Spring 2019

Testing and Validation Framework for Closed-Loop Physiology Management Systems for Critical and Perioperative Care

Farooq M. Gessa
Bucknell University, fmg005@bucknell.edu

TESTING AND VALIDATION FRAMEWORK FOR CLOSED-LOOP PHYSIOLOGY MANAGEMENT SYSTEMS FOR CRITICAL AND PERIOPERATIVE CARE

By

Farooq Mousal Gessa

A Thesis

Presented to the Faculty of
Bucknell University
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Approved: [illegible]

Advisor: Dr. Philip Asare

[illegible]

Department Chairperson: Dr. Michael Thompson, PhD

Joseph Tranquillo

Engineering Thesis Committee Member: Dr. Joseph V. Tranquillo, PhD

[illegible]

Engineering Thesis Committee Member: Dr. S. Mark Poler, MD

(Date: May 2019)

Acknowledgements

I wish to thank my advisor Dr. Philip Asare for taking me on and giving me the opportunity to work on such an exciting research project. His insightful advice and feedback helped me through my progress.

I am also thankful to my committee advisors Dr. Joseph V. Tranquillo and Dr. Mark Poler (MD), this thesis would not have been possible without their continued support and guidance.

My deepest gratitude to Dr. John Mcilwane for taking the time to explain the infusion protocol.

Dikendra, Win, Adit, Raphael and Sarah, the open connected medicine group, you have been such great colleagues.

Special thanks to my family and friends for the moral support.

Table of Contents

Chapter 1
Introduction

1.1 Background

In today's modern operating room, care of the surgical patient during an operation is provided by a team of individuals referred to as the care team. The care team monitors the patient's physiological status by regularly checking the vital signs on display monitors and other devices. Usually, the care process requires balancing administration the appropriate amounts of intravenous fluids and drugs, often employing infusion pumps whose parameters may be manually adjusted to keep the patient's vital signs at desired levels. During complex surgical procedures, more fluids are required. This necessitates the simultaneous use of multiple pumps and, in most cases, these pumps differ in purpose and monitored parameters. Further, utilization of multiple pumps can prove to be cumbersome, requiring meticulous attention to input parameters to avoid fatal medical errors.

To assist the team in such circumstances, we are designing and developing a proof-of-concept computer-based system also known as a closed-loop assistant (CLA). This system will handle all communication and connections between devices and evaluate data obtained from monitoring equipment. The system will also process the collected data and execute changes to the infusion pump parameters in order to let the care team concentrate on other tasks. A conceptual picture of the CLA is shown in Figure 1.1.

The focus of this thesis was on testing and validating systems like the CLA. Without proper testing and validation tools, it is very difficult to design a CLA that is safe and effective for use with patients.

1.2 Motivation

Since the care team will rely upon the stability of the CLA throughout the patient's surgery experience, it is crucial that the CLA is accurate to ensure the safety of the patient and success of the surgery. The CLA should have the ability to perform fundamental tasks such as informing the team of technical glitches like lost connection to other devices. It should also be able to yield control to teams in similar situations.

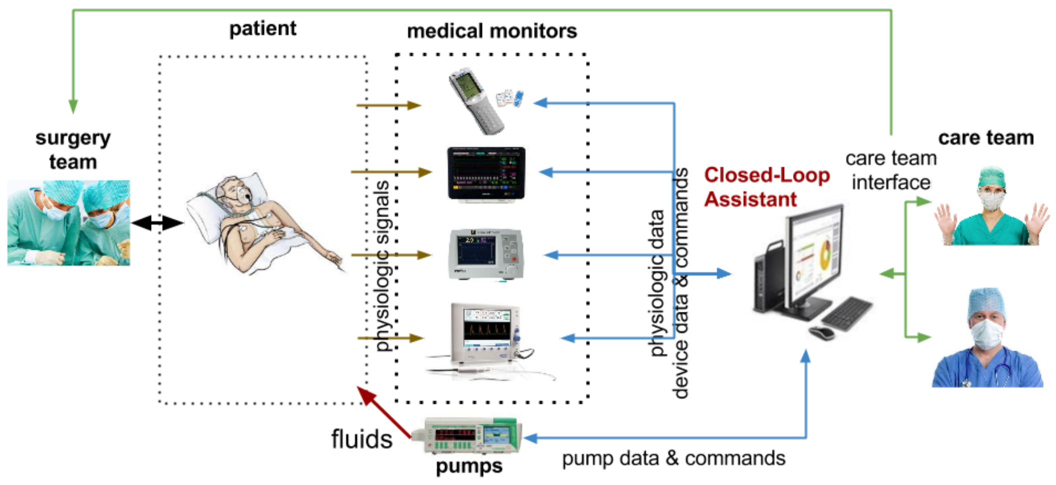


Figure 1.1. *CLA concept*

Patient safety being a major concern, methods for verifying and validating the CLA, in order to guarantee reliability, need to be developed. In addition, there should be a way to confirm that the designed system meets its original specifications. One way to do this would be to perform human trials but this process is complicated, potentially hazardous, not reproducible, and expensive. Thus, human trials necessitate intricate safety precautions. Animal studies are only somewhat easier, having similar limitations. Furthermore, performing repeatable physiological trials to evaluate design modifications is functionally impossible.

Models have been used before in this process. For example, in the study of glucose regulation in patients with type 1 diabetes, *in silico* patients have been developed and used to provide a basis for testing as well as optimization of diabetes treatment [1]–[5]. The same technique has been used to verify the safety properties of a closed loop system which consists of the heart and a pacemaker [6]–[8]. Therefore, to achieve the desired test coverage without worrying about test failures that can harm patients, we have to design a software patient model that simulates human physiology.

For more realistic testing, this patient model can be connected to the rest of our system through a physical simulator that generates and sends signals to the monitoring devices. The patient model is designed to mimic human dynamics and responses and therefore can be used to test our system in a closed loop. By mimicking the behavior of the environment in which the CLA is expected to be used, the *in silico* model will allow us to validate the CLA's performance by testing the software algorithm that controls the

CLA and observe its interactions with the rest of the system. Consequently, we will learn the response of the model to variations in the CLA's parameters. Having such a model will enhance safety by allowing us perform potentially dangerous tests in a virtual environment.

1.3 Research Objectives

My research broadly focused on developing tools for testing systems like the CLA. Such tools are critical to the development of CLA technologies. Figure 2 shows the conceptual picture of the CLA as well as the identified key enabling science and technologies. The patient models mentioned previously as critical to testing CLA interaction with patient physiology is indicated as T1. The intelligent algorithms and the platform on which they run which connect them to the various medical devices are labeled as T2 and T3 respectively. For this thesis we did not focus on the interaction between the CLA and the care team (T4), though this is important for full CLA validation.

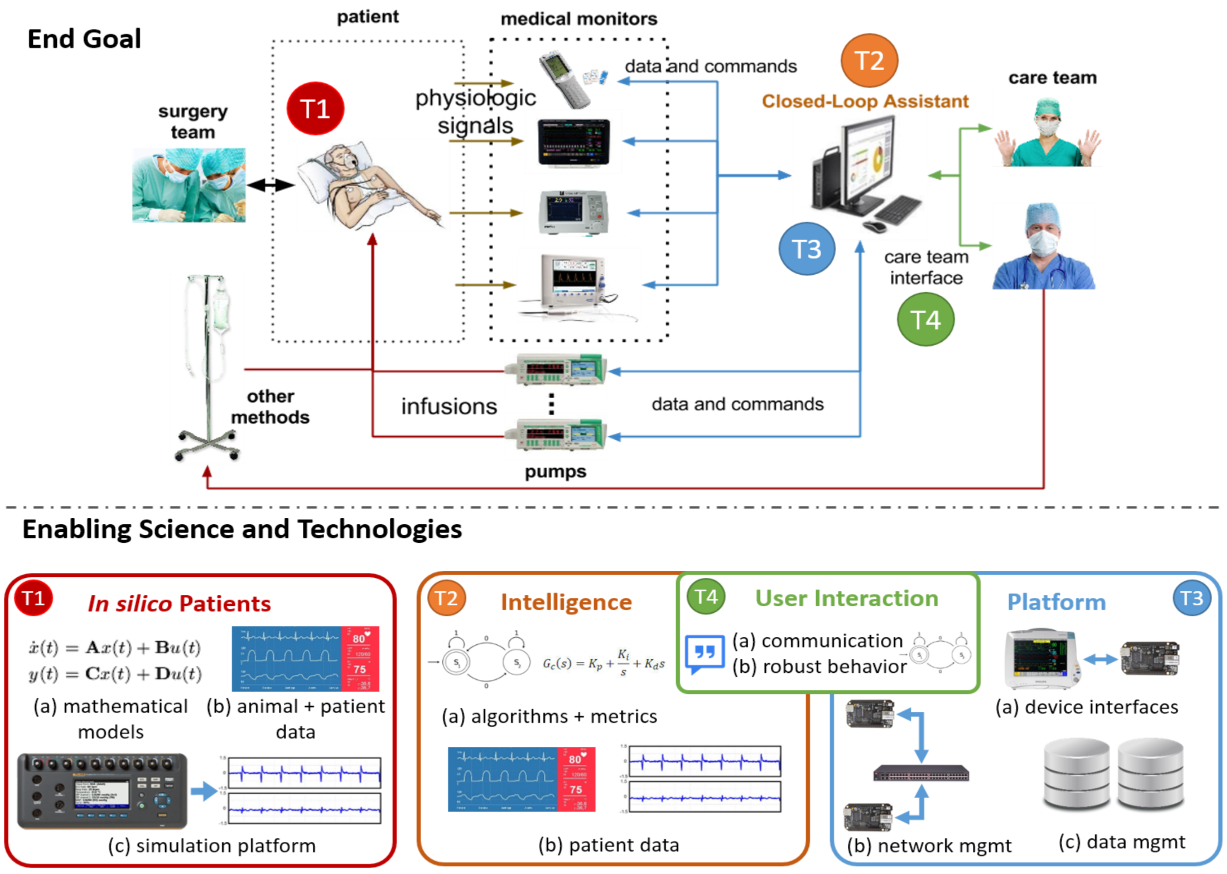


Figure 2. *The CLA concept with enabling science and technologies highlighted.*

Due to the distributed architecture, networking equipment, and interactions between cross-vendor devices present in the CLA, there is uncertainty in the system behavior. Testing allows to address this uncertainty because it facilitates the detection of defects and evaluation of system performance before deployment.

The two main objectives of my thesis were:

1. Develop pure software simulation of CLA-patient interaction to provide early insights into potential CLA behavior (Chapter 3)
2. Develop a framework for connecting a virtual patient model (T1) with a CLA (T2 and T3) consisting of real medical devices to facilitate realistic real-time, system-in-the-loop testing of CLA to complement the results obtained for pure software simulations in the first objective (Chapter 4).

Chapter 2
Background and Literature Review

2.1 Closed-Loop Systems in Medicine

One of the more well-known closed-loop system used in medicine is the pacemaker. It is a device that is implanted underneath the skin that senses the activity of heart (and sometimes other factors), and sends electrical signals to slow or increase the heart rate [9], [10]. A more recent system is the artificial pancreas [11], [12] used in the management of type 1 diabetes [13]. It consists of a continuous glucose monitor and an insulin pump, and algorithm (either implemented on the pump or separate device), that monitors the blood glucose and adjusts the insulin infusions based on this sensed information. These two systems share some similarities. In both cases the patient condition is chronic (or permanent). The devices compensate for lost function in the body. Both these systems are also used by the patient outside the clinical environment and provide care that enhance the quality of life for the patient.

There are closed-loop systems that are used in the clinical environment as well. Patients during surgery or in the intensive care unit (ICU) sometimes temporarily lose the ability to control their blood glucose properly. There is a commercial available computer-system called Glucommander [14]. [15] that directs clinicians on how to adjust insulin infusion based on the glucose test results input to the computer by the clinician. Although this can be described as more of a decision support system, the system can also be considered closed-loop because it decides on the infusion. There is also recent work in closed-loop management of fluids [16], [17], blood pressure [18], and anesthetics [19] during surgery. These systems are examples of the closed-loop assistant (CLA) technology that is the focus of this thesis. They focus on treatment of specific conditions that are temporary and are used under the supervision of clinicians in the clinical environment. In addition, CLAs have to deal with more complex physiological conditions since there can be multiple simultaneous treatment actions going on.

2.2 Testing of Closed-Loop Medical Systems

At different phases of development, closed-loop systems undergo various tests. Early in development, simulation is used to validate ideas and algorithms. When a feasible prototype is developed, pre-clinical trials using animals is usually undertaken. At later stages in the development various phases of human trials would be conducted [20].

However, not all closed-loop systems may go through all these phases. Where the closed-loop system can start out as a decision support system, animal trials are sometimes skipped altogether. The Glocummander systems [14] seems to have started off this way as an automation of an already existing clinical protocol. In this case the system functions as a decision support suggesting the new infusion but leaving the option of whether to follow that suggestion or do something else to the clinician who then implements whatever decision was taken.

For the diabetes case, a simulator was developed that was eventually approved as an alternative to animal trials [1], [4].

2.3 Testing of Medical Devices Using Simulations

As mentioned previously, simulations are typically used earlier in the development process to test and validate algorithm ideas. Recently, however, there has been interest in leveraging simulations for testing much later in the design and development process. The diabetes case, where the simulator has a patient population in addition to realistic models of the medical devices involved is one example. The pacemaker verification case mentioned earlier is another example. In that work, a hardware version of the virtual heart model that can interact with real pacemakers was developed [21]. The ability to run a test of a prototype of the real medical system with a simulated patient is typically called *in silico* testing.

There is recent interest by the FDA in *in silico* testing as part of the evidence provided to support the safety and efficacy claims made for closed-loop systems [22]. A recent paper by FDA authors reviewed the use of simulations in testing various closed-loop systems and offered suggestions on ways in which these simulations could be used to support device claims, focusing on the validity of the simulation results [23].

The work in this thesis is aimed at provide such testing capabilities for CLAs at the later stages of the design process in the hopes that this could help with safety and efficacy analysis of these devices for the regulatory process.

Chapter 3
Software Simulation of CLA-Patient Interactions[1]

3.1 Motivation

The most ideal controllable and repeatable testing scenario for a CLA is with real devices interacting with a realistic patient model. Pure software simulations, however, provide a lower overhead but useful environment in which to explore early designs. Because pure software simulations ran much faster than real-time, many scenarios can be explored quickly. Even in pure software simulation, some non-idealities in system behavior can be represented giving faster insights into potential issues and allowing for improving early concept designs before committing to hardware and software that is closer to the final product, which is also more expensive to iterate on. This chapter focuses on a software framework we developed that allows for quicker early explorations to complement the higher-fidelity real-time framework.

3.2 Architecture Overview and Design

A conceptual representation for the framework is shown in Figure 3.1. It is comprised of models of medical devices such as monitors and infusion pumps, a physiology management algorithm, and a patient physiology model. The monitor models the interaction with the patient to derive vital signs that are reported periodically to the algorithm. The pump models taking instructions from the algorithm and turning them into infusion actions which results in fluids or drugs entering the patient at a particular rate. The algorithm models software that receives data periodically for the monitors, decides on actions (or inactions) based on this data and then instructs the pump on the rate of infusion of various fluids or drugs. Although the platform that enables connectivity between the devices and the algorithm and also runs the algorithm is not shown explicitly, the communication between the different parts are modeled and real-world issues like loss of data and delays are captured in a conceptual manner.

[1] Most of the information in this chapter is reproduced from [24], which we presented at the 2018 Medical Cyber-Physical Systems Workshop in,Porto, Portugal

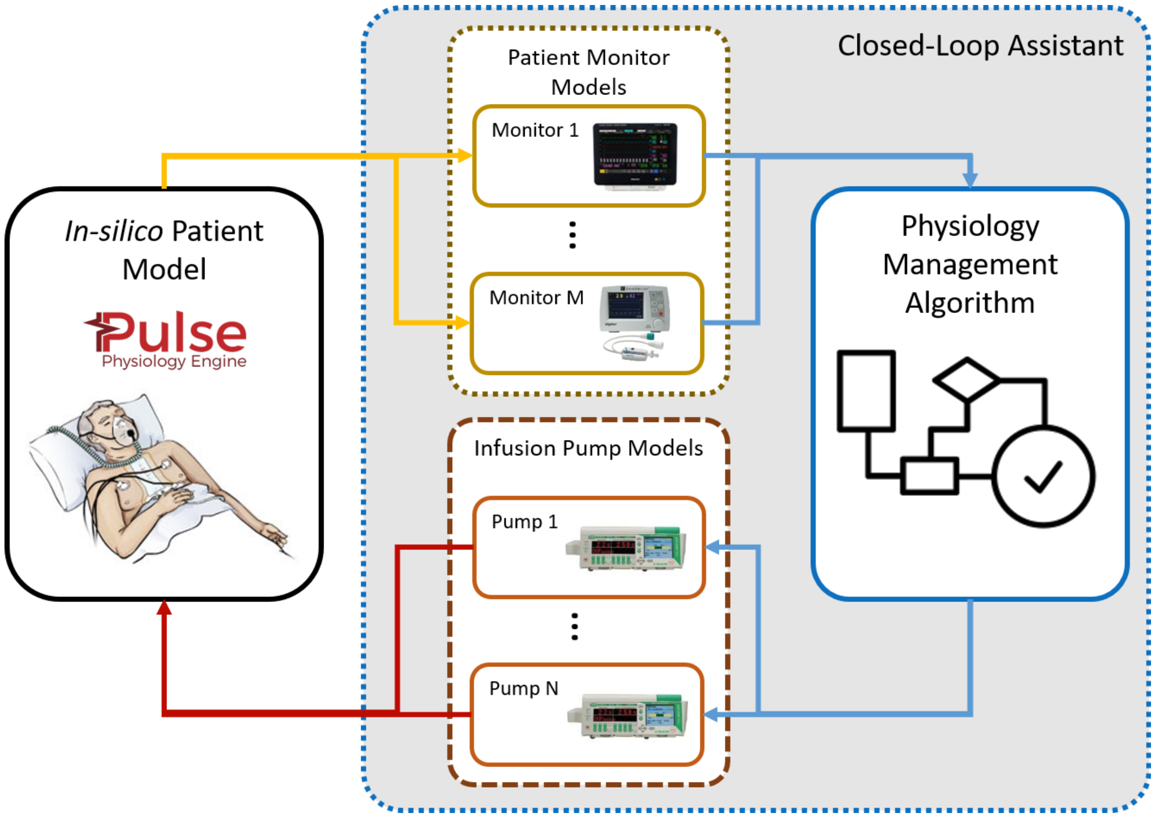


Figure 3.1. *Conceptual architecture of software simulation framework for CLA-patient interactions*

In spite of the fact that a only single a physiology management algorithm is demonstrated in Figure 3.1, the framework is capable of handling interactions between devices and several algorithms (with the potential for algorithms to interact with each other). In addition, variations in any part of the architecture (patient, devices, algorithm(s)) can be explored. The exploration of variations of such scenarios is facilitated by the modular architecture of the key components of the framework. For example, given a particular patient, we were able to explore the impact of various renditions of the CLA. This can be achieved by changing either the devices or algorithm, or their parameters. Likewise, we can also explore interactions between a particular CLA instance with different patients. This allows us to undertake a design space exploration using a patient population instead of a single patient model. The use of the patient population allows us to understand how robust a particular design is to inter-patient variability. This can also allow us to begin to see the potential safety level of our early concept design using some of the techniques developed in [25]. The conceptual picture of this design space exploration is shown in Figure 3.2.

