First Semester Report: On-Board Sound Intensity Data Acquisition System (OBSIDAS)

Muhammed Abdalla, Daniel Cardosi, Aurojit Chakraborty,
Joseph Lucatorto, and Javier González Santamaría

Executive Summary (Aurojit Chakraborty) — There is an ever-growing body of evidence that demonstrates the negative effects of noise pollution: not only does it harm local wildlife near major roadways, but it is also linked to health issues such as tinnitus, hearing loss, and heart disease. Automobile traffic is a major driver of noise pollution, yet the current sound-measurement tools used by the U.S. Department of Transportation's Volpe Center consist in outdated and cumbersome programs. The software to be developed, requested by Volpe, aims to make their road-acoustics research more streamlined and scalable by interfacing with an external microphone array to perform fully automated measurements of tire-pavement noise via the On-Board Sound Intensity (OBSI) method. The software will be a LabVIEW executable that leverages a C++ subroutine for all computations on sound data.

Index Terms— Audio input/output, Hardware/software interfaces, Software/Software Engineering, Sound and Music Computing

1 Introduction (Joseph Lucatorto)

When designing and building cars and roads, many variables must be considered to maximize their harmony not only with drivers but also with the environment and society, and noise production is chief among these variables. Given that tire-pavement noise is the predominant source of roadway noise at speeds above 30 miles per hour [1] and increases logarithmically with increasing speed [2], it is especially useful to measure on-board sound intensity because it corresponds to the energy generated at the interface of a vehicle's tire and the pavement beneath it and thus correlates with a vehicle's contribution to roadway noise. Access to a reliable system for this form of measurement may aid in the creation of design constraints for automobile manufacturers and traffic engineers alike, which will in turn produce quieter traffic.

Organizations such as the Volpe Center adhere to an existing methodological standard for road-acoustics research, known as the On-Board Sound Intensity (OBSI) method, which is detailed in AASHTO TP 76-10 [3]. This standard defines the entire measurement procedure, including specific numerical data, necessary calculations, and the configuration of all involved hardware. Per Fig. 1 below, sound-pressure data is collected using two pairs of specialized probes in a standardized array configuration, and a single measurement trial proceeds as follows: over a user-prescribed duration of time, the probes feed acoustic data to a laptop in the vehicle as it drives. The current project develops a software program that collects such measurements and computes a number of acoustical metrics, all specified by Volpe.

The software currently used by Volpe to gather OBSI data is outdated: it has a cumbersome interface, only functions on legacy operating systems, and has extremely limited sensor compatibility, leading to increased costs for researchers. Our team intends to create a brand new software package that adheres completely to the AASHTO-standardized process for OBSI measurements. This new software package will collect the same data as the Volpe Center's current software, with the added benefits of greater hardware flexibility, easy distribution, no overhead for license fees, a user-friendly GUI (similar to that of Fig. 2), and access to the source code.

- *Muhammed Abdalla is a Computer Engineering undergraduate student in the Boston University Department of Electrical and Computer Engineering, Boston, MA 02215. Email: muhabda@bu.edu*
- *Daniel Cardosi is an Electrical Engineering undergraduate student in the Boston University Department of Electrical and Computer Engineering, Boston, MA 02215. Email: dcardosi@bu.edu*
- *Aurojit Chakraborty is an Electrical Engineering undergraduate student in the Boston University Department of Electrical and Computer Engineering, Boston, MA 02215. Email: aurojit@bu.edu*
- *Joseph Lucatorto is an Electrical Engineering undergraduate student in the Boston University Department of Electrical and Computer Engineering, Boston, MA 02215. Email: jlucator@bu.edu*
- *Javier González Santamaría is a Computer Engineering undergraduate student in the Boston University Department of Electrical and Computer Engineering, Boston, MA 02215. Email: javiergs@bu.edu*

