Available online at www.eccomasproceedia.org
Eccomas Proceedia COMPDYN (2023) 4960-4971

COMPDYN 2023
9th ECCOMAS Thematic Conference on
Computational Methods in Structural Dynamics and Earthquake Engineering
M. Papadrakakis, M. Fragiadakis (eds.)
Athens, Greece, 12-14 June 2023

MODULAR DEPLOYMENT OF MICROPROCESSOR-CONTROLLED DATA ACQUISITION AND CONTROL WITHIN A DIGITAL TWIN

Matthew Bonney[1], **Matthew Tipuric**[1], **David Wagg**[1], **Ian Stothers**[2]

[1] Dynamics Research Group
The University of Sheffield, UK
e-mail: {m.bonney,m.tipuric,david.wagg}@sheffield.ac.uk

[2] St Others Ltd. UK
e-mail: st.others.ltd@gmail.com

Abstract. *One major component of building management is the understanding of the current state of the system, typically using sensor measurements. To achieve this, many building managers are interested in the deployment of a digital twin for each building that allows for remote inquiries. However, many commercial data acquisition systems use proprietary connections or data formats putting a large barrier for interoperability between sensors and digital twins. To remove this barrier, this work develops a transparent data acquisition system using microprocessors and the remote connection to this system through a python-based remote digital twin operational platform. The use of microprocessors allows for nearly complete control of the data processing from the analogue to digital converter, data sampling, and data recording. There are many possibilities to process this data, so this paper focuses on identifying the various possibilities and demystifying this computer science topics for engineers. To access this data, a remote connection allows for multiple users to simultaneously access the current operational data. Using an interface, such as the digital twin operational platform, reduces the required knowledge to promote a multi-disciplinary culture through shared data within the digital twin.*

Keywords: Digital Twins, Internet of Things, Open-Source, Microprocessor

1 Introduction

Recently, the concept of a Digital Twin (DT) has been introduced to a wide variety of industries and applications. Through this introduction, there have been many interpretations on what a DT is and what it does [1]. As of late 2020, there has been more than 30 definitions for what a DT is [2], and it is expected that there are many more since that specific literature review due to the rapid increase in popularity in both research and industry for DTs. Among the myriad of definitions, one of the most common concepts is the connectivity between the physical and digital twins. There are multiple methodologies that have been applied to establish this connectivity, as well as the meaning of "connected". The two main methodologies are: 1) connection via access to embedded sensors and industrial Internet of Things (IoT) [3, 4] or, 2) connection via a collection of data-sets obtained during regular intervals. Selection of the connectivity methodology is highly related to the application, security, industry, and systems.

An important component of the twinning concept is the multi-disciplinary nature of systems. Particularly when using IoT to establish connectivity, the incorporation of hardware, software, human factors, and decision making promotes highly collaborative and multi-disciplinary understanding of a system [5]. IoT refers to a network of sensors and computing to provide useful data to better understand the connected physical system(s). This connectivity is done typically through networking (such as Ethernet or WiFi) due to the difficulties associated with mechanical safety, physical distance between sensors/systems and users, and user ease-of-use.

Due to the complexity of this connectivity, IoT platforms tend to be closed-source and/or proprietary in nature. Some common commercial products include cloud data-services (e.g. Google Cloud, AWS, and Microsoft Azure) or low-code solutions (e.g. Blynk IoT, Particle, and ThingWorx). Due to this obfuscation of information, it is difficult to fully understand the processes being used, thus declining the level of trust in the results presented. In order to increase the trust in both the IoT platform and the DT, an open-source methodology is proposed. To demonstrate this methodology, this work discusses some of the various options for developing an IoT platform including both the software and hardware components of the setup.

2 Software Setup

The development of an IoT platform (specifically within a DT) is comprised of both software and hardware components. This section discusses several of the software components including:

- Network connectivity options
- Data access options
- User access options

While some of these components seem similar, there are some slight differences between these topics that require some specific discussions. The network connectivity options refer to protocols that the sensors are connected to the sensor network, the data access options refers to how the data is managed along the network, and the user access options refer to how the user accesses this network of information through the DT.

2.1 Network Connectivity Options

The methods of communication between devices (sensors primarily in this case) is dedicated into two main categories, TCP/IP and UDP. For the majority of uses, it is common to use

Transmission Control Protocol and Internet Protocol (TCP/IP). TCP is used for general internet access and is the most commonly used protocol for secure networking. The other major alternative is User Datagram Protocol (UDP), commonly used for fast networking. While the discussion here is based on the author's experience, the reader is referred to [6] and similar papers for a quantitative comparison between the protocols.

