S.A.R.A

Search And Rescue Assistant

Software Engineering | Group 8

Date: May 5, 2019

https://abhiek187.github.io/emergency-response-drone/

TEAM MEMBERS

Sahana Asokan
Won Seok Chang
Avnish Patel
Abhishek Chaudhuri
Shantanu Ghosh
Srikrishnaraja Mahadas
Sri Sai Krishna Tottempudi
Vishal Venkateswaran

Individual Contributions Breakdown

All team members contributed equally.

Table of Contents

Summary of Changes

- Updated the contributions breakdown matrix to indicate equality
- Reorganized references and labeled points of interest with numbers
- Added low battery alert to REQ4
- Changed descriptions for REQ5 and REQ6
- Added more detail about the drone and controller connection in REQ6
- Clarified useability in nonfunctional requirements
- Removed recoverability from nonfunctional requirements
- Made Drone a participating actor instead of an initiating actor
- Elaborated more on UC-6 and UC-7’s casual description
- Indicated that UC-5 and UC-8 could be considered for future work
- Updated the use case diagram
- Clarified more about the descriptions for UC-1.
- Altered main success scenario for UC-6
- Removed fully-dressed description for UC-8
- Recalculated the Effort Estimation section
- Calculated duration in effort estimation
- Changed description of responsibilities, associations, and attributes
- Merged Calibrator onto Controller
- Added association between the Controller and the Interface
- Removed attributes for Controller and Notifier
- Changed domain model accordingly
- Added the description for the traceability matrix of the domain model
- Removed system operation contract for ReturnToHome
- Removed mathematical model for AvoidObstacles and added information about GetData
- Removed interaction diagram for UC-8
- Updated the Class Diagram
- Updated the Data types and Operations Signatures subsection
- Added the design patterns for the class diagram
- Added the OCL contracts

- Changed package diagram
- Updated the Hardware Requirements with all additional hardware used and any hardware removed/changed.
- Tweaked the Mapped Hardware to Subsystems section to account for the Hardware Requirements section.
- Tweaked the Algorithm section to specify the method used to calculate speed.
- Removed Data Structures section
- Updated the Algorithms Section with the algorithms that were used
- Added old and new website design and explained why we changed the interface
- Added more detail to the GetData/GetStatus test
- Updated Project coordination and Progress report, so that it reflects recent work.

Customer Problem Statement

Customer Statement of Requirements

Search and rescue operations can often involve first responders and volunteers trying to cover a vast area in as little time as possible to save the most lives. These operations can be categorized by the environment take place in. They can further be categorized by the specific type of operation that needs to take place, such as in urban areas or in remote mountainous regions. The circumstances that could merit such operations could involve natural disasters such as earthquakes and hurricanes. Regardless of the type of operation, technology is being increasingly used to streamline the efforts of first responders and volunteers in their efforts to try to save as many people as possible. There have been many search and rescue missions in the past.[1] Many of these missions involved the use of large amounts of people and resources. Even with all the effort put by the people involved many lives were lost in the process. One such organization that is involved in search and rescue operations is the Coast Guard. The table below illustrates the statistics of these operations conducted by the Coast Guard from 2011 through 2015.[2]

Fiscal Year	Cases	Responses	Sorties	Lives Saved	Lives Lost	Unaccounted Lives
2011	20,512	43,954	21,566	3,793	735	392
2012	19,787	43,940	21,609	4,037	713	440
2013	17,803	38,272	19,420	3,753	651	252
2014	17,508	38,282	19,032	3,443	595	308
2015	16,456	37,215	18,781	3,536	603	330

The sheer number of cases and responses conducted by the Coast Guard shows how big of an issue search and rescue missions are in the United States. The table also emphasizes that concept that these operations are not always successful or efficient. This is based on the number of lives lost and the number of people not accounted for along with a high number of responses for the cases. Our method looks into a possible alternative approach to these search and rescue missions.

The Search and Rescue Assistant, or S.A.R.A., will modernize the techniques employed by first responders on search and rescue operations. A drone can cover more distance than a single person is able to. Currently, the most frequently used techniques to cover a lot of areas very quickly is to either use a helicopter or to use a lot of people. The problem with helicopters is that they usually have to fly in from somewhere else and that can take time. Another problem with the use of helicopters is its lack of ability to search in narrow or tight areas. The issue with using a lot of volunteers is that people end up risking their own lives to find survivors. Often times these search parties tend to be time-consuming and depending on the circumstances, unorganized.[3]

Ready-to-launch drones can be set up in minutes, which will save time. By attaching a phone camera to the drone, the user will able to see the video feed that the drone is transmitting. Doing so will reduce the risk of potentially sending people in harm's way to get the most accurate information about where people might be trapped. Additionally, this would also be cost-efficient. This would reduce actual labor since we would mainly be investing in developing an efficient algorithm, and the device. This algorithm would take one initial investment and would be developed for improvement. Due to the cost effectiveness of the device and the reusability of the algorithms involved, if the resources were to be available, it should be rather simple to manufacture multiple devices.

Naturally, an important aspect of an aerial vehicle of this nature is whether or not it can survive the challenges/harsh conditions it can face while in the field. To bolster S.A.R.A's ability to withstand these conditions, it will be able to avoid obstructions in its' path, in part due to the implementation of ultrasonic sensors. Using these sensors, and the usage of the primary camera, the user can easily maneuver through different obstacles that he/she may encounter during a search and rescue mission. To assist the end-user in knowing the immediate environment, a thermal imaging attachment will be mounted to the mobile phone that serves as the drone's primary camera. Image processing will not occur on the drone itself, but instead on a centralized hub located back on an emergency vehicle, which receives relayed images/video real-time so that emergency responders can quickly determine the best course of action. The sensors/equipment necessary to accomplish this will be either be purchased or obtained by the team members from existing laboratories/organizations.

Regarding the working environment, S.A.R.A. will have to be able to maneuver in potentially tight/enclosed spaces. In such an environment, being able to receive data on how close an object is to the drone is a specialized function ultrasonic sensors can provide. The drone can then properly take a course of action based on the proximity data it receives, such as change the amount of thrust in a particular direction or instead start pushing in an entirely different direction. With regards to processing visual data, the S.U.R.F. identification algorithm can be used to accurately determine an image’s correlation/accuracy to a specific desired target object. In this case, the target would be the human faces/heat signatures.

Even with many solutions to search and rescue operations, S.A.R.A. offers a new take on optimizing the field. One of the key priorities of search and rescue missions includes safety, not just for the missing people, but for the people involved in the rescue operation. This approach makes it easy for even a single person to actively investigate the search and rescue operation in a safe manner. There would be more focus on the actual goal of the mission instead of also worrying about the safety of the people working the rescue/search missions.

Glossary of Terms

Database - Server that will keep data of the drone and pictures from the drone camera.

UI - A physical program that allows the user to see the environment from the camera, information of the drone speed and health, and distance away from the objects.

Controller - A device that will allow the user to control the drone movements and avoid any obstacles.

Proximity Alert - Internal mechanism that will use proximity sensors to see if the drone is getting dangerously close to any obstacles in the flight path.

Wireless Connection - The connection between the drone, controller, and database that allows the user to stay in control of the drone.

Drone Sensors - Devices that allow the drones to detect its speed, distance from user, stability, to detect obstacles, the drone's health, etc. Examples include an IR/Thermal sensor, Accelerometer, and Gyroscope.

S.U.R.F. - Speeded Up Robust Features, an algorithm that finds key points of an image using Hessian Matrices and scaled space, making it simpler to compare different images and see if they correlate appropriately.

System Requirements

Functional Requirements

REQ1 - Database/Server
REQ2 - UI Screen
REQ3 - Controller
REQ4 - viewDroneCondition
REQ5 - Proximity Alert
REQ6 - Wireless Connection
REQ7 - GPS tracking
REQ8 - Infrared Sensor

Requirement	Priority	Description
REQ1	5	Data server that will store the information from the drone and allow the user to access it
REQ2	2	The user interface will allow the user to see the drones footage and any other relevant information
REQ3	1	The user should be able to control the drone's movements
REQ4	4	The phone mounted on the drone will send a signal to the controller to notify of its operating status and alert the user if the battery is low.
REQ5	1	The drone should be able to correctly identify any close obstacles.
REQ6	2	A connection between the drone and controller is established via remote control.
REQ7	2	The user will be able to know exactly where the drone is.
REQ8	3	This sensor will allow the user to detect heat signatures through any material walls

Nonfunctional Requirements

Usability - User will figure out the user-friendly interface for viewing the drone by labeling buttons, displaying drone data, and requiring very little taps on the screen.

Security - User will be able to use the interface without having to jeopardize his/her safety by using the drone from a reasonable distance.

Accessibility - The user will be able to run the software to operate the drone, on any smartphone regardless of the OS on the device.

Efficiency - User will be able to use the software with any accompanying hardware through a wireless connection.

User Interface Requirements

Requirement	Priority	Description
REQ1	1	The controller for the drone will have a live feed of what the camera is seeing
REQ2	3	It will also display various properties of the drone. Some properties include the speed of motors, drone battery level, and current location
REQ3	2	Proximity alerts will be sent to the controller so the user knows which direction to avoid
REQ4	4	The operating status of the drone will be sent to the controller so the user will know if they have to pull the drone back in case of low battery level.

Image 1[4]

Functional Requirements Specification

Stakeholders

Stakeholders

- Licensed User
- First Responders
    - Police Officers
    - Authorized Volunteers
    - Firefighters
    - EMT’s
    - Emergency Dispatchers

Actors and Goals

Actor	Type	Actor's Goal	Use Case Name
User	Initiating	To control the drone.	MoveDrone (UC-1)
User	Initiating	To view a live video feed of the drone.	ViewCamera (UC-2)
User	Initiating	To get the drone's current location.	GetLocation (UC-3)
User	Initiating	To get the drone's operating status.	GetStatus (UC-6)
First Responder	Initiating	To identify the emergency from the drone.	ViewCamera (UC-2), GetStatus (UC-6)
Drone	Participating	To move based on user input.	MoveDrone (UC-1)
Sensors	Participating	To locate nearby objects.	CheckObstacles (UC-4), AvoidObstacles (UC-5)
GPS	Participating	To track the current location of the drone.	GetLocation (UC-3)
Server	Participating	To store all of the data that the drone has obtained.	getData (UC-7), GetStatus (UC-6)

