Robotics Dashboard Web App for Laboratory Monitoring

Amaan Khan[†,1], Rory Butler[§,¶,2], Carla Mann[¶, 3], Arvind Ramanathan[¶,4]

[†]University of Illinois at Urbana-Champaign, Champaign, IL

[§]University of Chicago, Chicago, IL

[¶]Argonne National Laboratory, Lemont, IL

[1]amaanmkhan2@gmail.com, [2]rorymb@uchicago.edu, [3]cmann@anl.gov, [4]ramanathana@anl.gov

Abstract. Developing a self-driving biology lab with a suite of autonomous robots that independently conduct biology experiments pose numerous benefits to scientists as it can reduce human error and speed up the experiment. Such a lab contains many independent robots, which all must work together smoothly. However, it is difficult for lab operators to monitor, analyze, and debug the data from the numerous robotic systems. Therefore, an application that allows engineers and researchers to monitor messages, video streams, ROS topics, and provides interactive analysis tools on real-time data can significantly improve productivity, debugging, and ease of use. Here we describe our progress in creating a web dashboard application that provides these set of tools. Currently the dashboard application can display live camera video streams from robots. Additionally, lab monitors can send messages and commands back to the robots from the dashboard application as well as interact with various data visualizations.

INTRODUCTION

Creating a fully autonomous biology laboratory can pose numerous benefits to researchers. Having humans conducting biology experiments all the time can result in many errors during the experiment as well as take up valuable time that could be instead spent elsewhere in the research process. Additionally, a self-driving lab can also speed up biology experiments since robots are faster, can work continuously and in parallel. Our overarching goal is to develop a self-driving biology lab to be able to have robots conduct biology experiments autonomously.

Building such a lab requires many independent robotic systems to work together smoothly to complete the experiment. However, it is quite difficult for lab operators to monitor, analyze, and debug the data from the numerous robotic systems in the lab. Therefore, having a dedicated application that can allow engineers and researchers to monitor messages, video streams, data, provide interactive analysis tools on real-time data, and provide the ability for lab operators to control the various robots can significantly improve productivity, debugging, and ease of use. Image 1 is an example of what we are aiming to develop. In Image 1, there are many boxes with various data that is being sent from the numerous robots in the lab. For instance, in the top left there is an output of the live stream camera display from one of the robots. On the bottom is a visualization of some data that was sent from the robots. And on the right middle is a box displaying the statuses of the various ROS[1] (Robot Operating System) nodes. Having a dashboard app similar to the one shown in Image 1 is what we aim to develop.

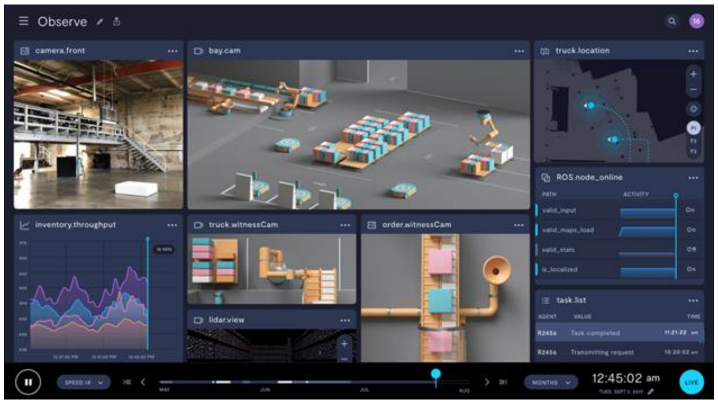

Image 1: *Example of a dashboard app similar to what we aim to develop. Top-left box displays live stream camera output, bottom-left box shows data visualization of the lab, and middle-right box shows statuses of ROS nodes.*
Source: *https://formant.io*[2]

In this paper, we describe the progress we made in developing such a dashboard application. We developed software to live stream the camera from a robot to the dashboard application, which resides in a different computer. We also developed the ability for users to send commands and messages back to the different robots in the lab, as well as interactive graph displays for users to interact with data visualizations.

