A ROBUST STOCHASTIC SUPERVISION ARCHITECTURE FOR AN INDOOR MOBILE ROBOT

Joaquín L. Fernández, Reid Simmons**, Rafael Sanz* and Amador R. Diéguez**

*Dept. de Ingeniería de Sistemas, Universidad de Vigo, Campus Lagoas-Marcosende, 36200 Vigo, Spain
Phone Int.: +34 986 812244, E-mail: {joaquin,rsanz,amador}@uvigo.es

**Computer Science Dept. Carnegie Mellon University, 5000 Forbes Avenue Pittsburgh, PA 15214 USA
Phone Int.: +1 408 555 1212, E-mail: reids@cs.cmu.edu

Abstract: Mobile robots operating in the real world need very reliable navigation capabilities to operate autonomously for long periods of time. However, it is almost impossible to specify in advance all the possible anomalous situations the navigation system can encounter, especially given noisy and imprecise sensor information and a dynamic and partially known environment. In this paper, we present an architecture for robust execution monitoring of the different tasks of the robot. This approach uses a set of monitors that get information about the robot's state and, rather than detecting fault states directly, detects significant differences between perceived and expected states. To deal with the uncertainty about the knowledge of the state of the system and the result of some actions, it uses a POMDP model to decide when is worthy to take recovery actions. We present the general approach and show its application in the domain of indoor mobile robot navigation.

Keywords: Supervision, mobile robots, detection, diagnosis and fault recovery, POMDP.

1. INTRODUCTION

One of the goals for intelligent mobile robots is the ability to operate autonomously in all situations, even to respond in advance before anomalous situations arise. Since it is almost impossible for the programmer to predict all the circumstances that might be encountered, however, a general mechanism is required to handle failure situations. For this purpose, different supervisory functions must be implemented.

Advanced supervision and fault diagnosis to improve reliability, safety and economy is a subject of intensive research. Most of this research is carried out in industrial processes. In the field of mobile robots, solutions to deal with errors strongly depend on the architecture used in the navigation system. For instance, error recovery is not employed in purely reactive systems because the system basically reacts to events. In the context of robot navigation, several architectures deal with exceptions and sensor errors in different ways [Noreils and Chatila, 1995][Stuck, 1995]. Monitoring and recovery processes are responsible for verifying that the robot is correctly executing its tasks, detecting when is not and handling exceptions.

The faults commonly handled are due to the sensor system either only at hardware level or also including software faults. While some architectures only consider faults in sensors and effectors, in others they also take into account the faults at the navigation level. The treatment of faults in this last group is always conditioned by the architecture of navigation used, leading to different supervision, fault detection and recovery systems.

In this paper, an exception detection and recovery architecture for mobile robots is presented. The architecture uses a set of monitors to gather information about the state of the robot [Fernández and Simmons, 1998] and is able to either (1) command a set of actions to recover from errors, (2) obtain information about the state of the robot, or (3) try to reach the goal. The idea is to provide coverage for many types of unexpected and unanticipated situations, while at the same time enabling the robot to quickly detect, and react to, specific contingencies. Our monitoring and recovery architecture is independent of the navigation architecture. Like other systems, the architecture used in our research combines reactive and deliberative behaviors. The approach described in this paper has roots in earlier work with a previous navigation system [Simmons, 1994b] [Fernández and Simmons, 1998].

2. SOME CONSIDERATIONS ABOUT THE SUPERVISOR

Our supervision architecture uses a stochastic model to decide which action to do for the next step, based on the current (estimated) state, current senor values, and the different alternative actions it can perform.

We take the role of execution monitors to be to detect *symptoms*. A symptom is defined as a significant difference between the observed state-of-the-world and the *expectation* with respect to the *nominal* situation. Depending on the metrics used to measure that difference, we can set up different monitors to look for different symptoms. In navigation, for example, one metric is the time the robot takes to achieve its goal. Another measure is the distance traveled by the robot. In the latter case, a symptom will be that the robot is moving too slowly, or not moving at all. The idea is that such symptoms, which we refer to as *exception situations*, will usually be associated with situations where the robot is stuck or is performing poorly named.