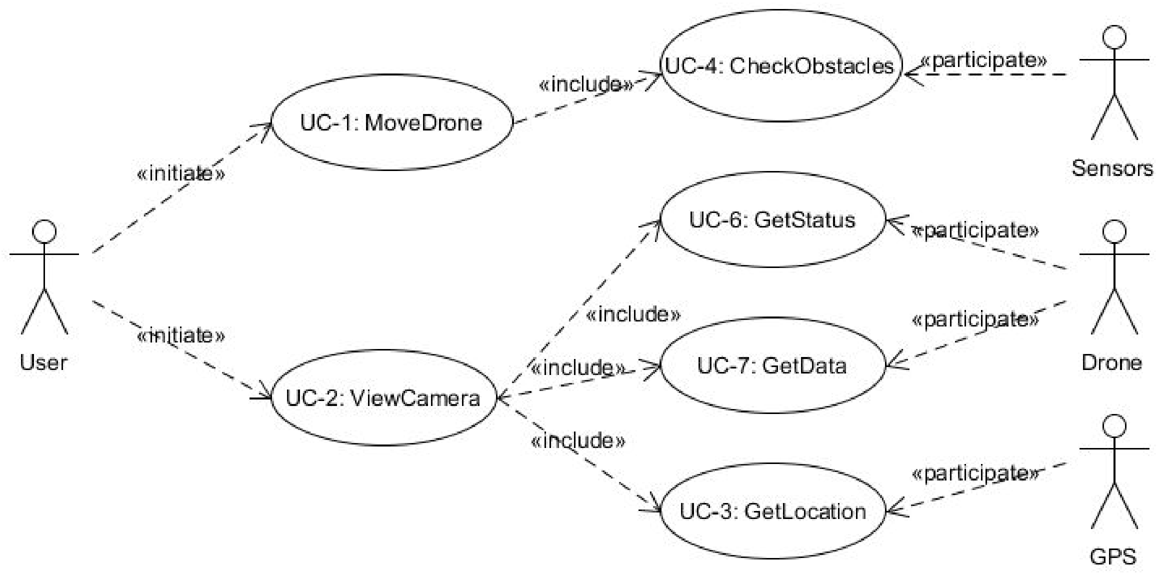
Use Cases

Casual Description

Use Case Name	Description	Requirements
MoveDrone (UC-1)	The user can move the drone using the controller.	REQ3, REQ6
ViewCamera (UC-2)	The user can view a video of the drone via the phone's camera.	REQ2
GetLocation (UC-3)	The user can detect the drone's location using GPS.	REQ2, REQ7
CheckObstacles (UC-4)	The drone can detect obstacles in its path.	REQ5, REQ8
AvoidObstacles (UC-5)*	The drone can avoid obstacles based on its surroundings.	REQ5, REQ8
GetStatus (UC-6)	The user or a first responder can check the current state of the drone, such as its power level or the phone's battery life.	REQ1, REQ2, REQ4
GetData (UC-7)	The user can check all of the data that the drone is transmitting through the sensors.	REQ1, REQ6
ReturnToHome (UC-8)*	The drone can safely autopilot back to the home (controller) in case the connection is lost. The user will know the drone's last location until it gets back.	REQ5, REQ7, REQ8

*This can be considered for future work.

Use Case Diagram



Traceability Matrix

Requirements	Priority	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8
REQ1	5						x	x	
REQ2	2		x	x			x		
REQ3	3	x							
REQ4	4						x		
REQ5	1				x	x			x
REQ6	2	x						x	
REQ7	2			x					x
REQ8	3				x	x			x
Total Priority	-	5	2	4	4	4	11	7	6

The traceability matrix above shows the relationship between the use cases and the functional requirements. It also ranks each of the use cases based on which use cases we believe have a higher priority. The matrix also ranks the priority of the requirements by what we think are the most important features for the drone to have. The way the matrix works is that each requirement has a set priority and if a use case incorporates a requirement; the priority points of the requirement are added to the use case. The priority points of each use case are the sum of the requirements for that use case. For example, use case 6 has a priority of 11 which comes from REQ1 (5), REQ2 (2), REQ4 (4). If the there values of the requirements are added together, you get the priority of use case 6 which is 11.

Fully-Dressed Description

<u>Use Case 6:</u>	GetStatus
Related Requirements:	REQ1, REQ2, REQ4
Initiating Actor:	Drone
Goal:	To get the drone's operating status
Participating Actor:	Server
Preconditions:	A signal between the drone and the controller is available
Postconditions:	Allows the user to know if the drone is active or not.

Main Success Scenario:

1. The user will know how much power the drone is using.
2. The user can see the phone's battery level and is alerted whenever it drops below 20%.

<u>Use Case 7:</u>	GetData
Related Requirements:	REQ1, REQ6
Initiating Actor:	User

Goal: Collect data on the various operations of the drone

Participating Actor: Server

Preconditions: Drone is on and a connection between the drone and controller is established.

Postconditions: Allows the user to manipulate and store that data.

Main Success Scenario:

1. The user can adjust motors speeds based on collected data.
2. The user uses the controller to move the drone if needed based on altitude.

<u>**Use Case 1:**</u> MoveDrone

Related Requirements: REQ3, REQ6

Initiating Actor: User

Goal: Ability to move the drone using a controller

Participating Actor: Drone, Controller

Preconditions: Drone is available
Controller is Available

Postconditions: Allows the user to maneuver the drone using a controller and a camera

Main Success Scenario:

1) The user sets the drone on the field.
2) The user uses the controller to test the drone's ability to move.
3) The controller will send signals to the drone which will allow the user to control and move it.

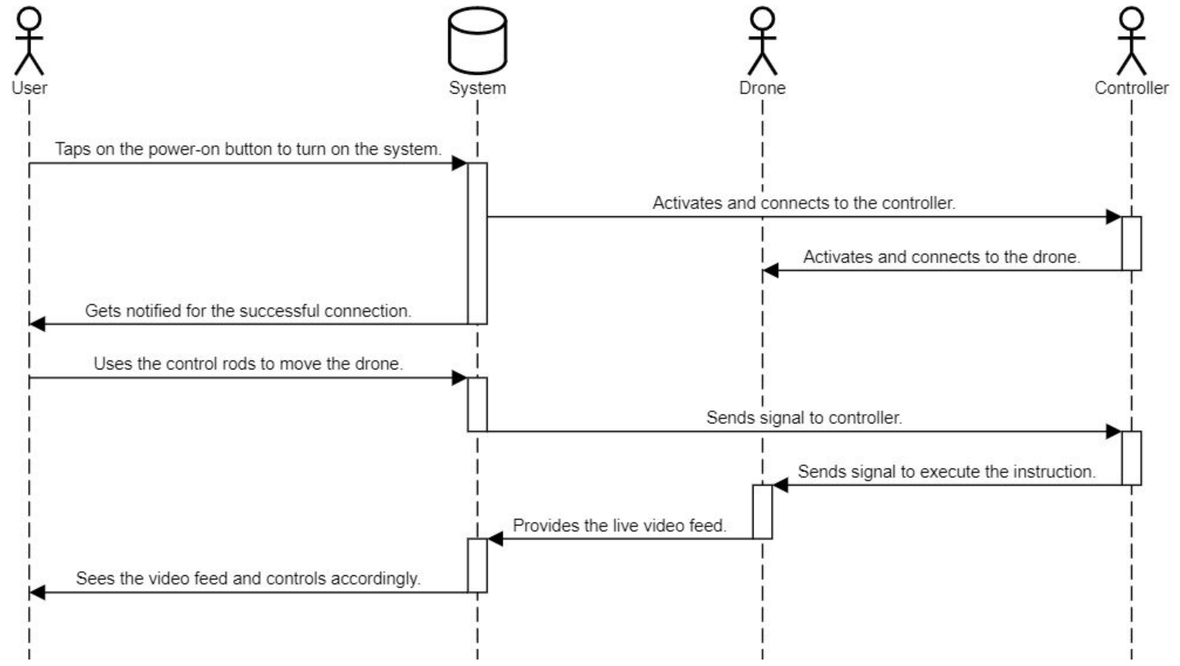
<u>**Use Case 3:**</u>	Get Location
Related Requirements:	REQ2, REQ7
Initiating Actor:	User
Goal:	Ability to detect the current location of drone
Participating Actor:	Drone, Controller
Preconditions:	The GPS is on and in a working condition. The connection between the drone and the controller is stable.
Postconditions:	Allows the user to retrieve the current location of drone displayed on the controller.

Main Success Scenario: 1)The controller receives the GPS signal from the drone.

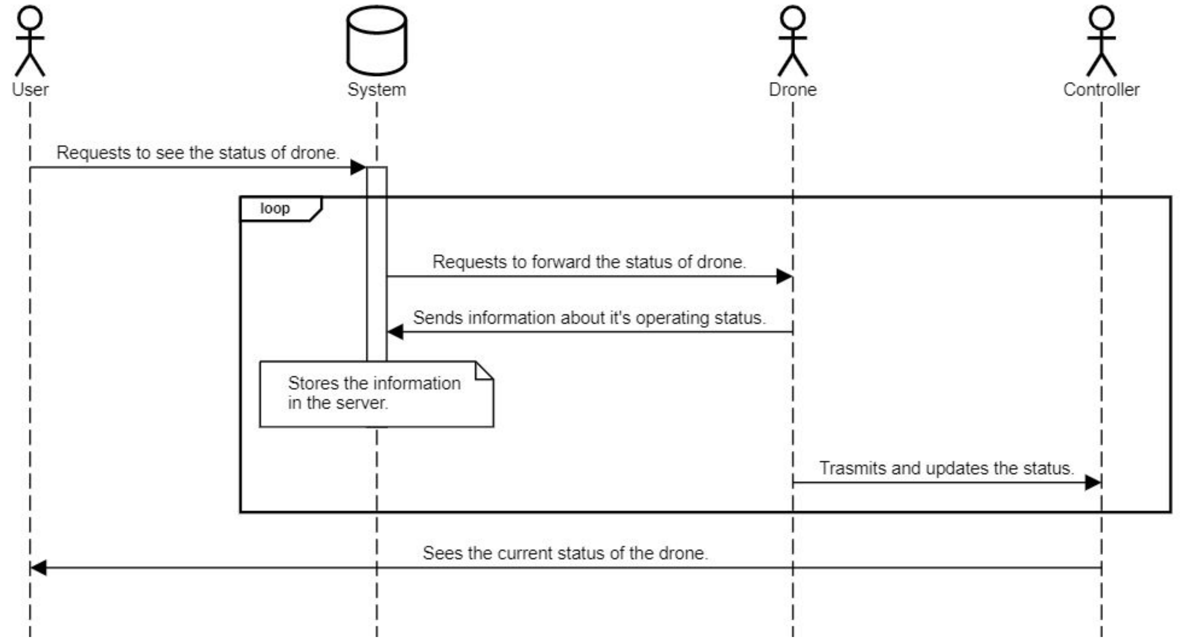
	2) The user can see the current location of the drone.
<u>**Use Case 4:**</u>	CheckObstacles
Related Requirements:	REQ5, REQ8
Initiating Actor:	Drone
Goal:	To enable drone to detect obstacles in its path.
Participating Actor:	Sensors
Preconditions:	The sensors are on and in a working condition. The physical mechanism of the drone is undamaged and operable.
Postconditions:	Allows the drone to detect obstacles that can possibly damage or interrupt its mission.
Main Success Scenario:	1)The sensors built on drone detect . 2) It alerts the user, thus the user can maneuver the drone.

