2013 IEEE 16th International Conference on Computational Science and Engineering

Modeling and Mitigation of Faults in Cyber-Physical Systems with Binary Sensors

Volkan Gunes, Steffen Peter, and Tony Givargis
Center for Embedded Computer Systems
University of California, Irvine, USA
{vgunes, st.peter, givargis}@uci.edu

***Abstract* — This paper presents an analysis framework for correct system operation (i.e. system success) of Cyber-Physical Systems (CPS) that deploy binary sensors with possible faults. We discuss potential faults in the interface part of such systems and address solutions for those faults in order to build dependable and reliable CPS applications. As a practical tool, we present a set of models in SIMULINK to help system designers extend simulations of general CPS applications that deploy binary sensor networks. We provide methodologies to add well-defined fault behaviors and offer assessment tools to measure the effects of possible faults on the overall system success. We demonstrate the feasibility of our contributions using a CPS application and explore various architectures for fault mitigation in a holistic design space exploration environment. With the ability to help system designers analyze and assess a non-trivial design space, the presented approach contributes to the design of fault-tolerant CPSs.**

Keywords—Cyber-Physical Systems, Fault Tolerance, Robust CPS, Binary Sensors

I. INTRODUCTION

Advances in digital electronics have led to a significant increase in the number of systems that couple the digital (cyber) world with the physical world via sensors and communication mediums (interface), namely what have become known as Cyber-Physical Systems (CPS). The application domains of CPS are, including but not limited to, industrial/factory automation, energy management, emergency response, aeronautics/air transportation, critical infrastructure, health care and medical applications, intelligent transportation, robotic for service and so on and so forth.

The design of CPS requires a significant amount of reasoning with respect to unique challenges, and complex performance and reliability requirements. Hence, dependability and reliability are two important requirements for CPS design. However, the potential component failures may prevent a CPS from performing dependable and reliable operations. Such failures may occur in the Cyber part, the Physical part or the Interface part and must be analyzed at the beginning stages of the design.

Failure is an inevitable incident for applications in the physical world. To ensure the satisfaction of the system outcome, potential failures must be handled to guarantee the expected behavior of the system. First of all, such failures need to be identified and categorized. Later, solutions for them should be addressed in a systematic way.

A variety of fault detection and mitigation techniques for faults in sensor systems have been proposed, ranging from component redundancy [1], over algorithm redundancy [2] to complex mathematical models [3]. However, these approaches are highly application-specific, as there is not a single perfect solution that suits each application or even application domain. An application developer has to perform an analysis on trade-offs between cost, design time, and mitigation of component failures. What is needed is a framework that allows system designers to test fault mitigation strategies even without strong background in fault theory.

This paper focuses on the sensor interfaces of the CPS and addresses the following questions:

- Which faults can occur in the sensor interfaces of the CPS using binary sensor networks?
- How can we cope with those faults?

The first question leads to the task of understanding and modeling possible faults, while the second question concerns fault tolerance strategies.

The contributions of this paper are:

- Fault characterization of binary sensor networks used in CPS applications.
- Providing a system framework for a CPS use-case application and developing reusable system models in the MATLAB/SIMULINK environment.
- Exploring the design space of the CPS use-case application for fault mitigation

In this study, we discuss the analysis of system success for a CPS application, namely the Falling Ball Example (FBE) [4], using binary motion sensors with fault probabilities and impact of the component faults, which may occur in the interface part of CPS, on the system outcome. The rest of this paper is organized as follows. Motivation, preliminaries, and related work are presented in Section II. A MATLAB/SIMULINK framework is introduced in Section III. Our demonstrative example is introduced in Section IV. Fault mitigation techniques are introduced in Section V. Experiments and evaluation are discussed in Section VI. Conclusion is given in Section VII.

978-0-7695-5096-1/13 $31.00 © 2013 IEEE
DOI 10.1109/CSE.2013.83

IEEE computer society

II. MOTIVATION, PRELIMINARIES, AND RELATED WORK

Cyber-Physical Systems rely on the tight integration of the cyber and physical systems. In this tight integration, the sensors play an important role. CPS applications deploy a wide range of sensors as interfaces between the physical world and the cyber systems. Binary sensors are among those sensors. Examples of binary sensors include motion detectors, break beam sensors, pressure mats, etc. Binary motion sensors are used in CPS applications to identify location, velocity, or trajectory of the physical target. Various studies have addressed deploying binary sensors for different applications.

An intruder tracking system is proposed in [5] to automatically track and capture photos of an intruder via a camera. In this study, the authors consider the problem of automated position estimation using a wireless network of inexpensive binary motion sensors to track an intruder. A high level robot behavior is evaluated in [6] under the presence of erroneous sensors. The authors do probability based analysis to see if the controllers satisfy a set of high level specifications for robotic behaviors.

