Synthesis of a Context-Aware Safety Monitor for an Artificial Pancreas System

Bulbul Ahmed

A Dissertation Presented to the Graduate Faculty

of the University of Virginia in Candidacy for the Degree of

Master of Science

Department of Electrical & Computer Engineering

University of Virginia

November, 2019

Abstract

Rapid advances in sensing and computing technology have led to the proliferation of medical cyber-physical systems (CPS) in personalized and clinical settings. However, the increasing device complexity, shrinking technology sizes, and shorter time to market have resulted in major challenges in ensuring the reliability, safety, and security of medical devices. This research presents a hybrid model- and data-driven approach for the synthesis of safety monitors that can continuously detect faults and mitigate hazards in medical CPS. The synthesis process starts with the identification of safety requirements using systems-theoretic (STPA) hazard analysis and specification of temporal logic rules for the detection of unsafe system context. The extracted temporal logic is further refined using the closed-loop simulation of the controller with a dynamic patient model. The final synthesized monitor is integrated with the target control software as a wrapper that only has access to the input-output interface (sensor and actuator values) and performs real-time execution of logic rules along with a simulation of the patient model. We demonstrate the effectiveness of our approach using a case study of a closed-loop artificial pancreas system (APS), consisting of an APS control software (OpenAPS) and a patient glucose simulator (Glucosym). The performance of the monitor is assessed in terms of timely and accurate detection of potentially unsafe controller commands due to hardware errors. Results show that the proposed monitor can correctly generate alerts and shows a significant increase in F1 score (up to 23%) with a trade-off of 33-minute decrease in reaction time but having a lower number of false positives, 175 compared to 649 for a baseline.

Keywords— safety, resilience, run-time verification, error detection, hazard analysis, cyber-physical system, medical device

Acknowledgments

This dissertation would not have been possible without the help of so many people in different ways. I would first like to thank my advisor Dr. Homa Alemzadeh for her continuous guidance, encouragement, and advice throughout the last two years. I would also like to express my heartfelt gratitude to Dr. James H. Aylor for providing me the opportunity to work on this project.

I would like to express my deepest appreciation to my final defense committee chair, Dr. Joanne Bechta Dugan for providing me the guidance and support from the very beginning of my journey at the University of Virginia. I would also like to thank my defense committee member, Dr. Lu Feng for her time.

I gratefully acknowledge the support from the U.S. National Science Foundation (NSF) (Award no. 1748737) for the research contained in this dissertation. Special thanks to Xugui Zhou for his invaluable support, interesting discussion in research and helpful insights on research.

Finally, and most importantly, I thank my parents for supporting me since my childhood and my wife for always encouraging me.

Contents

1 Introduction **1**

1.1 The Motivation 1

1.2 Objectives 2

1.3 Contributions 4

1.4 The Organization 5

2 Related Work **7**

2.1 Runtime Verification 7

2.2 Safety and Security Monitoring in CPS 8

2.3 Fault Detection and Tolerance in Medical Devices 10

3 Technical Background and Case Study **13**

3.1 Safety Context in Cyber-Physical Systems 13

3.2 Case Study 15

3.2.1 Artificial Pancreas 15

3.3 Development of Artificial Pancreas 18

3.3.1 OpenAPS 21

3.3.2 GlucoSym 23

4 Approach **25**

4.1 Design Methodology 25
4.2 Systems Theoretic Hazard Analysis 27
4.3 Closed-Loop System Simulation 32
4.4 Parameter Extraction from Temporal Logic 33
4.5 Automation of Context Table and LTL Rule Generation 34

5 Evaluation **39**

5.1 Experimental Setup 39
5.2 Fault Injection 40
5.3 Hazard Detection 41
5.4 Context-Aware Monitoring 44
5.5 Evaluation Metrics 44
5.6 Experimental Results 46

6 Conclusion **53**

6.1 Conclusion 53

List of Figures

1-1 System With The Safety Monitor 3

3-1 Artificial Pancreas. 14
3-2 Monitor does not consider the action of controller 15
3-3 Continuous Glucose Monitor (CGM) 16
3-4 Insulin Pump 17
3-5 Closed Loop System with OpenAPS Controller and GlucoSym. 18
3-6 Communication among OpenAPS, GlucoSym and Python Script 19
3-7 Glucose level of patient without insulin feedback. 20
3-8 Glucose level of patient with insulin feedback. 20
3-9 Recommended insulin rate. 21
3-10 OpenAPS Architecture and Input Output. 21
3-11 OpenAPS Algorithm 23
3-13 Sample parameters of GlucoSym simulator. 23
3-12 Mathematical equation used in GlucoSym simulator [1]. 24

4-1 Steps for Synthesis of Safety Monitor 26
4-2 Context Aware Monitor integrated as a Wrapper with the OpenAPS Control Software 27
4-3 Control Structure with the OpenAPS Controller's Process Model 30
4-4 Specification to LTL rules 36

5-1 Hazardous Data Points and Detection Results 42
5-2 Fault-free simulation 43
5-3 True Positive 46
5-4 False Positive 47
5-5 True Negative 48
5-6 False Negative 49
5-7 Average Fault Propagation Times for Different Scenarios (Unit: minutes) 52

List of Tables

3.1 Input Parameters of OpenAPS. 22
3.2 Input Parameters of GlucoSym. 24

4.1 STPA Context Table 33
4.2 Patient Parameter 35
4.3 Temporal Logic Rules Extracted from STPA Context Table and System Simulation 37

5.1 Rules of Baseline monitor 44
5.2 Summary for monitor performance in fault injection and fault-free cases 50
5.3 Hazard Number and Coverage for Different Fault Scenarios and Types 51

Chapter 1

Introduction

1.1 The Motivation

Rapid advances in sensing, computing, and low-power highly-integrated circuit technology have led to increasing deployment of the medical cyber-physical systems (CPS) in various personalized and clinical settings. The ability of these devices to process critical clinical information and provide direct treatment to patients makes them safety-critical devices. On one hand, advances in the development of medical devices promise to ensure the quality of treatment. On the other hand, the increasing device complexity, shrinking technology sizes, and shorter time to market have resulted in major challenges in ensuring reliability, safety, and security as the potential failure of these devices might have serious impact on patients' health and wellness. Recent studies have shown the susceptibility of medical devices, such as patient monitors, infusion pumps, and implantable pacemakers, to accidental faults or malicious attacks with potential adverse impact on patients [6, 16, 23, 5].

One of the possible reasons for such incidents might be the lack of consistent understanding and perspectives with respect to the health-care and engineering disciplines. This is due to the fact that researchers from different domains, specifically engineering and medicine, have developed their

different respective perspectives, concerns, and definitions regarding patient safety, security, and dependability. Besides, fierce market competition has been increasing among the medical device manufacturers that results in the tendency of achieving shorter time-to-market among the companies. The consideration of safety and security in the design of these devices is often overlooked in the design phase due to the limited time-to-market (TTM) and tight timing and resource constraints [6]. Further, some of the existing monitoring techniques rely on the fixed rules and general medical guidelines and often ignore the underlying context including dynamic system and patient status [33].

This work proposes to consider a new system-resilience design methodology which considers the many dimensions of complexity and design constraints to design a system that continuously prevents, detects, mitigates, or ameliorates hazards and incidents. To be specific, the goal of this research is to develop a general design methodology that provides the automatic inclusion of safety monitors in any functional design without the designer having to explicitly designate monitors during the design process.

1.2 Objectives

This work aims to develop a system-resilience design methodology in which a context-aware safety monitor is automatically synthesized from the safety requirements identified during the hazard analysis process. This monitor is then integrated with the control software of the target device to continuously detect and mitigate the delivery of unsafe control commands to patient at run-time. The use of the STPA hazard analysis further provides a common platform between different research domains (Engineering and medical) to collaborate with each other and share their respective perspectives. A very abstract view of a the proposed approach is shown in Figure 1-1.

In particular, we propose a hybrid model- and data-driven approach for synthesis of safety monitors, starting with hazard analysis using Systems Theoretic Process Analysis (STPA) to identify the underlying system context leading to potentially unsafe control actions and hazards. The

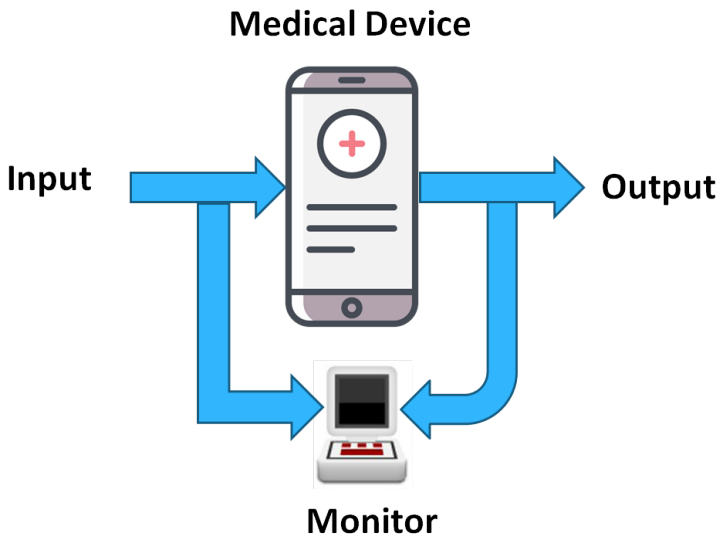

Figure 1-1: System With The Safety Monitor

logic for detection of unsafe system context is translated into linear temporal logic (LTL) rules that are further refined using the closed-loop simulation of controller with a dynamic patient model. Specifically, we use simulated patient models to learn the thresholds and parameters required for the execution of the LTL formulas. The safety monitor is then implemented as a wrapper integrated with the control software that only has access to the input-output interface (sensor and actuator values) and performs real-time execution of logic rules along with simulation of patient model. To demonstrate the effectiveness of our approach, we develop a closed-loop artificial pancreas system (APS), consisting of an APS control software (OpenAPS) and a patient glucose simulator (Glucosym). The performance of the monitor is assessed in terms of timely and accurate detection of potentially unsafe controller commands due to hardware errors to mitigate them before being delivered to the patient.

1.3 Contributions

In this study, we examine the challenges of lack of knowledge about fault tolerance in the design phase and shorter time to market by proposing a design methodology that ensures safety by preventing unsafe interactions among the system components.The primary contributions of this study can be summarized as follows:

- We develop a hybrid model- and data-driven approach for synthesis of context-aware safety monitors that can seamlessly be integrated with the control software at design-time to continuously detect and mitigate potentially unsafe control actions before they are delivered to patient's body at run-time.

- We exploit the information from the STPA hazard analysis process to generate safety monitoring rules that are continuously checked to detect and prevent unsafe control commands.

- We propose a general system resilience design methodology that can be applied to any medical cyber-physical system. Our proposed design methodology lets the designers not worry about safety during the design phase. The designer can focus solely on functional design aspects, thus, shortening the time-to-market while not compromising safety.

- The proposed method is capable of automatically generating the system safety context table based on the system specification information provided by the designer and can automatically convert the safety constraints from the context table into linear temporal logic that can be readily fed into the parameter learning tools (TeLEX).

- Our experimental results on the closed-loop OpenAPS system shows that the proposed context-aware monitoring technique outperforms a baseline monitor designed using medical guidelines with up to 23% increase in average detection accuracy (F1 score), significant reduction in the number of false positive alarms (from 649 to 175), greater improvement in terms of true negatives (from 1646 to 2120), while keeping the same true positive and false negative rates.

1.4 The Organization

The rest of this thesis is organized as follows:

Chapter 2 provides a review of the past related work covering run-time verification, safety monitoring, and fault detection and fault tolerance in medical cyber-physical systems.

Chapter 3 presents the technical background and the case study we have used to evaluate our proposed technique. In this chapter, we explain how the unsafe interactions among the components can compromise the safety of the system and show the development of the system that we used as our case study in this research.

