‘

“Unstable Bluff” Detection System

Team Members:

Omar Vizarraga
Nikhil Maharaj
Juan Conriquez
Roberto Burciaga
Patrick Stewart
Evan Paulino

System Description

Due to the instability of the bluffs in Del Mar, it has been assigned to our team to create a system that is able to detect and analyze any changes to these bluffs. The system will send notifications to nearby residents and train operators to alert of any hazards.

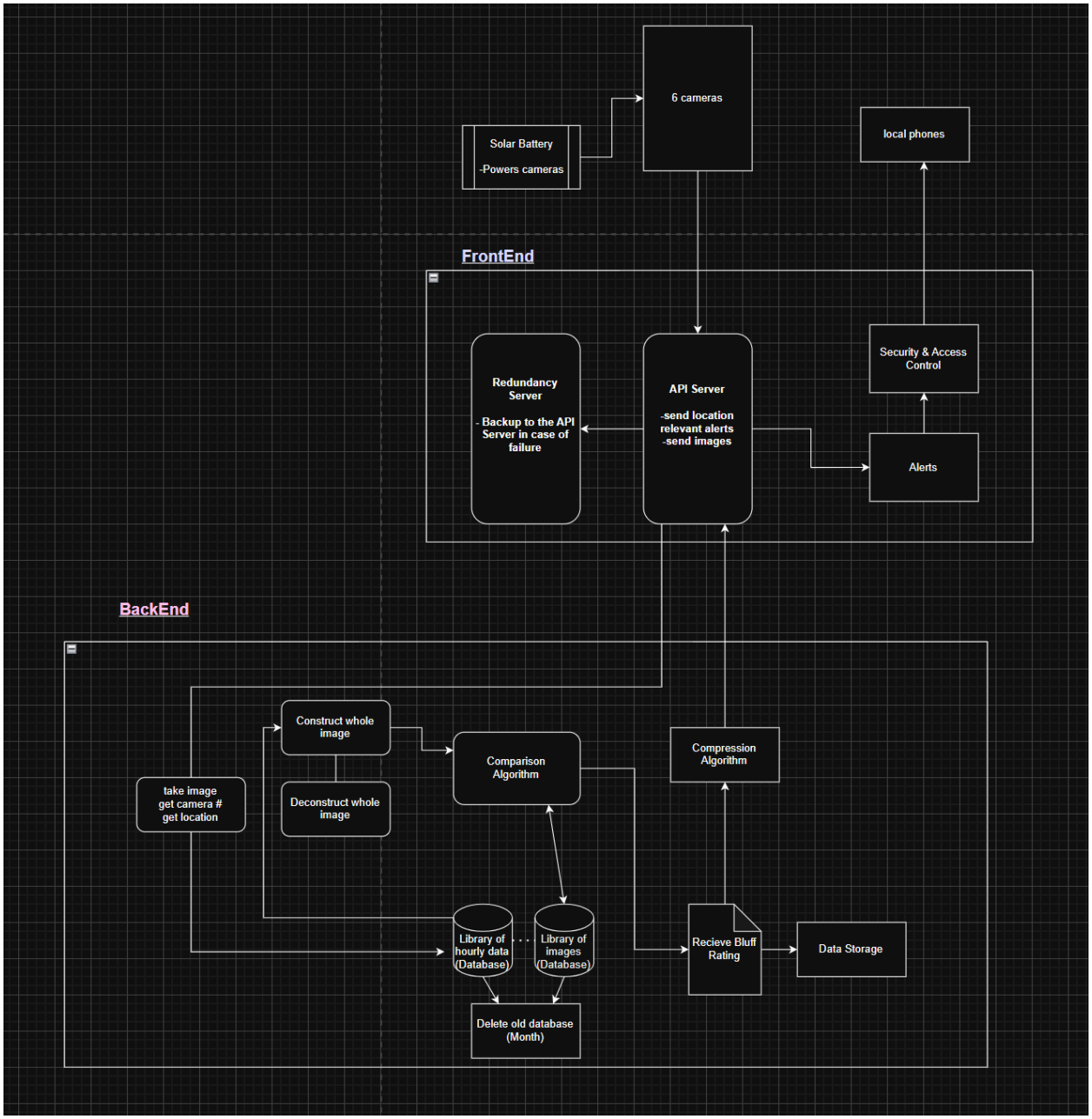
The system will use six fixed 10-megapixel wifi-enabled cameras that will monitor a 300 foot stretch of the Del Mar Bluffs. Each camera will cover a 50 x 50 ft space with an image resolution of 4,000 pixels per square foot. These images will also contain a 32-bit time stamp and a 32-bit geo location to help with the analysis. The cameras will take images in one hour intervals. The images will then be sent wirelessly, via wifi, to a local test facility where they will be stored and analyzed. The comparison between two images will be rated on a scale from 0 - 5 depending on the severity of the changes.

