Process of creating Liminal

This document outlines the technical processes and explains the code behind Liminal. This project is coded in python. The work was developed in Jupyter Notebook. The code accesses images available from each buoy as part of the BuoyCAM network.

You can find the GitHub repository to Liminal here.

A) Horizon detection:

Yi Qing Ng, a Research Assistant who was working with me on this project began the work of pulling these images from the NOAA BuoyCAM Network, slicing them into discrete images, and then processing them so that their horizon lines would align, and each image would blend into each other. The end result she achieved looked like the following:

This process consisted of the following technical processes:

1. Getting Buoy images

We got our images from the National Data Buoy Center. In our project, we specifically focused on the buoy images from Station 44065 (LLNR 725) - New York Harbor Entrance - 15 NM SE of Breezy Point , NY.

Images provided by buoys in the BuoyCAM network appear like this:

Image from 44065, New York Harbor

This image was taken from New York Harbor Buoy 44065 on 07/13/2024 2010 UTC. Photos are usually taken hourly during daylight hours.

Image from 46047, Tanner Bank

For example, the above image was taken at Tanner Bank, Buoy 46047 on 07/15/2024 1210 UTC.

Occasionally images in the strip taken during daylight hours blacked out. Usually this has something to do with the camera taking the photo:

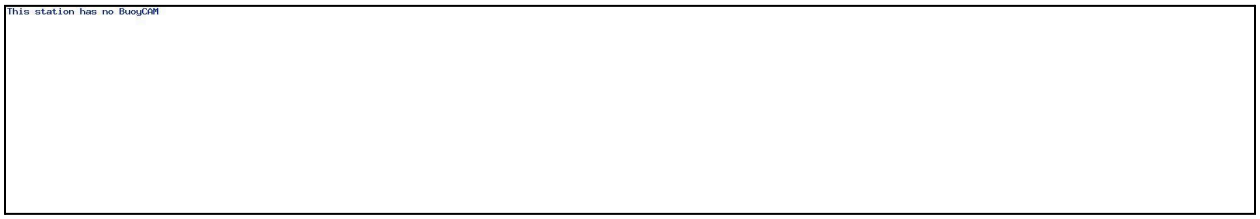


Image from 44065, New York Harbor

For instance, the above BuoyCam image is blank as the buoys occasionally malfunction.

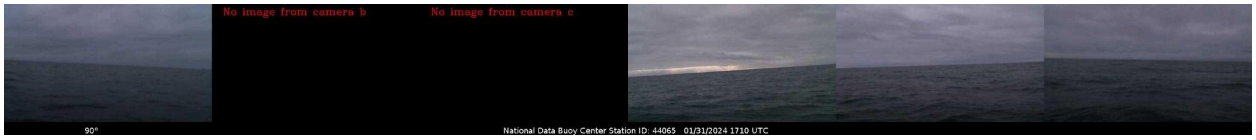


Image from 44065, New York Harbor

In some other instances, some BuoyCams at the same location may malfunction. From this, we learnt that there are six different cameras pointing in different directions at the same buoy location.

We used the buoyant API to draw these buoy images from the Buoy Cams in our code.

```
In [6]: pip install buoyant;
```

```
In [7]: pip install buoyant > /dev/null;
```

```
WARNING: Skipping /Users/pepi/anaconda3/l
adata entry 'name'
WARNING: Skipping /Users/pepi/anaconda3/l
adata entry 'name'
Note: you may need to restart the kernel
```

```
In [8]: from buoyant import Buoy
```

2. Initializing all the images and the folders needed

Because we ultimately want to create a continuously-updated video of buoy images in chronological order, we want to make sure that the images that we are processing are also arranged in a chronological order. That is why we get the current date and time at which we obtain each real-time buoy image:

3. Get current date and time

```
In [10]: # Get current time and date
import os
from datetime import datetime, timedelta

current_datetime = datetime.now()
print("Current Date and Time:", current_datetime)
```

```
Current Date and Time: 2024-08-09 09:25:46.452858
```

```
In [11]: current_date = current_datetime.date()
print(current_date)

current_time = current_datetime.strftime('%H:%M')
print(current_time)
```

```
2024-08-09
09:25
```

And then, we save them in a new folder that is labeled with the date at which the buoy images are saved.