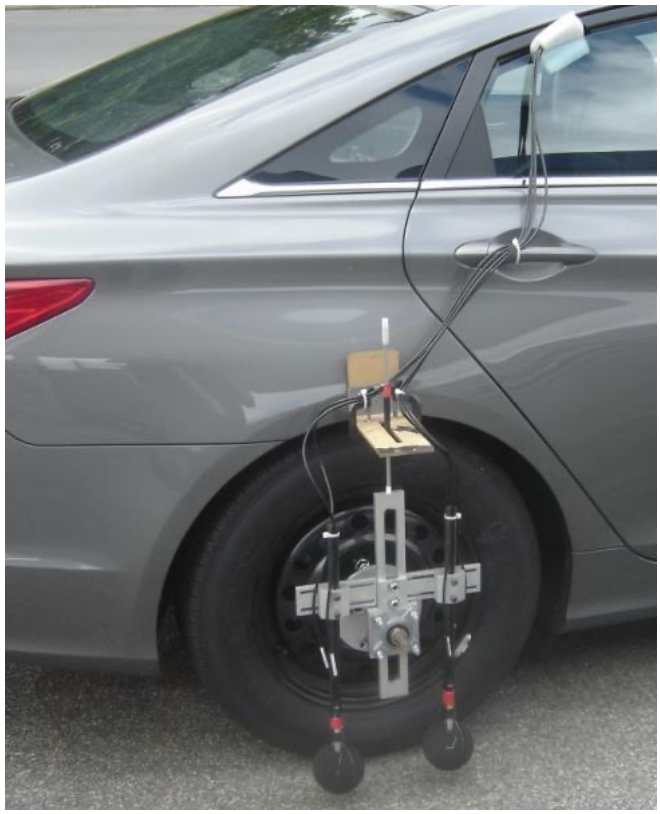


Fig. 1. OBSI probe configuration: each of the two probes consists of a pair of microphones.

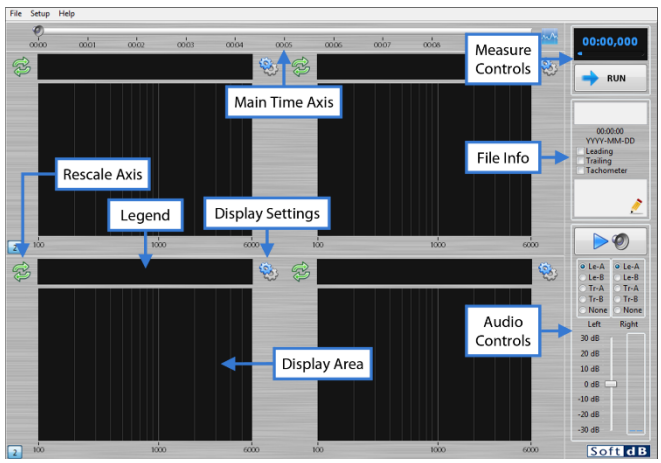


Fig. 2. Example of an OBSI-software user interface, taken from Soft dB's OBSI program.

2 Concept Development

2.1 Problem of Customer (Joseph Lucatorto)

Volpe is currently using outdated and unwieldy OBSI-measurement software. Consequently, there is a need to develop an updated and more manageable program. Outdated software can reduce employee productivity, in particular by failing to function with other, newer software that otherwise functions separately. Such is the case here; Volpe's lack of access to modern OBSI-measurement software creates inefficiencies across all stages of the research process, from cumbersome trialing to prohibitive hardware restrictions, yet purchasing new proprietary software could incur expensive, renewable license fees, which can be quite expensive. Our program will provide the Volpe Center with a solution that avoids such adverse consequences.

2.2 Product to be Delivered (Javier González Santamaría)

The product to be delivered is a LabVIEW-based measurement program, in two forms: the first is a standard .exe executable, and the second an editable VI file. We will be using LabVIEW 2017 and its associated libraries, and internally to the LabVIEW VI we will make use of C++ and the FFTW library for numerical computations.

The program will have a user-friendly graphical user interface, and it will be designed to work exclusively with a quad-probe microphone array, per AASHTO TP 76-10. The program will allow the user to calibrate the experimental probes individually, to set a trial time, and to initiate data collection. After data collection, the program will decompose the data into one-third-octave frequency bands with center frequencies ranging from 250 to 5040 Hz and thereafter calculate a variety of acoustical statistics relevant to OBSI evaluation (see Appendix 1 for details of requirements). Finally, the program will allow the user to easily export both processed and unprocessed data to a desired readable format such as a .csv file.

2.3 Alternative Solutions Considered (Daniel Cardosi)

The use of LabVIEW 2017 was not a decision on the part of our team but rather an explicit requirement of the Volpe Center; the internal use of C++ for numerical computation, likewise, was chosen over the use of Python to avoid possible issues arising from incompatibility between Python and LabVIEW, the latter of which only introduced native capability for calling Python scripts in the version of 2018.

Also considered as an alternative to the C++ subroutine was LabVIEW itself, which provides a built-in VI for DFT calculation, the most computationally intensive element of the subroutine. This VI, however, utilizes the canonical matrix formulation of the DFT in its computation, which has a

time-complexity of $O(n^2)$; the development of the subroutine in C++ therefore allowed a variety of more efficient O(nlog(n)) algorithms to be considered, of which there were eight:

1. Danielson-Lanczos/Runge-König (radix-2)
2. Cooley-Tukey (mixed-radix, decimation-in-time)
3. Gentleman-Sande (mixed-radix, decimation-in-frequency)
4. Transposed Stockham (in-place)
5. Pease (row-oriented Cooley-Tukey)
6. Stockham (row-oriented Transposed Stock ham)
7. Good-Thomas, Rader, Bluestein (prime sizes)
8. Duhamel-Hollmann (split-radix/ unsymmetric-butterfly)

From analyses of the above schemes for FFT-based DFT computation was selected the FFTW3 C library, specifically given the library's recursive implementation of the Cooley-Tukey algorithm incorporating stages of decimation both in time and in frequency, a modified split-radix architecture (after the Bernstein tangent algorithm) with adaptively chosen radices, and subroutines for prime-length inputs per the Good-Thomas (prime-factor) and Rader algorithms. Tentatively chosen were the halfcomplex-format real-to-real R2HC transform, expected to yield further reductions in runtime by exploiting Hermitian redundancy in outputs, the MEASURE planning-rigor flag, expected to do the same by balancing optimally the expenses of calculating an efficient transform-computation routine with the benefits in execution time thereof, and the DESTROY_INPUT restriction flag, expected to do the same by allowing for the overwriting of input arrays.