- Ratings between one and three are considered predictive of a significant slide.
- A rating of four indicates some significant change and further evaluation is required immediately to determine if rail traffic should be stopped and people evacuated from the area.
- A rating of five indicates that a major change occurred, rail traffic should be stopped immediately, and people evacuated from the area to avoid injury or death. In addition, the image indicating a major change should be compared to the last image at that location to determine if trains or people were in the area right before the bluff change and rescue operations initiated.

Ratings of 4 and 5 will send out alerts to railroad operators, lifeguards, and nearby residents.

Software Architecture Diagram Overview

SWA Diagram



SWA Diagram Link:
https://drive.google.com/file/d/1ATTj5aAgU1bRCzzvoQFFsT_n4rDxoqpE/view?usp=share_link

Architectural Diagram Components and Descriptions

Outside Components:

- Solar Energy Battery: This component is the power source for the camera. It will use solar energy to power the 6 cameras used to take the images.
- 6 cameras: This component is the cameras, with built in WIFI, that take pictures of the bluff. It sends the images to the API server, hence why there is an arrow pointing to the API server.
- Solar Energy Battery: A solar powered battery will provide energy for the cameras.
- Repeater: This component amplifies the signal for the test facility as well as the cameras. Creates a stronger signal to make sure information is able to be sent properly.
- Local phones/computers: These are the recipients of any alerts that are going to be sent out. Will be either sent as a text or as an email.

Frontend Components:

- API Server: This component receives images from the cameras then uploads the images to an hourly image database. This component also sends out alerts if the comparison of the images score a rating of 4 or 5.
- Alerts: This is a component that sends out alerts and requires information from the API server to send/display the correct message based on the rating of a comparison that was made.

Backend Components:

- Library of hourly data (database): Images from the API server are sent here to be stored.
- Construct Whole Image: This component takes 6 images from a certain time and constructs one big panorama image. It then sends the panorama image to the comparison component.
- Deconstruct Whole Image: This component deconstructs the panorama image that is created back into the 6 original images.
- Comparison Algorithm: This component compares images and creates a rating for the comparison between the two images. The rating is then sent to the database of comparisons.
- Comparisons Database: This database holds comparison objects that holds information regarding the comparison between two images. Such as the rating, which images that were compared, and its distinct comparison number.

Architectural Diagram Flow Description:

The SWA diagram consists of frontend, backend , and outside components. The frontend consists of components that make up how the interface looks and what the user will see when interacting with it. The backend consists of components that will store data and perform functions in order for the information to be sent back to the API server on the frontend to display information. The components that are outside either of these frames are actual physical components needed to retrieve and send information properly.

The arrows in the diagram are either one sided or double sided. The single sided arrows mean that a component interacts with another in a singular direction.

For example:

The API server has a single sided arrow pointing towards alerts. This means that the information sent to the alert component only interacts with it in that direction. The alert component can't send information back to the API server.

The double sided arrows mean that the components connected by these can interact both ways.