A trustworthy alarm system, namely Tru-Alarm, is proposed in [7] to detect intrusion objects in a bounded environment. Tru-Alarm is responsible for filtering meaningful information from a large volume of data including erroneous ones caused by sensor failures. A target tracking problem with wireless binary sensor networks is studied in [8]. The authors propose a distributed tracking algorithm that estimates target's location, velocity and trajectory under the imperfect binary sensor conditions.

A real-time user tracking system, namely FindingHuMo (Finding Human Motion), is proposed in [9]. The system can accommodate tracking of multiple (unknown and variable number of) users in any crowded Smart Environments. The system can detect and isolate motion trajectories of individual users via anonymous (not user-specific) binary motion data stream.

Besides the above-mentioned applications, binary sensor networks can be deployed in Smart Homes/Buildings. For example, a garage door can be controlled via a real-time location, velocity and trajectory detection of a car. In this scenario, false-positive faults may lead to safety concerns. Another example might be the alarm system of a Smart Building deploying break beam/motion detection sensors. In this scenario, false-positive faults may cause unnecessary panic in the environment. Therefore, false-positive and false-negative faults must be addressed and handled to guarantee trustworthy and reliable operations of CPS applications using sensor networks.

In order to make our discussion more clear, we need to clarify some terminology used throughout this paper. A binary sensor is a type of sensor which can return only a one-bit information in response to an occurrence of an event (e.g. a physical phenomenon).

A failure is the lack of success in the outcome of a system. Since definition of success is application-specific, system failure and success are defined by the designer according to the mission to be realized.

A fault is an error that occurs in a component of a system. Faults on binary sensors can be false-negative or false positive. A false-negative (FN) fault is a type of error, which indicates that a binary sensor component failed to detect an event even though the event occurred. A false-positive (FP) fault is a type of error which indicates that a binary sensor component fired even though an event did not occur.

In the real world, components do not fail frequently. Hence, faults are characterized statistically (i.e. using some probability model). Faults may occur over different domains (e.g. time domain or event domain). In our study, a false-positive fault is measured over the time domain. The following equation defines the occurrence probability of a false-positive fault.

$$P_{FP} = \frac{\Delta faults_{FP}}{\Delta t} \quad (1)$$

Where P_{FP} refers to the probability of a false-positive, $\Delta faults_{FP}$ refers to the number of false-positive faults, and Δt refers to a window of time.

On the other hand, a false-negative fault is measured over a number of real events. The following equation defines the occurrence probability of a false-negative fault.

$$P_{FN} = \frac{\Delta faults_{FN}}{\Delta event} \quad (2)$$

Where P_{FN} refers to the probability of a false-negative, $\Delta faults_{FN}$ refers to the number of false-negative faults, and $\Delta event$ refers to the number of events.

Various studies have addressed detection and mitigation techniques for faults in sensor systems. The majority of related work focuses on techniques for sensor readings of non-binary measurements in control loops. Kalman filters [10] or simplex architectures [3] are popular approaches for such systems. However, such approaches do not address binary sensors and sporadic activity patterns.

A broadly used approach to increase robustness of binary sensor systems is modular redundancy [1], in which redundant sensors are deployed to detect and mitigate sensor faults. Triple modular redundancy (TMR) has been applied in many avionic systems [11]. In the area of modular redundancy, voting strategies [12], i.e. mechanisms to identify faulty sensor readings, are still an active field of research.

To reduce the need for a high number of redundant physical sensors, the concept of algorithmic redundancy has been proposed [13]. As an example, model-based prognostics [2] can be applied to cope with missing or faulty sensor information. In this case, domain-specific knowledge

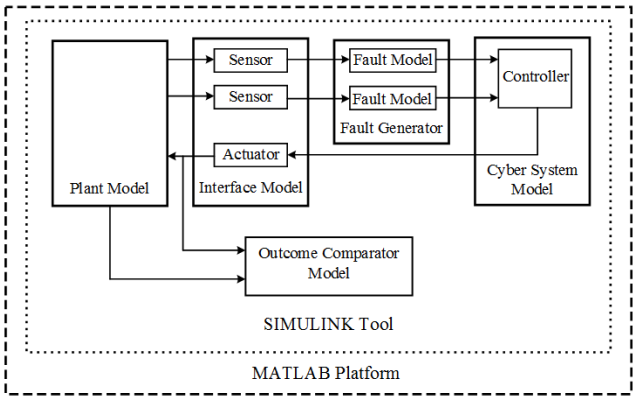


Figure 1. MATLAB/SIMULINK Framework.

such as physical models or plant models are used to compensate for faults and predict the correct values.

Isermann [14] provides a survey on these approaches and shows their application in automotive industry. However, no single solution exists that suits all faults and system requirements in a typical application. This is caused by contradicting goals for the system, such as cost, mitigation of false-negative faults, mitigation of false-positive faults, or system performance. Managing these trade-offs is a major concern of the framework that we present next.