Chapter 4 introduces the proposed design approach. In particular, this chapter contains the process of hazard analysis, collecting data traces through closed-loop system simulation, the extraction of the temporal logic rules, and the process of learning thresholds for the safety checking rules.

Chapter 5 presents the experimental evaluation of the proposed system. We assess the efficacy of the proposed monitor in the presence of hardware faults in memory of controller by artificially creating faults in the software controller of the target system and measuring the performance of the monitoring system in terms of accuracy and timely detection of hazards.

Chapter 6 concludes the thesis by providing a summary of insights and lessons learned and future research directions.

Code Availability

The code for the closed-loop APS system and the safety monitor is available at the following repository: https://github.com/gitguige/openaps_monitor/tree/master/myopenaps.

Chapter 2

Related Work

We conducted a literature review on the state-of-the-art fault tolerance and detection techniques in cyber-physical systems (CPS). Based on our survey study, we summarized different techniques and performed a comprehensive comparison among them.

Table I summarizes the previous work on model-based fault detection techniques in different application domains and compares them in terms of design approaches and the design stages at which they were applied.

Then, Next section categorizes recent works based on different approaches for ensuring safety in the medical cyber-physical systems and provides an overview of the research works in those categories.

2.1 Runtime Verification

Run-time verification of cyber-physical systems has been an active area of research because many bugs go undetected during the verification at the development stage. This types of bugs only appear at runtime and this necessitates the runtime verification in CPS. Also, the general approach of the verification follows an ad-hoc manner including trial and error basis. Rigorous verification

necessitates the involvement of formal methods.

The safety properties are usually described as Finite State Machines (FSM) or Linear Temporal Logic (LTL) and synthesized into run-time monitors [10] [17]. The most relevant and recent work in this area is [12] which presents robust online monitoring of signal temporal logic (STL) in CPS. However, to the best of our knowledge, none of the previous work on run-time verification have incorporated the notion of safety context in CPS and the information from the hazard analysis process for the specification of safety properties and synthesis of run-time monitors.

2.2 Safety and Security Monitoring in CPS

Another popular form of the techniques to ensure safety is the active monitoring scheme in which a monitor continuously observes the behavior of the system. The monitor usually contains some safety checking rules that are continuously checked and if there is any violation, the alarm or notification is generated. In [24], the authors proposed a complete framework for generating safety checking rules. In this work, the safety rules are generated through a hazard analysis technique and then validated by formal methods. They applied their technique to a mobile manipulator robot. While their approach presents a generic method to define safety checking rules, in an open environment for the extremely diverse scenarios, the formal analysis seems to be a non-trivial task.

A dynamic model-based detection and mitigation technique of safety-critical attacks on CPS in robotic surgery is presented by Alemzadeh et al.[5]. Aliabadi et al. [7] proposed a technique that dynamically mines system properties and infers likely specifications based on observational data. They mined the temporal properties from the smart meters and medical device and exploited those properties for intrusion detection.

[30] proposed a runtime security threat detection and mitigation technique that uses the adaptive risk assessment methodology. [34] proposed a verification approach that provides the advantage of real-time simulation to support for runtime verification.[32] proposed the synthesis of a safety guard that acts as a reactive component connected to the target system.

Ref.	Venue/Year	Target Application	Design Stages	Comments
Mashkoor et al. [26]	IEEE Embedded Systems Letters, 2016	Hemodialysis machine	Formal re verification, validation, code generation	Feedback loop exists from both verification and validation stages to requirements
Lee et al. [21]	Proceedings of the IEEE, 2012	Infusion pump and pacemaker	System requirements, system model development and verification, code generation and validation, implementation	In case of any constraint violation, the whole process is iterated until all constraints are satisfied.
Fleming et al. [14]	In Proceedings of the 5th IAASS Conference, 2011	Spacecraft	Identify hazards by STPA and define conditions using variables in system model, recursive identification of more refined conditions, repeat first two steps until all inputs in the process model are explained, identify possible conditions and eliminate impossible conditions	Safety was considered as a control problem rather than failure problem
Pajic et al. [29]	IEEE Transactions on Industrial Informatics,	Closed loop Medical Device	Simulink model of closed loop medical device including patient model with pharmacokinetic dynamics of drug absorption by patient body, Incorporating uncertainty in patient model to reflect variation between patients, Fault model development, Finding unsafe conditions, modification of fault model to restore safety.	N/A
Murugesan et al. [27]	IEEE Transactions on Industrial Informatics,2014	Generic Patient Controlled Analgesia (GPCA)	Requirement analysis in continuous and real time aspect, defining scope of a system, formal verification, code generation, testing the code	N/A

Along with the target device, this combined system ensures the satisfaction of the safety rules and at the same time, it minimizes the deviation from the normal system behavior.

In most of the previous safety monitoring approaches, the safety checking rules are generated without following any generalized methods and the same state variables observed by the monitored device are often used as variables by the safety checking rules. Thus the monitoring approach is not completely independent of the monitored device. Any error in the variables will certainly affect the monitoring process. To address these challenges, we propose a systematic approach for automatic synthesis of the safety checking rules from the STPA hazard analysis process and development of safety monitors that can independently work aside from the monitored device by only observing the input and output variables. In this work, similar to [5] we utilize the dynamic model of the physical process (patient glucose status) for detection of unsafe control actions before being delivered to the physical process. However, the idea of automated synthesis of the monitor from the safety properties identified using STPA hazard analysis is new.

2.3 Fault Detection and Tolerance in Medical Devices

One of the most related areas to this work is the model-based development and resilience by construction of medical CPS. In [8], a generic formal model of infusion pump systems was proposed. The authors first conducted a rigorous requirement and hazard analysis for infusion pumps and then proposed a generic model that ensures all the safety requirements to avoid the possible hazards. This led to a model-based development of generic patient-controlled analgesia (GPCA) pump model. In [26], a "Correct by construction" approach was presented for the trustworthy development of a hemodialysis machine. The whole design process was divided into four steps of Formal requirements specification, Verification, Validation, and Core generation. In [31], the authors proposed a formal approach for early validation of Medical Cyber-Physical Systems (MCPSs). They provided a model library composed of two models: patient model and device model. The proposed

models were built as reusable and customizable enough to be reused in different MCPS and could be extended to fit unpredictable MCPS. A fuzzy logic-based fault detection technique for wearable medical devices was proposed in [28]. In this approach, the wearable device contained a module to capture biosignals, a microcontroller and a module to transmit data to a Smartphone. The captured data was analyzed by the fuzzy logic algorithm to decide whether the output is abnormal, and if abnormal, it could identify the causes for the abnormality.

Another relevant work is Young et al. [33] which proposed a data authenticity monitoring system to ensure the authenticity of the collected data in a body area network assembled to provide treatment to the Type 1 diabetic patient. They specified a range of system properties related to type 1 diabetes treatment constraining glucose variability and insulin delivery. They expressed these properties in signal temporal logic and used these to monitor the property violation in the collected data. Their approach is completely unaware of the context of the patient.

Chapter 3

Technical Background and Case Study

This chapter presents an overview of the notion of safety context in CPS and introduces our case study of an Artificial Pancreas System (APS). The description of the major components of an APS and the step by step process of the development of the APS is also described in this chapter.

3.1 Safety Context in Cyber-Physical Systems

Cyber-physical systems (CPSs) are interconnected embedded systems that connect the cyber world with the physical world to provide the intended support. CPS gather information from the physical world through sensors and perform the necessary processing, including machine learning and control, to change the state of physical processes. Medical Cyber-physical Systems (MCPS) as a branch of cyber-physical systems promise to ensure safety in providing treatment to patients. As an example of the MCPS, consider the artificial pancreas system in Figure 3-1. In this system, a continuous glucose monitor continuously collects blood glucose information at a fixed time interval and sends the reading to a controller that calculates the recommended insulin and sends a command to the insulin pump accordingly. An unsafe control action provided by the controller may cause serious harm to the patient body and the potential cause of an unsafe control action might arise

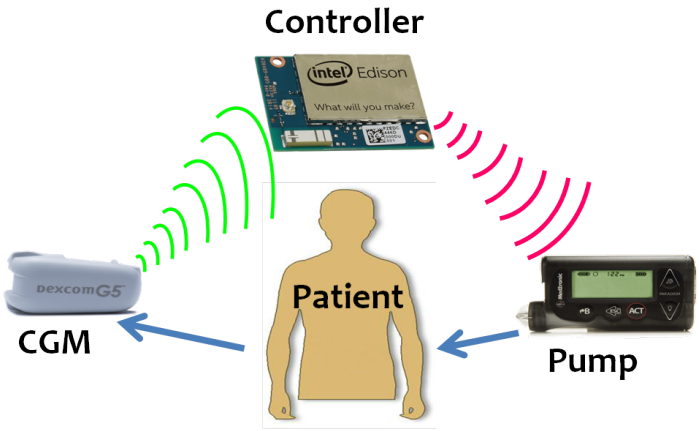

Figure 3-1: Artificial Pancreas.

from the measurement error of sensors, hardware fault or software error in controller algorithm or fault in the insulin pump. Even while all the components work perfectly, there still might be a possibility of the unsafe control action being delivered. As an example, for high blood glucose, if there is enough insulin in the patient's body, low insulin delivery seems alright. However, while the insulin in the patient's body is low, the low insulin delivery will place the patient at the risk of having hyperglycemia in the future. This shows the necessity of using the knowledge of underlying system context and patient status in design of MCPS.

Traditional rule-based monitoring system might include some fixed checking rules based on the medical guidelines that seriously suffers from the limitation of not considering the system context and the controller action. This type of monitor does not take controller action or system state into account that leads to a large number of false alarm. Examples of such events are shown in Figure 3-2. In this figure, we can see a baseline monitor designed using medical guidelines generates alarms when blood glucose value crosses a certain limit, 180 in this case. However, it does not consider the action of the controller that actively delivers sufficient insulin to bring the glucose in the normal range (shaded region). Thus, baseline creates a lot of false alarms.

Recent studies show the adverse events associated with different medical devices reported to

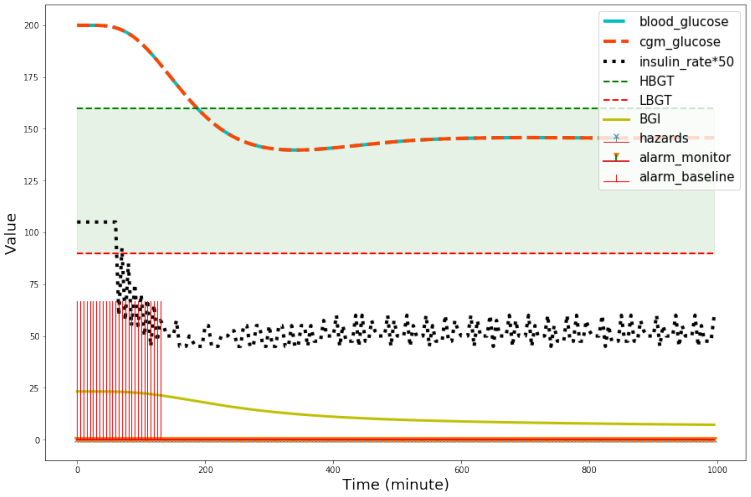

Figure 3-2: Monitor does not consider the action of controller

the U.S. FDA. For example, adverse events involving patient monitoring systems in hospitals where data errors and algorithmic inadequacies caused false alarms or missed detections, leading to patient harm [15][4]. This advocates that ensuring safety should be considered as primary objective while designing the medical cyber-physical systems.

3.2 Case Study

To demonstrate our proposed methodology, we exploited the artificial pancreas system (APS) as our case study and developed a closed-loop APS with the help of two open-source simulators: Open Artificial Pancreas System (OpenAPS) and GlucoSym. Figure 3-1 shows the basic diagram of an artificial pancreas system.