C++ thus provides more efficiency than other solutions, both in the control it allows over DFT computations and in its low-latency interoperation with LabVIEW 2017.

3 System Description

3.1 Client-Provided Hardware (Javier González Santamaría)

Volpe has provided the following hardware items that together constitute the input-signal chain of the OBSIDAS; the OBSIDAS software will interface with this signal chain through the additionally provided Native Instruments USB-4431 data acquisition system.

1. 4 GRAS 26AK preamplifiers
2. 4 GRAS 40AI paired microphones
3. 2 GRAS 12AQ 2-channel power modules
4. 1 Bruel and Kjaer 4231 calibrator
5. 4 GRAS 7-pin LEMO 10-meter extension cables
6. 4 BNC 25-foot extension cables

The signal chain is constructed as follows: each 40AI microphone is attached to a 26AK preamplifier, which is then connected to a single input channel of a 12AQ power module via a LEMO cable. The output of the power module is then routed to the USB-4431 DAQ, which is connected via USB to a computer loaded with the OBSIDAS software. Calibration proceeds by positioning the output driver of the 4231 calibrator directly against a 40AI microphone capsule and digitally adjusting the sensitivity parameter corresponding to the microphone.

3.2 Modular LabVIEW Application with Multithreaded Environment (Muhammed Abdulla)

This section of the report describes the software architecture proposed for the project. Before interfacing the data acquisition device and data-processing code to the LabView application, the solution needs to follow a modular implementation for realistic use and maintainability. There are four submodules (Fig. 5) that are expected to be built, all of which mutually exclusive from another giving maximum flexibility in scalability. The use of multiple SubVI's (submodules) for reusability further facilitates independent development, testing, and reuse of specific tasks such as data acquisition, processing, and storage, promoting efficiency and consistency across various applications.

In order to address further scalability and to meet the demands of expensive computation, a multithreading environment is incorporated into our LabVIEW implementation of OBSIDAS-LV. This design allows for parallel execution of data acquisition, transformation, processing, and relevant operations. Additionally, buffer queues are employed to create a robust flow of data between stages. Buffers also accommodate variations in

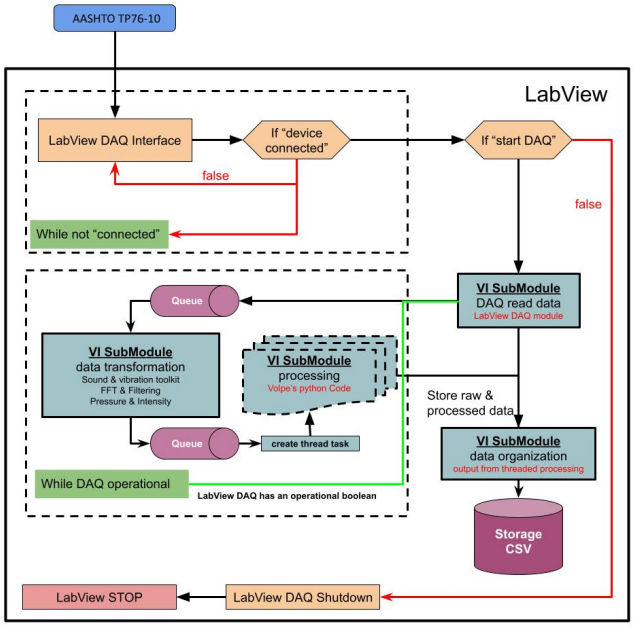


Fig. 3. Schematized diagram of program architecture.

processing speeds and prevent data loss. This design choice ensures a smooth information flow even under high-throughput conditions. The implementation of these queue buffers will be an array of the type to process, either string or integer. The way these buffers work is like a FIFO. Two of them will be implemented: after the read before data transformation and one after data transformation before data processing. The main methods of the queue buffers are push() and pop() by source and destination respectively.

Furthermore, the LabView implementation of OBSIDAS must have a write back to file operation. There are two watch cases for this: 1) the file format of data and 2) the organization of this data. Since OBSIDAS-LV is multithreaded, there is no sequencing determined to write out. On top of that, a common issue with multithreaded applications are race conditions. To mitigate race conditions, the processed data associated with its original data will have a timestamp or immutable value linked to it. By adding immutable tags to the processed data, OBSIDAS-LV maintains a chronological order in the data path to write back. LabVIEW, in conjunction with C++ and Python processing, can leverage timestamping to associate each piece of processed data with a specific time reference. This ensures accurate temporal alignment between the data generated in the LabVIEW environment and its corresponding processed results in the Python threads. Timestamping not only aids in organizing the data systematically but also facilitates easier retrieval and analysis, enhancing the overall coherence and reliability of the integrated system. This meticulous approach to temporal synchronization contributes to the effectiveness and precision of the LabVIEW application with multithreaded Python processing, ensuring the seamless handling of acquired data.

