THE MOTION-BASED GAMING PERIPHERAL MANAGEMENT (MBP-M) FRAMEWORK FOR ROBUST AND ACCURATE PROGRAM-DEVICE INTERACTION

A Thesis

Submitted to the Graduate School of the University of Notre Dame in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science and Engineering

by

Benjamin P. Bockstege IV

Dr. Aaron D. Striegel, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
August 2014
THE MOTION-BASED GAMING PERIPHERAL MANAGEMENT (MBP-M) FRAMEWORK FOR ROBUST AND ACCURATE PROGRAM-DEVICE INTERACTION

Abstract

by

Benjamin P. Bockstege IV

In recent years, a tremendous rise in consumer grade electronics has afforded bountiful opportunities for quantification and instrumentation of the human body, particularly through the use of motion-based gaming peripherals. Unfortunately, developing applications capable of reliably and robustly collecting data from such devices involves non-trivial complexities that must be accounted for. To combat these complexities, this thesis describes the Motion Based Gaming Peripherals Management (MBP-M) Framework. When integrated inside a program, the framework controls all device and data handling, thereby allowing developers to focus entirely on the intended use of device data. Explored in the thesis are examples of the use of gaming peripherals in non-gaming efforts and the core guiding principles, architecture, and data processing flow of the framework. Additionally, results of performance tests demonstrating the efficacy of the framework are presented and discussed, showing that the MBP-M Framework performs reliably and robustly and adheres to the guiding principles.
2.1  (a) The FitBit device for physical activity monitoring. (b) The Withings Smart Body Analyzer for weight, body composition, heart rate, and air quality monitoring. (c) The Withings Pulse O2 for activity, sleep, heart rate, and blood oxygen level monitoring.  

2.2  The Nintendo Wii Remote and Nunchuck attachment. Introduced alongside the Nintendo Wii in late 2006.  

2.3  A Nintendo Wii Balance Board. Introduced in late 2007 as a peripheral for the Nintendo Wii gaming system.  

2.4  An overview of the WeHab system. The patient stands on a Balance Board while the therapist controls the program using a Wii remote. Data are transferred from the board and Wii remote to WeHab using Bluetooth. Webcam video and audio is transferred via USB. Feedback is provided to both the patient and therapist through a large screen attached to the WeHab computer and located on a mobile cart. Data are saved to the local disk and encrypted upon completion of the session.  

2.5  (a) A Microsoft Kinect. Originally introduced as a peripheral for the Microsoft Xbox 360 in 2010. Re-released in a Windows-only form in 2012. (b) A Microsoft Kinect for the Microsoft Xbox One gaming system. Introduced alongside the Xbox One in 2013.  

3.1  A high-level overview of the MBP-M System.  

3.2  A high-level overview of the MBP-M System illustrating where the MBP-M System resides when included inside a program.  

3.3  The dataflows currently possible to generate from input coming from a Balance Board and Kinect. The middle items state the types of dataflows that may be generated. The images to the right are example visualizations that could potentially be generated from data provided by each dataflow.  

3.4  A class diagram for Instruments inside the MBP-M Framework. For Instruments there are two paths from the base class. An Instrument is either a direct representation of a device or is responsible for reporting of data in a specific format. See Figure 3.3 for example visualizations from COP and Skeleton Views.
3.5 A more detailed class diagram of the relationship between the Instrument base class and the specific device representations of Balance Boards and Kinects (Xbox 360 version). Standard UML syntax is used.

3.6 A high-level overview of the MBP-M Framework illustrating the location of each framework object in the data handling process.

3.7 The location of Channels, Minor Manipulations Channels, Output Instruments, and Dataflows inside the MBP-M Framework.

3.8 A class diagram for Channels inside the MBP-M Framework. There are currently three paths from the base Channel class; Balance Board Type, Multiple Type, and Skeleton Type Channels. Balance Board Type Channels take in Balance Board or similar data. Skeleton Type Channels take in data from devices which provide 3D joint locations. Multiple Type Channels bring together data input from multiple types of devices (eg. Balance Board and Kinect data).

3.9 An illustration of the events that take place when a device (a Nintendo Wii Balance Board in this case) reports new data. Shown are the steps the data takes going from the device to the Oracle main object and the updates that occur within the Oracle. Data logging occurs after delivery to the Oracle, but inside the Device Instrument object.

3.10 A basic overview of the events that take place on a tick of the Oracle’s timer. A Dataflow object representing the full data processing path for data coming from one or more Device Instruments is generated. Once the Dataflow is generated, a separate thread specific designated as the Dataflow processing thread is awoken and is instructed to run through the new Dataflow.

3.11 A timeline example of a brief window of MBP-M Framework operation to illustrate how the Oracle receives new updates from several Device Instruments. In this example, the Oracle is listening for updates from two Balance Boards and a Kinect.

3.12 Another timeline example of MBP-M Framework operation illustrating the interaction between Device Instrument data updates to the Oracle and ticks of the Dataflow generation timer. In this example, the Oracle is listening for updates from two Balance Boards and a Kinect.

4.1 A screenshot from the WeHab application presented in [1]. Shown is an example of a ‘trace’ being applied to a moving COP indicator illustrating where the COP was previously.

4.2 The device data collection and handling/visualization loops for the WeHab system.

4.3 The device data collection and handling/visualization loops for the MBP-M Framework.
4.4 CDF of the frame rates of the Main Program visualization frame rate on a single Balance Board setup for both the Laptop and Desktop when varying traces (see Figure 4.1) are applied.

4.5 CDF presenting the rate at which Balance Board data was received by the framework for sessions run on both the Laptop and Desktop when different trace lengths were applied to create view overloading.

4.6 CDF presenting the rates at which reports from the Kinect and Balance Board are received by the framework on the Laptop when only a Kinect and only a Balance Board are in use.

4.7 CDF presenting the rate at which Kinect data was received by the framework for sessions run on both the Laptop and Desktop.

4.8 CDF presenting the rate at which Balance Board data was received by the framework for sessions run on the Laptop for a variety of device configurations.

4.9 CDF presenting the time required to process data through a dataflow when trace applications of differing lengths are applied to Main Program data visualization on the Laptop.

4.10 CDF presenting the time required to process data through a dataflow for device use cases involving differing numbers of devices on the Laptop.
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Hardware Specifications for the Computers Used in MBP-M Framework Performance Testing</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>Instrument Frame Receipt Rates and Main Program Display Performance When Various Trace Lengths Are Applied to the Display of Data</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Distribution of Consecutive Device Updates Without Updates from Other Devices</td>
<td>59</td>
</tr>
<tr>
<td>4.4</td>
<td>Dataflow Processing Times for the Laptop and Desktop Computers When Traces of Varying Lengths Are Applied to Main Program Data Display</td>
<td>61</td>
</tr>
<tr>
<td>4.5</td>
<td>Dataflow Processing Times for the Laptop and Desktop Computers When Differing Device Configurations Are Used</td>
<td>63</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

For decades, there has been great interest in instrumentation of the human body as such instrumentation can lead to the quantification of a wide variety of aspects of the human body, health, and even behavior [1–17]. Such quantified aspects of health, behavior, and the body can range from the motion of specific limbs, to calorie intake, to mental faculties. Unfortunately, detailed quantification and instrumentation has long been out of reach for individuals as the devices capable of these quantifications have often been prohibitively expensive [18]. In recent years, however, drastically reduced monetary and power costs have led to consumer grade devices capable of monitoring a wide array of aspects of the human body [19–25] and the birth of the Quantified Self movement.

The Quantified Self movement [26] revolves around the concept of tracking quantifiable aspects of the life of an individual in order to encourage some form of personal improvement. These quantified aspects of life include many measures achievable through instrumentation of the human body and also sometimes include measures of mental status (e.g., life satisfaction, depression, mood) [19, 22, 27, 29]. The Quantified Self movement has naturally made itself best known within the realms of health-care given the numerous easily quantifiable metrics involved with tracking an individual’s health and the various possible uses of the collected data [30, 32]. For example, an individual could collect various health values of his/her life on a daily basis and then use these health values as part of attempts to force him/herself to adopt a healthier lifestyle. Likewise, automatically collected health data for indi-
viduals could potentially be used by health-care providers to aid in tracking patient health and decrease the time and frustration involved in patient diagnosis.

As noted earlier, recent years have seen tremendous new innovations in consumer grade devices [19–25]. Devices are smaller and require less power to operate while also being manufactured at lower prices and sometimes providing performance comparable to high-end versions of the devices [33]. A prime example of such consumer grade technology is the smart phone. Smart phones generally cost on the order of hundreds of dollars (as opposed to thousands) [34], have reasonable battery lives, and can contain computers, GPS, accelerometers, gyroscopes, microphones, wireless Internet connections, and more. Where high-end motion tracking systems can cost many thousands of dollars [18], a single smart phone that subjects likely already possess and other consumer grade devices can be used to collect similar, albeit less precise, data for a fraction of the cost.

An important area of interest for instrumentation of the human body is in instrumenting the motion of a body and its limbs [1–17, 32, 35–41]. By tracking the body and its limbs, observations can be made with regards to an individual’s balance capabilities, gait, limb movement precision, and posture [1–17, 32, 35–41]. As noted earlier, consumer grade devices have been decreasing in price, size, and power requirements while also increasing in capabilities. As such, consumer grade devices have begun to see use as a means to measure motion of the human body [1, 12–17, 36–51]. In particular, innovations in the realm of video gaming hardware have produced devices capable of motion tracking precision comparable to high-end devices while remaining well within the cost range of consumer grade electronics [23, 25, 33].

Motion based gaming peripherals, by virtue of their purpose as control devices for video games, must be accurate to a degree such that users do not become frustrated by the lack of game/device responsiveness [52] while also maintaining a price low enough that consumers will purchase the device. If a gaming peripheral lacks precision and
accuracy, users will naturally become frustrated as attempts to control the video game using the device result in game losses. Conversely, a perfectly precise and accurate device will not produce user frustration, but will see little use if the cost is too high. Motion based gaming peripherals must strike a middle ground between accuracy and cost to be profitable. For these reasons, gaming peripherals are well suited to the task of inexpensive instrumentation of the human body.