Although the same data can be transferred through either protocol, there are some significant differences between the protocols. A summarized list is shown in Table 1, but there are some specific differences to highlight. The first difference is the reliability of the transfer of data. For TCP, there is a "handshake" (a communication between devices for verification) to ensure that the requested data is sent and received. Once the requester has received the data, a signal is sent back to the source to verify the transfer. However for UDP, there is no handshake, so once the requester has established a connection, it purely receives the data. If there is a disconnection, the UDP protocol purely loses the data that was sent during the disconnect while TCP will ensure that all the data is received. Because of this handshake, there is extra latency when using the TCP protocol causing it to be a slower transfer speed. For most general internet usage, this latency is not perceivable, but can be noticed for real-time applications as jitter or lag.

Table 1: Main differences between TCP and UDP protocols

TCP	**UDP**
Secure	Unsecured
Slow	Fast
Guaranteed Transmission	No Guarantee
Critical Applications	Real-Time Applications
Packet Reorder Mechanism	No Reorder Mechanism
Advanced Error Checking	Basic Error Checking
20 Byte Header	8 Byte Header
Three-Way Handshake	No Handshake

The majority of the differences derive from the inclusion of a handshake protocol in the TCP protocol. This handshake makes the TCP protocol a three part transfer (request - send - receipt), while the UDP protocol is only a two part transfer (request - send). In addition to the handshake, TCP also introduces a larger header file attached to the transferred data. This means that for a singular piece of data, the packet for TCP, which paired with the handshake, is larger and slower than a comparable UDP packet for the same piece of data.

2.2 Data Access Options

The data access options commonly refer to the communication between machines. This communication can occur from sensor to sensor or from sensor to a centralised "broker" (also commonly called a server) such as a digital twin. Often, this communication is also referred to as messaging protocols [7]. While this section will give a general overview, it is not an extensive review of every messaging protocol developed.

One of the basic forms of messaging is the Constrained Application Protocol (CoAP). This messaging establishes a synchronous request/response direct connection between machines. Originally, this was established for tiny devices with low power to utilise REpresentational State Transfer (REST) operations. CoAP is a binary protocol that runs over UDP thus providing a direct connection with the least amount of lag possible. While this connection is quick, since it uses UDP, there is an issue with scalability. Since there is a direct UDP connection, the user must establish a connection to each machine, which has a limit for the number of sensors it can access.

One methodology to increase the scalability is to implement Message Queue Telemetry Transport (MQTT) protocol. This establishes an asynchronous publish/subscribe protocol that runs using the TCP[1] stack to the user. MQTT establishes a broker to facilitate connections between users and sensors. Since the user only have to interact with the broker, there is only one permanent connection, while there can be any number of connections between the broker and the sensors. Due to the broker setup, there is the ability to establish multiple queues and human-readable organisation of sensor information. One example would be to establish queues for similar sensors (accelerometer, temperature probes, etc.) or sensors that a physically nearby (a singular floor, specific cluster, etc.). MQTT excels on the ability for expansion, it's light-weight requirements, and its human-readable interactions/queues.

MQTT works well for establishing connections on dependable networks. However, as commonly implemented, if the network is disconnected the data that would be sent during this down-time would normally be lost. To address this issue, the Advanced Message Queuing Protocol (AMQP) has been developed, mainly by the financial industry [7]. The main advantage that this protocol introduces is a "store-and-forward" feature to account for network disruptions. While this feature provides robustness for network disruptions, it also introduces a much heaver load on the broker system compared to MQTT. This is similar to establishing a database buffer between the sensors and the broker. To summarise these various options, Table 2 provides a simple comparison between options.

Table 2: Summary of basic data access options

CoAP	MQTT	AMQP
Direct Connect	Broker	Broker
UDP	TCP or UDP	TCP
Header-less	Header	Store-and-forward
Fastest	Fast	Stored history

2.3 User Access Options

The last component for consideration is the platform for which the user access the data from the sensors. In general there are 3 main types of accessibility: high-code, low-code, and no-code. Each type requires a varying level of expertise from the user in order to operate. To summarise these options, Figure 1 gives a better understanding of the options.