Understanding how the software datapath and control flow is crucial for clients who will oversee the application and perform their own modifications. Since our architecture has the ability for parallel processing through a multithreaded environment, case statements are utilized for State-Based Operation. By implementing Finite State Machines, the control flow for the operational states of hardware and software components prevents unintended interactions and maintains system stability. It ensures that the LabVIEW application behaves consistently under various conditions, making it easier for clients to understand, use, and maintain. Overall, designing a multithreaded system with predictable states provides both structure and reliability for OBSIDAS-LV.

3.3 Hybrid LabVIEW-C++ Signal-Processing Submodule (Daniel Cardosi)

For each single trial of data collection, all computations on the raw audio data are executed via a LabVIEW submodule placed within the data path of the superordinate OBSIDAS-LV module detailed above. This submodule allocates all processing tasks to an external C++ script called by an instance of the LabVIEW-native Call Library Function Node.

As schematized in Fig. 4, the C++ script accepts arrays containing raw voltage-value sound data from LabVIEW, computes the required acoustical metrics in the frequency domain, and returns these values to LabVIEW with their appropriate precisions. As visible in the block diagram of Fig. 5, the LabVIEW VI interfaces with the C++ script to function as a self-contained submodule within the final LabVIEW executable; in the final OBSIDAS, the instance of the Acquire Sound Data subVI is to be replaced with the input arrays captured from the hardware input-signal chainn through the DAQ.

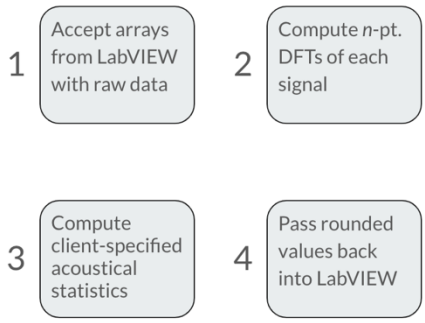


Fig. 4. Diagram of C++ script developed for per-trial computations.

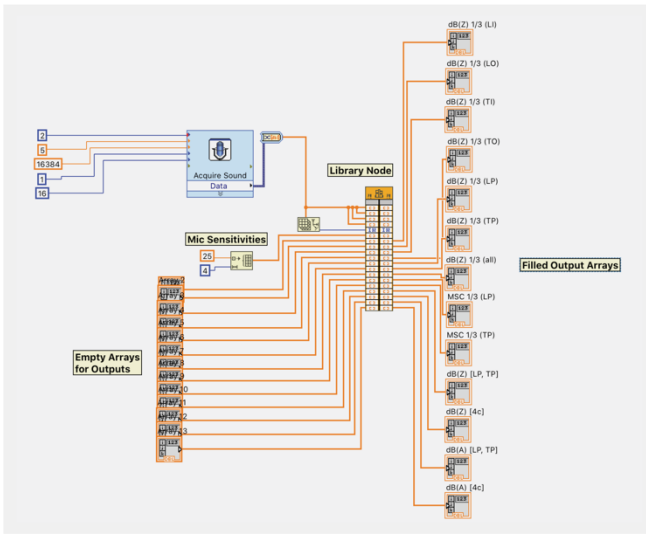


Fig. 5. Block diagram of LabVIEW VI developed for per-trial computations.

4 First Semester Progress

4.2 Summary of Progress (Daniel Cardosi)

Most significant in our progress heretofore is the complete development of a LabVIEW-C++ submodule that forms the core of the OBSIDAS system and is to be called on a trial-by-trial basis, as described in section 3.3 of this report. In particular, this submodule computes all client-specified acoustical metrics for the data of a single measurement trial, and its completion spans that of eight of the twelve expected deliverables originally listed in the client-prepared project-description document; remaining are the following four features, all to be developed externally to the submodule:

1. Microphone calibration via user-editable sen sitivity parameters
2. Manually triggered measurement trials of user-specified duration
3. Cumulative data storage across multiple trials
4. User-prompted data compilation and data ex port to a human-readable format

The C++ script of the submodule implements the FFTW library for DFTs and computes in the Fourier (frequency) domain all client-specified acoustical metrics from inputted audio data. The script meets all requirements for its incorporation into the signal-processing submodule of the final OBSIDAS, including support for four-channel input sequences and for user-defined microphone-sensitivity values. Additionally coded into the script is the computation of A-weighted sound pressure levels (SPLs), originally unspecified by the client but required per the AASHTO TP 76-10 Standard in which the official OBSI measurement method is delineated. Listed below are the specific metrics computed, where "OB14" denotes a collection of fourteen values, each computed in one of the fourteen third-octave-bands with center frequencies ranging from 250 to 5040 Hz:

1. Single-microphone unweighted SPL (OB14); four per trial
2. Single-probe unweighted SPL (OB14); two per trial
3. Across-probe-averaged unweighted SPL (OB14); one per trial
4. Across-probe-averaged A-weighted SPL (OB14); one per trial
5. Inter-microphone magnitude-squared coherence; two per trial
6. Entire-spectrum unweighted single-probe SPL; two per trial
7. Entire-spectrum unweighted across-probe-averaged SPL; one per trial
8. Entire-spectrum A-weighted across-probe-averaged SPL; one per trial

An initial testing procedure was performed on a preliminary version of the script via a parallel program developed in MATLAB R2023a, and at a later date a framework was created within the main function of the final C++ script to facilitate the comparative testing of FFTW options in the future; in particular, this framework utilizes the chrono STL sublibrary to record elapsed times between memory allocation and deallocation, equivalently between the point of FFT execution to the point of output-value storage in a vector containing the information of a one-sided transform, negative

frequencies being ignored and coefficients being scaled by two where necessary.

The functionalities of all coded functions were verified by simulating a four-channel input via the libsndfile C library, through which were read into C++ as vectors the client-provided .wav files obtained from real calibrations and tire-pavement-noise recordings on an existing OBSI-accordant probe fixture; Fig. 6 shows a portion of the command-line output generated upon the program's execution with the aforementioned pavement-noise example file as an input and with all sensitivity-parameters fixed at 25 mV/Pa, where LI = 'leading probe, inside microphone'; LO = 'leading probe, outside microphone'; TI = 'trailing probe, inside microphone'; TO = 'trailing probe, outside microphone'; LP = 'leading probe'; TP = 'trailing probe'; and MSC = 'magnitude-squared coherence'.

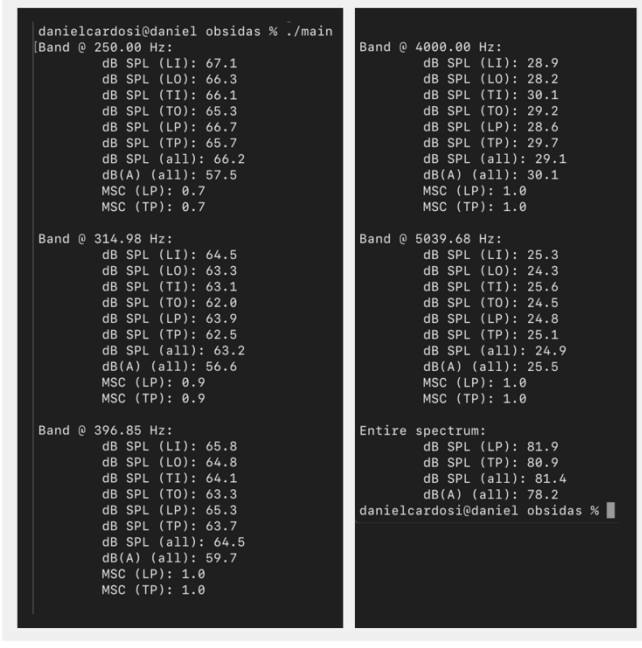


Fig. 6. Output (partially displayed) of C++ script for 4-channel noise-sequence input.

The LabVIEW VI that interfaces with the C++ script (see Fig. 5 above) was created in LabVIEW 2023 Q3 Community. The C++ codewas first augmented to include a LabVIEW-specific entry-point function (with C-linkage to remove type-decorations), the arguments of which were all necessary input and output arrays, passed in the form of pointers to doubles and preallocated in a LabVIEW-internal Call Library Function Node. To maintain portability despite the use of an external library (FFTW), FFTW was rebuilt statically into a single indexed archive of object files, specifically in the format of a Unix archiver file (of extension .a). The C++ source code was then compiled into a dynamic, shared library (of extension .dylib) linked to this archive, and this library was loaded into LabVIEW, again by way of the Call Library Function Node.

The functionality of the VI was verified with both uniform white noise, generated as a random-number sequence, and fixed-frequency pure tones, played through a JBL 305P MkII studio monitor and captured via an instance of the Acquire Sound Data subVI configured to match the sampling rate and bit depth of the hardware input-signal chain of the final OBSIDAS.

A Gantt chart detailing our progress in addition to future scheduled tasks is included below in Appendix 2.

4.2 Additional Individual Reports

(Muhammed Abdalla)

Over the course of the semester in Senior Design, my contributions have been pivotal to shape the trajectory of our assigned project. One of my key initiatives was to obtain a LabView 2017 license. This was a decision that was prolonged over the semester due to miscommunication and laboratory delays. We needed to acquire LabView on a desktop under a shared account that Team 8 can access only. This strategic move not only showcased proactive problem-solving but also significantly improved the overall flexibility of my team. By ensuring that my team members could contribute from various locations, we not only solved logistical challenges but also demonstrated adaptability in the face of evolving project dynamics.

