TASK LOGBOOK

Arnabi Dutta
Software Recruit
24/A02/017
Mentor - Sahil Kumar
Project Submission Github Repo

The theme for this task is Search and Rescue. A fire has broken out in a civilian area and your job is to gather information about the location of houses and buildings in the area. Your UAV is collecting images of the search area that look like the sample image given below.

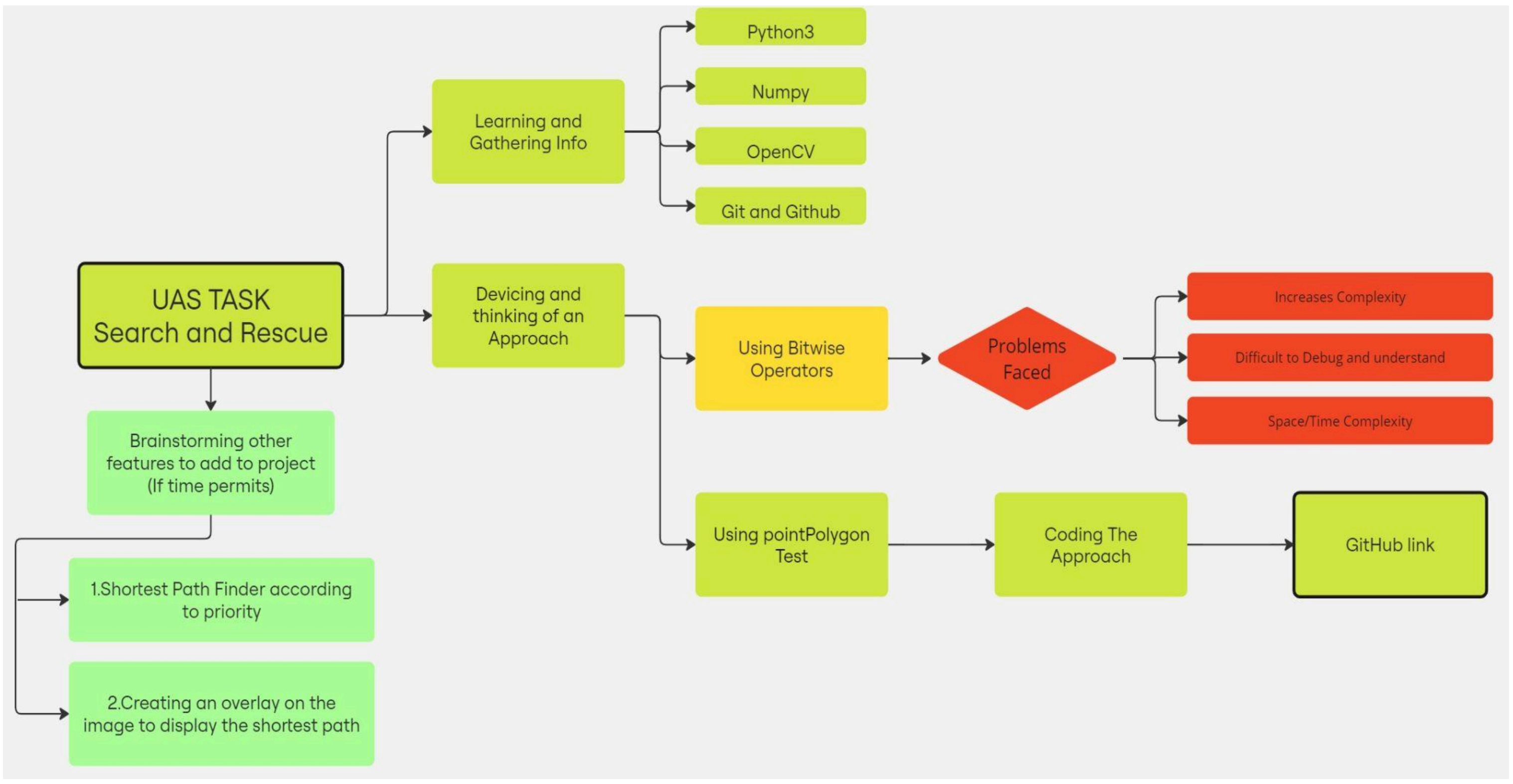
Information about the input image
- The brown area is burnt grass
- The green area is unburnt (green) grass
- The blue and red triangles are houses
- Each house colour has an associated priority
- Blue house have a priority of 2
- Red houses have a priority of 1

The task for you is to return the following information from the image Input A list of 10 images, similar to the sample image provided above

---

To utilise my time efficiently and bring forth the optimal possible productivity, I divided the task into subtasks and created a stepwise flowchart to visualise my project and allocate proper time.

