Case Study Results for Probabilistic Error Propagation Analysis of a Mechatronic System

Dr.-Ing. Andrey Morozov, Prof. Dr. techn. Klaus Janschek,
Faculty of Electrical and Computer Engineering, Technische Universität Dresden, Institute of Automation, 01063 Dresden, Deutschland, {andrey.morozov, klaus.janschek}@tu-dresden.de

Abstract

This paper addresses a probabilistic analysis of the propagation of errors through mechatronic systems caused by either hardware or software faults. System design, safety verification, testing, and many other activities that are aimed at dependable system development require such an analysis. The specifics of the mechatronic domain necessitate the development of an appropriate mathematical model that allows the proper mapping of the mutual interaction of heterogeneous system components such as software, hardware, and physical parts. A recently introduced new concept of system level error propagation analysis is based on the idea of simultaneous examination of two directed graph system models: a control flow graph and a data flow graph as abstraction of system operation. This paper describes the application of the discussed approach using a laboratory-developed mobile robot system. It is focused on design and computational issues, numerical results, and possible system dependability refinements based on the error propagation analysis.

Figure 1 A general structure of the proposed approach for error propagation analysis.

1 Introduction

This paper addresses a probabilistic analysis of the propagation of errors through mechatronic systems caused by either hardware or software faults. Such type of analysis supports a variety of engineering tasks. System design, reliability and safety estimation, testing, diagnostics, and many other activities that are aimed at dependable system development require a deep understanding of system’s behavior under erroneous conditions. The multi-domain characteristics of the mechatronic systems necessitate the development of an appropriate abstraction through mathematical modeling. An error propagation model for these types of systems must use a high abstraction level that allows the proper mapping of the mutual interaction of heterogeneous system components such as software, hardware, and physical parts.

In [1] and [2] we have introduced a new concept of a dual-graph probabilistic model for system-level error propagation analysis. The central idea is a synchronous examination of two directed graphs: a control flow graph and a data flow graph as abstraction of system operation. The structures of these graphs can be derived systematically during system development from a baseline system model, e.g. UML/SysML [3,4] or Matlab/Simulink.

The knowledge about an operational profile and properties of individual system components allow the definition of additional parameters of the error propagation model. At the next stage of the introduced approach we use a discrete time Markov chain that can be generated from the discussed dual-graph system representation. Specific computation of this Markov chain provides helpful probabilistic results for system development.

A general structure of the entire approach to error propagation analysis is shown in Figure 1. Section 2 introduces a dual-graph error propagation model. This model can be generated using a base-line system representation. In the presented case study we have used a UML activity diagram [3] as a baseline system model. After generation of the dual-graph model, error propagation analysis can be carried out. Section 3 briefly discusses an approach to error propagation analysis using an absorbing discrete time Markov chain in order to describe error propagation processes during system execution. Section 4 presents the main part of this paper. It addresses the real world application of the discussed approach using a laboratory-developed mobile robot. This section is focused on design and computational issues, numerical results, and possible system dependability refinements based on the error propagation analysis. Section 5 contains a conclusion and a discussion of the obtained results.
2 Error propagation model

The dual-graph error propagation model (EPM) is an abstract mathematical framework for error propagation analysis of mechatronic systems. The system under evaluation is considered to be defined by a set of independent elements. Each element represents an executable part of the system, e.g., a block of software source code, a hardware sensor that performs measurements, a controller or an actuator.

Two directed graph models are derived for this set of elements: a data flow graph (DFG) and a control flow graph (CFG). Nodes of both graphs represent the elements of the system. Arrows of the DFG define paths of data transfers between the elements, which are also seen as possible paths of error propagation. Arrows of the CFG represent control flow transitions between the elements, determining the order of their execution. The arrows of the CFG are weighted and show the probabilities of control transitions.