METHODS

We utilized a set of software tools and packages to develop a dashboard application. Namely, we used Python[3] programming language, Robot Operating System 2[1] (ROS 2), plotly.dash[4], and Flask[5]. Python is a very popular, high-level, interpreted programming language that is used widely in the space of robotics, which is why we chose to use it. ROS is an open-source robotics middleware suite that is a set of software frameworks for robot software development. Although the name has "operating system" in it, it is actually not an operating system but rather a set of software frameworks. Flask and plotly.dash are Python packages that are used for developing web applications. Plotly.dash is built on top of Flask and is used in particular for developing dashboard web applications.

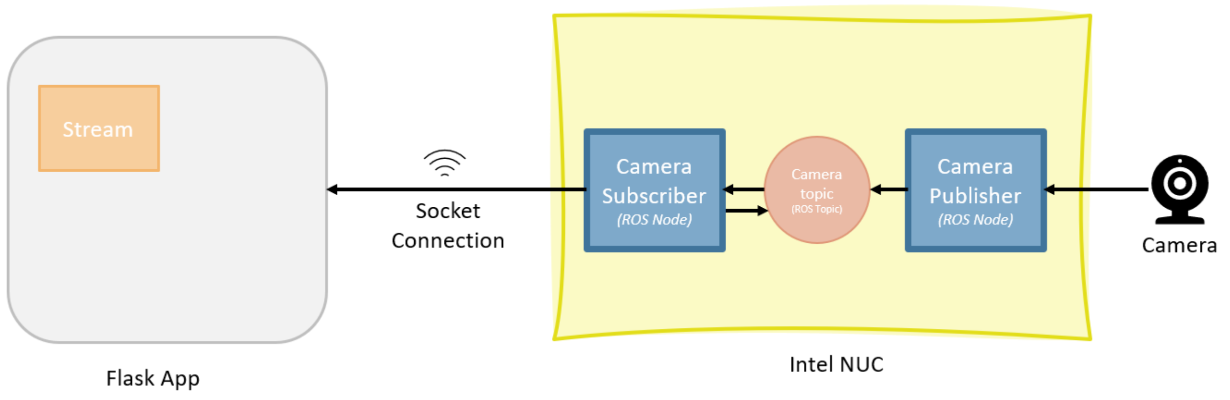

Figure 1: *Demonstrates how camera connected to a computer, such as an Intel NUC, streams to the Flask App. Data is sent to a ROS publisher, which publishes data to a ROS topic. Then a subscriber listens to that topic and sends any data to the Flask app (residing on separate computer) over a socket connection*

Figure 1 is explaining how we developed software to live stream the camera from the robot to the Flask web dashboard application. On the right in yellow is the Intel NUC (Next Unit of Computing), which is a powerful computer used often for edge computing and for running many Machine Learning and Deep Learning models. In this example, the Intel NUC is basically the "robot" in the lab, and attached to the Intel NUC is a USB camera. We also have a socket connection setup between the Intel NUC and the Flask dashboard web app, which is residing in a separate machine. Inside the yellow box represents the code that is going on inside the Intel NUC. The blue boxes are the ROS nodes and the red circle is a ROS topic. Firstly, the blue box on the right named "Camera Publisher" runs the video feed from the camera and constantly publishes the video frames to the Camera ROS topic. Then, the subscriber is constantly listening in to the Camera topic and whenever a new frame appears, it retrieves that frame and sends it over a socket connection to the Flask dashboard web application. The Flask app is then listening on the socket connection and whenever new data is sent, it retrieves it and displays it on the website.

This demonstrates how we were able to live stream a camera feed wirelessly over the network to the Flask app, which can reside on any computer as long as it is on the same wireless network. Now that we were able to establish a communication one-way from the robot to the dashboard app, we want to now establish a communication the other way so that users can send commands and messages back to the robot. This is accomplished through the architecture shown in Figure 2.