III. A MATLAB/SIMULINK SIMULATION FRAMEWORK

Model based design is a powerful design technique for CPS [15]. It allows creating models for each system component and verifying system behavior during each and every design stages. SIMULINK is a tool used to model and simulate the behavior of dynamic systems (those governed by differential equations). It is integrated with MATLAB, enabling the designer to incorporate MATLAB algorithms into SIMULINK models and export simulation results to MATLAB for further analysis and visualization [16].

A typical CPS simulation consists of a physical model, a cyber model, and an interface model. The general idea of our approach is to add a fault generator model and an outcome comparator model to this typical CPS simulation framework. The new components with typical CPS models are shown in Fig. 1. In addition to those new components, the whole simulation framework incorporates the physical system model, the interface model (including sensors and actuators), and the cyber system model.

The physical system model represents the physical phenomenon which needs to be monitored or controlled. The interface model represents the binary sensors and actuators that are responsible for data gathering and control actions, respectively. The cyber system model comprises the controller and other logic units. The controller unit performs control computations and outputs control action according to the result.

The outcome comparator evaluates the accuracy/ satisfaction of the system outcome after the control action. The control signal and information from the physical system is fed into the comparator. The comparator performs

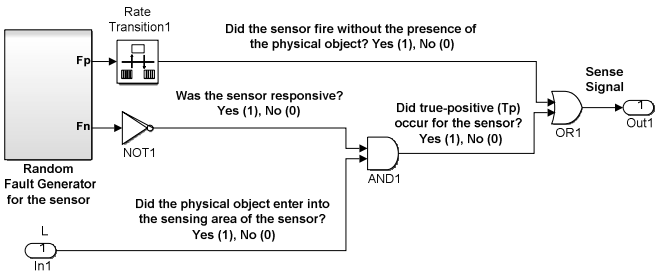


Figure 2. FP and FN fault generator model for one binary sensor.

comparison and evaluates correct functionalities. Success of the system is evaluated according to the definition of system success provided by the system designer. This definition is usually application-specific.

The fault generator model represents potential faults in binary sensors. We incorporated fault generation functionality into the CPS simulation framework to analyze the impact of false-negative and false-positive faults on the system outcome. Fig. 2 shows the fault generator model comprising both the false-negative (FN) and false-positive (FP) fault generation functionalities.

A major challenge of the fault generator is the fact that a false-negative fault is meaningful only when the sensor is supposed to detect an event, while a false-positive fault can occur in any window of time. This fact was also expressed in (1) and (2) in the previous section. In the following, we apply these equations to express the behavior, input and output signals of the fault generator model.

Each binary sensor *i* of the system has its individual fault generator model. Each time the model is evaluated in SIMULINK, the following variables are computed:

$$L_i = \begin{cases} 1 & if\ the\ physical\ object\ is\ in\ the\ sensing\ area\ of\ sensor\ i \\ 0 & otherwise \end{cases}$$

$$Fn_i = \begin{cases} 1 & if\ (L_i is\ true)\ and\ (sensor\ i\ doesn't\ fire) \\ 0 & otherwise \end{cases}$$

$$Tp_i = \begin{cases} 1 & if\ (L_i is\ true)\ and\ (Fn_i is\ not\ true) \\ 0 & otherwise \end{cases}$$

$$Fp_i = \begin{cases} 1 & if\ (L_i is\ not\ true) and\ (sensor\ i\ fires) \\ 0 & otherwise \end{cases}$$

Where L_i indicates the location of physical phenomenon according to the sensing area of the sensor i, Fn_i indicates the occurrence of false-negative fault for the sensor i, Tp_i indicates the expected behavior of the sensor i when the physical phenomenon is in its sensing area, and Fp_i indicates the occurrence of false-positive fault for the sensor i.

The above mentioned variable formulations show the signal behaviors in the fault generator model as shown in Fig. 2. L_i variable is provided to the fault generator model through In1 port. Fn_i and Fp_i are the outputs of random fault generator module. Tp_i refers to the output of AND1 logical operator as shown in Fig. 2. The fault generator model includes fault probabilities for false-negative and false-positive faults as a function of the model parameters.

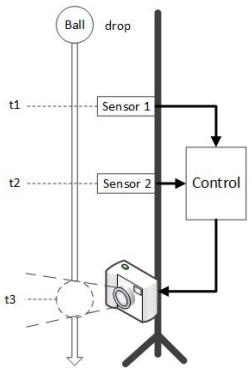


Figure 3. The Falling Ball Example Baseline Architecture.

Our framework can easily be extended to larger CPS applications using binary sensors. The plant model, the interface model, and the cyber system model (i.e. control algorithm) may need modifications according to the system definition of the relevant CPS application since they are application-specific. However, without any modification, the fault generator model can be integrated into larger CPS simulation frameworks since binary sensors show the same behavior (i.e. the provision of one-bit information in response to an occurrence of an event). Model based design in SIMULINK facilitates this integration. The fault generator model should be placed between the sensor model and the cyber system model. The outcome comparator may need some modification as well since it considers inputs from the plant model and the actuator model for the system evaluation as shown in Fig. 1.