Unfortunately, while motion based gaming peripherals are well suited to inexpensive instrumentation of the human body, the act of implementing programs that make use of such devices can be tedious and is littered with non-trivial complications. Whether a device is being used for purposes in line with the Quantified Self movement or simply for interaction with a user interface [42, 43, 47, 48, 53], there are several important duties that must be performed by the code acting as the interface between the program and the device. First, code that represents the device and determines how the program should respond to incoming data must be implemented. Further, it is generally good practice to keep device handling on a thread separate from the main program thread so the program may continue to operate even as the device-handling code receives new data. However, such threading introduces new complications in the form of keeping the values of variables and data structures thread safe such that all threads are using the same values for each variable and structure and are using these values at the correct time. For example, when a force platform reports a set of sensor values, the code interface between the device and program must report the sensor values in such a way that the values the main program is currently working with are unaffected. A lack of thread safety could lead to unusual and incorrect results produced from a combination of two separate frames of data that were not supposed to be combined.

Furthermore, the difficulties and complexities of program-device interaction are accented when multiple devices are in use concurrently. Questions such as which
device should take precedence when data from both devices arrive at the same time and what happens when the main program has not finished handling the data from one device but receives data from a different device must be answered. Moreover, not all devices report at the same rate and the rate at which these devices report is often not even consistent. For example, one device could report at 100 Hz while a second device reports at 30 Hz. How data from these heterogeneously reporting devices is fused is a question that requires careful deliberation on the part of the developer.

In addition to issues with regards to threading and device interfaces, the simple fact that not all developers approach coding problems in exactly the same manner and style is an issue. If multiple developers are working on the same program, multiple device handling approaches and styles could be taken. Therefore, it is good practice to simplify and standardize the organization of devices such that, in the best situation, the main program can interact with all devices in the same exact manner. Such an approach limits the mental burden for developers of remembering how to work with each device and simplifies the main program by removing the need for various versions of the same code to deal with each type of device.

While expert developers should not have difficulty integrating peripheral device data input into programs, doing so can still be a time-consuming and tedious prospect. For non-expert coders though, this task can be time-consuming, confusing, and overwhelming and can result in non-robust code. However, if a framework is provided that manages the aspects all devices have in common (e.g., threading, data logging, and data reporting), both expert and non-expert developers may be saved the time and frustration of implementing such functionality. Rather than learning how to work with threads and ensure variables are accessed safely, non-expert developers may instead focus entirely on the main program and the purpose of the software.

The Motion-Based Gaming Peripherals Management (MBP-M) Framework is designed to combat the difficulties of program-device communication by providing a
robust framework for fusing information across multiple motion capture gaming peripherals that is both accurate and easy-to-use. Following are the guiding principles behind the MBP-M Framework...

- **Ease of device handling:** Interacting with devices should be simple and efficient. The intricacies of device pairing should be minimized and robust troubleshooting and debugging assistance are essential.

- **Reliable device data logging:** Logging of data must be done quickly and reliably. Data provenance is of utmost importance to ensure that no data is missed and the system state is recorded reliably.

- **Robust framework and operation:** The framework must be reliable and robust to various errors that can emerge during operation including peripheral performance heterogeneity and varying system loads. Poor or computationally excessive visualization should be compartmentalized.

- **Accurate data extraction:** Data retrieval must be straightforward and consistent. The system should have the capacity for simple data extraction and the state of the system at any point as well as any data flows should be able to be replayed with the utmost of precision.

In addition to these guiding principles, the challenge of delivering the desired characteristics of the framework with minimal overhead exists. If the framework is perfectly robust and accurate, but requires significant amounts of time to process each frame of data, the usefulness of the framework is greatly reduced, particularly if device feedback is a part of the main program. A perfectly accurate depiction of the current Center of Pressure of a subject during a visual feedback based task is of little use if the subject is only provided a handful of frames per second.
In this chapter, the Quantified Self movement is briefly explored and examples of devices and applications in the movement are presented. Additionally, the three major motion based gaming peripherals available today are described in detail and examples of how each has been used for non-gaming purposes explored.

2.1 The Quantified Self Movement

As noted in Chapter 1, the Quantified Self movement [26] revolves around the concept of tracking a variety of quantifiable aspects of the life of an individual in order to encourage some form of self improvement. Values tracked can range from easily quantifiable items like calorie intake, heart rate, and the distance walked over a period of time, to more subjective measures such as mood and life satisfaction. Through instrumentation of the human body and measurements of health and lifestyle, an individual may set goals for him/herself to adopt a healthier lifestyle and easily track progress to said goals.

A great many adults in the United States already participate in some form of health-oriented self quantification, tracking values such as weight, diet, and exercise routines [54]. In an effort to provide supply in the face of clear demand for quantified health tracking, over five-hundred companies were developing such quantified health tracking tools as of 2012 [55] and websites such as [56] help provide interested individuals with a variety of these tools. Some of the most popular applications and devices currently available include Fitbit [19] (Figure 2.1(a)), Digifit [20], RunKeeper [21],
and Mood Panda [27]. Fitbit, Digifit, and RunKeeper are all tools aimed at monitoring physical activity and thereby encouraging users to improve health through exercise. Mood Panda and similar applications, however, focus instead on tracking the mood and emotional stability of the user, aspects of life which can have just as much influence over health as physical activity [57, 58]. A prime example of a company providing a variety of devices aimed at health-oriented self quantification is Withings [22]. One of the products provided by Withings is a ‘Smart Body Analyzer’ (Figure 2.1(b)) that measures weight, body composition, heart rate, and air quality and automatically updates a linked smart phone application. Withings also sells wireless blood pressure monitors, sleep monitors, and a small watch-sized monitor of activity, sleep, heart rate and blood oxygen levels (Figure 2.1(c)). Like the smart body analyzer, these devices also link with a smart phone application for automated data tracking. Linkage with a smart phone application and automatic updates in particular make the products sold by Withings stand out as one of the problems with self quantification is that manual data entry/collection can eventually become a tedious and repetitive task.

Beyond tracking health, there are also several other areas where the Quantified Self has been applied. For example, there exist tools like Klout [28] that use user data to quantify the influence of the user and map out connections. Other tools such as Daytum [29] instead simply let users record any data they deem worthy of being reported to the world at large - the locations an individual has had cups of coffee over the course of a year, the cities where another individual stayed in a hotel in 2013, and time one more individual spent on Facebook on a daily basis for example. Even such products as Facebook, Twitter, LinkedIn, and Tumblr could be considered quantified self tools as they involve the logging of information about an individual.
2.2 Use of Gaming Peripherals

Largely beginning with the introduction of the Nintendo Wii gaming system in late 2006, motion based gaming peripherals have received interest for their potential in a variety of research efforts. As noted earlier, this interest results particularly from the low cost, both in monetary and setup terms, of gaming peripherals as compared to high-end motion capture technologies such as forceplates and marker based motion tracking (e.g., Flock of Birds). The low monetary cost of gaming peripherals affords possibilities for wide scale adoption of developed applications where an application developed around forceplates costing on the order of tens of thousands of dollars, for example, would be severely restricted to only researchers and developers who could afford such an investment. Likewise, the low cost in time and effort required for setup of gaming peripherals as compared to certain high-end devices such as marker based motion tracking further affords new application opportunities as high setup time costs can restrict use in cases where time is limited or where individuals involved
cannot be expected to remain still for long periods of time (e.g., children or autistic individuals).

2.2.1 Nintendo Wii Remote

The Nintendo Wii Remote (Wiimote) [23] (see Figure 2.2) was introduced alongside the Nintendo Wii in late 2006. The Wiimote has a total of twelve buttons (one on the back and not shown in Figure 2.2). Additionally, on the forward facing side of the Wiimote is an optical sensor used to determine the direction in which the Wiimote is pointing with regards to a reference emitter. In normal use, the Wii sensor bar is used as the reference emitter as it emits infrared light from two sets of five infrared LEDs located on either end of the sensor bar. The Wiimote also contains an accelerometer capable of sensing 3-dimensional acceleration. Furthermore, the functionality of the Wiimote can be extended through the attachment of additional peripherals such as the ‘Nunchuck’ attachment seen in Figure 2.2. In 2009, the Wiimote was enhanced with the MotionPlus extension (and later full incorporation into the Wiimote). The MotionPlus extension introduces a pair of gyroscopes which allow for the capture of more complex movements and the rotational orientation of the Wiimote. Additionally, the Wiimote also features several forms of user feedback. The Wiimote can vibrate, play sounds through an internal speaker, or activate/deactivate the on-controller LEDs (see bottom of Wiimote in Figure 2.2). The Wiimote connects to the Wii wirelessly via Bluetooth. Connection via Bluetooth also affords Wiimote connections with computers and therefore opens the Wiimote to entirely new uses.

Since its release, the Wiimote has been repurposed for a variety of non-gaming based efforts. For example, GlovePIE [42] is an application that allows the Wiimote to be used on a Windows computer to emulate a mouse or joystick. Another developer has made use of the Wiimote to achieve inexpensive finger tracking, multi-point interactive whiteboards, and headtracking for virtual reality displays [43]. Researchers
have also made use of the Wiimote for data collection in cognitive psychology experiments [44] and even to acquire and analyze the hand gestures of orchestra conductors [45].

2.2.2 Nintendo Balance Board

The Nintendo Wii Balance Board [24] (see Figure 2.3) was introduced in 2007 as a peripheral for the Nintendo Wii. The Balance Board consists of a large, rectangular platform and four pressure sensors located on the feet of the board (at the four corners of the board). Using these pressure sensors, the weight of an individual can be measured and the Center of Pressure (COP) of an individual on the board calculated. Despite being an inexpensive gaming peripheral, the Balance Board is comparable in performance to high-end forceplates in COP tracking [33]. As with the Wiimote, the Balance Board communicates with the Wii via a Bluetooth connection and can therefore also form connections with non-Wii computers for a variety of
non-gaming efforts, particularly in balance focused health-care.

