Model-Free Execution Monitoring by Learning from Simulation*

Ola Pettersson, Lars Karlsson, and Alessandro Saffiotti
Center for Applied Autonomous Sensor Systems,
Department of Technology, Örebro University,
SE-70182 Örebro, Sweden
{*opn, lkn, asaffio*}*@aass.oru.se*

***Abstract*— Autonomous robots need the ability to plan their actions and to execute them robustly and in a safe way in face of a changing and partially unpredictable environment. This is especially important if we want to design autonomous robots that can safely co-habitate with humans. In order to manage this, these robots need the ability to detect when the execution does not proceed as planned, and to correctly identify the causes of the failure. An execution monitoring system is a system that allows the robot to detect and classify these failures. In this work we show that pattern recognition techniques can be applied to realize execution monitoring by classifying observed behavioral patterns into normal or faulty behaviors. The approach has been successfully tested on a real robot navigating in an office environment. Interesting, these tests show that we can train an execution monitor in simulation, and then use it in a real robot.**

***Index Terms*— Mobile robotics, monitoring, learning, diagnosis**

I. Introduction

Autonomous mobile robots that act in a changing and partially unpredictable environment must have the ability to plan their actions and to execute them robustly and in a safe way. Since the real world is dynamic and not fully predictable, the robot must also have the ability to detect when the execution does not proceed as planned, and to correctly identify the causes of the fault. An *execution monitoring system* is a system that allows the robot to detect and classify these faults. When a fault is detected and classified, the fault information can be used within the controller in order to recover from the fault situation. Therefore, a fast detection and accurate classification of the faults will help the system to find better recovery strategies and guarantee safety.

According to [8, p. 5] monitoring can be classified into *model-based* or *model-free* approaches. Inspired by this work we define these two categories as follows:

- *model-based* approaches use a predictive model in order to verify that the system evolves as expected, and
- *model-free* approaches solely observe the actual execution of the system, and detect certain patterns that indicate some problem.

Model-based execution monitoring is by far the most common approach within the field of robotics. Nevertheless, model-based approaches may have serious drawbacks. The main assumption when using model-based approaches is that a precise mathematical model of the system is available. In many complex systems it is difficult, or even impossible to create such models. This concerns, for example, behavior-based systems where the combination of different reactive behaviors may produce an overall behavior, sometimes called an *emergent behavior*, that is very difficult to model. Other examples of systems that are hard to model are systems that involve many different input variables or many unknown parameters. In general, model-based approaches run the risk to interpret as faults discrepancies between the estimated output and the measured output, which are instead caused by modeling errors. These modeling errors may be due to inaccuracy in some physical parameters, or to unmodeled dynamics caused by simplifying a higher-order model with a lower-order model.

Model-free execution monitoring does not suffer from these disadvantages. In a model-free approach the monitor is designed by learning from execution data without the use of a model. A model-free approach, therefore, is not limited by the accuracy of the a priori model provided to it, but rather by the extent of the data used to train it. This is perhaps the greatest drawback of this class of approaches: in order to provide reliable performance, the training data must provide an adequate coverage of all the fault situations. Unfortunately, it is not always feasible to force the robot into all known fault situations, and some of these situations can be catastrophic for either the robot or its operating environment.

In this paper, we study methods to realize model-free execution monitoring in behavior-based mobile robotics. We propose a pattern recognition approach incorporating two artificial neural networks that detect and isolate faults during plan execution. Suitable features extracted from the behavior activation levels act as input to the pattern recognition system, which is trained to recognize patterns of behavior activation which are characteristic of certain classes of faults. In order to overcome the above drawback, we let the execution monitor learn the fault classes safely in simulation, and we then use the system to monitor a real robot. We call this idea "learning from simulation." We report the results of a thorough evaluation of our approach in simulation, and we show that the system still performs

*This work was partly funded by the Knowledge Foundation in Sweden.

well when we apply it on a real robot.

II. Background

Model-based execution monitoring is by far the most common approach within the field of robotics. In this approach analytical models, often basic models from physics, are used in order to predict the system's outcomes. This approach rely on the concept of *analytical redundancy*. This means that two analytically generated quantities, most often the predicted state and the observed state, are compared. The resulting difference, called *residuals*, indicates the presence of a fault in the system.