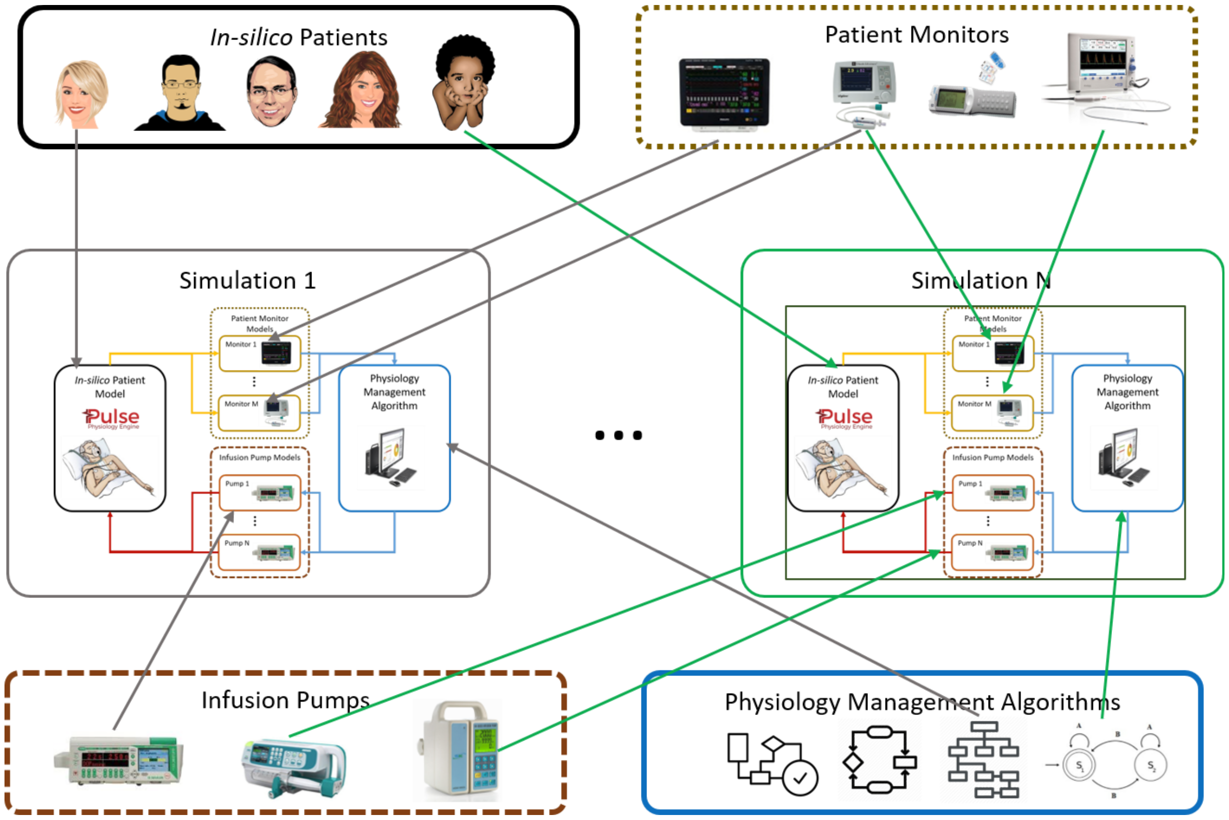


Figure 3.2. *Conceptual diagram showing the design space exploration using in-silico patient models*

3.3 Implementation

The software simulation framework is implemented using the Pulse Physiology Platform [26], an open-source physiology platform based on well-validated models that provides a software development kit (SDK) for interacting with the physiology models that allow you to control and extract patient state, and present inputs to the patient (e.g. infusion of fluids, or consumption of meals). We build our framework by developing models of monitors, pumps, algorithms, and the connections between these, and leveraged the Pulse SDK to connect the relevant parts to Pulse and to control to the overall simulation.

3.3.1 Pulse Physiology Engine

One of the main objectives of the Pulse Physiology Platform is to aid in the design, development and the adoption of physiologic modeling. Through the creation of a modular and extensible ontology for the simulation of human physiology, Pulse can speed up model development and also simplify integration with third party (for both hardware testing and software simulation). To facilitate its integration, the Pulse platform comprises a Common Data Model (CDM), a software framework, and the Pulse Physiology Engine, as indicated in figure 3.3. Pulse has been used in a number of medical simulations (e.g. [27], [28]), has a sizeable user community, and is developed by a company known for delivering quality open-source tools useful for medical and other applications. These reasons make it a good choice for software on which on our framework is based.

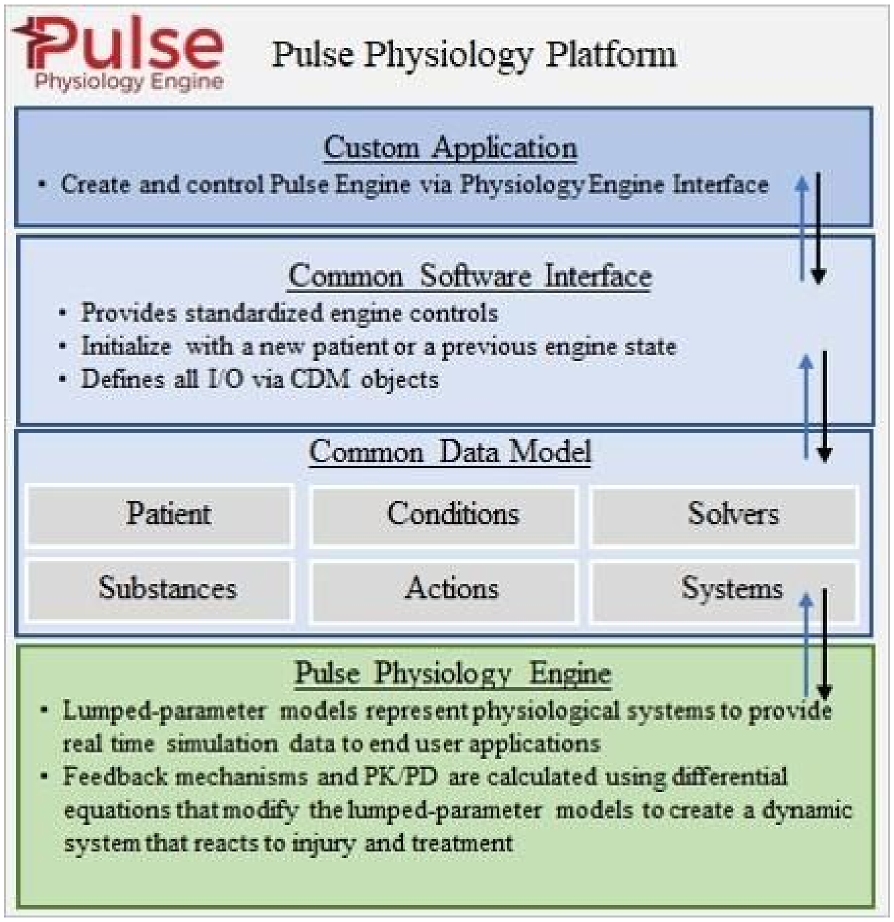

Pulse
Physiology Engine
Pulse Physiology Platform
Custom Application
• Create and control Pulse Engine via Physiology Engine Interface
Common Software Interface
• Provides standardized engine controls
• Initialize with a new patient or a previous engine state
• Defines all I/O via CDM objects
Common Data Model
Patient
Conditions
Solvers
Substances
Actions
Systems
Pulse Physiology Engine
• Lumped-parameter models represent physiological systems to provide real time simulation data to end user applications
• Feedback mechanisms and PK/PD are calculated using differential equations that modify the lumped-parameter models to create a dynamic system that reacts to injury and treatment

Figure 3.3. *Pulse Physiology Platform software architecture.*

3.3.1.1 Physiology Models

This engine is built on top of various abstract models (model architecture illustrated in Figure 3.4), that are used to simulate the feedback mechanisms and interactions between the systems, pharmacodynamics/pharmacokinetics (PD/PK) and the medical equipments like the anesthesia machine. These models are meant to represent the different systems of the body.

The particular systems of the body consist of numerical models which use of circuit analogues (like capacitors and resistors) to approximate behavior of a region of interest or even system of the entire body.

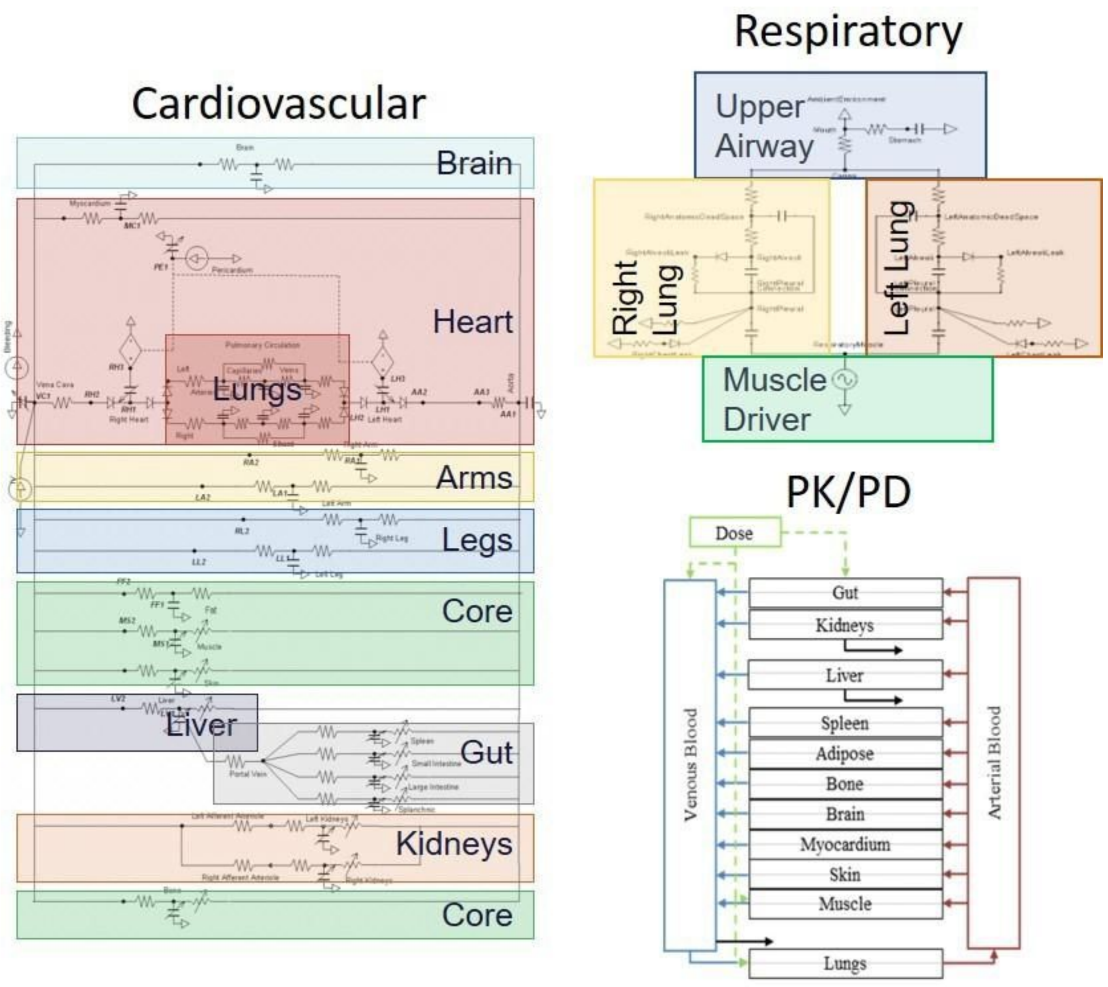


Figure 3.4. *Pulse Physiology Engine model architecture.*

The PK model symbolizes substance movement, particularly the movement of drugs through the whole body by means of clearance and diffusion focusing on the plasma concentration after a period of time. The moment the plasma concentration has been calculated accurately, the drug effects or even the PD effects on the vital signs are implemented. This particular model is just like the feedback mechanisms, since whenever the concentration of plasma increases, the lumped-parameter model [29] is modified which turn adjusts the individual parameters such as the heart rate, blood pressure and respiratory.

Also, the Pulse mechanisms that link the different systems together are important in the modeling of the impact of multiple interventions (for instance, combining drug infusions and ventilations) and also side effects of any interventions (for instance, interventions that only reduce blood pressure will lead to an increased heart rate).

For additional flexibility, Pulse provides for patient variability, eight patients are included by default in the Physiology Engine's repository. These patients are characterized by a several parameters that can be modified through the patient files in order to tweak their baselines. The individual files are then utilized in the initialization of the computational. Only after the engine stabilizes can conditions such as chronic obstructive pulmonary disease and renal stenosis be applied to the patient. This process allows for analysis of different health conditions and treatments. Additionally, it is possible to add custom patients by creating new patient files and changing values for the different parameters within those files.

3.3.1.2 The Common Data Model and Common Software Framework

As earlier stated, the Pulse architecture was specifically designed for reduction of model development time and increase the usability of the engine in simulations through a modular, extensible software system that represents the human physiology. For these goals to be achieved, the Pulse architecture provides a Common Software Framework and a Common Data Model (CDM).

The CDM specifies the data and the associated relationships with physiology simulation software in a dictionary-like format. The purpose of the Common Software Framework was to speed up the development time through the provision of a place to where common reusable algorithms can be implemented so as to ensure the basic functionality between programs is maintained. Consequently, the implementation of these common algorithms, simplifies reusability and validation which greatly speeds up model development time and ensure consistency in the results.

A single interface for both inputs and outputs and also controlling the engine defines all the methods, based on the CDM. This is an interface that enables hardware and software developers to integrate Pulse into their respective application. It provides a standardized approach for interacting with Pulse. For instance, controls to injecting messages, output computed data values and advancing the engine time are provided.

3.3.2. Modeling devices

The CLA comprises of algorithms (assumed to be on a separate computer) and medical devices as stated previously. In the software framework these devices are abstract models of patient monitors and pumps. Interaction between these device components and Pulse is achieved by utilizing Pulse's engine interface to directly observe the patient state and take appropriate action when needed. Similarly, the algorithm components indirectly observer and effect action on patient state by interacting with the device components.

3.3.2.1 Patient Monitors

Patient monitors directly interact with the Pulse physiology engine to access the patient's physiologic parameters. They can query these parameters at a rate whose equivalent sample period is a multiple of Pulse's timestep of 20ms (i.e. the monitor can get a value from Pulse every 20·N ms where N = {1,2, ...}). At the same time, patient monitors can output these physiologic variables to algorithms at a rate whose equivalent period is an integer multiple of its input sample period (i.e. the monitor can output values every inputPeriod·N where inputPeriod is how often it gets a value from pulse and N = {1, 2, ...}). Each one of these variables can be sampled and produced by the monitor at different rates provided there is a valid relationship between the rate at which a variable is sampled from Pulse and its output rate.

The variables passed to the algorithm from the patient monitors can either be unaltered values (i.e. as directly sampled from Pulse) or computed values (i.e. calculated from the input variables especially if the computed value requires multiple sequential unaltered values). For instance, we could model a monitor and computes the mean arterial pressure (MAP) from the systolic and diastolic blood pressures values in spite of the fact that it can we can directly obtain it from Pulse. Another example would be computing the value of stroke volume variation from a sequence of multiple stroke volume samples from Pulse.

Inaccuracies in values can be modeled by adding deviations from the true values before they are output to the management algorithms. Though the current implementation does not support modeling inaccuracies, it is extensible to support the inaccuracies plus other behaviors. This should be in the future revisions as more cases are being developed.