IV. Our Demonstrative Example

We investigate an instance of a lightweight CPS application, namely the Falling Ball Example (FBE) [1], suitable to be applied in the education of CPS engineers. The baseline architecture for FBE is shown in Fig. 3, which includes two binary motion sensors, a camera (as an actuator), a controller, a ball, and necessary appliances for the setup. The controller interprets sensors' views of the movement of the ball, which is in a free-fall from a certain height, to estimate when it reaches height of the actuator (camera) in its trajectory. Then, a course of action is taken by the controller to trigger the camera. The camera takes a picture of the ball, when it reaches the height of the camera. The time for camera to act is predicted based on the information which comes from motion sensors mounted above the camera. The controller utilizes the feed-forward control strategy, which is regarded as anticipation of the relationship between the physical system and its environment and taking a necessary course of action according to this anticipation.

Algorithm 1. The Falling Ball control program.

1. wait for a detect signal from sensor 1,
2. after receiving the signal from sensor 1, record the time (t_1),
3. wait for a detect signal from sensor 2,
4. after receiving the signal from sensor 2, record the time (t_2),
5. considering t_1 and t_2, compute the expected time (t_3) for actuator to act,
6. wait until t_3, then activate camera while compensating for the expected delay time in the system.

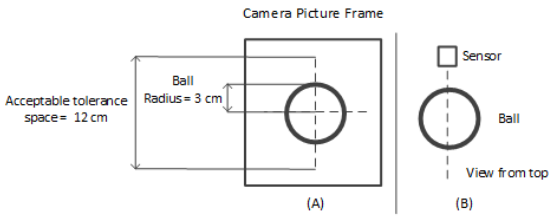


Figure 4. The ball placement: A) Ideal placement of the ball in the frame of the camera B) Correct alignment of the ball in its trajectory.

It is necessary to express the physical system as a part of the control algorithm with regular expressions. In case of a falling ball, we can apply the basic free fall equations. Using the known height of both sensors (h_1 and h_2), the value of gravity (a), and the time when the sensors triggered (t_1 and t_2), we can compute the speed of the ball at sensor 2, i.e. v_2, with the following equation:

$$v_2 = \frac{h_1 - h_2}{t_2 - t_1} + a\frac{t_2 - t_1}{2} \quad (3)$$

We can use this v_2 to compute the expected time (t_3) at the height of camera (h_3) with the following equation:

$$t_3 = \frac{at_2 - v_2 + \sqrt{{v_2}^2 + 2ah_2 - 2ah_3}}{a} \quad (4)$$

Computing (4) is the most complex step (step 5) in the small control program, which is shown as Algorithm 1.

Definition of success for FBE is that the falling ball should be in the middle of the snapshot with an acceptable tolerance space. The Percentage of Success (PoS) for FBE can be defined as follows:

$$PoS = \frac{Number\ of\ Successful\ Shots}{Total\ Number\ of\ Shots} * 100 \quad (5)$$

Successful shot denotes the placement of the ball in the center of the camera picture frame with a certain tolerance as shown in Fig. 4. The accepted tolerance space is 12 cm in the frame of a snapshot.

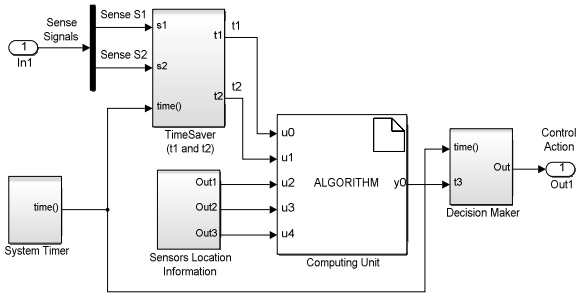


Figure 5. Controller subsystem inside Cyber system model.