Due to system complexity, many issues must be taken into account in order to develop a module to robustly detect and recover from exception situations. First, it is necessary to consider that, due to noise and modeling limitations, monitors can detect symptoms that are not always caused by a failure. If we could set up monitors that classify exactly the failure state of the robot then, depending on the state, we could take the best action for that state [Fernández and Simmons, 1998]. Unfortunately, this is not possible for most of the situations.

Another issue to consider is that the penalty of a false detection is not the same for all the problems. Even more, for the same problem in different situations the penalty of a wrong decision can be different. It is also necessary to bear in mind that the recovery mechanisms are not perfect. As with navigation, almost all the recovery actions can themselves fail (e.g., due to perception problems, environment exceptions, etc).

Finally, in order to make intelligent decisions about when is worthwhile to change the nominal action and execute a recovery action, the state of the system must be considered. However, this state is not deterministically known. Instead, we have a set of observations from different sensors, results of actions, etc. The actual state of the system should be estimated from all the history of these observations.

Our architecture uses a stochastic model of the system to deal with all these uncertainties. To date, we have built the model based in our experience working with the robot.

3. STOCHASTIC SUPERVISOR

Fault diagnosis and recovery can be seen as a sequential process involving three steps: symptom extraction using the monitors, identification, and recovery (if necessary). A supervisor is in charge of monitoring the system, searching for exception situations, and selecting recovery actions when necessary.

Real-world robotic systems have much uncertainty due to (1) limitations in the sensors, (2) dynamic, hard-to-model, and non-predictable environments, and (3) non-deterministic effects of actions. Therefore, in order to decide what the robot should do for next step, the supervisor must take into account the uncertainty in the observation about the system and in the action results, the action costs in different situations, and past information.

A decision process that supports the above requirements is the *Partial Observable Markov Decision Process* (POMDP) where the uncertainties are expressed using probabilities [Cassandra, 1998]. A POMDP Ξ can be described as a sextuple [Kaelbling et al., 1998]:

$$\Xi \equiv (S, A, Z, T, O, R) \qquad (1)$$

where:

- ✓ **S** is a finite set of states of the system
- ✓ **A** is a finite set of actions the robot can do.
- ✓ **Z** is a finite set of observations the robot can obtain.
- ✓ **T : S × A → Π(S)** is the state transition function. **T(s,a,s')** is the probability of ending in state **s'** when the robot is in state **s** and executes action **a**.
- ✓ **O: S × A → Π(Z)** is the observation function. **O(s',a,o)** is the probability of making observation **o** while doing action **a** and ending in state **s'**.
- ✓ **R : S × A × S → ℜ** is the reward function. **R(s,a,s')** is the expected reward for taking the action **a** in state **s** and ending in state **s'**.

Our approach, then, is to model the system as a POMDP and apply POMDP theory to find a policy that dictates which action to execute at each step. In next section, we describe how we model navigation as a POMDP. Section 5 describes how the robot uses this POMDP to decide what to do.

4. CONSTRUCTING THE POMDP MODEL

In order to describe the system as a POMDP, we have to define the different parameters (**S, A, Z**), probabilities (**T, O**) and rewards (**R**).

4.1. Parameters

States

In order to define the system states, the different exception situations must be determined. Even though we cannot identify all the possible exception situations, we can classify them.

Figure 1 shows a classification for some of the possible problems. An exception situation can be a combination of one or more problems.

To assure completeness, we need a cut through the tree of possible problems (Figure 1), at some level of detail. If we want to be more specific about the exceptions, we should use the subsets represented by the leaves of the tree. However, since we don’t have implemented so far enough mechanisms to identify every subset, we use only the items in squares in Figure 1, which are also shown in Table 1. Nevertheless, new subsets can be added easily if new observations are included. For example, instead of the non-navigable environment set, we can use the subsets obstacle and floor discontinuities, provided we have ways of detecting those situations.

Note that the state of the system is a combination of these possible problems. The state without any problem is labeled as the *nominal state*. The exception situations are all the possible combinations of the components shown in the Table 1. In particular, considering the seven exception components shown, the whole system can be in $2^7 = 128$ possible states. A detailed description can be found in [Fernández, 2000].