3.3.2.2 Physiology Management Algorithms

These receive patient physiologic data from the monitors and query any of the accessible physiological parameters from the patient monitor at rate whose equivalent sample period is a multiple of the monitor's output rate for that parameter (i.e. outputPeriodFromMonitor·N where N = {1, 2, 3...}).

Whenever new data retrieved, the algorithm can decide to either make a decision based on the current input or delay the decision until enough samples are collected. The algorithm will send a command instructing the pump to take specific actions depending on the on the decision. In addition, it is capable of making decisions in-between data arrivals since it has the opportunity to act in each time step of the simulation.

Since the case study we are working with operates on the order of minutes, we currently do not model any issues of the algorithms executing on a computational platform. In future as we encounter cases with finer time scales, reasonable issues like computation time causing delayed decision making and interventions will be included.

3.3.2.3 Pumps

The pump conceptually delivers medicines and fluids into the patient at specific rates. Pumps can receive and execute instructions which include start, adjust, or stop infusions. Generally, they have a delay parameter that models computation time and time for mechanical components to adjust to a command from the algorithm. To model an ideal situation without physical effects this delay can be set zero. Inaccuracies such as adjusting the rate before the infusion is applied or during the application of infusion between adjustments can also be modeled. However, during our simulation the delays were modeled but the framework can be easily extended to support inaccuracies as well.

3.3.3 Running a simulation

The framework execution cycle relies on Pulse and because of this dependency, the system is advanced at the same speed as Pulse which updates at discrete time steps each representing 20ms of wall clock time in manner corresponding to a synchronous reactive computation model [30].

Through the simulation engine component, the framework controls scenario events specifically actions that happen to the patient independent of the CLA. As long as all other components have been initialized, the execution cycle of the simulation engine we built on top of Pulse is as follows:

(1) Check for scenario events (including stop conditions)
(2) Set up and apply any scenario events to patient
(3) Execute one timestep of Pulse
(4) Execute patient monitors
(5) Execute algorithms
(6) Execute pumps

The update function in each of the CLA components is invoked on each timestep to act on the inputs at the point in time or not. There is also configuration file associated with the system that allows the user to set and modify certain properties of the devices such as the input and output rates, simulation running time, among others. The rate information from the configuration file is used to regulate when inputs and outputs are taken in and produced respectively. The benefit here, is that all changes to the properties in the file can be made without the need to recompile the code.

3.3.4 Simulation Development

Developing a simulation in our framework entails writing C++ code as well as putting information in a configuration file. The configuration settings are read and loaded into the CLA components using the provided methods. Inside these methods we perform basic error checking for example, confirming whether input and output out rates of the different models align. Provided there are no errors, an executable file is created in The folder where Pulse expects its executables to run from. That created executable is what runs the simulation.

3.4 Case Study and Results

We developed and simulated a case study simulation based on low blood pressure management in the ICU to demonstrate what the framework can do and the sorts of explorations we hope to avail users.

In light of the discussion with Dr. John McIlwaine, DO, Medical Director for the Center for Telehealth and eICU at the Geisinger Health System about they manage low blood pressure cases, a greater part of what is presented here is specific to Geisinger, however it is in accordance with general practice at other medical institutions. For demonstration purposes and simplicity's sake, several things were left out, in contrast to how it is actually done in hospitals.

3.4.1 Clinical Scenario and General Strategy for Intervention

Low blood pressure (hypotension) usually categorized as mean arterial blood (MAP) below 65mHg [31], [32] is not uncommon among patients in ICUs particularly those under post-operative care. It is crucial that MAP is regulated to ensure oxygen delivery to all organs in the body. One of the ways hypotension is managed is by infusing the patient with both saline and a vasopressor drug such as norepinephrine meant to narrow the blood vessels. Concurrently, the doctors may attempt to determine and address its underlying cause during this process.

The case study was centered around the intervention using norepinephrine infusion. A quasi-hybrid system shown in Figure 3.5 is a representation of procedures followed during the infusion process. The primary objective of hypotension management is to get the MAP into a tolerable range (i.e., between 65 mmHg and 85 mmHg) and keep it within that range.

The procedure comprises three parts. The first part involves the initial response to the hypotension where the patient is hit with the maximum allowed dose of the drug and fluid infusion is also started (the initial transition in Figure 3.5). In our case study this was maintained at a constant rate thus it was not included in Figure 3.5. The system then waits for the MAP to increase (in the state labeled "Wait for increase" in Figure 3.5), checking every five minutes.

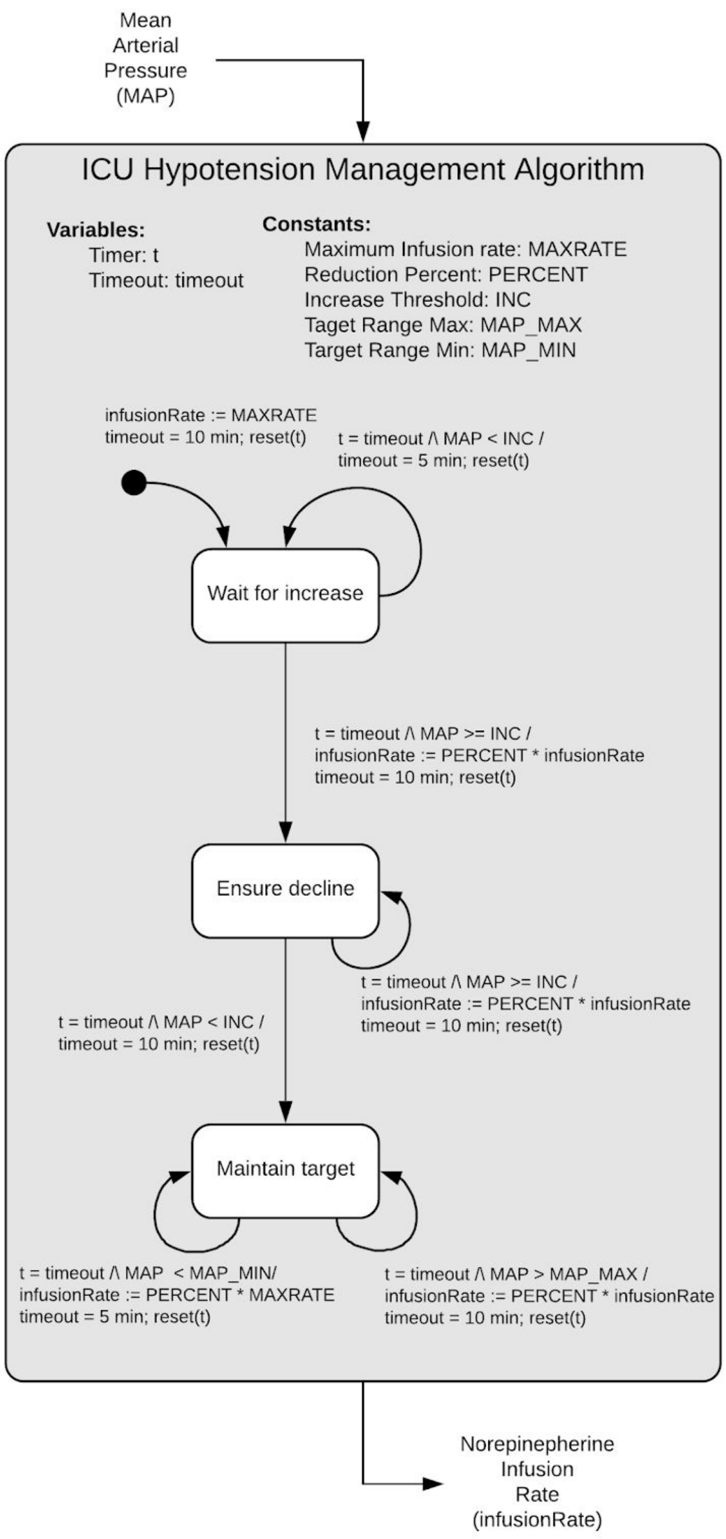
The second part starts after the MAP has increased to or beyond the threshold (denoted by INC in Figure 3.5), which is usually around 100 mmHg. The dose is reduced to some fraction of the maximum rate to avoid issues like high blood pressure and other side effects that can result from infusing a high dose of the drug. The dose is reduced until the MAP stops rising and begins to drop.

When the MAP drops to within the target range (denoted by MAP_MAX and MAP_MIN in Figure 3.5), the third part which ensures that MAP is maintained within this range (usually 65 mmHg and 85 mmHg) begins. Whenever the MAP falls below or goes above this range, the dose is increased and reduced respectively to counter this change. This process can continue for up to 24 hours or more, or until it is confirmed that infusion of norepinephrine is no longer required. The MAP is consistently checked every 10 minutes between actions, however, when it falls below range (i.e. 65 mmHg) the frequency increases to every 5 minutes.

3.4.2 Simulation Setup

To explore the variability in both the CLA and the patients it must interact with, we examined two slightly different infusion protocols across a small patient population using the framework. We used three of the patients provided with Pulse. The three patients were selected because they were able to withstand the initial hemorrhaging used to create hypotension and the result intervention. The parameters tha Pulse uses to define these patients are shown in Table 3.1. For more information on how Pulse uses patient parameters to define different physiologies see [33].

We mimic postoperative hypotension in every patient by starting off with a lower baseline MAP and hemorrhaging them until the MAP falls to 70 mmHg, whereupon the protocol is triggered. For each patient, we ran the whole scenario for one hour of simulation time.



Mean Arterial Pressure (MAP)
ICU Hypotension Management Algorithm
Variables:
Timer: t
Timeout: timeout
Constants:
Maximum Infusion rate: MAXRATE
Reduction Percent: PERCENT
Increase Threshold: INC
Taget Range Max: MAP_MAX
Target Range Min: MAP_MIN
infusionRate := MAXRATE
timeout = 10 min; reset(t)
t = timeout /\ MAP < INC /
timeout = 5 min; reset(t)
Wait for increase
t = timeout /\ MAP >= INC /
infusionRate := PERCENT * infusionRate
timeout = 10 min; reset(t)
Ensure decline
t = timeout /\ MAP >= INC /
infusionRate := PERCENT * infusionRate
timeout = 10 min; reset(t)
t = timeout /\ MAP < INC /
timeout = 10 min; reset(t)
Maintain target
t = timeout /\ MAP < MAP_MIN/
infusionRate := PERCENT * MAXRATE
timeout = 5 min; reset(t)
t = timeout /\ MAP > MAP_MAX /
infusionRate := PERCENT * infusionRate
timeout = 10 min; reset(t)
Norepinepherine Infusion Rate (infusionRate)

Figure 3.5. *State machine representation of hypotension management algorithm (i.e. the norepinephrine infusion protocol).*

Table 3.1. *Patient Parameters*

Parameter	**Value for Patient**		
	Diana (ExtremeFemale)*	**Gus**	**Hassan**
Sex	Female	Male	Male
Age (yr)	18	32	28
Height (in)	54	70	72
Weight (lbs)	90.2	190	185
Body Fat Fraction (0 to 1)	0.32	0.18	0.18
Right Lung Ratio (0 to 1)	0.5	0.525	0.525
Systolic Blood Pressure Baseline (mmHg)	90	90	90
Diastolic Blood Pressure Baseline (mmHg)	60	60	60
Heart Rate Baseline (BPM)	60	93	110
Respiration Rate Baseline (1/min)	12	14	18

* *See Pulse Patient Methodology [33]*

By parameterizing the infusion protocol (depicted in Figure 3.5) and tweaking one of the constants we were able to get two distinct strategies. That is, living all other parameters unaltered, we changed the only the percentage reduction in the dose in both algorithms. Table 3.2 shows the full infusion protocol parameters for both strategies.

For simplicity in the software framework, the modeled patient monitor outputs MAP directly from Pulse as opposed to computing it from systolic and diastolic pressures like real monitors do. We also assume the value is accurate and does not introduce any error.

Table 3.2. *Algorithm and General Simulation Parameters*

Parameter	**Symbol in Figure 3.5**	**Value**		**Units**
		Algorithm 1	**Algorithm 2**	
Patient MAP at start	-	70		mmHg
Saline rate	-	50		ml/hr
Norepinephrine concentration	-	16		μg/ml
Maximum norepinephrine infusion rate	MAXRATE	84	84	ml/hr
Percentage reduction in norepinephrine infusion rate	PERCENT	75	87.5	%
MAP threshold to start reducing dose	INC	80	80	mmHg
Upper threshold of MAP target range	MAP_MAX	80	80	mmHg
Lower threshold of MAP target range	MAP_MIN	75	75	mmHg

The timing of monitor, algorithm, and pump in terms of how often data is acquired, output, or decided on is shown in Table 3.3. For the algorithm, we considered two scenarios in terms of how often it checked and decided on the MAP. The first is based on information from Dr. McIlwane, which represents what clinicians do (we called the clinician behavior or the baseline), where the algorithm checks every 5 or 10 minutes depending on the algorithm state. This represents realistically what clinicians in the ICU are able to do given that they have multiple patients to check on as well as other clinical tasks to perform. The second scenario has the algorithm checking once every minute, since this is typically the rate at which data is recorded into electronic health records [34], and represents the capability of a computer system being able to monitor more frequently than a human.

Table 3.3. *Model Timing Parameters*

Parameters		**Value**		**Units**
		Clinician Behavior (Baseline)	**Computer System Behavior**	
Monitor Sampling From Patient	Rate	2		points/minute
	Period	30		secs
Monitor Output to Algorithm	Rate	1		points/minute
	Period	60		secs
Algorithm check period	Wait for increase state*	5	1	minutes
	After Wait for increase state (MAP > MAP_MIN)*	10	1	minutes
	After Wait for increase state (MAP < MAP_MIN)*	5	1	minutes
Pump Infusion Action Delay		15		secs

** See Figure 3.5*

3.4.3 Results

3.4.3.1 Example Output

Figures 3.6 shows plots of the various information extracted from an example simulation. The results are shown in such a way that they highlight how patient data is processed before it is presented on the monitor in a real patient monitor. For instance, the patient monitor has to convert the sensed input signals into digital values for further refinement before reporting them as the final observed values. The algorithm then sees a further sampled version of what the monitor observes. The patient data shown depicts the 'ground truth' MAP values coming directly from Pulse. In our simulation, the patient monitor samples data from Pulse every 30 seconds, and then reports it as observed data to the algorithm every minute. This shows that although a monitor can observe values at higher time resolutions, it may only report values to other systems at lower time resolutions so the algorithm's view of the patient may not necessarily be the same as the monitor's view of the simulated patients. The algorithm uses this information to decide when to send commands to pump to regulate infusion rates.

Though not noticeable in the figure, the pump has a delay of 15 seconds from when it gets command till when the new infusion rate starts.

Both the target MAP range and the period the CLA starts interacting are highlighted in all the plots to easily distinguish between the different parts of the scenario. From the results, we can also confirm that algorithm is indeed following the logic detailed in Figure 3.5 in line with the parameters in Table 3.2 by observing algorithm's response to the to changes in MAP data. Although we do not show explicitly which of the three states the algorithm is in, the framework can be extended to capture and present this information as well. Regarding this particular situation, it can be seen in Figure 3.6 that the Algorithm 1 using the clinician behavior (baseline) timing successively got the patient (Hassan) from a state of low blood pressure after a few minutes and then into the targeted range within less 30 minutes from the start of the infusion process and was also able to maintain his MAP in that range for remaining part of the simulation.

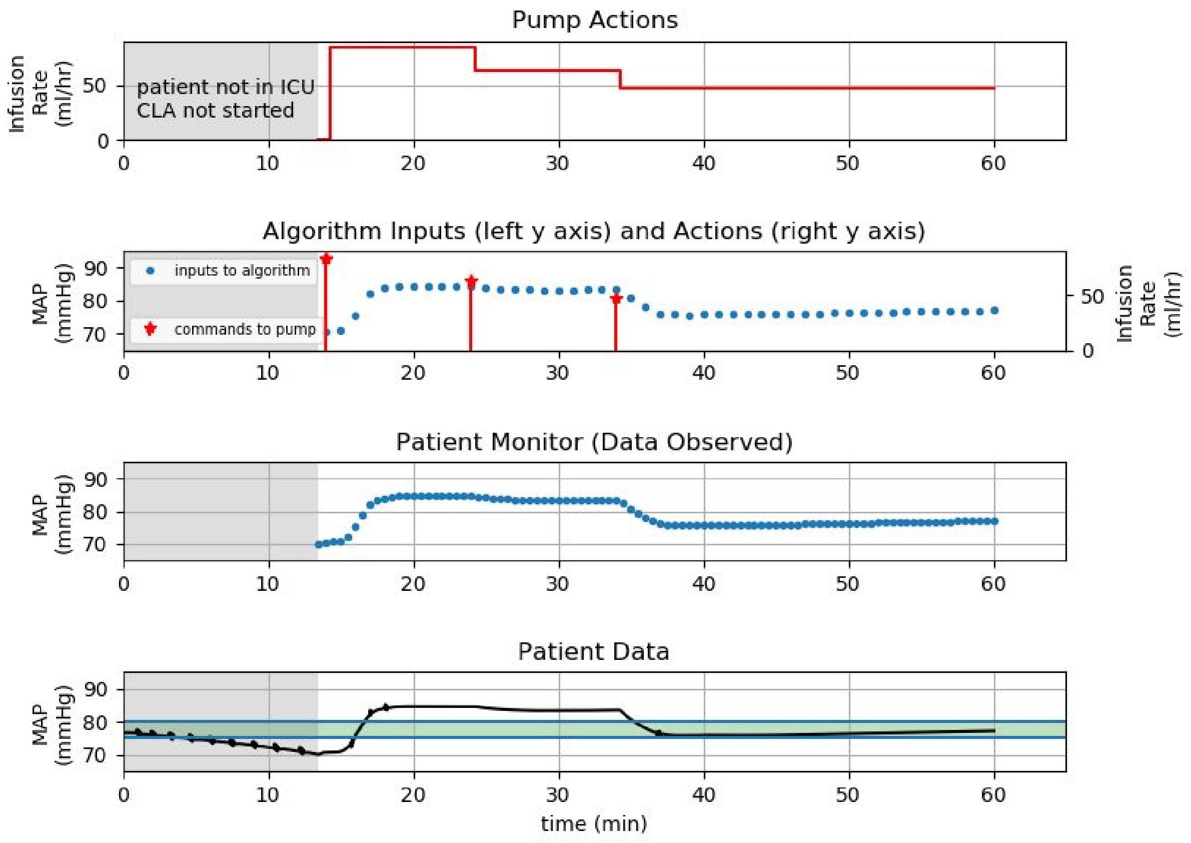


Figure 3.6. *Result of running CLA with algorithm 1 parameters from Table 3.1 with a single virtual patient.*

3.4.3.2 Single Patient Multiple CLA Designs

As mentioned earlier, one of the key advantages of this simulation framework is ability to explore the impact of designs along two dimensions: the impact of different design choices on a given patient, or the impact of patient variability on a given design. Both dimensions can be explored simultaneously to understand how different designs perform across the same patient population. Here, we look at the impact of different design choices on a single patient to get an initial sense of the kind of results we can expect from a design space exploration.

