

MATLAB-Based Real-Time Data Acquisition Tool for Multimodal Biofeedback and Arduino-Based Instruments: Arduino Firmata Data Acquisition (AfDaq)

Kulbhushan Chand, Dr. B. R. Ambedkar National Institute of Technology, India*

 <https://orcid.org/0000-0001-6502-0748>

Arun Khosla, Dr. B. R. Ambedkar National Institute of Technology, India

 <https://orcid.org/0000-0001-8571-7614>

ABSTRACT

AfDaq is an open-source, plug-and-play, MATLAB-based tool that offers the capabilities of multi-channel real-time data acquisition, visualization, manipulation, and local saving of data for offline analysis. The MATLAB Arduino package suffers from serious timing jitter during real-time data acquisition. This timing jitter associated with four main commands (analog read, digital read, digital write, and PWM set) available in MATLAB Arduino package is statistically analyzed, and a simple post-hoc timing jitter correction mechanism is proposed to acquire data points with high timing accuracy. The benchmark of the final program is conducted at various sampling rates for multichannel acquisition with 10 Hz comes as the maximum sampling rate for 5 channel recording. In the end, a use case of the developed tool for physiological data acquisition in multimodal biofeedback is presented. The software tool, data, and analysis scripts that support the findings of this study are released as an open-source project to support the replicability and reproducibility of the research.

KEYWORDS

Biofeedback, Firmata, Graphical User Interface, Open-Source Software, Optimum Sampling Rate, Physiological Signal Acquisition, Plug and Play System, Timing Jitter, Timing Jitter Correction Mechanism

1. INTRODUCTION

Researchers from different fields frequently require data acquisition devices to record various signals relevant to their field of interest. In some cases, scientific research requires the development of experimental setups and novel methods to capture unknown phenomenon and signals of interest while others rely on expensive proprietary data acquisition devices. However, there is a significant proportion of researchers including students of universities, school-going students, and hobbyists, for their preliminary data acquisition needs, requires an inexpensive and quick to deploy which can be realized with limited engineering capabilities, falls under their resource constraints, and easy to

DOI: 10.4018/JITR.299922

*Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

use. Particularly in the biofeedback domain, there is a need for affordable and quick to deploy data acquisition tool for quick prototyping and out-of-lab experiments (Fortin-Côté et al., 2019).

In recent years, in pursuit of affordable and flexible data acquisition systems, researchers have shifted their focus to using open-source electronics platforms like Arduino (D'Ausilio, 2012; Pearce, 2012; Polo et al., 2018). An Arduino is a low-cost open-source microcontroller board used for quick prototyping for electronics projects. The dedicated integrated development environment of Arduino is available for different operating systems like Windows, Mac OS, and Linux, and simplifies the process of editing, compiling, and uploading of software code into the microcontroller. The huge online community and support forums are an additional advantage of the Arduino ecosystem. The low cost associated with the Arduino ecosystem makes it stand apart from other hardware alternatives for data acquisition like myRIO (National Instruments, Austin TX, USA), LabJack (LabJack Corporation, Colorado, USA), and others. The cost of the cheapest alternatives is \$130 for the Labjack device whereas Arduino comes for \$10.

For the software part of data acquisition, software like MATLAB (Mathworks, Natick MA, USA) have been extensively used for its quick prototyping abilities and ease of use (Toulson, 2008). MATLAB is a widely used software in universities around the world. It offers very powerful data processing capabilities with an exhaustive library of functions spanning various fields. It also offers powerful and customized GUI tools for experiment building. Most of the researchers are already familiar with or have access to MATLAB in their university. A tool built in this environment will have data acquisition, visualization, and processing capabilities all in one place which significantly reduces the prototype building time. However, for an individual with no access to MATLAB, the high cost of MATLAB can be the biggest setback in using this tool.

Arduino family of boards can be easily integrated with MATLAB using official support packages to create a plug and play system (*MATLAB Support Package for Arduino Hardware Documentation - MathWorks India*, n.d.). However, this flexibility of prototyping comes at a cost of random timing jitters in communication between Arduino and MATLAB which causes latency issues in real-time data acquisition. In this paper, the authors present AfDaq¹, a MATLAB based GUI tool for real-time data acquisition from Arduino. The GUI is developed using the GUIDE tool available in MATLAB. For interacting with the Arduino, a set of commands is used which are provided by “MATLAB Support Package for Arduino Hardware”² (*Arduino Support from MATLAB - Hardware Support - MATLAB & Simulink*, n.d.). The timing jitter associated with various commands and sub-processes in the system was measured and descriptive statistics were computed. MAE (Mean Absolute Error) and the recorded sampling rate are used in the benchmark to define the selection criteria for system performance. A maximum of 5 channels (Analog or Digital) can be recorded simultaneously. The acquired data is displayed in real-time over a customizable scrolling plot at a maximum sampling rate of 10 Hz when acquiring from all 5 channels simultaneously. The tool also offers real-time data manipulation. The acquired data can be locally saved for offline processing. This is a plug and play system and the experimenter can be quickly up and running with the data acquisition process. The developed tool acquired data and analysis scripts used to compute results are released in the open-source domain for the researchers and to strengthen the replicability of the project.

The paper is organized as follows. In related work, the literature and the research gaps focussing the use of Arduino and MATLAB in the data acquisition system are discussed. Next, an overview of the developed tool is discussed. Preceding sections elaborate on the design and development methodology of AfDaq starting with the design of GUI, communication protocol, various design optimizations, timing jitter analysis, the proposed method of its correction, and program workflow. Next, the benchmark results of AfDaq are discussed. In the end, a use case of AfDaq for multimodal physiological data acquisition from a subject is discussed where during data acquisition, a stimulus is given to the subject and the corresponding effects on the physiological signals are noted. Finally, the authors conclude the findings, and a link to the repository is provided where the code, data, and analysis scripts are made available with an open-source license.

2. RELATED WORK

In literature, there is the wide-spread use of Arduino and MATLAB as data acquisition tools across various domains. There are a plethora of data acquisition projects completed using Arduino in the research, education, and hobby domain. Some of the notable use in research is in chemical instrumentation (Grinias et al., 2016), indoor environment quality (IEQ) monitoring (Ali et al., 2016), brain-computer interface (BCI) systems (King et al., 2014), seismic noise measurement (Saraò et al., 2016; J.L. Soler-Llorens et al., 2016; Juan Luis Soler-Llorens et al., 2019), biomedical signal acquisition (Polo et al., 2018), biofeedback devices (Kim et al., 2011), technologies for smart cities (Costa & Duran-Faundez, 2018; Mumtaz et al., 2018). In education, it is used in university laboratories for teaching students about electronics, practical experience with instrumentation, in-class projects (Mabbott, 2014; Urban, 2014), and laboratory experiments (Cao et al., 2015; Famularo et al., 2016; McClain, 2014). The hobbyist has also developed interesting projects with Arduino like musical instruments (Wu & Bryan-Kinns, 2019), robots (Salman et al., 2020), and drones (*ArduPilot*, 2013/2020).

There are also many data acquisition software applications developed using different platforms to acquire data from Arduino. Instrumentino (Koenka et al., 2014) and RTGraph (Sepúlveda et al., 2015) are python-based applications. ArduinoScope (Ishikawa & Maruta, 2010) is a processing based tool. Telemetry Viewer (Farahbod, 2016/2020) is a java-based real-time acquisition tool. There are few tools developed by researchers for their specific needs in the MATLAB environment (Bhoyar & Dr.S.S.Sonavane, 2015; Nichols, 2017; Juan Luis Soler-Llorens et al., 2019) and in the LabVIEW environment (Ferreira et al., 2015). Some experimenters have also developed non-Graphical User Interface (GUI) standalone data acquisition devices for Arduino (Ali et al., 2016; Gad & Gad, 2015).

The available acquisition tools in the literature require an additional step of uploading some code on Arduino to prepare it for communication with acquisition software running on PC. This creates an additional dependency for the system and extra steps for researchers, who are not familiar with the process of programming an Arduino board, to use the acquisition system. The “MATLAB Support Package for Arduino Hardware” mitigate this issue by handling the task of preparing the Arduino board with firmata code, so that it can receive the commands from MATLAB and become a complete plug and play system. The MATLAB software then acts as a master and takes care of all the acquisition processes including precise time stamps at which data is polled from Arduino. However, the communication in this setup is prone to severe timing jitters due to MATLAB being a high-level language, which further runs on a non-real-time operating system (like Windows). The downside of using firmata code is the increased latency as compared to custom code running on Arduino. This reduces the maximum sampling rate at which data can be acquired from Arduino. It is to be noted that in some of the biofeedback applications, a relatively low sampling rate of less than 50 Hz is sufficient during data acquisition (Alhamid et al., 2012; Frey et al., 2018; Geršak & Drnovšek, 2020; Luo et al., 2017; Moeyersons et al., 2016; Moraveji et al., 2011). Thus, even with the relatively low sampling rate, a useful plug and play real-time data acquisition system can be developed with Arduino and MATLAB.