Precision balance devices like the Balance Board afford the ability to incorporate real-time balance performance measurement and visual feedback directly into normal balance rehabilitation activities. Such Visual Feedback Forceplate Training (VFFT) allows therapists to directly target activities such as exploration of the limits of patient stability, targeted weight shifting, cognition of weight distribution, sit-to-stand, dizziness, and various other activities. A number of studies have demonstrated that providing such visual feedback yields improvements in postural sway [4], symmetry [5], dynamic balance [9], and functional abilities [10]. Notably, a common characteristic for many VFFT systems is that the relatively high cost of the systems often precludes their use in many clinical settings, let alone home environments for post-rehabilitation follow-up activities. For this reason, the low cost and high accuracy of the Balance Board as compared to high-end forceplates [33] has generated great intrigue in the possible applications of the Balance Board for balance rehabilitation. To begin, the weight-shift and balance activities in off-the-shelf Wii Fit games have been used for stroke rehabilitation [12] and cerebral palsy therapy [13].
ally, the ability to connect the Balance Board to a computer using Bluetooth has allowed researchers to create customized Balance Board applications. For example, researchers have used the Balance Board as a tool for COP tracking and feedback [14, 15] and as a change of standing posture (CSP) detector [16].

One noteworthy example of Balance Board use for visual feedback based research efforts and balance rehabilitation is WeHab [1]. Using the combination of one or two Balance Boards, a Wiimote, a standard Windows laptop, a large monitor, and the WeHab software, rehabilitation therapists may run patients through a variety of visual feedback oriented balance rehabilitation activities (see Figure 2.4). Additionally, the WeHab system also collects and logs metrics on patient balance performance and use of the system while the system is running, thereby allowing later analysis of patient performance.

While WeHab has been used successfully to gather balance performance data on patients in a large number of sessions [1], there are some problems with the implementation that violate the design principles behind the MBP-M Framework. To begin, WeHab is currently implemented in such a manner that much of the data logging and recording are handled with each displayed frame, violating the principles of Reliable device data logging and Accurate data extraction. As will be explained in greater detail in Chapter 4, this strategy can result in lost data as the rate of data visualization can be substantially lower than the rate of data collection. Furthermore, the rate of data visualization can be subject to change based on a variety of factors including the display of additional objects such as a trace of previous COP positions (see Chapter 4). Additionally, all the code for communicating with devices and most of the code for reacting to updates from the devices is located within a single class alongside code for data encryption, logging, system setup, and rehabilitation activity flow control, violating the principle of Ease of device handling. For these reasons, WeHab serves as an excellent example of a system that could benefit greatly from
Figure 2.4. An overview of the WeHab system. The patient stands on a Balance Board while the therapist controls the program using a Wii remote. Data are transferred from the board and Wii remote to WeHab using Bluetooth. Webcam video and audio is transferred via USB. Feedback is provided to both the patient and therapist through a large screen attached to the WeHab computer and located on a mobile cart. Data are saved to the local disk and encrypted upon completion of the session.
implementation of the MBP-M Framework.

2.2.3 Microsoft Kinect

The Microsoft Kinect [25] (see Figure 2.5(a)) was originally released as a peripheral for the Microsoft Xbox 360 gaming system in 2010. Contained in the Kinect are an RGB camera, an infrared based depth sensor, and a multi-array microphone, providing full-body 3-dimensional motion tracking, facial recognition, and voice recognition. In 2012, a new version of the Kinect designed specifically for use with Windows was released along with a software development kit (SDK), affording development of a wide array of new Kinect applications. In 2013 and alongside the release of the Microsoft Xbox One gaming system, another version of the Kinect was also released. This new Kinect (see Figure 2.5(b)) features various upgrades over its predecessor, including greater motion tracking accuracy, the ability to track without visible light, a wider field of view, tracking of additional joints, and tracking of multiple individual skeletons concurrently.

![Figure 2.5.](image)

(a) A Microsoft Kinect. Originally introduced as a peripheral for the Microsoft Xbox 360 in 2010. Re-released in a Windows-only form in 2012. (b) A Microsoft Kinect for the Microsoft Xbox One gaming system. Introduced alongside the Xbox One in 2013.
As with the Wii remote and Balance Board, the low monetary cost and setup time of the Kinect in comparison to motion tracking setups such as marker based approaches (e.g., flock of birds) [6, 8] and accelerometers [3, 35] has afforded a wide range of research efforts. Additionally, the lack of on-body markers when using the Kinect further opens application possibilities as setup time, cost, and user discomfort are all reduced. While cameras have also been used for human motion tracking [2, 7], the combination of depth sensing and an RGB camera affords the Kinect additional flexibility and applications.

The Kinect has been applied in a wide variety of research efforts ranging from post-neurotrauma rehabilitation to new forms of human-computer interaction. In the realm of health-care, the Kinect has often been used in combination with specially designed games for rehabilitation. For example, researchers have pursued the use of the Kinect and custom built games designed to make users perform specific therapeutic motions [36–39]. Likewise, other researchers have also used the Kinect alongside a specially designed game to train individuals suffering hemiparesis post-stroke [46]. Outside health-care, the Kinect has been used for gesture recognition [53], interaction with computers in novel ways [47], and making everything from walls to tables into touchscreens [48].

2.2.4 Combinations

In addition to using single devices, the low cost of gaming peripherals like the Balance Board and Kinect affords the opportunity to inexpensively combine data from multiple such devices and thereby produce a richer data set. For example, data from the Balance Board and Kinect have been combined by researchers to produce user-specific models for Center of Mass (COM) prediction [49, 51]. The same combination of devices has also been used to provide unique visual postural feedback by calculating COM from the Kinect and COP from the Wii Balance Board.
[40]. In addition, researchers have made use of multiple Kinects and projectors to allow for computer interaction both on surfaces and in open space [41].
CHAPTER 3

ARCHITECTURE

This chapter describes in detail the various components of the MBP-M Framework and the interactions each component has with the rest of the framework and the programs created by users of the framework. Additionally, the manner by which device data enters the framework, is logged, transformed, and finally delivered to the Main Program is explained and illustrated through examples.

As described in Chapter 1, the guiding principles of the MBP-M Framework are as follows...

- **Ease of device handling**: Interacting with devices should be simple and efficient. The intricacies of device pairing should be minimized and robust troubleshooting and debugging assistance are essential.

- **Reliable device data logging**: Logging of data must be done quickly and reliably. Data provenance is of utmost importance to ensure that no data is missed and the system state is recorded reliably.

- **Robust system and operation**: The system must be reliable and robust to various errors that can emerge during operation including peripheral performance heterogeneity and varying system loads. Poor or computationally excessive visualization should be compartmentalized.

- **Accurate data extraction**: Data retrieval must be straightforward and consistent. The system should have the capacity for simple data extraction and the state of the system at any point as well as any data flows should be able to be replayed with the utmost of precision.

From a designer/developer perspective, the first principle is important as it is directly related to making code development simpler and less time consuming. For the end user however, the final three principles are of the greatest importance as the
end user has no concerns about the ease of device handling or the inner workings of the program. The end user is only concerned with the robustness of the program and the accuracy and validity of the reported data. Applying these principles, the MBP-M Framework provides solid and flexible operation for end users and developers alike.

At its core, the MBP-M Framework is a dataflow management system. Using a variety of internal components responsible for data acquisition, data transforms, and data logging along with a structure to capture the flow of data, the MBP-M Framework covers the entire process of data handling from initial acquisition to output (see Figure 3.1). When used inside a program, the framework resides in the device interaction code of the program as illustrated in Figure 3.2. Additionally, to further separate device handling inside the framework from the functionality of the Main Program, the framework is run on its own thread. As the MBP-M Framework handles the entire process of data handling from initial acquisition to output for visualization, the only device handling code present in the main program involves the initial setup of the framework. Initial setup of the MBP-M Framework involves initialization of the framework itself and input of the parameters for the types of devices in use and the dataflow(s) to use. The possible dataflows that may be selected are derived from the types and number of devices provided. For example, if one Balance Board and one Kinect are provided, four dataflows are currently possible. Figure 3.3 shows the four currently possible dataflows that could be generated from Balance Board and Kinect input and also shows example visualizations that a developer could generate from the data resulting from each dataflow. The first two possible dataflows ignore one device while the third dataflow transforms and reports the Balance Board and Kinect data as separate entities (e.g., a COP indicator alongside a skeleton in the final visualization). Finally, the fourth dataflow merges the data from each device in some manner (e.g., using both Balance Board COP position and skeleton central hip X
3.1 Framework

As described above, the MBP-M Framework is flexible and robust, providing developers with an easy-to-use and easy-to-understand interface with devices. Presented below are the various base components of the MBP-M Framework...

- **Main Program:** The program into which the MBP-M Framework has been inserted. Has no direct communication with any elements of the framework beyond minimal interaction with the Oracle object (see ‘Oracle’ below).
Figure 3.2. A high-level overview of the MBP-M System illustrating where the MBP-M System resides when included inside a program.
Figure 3.3. The dataflows currently possible to generate from input coming from a Balance Board and Kinect. The middle items state the types of dataflows that may be generated. The images to the right are example visualizations that could potentially be generated from data provided by each dataflow.
• **Oracle:** Central object of the framework. Only one instance and handles management of all threads and objects. Also acts as the primary interface between the Main Program and the framework. Runs on a thread separate from the Main Program.

• **Instrument:** The MBP-M Framework representation of a device (e.g., Nintendo Wii Balance Board, Microsoft Kinect) or an output format. If representing a device, the Instrument either directly listens to or periodically queries its device to obtain data. If an Output, the Instrument takes in data to produce either a comprehensible collection of device data or a set of parameters a program can use for easy visualization of the data (e.g., normalized coordinates for an ellipse representing COP on a 2D plane).