4. Create a new folder every day a new day starts

In [12]:
```
def create_folder_everyday(base_path):
    current_date = datetime.now().date()
    current_folder_path = os.path.join(base_path, str(current_date))

    #if folder doesnt exist yet
    if not os.path.exists(current_folder_path):
        os.makedirs(current_folder_path)
        print(f"New folder created: {current_folder_path}")

    return current_folder_path


## Creating new folder every day for each buoy
base_path_ny = "downloadedImages/NY/raw_images"
base_path_sb = "downloadedImages/SB/raw_images"
base_path_sd = "downloadedImages/SD/raw_images"

#Call function
newFolder_ny = create_folder_everyday(base_path_ny)
newFolder_sb = create_folder_everyday(base_path_sb)
newFolder_sd = create_folder_everyday(base_path_sd)
```

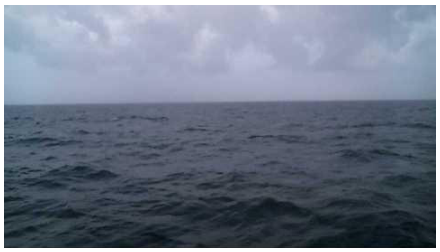
```
New folder created: downloadedImages/NY/raw_images/2024-08-09
New folder created: downloadedImages/SB/raw_images/2024-08-09
New folder created: downloadedImages/SD/raw_images/2024-08-09
```

3. Cropping each collage of 6 buoy images into their individual images

In order to create a video/ gif output of individual buoy images diffusing into each other, we would need to crop the college of buoy images obtained from the API into its individual images.

For instance, from this:

To this:



This is the part of the code that does this cropping process:

7. Now, crop BuoyCam image into its individual images

In [23]:
```
newWidth, newHeight = topPortion.size

# Calculate the width of each part
part_newWidth = width // 6

part_newWidth

save_folder_path = "downloadedImages/NY/processed_images"

#Crop image into 6 parts
for i in range(6):
    left = i * part_newWidth
    upper = 0
    right = (i + 1) * part_newWidth
    lower = newHeight

    part = topPortion.crop((left, upper, right, lower))
    part.save(os.path.join(save_folder_path, f"{current_date}|{current_time}|{i+1}.jpg"))

    display(part)
```

We then save the individual buoy images into a designated folder.

4. Detect the region of interest (ROI) in each individual buoy image

The region of interest (ROI) refers to a particular rectangular portion of each buoy image where the horizon most likely lies in. The following code is the function used to detect the region of interest of each individual image. Below, I have split the entire function up into different specific portions to describe the process of automating the ROI.

a) We start by defining what the approximate Hue, Saturation and Value (HSV) numbers are for the color of the ocean and for the color of the sky (so that we can differentiate those colors and get the horizon line that is in the middle)

```
def automateROI(image):
    # Convert the image to HSV color space
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define range of blue color in HSV
    # HSV Values (Hue, Saturation, Value)
    lower_blue = np.array([50, 10, 10])
    upper_blue = np.array([150, 255, 150])

    # Threshold the HSV image to get only blue colors
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # Bitwise-AND mask and original image
    segmented_image = cv2.bitwise_and(image, image, mask=mask)
```

b) We then convert the image to grayscale, so that it's easier to differentiate the colors. We apply Gaussian blur to the image to reduce noise, and then perform edge detection using the Canny edge detector.