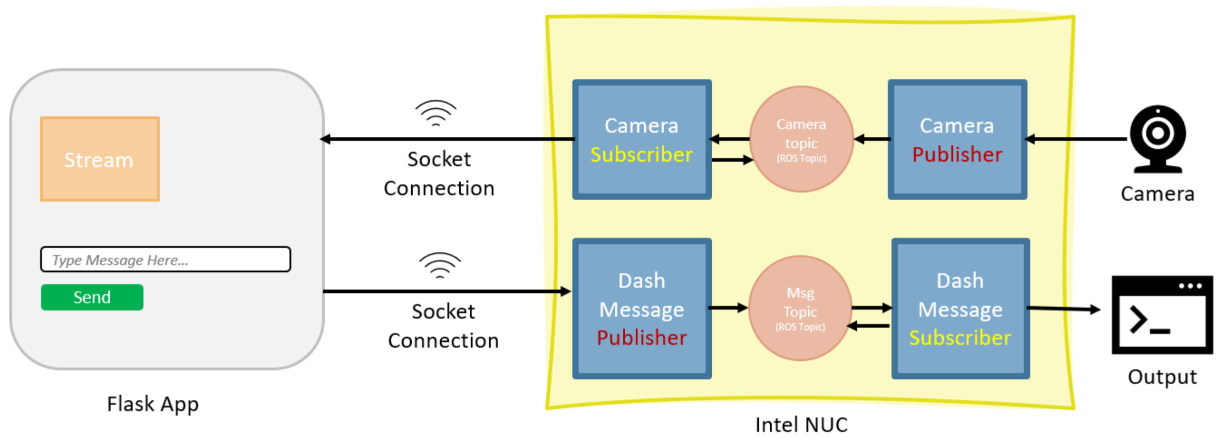

Figure 2. *Demonstrates the two-way communication. User can type in a message in the Flask app and click send. Message is sent over socket connection, in which the "Message Publisher" publishes the message to a ROS topic and a subscriber fetches that message and outputs it.*

In Figure 2, we have the live streaming camera logic the same as before, which is demonstrated by the fact that the publisher and subscriber model for the camera in Figure 2 is the same as that of Figure 1. Firstly, we added another socket connection representing the connection going from the dashboard application to the robot. In the Flask app, we added a textbox where the user can type in a message, and then a green "Send" button. When the send button is clicked, the message in the textbox is sent over a socket connection to the Intel NUC, in which a ROS node called "Dash Message Publisher" receives that data and sends it to a ROS topic called "Msg topic." The Dash Message Subscriber subscribes to the "Msg Topic" and whenever new data appears, it fetches it and then outputs it to the console. However, we can easily change it to executing a command instead of printing it to the console.

Additionally, using plotly.dash I demonstrated the ability to create an interactive graph where the user can zoom in, crop, etc with the data that is sent from the robots to the dashboard application. Currently it is just a normal distribution, but in the future this can be data from a sensor, or any other important data from a robot. The user can interact with the data while the live stream is playing, which is very good as this means it does not interfere with the live stream and is running in parallel.

Below is a screenshot of the current state of the dashboard, where the live stream, textbox with submit button, and an interactive normal distribution can be found.