• **Device Instrument:** The MBP-M Framework representations for specific device types. Devices can include such products as the Balance Board and the Kinect. The state of the Main Program also exists in the framework as a Device Instrument. All Device Instruments are run on their own thread separate from the Oracle and the Main Program.

• **Output Instrument:** MBP-M Framework objects responsible for the translation of data into an output format that is later provided to the Main Program. Output can be either purely data or a set of visualization parameters.

• **Display:** An Output instrument that translates incoming data into a specific set of parameters that can be easily visualized by the Main Program (e.g., normalized coordinates for an ellipse or coordinates and lengths for joints and bones on a skeleton).

• **Data Output:** An Output object that directly reports all device data rather than formatting the data into a set of parameters for easy visualization or focusing on a specific subset of the data.

• **Channel:** Performs a primary data transform on incoming data - Balance Board sensor values transformed into CoP X- and Y-coordinates for example.

• **Minor Manipulations Channel:** A special type of Channel responsible for optional minor manipulations that can be performed on most types of data - coordinate inversion and jitter application for example.

• **Dataflow:** A representation of the flow of data through the MBP-M Framework. Maintains a path of the specific Channels and Outputs device data is supposed to take.

• **Data Delivery Package:** An object in which data are stored for transport throughout the MBP-M Framework.
3.1.1 Location and Purpose of the MBP-M Framework Inside Programs

When used inside a program, the MBP-M Framework handles all aspects of device handling, interaction, and data reporting. As illustrated in Figure 3.2, The MBP-M Framework resides inside the Main Program, but only in the code responsible for device handling and interaction. The Main Program interacts directly with devices only during instantiation of the MBP-M Framework and when providing the framework with the parameters of what devices to use and dataflows to run.

As explained earlier through the four guiding principles, the MBP-M Framework exists to provide easy and robust device handling alongside reliable and consistent data logging. The encapsulation of all device handling and communication within the MBP-M Framework achieves the goal of easy and robust device handling as the Main Program must simply listen for and respond to device data updates sent by the Framework. Additionally, the position of the MBP-M inside the Main Program as a thread separate from the primary Main Program thread affords data collection, transformation, and reporting entirely disassociated with the visualization of said data. As such, data extraction and logging are isolated from any actions the Main Program may take and therefore kept reliable and consistent.

3.1.2 Oracle

The Oracle can be thought of as a central clearing house for the MBP-M Framework. The Oracle creates and maintains all Instruments, Channels, and Dataflows and all associated threads. Furthermore, the Oracle maintains a timer object on whose ‘ticks’ Dataflows are generated and activated using the most recent data received from each Device Instrument. Additionally, when Dataflows are complete, the Oracle is responsible for properly packaging the resultant data (either visualization parameters or simple data) and delivering said data to the Main Program. As the Main Program has no direct interaction with any internal MBP-M Framework objects
in order to maintain the simplicity of MBP-M use, the Oracle acts as the interface between the Main Program and the framework. The Main Program may request the status of Instruments in use and order the start and stop of data collection, but such interactions are handled entirely through the Oracle.

3.1.3 Instruments

The Instruments inside the Oracle are one of two concepts as illustrated in Figure 3.4. Instruments may either be Device Instruments (representing devices) or Output Instruments (responsible for transformation of data into an output format). All Device and Output Instruments, however, inherit from the base Instrument class. This base class contains the code common to the various types of devices and outputs, including logging, debugging and event handlers, thereby allowing the specific Device and Output Instrument classes to remain small and simple.

3.1.4 Device Instruments

Device Instruments are the MBP-M Framework’s representation of real-world devices (e.g., Balance Board, Kinect). The specific Instrument classes representing devices are themselves quite simple with their primary distinguishing features being the device they represent and how data from the device is obtained. Figure 3.5 provides a more detailed illustration of the relationship between the Instrument base class and the specific device representations of the Nintendo Balance Board and the Microsoft Kinect. Within the base Instrument class, the majority of operations are either handled completely or at least defined. Conversely, a minimal amount of code is implemented in the device classes such that only those functions and variables specific to each device are present. Some functions such as reportData, prepareLog, and start/stopDeviceReporting can also vary based on device. However, any code not device specific is relegated to the Instrument base class so as to maintain the
Figure 3.4. A class diagram for Instruments inside the MBP-M Framework. For Instruments there are two paths from the base class. An Instrument is either a direct representation of a device or is responsible for reporting of data in a specific format. See Figure 3.3 for example visualizations from COP and Skeleton Views.
simplicity of the device classes. In addition, the current state of the Main Program may also be recorded and tracked using a Device Instrument.

Two of the principles behind the MBP-M Framework are Reliable device data logging and Accurate data extraction. In accordance with these principles, device data received by a Device Instrument is immediately sent forward to the Oracle via a callback and then written to disk by the Instrument. The data is sent forward first as writing to disk is not an instantaneous operation. Additionally, Device Instruments are kept separate from the rest of the framework and in their own threads as shown in Figure 3.6, allowing data collection and logging to remain unhindered by operations.
3.1.5 Outputs

The children of the Output base class represent different methods for reporting data to the Main Program. As the Output base class is a child of the Instrument base class, the majority of logging and debugging are already in place, thereby allowing Output Instruments to remain fairly simple. Additionally, Output Instruments are only ever used at the end of Dataflow objects as shown in Figure 3.7. Output Instruments come in two major forms; Displays and Data Outputs.

Displays: Displays are Output objects designed for the purpose of data visualization.
Figure 3.7. The location of Channels, Minor Manipulations Channels, Output Instruments, and Dataflows inside the MBP-M Framework.
As such, Displays take device data and transform said data into a set of parameters which can then be used by the Main Program for easy visualization. For example, one Display may generate coordinate parameters necessary to generate an ellipse on a 2-dimensional plane based off of Center of Pressure (COP) coordinates provided by a Channel. Conversely, another Display may generate a customized skeleton of joint positions consisting of only a subset of joints selected by the Main Program based off of 3-dimensional joint location data collected by a Kinect (e.g., one version may report only hip, spine, shoulder, and neck joints for posture tracking purposes).

Data Outputs: In the case that visualization is not required or the developer plans to create a unique visualization and therefore needs all the data provided by a device, Data Output objects are used instead. Data Output objects simply coalesce data from one or more devices into a comprehensible whole for delivery to the Main Program. The Main Program may then make use of the data however the developer sees fit. For example, the developer may choose to display the data in a table or some form of specially designed window. Alternatively, the developer may wish to create a visualization similar to the third option in Figure 3.3 and needs all the device data points gathered by the framework. The developer may also choose to use the provided data to control custom built on-screen objects that the MBP-M Framework has no knowledge of.

3.1.6 Channels

Channels perform primary transformations of data from Device Instruments. A Channel receives raw data and performs some sort of transformation to produce new values with different possible uses. For example, a Channel receiving data from a Balance Board will generally convert the sensor values generated by the Balance Board into Center of Pressure (COP) X and Y coordinates which can then be used for a variety of purposes such as displaying a COP indicator on a 2-dimensional
plane (see Figure 3.3). As with Instruments and shown in Figure 3.8 Channels all inherit from a single base Channel class that contains all functions and variables common to all types of Channels. Also like Instruments, the code inside each specific Channel is quite simple. Each Channel simply performs that specific Channel’s data manipulation. Channel usage is always located inside Dataflows and is driven by the thread driving the Dataflow as shown in Figure 3.7.

3.1.7 Minor Manipulations Channels

Minor Manipulations Channels inherit from the base Channel class and perform various minor manipulations on data passed into them. Such minor manipulations can include coordinate inversion and applying artificial jitter to data. These minor manipulations are optional and are only used if the developer specifically has certain manipulations set to be used or if the end user of program chooses to activate them. As with normal Channels, Minor Manipulations Channels are located inside Dataflows and driven by the same thread driving the Dataflow (see Figure 3.7).

3.1.8 Dataflows

Dataflow objects provide a structure isolated from non data handling framework operation for the transformation and processing of device data. Contained inside a Dataflow object is a series of Channels through which device data from one or more devices are passed and concludes with an Output Instrument as illustrated in Figure 3.7. At a minimum, Dataflow objects consist of a single Channel, a Minor Manipulation Channel (for optional, user requested transformations), and an Output Instrument. Furthermore, as with Device Instruments and incoming device data, every transfer and transformation of data inside the Dataflow object is logged. The data transfer and transformation logs are, however, not written to disk until the close of the Main Program as writing to disk is not an instantaneous operation and would
Figure 3.8. A class diagram for Channels inside the MBP-M Framework. There are currently three paths from the base Channel class; Balance Board Type, Multiple Type, and Skeleton Type Channels. Balance Board Type Channels take in Balance Board or similar data. Skeleton Type Channels take in data from devices which provide 3D joint locations. Multiple Type Channels bring together data input from multiple types of devices (eg. Balance Board and Kinect data).
hinder Dataflow performance. Through the combination of logging inside Device Instruments and Dataflow objects, the entire process by which device data enters the framework, is transformed, and output to the Main Program is recorded and performed in a manner that is reliably consistent and easily followed/replayed. This reliability and consistency of logging ensures that the framework satisfies the design principles of Reliable device data logging and Accurate data extraction.