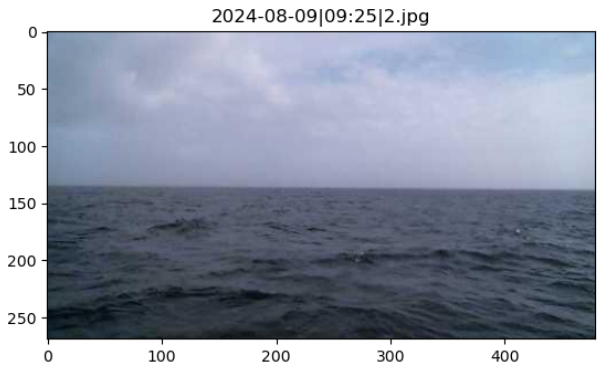
```
# Convert segmented image to grayscale
gray_segmented = cv2.cvtColor(segmented_image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur to reduce noise
blurred = cv2.GaussianBlur(gray_segmented, (5, 5), 0)

# Perform edge detection using the Canny edge detector
edges = cv2.Canny(blurred, 50, 150)

# Perform image dilation to close gaps in edges
kernel = np.ones((5, 5), np.uint8)
dilated_edges = cv2.dilate(edges, kernel, iterations=1)

# Find contours in the edge-detected image
contours, _ = cv2.findContours(dilated_edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

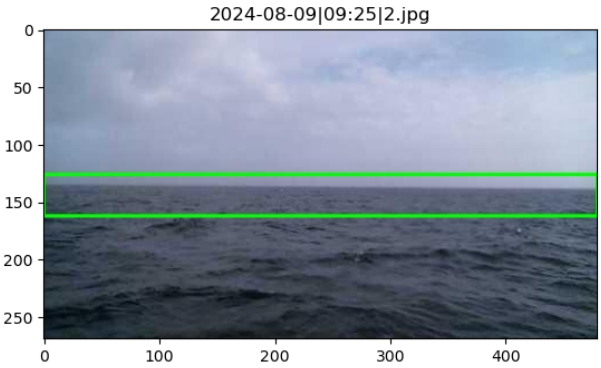
c) If there are no edges/ contours found, it will print the following statement: *"No contours/ no ROI found for {filename}"*. And if there are edges found, it will print the following statement: *"ROI found for {filename}".*

We then find the contour with the maximum area, and project a bounding rectangular box that would contain the horizon edge, which ultimately represents the ROI. We will then draw this bounding box on the image of each individual buoy image.

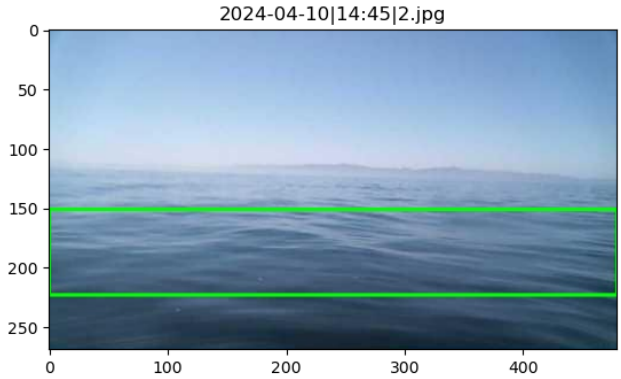
```
# If no contours found, return None
if not contours:
    print(f"No contours/ no ROI found for {filename}")
    print("")
    return None

if contours:
    print(f"ROI Found for {filename}")

# Find the contour with the maximum area (presumably the horizon line)
max_contour = max(contours, key=cv2.contourArea)

# Get bounding box of the contour
x, y, w, h = cv2.boundingRect(max_contour)

# Increase the height of the rectangle by 20 pixels
h += 20

# Create a rectangle covering the entire width of the image and with the height of the bounding box
roi_rectangle = np.array([[0, y], [image.shape[1], y], [image.shape[1], y + h], [0, y + h]])

# Draw the rectangle on the original image
result_image = cv2.drawContours(image.copy(), [roi_rectangle], -1, (0, 255, 0), 2)