3.2.1 Artificial Pancreas

Artificial pancreas is a connected environment of three basic subsystems: continuous sensing of blood glucose level, control algorithm that calculates required insulin, and insulin delivery sys-

tem. It replaces the manual checking of blood glucose and the insulin shot. It is a single system containing three major components mentioned above that continuously checks the patient's blood glucose level and automatically delivers the required insulin to the patient body.

3.2.1.1 Continuous Glucose Monitor (CGM)

Continuous Glucose Monitor (CGM): the Continuous Glucose Monitor (CGM) is the critical component in the artificial pancreas system that continuously tracks the blood glucose level at regular intervals, usually 5 minutes, throughout the day and night. Figure 3-3 shows different parts of a continuous glucose monitor.

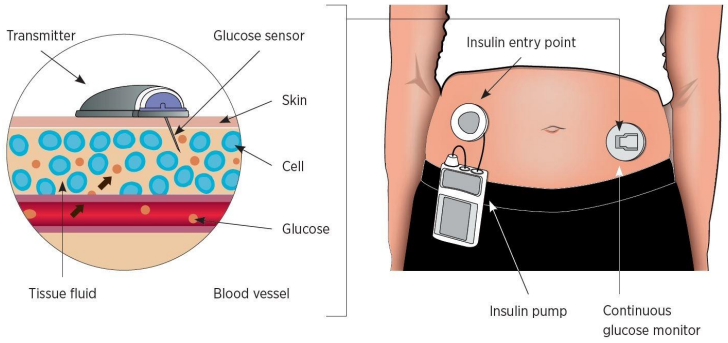

Figure 3-3: Continuous Glucose Monitor (CGM)

It helps users to control their glucose level by providing information about the effect of food ingestion, insulin delivery, exercise or other physical issues on their glucose level. CGM works by inserting a tiny sensor wire under the skin. It measures the glucose level of the fluid between cells that are known as the interstitial glucose level.

A small transmitter is used to transmit the real-time glucose level measurements to the receiver so the patient can observe the real-time glucose level in his/her blood.

Figure 3-4: Insulin Pump

3.2.1.2 Control Algorithm

The control algorithm is the key element of an artificial pancreas system. It uses the BG data from CGM and the actions of the insulin pump to calculate required insulin delivery rate at regular intervals. In past years, researchers have developed may promising control algorithms. Among them, Model Predictive Control [19], Run-to-run Control [13] and PID-based controllers [25] are the most prevalent algorithms.PID controller responds to the change in glucose level and MPC is based on the prediction of glucose dynamics. The run-to-run controller can learn the patient's daily routine and using this information it produces an optimized response.

3.2.1.3 Infusion Pump

The insulin infusion pump is a small, computerized device that continuously delivers short-acting insulin (basal rate) all day long. It delivers insulin through a tube that is connected to a cannula which remains inserted under the skin: typically into the fat area under the skin. Figure 3-4 shows an insulin pump and it's connectivity to the patient's body.

The basal insulin rate is usually set according to the suggestion of medical personnel. Most of the insulin pump comes with the built-in bolus calculator. A bolus is the high amount of insulin that needs to be pushed before food meal. Patients use the bolus calculator to calculate the required insulin based on the amount of carbohydrate they are going to take in.

3.3 Development of Artificial Pancreas

To demonstrate the ideas proposed in this work, we developed a closed-loop APS using the Open Artificial Pancreas System (OpenAPS) [2] as the controller and the open-source GlucoSym simulator [1] as our patient model. The overall system architecture of the developed closed-loop APS is shown in Figure 3-5. The description of about these two simulators is provided in section 3.3.1 and 3.3.2.

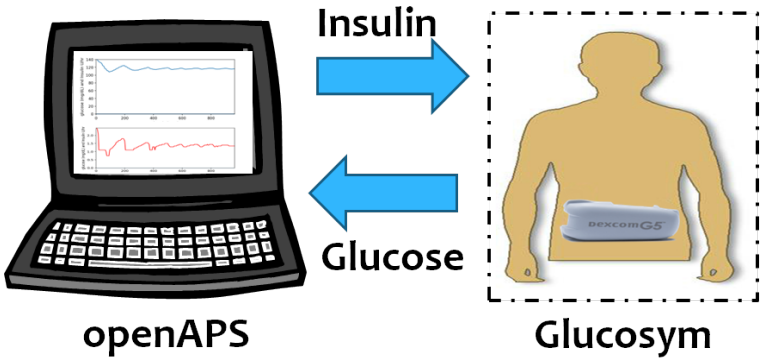

Figure 3-5: Closed Loop System with OpenAPS Controller and GlucoSym.

GlucoSym acts as both the patient model and the CGM sensor. As we are using a simulated patient model, the insulin pump is omitted and the output of the controller is directly fed into the GlucoSym simulator. A python script is developed to enable synchronized interaction between these two simulators and to update the necessary files to simulate the functionality of sensor and actuator and the interactions between the controller and insulin pump and patient in the real APS system.

OpenAPS is the controller that calculates required insulin delivery to the patient's body. The python script establishes the interface between OpenAPS and GlucoSym. It also performs necessary updates of different files to simulate the interaction between OpenAPS and infusion pump. Interaction among OpenAPS, GlucoSym and the python script is shown in Figure 3-6.

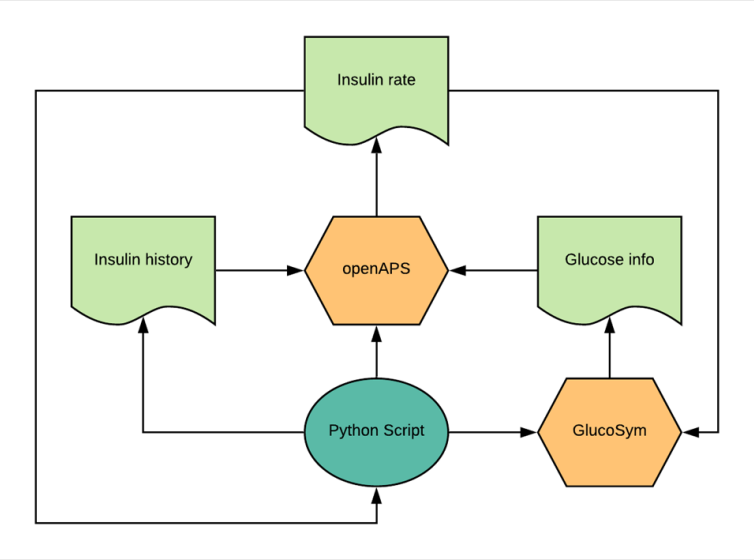

Figure 3-6: Communication among OpenAPS, GlucoSym and Python Script

In a real artificial pancreas system, OpenAPS pulls current information and as well as the history of insulin delivery from the pump every 5 minutes. To emulate this scenario, we set up one file that stores the current insulin delivery and also another file that keeps track of the history of insulin delivery. This helps the OpenAPS to calculate the amount of insulin stored in the patient's body at any given time (IOB). IOB is one of the important parameters that OpenAPS use during the calculation of insulin delivery.

The most challenging part during the development of the artificial pancreas was to emulate the elapse of time. The way the OpenAPS works is it pulls the insulin delivery information from the infusion pump every 5 minutes with a time-stamp and calculates the required insulin delivery based on the history of insulin delivery, current insulin delivery, and current blood glucose level in patient's body. To emulate the elapsing of time, we change the time of the machine that the OpenAPS is installed on to shift 5 minutes in the future. This gives the illusion that the OpenAPS is performing the required calculation every 5 minutes. However, this causes another challenge while evaluating the performance of the proposed system by recording system times. This difficulty is

resolved by recording the number of iteration and multiplying the iteration number by 5 the time is found, because, every iteration represents 5 minutes. Sometimes, the epoch time is recorded to overcome this issue while the time needs to be recorded is less than one iteration.

Figure 3-7 shows the simulation the glucose level of a particular patient while no insulin is fed to the patient. The simulation starts with the initial glucose value of 150. Figure 3-8 and 3-9 show the glucose level of the closed-loop system and recommended insulin by the controller respectively. The simulation was run for 1000 minutes with a patient model provided by GlucoSym simulator.

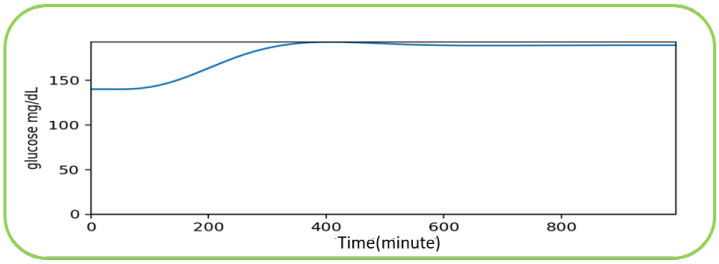

Figure 3-7: Glucose level of patient without insulin feedback.

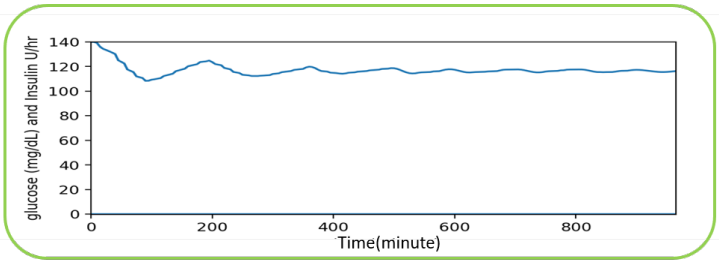

Figure 3-8: Glucose level of patient with insulin feedback.

The upper blood glucose level was set at 120 and the lower target was set at 110. In Figure 3-7, we can see that without any insulin feedback, the glucose does not have any intention to be lower while in Figure 3-8, the insulin brings the glucose back to the targeted range between 120 and 110.

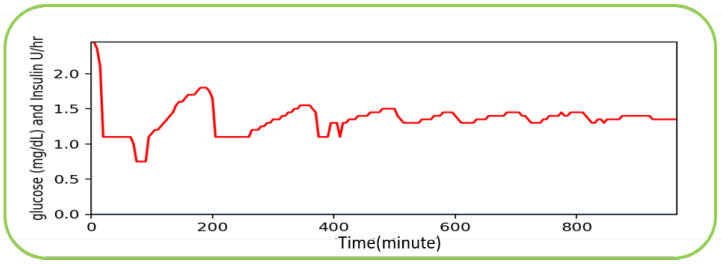

Figure 3-9: Recommended insulin rate.

3.3.1 OpenAPS

The OpenAPS is a safe and advanced APS controller that is developed by an open-source community [2]. It adjusts the insulin delivery of an infusion pump to automatically keep the blood glucose level of a diabetic patient within a safe range. The internal architecture and necessary input-output connections of OpenAPS are shown in Figure 3-10. The description of input parameters is listed in Table 3.1.

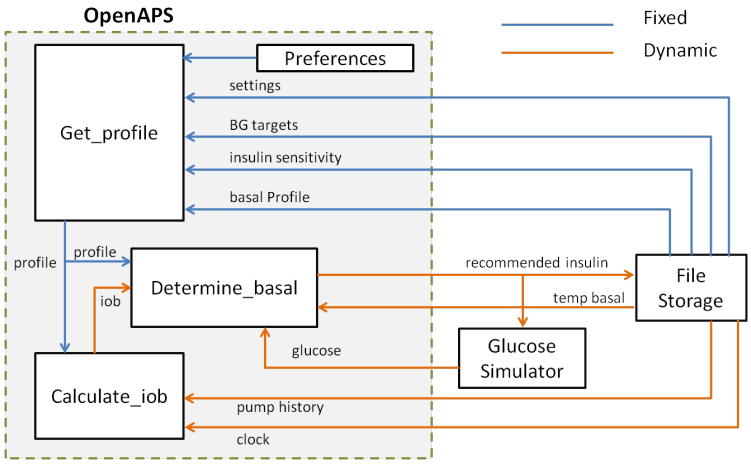

Figure 3-10: OpenAPS Architecture and Input Output.