System Sequence Diagrams

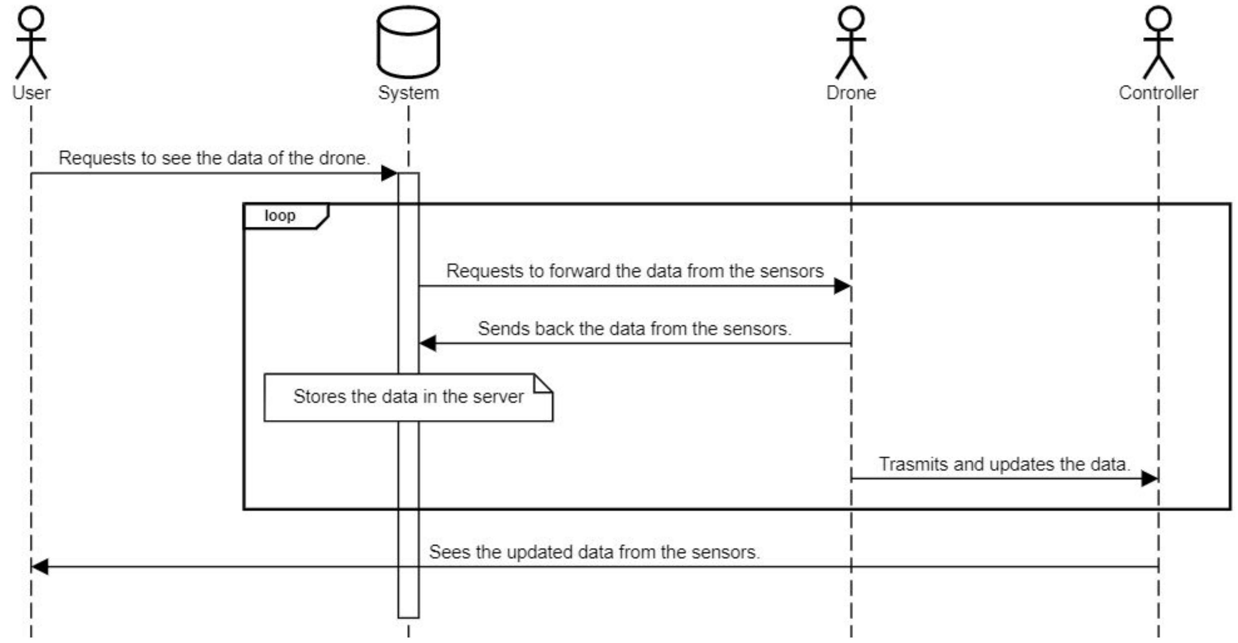
Use Case 1: MoveDrone


User
System
Drone
Controller
Taps on the power-on button to turn on the system.
Activates and connects to the controller.
Activates and connects to the drone.
Gets notified for the successful connection.
Uses the control rods to move the drone.
Sends signal to controller.
Sends signal to execute the instruction.
Provides the live video feed.
Sees the video feed and controls accordingly.

Use Case 6: GetStatus


User
System
Drone
Controller
Requests to see the status of drone.
loop
Requests to forward the status of drone.
Sends information about it's operating status.
Stores the information in the server.
Trasmits and updates the status.
Sees the current status of the drone.

Use Case 7: GetData


User
System
Drone
Controller
Requests to see the data of the drone.
loop
Requests to forward the data from the sensors
Sends back the data from the sensors.
Stores the data in the server
Trasmits and updates the data.
Sees the updated data from the sensors.

Effort Estimation using Use Case Points

	Use case Points
UUCP=UUCW+UAW	108
TCF	.955
UCP = UUCP × TCF	103.14

UAW
Simple=1
Average =2
Complex = 3

Actor Name	Description of Relevant characteristics	Complexity	Weight
User	To control the drone	Complex	3
User	To view a live video feed of the drone	Complex	3
User	To get the drone's current location	Simple	1
Drone	To check for and avoid obstacles	Complex	3
User	To get the drone's operating status	Simple	1
First Responder	To identify the emergency from the drone	Simple	1

Sensors	To locate nearby objects	Average	2
GPS	To track the current location of the drone	Average	2
Server	To store all of the data that the drone has obtained	Average	2

UAW(home access) = _3 * Simple + 3_ * Average + 3_ * Complex = 18

UUCW
Simple=5
Average =10
Complex = 15

Use Case	Description	Category	Weight
MoveDrone (UC-1)	The user can move the drone using the controller.	complex	15
ViewCamera (UC-2)	The user can view a video of the drone.	complex	15
GetLocation (UC-3)	The user can detect the drone's location using GPS.	average	10
CheckObstacles (UC-4)	The drone can detect obstacles in its path.	complex	15
AvoidObstacles (UC-5)	The drone can avoid obstacles based on its surroundings.	complex	15
GetStatus (UC-6)	The user or a first responder can check the current	simple	5

	state of the drone based on the emergency.		
GetData (UC-7)	The user can check all of the data that the drone is transmitting.	complex	15

UUCW = 1_ *Simple + _1 * Average + _5 *Complex = 1x5+1x10+5x15= 90

TCF

Technical Factor	Description	Weight	Complexity	calculations
T1	Distributed web-based System	2	5	10
T2	Performance objectives	2	3	6
T3	End-user efficiency	2	4	8
T4	Reusable code and design	1	2	2
T5	Easy to use	0.5	1	.5
T6	Moderately difficult to change	1	3	3
T7	Range of operation	1	3	3
T8	Signal Strength	1	3	3

TCF=C1+C2x Technical Factor Total=35.5

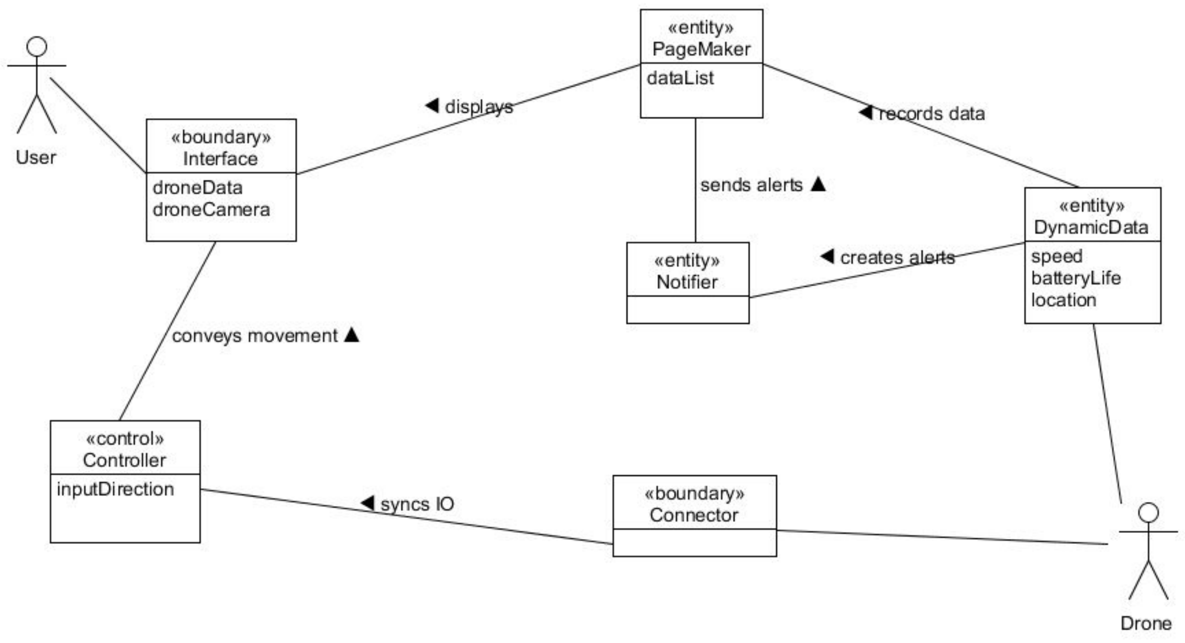
$$C_1 + C_2 \cdot \sum_{i=1}^{13} W_i \cdot F_i$$

C1=0.6, C2=0.01, Technical Factor Total=

TCF= .955
Duration = UCP*PF = 103.14*28 = 2887.92

Domain Analysis

Domain Model



The domain model is derived from the concepts, attributes, and associations from all the use cases and requirements.

Concept Definitions

Rs#	**Responsibility Description**	**Type**	**Concept Name**
Rs1.	Coordinate the actions that the user wants the drone to take.	D	Controller
Rs2.	Shows the physical data of the drone along with a live camera feed on a website.	K	Interface
Rs3.	Establishes a remote connection between the camera of the drone and the controller.	D	Connector
Rs4.	Renders the data onto the website.	K	Page Maker
Rs5.	Calculates the speed, battery life and the location of the drone.	D	Dynamic Data
Rs6.	Notifies the user of potential issues such as low battery or obstacles.	D	Notifier

Association Definitions

Concept Pair	Association Description	Association name
Page Maker ↔ Interface	The Page Maker prepares the logic for the Interface	Display
Dynamic Data ↔ Notifier	The Dynamic Data informs the Notifier if there are any issues that the user needs to know.	Create Alerts
Connector ↔ Controller	The Connector makes sure that the inputs given by the user with the Controller are properly executed with minimal latency.	Sync IO
Dynamic Data ↔ Page Maker	The Page Maker records the physical data of the drone on the website.	Record Data
Notifier ↔ Page Maker	The Notifier sends the alert information to the Page Maker	Send Alerts
Controller ↔ Interface	The movement of the drone by the Controller is picked up by the Interface's live video feed.	Convey Movement

Attribute Definitions

Concept	Attributes	Attribute Description
Interface	Drone data Drone camera feed	Used to show the user the drone’s physical data. Will show the user a physical view of the live camera.
Controller	Input direction	Up, down, left, right, forward, and back.
Page Maker	Data list	Data that will be rendered on the website.
Dynamic Data	Speed Battery life Location	Will look for specific types of drone data such as speed, battery life and the location of the drone at all times.

Traceability Matrix

		Domain Concepts						
Use Cases	Priority Weight	Interface	Controller	Connector	Page Make r	Dynamite Data	Notifier	Calibrato r
UC-1	5	x	x					x
UC-2	2	x			x			
UC-3	4			x		x		
UC-4	4						x	
UC-5	4		x					x
UC-6	11			x	x	x	x	
UC-7	7	x		x		x		x
UC-8	6		x				x	

System Operation Contracts

Operation	MoveDrone
Preconditions	• Drone is available • Controller is available • Application is open • The physical mechanism of the drone is undamaged and operable
Postconditions	• Allows the user to maneuver the drone using a controller and a camera • Get visual feedback about the movement

Operation	GetLocation
Preconditions	• The GPS is on and in a working condition • The connection between the drone and controller is stable
Postconditions	• Allows the user to retrieve the current location of drone displayed on the webpage

Operation	CheckObstacles
Preconditions	• The sensors are on and in a working condition • The physical mechanism of the drone is undamaged and operable
Postconditions	• Allows the drone to detect obstacles that can possibly damage

	or interrupt its mission using the sensors.