Flow of the Project


Python3
Numpy
OpenCV
Git and Github
Learning and Gathering Info
UAS TASK Search and Rescue
Devicing and thinking of an Approach
Using Bitwise Operators
Problems Faced
Increases Complexity
Difficult to Debug and understand
Space/Time Complexity
Brainstorming other features to add to project (If time permits)
Using pointPolygon Test
Coding The Approach
GitHub link
1.Shortest Path Finder according to priority
2.Creating an overlay on the image to display the shortest path

1. STEP 1 [Learning Libraries and Gathering Information]:

- ☑ https://www.python.org/downloads/release/python-378/
- ☑ basics of NumPy
- ☑ basics of OpenCV (Open Source Computer Vision Library)
- ☑ Basics of Git and github

2. STEP 2 [Devicing Approaches and selecting the best one]:

The steps I took:

1. Loading and Mask Creation:

I began by loading an image and converting it from BGR to HSV color space for easier detection of color regions.
Then, I created masks for four color ranges: red, blue, green, and brown. These colors represent different areas in the image (houses, green regions, burnt regions, etc.).

2. Detecting and Marking Houses:

Using cv.findContours(), I identified contours in the red and blue masks, representing houses.
I filtered the contours based on their area and shape (focusing on triangular ones) and drew them on the image. I also calculated the centroids of each detected house and marked them with circles.

Now I came across a dilemma on how to detect and classify Houses by Region.
After researching and discussing with a few friends, I found and tried the following two approaches:

a. Using Bitwise Operators

b. Using Point polygon Test

a. Using Bitwise Operators

After trying out this approach, I came across few issues and reasons why using bitwise operators for task search and rescue might not be the best approach in my case:

1. Complexity of Mask Creation:

Manually creating separate masks for each color and creating more masks of their combinations (red, blue, green, brown, redongreen, redonbrown, blueongreen, blueonbrown) added complexity to the code. Handling multiple masks increased both the space and time complexity, and made the overall implementation harder to manage and understand. The more masks I created, the more convoluted and confusing it became.
I realised this is not the right way to apply this.

2. Space and Time Complexity:
My thought process was Bitwise operators could introduce additional memory usage by storing large intermediary masks and operations. Since there are already several mask creation steps, increasing the operations would further slow down the process.

3. A major issue I faced is that I was making errors in applying bitwise operators. Since the task involved multiple contours and region overlaps, applying bitwise operations manually increases the chances of errors in combining or subtracting masks. This resulted in incorrect house detections or misclassifications of regions (green vs. brown), which lead to failure of output and inaccurate results.

I think I can solve this issue by reconsidering how I handle regions and reduce redundant steps.
Will discuss it with my mentor.

b. Using point polygon test:

Trying to think of a new approach and scratching my brain till 6 am at night (or morning lol, sleepless nights ftw), I stumbled the idea that what if
I can check if centroids of each triangle lie on a particular contour, after finding external contours of both the burnt grass and green grass.
Thus classifying if a certain coloured triangle lies on a certain contour.

To find an execution to my idea, I searched the internet for possible solutions and read OpenCV docs. Which is when I found out about the point polygon test.

Point Polygon Test
Performs a point-in-contour test.

The function determines whether the point is inside a contour, outside, or lies on an edge (or coincides with a vertex). It returns positive (inside), negative (outside), or zero (on an edge) value, correspondingly. When measureDist=false , the return value is +1, -1, and 0, respectively. Otherwise, the return value is a signed distance between the point and the nearest contour edge.
Parameters
- contour: Input contour.
- pt: Point tested against the contour.
- measureDist: If true, the function estimates the signed distance from the point to the nearest contour edge. Otherwise, the function only checks if the point is inside a contour or not.

src: Point Polygon Test OpenCV Docs

This was the perfect solution to my problem.

3. Classifying Houses by Region:

https://github.com/ArnabiDutta/UAS-Task-R2-2024/blob/master/pointpolytest.py

1. Found Contours:
I extracted the contours from the edge-detected image using cv.findContours(). I thought this would help me isolate individual shapes, such as triangles and external contours.

2. Detected Triangles Based on Contours:
I approximated each contour to check if it had 3 points (indicating a triangle) using cv.approxPolyDP(). I also filtered out small areas by checking if the contour area was above 500. I did this because small contours might not represent meaningful triangles.

3. Drew Triangles and Found Centroids:
For each triangle, I drew the contour on the original image using cv.drawContours() and calculated the centroid using cv.moments(). I thought it was important to mark the center for further analysis, so I added a white circle at the centroid.