High-code represents the modifying of the source code in order to communicate. This is typically the starting point of any program or setup and requires a high level of expertise is order to operate. Due to this required expertise, it is expected that the user will be the same (or similar) developer. Using a high-code option provides a high level of granular control, but greatly reduces productivity since the developer will have to search for specific lines of code in order to make any modifications.

To expand the desired user-base, the low-code option increases the ease-of-use for users. This is targeted towards tech user, or users with application expertise and light programming

[1]Some lightweight implementations of MQTT do use UDP, such as MQTT/UDP and MQTT-SN. These implementations are not particularly commonly used.

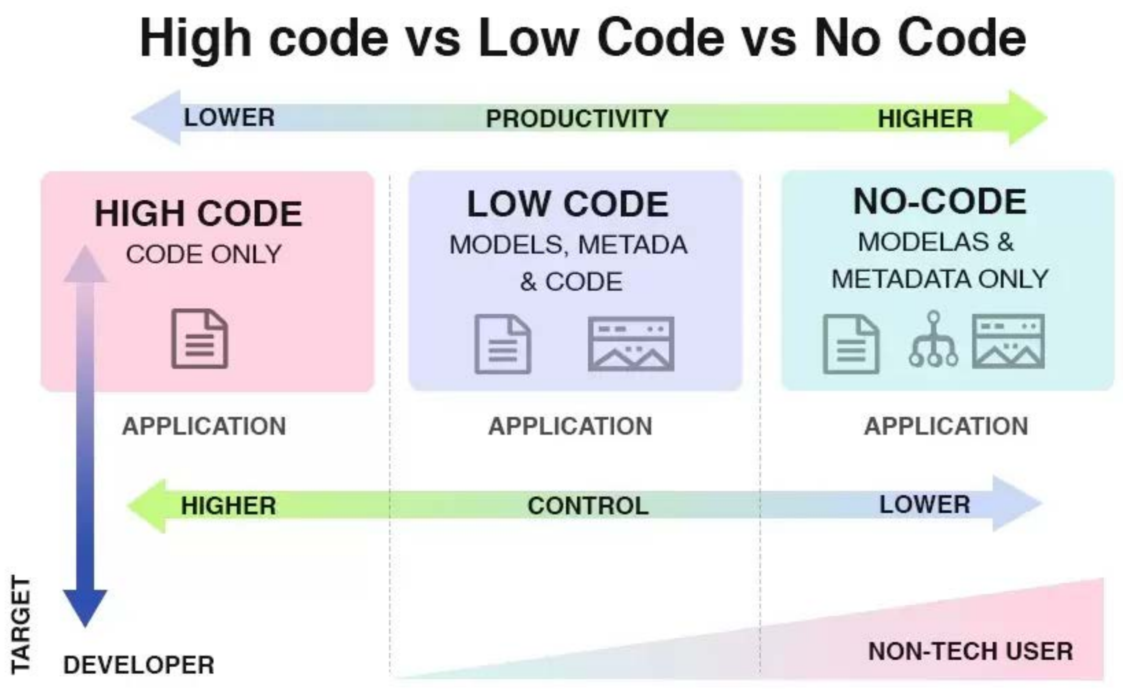
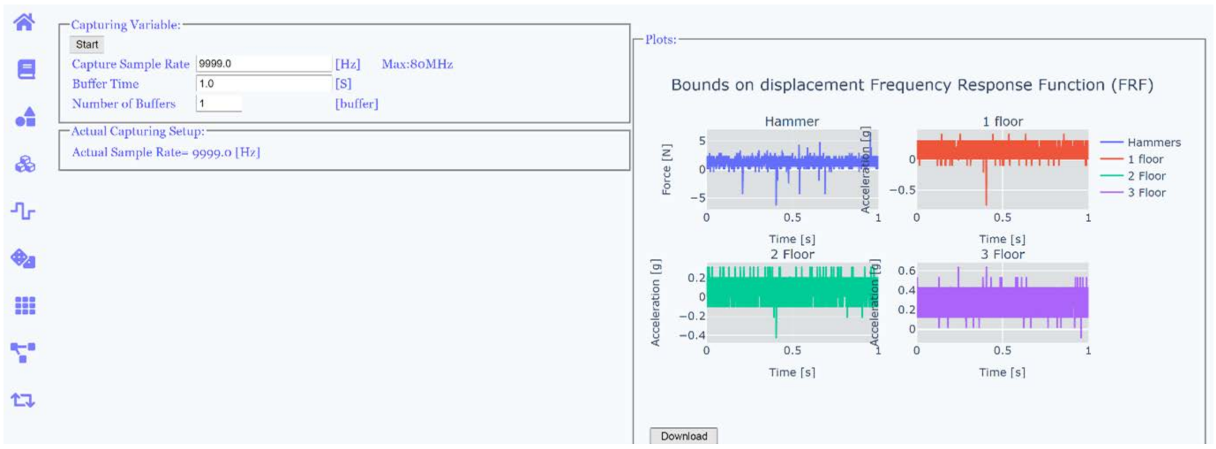