The first model-based execution monitor applied to a mobile robot was PLANEX [4] on board Shakey. Here the predicted outcomes from the robot's actions were compared to the measured world-state.

The Procedural Reasoning System (PRS) [7] was developed as the execution and monitoring system on board the mobile robot Flakey. In PRS every plan has an expected outcome that is monitored by the system. In other words, the residual is given by comparing the measured world-state to the expected world-state. When a fault is detected, precompiled recovery routines can be called to handle the problem. PRS has been developed further and implemented in many different mobile robot control architectures. See for example, Saphira [10] or the LAAS architecture [14].

Livingstone [21] is the name of the control architecture used in NASA's first New Millennium spacecraft. Livingstone performs state identification of the modeled system in order to detect faults and isolate them. Also here the residual is given by comparing the measured state to the expected state. In Livingstone 2 [11] the state identification is improved by the use of Markov decision processes.

The execution monitoring device on board the mobile robot Xavier is programmed in TDL [19]. In this approach a fault is defined as a significant difference between the observed state of the world and the expectation.

In [12] the monitoring process is designed as an expert system implemented in linear temporal logic (LTL). In this model-based approach a set of temporal fuzzy rules, operating on the activation levels of the behaviors, are used for fault detection and diagnosis.

Model-free execution monitoring [16] is rarely seen in the robotics literature, although some systems do exist. An early example is presented in [6] where a model-free monitor is applied to an autonomous rover moving across the surface of a planet. In this approach several variables are measured and compared to a number of limit values. These limit values are obtained from statistical measures from several simulations. The use of limit value checking for execution monitoring is also presented in [15]. Here two fuzzy variables are analyzed in order to detect different degrees of failure in the plan execution of a mobile robot. These degrees are used for choosing between different recovery strategies.

Recent work on execution monitoring in planetary rovers is given in [3]. Here the system is described as a discrete-time probabilistic hybrid automaton. Instead of defining the different states in the automaton from predictive models, the transition functions between states are given from statistical measures from several observed variables. The monitoring is realized using *particle filters* [20].

Although not focused on execution monitoring but still related to this, is work on the DD&P architecture [9]. This work shows that the activation levels of the behaviors form certain patterns that can be learned in order to extract information about the robot's state. Here an *echo state network*, which is a particular type of recurrent neural network, is used for learning.

In [17] the activation levels of the behaviors are analyzed in order to detect and isolate different types of faults. Here the execution monitor is realized by a pattern recognition system using principal component analysis and multivariate statistics.

III. Model-Free Execution Monitoring

The main assumption in our approach is that faults form specific patterns in the system output. If this assumption holds, execution monitoring can be based on pattern recognition techniques. We consider execution monitoring on a mobile robot that is controlled by a hybrid architecture evolved from [18] called the Thinking Cap. The Thinking Cap (TC) consists of a fuzzy behavior-based controller, and navigation planning. In the TC, 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:

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

Depending on how similar the current situation is to the context, the different behaviors are activated to different degrees. These activation levels form the input data given to the execution monitoring system. An example of the activation levels is shown in Figure 5.

Our proposed learning from simulation approach consists of two phases:

- *Learning phase* The simulated robot is forced into different success or fault scenarios. During this phase the system is given sampled input data together with its corresponding class. With the use of these learning pairs, a mapping function between input data and status classes is created. In our case, the input data is the activation levels of the behaviors when the plan is executed. The classes indicate the status of the execution, i.e., success or type of fault.
- *Decision phase* The execution monitoring system that has learned from experiences given from simulation is ported to a real robot. The mapping function is used to classify new input data samples given during plan execution.

A general pattern recognition system is often divided into the following three steps [13]: (1) *data acquisition*, first the data is collected, (2) *feature extraction*, the collected data is analyzed and reduced into a set of features, and (3)

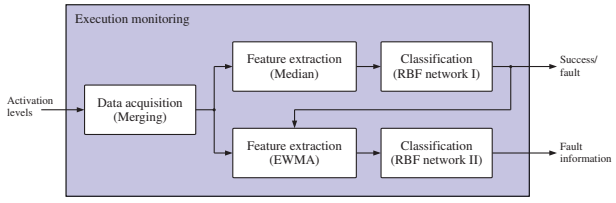

Fig. 1
SCHEMA OF A HIERARCHICAL EXECUTION MONITORING SYSTEM CONSISTING OF TWO PATTERN RECOGNITION SYSTEMS.

classification, the data is then classified into a set of known classes.