return result_image, y, h
```

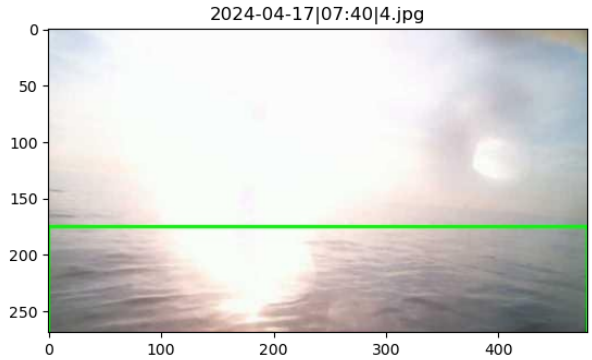
Example resulting image with the bounding box:

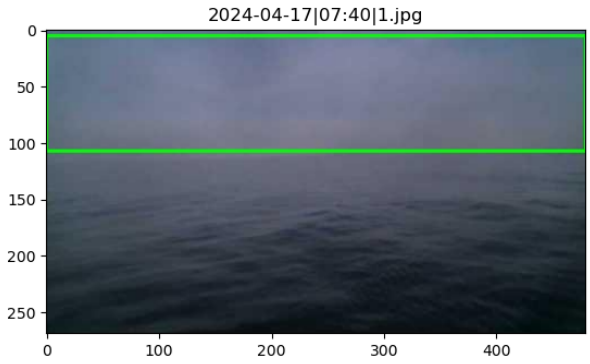


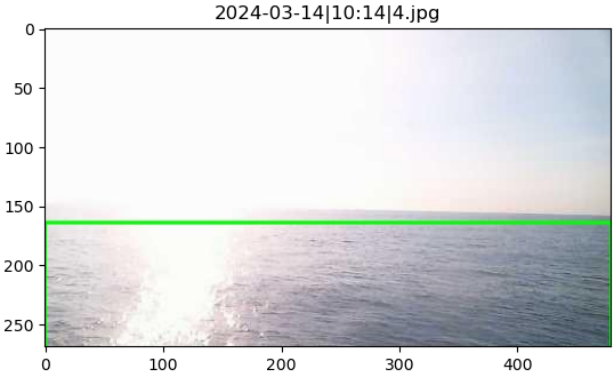


Sometimes, the bounding box might be a little inaccurate, either because of the threshold value used for the Canny Edge detection, or just because some of the images have a blurry horizon line to begin with. For instance:



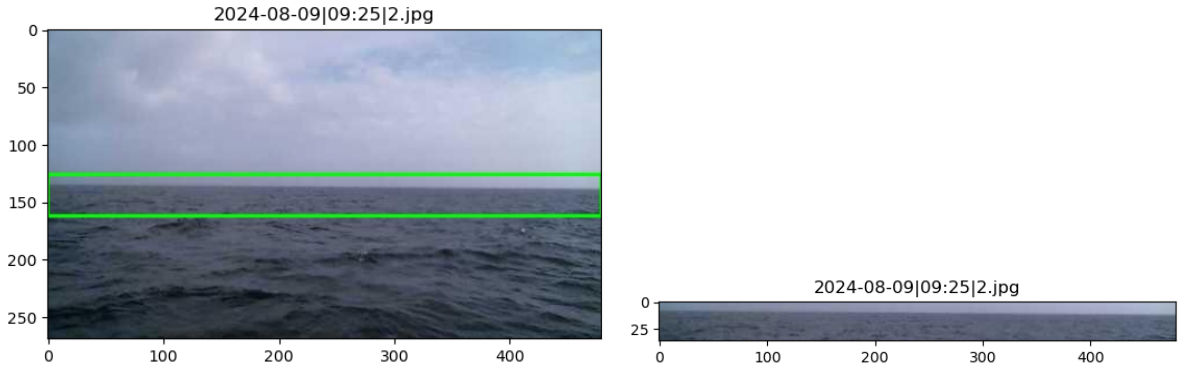






However, for the most part, the edge detection and ROI detection has been accurate.

5. Crop all the images that have a bounding box, into its bounding box.



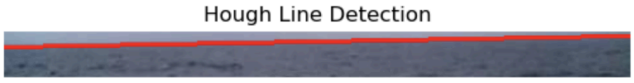
6. Detect horizon line

We then use [Hough Transform Edge detection] to detect the horizon line within the cropped ROI area. We deliberately cropped the original individual buoy image to its ROI first, so that the horizon line would be more easily detectable. If not, Hough Transform might detect a line in other parts of the image.

To do that, we once again have to transform the image into grayscale first, so that the differences in color in the sky and the water is more obvious, and thus the horizon line is more obvious:

If a horizon line is detected, the code would print a result like this, together with the horizon line superimposed on the cropped ROI image:

```
Image with detected line saved to: downloadedImages/NY/processed_images/cropped_images/detectedLines_images/2024-08
-09|09:25|1.jpg
```



If no horizon line is detected, it would print a result like this:

> *No lines detected in downloadedImages/NY/processed_images/cropped_images/2024-08-09|09:25|5.jpg. Moving on to the next image.*

However, there was a huge bug when trying to iterate through all of the images and rotate them all of the images at once.

Because of different lighting + exposure settings in each image, the threshold value of each image to get the Hough Line to show up is different. she used a standard threshold of 150, as I found that this value allowed MOST of the images to have a line detected. However, there are still many images that require a lower/ higher threshold.

If the threshold value is too low, lines are either inaccurate, or many lines will be detected.

For instance;

Threshold value of 150:

Threshold Value of 30:

Threshold value of 200: No lines can be detected.