Figure 3.7 shows the results of of running the two different algorithms, and varying the timing behavior in each algorithm, with a given patient (Hassan).

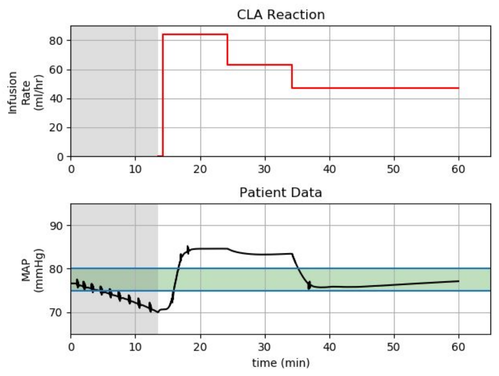
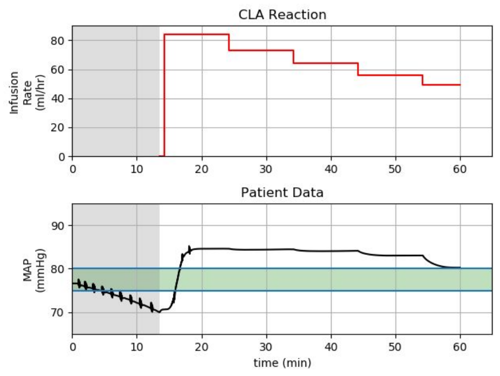
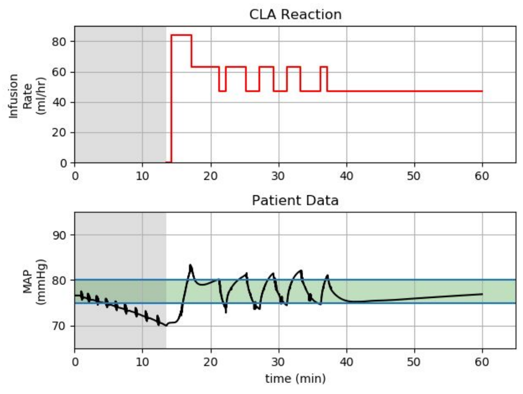
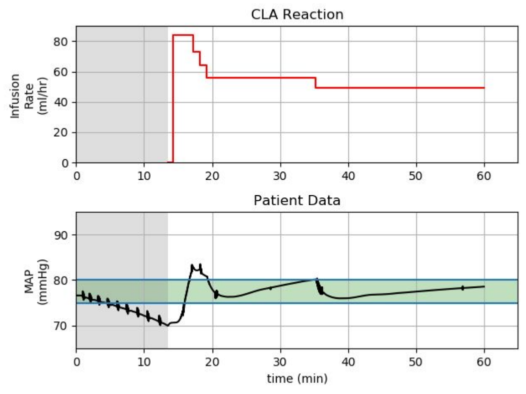
Timing Behavior	**Algorithm 1 Outcome**	**Algorithm 2 Outcome**
Clinician (Baseline)		
Computer System		

Figure 3.7. *Results of varying CLA behavior for a given patient (Hassan).*

The results show the advantage of simulation of over testing in biological systems (humans or animals). For each simulation, the CLA version starts with the exact same patient state. In addition, the only parameters that change are how often the CLA makes decisions on adjusting infusions based on the MAP and how much it reduces or increases the norepinephrine infusion rate. All other parameters remain the same. We see four different patient physiology responses for the four different behaviors.

Based on the results, it seems like for this simulation scenario, when adjusting infusions less often (clinician/baseline behavior), more aggressive rate changes (Algorithm 1) yield better results. However, when adjusting infusions more often (computer system behavior), less aggressive rate changes (Algorithm 2) yield better results. Although overall adjusting rates more often keeps the patient within the range more, the more aggressive rate changes cause the MAP to oscillate between the target boundaries whereas the less aggressive rate changes provide a smoother MAP response.

For each case, we can dig deeper like in Figure 3.7 to see how the specific behavior of the various entities may be contributing to the outcomes we observe.

3.4.3.2 Full Design Space Exploration: Multiple Patients Multiple CLA Designs

We can compare the four different CLA designs across the patient population as shown in Figure 3.8. Here we are only looking at the patient physiology outcome. However, for each patient, or each algorithm behavior we can examine the other outputs of the simulation to understand how the physiology outcome arises. It seems like all the CLA designs in this scenario have trouble keeping one patient in the desired range. Algorithm 2 with the computer system timing seems to do the best job of keeping the patients within the target range without too many oscillations in the MAP.

For larger populations, we could use these visualizations to get a general sense of how the CLA does, or we could define metrics using techniques in (Asare dissertation) to define the performance of the CLA and compute these metrics to compare different designs. This ability to examine the performance of different CLA design choices across a patient population is the kind of capability we hope to provide to designers. The results from the software-only simulation part of the framework allow for rapid exploration of various designs in the early development stages, especially when no prototype of the system exists. The results here can inform the design of a prototype. In the later stages of the development when a prototype exists, however, it would be more desirable to be able to test the actual prototype with virtual patients. This capability is described in the next chapter.

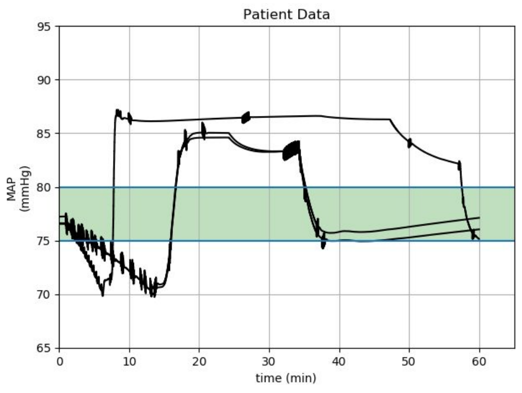
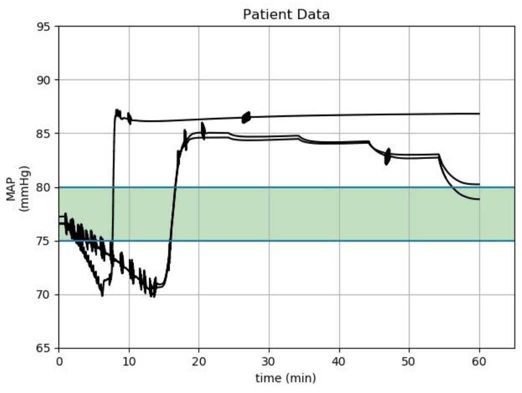
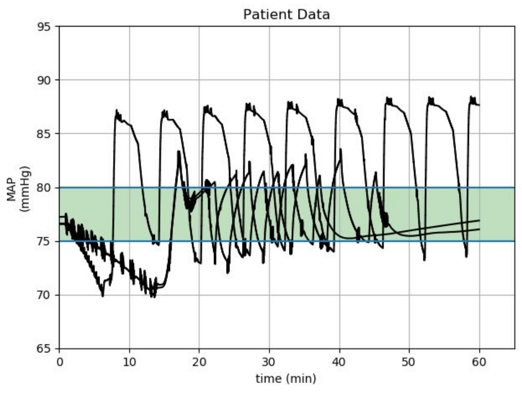
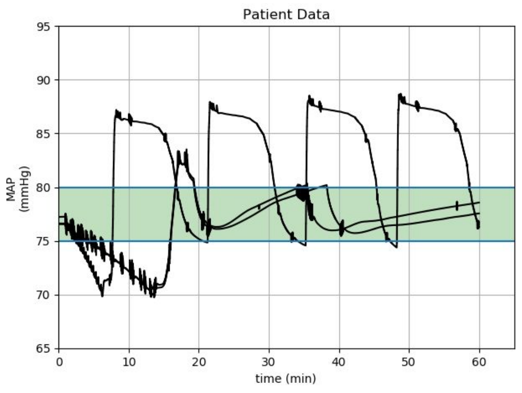
Timing Behavior	**Algorithm 1 Outcome**	**Algorithm 2 Outcome**
Clinician (Baseline)		
Computer System		

Figure 3.8. *Results of varying CLA behavior for a patient population (Hassan, Gus, and Diana). Gus and Hassan behave similarly. The algorithms have trouble with is Diana.*

Chapter 4
System-in-the-Loop Simulation

4.1 Motivation

As mentioned at the end of the previous chapter, once a prototype of the system exists, it is more desirable to test the actual prototype with virtual patients. This provides the repeatability and controllability advantages of simulation, while increasing the realism of the environment in which the design is tested. This chapter describes the design and implementation of the part of our framework that allows for such real-time testing with real CLA prototypes.

4.2 Architecture Overview and Design

The conceptual architecture for this part of the framework is very similar to the software only part of the framework as shown in Figure 4.1. The main difference is that we now have to develop physical versions of the lines that connect all the pieces. In particular, on the device-patient interaction side, we have to design and develop physical systems for converting the information from Pulse into physical signals, that mimic signals from an actual human, that the patient monitor can capture and interpret, as well as for providing the right vascular resistance to the pump and converting measured fluid flow out of the pump into fluid flow into the virtual patient model. On the devices-algorithm side, we either have to leverage an existing CLA that can connect to existing medical devices to receive patient data from the monitor and control infusions on the pump, or develop our own CLA for testing. Fortunately, we have our own research prototype medical application platform on which CLA behavior can be developed to leverage. In addition to connecting all the various pieces, we also have to ensure that information flows in and out of the virtual patient to guarantee real-time behavior of the overall system.

Similar to the case of the software-only simulation, what is most useful is the data that comes out of the simulation: not only the patient physiology data, but data about the behaviors of all the parts of the CLA. This means we must able to instrument the CLA system and capture the data flowing between the devices and the algorithm as well as the algorithms internal state and decision-making.

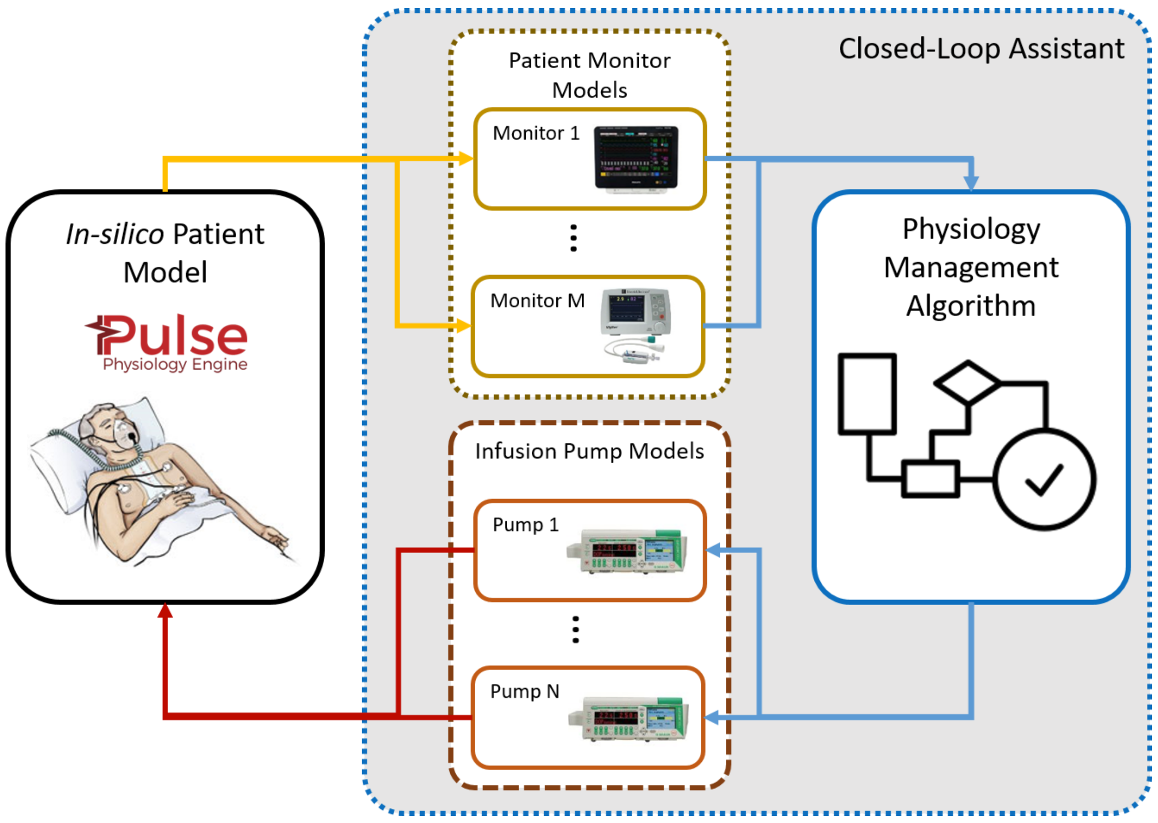


Figure 4.1. *Conceptual architecture of system-in-the-loop simulation framework for CLA-patient interactions*

4.3 Implementation

4.3.1 openmedap Platform

The research prototype medical application platform we used to implement the CLA for this work is openmedap [35]. It provides an 'abstraction layer' of the inner workings of the CLA system in order to enable the developers focus on building applications without the details of the devices being a major bottleneck. In addition, drivers for hardware devices such as the monitor and pump are provided to facilitate interaction and software development process.

4.3.1.1 Platform Design

Figure 4.2 shows the openmedap system architecture. From a connectivity or dataflow perspective, there are the medical devices that provide variables or abilities to act (e.g. infusing fluids) that the algorithm (or application) wants to make use of. The device drivers abstract away the specific devices and provide these variables and abilities to act to applications through the device manager. The device drivers and manager work together to ensure that all pieces act safely when connectivity fails.

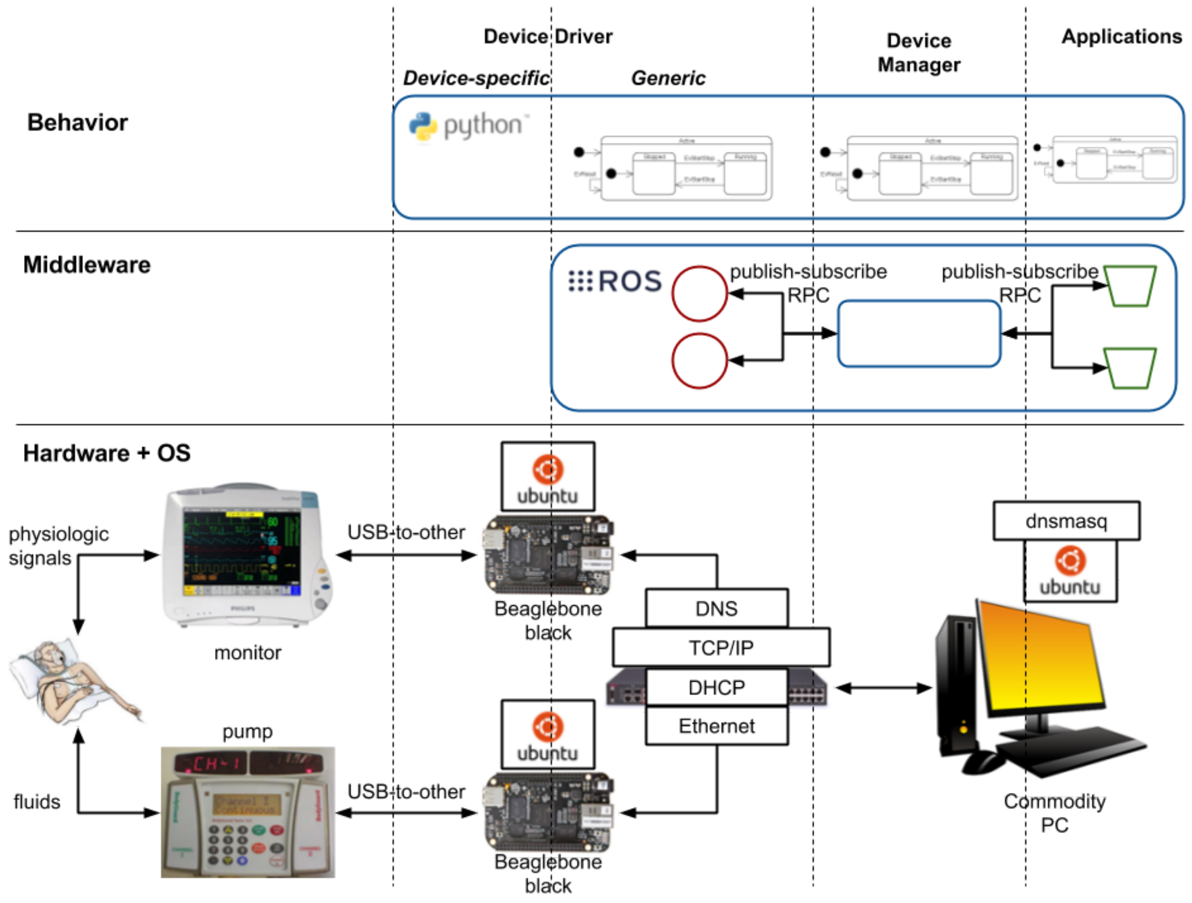


Figure 4.2. *openmedap architecture.*

Hardware

From the abstraction layer perspective, at the lowest level is the hardware, which consists of the devices (in this case a monitor and pump), a host computer, one network switch, and 2 single-board computers (SBCs) that serve as the devices drivers (one per device). In this case, the SBCs are the BlackBone Black (BBB) [36]. Each of the BBB is connected to a medical device and they host drivers of the attached devices to enable communication with the rest of the system and vice versa.