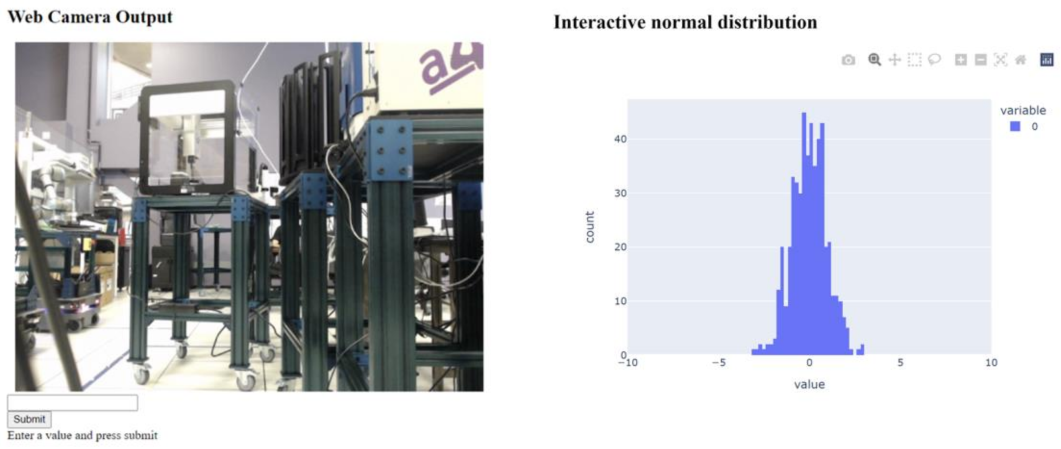
Web Camera Output
Submit
Enter a value and press submit
Interactive normal distribution
variable
0
count
value
40
30
20
10
0
−10
−5
0
5
10

__Image 2.__ Screenshot of the current dashboard

RESULTS

We were able to develop the infrastructure to have a two-way wireless communication between the Dashboard app and a robot. Currently, there is a live stream of a camera being sent to the dashboard, but due to the modular design of our software, it is very simple to change the data from a live stream to any other type of data. For instance, we can send data from a sensor following the same method.

Additionally, we developed the initial infrastructure to allow for users to send messages and commands back to the various robots. Currently, the proof-of-concept demonstrates the ability for users to send text messages, but it is very simple to change this to a command that can be executed by the robot instead of a message that the robot outputs to the console. Furthermore, we also developed the ability for users to interact with data visualizations on the dashboard application.

CONCLUSION

Through our work on this project, we have been able to develop the underlying base software to rapidly add new features for our dashboard. The two-way communication system makes it easy to send different types of data due to the modular design of our software. Additionally, our dashboard application can significantly increase productivity and it enables flexible and easy access to debugging information from outside the development environment.

For the future, we plan to make several updates and plan to incorporate several new features. Firstly, we plan to remove the socket connection layer and instead integrate the Flask app with ROS. ROS has the ability to send messages and data over the network; however, we chose to not initially develop it this way because it adds another layer of complexity. For the time being, a socket connection was simpler, but a socket connection adds additional latency and is not as scalable as instead integrating ROS within the dashboard application itself. Furthermore, we plan to add the feature to stream multiple camera streams. We have multiple robots in the lab, and each robot often has multiple cameras to capture different angles. Therefore, implementing a solution to stream multiple camera streams is a vital feature for the application. Additionally, we want to incorporate a log output of different ROS topics. The user should be able to click on a dropdown where they can choose which ROS topic they want, and from there the output window should display the various messages and log outputs that are being sent on that topic. Lastly, we want to improve the appearance of the web application using CSS.

Overall, this dashboard application is able to increase the ease of debugging and productivity of various lab operators developing technology for the autonomous biology lab. We were able to develop the underlying software while practicing good software design, which allows developers to easily add new functionalities and features.

ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy (DOE) Office of Science and DOE Office of Workforce Development for Teachers and Scientists (WDTS) through the Science Undergraduate Laboratory Internships (SULI) program. I also want to thank Rafael Vescovi for providing guidance and clarifications on ROS and the program architecture, as well as Casey Stone for setting up weekly tutorials and meetings. I would also like to thank Rory Butler, Carla Mann, and Arvind Ramanathan for their generous mentorship and support.

FOOTNOTES AND REFERENCES

[1]S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” Science Robotics vol. 7, May 2022.

[2]J. Linnell, Formant (2019).

[3]Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.

[4]Inc., P. T. (2015). Collaborative data science. Montreal, QC: Plotly Technologies Inc. Retrieved from https://plot.ly

[5]Grinberg, M. (2018). Flask web development: developing web applications with python. " O&#x27;Reilly Media, Inc."