Operation	**GetStatus**
Preconditions	● A signal between the drone and the controller is available
Postconditions	● Allows the user to know if the drone is active or not

Operation	**GetData**
Preconditions	● The drone is on and a connection between the drone and controller is established
Postconditions	● Allows the user to manipulate and store that data. ● Get webpage of data

Mathematical Model

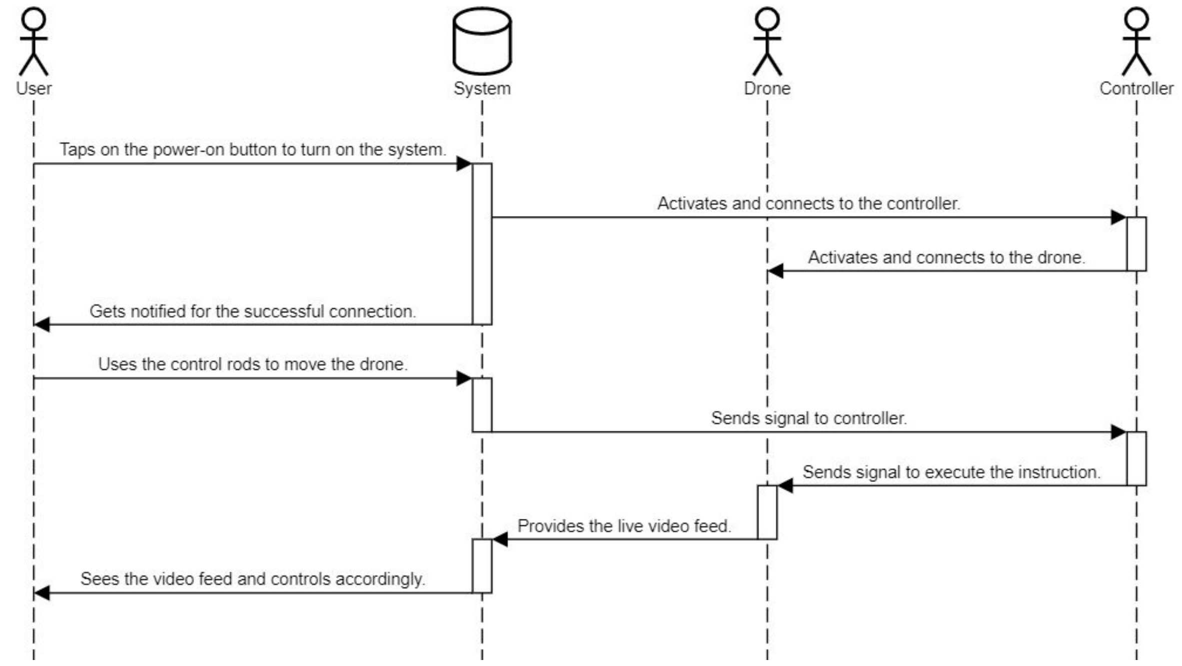
The drone will be calibrated to use math in order to move and avoid obstacles. This model is correlated with UC-1 and UC-7.

UC-1 (MoveDrone): A controller is used to move the drone. The user can control to rotate, move forward/backward, throttle, and strafe the drone. This will involve adjusting the speed of motors to change the velocity and angle of the drone.

UC-7 (GetData): The user will be able to retrieve the battery level, speed, and position of the drone. These values will be updated in real time via the drone’s wireless connection. The speed is calculated by taking the latitude and longitude, and when it updates, calculate the distance divided by the time it took to update.

Interaction Diagrams and Design Principles

Use Case 1: MoveDrone


User
System
Drone
Controller
Taps on the power-on button to turn on the system.
Activates and connects to the controller.
Activates and connects to the drone.
Gets notified for the successful connection.
Uses the control rods to move the drone.
Sends signal to controller.
Sends signal to execute the instruction.
Provides the live video feed.
Sees the video feed and controls accordingly.

The diagram for the first use case is displayed above. In this case, the User will first activate the system in order to gain access to the webpage. From there the user will use an RC controller to manually control the actions of the drone.

Design Principles:

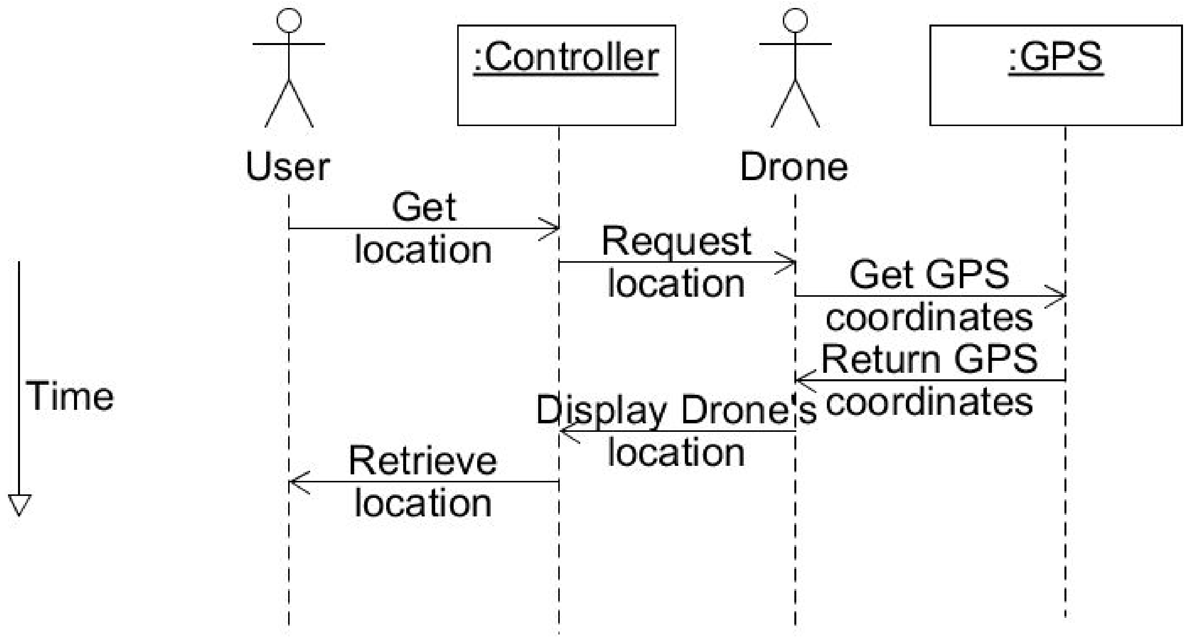
The design principles utilized in this use case include the Low Coupling Principle. This design principle is utilized as the communication links that exist are very short. Most of the communication is done between the User and controller, and then the controller and drone.

In this use case diagram, the controller could act as the publisher while the drone will act as the subscriber. The user utilizes the controller to send events to the drone which will work if it has received a valid event.

Alternative Solutions:

We originally planned to have the user control the drone using the webpage by having control buttons on the webpage. This idea was scrapped because there would be a great amount of latency introduced onto the system. Using the RC controller will mitigate the total latency and improve overall performance.

Use Case 3: GetLocation



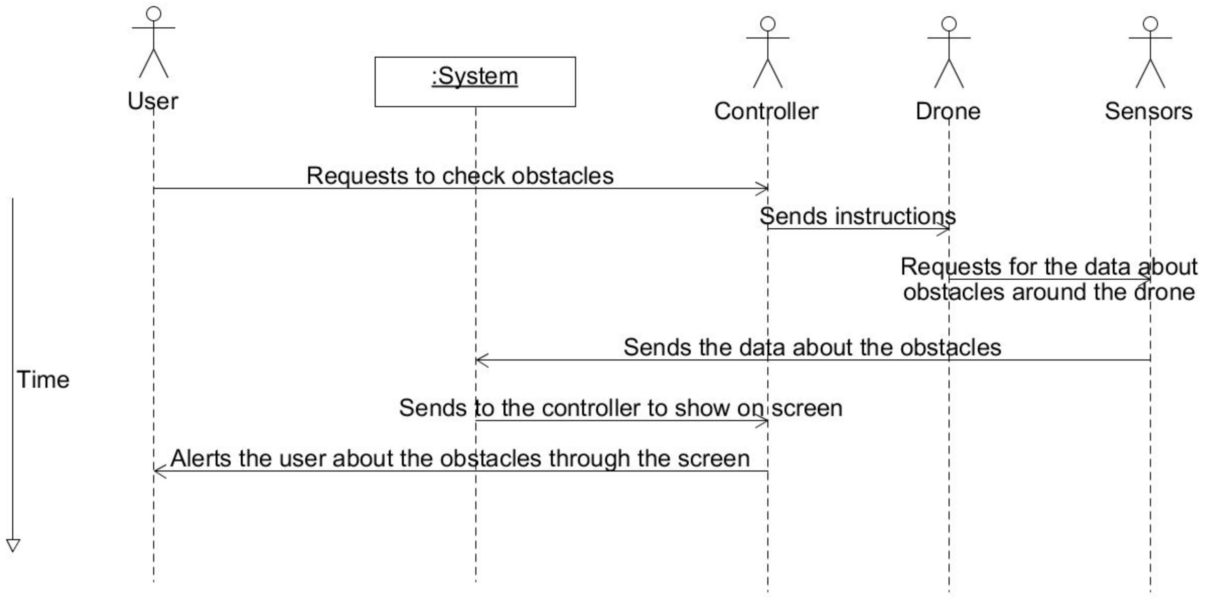
The diagram above demonstrates the interactions between classes in UC-3: get the location. Once the user has control of the drone and being able to maneuver around obstacles. First, the user sends a request to get the location of the camera mounted on the drone to the controller which then gets requests it to the phone. The phone then requests the coordinates to the GPS server and gets the coordinates and sends it back to the controller to make it visible. The user sees the phone's location based on longitude and latitude. The location of the phone was then updated by repeatedly getting requests to update the location. If the location every changed then the new location was then presented to the user, otherwise, the old location will still be shown to the user.

Design Principles:
The design principles employed in the process of assigning responsibilities were the expert doer principle and high cohesion principle. The expert doer principle is used because each of the classes is an expert for specific functions. An example, the phone is responsible for getting the coordinates from the GPS server and relaying it back to the controller for the user to see.

This use case diagram will also work with the publisher-subscriber design pattern. The controller will act as the publisher while the phone and the GPS act as the subscribers. The controller will send a request to update the current GPS location of the phone to the GPS, which in turn will verify the location of the drone and then return the coordinates.

Alternative Solutions:
The original idea to get the location was to use the drone but that was scrapped in favor of using the phone. This is because the phone had a built-in GPS, which we were able to access in order to get the location of the phone. The phone is going to be mounted onto the drone so the location of the phone is also going to be the location of the drone as well.