The methodology we employ is to train two radial basis function (RBF) networks by running a simulated robot on a number of scenarios where different plans are executed, both with successful results and with various faults. One network is used for fault detection, and one for fault isolation, as shown in Figure 1. The input data to the classification are features which are extracted from the activation levels of the fuzzy behaviors. The features are extracted from two specialized feature extraction methods in order to obtain better performance.

A. Data Acquisition

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. A straightforward solution to this problem is to group together all behaviors of a certain type, and to merge the activation levels connected to behaviors of the same type. For example, all activation levels connected to the behavior type CROSS(O) can be merged together independently of what door O the behavior refer to.

Let $\mathcal{T}^d$ be a fixed set of behavior types, and $T \in \mathcal{T}^d$ be one of these types. Let $C_j(s)$ denote the activation level connected to behavior B_j for each fuzzy rule j in a given B-plan P. Then the *merged activation level* for behavior type T in state s is computed by:

$$\bar{C_T}(s) \quad = \quad \max_{B_j \in T} C_j(s). \tag{1}$$

The activation level merging transforms the data into space $\mathcal{A}^{n \times d}$ where d is the number of behavior types given in $\mathcal{T}$. Since the number of behavior types d is fixed, and independent of the goal and the environment, the data is also plan independent.

B. Feature Extraction

The merged activation levels are passed to two different feature extraction functions; one specialized for fault detection and the other for fault isolation. In order to optimize the performance in terms of reaction speed and isolation power different feature extraction methods have been evaluated. The evaluated feature extraction methods are the following:

- cumulative sum (CUSUM),
- exponentially-weighted moving average (EWMA),
- sample variance, and
- median.

All the evaluated feature extraction methods use samples from several time instances. Therefore the feature extraction is performed on a set of samples within a time window. Let w denote the size of the window, that is, the number of samples within the window. If the notion of time within the time window is neglected, the time window for behavior type T can be written as a column vector

$$\mathbf{x}_T \quad = \quad [x_{1T}, x_{2T}, \ldots, x_{wT}]^{\mathrm{T}}. \tag{2}$$

The size of w is a parameter of interest in the performance optimization.

A cumulative sum (CUSUM) is the partial sum of the samples within the given time window. Since the CUSUM is a very simple calculation it has a very low computational cost. The exponentially-weighted moving average (EWMA) is calculated within time window $\mathbf{x}_T$ according to

$$\bar{x}_T \quad = \quad \frac{1}{w} \sum_{i=1}^{w} x_{iT}^2. \tag{3}$$

The sample variance within time window $\mathbf{x}_T$ is given by the following equation:

$$\mathrm{var}(\mathbf{x}_T) \quad = \quad \frac{1}{w-1} \sum_{i=1}^{w} (x_{iT} - \bar{x}_T)^2, \tag{4}$$

where $\bar{x}_T$ is the sample mean given as:

$$\bar{x}_T \quad = \quad \frac{1}{w} \sum_{i=1}^{w} x_{iT}. \tag{5}$$

The median is calculated by sorting the samples within the window and selecting the middle value.

The different feature extraction methods were evaluated in simulation by comparing the classification errors from the full execution monitoring system including classification. During the experiments window sizes between 20–120 samples were evaluated.

The extracted features are normalized with respect to the mean and the variance and stacked into a new feature data matrix $\mathbf{X}_F \in \mathcal{F}^{l \times d}$, where $\mathcal{F}$ represents the feature space. Each row in $\mathbf{X}_F$ represents a feature vector $\mathbf{x}_i$, where $i = 1, 2, \ldots, l$. These feature vectors are passed to the radial basis function networks.

C. Classification

Since artificial neural networks perform very well in pattern recognition problems [1], and have nonlinear discrimination power, we have chosen to use radial basis function (RBF) networks for classification. The motivation of using RBF networks instead of the most popular multilayer feed forward networks with backpropagation is that the training is much faster for the former [2].