```
No lines detected in downloadedImages/NY/processed_images/cropped_images/cropped_2024-02-07
|09:23|1.jpg. Moving on to the next image.
```

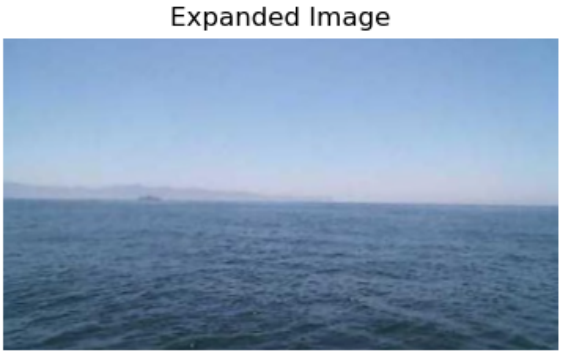
We decided to stick to a threshold value of 150.

7. **Rotate each image such that their horizon lines are perfectly horizontal (y = constant)**

We did this by first calculating the gradient of each line detected by Hough Line Transform, and then we saved their gradients into a list.

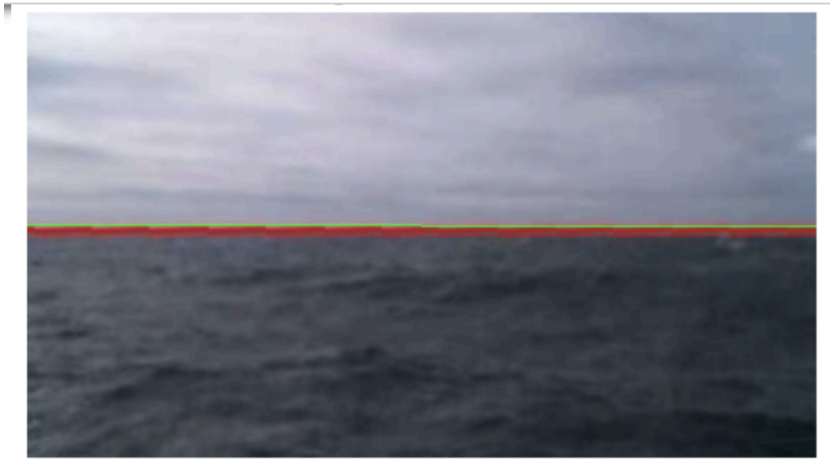
We then rotated each image, according to its gradient.
For instance: On the left is an image with its horizon line detected. We then rotated it by its gradient so that the horizon line is perfectly horizontal.

8. **Expand each image by 20% so that we will not be able to see any of the black borders on each of the images.**



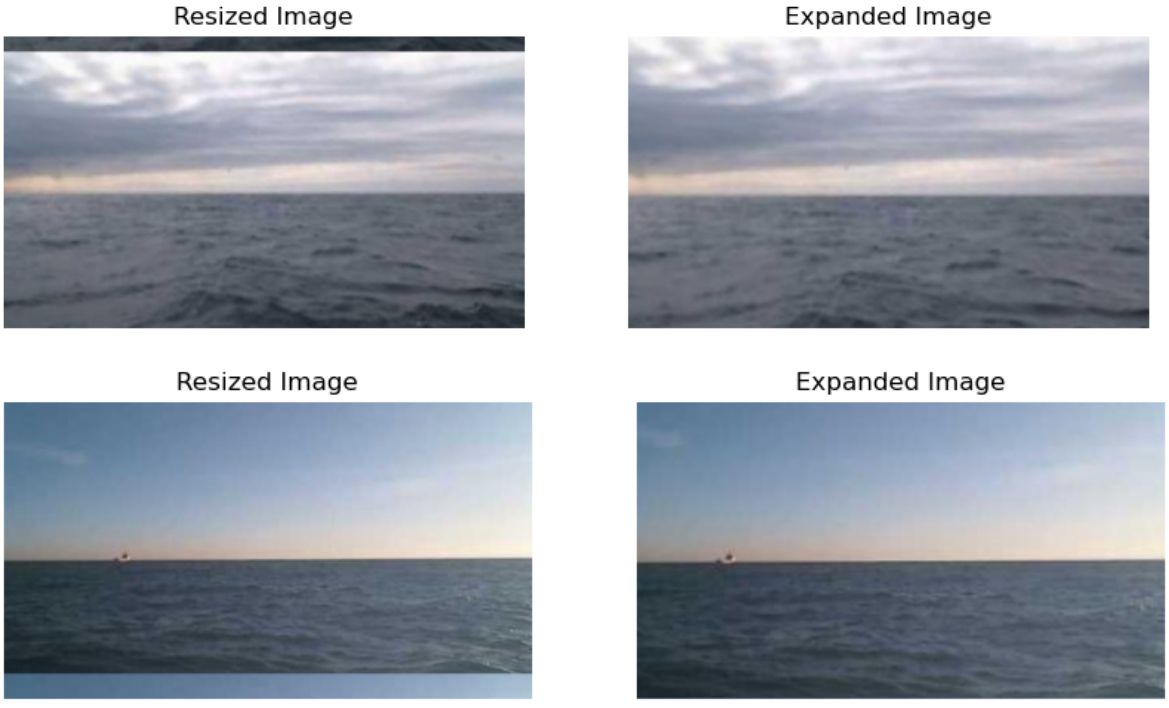
9. **Now, we want to align all of the horizon lines, such that they are on the same y-plane.**

In order to do that, we first got the y-coordinates of all the horizon lines. For instance:



