DESIGNING AND IMPLEMENTING A CONTROL INTERFACE FOR INFRARED LED PROJECTOR SYSTEMS

by

Casey K Campbell

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Spring 2019

DESIGNING AND IMPLEMENTING A CONTROL INTERFACE FOR INFRARED LED PROJECTOR SYSTEMS

by

Casey K Campbell

Approved: _____

Fouad Kiamilev, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical & Computer Engineering

Approved: _____

Levi T. Thompson, Ph.D.
Dean of the College of Engineering

Approved: _____

Douglas J. Doren, Ph.D.
Interim Vice Provost for Graduate & Professional Education

ACKNOWLEDGMENTS

I would like to thank my parents, Cathy and Kevan, and my siblings, Ryan and Shannon, for supporting me through my life and academic career. I would not be where I am today without their love and support.

I would also like to thank my advisor, Dr. Fouad Kiamilev, for allowing me the opportunity to work as a research assistant on the CVORG team. Working under his guidance has taught me invaluable lessons about being both an academic and a person of high character.

The CVORG team provided a wonderful work environment, pushing me to achieve my personal goals while also maintaining a strong sense of community working towards a common objective. I would especially like to thank Jeffrey Volz, Zachary Marks, Benjamin Steenkamer, Aaron Landwehr, Peyman Barakhshan, Miguel Hernandez, Hamzah Ahmed, Furkan Cayci, and everyone else at CVORG, past and present, for all their help over these past few years. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Infrared (IR) detectors are becoming a widely used tool in modern research. IR source generators are needed to test and improve these IR detectors. In 2014, the CVORG team at the University of Delaware developed the first IR Scene Projector (IRSP) using IR light emitting diodes (IRLEDs). This IRSP was called the SLEDS system. IRLEDs are capable of faster framerates and better image quality than the previous generation of IRSP technology. New types of SLEDS projectors with higher resolutions were then created. The current goal of the SLEDS project is to produce a complete system that can replace the previous iteration of IRSP technology.

One of the integral parts of the SLEDS system is the control interface. In order to effectively test and control the SLEDS technology, an advanced user interface was developed. This interface, called the Projector IRLED Native GUI (PING), was created to scale with the growth of SLEDS technology. As the capabilities of the projector system evolved, the control interface evolved with them. PING is composed of three main components: the HTML frontend, backend controller, and the backend service architecture. The three components work in conjunction to initialize, control, test, and analyze the SLEDS systems. PING also communicates with the other hardware components of the SLEDS system to maintain efficient system operation.

Chapter 1

INTRODUCTION

1.1 Background

Light in the infrared (IR) wavelength spectrum is incredibly versatile. It is used in multiple different medical, military, and scientific applications. Since it is invisible to the naked human eye, cameras capable of capturing IR light waves have been developed. These cameras are integral for detecting of IR radiation and testing IR-based technologies. One of the emerging IR technologies is the Infrared Scene Projector (IRSP). IRSPs are used to emit a controlled IR display. This IR display can be used for developing and testing IR-sensitive equipment, such as infrared sensors. The benefit of an IRSP is that it can display these scenes in real-time, testing the limits of IR sensor technology. IRSPs are usually better for testing IR sensors than natural IR sources [13].

The most prevalent technology currently used for IR scene projection is resistor arrays. While useful, these resistor arrays have drawbacks and limitations. Due to slow heating time, they are unable to project displays at high framerates. They are also unable to separate apparent temperature from thermal temperature or display objects considered "cold" relative to the rest of the scene [6, 10, 20, 19]. To overcome the limitations of the previous technology, a new type of IRSP was developed using light-emitting diodes. In 2014, the CVORG research team developed the first infrared LED scene projector (IRSP). The research and development was conducted at the University of Delaware under Dr. Fouad Kiamilev in conjunction with the University of Iowa [16].

The original super-latticed LED system (SLEDS) had a resolution of 512x512 and ran at 100 Hz [14]. After the successful creation of the original SLEDS, two

new sets of 1024x1024 resolution projectors were developed: the Nightglow SLEDS (NSLEDS) projectors and the two color SLEDS arrays (TCSA). The goal of the arrays were to push both the resolution and framerate of the IRLED projector design. The NSLEDS arrays are currently still in the development and testing phase. The TCSA arrays were tested extensively for to examine properties that could be applied to all IRLED projector technologies [22]. A third variation of projectors called HDILED with a resolution of 2048x2048 are currently being developed. The current goal of SLEDS research is to improve the current iteration of the IRLED technology.

1.2 Problem Formulation

In order to effectively run the SLEDs system, a control interface needed to be created that would maximize performance and functionality. The main purpose of the control interface would be to process user input and perform the corresponding actions on the hardware system. In the early stages of the SLEDs system, the system was controlled by a laptop directly connected to the control electronics. The only functions of the laptop were to load the control firmware onto the hardware and pass the images and videos along to the projector for display [14].

As the scope of the SLEDs system evolved over time, more control functionality was needed to match the capabilities of the projector system. As a result, a graphical user interface (GUI) was developed for more streamlined user control. The GUI needed to be designed with two major elements in mind:

- A comprehensive structure - In order for the GUI to be efficient, it needed to be easy to operate. Therefore, a comprehensive structure was needed to maximize usability.
- Scalability - To keep up with the advancing projector system, the GUI needed to be designed such that new features or capabilities could be added without disrupting the previously established structure.

Using these design methodologies, the Projector IRLED Native GUI (PING) was developed. This paper will document the design process and functionally of PING.

Chapter 2

ARCHITECTURAL DESIGN

2.1 SLEDS System Components

2.1.1 IRLED Projector and Housing

The LEDs in the SLEDS projectors are built from 10V CMOS pass transistor pairs. These transistors pass the analog input to transistors that drive the LEDs. These transistors can pass varying voltage levels to light the LEDs at different intensities. The higher the voltage, the stronger the IR radiation [3]. Each LED is driven by a read-in integrated circuit (RIIC) on a board that the projector is mounted to. The RIIC and projector are mounted into a Dewar chamber to maintain a vacuum. The Dewar is also used to cool the projector down to cryogenic temperature (around 77 degrees Kelvin) for optimal performance. A custom packaging was developed in an attempt to increase the Dewar temperature performance [12]. Figure 2.1 shows the projector and RIIC mounted in the Dewar.

2.1.2 Close Support Electronics

The Close Support Electronics (CSE) are responsible for a variety of tasks required for the projector to operate. The main components of the CSE are the HDMI input/output cards, the FPGA, the Interface Boards, the Digital to Analog Converters (DACs), and amplifiers. All of these components are packaged into a sturdy custom casing. The CSE receives generated scenes via the HDMI cards. The data is read by the TB-6V-LX760-LSI FPGA and stores it in the local BRAM. The FPGA uses a custom-developed firmware to work as the controller for the hardware system. The data is piped to the array when all 16 data channels used for pixel control are ready to write to the array. The FPGA writes 16 pixels to the array at a time [18]. The

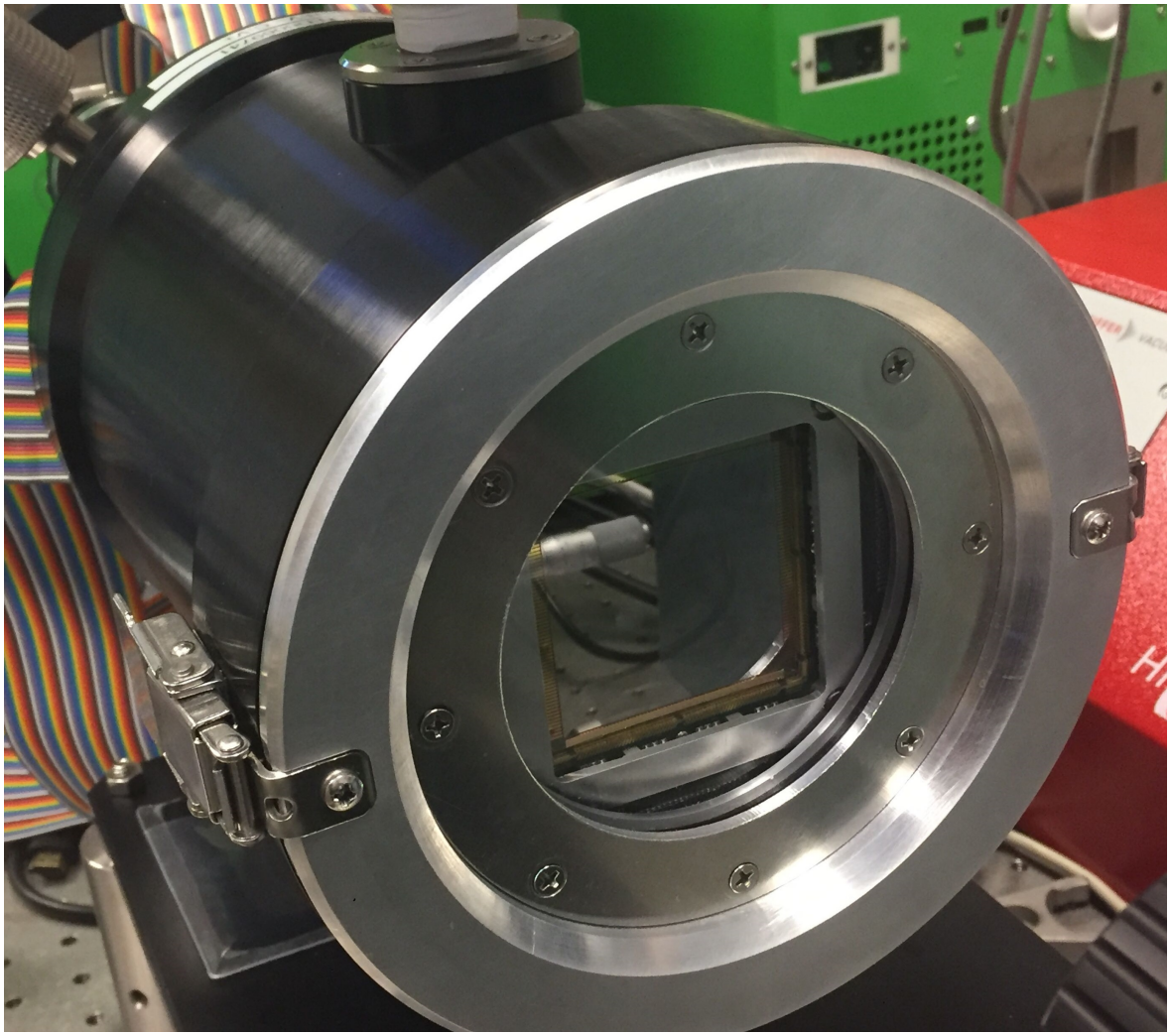

Figure 2.1: The the projector and RIIC mounted in the Dewar.

pixel data is passed to the DACs, which convert the digital data to an analog system the projector can use. The analog signals are then amplified to the necessary voltage levels. Finally, the amplified signals are passed to the interface boards which connect the CSE to the projector via ribbon cables. Figure 2.2 shows the CSE components housed in their custom casing.

2.1.3 The IR Camera

Since the output of the SLEDS projector is in the infrared spectrum, it is invisible to the naked eye. Therefore, an IR Camera is used to observe the projector

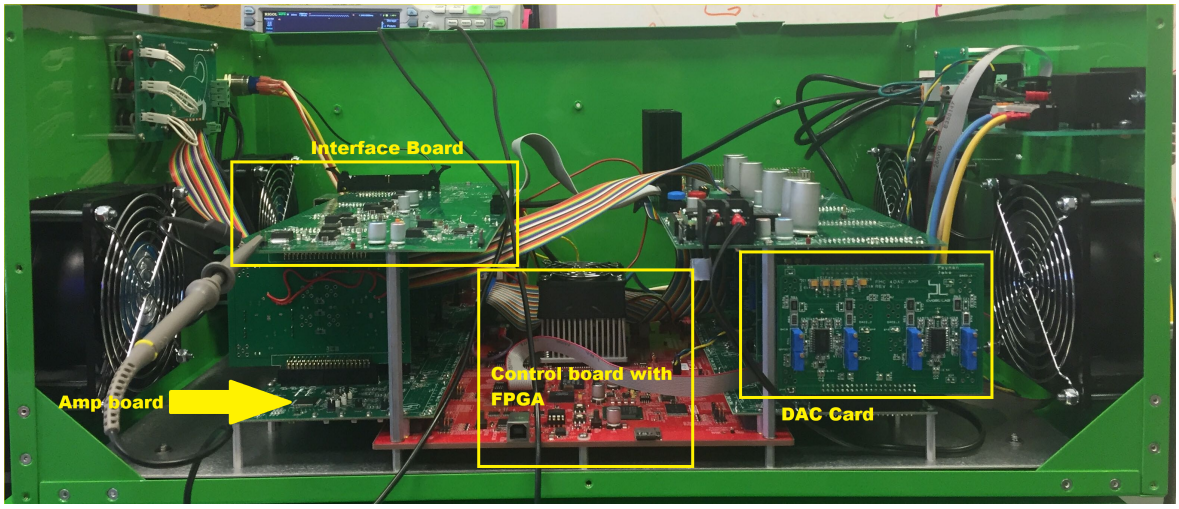

Figure 2.2: The CSE Hardware.

output. The IR Camera used in the system is a FLIR SC8200 camera. The camera is controlled by Ethernet and camera link. The camera communicates with the camera link card in the NUCPC. The camera link card used is the Visionlink-F4 camera link frame grabber [21]. An image of the camera can be seen in figure 2.3.

Figure 2.3: The FLIR IR Camera.

2.1.4 The NUCPC

The NUCPC is used for scene generation and software-side control. The NUCPC runs a custom-built Scientific Linux kernel. Originally it was primarily responsible for non-uniformity correction (NUC) of images sent to the projector. Its scope evolved with the capabilities of the system. The NUCPC communicates with and commands all other components of the system.

2.2 Interface Architecture Overview

At the highest level, PING is separated into two main components: the frontend and the backend. The frontend acts as the visual interface for PING using HTML, CSS, and JavaScript. The backend runs on the NUCPC. Using Python, the backend communicates with the frontend, processes user requests, and transfers those commands to the hardware. The backend also maintains the connections to the CSEs, the IR camera, and the temperature monitor.

The frontend and backend communicate with each other over a WebSocket. The WebSocket allows full two-way communication over a single Transmission Control Protocol (TCP) link [8]. Using serial connections, the backend is able to send and receive commands to the CSE hardware, the infrared camera, and the temperature monitor. PING sends serialized packets to the firmware on the CSE's FPGA. The backend also transfers and receives processed display frames from the CSE over HDMI. For the infrared camera, PING is able to set camera configuration settings and save frames captured in the camera's memory. The temperature reader measures the current temperature of the projector enclosure, and PING periodically pulls the current value for data logging.

All user connections to the NUCPC are through secure shell (SSH) tunneling on the same local network as the machine. Using SSH, the user initializes the web server by running the Python script "web_interface.py". The user then connects to PING on localhost port 8087 on their machine. A connection is established between the user's machine (the client) and the NUCPC server. Only one connection on the

port is allowed at a time to prevent conflicting commands from being sent by two different users.

Figure 2.4 demonstrates the architecture setup of the main hardware components of the SLEDS system.

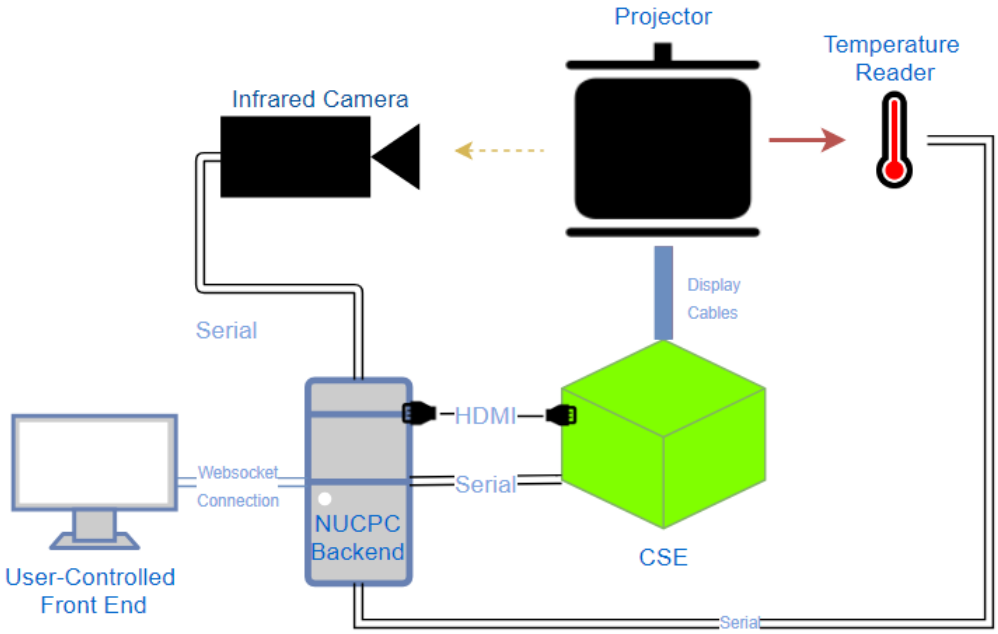

Figure 2.4: The architecture of the SLEDS system.