In addition to addressing logistical concerns, I played a role in establishing the project's technical foundation. By recognizing the importance of a well-designed system architecture, I took the lead in crafting a modular framework. My software architecture conveys a modular development plan as well as positions the project for future expansions or modifications with minimal disruptions in the software. The modular approach underscores a commitment to thoughtful engineering practices, reinforcing the project's resilience and adaptability in the face of evolving requirements and challenges. I was able to start pushing one section of the architecture to my

teammates and take on the rest of the data path of the process.

Another significant aspect of my contributions revolves around the meticulous focus on data collection for hardware components. Understanding the central role of reliable data in engineering analyses, I directed efforts towards ensuring the precision and accuracy of our data collection processes. This emphasis on data quality not only contributes to the project's overall integrity but also establishes a solid foundation for informed decision-making. The meticulous approach to data collection reflects a commitment to the highest standards of engineering, ensuring that our project is built on a robust and reliable information base.

(Joseph Lucatorto)

Over the course of the semester, I took part in many areas of my group's project. Early on, as our group communicated with the Volpe Center about the details of our project, I focused on outlining our goals in our written material. I wrote and edited many parts of our PDRR Written Report, mainly in the sections "The Need for this Project" and "Problem Statement and Deliverables". In addition to writing the body of these sections, I did most of the background research and included relevant scholarly articles in our references section. In meetings with our client, I took notes of the points raised by them in order to better define the goals we would need to set going forward. I also worked with Javier on the initial stages of the layout of our program; brainstorming on potential features to include during some lab sessions. After we received our equipment from Volpe, however, I became much more involved in the hardware side of our project. For the initial tests of our hardware, I assembled all of the necessary equipment and, with the help of other group members, was able to produce an output signal that corresponded to the input generated by our 1kHz calibration unit. This initial test utilized the oscilloscope at our bench in PHO 111, however, later on I managed the equipment for our first test with Daniel's MATLAB program, where we connected our hardware to a PC.

As our work on this project continues into next semester, I am considering how my contribution to my group will change over time. Although, at the moment, our hardware works fairly reliably, we may need to continue to experiment with our setup as we develop our OBSIDAS program. Eventually, some changes may need to be made as we transition from testing in a lab to in an actual vehicle. In addition to contributing to the hardware aspects of this project, I would like to help where I can in other areas. Some potential areas for my contribution include the layout of our program, as I worked previously with Javier on some initial ideas for our layout, and the software of the program itself, which I was hesitant to get involved with due to my lack of experience, but would be willing to learn more about as the project continues.

(Javier González Santamaría)

This semester, I've been actively involved in the On-Board Sound Intensity Data Acquisition System project, taking on responsibilities ranging from hardware testing to LabVIEW interface development. Despite having no prior experience with LabVIEW, I undertook projects to get acquainted with the tool.

In hardware testing, I ensured the synchronization and functionality of preamplifiers, intensity microphones, and calibrators. The successful test outcomes, as detailed in the logbook, validated the reliability of the equipment provided by our client. These tests involved intricate processes such as connecting preamplifiers, intensity microphones, and calibrators, and conducting FFT analysis, contributing to a comprehensive evaluation of the sound intensity measurement system.

On the software front, my focus shifted to LabVIEW, where I initiated the development of the user interface. Inspired by existing interfaces, particularly SoftdB's OBSI Module, I crafted a LabVIEW interface that accommodates multiple tabs for streamlined signal visualization. This interface aims to provide a user-friendly experience, facilitating the interpretation of sound data captured by each microphone.

Collaboration on various project assignments, such as presentations, essays, and testing plans, was crucial in overcoming challenges, including initial client communication issues and acquiring LabVIEW licenses. The anechoic chamber testing showcased our commitment to evaluating the system under different conditions.

5 Technical Plan

We intend to have a working product to deliver to Volpe by the beginning of March of 2024. By dividing the remaining items into two categories, Signal Processing and Software Datapath, we will streamline the development process and maximize our productivity.

5.1 Development Plan for Software Datapath (Muhammed Abdulla)

1. Functional Verification: verify each thread's designated function, such as data acquisition and processing; confirm accurate interfacing with the data acquisition device.

2. State-Based Operation: test Case Statements for smooth state transitions; confirm FSM controls application behav ior in different states
3. Thread Synchronization: test synchronization to ensure data consistency and prevent race conditions; verify the effectiveness of synchronization mechanisms between threads.
4. Queue Buffering: validate Queue Buffer implementation to mitigate data loss; assess the buffering system's ability to handle rate variations without data loss.
5. Error Handling: induce errors and confirm the application handles and reports errors; verify the presence of error recovery mechanisms for application robustness.
6. Performance Testing: confirm that we can collect data and process it without contention; monitor resource utilization for potential bottlenecks.
7. Timestamp Accuracy: ensure data is ordered during writeback; validate consistent association of timestamps with correct data points; validate the application's ability to handle parallel tasks without compromising accuracy.
8. Long-Term Stability: conduct prolonged testing to assess long-term stability; check for memory leaks, thread deadlocks, and other issues over extended usage periods.
9. Debugging: run script or connect microphone array to the laptop; configure field parameters appropriately and start the executable; monitor the flow of data through the submodules; compare single unit test cases compared to fully integrated unit test cases.