The shaded region indicates the OpenAPS controller. "File storage" section reflects the behavior of the insulin pump. The functionality of OpenAPS can be divided into three processes. Get_profile process accepts pump settings, Blood glucose target (BG target), insulin sensitivity,

basal profile, and preferences as inputs and creates a profile that is required to calculate both "insulin onboard active in the body" (IOB) and "recommended insulin delivery". The Calculate_iob process gets profile, clock and pump history as input and calculates IOB (insulin on board).

Finally, the Determine_basal process accepts profile, IOB, blood glucose and current insulin delivery (temp_basal) and calculates the suggested insulin delivery to the patient.

Table 3.1: Input Parameters of OpenAPS.

Input	Description
Settings	Various settings specific to the pump
BG targets	High/low glucose targets set up in the pump
Insulin Sensitivity	The expected decrease in BG as a result of one unit of insulin
Basal profile	The basal rates that are set up in the pump
Preferences	User defined preferences
Pump history	Last 5 hours data directly form the pump
Clock	Date and time that is set on the pump
Temp_basal	Current insulin delivery rate set up in pump
Glucose	Glucose level sensed by CGM

More specifically, OpenAPS collects the previously delivered insulin amount and combined with the duration of the activity, it calculates the net IOB. Using the glucose sensor readings, OpenAPS then calculates the eventual BG (Blood Glucose) using the following equation:

$$CurrentBG - ISF * IOB = eventualBG$$

where $CurrentBG$ is the current Blood Glucose, ISF is the Insulin Sensitivity Factor, IOB is the Insulin on Board and $EventualBG$ is the estimated BG by the end of current insulin delivery.

While current BG is below a threshold value, OpenAPS continues to issue temporary insulin delivery to zero until BG is not rising. Otherwise, OpenAPS determines whether the glucose values

Algorithm 1: OpenAPS Algorithm

```
if BG is rising, but E_BG < BG_Target then
    cancel any temp baasal;
else if BG is falling, but E_BG > BG_Target then
    cancel any temp basal;
else if E_BG > BG_Target then
    calculate 30 min temp basal;
    if recommended temp > existing basal then
        issue the new high temp basal ;
    else if recommended temp < existing basal then
        issue the new low temp basal ;
    else if 0 temp for >30m is required then
        extend zero temp to 30 min ;
```

Figure 3-11: OpenAPS Algorithm

are rising or falling more than expected. In that case, it performs the course of actions shown in Algorithm 1 in Figure 3-11.

3.3.2 GlucoSym

GlucoSym is an open-source human body glucose simulator that was developed to help building and testing automatic insulin delivery systems. This simulator mainly works based on the mathematical model shown in Figure 3-12. Using this simulator, one can compare multiple algorithms with the same patient model, test on a patient profile with parameter variations during the day, or test against a population of patients. Sample parameters of this simulator are shown in Figure 3-13. The description of different parameters is listed in Table 3.2.

{ "dose": 0.0, "dt": 5, "index": 0, "time": 1080, "events": {"bolus": [{"amt": 0.0, "start": 60}], "basal": [{"amt": 1.3, "start": 0, "length": 600}], "carb": [{"amt": 0.0, "start": 600, "length": 90}]} }

Figure 3-13: Sample parameters of GlucoSym simulator.

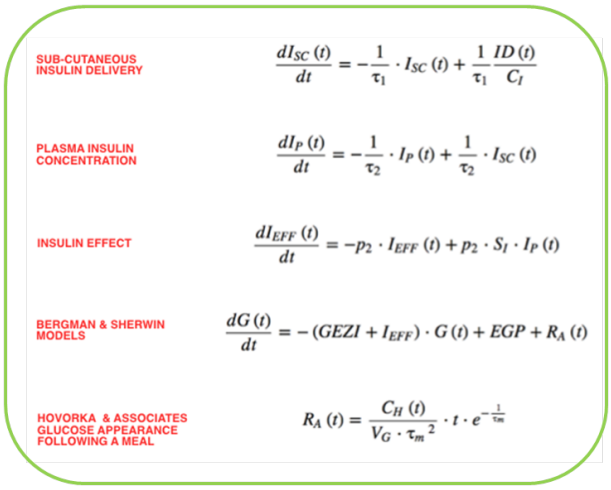

Figure 3-12: Mathematical equation used in GlucoSym simulator [1].

Table 3.2: Input Parameters of GlucoSym.

Input	Description
dose	Insulin dose in units given during the time-step. In the case of a basal (insulin delivery) adjustment, we need to calculate how much insulin will be given in the time-step defined by "dt" (i.e. how many insulin units will be given in 5 minutes by the set basal profile or temporary basal?).
dt	Change in time each step in minutes.
index	Current index from the start of the simulation, starting at 0.
time	Total simulation run-time in minutes.
basal	The delivery of insulin.
events	Events are set so that the simulator will consider them during the run. The events were sent on-the-go.

Chapter 4

Approach

This chapter introduces the overall methodology. First, a high-level overview of the entire approach for automated synthesis of the safety monitor is provided and then each step is described in more detail.

4.1 Design Methodology

Figure 4-1 shows the overall steps in our approach for the synthesis of a context-aware safety monitor. We first conduct a rigorous hazard analysis using STPA to identify the potential unsafe scenarios that might lead to safety hazards for the patient. This is done by developing a system context table consisting of all the control actions and different system contexts that might lead to unsafe control actions. The output from this step is the logic describing safety requirements or set of safety constraints that should not be violated by the target controller. Then, by performing a closed-loop system simulation, we generate simulation traces that can help with further refinement of the logic learned from the hazard analysis step.

Finally, the logic describing the unsafe system context is translated into linear temporal logic and is provided along with the simulation data to a tool for automated extraction of logic rules

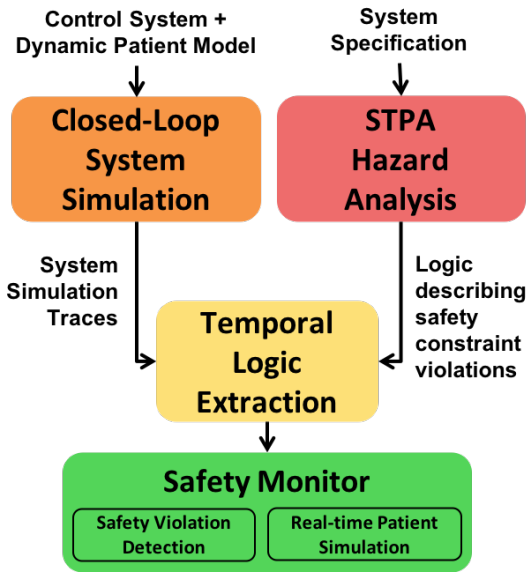

Figure 4-1: Steps for Synthesis of Safety Monitor

for the implementation of the monitor. The final monitor is composed of the extracted logic rules for detection of safety violations along with a simulated patient model that predicts the outcome of providing a control command or the risk of harming the patient in a near-future before the command is actually delivered to the patient.

Figure 4-2 shows an example of the context-aware monitor synthesized from the STPA context table and integrated as a wrapper with the OpenAPS control software. As shown in the figure, the monitor only has access to the input and output of the target control software and is only designed based on the knowledge of system specification and hazard analysis process. It should be noted that since the monitor receives the sensor and actuator data from the target controller, it cannot detect any faults in the sensors or output interface of the controller. The main focus is instead on detecting any flaws in the control algorithm or software and hardware faults in the controller itself. We assume that the sensor data is not faulty and the same sensor data is delivered to the OpenAPS controller and the monitor. The monitor has access to the control command generated by the OpenAPS controller before it is delivered to the pump. More details on different steps of the

synthesis process are provided next.

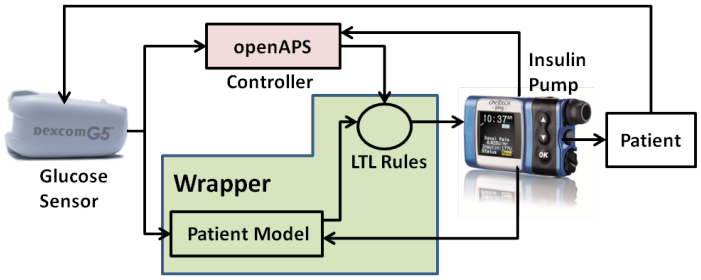

Figure 4-2: Context Aware Monitor integrated as a Wrapper with the OpenAPS Control Software

4.2 Systems Theoretic Hazard Analysis

STPA[22] is a hazard analysis technique based on control and systems theories that, unlike traditional risk/hazard analysis techniques, focuses on the dynamic behavior of a system. It is in particular appropriate for hazard analysis in CPS because it models the system as a hierarchical control structure with multiple control loops, including the human operators, automated controllers, and cyber and physical processes. While traditional techniques are based on reliability theory and use component failure as the key point for hazard analysis, STPA is based on systems theory and considers safety as an emergent property of the system and tries to solve it as a control problem. STPA can be divided into four parts:

1. Defining accidents or losses

2. Identifying the potential unsafe control actions

3. Create safety requirements using the identified control action

4. Determine how each unsafe control action can occur

Basically, STPA starts with defining the accidents followed by identifying the hazards associated with these accidents. Then a high-level functional control structure is drawn that is used to identify the unsafe control actions and their potential causes.

The two main steps of STPA hazard analysis process can be summarised as follows:

- Identifying Unsafe Control Actions: An unsafe control action is defined as one the following four types:

    1. A control action required is not provided.
    2. An incorrect control action is provided.
    3. A control action is provided too early or too late.
    4. A control action is stopped too early or applied too long.

- Identifying the Causes of the Unsafe Control Actions: This step is done by analyzing the contributing causes to unsafe control commands, including faults in the sensor measurements and controller input, errors in the control algorithm, or faults in the actuators.

Accidents: Based on the functionality of the OpenAPS controller, and according to the STPA Analysis [22], we first characterized two types of accidents that might happen to the patients due to faults in the APS controller:

- **A1:** hypoglycemia: Blood glucose level is more than 280 mg/dL.
- **A2:** hyperglycemia: Blood glucose level is less than 70 mg/dL.

We used the concept of risk index described in [11] to define the hazardous situation that eventually causes hyper- or hypo-glycemia (Accident) in the patient's body. In [11], the authors introduced the notion of risk index that captures both the glucose variability and its associated risks for hypo- and hyperglycemia. As a first step of the risk index calculation, the blood glucose scale is transformed into a symmetric scale using the following formula.

$f(BG) = 1.509 * [(ln(BG))^{1.084} - 5.381]$

Then BG risk function is then computed and the left and right branches are separated to get low (LBGI) and high (HBGI) BG risk index. This computaion is done using the following formulas.

$r(BG) = 10 * f(BG)^2$

$rl(BG) = r(BG)$ if f(BG)<0 and 0 otherwise

$rh(BG) = r(BG)$ if f(BG)>0 and 0 otherwise

For a series of blood glucose values BG1 , BG2, ……., BGn , low (LBGI) and high (HBGI) BG risk indices are calculated as follows.

$LBGI = 1/n \sum_{i=1}^{n} rl(BG_i)$ $HBGI = 1/n \sum_{t=1}^{n} rh(BG_i)$

The overall BG risk index BGRI is determined by summing up LBGI and HBGI.

$BGRI = LBGI + HBGI.$

The authors claim that LBGI and HBGI separate the glucose variation into two independent regions and it also equalizes the amplitude of these two regions based on the risk associated with the amplitude. Thus, BG fluctuation from 180mg/dL to 250mg/dL and BG fluctuation from 70mg/dL to 50mg/dL are treated as equal in terms of risk associated with them although, in terms of amplitude variation, they are quite different.