Table 1. Basic problems considered in the supervisor

Name	**Description**	**Source**
NNP	Non Navigable Path	Environment
HP	Hardware Problems	Hardware
RT	Reactive Problems	Soft. (React.)
PP	Perception Problems	Soft. (Per.)
WD	Wrong Direction	Soft. (Nav.)
LST	Robot Lost	Soft. (Nav.)
NNA	Non Navigable Alternative	Environment

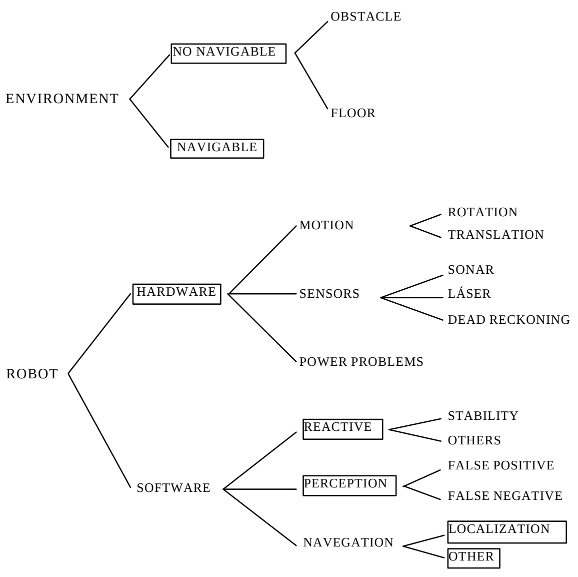

Figure 1. Classification of the possible problems

Actions

An action is a single procedure performed by the robot in order to accomplish a task or recover from an exception situation. Table 2 shows the six actions developed in our robot system.

Observations

Information about the system comes from two different sources: monitors and actions. A monitor informs the *devisor* (decision making module) whenever its monitored condition is detected. In addition, the decisor uses the absence of monitor firings at each step to reinforce the nominal state probability. Some actions communicate the results of their successes. Note that neither sources of information is perfect and they only provide symptoms about the state of the robot. Table 3 lists the possible components of the observations the system can get in one step. These observations could be any combination of the components.

Table 2. Basic robot actions

Action	**Description**
FP	Nominal action. Follow planned path
NP	Request a new path
RL	Relocalize using other methods
GO	Go opening
MA	Move away from the current position
SH	Send a message for assistance.
GU	Give up on this task.

Table 3. Basic Observations

Obs.	**Description**	**Source**
EPS	Error Position (stopped)	Monitor
EPM	Error Position (moving)	Monitor
LD	Loop Detected	Monitor
SD	Spinning Detected	Monitor
BO	Blockage Overtaken	Action (GO)
ALT	Alternative path found	Action (NP)
NNA	Non Navigable Alternative	Environment

The monitors are implemented as one independent process that uses TCA control constructions to coordinate with the executing navigation tasks [Simmons, 1994b]. A detailed description about how the monitors work is presented in [Fernández, 2000].

4.2. Probabilities

The POMDP model of the system is defined using a syntax similar to the one presented in [Cassandra 1998], with some new features added to the syntax. In the following, we describe how we define the probability models used.

Transition probabilities

Since the transition probability is a function:

$$T : SxA \rightarrow \Pi(s) \qquad (2)$$

there are a total of $|S| \times |A| \times |S|$ (114,688) probabilities to specify. We can significantly reduce the number and effort of specifying probabilities if we assume that the components of the states are independent. While this is correct for most of our cases, there are some obvious relations among some components that need to be modeled. For example the reactive and perception problems. Fortunately, the extended syntax of our model allows us to represent them as particular cases.

Observation probabilities

For each state and action, the probability of getting each possible observation must be established:

$$O : SxA \rightarrow \Pi(z) \qquad (3)$$

Again, instead of specifying all |S| × |A| × |Z| (57,344) probabilities, we can reduce the number by considering that the observation components are dependent on the state and the action, but are independent of the other components. The number of necessary transitions to define can also be cut down by taking into account that not all the components can be observed for all the actions. Table 4 shows the possible observations obtained during the execution of each action.

Table 4 Possible observations for each action

Actions	**Observations**
Follow path	EPS EPM LD SD
Go through opening	BO
Relocalize	
New path	ALT
Move away	
Give up	
Rep. hardware prob.	