Figure 1: Summary of user access options. Taken from [8]

knowledge. For example, if the IoT sensors are attached to a manufacturing robot, the expected user for the low-code option would be the mechanical engineers or technicians. This accessibility option uses specifically made techniques/functions/classes to allow users to have access to functions/techniques without the need to modify source code. In addition to this ease-of-use, this option is also very helpful for other aspects such as collaborative working and version controlling. If there is a change in the algorithm implementation (typically for reduced computational effort), the end-user's code does not need any modifications, just a simple update of the package. For many engineering applications, this is the furthest that most platforms will obtain since the expected users will almost always be technical experts.

While low-code is typically the end-point for most engineering applications, the barrier for entry is till too high for many users and decision makers, making it difficult to market to a general audience. For these users, a no-code option is required. This reduces the required expertise of the user to a general, possibly trained, person. The wide-spread nature of the user-base, with limited access to source code, greatly benefits commercial interests in the development of software and platforms. This commonly reduces the user's experience to simple drag-down menus, button selects, and graphs.

One example of a no-code solution that also is open-source is the Digital Twin Operational Platform (DTOP) Cristallo [9] and other DTOPs created by the authors [10, 11]. This uses a python back-end and HTML front-end to reduce the knowledge burden on users to operate the digital twin. A demonstration of how this DTOP system can be used in a IoT context can be found in [12] where the DTOP system was used to activate and collect sensor readings though a direct connection. The user-interface for this access is shown in Figure 2. This interface allows the user to input the simple parameters such as sample rate and the measurement time. The user simply has to input these values into a HTML page and click "start". The DT will then activate the data acquisition system to collect the data and display it to the user. Additionally, the user can download this data for further analysis.


Capturing Variable:
Start
Capture Sample Rate 9999.0 [Hz] Max:80MHz
Buffer Time 1.0 [S]
Number of Buffers 1 [buffer]

Actual Capturing Setup:
Actual Sample Rate= 9999.0 [Hz]

Figure 2: DTOP user interface for demonstration IoT interaction. Taken from [12]

3 Hardware Setup

The other major component of an IoT platform is the hardware physically connected to the system. This connection is done through various sensors such as accelerometers, temperature probes, pitot tubes, strain gauges, and a multitude of other sensors. The output of these sensors can vary, but generally fall into two main categories: voltage or current output. Using a voltage reading is common for non-intrusive electronics or systems that are sensitive to electronic coupling. For mechanical systems, sensors like accelerometers commonly use voltage outputs via piezoelectric components that create an electrical voltage when under mechanical strain. One common issue with voltage-based sensors is the voltage loss when the signal is being transported on long wires. In contrast, current-based sensors have a variable resistance across a constant voltage to relate to the output. Commonly, these sensors are limited to the current range of $4-20$mA where an output of 4mA refers to the zero point of calibration and 20mA corresponds to the maximum value recordable.

The selection of sensors is highly dependent on the system, quantity of interest, accessibility, and cost. With the selection of the sensors, the Data AcQuisition (DAQ) system must also be designed. There are many commercial options for these DAQ systems, but tend to require proprietary software packages in order to operate these systems commonly with low- or no-code interfaces. However, some provide drivers for high-code incorporation into other frameworks. The demonstration in Figure 2 is one example of this type of setup. This work aims to implement a microprocessor-controlled network of DAQ systems to connect the physical and digital twins together in an open-source and modular framework. This modularity is especially important when considering the multi-disciplinary nature of complex engineering systems and the variety of sensors required to measure these quantities.