Using the notion of risk index and the fault-free simulation data, we created the ground truth by labeling hazardous data points that are later used to evaluate the proposed monitor. As our proposed monitor detects the hazardous points, this helps to prevent accidents as hazards are the previous state of the accidents.

Hazards: We also identified the potential hazards based on the risk [11] index that might cause those accidents:

- **H1:** Low Blood Glucose Index (LBGI) is more than 5 and keeps increasing.
- **H2:** High Blood Glucose Index (HBGI) is more than 9 and keeps increasing.

Safety Control Structure: We then modeled the safety control structure of the closed-loop APS as shown in Figure 4-3. We identified different system conditions in which different control actions might possibly be unsafe and lead to a hazardous situation. First, we defined the state variables and then analyzed all the control actions with different combinations of these state variables in which the control actions are considered as unsafe. In STPA, to identify unsafe control actions, the following 4 specific scenarios are considered:

- A required control action was not provided and.
- A control action is provided when not required.
- A potentially safe control action is provided too late or too early.
- A safe control action is provided too long or is stopped too soon.

However, for simplicity, in this case, we considered the first two of the above four scenarios.

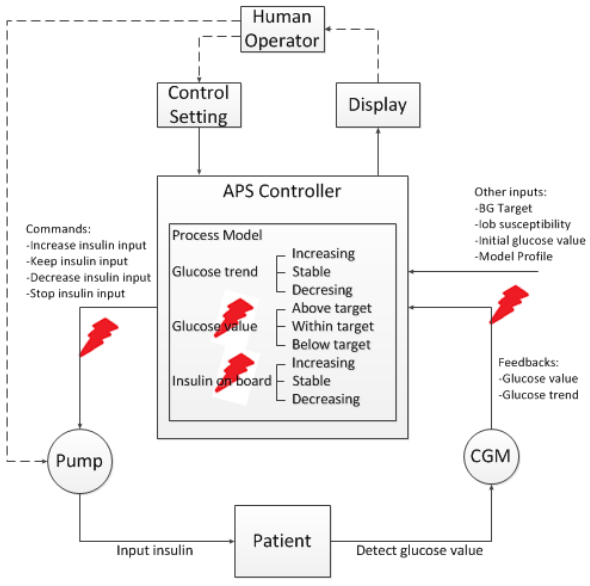

Figure 4-3: Control Structure with the OpenAPS Controller's Process Model

STPA Context Table: The construction of a context table starts from selecting the controller, process variables and control actions from the control structure. In the initial stage, the context table is generated with all the combinations of the controller actions and the process variable. Different combinations of process variable values represent the underlying system context.

Unsafe Control Actions: We identified the potential unsafe system states or actions that might cause hazards leading to accidents:

- **U1-1:** Controller sends insulin delivery command while BG is low (Insulin is not needed).
- **U1-2:** Controller sends high insulin delivery command when low insulin is required.
- **U2-1:** Controller does not send insulin delivery command while BG is high (Insulin is needed)
- **U2-2:** Controller sends low insulin delivery command when high insulin is required.

However, all the contexts are not hazardous, some are redundant. Next step involves the pruning of the non-hazardous and redundant rows from the context table that gives us the final context table. Table 4.1 shows the final context table for the artificial pancreas system. This is the final context table after the necessary pruning of the non-hazardous and redundant rows. The first column contains the rule number. The second column indicates different control actions provided by the controller, the next two columns are the process variables BG(Blood Glucose) and IOB(Insulin On Board). The last column determines whether the control action leads to a hazardous situation in a specific context if that control action is provided or is not provided. dec_insulin, inc_insulin and zero_insulin refer to decrease, increase and stop insulin delivery respectively. As an example, the first row of the context table includes a specific context where the blood glucose (BG) level is greater than the blood glucose target (BGT) and is rising, with insulin on board (IOB) below a certain threshold and also rising. In this context, providing a dec_insulin command might lead to a hazardous situation.

Our original context table for OpenAPS included several redundant rows corresponding to similar or not hazardous system context which we pruned in order to generate the summarized context table here. Each row of the final context table represents a safety property or constraint that if violated by the controller might lead to one of the safety hazards identified earlier. Thus, each row of the context table can be used as a logic rule to be checked by a safety monitor that has access to all the values of state variables and the control commands from the controller to detect any potential harmful control actions.

STPA helps to conduct hazard analysis using the specification only. In this way, there is no need to learn about the internal design implementation of the system during the hazard analysis process. It also considers system-context while identifying unsafe control actions. These two are the main motivation behind using STPA in our design methodology, because it helps generating safety rules that are context-aware and also as this process does not need any knowledge about the internal design of the device, it can be used as a general methodology for any MCPS.

4.3 Closed-Loop System Simulation

As shown in Table 4.1, the logic rules extracted from the STPA context table are based on undefined thresholds (e.g., Th1 in row 1) that should be refined based on the target controller and adjusted using data from the patient. In other words, the inclusion of system context is driven not only by the STPA model but also by the data collected from a closed-loop system consisting of an artificial pancreas controller and a patient model.

To achieve this goal, we developed a closed-loop system by integrating the OpenAPS control software with the Glucosym patient simulator and collected data traces from a population of patients with different characteristics and body parameters provided by the Glucosym patient simulator. These data traces were then used for parameter tuning of the logic rules extracted from the STPA context table.

Table 4.1: STPA Context Table

Rule No	Control Action	BG	IOB	Hazardous Control Action?	
				Provided	Not Provided
1	dec_insulin	>HBGT+Th12; rising	falling; < Th1	yes	
2	dec_insulin	>HBGT+Th13; rising	stable; < Th2	yes	
3	dec_insulin	>HBGT+Th14; falling	rising; < Th3	yes	
4	dec_insulin	>HBGT+Th15; falling	falling; < Th4	yes	
5	dec_insulin	>HBGT+Th16; falling	stable; < Th5	yes	
6	inc_insulin	< LBGT+Th17; falling	rising; > Th6	yes	
7	inc_insulin	< LBGT+Th18; falling	falling; > Th7	yes	
8	inc_insulin	< LBGT+Th19; falling	stable; > Th8	yes	
9	zero_insulin	>HBGT+Th20	< Th9	yes	
10	zero_insulin	< Th21	*		yes
11	keep_insulin	> HBGT+Th22; rising>Th23	not rising; <Th10	yes	
12	keep_insulin	< LBGT+Th24; falling	not falling; >Th11	yes	

$*HBGT = 95mg/dL, LBGT = 160mg/dL$ [18][9]

Table 4.2 shows the patient population used in our simulations along with their corresponding body parameters. For each patient, we collected 13 data traces with 1500 minutes worth of runtime in the real world. Each simulation started with a different initial value for a blood glucose level. This gave us a total of 13*9 data traces. We used data from 7 patients for learning the parameters for our safety monitor and the rest of the data traces were used for validation and testing of the monitor.

4.4 Parameter Extraction from Temporal Logic

We used a temporal logic synthesis tool called TeLEX developed by Jha et al.[20] to learn different thresholds used in the context table rules. These thresholds are not clinical thresholds

but they are the thresholds used in the safety checking rules in the monitor. TeLEX [3] is a passive learning approach that requires two inputs: a set of observed data traces and parameterized temporal logic formula. TeLEX learns the value of the parameters in the temporal logic formula in such a way so that all the temporal logic properties are satisfied by all the input data traces. In our case, we have collected the data traces and we already have a set of safety rules from which we need to learn the value of the thresholds. This leads us to choose TeLEX as the learning agent. As opposed to the other learning schemes, TeLEX has the advantage of learning the boundary of the parameters for legal behavior by observing only the positive (non-faulty) example.

As the first step of parameter learning, we converted all the rules of the context table in the format of temporal logic. The set of rules described in the temporal logic format are shown in Table 4.3.

These formatted rules were then fed into TeLEX along with data from simulation traces. In this case, we collected the data traces for 7 patients with 13 initial BG values while not injecting any faults to the system. TeLEX basically goes over all the data and finds the values of the thresholds that satisfy the rules for all the data traces.

4.5 Automation of Context Table and LTL Rule Generation

We developed a framework for the automatic generation of the context table and the conversion of the context table rules into LTL formulae. Figure 4-4 shows the entire work-flow of the whole process. The framework is independent of the system as long as the system has a control algorithm and the actuator to implement the control actions provided by the controller.

The entire process starts with the user input: state variable and control action. The framework let users input all the state variables, possible values of the state variables, and the control actions

Table 4.2: Patient Parameter

Patient	BW	C_I	τ_1	τ_2	V_G	$1/p_2$	EGP	I_{EFF}
A	89	20.1	49	47	253	0.01	1.33	0.002
B	63	12.81	41	10	261	0.01	0.6	0.004
C	65	9.09	71	70	199	0.02	1.07	0.003
D	116	18.13	91	70	337	0.008	0.98	0.00002
E	64	15.35	46	46	188	0.01	0.6	0.004
F	51	5.88	68	30	104	0.009	0.603	0.001
G	77	18.06	60	60	263	0.01	1.11	0.0023
H	65	5.4	95	37	137	0.01	1.3	$1e^{-8}$
I	100	8.75	131	21	193	0.01	1.27	0.006

BW=Body Weight.
C_I=insulin clearance (dl/min).
τ_1, τ_2=Time constant associated with insulin movement between the SC delivery site and plasma (min).
V_G=Distribution volume in which glucose equilibrates (dl).
$1/p_2$=Delay in insulin action following an increase in plasma insulin (1/min).
EGP=Endogenous glucose production rate that would be estimated at zero insulin (mg/dl/min).
I_{EFF}=Effect of glucose per se to increase glucose uptake into cells and lower endogenous glucose production at zero insulin (1/min).

of the system on target. Taking all these inputs, the framework generates a context table with all the possible combinations of the values of the state variables and the control actions. The next stage involves manual labeling of the context table.

Each row of the table is labeled as hazardous or not based on context and action. Then the elimination of the redundant rows is performed. In the final stage, the labeled context table is fed to the framework that converts all the rules into a Linear Temporal Logic (LTL) format. The LTL rules are used in the parameter learning phase of the synthesis of the safety monitor as the learning tool (TeLEX) is required to have parameterized LTL rules to learn the value of different parameters.