All the monitors and, therefore, all the observations that come from the monitors are only active while navigating. That means that we can only get these observations during the Follow Path (FP) action. Therefore, the probability is zero of getting these observations for any other action. The probabilities to get a combination of several observations are calculated simply by multiplying the individual probabilities for each observation. For example, the probability that only the Error Position Static (EPS) is fired during navigation will be:

$$\mathrm{P(EPS)} = \mathrm{P(EPS)} \times \mathrm{P}(\{\overline{\mathrm{EPM}}\}) \times \mathrm{P}(\{\overline{\mathrm{LD}}\}) \times \mathrm{P}(\{\overline{\mathrm{SD}}\}) \qquad (4)$$

Because

$$\mathrm{P}(\overline{\mathrm{BO}}) = \mathrm{P}(\overline{\mathrm{ALT}}) = 1 \qquad (5)$$

for the New Path (NP) action.

4.3. Rewards

There are different ways we can define the rewards and this is somewhat subjective for the user. It is possible to set rewards for an action, rewards for all the states that include a failure component in the start state or in the end state. For the system we are dealing with, a cost for each action has been selected according to time needed and giving a high penalty to some actions like Give Up (GU). The following criteria have been followed:

- ✓ Reward for reaching the state “everything ok”
- ✓ Reward for doing Follow Path (FP) when it is in nominal state.
- ✓ High penalty to the action Send Help (SH).
- ✓ High penalty to the action Give Up (GU).

5. EXCEPTION IDENTIFICATION AND RECOVERY

The task of exception identification and recovery is to determine the exception situation and decide on the appropriate response. The decision process is done in two phases (Figure 2): (1) the *belief state* of the robot is updated according to the information gathered in the last step and, (2) the decisor selects the action according to the *belief state* of the system.

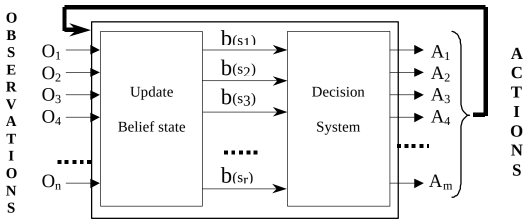

Figure 2. Stochastic supervisor

5.1. Updating the belief state

The system does not know the state of the robot exactly because of the uncertainty in the information sources. Instead, it maintains a *belief state* (a distribution probability b over the states S). An interesting property of the *belief state* is that, under the Markov property, it summarizes all the past information about the state of the robot. That is, the *belief state* is a *sufficient statistic* for the history. We use **b(s)** to denote the probability that the robot is in state **s** and $\mathbf{b^a_z(s)}$ the probability that the robot ends in state **s** after executing **a** and observing **z**. The *a posteriori* probability for each state when action **a** is executed and **z** is observed can be obtained [Kaelbling et al., 1998] by:

$$b_z^a(s') = \frac{o(a,s',z)\sum_s t\,(s,a,s')b(s)}{\sum_{s,s''} o(a,s'',z)t\,(s,a,s'')b(s)} \qquad (6)$$

5.2. Decisor

Since the *belief state* is a *sufficient statistic*, the optimal decision is one based solely on the current *belief state*. To determine the optimal decision, it is necessary to have some measure to evaluate decisions. That measure comes from the rewards that were defined in Section 4.3. While an exact solution of the POMDP model will produce an optimal policy Π{MLS}, unfortunately, with known algorithms, this is not feasible for models with as many states, actions, and observations as the model presented here. Instead, we use approximate solutions proposed in [Cassandra 1998]. These solutions are based on calculating the optimal policy for the underlying Completely Observable Markov Decision Problem (COMDP), assuming no state uncertainty. The different POMDP approximate solutions depend on how to select the action to execute given the COMDP $\mathbf{\Pi_{CO}(s)}$ policy and the *belief state* **b(s)**.