3.1 Microprocessor Controllers

In contrast to the proprietary nature of commercial DAQ systems, one alternative (both technically and financially) is the utilisation of microprocessors to interface between raw sensor readings (mV or mA) and human-readable data (in units such as g, C, etc.). The inclusion of microprocessors allow for the embedding of meta-data into the data and supplying any power or signal requirements. While the actions performed is not limited to microprocessors, these provide a flexibility not available for commercial solutions. While there are many different types of microcontroller available (Singh et al. [13] provide a comparison of some of the boards available in 2020), two makes will be discussed here for comparison. These are STMicroelectronics (STM) and Raspberry

STMicroelectronics is a global leader is semiconductor manufacturing. They produce a while variety of processors and associated circuitry for various uses in systems. These processors run using either C or C++ programming for hardware-level programming. This provides the quickest operation of programming available. Some of the most commonly used STM development boards are the black-pill (STM32F411) and the more expensive Nucleo range. These boards excel for high speeds at low cost, but do require a higher degree of technical knowledge than some competitors.

As an alternative, Raspberry is a UK-based company focusing on ease-of-use and accessibility. Commonly, these are used to teach programming at schools; this shows one of the advantages the Raspberry has compared to STM, ease-of-use. This focus on ease-of-use and teaching means that there are a large amount of resources available for new designers and open-source low-code packages already developed. Raspberry Pis (processors created by Raspberry) run using a lightweight build of python call micro-python or C. The default inclusion of python provides easier to read programming such that non-experts can understand the process flow and grow the inherent trust in the processing. Another advantage is the capability for many of the processors to have an operating system installed with a graphical user interface. This is an adaptation of Linux and several processors have the capability of running Linux directly. Some of the common controllers include Raspberry Pi 4 (with operating system) and Raspberry Pi Pico (without operating system and is similar to STM black-pill).

4 Example Implementation Demonstration

This section discusses one possible implementation for a Building Information Management Digital Twin (BIM-DT) and an example messaging system on hardware. This is selected as an example due to the large scale nature of buildings and the variety of sensors needed to create a DT (to measure aspects such as earthquake vibrations, room conditions such as temperature, and energy usage for efficiency and certification). This section contains two subsections, Section 4.1 gives a possible architecture design from sensors to digital twin to user and Section 4.2 shows a demonstrative example messaging system using CoAP.

4.1 Example Architecture

The example architecture for this setup is shown on a scaled 3-storey structure (used in DTOP-Cristallo) in Figure 3. This contains the entire design from sensors, to server, to user.

In Figure 3, the teal squares represents the microprocessors proposed for this setup. These are Raspberry Pi Pico W that cost less than £10 each and has both wired and wireless connectivity options. These Picos are attached by wires to three different types of sensors via the general purpose input/output pins. The black sensors are accelerometers to measure seismic activity and wind induced vibrations; the red sensors are temperature sensors that tie into the climate control system, and the green sensors are occupancy/motion sensors to identify if someone is using the space and adjust the lights accordingly. Since this system has multiple controllers, a MQTT broker is setup on a Raspberry Pi3 with wireless connections to each controller. The W variant of the Pico has the WiFi connectivity that reduces the amount of wires needed to be sent

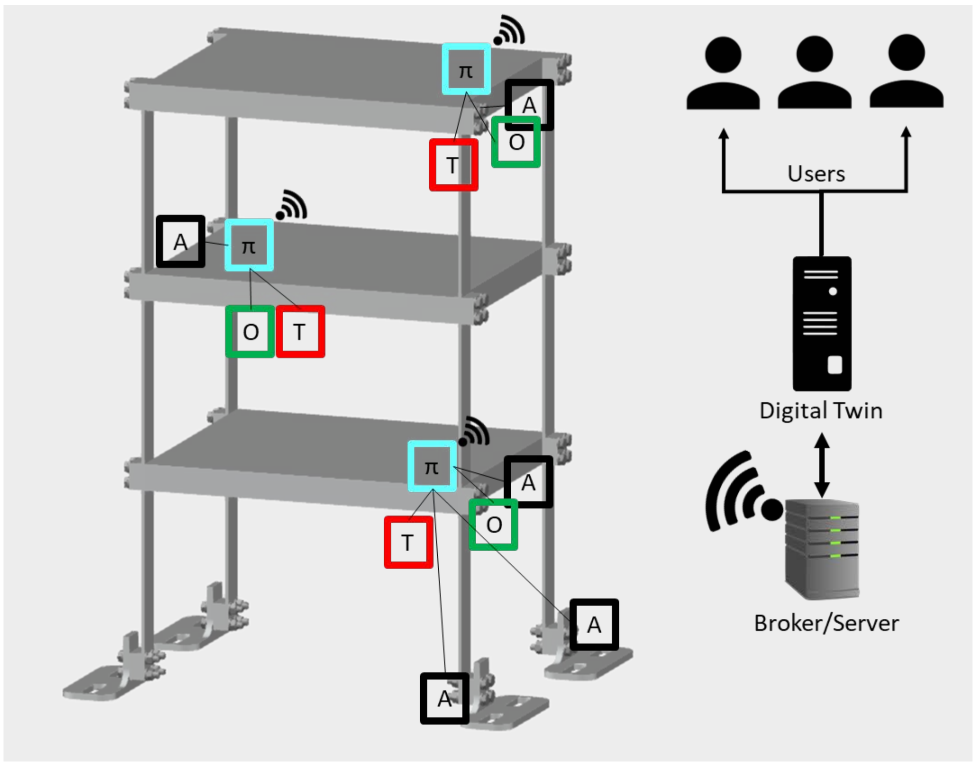


Figure 3: Example sensor layout for building management. Teal are microprocessors, black are accelerometers, red are temperature sensors, and green are occupancy/motion sensors.

throughout the building and reducing the voltage loss due to long cabling. This also allows for more flexibility when deploying new sensors or redesigning each floor (for example changing the amount of rooms on a floor).

With a centralised server using MQTT, the DT can access this directly and gain access to each and every sensor of interest. This can be used to directly control aspects of the building operation such as lighting and climate control in "real-enough" time (lights operate in less than a half second and climate control on the scale of seconds). This control can be done automatically through algorithms or manual through the user interaction with the DT. For more time critical control, such as for emergency systems, local control could be achieved on the relevant Picos. The Raspberry Pi Pico can run the control logic on the inbuilt microprocessor to reduce overall processing time, while still providing override or monitoring information to the DT. This use of local control algorithm would be required for any active seismic control mechanisms implemented within the building due to the need to respond quickly to sudden earthquakes, for example.

4.2 Octave-based CoAP

To show an example implementation of a messaging system on hardware, this section discusses a CoAP messaging protocol with a UDP connection to a STM microprocessor (black-pill). There are 5 sensors used and are piezoelectric sensors placed along the length of a long slender beam, similar to the support pillar from the scaled 3-storey structure in Figure 3. This setup is seen in Figure 4 with the circles on the beam being the piezoelectric sensors, attached to the STM black-pill (outlined in red) with an external Ethernet port (outlined in black) that is connected to the user's computer.

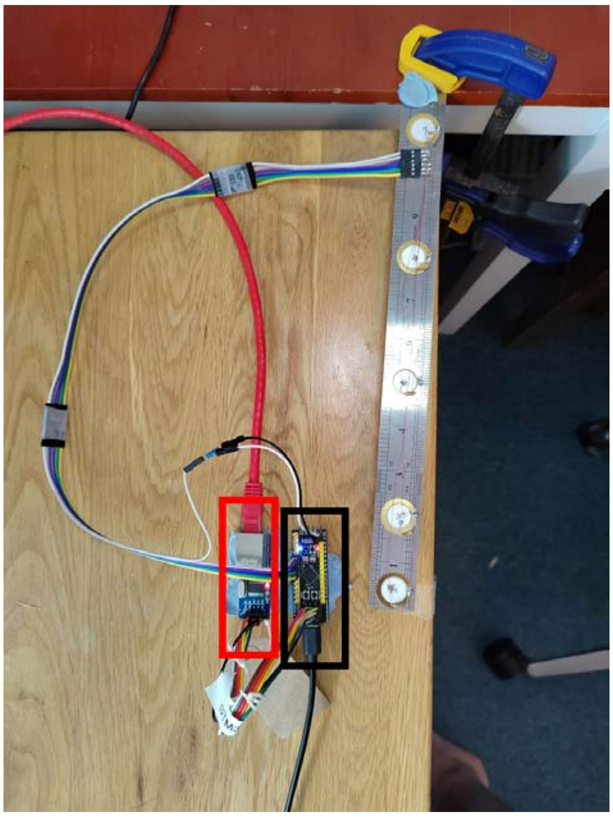


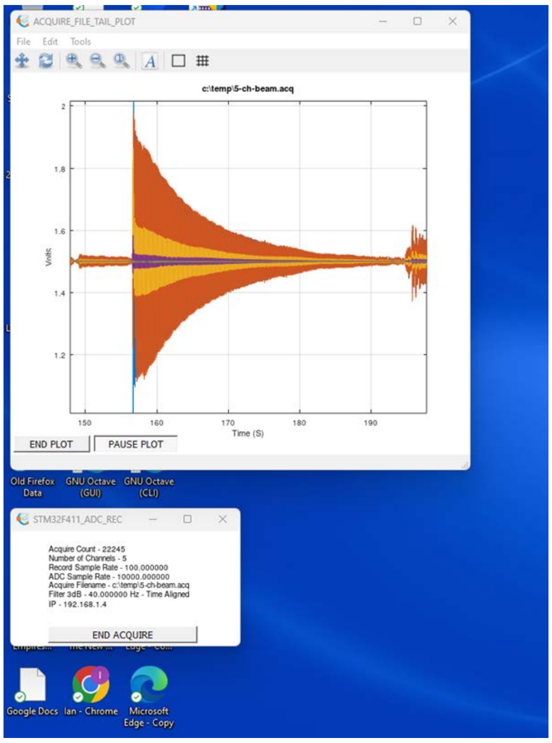
Figure 4: Physical setup for STM-based CoAP connectivity, with the STM black-pill outlined in black and the external ethernet port outlined in red

The piezoelectric sensors take the mechanical deformation to produce a voltage that is read by the STM black-pill. These sensors are common in building health monitoring due to the physically lightweight materials and relative inexpensive cost (under £1 each). Overall, this entire hardware system costs less than £20.

In order for the user to access this data, a CoAP messaging protocol is implemented using a mixture of C for the hardware processing and Octave (similar to Matlab) for the software processing. For the hardware, the board is programmed to take the appropriate signals to generate digital signals and establish the scheduler to facility the CoAP messaging. The processing and visualisation is performed via software on the user's computer via purely Octave programming. As a demonstration of a seismic shock, the base is excited through an impact of the table that the base is attached to and the Octave results in shown in Figure 5.

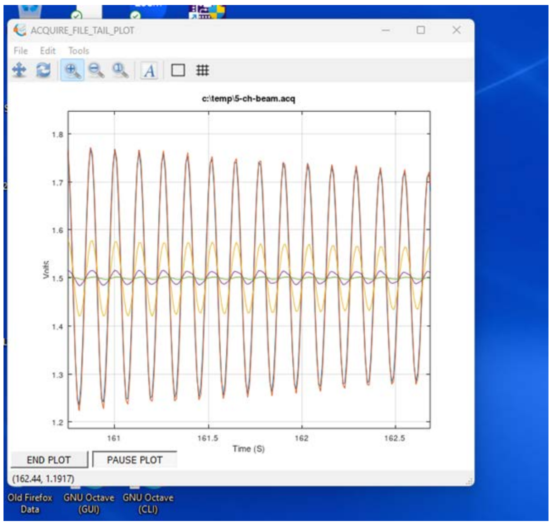
This implementation CoAP involves two main portions, the gathering and the visualisation of the data packet. In Figure 5a, the bottom window is active collection of data and storing it locally via a binary file. This collection is continuous until the user clicks "END ACQUIRE" that terminates this connection. One interesting aspect of this implementation is the use of a scheduler that allows for multiple users to create UDP stack connections for retrieving the data. In this example, the computer is connected directly to the microprocessor (see Figure 4) so only one user can connect at a time. However, if this microprocessor was connected to a router, then multiple people would be able to connect and access this data simultaneously.

The second window in Figure 5a (the top) shows a graphical representation of the sensor data. This shows a selected time section from the streaming binary file being written to in the first window. One advantage of this binary file is the ability to do quick (non-sequential) search-



```
Acquire Count - 22245
Number of Channels - 5
Record Sample Rate - 100.000000
ADC Sample Rate - 10000.000000
Acquire Filename - c:\temp\5-ch-beam.acq
Filter 3dB - 40.000000 Hz - Time Aligned
IP - 192.168.1.4
```

(a) Ring down data for all sensors



(b) Zoomed in graph to visualize sensor differences

Figure 5: Octave output of CoAP messaging

ing with limited loading that is not available in ASCII files such as CSV format. Because this visualisation is purely reading a local binary file, the user can select what time limits and channels to investigate. Figure 5b shows a more detailed view of the response of the sensors. This also shows the various sensors and the relative amplitudes of vibration caused by the geometric location of the sensors. While these plots only shows the data in voltage, the microprocessor can also inject meta-data into the signal to denote the units of measurement, physical location, calibration information, and other pieces of data such that the data received by the user is self-describing.