```
y-coordinate of line: 161
```

Then, we get the average of all of these y-coordinates. This y-average will then be the value which all horizon lines will align themselves to. This will require some of the images to be further expanded so that there wouldn't be any blank space below/ above the image as their horizon lines get shifted up/ down to align with the y-average value.



10. **Finally, creating the video/ gif of all the expanded images with their horizon lines aligned.**

B) Outstanding issues

Eventually it was decided to drop images where the horizon line are undetectable as well as blacked out images from when the cameras did not return good image values or nighttime exposures were too dark. Whether or not this process could be better optimized is something that could be revisited.

There is still a big question about the degree of merging between images and the pace at which they transition. The video linked here is a lot smoother, but still doesn't have the right feeling of liminality. Perhaps there may be some image blending or other processing that can keep the brightness of the image, but increase transparency, and actually blend/average the images together more. I am seeking very subtle movement, an uncanny quality, akin to (but obviously not exactly like) After the Deluge by Paul Pfieffer[1].

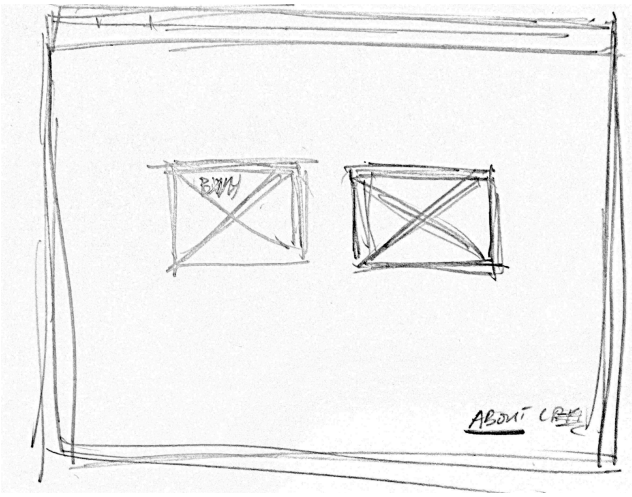
The project needs to develop a proper pipeline to retrieve, store, display and then refresh images. The current code with images is very large. This isn't necessarily a problem, but storage and retrieval of images pulled from the BuoyCAM network is another outstanding issue to address.

Additionally the image and/or video output needs to be able to be presented on a website and as an installation. Each format requires that the processed images have some kind of pipeline to end up in their final forms.

C) Website:

[1] https://www.thomasdanegallery.com/artists/47-paul-pfeiffer/works/17720/

The processed images from each buoy should be displayed as two images side by side. The site is envisioned with minimal UI, with only a small link to an about page at the bottom right hand corner of the page. One of the main aesthetic elements to get right would be the aspect ratio of the images and proportions relative to the rest of the page.

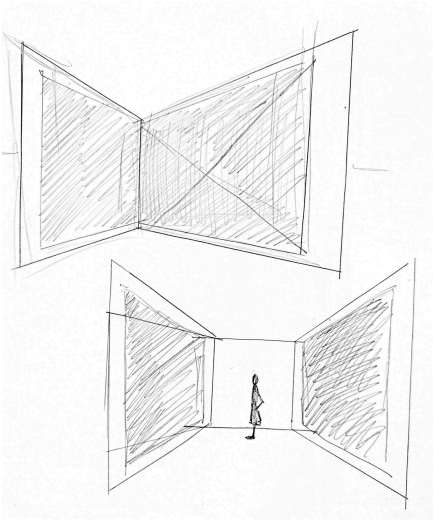


The web application is currently built on Flask, which you can find in the github repository.

However, as of Aug 20, 2024, Pepi has not managed to finish the Flask website and thus, the web application is currently not live. The code can still be run in Jupyter Notebook.

D) Installation:

In the installation view the images should be able to be projected independently.



The above image is just a sketch, and might need to be adjusted according to image quality and size of projectors. To be honest, it will be hard to assess final dimensions until looking at the

output images and how they appear projected. Perhaps the website itself would look better as a projection. Still this is another form the work is envisioned to take. Additionally in order to accommodate this view it may require importing images into another system (such as touch designer.)