EXPLORING A SUPERVISORY CONTROL SYSTEM USING ROS2 AND IOT SENSORS

M. A. Roach[1], J. Penney[1], B. H. Jared[1]

[1]Mechanical, Aerospace, and Biomedical Engineering, University of Tennessee, Knoxville, TN 37996

Abstract

Whether collecting data from process monitoring sensors or controlling a system of multiple actuators and electrical systems, a powerful supervisory control system must be developed for additive manufacturing (AM) systems. The Robot Operating System version 2 (ROS2) is a set of software libraries that can be used to control robotics systems and has tools for sensor value publishing. This research project is exploring the use of computational nodes connected to process monitoring sensors and robotic or electrical systems to allow for a more in-depth knowledge of the system health and process as well as open the possibilities of process control. These nodes can be connected and controlled by the ROS2 architecture. Work will be discussed exploring the reliability and speed of common AM processes and sensors such as robot controllers and thermal monitoring.

Background

When switching from traditionally cast metals to AM created metals, the properties of the metal are no longer guaranteed and may not be homogenous throughout the volume of the material. Due to this downside in AM, the need for process monitoring is now formed, as a method of watching signals and changes during the metal deposition process can be beneficial and may indicate process conditions have deviated significantly enough to allow a defect to form in the material.

In this research, the AM methods of interest are wire-fed directed energy deposition (DED) such as wire-arc AM. DED is traditionally done by programming 2- and 3-dimensional paths which follow sequential slices of a pre-defined geometry on top of a 2- and 3-dimentional substrate. These paths are then used by robots or CNC equipment to manipulate a deposition head along the path or manipulating the build area about the deposition head along the pre-defined path. The deposition head consists of an energy source to melt the material with a wire-feeding mechanism to supply the melt pool with new material for the purpose of deposition.

While this paper does not focus on a specific DED setup or system, the methods and results are applicable not only to many DED systems but also many other AM systems in general. The reason for this is that many AM systems rely on similar sensors to monitor the deposition process. Such sensors can include thermal monitoring, robot or CNC equipment programmed and actual position monitoring, 2- or 3-dimensional imaging sensors, and power monitoring. There exist many methods for AM system controls, collection of sensor information, open- and closed-loop control, as well as path planning to allow the workflow from pre-defined geometry to final part. In this paper, the specific system under investigation is defined in the remainder of this section.

This specific system consists of a KUKA KR6-2 robotic manipulator with a Fronius TPS 4000 CMT Advanced robotic weld head. This welding head uses the pre-defined welding profiles generated by Fronius and are executed by the KUKA ArcTech package in job mode. The robot path is generated using Octopuz path planning and robot simulation software where the final files (.src and .dat) are copied over to the KUKA controller and run by the operator to print the desired geometry. During the printing process, a manual pause is added between each sequential layer to prevent the heat in the part from building up. This is normally done in one of two ways; either a 120 second delay is manually coded into the path plan using Octopuz, or the program is manually paused by the operator after each layer to allow for a manual temperature measurement using a handheld non-contact IR thermometer to monitor the temperature of the deposited material until it passes a pre-

defined threshold. This threshold varies for different materials and is not standardized. The following sections will explore the use of ROS2 and IoT sensors to work towards the automated thermal monitoring and path plan resuming without the need for operator intervention.

This system is a development system; therefore, it integrates many different sensors and process monitoring equipment. This suite of sensors is always changing based on research needs, but for this research the configuration is as follows. The robot is monitored using the KUKA.RobotSensorInterface (RSI)[1] package which reports back current tool positions as well as error dialogue and robot status. This is connected to the existing Windows 10-based control computer via an ad-hoc ethernet network. Also connected to this network are two cameras, a FLIR A50 thermal camera and a XIRIS XIR-1800 welding camera. These cameras watch the bulk temperature distribution across the deposited material and the arc and melt pool dynamics and temperature, respectively. Finally, the remainder of the sensors are currently integrated with a custom LabVIEW VI that includes 4 k-type thermocouples, welding voltage and current, and the previously mentioned RSI. The two cameras listed above are controlled by their own software and do not integrate with the LabVIEW system.

Issues have occurred frequently with the maintenance of the LabVIEW controller VI that include driver mismatches, the lack of timestamp recording for time-series measurements, and the ability to easily add such a feature into the existing VI. Due to the way in which the VI is executed, the sensor acquisition and control is collected on a time-defined loop and lacks the ability for a more intelligent data collection method. Along with these issues, the need to purchase LabVIEW licenses and National Instruments equipment quickly increases the cost of development of these systems. Especially when a specific sensor, such as the welding camera, does not have existing compatibility with LabVIEW. ROS2 will be explored to determine if it can solve these problems and still deliver adequate performance.

ROS2 Introduction

As their website states “ROS (Robot Operating System) is an open-source software development kit for robotics applications. ROS offers a standard software platform to developers across industries that will carry them from research and prototyping all the way through to deployment and production.”[2] This software has been used in many AM systems at many institutions and has proven to be a valuable tool in this research space. The ROS ecosystem advertises 8 main benefits on their website and some of these benefits were seen directly with this research. These benefits include a substantially large and active global community of developers, a continued and proven use in systems spanning single-student projects to large-scale collaborations and competitions, customizable and shorter time to market, multi-domain due to the generality of the software model, multiplatform so it runs on systems from Mac and PC to Linux and embedded platforms and numerous programming languages from C++ and Python to MATLAB, completely open source, non-restrictive Apache 2.0 license, and finally industrial support specifically from ROS2[3]. While not all these claims can be verified in this research due to the limited scope and timeline, future work as well as existing ROS-based projects continue to prove its viability.

ROS2 comes in two major versions, ROS and ROS2. ROS2 is the newest version and focuses on the industrialization of the software framework. ROS and ROS2 operate individually but can still communicate via the ROS1 to ROS2 bridge. This is especially helpful when existing packages and libraries have not yet been migrated by their developers. This research will focus entirely on ROS2.

The ROS2 environment is composed of discrete elements, all with a defined task such as data acquisition for one sensor or a decision-making algorithm output. To begin, all work is done within a Workspace[4]. This is a simple folder with defined subfolders like src for organizing your system as a whole. For example, a complete AM system would operate within one workspace if only one computer was used. The next organizational element is a Package[5]; these are the backbone of ROS2 and its modularity. Packages are collections of executable scripts and resources to control a specific entity like a sensor, robot, or specific system, for example existing packages can be used to interface with the KUKA KR6-2 and RSI. Within these

organizational elements and executable scripts, the main ROS2 elements are called and run to operate the target system.

The most important element is a Node[6], these are responsible for a singular and modular purpose, for example collecting data from a sensor, or conducting computations or machine learning classifications. Usually, each Node is initialized by a single executable file, like a Python script or compiled C++ executable, using object-oriented programming. Similarly multiple Nodes can be initialized in a single executable file. The next element is the Topic[7]. These are how data is moved between Nodes. These are defined within the Nodes to publish and subscribe to specific Topics. A simple example is shown below in Figure 1, where three nodes are running and publishing or subscribing to one Topic by sending and receiving the Message. A fourth element is shown called “ros2 topic echo” which is an example of using the command line interface ability of ROS to print the published Topic Messages to the terminal.

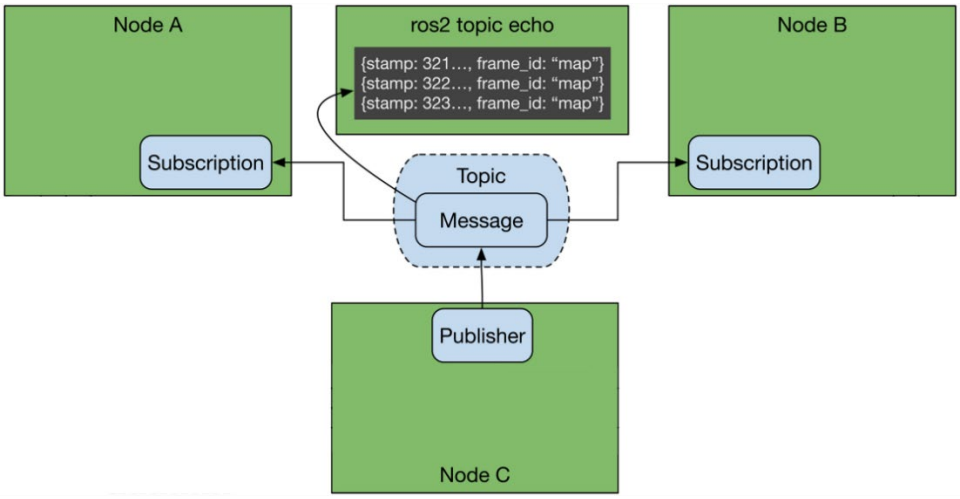

Figure 1: Basic ROS2 structure[8]

More advanced capabilities exist in ROS2 that expand the functionality to cover nearly all actions; this is done using Services[9] and Actions[10]. Services provide data when they are specifically called by a Node. These can be used for actions like checking an enabling signal before an operation is executed, but the enabling signal does not need to be published continually to rest of the system. Finally, an Action is a system task with a goal, feedback, and a result. Using an Action is helpful when the system must do something such as move the tool to a pre-defined location; this action will send the target coordinates, return the coordinates as the tool is moved, then the final status of the system is sent upon completion of the goal. An example of this system can be seen in Figure 2, where the same nodes from Figure 1 are used, but with different elements. Note, that all Nodes, Topics, Services, and Actions in this example can coexist in one environment, rather than how they are depicted as two systems.

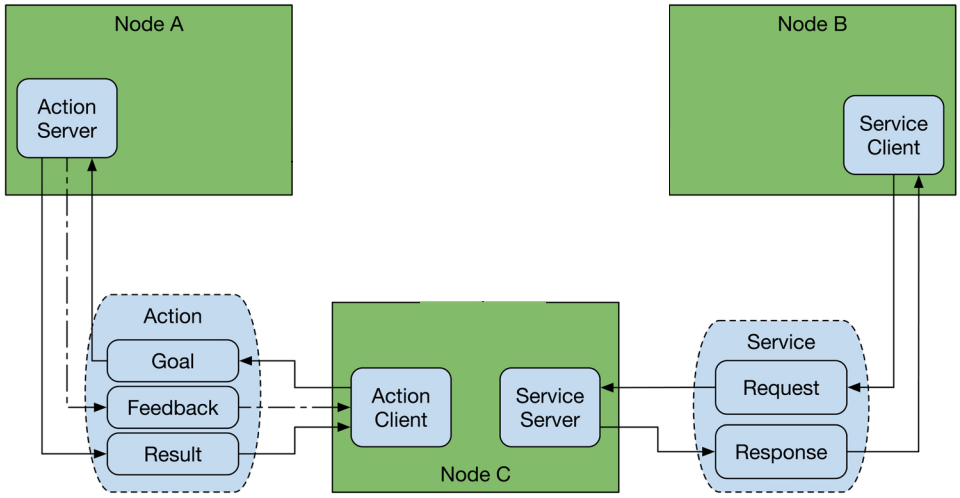

Figure 2: More advanced ROS2 system[8]

There exist other elements and capabilities within ROS2, but in this research only Nodes and Topics will be used. The authors acknowledge that for a complete AM system, all elements must be used for the most optimal supervisory control system.

An additional comment can be made in regards to the use of generative AI, such as ChatGPT[11], for reducing the time required to develop ROS2 nodes. For example, the following prompt can be used to generate a starting point for a Python script that initializes a node, connects to a specific sensor, and publishes that sensor value to a topic: “write a python script for a ROS2 node that connects to ‘specific sensor’ and publishes the value as a 32-bit float message”. If the specific sensor has a public development kit that has a python library option, and the generative AI has been able to learn about it, the response should allow for a more streamlined start to a node script. This may not work in all cases and may also require further program troubleshooting and installation of required libraries.

Methods

To test the capabilities of the ROS2 software, an experimental setup is created. This consists of a simple system designed to simulate an AM system thermal monitoring setup and can be seen in Figure 3. The major components are a Raspberry Pi 4b 8GB (RPi), 500C Ohaus hotplate, and an Omega OS37-10-K-HIE adjustable IR sensor with k-type thermocouple output. By heating a target material on the hotplate to a representative temperature of 90C, the RPi can run the ROS2 system to test the basic functionality by interfacing with the pyrometer for simple temperature monitoring.

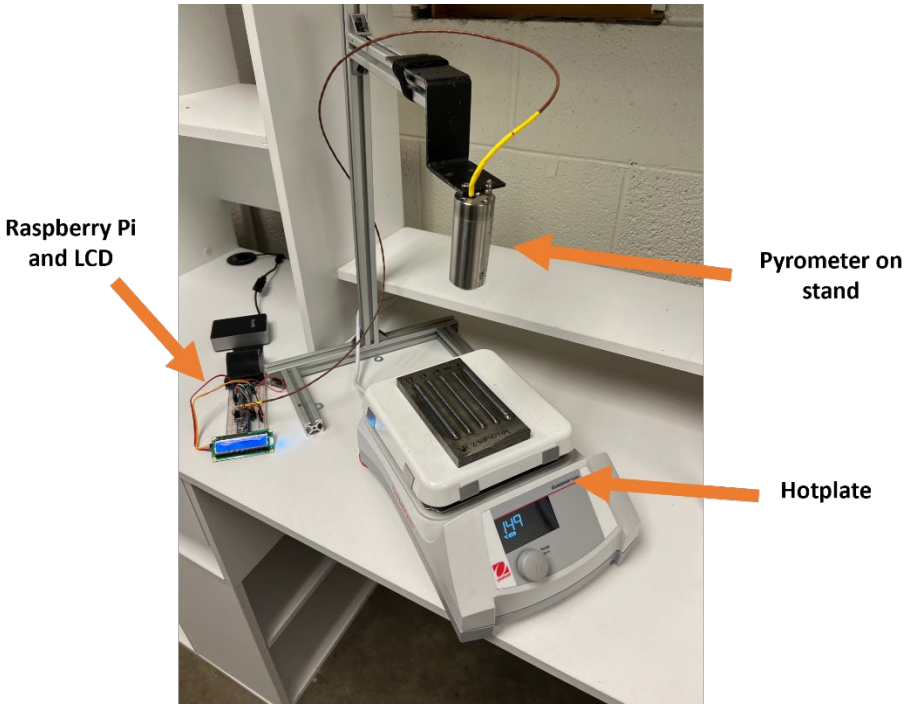

Figure 3: Simple testbed system

The RPi is connected to a standard breadboard using a ribbon cable and a general-purpose input/output (GPIO) cobbler. This allows for the interface pins on the RPi to be made available on the breadboard rows. This system must be able to interface with the pyrometer, collect time series data, and display the pyrometer temperature to a theoretical operator. To interface with the pyrometer an Adafruit MAX31855 thermocouple amplifier breakout board is connected to the breadboard. This breakout board communicates with the RPi via Serial Peripheral Interface (SPI) by using the 3.3 volt and ground pin for power, SCLK and MISO pins for SPI interface, and GPIO 5 for data. To display the current temperature, a basic 2x16 LCD screen is connected to a basic Inter-Integrated Circuit (I^2C) board. This board connects to the RPi using the 5 volt and ground pins, and the SDA/SCL data and clock pins. Figure 4 shows a full wiring diagram of the RPi and its peripherals.

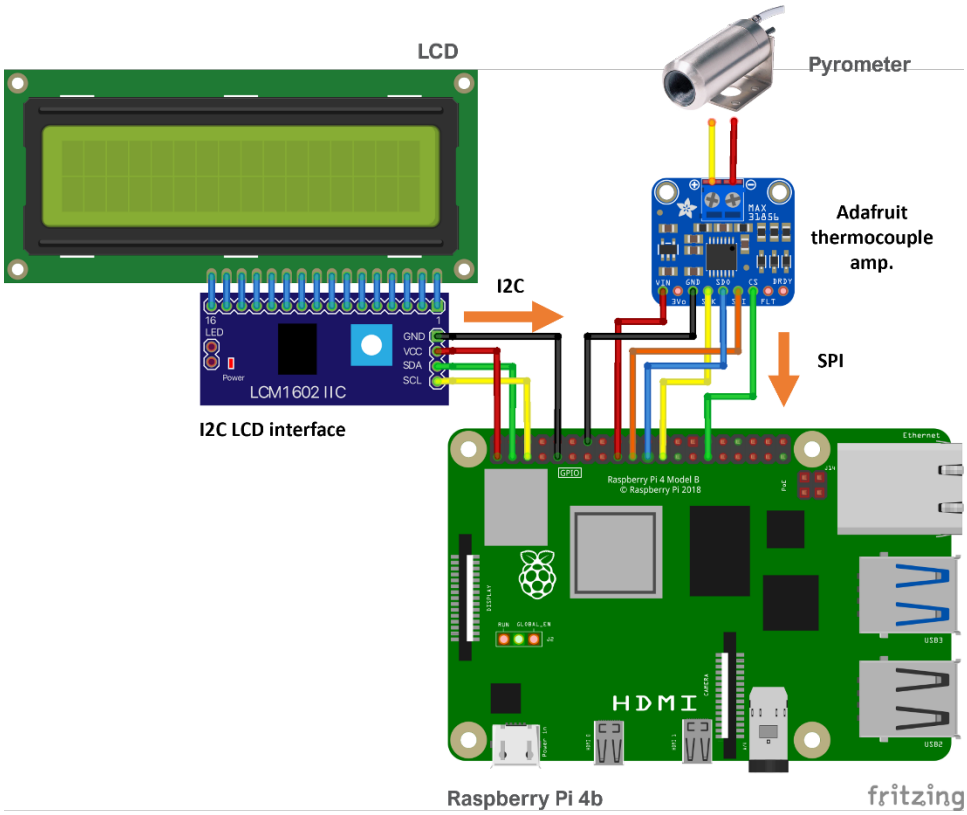

Figure 4: Electrical connections for testbed system

ROS2 Architecture

With hardware connected property, the ROS2 software can now be developed. The RPi is flashed with the Ubuntu Desktop 22.04 LTS 64-bit image[12]. The operating system is set up using standard configuration, Secure Shell Protocol (SSH) enabled, and networked using Wi-Fi. SPI and I^2C interfaces are enabled through raspi-config. The RPi is accessed using Microsoft Visual Studio Code[13] via SSH for the remainder of development. In order to execute the node responsible for communicating with the pyrometer over SPI, the Ubuntu user invoking the ROS node needs access to the "/dev/spidev0.0" access group, found by using the terminal command "ls -l /dev/spidev0.0".

ROS2 Humble is installed using the standard Ubuntu Debian package "ros-humble-desktop"[14]. A testbed workspace was created in the home drive of the main user. Now, the main structural elements of the ROS2 environment can be created. Four packages were created for this testbed and include: Pyrometer Package to interface with the pyrometer and breakout board, LCD Package to display values on the LCD screen, HDF5 Package to archive the temperature time series data as an HDF5 file, and a Bring-Up Package to make it easier to start the testbed system. This general environment can be seen in Figure 5.

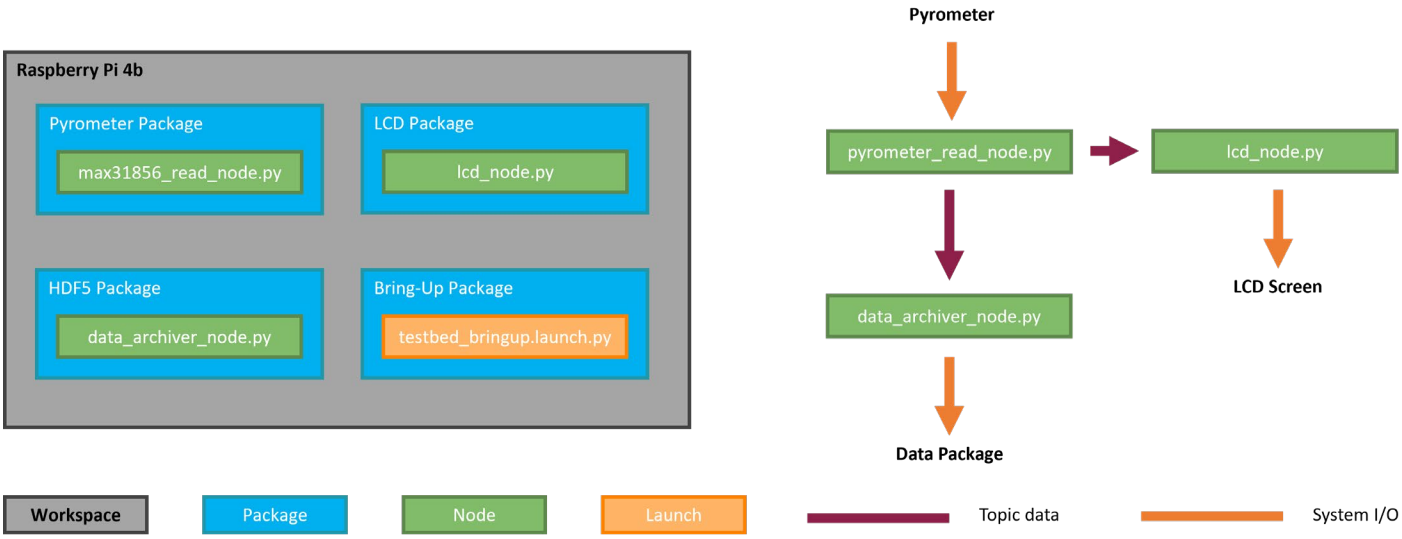

Figure 5: Testbed software structure and data flow diagram

Each package is modular and can be used by any system that uses the same sensor or interface. For example, all AM systems that use the pyrometer (or, more realistically, simply the MAX31855 breakout board) can use the Pyrometer Package. The only exception is the Bring-Up Package which is specific to the system configuration and contains the ROS Launch file, which can be thought of as the orchestrator of the system, that selects and runs the required nodes to boot-up the system as expected. These packages are all simply folders within the "src" folder of the Workspace and themselves contain a "src" folder that contains the executable files that run each Node of the system. These package folders are synced with GitHub[15] to maintain versioning and monitor changes.

The data flow diagram seen on the right of Figure 5 illustrates how the topic data is sent between the nodes and then how that data is collected and displayed through system inputs (pyrometer value) and outputs (HDF5 data package and LCD screen). Once the Launch file is executed, all nodes and topics will function to run the system as defined. This testbed system uses Python 3.10 to program the nodes for all packages. Each node contains a single executable Python script as seen in Figure 6.

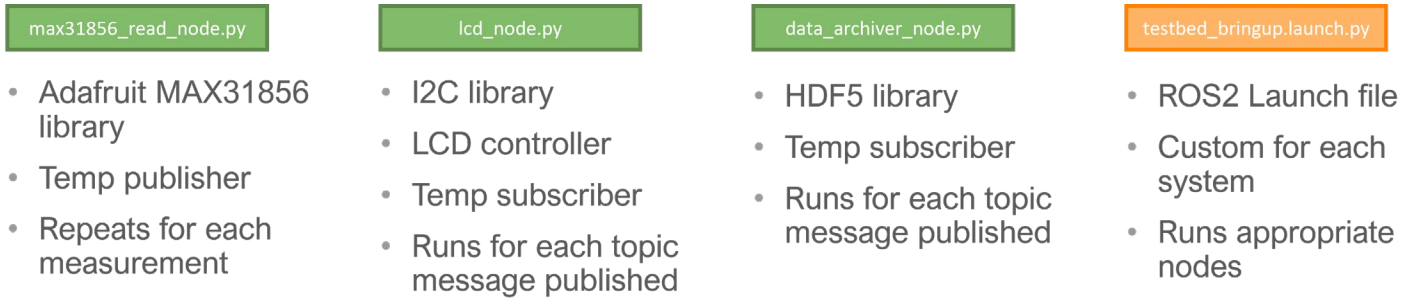

Figure 6: Python files and their contents

Within the Pyrometer Package, the python file ("max31856_read_node.py") utilizes the Adafruit MAX31856 Python library, specifically written for this breakout board, and installed using the command "sudo pip3 install adafruit-circuitpython-max31855"[16], [17]. This library connects to the board and then creates a Python class object that can be called to acquire a temperature measurement. Additional functionality includes the definition of the pyrometer minimum and maximum measurement limits as well as error handling for communication and thermal bounds of the system. The Python script also initiates a topic publisher with the topic name "temperature" and a message format of a 32-bit float number. The MAX31855 python library uses a

delay in the acquisition of a temperature so this node has the temperature measurements collected and published as fast as the library allows.

Within the LCD Package, the python file ("lcd_node.py") utilizes the I^2C library[18] and a library for controlling the LCD via the 16-pin interface[19]. The script also initializes a topic subscriber, subscribing to the topic of "temperature". Every time the subscriber is given a new message from the topic, it runs a callback function that updates the LCD display with the new temperature, using the LCD and I^2C libraries.

Within the HDF5 Package, the python file ("data_archiver_node.py") and utilizes the HDF5 library[20]. This script initializes an HDF5 file with the correct structure and metadata. Similar to the LCD Package, the script initiates a topic subscriber that runs a callback function that adds the temperature value and timestamp to the HDF5 file. Once the node is terminated, the HDF5 is closed and ready for retrieval.

Within the Bring-Up Package, the python file ("testbed_bringup.launch.py"). As mentioned previously, this package is custom for each system. In this script, the launch sequence is relatively simple as it only launches the three main nodes (pyrometer, lcd, and data archiver). These nodes can be run using the terminal and normal ROS2 interface commands, but using the launch file reduces the need to open three terminals and run each node separately.

Experimental Methods

The final experiment for this research was to record data while completing the pyrometer calibration procedure. First, the RPi was connected to the breakout boards and pyrometer, then powered on. The hotplate was loaded with a sample material of steel with steel weld beads and the pyrometer was positioned directly above and set at the correct height for the pyrometer measuring distance and set to read 75% of the target temperature of 90C. The ROS2 launch file was then started. The hotplate setpoint was adjusted to 90C and allowed to reach setpoint and remain stable for many minutes. The pyrometer was then calibrated to show the correct reading of 90C then the hotplate was turned off and allowed to cool to ambient temperature over many minutes. The ROS2 network was then terminated and the HDF5 file was recovered.

Results

The resulting time series data for temperature from the HDF5 file is shown in Figure 7. The initial temperature can be seen raising from a starting point of about 55C and leveling out at slightly less than 70C. After the spike in temperature caused by the manual calibration procedure, the temperature reads 90C. As the hotplate is cooled, the pyrometer temperature settles to slightly above 70C, indicating either further calibration processes are needed, or the thermocouple amplifier is not operating as expected.

Final temperature data was collected at around 4Hz which is most likely limited by the Adafruit Python library using a hard-coded delay. A higher data collection frequency may be possible by changing developing a more efficient Python function that collects data without a defined delay. This data collection frequency is high enough for many AM sensor needs, but other sensors will need higher throughputs for both speed and message size.

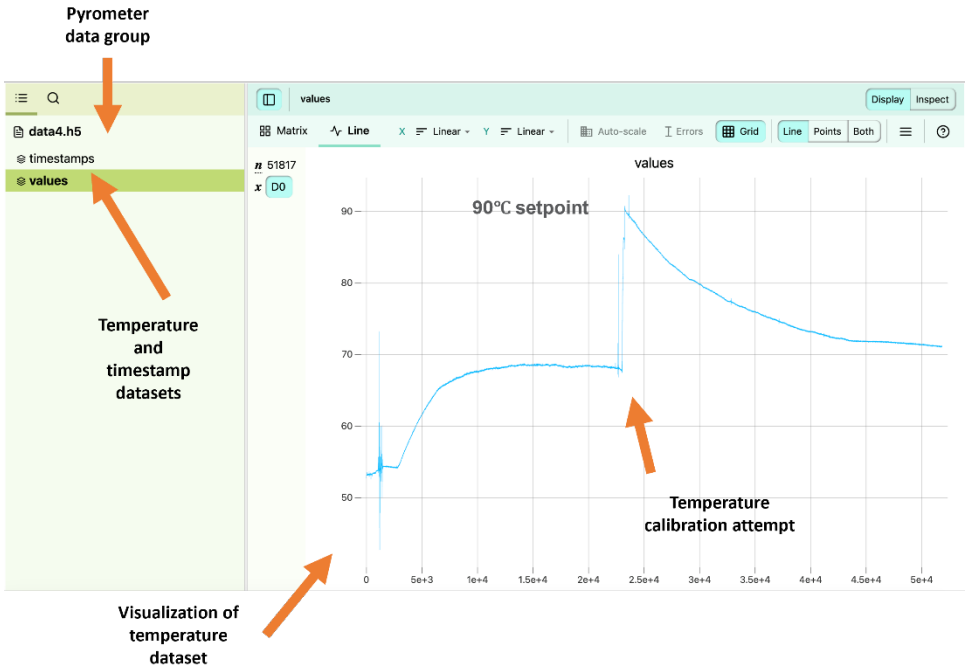

Figure 7: HFD5 file interface and temperature graph

The purpose of this research was to test the feasibility of using ROS2 as a process monitoring and control system for AM. This research shows the setup of a ROS2 environment to be relatively simple and well documented in support and online communities. Rather than writing Python scripts for both sensor data acquisition and system operation, only the sensor interface needs to be written, while ROS2 manages operation. Additionally, the Python scripts defining ROS2 nodes can be significantly short and quick to write, as initializing a ROS node takes only a few lines of code and with an existing Python library sensor integration may only take one or two more lines of code. Given the highly modular design, packages can be easily interchanged throughout different AM systems, and unique versions of nodes can be run from the same package, as defined by the needs of the system. Such new node version can also be made available to other systems that use that package, thus expanding the features of all systems.

Conclusion

This research has shown that the collection and recording of pyrometer sensor data at 4Hz is possible using ROS2 and allows for highly modular and scalable systems of increasing complexity. Future research will focus on the development of additional packages for sensors such as the FLIR A50 thermal camera and XIRIS XIR-1800 which have existing software development kits (SDKs) which allow for integration into the ROS environment. Further research also needs to be done for developing more functionality through decision-making nodes, possibly powered by machine learning. Work is also being done to implement HDF5 files and define a more standardized AM data management framework. Finally, the functionality of ROS2 systems can be expanded by utilizing the ability to network multiple computers running ROS. Bringing data computation closer to the source of creation will allow for a more efficient system that is not controlled by a centralized computer running all actions, acting as a single point of failure.

Acknowledgements

The figures and text in this report are human generated. The python scripts used in this research were partially developed using generative AI[11]. Source code is available via requests to Authors and may be made available as a public ROS package in the future.

References

[1] "KUKA.RobotSensorInterface," *KUKA AG*. https://www.kuka.com/en-us/products/robotics-systems/software/application-software/kuka_robotsensorinterface (accessed Sep. 11, 2023).

[2] "ROS: Why ROS?" https://www.ros.org/blog/why-ros/ (accessed Sep. 11, 2023).

[3] "ROS-Industrial," *ROS-Industrial*, Aug. 18, 2023. https://rosindustrial.org (accessed Sep. 11, 2023).

[4] "Creating a workspace — ROS 2 Documentation: Humble documentation." https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html (accessed Sep. 11, 2023).

[5] "Creating a package — ROS 2 Documentation: Humble documentation." https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html (accessed Sep. 11, 2023).

[6] "Understanding nodes — ROS 2 Documentation: Humble documentation." https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (accessed Sep. 11, 2023).

[7] "Understanding topics — ROS 2 Documentation: Humble documentation." https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (accessed Sep. 11, 2023).

[8] "Robot Operating System 2: Design, architecture, and uses in the wild | Science Robotics." https://www.science.org/doi/10.1126/scirobotics.abm6074 (accessed Sep. 11, 2023).

[9] "Understanding services — ROS 2 Documentation: Humble documentation." https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html (accessed Sep. 11, 2023).

[10] "Understanding actions — ROS 2 Documentation: Humble documentation." https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html (accessed Sep. 11, 2023).

[11] "ChatGPT." https://chat.openai.com (accessed Sep. 11, 2023).

[12] "Ubuntu for Raspberry Pi," *Ubuntu*. https://ubuntu.com/raspberry-pi (accessed Sep. 11, 2023).

[13] "Visual Studio Code - Code Editing. Redefined." https://code.visualstudio.com/ (accessed Sep. 11, 2023).

[14] Open Robotics, "The Robot Operating System 2 (ROS2)." Open Robotics. [Online]. Available: https://www.ros.org/

[15] "GitHub: Let's build from here," *GitHub*. https://github.com/ (accessed Sep. 11, 2023).

[16] "Introduction — Adafruit MAX31855 Library 1.0 documentation." https://docs.circuitpython.org/projects/max31855/en/latest/ (accessed Sep. 11, 2023).

[17] "Release 3.2.19 - Updated .pylintrc, fixed jQuery · adafruit/Adafruit_CircuitPython_MAX31855," *GitHub*. https://github.com/adafruit/Adafruit_CircuitPython_MAX31855/releases/tag/3.2.19 (accessed Sep. 11, 2023).

[18] K.-P. Lindegaard, “smbus2: smbus2 is a drop-in replacement for smbus-cffi/smbus-python in pure Python.” Accessed: Sep. 11, 2023. [Online]. Available: https://github.com/kplindegaard/smbus2

[19] “Adafruit_LiquidCrystal.” Adafruit Industries, Sep. 07, 2023. Accessed: Sep. 11, 2023. [Online]. Available: https://github.com/adafruit/Adafruit_LiquidCrystal

[20] A. Collette, “h5py: Read and write HDF5 files from Python.” Accessed: Sep. 11, 2023. [MacOS :: MacOS X, Microsoft :: Windows, POSIX :: Linux, Unix]. Available: https://www.h5py.org/