Faults can be activated in the elements during their execution and result in the occurrence of errors. The occurred errors propagate to other elements through the data transfer paths. Error propagation through the system comprises two aspects: error propagation between the elements and error propagation through the elements. The error propagation between the elements is determined by the DFG structure. The error propagation through the elements depends on the properties of a particular element. Each element is defined using three parameters: fault activation probability (FAP), error propagation probability (EPP), and error detection probability (EDP). The two graphs and these parameters of all of the elements describe an operational profile of the system. Figure 2 shows an example of a dual-graph error propagation model with a list of the element level parameters.

Figure 2 Examples for a control flow graph, a data flow graph, and a list of the probabilistic parameters of elements of a dual-graph error propagation model.

3 Discrete time Markov chain

Simultaneous probabilistic analysis of the control and data flow graphs forms the backbone of the introduced approach. The most suitable and comprehensive way to perform this analysis is the application of a discrete time Markov chain (DTMC) model on top of the EPM.

Figure 3 shows an example of a DTMC generated from the EPM depicted in Figure 2. Each state of the DTMC is characterized with four parameters: \(E^{\text{ref}}\) defines an element of the system that will be executed next, \(E^{\text{fa}}\) is a set of the elements where a fault has been activated, \(E^{\text{fp}}\) is the set of the elements where errors have been already propagated, and \(E^{\text{ed}}\) is a set of the elements where the errors have been detected.

Operation of a system starts with the initial state. A state change happens after each execution of a system element. The probability of each state change can be easily derived from the EPM. This process continues until one of the final states has been reached. A final state describes a possible, erroneous or error-free system execution scenario. This allows us to compute the probabilities of the system execution scenarios using the mathematical framework of the DTMC. The DTMC can be fully automatically generated using the data contained in the EPM. An extensive description of the EPM, DTMC, and all the algorithms are given in [1] and [2].
4 Case study

The example system in question is a caterpillar mobile robot, shown in Figure 4. As in a typical mechatronic system, it consists of mutually coupled hardware, software, and physical components. For the sake of the simplicity and transparency of the proposed approach, it has a simple basic functionality that avoids difficult path planning and motion control algorithms. However, an extension to more complex and thus more realistic system functionality is obvious and straightforward.

The task of this system is to move a robot from its current location to a target point. An abstraction of the caterpillar mobile robot system as a control system block diagram is shown in Figure 5. A controlled variable is a pose (position and orientation) of the robot. The control loop shown in this figure consists of four blocks: "control software", "servo motors", "robot dynamics and kinematics", and "navigation camera". The "control software", installed in a PC, plays the role of a controller. The "servo motors", connected with the caterpillars, serve as actuators. The "navigation camera" represents a sensor of the system. During each control-cycle, the control software determines the current location and orientation of the robot using the navigation camera that is located above the scene. After that it computes the speeds for the caterpillars and the length of a time interval for the servomotors activation. Next, the control software sends corresponding commands to the servomotors that correct the robot motion to go towards the target point. This procedure is repeated until the robot reaches the anticipated target point. A prototype of the robot is developed using a Lego NXT Mindstorms framework. Control software is implemented in Java and installed in a PC with Windows XP OS. A Bluetooth connection enables communication with the robot. A regular USB web-camera is used for observation of the scene.

The obtained data form a complete dual-graph error propagation model. After that we generate a DTMC using the obtained EPM. Next, the DTMC is applied for the model-based estimation of the probabilities of various scenarios of system execution. Real-world experiments with the faulty version of the system provide a statistical evaluation of the same probabilities. Finally, the experimental results and the model-based prediction are compared in order to prove the applicability of the approach.

It is obvious that the accuracy of the presented approach strongly depends on the estimation of the element level properties. This particular case study only evaluates the introduced system level model. Therefore, artificial fault injection experiments are used to determine fault activation, error propagation, and error detection probabilities. It gives a very precise evaluation of the element level parameters. Hence, we can estimate the accuracy of the system level model.

Figure 6 shows the task structure of the case study performed for evaluation of the error propagation analysis approach. It starts with system decomposition into several executable elements. After that faults are artificially injected into some of these elements in order to obtain a faulty version of this system. Examination of the control and data dependencies between the elements of the original system provides the structures of the control flow and data flow graphs. A number of test runs of the original system enable the determination of the control flow probabilities. Comparison of the elements of the original and faulty versions of the system allows the properties of individual elements to be estimated, such as the probabilities of fault activation, error propagation, and error detection.

Figure 5 A block diagram of a control loop of the "caterpillar mobile robot" reference system.

Figure 6 A structure of the case study.
The system is decomposed into seven executable elements as it shown in Figure 7. The element "Init" defines an initialization of the system. Execution of this element represents a strict software activity: A user is asked to click on the target point and on the two colored markers for calibration. Coordinates of the target point and RGB representations of the colors of the markers are the outputs of this element. This data is used as input of the next element "RcgPic". The user can initiate fresh initialization in case of an incorrect input that is represented by a control flow loop back to "Init". "RcgPic" stands for picture recognition. It incorporates several software and hardware activities in order to determine the pose of the robot. At first, it takes a snapshot using the navigation camera. After that it recognizes the positions of the front and back colored markers. Finally, it computes the position of the robot and compares it to the target position. The result of this comparison determines the control flow decision after the execution of this element. The system stops if the robot has reached the target point. If the color markers have not been found for some reason, the control is then transferred back to the element "Init". Otherwise, the execution continues with element "GuiOut". "GuiOut" is a software method that displays the result of the picture recognition to a graphic user interface (GUI).

After that, the element "CalcAngle" is executed. "CalcAngle" is also a strict software element that computes the angle between the robot's current orientation and the direction from the center of the robot to the target point. Depending on the value of this angle, either "RegStep" or "CorrStep" is executed. "RegStep" determines the speed and the time interval for straightforward movement of the robot, considering the distance from the robot to the target point. "CorrStep" computes two different speeds for the caterpillars in order to correct the robot's orientation.

The last element "NxtGo" is a mix of software, hardware, and physical activities. It generates a software command for the servo motors (software) and activates the servo motors during the defined time interval (hardware). The physical behavior of the robot (dynamics and kinematics) is also represented by this element. This element produces an abstract output "scene" that describes the environment and the robot after the motion and it goes on further as an abstract input to the element "RcgPic".

This system has been realized according to the UML representation as described above. Hence, the structure of the control and data flow graphs can be easily derived (see Figure 8). Also, a number of statistical experiments (about 500 runs) have been performed in order to obtain the probabilities of control flow transitions. These probabilities are also shown on the CFG in Figure 8. For instance, after the execution of "CalcAngle", "CorrStep" is executed in 69% and "RegStep" in 31% of the cases. Similarly, after the execution of "RcgPic," the system stops with the probability of 0.1147.

**Figure 7** The UML activity diagram of the "caterpillar mobile robot" reference system.

### 4.1 System level analysis

An activity UML diagram [3] of the mechatronic system under consideration is shown in Figure 7. This type of UML diagram is usually applied for workflow description. Rounded rectangles in the activity diagram represent so-called system activities, which are interconnected with the arrows (highlighted in black in Figure 7) that define the control flow of the system. Diamonds represent different types of control flow decisions. The solid black circle stands for the initial state of the workflow. The encircled black circle shows the final state. In addition to the control flow representation, the UML activity diagrams can also model data flow. Small colored boxes represent the required data inputs and the provided data outputs of the activities. Colored, thin arrows show the data flow paths. Because of such explicit representation of the control and data flows, this activity diagram is used as a base-line system model within a formal design process.

The UML diagrams are usually applied to describe only the software part of the systems. However, the activity UML diagrams can model very abstract entities. In this particular case, each activity, shown in Figure 7 represents not just the action of the robot control software, but also the hardware and even the physical motion of the robot. The described activities are considered to be actions of the executable elements of the system. Therefore, there is one-to-one correspondence between these activities and the elements of the dual-graph error propagation model of the "caterpillar mobile robot" reference system.
4.2 Element level analysis

A software implemented fault injection (SWIFI) is used in order to obtain a faulty version of the reference system. During the development, debugging, and testing of the original system, the information about introduced software and hardware bugs and design faults has been tracked and saved into a special fault list. Only the faults from this list have been injected in order to ensure the realistic behavior of the faulty version.

However, not every fault is suitable for this case study. First of all, it is only important to inject the faults that will result in erroneous data outputs. For instance, the faults that lead to immediate system failure or performance drop were not considered. Also, the faults can be classified as permanent faults and temporary faults [5]. A permanent fault is activated on each execution of an element meaning that the fault activation probability of this element equals to 1. It is not interesting for the case study, because such faults are always visible and do not require any probabilistic analysis. A temporary fault is activated only under specific execution conditions like specific values of input parameters, environmental impact, or a control flow decision. Our SWIFI experiment only considers temporary faults with low fault activation probabilities. Using this principle, the faults have been injected into all elements except "GuiOut" because it has no data output. Software bugs have been injected into "Init", "CalcAngle", "CorrStep", and "RegStep". In the elements "RegPic" and "NxtGo", hardware faults have been emulated by the software means. A complete list of injected faults is given in [2].

After the fault injection, a separate statistical analysis for each individual element was performed. Fault activation probabilities were estimated by executing the original and faulty versions of the elements in parallel. Error propagation probabilities were estimated using the erroneous and error-free inputs for the faulty elements. A software model of physical robot movement has been developed for the element "NxtGo" and used parallel to the real-world robot. The element "GuiOut" has a non-zero error detection probability. This probability has been determined using the defined difference between the erroneous robot pose and the expected one that is detectable by the naked eye.

4.3 Application of error propagation model

The described system level and element level analyses of the reference robot control system allow the construction of an error propagation model that is shown in Figure 8. The CFG has three forks that result in the existence of control flow cycles. The DFG also contains cycles. A DTMC has been generated using this EPM. The generation time was approximately one minute on a commodity laptop. The generated DTMC contains 30431 arcs and 11887 states: 1 initial state, 9944 transient states, and 1942 absorbing states. Thus, there are 1942 possible system execution scenarios and the probabilities of absorption in these final states define the probabilities of these scenarios. These probabilities have been computed using the methods described in [2]. The computation took about 30 seconds.

4.4 Experiments

The faulty and original versions of the system are run in parallel for statistical evaluation of system behavior. Both versions of the system work according to the same control flow. Control flow decisions for the original system have been provided by the faulty one. In contrast to the control flow, two separate data flows are kept: error-free data flow of the original system and data flow of the faulty system that can contain errors. The physical movement of the robot is controlled by the faulty system.

Scenario 1 | Scenario 2 | Scenario 3
--- | --- | ---
Model: 31.27 % | Model: 15.45 % | Model: 10.67 %
Experiments: 32.96 % | Experiments: 18.60 % | Experiments: 10.99 %

Figure 8 Control flow graph and data flow graph of the reference system with fault activation, error propagation, and error detection probabilities.

Figure 9 Three of the most frequent execution scenarios.
The original system uses a software-implemented model of robot kinematics. During the system operation, an original version and a faulty version of each element are executed using the erroneous data flow in order to detect fault activation. Likewise, a faulty version of an element is executed using the error-free and the erroneous data flows in order to detect error propagation through the element. In addition, an error detection mechanism is implemented in the element ‘GuiOut’. It is based on the defined delta between a pose of the real robot and the software model.

The described system has been run 446 times and has demonstrated 52 different execution scenarios. The first three results according to the statistical estimation of their probabilities are shown in Figure 9. An execution scenario without fault activations (Scenario 1) was the most common. Approximately each third run of the system was fault-free. The next two execution scenarios are as follows: (Scenario 2) fault activation in 'CorrStep', error propagation through 'NxtGo' and 'RegPic', error detection in 'GuiOut', and (Scenario 3) fault activation in 'CalcAngle', error propagation through 'CorrStep', 'NxtGo', and 'RegPic', and error detection in 'GuiOut'. The probabilities of these scenarios equal 0.1860 and 0.1099 respectively.

Twelve of the most frequent execution scenarios have been selected for evaluation. Each of them has happened at least six times during the experiments. This selection has been made because the normal probability density approximation works well as long as \( np > 5 \) and \( n(1 - p) > 5 \). Where \( n = 446 \) is a population size, and \( p = 1/n \) is a proportion of successes. Statistical estimation of other scenarios is considered unreliable because of the small number of their occurrence. The computational error of this estimation is less than 1% for the standard 95% confidence level.

Table 1 shows twelve execution scenarios of the selected system. The first column describes the scenarios in terms of fault activation (FA), error propagation (EP), and error detection (ED). The column ‘Exp.’ (Experiments) shows the probabilities of the scenarios obtained by the statistical evaluation of the real system. The column ‘Mod.’ (Model) shows the probabilities of the scenarios predicted by the error propagation model. The last column ‘Diff.’ shows the discrepancy.

The model prediction results and the experimental results are quite similar. The average difference is about 0.0178, and the maximum difference is 0.0322. However, only three of first execution scenarios have a probability greater than 0.1. The probabilities of the other scenarios are lower, and the relative difference between the experimental and model result becomes more and more significant. Scenarios with the relative difference higher than 50% of the predicted value are highlighted in the table. Nevertheless, these results prove that the model can accurately predict the most probable scenarios. At least it can distinguish between realistic and practically impossible execution scenarios. For example, the sorted list of predicted execution scenarios has been crosschecked with the experiments. The first predicted scenario that was not met among the experiments has the predicted probability of only 0.0074. It means that the likelihood that this execution scenario will be among our 446 experiments is only 3.5%. All other scenarios with probabilities greater that 0.0074 have been met at least once.

The estimated probabilities of the system execution scenarios play a valuable role in development of dependable systems. This information helps to define an effective testing strategy, to perform accurate reliability estimation, and to speed up error detection and fault localization processes. The obtained results allow us to conclude that the error propagation model and the DTMC approach are applicable for analysis of a typical mechatronic system.

5 Conclusion

Table 1 shows twelve execution scenarios of the selected system. The first column describes the scenarios in terms of fault activation (FA), error propagation (EP), and error detection (ED). The column ‘Exp.’ (Experiments) shows the probabilities of the scenarios obtained by the statistical evaluation of the real system. The column ‘Mod.’ (Model) shows the probabilities of the scenarios predicted by the error propagation model. The last column ‘Diff.’ shows the discrepancy.

The model prediction results and the experimental results are quite similar. The average difference is about 0.0178, and the maximum difference is 0.0322. However, only three of first execution scenarios have a probability greater than 0.1. The probabilities of the other scenarios are lower, and the relative difference between the experimental and model result becomes more and more significant. Scenarios with the relative difference higher than 50% of the predicted value are highlighted in the table. Nevertheless, these results prove that the model can accurately predict the most probable scenarios. At least it can distinguish between realistic and practically impossible execution scenarios. For example, the sorted list of predicted execution scenarios has been crosschecked with the experiments. The first predicted scenario that was not met among the experiments has the predicted probability of only 0.0074. It means that the likelihood that this execution scenario will be among our 446 experiments is only 3.5%. All other scenarios with probabilities greater that 0.0074 have been met at least once.

The estimated probabilities of the system execution scenarios play a valuable role in development of dependable systems. This information helps to define an effective testing strategy, to perform accurate reliability estimation, and to speed up error detection and fault localization processes. The obtained results allow us to conclude that the error propagation model and the DTMC approach are applicable for analysis of a typical mechatronic system.

6 Literature


This work has been supported by the Erasmus Mundus External Cooperation Window Programme of the European Union.