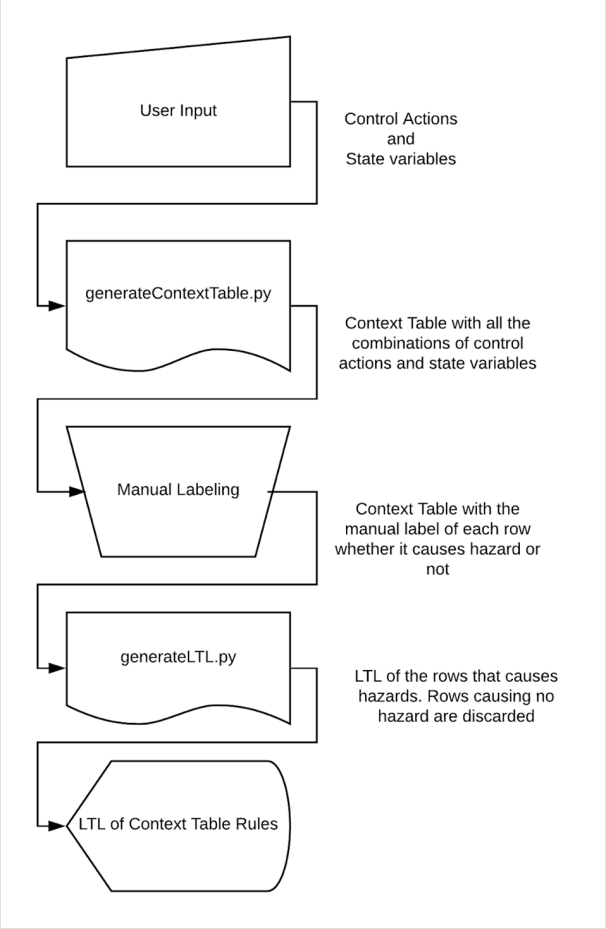

Figure 4-4: Specification to LTL rules

Table 4.3: Temporal Logic Rules Extracted from STPA Context Table and System Simulation

Rule	**Temporal Logic**
1	□(((BG>HBGT+Th12)∧(ΔBG>0))∧((ΔIOB<0)∧(IOB<Th1))⇒¬ decrease_insulin)
2	□(((BG>HBGT+Th13)∧(ΔBG>0))∧((ΔIOB=0)∧(IOB<Th2))⇒¬ decrease_insulin)
3	□(((BG>HBGT+Th14)∧(ΔBG<0))∧((ΔIOB>0)∧(IOB<Th3))⇒¬ decrease_insulin)
4	□(((BG>HBGT+Th15)∧(ΔBG<0))∧((ΔIOB<0)∧(IOB<Th4))⇒¬ decrease_insulin)
5	□(((BG>HBGT+Th16)∧(ΔBG<0))∧((ΔIOB=0)∧(IOB<Th5))⇒¬ decrease_insulin)
6	□(((BG<LBGT+Th17)∧(ΔBG<0))∧((ΔIOB>0)∧(IOB>Th6))⇒¬ increase_insulin)
7	□(((BG<LBGT+Th18)∧(ΔBG<0))∧((ΔIOB<0)∧(IOB>Th7))⇒¬ increase_insulin)
8	□(((BG<LBGT+Th19)∧(ΔBG<0))∧((ΔIOB=0)∧(IOB>Th8))⇒¬ increase_insulin)
9	□(((BG>HBGT+Th20)∧(IOB<Th9))⇒¬ zero_insulin)
10	□((BG<Th21)⇒zero_insulin)
11	□(((BG>HBGT+Th22)∧(ΔBG>Th23))∧((ΔIOB<=0)∧(IOB<Th10))⇒ ¬keep_insulin)
12	□(((BG<LBGT+Th24)∧(ΔBG<0))∧((ΔIOB>=0)∧(IOB>Th11))⇒¬ keep_insulin)

Chapter 5

Evaluation

We evaluated the proposed monitor in terms of detection time and accuracy. As a first step of the entire evaluation process, we first injected different hardware fault in the artificial pancreas controller. Then we measured the accuracy of the detection of the proposed monitor. We further measured the detection latency (time between fault injected and alarm generation) and the reaction time (time between alarm and the occurrence of hazard). We also compared the results with a baseline monitor that does not depend on the context of the patient. We divided the process into the following subsections.

5.1 Experimental Setup

We ran our experiments on a virtual machine of Ubuntu 14.04 LTS. It takes 3-5 seconds to run one iteration of the experiment which represents 5 minutes in the real OpenAPS control system. We ran 200 iterations for each test which in total equals to 16 hours of real world patient data. We used 9 patient models with different weights and IOB susceptibilities or rate of insulin consumption as our test cases. The performance of OpenAPS is assessed in presence of transient and permanent faults affecting the memory and logic of the OpenAPS controller and the performance

of the monitor in timely detection of unsafe commands is assessed in both fault-free and faulty scenarios and comparing to a baseline monitor with no context-awareness.

5.2 Fault Injection

We tested the robustness of OpenAPS and performance of our monitor by injecting faults, into the variables storing the input data of the controller, that is the glucose feedback values from CGM, and the output commands delivered from the controller to the pump. These high-level faults were injected to represent low-level hardware faults that might lead to erroneous state values in the controller. We assume that the sensor data is not faulty and the same sensor data is delivered to the OpenAPS controller and the monitor and monitor has access to the control command generated by the OpenAPS controller before it is delivered to the pump. According to the context table introduced in the former section, we injected 5 types of faults for each of the two variables, blood glucose (BG), and insulin output (rate), to the control software:

- Hold the last value of BG and rate, to represent the fact while BG and rate value do not update and keep the last value. It might cause due to the problem while writing to the register where these two values are stored.

- To represent the bit-flip in the exponent section of the binary representation of BG and insulin rate, we intentionally doubled and halved those values at random time.

- Two other faults, decrease and increase in the value of BG and insulin rate were injected to simulate the bit flip in magnitude section of the binary representation of those values.

All of the above faults are injected at a random time point and for a random duration of time during the simulation. The manipulation of the insulin rate is based on the assumption that the same insulin rate to be delivered to the pump (even if it is faulty) is given to the context aware

monitor before the command is actually delivered to the pump. The effectiveness and accuracy of the monitor in the detection of hazard and unsafe actions were evaluated with both fault-free and faulty data. To be specific, we trained our monitor without faults on 8 patients, and then injected faults to the two remaining patients, patient A, and patient H, with different initial glucose values:

- Patient A with a initial glucose value from 80 to 200
- Patient H with a initial glucose value from 80 to 200

5.3 Hazard Detection

We used the notion of the risk index [11] described in Section 4.2 to label the data points in our simulation traces as normal or hazardous and create the ground truth for verifying the safety monitor.

The idea of labeling data is that if the potential risk of insulin over-dose or under-dose crosses a certain threshold and keeps increasing, we mark that data point as hazardous. To be specific, we measure the increase in the risk index for the 10 minutes ahead of time using our patient simulation model and if the current control command for insulin delivery leads to the increase in risk index, we label that data point as a hazardous situation. This is a similar idea as the model-predictive control where the future estimated state of the physical process is considered for deciding on the control action by the controller.

Figure 5-1 shows different hazardous points represented by the taller vertical lines. Insulin rate and risk index value are scaled up to be plotted in the same figure. The X-axis represents the simulation time, with each point representing a 5-minute interval in reality. As we can see, after around 120-time points, a stuck-at fault is injected and blood glucose is fixed at the value of 300. Consequently, this leads to a continuous increase in the risk index (BGI). We mark these points as hazardous and our monitor is expected to detect this situation.

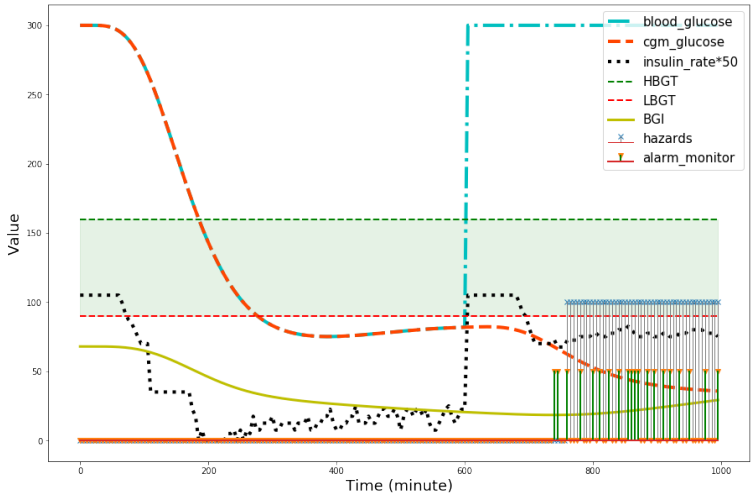

Figure 5-1: Hazardous Data Points and Detection Results

Figure 5-2 shows same simulation as the Figure 5-1 but without injecting any fault. The blood glucose starts at 300 and the controller provides sufficient insulin to bring it back to the normal range. As opposed to the faulty scenario, once the blood glucose comes to the normal level, it remains in the range like no other disturbances (meal, exercise, etc.) has appeared. On the other hand, in Figure 5-1, after injecting fault, the controller gets the faulty BG value (300) and provides insulin based on that. This keeps lowering the actual blood glucose (cgm_glucose) even further and places the patient at risk of having hypoglycemia.

The rules described in the context table play a key role in detecting hazardous conditions. After learning all the thresholds using parameter learning from simulation traces, the rules are then translated into logic code and the monitor uses these rules as a look-up table.

Both the monitor and the OpenAPS controller get the same inputs from CGM and the infusion pump. In Figure 5-1, BG is the blood glucose value used in OpenAPS controller to calculate insulin delivery in the next cycle and cgm_glucose is the glucose value used by the proposed monitor. Both values stem from the same CGM. However, as we are simulating the controller's memory fault, the two different names help to distinguish which value is used by which agent. For

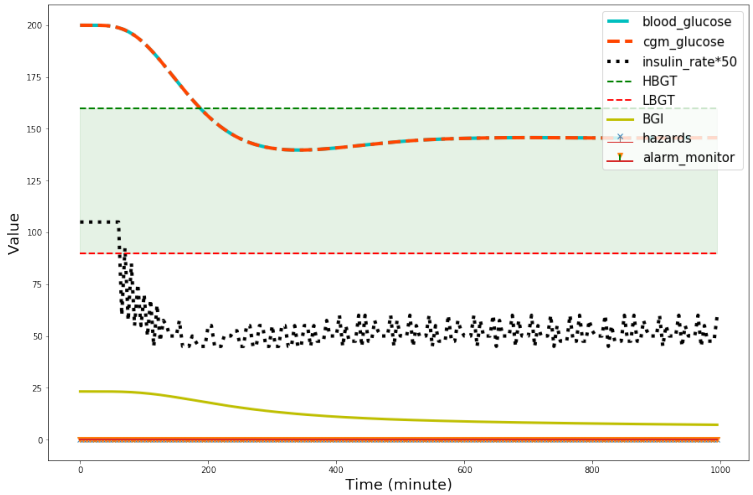

Figure 5-2: Fault-free simulation

example, before injecting the fault, both have the same value. However, after fault-injection, such as the controller's memory fault, BG is stuck at 300 and consequently, the controller increases the insulin delivery that leads to a hazardous situation. However, the blood glucose value received by the monitor (cgm_glucose) is not affected by the fault. Hence, we can see a significant drop in the cgm_glucose value. This is because the controller is totally unaware of the fault and calculated the insulin rate based on the faulty value of the blood glucose.

Figure 5-1 shows the detection of different hazardous conditions. The detection is represented with the shorter vertical lines. The controller and the monitor get the necessary inputs every 5 min and the controller calculates the required insulin in the next cycle while the monitor checks the current context against all the rules in the context table. Violation of any rules causes to raise an alarm. We can see, in Figure 5-1, the monitor keeps raising alarms a little earlier than the hazardous situation starts. This simulation was conducted for a patient model having important body parameters such as body weight 64 kg, insulin clearance rate 15.35 dl/min. The simulation was conducted for 1500 minutes in reality with the initial glucose value of 300.

Table 5.1: Rules of Baseline monitor

Rule No.	**Description**
1	$\phi 1 = \Box(BG>70) \wedge (BG<180))$
2	$\phi 2 = \Box((\Delta\ BG>-5) \wedge (\Delta BG<3))$
3	$\phi 3 = ((BG<\lambda_{10}) \Rightarrow \Diamond_{[0,\alpha]}(BG>\lambda_{10}))$
4	$\phi 3 = ((BG>\lambda_{90}) \Rightarrow \Diamond_{[0,\alpha]}(BG<\lambda_{90}))$

5.4 Context-Aware Monitoring

To evaluate the importance of considering the context in safety monitoring, we did a comparative analysis between our monitor and a baseline monitor. This baseline monitor was implemented based on four data authenticity rules described in the recent work [33]. These rules were defined based on the medical literature and without considering the notion of safety context which is the main focus of our monitor synthesis technique. Another main difference is that our monitor relies on run-time simulation for prediction of the next glucose level in the patient upon execution of a given insulin command to indicate whether the command might lead to a hazard or not. Table 5.1 shows the rules used to implement the baseline monitor.

The first rule in table 5.1 indicates that BG must remain in the range of 70 mg/dL to 180 mg/dL. The second rule states that BG cannot increase by more than 3 mg/dL/min or decrease by more than 5 mg/dL/min. The third rule represents the fact that BG should not stay below the 10th percentile threshold for more than alpha minutes. Similarly, rule 4 indicates that BG should not remain above the 90th percentile threshold for more than alpha minutes.

5.5 Evaluation Metrics

The resilience of the OpenAPS system in the absence and presence of faults and the performance of the safety monitor were evaluated using the following metrics:

- ***Hazard Coverage:*** We measure the coverage of safety-critical faults by the fault injection experiments using hazard coverage, defined as the conditional probability that given an injection of a fault in the system, it leads to an unsafe state or hazardous condition.

- ***MTTH:*** Mean Time to Hazard represents the average time between activation of a fault in the system until the occurrence of a safety hazard. This is an important metric indicating the amount of time budget we have for the detection and mitigation of faults before they lead to safety hazards and incidents.

- ***Detection Coverage:*** Detection coverage represents the percentage of potential unsafe control actions (e.g., an ultra overdose insulin injection) that are detected by our monitor. The true number of unsafe control actions leading to hazards are indicated by the risk index as defined in Section 5.3. We measure the detection coverage using the average number of true positive (TP), true negative (TN), false positive (FP), and false-negative (FN) monitor alerts as well as average micro F1 score. We use the micro F1 score because it considers the number of samples in each experiment.

- ***True-positive (TP):*** True-positive [Figure 5-3] is defined as the generation of alarm before the hazard occurs. This ensures the detection of the hazardous situation earlier than it happens and thus prevent the critical situation.

- ***False-positive (FP):*** False-positive [Figure 5-4] is defined as the generation of the alarm while there is no hazard. This falsely detects the good scenario as hazardous.

- ***True-negative (TN):***True-negative [Figure 5-5] is referred to the fact that while there is no hazard, there is no alarm.

- ***False-negative (FN):***False-negative [Figure 5-6] is defined as the generation of alarm after the hazard occurs. This is the most severe case where the hazardous situation goes undetected.

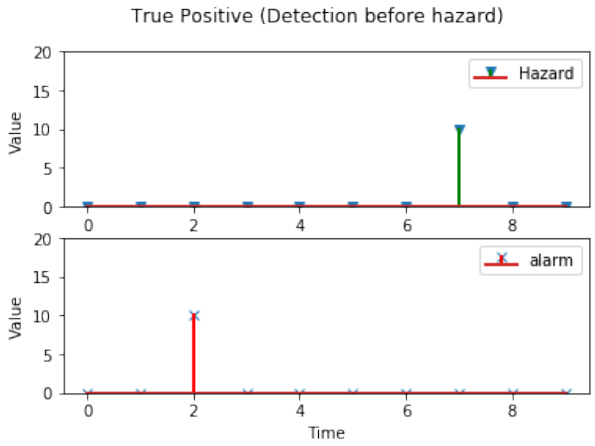

Figure 5-3: True Positive

- ***Detection Latency:*** We measure the time between the activation of fault until the detection of unsafe control action by the monitor as the detection latency. This metric provides us a measure of the fault propagation time in the system and also impacts the reaction or recovery time.

- ***Reaction Time:*** Reaction time is the maximum time that the patient and physicians have to respond to a monitor alert and take action, before a hazard and potential harm to patient happens. We calculate the reaction time by finding the time difference between a monitor alert and the occurrence of a hazard.

5.6 Experimental Results

In this section, we present the experimental results of our case study on the OpenAPS controller. We injected different faults in the OpenAPS controller and show the efficacy of our proposed technique in terms of hazard analysis, safety monitor performance, and time analysis.

Hazard Analysis: In Table 5.3 we present the results of assessing the resilience of OpenAPS controller to different fault scenarios and types.

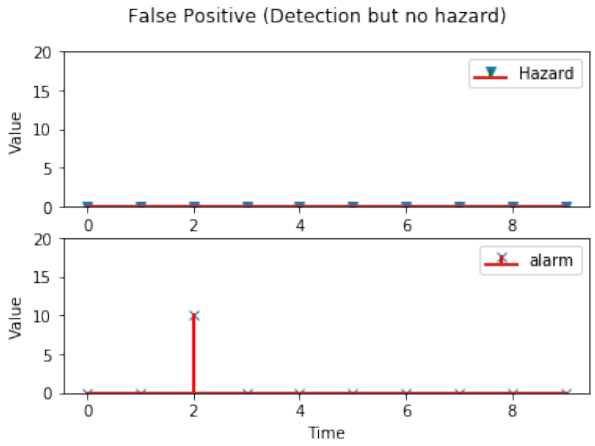

Figure 5-4: False Positive

Safety Monitor Performance: In Table 5.2, we show different performance metrics for both the baseline and the context-aware monitor. We measured the performance of monitors for both the fault-free and faulty system. We further evaluated the performance of the context-aware monitor without and with learning different thresholds. To evaluate the performance in a faulty environment, the system is tested for different faults described in section 5.2. The second and third column in Table 5.2 represents the evaluation for faulty and fault-free scenarios respectively for both baseline and context-aware monitor. The column named Context-aware represents the result of the monitor with learned threshold value while Context-aware_WOT represents the result of the monitor without learning the thresholds.

According to the context table introduced in the former section, we injected 5 types of faults for each of the two variables, blood glucose, and insulin output, to the control software

In Table 5.2, we show different performance metrics for both the baseline and the context-aware monitor. We measured the performance of monitors for both the fault-free and faulty system. We further evaluated the performance of the context-aware monitor without and with learning different thresholds.

To evaluate the performance in a faulty environment, the system is tested for different faults

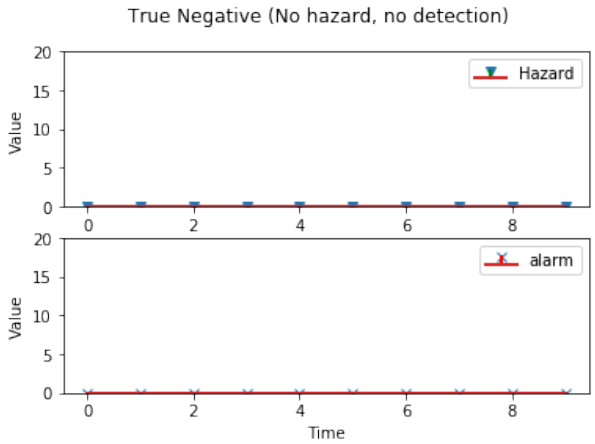

Figure 5-5: True Negative

described in 5.2. The second and third column in Table 5.2 represents the evaluation for faulty and fault-free scenarios respectively for both baseline and context-aware monitor. The column named Context-aware represents the result of the monitor with learned threshold value while Context-aware_WOT represents the result of the monitor without learning the thresholds.

We conducted the fault-free simulations for all the 10 patients with 7 initial glucose (80 to 200 with an interval of 20) value for each. This gives us a total of 70 simulations for fault-free simulations. For faulty scenarios, we conducted a total of 1232 simulations for fault-injection in insulin output and a total of 1456 simulations for manipulating glucose values by injecting faults. That provides a total of 2688 simulations for faulty scenarios. This is represented in row 2 of Table 5.2. Row 3 represents the number of hazardous simulations found out of the total simulation number for both the faulty and fault-free scenarios. This row shows that for faulty scenarios, out of 2688 simulations, 393(14.62%) simulations turn out to contain hazardous scenarios and for fault-free scenarios, out of 70, only 1 simulation (1.43%) contains hazardous data point.

The row named Alernum represents the total number of alert/alarm generated during the entire 2688 simulations. We can see that baseline generates a lot more alarms (1042) than the context-aware monitor (568) while detecting the same number of hazardous scenarios (393). Consequently,

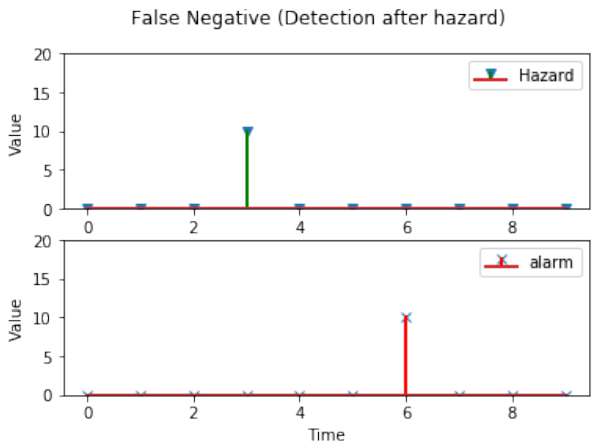

Figure 5-6: False Negative

both monitor have the same number of True Positives (393), however, as baseline generates a lot more unnecessary alarms, it gives a high number of False Positives (649) compared to the context-aware monitor (175). This is shown in the row TP (True-positive) and FP(False-positive) respectively where we can see that the baseline monitor generates 3.7 times number of false positives than the context-aware monitor. In terms of FN (false negative), both the context-aware and baseline monitor perform a great job by not missing any hazardous situations, however, context-aware monitor outperforms the baseline in terms of TN(True-negative): 2120 for context-aware and 1646 for baseline monitor.

The context-aware monitor also outperforms the baseline monitor in terms of Alertnum, TN, FP, and f1 score. However, Context-Aware monitor shows 1 false negative (FP) and 0 true positives (TP) while baseline shows 0 false negatives (FN) and 1 true positive (TP). This is due to the fact that although no fault was injected, there was one case out of 70 was labeled as hazardous. However, not all the hazardous situations lead to accidents and this situation lies in that category. The controller takes appropriate action to mitigate the hazardous situation and prevents from reaching the accidental situation, thus the monitor does not generate any alarm.

We also evaluated the efficacy of the parameter learning scheme by measuring the same metrics

Table 5.2: Summary for monitor performance in fault injection and fault-free cases

Fault Scenarios	**Fault Injection**			**Fault-free**		
No. Simulations	2688			70		
Hazard-num	393 (14.62%)			1 (1.43%)		
Monitors	**Baseline**	**Context-Aware-WOT**	**Context-Aware**	**Baseline**	**Context-Aware-WOT**	**Context-Aware**
Alertnum	1042 (38.76%)	1505 (55.99%)	568 (21.13%)	12 (17.14%)	38 (52.29%)	0
TN	1646	1183	2120	58	32	69
TP	393	393	393	1	1	0
FP	649	1112	175	11	37	0
FN	0	0	0	0	0	1
F1_micro	0.76	0.59	0.93	0.84	0.47	0.99

for the context-aware monitoring with learning and without learning. Context-Aware_WOT monitor represents the monitor without learning. From Table 5.2, it is obvious that without learning, the monitor performs worse than both the baseline and the learned monitor in terms of Alertnum, TN, FP, and F1 score. This shows the necessity of using the learning scheme for being adaptive to patient profiles and context during the monitor development stage.

Time Analysis: In Figure 5-7, we present the timing analysis of the monitors in terms of the time of injecting fault, generating alarms and occurrence of a hazards. We represent the analysis for both the baseline and the context-aware monitor. We further compared the analysis with the context-aware monitor without learning the threshold.

In faulty cases, the reaction time of two monitors (209 minutes for baseline and 176 minutes for the context-aware monitor) is shorter than fault-free cases (270 minutes), meaning that our fault

Table 5.3: Hazard Number and Coverage for Different Fault Scenarios and Types

Scenarios	Fault Type	Total No. Simulation	Hazard Coverage
9	holdrate[a]	112	0.00%
10	doublerate	224	0.00%
11	halfrate	224	6.25%
12	bitflip_addrate	336	41.96%
13	bitflip_decrate	336	29.46%
Summary of Rate		1232	20.61%
14	holdglucose	112	0.00%
15	doubleglucose	112	18.75%
16	halfglucose	112	0.89%
17	bitflip_addglucose	560	20.89%
18	bitflip_decglucose	560	0.00%
Summary of Glucose		1456	9.55%

[a]Increase insulin input.
[b]Insulin input stuck at a very high level like [1, 3].
[c]Decrease insulin input.

injection technique helps to imitate severe situation to examine the monitor's performance. Even in such an adverse scenario, the context-aware monitor can still detect unsafe control commands in a timely manner and leave enough time (nearly 3 hours) for patients and physicians to take any action in response. We also observe that baseline monitor has 30 minutes advantage in reaction time, however, this is at the cost of generating much more unnecessary alerts (see Table. 5.2), which is also underlined by the negative average detection latency (alerts happens before a fault is injected).

In Figure.5-7, the context-aware_WOT monitor has a much longer reaction time than baseline, but keeps a much lower negative detection latency, reflecting the benefit of being context-aware and also exhibiting the drawback of the unknown threshold. It is a trade-off between less number of false positives and longer reaction time. To conclude, with learned threshold, the context-aware monitor outperforms the other two monitor in terms of number of false positives and the accuracy

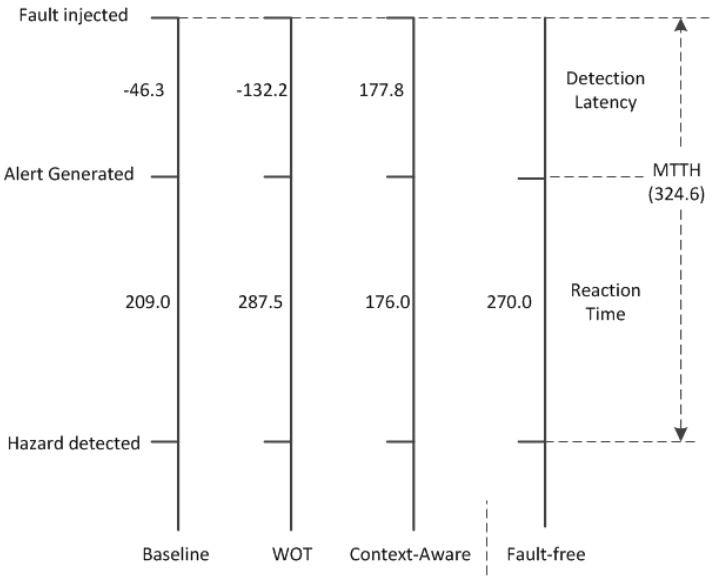

Figure 5-7: Average Fault Propagation Times for Different Scenarios (Unit: minutes)

on predicting unsafe control actions, while keeping a safe reaction time.

Chapter 6

Conclusion

6.1 Conclusion

Recent years have been undergoing a dramatic revolution in the development of safety-critical medical cyber-physical systems (MCPS). MCPSs are life-critical devices and they improve the effectiveness of patient care by providing direct treatment based on the data collected by different sensors. This leads to the increasing use of the medical device in patient treatment in both hospital and outpatient settings. Now, not only a single device but also a distributed system of networked medical devices monitors the state of the patients and sometimes control the different physiological conditions by providing treatment remotely. This obviously makes the patient care so much more effective compared to the traditional treatment system. However, this also increases the complexity and a significant number of challenges in the development of modern medical devices. One of the most important challenges is ensuring the safety of patients that is of utmost importance. However, sometimes due to the lack of sufficient knowledge on the safety and fault tolerance, and the increasing competition among the manufacturers to deploy the device in the market, often the safety aspect in the medical device is ignored.

In this study, we focused on this issue and proposed a general design methodology that automatically synthesizes a safety monitor. We developed a framework for the automated synthesis of context-aware safety monitors for medical CPS from the high-level requirements identified during the hazard analysis process. We used STPA as a systems and control-theoretic hazard analysis approach to identify critical safety context that might lead to the generation of unsafe control commands. Our synthesis approach also utilized the data-driven simulation of the controller along with a dynamic patient model to further refine the logic extracted for the design of the monitor.

To the best of our knowledge, our hybrid model- and data-driven synthesis approach is the first attempt at using the information from the STPA hazard analysis process to generate safety checking rules that can detect and prevent unsafe control commands in CPS control software. Our experiments using a case study of a closed-loop artificial pancreas system in both fault-free and faulty scenarios show preliminary encouraging results on the ability of monitor in accurate and timely detection of unsafe control actions and hazards.

However, there are several limitations to be addressed in the future. One limitation is the frequent checking of the context by the safety monitor (every 5 minutes), while the human body usually takes much longer time to show the effect of insulin on blood glucose levels. Another limitation is the small population of 9 patients used for learning the parameters of the safety monitor. Besides, we learned from all the patients to set the parameters. However, it might not be a good approach to learn from a group of patient and use the monitor to another patient whose body parameter is completely different. A good future direction might be an online learning approach that will continuously learn the threshold from the patient who uses the monitor. Future work might also include focusing on defining more suitable context-checking intervals and learning patient-specific thresholds using a larger population of patients. We also did not consider other factors, such as exercise, meal ingestion and, hormone levels which should be accounted for when monitoring patients with diabetes.

While defining TP, FP, TN, and FN, we followed a very simplified approach where we did

not consider if the alarm is generated too early or too late. In the future, an analysis of alarms at different times before and after the fault injection might provide a more accurate evaluation of the monitors. Also focus on evaluating the applicability of the proposed synthesis approach on a wider range of medical CPS, in particular, those with tighter timing and resource constraints should be considered.

Bibliography

[1] Glucosym. https://github.com/Perceptus/GlucoSym.

[2] Openaps reference design. https://openaps.org/reference-design.

[3] Temporal logic extractor. https://github.com/susmitjha/TeLEX.

[4] Homa Alemzadeh. *Data-driven resiliency assessment of medical cyber-physical systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[5] Homa Alemzadeh, Daniel Chen, Xiao Li, Thenkurussi Kesavadas, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. Targeted attacks on teleoperated surgical robots: Dynamic model-based detection and mitigation. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 395–406. IEEE, 2016.

[6] Homa Alemzadeh, Ravishankar K Iyer, Zbigniew Kalbarczyk, and Jai Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security & Privacy*, 11(4):14–26, 2013.

[7] Maryam Raiyat Aliabadi, Amita Ajith Kamath, Julien Gascon-Samson, and Karthik Pattabiraman. Artinali: dynamic invariant detection for cyber-physical system security. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 349–361. ACM, 2017.

[8] David E Arney, Raoul Jetley, Paul Jones, Insup Lee, Arnab Ray, Oleg Sokolsky, and Yi Zhang. Generic infusion pump hazard analysis and safety requirements version 1.0. *Technical Reports (CIS)*, page 893, 2009.

[9] American Association of Clinical Endocrinologists and American Diabetes Association. Standards of care in diabetes. *Diabetes Care*, 33(1):11–61, 2010.

[10] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 44–57. Springer, 2004.

[11] William Clarke and Boris Kovatchev. Statistical tools to analyze continuous glucose monitor data. *Diabetes technology & therapeutics*, 11(S1):S–45, 2009.

[12] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30, 2017.

[13] FJ Doyle, B Srinivasan, and D Bonvin. Run-to-run control strategy for diabetes management. In *2001 Conference Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 4, pages 3159–3162. IEEE, 2001.

[14] Cody Fleming, Takuto Ishimatsu, Yuko Miyamoto, Haruka Nakao, Masa Katahira, Nobuyuki Hoshino, John Thomas, and Nancy Leveson. Safety guided spacecraft design using model-based specifications. In *Proceedings of the 5th IAASS Conference*, pages 17–19. Citeseer, 2011.

[15] U.S. Food and Drug Administration. Maude adverse event report: Philips medical systems philips intellivue x2 portable patient monitor, mdr report 2154693. https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfMAUDE/detail.cfm?mdrfoi__id=2154693, 2010.

[16] Daniel Halperin, Thomas S Heydt-Benjamin, Benjamin Ransford, Shane S Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 129–142. IEEE, 2008.

[17] Klaus Havelund and Grigore Roşu. Synthesizing monitors for safety properties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356. Springer, 2002.

[18] A M Hersh, E L Hirshberg, E L Wilson, JF Orme, AH Morris, and MJ Lanspa. Lower glucose target is associated with improved 30-day mortality in cardiac and cardiothoracic patients. *Chest*, 154(5):1044–1051, 2018.

[19] Roman Hovorka, Valentina Canonico, Ludovic J Chassin, Ulrich Haueter, Massimo Massi-Benedetti, Marco Orsini Federici, Thomas R Pieber, Helga C Schaller, Lukas Schaupp, Thomas Vering, et al. Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. *Physiological measurement*, 25(4):905, 2004.

[20] Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. Telex: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design*, pages 1–24, 2019.

[21] Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alexander Roederer, et al. Challenges and research directions in medical cyber–physical systems. *Proceedings of the IEEE*, 100(1):75–90, 2011.

[22] Nancy Leveson. *Engineering a safer world: Systems thinking applied to safety*. MIT press, 2011.

[23] Chunxiao Li, Anand Raghunathan, and Niraj K Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *2011 IEEE 13th International Conference on e-Health Networking, Applications and Services*, pages 150–156. IEEE, 2011.

[24] Mathilde Machin, Jérémie Guiochet, Hélène Waeselynck, Jean-Paul Blanquart, Matthieu Roy, and Lola Masson. Smof: A safety monitoring framework for autonomous systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(5):702–715, 2016.

[25] Gianni Marchetti, Massimiliano Barolo, Lois Jovanovic, Howard Zisser, and Dale E Seborg. An improved pid switching control strategy for type 1 diabetes. *ieee transactions on biomedical engineering*, 55(3):857–865, 2008.

[26] Atif Mashkoor and Miklos Biro. Towards the trustworthy development of active medical devices: a hemodialysis case study. *IEEE Embedded Systems Letters*, 8(1):14–17, 2015.

[27] Anitha Murugesan, Mats PE Heimdahl, Michael W Whalen, Sanjai Rayadurgam, John Komp, Lian Duan, Baek-Gyu Kim, Oleg Sokolsky, and Insup Lee. From requirements to code: Model based development of a medical cyber physical system. In *Software Engineering in Health Care*, pages 96–112. Springer, 2014.

[28] Cristina C Oliveira and José Machado da Silva. A fuzzy logic approach for highly dependable medical wearable systems. In *2015 IEEE 20th International Mixed-Signals Testing Workshop (IMSTW)*, pages 1–5. IEEE, 2015.

[29] Miroslav Pajic, Rahul Mangharam, Oleg Sokolsky, David Arney, Julian Goldman, and Insup Lee. Model-driven safety analysis of closed-loop medical systems. *IEEE Transactions on Industrial Informatics*, 10(1):3–16, 2012.

[30] Aakarsh Rao, Nadir Carreon, Roman Lysecky, and Jerzy Rozenblit. Probabilistic threat detection for risk management in cyber-physical medical systems. *IEEE Software*, 35(1):38–43, 2017.

[31] Lenardo Silva, Hyggo Almeida, Angelo Perkusich, and Mirko Perkusich. A model-based approach to support validation of medical cyber-physical systems. *Sensors*, 15(11):27625–27670, 2015.

[32] Meng Wu, Haibo Zeng, Chao Wang, and Huafeng Yu. Safety guard: Runtime enforcement for safety-critical cyber-physical systems. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[33] William Young, John Corbett, Matthew S Gerber, Stephen Patek, and Lu Feng. Damon: A data authenticity monitoring system for diabetes management. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 25–36. IEEE, 2018.

[34] Xi Zheng, Christine Julien, Hongxu Chen, Rodion Podorozhny, and Franck Cassez. Real-time simulation support for runtime verification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(4):106, 2017.