Chapter 3

INTERFACE FRONT END ARCHITECTURE

PING's frontend architecture is built using a combination of HTML, JavaScript, and CSS. HTML is used to define the content of the web pages the user sees. CSS is used to specify the layout of the web pages. Most of the CSS used for the user interface is part of a bootstrap template that was used to shape the visual framework of PING. Custom CSS was developed for unique HTML classes or objects, (ex. a status light used to indiciate the CSE trip status). The final component, JavaScript, is used to program the behavior of the web pages. JavaScript is also responsible for managing the WebSocket communication with the backend.

3.1 Main HTML Framework

The frontend is composed of two main files: "index.html" and "index.js". "Index.html" functions as the main page that the rest of the frontend interface is built around. When the user first connects to the server on the NUCPC, the server's default response is to send "index.html" back to the client. Once "index.html" has been received by the client, a group of JavaScript commands within the page are called to load "index.js", required js and CSS files, and the html tabs.

The HTML tabs are separate HTML files that are selectively displayed within a container in the center of the main page. Each HTMl tab has its own JavaScript file which is loaded after the tab layout file has been received by the client. The HTML and JavaScript files were written into individual tabs in order to maintain better organization. The tab-specific JavaScript files need to be loaded in their corresponding tab HTML file to prevent issues with function importing. Figure 3.1 demonstrates the frontend file structure and the steps in which these files are loaded.

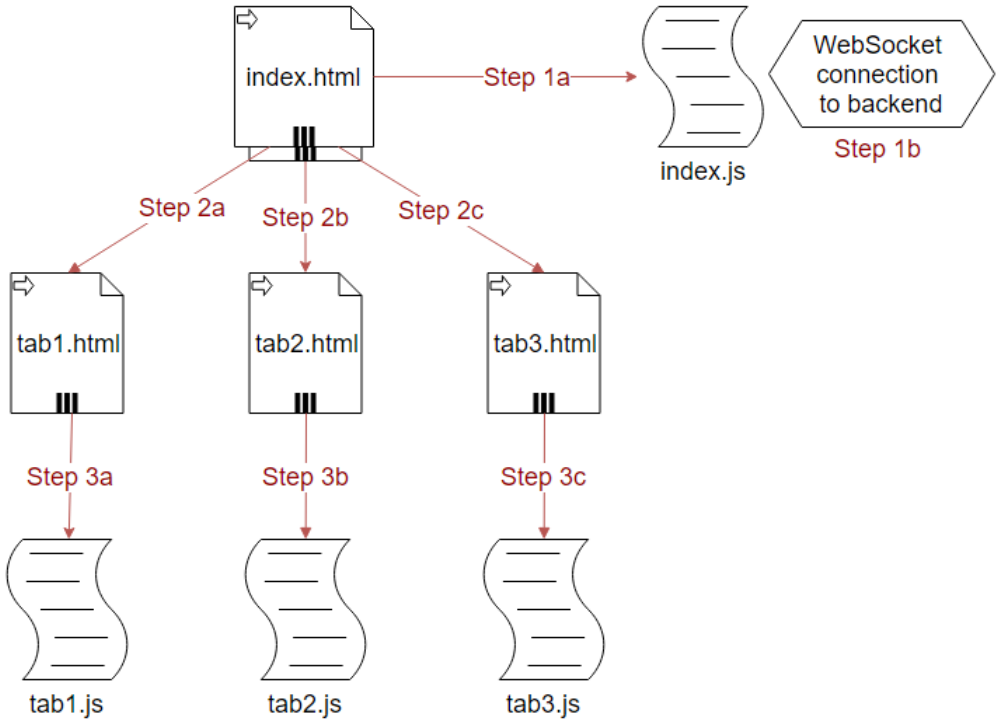

Figure 3.1: PING frontend framework. Step 1: index.js is loaded and the websocket is established. Step 2: The tab HTML files are loaded. Step 3: The tab JavaScript files are loaded.

3.2 Main JS Framework

"Index.js" contains utility functions needed by all the HTML tabs and is responsible for handling communication with the backend. Most of the utility functions in "index.js" are used to prepare input data to be sent over the WebSocket and to unpack and process responses from the backend. The WebSocket data preparation and structure will be covered in section 5.2. "Index.js" also handles functionality associated with aspects of the main HTML page, such as the CSE status bar, CSE trip control, and the backend status console. The tab-specific JavaScript files contain functions predominantly used to control their corresponding HTML file.

3.3 jQuery

JQuery is used for most of the HTML page control/manipulation. JQuery is a JavaScript library made explicitly for fast and effective HTML traversal and manipulation [11]. Using jQuery's built-in API's, PING is able to manipulate HTML objects by referencing their ID, name, class, etc. These references can also be tied to event listeners linked to user input, such as clicking a button. An example jQuery command can be seen in Figure 3.2.

```
1. $("#window_offset_x-init").on('change', function(){
2.     round_x_offset("window_offset_x-init");
   });

   function round_x_offset(offset){
       if(projector_height == undefined || projector_width == undefined){
           var projector = $("#projector").val().split(",");
           projector_width = parseInt(projector[1]);
           projector_height = parseInt(projector[1]);
       }
3.     var value = $("#" + offset).val();
       if(value % 2 != 0){
           value = 2 * Math.round(value / 2);
       }
       if(value >= projector_width){
           value = 2 * Math.round(projector_width / 2) - 2;
       }
       if(value < 0){
           value = 0;
       }
4.     $("#" + offset).val(value);
   }
```

Figure 3.2: An example jQuery command.

In this example, an HTML number input's value is being rounded to the nearest even number (while also accounting for over and underflow) when the input is changed by the user. Step 1 is an on-change event handler based on the input id "window_offset_x-init". The function "round_x_offset" is called with same id as its input when the event handler detects a change is detected (Step 2). Using the jQuery val()

API, the current value of the HTML input is pulled (Step 3), modified, and reassigned to the input id (Step 4). JQuery was used as an integral part of the frontend user interface JavaScript design because it provides functionality such as this example for easy interface control.

Chapter 4

DESIGN OF GRAPHICAL FRONT END

The following chapter will describe the different pages of the PING frontend and their functionality.

4.1 Index Framework

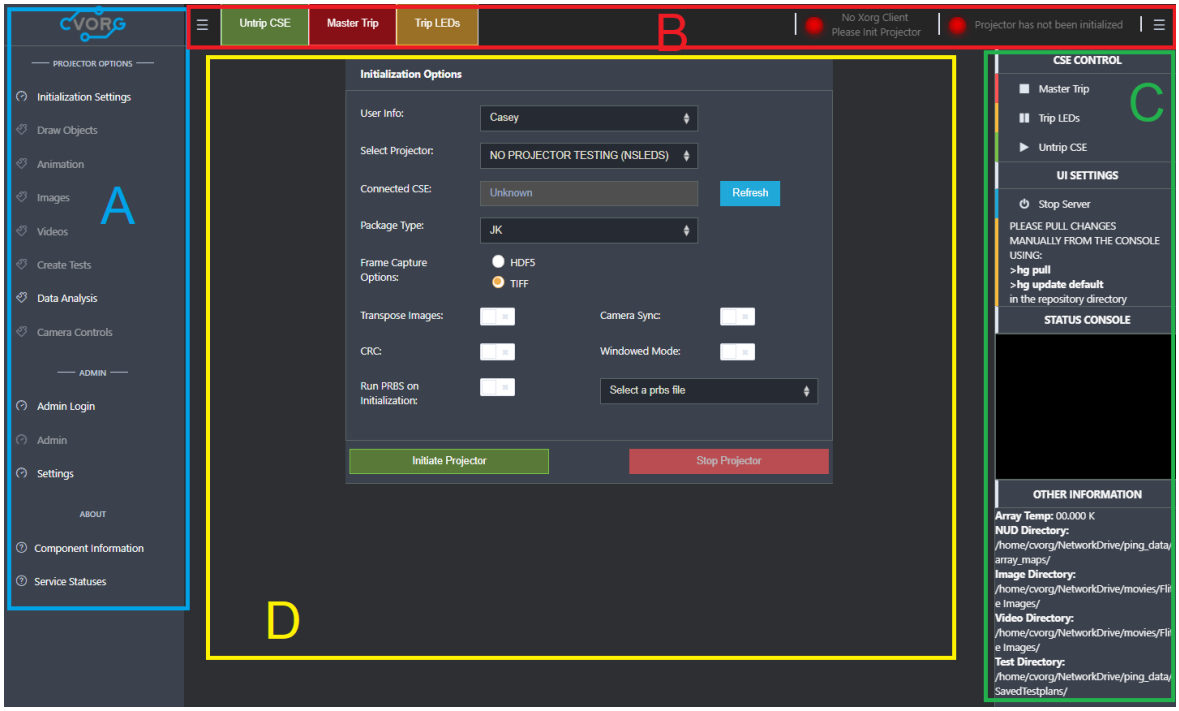

Figure 4.1: The frontend index framework. Section A is the tab selector. Section B is the CSE Control panel. Section C is the system information/control sidebar. Section D is the tab container.

The index framework is used as the main architectural template for the frontend. As discussed in the previous chapter, the index framework (index.html) loads

the individual tabs and displays them inside a container. The main functionality of the index framework page is for system control, feedback, and navigation.

The first section of the index framework (Section A in Figure 4.1) is the tab selector. The tab selector allows the user to navigate between the different frontend pages. When a new tab is selected, the frontend hides the currently displayed tab and loads the new tab into the HTML container area (Section D).

Section B of the index framework is the CSE control panel. The CSE control panel allows the user to remotely trip and untrip the CSE hardware controller. If the system is tripped, no current is sent to the projector, so no images are displayed. There are two types of CSE trips: the master trip and the LED trip. The master trip button trips the LEDs and the DAC cards. The LED trip button only trips the projector LEDs. This trip mode allows the user to observe current and voltage readings on the DACs without writing to the projector. The CSE control panel also contains two status bars. The first (left) is the sync status. The sync status tracks whether or not the system is synced and running at the correct framerate. If the system is synced, the status light is green; if the system is not synced, the status light is yellow; if the projector hasn't been initialized, the status light is red. The second status tracks the CSE trip status. If the system is ready (untripped), the status light is green; if the system is tripped, the status light is red and the reason the system was tripped is also displayed; if there is communication problems with the CSE, the status light is yellow.

Section C is the system information/control sidebar. The sidebar contains another set of buttons to trip/untrip the system and a button to kill the PING server remotely. Below these buttons is the status console. The status console provides feedback from the backend to the user. The status console will be covered in more detail later. Below the status console are a list of directories where useful files can be located. Most of the directories listed are folders where PING saves different types of collected data.

4.2 Projector Initialization Tab

The projector initialization tab is used to start and stop the projector. The projector initialization screen is the tab selected in figure 4.1. The projector selected by the user dictates how the backend is configured for image processing. The projector can also be initialized in windowed mode to display images in a smaller window on the projector (this will be discussed in more detail later. Other important options that can be selected in projector initialization include transposing images, camera initialization, and HDMI card testing using PRBS. Once the projector has been initialized, a second window will appear below the initialization settings. The second window can be used to change the modeline (framerate) and to change the window offset if windowing was enabled on start-up. The projector can be stopped and reinitialized to change the settings.

4.3 Draw Objects Tab

The draw objects tab allows the user to draw basic shapes on the projector. The object types can be selected in Section A in Figure 4.2. Multiple objects can be selected at once. The properties of the selected objects can be modified in Section B of the tab. These properties include size, location and color. The start, end, and spacing of grids are also set here. Whenever a new object is selected or a property is changed, the screen preview (Section C) is updated. The screen preview is generated in the backend. It shows a preview of what the projector will display after the new objects are drawn. Once the draw objects button is pressed, the new objects are displayed on the projector. The objects are displayed as a static image and remain on the projector until they are cleared by the user. If the array was not cleared, the new objects overlap the previously drawn objects.

4.4 Animations Tab

The animations tab displays frame patterns that are generated in real-time. Unlike the Videos tab, which processes all frames before display, the animations tab

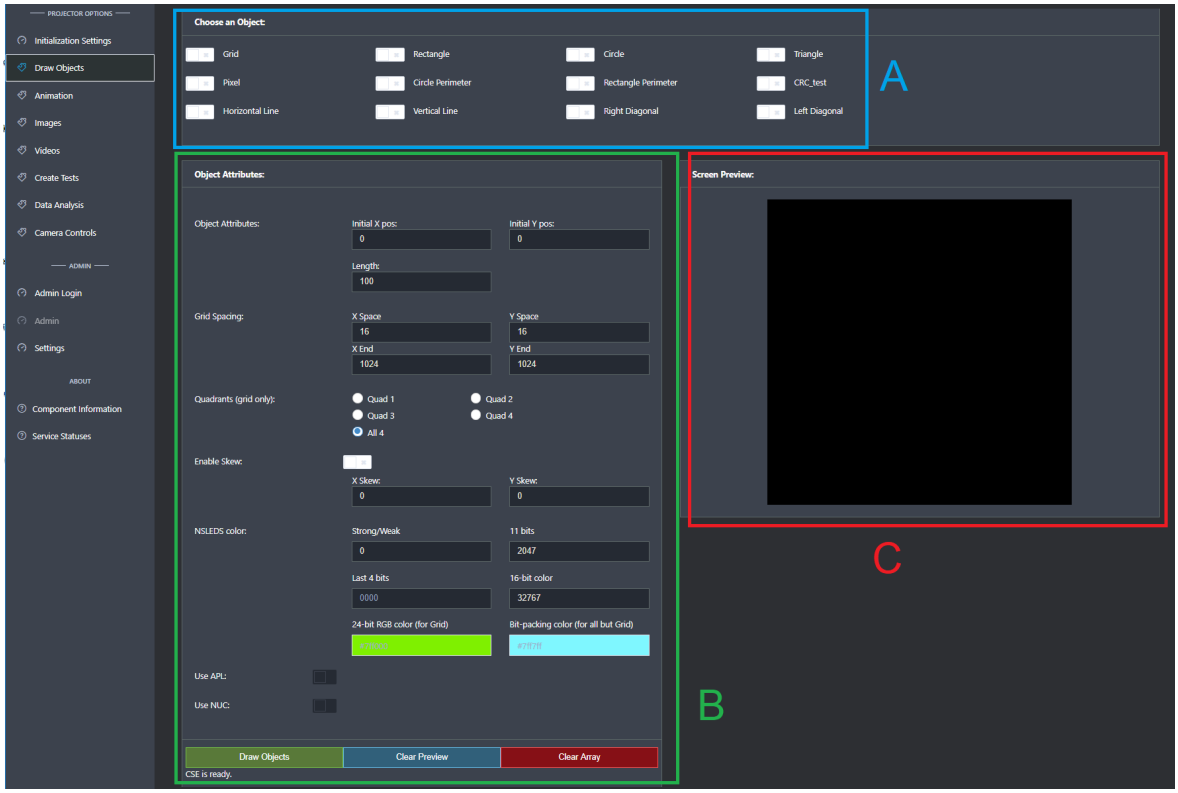

Figure 4.2: The Draw Objects tab. Section A is the object select. Section B is the object configurations. Section C is the screen preview.

generates and then displays each frame sequentially. This can be difficult to accomplish at high framerates, since generating the frames can take a significant amount of time. For example, frames displayed on a 1Kx1K pixel projector take about four microseconds to generate. At a framerate of 120HZ each frame must be displayed in five microseconds. Besides generating the frame, this five microsecond period also includes sending the frame to the hardware, displaying the frame on the projector, and removing it so the next frame can be shown. Since the frame takes four microseconds to generate, only one microsecond is left to perform all the other tasks. If the frames take too long to be displayed, the system will be desynchronized and fail to show any output. As the projector size increases, the frame generation time increases. Therefore, the larger the projector, the lower the max framerate for the animations. Despite this, the on-the-fly scene generation is an important aspect of the projector system.