1. *Most Likely State* (MLS) simply chooses the optimal COMDP action associated with the state that has the highest probability:

$$\Pi_{MLS}(b) = \Pi_{COMDP}(\arg\max_{s} b(s)) \tag{7}$$

2. *Action Voting* (AV) assigns a probability over the actions:

$$P_a(b) = \sum_{s} b(s)\mathrm{d}(\Gamma_{CO}(s), a) \tag{8}$$

where:

$$\mathrm{d}(x,y) = 1 \text{ if } x == y\text{; } 0 \text{ otherwise} \tag{9}$$

Then, selects the action with highest probability:

$$\Gamma_{AV}(b) = \arg\max_{a} P_a(b) \tag{10}$$

3. *Q-MDP control strategy* uses the function value $V^a(s)$ and is given by:

$$P_a(b) = \sum_{s} b(s) V'^{a}(s) \tag{11}$$

where $V^a(s)$ is the reward of executing *a* in the next step when the robot is in *s*, considering that after the next step all uncertainty is removed and the system behaves optimally.

4. The *Dual Control Mode strategies* (D-X, A-X) have two different modes of operation depending on a threshold factor K. They use the X strategy where X can be any of the aforementioned strategies (MLS, AV, Q_MDP) when the entropy is under K and select the action that most reduces the *expected state entropy* (D-X) or the *expected action entropy* (A-X).

6. EXPERIMENTAL RESULTS

An event simulator was used to compare the performance of the different heuristic approximations. The best performing strategy was then implemented on the robot. Then, we created some exception scenarios with the robot to see how it behaves.

Since we do not have the optimal solution to compare the simulation results, we have used an omniscient (OMNI) controller to find a superior bound on the reward. The results in Figure 3 show the average reward for different methods. The considerable difference of the results for this method gives us an idea of the importance of uncertainty in the system. We assume that the robot always has tasks to do and we use a discount factor of 0.85 to calculate the reward. With regard to the parameterized algorithms (A-X, D-X), the figure illustrates best results from the range of parameter settings used. We have considered two starting situations. In the first situation the robot starts from the nominal state and the state is known. In the second situation the starting state is randomly selected.

From Figure 3, we can see that MLS performance is one of the best decision strategies. Since it is also simple and fast to compute, we decided to use it for the tests in the robot.

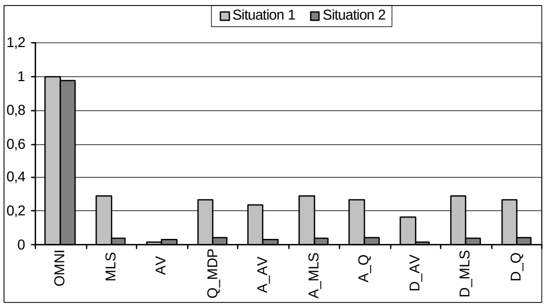

Figure 3. Reward for the different decision methods.

6.1. Robot tests

Two robot platforms have been used in this research: *Xavier* (Carnegie Mellon University-USA) and *Rato* (University of Vigo-SPAIN). *Xavier* is designed with a layered architecture, consisting of task scheduling, path planning, navigation, and obstacle avoidance components, each of which relies on the abstraction provided by the previous level [Simmons 2000]. Obstacle avoidance is performed by the Curvature-Velocity method and navigation is done using Partially Observable Markov Decision Process (POMDP) models [Simmons and Koenig, 1995]. Path planning uses an A* algorithm strategy. The robot architecture is implemented as a collection of asynchronous processes. System integration is performed using the Task Control Architecture (TCA) [Simmons 1994a]. The *Xavier* navigation system has also been implemented in *Rato*, with minor changes, proving the versatility of the architecture.

The supervisor, using an MLS, decisor has been tested in both robots (*Xavier* and *Rato*) showing the desired behaviors for a variety of situations. Because of the low failure rate of the navigation system, we have created artificial exception situations.

The simplest scene is to block a corridor that is in the planned path of the robot. The position monitor (EPM) is fired several times increasing the probability of reactive problems (RT), perception problems (PP), or a non-navigable path (NNP), among others. Once the supervisor is confident about the existence of an exception situation, even if it does not know exactly *why* such a situation exists, it decides to execute Go Opening (GO). The observation that there is no opening increases the probability of a non-navigable path (NNP). Since it is quite confident about a NNP situation, decides to execute New Path (NP).