As shown in Figure 2 a RBF network can be described as a two-layer network. We have chosen an approach,

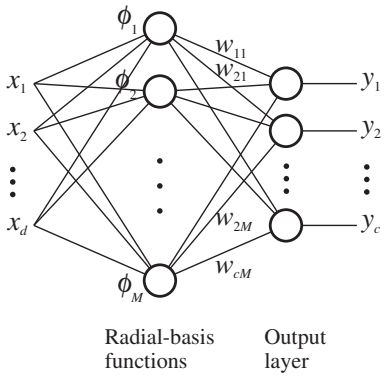

Fig. 2
THE STRUCTURE OF A RADIAL BASIS FUNCTION NETWORK.

presented in [2], where layer 1 consists of a number of nonlinear radial basis functions ϕ_j, where $j = 0, 1, \ldots, M$, with fixed centers and parameters in the nonlinearities. Hence, the input data is translated into a nonlinear space. The output layer then 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:

$$y_k(\mathbf{x}_i) \quad = \quad \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}_i), \qquad i = 1, 2, \ldots, l, \quad (6)$$

where l is the number of feature vectors, 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:

$$\phi_j(\mathbf{x}_i) \quad = \quad e^{-\frac{\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2}{2\sigma^2}}, \quad (7)$$

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, that is, train our network, by minimizing a suitable error function such as the sum-of-squares error function:

$$E \quad = \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{c}\left(y_k(\mathbf{x}_i) - t_{ik}\right)^2, \quad (8)$$

where t_{ik} is the target value, that is, the correct classification, for output neuron y_k given input vector $\mathbf{x}_i$.

Since the features are normalized in variance the static value σ was set to 1 in all experiments. Two separated networks are trained for fault detection and fault isolation respectively.

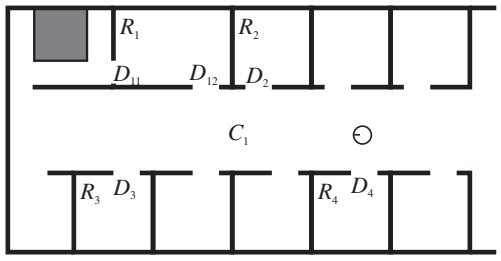

Fig. 3
THE OFFICE ENVIRONMENT.

IV. EXPERIMENTS

First a large number of simulated experiments were performed for evaluation of our proposed execution monitoring method. Different feature extraction methods were tested in order to optimize fault detection and isolation performance. The execution monitoring system that has learned from experiences given from simulation is then ported to the real robot. The motivation of the real robot experiments is to strengthen the previous statistical results given from simulation.

Both the simulation and real robot experiments are performed in indoor office environments including obstacles. During simulation several different office environments were evaluated. The real robot environment is shown in Figure 3. During the experiments the robot is given a simple service task, that is, to collect mail from one or many of the offices in the corridor.

Three different scenarios are tested, both in simulation and in the real robot experiments, namely:

1) success,
2) fault caused by a closed door (Fault CD), and
3) fault caused by a corrupted world-model (Fault CWM).

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.

A. Simulation Results

The simulation takes place in the Nomadic simulator, which includes simulation of sensor and positioning noise. Models are used within the simulator in order to simulate the interaction between the robot hardware and the environment. Although, no models are used in order to predict the robot behavior. All three scenarios were simulated 15 times, giving a total of 45 full simulated runs. Various amounts

TABLE I

MONITORING PERFORMANCE WHEN EWMA IS CHOSEN FOR FEATURE EXTRACTION.

Reaction speed	Time 2 s		Time 4 s		Time 10 s	
Robustness	$\bar{E}$	s	$\bar{E}$	s	$\bar{E}$	s
False alarms	0.095	0.079	0.107	0.073	0.036	0.058
Missed alarms	0.111	0.054	0.096	0.053	0.097	0.110
Isolation performance	$\bar{E}$	s	$\bar{E}$	s	$\bar{E}$	s
Fault CD	0.075	0.089	0.086	0.208	0.089	0.162
Fault CWM	0.012	0.038	0.007	0.032	0.028	0.023

TABLE II

MONITORING PERFORMANCE WHEN MEDIAN IS CHOSEN FOR FEATURE EXTRACTION.