The first animation in the animation tab is the bouncing ball. The bouncing ball was the first animation created. It generates a circle object which bounces around the screen, reflecting off of the edges of the projector.

The second animation is used for camera mapping. The camera mapper is used to correlate the projector output to the captured camera input. Since the projector and camera resolutions can both be variable, a map is used to predict the exact location of pixels in the camera capture. The camera mapper displays single pixels with known locations to find small regions-of-interest (ROIs) representing the known pixel locations in the camera capture [5]. This model can then be used to estimate the location of any pixel on the array. Figure 4.3 shows an example camera map plot. The squares represent generated ROIs wrapped around the drawn pixels. The color of the ROI lines indicates the intensity of the pixel.

The third function of the animation tab is non-uniformity detection (NUD). NUDs are used to map the brightness model of the array. Every pixel is swept over a range of desired brightness values. The camera mapper captures the output brightness and uses the output data to find pixels whose performance is non-uniform. [5] (Example: figure 4.4).

The final animation in the animation tab is the pixel sweep. The pixel sweep is used for data analysis of individual pixels. The user selects a pixel or set of pixels to be observed individually. Each pixel is swept at increasing brightness values. The camera mapper is used to capture the output brightness of each pixel.

4.5 Images Tab

The images tab is used to load and display static images. These images can be default file types such as JPEG, PNG, etc. The Image Tab can also read special IMG files. These IMG files are 2-dimensional arrays containing individual frame pixel values. The file is selected using the file browser (Figure 4.6). The file browser is opened by clicking the browse button located in Section A of Figure 4.5 (The Images Tab).

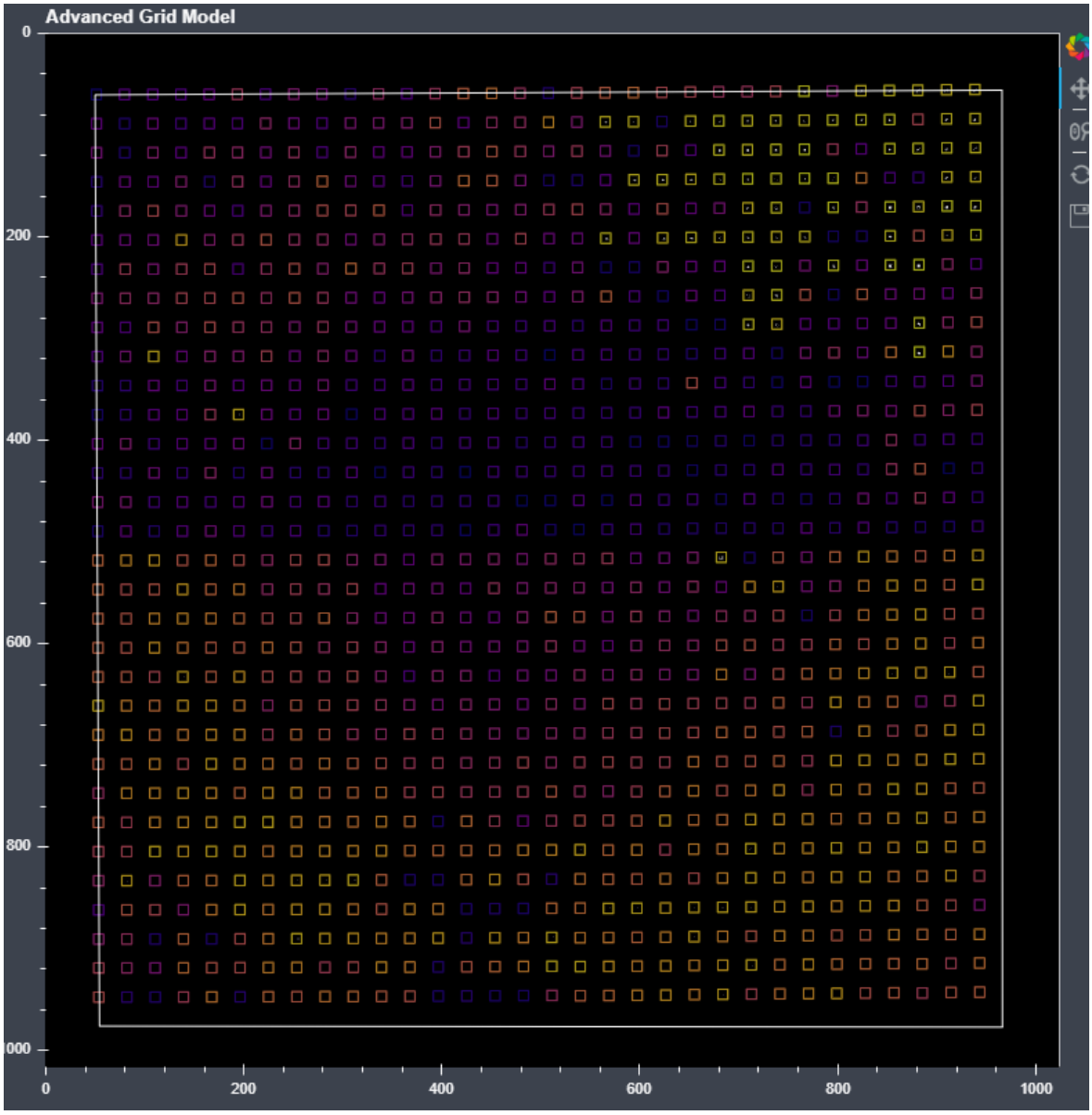

Figure 4.3: An Example Camera Map.

The file browser allows the user to navigate through the file system on the NUCPC. The NUCPC is connected to a network storage drive containing data collected and saved by PING as well as images and movies used to test the projector. Directory navigation is handled by an administrative controller in PING. Both the administrative controller and the network drive will be discussed in later sections.

Once an image has been selected, the backend generates a preview of the selected

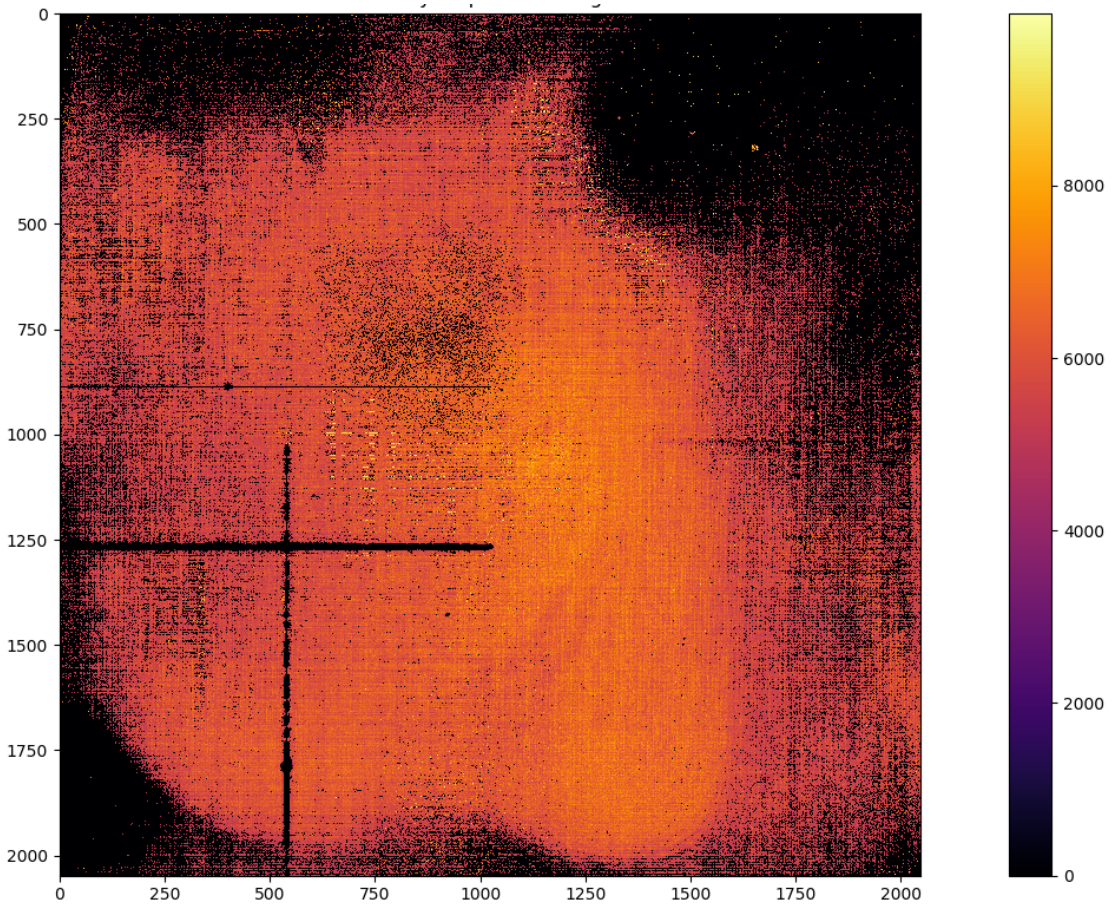

Figure 4.4: An Example NUD captured from HDILED1. The color indicates the radiance for each pixel on the array. The hotter the color the higher the captured brightness.

image that is displayed in Section C in the images tab. The image can be manipulated using the tools in Section B of the tab. The possible image manipulation tools include offset (brightness), gain (contrast), image flipping, image rotation, and origin shifting.

4.6 Videos Tab

The videos tab generates movies and displays them on the projector. Like the Image Tab, Section A (Figure 4.7: The Videos Tab), has a file selector. The videos tab can read mp4 files, image array formats, and preprocessed frame files. Preprocessed frame files contain serialized frame data that is ready to be displayed on the projector. Once a file is selected, the first frame of the movie is displayed in the video preview

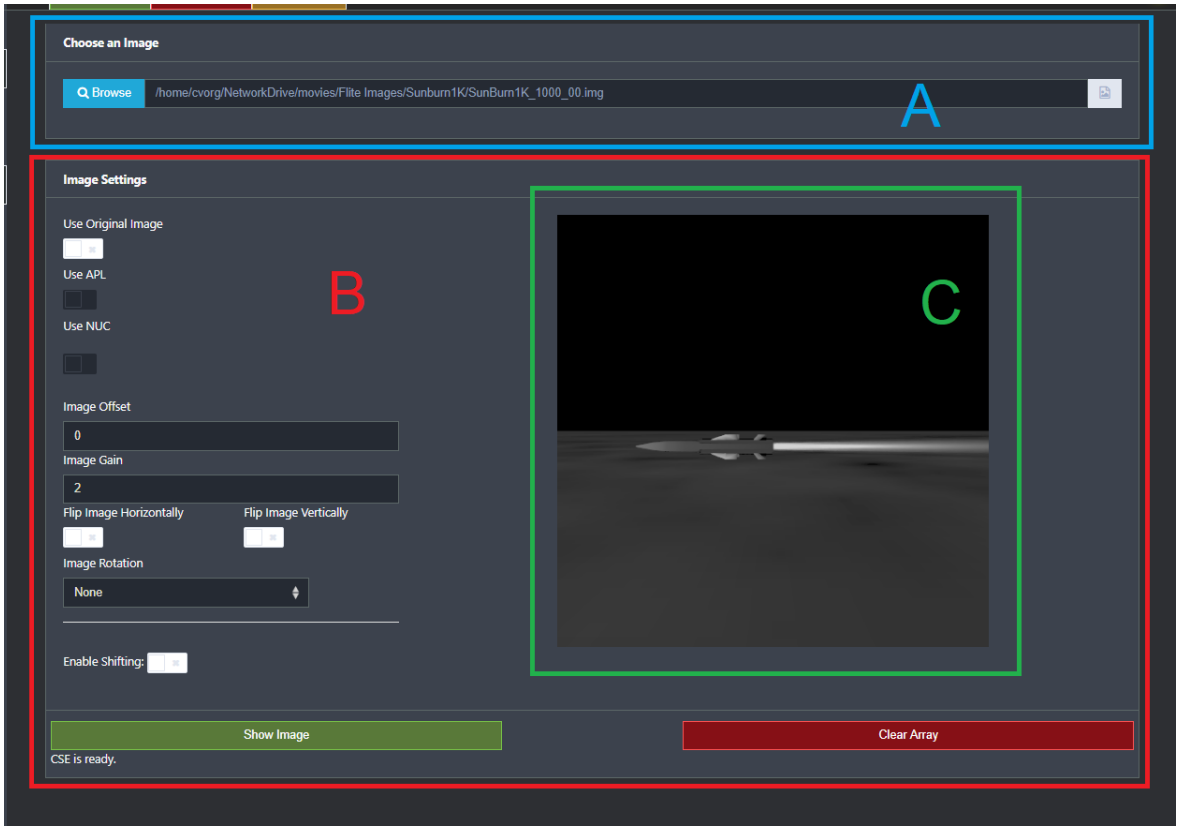

Figure 4.5: The Images tab. Section A: Opens browser and shows selected file. Section B: Image settings. Section C: Image preview.

Figure 4.6: The file browser.

box (Section C). Clicking the "Preview Video" button in Section B will play the first 100 frame of the movie in the video preview box

Section B, the video settings, has most of the same options as the Images tab.

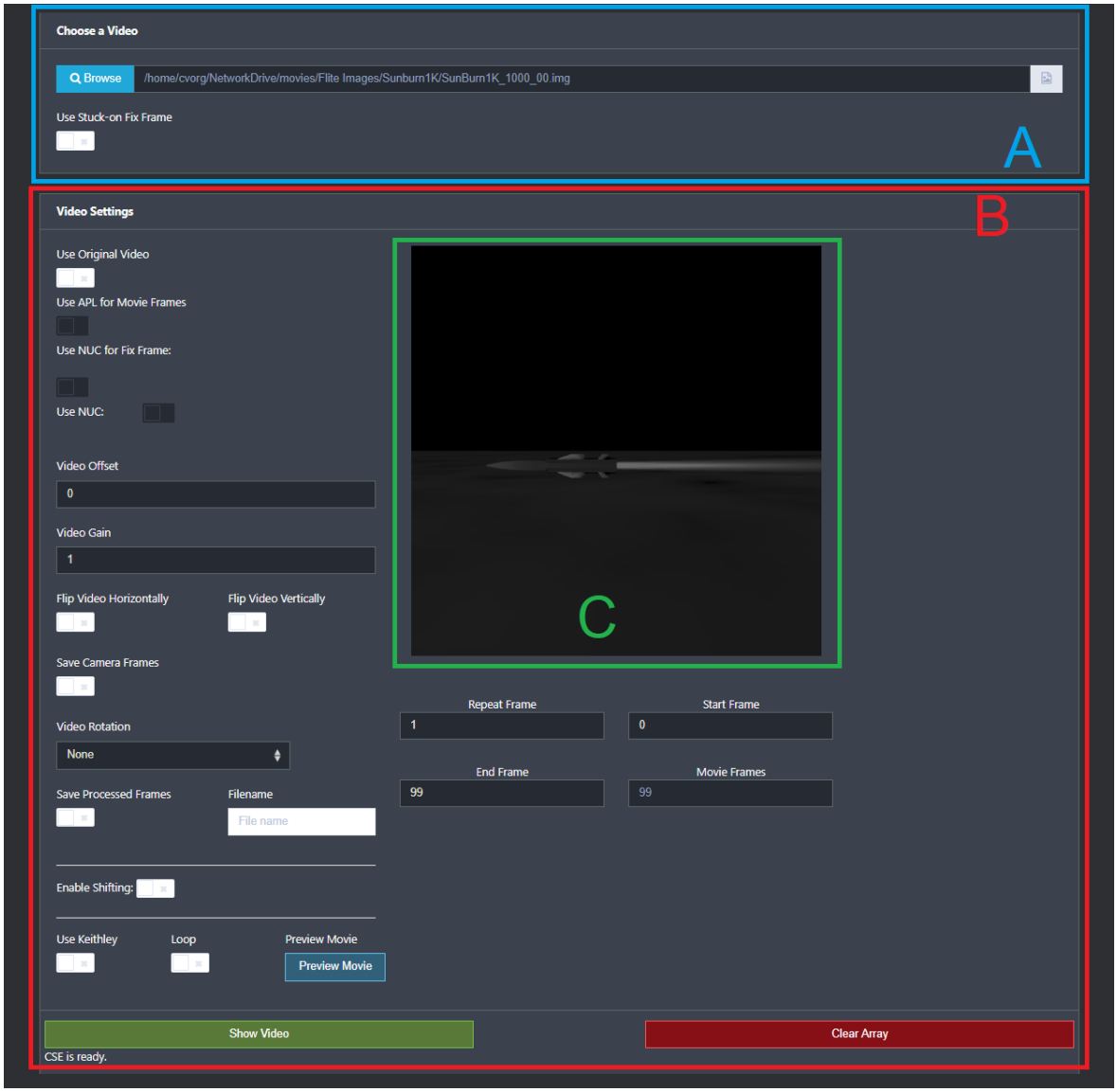

Figure 4.7: The Videos tab. Section A: Opens browser and shows selected file. Section B: Video settings. Section C: Video preview.