4. Checked if Triangle's Center is in a Region:
I created a function centre_in_region() to check if the centroid of a triangle lies inside an external contour using cv.pointPolygonTest().
This finally helped in determining whether the triangle is located within a specific area - burnt grass or green grass.

4. Priority and Rescue Ratio Calculation:

For each image, I tallied the houses based on their location (green or burnt regions). The rescue priority was calculated based on a weight system, where red houses had a priority of 1 and blue houses had a priority of 2.

I computed the "rescue ratio" as the ratio of houses on burnt ground to those on green ground.

5. Generating Rescue Sequence:

I processed 10 images, calculating the rescue ratios for each.

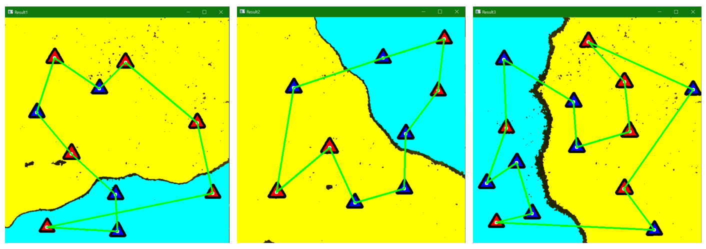
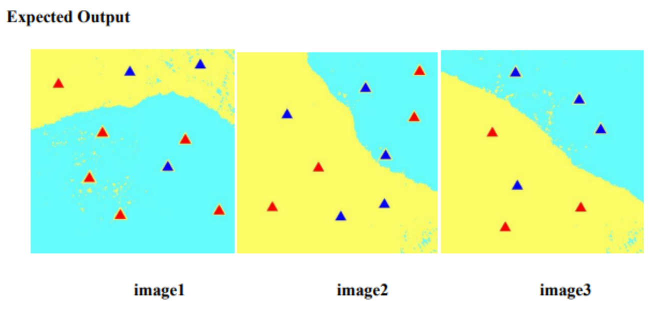
Finally, I created a dictionary mapping each image to its rescue ratio and sorted the images based on these ratios to determine the priority sequence for rescue operations.

STEP 3 [Coding the project and debugging errors]:

https://github.com/ArnabiDutta/UAS-Task-R2-2024
I coded the project in python and tested it using the provided test cases.

MY OUTPUT	EXPECTED OUTPUT

no_houses= [[6, 4], [6, 4], [7, 6], [3, 4], [6, 3], [6, 4], [2, 6], [7, 5], [6, 4], [6, 4], [6, 4]]
 	
priority_houses = [[8, 6], [9, 6], [10, 10], [4, 7], [9, 4], [10, 5], [3, 10], [10, 8], [8, 5], [8, 5], [8, 6]]
 	
Priority Ratio = [1.3333333333333333, 1.5, 1.0, 0.5714285714285714, 2.25, 2.0, 0.3, 1.25, 1.6, 1.6, 1.3333333333333333]
 	
sequence in terms of rescue ratio in descending order= ['image5', 'image6', 'image10', 'image9', 'image2', 'image11', 'image1', 'image8', 'image3', 'image4', 'image7']
	n_houses = [[3,6],[5,4],[4,3]]
	
	priority_houses = [[5,7],[8,6],[5,6]]
	
	priority_ratio = [0.71,1.33, 0.83]
	
	image_by_rescue _ratio = [image2, image3, image1]

STEP 4 [Taking it up a Notch]:

It is my habit to think and go above and beyond every task. After brainstorming ideas with friends that could elevate the project, the one that feels the best to implement is Route Planning.
Route planning involves finding the shortest path according to the priority of houses and displaying them on an overly over our image.
UPDATE:
This project is a case similar to the travelling salesman problem. There are multiple approaches to this with different kinds of approximations. Looking into it and trying to figure it out, I found using the Nearest Neighbour Heuristic to be the most approachable method as per my current level of understanding.

Github file to my approach: Route Planning Basic

I would like to discuss a better approach, fix my errors and develop this feature of the project a bit more with my mentor.

FUTURE IMPROVEMENTS I WISH TO ADD WITH THE HELP OF MY MENTOR:

1. This route planning does not depend upon the priority of the house or its position on burnt grass or green grass. I want to devise or implement an algorithm that plans route eradicating these flaws.
2. A better heuristic to improve precision for large data sets.
3. A better way of displaying the output images.
4. Any other cool ideas as recommended by mentor.

STEP 5 [Discussing with Mentor]:

Discussing my project with my mentor, getting their valuable insight on it and implementing them. Discussing errors and mistakes I was making, ways to do the task more efficiently.