Operating System and Network Services

Above the hardware level is the operating system level that provides services that higher layers can access. All the hardware in the system run on The Ubuntu [37] operating system. The current version of openmedap runs on Ubuntu 16.04.6 LTS [38], which is a long term support version and will have maintenance updates until 2022.

Ubuntu was chosen as the operating system because of the software available to control more easily the operating system services that the platform needs. The main services are the domain name system (DNS), the dynamic host connection protocol (DHCP), and transmission control protocol/internet protocol (TCP/IP) deal with network connectivity and communication and allow us to connect the host computer to the BBB in ways the make the other layer so the platform run well.

The host computer (which also acts as a router) manages all the data communication across the network. By using the dnsmasq tool the host PC is able to act as both DNS and DHCP server. In other words, it can automatically assign, reclaim and manage the IP addresses centrally in a pool. The dnsmasq [39] tool is a lightweight infrastructure for small networks used for providing both Domain Name Server (DNS) and Dynamic Host Configuration Protocol (DHCP) server capabilities. Because computers in a network do not communicate in a human-like manner rather they use IP addresses that reference to a certain device in a network. The DNS server is used to automatically convert IP address like '192.168.xxx.xxx' to their respective domain name such as 'example.com' and vice versa. This enables the computer to be identified by a 'user-friendly' name over the network.

The DHCP server automatically assigns and distributes IP address to computers in a defined network range. It also packages other network parameters such as the subnet mask and the default gateway to each device in a network. Without the DHCP server, each computer added or removed from the network would require their IP addresses to be manually configured and reclaimed respectively. The DHCP server helps to solve this problem by performing dynamic assignment of IP addresses and returning IP addresses that are no longer in use to the DHCP pool which manages them centrally for reallocation to other DHCP-enabled clients. Since IP addresses are assigned dynamically, the same computer may not get the same IP address each time it joins. This is why the DNS is important, since it keeps track of which computer name (which doesn't change) is assigned to which IP address. That way we can keep referring to the same computer by name even if its IP address changes each time it joins the network.

Middleware: Robot Operating System (ROS)

Above the operating system level is the middleware which handle messaging processes within the application and provides simpler mechanism for communication between the application, devices drivers, and the device manager than the lower-level operating system mechanisms. The current version of openmedap uses the robot operating system (ROS) [40] as its middleware because of the flexibility it provides as the system evolves over time. ROS is used to handles connections and communication between the different software pieces (nodes) that make up the various endpoints of our system. These endpoints include the device drivers, the device manager, and the applications.

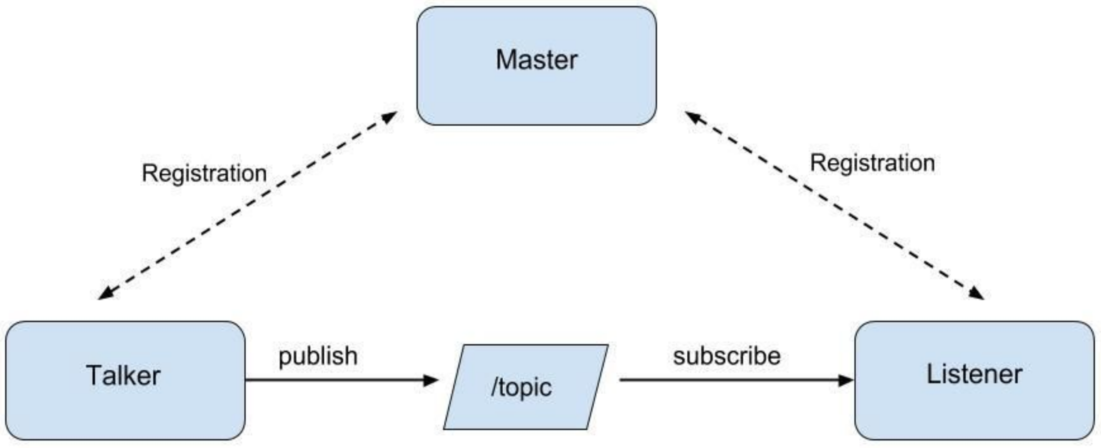


Figure 4.3. *Overview of ROS interactions showing two nodes named "Talker" and "Listener". Master is the part of ROS the manages all the interactions between nodes.*

ROS is made up of various packages, libraries and tools that help it to perform its functionalities. Among these tools are communication processes referred to as nodes. ROS encourages modularity by ensuring that application is separated into components where each performs a specific task. These components are referred to as nodes. In the current setup, the endpoints (pump, monitor and the computer hosting algorithm) are the nodes.

In order for these nodes to communicate, messages are published (e.g. by Talker node) through a particular channel (or topic). The message are received by all the nodes (e.g. Listener node) subscribing to that topic. The setup is shown in figure 4.3. The peer-to peer model used by ROS to enable nodes to either transmit or receive data known as the publish/model.

Before establishing a peer to peer communication, the nodes need to register with the 'Master'. The Master is mainly concerned with registration of names and any other information pertaining to the nodes on the network. This is useful because it ensures that all the nodes can easily locate each other, otherwise, it would be impossible for them to communicate. It is usually invoked by issuing 'roscore' command in the terminal.

Furthermore, ROS supports inter-process messaging between components that are distinct from each other and are operating in different programming languages that are different from each other. This feature greatly simplifies communication between components in our system which are built in different languages (i.e., python and C++ components).

4.3.1.2 Example Applications

Openmedap comes with example applications that demonstrate its capabilities for connecting to medical devices as well as for development software that makes use of these medical devices. There are three current example applications: an application that displays the heart rate and oxygen saturation from a Philips Intellivue patient monitor (patient monitor demo), one that starts, adjusts, and displays infusions from a BodyGuard 121 Twins infusion pump developed by CME America (pump demo), and an application that reacts to data coming from the Philips monitor by adjusting the infusion rate on the CME pump (closed-loop demo).

Patient Monitor Demo

This demo application demonstrates the communication management capabilities of openmedap and also its ability to send messages from a device to an application for display. The application connects to a receives information from a device driver for the Philips Intellivue patient monitor (in this case an MP50).

On startup, the application checks to see if the device is available for connection. If it is not, the application indicates this and does not allow the user to start the monitoring. Once the device is detected, the connection is indicated as green and the user can now start the monitoring. At any time during the monitoring, if the device is disconnected, the user is alerted. Once the user acknowledges receipt of the alert, the application goes back to the start screen and indicates in red or yellow that the driver or device connection is lost respectively.

Pump Demo

The Pump Demo application provides an interface for users interact with a pump (in this case the CME America BodyGuard 121 Twins pump). Like the Patient Monitor Demo, this application also displays the device and driver connection status on startup, and monitors this connection while the pump is running in order to alert the user if the connection is lost while the pump is running. The application lets the user initializes pump with specific setting parameters (i.e. bag volume and infusion rate) provided the pump is connected. While the pump is running, it displays the current rate and volume left to be infused. The user can also adjust the infusion rate while the pump is running. They can also pause, resume, or completely stop the infusion.

Closed-Loop Demo

The Closed-Loop Demo application demonstrates automated monitoring and infusion. The application monitors the heart rate and oxygen saturation and adjusts the pump infusion rate based on the heart rate value. Like the other two applications, it also continually monitors the connection to the two devices it works with and will alert the user and stop operation if a connection is lost. On startup, if both devices and drivers are connected, the user can start the application. Both the monitor values and the current state of the pump are displayed in the application while in action so the user can see what data the algorithm that adjusts infusions is acting on and what actions it takes.

4.3.2 Time synchronization

The CLA being a distributed system, it demonstrates the critical importance of clock synchronization. Typically, each device on the network runs off its own internal clock. This can be a major constraint for applications that depend on the clock accuracy and synchronicity in such systems (for instance the logging service on which the system may relies to record events in the order in which they occurred). To overcome this limitation the Network Time Protocol (NTP) was utilized.

NTP is a standard computer protocol used throughout the internet as a means of achieving and maintaining synchronization of clock times among communicating computers and other network infrastructure in a network [41]. When computer devices communicate in a network, the timestamps of the data packets, time difference, and the latency duration are encoded on the message and when the receiver gets the message, it decodes and reads the information. NTP is an application layer protocol and in its implementation, NTP comprises the following phases, network time servers (NTS), NTP stratum, precision oscillators, and the client software.

A network time server is a device (which in our case is the host computer) that receives precise time from an external hardware clock and provides the accurate timing reference to the network. This accurate time is maintained internally and passed to the network time clients upon request. The network time server then provides a time reference to all devices in a network. NTP stratum is a series of hierarchical protocols. Each stratum is synchronized to the level above in the hierarchy. At the top level, there is a stratum 1 server which provides an accurate time reference to an external hardware clock. The precision oscillators are used to provide time reference backup, examples include rubidium and an oven-controlled crystal oscillator.

In practice, the NTP works in a request and response format where the NTP clients (beaglebones connected to the CLA components in our case) initiate a request to the time servers then it synchronizes it's time to match the time on the server clock. The client is also able to calculate the link delay (latency) and its local offset (The absolute difference on the time value of the two clocks). It includes a pair of peer/poll processes for each reference clock or remote server used as a synchronization source. Packets are exchanged between the client and server using the on-wire protocol. The poll process sends NTP packets at intervals ranging from 8 s to 36 hr. The peer process receives NTP packets and performs the packet sanity tests and the results of various access control and security check.

Once synchronized, the client updates the clock about once every 10 minutes, usually requiring only a single message exchange. This transaction occurs via the User Datagram Protocol on port 123.

The architecture for our NTP implementation is shown in Figure 4.3.

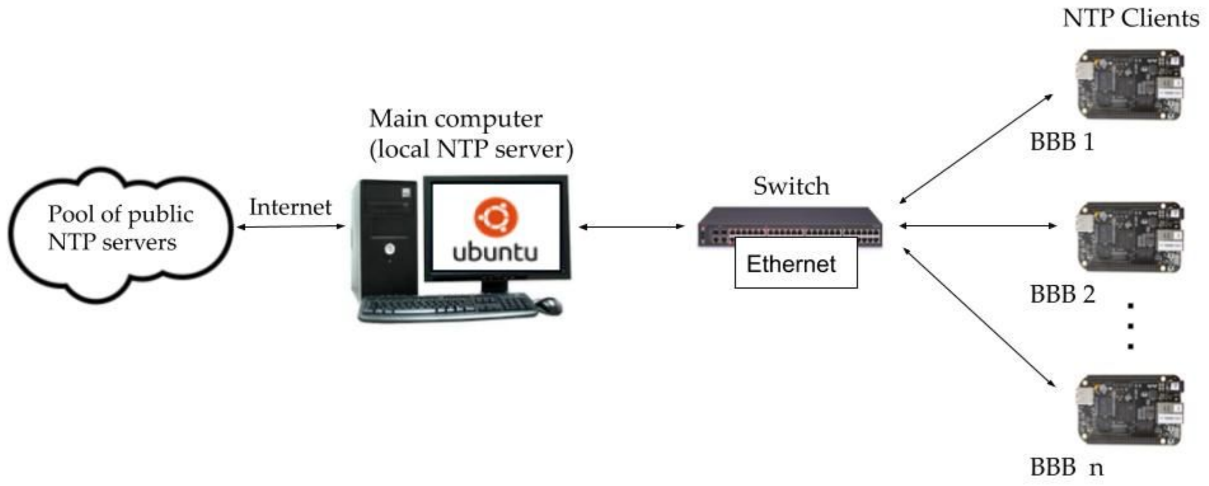
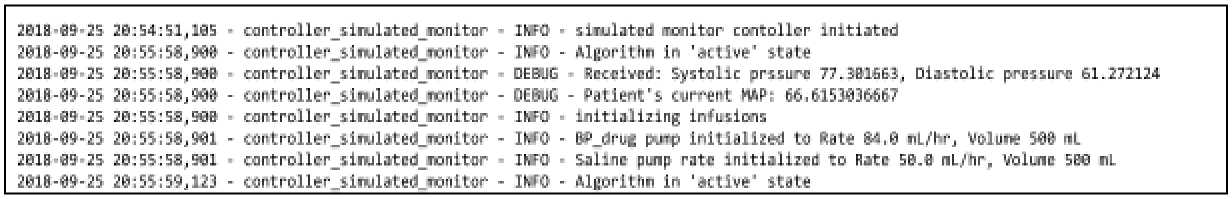


Figure 4.3. *NTP Architecture*

4.3.3 Instrumenting the CLA

To be able to gather and collect data generated during simulations, some form of instrumentation was required. We developed a logger using the python. The logger records its data in CSV files. The recorded data contains all the events as well as the time at which they occurred during the simulation run. For instance in case of the physiology algorithm, the logger was able to capture the invasive blood pressure (IBP) values received from monitor on each time step, instructions that were sent to the infusion pump and other useful things which the user can analyze at a later point. The information captured is formatted in such a way that it can easily understood by the user. Figure 4.4 is a snippet showing the kind of formatted messages captured by the logger. Each instance of an event has the date, time, module name, logging severity level and the message.



```
2018-09-25 20:54:51,105 - controller_simulated_monitor - INFO - simulated monitor contoller initiated
2018-09-25 20:55:58,900 - controller_simulated_monitor - INFO - Algorithm in 'active' state
2018-09-25 20:55:58,900 - controller_simulated_monitor - DEBUG - Received: Systolic prssure 77.301663, Diastolic pressure 61.272124
2018-09-25 20:55:58,900 - controller_simulated_monitor - DEBUG - Patient's current MAP: 66.6153036667
2018-09-25 20:55:58,900 - controller_simulated_monitor - INFO - initializing infusions
2018-09-25 20:55:58,901 - controller_simulated_monitor - INFO - BP_drug pump initialized to Rate 84.0 mL/hr, Volume 500 mL
2018-09-25 20:55:58,901 - controller_simulated_monitor - INFO - Saline pump rate initialized to Rate 50.0 mL/hr, Volume 500 mL
2018-09-25 20:55:59,123 - controller_simulated_monitor - INFO - Algorithm in 'active' state
```

Figure 4.4. *Snippet of logged messages*

4.3.4 Simulating Pulse Virtual Patients in Real-Time

Pulse drives the entire execution process and as stated earlier, it can execute and produce outputs relating to the patient's state at a rate of 50Hz by default. Since we are using actual medical devices in this case, it is desirable that we run simulations in real time in order to try to mimic what actually happens in a clinical setting in real time (specifically the ICU in our case). The synchronization process which involves sending the data from Pulse to monitor and initiating infusion is handled within the Prosim driver update method. Figure 4.5 shows how the synchronization between Pulse and the system clock time is achieved. N and t denote the total number of simulation steps and time between updates in the real world respectively (put simply,

N = TotalSimulationTime/t, where t = {1,2,3…}). The steps involved in this process are as follows.