In the case of a non-observable obstacle, things are a bit more complicated. First, since the robot does not detect the obstacle, the probability of PP/RP *(combination of PP and RP)* increases, but NNP decreases. Given that, the robot first decides to use Move Away (MA), which decreases the probability of reactive problems. Since the position monitor keeps firing, however, the robot will end up in a non-navigable path/perception problems (NNP/PP) situation where it decides, once again, to use a New Path (NP).

7. CONCLUSIONS

We have presented a general approach to execution monitoring that uses *Partial Observable Markov Decision Processes* to detect *exceptions* between the expected and observed states of the world. This approach has been used in the context of indoor mobile robot navigation.

Instead of errors, we deal with *exception situations* since this is a more general concept that includes all situations not contemplated in the design phase that can keep the robot from reaching its goal.

While we do not have complete coverage of the exception space, the monitors developed do cover a very wide range of the exceptions that commonly occur in indoor navigation. Some of these monitors, such as a watchdog timer, need little domain information, and are very easy to encode. Most, however, are very task-specific, and needed to be tuned for the particular robots and the environments in which they normally operate (for instance, battery characteristics, or average speed). While we would like to learn such monitors automatically, our experience indicates that it would be a difficult task, in general. For now, we continue to experiment with the set of hand-coded monitors we have developed, and we working on adding more sophisticated recovery strategies using the same stochastic model to decide which strategy to use at each moment.

The biggest strength of this model is that, if the model matches the real system, the decisor will perform quite well. On the other hand, the main shortcoming is the tedious task of finding the correct parameters, probabilities, and rewards. Currently, we need to set up the probabilities by hand, based on experience, and tune the rewards so that the robot will show the desired behaviors. We believe, however, that this can be done reinforcement learning, and we are trying to extend our research in this way. In any case, the results we currently achieve are much better than with a deterministic system, even though the probabilities may not be completely accurate.

ACKNOWLEDGEMENTS

This work has been partially supported by the *Comisión Interministerial de Ciencia y Tecnología* (CICYT), project TAP93-0968-E. We would like to thank all the people that have influenced this work, particularly the members of the Robot Learning Lab in CMU. We also thank the *Xunta de Galicia* for funding Joaquin's visit to Carnegie Mellon.

REFERENCES

[Cassandra, 1998] A. R. Cassandra. Exact and approximate algorithms for partially observable Markov decision Process. PhD Thesis, Department of Computer Science, Brown University, Providence, Rhode Island, May 1998.

[Fernández and Simmons, 1998] J.L. Fernández and R. G. Simmons (1998). Robust Execution Monitoring for Navigation Plans. Proc. 1998 IEEE/RSJ International Conference, pages 551-557, Victoria, B.C., Canada, 1998.

[Fernández, 2000] J. L. Fernández. Supervision, detection, diagnosis and exception recovery in autonomous mobile robots. Ph.D. Thesis. University of Vigo. March 2000.

[Kaelbling et al., 1998] L. P. Kaelbling, M. L. Littman and A. R. Cassandra. Planning and acting in Partially Observable Stochastic Domains. Artificial Intelligence (101) 1-2: 99-134, 1998.

[Noreils and Chatila, 1995] F. R. Noreils and R. Chatila. Plan Execution Monitoring and Control Architecture for Mobile Robots", IEEE Transactions on Robotics and Automation, 11(2): 255-266. 1995.

[Simmons, 1994a] R. G. Simmons. Structured Control for Autonomous Robots. IEEE Transactions on Robotics and Automation. 10(1), February 1994.

[Simmons, 1994b] R. Simmons. Becoming increasingly reliable. In Proc. 2nd Int. Conference in Artificial Intelligence Planning Systems, Chicago, IL, June 1994.

[Simmons and Koenig, 1995] R. G. Simmons and S. Koenig. Probabilistic Navigation in Partially Observable Environments. IJCAI Intl. Conference, Montreal Canada, July 1995.

[Stuck, 1995] E. R. Stuck. Detecting and Diagnosing Navigational Mistakes, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Pittsburgh, PA, August 5-9, 1995.

[Simmons, 1994b] R. Simmons, J. L. Fernández, R. Goodwin, S. Koenig and J. O'Sullivan. Lessons learned from Xavier, IEEE Robotics & Automation Magazine, 7(2): 35-39, June 2000.