Reaction speed	Time 2 s		Time 4 s		Time 10 s	
Robustness	$\bar{E}$	s	$\bar{E}$	s	$\bar{E}$	s
False alarms	0.083	0.049	0.096	0.078	0.079	0.131
Missed alarms	0.104	0.047	0.116	0.062	0.071	0.075
Isolation performance	$\bar{E}$	s	$\bar{E}$	s	$\bar{E}$	s
Fault CD	0.078	0.120	0.101	0.175	0.055	0.173
Fault CWM	0.019	0.038	0.011	0.049	0.017	0.026

of obstacles were used, from no obstacles at all to highly cluttered rooms. In the two fault scenarios different doors were closed.

It is of great importance that quantifiable criteria are used when evaluating the performance of a monitoring system. Inspired by [8] we have chosen to measure the following criteria when evaluating this hierarchical approach:

- *Reaction speed*, that is the ability of the technique to detect faults with reasonably small delays after their arrivals.
- *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 system, on the size of faults, noise, disturbances and model errors, and on the design of the algorithm.

The reaction speed depends on the number of samples that are stored within the time window. Since the sampling rate is 10 Hz, the reaction speed is given as $\frac{w}{10}$ seconds, where w denote the size of the window. The performance in robustness and isolation is here calculated by evaluating the mean error $\bar{E}$ and the standard error deviation s. To get an unbiased measure of these statistical measures, the *leave-one-out* procedure [5] was used. In this method, data from one simulation run is left out and used for testing and the rest is used for training. This procedure is then repeated until all simulation runs have been left out once for testing.

In Table I and II the performance using EWMA and median for feature extraction is presented. We have omitted the performance measures from CUSUM and sample variance, since these feature extraction methods provided significantly worse results. As can be seen in Table I and II, the robustness is improved when a larger time window, that is, a slower reaction speed, is used. This means, there is a trade-off between reaction speed and robustness.

B. Real Robot Results

Since our Nomad 200 robot that is simulated is retired, one of our Magellan Pro robots (shown in Figure 4) is used in the real robot experiments instead. The Nomad 200 and the Magellan Pro, are equipped with similar sensors but differs in size and steering. The Nomad 200 is approximately 0.6 m in diameter and is driven by three steering wheels. The Magellan Pro robot is approximately 0.40 m in diameter and is driven by a 2-wheel drive with differential steering. This means, that the execution monitor is trained with data from one type of robot platform in simulation, and evaluated on an other type of robot platform in the real robot experiments.

Fig. 4

OUR MOBILE ROBOT REACHES A CLOSED DOOR.

Each scenario, that is, success, fault CD, and fault CWM, is tested three times using our Magellan Pro robot. Figures 5–7 show examples from execution of the success, fault caused by a closed door, and fault caused by a corrupted world-model scenarios, respectively. Each figure is divided into two sub figures: The top figure shows the merged activation levels that are used as inputs to the execution monitor, and the bottom figure shows the corresponding output given from the execution monitor. The time is given in units of 0.1 seconds.

Figure 5 shows an example of the success scenario. Here the CROSS behavior becomes active when the door is successfully crossed. This scenario is 100 % correctly classified by the execution monitor.

An example of the fault caused by a closed door scenario is shown in Figure 6. In this scenario almost all behaviors become inactive when the door is detected closed and the robot cannot proceed its execution of the plan. The fact