5.2 Plan for Signal-Processing Submodule (Daniel Cardosi)

As described in this report, the signal-processing submodule is fully functional with LabVIEW 2023 on a machine running MacOS, but yet untested is its compatibility with LabVIEW 2017 on Windows, a specific requirement of the clients. We plan to achieve this compatibility via a rebuild of FFTW into an archive of COFF binaries rather than of ELF binaries (i.e., an archive of extension .lib as opposed to .a) and a recompilation of the C++ source code into a Windows-native dynamic-link library (of extension .dll, as opposed to .dylib).

Additionally remaining are the ancillary tests of the various FFTW routines, for which we have already developed a framework (q.v. section 4.2 above). These tests will facilitate the selection of a maximally efficient DFT-computation routine, and their completion will mark the completion of the entire signal-processing submodule; thereafter the OBSIDAS software will require additions only to the components involved in interfacing with the client-provided hardware, with the user, and with the hard drives of the computers on which the software is to be operated, all of which are described above.

6 Budget Estimate (Aurojit Chakraborty)

Item	Description	Cost
1	LabVIEW 2017 Full	$1664
2	LabVIEW Application Builder	$518
3	LabVIEW Sound and Vibration Toolkit	$824
	Total Cost	$3006

By request of the client, our program will make use of LabView 2017 along with the following two packages: LabVIEW Application Builder and LabVIEW Sound and Vibration Toolkit. As National Instruments requires that both subscriptions and addons be purchased, Boston University covered these costs for our development purposes; however, given that Volpe is already in possession of all required licenses, our final software will necessitate no further monetary expenses on their behalf.

7 Acknowledgments

We would like to thank our clients Sophie Son, Aaron Hastings and Robert Downs from the Volpe Center for their assistance thus far, as well as our professors Alan Pisano, Osama Alshaykh and Michael Hirsch for their guidance throughout this project.

8 References

[1] R. O. Rasmussen, R. J. Bernhard, U. Sandberg, and E. P. Mun,, “The Little Book of Quieter Pavements,” *FHWA-IF-08-004*, Federal Highway Administration, Washington, DC, 2008.

[2] U. Sandberg, "Tyre/Road Noise Myths and Realities," *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, 2001.

[3] American Association of State and Highway Transportation, "Standard Method of Test for Measurement of Tire/Pavement Noise Using the On-Board Sound Intensity (OBSI) Method," AASHTO Specification TP 76-10, 2010.

9 ATTACHMENTS

9.1 Appendix 1 – Engineering Requirements

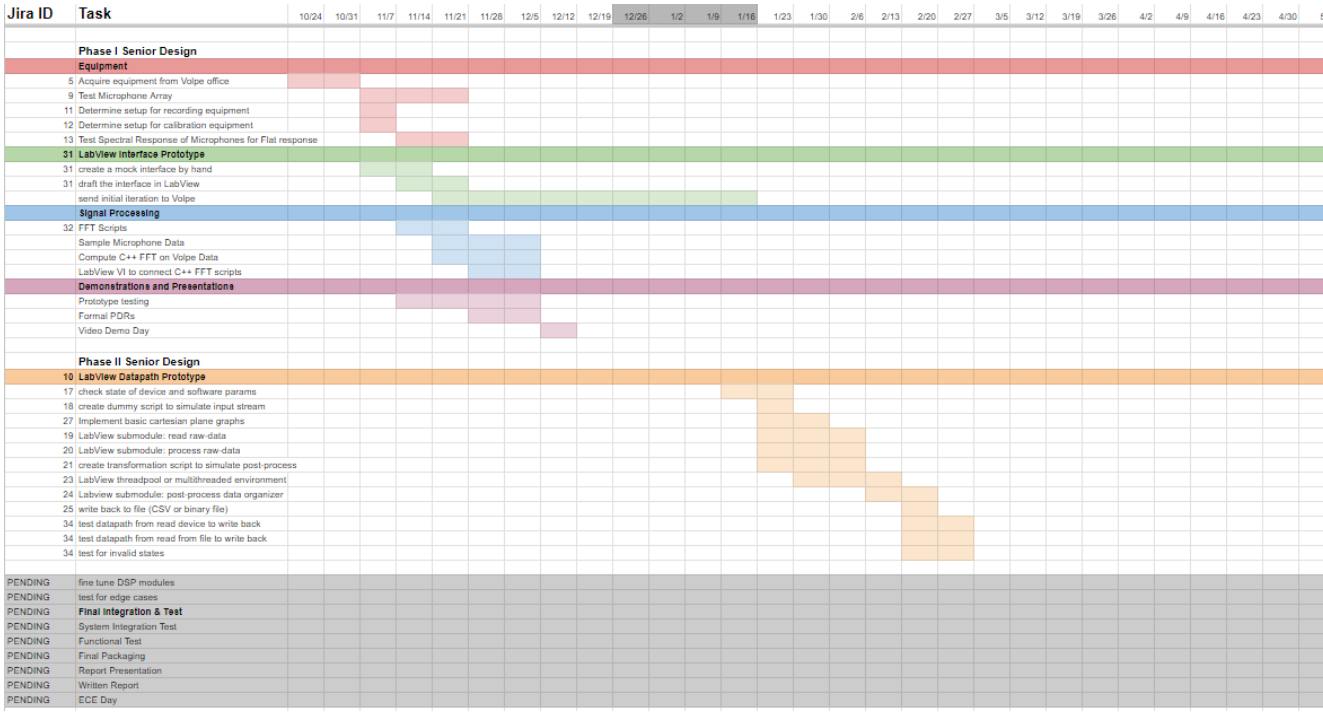
The Volpe Center has provided a hierarchical list of features and specifications, both reproduced verbatim and tabulated below.