Before a Dataflow can be generated, the Oracle must have copies of recent data from devices, an acquisition process illustrated in Figure 3.9. When a Device Instrument object receives data from or queries the device it represents, that data is reported to the Oracle via a callback. The Oracle responds to the receipt of data from a Device Instrument by registering the received raw data package as the most recently received package from said Device Instrument. Only the most recently received package from each Device Instrument is tracked so as to keep data handling predictable and to prevent issues of non-real-time data reporting that can arise if an attempt is made to send every single frame reported by a device through Dataflows. For example, a Balance Board can report data at rates of approximately 100 Hz, but the MBP-M Framework is hardwired to generate Dataflows at a rate of 60 Hz. If the framework attempted to send every incoming frame of Balance Board data through a Dataflow, the framework would rapidly fall behind and be entirely unable to keep pace with the Balance Board data. Such a situation would result in the output of Balance Board data that is no longer real-time, an issue of particular importance when the Balance Board data is used for some form of visual feedback. The process of data receipt and tracking data updates continues unabated until either the program is closed or the Oracle is ready to transfer the data to a Dataflow.

When the Oracle has copies of recent device data, Dataflows are generated on the ‘ticks’ of a timer. This timer is located inside the Oracle and runs on a thread separate from the rest of the framework. As shown in Figure 3.10 and described
Figure 3.9. An illustration of the events that take place when a device (a Nintendo Wii Balance Board in this case) reports new data. Shown are the steps the data takes going from the device to the Oracle main object and the updates that occur within the Oracle. Data logging occurs after delivery to the Oracle, but inside the Device Instrument object.
earlier, contained in a Dataflow is the order of Channels to pass device data through and the concluding Output Instrument which will convert the transformed data into an appropriate form of Output. The most recent data for each device is sent through the Dataflow object, but whether or not the data is actually acted upon and output to the Main Program depends on the Channels and the Output Instrument. For example, a Balance Board and a Kinect may both be in use, but the Main Program can request to receive only the Balance Board data (see the first example in Figure 3.3). In such a case, a Dataflow object is generated which contains Channels and an Output Instrument devoted to Balance Board Center of Pressure (COP) reporting and ignores the Kinect data.

Once a Dataflow object has been generated, the Dataflow is handed off to a dedicated thread which proceeds to run the most recent device data through the Dataflow object. Starting with the first Channel in the Dataflow, the device data are systematically passed through each enqueued Channel and the concluding Output Instrument. As shown in Figure 3.7 and Figure 3.10, the first enqueued object in a Dataflow is always a Channel that performs a transformation on raw data. Next, the manipulated data are passed into either more Channels for additional transformations or into a Minor Manipulations Channel optional, minor manipulations (such as coordinate inversion) as requested by the Main Program. After all data transformations are completed, the data are passed into an Output Instrument for packaging as either a set of visualization parameters or simple data values. Finally, the fully transformed and packaged data are handed off to the Oracle for delivery to the Main Program, whereupon the Main Program uses the data for the purpose chosen by the developer. Additionally, an option also exists for the Main Program to provide its current status to the MBP-M Framework for logging. Such an action takes place once the Main Program has finished using the latest frame of device data.

The use of Dataflow objects alongside the structure of the MBP-M Framework
Figure 3.10. A basic overview of the events that take place on a tick of the Oracle’s timer. A Dataflow object representing the full data processing path for data coming from one or more Device Instruments is generated. Once the Dataflow is generated, a separate thread specific designated as the Dataflow processing thread is awoken and is instructed to run through the new Dataflow.

Most Recently Received Data

- BB_Instrument_0: Frame 5
- BB_Instrument_1: Frame 4
- Kinect_Instrument_0: Frame 2

Generate Data Flow

Send Most Recent Data through Flow

Fully Transformed Data

Deliver to Main Program
resolves several difficulties incurred by multiple device use. Namely, issues with data integrity and temporal accuracy are resolved along with ensuring real-time data output. When multiple devices are reporting to a single program, issues of data integrity and accuracy can arise. These issues can take many forms and crop up in several locations in the data handling, manipulation, and output process. Depending on the method by which data are transferred between device and computer, there is the chance that data may take a variable amount of time to arrive or even fail to arrive. For example, a wireless connection could be delayed or corrupted by objects between the device and computer or even a microwave in use nearby. Conversely, in the case of the Kinect, data can reach the computer at an inconsistent rate as a result of the data handling that takes place before the data is even reported to the Device Instrument. Moreover, there can also be several types of issues related to threading in the MBP-M Framework itself. If only a single thread is used for the entire system (Oracle, Device Instruments, Dataflows, and Main Program), the time required to process a single frame of data can take long enough that subsequent frames of data are not logged or logged at the incorrect time, introducing inaccuracies in the reported data. On the other hand, multiple threads with insufficient thread safety measures can lead to inaccurate data if the threads complete out of order or try to access a single object at the same instant. In addition, a quirk of using multiple threads exists in that while two threads may be instantiated in one order, it is entirely possible that the second thread will complete before the first. Partially because of this quirk of multi-threading, the MBP-M Framework currently only allows a single Dataflow to be processed at any one time.

In acknowledgment of the above difficulties incurred by multiple device use, a combination of threads and ‘locks’ are used in the MBP-M Framework. To begin and as noted earlier in this chapter, each Device Instrument is run in its own thread. Individual Device Instrument threads allow for device data receipt and logging unim-

36
peded by any other actions taken by the MBP-M Framework. Furthermore, rather than the Oracle querying the Device Instruments for the most recently received data, each individual Device Instrument notifies and provides the Oracle with the data as soon as the data is received. Such a strategy, however, can be problematic without proper thread safety. For example, if two Balance Boards report new data at or very near the same instant, issues of concurrent writes to a single data structure can create errors and potentially disrupt the order of updates to the Oracle. Furthermore, if a tick of the Oracle’s Dataflow generation timer occurs while a write is in process, the Dataflow could receive an inaccurate report of the current status of each device (see Sub-Section 3.4 for an illustrated example of such a situation). Such a situation is particularly problematic in the case of devices that report slowly as the Dataflow could be trapped using an old frame of data while the new frame of data was being logged. To prevent such read/write problems, locks are put on the relevant data structure whenever a Dataflow attempts to obtain the most recently received device data and when Device Instruments attempt to update said data. Additionally, running Dataflows on a thread separate from the rest of the MBP-M Framework allows the framework to isolate the transformation and output of data from the collection of data such that neither may interfere with the performance of the other.

In addition to the difficulties of out of order data handling, there is also an issue of non-real-time data output. Many programs that use peripheral devices are based around the idea of visual feedback and for such a program to work properly and avoid frustrating the user, the visualized data must be as close to real-time as possible. As noted earlier, however, some devices are capable of reporting data at rates exceeding the MBP-M Framework’s hardwired data processing rate (e.g., Balance Board reports at $\approx 100$ Hz while MBP-M Framework works at 60 Hz). As such, decisions must be made as to how much data should be handled and output and what to do with the data not processed. Attempting to process and output every single frame of data
produced by each Instrument has a high potential to lead to visualizations being multiple frames late; a gap that only grows larger the longer the program is active. To resolve this issue of non-real-time data output, the MBP-M Framework uses the earlier described strategy illustrated in Figure 3.10 of using only the most recent frame of data from each Device Instrument. By only using the most recent frame of data received from each Device Instrument, the above described backlog of frames is prevented and the final visualization produced by the Main Program represents as close to real-time data as possible.

3.2 Intra-Object Interactions

An important part of the MBP-M Framework is the isolation of different data handling operations. Data collection and logging from a device are kept entirely separate from data transformation and visualization, operations which are also kept entirely separate from each other. A Device Instrument representing a Balance Board does not know and does not need to know how the sensor values it reports will be used or if they will actually be used. All the Device Instrument knows is that it periodically receives updates from the Balance Board it represents and must collect the sensor values at such instances, package the values, and finally deliver the package to the Oracle via a callback. Likewise, Channels have no knowledge of exactly where incoming data comes from or where the data will be delivered. Channels do, however, know what type of data they are supposed to operate on and from what types of Device Instruments. The Channels know what data they are supposed to work with, but do not know where the data they are actually working with came from. Only the Oracle and Dataflow objects have knowledge of how data is transferred through the framework. Additionally, the Main Program has neither interactions with nor knowledge of the inner workings of the MBP-M Framework, thereby simplifying use of the framework for developers. Developers simply need to instantiate the framework
and then respond to the data that comes out of the framework.

In order to maximize flexibility and simplicity of data transfer between objects in the MBP-M Framework, a specialized object called a Data Delivery Package was implemented. The Data Delivery Package is a simple object that takes advantage of the fact that all classes in the C# programming language inherit from the same ultimate base `Object` class. The Data Delivery Package contains a mapping between variable names and values where the variable names are text-based strings and the values are `Objects`. The variable names used in Data Delivery Packages remain consistent in the framework through the use of static enums that all Instruments, Channels, and Outputs posses and can access. Just as with the Oracle, the Data Delivery Package simply transporting the data from one location to another in the framework and makes absolutely no modifications to the data. Unlike any other MBP-M Framework Object, however, the Data Delivery Packages have no logging functionality.

3.3 Flexibility

The framework described above, while robust, also allows for a great deal of flexibility and developer customization. If a developer finds that he/she needs to perform a more specific type of manipulation on data provided by a certain kind of device, all he/she must do is create a new Channel object inheriting from one of the existing base Channel types or even create an entirely new type if necessary. On the other hand, if the developer wants to start gathering data from a new type of device, he/she simply needs to create a new class inheriting from the base Instrument class and implement the code that will collect data from the device. In both cases, there exists ample documentation throughout the entire framework to aid in understanding and to make creating new classes as simple as possible. Also provided are template device, Channel, and Output classes which further explain the steps needed to create a new class of each type.
The MBP-M Framework currently supports connections with the Nintendo Wii Remote (Wiimote), Nintendo Wii Balance Board, and Microsoft Kinect (1st generation Kinect for Windows). As the Wiimote and Balance Board connect via Bluetooth, the framework uses the 32feet.NET library (Version 3.4) \cite{59} to establish and manage Bluetooth connections. After connection, the WiimoteLib library (Version 1.7) \cite{60} is used to send commands to and receive data updates from the Wiimote and Balance Board. The Kinect, on the other hand, is connected via USB and communication is achieved through the Microsoft developed Kinect for Windows SDK (Version 1.8) \cite{25}. If developers wish to implement a new device in the MBP-M Framework, they need only create a new Device Instrument class representing the new device and include the libraries and code required to send commands to and receive data from the devices.

3.4 Example Time Windows

Figure 3.11 presents a brief and simple window of MBP-M Framework use during which two Balance Boards and a Kinect are reporting data. When the Oracle receives a new frame of data from a Device Instrument, a mapping is created between the reporting Instrument and the data. If a mapping already exists between the reporting Instrument and a data package, then the mapped data is replaced with the new data. It can be the case that the frame of data being overwritten was never processed and output by a Dataflow, but as explained in Sub-Section 3.1.8 attempting to process every frame of data provided by a device can lead to non-real-time data output. Even if a frame of data is never output to the Main Program, the data is still logged and saved to disk by the respective Device Instrument.

Figure 3.12 presents another brief window of MBP-M Framework use. Where Figure 3.11 illustrated how the Oracle’s mappings Device Instruments and most recently received data are updated with incoming device data, Figure 3.12 shows the
Figure 3.11. A timeline example of a brief window of MBP-M Framework operation to illustrate how the Oracle receives new updates from several Device Instruments. In this example, the Oracle is listening for updates from two Balance Boards and a Kinect.
manner in which device data updates and the Dataflow generation timer interact. As noted earlier, to assuage thread safety concerns (and comply with the MBP-M design principle of *Robust framework and operation*), the Oracle mappings of Device Instruments to most recently received data are locked against any read or write request coming from other threads whenever a device update arrives or the Dataflow generation timer ticks. A lockdown occurs because, as illustrated in Figure 3.12, it is possible for the Dataflow generation timer to tick and a new frame of device data to arrive in the Oracle at very near the same instant. If the Oracle’s mappings of Device Instruments to most recent data is locked down, the thread(s) waiting for read/write access wait a short period of time for the lock to be removed and then proceed with their read/write. As both read and write operation are completed rapidly, the wait time for access remains short (on a scale of nano-seconds) and is non-disruptive to framework operation. A queue of the most recent write attempts by each device is also maintained to avoid the risk of one device being locked out of updating by the other devices. As multi-threaded environments have no natural concept of ordering, a single thread could be forced to wait through several locking cycles even if it has waited longer than other threads waiting for access. For example, updates from one Balance Board could remain unreceived for several concurrent frames if the other device update threads consistently receive access before this one Balance Board.
Figure 3.12. Another timeline example of MBP-M Framework operation illustrating the interaction between Device Instrument data updates to the Oracle and ticks of the Dataflow generation timer. In this example, the Oracle is listening for updates from two Balance Boards and a Kinect.
CHAPTER 4

PERFORMANCE RESULTS

This chapter presents tests run using a variety of device configurations and conditions to assess the performance of the MBP-M Framework. The results from these tests are discussed and linked with the complexities of motion based gaming peripheral based software development and the guiding principles of the framework.¹

4.1 Demonstration of Framework Isolation and Performance Tests

To illustrate the isolation of data logging in the MBP-M Framework and test the performance of the framework, several batteries of tests were run. All tests were run using the same, simple format. For each session, a subject stood quietly for 65 seconds in front of a Kinect, on one or two Balance Boards, or some combination. As the beginning and end of each session can include dirty data gathered as the subject was entering and leaving the Kinect field of view or stepping on and off the Balance Board, the first fifteen seconds and final five seconds of each session were ignored during analysis. From the forty-five seconds of clean data, several thousand data points are logged with the exact number dependent on the data being logged. For example, a Balance Board Instrument log reporting at a rate of approximately 100 Hz can contain upwards of 4500 log entries whereas the Dataflow log can contain nearly 3000 log entries (≈66 Hz * 45 seconds). For analysis, the thousands of log entries are condensed into averages over jumping one second windows, producing 45

¹These results are also presented in [01]
TABLE 4.1
HARDWARE SPECIFICATIONS FOR THE COMPUTERS USED IN MBP-M FRAMEWORK PERFORMANCE TESTING

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Computer 1 - Laptop</th>
<th>Computer 2 - Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Windows 7 64-bit</td>
<td>Windows 7 64-bit</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Core i5-2450M @ 2.50 GHz</td>
<td>Intel Core i7-4770 @ 3.40 GHz</td>
</tr>
<tr>
<td>CPU Cores</td>
<td>2 Cores</td>
<td>4 Cores</td>
</tr>
<tr>
<td>Threads</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Hyper Threading</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RAM</td>
<td>4 GB</td>
<td>4 GB</td>
</tr>
</tbody>
</table>

data points for each session data file (Device Instrument logs, Dataflow log, Main Program log, Oracle log).

All tests were run on two separate computers - a laptop and a desktop (hereafter referred to as such). The hardware specifications for the two computers are presented in Table 4.1. These two computers were used for the purpose of illustrating differences in framework performance between a ‘typical’ laptop computer and a powerful, dedicated desktop computer. For the Laptop, tests were repeated ten times while for the Desktop, tests were repeated five times. The tests on both computers were run using a custom built WPF application with the framework integrated into its operation. This WPF application possessed simple controls for starting/ending sessions and a visualization region on which device data was displayed (generally as COP or skeletal joint coordinates on a 2-dimensional plane).

To demonstrate the isolation of the MBP-M Framework from actions taken in
the Main Program, several batteries of tests were run in which Main Program data visualization was overloaded. In view overloading, the frame rate of data visualization is engineered to remain consistently low as a result of extensive time requirements for display operations. If an application is designed in such a way that device data logging is not kept separate from display of data, it is entirely possible for incoming device data to be missed. Such an application design violates the MBP-M Framework design principles of Reliable device data logging and Accurate data extraction as data logging is neither reliable nor an accurate depiction of all of the reported device data. An example of view overloading can be found in the Center of Pressure (COP) tracing option of the Wehab application described in [1] and Chapter 2. Tracing COP involves maintaining a set of previous COP positions and displaying these previous positions in some manner alongside the new COP location. If frames are processed at 60 Hz (with no regard to slowdown), then a 500 point trace would include COP points processed over the past 8.33 seconds. In both WeHab and the MBP-M Framework testing application, the trace is a series of lines as seen in Figure 4.1. Unfortunately, the time required for an application to draw every one of potentially hundreds of lines in a trace is non-trivial and therefore decreases the display frame rate (see Section 4.2).

Figure 4.2 shows the device data collection and data handling/visualization loops for the WeHab system. While device data is collected by WeHab at a rate consistent with the rate at which device updates are made, logging of device data is completed inside the main User Interface (UI) dispatcher thread. As the timer used by WeHab is a Dispatcher timer, the timer runs on the same thread as the UI thread. Therefore, the interval between timer ticks is influenced by any other operations taken in the UI thread, including data visualization. As such, while the current state of the Balance Board is current when logging takes place, not every device update is logged and the time between logs can vary widely depending on other events in the UI thread.
Figure 4.1. A screenshot from the WeHab application presented in [1]. Shown is an example of a ‘trace’ being applied to a moving COP indicator illustrating where the COP was previously.
Unlike WeHab, the MBP-M Framework isolates device data logging from visualization, allowing device data logging to continue unhindered by poor display frame rate. Figure 4.3 shows the device data collection and data handling/visualization loops for the MBP-M Framework. As described in Chapter 3, when devices send updates to the framework, the data in the update is immediately sent forward to the Oracle to be registered as the most recent data from the device and then logged to disk by the Device Instrument. This strategy ensures that all device data is logged no matter the events or actions that take place in the Main Program or even the framework Dataflow processing thread. Also unlike WeHab, the Dataflow generation timer in the framework is run on a separate thread from both the UI and the framework itself, allowing a much more consistent tick interval that is uninfluenced by Main Program and framework actions. To demonstrate this isolation of device data logging from visualization in the MBP-M Framework, tests were performed in the same format as described earlier, but with only a single Balance Board in use and COP traces of length 0, 100, 250, and 500 applied to the Main Program display.

In addition to demonstrating the isolation of data logging and visualization when using the framework, tests meant to assess the performance of the MBP-M Framework were also run. As described earlier, configurations of devices involving several combinations of Balance Boards and a Kinect were used for multiple sessions each. The performance results of the framework are drawn from these tests. In particular, the ability of the framework to achieve consistent and reliable device data logging based on the number and type of devices in use was assessed (necessary to satisfy the design principles of **Reliable device data logging** and **Accurate data extraction**). Not all devices report data at the same rate and these rates are often not even perfectly consistent. As such, the framework must be able to properly and reliably log and make use of the data arriving from each connected device even if the devices report data at inconsistent rates. In addition to demonstrating the ability of the framework
Balance Board and Wiimote Updates

Update sent from device \rightarrow \text{State of device set to the update}

Data Logging

Potential slowdowns due to tracing and other visualizations

Dispatcher

Timer tick (15 ms interval)

Update views (move COP, apply trace, update targets)

Calculate COP from latest Balance Board state

Log all device data and visualization state

Figure 4.2. The device data collection and handling/visualization loops for the WeHab system.
Figure 4.3. The device data collection and handling/visualization loops for the MBP-M Framework.
to maintain consistent device data receipt rates no matter the devices in use, the
above described tests were also used to assess the time required for Dataflows to
fully process data. As stated in Chapter 1, an overarching goal of the framework is
to adhere to each of the design principles with minimal overhead cost. A perfectly
accurate and robust framework is of little use if a high cost in system resources and
handling time is incurred such that the framework is responsible for slowdowns in
the Main Program.