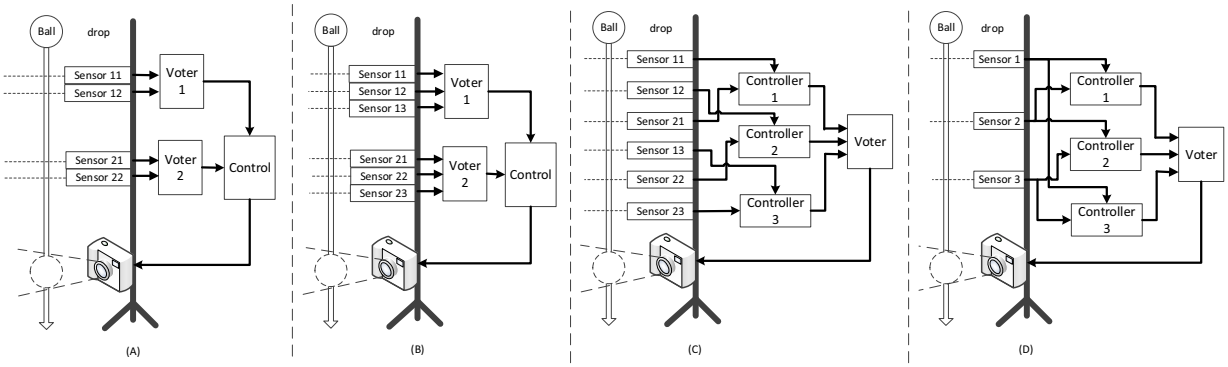


Figure 6. Fault mitigation techniques for the FBE (A) Dual Sensor Redundancy (SR4), B) Triple Sensor Redundancy (SR6), (C) Triple Algorithm Redundancy with 6 sensors (SR6CR3), D) Triple Algorithm Redundancy with 3 sensors (SR3CR3).

We developed the baseline architecture in SIMULINK and our cyber system model is shown in Fig. 5. The system timer unit is a counter that increment based on the sampling frequency as in microcontroller timers that increment based on the clock cycle. The timesaver unit is responsible for saving the time when the sense signal becomes true for a sensor. The timesaver provides t_1 and t_2 to the computing unit. The computing unit is responsible for processing the data provided by other units (e.g. t_1, t_2, and sensor information). It uses (4) to compute t_3. System time and t3 are fed into the decision maker unit. The decision maker unit is responsible for taking necessary control action in due time (i.e. t_3).

We incorporated sensor fault generation and system outcome comparison functionalities into the baseline architecture in order to assess the system success under faulty sensor behaviors. Faulty sensor behaviors affect the success of the baseline architecture adversely for sure. So that is the motivation for us to investigate fault mitigation techniques which we address in the subsequent section.

V. FAULT MITIGATION TECHNIQUES

In case sensor faults occur, false-negative and false-positive faults affect the system success of the baseline architecture adversely. In this section, we describe a set of strategies to alleviate the effects of those faults on the system success.

A. Component Redundancy

As a solution to those component faults, namely false-negative and false-positive faults, the traditional approach is to apply sensor redundancy. Dual sensor redundancy is shown in Fig. 6 (A). It is denoted as SR4. It comprises four sensors, two voters, and one controller. Each voter is responsible for applying selection logic on the sensor signals (i.e. AND or OR). SR4 with AND logic for voting is denoted as $SR4_1$. SR4 with OR logic for voting is denoted as $SR4_2$. Each voter either applies logical AND or logical OR operations to both sensor signals. In this scenario, AND operation means that both sensors should agree on the occurrence of the event. The system misses the event in case one sensor has false-negative fault. OR operation means that if only one sensor detects the event, that's enough. The sense signal will be forwarded to the subsequent system units, as well as false-positive ones. The output of $Voter_1$ represents t_1 and the output of $Voter_2$ represents t_2.

Triple sensor redundancy approach is shown in Fig. 6 (B). It is denoted as SR6. It comprises six sensors, two voters, and one controller. The voter mechanisms ensure that at least two of three sensors agree on the sense signals. Agreement of sensors on the presence of the physical phenomenon (i.e. the falling ball) will prevent sensor component faults (i.e. false-positive and false-negative) to affect the system outcome adversely.

B. Algorithm Redundancy

Besides component redundancy technique, another technique to mitigate sensor component faults is control algorithm redundancy. Triple algorithm redundancy approach is shown in Fig. 6 (C) and (D) with the use of different number of sensors. Triple algorithm redundancy with six sensors is denoted as SR6CR3. It comprises six sensors, one voter, and three controllers as shown in Fig. 6 (C). Triple algorithm redundancy with three sensors is denoted as SR3CR3. It comprises three sensors, one voter, and three controllers as shown in Fig. 6 (D). Both architectures deploy three controller units in the cyber part. Each controller waits for data from different sensors located in the trajectory of the falling ball. The output of the each controller represents an instance of t_3. The voter mechanism ensures that at least two of three controllers agree on the value of t_3. When at least two controllers provide the same value of t_3 than that instance of t_3 is considered to be correct.

VI. EXPERIMENTS AND EVALUATION

We developed system models for the above mentioned architectures in SIMULINK in order to make a comparison of these architectures and an assessment on the design space exploration of our use-case application. In the system simulation, we followed an episode scenario where falling ball activity occurs randomly during a time interval. The time interval is 2 seconds. We defined false-negative fault as a random binary number generated once throughout the

TABLE I. SIMULINK Model Configurations.

