CAVECamX Final Report

by Linda Shih and Helen Toma

Abstract - The CAVECamX project aims to develop a device for generating immersive 3D panoramic images that is a significant improvement over the previous generation CAVECam by providing more flexible algorithms and more intuitive interfaces. This will provide scientists and researchers an accurate and immersive way to document environments. The CAVECamX uses two Sony QX1 cameras mounted on top of a high-precision gimbal to stitch a series of images to produce a high resolution 3D panoramic image. The sensors used in this device will improve the quality of the 3D content by improving the resolution and depth perception compared to the current generation of CAVECams. Currently we have produced an algorithm that dynamically generates orientations for any given set of camera parameters, a web interface for remote connection and control of one Sony QX1 camera, and software modules to gather and aggregate the data coming from the CAVECamX. The goal for the CAVECamX is to increase access to 3D visualization technology by providing a simple and easy-to-understand interface for generating 3D panoramic images.

Introduction

The camera control portion of the CAVECamX project aims to create a simple and seamless way to interact and get feedback from the cameras. Since the Sony QX1 cameras used by the CAVECamX do not have an LCD, this makes it incredibly inconvenient to view images on the SD card and change camera settings such as ISO, shutter speed, and aperture. Additionally, the CAVECamX uses two Sony QX1 cameras, which means that any configuration or action performed on one camera must be mirrored on the other as well. This leads to slow and clumsy setup and usage of the CAVECamX.

To combat this problem and make interaction with the cameras simpler for users such as archeologists and researchers, we developed a web-based interface that would allow the user to be able to make settings adjustments or get feedback from the camera such as notifications or viewing images from the SD card, all without having to physically touch the cameras. Currently, we have implemented several important features such as:

- Focusing via Live View
- Image Notifications
- View/Download Images

Focusing using the live view allows users to set a new focus point on the camera without needing to physically adjust the camera lens or input values into a command prompt. The user simply clicks on the object that they wish to be in focus, and those points will be interpreted and sent to the camera. Again, the Sony QX1 cameras do not have LCDs, so this feature paired with the live view creates a very intuitive and user-friendly way to frame a picture and set focus. This lets them to make subsequent settings configurations as they wish, which will all be reflected on the live view in the web interface.

Camera notifications provides feedback to the user about what is happening on the camera side. There are two ways of taking a picture: using the shutter button on the camera body, or through the web interface. By implementing a feature that will be able to tell the user information about a picture, future work can use this data to tell the gimbal when it can move to the next location. Data such as:

- When the image has completed capturing
- Where the shutter call was made from
- The resulting image URL

provides the user a way to know when it's okay to start taking the next picture, if the execution was made from the camera body or the interface, and gives them the option to review the latest picture to see if they want to retake it.

Lastly, the option to view or download images off of the SD card provides users an easy way to remotely review pictures currently on the SD card without the need to remove it from the camera. When the "View/Download Images" tab is clicked, it page begins to load low-res images in the form of small thumbnails and corresponding full-resolution images from the camera. When finished loading, clicking on a thumbnail will open a full-resolution image in a new browser tab.

With these three basic features added to the web interface, we believe that using the CAVECamX will be much simpler to use and intuitive to a layperson. Being able to get feedback from the camera knowing when a picture has finished capturing will be able to help the gimbal know when it's safe to move to a next location. This opens door for more work in the future on the CAVECamX.

Technical Material

Server - The server side of the web-based interface is written in Python and acts as a bridge between the client web page and the camera. The server is responsible for establishing a Wi-Fi connection with the camera, listening to client requests, and sending those requests to the camera if they are made correctly. Any requests that are made to the camera are then translated to a corresponding Sony API call by the server, which is then sent to the camera. These API calls must follow a protocol specified by the Sony Camera Remote API v2.00 document. The differentiation of calls can differ by name, parameters, version, and camera mode. The Sony API calls are formatted as:

```
name_of_function(camera_mode, parameters, version)
```

Focusing via Live View: In order to get the camera to focus on a specific point, the Sony API document provides an API call named `setTouchAFPosition`, which will focus on a given point given the location of the point in percentages. When a user clicks on a point on the live view, those points will be converted to percentages, passed to the server, which are then passed to the camera.

