Unstable Bluff Detection System

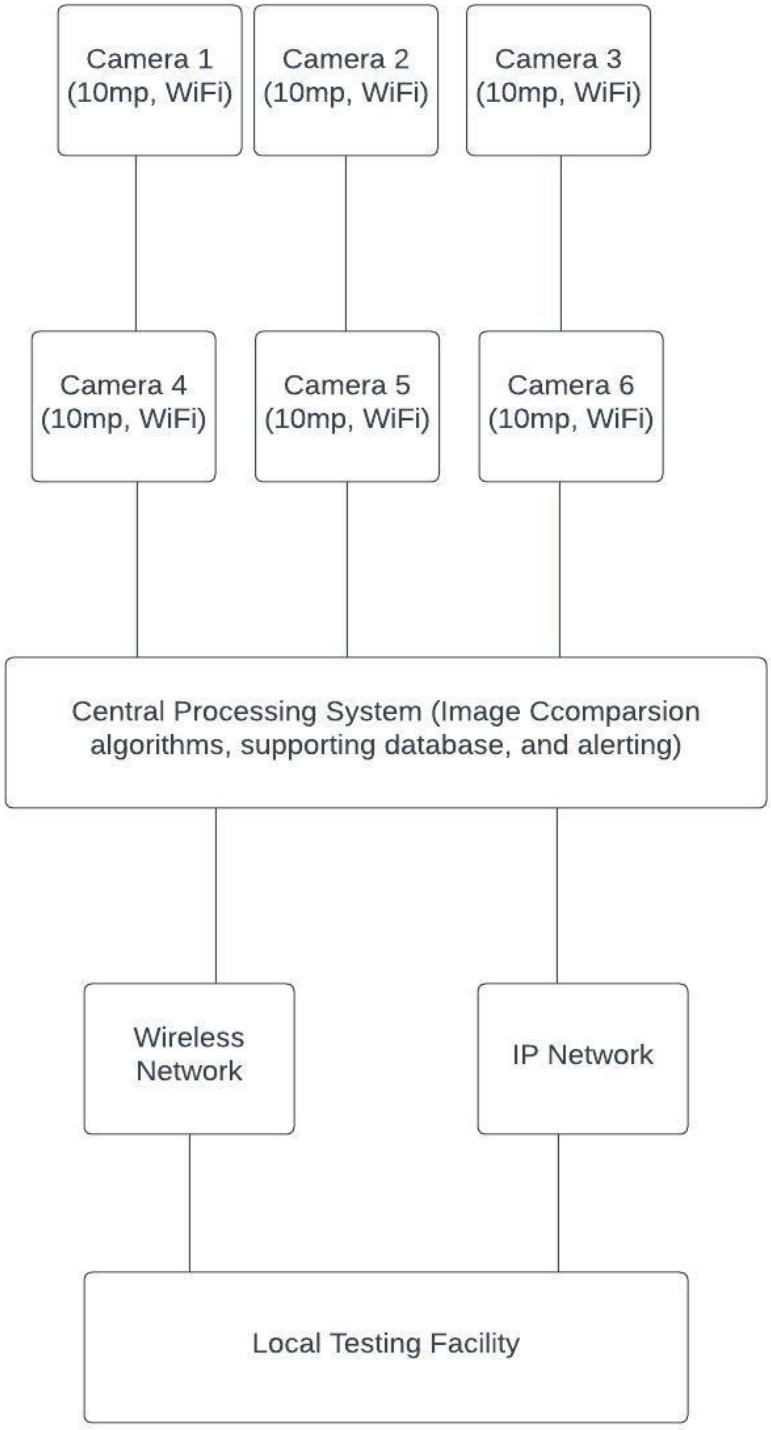
Prepared by: Jessica Chammas, Ashley Olson, Jared Pueblo

March 25, 2023

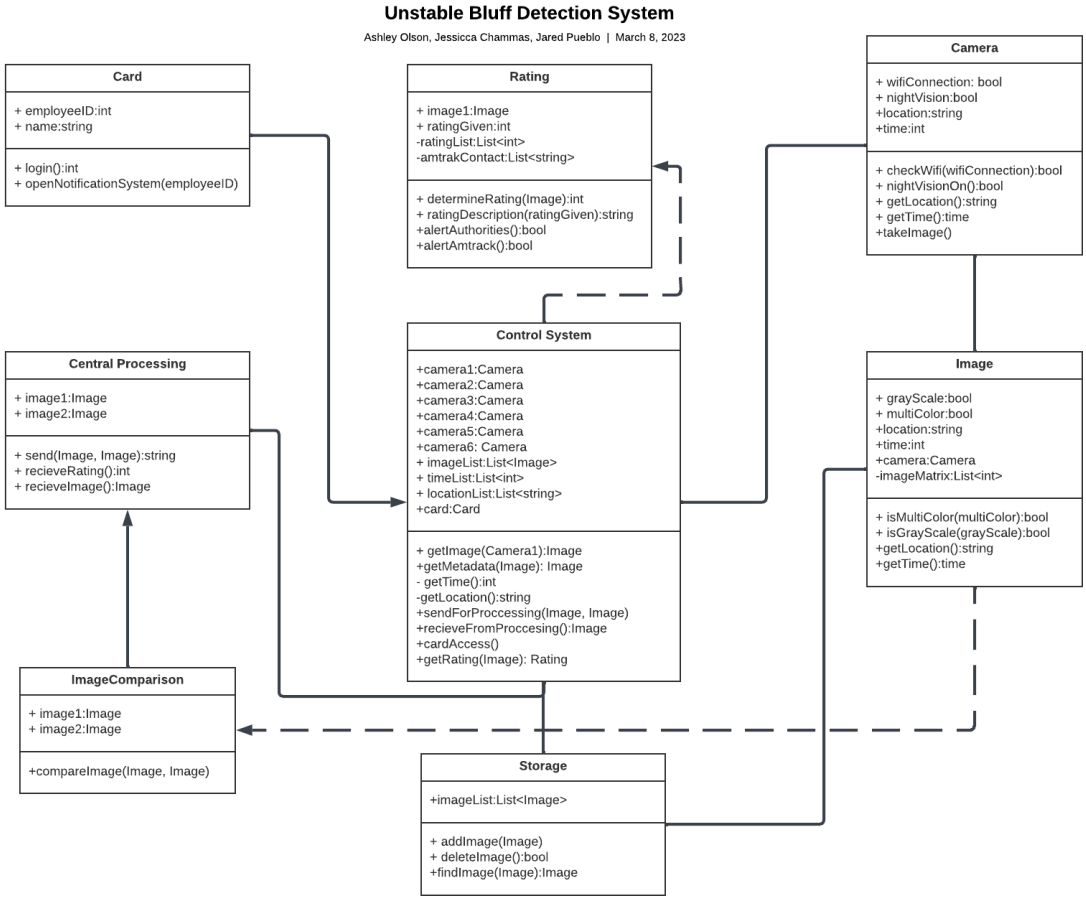
System Description

This system is designed to monitor a 300 foot stretch of bluffs in Del Mar, California to prevent hazards to train tracks at the top of the bluffs as well as potential harm/ loss of life to people sitting on the beach near the bluffs. The system uses six fixed cameras mounted on poles that each capture a 50-foot by 50-foot image of the top of the bluff. Each image includes a timestamp and geolocation information. These images will be sent to the central site processing system which has image comparison algorithms and databases implemented to rate any changes in the bluffs from zero to five, with ratings of four and five causing alarms to go off and messages sent to rail operators and lifeguards.

Software Architecture Overview



UML Class Diagram



Description of Classes

The Control System class' design is for being able to share information of the Unstable Bluff detection system with other classes and its primary purpose is to be able to send this information to the central testing site. The Control System has a dependency with the RatingSystem class and associations with the CentralProcessing and Camera classes.

The ImageComparison class is in charge of gathering the data of any two images taken from the same camera to be compared if an unstable bluff has occurred. Then, the data should be sent over to the Central Processing class for making additional connections through wireless networks and IP addresses to ultimately send to the central testing site. The Image Comparison class has a generalization to the Central Processing class and a dependency from the Image class.

The Image class will hold the locations and time of the images taken through the Camera class. The Image class will hold the 32-bit timestamp and 32-bit geolocation. The class will check the color of the image and determine the color along with its locations and time stamp. The Image class has a dependency to the Image Comparison class and an Association from the Camera class.

The Camera class will make sure that the camera in the detection system is connected to the Wifi network and take an image. The Camera class will also detect if night vision has been activated and add metadata of where the location and the time of the image is taken. The Camera class has an association with the Image class and Control System class.

The Rating class is in charge of receiving an image and determining its rating. If the rating is 4-5, then the class will alert authorities (lifeguards included) and Amtrak. A description

will be given as well with what type of severity the bluff is at to show the risk of danger for the community. The Rating class has a dependency from the Control System class.

The Central Processing class will take two images and send them out to the testing facility. When the images are scanned for processing, the class will receive a rating along with its corresponding image for validation. The Central Processing class has an association from the Control System class and a generalization from the Image Comparison class.

The Card class is for Amtrak members who have a card and wish to logon to the system. The class provides a login with the card and access to the Control System to be able to access images in addition to camera feed. The Card class has a generalization to the Control System class.

The Storage class is meant for keeping track of all images taken from each camera and stores them within a list that can be modified accordingly. The class will be able to add, delete, and find images based on the metadata provided within each image. The class has an association to both the Control System class and the Image class.

Description of Attributes

The attributes of the card class include an integer value for the Employee ID card, for those Amtrek Members who are eligible for an employee ID. The card class also contains a string which contains the name of the Amtrak Member who is trying to login to the system. Each name has their own unique ID.

The Central Processing class's attributes are the two images that are taken in through the send and receive operations which will send the images to the testing facility to be tested against previous images for any changes.

The Image Comparison Class's attributes are the two images that are taken in through the compare image function, which are then compared against previous images for any unstable bluffs.

The Rating class's attributes include an image that is being rated due to the amount of change that has occurred, with an integer that assigned its rating. The rating is placed with a list of integers that will save these ratings for one whole month past the incident. The class also includes a string list that includes the Amtrak Contact list, in the case that the rating is a 4 or 5, as the system will automatically set the alarms and notify the nearby authorities.

The Control System class's attributes include the six cameras which will be continuously photographing the 300 foot stretch of bluffs which will take images that will include a 32-bit timestamp and a 32-bit geo location. The image will be included in the list of images, and the times will be recorded in the list of ints, with the location being noted in the list of strings.

The storage class's attributes will include the list of images as the class will keep track of the images and access one month's worth of hourly data plus one image per day for

images older than one month for up to one year.

The camera class's attributes will include a boolean value that will represent whether the wiFi is connected or not, as well as a boolean value that will indicate whether the night vision is activated. The camera class has a string value to detail the location of the image where the camera is recording, and an integer to display the time that the photo was taken.

The image class's attributes include a boolean value to detail whether the photograph is a gray scale image and another boolean value to indicate whether the photograph is a multicolor image. The class will include a string to detail the images location, an integer to note the time the image was taken, and a Camera object to be the physical object that takes the image. The class also includes a list of integers that can be used to create the image matrix.

Description of Operations

The control systems' class operations include get operations which involve having to call these different classes to access the information from these classes such as from the Image class and the Rating class. The purpose of this class is it will call and get information and data from the other classes such as calling from class image to retrieve data from the Image class and even the sendForProcessing method and receiveFromProcessing, it will use class Image which sends and retrieves Image data and information to and from the Control System. This class will get information to be used for the system itself such as getTime and getLocation to provide the current time being of the photos kept and taken. This class is mainly to obtain and use image information about the system to be able to use the information obtained which will use the parameters of Image and Camera1; this class also receives data types int and string.

The rating class operations include determining the rating which has the parameter of class Image, using the data type int which is calling the class to determine the scale by accessing the data from the Image class. This class also includes boolean data types which is created within the Rating class itself because it will use the Image class to determine whether or not the Authorities and/or the Amtrak itself should be alerted due to the scale score of the rating.

The card class operation briefly is an int data type which simply is created to input the ID information to determine access for the control system. This class also includes a notification system that uses the parameter of the employee ID to grant access and gain access to the control system itself.

The central processing class simply sends Image class information, of data type string, to send and process the image in this class from the called image class. The receiveRating and receiveImage methods both get information from the Rating class and the Image class to process the integer value for the rating and the image data from the image class.

The image comparison class is just a class created to take the image that is being observed at the time being and compare it with image 2 from the image class which in this situation, it would be the current image compared with the most recent image.

The storage class contains memory allocation as it will allow images from the camera class, then onto image class, then to storage to be stored. This class involves adding images from the image class and is stored on the image list and where we can delete images and find images from the Image class.

The camera class contains wifi connection and night vision which are both boolean functions because it checks whether or not the camera is connected to the wifi in terms of wifi connection and night vision will check if it is on. This class also contains get methods which include getLocation and getTime so it will also include the time and location of when and where the images will be taking place. The takeImage method will capture the image which could be sent to the image class.

The image class contains two boolean functions which consider the color of the images such as colored or gray scaled. This class will also include the location and time of when the image was captured and equivalent to the control system class and the camera class, getLocation is of data type string and getTime is of data type int.

Development Plan and Timeline

- Brainstorming software ideas for this computer system: ~ 1 week
- Create outline for software development plan: ~ 1 week
- Create software architecture and UML diagram with revisions and corrections as a team: 1-2 weeks
- Implement coding in a mutual language and develop the program together as a team of software developers: 2 months

- Set up the hardware needed in the unstable bluffs location in Del Mar, California with the camera system(s): 2-3 weeks
- Install and test the software with the hardware combined: ~ 2 weeks
- Leave room for debugging and technical difficulties especially for safety purposes and reliability of the correctness of the software: 6-12 months
- Release software for usage and continue maintenance

Tasks

Jessica's task: System Description, Software Architecture Overview, Description of Attributes, and overlooking UML class diagrams

Ashley's task: creating and developing UML class diagrams, Description of classes, overlooking entirety of project

Jared's task: Creating development plan and timeline, Description of operations, overlooking UML class diagrams

Verification Test Plan

Unit Test:

1. "checkWifiConnection(wifiConnection):bool"

The first function we are testing is from the Camera class, “checkWifiConnection(wifiConnection):bool” which checks if the wifi is connected as this is important to ensure that the camera is working properly as the camera is connected to the Wifi network.

Test Cases:

To test this function we need to test different inputs, such as the instance in which there is no connection, as we would expect the output of this boolean function to be ‘false’ indicating that the wiFi is not connected.

Test1: No connection

Input: wifiConnection = None

Expected Output: False

A second test instance we need to account for is when the wifi is connected and is working properly, as this function should then return the output of ‘true’ which indicates that the wifi is connected.

Test2: Connection

Input: wifiConnection = ‘connected’

Expected Output: True

A third instance we need to test is in the case that the wifi has disconnected, in which case this function should return the output of ‘false’ to indicate that the connection is not working properly.

Test3: Disconnected

Input: wifiConnection = ‘disconnected’

Expected Output = False

2. send(Image, Image):string

The second function we are testing is from the Central Processing class, “send(Image, Image):string” which takes in two images to be compared and will send them out to the testing facility. The output of this function will indicate whether the images were sent properly or not.

Test Cases:

The first instance we are accounting for is in the case where we send two valid images, as the expected output of this case would be some variation of “Images successfully sent for processing” to indicate that these images are valid types.

Test 1: Sending Valid Image Files

Input1: ‘bluff.jpg’

Input2: ‘bluff2.jpg’

Expected Output: send() returns ‘Images successfully sent for processing’

The second instance we are accounting for is the case in which the images attempting to be sent are of invalid types. The expected output of this case would be, “Error: Invalid image type” to indicate that the images were not sent as they belong to an invalid image type.

Test 2: Sending Invalid Image Files

Input1: ‘invalid.tx’

Input2: ‘ivalid.doc’

Expected Output: send() returns ‘Error: Invalid image type’

A third case we must test for is in the instance when two identical images are being attempting to be sent to the testing facility, as the function should return the message that “Error: Images are identical” to reveal that these images are identical and thus cannot be compared.

Test3: Identical images

Input1: ‘bluff.jpg’

Input2: ‘bluff.jpg’

Expected Output: send() returns ‘Error: Images are identical’

A fourth instance we must account for is the case in which the images are blank/not existent, as this function should output an error message that reveals, “Error: Image file not found” to indicate that the inputs were blank, and thus cannot be sent to the testing facility.

Test4: Images not Existent

Input1: ‘blank.jpg’

Input2: ‘blank.jpg’

Expected Output: send() returns ‘Error: Image file not found’

Integration Test

1. **Rate Bluff Based on Image Comparison**

To determine the rating of a bluff, we will use in the Rating class, the function “determineRating(Image):int”. Here, we will take an image which has been given as an input then return a number between 0 and 5 to show the rating in the proper circumstance.

Test Case 1: Image in color with unstable bluff of no identifiable change rated

Input: ‘bluff.jpg’

Expected Output: 0

Test Case 2: Image in color with unstable bluff of predictable slide rated

Input: ‘bluff.jpg’

Expected Output: 1, 2, or 3

Test Case 3: Image in color with unstable bluff of a significant change/ danger zone rated

Input: ‘bluff.jpg’

Expected Output: 4

Test Case 4: Image in color with unstable bluff of a major change/ evacuation should occur rated

Input: ‘bluff.jpg’

Expected Output: 5

Possible circumstances to account for include a missing image. If an image is not discovered, then an error should be thrown mentioning how a rating cannot be determined due to there not being an image given from the Central Processing class’ “recieveImage():Image” function.

Test Case 5: Image in color with unstable bluff of predictable slide rated

Input: (None)

Expected Output: “Error: Unable to determine rating, no image found”

Another circumstance that could have occurred is that a rating wasn’t attached in the metadata of the image from the Central Processing class’ “recieveRating():int” function. If so, then an error should occur saying that no rating was attached to the corresponding image.

Test Case 6: Image in color with unstable bluff of predictable slide rated

Input: (None)

Expected Output: “Error: Unable to determine rating, rating not found in image”

Lastly, another scenario could be the color of the image taken from the Image class’ “isGrayScale(grayScale):bool” and “isMultiColor(multiColor): bool” functions. If the image in it’s metadata is grayscale, then it’s rating should correspond no differently to its color counterpart.

Test Case 7: Image in grayscale with unstable bluff of no identifiable change rated

Input: ‘bluff.jpg’

Expected Output: 0

Test Case 8: Image in grayscale with unstable bluff of predictable slide rated

Input: ‘bluff.jpg’

Expected Output: 1, 2, or 3

Test Case 9: Image in grayscale with unstable bluff of a significant change/ danger zone rated

Input: ‘bluff.jpg’

Expected Output: 4

Test Case 10: Image in grayscale with unstable bluff of a major change/ evacuation should occur rated

Input: ‘bluff.jpg’

Expected Output: 5

2. **Storing an Image to the Database**

After an image is taken from the Camera class using “takeImage()” it is taken to the Control System class and uses “getImage(Camera1):Image” to capture an image based on its corresponding camera and finally, the Storage class’ function “addImage(Image)” will take in an image and it should be added to the imageList where essentially the database of all the pictures are held.

Test Case 1: Image in color with unstable bluff of no identifiable change

Input: ‘bluff.jpg’

Expected Output: (None)

One error that may occur along the way could be that the Control System class failed to connect to the a camera. Therefore, in the Control System class, “getImage(Camera1):Image”, should produce an error before getting to the Storage class confirming that there was no connection to the camera.

Test Case 2: Image in color with unstable bluff of no identifiable change

Input: (None)

Expected Output: “Error: Unable to find image to add to database, no camera connected”

System

- Input: the picture captured from camera

For this system test, it uses the camera to capture pictures every hour and it compares using the ImageComparison algorithm with the previous image taken. The path taken for this is we first start at the camera where the photo will be taken and then is processed as an image. From image, we will be sending the image directly to ImageComparison and also sending the image to storage. In terms of ImageComparison, the image will be sent to Central Processing where the two images will be compared and therefore be sent to the control system where it will be sent to the rating class where the rating class will determine whether the authorities and the amtrack will be alerted or not because in this case, we have no user to alert other than the main system itself because this system does not require a user to operate on its own due to the image comparison algorithm.

- Output: whether the system alerts user

- Input: the rating

For this system test, it is highly similar to the previous example with having the input be the picture. The path taken for this is starting off with the camera capturing the picture and the picture being sent to both the control system and the image class. From the image class perspective, it will be sent to both the storage class to store the image in the memory of this system and also directly sent to the ImageComparison class where the two images will be compared. From the ImageComparison class, the two images compared will be sent to the Central Processing class where the control system will receive the images processed and will be

sent to the rating class where similarly to the last example, this class will be determining whether authorities and/or amtrak will be alerted if the rating given is high enough to alarm the system. The boolean value will be sent back to the main control system and determine whether the alert is necessary or unnecessary.

- Output: sent back to the control system and alerts the system