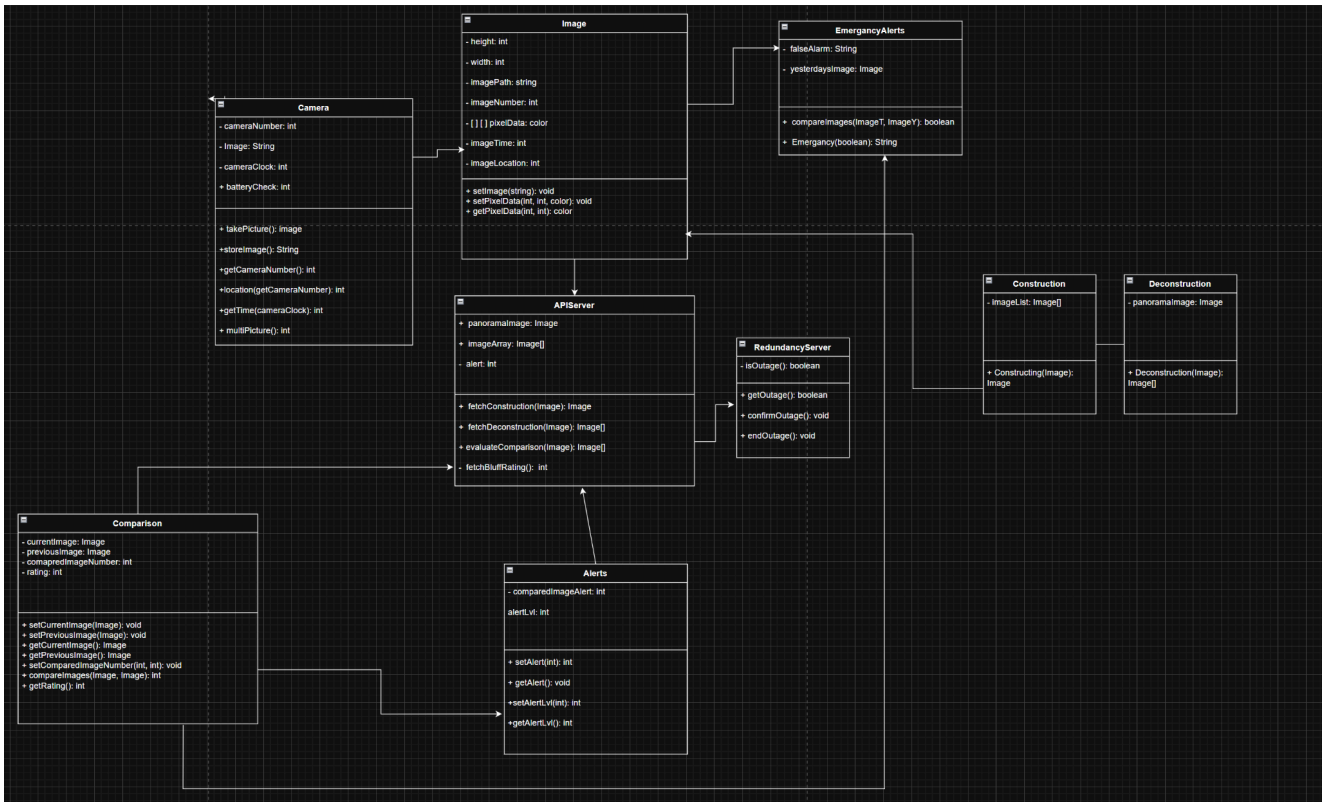
For example:

The API server, repeater, and cameras are all connected using double sided arrows. This allows for the cameras to send information such as images to the API server. This also means the API server can send the camera information. This can be information regarding changes to the camera's settings in order to change the time interval at which pictures are taken.

Therefore, following the arrows from component to component in the diagram shows how all components interact to make up the system.

UML Class Overview

UML Diagram



UML Class Diagram Link:
https://drive.google.com/file/d/1hni8QpLTemc4JfEW2FYqdzLGweVK2kJV/view?usp=share_link

Description of Classes + their Attributes and Methods:

Image:

The image class will hold all the information regarding an image.

Attributes:

- height: int
    - This class variable is private and stores the height of the image, in pixels, and is an integer type.
- width: int
    - This class variable is private and stores the width of the image, in pixels, and is an integer type.
- imagePath: string
    - This class variable is private and stores the path of the image file. The variable type is a string.
- imageNumber: int
    - This class variable is private and creates a number to be associated with an image. The variable type is an integer.
- [ ] [ ] pixelData:
    - This class variable is private and stores the image's color data at every pixel in a 2D array and is a color type.
- imageTime: int
    - This class variable is private and will store the time that an image was taken. It will store the time in the format of:
        - MM/DD/YYYY HH:MM:SS
    - Variables are defined respectively to the order of the format presented above.
        - MM: Represents the month from 01 - 12.
        - DD: Represents the day from 01- 31.
        - YYYY: Represents the year 0 - current year.
        - HH: Represents the hour of the day from 00 - 23.
        - MM: Represents the minute of the day from 00 - 59.
        - SS: Represents the second of the day from 00 - 59.
    - This variable is an integer type that is stored as a binary number. The binary integer value can then be converted into base 10 to make it easier to read when referring to the date and time of the picture.
- [ ] [ ] imageLocation: int
    - This class variable is private and will store the location of where the image was taken. The image location will be a coordinate in reference to the top left corner of the 100 ft x 300 ft square ft area captured by the six cameras. The top left most square foot will be (0, 0) and the rest of the square foot coordinates will follow a 2D array index pattern.