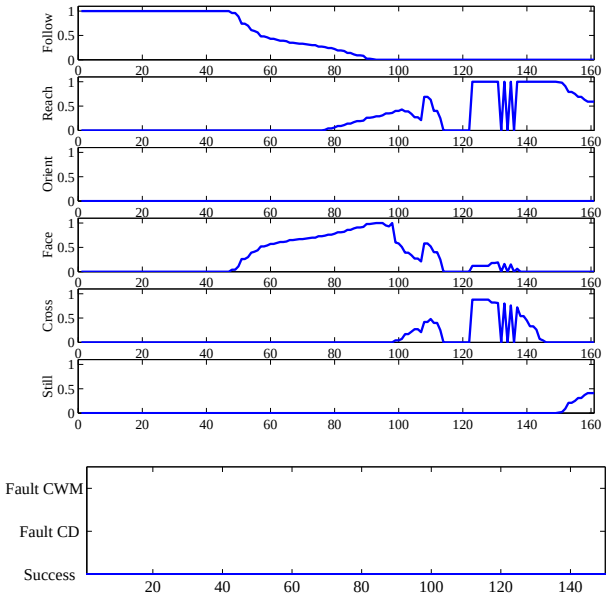

Fig. 5

AN EXAMPLE OF THE SUCCESS SCENARIO. TOP: THE CROSS BEHAVIOR BECOMES ACTIVE AFTER APPROXIMATELY 10 SECONDS. BOTTOM: THE OUTPUT SIGNAL FROM THE EXECUTION MONITOR, INDICATING SUCCESS DURING THE ENTIRE SCENARIO. THE TIME IS GIVEN IN UNITS OF 0.1 SECONDS.

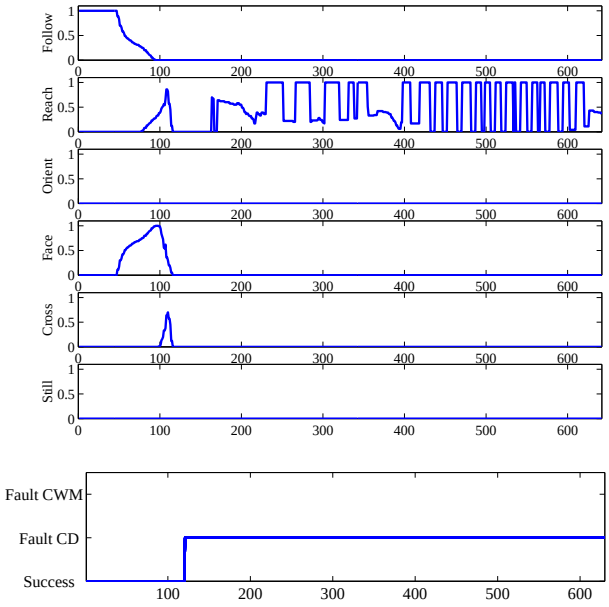

Fig. 6

AN EXAMPLE OF THE FAULT CAUSED BY A CLOSED DOOR SCENARIO. TOP: AFTER SOME TIME ALL BEHAVIORS, EXCEPT FOR REACH, BECOME INACTIVE. BOTTOM: THE OUTPUT SIGNAL FROM THE EXECUTION MONITOR, INDICATING FAULT CD AFTER APPROXIMATELY 12 SECONDS. THE TIME IS GIVEN IN UNITS OF 0.1 SECONDS.

that the goal point is located just on the other side of the closed door confuses the REACH behavior. Therefore this behavior is still activated even thought the door is closed. Nevertheless, this scenario is also 100 % correctly classified by the execution monitor.

In Figure 7 an example of the fault caused by a corrupted world-model is shown. Here the activation levels of the behaviors REACH, ORIENT, and FACE are oscillating when the robot keeps on trying to cross the closed door. In this scenario the execution monitor has some problems in determining the correct output. Here a little more than 20 % of the execution time is misclassified.

V. CONCLUSIONS

Model-based execution monitoring is by far the most common approach within the field of robotics. Nevertheless, model-based approaches require that precise analytical models of the monitored system are available, and are sensitive to modeling errors. In this paper, we have presented a model-free approach to robot execution monitoring, and we have shown its effectiveness in both simulated and real-world experiments. We have also shown that we can train the monitor in simulation, thus overcoming the problems connected with the generation of fault situations on a real platform, and then apply it on a real robot.

It should be noted that our approach relies on just one assumption: the use of a behavior-based controller in which behaviors have activation levels. In this sense, our approach is rather general, since many of the current architectures for autonomous robotics satisfy this assumption. In virtually all

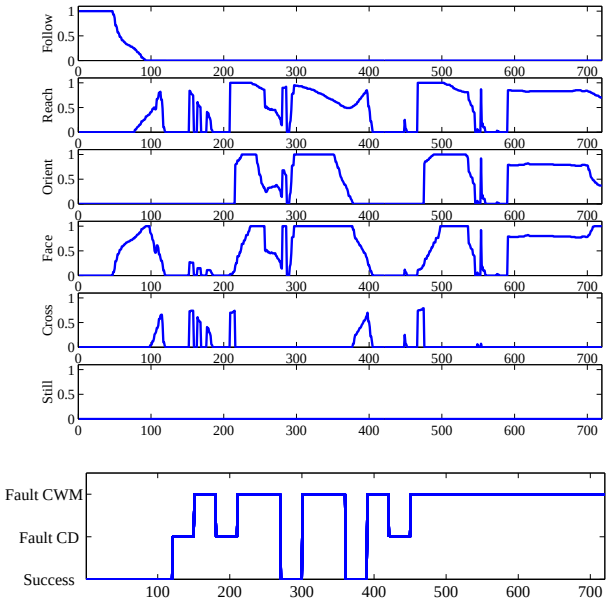

Fig. 7

AN EXAMPLE OF THE FAULT CAUSED BY A CORRUPTED WORLD-MODEL. TOP: THE OSCILLATING BEHAVIOR PERFORMED BY REACH, ORIENT, AND FACE. BOTTOM: THE OUTPUT SIGNAL FROM THE EXECUTION MONITOR. THE TIME IS GIVEN IN UNITS OF 0.1 SECONDS.

other matter our work is not dependent on the architecture or the robot. In our model-free approach the monitor only incorporates knowledge about the relation between the activation levels of the behaviors, and the fault classes. Models describing the environment and the robot hardware are needed within the simulator. Nevertheless, no models of the robot controller are used; therefore, no prediction of the robot behavior is performed.

REFERENCES

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

[2] 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.

[3] R. Dearden, F. Hutter, R.G. Simmons, S. Thrun, V. Verma, and T. Willeke. Real-time fault detection and situational awareness for rovers: Report on the mars technology program task. In *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, USA, 2004. To appear.

[4] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[5] K. Fukunaga and D.M. Hummels. Leave-one-out procedures for nonparametric error estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):421–423, 1989.

[6] E. Gat, M.G. Slack, D.P. Miller, and R.J. Firby. Path planning and execution monitoring for a planetary rover. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 20–25, Cincinnati, OH, USA, 1990.

[7] M.P. Georgeff and A.L. Lansky. Reactive reasoning and planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 677–682, Seattle, WA, USA, 1987.

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

[9] J. Hertzberg, H. Jaeger, and F. Schönherr. Learning to ground fact symbols in behavior-based robots. In *Proceedings of the European Conference on Artificial Intelligence*, pages 708–712, Amsterdam, Netherlands, 2002.

[10] 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.

[11] J. Kurien and P.P. Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 370–377, Austin, TX, USA, 2000.

[12] K.B. Lamine and F. Kabanza. Reasoning about robot actions: A model checking approach. In M. Beetz, J. Hertzberg, M. Ghallab, and M.E. Pollack, editors, *Advances in Plan-Based Control of Robotic Agents*, number 2466 in Lecture Notes in Computer Science, pages 123–139. Springer-Verlag, Berlin, Germany, 2002.

[13] Y. LeCun and Y. Bengio. Pattern recognition. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 711–715. MIT Press, Cambridge, MA, USA, 1995.

[14] F.R. Noreils and R.G. Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, 11(2):255–266, 1995.

[15] S. Parsons, O. Pettersson, A. Saffiotti, and M. Wooldridge. Robots with the best of intentions. In M. Wooldridge and M.M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, number 1600 in Lecture Notes in Artificial Intelligence, pages 329–338. Springer-Verlag, Berlin, Germany, 1999.

[16] O. Pettersson. *Model-Free Execution Monitoring in Behavior-Based Mobile Robotics*. Ph.D. Thesis, Örebro University, Department of Technology, Örebro, Sweden, 2004.

[17] 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.

[18] 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.

[19] R.G. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1931–1937, Victoria, BC, Canada, 1998.

[20] V. Verma, G. Gordon, R.G. Simmons, and S. Thrun. Real-time fault diagnosis. *IEEE Robotics and Automation Magazine*, 11(2):56–66, 2004.

[21] B.C. Williams and P.P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 971–978, Cambridge, MA, USA, 1996.