Mountain Lion Detection Unit Software Design Specification

Prepared by:

Bradley Mustoe, Luke Files, Nathan Morales

October 8, 2024

1 Introduction and Overview

This device will detect mountain lions and other various animals in the surrounding area. It will create safety in our parks and recreation centers mitigating potential animal attacks. This will create a sense of comfort in knowing that you are in an area which is monitoring animal threats, thus creating peace of mind. In this document we will cover how the system is used, what features will be included and any supporting information regarding this technology.

2 User Requirements

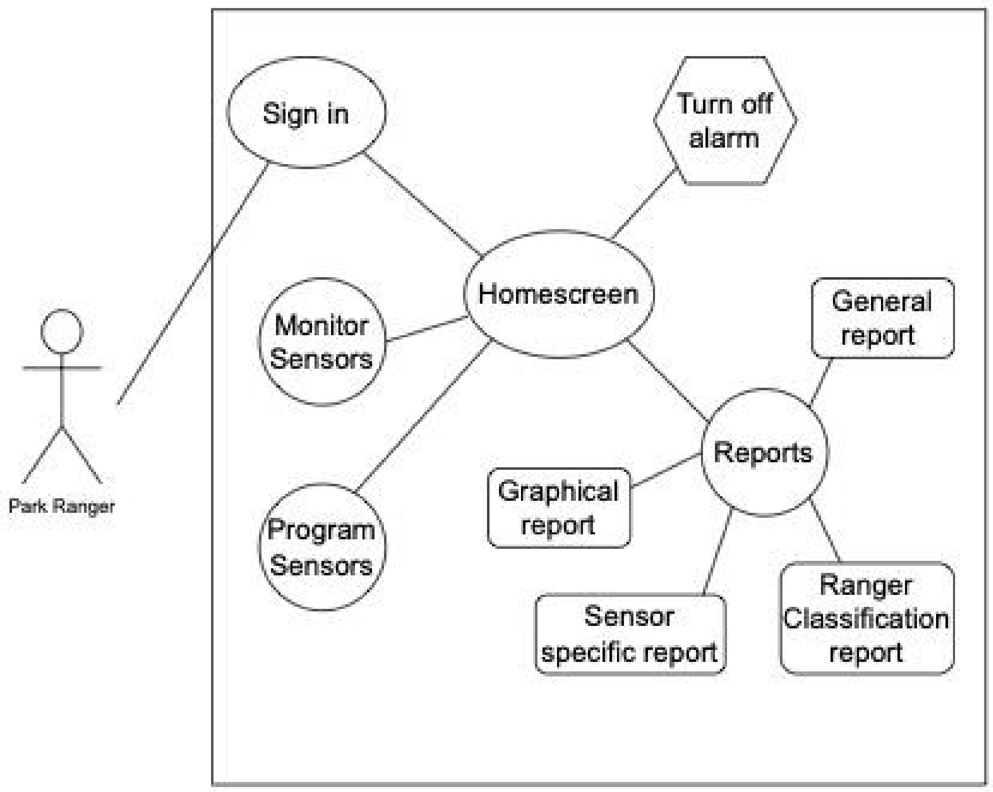
The ranger must place detection sensors around the park to cover the surrounding area. Only the park ranger will have access to control and monitor the detection controlling computer. It will be located in the park ranger station and will sound an alarm when a sensor is alerted. It will not stop on its own, the park ranger must turn off the alarm manually from the controlling computer. He will have access to request all of the various reports stored on the controlling computer. The ranger will be the main and/or only user tasked with placing sensors, turning off alarms, accessing reports, and monitoring sensors.

3 System Requirements

3.1 Functional Requirements

The system will detect noises based on threat levels. The three threat levels will be definite, suspected, or false. Every threat will be recorded and saved for up to 30 days. Past 30 days a summary will be saved up to a year. The ranger will be able to request reports by date and classification. These reports will be accessible from the controlling computer. There are multiple reports this system will store. A report showing all mountain lion detections by date and threat level classification. A report showing all mountain lion detections at a specific sensor location. A graphical report showing detections on a map of the park and areas within 2 miles of the park. Finally, A report showing detection classifications by the ranger. If a detection occurs, It will sound an alarm that will not stop until the park ranger turns off the alarm. Once the alarm is turned off it will not sound again until another separate noise is detected at a different location. The control computer in the ranger station will interact with the detection sensors which will be placed around our park or recreation centers. These sensors will have a 5 square mile radius. The sensors will determine the type of noise detected, the strength of the noise, and the location of the detected noise to within 3 meters. The sensors will be programmable for versatility.

Use Case Diagram



- The park ranger starts by signing into the controlling computer.
- Then from the home screen he has 4 options to go from there:
- Turn off alarm button
- Program sensors
- Reports
- Monitor sensors
- From the reports screen you're given the options of 4 different reports to chose from

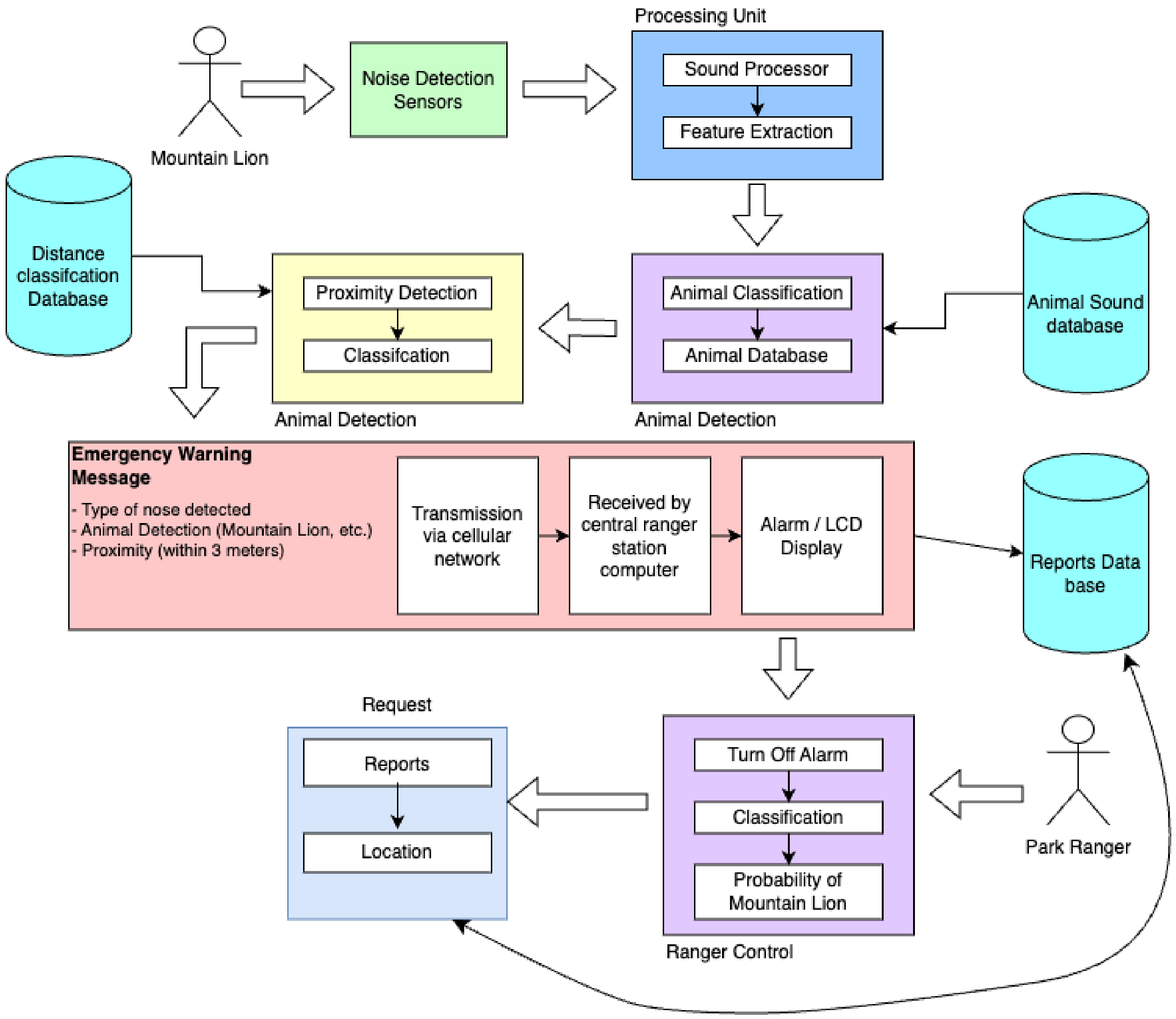
3.2 Non-functional Requirements

This system must be reliable to ensure the safety of the public. The sensors must be portable for easy access to place around parks or recreation centers where they're needed. They must be durable enough to withstand the weather conditions and surrounding environment. This includes the sensors being waterproof to withstand rain.

4 Other

We will be implementing this in San Diego but it must be a version capable of being duplicated for other parks. There may be a need for these in other areas and therefore must be adaptable. (Ex. Areas that snow etc.)

Software Architecture Diagram



1 Processing Unit

The processing unit is the first aspect of the software architecture that will be encountered when a mountain lion is detected. It will require the noise detection sensors to be connected to it so that it can detect nearby mountain lions. The processor will first contain a sound processor which will take the raw sounds that are received from the sensor and will process these to files that can then be used. The processing unit will then contain a feature extraction so that it can detect specific attributes of the sound that will then later be used.

2 Animal Detection

The next component of the software will be the Animal Detection unit. This feature will need to use the extracted features and sound files from the processing unit to categorize the animals. It will use the animal database to compare the different features of the sounds and correctly analyze the mountain lions. The second piece of the animal detection unit is the proximity detection. This will analyze the volume and direction of the sound to be able to accurately pinpoint the location of the mountain lion.

3 Emergency Warning Message

The emergency warning message is the warning that will pop up on the LCD screen with the information about the mountain lions. This message will be sent through a transmission via cellular network where it will then be received by the main server at the central park ranger station. It will then display this message on the screen, and will sound an alarm, only to be turned off by the park ranger. This message will contain a description of the type of noise detected, the species of the animal, and the location within 3 meters. The warning message will also connect directly with the main database, as well as with the report requests unit.

4 Ranger Control

The ranger control feature is the next aspect of the program and will be used primarily on the main ranger computer. It will connect directly with the error message previously received, where the ranger can then turn off the alarm that is going off. The ranger can then confirm the previous classifications of the type of mountain lion, where the report will then be sent to the report requests unit.

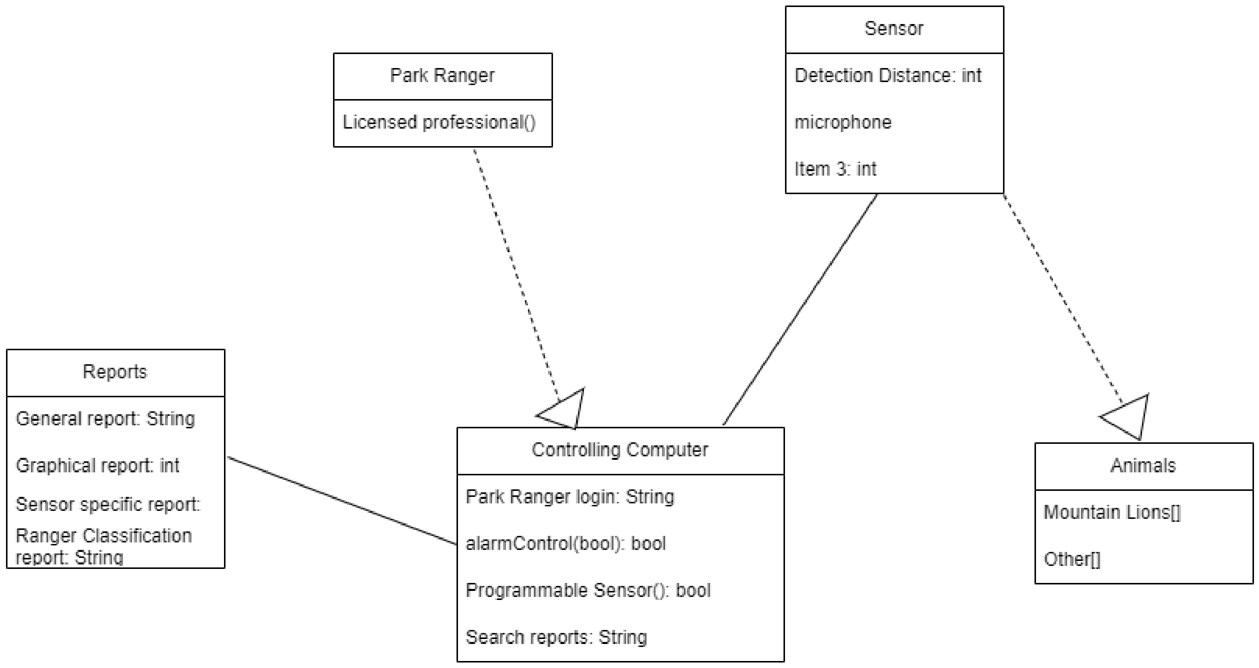
5 Report Requests

The report Requests tab processes the reports of the location of the mountain lion as well as the location and compresses this information. After compilation, it will then link directly to the database and store the information here.

6 Databases

Our software requires three separate databases that will be used to hold all of our information that will be used in the software. The first of the databases in our system is an animal database that stores all animals we have recorded. It is useful because we can compare new sounds to classify the animals. The second database that we have implemented is the distance classification that holds a variety of sounds from various distances so we can cross-reference the recordings we have and verify the accuracy of the predictions of locations. Our final database is used to store all of our reports and new recordings. This database will contain all reports that are compiled with all necessary information, including animal type, threat level, and location.

<u>UML Class Diagram</u>



This UML class diagram will describe the organizations of each class in the program. To begin with, the Park Ranger class uses the Controlling Computer class to log in. This class includes an Alarm Control function, Programmable Sensors, Search Reports, and a login. This class will have access to the Sensor class which includes the detection distance as an int from the placed sensors, a microphone, and an animal item classification from the Animals class.

The Alarm Control function takes a bool parameter from the Programmable Sensor function in order to trigger off the alarm. The class also has access to the Reports class which includes General Report, Graphical Report, Sensor Specific Report, and a Ranger Classification Report. A string of search reports will be generated in which the ranger will be able to view and analyze them. Lastly in order for the Sensor class to function it has to use the Animals class that has an array of mountain lions and other wild animals storing it in the Sensor class.

Development Plan & Timeline

For the development of this product our team will be working for the next 5-6 months. This product is needed by summer 2025 giving us ample time to test and implement this product with extra time to account for unexpected delays and necessary updates.

Planning phase (2-3 weeks)

In the first month we will be on site meeting with park rangers to confirm necessary requirements functional and non functional. During this phase we will be identifying sensor technologies suitable for our functions and decide on the communication protocol between the sensors and the central control system. We will also be creating a budget analysis for hardware costs. This will be carried out by our project management team, engineering team, and finance team.

Prototype phase (4-5 weeks)

The following month will be our prototyping design phase. In this month we will create great strides towards the final version of the product where we design the sensors will the necessary capabilities of covering a 5 square mile radius and locating the detection within a 3 meter accuracy. This will also include the designing of the controlling center user interface in which the implementation of a manually controlled alarm system will be applied. Amongst the initial creation of our prototyped product this period will also include some initial testing with simulated environments for noise detection. This will be carried out by our hardware/software development teams alongside our testing team.

Hardware and software phase (8-9 weeks)

After our prototyping phase the next two months will be designated for hardware and software integration. This will include the full completion of sensor programming to detect the three threat levels along with implementing a graphical component on the controlling computer to show the detection location. We will design and integrate report generation tools so the ranger can request the various reports and ensure the data is stored for up to 30 days with summaries of the information up to a year. This phase will also include the enabling of remote updates to the sensors and ensure seamless communication between the sensors and controlling computer. This phase will be handled by our hardware and software development teams.

Testing phase (3-4 weeks)

Nearing the finished product this next month will be the testing a calibration phase. We will be testing the product in parks and simulation noises to ensure accuracy and desired outcomes. The system will be calibrated for accurate location detection and threat levels. This phase will also include the user testing with park rangers on site. We will be teaching users how everything works, how to place and monitor sensors, generate reports, and turn off the alarm. We will also be collecting feedback on any usability issues or system improvements. Our testing and calibration teams will work in congruence to finish this phase.

Finalization phase (3-4 weeks)

Our last month will include the actual deployment of the product for the real world. The user manuals will be finalized during this process while we address any last minute improvements discovered during our previous testing phase. Our team will provide continued on site support during the initial launch of the product. This final phase will be initiated by our deployment team with the secondary help of our customer success team.

Post launch

Given that our product will be ready well ahead of required schedule will allow us any time for necessary updates and improvements realized throughout the finalization phase before summer of 2025. Our team is committed to the ongoing support in monitoring the system performance while providing regular software updates. All teams on the project will be responsible for the continued success of this product.

1. Unit Testing

For our unit tests we will be testing the most basic and important functions of our overall system. We will be testing the functionality of our sensors. The two main focal points for our sensor will be to test the detection of audio as a whole and pinpointing the location of the detected noise.

I. Test audio detection by sensors

The test procedure will incorporate an example environment to simulate real world use. We will place the sensors in an open area, connected to power, and the processing unit. We will play various audio clips including our main target of mountain lions alongside other animals and background noises. The mountain lion sounds will be tested at different audio levels. A variety of background noises and other animals will also be tested.

Expected outcome:
Success: The sensors accurately detect mountain lion sounds with threat levels.
It ignores irrelevant sounds, and returns a high accuracy score.
Failure: The sensor fails to detect mountain lions or incorrectly alarms irrelevant audio

Pseudo Code

```
TestAudioDetection():
    sensor.initialize()
    play_audio(mountain_lion_roar)

    if sensor.detected_animal == "mountain lion":
        print("Test Passed: Mountain lion detected")
    else:
        print("Test Failed: Detection error")

    play_audio(background_noise)
    if sensor.detected_animal == None:
        print("Test Passed: No false detection")
    else:
        print("Test Failed: False positive detection")
```

II. Test location detection accuracy

During this test we will have the same setup with our previous test of an open area where we will simulate a variety of noises, specifically mountain lion noises. We will be testing the systems ability to accurately detect the location within the required 3 meter radius of the source.

Expected outcome:

Success: The system accurately detects the location of the noise within 3 meters of the actual source.

Failure: The system reports a detection more than the 3 meter radius indicating an issue with our detection algorithm.

Pseudo Code

```
TestLocationDetectionAccuracy():
    sensor.initialize()
    known_location = (50, 30)
    simulate_noise_at_location(known_location)
    detected_location = sensor.detect_location()

    if calculate_distance(known_location, detected_location) <= 3:
        print("Test Passed: Location detected accurately within 3 meters")
    else:
        print("Test Failed: Location detection exceeds 3-meter accuracy")

    edge_location = (5000, 5000)
    simulate_noise_at_location(edge_location)
    detected_edge_location = sensor.detect_location()

    if calculate_distance(edge_location, detected_edge_location) <= 3:
        print("Test Passed: Edge location detected accurately within 3 meters")
    else:
        print("Test Failed: Edge location detection exceeds 3-meter accuracy")

    outside_location = (7000, 7000)
    detected_outside_location = sensor.detect_location()

    if detected_outside_location == None:
        print("Test Passed: No location detected for out-of-range noise")
        else:
                print("Test Failed: False detection for out-of-range noise")
```

3. Functional Testing

I. Testing the Alarm Activation and Deactivation

The goal of this test is to ensure that the alarm will activate and deactivate with the detection of a mountain lion. To test this, we will provide a real mountain lion sound that has already been captured, as well as a nature audio sample that does not contain a mountain lion. Upon providing these sounds, we will have a boolean be turned to TRUE if the alarm is playing, and FALSE if it is not. If it is false, the system will automatically fail. If it is true, the system will continue and test whether the alarm can be turned off by manually turning off the alarm, then rechecking the boolean.

Expected Outcome:
From this test, we expect for the alarm to sound when the mountain lion sound is fed to the machine, and to not play the alarm when a sound is provided that does not contain a mountain lion. The combination of these two inputs should provide an accurate understanding of whether the alarm will sound for a mountain lion.

Pseudocode

```
mountainLionSound = new Sound("mountain_lion")
sensor.detectSound(mountainLion)

boolean alarmState = rangerStation.checkAlarm()
if alarmState = true:
        print "alarm was triggered successfully"

        ranger.turnOffAlarm()
        alarmState = rangerStation.checkAlarm()
        if alarmState = true:
                print "alarm turned off successfully"
        else:
                print "alarm did not turn off: FAIL"
else:
        print "alarm did not sound: FAIL"
```

```
fakeSound = new Sound(“not_mountain_lion”)
sensor.detectSound(fakeSound)

boolean secondAlarmState = rangerStation.checkAlarm()
if secondAlarmState = true:

    print “alarm was triggered: FAIL”
else:
    print “alarm did not sound succesfully”
```

II. Testing the Report Generation

The goal of our report generation is to be able to retrieve reports of the mountain lions with the dates, location, and threat level classification. In order to test this, we would provide simulated mountain lion sounds to the sensors. We would provide several of different audios, allowing for several testing of whether the locations are valid, the sounds are accurate, and the threat is accurate.

Expected Outcome:

From this test, we expect the report to contain all accurate information gathered from the fake sounds that are provided.

Pseudocode

```
for i from 1 to 100:
    sound = simulateSound(randomMountainLion())
    sensor.detectSound(sound)
    Boolean nextDay = randomBoolean()
    if nextDay == true:
        currentDay += 1

report = ranger.requestReport(“mountain_lion”, lastDays = 30)

if report.animalType == sensor.getAnimals()
    print “report accurately compiled animals”
else:
    print “report did not compile animals”
if report.animalDate == sensor.getAnimalDate()
    print “report accurately logged the date the animals were sensed.”
else:
    print “report did not compile the date the animals were logged”
if report.animalType == sensor.getAnimalRange()
    print “report accurately compiled animal locations”
```

```
else:
        print “report did not compile animal locations”
if report.animalType == sensor.getAnimalThreat()
        print “report accurately compiled animal threat levels”
Else:
        print “report did not compile animal threat levels”
```

2. System Testing

To test the functionality of our Mountain Lion detection system we will ensure that our requirements are met in order to confirm that all aspects of the system behave accordingly and produce the expected outcomes.

I. Testing an animal walking by the Sensor:

A simulated animal would be walking through within the range of the sensor, ensuring that even the lowest amount of noise volume be detected. It would be recorded and sent back to the system where various test levels of analysis and classification would be performed.

Expected Outcome:

Data would be sent to the system in which the sound would be analyzed to determine its location and threat level which in addition would be sent to the database for later use. The alarm will go off immediately at the location of the threat level if it's too high and continue sounding until the park ranger receives a notification of it and manually turns it off.

II. Testing the Report Compilation:

A report would be generated by retrieving the recorded data and animal classification from the database in which the park ranger will be able to access and view all the information present. The ranger would be able to navigate through each report classified either as date or threat level to begin analysis and make conclusions.

Expected Outcome:

Reports should be generated accurately with as much details as possible and the park ranger can easily navigate and view them as a pdf file.

Data Management Strategy:

We added specific databases to our software architecture diagram to include the different information we will be covering with our system. Starting with only one database we have since improved our design to include three databases for quick data processing and retrieval. Our three databases will be a distance classification database, animal sounds database and a reports database. This updated design will help ensure scalability, efficiency and organized storage of critical information. Through implementing a structured, multi-database approach, we are improving the efficiency of the system, reducing response times, and ensuring data integrity.

Design Choices:

We chose to use SQL for our system because by using three databases we want to ensure key elements in the functioning of our system remain perfect. We want our data inside each database to be firmly structural and consistent, having it be reliant as when specific classes need to retrieve different types of data or parameters, such as generating reports using the Report Database knowing that retrieval would be reliant as other types of data won't be included in there, but instead on other databases. We also want our system functioning at its intended pace by having our query remains as fast as possible when requesting data in order to have our alarm system be triggered depending on its distance by using the Distance Classification Database. Data across each database would be strongly secured as we can't allow any modifications from unauthorized access that could potentially harm or have the system not function as properly as databases won't remain consistent and reliable anymore. Lastly if more sensors at different locations want to be added our databases would easily be allowed to expand horizontally.