2014 IEEE 16th International Conference on e-Health Networking, Applications and Services (Healthcom)

A Management System for Motion-Based Gaming Peripherals for Physical Therapy Instrumentation

Benjamin Bockstege, Aaron D. Striegel, {*bbockste, striegel*}@nd.edu
University of Notre Dame

***Abstract*—The rise in motion-based gaming peripherals has afforded intriguing opportunities for low-cost instrumentation of health-oriented activities. One particular activity, that of physical therapy, is of considerable interest as traditional systems in the area cost on the order of tens of thousands of dollars. However, while recent research has shown that gaming peripherals can deliver high quality instrumentation, non-expert programmers face considerable challenges in delivering robust and accurate instrumentation outside of the lab environment. Furthermore, when one considers how to fuse data across multiple peripherals, the heterogeneity of peripheral performance significantly complicates recording useful data. To that end, this paper seeks to describe our approach for delivering a robust, accurate, and scalable framework for motion-based gaming peripherals, specifically targeted at physical therapy in the clinical and research settings. We describe the principles of our framework and composition of data flow through a variety of illustrative examples. Finally, we conclude with several experimental setups designed to demonstrate the efficacy of the framework drawn directly from our experience in live clinical settings.**

I. Introduction

Improvements in instrumentation have often been a significant driver in health care advances. That trend has only accelerated in recent years as clinical-grade instrumentation has been revisited to study the extent to which consumer-grade gear can complement (ex. Quantified Self [1]) if not wholly replace non-life-essential functionality. In particular, the field of physical therapy has benefited from the adaptation of consumer-grade electronics, namely the adaptation of motion-centric games and their associated gaming peripherals. While the costs of various gaming peripherals such as the Nintendo Wii Balance Board and Microsoft Kinect are quite appealing, it is the ability for researchers to interface and interact with the peripherals that truly drives research. With the catalyst of inexpensive gaming peripherals for sensing, researchers can derive customized systems for therapy and instrumentation and focus on the key task of utilizing the sensed information. The end result are systems that cost a fraction of existing clinical systems while taking advantage of gaming peripherals many therapists have already purchased for on-site use. Examples abound of researchers taking advantage of gaming peripherals for physical therapy including the Wii Balance Board in [2]–[4] and the Kinect in [5]–[8]. More recent research has even attempted to fuse both peripherals together in order to obtain richer datasets and feedback [9]–[12].

Unfortunately, as many researchers have discovered, the fusion of data from multiple gaming peripherals is decidedly non-trivial. For the case of simple fusion such as using two Wii Balance Boards (one for each foot [4]), variations between different devices results in fluctuations where individual device sample rates vary. That problem is further exacerbated when the devices themselves become heterogeneous such as with fusing information from different types, ex. Wii Balance Board (≈100 Hz) and Microsoft Kinect (≈30 Hz). Furthermore, consideration for multiple devices often involves *threading* which introduces its own unique blend of challenges. When coupled with the fact that effective therapy often requires smooth and accurate feedback [13], an area that was already difficult for non-computer scientists becomes nearly overwhelming. It is this need that serves as the basis for this paper, namely how to create a robust framework for fusing information across multiple motion capture gaming peripherals that is accurate and easy-to-use. By using this framework, researchers and developers may focus on the use of device data rather than the complexities of device and data handling. Our design is in part informed by our prior work with the WeHab [3] software suite and the complexities afforded in clinical settings when growing from a single Wii Balance Board to multiple balance boards and Microsoft Kinect as inputs. Our *Motion-Based Peripheral (MBP) Management* Framework aims to deliver such a platform and is built on four guiding principles:

- *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:* Reliability and robustness of the overall system is essential. 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.

With these four guiding principles in place, the challenge is how to deliver not only the desired characteristics of the framework but how to deliver such characteristics with minimal overhead. In this paper, we present the architectural design

978-1-4799-6644-8/14/$31.00 ©2014 IEEE

characteristics that allow us to deliver the robust and accurate MBP Management framework. In each section, we discuss the rationale for various design decisions as informed by clinical [3] and experimental experience [4] with the WeHab software suite of which the framework will replace the core instrumentation mechanisms. The MBP-M Framework itself is available via open-source through our project Wiki page[1].

The remainder of the paper is organized as follows. The MBP-M Framework and the flow of device data through the system are described. Next, the performance of the MBP-M Framework during several problematic scenarios is explored in order to demonstrate the robustness and reliability of the framework and its logging functionality. Finally, takeaways regarding the MBP-M Framework are presented and future plans explained.

II. MBP Management Framework

The MBP Management (MBP-M) framework is at its core, a dataflow management system. Using a variety of internal components responsible for data acquisition, data transforms, and data logging, the MBP-M Framework covers the entire process of data handling from initial acquisition to output (see Figure 1).

A. Framework

To begin, the most important component of the MBP-M framework is the object we dub the *Oracle*. The *Oracle* object can be thought of as a central clearinghouse for the system. The *Oracle* creates and maintains all *Device*, *Channel*, and *Output* objects along with all threads. Additionally, the *Oracle* maintains mappings of the most recently received data from each active device and is responsible for the initialization of data processing via *Dataflow* objects. The *Oracle* also acts as the only means of communication and interaction between the program inside which the framework has been integrated and the framework.

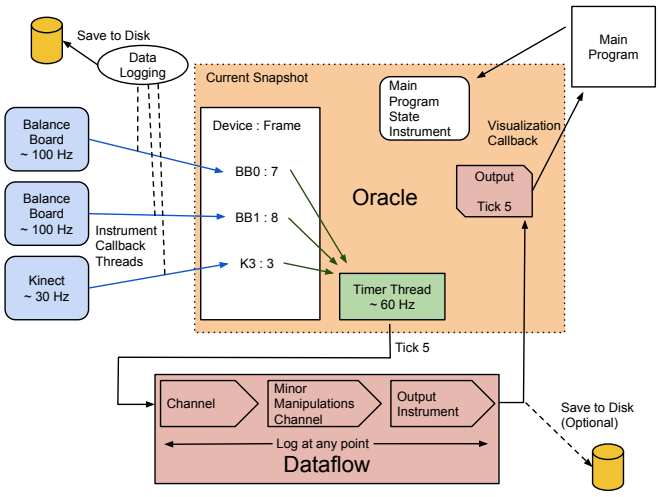

Fig. 1. A high-level overview of the MBP-M Framework.

[1]http://netscale.cse.nd.edu/MBP-M

Instrument objects in the MBP-M Framework are one of two types: *Devices* and *Outputs*. *Devices* are representations of peripheral devices in the system. All *Devices* inherit from a base *Instrument* class containing the core functionality common to the various types of *Devices*, including logging, debugging and event handlers. Specific *Device* classes represent such peripherals as the Kinect and Balance Board with their primary distinguishing features being how data from the peripheral is obtained and how log entries are structured.

For reporting data out of the MBP-M Framework, *Outputs* are used. As *Outputs* are also children of the *Instrument* base class, the core functionality of logging, debugging, and event handling are already in place. There exist two major types of *Output* representing the major approaches to data delivery: *Display* and *Data*. *Displays* are designed for the purpose of molding data into an easily visualized format while *Data Outputs* are used when data are only intended for consumption.

Channel objects perform transforms of data from *Devices*. A *Channel* receives data and performs some sort of manipulation(s) to transform the data into a different form. For example, one *Channel* may take in raw sensor values from two Balance Boards and fuse these values into a single two-dimensional Center of Pressure (COP) coordinate. Minor Manipulations Channels, on the other hand, perform minor, optional data transforms such as coordinate inversion and applying jitter. Like *Instruments*, *Channels* inherit from a base class containing the core functionality common to all *Channels*.

For logging, each *Instrument* and *Channel* maintains its own log file so that exactly what and when each object performed an action is recorded in an easily queried manner. While the specifics of log entries for each type of *Instrument* and *Channel* vary, logs follow a template of recording the time, the current frame number of the logging object, and data points specific to the type of *Instrument* or *Channel*. Additionally, while all *Instruments* and *Channels* create log entries for every received frame of data, only *Devices Instruments* write said data to disk on receipt. Due to the time required to write to disk and the need to maintain low overhead in the framework, the logs from *Channels* and *Outputs* are stored together as *Dataflow* log entries and written to disk as one file on closure of the main program. In this manner, data retrieval is both straightforward and consistent. Every incoming frame of *Device* data is logged on receipt and the specifics of how, when, and in what order data were processed and transformed are preserved. An end-user may go to the on-disk logs to determine exactly what data came into the system and how said data was transformed into the output he/she saw.

Figure 2 illustrates the steps the MBP-M Framework takes to transfer data from a peripheral device to the *Oracle* and the first steps taken when the data processing timer inside the *Oracle* 'ticks'. When a Balance Board reports sensor values, the *Device* object representing said Balance Board first creates a copy of the data and delivers the data to the *Oracle* via a callback then logs the sensor values. The *Oracle* responds to the callback by registering the delivered data as the most recently received data from the *Device*. Since each *Device* is an independent thread and *Device* report rates are heterogeneous, the *Oracle* can see inconsistently spaced and ordered updates.

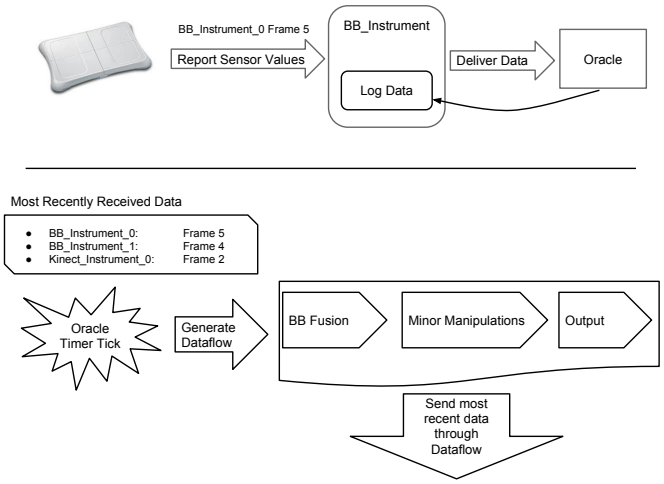

Fig. 2. An illustration of the events that take place when a device reports new data and what occurs on a 'tick' of the *Oracle's* data processing timer.

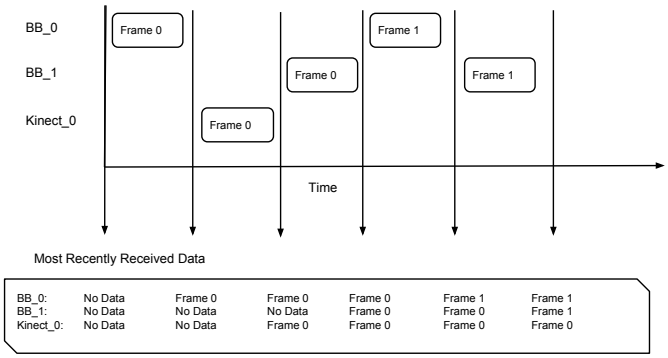

Fig. 3. A time-line example of a brief window of MBP-M System operation to illustrate how the System is updated as new frames of data from *Devices* are received by the *Oracle*.

For example, the Kinect reports at ≈30 Hz while the Balance Board reports at ≈100 Hz. The *Oracle* can receive several updates from a Balance Board in the time required for a single update from the Kinect.

Figure 3 presents a brief example during which two Balance Boards and a Kinect are reporting data. Figure 3 illustrates how the *Oracle* keeps track of *Device* data with each *Device's* frame report. When a *Device* reports a new frame of data, the *Oracle* registers this new frame of data as the most recently received frame from the sending *Device*. Note that as illustrated in Figure 3, the heterogeneity of device reporting makes it possible for multiple frames of data from both BB_0 and BB_1 to be received and logged in the time between two Kinect_0 frames.

To begin processing received data, the *Oracle* contains a timer running on a thread separate from the rest of the framework whose purpose is to initiate a series of steps that lead to the eventual output of collected data as illustrated in Figure 2. A timer is used to avoid difficulties caused by the cumulative frame rates of devices and to keep the system close to real-time. A Balance Board can report data to the framework at over 100 frames per second, and if two Balance Boards are in use the framework is receiving nearly 200 frames of data per second to be processed and visualized. If the framework started processing and visualizing data the instant said data came in from a *Device*, the framework would rapidly become swamped in a backlog of data as it simply cannot keep pace.

On the 'tick' of the timer, a *Dataflow* object is generated. As illustrated in Figures 1 and 2, a *Dataflow* contains any number of data transformations (*Channels*), an *Output*, and the most recently received data from each *Device*. Once a *Dataflow* object has been generated, it is handed off to another thread which proceeds to pass the attached data through each enqueued *Channel* and *Output* of the flow. Note that the ticking of the timer is consistent and can be tuned to an appropriate level by the programmer. For example, if the programmer knows that the display frame rate will be 30 Hz at best, the timer can be tuned to tick at a rate which will have data processed and delivered at approximately 30 Hz. On completion of data processing, the processed data is returned to the Oracle for final output.

III. Performance

In order to illustrate the benefits of using the MBP-M Framework, problematic scenarios and the potential overhead cost of the framework are explored. In the results below, the metrics used consist of display frame rates (Hz) and display render times (ms) for illustrating the problems caused by overloading the display of a frame of data, device reporting rates (Hz) to illustrate device report rate heterogeneity, and *Dataflow* processing and write times (ms) to quantify the overhead cost of the framework. The data were all collected on an HP Probook 4730s running Windows 7 64-bit with a 2.50GHz Intel Core i5-2450M CPU (two cores capable of two threads each) and 4GB Ram. For Bluetooth connection with the Balance Boards, the 32feet.NET library (version 3.4) [14] was used. The WiimoteLib library (version 1.7) [15] was used for communicating with the Balance Board. Additionally, an original Kinect sensor was used alongside the Kinect for Windows SDK (version 1.8) [16]. All sessions were run for sixty-five seconds and for analysis, the first fifteen and final five seconds of each test were ignored. In addition, all sessions were run using a WPF program with simple controls and a single visualization region on which data output was displayed. The MBP-M Framework was integrated into this program.

A. View Overloading

If software is coded without threading and appropriate attention to data provenance, an overload of data visualization can lead to data being ignored or miss-handled. For example, if a trace of previous coordinates is applied to a moving COP indicator visualized with each new frame of data from a Balance Board such that a set number of previous points are also visualized, the length of the trace will directly influence

TABLE I
INSTRUMENT FRAME RECEIPT RATES AND DISPLAY PERFORMANCE WHEN VARIOUS TRACE LENGTHS ARE APPLIED TO THE DISPLAY

	Average Instrument Frame Receipt Rate (Hz)	Average Display Frame Rate (Hz)	Average Display/Render Time (ms)
No Trace	101.76 ± 0.84	65.41 ± 1.77	0.00 ± 0.08
100 Points	101.86 ± 0.84	62.79 ± 3.68	7.94 ± 1.87
250 Points	101.67 ± 2.08	23.12 ± 2.08	29.06 ± 4.84
500 Points	101.95 ± 0.77	11.06 ± 1.61	76.54 ± 11.04

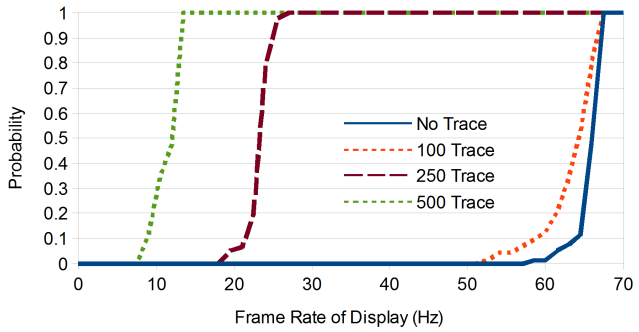

Fig. 4. Cumulative Distribution Function (CDF) of the frame rates of visualization on a single Balance Board setup.

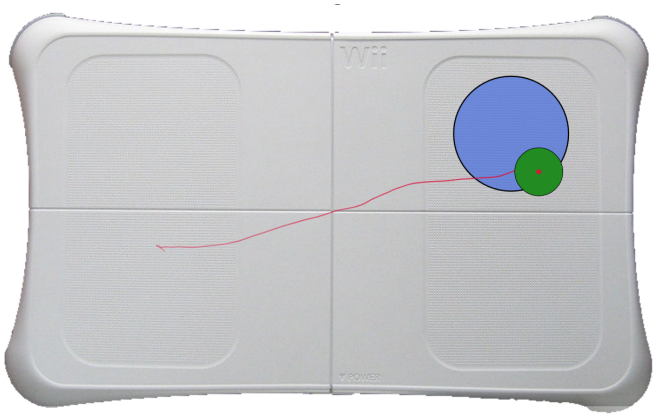

Fig. 5. A screenshot from the WeHab application presented in [3]. Shown is an example of a 'trace' being applied to a moving CoP indicator illustrating where the CoP was previously.

how much time is required for the display to complete its 'draw' operation. Such an increase in drawing time decreases the number of frames displayed per second as evidenced in Figure 4. If frames are being 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.

Figure 5 presents an example of a trace in the WeHab application described in [3]. Unlike the MBP-M Framework, WeHab was designed such that logging occurs at the same moment the data is displayed. As described above and discovered during development, drawing a long trace of data points can have a significant detrimental effect on the display frame rate. When logging is coupled with display and a substantial amount of time is required to completely display a single frame, multiple frames of data reported by a device can be ignored while the program works to visualize the previous

TABLE II
DISTRIBUTION OF CONSECUTIVE DEVICE UPDATES WITHOUT UPDATES FROM OTHER DEVICES

Interim Updates	BB / Kinect		2x BB / Kinect		
	BB	Kinect	BB 1	BB 2	Kinect
0	69.3%	0.0%	13.0%	13.9%	0.1%
1	30.7%	1.5%	51.5%	50.5%	0.3%
2	0.0%	17.2%	28.4%	27.5%	1.2%
3	0.0%	39.8%	6.6%	7.5%	1.4%
>= 4	0.0%	41.5%	0.5%	0.6%	97.0%

frame. While it seems at first that logging data at the same moment as data display is a poor design decision due to the potential of slowdowns, the ability to log the exact state of the system at the same moment the user sees the output is useful for understanding exactly what the user saw at a given time.

The MBP-M Framework avoids the slowdown caused by view overloading through its use of multi-threading. 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 some sort of slow down like an overburdened display. The success of this strategy is evident in Table I. The stable Balance Board report receipt rates compared against the decreasing display frame rate as longer traces are applied show that the rate at which frames of Balance Board data are received by the framework remains unaffected by slowdowns in visualization. If all data logging were coupled with visualization, then upwards of ninety frames could be ignored by the system every second when a 500 point trace is being visualized.

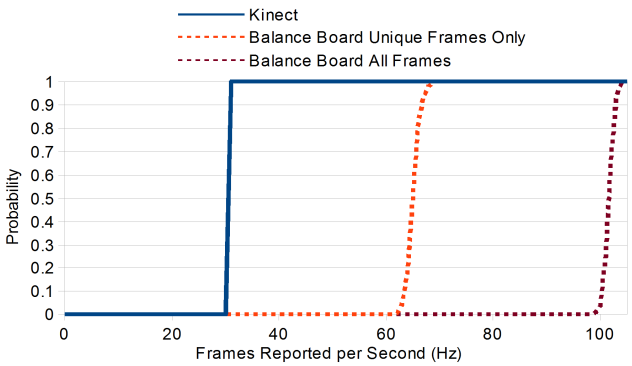

Fig. 6. CDF presenting the rates at which reports from the Kinect and Balance Board are received by the system.

B. Heterogeneous Instrumentation

An unfortunate fact about devices is that not all devices collect data at the same rate and the rate at which this data

is received is not always consistent. Figure 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 that although the Balance Board reports at ≈100 Hz, only about 64% of frames are actually unique (at least one reported sensor value is different from the previous frame). When only unique frames are sent to the Oracle, only ≈65 frames are reported per second. While Figure 6 does show report rates for the Kinect and Balance Board to be quite consistent, the consistency is not perfect.

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. If the Kinect reports at less than half the rate at which the Balance Board reports, how often should the *Channel* perform and pass along its transformations? The MBP-M Framework is designed such that *Channels* always use the most recently received data from each of the devices and perform transforms on a consistent interval (inside *Dataflows*). Therefore, a *Channel* receiving data from 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. This approach makes sense because, 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.

Table II presents situations in which a Channel listens to both a Balance Board and a Kinect in one case, and two Balance Boards and a Kinect in the other case. Table II shows how the total number of *Device* updates to the *Oracle* between updates for each specific *Device* were distributed. 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.

C. MBP-M Framework Overhead

While it is important that the MBP-M Framework provides 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. As long as *Dataflow* processing time remains low enough such that the display frame rate remains above ≈60 Hz (or 30 Hz when a Kinect is in use due to 30 Hz report rate), then the MBP-M System is performing acceptably as a display frame rate of 60 Hz appears fluid and understandable for most users [17]. In order for the display frame rate to remain above 60 Hz however, *Dataflow* processing time combined with the Dataflow generation timer interval of 15 milliseconds must take less than 17 ms (60 frames * 17 ms = 1020 ms). However, a frame rate of 30 Hz is also seen as acceptable [17], so a processing time of 22 ms could also be considered viable. As evidenced by the processing times in Figure 7, overhead caused by the framework during *Dataflow* processing remains largely below 3 ms no matter the number of devices actively reporting data to the framework. However, there does exist a small drop in performance as more devices are added. The increased Dataflow processing times seen with the addition of more devices can be attributed to the limited resources available through the four logical cores of the computer CPU and the simple fact that more devices in use means more threads are active, placing a greater burden on the CPU.

The MBP-M Framework currently writes only *Device* data to disk with every received frame as, unfortunately, writing to disk is not an instantaneous operation and can take several milliseconds as shown in Figure 8. For Figure 8, the framework was set to write to disk the log of every *Dataflow* object on completion of *Dataflow* processing (rather than on closure of the program). While the single *Device* examples in Figure 8 consistently have write times of less than ten milliseconds, adding a second (or third) *Device* has a detrimental effect on write time. With more *Devices* actively reporting to the framework, more writes to the disk must be made every second as each *Device* is attempting to write to disk every time it has a new frame to report. Since the *Devices* run on separate threads from the rest of the framework and from the *Dataflow* processing thread, multiple attempts to write to disk can occur concurrently. As the computer has limited resources and can only write to one location on the disk at a time, multiple requests to write to the disk in the same moment naturally require more time.

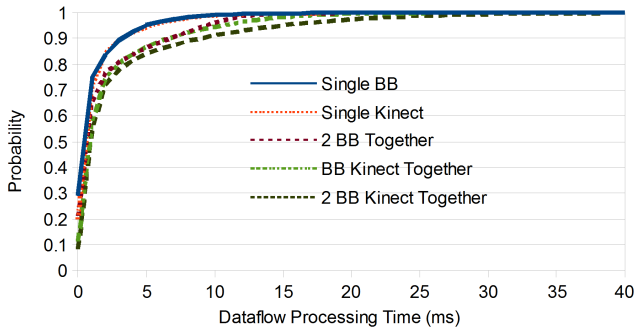

Fig. 7. CDF presenting the time required to process data through a Dataflow for several possible device use combinations.

A potential solution to the issue of writing to the disk for every frame of data that passes through a *Device* (or *Dataflow* if also writing *Dataflows* to disk immediately) is to write log entries to disk in batches. Like Figure 8, Figure 9 represents results gathered when the framework was set to write to disk the *Dataflow* log entries on completion of *Dataflow* processing. Figure 9 shows that increasing the interval between writes to the disk such that each write consists of the frames logged since the last write can improve *Dataflow* processing time while still recording all processed data. However, by increasing the interval between log writes, risk of lost or inaccurately

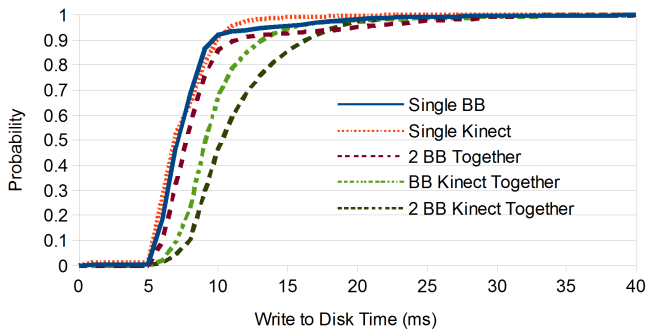

Fig. 8. CDF presenting the time required to write one frame of processed data to disk for several possible device use combinations.

recorded data increases. For example, if *Device* log entries are written to disk in batches of 1000 and the computer either crashes or loses power, hundreds of log entries would be lost representing potentially several seconds of device use. A balance must be achieved between improved framework performance and the risk of data loss.

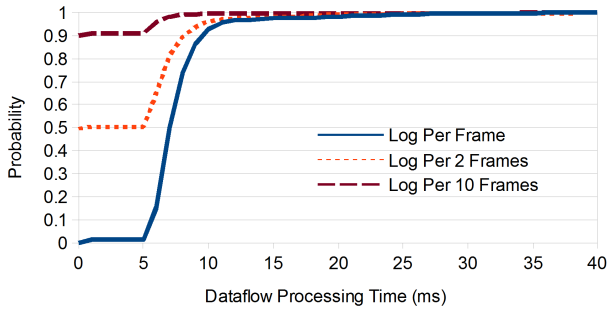

Fig. 9. CDF presenting the time required to process data through a *Dataflow* with three different log batching strategies when a single Balance Board was in use.

IV. Conclusion/Discussion

Low-cost motion-based gaming peripherals are a flexible and inexpensive approach to instrumentation of health-oriented activities. However, as described in this paper, there are complexities in the development of software that makes use of motion-based gaming peripherals that must be attended to lest data be lost or the developed software lack robustness. Our MBP-M Framework provides an easy to use and reliable platform upon which to develop software aimed at making use of motion-based gaming peripherals. The MBP-M Framework is designed to offer accurate data logging safe from any actions taken outside the framework and is robust against the various difficulties incurred when working with motion-based gaming peripherals, thereby allowing users of the framework to focus on the use of data rather than on the collection of data.

In the near future, a visualization suite with the MBP-M Framework at its core and a focus on consistent data logging will be constructed. This suite will exist as both a viable program for interaction with low-cost gaming peripherals and as an example/template of how the MBP-M Framework can be used.

V. Acknowledgments

We would like to thank for their feedback Notre Dame Professors James P. Schmiedeler, Michael A. Villano, and Charles R. Crowell along with Notre Dame graduate students Michael W. Kennedy and Mitchell D. Kajzer. This project was funded in part by grant # UL1 RR 02576 from the Indiana CTSI and from grant # IIS-1117706 from the National Science Foundation.

References

[1] M. Swan, "The quantified self: Fundamental disruption in big data science and biological discovery," *Big Data*, vol. 1, pp. 85–99, 2013. doi:10.1089/big.2012.0002.

[2] R. Brown, H. Sugarman, and A. Burstin, "Use of Nintendo Wii Fit for the treatment of balance problems in an elderly patient with stroke: A case report," *International Journal of Rehabilitation*, vol. 32, pp. 109–110, 2009. doi:10.1097/00004356-200908001-00144.

[3] M. W. Kennedy, J. P. Schmiedeler, A. D. Striegel, C. R. Crowell, M. Villano, and J. Kuitse, "Enhanced feedback in balance rehabilitaiton using the Nintendo Wii Balance Board," in *Proc. of IEEE HealthCom 2011*, pp. 162–168, 2011. doi:10.1109/HEALTH.2011.6026735.

[4] M. W. Kennedy, C. R. Crowell, A. D. Striegel, M. Villano, and J. P. Schmiedeler, "Relative efficacy of various strategies for visual feedback in standing balance activities," *Experimental Brain Research*, vol. 230, pp. 117–125, 2013. doi:10.1007/s00221-013-3634-x.

[5] B. Lange, C. Chang, B. Newman, A. S. Rizzo, and M. Bolas, "Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor," in *Proc. of IEEE EMBC 2011*, pp. 1831–1834, 2011.

[6] B. Lange, S. Koenig, E. McConnell, C. Chang, R. Juang, E. Suma, M. Bolas, and A. Rizzo, "Interactive game-based rehabilitation using the Microsoft Kinect," in *Proc. of IEEE VRW 2012*, pp. 171–172, 2012.

[7] B. Lange, S. Koenig, C. Change, E. McConnell, E. Suma, M. Bolas, and M. Rizzo, "Designing informed game-based rehabilitation tasks leveraging advances in virtual reality," *Disability and Rehabilitation*, vol. 34, pp. 1863–1870, 2012. doi:10.3109/09638288.2012.670029.

[8] Y.-J. Chang, C. S.-F., and H. J.-D., "A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities," *Research in Developmental Disabilities*, vol. 32, pp. 2566–2570, 2011. doi:10.1016/j.ridd.2011.7.002.

[9] A. Gonzalez, M. Hayashibe, and P. Fraisse, "Estimation of the center of mass with Kinect and Wii Balance Board," in *Proc. of IEEE/RSJ IROS 2012*, pp. 1023–1028, 2012.

[10] A. Gonzalez, M. Hayashibe, and P. Fraisse, "Three dimensional visualization of the statically equivalent serial chain from Kinect recording," in *Proc. of IEEE EMBC 2012*, pp. 4843–4846, 2012.

[11] A. Gonzalez, M. Hayashibe, and P. Fraisse, "Subject-specific center of mass estimation for in-home rehabilitation - Kinect-Wii board vs. Vicon-Force plate," *Converging Clinical and Engineering Research on Neurorehabilitation*, vol. 1, pp. 705–709, 2013.

[12] A. Dutta and A. Banerjee, "Low-cost visual postural feedback with Wii Balance Board and Microsoft Kinect - a feasibility study," in *Proc. of IEEE PHT 2013*, pp. 291–294, 2013.

[13] M. C. Dault, M. de Haart, A. C. H. Geurts, I. M. Arts, and B. Nienhuis, "Effects of visual center of pressure feedback on postural control in young and elderly healthy adults and in stroke patients," *Human Movement Science*, vol. 22, pp. 221–236, 2003. doi:10.1016/S0167-9457(03)00034-4.

[14] "32feet.net," May 2014. 32feet.codeplex.com.

[15] B. Peek, "Managed library for Nintendo's Wiimote," April 2014. wiimotelib.codeplex.com.

[16] "Kinect for Windows," May 2014. http://www.microsoft.com/en-us/kinectforwindows/.

[17] M. Claypool, K. Claypool, and F. Damaa, "The effects of frame rate and resolution on users playing first person shooter games," in *Proc. of SPIE 6071, Multimedia Computing and Networking 2006*, January 2006. doi:10.1117/12.648609.