- Example:

(0, 0)	(0, 1)	(0, 2)	(row, column)
(1, 0)	(1, 1)	(1, 2)	…
(2, 0)	(2, 1)	(2, 2)	…
(row, column)	…	…	(row, column)

Methods:

+ setImage(string): void

- This method is public and takes in a filepath string and sets the attributes of the Image object according to the image the filepath directs to.

+ setPixelData(int,int,color): void

- This method is public and takes in two ints indicating the pixel row and column, and a color to set that pixel to.

+ getPixelData(int,int): color

- This method is public and takes in two ints indicating the pixel row and column, then returns the color of this pixel.

Comparison:

The Comparison class will hold attributes regarding the comparison of two images taken at separate times. It will be able to compare images and create a Comparison object that stores the rating for the two images that were compared. It will also store the two images that were compared along with creating a distinct number associated with the images that were compared for reference.

Attributes:

- currentImage: Image

- This class variable is private and stores the current image to be compared to a previous image. The variable is an image type.

- previousImage: Image

- This class variable is private and stores the previous image to be compared to a current image. The variable is an image type.

- comparedImageNumber: int

- This class variable is private and stores a unique number associated with the comparison of two images that were compared. The variable is an integer type.

- rating: int

- This class variable is private and stores the rating of the comparison between the current and previous images that were being compared. This variable is an integer type.

Methods:

+ setCurrentImage(image): void
  - This method is public and it takes an image object as an argument that sets the value of the currentImage. It has no return value.
+ setPreviousImage(image): void
  - This method is public and it takes an image object as an argument that sets the value of the previousImage. It has no return value.
+ getCurrentImage(): Image
  - This method is public and gets the current image of a Comparison object. Its return value is an image type.
+ getPreviousImage(): Image
  - This method is public and gets the previous image of a Comparison object. Its return value is an image type.
+ setComparedImageNumber(int, int): void
  - This method is public and uses two integer arguments that are image integer values. It sets a unique value to these two images being compared that can be used as a reference. It has no return value.
+ compareImages(Image, Image): void
  - This method is public and takes two image arguments that are the currentImage and previousImage. It will compare the images and sets the rating value from 0 - 5 depending on the severity of the changes between the images. It has no return value.
+ getRating(): int
  - This method is public and it returns the rating of a Comparison object. It returns an integer value.

Camera:

The camera class functions as the capturer of physical data into data we can compare and alert about.

Attributes:

- cameraNumber: int
    - This private class variable holds the camera's specific number so that it may compare with its own database of images.'
- Image: String
    - This private class String variable will hold the image to compare
- cameraClock: int
    - This private class int variable keeps an internal clock counting to log images and make GIS tracking available through comparing different timed images.

+ batteryCheck: int
    - This class returns the amount of battery the camera has, which will probably change to display as having to check every single time would be too inefficient

Methods:

+ takePicture(): image
    - This method signals the camera to capture an image at that moment.
+ storeImage(): String
    - This method stores the image with the time of the clock, and the number of the camera, to log it into the database later and ease comparison later.
+ getCameraNumber(): Int
    - This gets the number of the camera the image was taken on.
+ location(getCameraNumber): Int
    - This uses the getCameraNumber() function to find the camera used and access the preset locations of the cameras. From there you can tell what part of the bluff is being compared.
+ getTime(cameraClock): Int
    - This method can be used to call for the time of the image.
+ multiPicture(): int

- This method makes it so you can take a rapid succession of pictures in case of movement. This helps the before and after, and maybe to try and determine the cause of instability.