Use Case 4: CheckObstacles


:System
User
Controller
Drone
Sensors
Requests to check obstacles
Sends instructions
Requests for the data about obstacles around the drone
Sends the data about the obstacles
Time
Sends to the controller to show on screen
Alerts the user about the obstacles through the screen

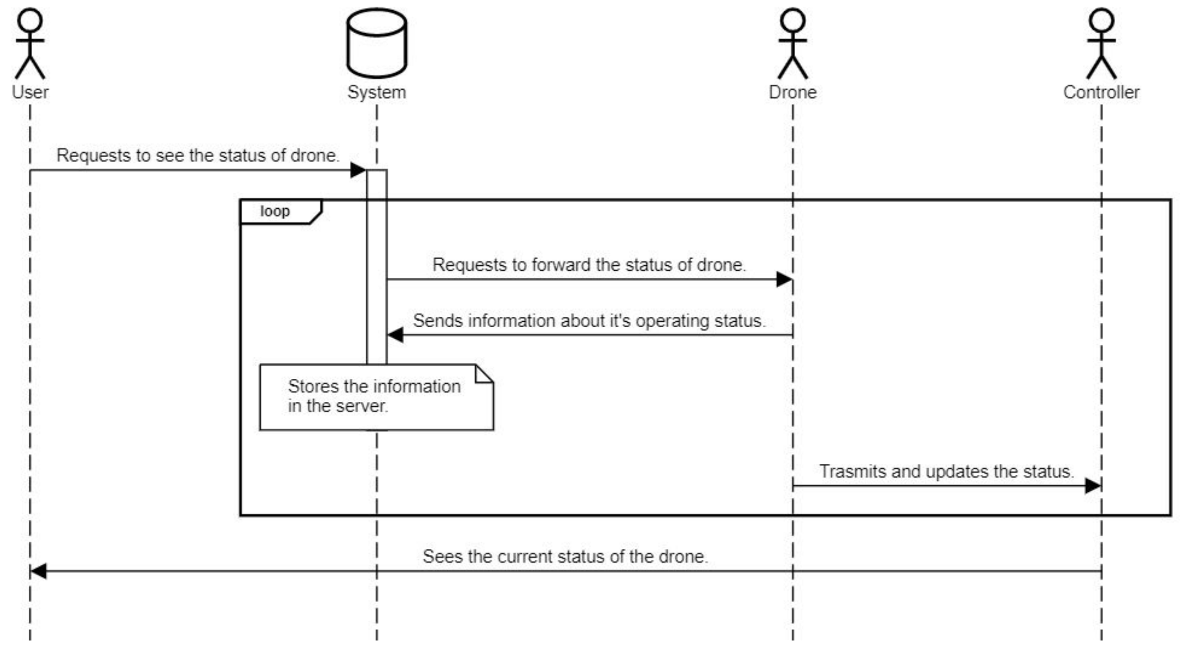
The way that the diagram above is implemented is that the user is able to get an alert that an object is a certain distance away. The sensors on the drone are able to locate objects that are close to the drone. If an object is under a certain distance then the user should get an alert saying that there is an object a certain distance away from this one sensor.

Design Principle:
The design principle for this use case is the expert doer principle and high cohesion principle because the parameters for the obstacle is super specific. And that data should be focused on because it can affect the overall behavior of the drone. It is also important that the specific obstacles that are being checked for are being communicated to other sources.

Alternative Solutions:
The original implementation for this use case was to have sensors all over the drone that would be able to detect objects in any direction. This idea was replaced by having a sensor on the left and right sides of the drone. This was done in order to reduce the possibility of a false reading from another drone.

This could happen as the echo from one sensor was picked up by another sensor, which could occur in confined environments. The user is able to see in front of the drone and behind the drone due to the front and rear-facing cameras. Their only blind spots would be on their peripherals. This could be solved by having a sensor on the right and left-hand sides, which will be able to detect any objects.

Use Case 6: GetStatus


User
System
Drone
Controller
Requests to see the status of drone.
loop
Requests to forward the status of drone.
Sends information about it's operating status.
Stores the information in the server.
Trasmits and updates the status.
Sees the current status of the drone.

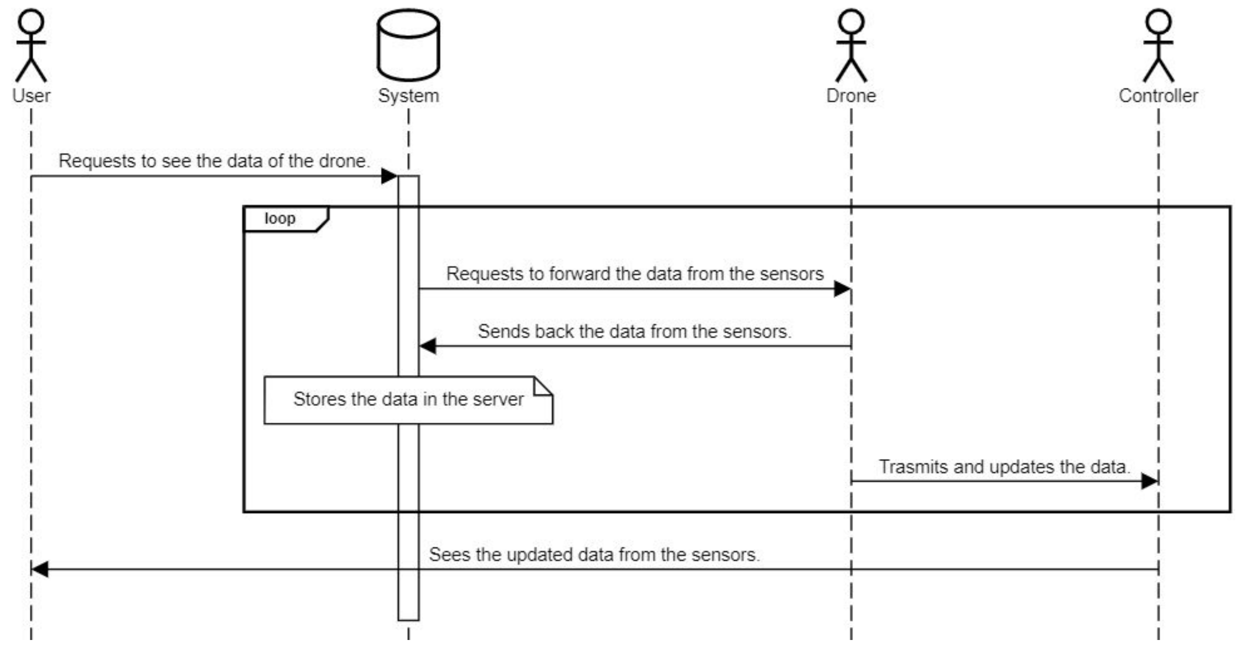
The interaction diagram for use case 6 is displayed above. The drone basically sends a signal to the system which the user can see the result of through the controller. The part of this use case is to let the user know of the operation status of the drone, in particular, the battery level. The user is also able to view the past values sent by the drone.

Design Principles:
The design principles utilized by this use case are Expert Doer Principle, High Cohesion Principle, and the Low Coupling Principle. Since this use case is the only use case that knows about the battery status it makes sense that the

Expert Doer Principle is used. As for the High Cohesion Principle, the only computation done by this part is the battery level. The Low Coupling Principle deals with the concept that this use case does minor communication between the drone and the controller.

Alternative Solution:
Originally we had planned to show the battery level of the drone on the webpage. This idea was replaced as soon as we came to utilize the RC controller. The reason for this is because the RC controller has a warning that will automatically inform the user when the battery of the drone is low. So instead we replaced the battery of the drone with the battery level of the phone. This is because the user will then be able to know that they need to pull back the drone and replace the phone counted camera. The phone is being utilized for many purposes so it is important to know how much charge is left on the phone.

Use Case 7: GetData


User
System
Drone
Controller
Requests to see the data of the drone.
loop
Requests to forward the data from the sensors
Sends back the data from the sensors.
Stores the data in the server
Trasmits and updates the data.
Sees the updated data from the sensors.

The diagram above demonstrates how the user, controller, and drone interact with each other to show necessary data of drone to the user, so the user can control the drone. When the system needs data, the drone sends the data that is saved on it to the controller upon the request by the controller. When the controller receives the data, it displays it on the webpage, so the user can see the data and make necessary judgments of controlling the drone. The user will verify its execution by the updated live-feed.

Design Principles:

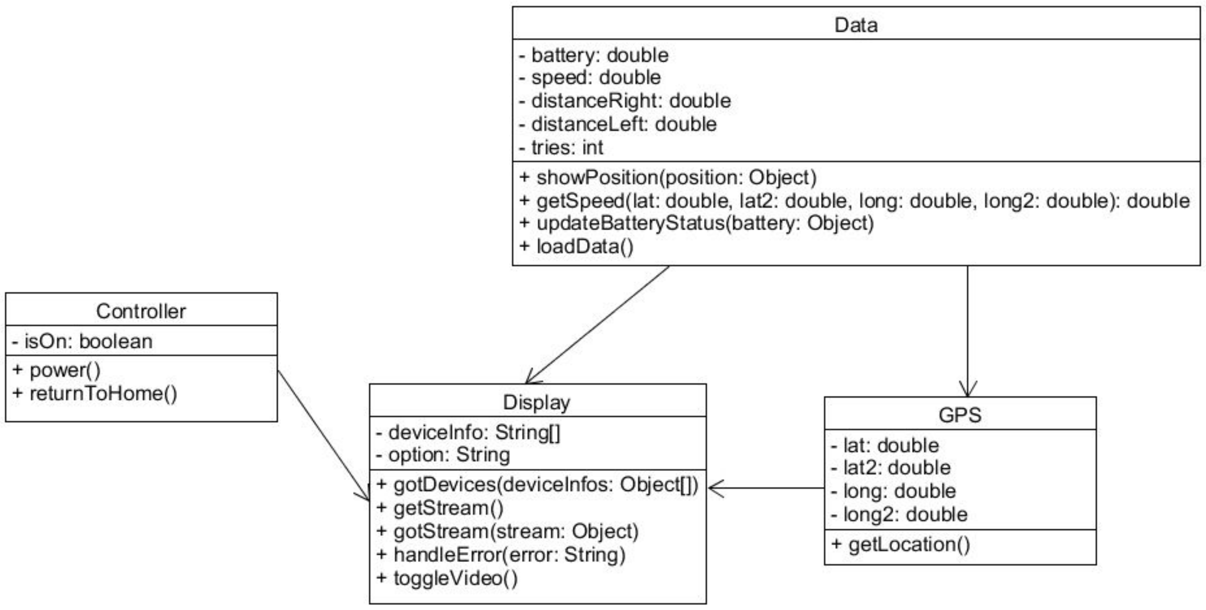
The design principle of this use case is High Cohesion Principle. There is more focus on displaying the necessary data and sending instructions to drone to control it, rather than having a high responsibility of computing data.

Alternative Solutions:

The original idea to get the data was to refresh the data every few seconds and sent that to the user via the web page. This was eventually changed to send the data only when something changes. For example, the location is only to update and be sent to the webpage when the system notices that the current location of the phone has changed. This idea was implemented because it makes sense to inform the user of any changes in the data received as that informs the user that something is happening.

Class Diagram and Interface Specification

Class Diagram



Data Types and Operation Signatures

1. **Controller:** The main role of this class is to use the help of the other classes to maneuver and utilize the drone.

   Operations

   1. power(): This function turns the drone on and off.
   2. returnToHome(): The drone returns to where it was launched

   Attributes

   1. isOn: Boolean -It has a value of true or false depending on if the drone is on or not.

2. **Data:** The system class mainly deals with the inner mechanisms of the drone and its operational status.

   Operations

   1. showPosition(position: Object) - Void: Get the position of the drone by reading the device's data.
   2. getSpeed(lat: double, lat2: double, long: double, long2: double) - Double: Get the speed of the drone by using the device's location to calculate speed.
   3. updateBatteryStatus(battery: Object) - Void: Get the battery status from the camera device on the drone.

4. loadData() - Double: The point of this function is to get the data from the sensors.

Attributes

1. battery: Double - The amount of battery life left on the drone.
2. speed: Double-The speed of the drone in meters per second.
3. distanceRight: Double - The distance from the right sensor to the measured object.
4. distanceLeft: Double - The distance from the left sensor to the measured object.
5. tries: Integer - The number of times it took before the location was able to update.

3. **Display:** This class is mainly responsible for showing and updating the graphical user interface.

Operations

1. gotDevices(deviceInfos: Object[]) - Void: Allows the webpage to access multiple cameras on the device if there are any.
2. getStream() - Void: Sets up the drop down menu for the video menu.
3. gotStream(stream: Object) - Void: Sets up the video feed.
4. handleError(error: String) - Void: Prints any error that comes with displaying the video feed.
5. toggleVideo() - Void: Switches the camera feed from being visible to not being displayed and vice-versa.

Attributes

1. deviceInfo: String Array - List of all compatible devices.
2. option: String - Text that shows the drop-down menu.

4. **GPS:** The class GPS is used to implement the location component of the drone. The functions of this class are displayed below.

   Operations

   1. getLocation()- This uses the GPS to get the location of the drone in the form of latitude and longitude.

   Attributes

   1. lat: double - The current latitude of the drone.
   2. lat2: double - The past latitude of the drone.
   3. long: double - The current longitude of the drone.
   4. long2: double - The past longitude of the drone.

Traceability Matrix

Domain Concepts/ Software Class	Controller	Data	Display	GPS
Controller	X			
Interface		X	X	
Connector	X	X		X
Page Maker			X	
Dynamic Data		X		X
Notifier		X	X	
Calibrator	X	X		

The software classes were developed from the domain concepts based on the required functionality of each domain concept. The core functions of the drone involved movement, operational status, user interface, and location. Each of these core functions represents a software class. So the traceability matrix above represents how each of the domain concepts corresponds to the core functions of the drone i.e. the software classes.

Design Patterns

One of the design patterns that is mainly used in our code is the publisher-subscriber pattern. This is based of event like structure. So if a certain event occurs, a certain part of the code runs. Some events that occur in our code is permission requirements from the user. So in order to run the location and camera feed the user need to grant permission to the program. This style of coding allows us to add more components in the future if we decide to do so. This also allows us to change the “publisher” if we need to for various events and we can also add “subscribers” as well.

The publisher-subscriber model is predominantly used in aspects of the interface that are constantly updating. This include the speed, location, camera feed, sensor data, and battery levels. Each of these are event based, and each event leads to a possible update of the attributes.

Object Constraint Language (OCL) Contracts

Invariants

- The browser the user is using needs to support Javascript and support HTML5 video. It also needs to be able to support the functions in the code.
- The battery level that is going to be displayed is always going to be greater than 0.
    - Context controller: getBattery()
    - label.textContent = `${Math.round(battery.level*100)}%`;
- The user must have a good wireless connection between the Pi and the device that is in use. The connection should preferably be at a frequency of 2.4 GHz.
- The Raspberry Pi will always be powered on, upon starting the flight and will thus give readings within the range of ~ 2 cm to 200 cm.

Preconditions

- To execute get location, the code needs to make sure the user allows their location to be accessed
    - Context: controller-getlocation()
    - Pre: if (navigator.geolocation)
- In order to view the camera feed of the device, user permission must be granted.
    - Context: controller-getdevice()
    - Pre: navigator.mediaDevices.enumerateDevices() .then(gotDevices).then(getStream).catch(handleError);
- In order for the program to output a distance between the drone and the object, the distance has to fall under the threshold distance of 2 m.

Postcondition

- The post condition for the speed is the location. If the location has been updated then the speed will also update. If it has not been updated then it will not update.
    - Context: velocity=getSpeed(lat,lat2,long,long2)
    - Post: if(lat === lat2 && long === long2)
- The post condition for the power button is to toggle the videofeed. If the feed is currently visible it will turn it off. If it is not visible it will show the video feed.
    - Context: controller-togglevideo()
    - Post: document.querySelector('.fa-power-off').onclick = toggleVideo;
- A post condition for the sensor reading is the automatic updating of a text file in 1 second intervals to update the distance measured. If this doesn't happen, then the precondition has not been met yet.

System Architecture and System Design

Architectural Styles

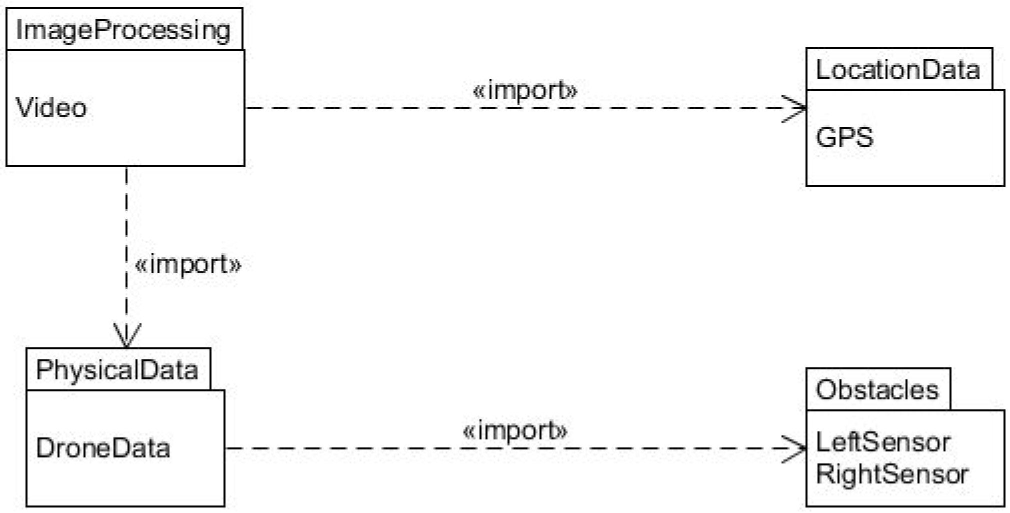
REST: Since all the information about the drone is displayed on a website using HTML, it complies with a RESTful API. The video feed and physical drone data are gathered from servers controlled by other subsystems and represented as hypertext.

Client/Server: The client is the person controlling the drone and the server is the information picked up by the drone. When the user wants to throttle the drone, for instance, a request is sent to the drone’s motors to move up or down, which then responds with movement seen by the live video feed.

Layered: Certain services of the drone depend on each other. For example, getting the current location of the drone is initiated by the controller, which depends on the drone requesting its location, which depends on the coordinates returned by the GPS.

Uniform interface: All resources should be reachable from any devices. It should not be limited to only one device. The website should be simple but effective.

Identifying Subsystems



The ImageProcessing package contains the Video class for displaying the live video feed. It imports the LocationData and PhysicalData in order to display those on the website.
The LocationData package has a GPS class to request its location and use it to calculate the speed of the drone.
The PhysicalData package contains the DroneData class to display the operating status of the drone. It imports the Obstacles Package to retrieve the sensor data.
The Obstacles package has classes for each of the two sensors. They each return the distance from their nearest object.

Mapping Subsystems to Hardware

1. Physical Data

The majority of the hardware for physical data will incorporate the Raspberry Pi, which will send the required signals to the software component. The motors will also be involved when it comes to the detection of speed.

2. Image Processing

The hardware needed for this substem will include a camera inside a phone that will be mounted to the drone. The mobile device is going to be a Samsung Galaxy S4. The rear camera is a 13.0 MP autofocus camera with LED flash, with a Sony IMX091PQ sensor. We are also using an infrared lens to be able to detect people. With the added thermal sensor called SeekThermal Compact, we were able to also include another feed of the phone with thermal imaging capabilities.

3. Obstacles

The hardware that is mapped from the Obstacles' subsystem is the ultrasonic sensor that will be attached in multiple locations around the drone. The model of the ultrasonic sensor that is going to be used is HC-SR04.

4. Location Data

The mapped hardware for the Location Data subsystem would be the GPS, which is inside the smartphone that is mounted to the drone.

Persistent Data Storage

The drone will be equipped with Raspberry Pi. Even with the drone powered off, this system will be capable of saving any data from previous flights. However, this data can be transferred to another system since it is unnecessary for the drone to carry all the data from previous flights. But, during regular flight time, the drone will be overwriting data to a text file constantly, which will be read by the webpage. The overwriting of the data allows the webpage to not read through too much data.

Another form of data storage is the image component of the drone. Even though it is a live feed, it will require some form of data storage through cache memory. This is due to the fact that the image will be required to be transferred from one device to another. This also applies to the thermal imaging software that will be installed on the device that acts as the camera on the drone. Similarly, the ultrasonic sensor will also have a cache data component. The ultrasonic sensor will have to transmit the distance between the drone and any obstacles to the controller. This data does not need to be stored for a long time but is still required if any action is needed to be taken by the drone.

Network Protocol

For managing the network that our system will make use of, the HTTP communication protocol will be utilized. This was chosen because the data that is being transmitted ends up as part of a browser-based display, for which HTTP can be used for simpler client-server interactions. The webpage that the drone operator sees requires data regarding the drone's location (GPS), the live camera feed, and other drone-related physical data (battery level, speed, etc.). These need to be delivered across the drone's connection to the operator's device, which naturally calls for a web-based communication protocol layered around the TCP/IP.

Global Control Flow

Our project can be noted as both procedure-driven and event-driven. The reason behind this is because initially, the same steps have to be taken to initially operate the drone however it is mainly an event-driven system because the case for why this drone is being used is different. There are a lot of situational factors so the user must generate a different series of actions in a different order depending on the specific case we are looking at. So it is mainly event-driven because it is very unlikely that the same steps will be taken in the same order for more than one event.

Our system is an event response type with concern for real-time. Since it is real-time, it is not periodic. It is not periodic because the time differs for different situations. There are no time constraints for each case because we don't know how long each case would take.

Hardware Requirements

The access to control the drone can be done through any touch-enabled device with an internet browser such as a smartphone or tablet. The device requires a minimum of 1 GB since to process the live-feed video from drone smoothly. There will also be a camera mounted on a phone that is placed on the drone in order to capture video. The camera should be able to record steady 1080p video. This can be done by using any phone camera that is better than or equal to that of the Samsung Galaxy S4, which has an image sensor from Sony called IMX091PQ. In addition, the phone will also have thermal imaging sensor attached to the mounted phone. This can read temperature differences from -40 ℉ to 626 ℉. The camera will also have a viewing angle of 36° and a maximum viewing distance of 1000 ft. The exterior of the drone will have a Raspberry Pi mounted underneath the drone. The device that will process the live-feed from the camera has to have a colored display of a resolution of at least 1920 x 1080 to allow the user to see where the drone is clearly. This can be done with any modern display devices like smartphones or tablets. Because of the quality of the image that is transmitted from the camera on the drone, the connection between the controller and the drone has to operate smoothly, with relatively low latency. The wireless connection bandwidth is a 2.4 GHz connection. The 2 HC-SR04 ultrasonic sensors we used were utilized in conjunction with a Raspberry Pi. The sensors have to be connected to the GPIO pins on the PI through jumper wires. The jumper wires had to be male to female connectors, where the female connectors would go to the Pi and the male connectors connect to the channels on the breadboard which the ultrasonic sensor is mounted on. The Raspberry Pi model that was used was a model 3B that had an onboard 802.11n WLAN adapter. The Raspberry Pi needs to be powered by a 5.1 V power source up to 2.5 mA of current. This can be done by using a power bank/portable battery pack that can output the desired voltage and current.

Algorithms and Data Structures

Algorithms

The main factor of this project is to have a safe and efficient flight for the drone. The drone will not be capable of performing tasks such as search and rescue if the drone is not durable. To accomplish this goal is to control the velocity of drone and locate any obstacles on its way. Calculating the velocity and distance between the drone and obstacles involve complex algorithms.

The velocity of the drone can be calculated by using the formula that states that $v = a * t$. The variable v will stand for velocity, a will stand for acceleration, and time stands for time. Of course, though, we will have to account for other factors such as thrust and pitch for when we are going over the drone's movements. The total amount of thrust is going to be equal to the following equation.

$Ft = Ft0 * (\frac{Vmax - V}{Vmax}) - Fd.$

F_D is the drag force.

F_{t0} is the force of thrust when the velocity is at 0 meters/second.

$F_D = 0.5 * \rho * Cd * [(A(front) * cos(P(max) - P(motor))) + (A(top) * sin(P(max) - P(motor)))] * v^2$

For the above equation, the constant ρ is going to be equal to the density of air, while the constant C_d is equal to the drag coefficient. The variable P_{motor} is the pitch of the motor, while P_{max} is the maximum pitch the drone is able to achieve without losing altitude.

$P_{max} = cos^{-1}(\frac{m}{T0})$

The variable m is equal to the mass of the drone and the variable T_0 is equal to the total thrust of the drone.

When the ultrasonics sensors recognize any obstacles, the drone needs to know where the obstacles are. The drone is equipped with two ultrasonic sensors. The two sensors should locate the exact location of obstacles and alert the user if necessary, so the user can maneuver the drone. They are placed on the right and left sides of the drone to cover the blind spots that the primary drone camera cannot cover. This can be also used in the function such as "return to home" when the drone autonomously returns to the base.

The distance between two points can be calculated based on the pulse sent by the trigger pin on the sensor. The trigger pin sends a 10 µs TTL, 40 Hz pulse toward the object that the sensor has to measure with respect to. The code is designed to calculate the distance by timing how long the pulse sent, takes to return to the echo pin. This time is then divided by 2 since the distance to an object is a calculation. In addition, we also know the speed of sound in air to be 343 m/s or 34300 cm/s. The latter will be used because the Pi measures the distances by default in centimeters. So, considering we now know the speed and the time it takes for the pulse to reach the measured object, we can now solve for distance. Using a simple physics kinematics equation we can see that:

$d_{OBJECT} = v_{SOUND} \times t \Rightarrow d_{OBJECT} = 0.5 \times 34300\ cm/s \times t \Rightarrow d_{OBJECT} = 17150\ cm/s \times t$

The speed can be calculated by using latitude and longitude. This is first done by utilizing the *Haversine Formula* to get the distance between two points given latitude and longitude. This could be done by the following formula: distance = 2*r* $\sqrt{sin^2(\frac{lat2-lat1}{2}) + cos(lat1) * cos(lat2) * sin^2(\frac{long2-long1}{2})}$. This distance was then divided by the number of tries the system attempted to get a new location to get the speed of the drone.

However, since the drone is equipped with the ultrasonic sensors, it can use the time it took for an ultrasonic wave to travel to an obstacle to calculate the distance between them. The equation will be,

$distance\ to\ object = \frac{time * speed}{2}$.

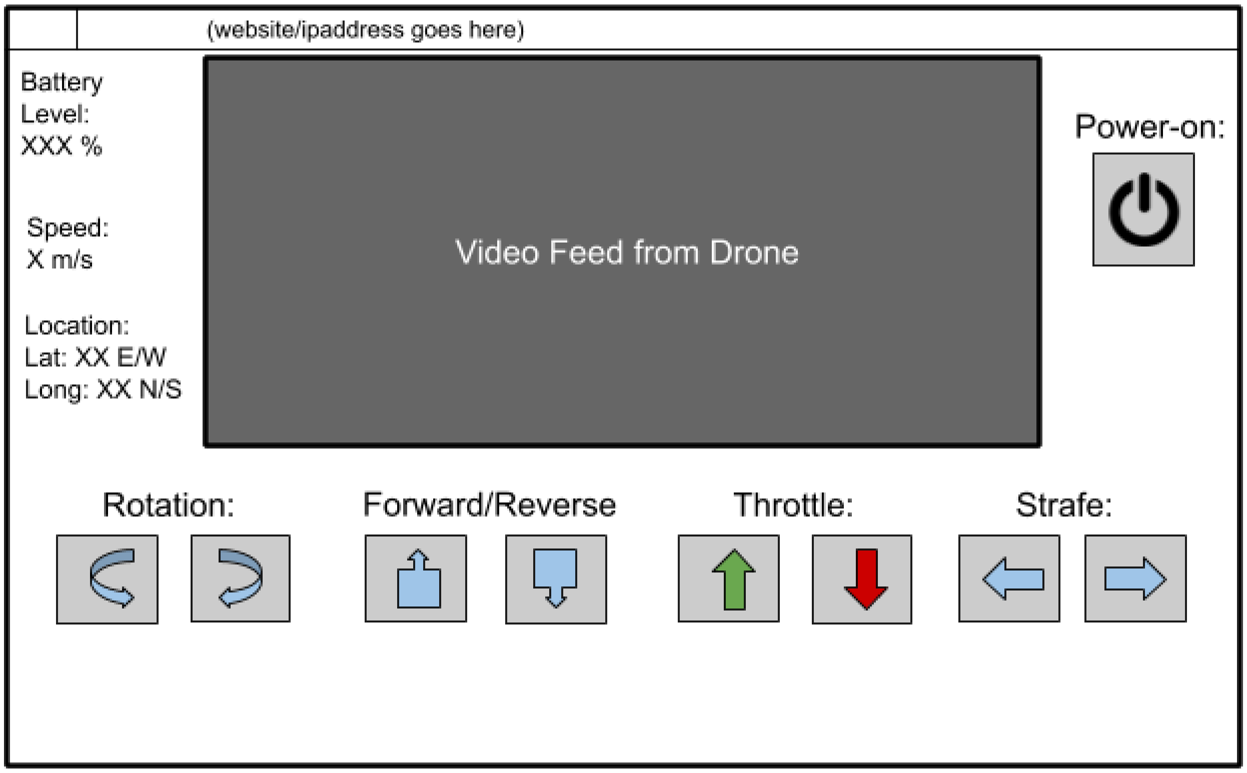
Time is divided by two since the time it took is an ultrasonic wave is to be emitted and reflected back to the drone combined. Only one way is needed.

Speed is the speed of the ultrasonic wave, which will be 340 meters/second in the air. Since the speed of the ultrasonic wave is significantly greater than the speed of the drone, that speed of drone can be ignored in the calculation.

If the distance is less than a safe distance, the user will notice the drone through the alert on screen and will be able to maneuver. It is very important that our algorithm is consistently checking this distance because it can alert a safety issue if needed.

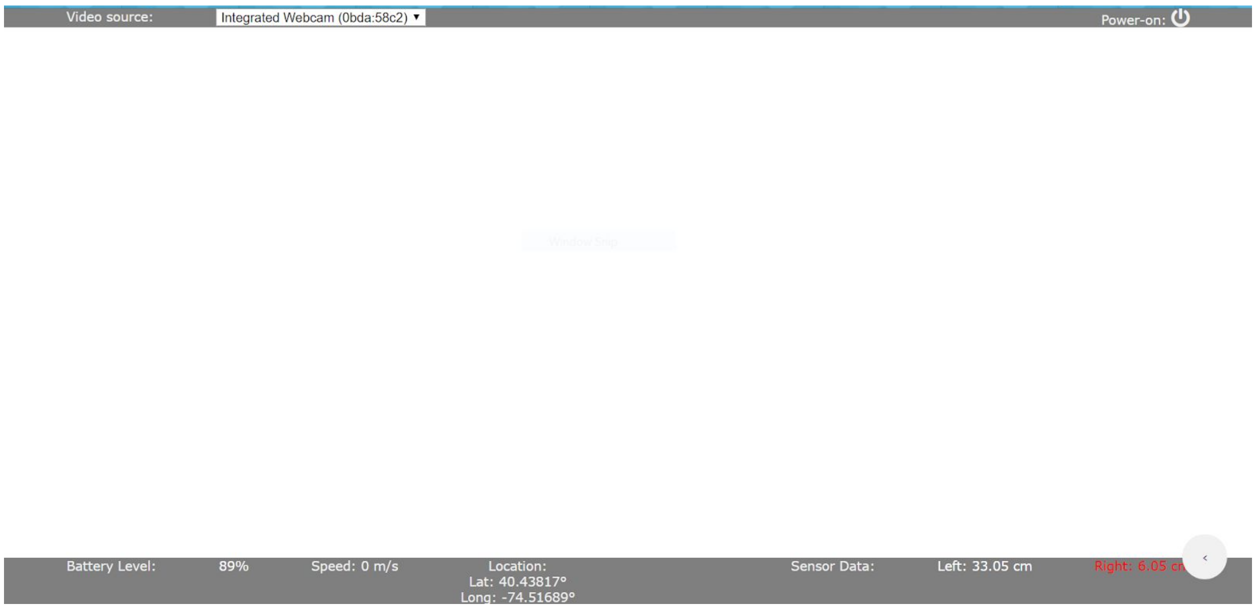
User Interface Design and Implementation

The design of our user interface went through several revisions since its inception.



Our initial mock-up

Since we didn't follow through with our plan to make the website a controller, we got rid of the controller inputs and rearranged all the other data so that the video feed can take up as much room as possible. That way, the user can have a good field-of-view when navigating the drone through obstacles. This new redesign minimizes the user effort while browsing the website and makes the interface easy to understand to anyone wanting to view the drone's flight.


Video source: Integrated Webcam (0bda:58c2) ▾

Power-on:

Battery Level: 89%

Speed: 0 m/s

Location:
Lat: 40.43817°
Long: -74.51689°

Sensor Data: Left: 33.05 cm Right: 6.05 c

Our final design (video off)

The user can access this site at:
https://abhiek187.github.io/emergency-response-drone/1_code/controller/control.html
If the user bookmarks this site, it will only take 1 button press to load this page. The user can then change the video source through a dropdown menu (requiring 2 button presses) and power the drone video on and off (1 button press). All the other features will change based on external events. The battery level mimics the device's battery level and will alert the user if it drops below 20%. The speed changes based on the change of location in latitude and longitude. To the right is sensor data from the Raspberry Pi's ultrasonic sensors. They can detect the distance of their closest objects and update their data in real time (assuming the user has a connection to the Raspberry Pi). The user will be alerted if an object is less than a foot away.

Design of Tests

A.

Types of tests:

1. MoveDrone- The user should be able to control the movement of the drone. Any instruction that the user gives via the controller should be received and performed by the drone. It should be able to successfully complete each type of movement. This includes rotating clockwise and counterclockwise, moving forward/backward, moving up/down and finally moving right/ left. So there will be a test for each individual type of movement.

2. GetData/GetStatus - The user should be able to receive and view the data the drone is transmitting. The battery level should match the device's battery level and update in real time. If it's below 20%, the text should turn red and a low battery alert should be displayed on the website. The location should update every few seconds every time the person moves the device and the speed should change as well based on the change of location.

3. ViewCamera - The user should be able to receive the video feed from the drone. The drone will transmit the video feed from its camera to the website. The video input should list all the possible camera options built into the device and display the right image (i.e. front and back-facing cameras should display the right thing). The video feed should continue to transmit no matter what action the drone takes.

4. CheckObstacles- The user should accurately be able to detect any nearby obstacles. It should work in all directions that the sensors are facing. Once it finds the obstacles, it needs to notify the user of the distance and direction of the obstacle. When testing the website directly on the Raspberry Pi, the sensor data should update every time the Python script updates the text file.

Like the battery level, the text should turn red to warn the user if they're within a foot (~ 30 cm) from an obstacle.

5. Power- The video feed from the camera should be able to turn on or off based on the user's decision. Once the user decides to turn off the camera then the video feed on the web page should then become blank.

6. Display- Runs the interface and shows all the features incorporated into the interfaces. This includes the video feed from the camera—regular and thermal and the data values of the drone from the Pi.

B.

ViewCamera-We ended up testing various approaches to see which version would be clear and see how far the range for each approach was. During testing, we had two different approaches which were screen sharing and an app called Alfred. The test involved establishing a connection between the phone and computer and then seeing how far the phone can be taken away from the computer and still have a clear image. At first, when both approaches were close to the computer, the screen share was very clear and Alfred had good clarity. However, the quality of the screen-sharing app rapidly decreased as the phone moved away when using screen sharing. When it came to Alfred, the quality remained the same for a while, but as the phone moved farther and farther away, the quality decreased. This is because the phone was starting to go out of range of the wifi.

GetData/GetStatus-One of the data values that we have tested includes the location of the device in use. This was done by running the code for the controller in three different locations and seeing that these are in fact different in latitude and longitude. The test was conducted in Edison, South Brunswick, and East Windsor. As expected each of these values was different. In Edison, the values were: Lat: 40.55° and Long: -74.39°. In South

Brunswick, the values were: Lat: 40.38 ° and Long: -74.54°. In East Windsor, the values were: Lat: 40.25° and -74.53°.

Display- We tested the display to check for how the controller responded to the user inputs and how fast or slow the latency of the device was. This was done by triggering different buttons to observe the behavior of the controller. We also checked the controller to see if the different latitude and longitude coordinates mentioned above were displayed on the screen for three different locations. This was also tested using different devices like a laptop, a smartphone, or a tablet. In all of these cases, the live feed was present, the buttons were working, and the data from the drone was displayed correctly on the screen.

Check Obstacles- To test the responsiveness of the ultrasonic sensors, we set up different test cases and measured the time it took for a signal to be outputted from the sensor. This was done by having the sensor emit several sound waves at approximately 40 Hz, and then observing the output based on the time it took for the wave to reach back to the sensor. This was repeated for objects placed in a straight line approximately 1 meter, 2 meters, and 4 meters away. Although the accuracy of the sensor did slightly decrease the further the distance became, it was able to accurately determine the distance to within < 5 cm of the target for larger distances and < 1 cm for smaller distances.

C.

We are planning on using the Horizontal Integration method for our testing. More specifically we will be using the Top-down integration version for our testing purposes. The first test we will have to do is display because the controller needs access to the buttons in order to run the rest of the tests. Once the display is on the next test will be power. This will turn the drone on and allow us to test the functionality of the drone. The next thing that will be tested will be ViewCamera. This will allow the user to see the video that the drone is transmitting. Once the video feed is working the next step is to see if the drone can move. A test will be conducted to see that it is possible to maneuver the drone in all 6 directions. Once the drone is in operation the next step is to test the check obstacles function. This will let the user know if there are any obstacles nearby. Once the drone is in stable operation the final step would be to use the getData/getStatus to see the properties of the drone in action.

History of Work, Current Status, and Future Work

Merging the Contributions from Individual Team Members

Shantanu came up with the project idea and was able to explain how we could contribute to the project during weekly meetings. We decided to split the work into four subgroups: image processing, location data, physical data, and obstacles. Since not everyone can make it to the weekly meetings, each subgroup has set up their own meeting times to discuss specific functionalities to be implemented in this project. This also ensures that each person can discuss how they will contribute toward building S.A.R.A.

Krishna Mahadas created and shared the Google Drive for our project so we could easily collaborate on creating the reports.

Abhishek manages the GitHub repository to maintain the project code and divide the work among the team. Each branch corresponds to the different subgroups. Each person works on their subgroup work and when it's ready to be implemented, it is merged into the master branch.

A website is going to be made and developed with relevant updates to the project. This will be managed by Abhishek. Other team members will help.

Project Coordination and Progress Report

Image Processing:
The image processing component of the project mainly implements the use case ViewCamera. So far we have already been able to display what the phone camera is seeing on other devices such as a pc. We tried using multiple third-party applications and features of the Android phone to see which works best. Some third-party applications we tried are Sidesync, Alfred and IPwebcam. All of these applications are able to display decent quality video feed for a reasonable range using wifi. Another approach was using the screen mirroring function of the android. This approach also uses wifi and provides a really good quality image. However, it does not have much of the range due to the fact the phone needs to be close to where ever the display is being transmitted to. So we decided to try using the Alfred application at first but later switched to Sidesync. The Sidesync application allowed us to have a better version of screen mirroring and actually control the phone from the pc. So we are able to run our interface on the phone and then display it on the laptop. This is being done by using the HTML code of the webpage version of the application since the camera feed is coming from the phone camera itself. A prototype of the controller can be found on our website. The last aspect of image processing was to implement the thermal video feed. This was done using a micro usb thermal camera, and once again the Sidesync app is used to transfer the thermal feed from the phone to the remote pc.

Location Data:
The use case that is the main function of location data is GetLocation. We already have code for this use case in HTML that provides the location of the given device in latitude and longitude form. Since the Sidesync app transfer

the phone screen to the laptop, and the phone is running our interface, the location of the phone is transferred to the laptop.

Physical Data:
The physical data part of the project deals with the GetStatus and GetData use cases. It will also include the MoveDrone use case. Due to the hardware component of the project, this part of the project was in effect once the drone is in full operation. Specifically, this part of the project will depend on the use of a Raspberry Pi and the interface. The physical data component uses values from getLocation() to calculate the speed. It is also used to get the current battery levels of the device in operation.

Obstacles:
The Obstacles section of the project deals with the remainder of the use cases. The use case for the Obstacles section only includes CheckObstacles. Similar to the physical data component, this section was mainly in effect once the drone was working. In the final drone build, we decided to use two ultrasonic sensors on the drone to check for nearby obstacles in the left and right directions. They utilize the Raspberry Pi and Python to detect the obstacles around the drone and store the measured distances in a text file that will be read by the webpage and displayed.

History of Work

- Milestones:
  - *Drone Camera Transmission:* Be able to provide a reliable stream from the onboard phone camera to a mobile device set aside to mock the operator's control device.
    - Date of Completion: March 8th, 2019
  - *Hardware-Associated Tasks:* After all necessary hardware components arrive between March 1st-3rd, the construction of the drone frame to fit the needs of the project. This includes mounting the camera to the drone frame.
    - Date of Completion: April 21st, 2019
  - *Webpage Integration:* Collecting all relevant data and finalizing transmission/display of said data to the operator's webpage.
    - Date of Completion: March 22nd, 2019
  - *Sensors:* Managed to get the sensors incorporated with the Raspberry Pi and be able to record the distance from an object.
    - Date of Completion: April 21st, 2019

The milestones that have been completed so far and the planned milestone achievements so far are slightly different from each other. We initially experienced delays in receiving the hardware components on time so we could not meet up some of the expected milestones on the hardware side. We did manage to complete several of the other planned achievements on time. These achievements included getting the video feed from the mobile's camera and getting the webpage integration done on time.
The future plans for this project will be focused more on integrating everything around the drone and making sure that the various subsystems

can work in cohesion. All future work for this project is going to be focused on the drone itself and not on the systems that utilize the drone or that can be placed on the drone.

Key Accomplishments:

- Webpage Integration
- Camera Transmission
- Getting location and calculating the speed
- Getting the infrared camera
- Connecting the sensors onto the Raspberry Pi
- Writing frequent readings of sensors into a file
- AJAX connection from sensor data to the website

Breakdown of Responsibilities

- Project divisions:(all tasks that are in progress/to be completed)
  - Visual Data Processing:
    - Shantanu: Management of the main wireless network/communication of data. Also responsible for the bringing most of the hardware for the drone.
    - Abhishek: Webpage development/Data handling on operator-side
    - Krishna Mahadas: Onboard camera handling, transmission
  - Obstacle Management
    - Vishal: Managing sensor data, implementing detection/assoc. movement
  - Location Data
    - Avnish: Gathering onboard GPS data, transmission
  - Physical Drone Data
    - Krishna Tottempudi: Determining overall operational status from the collected data.
    - Sahana: Determining power levels/operational lifespan of drone real-time
    - Won Seok: Determining the strength of signal/connection to the operator

All other contributions to the project can be found in the individual contributions breakdown matrix on page 2.

References

1. “Drone Sense” https://www.dronesense.com/?gclid=EAIaIQobChMIqo-45_Gx4AIVwoCfCh2CbA0QEAAYASAAEgKMu_D_BwE
2. Byran, Cantfil. “ United States Coast Guard Search and Rescue Summary Statistics 1964 thru 2015.” https://www.dco.uscg.mil/Portals/9/CG-5R/SARfactsInfo/SAR%20Sum%20Stats%2064-16.pdf
3. Rhode, Steve.“DRONE SEARCH-AND-RESCUE STUDY REVEALS POTENTIAL, LIMITS” https://www.aopa.org/news-and-media/all-news/2018/october/01/drone-study-reveals-potential-and-limits
4. “Image 1” https://s.yimg.com/ny/api/res/1.2/2P8Y6UqlB8dKOiVIg9Rscg--~A/YXBwaWQ9aGlnaGxhbmRlcjtzbT0xO3c9ODAw/http://media.zenfs.com/en-US/homerun/digital_trends_973/8122e594705a009db372bf32720d9fe9
5. “Using a Raspberry Pi distance sensor (ultrasonic sensor HC-SR04).” https://tutorials-Raspberrypi.com/Raspberry-pi-ultrasonic-sensor-hc-sr04/
6. “Raspberry Pi Distance Sensor: How to set up the HC-SR04” https://pimylifeup.com/Raspberry-pi-distance-sensor/
7. “HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi” https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-Raspberry-pi
8. “Installing GPIO” https://gpiozero.readthedocs.io/en/stable/installing.html
9. “The Equations for Speed” https://quadstardrones.com/the-equations-for-speed/
10. “Implementing an in-browser camera” https://davidwalsh.name/browser-camera
11. “Tracking current location” https://www.w3schools.com/html/html5_geolocation.asp