Simulation Time (t_{sim})	2 sec
Simulation Time of the Plant (t_{plant})	0.6 sec
Simulation start time for the Plant (t_{start})	random number that ranges between 0 and 1.4 sec ($t_{sim} - t_{plant}$)
Simulation stop time for the Plant (t_{stop})	$t_{start} + t_{plant}$
Sampling Time of FN fault	2 sec
The occurrence probability of FN fault	ranges from 0 to 0.05 with step size of 0.005
Sampling Time of FP fault	200 msec
The occurrence probability of FP fault	ranges from 0 to 0.05 with step size of 0.005

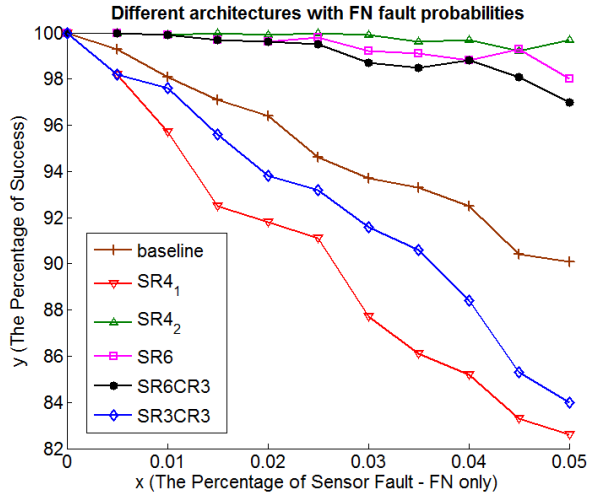
simulation. Because of that, sampling time of FN fault is the same as system simulation time (i.e. 2 seconds). We defined false-positive fault as a random binary number generated in a certain sampling time (i.e. 200 miliseconds). We considered fault probability range from 0 to %5. Since false-positive faults are generated in the given probability range and sampling time, the work is left to the cyber system to find out trustworthy results. The SIMULINK model configuration parameters for the simulation are summarized in Table 1.

A. Simulation Results

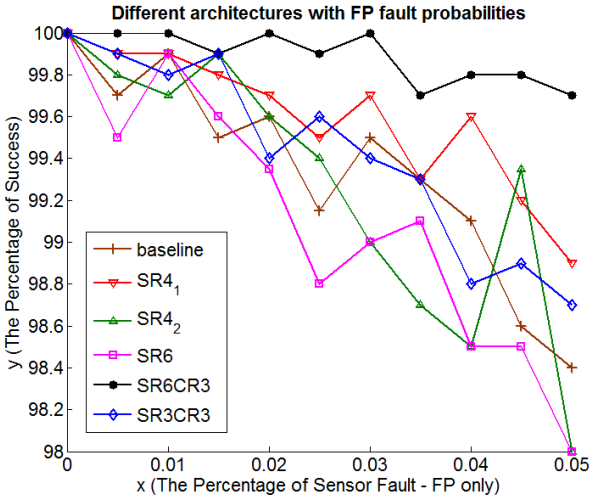
We compared six architectures, namely baseline architecture, dual sensor redundancy with AND logic denoted as $SR4_1$, dual sensor redundancy with OR logic denoted as $SR4_2$, triple sensor redundancy denoted as SR6, triple algorithm redundancy with six sensors denoted as SR6CR3, and triple algorithm redundancy with three sensors denoted as SR3CR3, in our experiments to see better approach for achieving more dependable and reliable system outcome. The result of simulations in SIMULINK were exported to MATLAB workspace and visualized through a MATLAB script for the analysis of the framework, as shown in Fig. 7.

Fig. 7 shows the system success rates for the given fault probabilities. X-axis represents the probability percentage of fault occurrence in binary sensors. Y-axis represents the percentage of success for the CPS application as formulated in (5). Fig. 7 (A) represents the system success rate in case only false-negative fault occurred. The probability of false-positive fault occurrence is assumed zero. $SR4_2$ shows the best performance in this category. It provides about %10 improvement in %5 fault probability (only FN) compare to the baseline architecture.

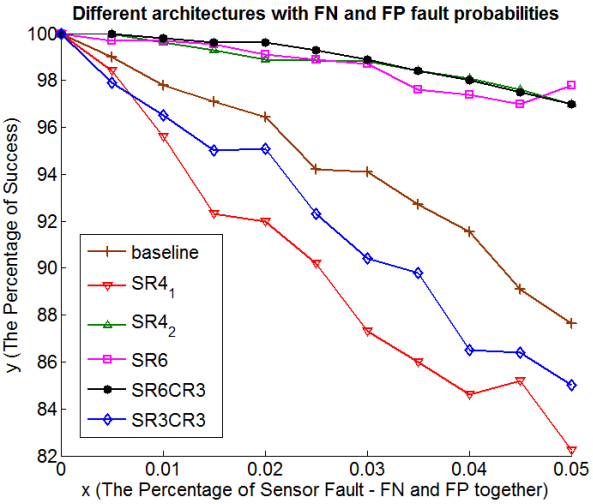
Fig. 7 (B) represents the system success rate in case only false-positive fault occurred. The probability of false-negative fault occurrence is assumed zero. SR6CR3 shows the best performance in this category. It provides about %2 improvement in %5 fault probability (only FP) compare to the baseline architecture.



(A)



(B)



(C)

Figure 7. Success rate of different architectures with false-positive and false-negative error rates: A) False-Negative only, B) False-Positive only, C) False-Negative and False-Positive together.

TABLE II. Ranking of the architectures according to three criteria (cost, FN, and FP).

Architecture	Cost	Only FN	Only FP	FN + FP
Baseline	1	4	4	4
SR41	3	6	2	6
SR42	3	1	5	1
SR6	4	2	6	3
SR6CR3	5	3	1	2
SR3CR3	2	5	3	5

Fig. 7 (C) shows the system success rate in case both false-positive and false-negative faults are taken into account. In this category, $SR4_2$, SR6, and SR6CR3 show better performance than the others. They provide about %9 percent improvement in %5 fault probability (i.e. for both FN and FP) compare to baseline. Each one of FN and FP has %5 occurrence probability.

Fig. 7 (A) and Fig. 7 (C) are very similar because success rate in Fig. 7 (B) ranges in a small number (i.e. between %98 and %100). As shown in Fig. 7 (C), success rate significantly decrease with the increasing probability of false-negative (FN) and false-positive (FP) occurrence for the baseline, $SR4_1$, and SR3CR3 architectures. When we compare the three plots, we can conclude that the selection of architecture depends on the significance of false-negative and false-positive faults on the system success. As we mentioned earlier, the definition of system success is application-specific. Fig. 7 indicates that some architecture may be best choice for mitigating false-negative fault but may not be the best choice in false-positive fault mitigation (e.g. $SR4_2$). Furthermore, some architecture may be the best choice for mitigating false-positive fault but may not be the best choice in false-negative fault mitigation (e.g. SR6CR3). The design space exploration will end up with different choices if we provide the objective function with weighted false-negative and false-positive faults. This trade-off needs to be considered by the designer according to the system outcome requirements.

B. *Design Space Exploration*

The data presented so far shows the general variability of the effect of fault mitigation techniques in CPS. However, system success is not the only metric that is important in the design of CPS. In addition, we have to consider aspects such as cost, energy consumption etc. before making a design decision. In this section, we compare the system success results from the previous section with the cost for sensors to set up the required redundancy. The idea is to apply a Pareto analysis to rule out system setups that cost more and provide less system success. We will see that the result of design space exploration differs depending on our system goal.

To conduct the analysis, we ranked the architectures according to three criteria, namely cost, FN, and FP as shown in Table 2. The cost metric is derived from the number of sensors and controllers; the system success is evaluated according to Fig. 7. In the ranking, a 1 indicates the best choice and a 6 indicates the worst choice. According to that ranking, we draw the plots in Fig. 8 in order to find out Pareto-optimal solutions in the design space. An ideal system would be in the lower left corner. For example, if we compare two systems (circles) in the graph, e.g. S_1 and S_2, while S_2 is more expensive (further right) and has less system success (further up) than S_1, we can rule out S_2 from the design space. The results of this analysis are the followings:

- If we consider cost and only FN faults, Pareto-optimal solutions contain {Baseline, $SR4_2$} as shown in Fig. 8 (A).

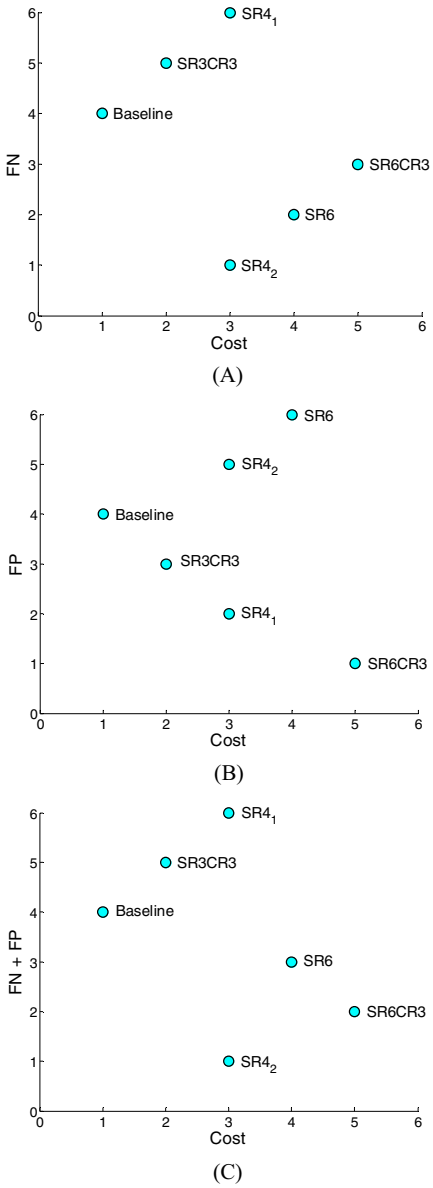


Figure 8. Design space of target architectures:
A) Cost/False-Negative (FN) trade-off
B) Cost/False-Positive (FP) trade-off
C) Cost/False-Negative (FN) and False-Positive (FP) trade-off

- If we consider cost and only FP faults, Pareto-optimal solutions contain {Baseline, SR3CR3, $SR4_1$, SR6CR3} as shown in Fig. 8 (B).
- If we combine both faults according to the parameters set in TABLE , Pareto-optimal solutions contain {Baseline, $SR4_2$} as shown in Fig. 8 (C).

Indeed, if we change parameters or the system goals, the result of design space exploration will change. As our results indicate, this change is partly non-intuitive and non-trivial. In the example, the small design space for the combined faults was surprising for us. Initially, we expected a direct trade-off between cost and system success. The results, however, show that we can remove 67% of the design options. These results underline the importance of an evaluation framework for simulations as it has been proposed in this paper.

VII. CONCLUSIONS

In this paper, we outlined the design space of a fault-tolerant CPS application to ensure reliable operations in case of sensor faults which hinder system success. As a use case, we compared a range of architectures which add redundancy to mitigate faults. Our results indicate that even for small examples the design space is complex and the answer for the best system setup is non-trivial, in particular when taking into account the aspects such as cost or required precision. Therefore, it is mandatory for a system designer to be able to analyze this design space efficiently. This requires the ability to set up systems with a well-defined fault model and flexible evaluation logic. A SIMULINK framework consisting of components that provide this functionality have been introduced in the first part of this paper. With the described use case and the result of design space exploration analysis, we could demonstrate the effectiveness of this framework to simulate faults and analyze their effects.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under NSF grant number 1016789 and 1136146.

REFERENCES

[1] E. P. Kim and N. R. Shanbhag, “Soft n-modular redundancy,” Computers, IEEE Transactions on, vol. 61, no. 3, pp. 323–336, 2012.

[2] M. Daigle and K. Goebel, “Model-based prognostics under limited sensing,” in Aerospace Conference, 2010 IEEE. IEEE, 2010, pp. 1–12.

[3] X. Wang, N. Hovakimyan, and L. Sha, “L1simplex: Fault-tolerant control of cyber-physical systems,” in in International Conference on Cyber-Physical Systems (ICCPS13), Cyber-Physical Systems Week, 2013.

[4] S. Peter, F. Vahid, D. D. Gajski, T. Givargis, "A Ball Goes to School - Our Experiences from a CPS Design Experiment", in 1st Workshop on Cyber-Physical Systems Education (CPS-Ed), April 2013.

[5] J. Schiff and K. Goldberg, “Automated intruder tracking using particle filtering and a network of binary motion sensors,” in Proc. of IEEE CASE, pp. 580-587, Oct. 2006.

[6] B. Johnson and H. Kress-Gazit, “Probabilistic analysis of correctness of high-level robot behavior with sensor error”, in Proc. of Robotics: Science and Systems (RSS), June 2011.

[7] L. Tang, X. Yu, S. Kim, J. Han, C. Hung and W Peng. “Tru-Alarm: Trustworthiness Analysis of Sensor Networks in Cyber-Physical System”, in Proc. of IEEE International Conference on Data Mining (ICDM), 2010.

[8] Z. Wang, E. Bulut, and B. K. Szymanski, “A distributed cooperative target tracking with binary sensor networks,” IEEE International Conference on Communications Workshops, ICC’08, pp 306-310, May 2008.

[9] D. De, W.-Z. Song, M. Xu, C.-L. Wang, D. Cook, and X. Huo, “Findinghumo: Real-time tracking of motion trajectories from anonymous binary sensing in smart environments”, in Proc. of the 32nd IEEE International Conference on Distributed Computing Systems, ICDCS ’12, pages 163-172, 2012.

[10] D. Willner, C. Chang, and K. Dunn, “Kalman filter algorithms for a multi-sensor system”, in Decision and Control including the 15th Symposium on Adaptive Processes, 1976 pp. 570–574.

[11] Yeh, Y. C. "Triple-triple redundant 777 primary flight computer." *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*. Vol. 1. IEEE, 1996.

[12] D. M. Blough and G. F. Sullivan, “A comparison of voting strategies for fault-tolerant distributed systems,” in Reliable Distributed Systems., Ninth Symposium on. IEEE, 1990, pp. 136–145.

[13] S. Neema, T. Bapty, S. Shetty, and S. Nordstrom, “Autonomic fault mitigation in embedded systems,” Engineering Applications of Artificial Intelligence, vol. 17, no. 7, pp. 711–725, 2004.

[14] Isermann, Rolf; Schwarz, R.; Stolzl, S., "Fault-tolerant drive-by-wire systems," *Control Systems, IEEE* , vol.22, no.5, pp.64,81, Oct 2002

[15] J.C. Jensen, D. Chang, E.A. Lee, "A Model-Based Design Methodology for Cyber-Physical Systems", First IEEE Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy), July 2011.

[16] SIMULINK in R2013a, Mathworks. [Online]. Available: http://www.mathworks.com/products/simulink/