Requirement	Value, range, tolerance, units
User-editable sensitivity-parameter inputs for microphone calibration	Precision of 1/10 mV/Pa for sensitivities
Manually triggered measurement trials of user-defined durations	Precision of 1/10 sec for durations
User-prompted data compilation and data export to a human-readable format	.CSV format for exported data
Computation of single-microphone SPLs (x4), single-probe SPLs (x2), overall SPLs (x1) in third-octave bands with centers from 250–5040 Hz; computation of single-probe SPLs (x2) and overall SPLs (x1) for entire spectrum	Precision of 1/10 dB for all SPLs
Computation of inter-microphone coherences in third-octave bands with centers from 250–5040 Hz	Precision of 1/10 (unitless) for all coherences
Program builds an LabVIEW-2017-compatible executable	N/A
Program simultaneously reads and processes data	N/A

Features:

1. The user should be able to individually calibrate the four microphones using sensitivity-parameter inputs with a precision of 1/10 mV/Pa.
2. Trials should be manually triggered, and the program should allow trial length to be pre-specified by the user, ranging from 3 to 60 seconds with a precision of 1/10 seconds.
3. On each trial, sound pressure level data should be collected for all four microphones in one-third-octave bands, with center frequencies from 250 to 5000 Hz, with a precision of 1/10 dB. Coherence data between the microphones in each probe should be computed as well, separately for each band, with a precision of 1/10 (unitless).
4. The program should use the acquired single-microphone sound pressure level data to calculate the corresponding single-probe sound intensity levels, both individually for each one-third-octave band and as a logarithmic sum of all single-band intensities. Each of these values should have a precision of 1/10 dB.
5. The computed single-probe sound-intensity-level data should be used to compute the corresponding overall data, consisting of the logarithmic average of the two single-probe intensity levels for each one-third-octave band as well as the logarithmic sum of all logarithmic averages, all computed values again having a precision of 1/10 dB.
6. The program should save all measured and computed data for each trials and store this data across multiple measurement trials.
7. The user should be able to compile and export stored multi-trial data to a human readable format (CSV is preferred), and files should be named by date and time.

9.2 Appendix 2 – Gantt Chart



Jira ID	Task
	Phase I Senior Design
	Equipment
5	Acquire equipment from Volpe office
9	Test Microphone Array
11	Determine setup for recording equipment
12	Determine setup for calibration equipment
13	Test Spectral Response of Microphones for Flat response
31	**LabView Interface Prototype**
31	create a mock interface by hand
31	draft the interface in LabView
	send initial iteration to Volpe
	Signal Processing
32	FFT Scripts
	Sample Microphone Data
	Compute C++ FFT on Volpe Data
	LabView VI to connect C++ FFT scripts
	Demonstrations and Presentations
	Prototype testing
	Formal PDRs
	Video Demo Day
	Phase II Senior Design
10	**LabView Datapath Prototype**
17	check state of device and software params
18	create dummy script to simulate input stream
27	Implement basic cartesian plane graphs
19	LabView submodule: read raw-data
20	LabView submodule: process raw-data
21	create transformation script to simulate post-process
23	LabView threadpool or multithreaded environment
24	Labview submodule: post-process data organizer
25	write back to file (CSV or binary file)
34	test datapath from read device to write back
34	test datapath from read from file to write back
34	test for invalid states
PENDING	fine tune DSP modules
PENDING	test for edge cases
PENDING	**Final Integration & Test**
PENDING	System Integration Test
PENDING	Functional Test
PENDING	Final Packaging
PENDING	Report Presentation
PENDING	Written Report
PENDING	ECE Day