(1) Pulse Advance time
(2) Send physiologic variables to monitor
(3) Initiate infusions
(4) Wait until next update

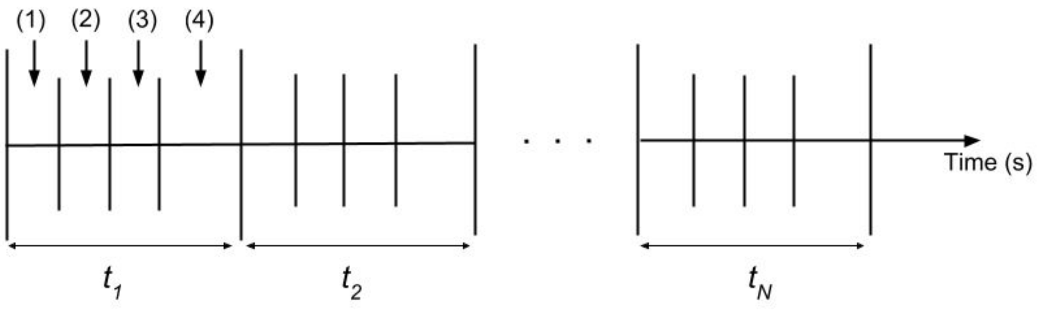
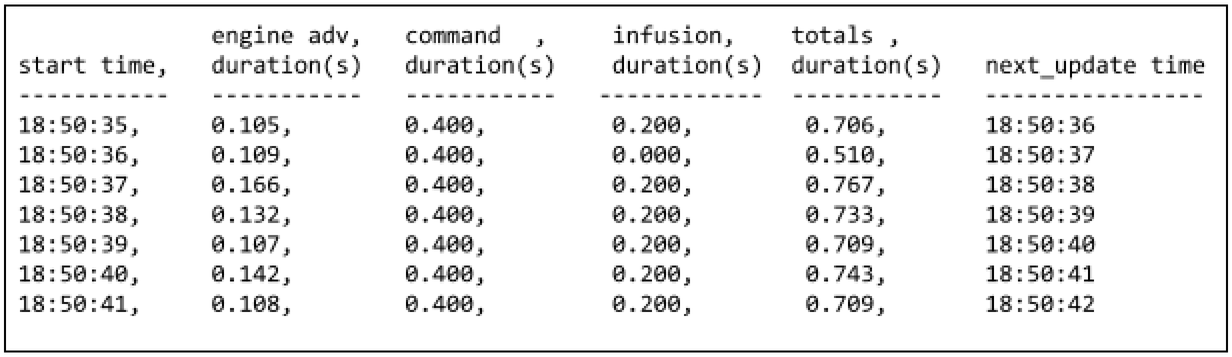


Figure 4.5. *Pulse to real-time synchronization*

The Pulse Advance time should be equal to the update period t in real world. The time information is set by the user in the configuration file and for the case study the update period was set to 1s. Figure 4.6 is an excerpt of the recorded data demonstrating a 1 second interval between each timestep in a real-time simulation. We analyze the real-time performance of the system in Section 4.4 where we present and discuss the results of evaluating this realtime framework based on the case study introduced in Section 3.4.



```
              engine adv,   command   ,   infusion,    totals ,
start time,   duration(s)   duration(s)   duration(s)  duration(s)    next_update time
-----------   -----------   -----------   ------------ -----------    ----------------
18:50:35,     0.105,        0.400,        0.200,        0.706,        18:50:36
18:50:36,     0.109,        0.400,        0.000,        0.510,        18:50:37
18:50:37,     0.166,        0.400,        0.200,        0.767,        18:50:38
18:50:38,     0.132,        0.400,        0.200,        0.733,        18:50:39
18:50:39,     0.107,        0.400,        0.200,        0.709,        18:50:40
18:50:40,     0.142,        0.400,        0.200,        0.743,        18:50:41
18:50:41,     0.108,        0.400,        0.200,        0.709,        18:50:42
```

Figure 4.6 *Snippet of recorded real-time durations*

4.3.4.1 Virtual Patients with Simulated Devices in Real-Time

Before committing to a setup with real medical devices, we propotyped a scenario where everything operated in real-time but with simulated devices. Where the virtual patient interacts with the openmedap platform and openmedap-based algorithm through a simulated pumps and monitors. This allows us to identify any issues in the real-time architecture while eliminate issues from the devices themselves as possibility. It also serves as a useful real-time testing framework for those who may not have access to the medical devices.

Pulse which implements the virtual patient needs a way to interact with the openmedap-based CLA. We rely on ROSBridge [42] to to achieve this functionality as shown in Figure 4.7. ROSBridge is a collection of different ROS packages that enable non-ROS applications (like our patient simulation in Pulse) to interact with ROS applications. This package allows non-ROS apps to create ROS channels, through which data can be published and received as JSON strings using ROSBridge API.

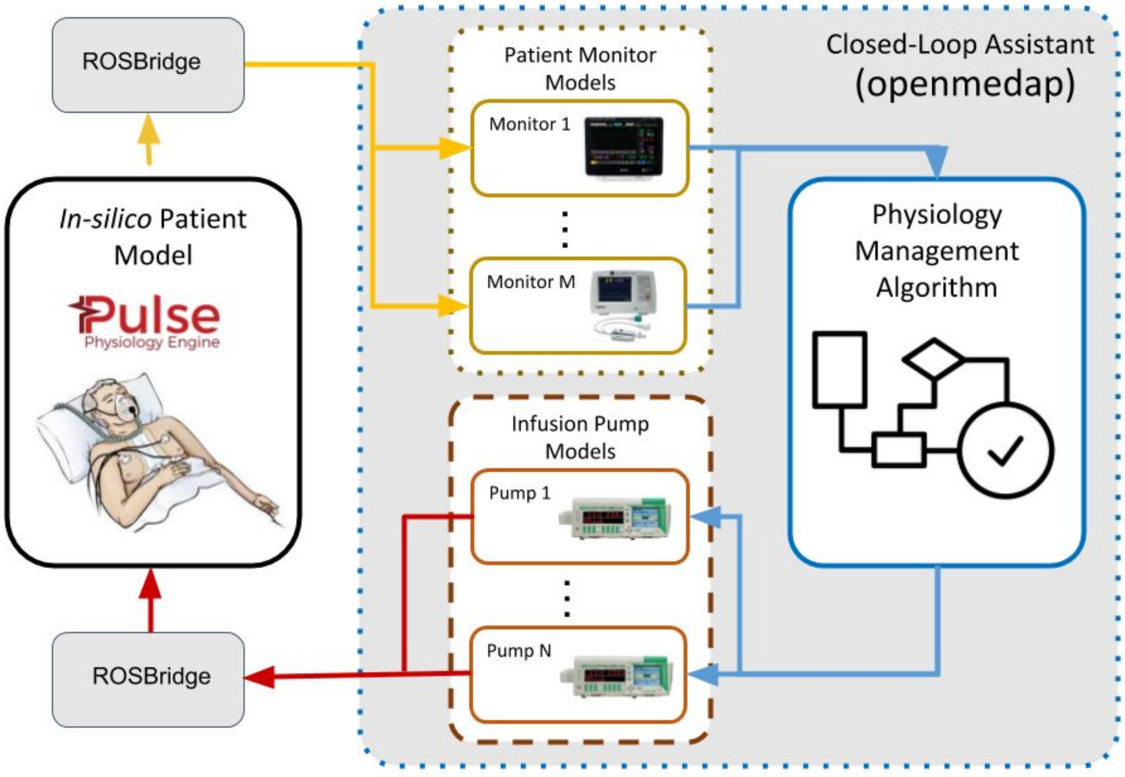


Figure 4.7. *Conceptual picture of Pulse interacting with openmedap-based CLA algorithm in real-time using the simulated devices.*

The simulated monitor publishes to the '/monitor' channel which the openmedap-based algorithm subscribes to in order receive patient related information. The data published on this channel includes both the systolic (ABP_{sys}) and the diastolic (ABP_{dias}) blood pressure values. The algorithm uses this data to compute the mean arterial blood pressure (MAP) (using the equation 4.1), on which it bases to make decisions (i.e., send commands to pump). The value obtained from the formula is only an estimate of the true MAP value, which is more accurately derived using the area under the curve of the arterial blood pressure waveform.

$$MAP = \frac{1}{3} \cdot ABP_{sys} + \frac{2}{3} \cdot ABP_{dias} \qquad (4.1)$$

The pump provides a '/pump' channel where it receives action commands from the algorithm. Pulse intercepts the messages from the algorithm to the pump and implements the action that the algorithm wants in the simulation.

4.3.4.2 Virtual Patient with Real Devices in Real-Time

In contrast to the simulated devices setup, here the virtual patient interacts with the openmedap platform via real devices. The architecture is shown in Figure 4.8

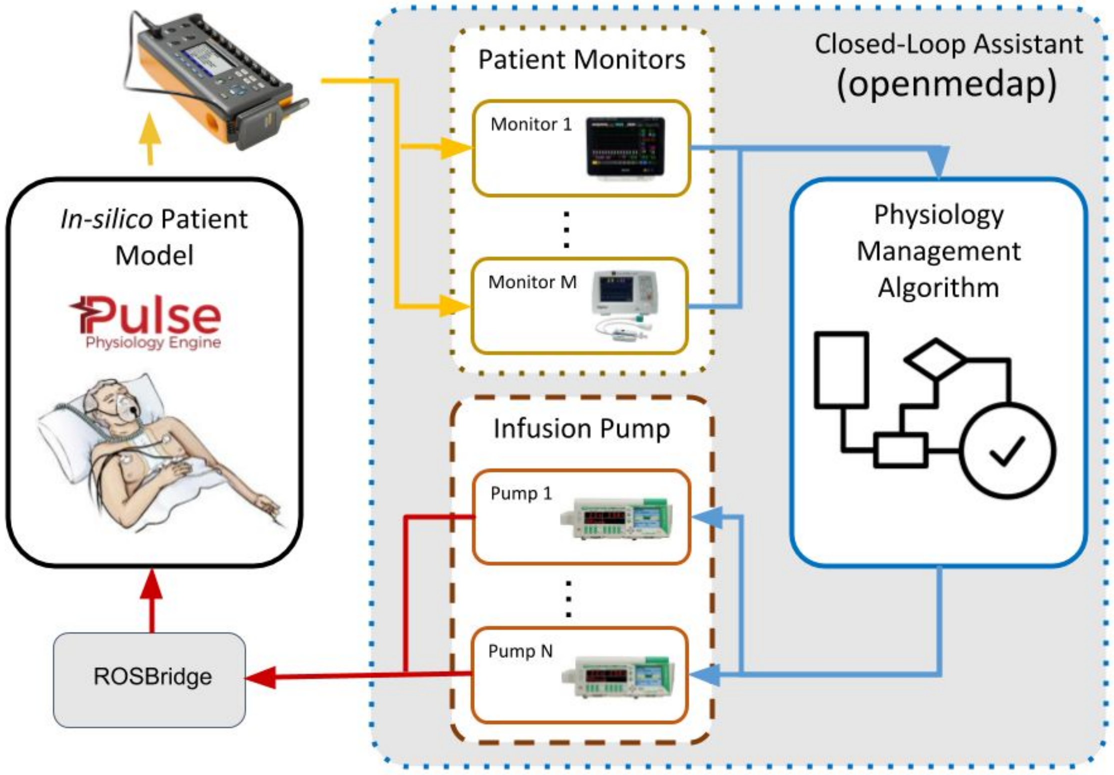


Figure 4.8. *Conceptual picture of Pulse interacting with openmedap-based CLA algorithm using real devices.*

In order for Pulse to interact with the monitors we use The Fluke Biomedical Prosim 8 patient vitals simulator hardware [43]. The Prosim 8 hardware is a device that converts the physiologic values from Pulse into physical signals that can be detected by standard patient monitors. Pulse communicates with the Prosim 8 hardware via a driver that we developed.

Currently, Pulse with the pump via ROSBridge. We are currently working on a solution that does not require intercepting the messages to adjust infusion rates to the pump.

This would make the interface between the virtual patients and the medical devices (patient monitor and pump) fully agnostic to the specific devices used.

4.4 Case Study and Results

We repeated the scenarios using the clinician (baseline) timing behavior of the algorithm that were run in the software-only framework in the real-time framework both for the case with simulated devices and for the case with real devices. These allowed us to compare how the results change as we increase the realism of the test environment from low level for realism in the software-only simulations to the higher level of realism in system-in-the-loop simulation with real devices. We worked with the clinician baseline because that is the scenario against which we can validate the data in the future once we have access to clinical data.

4.4.1 Real-Time Performance

In order for this part of the framework to be useful, it must be able to operate in real-time on reasonable computing resources. Earlier, in Section 4.3.4, we showed a snippet of the logged timing data to show that the system updates once every second as intended. For our case study, updates once every second was chosen because this is the minimum rate at which monitors and pumps are able to respond over their respective computer interfaces. It also represents the order of magnitude of time at which humans who interact with the system are able to reasonably respond.

The specification for the computer on which virtual patient was run for all real-time experiments is shown in Table 4.1.

Table 4.1. *Virtual patient computer specifications*

Parameter	Value
Brand	Lenovo ThinkCentre M83 Tiny
CPU	Intel® Core™ i5-4590T CPU @ 2.00GHz (4 cores)
RAM	15.6 GiB
Operating System	Ubuntu 16.04 LTS

We found in experimentation that capturing the infusion information from the pump works consistently when there is a 0.2s timeout on reading the information. In measurements, we found that this timeout dominates the infusion update. For updating the Prosim 8 hardware, we send four different commands and wait 0.1s between sending commands. In measurements, we found that this waiting time dominates the overall updates to the Prosim 8 and measured consistently that the Prosim 8 hardware took 0.4s to update. This means the remaining 0.4s in the update time is what Pulse has to work with. Figure 4.9 shows how long each 1s step of the simulation took to update in Pulse over the course of each of the 12 one-hour real-time simulations. Figures 4.10 and 4.11 shows the distribution of Pulse update time values for each simulation for the simulated devices and real devices cases respectively.

In all cases, none of the simulations exceed the 0.4s bound at any point during the simulation. Most of the simulations stay around the 0.2s mark or less.

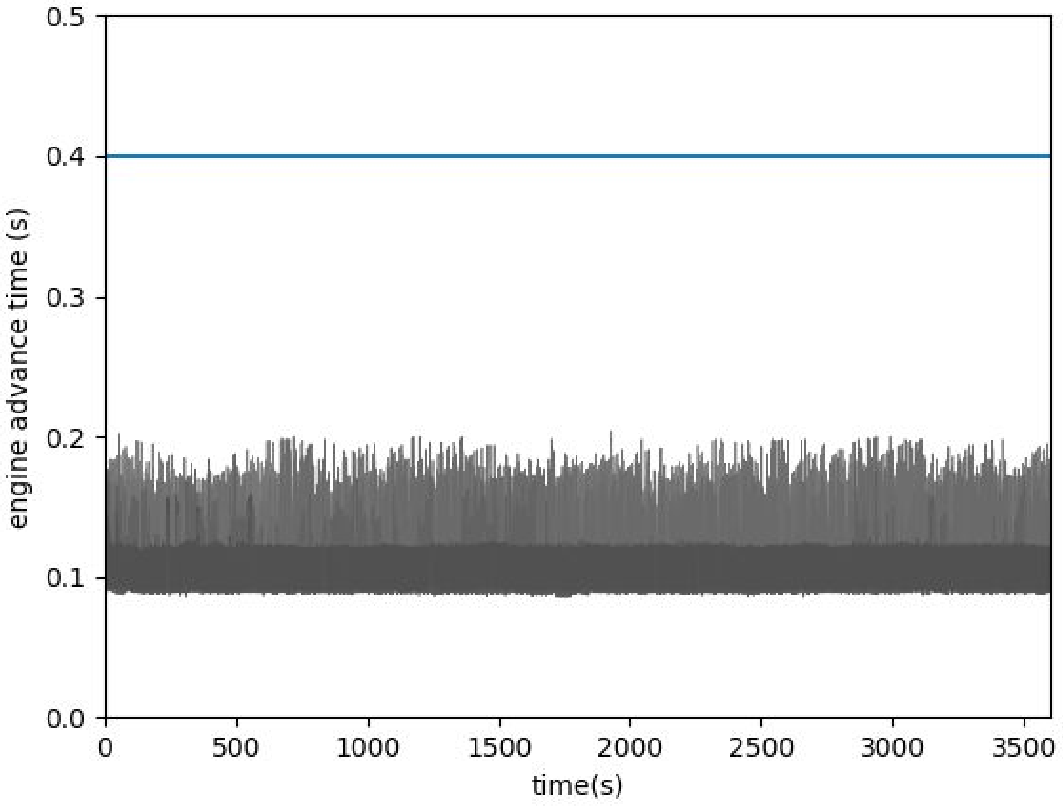


Figure 4.9. *Pulse time to simulate 1s of simulation time at each point in the real-time simulation for 12 different simulations. The blue line indicates the maximum allowed time.*