5 Remarks

During the course of this work, there was a multitude of knowledge gained through the investigation of multi-disciplinary aspects of IoT-based digital twins. It was found that there any many solutions to the same problem, for example the hardware selection between STM and Raspberry. This variety of solutions made the collection of information difficult, thus this paper has focused on the distillation of information gathered for a mechanical/industrial engineering audience. The first example shows a planned designed system from sensor, to server, to digital twin, to user, while the second example shows an implemented CoAP connection from a STM microprocessor to a low-code environment within Octave.

Future work by the authors will include a physical demonstration of the architecture proposed in Section 4.1. This will include the use of various sensors, multiple controllers, a centralised MQTT server, and a DTOP for user interactivity with the various sensors and data. To aid in the open-source dissemination and demystification of these platforms, the software component will be publicly available via GitHub when the finalised journal paper is published.

Acknowledgement

The authors would like to acknowledge the support of EPSRC via the DigiTwin project (grant number EP/R006768/1). The authors would also like to specially thank the coauthor Ian Stothers for the open-source, commercial demonstration free-of-charge for Section 4.2.

REFERENCES

[1] D. J. Wagg, K. Worden, R. J. Barthorpe, and P. Gardner. Digital Twins: State-of-the-Art and Future Directions for Modeling and Simulation in Engineering Dynamics Applications. *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, 6(3), 05 2020. 030901.

[2] C. Semeraro, M. Lezoche, H. Panetto, and M. Dassisti. Digital twin paradigm: A systematic literature review. *Computers in Industry*, 130:103469, 2021.

[3] J. Lee, E. Lapira, B. Bagheri, and H. Kao. Recent advances and trends in predictive manufacturing systems in big data environment. *Manufacturing Letters*, 1(1):38 – 41, 2013.

[4] T. H.-J. Uhlemann, C. Schock, C. Lehmann, S. Freiberger, and R. Steinhilper. The digital twin: Demonstrating the potential of real time data acquisition in production systems. *Procedia Manufacturing*, 9:113 – 120, 2017. 7th Conference on Learning Factories, CLF 2017.

[5] M. S. Bonney, M. de Angelis, M. Dal Borgo, and D. J. Wagg. Contextualisation of information in digital twin processes. *Mechanical Systems and Signal Processing*, 184:109657, 2023.

[6] F. T. AL-Dhief, N. Sabri, N. M. A. Latiff, N. N. N. A Malik, M. A. A. Albader, M. A. Mohammed, R. N. AL-Haddad, Y. D. Salman, M. K. A. Ghani, and O. I. Obaid. Performance comparison between TCP and UDP protocols in different simulation scenarios. *International Journal of Engineering & Technology*, 7(4.36):172–176, 2018.

[7] T. M. Tukade and R. Banakar. Data transfer protocols in IoT-an overview. *International Journal of Pure and Applied Mathematics*, 118:121–138, 01 2018.

[8] iTech India. Will low code/no-code solutions overtake custom software?, March 2022.

[9] M. S. Bonney, M. de Angelis, M. Dal Borgo, L. Andrade, S. Beregi, N. Jamia, and D. J. Wagg. Development of a digital twin operational platform using python flask. *Data-Centric Engineering*, 3:e1, 2022.

[10] M. S. Bonney, P. Gardner, D. Wagg, and R. Mills. Case study of connectivity of digital twins and experimental systems. In *Proceedings of the 8th International Conference on Computational Methods in Structural Dynamics and Earthquake Engineering*, pages 1416–1425, Streamed from Athens, Greece, June 2021.

[11] M. S. Bonney, M. de Angelis, M. Dal Borgo, and D. Wagg. Digital twin operational platform for connectivity and accessibility using Flask Python. In *Proceedings of the 24th International Conference on Model-Driven Engineering Languages and Systems*, 2021.

[12] R. Wang and M. S. Bonney. Novel data acquisition utilising a flask python digital twin operational platform. In Matt Allen, Sheyda Davaria, and R. Benjamin Davis, editors, *Special Topics in Structural Dynamics & Experimental Techniques, Volume 5*, pages 7–13, Cham, 2023. Springer International Publishing.

[13] D. Singh, A. Sandhu, A. Thakur, and N. Priyank. An overview of iot hardware development platforms. *International Journal of Emerging Technolgy*, 11:155–163, 2020.