Python Server — Sony API v2.00

```
set_focus(13.3, 78.3) → setTouchAFPosition(camera, [13.3, 78.3], 1.0)
```

Image Notifications: It is important to know how and when the camera is being used and when it is done taking pictures, so this feature helps the user to get notifications whenever a picture is taken and if it was taken via shutter button on the camera body or from the web interface. The Sony API provides "awaitTakePicture" API which was used to help user to get the notifications on the web interface. This API gives a feedback to make the user wait until the camera is done, and is able to use this feedback to get this feature working. The API returns a URL of the picture taken. The notifications update by comparing the new generated URL with the previous inside the Python Server.

Python Server — Sony API v1.00

```
await_take_pic() → sony_api_call("awaitTakePicture", [])[0][0]
```

View/Download Images: To be able to view images on the camera SD card, a series of calls must be made. Per the Sony API document, the camera must be in "Contents Transfer" mode in order to access any part of the SD card. After switching to this mode, the server must:

1. Check if an SD card is present
2. If so, change to "Contents Transfer" mode
3. Check the number of images on the SD card
4. Make requests for image information
5. Repeat Step 4 if necessary

6. Collect relevant image information, package, then send

The Sony API call for SD card contents can only deliver up to at most 100 images from the SD card at a time. If the SD card happens to have more than 100 images, then the API must be called several times (in some loop) in order to get all the images.

Python Server	Sony API v2.00
1. `get_storage_information()` →	`getStorageInformation(camera, None, 1.0)` Returns: - `"No Media"` → done - `"storage:memoryCard1"` → change camera mode
2. `set_camera_function("Contents Transfer")` →	`setCameraFunction(camera, "Contents Transfer", 1.0)`
3. `get_content_count()` →	`getContentCount(content, {"storage:memoryCard1", "flat"}, 1.2)` Returns: - `0` → done - `1+` → request image information
4. `get_content_list()` →	`getContentList(content, [params], 1.3)` Returns: - `[image1, image2, image3…]`

5. Repeat Step 4 if there are more than 100 images, since the API call can only return a maximum of 100 images at once.
6. Take relevant image information such as small image URL and the full-res image URL and send them to the web client to be displayed as a grid of thumbnails.

Python Server	Web Client
`[small_img1, small_img2, …]` →	display small_img1, display small_image2, …
`[large_img1, large_img1, …]` →	open full-res when corresponding image is clicked

Client - The client is the web interface, or the end product that the user will be able to see and use, which will send requests to the server according to the actions of the user. The web address currently being used is localhost:5000 in a web browser.

This portion uses HTML, CSS, JavaScript, and JQuery for its implementation. HTML and CSS are used for displaying and formatting web page content, and JavaScript and JQuery are used for the functionality such as switching tabs and sending/getting information to and from the server.

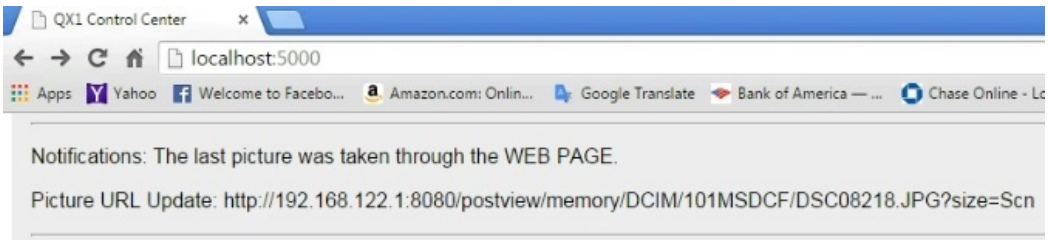
When a user clicks or inputs data into the web page, the JavaScript will trigger actions to occur. Requests may be sent to the Python server to obtain data, or requests may be sent to the server to change modes or settings.

Focusing via Live View: When the user clicks on a specific point on the live view, JavaScript is able to return the specific coordinates of the mouse click. However, since the Sony API call for focusing must take in *percentages* as arguments, additional calculations must be made before making the API call:

```
x_coordinate/liveview_width*100 = x_percentage
y_coordinate/liveview_height*100 = y_percentage
```

These percentages are then passed to the server to make the request to focus on those points.

Image Notifications: When the user clicks on “Take Picture” button or the shutter button on the camera body, the camera’s API generates a URL that is returned to the Python Server. If the picture was taken through the web page, then “actTakePicture” API gets called and returns the new URL, and if the picture was taken manually, the “awaitTakePicture” API gets called and it returns a new URL. By comparing the stored URLs returned from each call, we are able to distinguish the camera usage and know if the camera is done taking pictures, then the notifications get updated in the Python Server and sent to JavaScript to enable HTML to update the web interface with the correct URL and notifications.



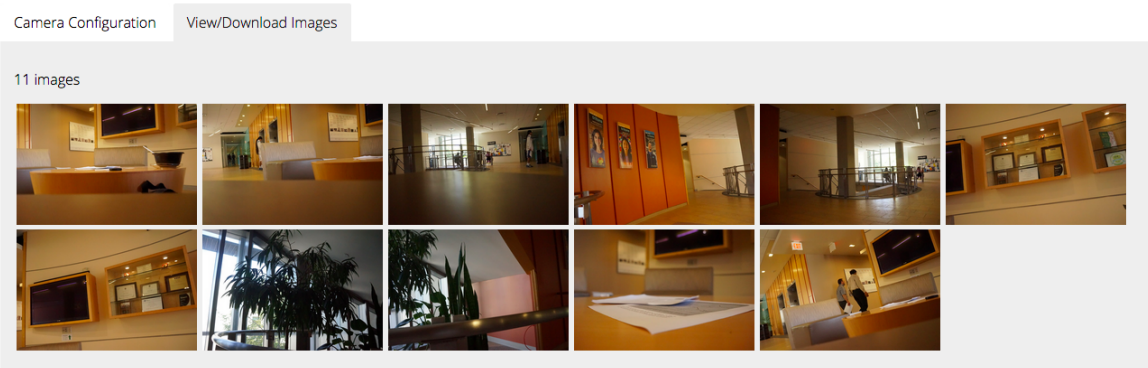
View/Download Images:

Tabbing: A new tab for viewing and downloading images was added to the web interface using HTML and CSS for formatting and style, and JavaScript for actual functionality of clicking on the tabs. In order to improve performance of the server and not overload the QX1 camera, switching tabs would halt any activity in the other tab. The default "Camera Configuration" tab queries the camera for ISO, shutter speed, and aperture values every 3 seconds. Clicking the "View/Download Images" tab breaks that loop, and resumes the loop when switching back to the "Camera Configuration" tab. When on the "View/Download Images" tab, another loop is triggered to start querying the camera for SD card contents. Similarly, this loop is stopped when in the configurations tab.

Camera Configuration Tab	→	View/Download Images Tab	→	Camera Configuration Tab
- settings loop ON		- settings loop OFF		- settings loop ON
- contents loop OFF		- contents loop ON		- contents loop OFF

Displaying Thumbnails: Thumbnails of the original images are displayed in a grid-like format on the "View/Download Images" tab. CSS is used to scale down the size of the images and add a small margin to separate each thumbnail. JavaScript is used to convert each element of the small URL array returned from the server, into a displayable image on the web page. Additionally, it wraps the thumbnail with a link to the corresponding element in the large URL array. This allows the thumbnail to open a new tab displaying the full-resolution image when clicked.

```
<a href=highresX> <img src=lowresX/> </a>
```



Milestones – Initially Linda and Helen decided to divide up the work by frontend and backend work. Linda would handle Python code on the server side, whereas Helen would be in charge of frontend HTML, CSS, and JavaScript work. However, due to the different schedules and experience, they decided to split up the work by milestones/features instead.

1. **Focusing via Live View** (4/24) Done:

   This milestone was to be completed by Linda on 4/24, but was completed 4/21. Due to this early finish, Linda started on milestone 3, while Helen was assigned milestone 2 of the project. The focusing via live view was completed by grabbing the coordinates of each mouse click on the live view, converting those values to a percentage, then calling the appropriate Sony API for it.

   Some problems that were encountered during this milestone were that the Sony API call is value on the format on the variable types for the functions that it provides. For a few days I could not understand why it wasn't working, and later realized that it could only accept type double, instead of string or integer values.

Additionally, the focusing functionality broke a few weeks later. Clicking on a point on the live view triggered no response from the camera, although the server would state that the API call to the camera was successfully made. Linda later found that the camera focus mode was somehow changed to “MF” which is incompatible with her current implementation. She changed the camera focus mode to “AF” and seemed to fix the problem.

2. **Image Notifications** (5/5) Done:

    Helen started working on this Phase during week 4. Many changes were requests after she started working on it which made the delivery delayed for this feature. After few changes to the due date, it was finally settled to have the due date moved to 6/5. This feature was done on 6/1, but she kept working on making the picture URL to display the picture until 6/6, but this feature could not be accomplished because the URL is not set and it keeps updating. Helen tried to use different HTML functions and multiple JavaScript syntax but to no avail.

    By the end of week 8, this feature was accomplished first by displaying only the URL and keep track of the image name that is appears as part of the URL. This allowed us to test the code and be able to get feedback from the camera when it is used manually to take pictures. It also helped us to distinguish the different calls that are made to take a picture.

    By the beginning of week 10, the notifications messages where up. This was tested first by sending notifications to the server then eventually moved them to HTML. Those notifications were tested and they were updating as the camera takes pictures to inform the user about when the camera is done with the panorama and if the picture was taken manually. For the rest of week 10, I kept working on the clickable URL to make it work but unfortunately could not get the URL to display the picture when it is clicked.

    After using “awaitTakePicture” to get the web interface to display and update the picture URL and notifications, a bug was introduced. The camera started taking two pictures after every other click on “Take Picture” button. After further investigations, Helen discovered the “awaitTakePicture” API was what causing this bug although the Sony API document states that the API should only inform the user about the camera usage and should not take a picture. Helen was able to fix this bug by calling the API differently.

3. **View/Download Images** (5/15) Done:

    Linda started working on this milestone after finishing the first milestone. She began by adding tabbing functionality and some minor user interface improvements to the web page. This milestone was completed on 5/26 instead of the original 5/15 deadline. The original 5/15 deadline was set so that the code could be deployed and used in Guatemala, but due to some missed deadlines along with the gimbal team as well, the code ended up not being used. This allowed us to push the deadline to the end of the quarter.

    After studying the Sony API document, Linda found that grabbing image information off the SD card involved a sophisticated series of calls to and from the camera. It involved first switching to the “Contents Transfer” mode of the camera, checking if the SD card was present, and then checking how many images were on the SD card. If there was more than 1 image, then specific image information would be requested from the camera so that it could be displayed on the web page. Functionality had to be added for when the user switches back to the other tab, in which case the camera mode must switch back to “Remote Shooting”.

    Some problems that Linda faced were that the Sony API was not entirely clear on how the sequence of calls would occur. Sometimes there would be a 1-2 second lag when the camera was switching modes. A 3 second delay was added in order to account for this camera lag. On top of this, the API document wasn’t clear that when querying for SD card content such as number of images or specific image information, a completely separate new URL had to be used. This problem took about 2 weeks to solve and was mainly attributed to the vague document and

unclear specifications that it provided. Additionally, confusing statements such as " `The client should check the parameter "cameraFunctionResult" of "getEvent (v1.0)" to get result of setting camera function. The callback of this API is for starting execution, not for getting result of setting.`" This note was extremely misleading and ended up being an optional API call that did not need to be executed.

A persisting problem that occurred was that changing modes from "Remote Shooting" to "Contents Transfer" resulted in a frozen live view. While this is expected, any attempts to restart or reinitialize the live view on the "Configurations" tab was a failure. Linda attempted to recreate a LiveView object, restart the live view thread, as well as recreate the live view image. None of these things worked. While there currently no fix, a workaround is that restarting the server gets the live view unfrozen and working again.

Things We Didn't Have Time For:

While several project goals were accomplished, they were not completed to perfection. Problems which arose during the course of the quarter still persist, which we have not been able to get around to fixing or implementing a workaround:

1. Frozen Live View: When switching from "View/Download Images" tab to the "Camera Configurations" tab, the live view freezes. This issue is due to the fact that the camera itself must shut off the the live view when switching camera modes. However, getting it to restart doesn't seem to work. Linda suspects that it has something to do with Python threading and the way that she is restarting the live view thread is incorrect. Although she has tried everything that she could think of, she has set aside the issue to work on other features. The current workaround is to just restart the Python server and reload the live view from the beginning.
2. Focusing Confirmation: After Linda completed Phase 1, Focusing via Live View, Eric thought that it would be nice to add a feature where the web interface could tell the user whether or not the focusing was successful or not. Upon successful focusing on a given point, a green box would appear around the live view box which would serve as visual confirmation to the user. If unsuccessful, then a red box would appear around the live view to indicate that the user should retry. This feature was not implemented but would have been a useful feature to have as part of the web interface.
3. Image Sorting: With the current implementation of displaying images from the SD card, images are loaded in order of how the camera API decides to return the image information. The order of image information returned from the camera seems to be *sometimes* out of order from how the images appear on the SD card, which are usually organized by date. While sorting pictures by date appears to be doable, this will take extra work to extract further image metadata (such as date taken), and then a sorting algorithm to display the images. Currently the images seem to be generally sorted by date, but not *exactly*.
4. Displaying Picture Using Its URL: The URL displayed on the web interface is generated and updated as the camera takes pictures, and each picture has its own URL. The URL is clickable now; however, the new tab that opens is blank and cannot display the picture taken. To solve this problem, a lot of the time was spent trying to make JavaScript and HTML "grab" the URL and open it in a new window/tab. Unfortunately, this task could not be accomplished due to the nature of Sony API "awaitTakePicture", which was used to display and extract the generated URL which makes the URL change as the camera takes pictures.
5. Control of 2 Sony QX1 Cameras: This was more of a really hopeful stretch goal. While the web interface can control only 1 of the cameras, controlling both cameras would be optimal for the CAVECamX. However, this would mean that the interface would somehow need to connect to 2 different Wi-Fi connections in order to configure both cameras. There isn't a clear way to implement this yet, and this portion will require some time.

Conclusion – This web-based interface is able to help users of the CAVECamX by simplifying both the usage and setup of the cameras. Without a web interface, users must manually configure settings on the camera, and viewing SD card images must involve taking the SD card out of the camera and inserting it into a computer to be seen. The interface allows users to adjust settings, set new focus points, take pictures, and lets them view pictures on the SD card, all without having to physically touch the camera.

We worked on implementing features that we thought would be the most important to a user of the web interface. Being able to set a specific focus point using the live view would allow users to focus on an object with a simple click of their mouse instead of having to do it manually or enter in coordinates into a command prompt. Getting feedback from the camera to when a picture has finishing capturing an image gives users the option to review the latest picture taken. Additionally, knowing when a picture is done can provide information to the gimbal to let it know when to move to the next location. The ability to review images on the SD card and optionally download the full resolution image removes the need to take the SD card out of the camera (and insert it into a laptop) just to review pictures.

While we did implement most of these features, new problems were constantly being faced. Limited camera availability, poor and vague API documentation by Sony for the QX1 cameras, small online community, little to no online help, and unfamiliarity with languages such as HTML and JavaScript were major factors that slowed down our progress. We successfully completed Phase 1, Phase 2 does not yet open up the image in a new tab when clicked, and Phase 3 freezes the live view as well as displays unsorted images. There are currently workarounds for these problems which involve just a simple restarting of the server and refreshing the web page.

The work that we did this quarter opens doors for future developers to add more features to the web interface. Hopefully, in the future it will have a modern UI with sliding windows and interactive panels. Features like a focus confirmation box, more settings adjustment options, maybe even displaying image name and date for the SD card review section, would all be very helpful to the future user of the CAVECamX. Additionally, being able to get camera feedback can be used for the gimbal so that it knows when it is ready to move to the next location to take a picture. This way, the entire system hopefully will become fully integrated.

Although the web-based interface is still in its growing stages, we hope that with the few major implementations of these new features our work will create more future development. The interface provides a simpler yet useful interface for all users of the CAVECamX. With proper documentation and organization of code, we hope that the interface will still continue to be developed after this quarter.

References –

Sony Camera Remote API beta SDK v 2.00
https://developer.sony.com/downloads/camera-file/sony-camera-remote-api-beta-sdk/