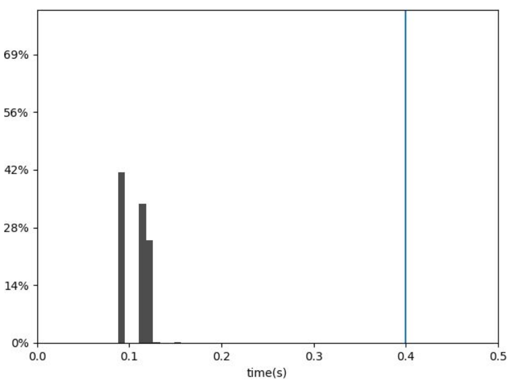
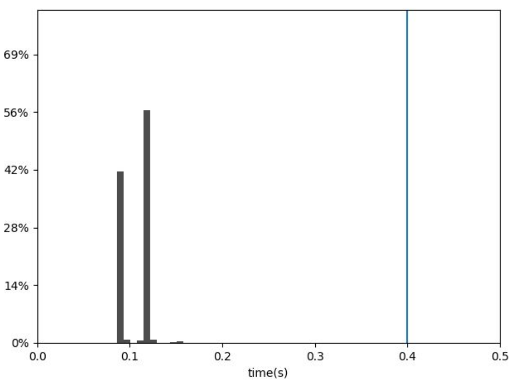
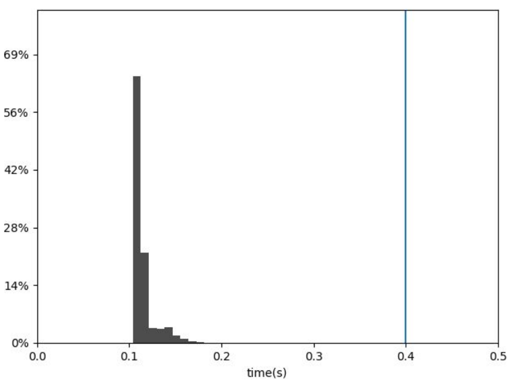
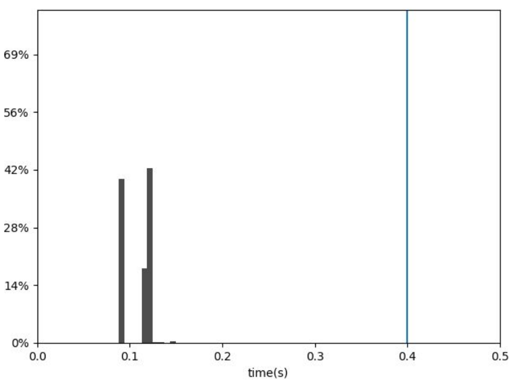
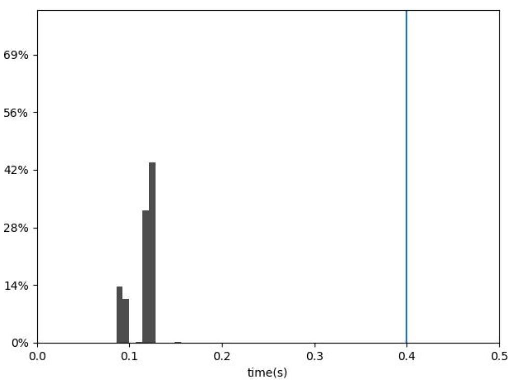
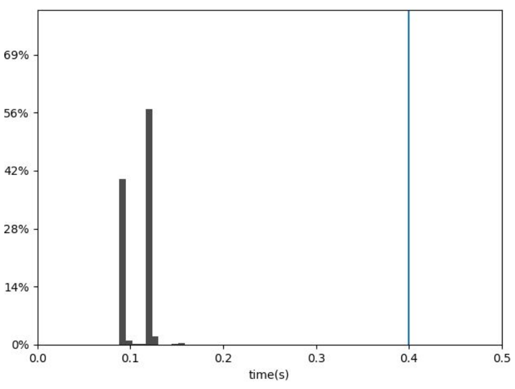
Patient	Algorithm 1	Algorithm 2
Diana		
Gus		
Hassan		

Figure 4.10. *Distribution of Pulse time to simulate 1s of simulation time for the simulated devices case. The blue line indicates the maximum allowed time.*

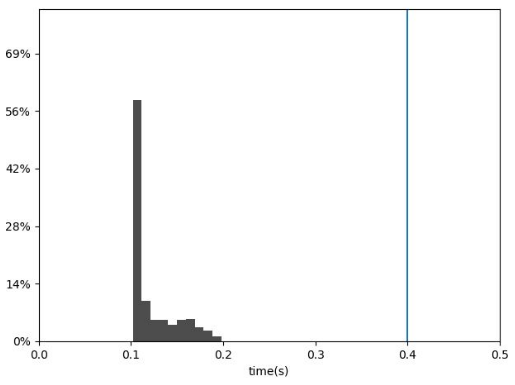
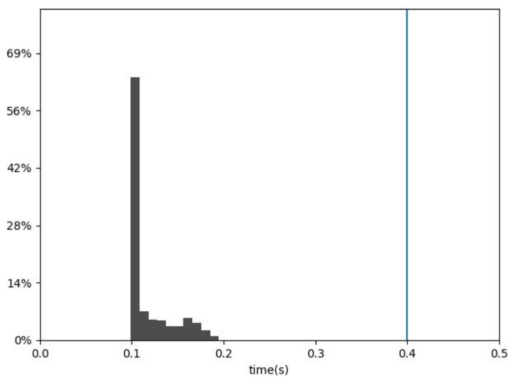
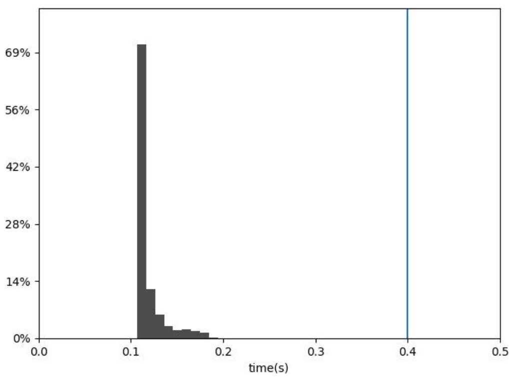
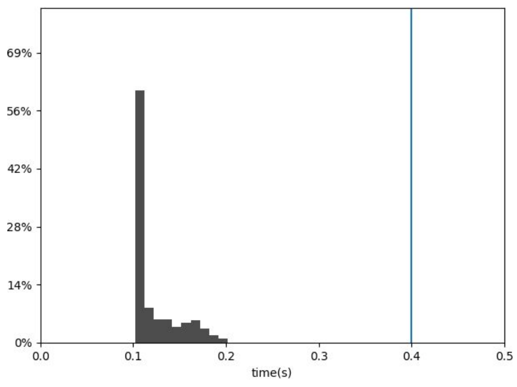
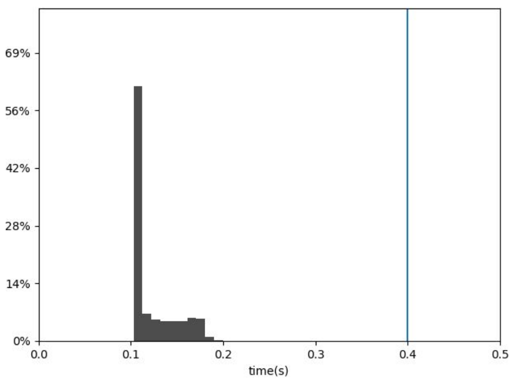
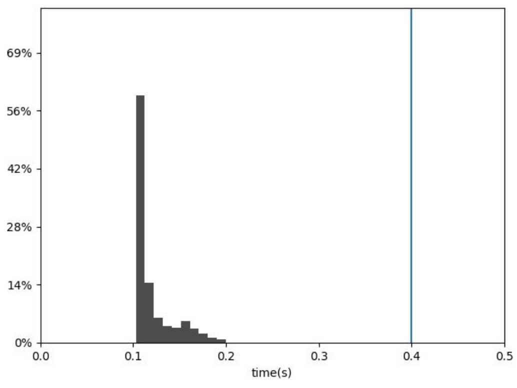
Patient	Algorithm 1	Algorithm 2
Diana		
Gus		
Hassan		

Figure 4.11. *Distribution of Pulse time to simulate 1s of simulation time for the real devices case. The blue line indicates the maximum allowed time.*

4.4.2 Comparison of Patient Outcomes Across Simulation Types

One simple comparison of the difference in behavior of the different simation types (software-only, real-time with simulated devices, real-time with real devices), is to examine the 'ground truth' MAP of the virtual patient for the same simulation scenario. If there is any significant disagreement between values across the simulations, we can use the other logged data to examine the behavior of other parts of the system using the data that is logged. Figure 4.12 shows the plot of the MAP for all three simulation types for each patient under the baseline timing behavior of the algorithm. For these plots the results mostly agree with the exception of the versions with simulated devices for Gus and Hassan. We can dig deeper into these specific simulations to understand why the differences arise.

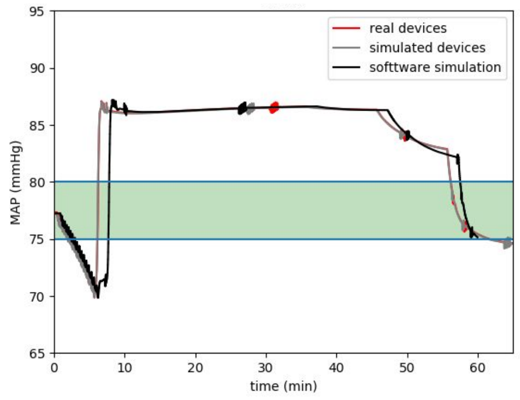
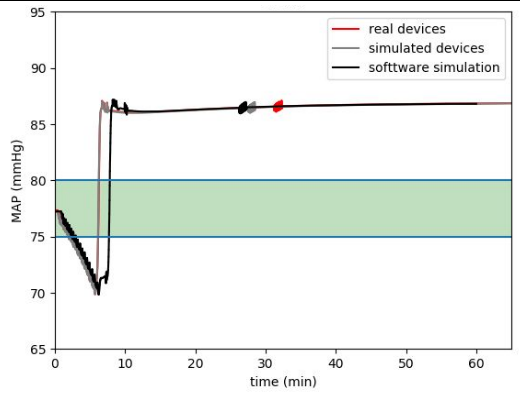
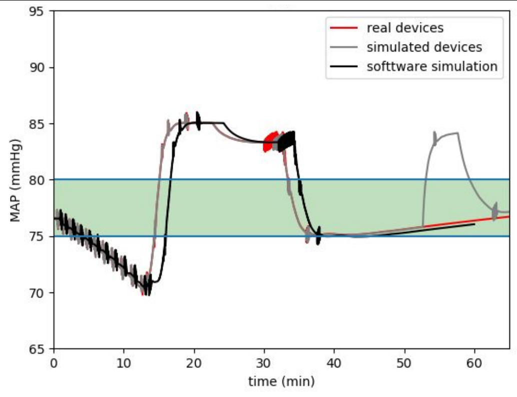
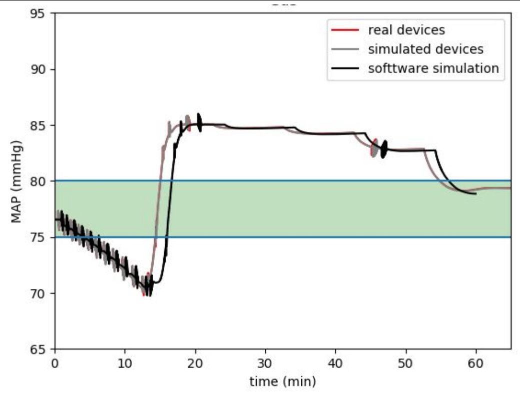
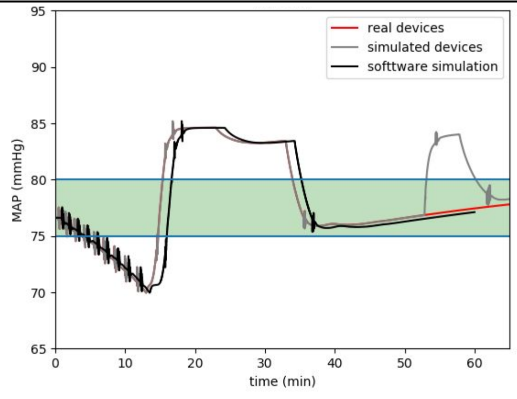
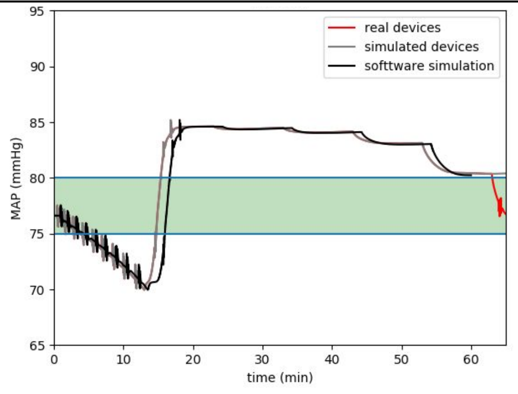
Patient	Algorithm 1 Outcome	Algorithm 2 Outcome
Diana		
Gus		
Hassan		

Figure 4.12. *Comparison of patient outcomes across simulation types (baseline timing behavior).*

Figure 4.13 shows the case of Gus with the simulated devices in real-time under algorithm 1 indicating the algorithm's view of the patient's MAP, as well as the 'ground truth' patient MAP. Remember from equation 4.1 in Section 4.3.4.1 that for the simulated devices case, the algorithm estimates the MAP from the systolic and diastolic arterial pressures reported by the simulated monitor, and that this equation is an approximation that does not account for the area under the curve of the arterial pressure waveform which is the more accurate MAP measure. You can see in this Figure that the algorithm consistently underestimates the MAP which causes it to increase the norepinephrine infusion although the actual MAP is within range at around the 53-minute mark.

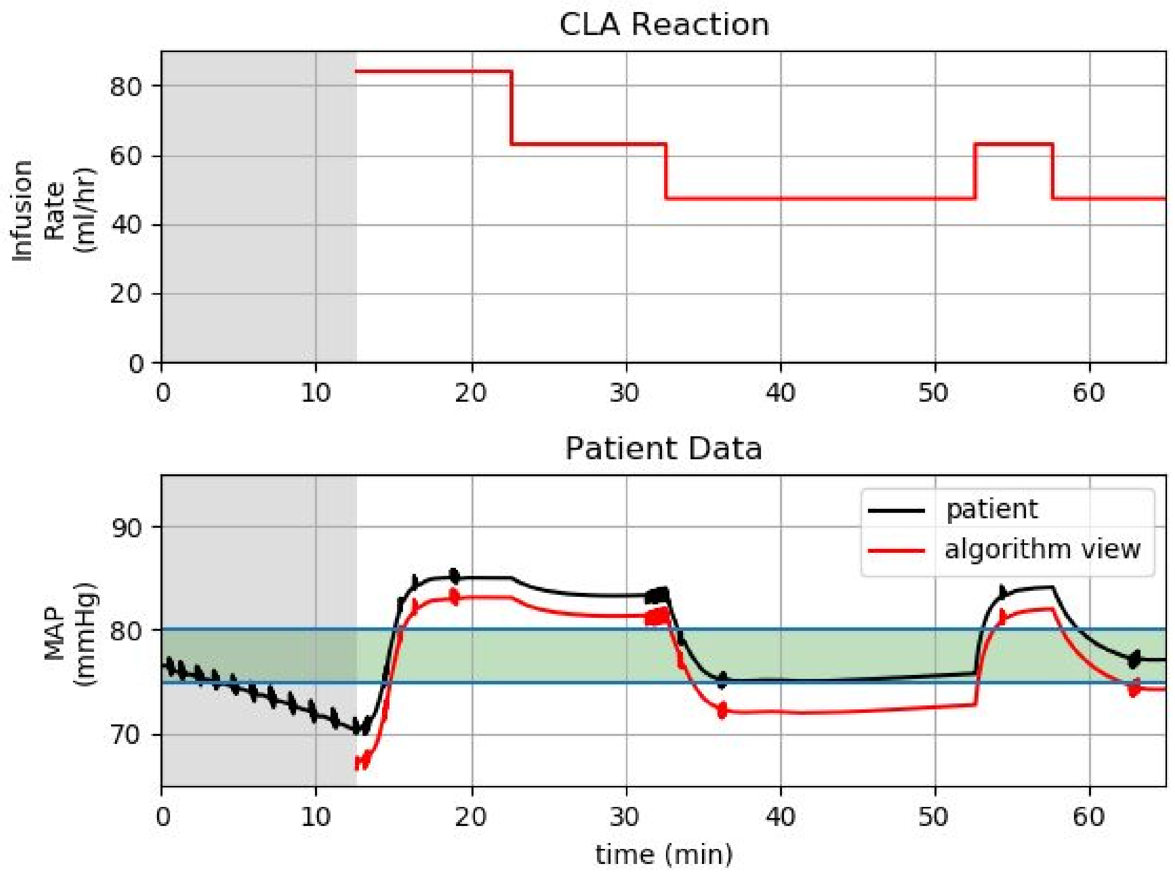


Figure 4.13. Result of simulating Gus in real-time under algorithm 1 with baseline timing behavior using the simulated devices.

The reason why the increase in infusion is not immediate is because around the 33-minute mark it changes the infusion to continue to ensure the decline of the MAP and waits 10 minutes to check the MAP again, as spelled out in the algorithm description in Section 3.4.1 and shown in Figure 3.5. At the 43-minute mark, the algorithm realizes that the MAP has reduced below the intended decline threshold and switches states to keeping the MAP within the target range. While switching states, the algorithm does not adjust the infusion. This is the specified this behavior in Section 3.4.1 and Figure 3.5. After the state switch, the algorithm waits another 10 minutes before checking the MAP to adjust infusions. At the 53-minute mark it estimates that the MAP is below the target range so it increases the infusion. It then waits 5 minutes as specified, and then adjusts the infusion again at the 58-minute mark.

The real devices simulation as shown in Figure 4.12 does not have this problem because it uses the MAP value from the patient monitor which computes the MAP from the arterial waveform.

The real devices simulation would sometimes encounter problems since it was operating on the full openmedap platform. Figure 4.14 shows one such case (Hassan under algorithm 1 with the baseline timing behavior) where the real-time simulation does not agree with the software-only simulation. Upon further examination in Figure 4.15, we can see that the algorithm stops receiving MAP values from the patient monitor around the 23-minute mark due to a lost connection to the monitor. Because of this, the algorithm is unable to react to the MAP values and change infusions, and the pump continues to infuse and the most recent rate that it was programmed to use by the algorithm, which results in the patient MAP behavior seen.