Alerts:

The alert server class functions as a back-end and physical indicator about the camera's readings. The alert messages should are intended to be sent to railroad operators, lifeguards, and nearby residents.

Attributes:

comparedImageAlert: int

alertLvl: int

Methods:

setAlert(int): int

- Updates the alert to see if it will be sent, what level bluff warning it should display, etc

getAlert(): void

- Retrieves the alert and can be sent to local phones.

setAlertLvl(): void

- Sets the possible alert levels by setting a threshold for comparison in the level of risk of the bluff

getAlertLvl(): int

- Gets the Alert level from the image to send back in the alert

isAlertRead(comparedImageAlert): boolean

- Allows the people who have already received the bluff alert to close it once read.

RedundancyServer:

The redundancy server exists as a backup API server/database node so as to ensure the system keeps running even if the main server fails. This could happen if there are too many requests or pings to the main server.

Attributes

- isOutage: boolean
    - This is a private boolean variable that indicates whether the API server is down.

Methods

+ getOutage(): boolean
    - Returns the status of the server outage.

+ confirmOutage(): void
    - Sets isOutage to true.

+ endOutage(): void
    - Sets isOutage to false.

API Server:

The API server class functions as a managerial position, where if we need to compare images for alerts and fetch their rating, construct or deconstruct total panorama images of the cliff side from the 6 cameras, and or send Alerts to their corresponding levels. The API server class will allow our algorithms/ classes and databases to act swiftly without issue.

Attributes:

panoramaImage: Image

- The attribute allows for a pre-made variable of the Image type to create a single, compiled image (comprising all 6 images) of 50x50ft.

imageArray: Image[]

- The attribute allows for a pre-made variable of the Image array type, so that the 6 images taken can be accessible at any time in between the methods.

alert: boolean

- This attribute is considered to be our “standard” alert, as when there are no alerts it is assigned 0 to not alert, but always given a different alert value based on the method's return.

Methods:

+ fetchConstruction(Image): Image

- This method is used by the API when tasked to construct the panorama picture (50x50ft) to get the order of the construction and get the constructed image by another class in the program.

+ fetchDeconstruction(Image): Image[]

- This method is used by the API when tasked to deconstruct the panorama picture (50x50ft) to get the order of the deconstruction and get the deconstructed image by another class in the program.

+ evaluateComparison(Image): Image[]

- This method is used by the API when tasked to make the comparison and evaluate the difference between photos taken by cameras 1-6 through a different class in the program.

- fetchBluffRating(): int

- This method is used by the API when tasked to construct the panorama picture (50x50ft) to get the constructed image by another method in the program.

Construction:

This class is used to take in the 6 photos along the cliff side, and constructs a single panoramic image.

Attributes:

imageList: Image[]

- The imageList Attribute serves as a private array list placeholder. This allows for future moderation when the class receives an actual image array to construct a single panorama image.

Methods:

+ Constructing(Image): Image

- The Constructing method is used to take in the image array and combine all images to make a single panorama image.

EmergencyAlerts:

This class is used as the final two-step verification method for when the bluff rating is a rating of 5. Once declared a rating 5, the program will use this class to retrieve today's and yesterday's images and compare them to reverify if it's true or a false alarm.

Attributes:

falseAlarm: String

- This attribute is only used when the images both look the same and don't equate to a bluff occurring at alert of level 5 and produce a false alarm response print.

yesterdaysImage: Image

- This attribute is used as a placeholder for when the Emergency alert doesn't produce an image to go back to review and just uses the same photo as the current day to produce a false alarm.

Methods:

+ compareImages(ImageT, ImageY): boolean

- The compareImages method is used to compare both ImageT (Image Today) and ImageY (Image Yesterday), and compare the images' pixels via monochrome or colored to verify any major pixel differences. Once compared, a boolean is returned to signify either true significance for an emergency or false.

+ Emergency(boolean): String

- The Emergency method is the last step towards this two-step verification method for an Emergency Alert. The boolean will determine whether a String is returned that states to evacuate or false alarms.

Security & Access Control:

- The program itself requires a necessary stop-block method in case of unauthorized access attempts to the server.

Compression Algorithm:

- Before the system sends out the bluff rating information, the data must go through the data-preprocessing unit in the form of the compression algorithm.