4.2 Results: View Overloading

One of the areas in which poor design and development choices can adversely affect
data collection and logging is data visualization. In particular, overloading the display
of a program in such a manner that visualization requires a substantial amount of time
can cause significant reductions in program performance. As explained earlier, view
overloading for the framework performance tests was achieved by applying traces
(see [1] and Figure 4.1) of lengths 100, 250, and 500 previous COP points to the
visualization of data. The length of the trace directly influences the amount of time
required for the Main Program display to complete a ‘draw’ operation and therefore
decreases the number of frames displayed per second as evidenced in Table 4.2. From
Table 4.2, it can be seen that as the length of the trace increases, the average time
required to display a single frame of data increases as well (ANOVA p-value of less
than 0.001 for all). In hand with the increase in display time is a sizable decrease in
display frame rate to points that render the program nearly unusable [52] (ANOVA
p-value of less than 0.001 for all except between no trace and 100 point trace for the
Desktop). Figure 4.4 graphically shows the differences in display frame rate for each
trace length for both the Laptop and Desktop and the distribution of the display
frame rates. In line with the results presented in Table 4.2, Figure 4.4 illustrates that
longer traces result in lower frame rates for both the Laptop and Desktop, but that
TABLE 4.2

INSTRUMENT FRAME RECEIPT RATES AND MAIN PROGRAM DISPLAY PERFORMANCE WHEN VARIOUS TRACE LENGTHS ARE APPLIED TO THE DISPLAY OF DATA

<table>
<thead>
<tr>
<th>Instrument Frame Receipt Rate (Hz)</th>
<th>Average Display Frame Rate (Hz)</th>
<th>Average Display Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>Desktop</td>
<td>Laptop</td>
</tr>
<tr>
<td>No Trace</td>
<td>102 ± 2</td>
<td>104 ± 1</td>
</tr>
<tr>
<td>100 Points</td>
<td>102 ± 2</td>
<td>104 ± 1</td>
</tr>
<tr>
<td>250 Points</td>
<td>101 ± 2</td>
<td>104 ± 1</td>
</tr>
<tr>
<td>500 Points</td>
<td>100 ± 3</td>
<td>105 ± 1</td>
</tr>
</tbody>
</table>

the Laptop with its less powerful CPU and fewer cores capable of fewer concurrent threads is more seriously affected.

Table 4.2 also shows that the rate at which frames of device data are received and logged by the framework remains largely unaffected by view overloading, as intended. The time between Balance Board frame receipts was measured using a Stopwatch object with a millisecond granularity. On data receipt, the stopwatch is immediately stopped, the elapsed time retrieved, and the stopwatch reset and restarted. In this manner, the device data receipt rates in Table 4.2 and Figure 4.5 are a measure of the time elapsed between updates the device makes to the framework. As with Table 4.2, Figure 4.5 shows that the rate at which frames of Balance Board data were received by the framework on the Desktop remained remarkably consistent. For the Laptop,
Figure 4.4. CDF of the frame rates of the Main Program visualization frame rate on a single Balance Board setup for the both the Laptop and Desktop when varying traces (see Figure 4.1) are applied.

however, it can be seen that the rate at which Balance Board data was received and logged was below 100 Hz approximately 30% of the time when traces of length 250 and 500 were applied. The decrease is small enough, though, as to make no difference in overall framework performance. This slight drop seen in device data receipt rate with increasing trace length and the generally slower performance for the Laptop (ANOVA p-value of less than 0.001 for all when compared with the Desktop) can again be attributed to the less powerful Laptop CPU as compared to the Desktop CPU.

The MBP-M Framework avoids being affected by view overloading and suffering the slowdown seen in the average display frame rates and average display times of Table 4.2 through its use of multi-threading. As described in Chapter 3 all data logging is handled on threads separate from the Main Program’s display thread, thereby allowing logging to continue unhindered even if the main program experiences
Figure 4.5. CDF presenting the rate at which Balance Board data was received by the framework for sessions run on both the Laptop and Desktop in which different trace lengths were applied to create view overloading.

some sort of slow down like an overburdened display. As already noted, the success of this strategy is evident in the instrument frame receipt rates of Table 4.2 and Figure 4.5. Despite decreasing display frame rates caused by the application of longer traces, the rate at which frames of Balance Board data are received and then logged by the framework remains consistent.

When logging is coupled with display (like in the WeHab application) and a substantial amount of time is required to completely display a frame, multiple frames of data reported by a device can be ignored while the program works to visualize a previous frame. For example, if a 500 point trace is applied and device data is logged with every displayed frame, then (using the Laptop values) nearly ninety frames of device data remain unlogged every second. While it seems at first that logging data at the same moment as data display is a poor design decision due to the potential and danger of slowdowns, the ability to log the exact state of the system at the same
moment the user sees the output is quite useful for understanding exactly what the user saw at a given time. Logging the state of the system at the same moment as the data is visualized can still be done, but such logging should remain separate from device data logging as device data logging occurs at a generally greater rate and must not be interrupted for accurate and reliable logging to be achieved.

4.3 Results: Heterogeneous Instrumentation

An unfortunate fact about motion based gaming peripherals is that not all devices collect data at the same rate. Worse still, the rates at which these devices collect data are not always consistent. Figure 4.6 shows that the Kinect and Balance Board clearly report at different rates with the Kinect reporting less than half as often as the Balance Board. It should also be noted, however, that although the Balance Board reports at $\approx 100$ Hz, only about 64% of frames are actually unique (at least one reported sensor value is different than in the previous frame). When only unique frames are sent to the Oracle, only $\approx 65$ frames are reported per second. Furthermore, as stated earlier with regards to consistent device reporting rates, although Figure 4.6 does show report rates for the Kinect and Balance Board to be highly consistent, the consistency is not perfect. Likewise, Figure 4.7 shows that the rates at which the Kinect reports to the framework on both the Desktop and Laptop when several different Kinect inclusive device configurations are used is highly consistent, but again is not perfectly consistent as noted particularly by the slightly lower performance of two Balance Boards and a Kinect on the Laptop. This slightly lower data receipt rate can again be attributed to the less powerful CPU present in the Laptop as compared to the Desktop CPU and its ability to run fewer threads concurrently. Similar device update inconsistency as a result of CPU capabilities and the number of devices in use can be seen in Figure 4.8. In Figure 4.8, the rates at which the Balance Board reports to the framework in Balance Board inclusive device configurations are shown.
Figure 4.6. CDF presenting the rates at which reports from the Kinect and Balance Board are received by the framework on the Laptop when only a Kinect and only a Balance Board are in use.

to vary between 94 and 104 Hz, although the vast majority of session time sees device report rates upwards of 102 Hz. The most important point regarding Figures 4.7 and 4.8 however, is that no matter the number of devices in use, the framework still receives and logs device data at the same rate at which devices updates arrive and never fails to log a frame. Furthermore, even though device report rates show some inconsistencies, it is also important to note that the rates remain essentially the same and remain as such no matter the number of devices in use.

A system that does not account for discrepancies in frame rate between devices in a safe and predictable manner could potentially produce poor output for the end user. For example, if reports are throttled to the rate of the slowest reporting device, the system could produce poor display frame rates and therefore negatively impact user satisfaction if the slowest reporting device is below 30 Hz. On the other hand, a system that simply reports with every newly received frame of data would
Figure 4.7. CDF presenting the rate at which Kinect data was received by the framework for sessions run on both the Laptop and Desktop.

Figure 4.8. CDF presenting the rate at which Balance Board data was received by the framework for sessions run on the Laptop for a variety of device configurations.
be inconsistent and unpredictable in both report rate and the order in which devices report to the Main Program, a violation of the MBP-M Framework’s principles of Reliable device data logging and Accurate data extraction.

The lack of perfect consistency between different types of devices and even between devices of the same type raises questions of what should happen when, for example, a single Channel modifies and passes along a combination of data coming from a Balance Board and a Kinect. Since the Kinect reports at less than half the rate at which the Balance Board reports, with what frequency should the Channel perform and pass along its transformations? As described in Chapter 3, the MBP-M Framework is designed such that Dataflows, and therefore Channels, always use the most recently received data from each of the devices currently active in the framework. Therefore, a Channel listening to both a Balance Board and a Kinect will potentially use the same Kinect data multiple times as that data comes from the most recently known state of the Kinect. While the most recently known state of the Kinect may remain the same between consecutive dataflows, the most recently known state of the Balance Board is changing and therefore, the Channel’s output is also changing and must be reported.

Table 4.3 presents situations in which a Dataflow listens to a Balance Board and a Kinect in one case, and two Balance Boards and a Kinect in the second case. Table 4.3 also shows the distribution of the total number of device updates to the Oracle between updates for each specific device. For two Balance Boards and Kinect, in the time between 97% of consecutive Kinect reports at least 4 updates from the Balance Boards arrived in the Oracle. Conversely, for one Balance Board and one Kinect, the time between 69.3% of consecutive Balance Board updates to the Oracle saw no Kinect updates arriving in the Oracle. Both of these examples help illustrate that a potentially large number of Balance Board updates can reach the Oracle between Kinect updates.
4.4 Results: MBP-M Framework Overhead

While it is important that the MBP-M Framework provide robust and accurate data logging, it is also important that the system does no harm. Overhead imposed by such matters as data manipulation, logging, and data transfer must be kept to a minimum. Even if the framework provides perfectly accurate, consistent, and reliable data logging and reporting, the framework would be of little use if it requires several seconds to process every single frame. Furthermore, as motion-based gaming peripherals are particularly useful for and were originally designed for the purpose of visual feedback, high-speed data handling is essential in a broad number of possible applications.