The videos tab also has off frame options and the ability to save preprocessed frames. Off frames are blank frames periodically inserted into the video. The user determines how many off frames appear and how often they appear. The off frame options are part of the video settings in Section B. The user also has the option to create an off frame with a unique pattern by selecting the "Use Stuck-on Frame" box in Section A. This box opens another window that is used to create a custom pattern for the off

frame.

When the user clicks "Show Video" the backend processes the movie frames so they can be displayed on the projector. Unlike animations, the movie frames are all processed at once and then displayed on the projector. Since this takes time, the user has the option to save the processed frames to a file so they can be loaded and displayed quickly in the future.

4.7 Test Tab

The test tab is a design suite that allows the user to create a fully customized preprocessed frame pattern. The frame pattern is built using the same objects available in the draw objects tab. The test tab uses these objects to perform color sweeps, position sweeps, size sweeps, or a combination of these different sweeps. Objects can also be displayed statically for a given number of frames.

Each test is composed of sets of sweeps called "frame objects". These frame objects run their sets of sweeps simultaneously. A frame object can have any number of sweeps, but the duration of a frame object is based on the longest. For example, if a sweep lasting for 100 frames and another sweep lasting for 200 frames are in the same frame object, the frame object will last for 200 frames. If a sweep in a frame object is shorter than its frame object, the user can keep the last frame statically displayed or clear it from the projector. Below is an example test setup:

In the example (Figure 4.8), a test with two frame objects is created. The first frame object has four separate objects: a square drawn in the top left, a square drawn in the top right, a square drawn in the bottom left, and a square drawn in the bottom right. Each object is surrounded by blank frames. The placement of the blank frames and the drawn squares will make the test appear to move a square around the four corners of the projector. Each square/blank frame sweep is repeated ten times. The first frame object lasts for forty frames since the four sets of squares/blank frames are displayed at the same time.

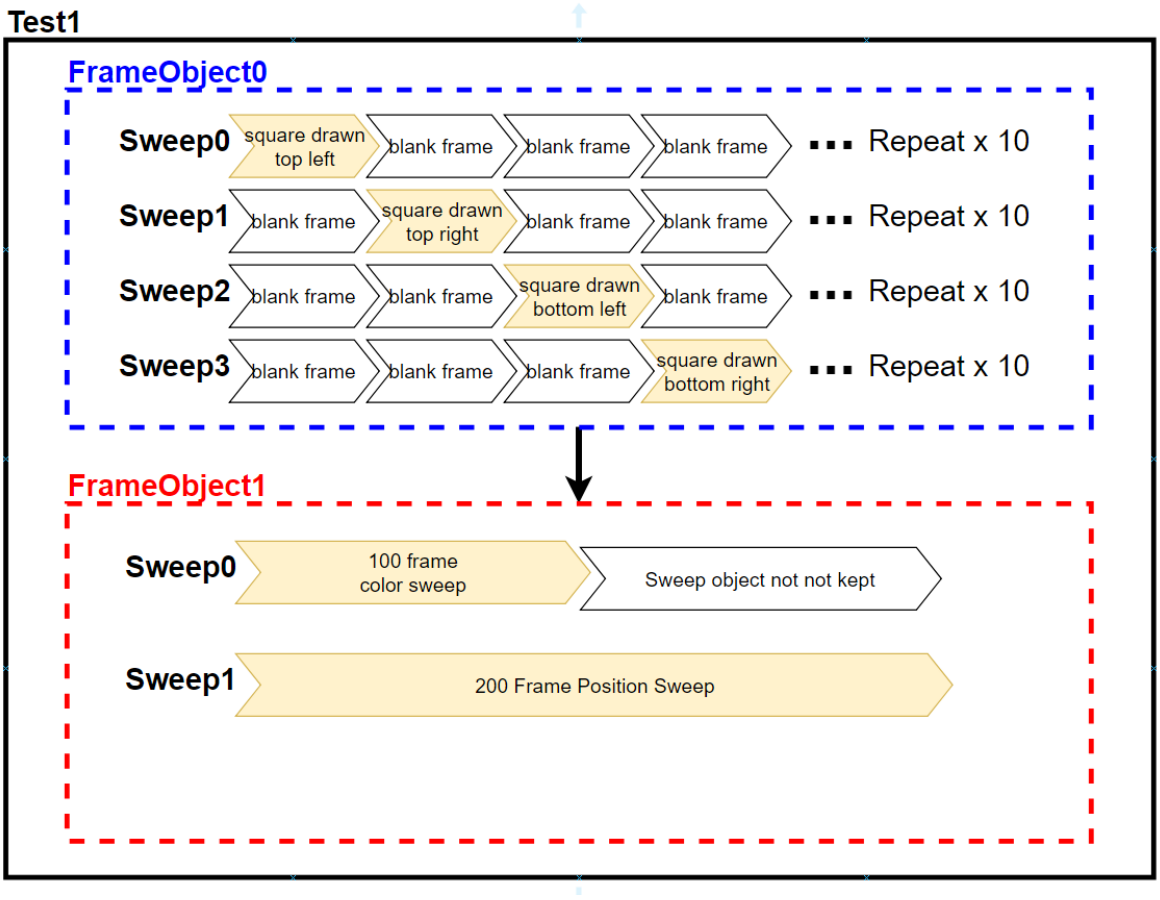

Figure 4.8: Example test framework.

After FrameObject0 has finished displaying, the test then displays FrameObject1. This frame object contains two sweeps: a color sweep lasting 100 frames and a position sweep lasting 200 frames. Both sweeps run at the same time. Since the color sweep (Sweep0) finishes first and the last frame is not kept on the screen, the remaining frames of the position sweep (Sweep1) continue playing by themselves. When the last sweep has finished, the test is complete.

4.8 Data Analysis Tab

The Data Analysis tab is used to analyze data captured by PING. PING captures and saves frames recorded by the IR camera. The frames are saved into data files and stored on a network drive the NUCPC is connected to. The data analysis tab reads

these files and uses the frame data to produce a data plot. The user selects a file using the frontend file browser and enters information about the file, such as how many total frames there are and how many blank frames are in the start and end of the file. The user then chooses the dimensions of the area that they want to observe. The backend then uses these configuration settings to generate a plot of the apparent temperature and radiance based on frame number. The plot is generated using a Python library called bokeh [1]. The bokeh plot is intractable, allowing the user to zoom in and out of the plot and save it to their machine. An example bokeh plot can be seen in figure 4.9.

4.9 Camera Control Tab

The camera control tab displays the current camera settings and allows the camera settings to be changed. Two of the most important functions of the camera control tab are Auto-NUCing and maximum framerate calculation. Auto-NUCing automatically configures camera light absorption and helps prevent non-uniformity camera during camera capture. The maximum camera capture framerate is calculated by the window size.

4.10 Administration Tabs

There are two administration tabs: the login tab and the administration settings tab. The administration login tab is used to authenticate whether the current user should have access to the settings in the administration settings tab. Once the correct authentication is provided, the user can open the administration settings tab.

The first function of the administration settings tab is to configure the maximum current level of different voltage lines on the projector system. If the maximum voltage levels are detected, the system automatically trips to prevent damage to the projector. If the maximum voltage level is set too high, damage to the projector is likely to occur. By putting this setting behind the user authentication screen, the chance of a max voltage value being set too high is reduced. The second function of the administration

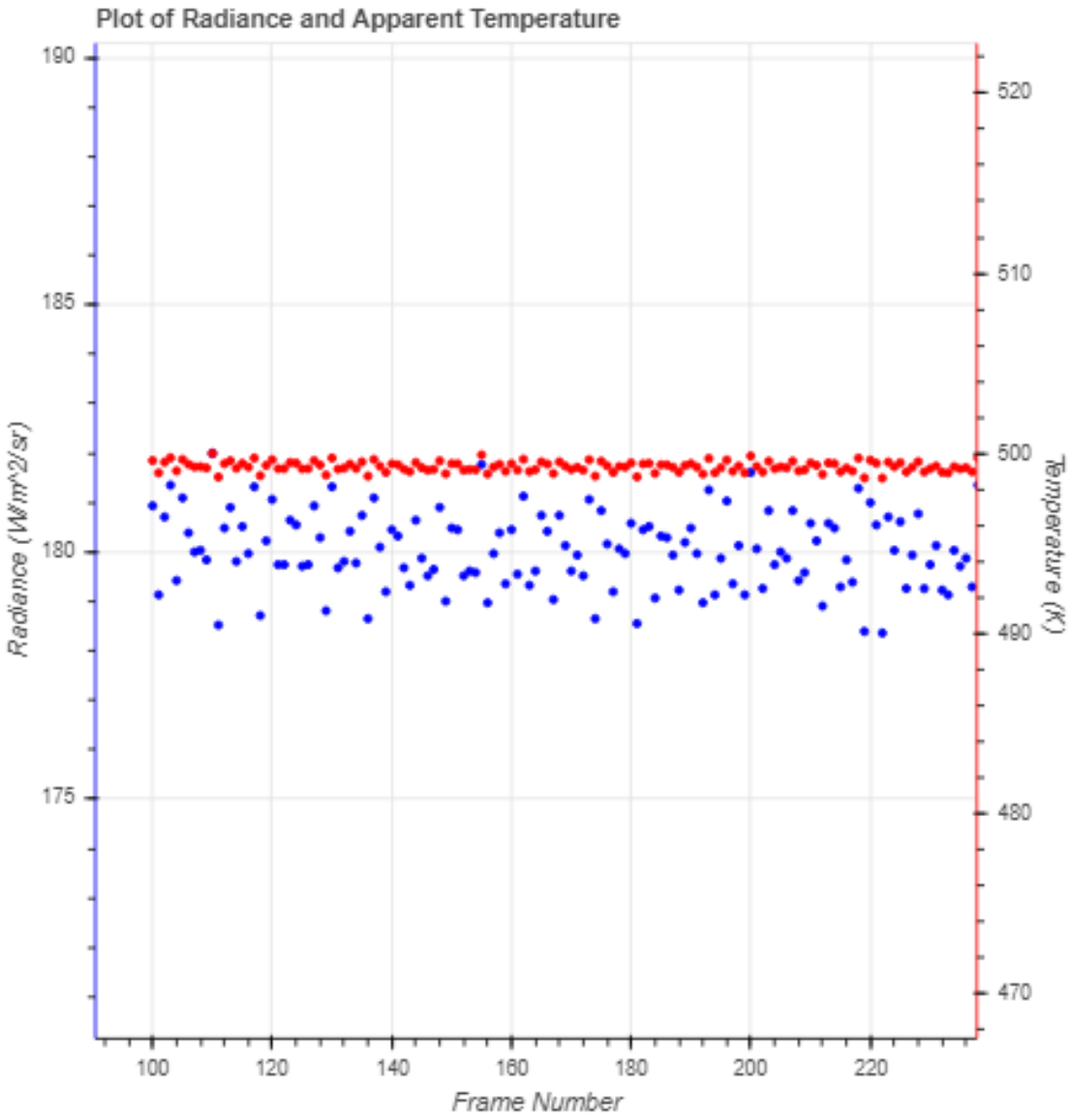

Figure 4.9: Example data analysis plot.

settings tab is to create or remove the user profiles used for record keeping. User profiles are handled by an administrative control library in the backend. The last function of the administration settings tab is to enable or disable the system's requirement of a temperature reader. By default, the projector system will not initialize unless a temperature reader is connected. The temperature reader is very important for monitoring and data collection. If the user absolutely needs to disable this requirement,

they can do so if they have the privileges needed to login to the administration settings tab.

4.11 Settings Tab

The settings tab is used to configure projector system settings that don't require administrative access. The first group of settings are the projector/CSE settings:

- Auto Trip: Enables/disables the projector automatically tripping if it has been idle for too long.
- Max Idle Time: The duration in seconds the projector can be idle until the system automatically trips.
- Auto Clear: If enabled, PING will clear the array after any video/animation has finished.
- Tripped Run Warning: If enabled, a pop-up window will appear when the user tries to display something while the system is tripped.

The next group of configurable settings are the firmware settings. The firmware settings are used to change the duration and delay of the write-enable signals in the firmware. The duration and delay are set using "tick" units, which are 10 nanosecond periods. The write enable duration determines how long the write enable signal is active. The write enable delay determines how long to delay the write enable signal being sent. This prevents the data timing from overlapping and getting corrupted.

The settings tab can also change the DAC mode and row. The DAC modes are used by the CSE hardware to run at different resolutions and framerates. The DAC row can also be changed to set which channels the DACs write to. The DAC row controls which row in each set of 16 DAC channels is written to first [2].

The last group of settings is used to configure the Keithley current readers connected over USB. The user first sets how many Keithleys are connected (0-2). Each connected Keithley must then be selected from the list of all serial connections detected by the NUCPC.

4.12 Component Information Tab

The component information tab displays detailed information about the different parts of the system. There are four sections of component information:

- Firmware Information: Reads the firmware information from the CSE and displays that information on the frontend. This information includes the firmware name, version, the CSE hardware ID, and the projector. The firmware information section can also run hardware tests to make sure different components are working properly.
- Xorg Service Information: Reads the Xorg service settings and displays them to the user. These settings include window size, set framerate, and sync status. The Xorg service (as well as all other services) will be discussed in chapter 5.
- CSE Information: Reads the main/secondary interface board IDs and voltage line settings.
- GUI Information: Displays a list of all updates to the PING repository so the user can keep up to date on all recent and past changes.

4.13 Service Status Tab

Figure 4.10: The Service Status tab.

The service status tab tracks the current state of the backend services. Each service has a status light to indicate whether it is running or not. If the status light is red, the service is not running or the web interface is not connected to the service. If the status light is green, the web interface is connected to the service. If the light is yellow, the status is unknown. Each service status also lists detailed information about

the service settings. The user also has the ability to start, stop, restart, or manually poll the status of each service. The service statuses are updated every 10 seconds.

Chapter 5

INTERFACE BACK END ARCHITECTURE

The backend of the interface is written in Python. The backend runs on the NUCPC and acts as the interface between all the separate components of the system (such as the CSE, camera, etc). The NUCPC also contains a directory called the Network Drive which is used for file storage and data collection. There are two main sections of the backend: the controller and the services. The controller is responsible for handling user input, communicating with the services, and controlling the projector. The services are Python applications designed to run constantly on the host machine (the NUCPC). These services run independently of the user interface. Connections to the services are established using ZeroRPC clients.

5.1 Controller Overview

The backend controller hosts the local web server on the NUCPC. The web server controls the projector based off user input. Once the server is initialized, in runs indefinitely unless manually stopped.

5.1.1 The Web Interface

The web interface file (web_interface.py) is the main file of the backend. This file contains the server class, which creates the web server and hosts it on localhost port 8087. On this port, the web interface maintains the WebSocket connection and handles requests from the frontend. The web interface also creates the zerorpc clients to communicate and utilize the services. The functions of the web interface class are organized into different sections.

The first group of functions in the web interface are responsible for handling incoming file requests from the client, as well as functions to start and stop the server. When the client first connects to server, a default request is sent, and "index.html" is sent as a response. The client then requests the necessary HTML, JavaScript, CSS, and image files. The web interface routes these requests based on the folder location of the given file (for example, JavaScript files for the individual HTML tabs are located in the /js/tabjs/ folder in the PING directory).

The next group of functions are responsible for initial projector configuration. The main function of this group, init_DVI(), initializes the projector based off of the user's chosen settings. If an error occurs during any point of the initialization process, the user is alerted immediately. Three of the most important steps during projector initialization are creation of the DVI module, connection to the Xorg service, and connection to the camera (these will all be discussed in more detail later. The initialization functions are also responsible for setting the frame rate and window size of the projector.

The next groups of functions handle the different type of image displays on the projector. The functions for drawing objects generate patterns based off of the desired objects, color, size, and location. These objects are layered on to a screen array that overlaps the previously drawn objects. The animation functions generate and display the desired animations in real time. The image and video functions load the selected files, make the desired changes (rotation, offset, gain, etc), process textures that can be displayed on the projector, and display the static texture or dynamic movie. The final group of display functions are used to generate the custom tests designed by the user in the Test Tab.

The final group of functions in the web interface are the routing handler functions. The routing functions are responsible for processing the input data from the client requests and calling the necessary python functions to accomplish the desired tasks. All the routing functions expect a JSON array variable called "data" that holds the parsed data received from the client. If a response is expected by the client, the

routing functions return the desired output to be packed into another JSON array.

5.1.2 The DVI Module

The DVI module (Dvi_functions.py) is responsible f or generating the frames that are displayed on the projector. It generates the frames for the Draw Objects, Show Image, Show Video and Test Tabs. The main class in the DVI module is the DisplayModules class. An instance of the class (called self.dvi_module) is created in the init_Dvi() function in the web interface.

The frames are initially generated using NumPy. NumPy is a Python library used for scientific computation. The advantage of NumPy is that it is very efficient and can perform operations at high speeds [7]. PING uses NumPy to generate and represent the projector frames as NumPy arrays. Each frame is a 2D array of color values used as the desired output of the DAC cards to the pixels. A movie is composed of an array of these frames. After the NumPy frames have been generated, the DVI Module bit packs the frames and the GLFW display converts the bitpacked frames to textures that can be used by the firmware (both of these processes will be discussed later).

5.1.3 Administrative Management

Administrative control is handled by the admin_control.py library. The web interface contains an instance of the admin management class to access it functions. The admin controller in PING is responsible for user management, repository information, and directory traversal. The user data is stored in a JSON file called "userData.json" inside the Network Drive. Users can be added or removed from this file. Each authorized user has their own entry in the data file. The user entry contains the saved settings from the last time the user initialized and operated the projector. For example, the saved values under the user's initialization settings include the projector they used, the package type, and which checkboxes were selected.

When the client selects an option from the user select in the Initialization Settings tab, the web interface requests that user's saved information from the admin manager. The admin manager loads and parses the user data file and sends the requested information back to the user. When a user's settings need to be updated, the web interface passes the new settings to the admin manager. The admin manager then loads the user data file as a JSON array and updates the user's settings. Finally, the admin controller overwrites the original file with the edited JSON array containing the new user settings.

The admin control class also generates the repository information table in the Component Information tab. The admin controller iterates through all the change sets in the PING repository, generating an HTML table entry for each change log. The list of change logs is returned to the web interface so that it can be displayed on the frontend.

Like the repository information table, the admin controller also generates HTML for the file selector in the frontend. When a new directory is requested by the client, the admin controller returns a string of HTML code containing all of the file and directories in the desired directory. The admin controller iterates through the selected directory and puts HTML wrappers around each file or directory. The list is sorted in alphabetical order and excludes hidden files within the directory. When the client requests a file, the admin controller checks to make sure the specified file type is valid for the desired operation. For example, if the user is selecting a file for the Image Display tab, they will not be able to select non-image file types.

5.1.4 EDT PDV Controller

EDT PDV is the driver to control the camera card in the NUC PC. In PING, PDV is used to capture images from the camera. The PDV controller interfaces with the frame grabber in the NUCPC to send capture commands to the camera. On initialization, the PDV class configures the settings on the frame grabber to match those of the FLIR camera. Once the setup is complete and the connection is established, the

PDV controller is able to send commands to the frame grabber to pull frames from the FLIR camera's memory buffer.

5.2 Communication with the frontend

Communication with the frontend is the most important job of the backend controller. In order to efficiently run the system, the WebSocket communication needed to quickly and accurately pass along a broad spectrum of requests to the backend.

On initialization, the web interface creates an instance of PING's Server class. The server class creates a Bottle object called self._app, a distinct web application that handles routing. Bottle is a Web Server Gateway Interface (WSGI) [9]. It is enclosed in a WSGIServer. The WSGIServer acts as the listener for the desired port, and passes the requests to the Bottle object. The Bottle object uses the function _route() in the Server class to route incoming requests from the client. The web server runs until the stop function is called or the web interface is killed.

5.2.1 Routing

The WSGI routing function handles the following requests (Asterisks are used in place of specific file names):

- /: initial request sent when user first attempts to access port 8087. Returns the index.html file.
- /*: request for html file (ie init_tab.html). Returns said file.
- /websocket: request for the interface to carry out a specific task a specific task (this will be covered in the next section).
- /js/*: request for non tab-specific JavaScript file (ie jQuery).
- /js/tab_js/*: request for tab-specfic JavaScript file (ie init_tab.js).
- /css/*: request for CSS file.
- /img/*: request for image files displayed in the GUI.

5.2.2 JavaScript Websocket Routing

Once index.js has been received by the client, it sends a request to create a direct connection to the host. The client creates a JavaScript WebSocket object to establish a connection to the host using the /websocket route in the Bottle routing function. This WebSocket object is used to send a data packet to the host to perform a specific task.

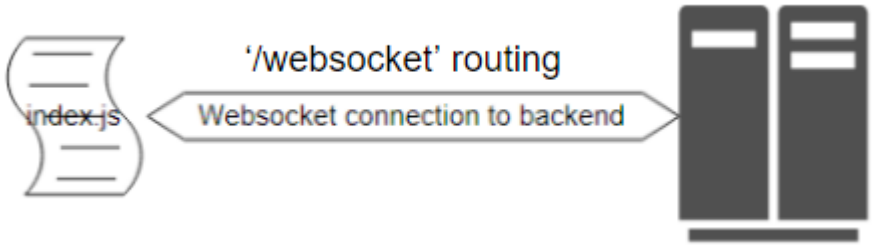

Figure 5.1: The WebSocket link.

The packet being sent across the web socket is a stringified JSON array which contains two keys:

1. Routing: the name of the action to be performed on the server side. Routing tells the host which action is performed on the backend.
2. Data: the data needed to perform that function. Data varies depending on what task the client is attempting to perform. Data is a JSON object comprised of key-value pairs of input data.

The packet's contents must be converted to a string before being sent by the web_socket, otherwise the web socket will not be able to send the data.

If the data needed for the desired routing task is taken from a form, or no input data is needed, the packet is prepared using the function "prepare_websocket_data" in "index.js". If the data is being pulled from the inputs of an HTML form, "prepare_websocket_data" uses jQuery to pull the input names and values and automatically assigns them to a JSON array. If no inputs are needed, "prepare_websocket_data" generates an empty JSON object. If the necessary inputs are not inside a form, the data object for the packet is generated manually in a separate JavaScript function.

Once the packet has been generated and sent by the client, it is received by the WSGI server within the host. Since the request from the client was received using the "websocket" route, the packet is passed along to the WebSocket route handler. The WebSocket route handler parses the received data and converts it back to a JSON array. It then uses the routing value in the JSON array to call a corresponding routing function. The routing function unpacks all the values in the packet's data object and carries out the required tasks for that specific route.

If the task requested by the client expects a response back, the routing function returns the output value to the WebSocket route handler. The WebSocket route handler creates a packet with the same structure as the client, using the original routing name and the data returned from the routing function. The response packet is then sent back over the WebSocket. When a response is received by the client, a response handler API uses a switch-case statement to handle the response based on the routing in the packet. The next section demonstrates an example WebSocket communication.

5.2.3 Example WebSocket Communication

The following example demonstrates the communication process for initializing the projector.

The first step (Figure 5.2) of the WebSocket communication is initiated by the user interface. The user interacts with the frontend (in this case the "Initialize Projector" button) to carry out a desired task. When the button is pressed, the HTML object initiates the "init_proj_click" JavaScript function. This function begins the preparation of the packet.

In the next step, the packet is generated. The inputs on the Initialization Tab are inside an HTML form. Therefore, the input values can be retrieved using the "prepare_websocket_data" function. In this case, the form name will be "init_form" and the routing name will be "initDVI". The names of the inputs and their values are pulled from the form and placed into a JSON array. For example, the projector being used is called "HDILED1", so the data object has an entry with the key "projector"

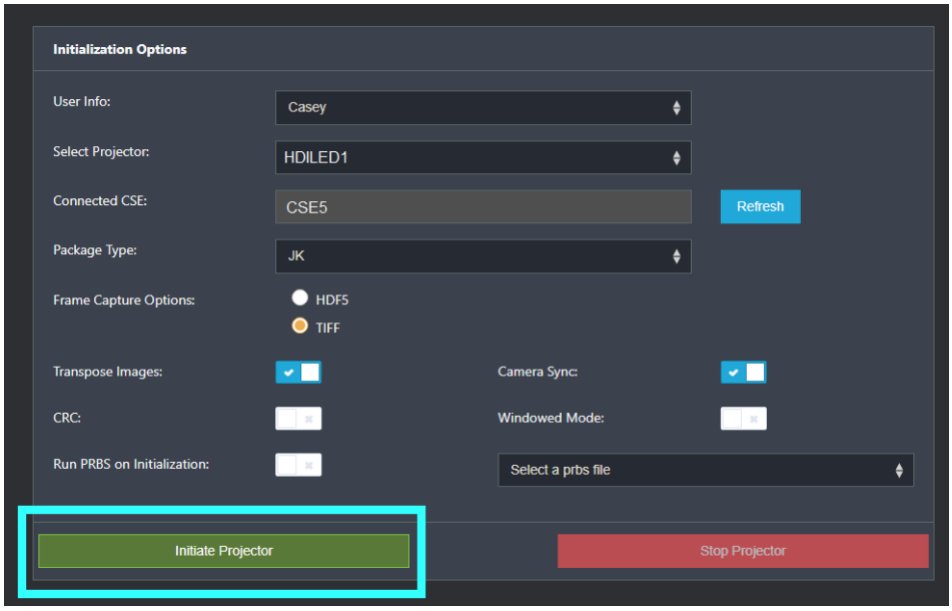

Figure 5.2: Step 1: The user presses the "Initialize Projector" button.

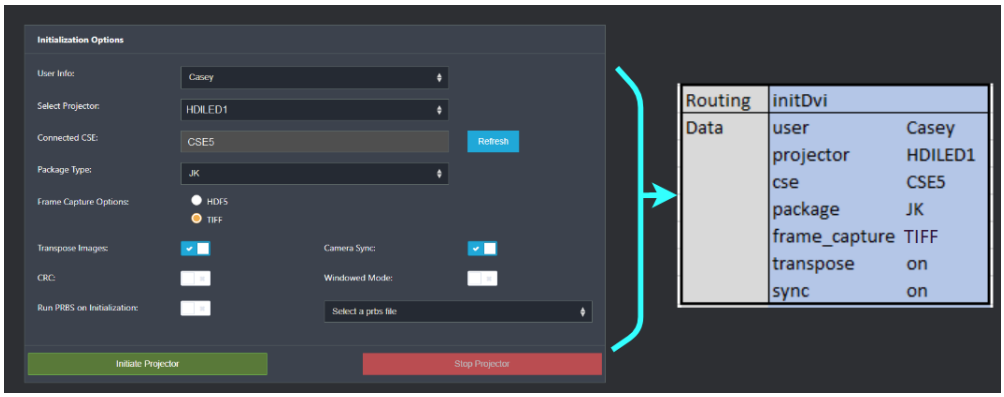

Figure 5.3: Step 2: The packet is created.

and the value "HDILED1". Checkboxes and switches are handled differently than normal inputs. If a checkbox/switch is selected, it is added to the data object with a value of "on". If it is not selected, it is not added to the data object.

The form data pulled from the page is assigned to the value of the packet's "data" key. The value of the "routing" key is set to the string "initDVI", since that was the input given from the "prepare_websocket_data" function. Figure 5.3 is a visualization of the packet created from the HTML page. The gray boxes are the keys and the blue boxes are their values. the data value has the form key/value pairs set across

from each other.

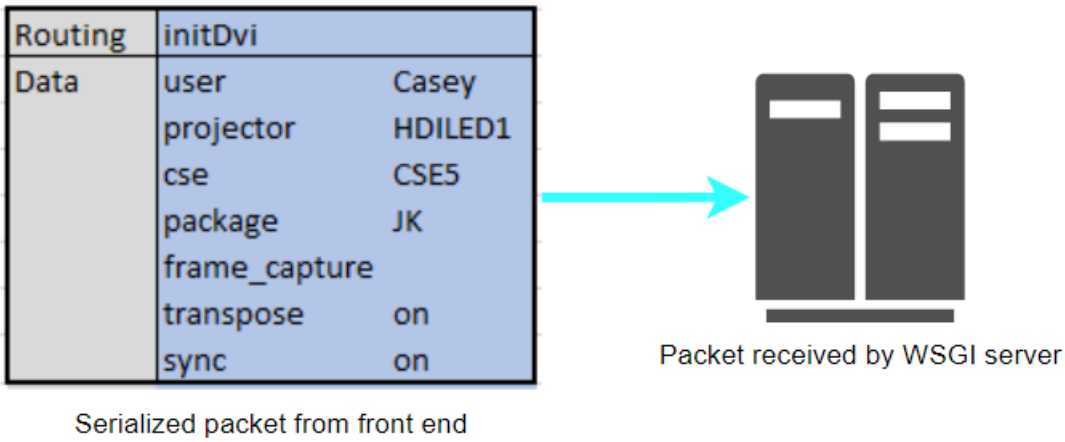

Figure 5.4: Step 3: The packet is transmitted to the backend.

Once the packet has been created, it needs to be sent over the WebSocket. The packet cannot be sent over the WebSocket if it is not converted to a string. Once it has been serialized, the packet is sent along the dedicated JavaScript WebSocket link. The packet is then received by the WSGI server in the web interface (Figure 5.4).

```
def handle_websocket(self):
    self.wsock = request.environ.get('wsgi.websocket')
    if not self.wsock:
        abort(400, 'Expected WebSocket request.')
    while True:
        try:
            message = self.wsock.receive()
            loaded = json.loads(message)
            reformatted = self.reformat_JSON(loaded)
```

Figure 5.5: Step 4: The packet is loaded and reformatted.

The packet received by the WSGI server was sent over the "websocket" route, so it is passed to the websocket handler function. Figure 5.5 shows the start of the websocket handler function. First, the message is pulled from the WSGI socket. Then, the data is converted from a string back to a JSON array. It is then reformatted so the values can be accessed more easily.

After the data has been reformatted, the getattr Python API is used to call the necessary routing function (Figure 5.6). Getattr uses a string to return the attribute

with that name in the specified scope. The websocket handler uses getattr to return a function call attribute. In this case, the desired function call is the routing name "initDVI". The reformatted data variable is passed as the argument for the "initDVI" function.

```
result = getattr(self, reformatted['routing'])(reformatted)
```

Figure 5.6: Step 5: The desired routing function is called and the reformatted data is passed into it

The "initDVI" routing function is responsible for carrying out the user's desired task using the given data. In this case, "initDVI" initializes the projector, updates the user's saved Initialization settings in the admin manager, and checks the CSE's trip status. It then returns the output status of the initialization attempt. If the initialization succeeded, the status will return a dictionary with an initialization status of True, the window dimensions, and no error message. If the initialization failed, the status will return a dictionary with an initialization status of False, no window dimensions, and an error message indicating where the process failed. This example will assume initialization passed.

```
def initDvi(self, data):
    #if gui_testing paramter is false then initializate projector normally
    if (not self.gui_testing):
        status = self.init_DVI(data)
        self.admin_management.update_user("Initialization Settings", data)
        self.send_cse()
        return status
```

Figure 5.7: Step 6: The initDVI routing function.

The initialization status JSON object is returned to the routing function. The routing function then creates the response packet sent back to the frontend. The routing of the response packet is set as "initDVI" to match the request. The response packet status is set as the JSON object returned from the "initDVI" function. This response packet is then serialized and sent back along the WebSocket (Figure 5.8).

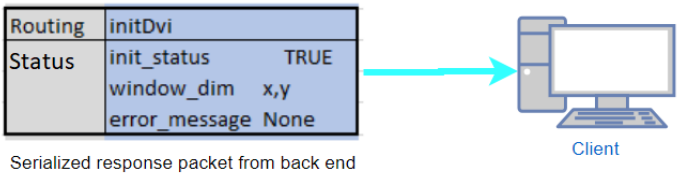

Figure 5.8: Step 7: The return packet.

The packet is received by the JavaScript WebSocket, which is listening for messages from the backend. The response is then sent to the response handler. The response handler parses the JSON string back to a JSON array. It uses a switch case to find what to do with the response. In this case, since the initialization status is True, the JavaScript unlocks the rest of the HTML tabs. If initialization status had returned False, the error message would have been displayed in a pop-up box.

5.2.4 Updating the Status Console

The frontend interface status console relays information about the current status of the backend. By providing feedback to the user, the status console helps keep the user up to date on PING's operation. The feedback lets the user know that their inputs are being registered by the backend. This prevents unnecessary command repetition. If a user presses a button and sees a message in console, they will not repeatedly press the button and send multiple instances of the same command to the host.

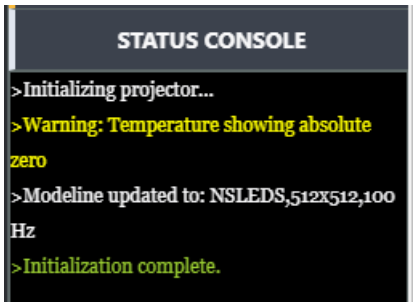

Figure 5.9: Example Status Console output.

The console feedback is updated in the web interface using the update_console function. The console is updated whenever a user command is registered, a major task is completed (i.e. projector initialization, modeline change, processing frames,

displaying frames) and when an error or failure occurs. The update console function takes a message and an optional color as arguments (both strings). Color guideline:

- Non-important update - no color (white)
- Success - green
- Significant failure or error - red
- Warning/important update - yellow

```
if hasattr(self, "dvi_module"):
    self.update_console("Initialization complete.", "yellowgreen")
    return {'init_status':True, 'window_dimensions':[self.width, self.he
else:
    self.update_console("Initialization failed.", "red")
    return {'init_status':False, 'window_dimensions': '', 'error_messag
```

Figure 5.10: Example status console call. During projector initialization, if the projector successfully initialized, update_console is called with a green success message. If the initialization failed, update console is called with a red failure message.

5.3 Service Architecture

The goal of the service architecture is to allow the independent services to run asynchronously as well as to enable encapsulation. Services run independently of the web interface. The services exist as part of the NUCPCs Linux path. Each service runs on a separate port in the 5000's on the NUCPC. The client libraries and shared libraries are also part of the NUCPCs python path.

There are three components of each service: the application, the service, and the client. The application allows control of the service directly from the Linux command line. From the command line, the services can be started, stopped, restart, and have their status checked. The service is the Python script run by the application. The service component is what runs on the NUCPC and is responsible for service function. The client script is used to create connections to the services so they can be controlled remotely.

5.3.1 Zero RPC

The services communicate over a remote procedure call (RPC) interface. Each service runs a ZeroRPC server on a specified port. ZeroRPC is a Python library used to implement RPC calls. In PING, it is used for communication between server-side processes.

5.3.2 The Xorg Service

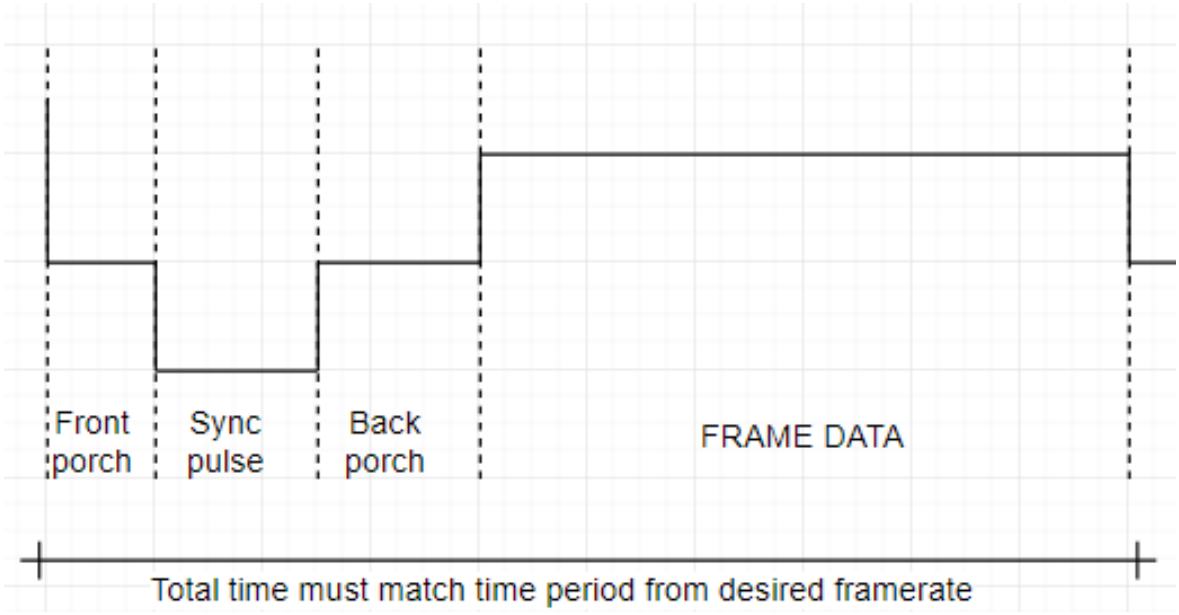

Figure 5.11: An example front porch, sync pulse, back porch pulse.

The Xorg Service was the first service implemented in PING. It runs on port 5001 on the NUCPC. The Xorg Service manages the Xorg Server, which controls the input and output of the HDMI display cards. The main functions of the Xorg Service are to set the projector modelines and maintain framerate synchronization. The projector modelines set the framerate and window size for the HDMI card. For a given framerate, each window dimension needs a specified front porch, sync pulse, and back porch signal length for the video to be displayed properly. The front porch occurs after the previous frame data has been displayed. The sync pulse is used to maintain framerate synchronization. The back porch occurs after the sync pulse, before the frame

data is displayed. These three values must be set such that framerate synchronization can be maintained and the data still has time to be displayed [15].

The Xorg Service contains a list of modelines for each projector. When the user initializes the projector, the list of modelines for the selected projector is sent to the frontend after initialization. When the user selects a new modeline, the Xorg Service configures the Xorg server with the new settings. The Xorg Service also monitors whether the HDMI cards are synced or not. If the service detects a sync loss, it alerts the frontend that an error has occurred.

5.3.3 The VidCap Service

The Video Capture (VidCap) Service is primarily used for testing the CSE HDMI cards and their custom firmware. It runs on port 5001. The VidCap service is responsible for frame display on loopback. Loopback is used to display the CSE output without passing the image data to the projector. The HDMI output ports on the CSE return the frames back to the NUCPC after they have been passed through the HDMI cards.

The client machine interfaces with the VidCap service using sVision. sVision acts as the client responsible for communicating with the VidCap service. When the sVision connection is initialized, the VidCap Service opens a display window on the client machine. If the window dimensions match the dimensions of the frames given to the CSE HDMI cards the window will provide the client with an accurate display of the image read by the CSE.

5.3.4 The CSE Service

The CSE Service works as the interface between the NUCPC and the CSE. It is responsible for changing firmware settings and checking firmware statuses. The CSE Service contains two custom libraries: the micro controller and the CSE controller. Each instance of the micro controller in the CSE Service acts as an interface to a different CSE. The CSE Service contains an array of the current micro controller objects

that it controls. When a CSE is connected or disconnected, the list of micro controllers is updated. Each micro controller has a separate instance of the CSE controller.

The CSE controller is responsible for handling communication with the CSE firmware. The CSE controller interfaces with the CSE by sending packets to the firmware controller code. The firmware controller is a compiled C project responsible for translating and passing commands to the firmware. The firmware controller also reads values from the firmware and passes them back to the CSE controller. The packets sent to and from the CSE controller are different based on which command is being carried out. Each packet sent has a header containing an operation code. The operation code determines which process the packet is for. The CSE controller, firmware controller, and firmware have matching lists of all the hex operation codes. The operation codes work similar to the WebSocket routing tags.

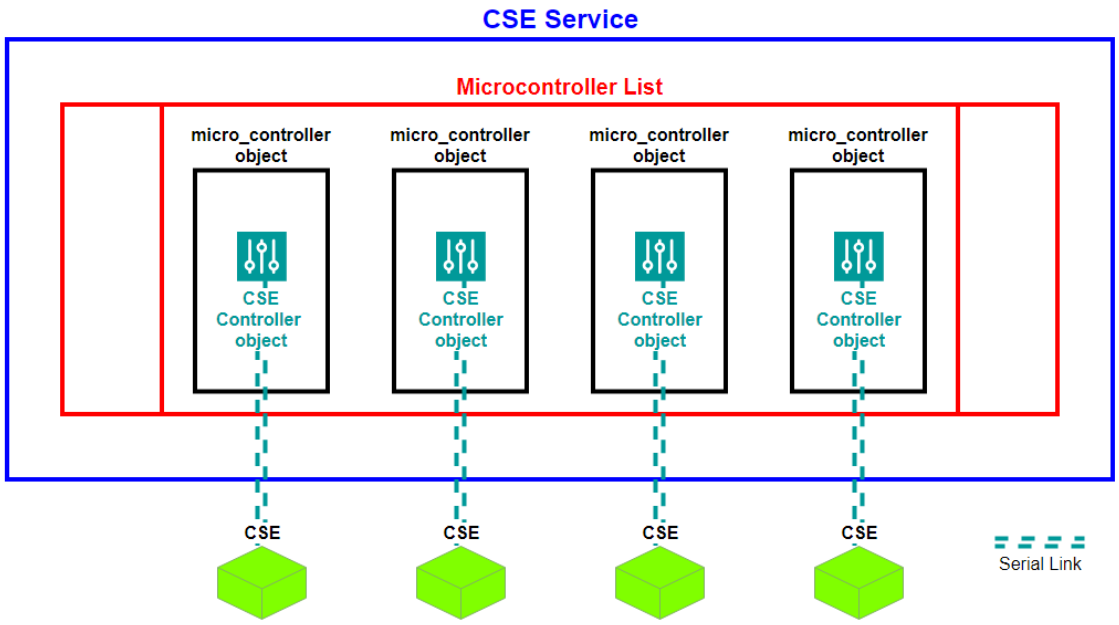

Figure 5.12: CSE Service structure. The CSE Service has a list called microcontroller list which has an instance of the microcontroller class for each connected CSE.

The most important task of the CSE Service is tracking the status of the connected CSEs. CSE tracking is handled asynchronously. The service monitors the trip

status of the CSE, how long the CSE has been on, and how long the system has been untripped without use. The CSE trip status is polled every three seconds. The possible trip statuses are tripped and ready. If the system is tripped, current will not be drawn by the pixels. If the system is ready, the LEDs have been untripped and can draw current. Images are drawn on the projector when it is untripped.

The time the projector has been on is updated whenever the CSE status is checked. If the CSE is still connected, then time is added to the tracker. The tracker also monitors how long the system has been on since a frame was shown on the projector. Whenever a frame is displayed, a variable tracking the most recent display time is updated to the current time. Each time the CSE status is checked, the amount of time since the last frame was drawn is measured. If the CSE has been idle for a preset length of time, a warning message is sent. The warning message is sent to the system administrator and the last person to initialize the projector. When the CSE is turned off, the amount of time the CSE was running is written in a log file for record keeping. Each CSE has a designated log file for run time.

When the CSE is in the ready state, current is drawn to the array whether or not a frame is displayed. Most of this drawn current comes from stuck-ons pixels. These stuck-on pixels draw current at all times if the system is untripped. In order to prevent unnecessary wear on the projectors, the CSE Service has a feature that automatically trips the CSE. The service tracks how long the service has been in the ready state without use. If a certain amount of time has passed (the default time is two minutes, but it can be configured by the user) the CSE Service trips the system.

5.3.5 The Log Service

The Log Service creates, updates, and saves logs of CSE and projector activity. There are three main types of logs: projector usage logs, system overview logs, and test logs.

The projector usage logs track projector usage. On initialization, PING records the date and the start time. While the projector is running PING tracks the cumulative

time the projector is tripped and untripped. When the server is killed or the projector is stopped, the end time is recorded and all the data is saved to a log. Each projector has its own log file containing the trip and untrip time of each instance the projector is turned on. A python script was developed to parse the logs and process the data. Figure 5.13 shows a sample from one of the usage log files.

```
############### begining of log ################
Date: ..................................Mon__18-06-2018
Start_Time: ............................17:52:52
End_time: ..............................18:40:01
Operation Time: ........................00:47:09
Untripped Time: ........................00:22:12
Tripped Time: ..........................00:24:56
################# end of log ###################

############### begining of log ################
Date: ..................................Mon__18-06-2018
Start_Time: ............................18:40:30
End_time: ..............................19:17:31
Operation Time: ........................00:37:00
Untripped Time: ........................00:08:37
Tripped Time: ..........................00:28:22
################# end of log ###################
```

Figure 5.13: Two example logs taken from the NSLEDS 3 projector usage log. The logs contain the date the projector was initialized, the start time, the stop time, the total time the projector was initialized, the total time the projector was untripped, and the total time the projector was tripped.

The system overview logs are also created on initialization. The goal of the system overview logs is to record PING's activity sessions. Each projector has a log directory. A new folder labeled with the date and time is created every time the projector is initialized.

The log folder contains the overview log and test logs. The overview log records the user, the array, the CSE, the firmware settings, the NUCPC hardware and software

NSLEDS3_2019-04-08__15-43-34.txt

```
**********  System Info  **********
Gui Array:                              NSLEDS3
CSE:                                    152196367
Package Type:                           JK
User:                                   Peyman
Gui Version:                            320
Firmware Array:                         N/A
Firmware Id:                            N/A
Firmware Build date(YY/MM/DD/sec):      2018/12/21 14:23:00 EST
Firmware Dimension:                     N/A
Firmware Offset:                        N/A
FMC channel:                            N/A
Number of DAC Channels:                 N/A
Transpose in Firmware:                  N/A
Main Interface board version:           Name:CSE1,ID:0x4D4800,Interface Board version 1.2.4,hardware trip enabled,commit 156
Secondary Interface board version:      ID:4D4801,Interface Board 1.2.4 secondary board,commit 156
VSE10 Trip level:                       2.99999982119 Amps
VSS10 trip level:                       2.99999982119 Amps
OS:                                     64bit Scientific Linux version 7.5 Nitrogen
CPU:                                    Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz 8 cores
Ram:                                    Total=62.7416343689 GB, used=1.07783508301 GB, free=12.0691070557 GB, Available=3.6 GB
GPU:                                    NVIDIA Corporation GK104GL [Quadro K5000] (rev a1)
Disk:                                   Root_Capacity=49.9755859375 GB, Root_used=42.566696167 GB, Home_capacity=174.434318542 GB,
                                        Home_used=159.692173004 GB
*********************************************

################## Test #1 ###################
Date:                                   2019-04-08
Start_Time:                             15:43:49
End_time:                               15:43:56
Duration:                               7.84805393219
Source Tab:                             Animation Tab
Animation:                              dp_mapper
Type:                                   animation_distortion_point_mapper
Settings:                               {u'dpm_ypos_text': u'0', u'dpm_xpos_text': u'0', u'dpm_pixel_color_text': u'65520', u'dpm_gr
Test Result:                            Test was successful
############### end of Test #1 ###############
```

Figure 5.14: An example overview log for NSLEDS 3

information, etc. It also contains a brief record of each test run on the projector. The test logs contain more detailed information about the tests.

5.3.6 The Temperature Reader Service

The Temperature Reader Service is responsible for controlling the temperature reader connected to the projector. The connection to the temperature reader is maintained over a serial port. The Temperature Reader Service tracks the temperature asynchronously. The service periodically polls the current temperature and assigns the updated value to a global variable. That value can be pulled from the service whenever it is needed. In PING, the temperature is mainly used for data collection and analysis. The tracked temperature can be used to categorize whether the system was run at room temperature or whether it was cooled to cryogenic temperature. If no

```
################## Test #1 ###################
Date:                                   2019-04-08
Start_Time:                             15:43:49
End_time:                               15:43:56
Duration:                               7.84805393219
Source Tab:                             Animation Tab
Animation:                              dp_mapper
Settings:                               {u'dpm_ypos_text': u'0', u'dpm_xpos_text': u'0',
                                        u'dpm_pixel_color_text': u'65520', u'dpm_grid_color_text': u'65520',
                                        u'dpm_row_space_text': u'64', u'dpm_grid_yspace_text': u'32',
                                        u'dpm_grid_xend_text': u'1024', u'dpm_grid_xpos_text': u'0',
                                        u'radio': u'dp_mapper', u'routing': u'showAnimation',
                                        u'dpm_grid_xspace_text': u'32', u'dpm_off_frames_text': u'5',
                                        u'dpm_grid_ypos_text': u'0', u'dpm_col_space_text': u'64',
                                        u'dpm_grid_yend_text': u'1024', u'browserText': u'',
                                        u'dpm_threshold_text': u'600'}
Temperature on Start:                   306.95
Temperature on End:                     306.99
APL Used:                               False
NUC Used:                               False
CSE status upon completion:             ready
Min Input DAC count:                    0
Max Input DAC count:                    65520
Camera Integration time:                0.0006 s
Camera Frequency:                       100.0 Hz
camera_info:                            1024x928 0x48 window (4-tap/14-bit, synced)
Test Result:                            Test was successful
############## end of Test #1 ###############
```

Figure 5.15: An example test log. This test file contains the detailed information from test 1 in the overview log.

temperature sensor is connected, the system cannot initialize.

5.4 Shared Python Libraries

The backend also has a group of custom Python libraries needed by both the interface controller and the services. Since the interface controller and services are located in different places on the NUCPC, these libraries could not be accessed by both sections if they were only placed with one or the other. In order to minimize duplicate files, these shared libraries are added to the global NUCPC path.

5.4.1 OpenGL

OpenGL is an API used to render 2D and 3D graphics. In PING, it is used to generate the frame textures used on the projector. PING creates these textures using

a library that interfaces with GLFW. GLFW is an open-source cython library used to run and control OpenGL.

5.4.2 Error Handling

As with any software architecture, PING should ideally never encounter unexpected errors. In some cases, errors can represent expected failure states. In these cases, explicit exception catching is used to handle the expected error state. Figure 5.16 demonstrates the flow of exception handling in PING's APIs. An example of this behavior can be seen when the web interface attempts to connect to the services. When ping initializes, it creates instances of the service client classes. If a socket error is detected, the service has not been started on the NUCPC. Rather than crash the web interface, the caught exception is noted. The web interface records which service failed to connect and notifies the client. If an unexpected exception is detected (any exception besides a socket error), this indicates something is wrong and the server should not finish starting up.

Error handling is also managed using a shared library. The PING error library contains custom exception classes. These custom exceptions were developed to provide error descriptions directly related to aspects of PING. The "CommunicationError" exception is an example from the PING error library that handles CSE timeouts. If a CSE connection times out, this exception is raised. The exception has different outputs based on what part of the hardware connection failed (either the Super FPGA, main interface board, or secondary interface board). The exception is passed through the CSE Service and across the ZeroRPC link to the in the web interface. From there, the web interface notifies the client that a CSE connection has timed out.

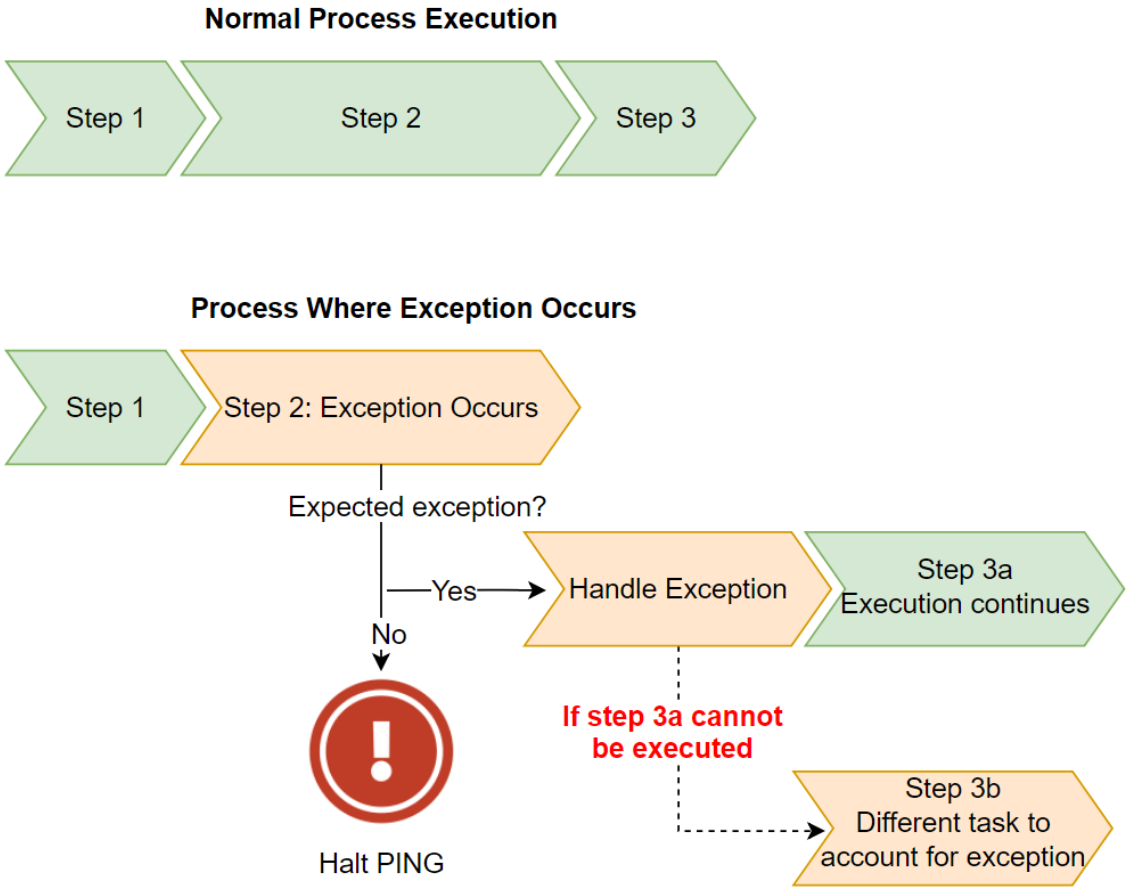

Figure 5.16: Process execution for APIs in PING. The normal process execution (top) demonstrates normal functionality: If no issues are encountered, the steps execute as intended. If an exception occurs (bottom), PING determines whether the exception is expected or not. If the error is an expected exception it is handled. Then the process either continues as normal or executes differently based on the exception that occurred. If the exception is not expected, PING is halted.

Chapter 6

FEATURE IMPLEMENTATION

The following sections detail different processes PING is responsible for. These processes are integrated into the backend and are vital for projector operation and analysis.

6.1 Scene Generation

As mentioned through the previous descriptions of PING's architecture, the backend is responsible for SLEDS scene generation. The ability to create and display scenes on the fly from a variety of different sources makes PING an incredibly versatile tool for SLEDS control. PING uses NumPy to store images in 2-dimensional arrays matching the projector resolution. Once the adequate manipulations have been made to the NumPy frames (for example, adding objects in the draw objects tab or rotating/flipping an image/video), the frames are bit-packed and converted into HDMI textures using GLFW. These textures are then sent over the HDMI connection between the NUCPC and the CSE where they are rendered on the projector.

6.2 Bit-Packing

One of the most important processes PING is responsible for is bit-packing the images. Bit-packing is used to condense the images that are sent to the projector. Once the scenes have been generated into NumPy arrays, the values of horizontally neighboring pixels are merged into a single value to reduce the width of the image by half. The bit-packed frames allow data to be transferred faster to help increase the maximum framerate. The firmware expects these bit-packed images and is responsible

for unpacking them when they arrive. Without the bit-packing process, the images displayed on the projector will not be accurate.

6.3 Windowing

Windowing is a feature developed to potentially increase the possible framerate of the SLEDS system. When enabled, scenes are written to a subsection of the projector. Since less pixels are written to, more frames can be displayed in the same amount of time. HDILEDs, the 2048x2048 resolution projector, currently has a maximum stable framerate of 60 Hz. Using a 1024x1024 window on the projector, the framerate can be sped up to approximately 200 HZ. An upgraded firmware capable of variable-framerate subwindowing is currently in development. This firmware will use windowing to display subsections of the projector at different framerates. The foreground of the scene (a moving object) will run at higher framerates, while the background of the scene (non-important areas) will run at lower framerates.

6.4 Average Pixel Linearization

Due to the nature of the projector manufacturing process, not all of the pixels follow a uniform brightness pattern. Ideally, as the voltage drawn by a pixel linearly increases, the brightness should linearly increase. Average pixel linearization (APL) was developed as a method for improving the uniformity of the arrays. In this process, the most uniform part of the array is observed and an algorithm is developed to optimize lower brightness values for the non-uniform pixels. A unique algorithm is developed for each projector, since no two projectors have the exact same non-uniformity [4]. In figure 6.1, an image displayed on a projector without APL applied is placed next to an image displayed with APL applied. Comparing the two, it is clear that the lower-emittance areas of the APL image are more prominent. The edges of the bright area are also sharper on the right image.

In PING, the APL is applied to the NumPy frames before they are converted to textures. The user has the option to apply APL to any type of generated scene.

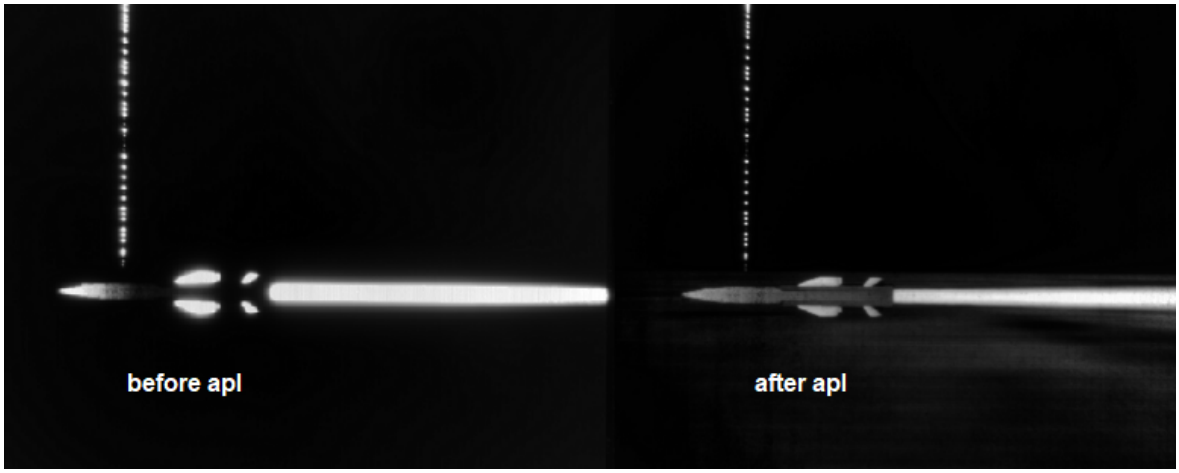

Figure 6.1: Two versions of the same image displayed on a projector captured by the FLIR camera. For the left iamge, APL was not applied. For the right image, APL was applied.

While image quality is improved, scenes using APL take longer to generate since it takes time for the algorithm to be applied to the whole frame. This is not an issue for scenes that are processed entirely before being displayed. However, for the animation tab, which generates and displays scenes frame by frame, the added processing time can cause the scene display to drop below the desired framerate.

6.5 Data Capture, Data Storage and the Network Drive

Another feature of PING mentioned in previous sections is its ability to capture and store data. PING is responsible for capturing four main categories of data: log files of projector use, temperature data, preprocessed frames, and captured camera data. As discussed in section 5.3.5, PING records three type of projector logs: operation time logs, system overview logs, and test logs. These logs are used to analyze system longevity and how the projector reacts to different tests. Temperature logs are used to measure the projector system's internal temperature through operation. The temperature reader service has an option to track the periodically polled temperature and save the time-temperature data values to a log file.

During the scene generation process, PING has the option to save the frames that have been processed and are ready to be displayed on the projector. This option

is available for movies and custom tests. These preprocessed frames are saved to a file which can be loaded in the videos tab. Doing so allows the user to replay the same scene at any later time without having to regenerate the frames. For longer movies or tests, this can save significant time.

The final data type captured by PING, captured camera data, can be acquired whenever the FLIR camera is connected. The frames captured by the camera can be used to track brightness of areas based off of the saved frames. This can be useful for mapping brightness curves for individual pixels or large objects.

The data captured by PING is stored in a specific directory called the "ping data" directory. This data directory has subdirectories based on what kind of data was captured or how it was capture. In order to prevent disjointed data collection, the ping data directory is stored on a network drive. This network drive is accessed by all the NUCPCs on the same network, so they all share the same ping data directory. If a NUCPC is not connected to the network drive, it will create a local version to store collected data until it can be reconnected. The network drive also holds files for scene generation and display. It contains a collection of media that the image and movie tab file navigators open to by default.

6.6 Interface Reliability

One of the major challenges PING faced in early development was instability. During the early stages of the software's life, feature growth outpaced reliability testing. Testing of new features was done to confirm basic reliability, but bugs were still prevalent. The solution to this instability problem was live testing. Software bugs were patched as they were discovered, allowing deep set problems to be traced and removed. After PING's capabilities reached a satisfactory level, feature growth slowed down and the development focus shifted to maximizing the software's stability. With the help of proper error handling implementation and heavier feature testing, PING has reached an optimal reliability.

Chapter 7

EXAMPLE USE CASES

The following chapter will outline practical applications of the PING software used to test different aspects of the projector.

7.1 Radiance Degradation Test

This test compared array packaging methods to determine which was superior. Different NSLEDS projectors used different methods of attaching the projector to the cold finger during fabrication. The cold finger is used to remove heat. It helps maintain low internal temperature for "off" pixels, which helps maximize brightness. If the projector's internal temperature is not kept low, the maximum radiance will be limited to prevent pixels from overheating. To observe the effectiveness of different packages, a test was developed in PING to track the radiance of an individual pixel based on how far away a large bright object was drawn. The observed pixel was drawn in the top left corner of the projector. A square of pixels with dimensions of 20x20 was drawn in the bottom right corner. This square was then moved closer and closer to the observed pixel, and the radiance of the pixel relative to the square was recorded. Figure 7.1 shows a frame captured during the test.

The two projectors tested were NSLEDS1 and NSELDS3. NSLEDS1 is packaged using only an epoxy. NSLEDS3 is packaged using an epoxy and indium bumps. The single pixel's maximum radiance percentage was measured based on how close the square was to the pixel. The maximum radiance percentage is a ratio of the measured output light with the square to the maximum output light of the pixel with no other objects drawn. The test was generated and displayed by PING and the data was also captured and saved by PING. Figure 7.2 shows the graph of radiance percentage

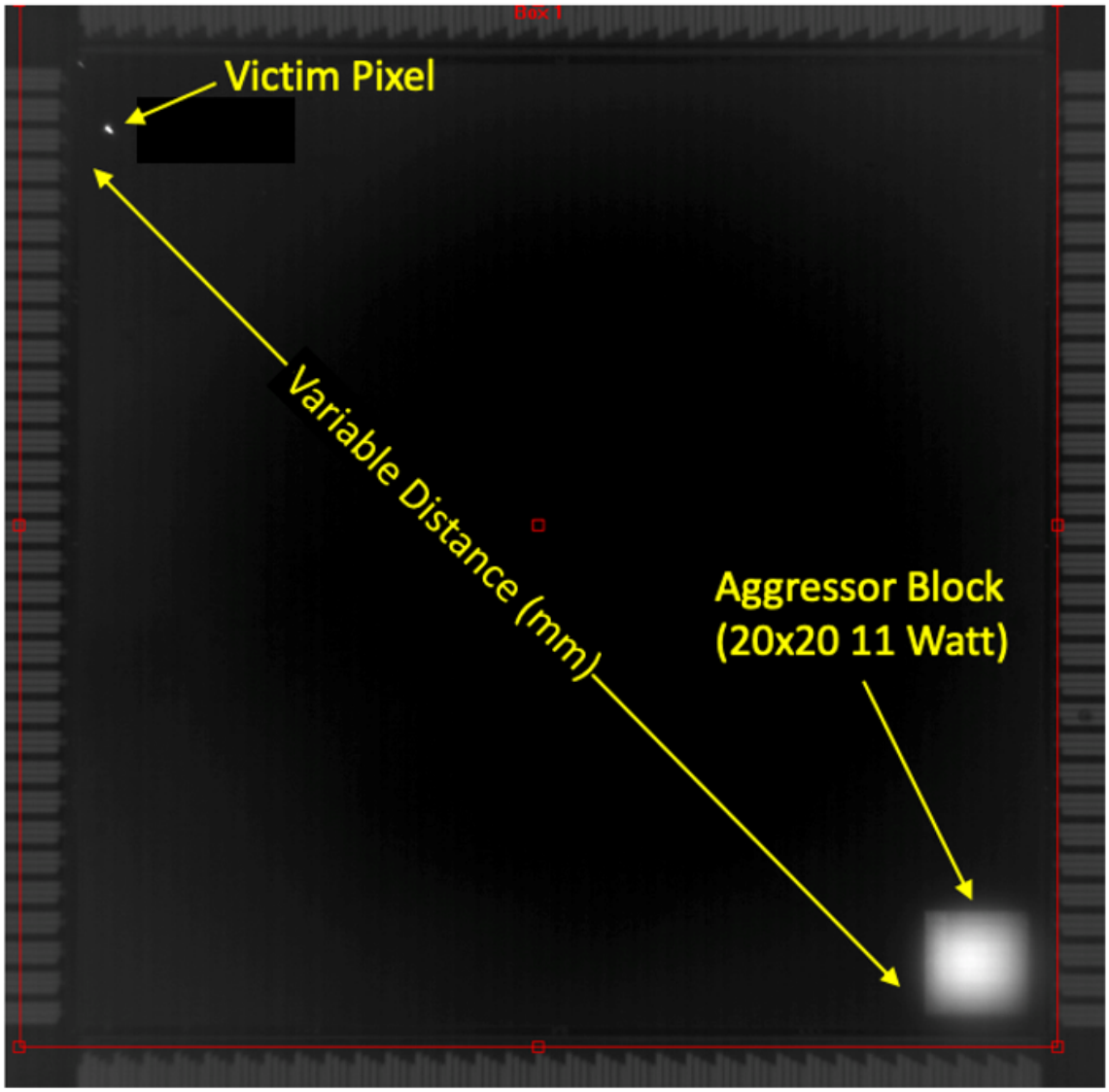

Figure 7.1: Radiance Degradation Test Frame.

based on the measured pixel's distance from the square. Based on the plot, it can be determined that NSLEDS3 has a higher maximum radiance with the presence of the square (95% vs 91%). NSLEDS3 also has significantly higher worst-case degradation. When the square is drawn 1 millimeter away from the pixel, NSLEDS3 maintains approximately 88% maximum radiance, while NSLEDS1 maintains 55% maximum radiance. Therefore, it can be determined that the indium bumps-epoxy combination

maintained much better temperature control than the epoxy by itself.

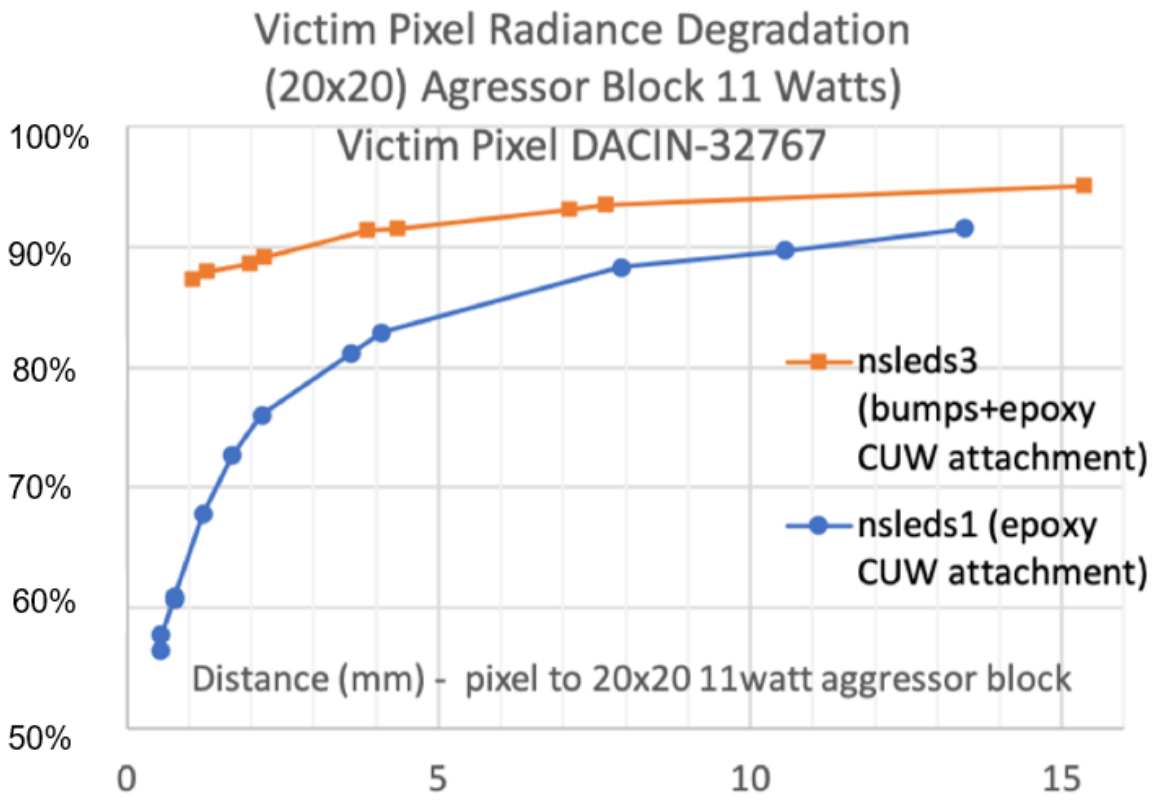

Figure 7.2: Maximum Radiance Comparison Plot of NSLEDS3 vs NSLEDS1 using the radiance degradation test created by PING.

7.2 Dewar Temperature Testing

Another example use-case for PING is demonstrated by the data collected and displayed in figure 7.3. The temperature tracker in the Temperature Reader Service two test two different setups: using a normal Dewar with liquid nitrogen and using a gravity-fed Dewar to hold the liquid nitrogen. Liquid nitrogen is used to cool the projector system to cryo temperature (77 degress Kelvin). The purpose of the test was to compare how long it took both methods to return to room temperature.

As seen from the data, the gravity-fed Dewar held cryogenic (cryo) temperature much longer. The gravity-fed Dewar stayed below 80 degrees Kelvin for 34 hours and 54 minutes. The Dewar without the gravity feeder stayed below 80 degrees Kelvin for

only 5 hours and 49 minutes, meaning the gravity-fed Dewar stayed cooler for over five times longer. The gravity-fed Dewar also returned to room temperature faster after running out of liquid nitrogen (5 hours and 54 minutes vs 7 hours 52 minutes). The only advantage of the Dewar without the gravity feeder is the cooling time to 77 degrees Kelvin. While the gravity-fed Dewar took longer to cool to cryo (12 minutes) than the normal Dewar (7 minutes), the time difference is inconsequential.

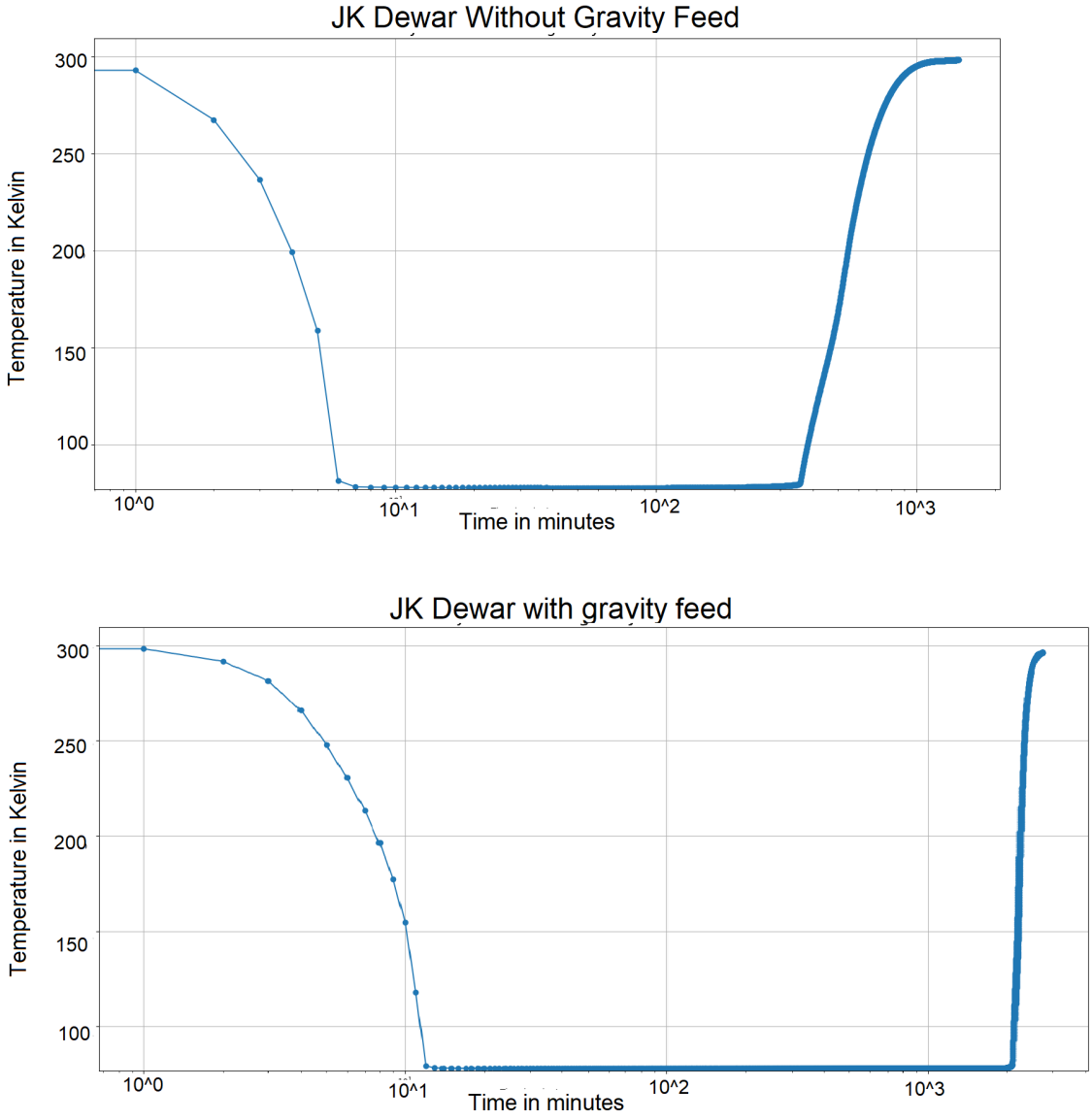

Figure 7.3: Testing how the Dewar holds cryo temperature without and with the gravity feeder. The top graph shows the temperature tracking without the gravity feeder. The bottom graph shows the temperature tracking with the gravity feeder.

Chapter 8

FUTURE WORK

The following sections will describe changes and feature that are planned to be added in the future.

8.1 Python3 Rewrite

One of the most pressing future changes to PING is migrating the code from Python2 to Python3. When PING was first created, Python2 was the more widely used version. In 2020, support for Python2 will end [17]; therefore, PING will need to be rewritten using Python3. The transition to Python3 will start with the services. Since the services run independently of the web interface, they will have the least potential to cause major problems early in the transition's life. Once the services are rewritten, the web interface and its other libraries will be transitioned.

8.2 Web Interface as a Service

Eventually, the backend controller will be converted to a service. Currently, the web interface is run from the repository directory on the NUCPC. By converting the web interface to a service, the user will be able to run the interface at any time on the NUCPC. This will also allow the interface to run at all times, making client connections simpler. Furthermore, if the web interface is running at all times, configuration settings can be saved and more potential logging opportunities will become possible.

8.3 Independent Script Running

While PING has a vast array of features, some new desirable features may have limited scope and functionality. These niche features would be inefficient to implement,

as the added bloat to the front and backend interfaces would outweigh their usefulness. One possible solution to address such features is to implement a way to run independent scripts through PING. The user would have the option to import a prewritten Python script and run it using PING's architecture. This would allow the user the flexibility to handle small tasks without having to overcomplicate PING. The only downside to the system would be a lack of monitoring and control of the imported code. This issue will have to be addressed before the capability to run independent scripts is added.

Chapter 9

CONCLUSION

PING was originally created to provide a comprehensive and scalable interface for controlling the SLEDS IRSPs. Through its design process, PING has evolved to meet the needs of the end users. In the current phase of the SLEDS research and development process, PING is used for system testing and data collection. All current generation NUCPCs have incorporated the service-oriented architecture and are updated periodically with the most recent changes to the controller interface. All projector operation is handled by PING. Tweaks are consistently made to add new features and fix minor software bugs. Overall, PING has met and exceeded its goals to become a reliable user interface for control of the SLEDS systems.

BIBLIOGRAPHY

[1] Bokeh user guide. https://bokeh.pydata.org/en/latest/docs/user_guide.html. Accessed: 2019-04-18.

[2] H. Ahmed. Designing a modular and scalable firmware for infrared led scene projectors. Master's thesis, University of Delaware, 2016.

[3] P. Barakhshan. Thermal performance characterization of a 512x512 mid-wave infrared super lattice light emitting diode projector. Master's thesis, University of Delaware, 2016.

[4] P. Barakhshan, G. Ejzak, M. Hernandez, A. Landwehr, N. Waite, K. Nabha, F. Kiamilev, R. J. Ricker, S. Provence, J. P. Prineas, and T. F. Boggess. Gamma correction and operability of irled scene projectors. GOMACTech, Mar 2018.

[5] P. Barakhshan, J. Volz, R. J. Ricker, M. Hernández, A. M. Landwehr, S. R. Provence, K. Nabha, R. Houser, J. P. Prineas, C. Campbell, F. E. Kiamilev, and T. Boggess. End to end testing of irled projectors. *2018 IEEE Research and Applications of Photonics In Defense Conference (RAPID)*, pages 1–4, 2018.

[6] P. T. Bryant, J. Oleson, B. Lindberg, B. Anderson, K. Sparkman, S. W. McHugh, J. Lannon, D. Vellenga, S. Goodwin, and S. L. Solomon. Mirage: developments in irsp system development, riic design, emitter fabrication, and performance. 5092, 2003.

[7] T. S. Community. Numpy reference documentation. https://docs.scipy.org/doc/numpy/reference/. Accessed: 2019-04-01.

[8] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455, RFC Editor, December 2011.

[9] M. Hellkamp. Bottle api reference. https://bottlepy.org/docs/dev/api.html. Accessed: 2019-04-01.

[10] J. James, J. LaVeigne, J. Oleson, G. Matis, J. Lannon, S. Goodwin, A. Huffman, S. Solomon, and P. Bryant. Oasis: cryogenically optimized resistive arrays and irsp subsystems for space-background ir simulation. 6544, 2007.

[11] jQuery Foundation. jquery api documentation. https://api.jquery.com/. Accessed: 2019-04-01.

[12] Z. Marks. Designing an advanced packaging system for infrared scene projectors. Master's thesis, University of Delaware, 2017.

[13] R. McGee, F. Kiamilev, N. Waite, J. Marks, K. Nabha, G. Ejzak, J. Dickason, J. Benedict, and M. Hernandez. 512x512, two-color infrared led scene projector. GOMACTech, Mar 2014.

[14] K. Nabha. 100 hz 512x512 sleds system design. Master's thesis, University of Delaware, 2014.

[15] J. D. Neal. Hardware level vga and svga video programming information page. http://www.osdever.net/FreeVGA/vga/crtcreg.htm, 1998. Accessed: 2019-04-07.

[16] D. T. Norton, J. T. Olesberg, R. T. McGee, N. A. Waite, J. Dickason, K. W. Goossen, J. Lawler, G. Sullivan, A. Ikhlassi, F. Kiamilev, E. J. Koerperick, L. M. Murray, J. P. Prineas, and T. F. Boggess. 512×512 individually addressable MWIR LED arrays based on type-II InAs/GaSb superlattices. *IEEE Journal of Quantum Electronics*, 49(9):753–759, Sep 2013.

[17] B. Peterson. Python 2.7 release schedule. https://legacy.python.org/dev/peps/pep-0373/. Accessed: 2019-04-10.

[18] R. McGee, F. Kiamilev, N. Waite, J. Marks, K. Nabha, G. Ejzak, J. Dickason, J. Benedict, and M. Hernandez. 512x512, 100hz mid-wave infrared led scene projector. GOMACTech, Mar 2015.

[19] K. Sparkman, J. Laveigne, S. McHugh, J. Kulick, J. Lannon, and S. Goodwin. Scalable emitter array development for infrared scene projector systems. page 90711I, 05 2014.

[20] P. T. Bryant, J. Oleson, B. Lindberg, K. Sparkman, S. W. McHugh, and S. L. Solomon. 1024 x 1024 large-format resistive array (lfra) design, fabrication, and system development status. *Proceedings of SPIE - The International Society for Optical Engineering*, 09 2003.

[21] E. D. Team. Visionlink f4. https://edt.com/product/visionlink-f4/. Accessed: 2019-04-10.

[22] J. Volz. Designing analysis and tools to measure the longevity of infrared led scene projectors. Master's thesis, University of Delaware, 2018.