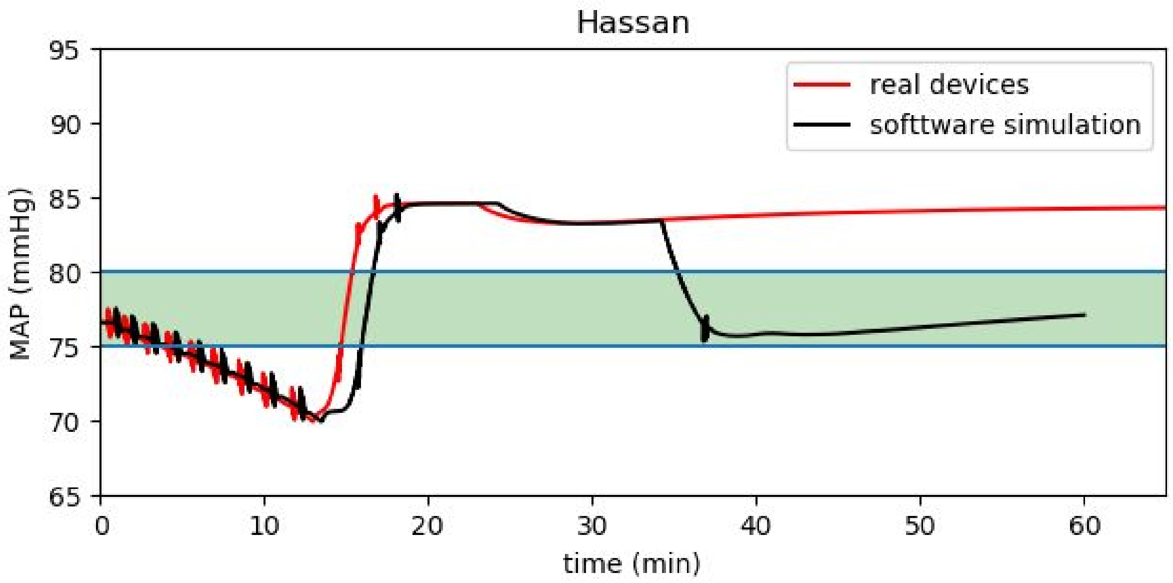


Figure 4.14. *Example of when real-time with real devices behavior deviates from software-only-simulation behavior for Hassan under algorithm 1 using baseline timing.*

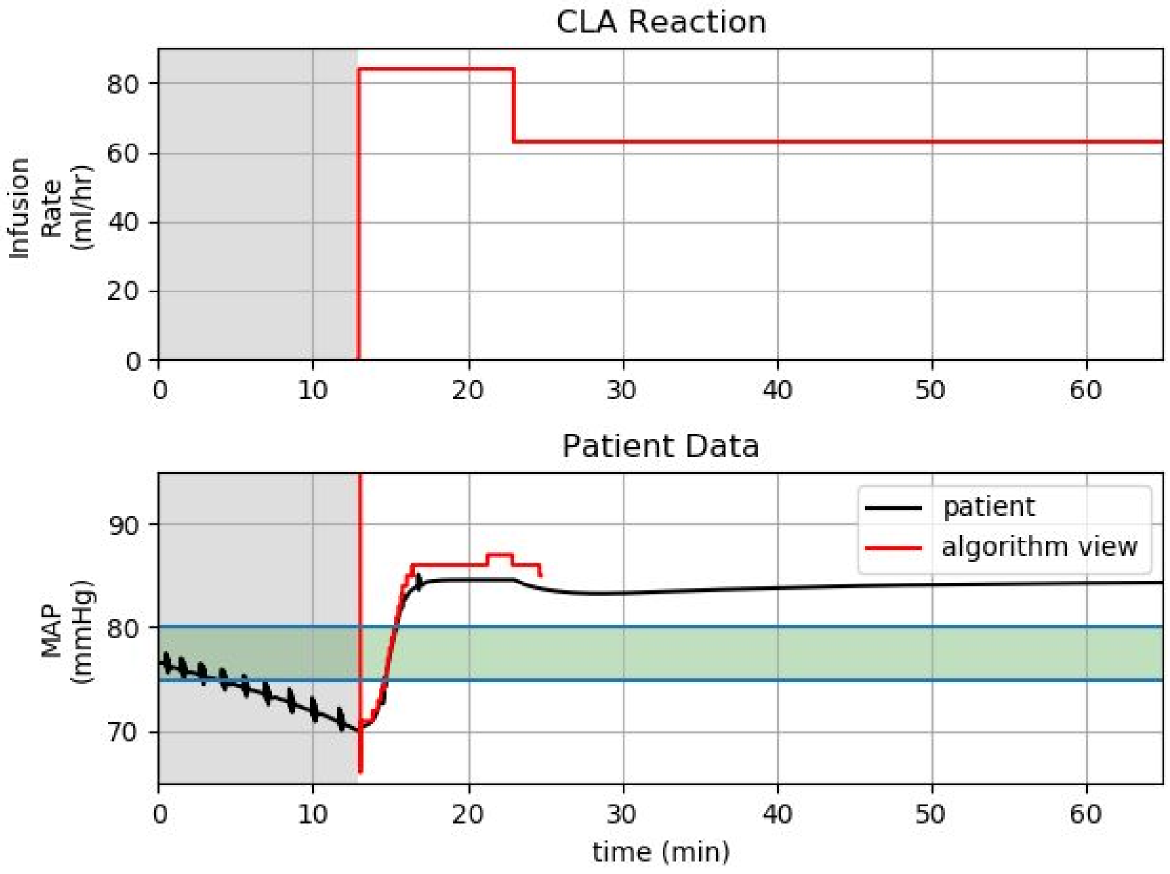


Figure 4.15. *More in-depth view of algorithm behavior of real-time simulation shown in Figure 4.14.*

Chapter 5
Discussion and Conclusion

In this thesis we presented the motivation for, design, and development of a framework to examine the efficacy of a physiological management strategies or systems in improving patient outcomes. To demonstrate the utility of the framework, we simulated and compared different algorithm strategies across a group of patients through both software-only only simulation and system-in-the-loop simulation. The software-only framework allowed us to quickly gain insights into which physiology management strategies or systems had the potential to result into better outcomes. The system-in-the-loop framework allowed us to understand better the behavior of different systems or strategies in a more realistic setting. In Chapter 4, we showed how this framework and its logging mechanisms can be used to understanding deviations of real-time behavior from the software-simulation-only (more ideal) behavior.

Although the framework is targeted at testing and validating technologies for managing physiology (CLAs), it actually serves as a good framework for also evaluating protocols that humans would apply manually. The case study in Chapter 3 illustrated this where we modeled the behavior of clinicians as a baseline. Since the real-time virtual patient system actually only interfaces directly with the medical devices, it cannot tell whether what is using the data from the monitor and adjusting the pump is a human or a computer algorithm. This means we can also use that part of the system-in-the-loop framework to evaluate clinical protocols applied by humans. More importantly, since the CLA is an assistive tool and would be supervised by a clinician who can take over treatment by working directly with the devices, the approach to the real-time simulation actually allows for evaluating the CLAs that are working with clinicians. This can prove to be a useful tool for learning about how clinicians might interact with CLAs and also for training clinicians to work with CLAs.

The biggest limitation of the work is in the patient models and patient population. Although we were able to show proof-of-concept of the kinds of exploration a designer would be able to undertake, because of the limited set of patients in Pulse and also the fact that the patients are not currently specifically designed to properly model patients undergoing surgery or in critical care in the ICU, were a not able to make any meaningful claims about the clinical relevance of our the results obtained from simulation. One important direction for future work is to improve the patient models to they can represent such physiologies and also to expand the number of different patients so that more inter-patient variability can be explored, which would in turn help to increase the robustness of systems that leverage the simulation capabilities we provide for testing and validation.

References

[1] B. P. Kovatchev *et al*, "In Silico Preclinical Trials: A Proof of Concept in Closed-Loop Control of Type 1 Diabetes," *J Diabetes Sci Technol,* vol. 3, *(1),* pp. 44-55, 2009. Available: https://doi.org/10.1177/193229680900300106.

[2] L. Magni *et al*, "Evaluating the Efficacy of Closed-Loop Glucose Regulation via Control-Variability Grid Analysis," *J Diabetes Sci Technol,* vol. 2, *(4),* pp. 630-635, 2008. Available: https://doi.org/10.1177/193229680800200414.

[3] C. D. Man, R. A. Rizza and C. Cobelli, "Meal Simulation Model of the Glucose-Insulin System," *IEEE Transactions on Biomedical Engineering,* vol. 54, *(10),* pp. 1740-1749, 2007. DOI: 10.1109/TBME.2007.893506.

[4] S. D. Patek *et al*, "In Silico Preclinical Trials: Methodology and Engineering Guide to Closed-Loop Control in Type 1 Diabetes Mellitus," *J Diabetes Sci Technol,* vol. 3, *(2),* pp. 269-282, 2009. Available: https://doi.org/10.1177/193229680900300207.

[5] M. Breton and B. Kovatchev, "Analysis, Modeling, and Simulation of the Accuracy of Continuous Glucose Sensors," *J Diabetes Sci Technol,* vol. 2, *(5),* pp. 853-862, 2008. Available: https://doi.org/10.1177/193229680800200517.

[6] Z. Jiang, A. Connolly and R. Mangharam, "Using the virtual heart model to validate the mode-switch pacemaker operation," *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, August 2010, . DOI: 10.1109/IEMBS.2010.5626262.

[7] Z. Jiang and R. Mangharam, "Modeling cardiac pacemaker malfunctions with the virtual heart model," *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, August 2011, . DOI: 10.1109/IEMBS.2011.6090051.

[8] Z. Jiang *et al*, "Real-time heart model for implantable cardiac device validation and verification," *22nd Euromicro Conference on Real-Time Systems,* July 2010, . DOI: 10.1109/ECRTS.2010.36.

[9] U.S. National Library of Medicine. “Heart pacemaker: MedlinePlus Medical Encyclopedia.” [Online]. Available: https://medlineplus.gov/ency/article/007369.htm. [Accessed Apr. 25, 2019]

[10] National Heart, Lung, and Blood Institute (NHLBI). *Pacemakers*. [Online]. Available: https://www.nhlbi.nih.gov/health-topics/pacemakers. [Accessed Apr. 25, 2019]

[11] U.S. Food and Drug Administration. “The Artificial Pancreas Device System”. [Online]. Available: https://www.fda.gov/medical-devices/consumer-products/artificial-pancreas-device-system [Accessed Apr. 25, 2019]

[12] U.S. Food and Drug Administration, "What is the pancreas? What is an artificial pancreas device system?". [Online]. Available: https://www.fda.gov/medical-devices/artificial-pancreas-device-system/what-pancreas-what-artificial-pancreas-device-system [Accessed Apr. 25, 2019]

[13] National Institute of Diabetes and Digestive and Kidney Diseases. “Type 1 Diabetes”. [Online]. Available: https://www.niddk.nih.gov/health-information/diabetes/overview/what-is-diabetes/type-1-diabetes. [Accessed Apr. 25, 2019]

[14] P. C. Davidson, R. D. Steed and B. W. Bode, "Glucommander: A computer-directed intravenous insulin system shown to be safe, simple, and effective in 120,618 h of operation," *Diabetes Care,* vol. 28, *(10),* pp. 2418-2423, 2005. Available: https://doi.org/10.2337/diacare.28.10.2418.

[15] Glytec. “eGlycemic Management System®”. [Online]. Available: https://www.glytecsystems.com/Solutions.html. [Accessed Apr. 25, 2019]

[16] A. Joosten *et al*, "Feasibility of Fully Automated Hypnosis, Analgesia, and Fluid Management Using 2 Independent Closed-Loop Systems During Major Vascular Surgery," *Anesthesia & Analgesia,* vol. Publish Ahead of Print, 2018. Available: https://doi.org/10.1213/ane.0000000000003433.

[17] A. Joosten *et al*, “Implementation of closed-loop-assisted intra-operative goal-directed fluid therapy during major abdominal surgery: a case-control study with propensity matching”. *European journal of anesthesiology*, vol. 35, (9), pp. 650–658, 2018. Available: https://doi.org/10.1097/EJA.0000000000000827

[18] A. Joosten *et al*, "Automated Titration of Vasopressor Infusion Using a Closed-loop Controller: In Vivo Feasibility Study Using a Swine Model," *Anesthesiology,* vol. 130, *(3),* pp. 394-403, 2019. Available: https://doi.org/10.1097/ALN.0000000000002581.

[19] M. Janda *et al*, "Clinical evaluation of a simultaneous closed-loop anaesthesia control system for depth of anaesthesia and neuromuscular blockade," *Anaesthesia,* vol. 66, *(12),* pp. 1112-1120, 2011. Available: https://doi.org/10.1111/j.1365-2044.2011.06875.x.

[20] U.S. Food and Drug Administration. "The Device Development Process". [Online]. Available: https://www.fda.gov/ForPatients/Approvals/Devices/default.htm. [Accessed Apr. 25, 2019]

[21] Z. Jiang *et al*, "Heart-on-a-Chip: a closed-loop testing platform for implantable pacemakers," 2014.

[22] B. Parvinian *et al*, "Regulatory Considerations for Physiological Closed-Loop Controlled Medical Devices Used for Automated Critical Care," *Anesthesia & Analgesia,* vol. 126, *(6),* pp. 1916-1925, 2018. Available: https://doi.org/10.1213/ANE.0000000000002329.

[23] B. Parvinian *et al*, "Credibility Evidence for Computational Patient Models Used in the Development of Physiological Closed-Loop Controlled Devices for Critical Care Medicine," *Front. Physiol.,* vol. 10, 2019. Available: https://doi.org/10.3389/fphys.2019.00220.

[24] F. Gessa, P. Asare, R. Clipp, A. Bray and S. M. Poler, "A Test and Validation Framework for Closed-Loop Physiology Management Systems for Critical and Perioperative Care," *Medical Cyber-Physical Systems Workshop*, Porto, 2018.

[25] P. Asare. "A Framework for Reasoning about Patient Safety of Emerging Computer-Based Medical Technologies," *University of Virginia*, 2015

[26] A. Bray *et al*, "Pulse Physiology Engine: an Open-Source Software Platform for Computational Modeling of Human Medical Simulation," *SN Compr. Clin. Med,* vol. 1, *(5),* pp. 362-377, 2019. Available:https://doi.org/10.1007/s42399-019-00053-w.

[27] R. Brown *et al*, "Enhancing combat medic training with 3D virtual environments," *IEEE International Conference on Serious Games and Applications for Health (SeGAH),* May 2016. Available:https://doi.org/10.1109/SeGAH.2016.7586266.

[28] L. Potter et al, "Physiology informed virtual surgical planning: A case study with a virtual airway surgical planner and BioGears," *Medical Imaging 2017: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 10135. Available:https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10135/101351T/Physiology-informed-virtual-surgical-planning--a-case-study-with/10.1117/12.2252510.short. DOI: 10.1117/12.2252510.

[29] R. B. Clipp et al, "Pharmacokinetic and pharmacodynamic modeling in BioGears," *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC),* August 2016. Available: https://doi.org/10.1109/EMBC.2016.7590986.

[30] S. A. Edwards and E. A. Lee, "The semantics and execution of a synchronous block-diagram language," *Science of Computer Programming*, vol. 48, (1), pp. 21-42, 2003. Available: http://www.sciencedirect.com/science/article/pii/S0167642302000965. DOI: 10.1016/S0167-6423(02)00096-5.

[31] K. Maheshwari *et al*, "The relationship between ICU hypotension and in-hospital mortality and morbidity in septic patients," *Intensive Care Med.*, vol. 44, *(6)*, pp. 857-867, 2018. Available: https://doi.org/10.1007/s00134-018-5218-5.

[32] B. Yapps *et al*, "Hypotension in ICU Patients Receiving Vasopressor Therapy," *Scientific Reports*, vol. 7, *(1)*, pp. 8551, 2017. Available:https://doi.org/10.1038/s41598-017-08137-0.

[33] Kitware Inc. "Pulse: Patient Methodology". [Online]. Available:https://physiology.kitware.com/_patient_methodology.html. [Accessed Apr. 25, 2019]

[34] Capsule Technologies. "Advanced Medical Device Integration". [Online]. Available:https://www.capsuletech.com/integration [Accessed Apr. 25, 2019]

[35] P. Asare, C. Liu, Y. Xu, W. Kyaw, D. Karki, Y. Mittal, F. Gessa, A. Acharya, M. Qureshi and S. M. Poler, "Enabling Translational Research on Integrated and Closed-Loop Medical Systems Using an Open-Source Approach," in *40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2018)*, 2018.

[36] BeagleBone.org Foundation. "BeagleBone Black". [Online]. Available:https://beagleboard.org/black [Accessed Apr. 25, 2019]

[37] Canonical Ltd. "Ubuntu". [Online]. Available:https://www.ubuntu.com/ [Accessed Apr. 25, 2019]

[38] Canonical Ltd. "Ubuntu 16.04.6 LTS (Xenial Xerus)". [Online]. Available: http://releases.ubuntu.com/16.04/ [Accessed Apr. 25, 2019]

[39] Simon Kelley. "Dnsmasq". [Online]. Available http://thekelleys.org.uk/dnsmasq/doc.html [Accessed Apr. 25, 2019]

[40] Open Robotics Software Foundation. Robot Operating System (ROS). http://www.ros.org/ [Accessed Apr. 25, 2019]

[41] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, (10), pp. 1482-1493, 1991. Available:https://doi.rog/ 10.1109/26.103043.

[42] Open Source Robotics Foundation. "ROSBridge". [Online].

Available:http://wiki.ros.org/rosbridge_suite [Accessed Apr. 25, 2019]

[43] Fluke Biomedical, "ProSim 8 Vital Sign and ECG Patient Simulator". [Online]. Available:https://www.flukebiomedical.com/products/biomedical-test-equipment/patient-monitor-simulators/prosim-8-vital-signs-patient-simulator [Accessed Apr. 25, 2019]