- The algorithm will simply compress the data and remove all noise and extraneous information. This will allow the server to work more efficiently given it will not have to undertake a bigger data-load.

UML Class Description:

The UML diagram paints a clearer image of how each major component of our software will work with one another. In this instance arrows are a bit more specific seeing that they connect to either a method within the class or an attribute.

This diagram provides more specific insight into the classes we will have to program for the entire product to be complete. For example, the connection between the image upload and the sorting algorithm can be traced from the way the camera captures, stores and finally compares and rates bluff warnings

The API server once again is the meeting point for these classes and converges all of the back end processes to a visible warning to locals and administrative users. The Uses cases were not specified as the audience would be broad and was not specified in the description. We can only assume that this warning will be accessible to essentially anyone.

Development plan and timeline:

The tasks for this architectural diagram and UML chart, were started as a free flow web of ideas in which each team member could contribute something to later be added into the visual charts. Additionally, a lot of us added our names into the document to inform each other what we wrote and were specifically going to work on in the charts. Hopefully these will help you, the reader, as well.

Each team member was responsible for not only contributing, but also making sure those contributions were properly logged into the team's github.

Group Member Responsibilities:

Omar:

- Write the software description
- Add comparison, image, and alert class into UML class Diagram
- Provide descriptions of the attributes, methods, and class description for these classes
- Help with how the classes are connected
- Reorganize and edit SWA diagram arrows so it is easier to understand how the components flow and interact with each other
- Add database of comparisons to SWA diagram
- Add descriptions of the SWA diagram components and explanation of how it flows

Roberto:

- Construct the SWA diagram with the frontend and back end
- Add the cameras, API server, construct whole image, deconstruct whole image, comparison algorithm, hourly data database, and image database components into the SWA diagram
- Connect these components to show how they flow together
- Add APIServer, emergencyAlerts, Construction, and Deconstruction classes to UML class diagram
- Give description of these classes, their attributes, and their methods.

Nikhil:

- Add Compression Algorithm (Data Pre-processing), a Redundancy Server, and Security & Access Control modules onto the SWA Diagram and into the program proposal group document.

Juan:

- Fill in camera and alert class in the UML class diagram with attributes and methods
- Add or build upon these two things in the SWA program
- Architectural Diagram flow description (UML)

Evan:

- Add more external components to the cameras like the solar energy and the long-range connectivity device for the server, to better optimize the uptime of our cameras and the efficiency of transferring data
- Add different functions to integrate the use of the battery into the camera, like checking the battery and also taking multiple pictures.

Patrick:

- Add solar battery to SWA diagram
- Add redundancy server to UML diagram
- Organize document components and formatted layout
- Revise class attributes and methods
- Create redundancy server methods and attributes
- Final revision and submission

Timeline:

All members are expected to finish their responsibilities at least an hour before the assignment due date. This will give team members time to review and fix any issues.

Due Date: October 11, 11:59 p.m.

Test Plan:

This section lays out test plans to verify components work as intended. There are at least two test sets for each granularity level: unit, integration, and system level tests. These sets each consist of one or more tests that target a feature. We explain the tests as much as possible, identifying what features of our design we are testing, what the test sets/vectors are, and how our selected tests cover the targeted features.

Tests Cases:

1. **Camera Capture**
2. **API server connectivity and Camera connectivity**

Unit Tests:

Solar Charge Capacity Test:

This test will check if the solar chargers that are hooked up to the battery are keeping the cameras charged during the night by running the batteryCheck hourly after sundown.

- Input: running batteryCheck with battery above 15%
- Expected output: nothing
- Input: running batteryCheck with battery at 15% or lower
- Expected output: low battery warning

Server Connection Test:

This will test the long-range connector by taking test images and seeing if it transfers through the server to the main hub.

- Input: activation of cameras
- Expected output: properly transmitted photos in main hub

Camera Capture Information Test:

This test makes sure that each image taken gives the correct timestamp and geolocation of the image. An image will be taken and the timestamp and geolocation at that moment will be recorded and compared to the image's logged data. This reassures that the image's timestamp and geolocation information are accurate to the actual time and geolocation.