During the literature survey, the authors didn't find any work where researchers have optimized the data acquisition process using “MATLAB Support Package for Arduino Hardware”.

3. OVERVIEW OF THE DEVELOPED TOOL

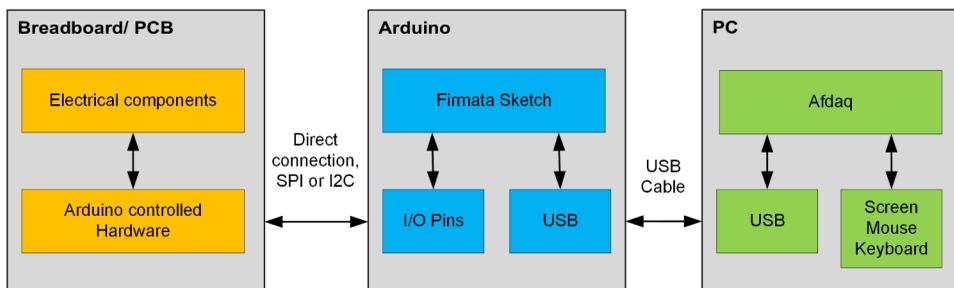
The basic overview of the setup for data acquisition using AfDaq is shown in Figure 1. The entire setup can be separated into three sections. On the left side is the breadboard circuitry which consists of various electrical components for interfacing sensors to Arduino I/O pins. This section deals with the conversion of electrical signals, voltage or electrical power level conversion, signal conditioning, and means to interface sensors with the input requirements of Arduino. Most of the time the electronic

circuitry is made on a breadboard for quick prototyping and later transferred to a custom-made PCB for final production.

The middle section is the Arduino board which acts as an extension to allow easy interfacing with the on-board ATMEGA328P microcontroller. It provides an onboard serial to USB interfacing chip for easy communication with USB devices, 5V and 3.3V voltage regulator, crystal oscillator, and sockets for ease in connection with analog and digital pins and external devices. The Arduino, on one side interfaces with Breadboard/PCB via direct connections to I/O pins, SPI (Serial Peripheral Interface), or I2C (Inter-Integrated Circuit) and another side with PC running AfDaq software via USB Cable. A special code called firmata runs on Arduino which provides functionality to receive standard commands from the serial port. By these commands, Arduino can be instructed to do a variety of tasks, like controlling I/O pins, using the I2C bus, data transfer on serial communication, etc. The power supply for Arduino and breadboard PCB is usually independent and not derived from the USB port of the PC. This is a preferred way of powering when Arduino and circuitry demand high power which PC may not be able to deliver reliably.

The right section is the PC running AfDaq which is the MATLAB GUI tool. Using this users can interact with and easily acquire real-time data directly from Arduino within the MATLAB environment. The AfDaq and Arduino work in a master-slave configuration, with AfDaq being the master. The application works as a real-time data visualizer and data logger. It keeps track of precise time intervals, acquire data from Arduino over the serial port, plot real-time data, performs basic data manipulation, append time-stamped data in arrays and continuously keeps track of timing jitter and minimizes it. As a data logger, it allows for the saving of data for offline analysis. The AfDaq uses the MATLAB-Arduino package, which provides a set of functions to send and receive commands/data from the Arduino.

Figure 1. A basic overview of the setup for data acquisition using AfDaq



4. DESIGN AND DEVELOPMENT

The GUI has been designed using MATLAB v2017b and also tested with v2015b and v2018b versions for backward and forward compatibility under the Windows operating system. The software is developed by keeping some design considerations in place. The main considerations are a simple plug-and-play system to reduce the system setup time; real-time acquisition with maximum sampling frequency possible; simple and easy to operate user interface; modular design to allow the further modifications to the software (if required); and provision of data logging and analysis to utilize the full data processing capabilities of the MATLAB. Some of these design considerations are dependent on each other. For example, to achieve the plug-and-play requirement, the MATLAB Arduino package is used which also adds up the timing jitter associated with its usage. Another dependency is in achieving a higher sampling rate which is limited by the number of simultaneous channels acquired

and the number of simultaneous lines plotted, with the increased lag being the confounding factor. While designing the program, the various trade-offs are decided between design-considerations based on their inter-dependencies.

4.1. Graphical User Interface

Figure 2 shows the screenshot of AfDaq. The GUI is segmented and named into various panels as per their functionality. For better visual aesthetics, these panels are arranged into four main sections namely – settings, information, plot, and control. The user controls the state of the GUI from the control panel. Different panels are made active according to the current state of the GUI. The settings section marked with number 1, deals with various settings. Starting from the top is the “Arduino” panel, which allows the user to select the COM port and establish a connection with Arduino. Below is the “Settings” panel which allows the user to customize various settings like session duration, sampling frequency, and channel type. The user can select a maximum of 5 channels. Each of these channels can be analog or digital and can be selected from the drop-down list. The analog channels are the first 5 channels of the Arduino ranging from A0 to A4 or PINs 23 to 27. The Arduino uses 10-bit ADC which allows for a maximum of 1023 levels of digitized analog values. The digital channels are the digital pins D2 to D6 or PINs 4, 5, 6, 11, and 12. The digital value is represented as either 1 or 0. The real-time data from channel1 can be scaled as per the custom scaling function. The “Display” panel is used to manually or automatically set the vertical axis of the displayed plot.

Figure 2. GUI for the data acquisition system



Sections - 1 settings, 2 information, 3 plot, 4 control

The information section marked with number 2, is the bottom part of the GUI. It includes the “Status Message” text for feedback to the user about the current state of the GUI. The “Daq information” panel displays the date, time, and other important parameters. The “Results panel”

displays the parameters computed from the acquired data and is updated after the completion of the acquisition.

The plot section marked with number 3, consists of a “Live Plot” panel which covers the major part of the GUI. The plot is updated in real-time with each sample received. For the distinguishable representation of the digital values (either 1 or 0), they are multiplied by 150 and each digital channel is staggered by even spacing to separate them from each other. Thus, the digital values cover the entire vertical axis of the plot but the analog values (range = 0 to 1023) are plotted as such.

At the bottom right, the control section is marked with the number 4. It includes the “Control” panel by which users can start/stop the acquisition, log the acquired data, and reset the state of the GUI. The reset button initializes the GUI to its first state after opening. The acquired data can be logged in to user-defined directory as .xlsx (Excel) or .csv (comma separated values) format. For viewing and offline analysis, the saved file can be opened in spreadsheet programs such as Calc (LibreOffice), Excel (Microsoft Office), or MATLAB.

The snapshot in Figure 2 shows the initialization of GUI. The program displays the real-time plot of the acquired signals. After the successful acquisition, the user can interact with the plot via various figure controls provided by MATLAB. The final number of samples acquired and the quality of data is displayed in the results panel. The data quality is the proportion of “successfully acquired samples” out of “estimated samples to be acquired” in the actual session duration. The data quality gives an idea about the number of dropped samples occurring because of the lag in data acquisition. If the user is satisfied with the acquisition the data can be saved locally for offline storage and analysis.

4.2. Communication Protocol

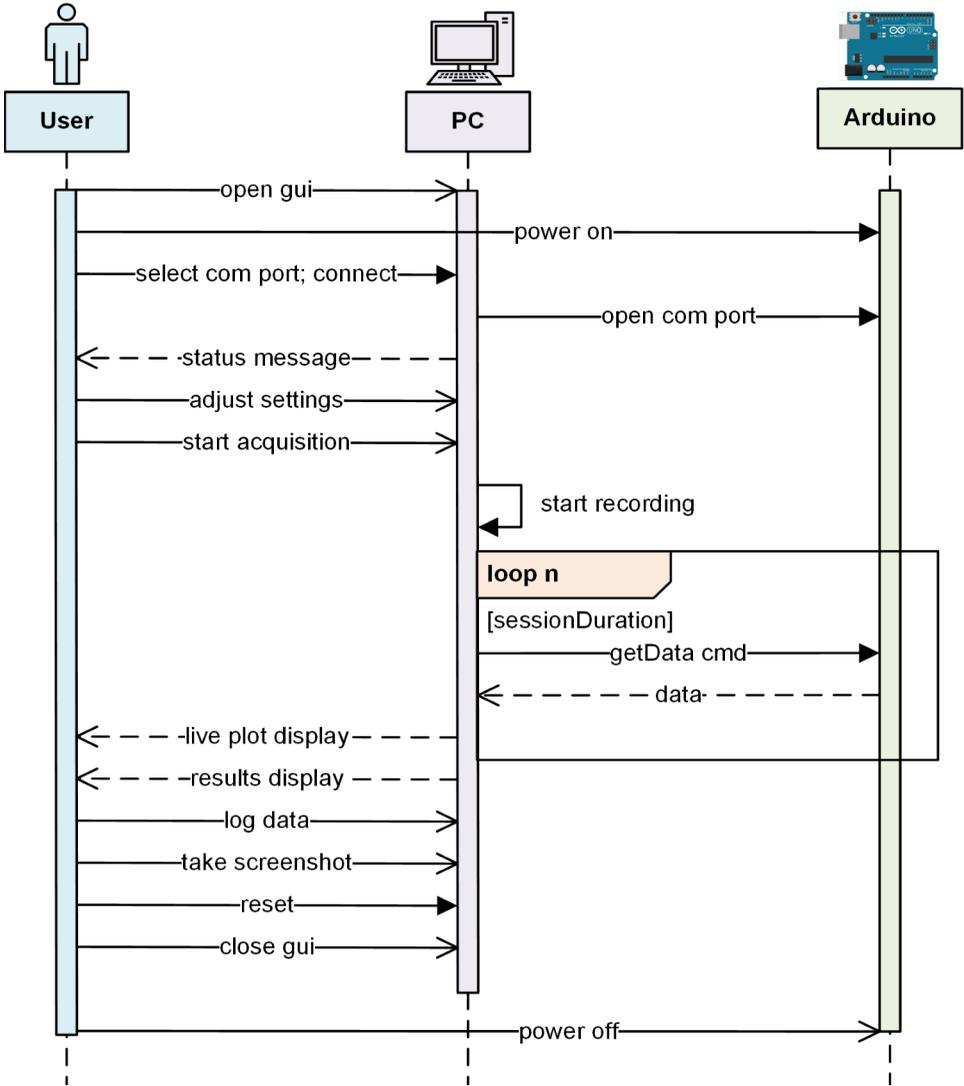
Arduino hardware can be connected to MATLAB in numerous ways. MATLAB has provided a “MATLAB support package for Arduino Hardware”, which contains a set of functions to directly control Arduino. The package provides various functions for acquiring analog and digital sensor data from the Arduino board, control other devices using digital and PWM outputs, access devices and sensors connected over SPI or I2C interface with the Arduino board, and many others. A wide range of boards from the Arduino family and Arduino-compatible devices are supported by this package. Since MATLAB is a high-level interpreting language, the results of the operation are available without compilation. Table 1 lists some of the functions/commands available in MATLAB which can be used to get data or control the Arduino board. Only the name of the command and a brief description is shown. For proper syntax and detailed description with example code, readers are advised to refer to the comprehensive documentation which is made available online by MathWorks (*MATLAB Support Package for Arduino Hardware Documentation - MathWorks India*, n.d.). The communication between AfDaq and Arduino occurs over the firmata protocol. Firmata is a communication protocol based on the midi message format and is used for communication between microcontrollers and software running on a computer/smartphone (*Firmata Firmware for Arduino*, 2012/2020). When the Arduino object is instantiated, the MATLAB support package automatically prepares the Arduino board to receive communication by uploading the firmata code on it.

The program workflow is designed to be simple and straightforward. Figure 3 shows the sequence diagram detailing the interaction between the user, AfDaq, and Arduino. The only interaction with Arduino is to connect it with the PC and the rest of the steps are performed via AfDaq. The initial step is to open the AfDaq tool in MATLAB and then connect the Arduino with the PC using a USB cable. The user can manually reset the GUI to see the list of active COM ports and make a connection with the Arduino. After a successful connection, the settings panel will become active. The user can modify various settings as per the requirements. A custom ID string can be set to distinguish the different sessions. The next step is to start the acquisition process, which runs for the set defined time or can also be stopped manually. A live signal-plot is displayed during acquisition. The results are computed after the acquisition process finishes. In the end, the user can log the data or take a snapshot of the GUI, before closing the tool.

Table 1. Some of the functions/commands available in the MATLAB support package to interact with the Arduino board

Function	Description
arduino	Connect Arduino by specifying the port and board name
arduinsetup	Launch Arduino hardware setup interface
configurePin	Set the pin mode (input/output) of the pin
readDigitalPin	Read data (0/1) from digital pin
writeDigitalPin	Write data (0/1) to digital pin
writePWMVoltage	Generate PWM signal of specified voltage on digital pin
writePWMDutyCycle	Generate a PWM signal of specified duty cycle on digital pin
readVoltage	Read voltage (0 to 5V) from the analog pin

Figure 3. Sequence diagram showing the interaction between the user, PC, and Arduino



4.3. Various Design Optimizations

AfDaq is developed in MATLAB software running on the Windows Operating system. The main goal of the design optimization is to reduce the timing jitter from as many sources as possible. To achieve this goal, various optimizations have to be made in MATLAB code and Windows environment during designing and testing the application.

There are two main bottlenecks in using MATLAB for this work. Firstly, the interpreted code execution in MATLAB always takes a longer time than the compiled code. Secondly, the MATLAB is designed for offline-data processing and is not optimized for designing real-time constrained applications. However, MATLAB natively provides a Just-In-Time (JIT) compilation feature (*MATLAB Execution Engine*, n.d.), which optimizes functions on-the-fly when code is executed additional times, but its performance is limited and case-specific (Chevalier-Boisvert et al., 2010). To make use of JIT acceleration, readings are always taken on the second run of the program after a cold start³. Further optimizations used are, making pre-allocation of arrays, using *clearvars* instead of *clear all*, and vectorizing the code wherever possible. For time measurements, *tic/toc* commands are used with sufficiently large readings (100 readings) to reduce the random effects of various background processes in Windows.

Windows itself is not a real-time operating system. At any given moment, many applications, background processes, and services are using system resources. The throttling of processor, virus scanner, disk access, memory, and cache occupied by other applications are a few of the things that can affect the performance of AfDaq in MATLAB. To keep check of these activities to a minimum, some of the important steps taken are - pausing Windows update, disabling non-essential services from third-party applications, and defragmenting the hard disk. During acquisition, the internet is turned off and the disk activity of background processes is monitored via Window Task Manager. Readings are discarded if there is significant background disk activity⁴.

The tool is intended to be used by a wider audience, without any specifications of high-end computer system requirements. Therefore, a mid-range laptop for benchmarking purposes is used with specifications of i3-3120M processor, 4GB DDR3 RAM, AMD Radeon HD 7500M graphics card running Windows 10 pro (64 bit) and MATLAB v2017.

4.4. Timing Jitter Analysis

Tight control over the sampling frequency is a must for reliable data acquisition, which means the samples must be acquired after an exact and same time interval. The variability in the time taken to acquire samples during acquisition is called timing jitter. When the sampling frequency is low enough, the system has adequate time to accommodate for any timing jitter and reliably acquire data samples at the set sampling frequency. However, when the sampling frequency is relatively increased, then the system may not be able to perform all the tasks and leave enough buffer time to accommodate timing jitter.

The AfDaq runs on Windows which is not a real-time operating system. There are many background processes and programs that can steal away CPU time and other system resources and thus effects the acquisition process. The resource demands from these background processes are highly variable and dependent upon system configuration and installed software and thus cannot be known a priori. Figure 4(a)-(b) shows the time taken by four different commands and their probability density plot. The density plot comes out to be highly skewed right-tailed. The median is an appropriate descriptive measure than the mean for highly skewed plots and is marked with a red line in each of the plots. The value of the median ranges from 12.54 ms to 14.98 ms for the command shown.

The time variability adds up when commands are called multiple times per sample. This is represented by the probability density plot in Figure 4(c), where the Analog Read command is called 1 to 5 times per sample, and the corresponding time taken is calculated. Since the density plots are highly skewed, the median is represented and labeled for each case. Figure 4(d) shows the ridgeline probability density plots of time taken for calculations (appending data into arrays, if-else conditions,

and real-time data analysis part) and plotting of multiple lines in AfDaq. The median (50% quantile line) is shown on each plot which divides the density plot into two halves in terms of area. The time taken for calculations is 0.7 ± 0.18 ms, which shows very little variability. However, the timing jitter of the multiple lines plot is interesting. The time taken for plotting a single line is 0.01 ms with a standard deviation of less than 0.05 ms. Any value less than zero in Figure 4(d) is a representational error. As the number of lines increases the density plot gradually shifts to a median value of 6 ms. This is because of the “drawnow limitrate” function used in plotting, which is an optimization feature within MATLAB for efficient plotting.

Figure 4. (a) Timing jitter in MATLAB Arduino package for various commands computed for 100 samples, (b) Probability density plot of the timing jitter. The median value is represented by a red line in each plot, (c) Probability density plot of multiple calls to Analog Read command per sample, (d) Ridgeline probability density plot showing timing jitter associated with calculations and multiple line plots

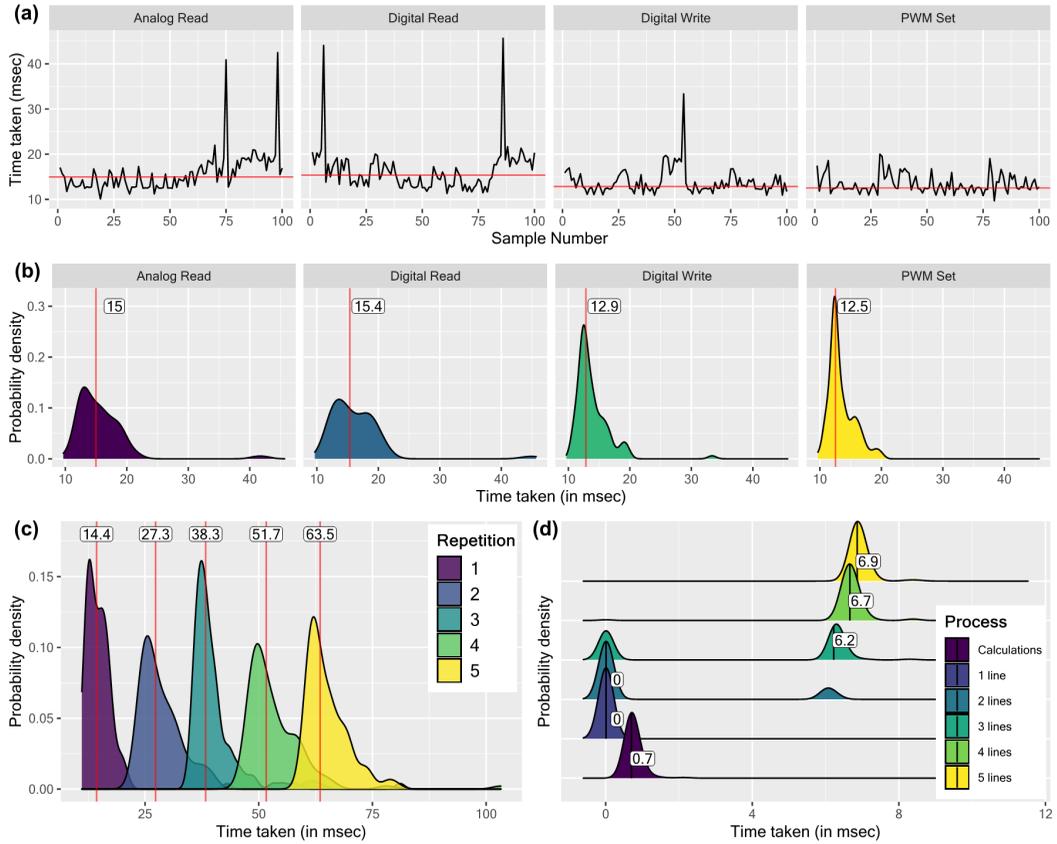


Table 2. Descriptive statistics of timing jitter of various commands and sub-processes

		Descriptive Statistics ^a				
		MEAN	MEDIAN	STD	MIN	MAX
Commands ^b	AR	15.73	14.98	4.56	10.09	42.5
	DR	16.32	15.37	4.96	11.31	45.67
	DW	13.89	12.85	2.89	10.91	33.37
	PS	13.46	12.54	2.08	9.67	20.02
Multi-channel ^c	I	14.92	14.35	3.65	11.09	43.03
	II	29.41	27.31	7.09	22.96	63.25
	III	39.96	38.33	5.85	35.22	81.11
	IV	53.6	51.65	7.18	47.47	103.24
	V	65.07	63.51	4.39	59.9	80.28
	Calculations ^d	0.76	0.7	0.18	0.69	2.13
Multiple Plots	I ^e	0.01	0.01	0	0.01	0.02
	II	1.11	0.01	2.35	0.01	6.35
	III	3.79	6.22	3.19	0.01	9.47
	IV	6.69	6.66	0.84	0.01	10.9
	V	6.92	6.86	0.25	6.67	8.5
	Work ^f	26.78	25.06	8.48	12.28	41.36
	Pause ^f	73.21	74.94	8.49	58.65	87.7

^aAll descriptive statistics are in ms and calculated for 100 samples each. ^bCommands used are AR = Analog Read, DR = Digital Read, DW = Digital Write, PS = PWM Set. (Fs = 1 Hz) ^cMulti-channel is the number of simultaneous calls of Analog Read command. (Fs = 1 Hz) ^dCalculations involve work excluding commands and plotting. (Fs = 1 Hz) ^eValues less than 0.01 are rounded off to 0 ^fWork includes all the active processes, and Pause is an idle wait period. (Fs = 10 Hz)

This timing jitter poses a serious concern in data acquisition in two ways. Firstly, the timing information associated with data samples is not precise. Secondly, this timing jitter adds up and creates timing drift over a while during the data acquisition process. This rate of drift is variable because of variability in the effects of background processes. Therefore, the timing drift is difficult to predict and compensate for before data acquisition. To ascertain the amount of variability in the timing of sub-processes, the mean, median, standard deviation, min, and max values are computed for 100 samples of data, as shown in Table 2. The timing jitter associated with commands and their multiple runs is found to be highly skewed and right-tailed as evident from the median and min-max values. The median is used as a comparison parameter since comparing means can be misleading for skewed distributions in the presence of outliers. These distributions are shown in Figure 4(b)-(d). The large values of standard deviation are noted in the use of commands and their multiple runs, whereas the calculations and plotting processes are relatively less variable. The columns Work and Pause are computed for single-channel acquisition using the Analog Read command at a sampling frequency of 10 Hz.

For the sampling interval of 100 ms, the median time taken by all the work (acquisition, calculations, and plotting) is 25.06 ± 8.48 ms, and idle time (time taken to wait out for sampling interval) is 74.94 ± 8.49 ms. So, the work is done for nearly a quarter of the sampling interval at 10 Hz, and in this case, there is enough time to take into account the worst-case scenario of timing jitter. However, this median time of work is increased when the acquisition is multichannel (multiple commands) with multiple real-time plots.

4.5. Timing Jitter Correction

In this work, timing jitter cannot be known before the data acquisition, and post-adjustment will result in a variable sampling rate. The best option is to correct for timing jitter in real-time during data acquisition. A simple method for time jitter correction during real-time acquisition is developed. The method is based on post-hoc time jitter adjustment, in which the time jitter that occurs in the previous sample is used to adjust the sampling time interval in the present sample. This can be easily understood in the visual representation of the method as shown in Figure 5 and the flowchart of the algorithm in Figure 6. The visual representation shows acquisition at a sampling interval of T ms, shown by dotted lines. The label t_{ic} shows the start of acquisition time and the label t_s shows the subsequent time stamps. The W is the work period which includes all the acquisition, data framing, and plotting related work for a single sample. The P is the pause period, which is used to wait out for the remaining time until the sampling interval completes. The number associated with P and W represents a consecutive sample number. Timing jitter is represented by t_j which is equal to the excess difference of timestamp and the estimated sample time based on sampling interval. In other words, it is the overshoot of measured timestamps from the ideal timestamp. The final objective of the method is to minimize the t_j by using its measured value during the previous loop, to adjust the pause period in the current loop.

Figure 5. The timing representation of post-hoc time jitter correction mechanism for the first two samples

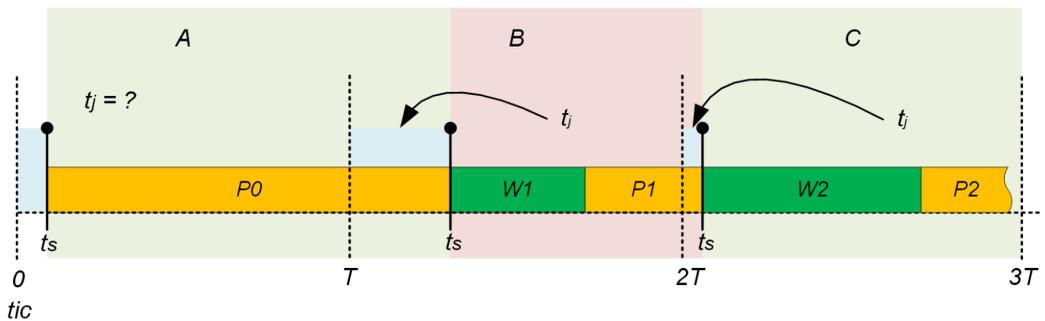
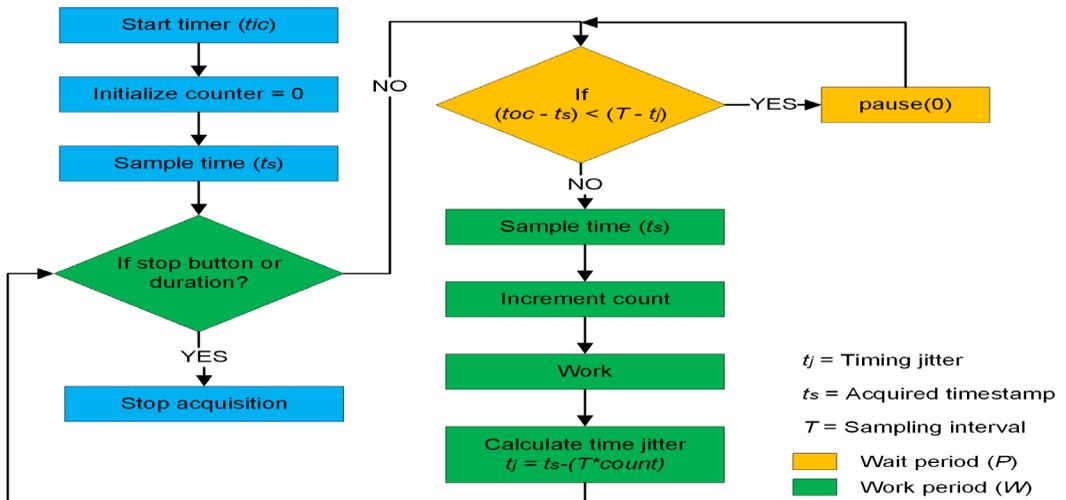


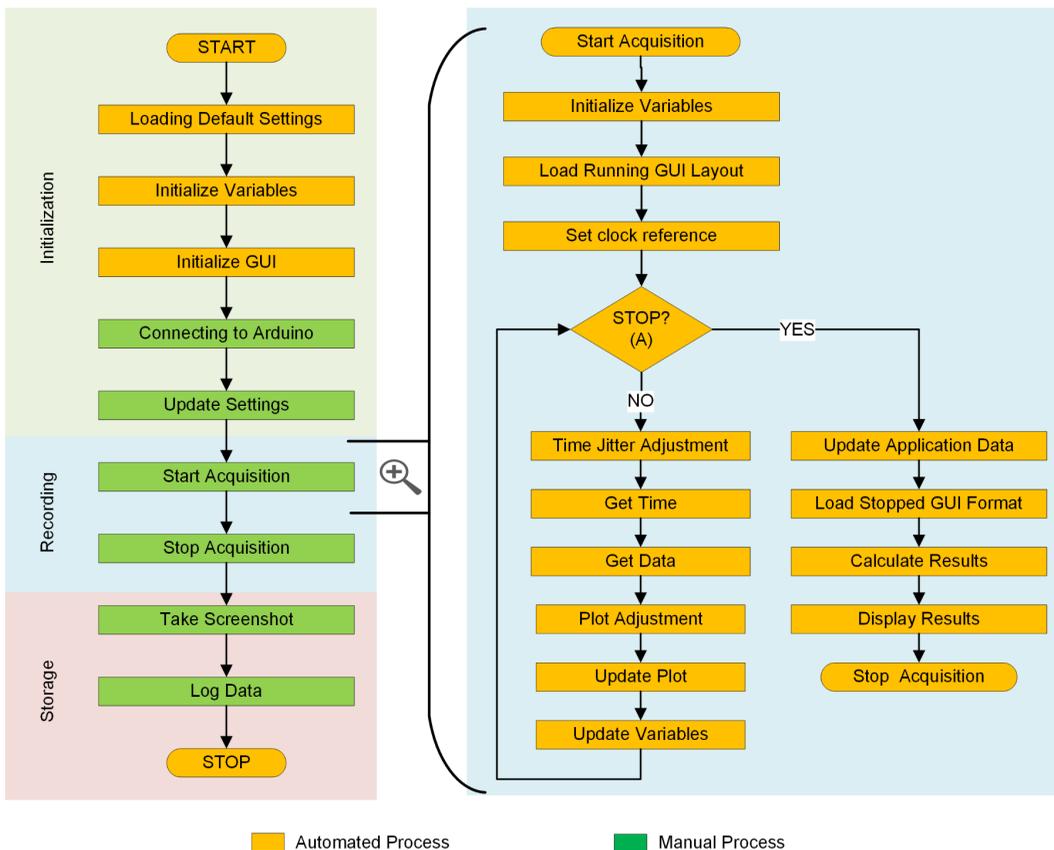
Figure 6. Flowchart of the post-hoc timing jitter correction mechanism



When the acquisition starts, there is no information on the t_j and it was assumed to be zero. Before starting the acquisition loop, i.e. before region B in Figure 5, the timer and counter are initialized, a time sample is taken and the pause period runs for one extra time at the beginning which is used to estimate the initial value of t_j . After $P0$ completes, the first acquisition loop starts, and subsequently, each loop contains a work period (W) and a pause period (P). In region B , the work period ($W1$) starts with taking a time sample, and the rest of the work is acquiring data from Arduino, data manipulation, and real-time plotting. At the end of the work period, the t_j is estimated and is used to adjust the pause period ($P1$) so that the sum of $W1$ and $P1$ is equal to the set sampling time. In the next iteration, in region C the second time sample is taken and the sampling process is repeated. In this way, timing jitter is tracked and corrected in each iteration.

4.6. Program Workflow

Figure 7. Flowchart of the acquisition process



A flowchart of the program workflow is shown in Figure 7. The left side of the flowchart is the main process flow and the right-side of the flowchart is the expanded view of the data acquisition process. The flowchart is color-coded to show automated and manual processes. The automated processes are the ones that do not require the user’s intervention whereas the manual processes require some input

or action from the user. The program flow is subdivided into three main procedures: initialization, recording, and storage.

In the initialization procedure, the program initializes GUI in its default state in which the variables are loaded with their hard-coded initial values and the various panels (like settings, control, display, and result) are in the inactive state. In this procedure, the program requires the user input for connecting the Arduino and updating the AfDaq settings. The user can set the various settings in the settings panel as shown in Figure 2. Once the settings are finalized the user can start the acquisition process from the control panel of the GUI, which initiates the recording procedure. In the recording procedure, the program enters into an automated process to acquire, manipulate, and plot data in real-time. The program loads the running GUI format which disables different panels as a fail-safe mechanism to avoid any unintended input from the user disrupting the acquisition process. The acquisition process runs until the session duration times-out or the user manually stops the process. Upon completion of the acquisition, the program loads the stopped GUI format, computes the various results, and displays them in the result panel. The final procedure is named storage in which the user can take a screenshot of the frontend of the GUI and/or save data in Excel or CSV format locally for offline processing. The various settings of the AfDaq are also stored along with data.

5. BENCHMARK RESULTS

To ascertain the reliability of the jitter correction mechanism, the AfDaq is benchmarked at various sampling rates from 1 to 50 Hz for multi-channel data acquisition. Two reliability parameters computed are: MAE (Mean Absolute Error) and Recorded Sampling Rate. MAE is computed as

$$MAE = \frac{\sum_{n=1}^N |T_{aq}(n) - T_{ref}(n)|}{N} \quad (\text{Eq. 1})$$

where T_{aq} is the Acquired timestamps taken from the saved data, T_{ref} is the Reference timestamp computed from the set sampling rate and N is the number of samples.

Table 3 lists the results of the benchmark. The columns I, II, III, IV, and V corresponds to the multichannel acquisition of the Analog Read command at A0, A1, A2, A3, and A4 pins respectively, e.g. Column I is a single-channel acquisition from A0 pin, and column II is double-channel acquisition at pin A0 and A1. For each case, 100 samples are taken and the mean absolute error is computed. For the first reliability parameter – MAE, the shaded values in Table 3 are where MAE is equal to 0 and corresponds to the best-case error-free data acquisition. As per this criterion, for 1 to 5 channel acquisition, a maximum sampling rate of 20 Hz, 15 Hz, 10Hz, 10Hz, and 10 Hz respectively should be used.

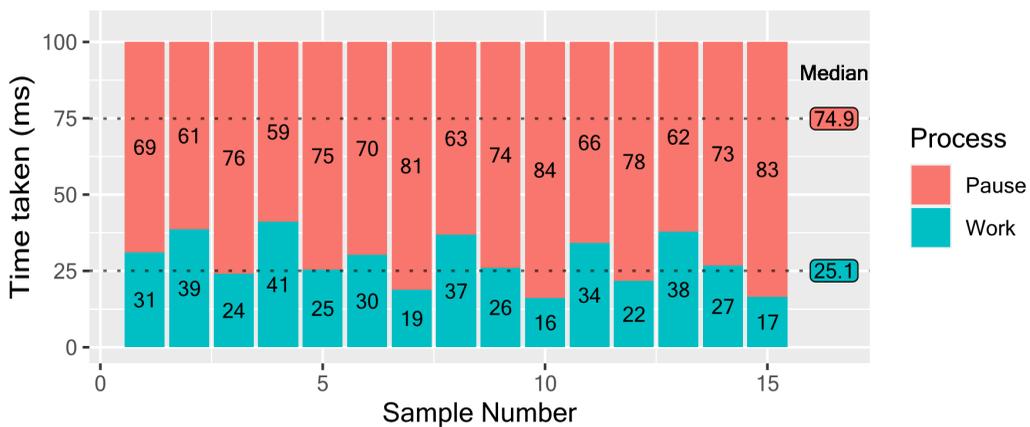
The recorded sampling rate is the effective sampling frequency at which acquisition takes place and it is always less than or equal to the set sampling frequency. The recorded sampling rate is computed as

$$Recorded\ Fs = \frac{N*1000}{T_{max}} \quad (\text{Eq. 2})$$

where T_{max} is the time taken in ms for N samples. For the second reliability parameter – recorded sampling rate, the values marked in are when the recorded sampling rate is the same as the set sampling rate. As per this criterion, for 1 to 5 channel acquisition, a maximum sampling rate of 40 Hz, 25 Hz, 15Hz, 10Hz, and 10 Hz respectively should be used. From Table 3, for 3-channel acquisition at a set sampling rate of 20 Hz, the recorded sampling rate comes out to be 20.4 Hz, which is larger than the

set sampling rate of 20 Hz. This is an approximation error due to a smaller number of samples ($N = 100$) and will come out to be 20 for larger values of N . It is to be noted that the best cases based on this reliability criteria are more than that based on MAE. In Table 3, the recorded sampling rate comes out to be approximately equal to the set sampling rate for MAE of less than 50. This is because small values of MAE represent the small jitter in the recorded time stamps which can be compensated by the post-hoc time jitter mechanism. Whereas when the larger values of MAE occur, the correction mechanism keeps diverging from reference timestamps due to larger lag. From the benchmark results, the sampling rate of 10 Hz comes out to be the best case to acquire any number of channels, with a maximum of 5 channels.

Figure 8. Time consumed by two main subparts (Work and Pause) during each acquisition loop (sample number) after time-jitter correction, recorded at a sampling rate of 10 Hz



To visualize the proposed jitter correction mechanism at work, the authors ran a single-channel test acquisition at a sampling rate of 10 Hz or a sampling interval of 100 ms and measures the time taken for sub-processes in each sample. Figure 8 shows the division of sampling interval (in ms) between Work and Pause parts during the first 20 samples. The Work part represents the various sub-processes that involve the acquisition, real-time data manipulation, calculations, and plotting. The Pause part represents the idle wait time in the program which ran till the time for sampling interval completes. The timing jitter in the Work part comes out with a median value of 25.1 ms calculated from 100 samples. The time taken by the Pause process is such that the total of time taken by the Work and Pause process will be equal to the sampling interval. Thus, the timing jitter in the Pause process is dependent on that of the Work process and has a median value of 74.9 ms in this case when calculated from 100 samples.

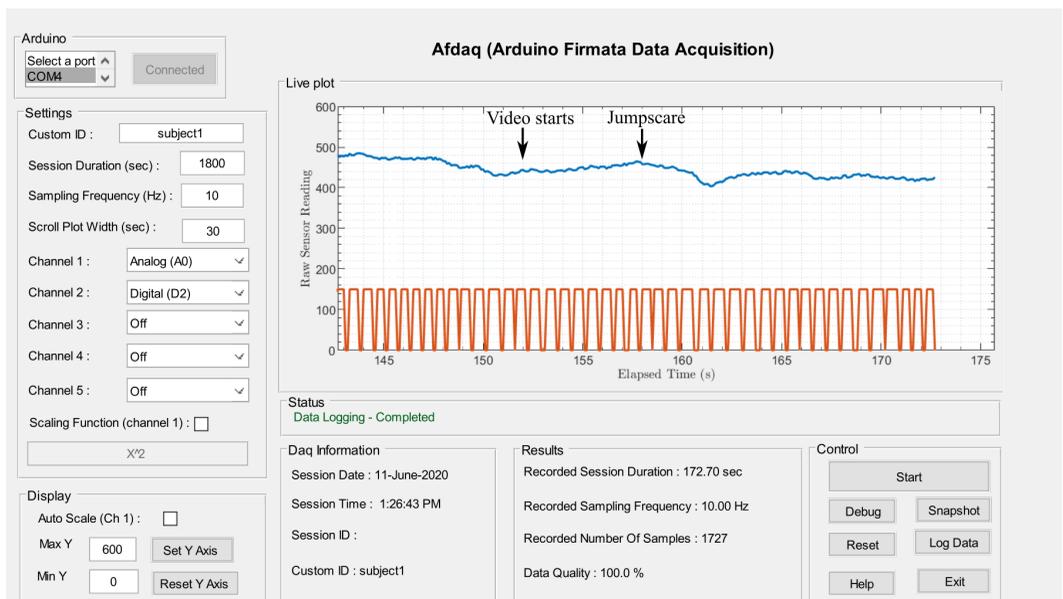
Table 3. Benchmark of AfDaq at various sampling rates

Set Fs (Hz)	Mean Absolute Error (ms) ^a					Recorded Fs ^b				
	I	II	III	IV	V	I	II	III	IV	V
1	0 ^c	0 ^c	0 ^c	0 ^c	0 ^c	1 ^d	1 ^d	1 ^d	1 ^d	1 ^d
5	0 ^c	0 ^c	0 ^c	0 ^c	0 ^c	5 ^d	5 ^d	5 ^d	5 ^d	5 ^d
10	0 ^c	0 ^c	0 ^c	0 ^c	0 ^c	10 ^d	10 ^d	10 ^d	10 ^d	10 ^d
15	0 ^c	0 ^c	2	157	898	15 ^d	15 ^d	15 ^d	14.4	11.8
20	0 ^c	4	48	924	1735	20 ^d	20 ^d	20.4 ^d	14.7	11.8
25	1	5	497	1408	2285	25 ^d	25 ^d	20.4	14.7	11.7
30	2	117	817	1761	2591	30 ^d	28.1	20.4	14.6	11.8
35	2	393	1042	2010	2799	35 ^d	28.1	20.3	14.5	11.9
40	2	536	1250	2217	3000	40 ^d	28.3	20.4	14.5	11.8
45	2	655	1328	2315	3114	44.9	28.6	21	14.6	11.9
50	8	767	1500	2456	3255	49.9	28.6	20.3	14.6	11.9

^a MAE is the mean of the absolute difference between Acquired and Reference time stamps (rounded-off to the nearest integer).
^b Recorded Fs is estimated from time taken to acquire 100 samples (rounded-off to single decimal place).
^c Acceptable values of MAE are taken as less than 0.5 ms.
^d Acceptable values of Recorded Fs are taken as equal to Set Fs.

6. USE CASE – MULTIMODAL PHYSIOLOGICAL DATA ACQUISITION

Figure 9. Acquisition of EDA and HR signal (blue and orange lines respectively) from a subject using AfDaq. A short video clip containing jump scare is manually shown at marked time intervals.



As a use case, multimodal physiological data from a subject (female, 23yrs.) is acquired using AfDaa. The investigation is conducted following the principles embodied in the Declaration of Helsinki and informed consent to participate in the test is obtained from the subject before data acquisition. Electro-Dermal Activity (EDA) and Heart Rate (HR) is acquired at a sampling rate of 10 Hz. For EDA and HR sensors low-cost and popular products from Seeed Studio – Grove-GSR Sensor (*Grove - GSR Sensor - Seeed Wiki*, n.d.) and Grove - Ear-clip Heart Rate Sensor (*Grove - Ear-Clip Heart Rate Sensor - Seeed Wiki*, n.d.) are used. The EDA sensor has an analog output range of 0-3.5V when operating at a rail-to-rail voltage of 5V. It is connected to the Analog pin A0 of Arduino Uno. The HR sensor is a photoplethysmograph based sensor, which can be connected to the ear-lobe and has a digital output of HIGH and LOW indicating the presence and absence of the heartbeat. It is connected to the digital pin D2 of Arduino Uno. Other settings are as shown in Figure 9. Both EDA and HR are acquired from the same subject in real-time. The EDA signal is taken from the distal phalanges of the index and middle finger of the non-dominant hand and the HR signal is taken from the left ear-lobe. During acquisition, the subject was asked to remain in a calm state until the stimulus is provided. To elicit autonomic sympathetic arousal response a jump scare video as a stimulus is manually shown to the subject via a smartphone. The video starts at the 152s mark and has a jump scare at the 158s mark of acquisition time.

From the real-time data in Figure 9, it is clear that after the jump scare the EDA signal has a significant drop in value due to decreased resistance (or in other terms increased conductivity) of the skin. This is the expected behavior from the human body in the sympathetic state (Kushki et al., 2011). The heartbeats are shown by peaks which at times become slightly narrower and wider which corresponds to heart rate variability (change in heart rate) during normal respiration (Zhu et al., 2019). The acquisition was manually stopped at the 173 sec time mark followed by data logging for offline analysis.

7. CONCLUSION

In this paper, the detailed journey of design and development of a MATLAB GUI based real-time Arduino data acquisition tool is presented. The tool was developed in keeping view of making a quick-to-deploy system with simple GUI, real-time visualization of signals, and customizability options which appeal to a wide range of audience. The acquisition process was designed to be a simple plug and play system so that less experienced programmers can easily use it, which greatly simplifies the task of data acquisition. The timing jitter is of the main concern when using the MATLAB Arduino package which is not addressed in the literature. Therefore, the descriptive statistics of timing jitter for all the sub-processes were analyzed and a post-hoc time jitter correction method was developed. However, due to the inherent delay in the MATLAB support package for Arduino, and the various interdependencies between some of the design considerations, the maximum sampling frequency is limited. The same was observed in the benchmark results. The maximum sampling rate of 10 Hz was achieved when acquiring data from 5 channels simultaneously, which is sufficient for the various forms of biofeedback applications like EDA biofeedback (Geršak & Drnovšek, 2020; Villarejo et al., 2012), Respiration Rate (RR) biofeedback (Frey et al., 2018; Moraveji et al., 2011) and Skin Temperature (SKT) biofeedback (Alhamid et al., 2012). This open-source tool can benefit a large section of researchers working in biofeedback or other domains where a higher sampling rate may not be required.

8. DATA AVAILABILITY

This is open data, open code, and replicable research. The final product - AfDaa is released as an open-source tool under the GPLv3 license. The software tool, data, and analysis scripts that support the findings of this study will be available as an open-source project hosted at the Open Science

Framework repository and can be accessed using DOI: [10.17605/OSE.IO/VCTJM](https://doi.org/10.17605/OSE.IO/VCTJM) following an embargo period until the paper is accepted/published.

FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.

REFERENCES

- Alhamid, M. F., Eid, M., Alshareef, A., & El Saddik, A. (2012). MMBIP: Biofeedback system design on Cloud-Oriented Architecture. *2012 IEEE International Symposium on Robotic and Sensors Environments Proceedings*, 79–84. doi:10.1109/ROSE.2012.6402610
- Ali, A. S., Zanzinger, Z., Debose, D., & Stephens, B. (2016). Open Source Building Science Sensors (OSBSS): A low-cost Arduino-based platform for long-term indoor environmental data collection. *Building and Environment*, *100*, 114–126. doi:10.1016/j.buildenv.2016.02.010
- Arduino Support from MATLAB - Hardware Support—MATLAB & Simulink. (n.d.). Retrieved October 2, 2020, from <https://in.mathworks.com/hardware-support/arduino-matlab.html>
- ArduPilot. (2020). <https://github.com/ArduPilot/ardupilot>
- Bhoyar, & Sonavane. (2015). Health at Home System Design for Remote Patient Monitoring. *International Journal on Recent and Innovation Trends in Computing and Communication*, *3*(8), 5303–5306. doi:10.17762/ijritcc.v3i8.4834
- Cao, T., Zhang, Q., & Thompson, J. E. (2015). Designing, Constructing, and Using an Inexpensive Electronic Buret. *Journal of Chemical Education*, *92*(1), 106–109. doi:10.1021/ed500509p
- Chevalier-Boisvert, M., Hendren, L., & Verbrugge, C. (2010). Optimizing Matlab through Just-In-Time Specialization. In R. Gupta (Ed.), *Compiler Construction* (pp. 46–65). Springer. doi:10.1007/978-3-642-11970-5_4
- Costa, D. G., & Duran-Faundez, C. (2018). Open-Source Electronics Platforms as Enabling Technologies for Smart Cities: Recent Developments and Perspectives. *Electronics (Basel)*, *7*(12), 404. doi:10.3390/electronics7120404
- D'Ausilio, A. (2012). Arduino: A low-cost multipurpose lab equipment. *Behavior Research Methods*, *44*(2), 305–313. doi:10.3758/s13428-011-0163-z PMID:22037977
- Famularo, N., Kholod, Y., & Kosenkov, D. (2016). Integrating Chemistry Laboratory Instrumentation into the Industrial Internet: Building, Programming, and Experimenting with an Automatic Titrator. *Journal of Chemical Education*, *93*(1), 175–181. doi:10.1021/acs.jchemed.5b00494
- Farahbod, F. (2020). *TelemetryViewer*. <https://github.com/farrellf/TelemetryViewer>
- Ferreira, F., Carvalho, V., Soares, F., Machado, J., & Pereira, F. (2015). Vital Signs Monitoring System Using Radio Frequency Communication: A Medical Care Terminal for Bedridden People Support. *Sensors & Transducers Journal*, *185*(2), 7.
- Firmata firmware for Arduino. (2020). <https://github.com/firmata/arduino>
- Fortin-Côté, A., Beaudin-Gagnon, N., Campeau-Lecours, A., Tremblay, S., & Jackson, P. L. (2019). Affective Computing Out-of-The-Lab: The Cost of Low Cost. *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 4137–4142. doi:10.1109/SMC.2019.8914646
- Frey, J., Grabli, M., Slyper, R., & Cauchard, J. R. (2018). Breeze: Sharing Biofeedback through Wearable Technologies. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12. doi:10.1145/3173574.3174219
- Gad, H. E., & Gad, H. E. (2015). Development of a new temperature data acquisition system for solar energy applications. *Renewable Energy*, *74*, 337–343. doi:10.1016/j.renene.2014.08.006
- Geršak, G., & Drnovšek, J. (2020). Electrodermal activity patient simulator. *PLoS One*, *15*(2), e0228949. doi:10.1371/journal.pone.0228949 PMID:32023317
- Grinias, J. P., Whitfield, J. T., Guetschow, E. D., & Kennedy, R. T. (2016). An Inexpensive, Open-Source USB Arduino Data Acquisition Device for Chemical Instrumentation. *Journal of Chemical Education*, *93*(7), 1316–1319. doi:10.1021/acs.jchemed.6b00262 PMID:27453587
- Grove—Ear-clip Heart Rate Sensor—Seed Wiki. (n.d.). Retrieved October 2, 2020, from https://wiki.seedstudio.com/Grove-Ear-clip_Heart_Rate_Sensor/

Grove—GSR Sensor—Seed Wiki. (n.d.). Retrieved October 2, 2020, from https://wiki.seedstudio.com/Grove-GSR_Sensor/

Ishikawa, M., & Maruta, I. (2010). Rapid Prototyping for Control Education using Arduino and Open-Source Technologies. *IFAC Proceedings Volumes*, 42(24), 317–321. 10.3182/20091021-3-JP-2009.00060

Kim, D. K., Kim, J., Lee, E. C., Whang, M., & Cho, Y. (2011). Interactive emotional content communications system using portable wireless biofeedback device. *IEEE Transactions on Consumer Electronics*, 57(4), 1929–1936. <https://doi.org/10.1109/TCE.2011.6131173>

King, C. E., Wang, P. T., McCrimmon, C. M., Chou, C. C. Y., Do, A. H., & Nenadic, Z. (2014). Brain-computer interface driven functional electrical stimulation system for overground walking in spinal cord injury participant. *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 1238–1242. 10.1109/EMBC.2014.6943821

Koenka, I. J., Sáiz, J., & Hauser, P. C. (2014). Instrumentino: An open-source modular Python framework for controlling Arduino based experimental instruments. *Computer Physics Communications*, 185(10), 2724–2729. <https://doi.org/10.1016/j.cpc.2014.06.007>

Kushki, A., Fairley, J., Merja, S., King, G., & Chau, T. (2011). Comparison of blood volume pulse and skin conductance responses to mental and affective stimuli at different anatomical sites. *Physiological Measurement*, 32(10), 1529–1539. <https://doi.org/10.1088/0967-3334/32/10/002>

Luo, H.-J., Lin, S.-X., Wu, S.-K., Tsai, M.-W., & Lee, S.-J. (2017). Comparison of segmental spinal movement control in adolescents with and without idiopathic scoliosis using modified pressure biofeedback unit. *PLoS One*, 12(7), e0181915. <https://doi.org/10.1371/journal.pone.0181915>

Mabbott, G. A. (2014). Teaching Electronics and Laboratory Automation Using Microcontroller Boards. *Journal of Chemical Education*, 91(9), 1458–1463. <https://doi.org/10.1021/ed4006216>

MATLAB Execution Engine. (n.d.). Retrieved September 10, 2020, from <https://in.mathworks.com/products/matlab/matlab-execution-engine.html>

MATLAB Support Package for Arduino Hardware Documentation—MathWorks India. (n.d.). Retrieved October 2, 2020, from <https://in.mathworks.com/help/supportpkg/arduinoio/>

McClain, R. L. (2014). Construction of a Photometer as an Instructional Tool for Electronics and Instrumentation. *Journal of Chemical Education*, 91(5), 747–750. <https://doi.org/10.1021/ed400784x>

Moeyersons, B., Fuss, F. K., Tan, A. M., & Weizman, Y. (2016). Biofeedback System for Novice Snowboarding. *Procedia Engineering*, 147, 781–786. <https://doi.org/10.1016/j.proeng.2016.06.318>

Moraveji, N., Olson, B., Nguyen, T., Saadat, M., Khalighi, Y., Pea, R., & Heer, J. (2011). Peripheral paced respiration: Influencing user physiology during information work. *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 423–428. 10.1145/2047196.2047250

Mumtaz, Z., Ullah, S., Ilyas, Z., Aslam, N., Iqbal, S., Liu, S., Meo, J. A., & Madni, H. A. (2018). An Automation System for Controlling Streetlights and Monitoring Objects Using Arduino. *Sensors (Basel)*, 18(10), 3178. <https://doi.org/10.3390/s18103178>

Nichols, D. (2017). Arduino-Based Data Acquisition into Excel, LabVIEW, and MATLAB. *The Physics Teacher*, 55(4), 226–227. <https://doi.org/10.1119/1.4978720>

Pearce, J. M. (2012). Building Research Equipment with Free, Open-Source Hardware. *Science*, 337(6100), 1303–1304. <https://doi.org/10.1126/science.1228183>

Polo, A., Narvaez, P., & Robles Algarín, C. (2018). Implementation of a Cost-Effective Didactic Prototype for the Acquisition of Biomedical Signals. *Electronics (Basel)*, 7(5), 77. <https://doi.org/10.3390/electronics7050077>

Salman, F., Cui, Y., Imran, Z., Liu, F., Wang, L., & Wu, W. (2020). A Wireless-controlled 3D printed Robotic Hand Motion System with Flex Force Sensors. *Sensors and Actuators. A, Physical*, 309, 112004. <https://doi.org/10.1016/j.sna.2020.112004>

Saraò, A., Clocchiatti, M., Barnaba, C., & Zuliani, D. (2016). Using an Arduino Seismograph to Raise Awareness of Earthquake Hazard Through a Multidisciplinary Approach. *Seismological Research Letters*, 87(1), 186–192. <https://doi.org/10.1785/0220150091>

Sepúlveda, S., Reyes, P., & Weinstein, A. (2015). Visualizing physiological signals in real-time. In K. Huff & J. Bergstra (Eds.), *Proceedings of the 14th Python in Science Conference* (pp. 182–186). <https://doi.org/10.25080/Majora-7b98e3ed-01c>.

Soler-Llorens, J. L., Galiana-Merino, J. J., Nassim-Benabdeloued, B. Y., Rosa-Cintas, S., Ortiz Zamora, J., & Giner-Caturla, J. J. (2019). Design and Implementation of an Arduino-Based Plug-and-Play Acquisition System for Seismic Noise Measurements. *Electronics*, 8(9), 1035. [10.3390/electronics8091035](https://doi.org/10.3390/electronics8091035)

Soler-Llorens, J. L., Galiana-Merino, J. J., Giner-Caturla, J., Jauregui-Eslava, P., Rosa-Cintas, S., & Rosa-Herranz, J. (2016). Development and programming of Geophonino. *Computers & Geosciences*, 94(C), 1–10. <https://doi.org/10.1016/j.cageo.2016.05.014>

Toulson, R. (2008). Advanced rapid prototyping in small research projects with Matlab/Simulink. *2008 IEEE International Symposium on Industrial Electronics*, 1–7. [10.1109/ISIE.2008.4677317](https://doi.org/10.1109/ISIE.2008.4677317)

Urban, P. L. (2014). Open-Source Electronics As a Technological Aid in Chemical Education. *Journal of Chemical Education*, 91(5), 751–752. <https://doi.org/10.1021/ed4009073>

Villarejo, M. V., Zapirain, B. G., & Zorrilla, A. M. (2012). A Stress Sensor Based on Galvanic Skin Response (GSR) Controlled by ZigBee. *Sensors (Basel)*, 12(5), 6075–6101. <https://doi.org/10.3390/s120506075>

Wu, Y., & Bryan-Kinns, N. (2019). Musicking with an interactive musical system: The effects of task motivation and user interface mode on non-musicians' creative engagement. *International Journal of Human-Computer Studies*, 122, 61–77. <https://doi.org/10.1016/j.ijhcs.2018.07.009>

Zhu, J., Ji, L., & Liu, C. (2019). Heart rate variability monitoring for emotion and disorders of emotion. *Physiological Measurement*, 40(6), 064004. <https://doi.org/10.1088/1361-6579/ab1887>

ENDNOTES

- ¹ AfDAQ translates to Arduino Firmata Data Acquisition.
- ² User have to explicitly only install this free add-on package in MATLAB before using AfDAQ.
- ³ Cold start refers to case when application restarted. In this context, the first acquisition in AfDAQ is a cold start and all the next acquisitions till the AfDAQ is closed are hot start.
- ⁴ Overall disk activity shown by task manager as more than 50% for 5 seconds is assumed to be significant for causing data corruption, and the acquired readings are discarded.

Kulbhushan Chand received the B.Tech. degree in Electronics and Communication Engineering from the Guru Nanak Dev University, India, in 2010, and the M.Tech. degree in Electronics and Communication Engineering from the Lovely Professional University, India, in 2013. He is currently working towards the Ph.D. degree in Electronics and Communication Engineering with Dr. Arun Khosla at the Dr. B R Ambedkar National Institute of Technology Jalandhar, India. His research interests include biofeedback, embedded systems, wearable physiological sensors and computing, healthcare and gamification.

Arun Khosla is working as Professor in the Department of Electronics and Communication Engineering at Dr B R Ambedkar National Institute of Technology, Jalandhar, Punjab. His research interests include soft computing, artificial intelligence and machine learning, assisted technologies, healthcare and gamification.