The framework is currently hardwired to generate and run a Dataflow every fifteen milliseconds, thereby achieving a maximum report rate to the Main Program of $\approx 66.7$ Hz. As long as Dataflow processing time remains low enough such that the display frame rate remains above $\approx 60$ Hz (or 30 Hz when the Kinect alone is in use due

### TABLE 4.3

**DISTRIBUTION OF CONSECUTIVE DEVICE UPDATES WITHOUT UPDATES FROM OTHER DEVICES**

<table>
<thead>
<tr>
<th></th>
<th>BB and Kinect</th>
<th>2 BB and Kinect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BB</td>
<td>Kinect</td>
</tr>
<tr>
<td>0</td>
<td>69.3%</td>
<td>0.0%</td>
</tr>
<tr>
<td>1</td>
<td>30.7%</td>
<td>1.5%</td>
</tr>
<tr>
<td>2</td>
<td>0.0%</td>
<td>17.2%</td>
</tr>
<tr>
<td>3</td>
<td>0.0%</td>
<td>39.8%</td>
</tr>
<tr>
<td>$\geq 4$</td>
<td>0.0%</td>
<td>41.5%</td>
</tr>
</tbody>
</table>
to the \(\approx30\) Hz report rate of Kinect), then the MBP-M Framework is performing acceptably. 60 Hz is considered acceptable as a display frame rate of 60 Hz appears fluid and understandable for most users \[52\]. In order for the display frame rate to remain above 60 Hz, the combination of the Dataflow generation interval of 15 milliseconds and Dataflow processing time must remain below 17 milliseconds (60 frames \(\times\) 17 ms = 1020 ms). A frame rate of 30 Hz is also seen as acceptable \[52\], however, so a combined processing time of 22 ms can also be considered viable.

*Processing Time and View Overloading:* As evidenced by the processing times in Table 4.4, overhead caused by the framework during the Dataflow handling process is kept quite low, but can be influenced by view overloading in the Main Program (Laptop and Desktop processing time differences are statistically significant with a p-value of less than 0.001). This observation is further reinforced by Figure 4.9 where it can be seen that approximately 70\% of Dataflow processing times remained consistently low no matter the trace length, but the longer trace lengths were accompanied by greater percentages of processing times in excess of five milliseconds. While Dataflow processing completes before the data is sent to the Main Program and is run on a thread separate from both the Main Program and the Oracle, the simple fact exists that system resources on a computer are limited. Therefore, as longer traces are applied to a COP visualization and more time is required to complete the draw operation, the draw operation ties up system resources for longer periods of time. Notably, however, the effect on Dataflow processing time of applying a trace to data visualization is quite small in comparison to the display frame rate drop caused by drawing the trace. Additionally, as was the case with the lower Main Program display frame rates seen on the Laptop against the Desktop, the generally longer Dataflow processing times for the Laptop can again be attributed to the less powerful CPU inside the Laptop as compared to the Desktop. Despite the differing processing times based on degrees of view overloading, the important point to take from Table 4.4 and
TABLE 4.4

DATAFLOW PROCESSING TIMES FOR THE LAPTOP AND DESKTOP COMPUTERS WHEN TRACES OF VARYING LENGTHS ARE APPLIED TO MAIN PROGRAM DATA DISPLAY

<table>
<thead>
<tr>
<th>Dataflow Processing Time (ms)</th>
<th>Laptop</th>
<th>Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB Trace 0</td>
<td>1.5 ± 2.2</td>
<td>0.8 ± 1.2</td>
</tr>
<tr>
<td>BB Trace 100</td>
<td>1.8 ± 2.3</td>
<td>0.4 ± 0.5</td>
</tr>
<tr>
<td>BB Trace 250</td>
<td>2.2 ± 2.7</td>
<td>0.5 ± 0.8</td>
</tr>
<tr>
<td>BB Trace 500</td>
<td>2.4 ± 3.1</td>
<td>0.6 ± 1.1</td>
</tr>
</tbody>
</table>

Figure 4.9 is that the Dataflow processing time remains low and largely consistent no matter the degree of view overloading.

Processing Time and Differing Device Configurations: Table 4.5 shows that the total time to process a frame of data combined with the Dataflow generation interval consistently remains below 17 ms for the Desktop no matter the device configuration, but can grow to 19 ms for the Laptop as more devices are added (processing time differences between the Laptop and Desktop for each case are statistically significant with a p-value of less than 0.001). This observation is supported by Figure 4.10 from which it can be seen that Dataflow processing times are highly consistent for single and dual device usage not only in average time, but also in the distribution of processing times. Moreover, Figure 4.10 also shows that between 70 and 80% of Dataflow processing
times were below three milliseconds no matter the number of devices in use, although greater numbers of devices were linked with greater percentages of processing times above five milliseconds. Such behavior is expected because although device data handling and logging take place in separate threads from the Dataflow, the existence of more threads with higher priority (Device Instrument update threads) means that the Dataflow processing thread is not handled as soon or as quickly as when fewer high priority threads are in action. Notably, the Laptop again did not perform as well as the Desktop. This difference in performance can once again be attributed to the less powerful CPU in the Laptop and, in particular, the fewer cores in the Laptop CPU and the ability of the Laptop to only run four concurrent threads as compared to the eight concurrent threads the Desktop CPU can run. Additionally, although differences do exist in performance between device configurations, it is noteworthy that these differences are generally small enough such that the final output of data is
### TABLE 4.5

**DATAFLOW PROCESSING TIMES FOR THE LAPTOP AND DESKTOP COMPUTERS WHEN DIFFERING DEVICE CONFIGURATIONS ARE USED**

<table>
<thead>
<tr>
<th></th>
<th>Dataflow Processing Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Laptop</td>
</tr>
<tr>
<td>BB Alone</td>
<td>1.5 ± 2.2</td>
</tr>
<tr>
<td>Kinect Alone</td>
<td>1.6 ± 2.2</td>
</tr>
<tr>
<td>BB and Kinect</td>
<td>2.7 ± 3.8</td>
</tr>
<tr>
<td>2 BB</td>
<td>2.32 ± 3.2</td>
</tr>
<tr>
<td>2 BB and Kinect</td>
<td>3.5 ± 5.8</td>
</tr>
</tbody>
</table>

minimally affected with the possible exception of two Balance Boards and a Kinect on the Laptop. Moreover, as was the case with Dataflow processing times and varying degrees of view overloading, the large majority of Dataflow processing times are below three milliseconds no matter the number of devices in use, showing that the framework satisfies the guiding principle of **Robust framework and operation.**
Figure 4.10. CDF presenting the time required to process data through a dataflow for device use cases involving differing numbers of devices on the Laptop.
CHAPTER 5

CONCLUSION

This thesis has explored a newly developed software package called the Motion Based Gaming Peripheral Management Framework. The guiding principles behind the framework were described and later shown to have been adhered to through results obtained from performance tests of the framework. The performance tests described in this thesis aimed to demonstrate the isolation of the framework from data visualization and the performance of the framework under varying device use loads. Additionally, a variety of cases in which gaming peripherals have been used for non-gaming purposes were presented alongside a brief exploration of the Quantified Self movement.

The MBP-M Framework is a software package designed towards the goal of simplifying the development of programs centered around the use of low-cost motion-based gaming peripherals such as the Balance Board and Kinect. The framework implements device and data handling reliably and robustly, allowing developers to focus entirely on the primary purpose of their program. The MBP-M Framework can be found in a Notre Dame subversion repository[^1] and is intended for release as an open source project available for use by any interested parties or individuals. Included with the framework is a demo application with the framework at its core that acts as an example of framework integration and a template from which developers may create their own applications.

[^1]: http://netscale.cse.nd.edu/svn/WeHab/framework-mbpm
To conclude this thesis, I now cover several final questions with regards to the MBP-M Framework to be answered in the future...

- **Other devices:** The Wiimote, Balance Board, and Kinect cover a wide swath of gaming peripheral use in research and third-party application development, but other devices also exist and more are always in development. For example, as the second generation Kinect for Windows has recently been released and consists of a sizable number of hardware improvements, it would be wise to implement a Device Instrument representing this new Kinect. On the other hand, some devices with smaller user bases, such as the Leap [62], could also be implemented in the framework. These facts raise the question of what other devices should be included in the base version of the MBP-M Framework?

- **System performance and debugging:** With its substantial logging functionality, the MBP-M Framework could also log data regarding the state of the computer on which the framework is running. Logging system state could assist developers in understanding their programs’ degree of system resource use. Similarly, the Main Program could send error reports to the framework which would then log the errors with appropriate additional data (e.g., timestamps, system state, framework state, program state...). Should the MBP-M Framework log more than device data and performance results and thereby aid developers not only in data logging, but also debugging?

- **MBP-M Framework provided visualizations:** In addition to data logging and device handling, the framework could be further enhanced to provide visualization options. For example, the framework could include pre-built data visualizations such as graphs or two-dimensional Center of Pressure indicator planes that could be output to the Main Program. Users of the framework could then either use these framework generated visualizations or use the visualizations as templates for custom built visuals. However, generating visualizations would need to be performed in such a manner that the operation of the rest of the framework remains unaffected. Should the MBP-M Framework provide such visualization options?

- **Multiple concurrent Dataflows:** The MBP-M Framework currently only allows one Dataflow to be processed at a time. By allowing multiple Dataflows to run concurrently, however, the framework could potentially improve its efficiency such that Dataflows could be running constantly and the Oracle’s Dataflow generation timer could instead tick and report the most recently completed Dataflow to the Main Program. For such a strategy to work, steps would need to be taken to ensure that Dataflows complete in the proper order and that each Dataflow is using the correct set of data. Additionally, logging for multiple concurrently running Dataflows would be more complex and more difficult to accurately replay. Should the MBP-M Framework allow multiple Dataflows to run concurrently?
• *Log Batching*: The MBP-M Framework currently writes every frame of device data to disk immediately. Unfortunately, writing to disk is not an instantaneous operation and can take several milliseconds to complete. A potential solution to the time required to write to disk could be to batch device data writes such that multiple concurrent device data log entries are saved to a queue until the queue reaches a set length or data size, at which point the entire queue is written to disk. As write to disk time is a combination of seeking and writing times, experiments would need to be run to determine the optimal number of log entries to save to the queue such that the write time does not grow too great. Conversely, saving device data to a queue also carries some risk with regards to operation of the Main Program. For example, if the queue waits for 1000 log entries before a write to disk and the Main Program crashes, hundreds of device data entries could be lost. A balance must be achieved between performance and risk. Should the MBP-M Framework write device data to disk using a batching system?


42. “Glovepie,” July 2014.