- Input: activation of camera
- Expected output: accurate timestamp and geolocation fields

Camera “only one” Capture Test:

This test ensures that the cameras all use an image taken at the same time for accurate comparison. Therefore, we run a unit test that checks that only one image was taken per camera and that no duplicate replaces it. (This could be caused by an information or timing error which “Camera capture Information Test” checks for). A function loops to test how many times capture() is called and we use that to check.

- Input: signal to activate cameras
- Expected output: one image (capture() call) per camera, with matching timestamps

Integration Tests:

Redundancy Server Test

Test outage by cutting off power to API, if redundancy is notified, receive a isOutage boolean true statement

- Input: power off API server
- Expected output: isOutage = true
- Input: disconnect API server from system
- Expected output: isOutage = true
- Input: ensure connected API server and power
- Expected output: isOutage = false

Server Upload Interval Test:

Tracks the time between taking a photo, uploading it, transferring to the main hub, and projecting it to a display to get an accurate time displacement period so the users know when they can expect an upload.

- Input: activation of cameras
- Expected output: an upload within 30 seconds

Camera Capture Transfer Test:

An image will be taken then the database will be checked. The images will be compared to make sure that the images were successfully transferred and match the ones taken. Any WIFI or IP problems will be checked through this test.

- Input: transmission of images to the server
- Expected output: images matching the input on the server

Redundancy Server Backup Test:

The redundancy server serves as a backup and thus the test will take inputs from the redundancy server backup and recheck with the API and itself before concluding that all inputs and outputs through the API are sound and coincident.

- Input: an identical set of images transmitted to both API and redundancy servers
- Expected output: A boolean value reflecting the equality of the server outputs; true will indicate that the servers are equal, and false will indicate some kind of discrepancy.

Compression Algorithm Test:

The compression algorithm test will intake both the pre and post processed data (in small, randomized, but corresponding packets and compare the file size and whether or not there is any noise in the data. The compression should be fairly rudimentary, but the file size should decrease and thus the output will be a boolean indicating whether or not the file sizes have decreased or not.

- Input: corresponding packets of image files
- Expected output: identical image files with a file size at least 40% smaller
- Input: corresponding packets of processed image files
- Expected output: identical image files with a file size at least 40% smaller

System Tests:

Camera Capture Accuracy Test:

This test will take two images. Once the second image has been taken we will compare what the API server does to what it should do. If there was a notable change between the two images then the API server should have sent alert messages out. If there wasn't a notable change between the two images then the API server should not send out alerts.

- Input: activation of camera twice, without altering monitored subject
- Expected output: no alert
- Input: activation of camera, altering monitored subject, activation of camera again
- Expected output: alerts sent out from API server

Panorama Capture Test:

When the array of images from all the cameras sends them over to the construction class to make the Panorama, it should be transferred over to the comparison algorithm. If it notices that the Panorama constructed does not produce a whole image (because one picture wasn't taken from the list of 6 cameras), the Bluff Detection team should be notified via a message to inspect the camera for replacement or repairs.

- Input: activation of all 6 cameras with capture()
- Expected output: a completed panorama
- Input: activating all 6 combinations of only 5 cameras at a time
- Expected output: error message regarding missing camera
- Input: activating all 14 combinations of only 4 cameras at a time
- Expected output: two error message regarding missing cameras
- …
- Input: activation of capture() with all cameras non-functional
- Expected output: six error messages regarding all cameras

Server Backup Test:

After a power outage we must ensure that the server that has all the data can pick up where it left off in case of a power outage. This will be through consistent saving intervals to ensure data loss is minimal.

- Input: run a control bluff evaluation. Then, run test evaluations that are interrupted by a power shutoff, ranging from 10 minutes to 3 hours, 12 hours, and 24 hours—all at each step of the bluff evaluation process.
- Expected output: continuation of the evaluation process, identical to the control run.

Phone Alert System Test:

Most commonly seen in everyday life, when safety protocols need to be ensured, a simulated emergency might sound to “Test” its intended functionality. What will happen is a simulated rated 5 bluff rating will sound to test if phones in the approximate area get the notifications. For that to happen the test will stimulate the Alerts in Bluff ratings, and then the API Server should direct it normally to the Alerts feature which will notify phones of a simulated bluff evacuation emergency. If the whole process does not occur, it means the test failed.

SWA Diagram with Placement of Test Cases: