Section 4

Technical implementation

In this section the technical implementation will be analyzed, with particular attention to the analysis and the explanation of the MATLAB code used for the creation of our segmentation program.

To run our program fourteen MATLAB files are necessary. There is one principal file called "ElicioDome_Progetto_Oil_Spill_Detection" that works as a "main" (like for example in Java programming language) and there are thirteen more files containing all the functions used to implement the various segmentation algorithms seen in the previous sections of this report. The choice of dividing the complete code in multiple files has been done to improve code readability and code maintenance by having a sort of Object Oriented Programming approach.

In the following there will be different paragraphs with the same titles as the MATLAB files used, to analyze and explain in detail different part of used code. Sometimes in the various snippets that will follow there will be the sign " **[…]** ", meaning that in the original file there is more code which functionalities are very easy to understand or which have already been explained. However, most part of the code has comments, so it is really easy to understand some parts that won't be analyzed here.

ElicioDome_Progetto_Oil_Spill_Detection.m

This file works as the "main" for the entire program. Here are contained all the commands that permit the user to choose what type of image he wants to analyze (only sea or sea and land images), to load the chosen image from specific paths, to choose the segmentation method to use and to modify or not some of the parameters for every method to improve its performance.

When the program is run, the user can make his first choice, having three possibilities: analyzing sea, sea and land images or exit the program causing the stop of the MATLAB program thanking the user for using it. If the user enters an input different from the three specified numbers (0 - 1 - 2), the program stops and the user have to re-run it again (*Figure23*).

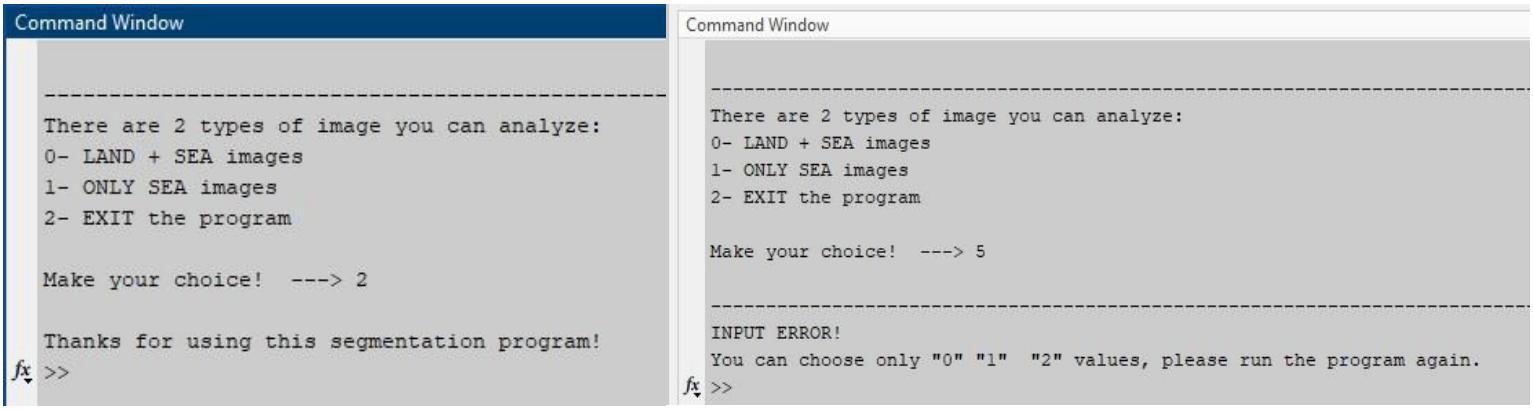

Figure 23: Example of running MATLAB program with user choosing (a) to exit and (b) input a wrong value

Instead, if the user inputs 0 or 1, he decides the image type to analyze, so the program displays the available number of that type of images permitting the user to choose the image he wants. Also here there are some controls to make the user inputs only correct values (for example if there are 219 land images, he can't choose the 220th image) (*Figure24*).

```
1.  while choice~=2
2.
3.  %Set the path to load images and labels
4.  […]
5.  imagesDir = 'C:\Users\Domenico[…]
6.  landImagesDir='C:\Users\Domenico[…]
7.  labelsDir = 'C:\Users\Domenico[…]
8.  landLabelsImagesDir='C:\Users\Domenico[…]
9.  […]
10.
11. if choice==0
12.     close all,tic
13.     flag=0;
14.     fprintf("\nThere are 219 images of this type, […]
15.     choice2=input("Make your choice!  ---> ");
16.
17.     if choice2<1 || choice2>=220
18.         fprintf("\nWRONG NUMBER!\nThere are 219 images of this type, please enter […]
19.     […]
20.     end
21. elseif choice==1
22.     close all,tic
23.     flag=1;
24.     fprintf("\nThere are 783 images of this type, please enter […]
25.     if choice2<1 || choice2>=784
26.         fprintf("\nWRONG NUMBER!\nThere are 783 images of this type, please […]
27.     […]
28.     end
29. elseif choice==2
30.     close all
31.     fprintf("\nThanks for using this segmentation program!\n")
32.     return
33. else
34.     close all
35.     fprintf("\nINPUT ERROR!\nYou can choose only ""0"" ""1""  ""2"" values, […]
36.     return
37. end
```

Figure 24: A part of the code that permits the user to make his first choice

If the user input is 0 or 1 he decides to analyze an image, so the desired image and its respective ground truth are loaded in some variables called *"I"* and *"groundTruth"*. Then the original image is transformed in a grayscale one that will be used with the available segmentation methods. Like the first choice, the user can make others: for example, he can choose which segmentation method to use. As said in the previous sections of this work, there are 6 methods available to segment images containing only sea and oil spills and there are 2 methods to segment images containing sea and land. Also here there are some controls to make the user inputs only correct values for his choices (Figure25).

```
%If flag==true you're considering ONLY SEA images, otherwise LAND+SEA images
if flag==true
    my_img(fileidx).img=imread([…])
    my_label(fileidx).img=imread([…]
    I=im2double(my_img(fileidx).img);
    label=my_label(fileidx).img;
else
    %Land & oil spills images
    my_LandImg(fileidx).img=imread([…]);
    my_LandLabel(fileidx).img=imread([…]));
    I=im2double(my_LandImg(fileidx).img);
    label=my_LandLabel(fileidx).img;
end
%Load and show image and label to use
groundTruth1=im2double(label);
   […]
figure('WindowState', 'maximized');
subplot(1,4,[1 2]),imshow(I),title('Original Image');
subplot(1,4,[3 4]),imshow(groundTruth1),title('Ground truth');
%RGB to gray conversion
grayImg=rgb2gray(I);

if choice==1
    fprintf("\n\nThere are different segmentation method you can use […]
    fprintf("1 - MANUAL THRESHOLDING SEGM.\n")
    fprintf("2 - AUTOMATIC THRESHOLDING SEGM.\n")
    fprintf("3 - LOCAL ADAPTIVE THRESHOLDING SEGM.\n")
    fprintf("4 - SUPERPIXEL + OTSU THRESHOLDING SEGM.\n")
    fprintf("5 - FUZZY LOGIC EDGE DETECTION & SEGM.\n")
    fprintf("6 - K-MEANS CLUSTERING & SEGM.\n")

    choice3=input("Make your choice!  ---> ");
    switch choice3 […]

% If user chooses images with SEA & LAND:
elseif choice==0
    fprintf("\n\nThere are different segmentation method you can use […]
    fprintf("1 - AUTOMATIC THRESHOLDING SEGM.\n")
    fprintf("2 - K-MEANS CLUSTERING & SEGM.\n")

    choice3=input("Make your choice!  ---> ");
    switch choice3 […]

end
```

Figure 25: A part of code that permits the user to choose the image to analyze and one segmentation method

After the user has made his choice on the segmentation method to use, a **switch** command is used to say MATLAB how to continue running the program. For instance, if the user chooses the manual thresholding segmentation method by giving in input the number 1, MATLAB executes the switch case number 1. All switch cases behave similarly, so only the first one will be analyzed here.

First there is the instantiation of some variables to some default values. These variables are the parameters that the user could later modify to improve the chosen segmentation method performances. At this point there is the call to the function that implements the chosen segmentation algorithm that takes as input parameters the instantiated variables cited before. After the segmentation has been done and its results are shown, the program asks the user if he wants to modify some parameters by manually changing some values to see how these changes affect segmentation's performance. Instead, if the user is satisfied with the results, he may decide to go ahead and start segmenting a new image, even by another method (*Figure26*) (*Figure27*).

```
1.      fprintf("\n\nThere are different segmentation method you can use […]
2.      fprintf("1 - MANUAL THRESHOLDING SEGM.\n")
3.      fprintf("2 - AUTOMATIC THRESHOLDING SEGM.\n")    […]
4.
5.      choice3=input("Make your choice!  ---> ");
6.      switch choice3
7.          case 1         % --------------- MANUAL THRESHOLDING ----------------------- %
8.              %Default value for thresholds and minimum area of the blobs
9.              addThreshval=0.06;
10.             areaToConsider=45;
11.
12. [BWmanual,max_value]=manual_thresholding(grayImg,groundTruth1,addThreshval,areaToConsider)
13.             figure('WindowState', 'maximized');
14.             segmentation_evaluation(groundTruth,BWmanual);
15.             %User can manually improve segmentation results by changing some parameters
16.             fprintf("\nWould you like to improve segmentation results? […]
17.             yesOrNo=input("Press ""y"" or ""n""  --->",'s');
18.             while strcmpi(yesOrNo,'y')
19.                 close all
20.                 %Maximum threshold value at the last analysis done:
21.                 fprintf('\n\nMaximum threshold value NOW: %f\n',max_value)
22.                 %User can modify threshold to recognize oil spill
23.                 fprintf("\nYou can adjust the segmentation results by increasing or […]
24.                 addThreshval=input("-- THRESHOLD VALUE -- \nMake your choice […]
25.                 if isempty(addThreshval)
26.                     addThreshval=0;
27.                 end
28.
    [BWmanual,max_value]=manual_thresholding(grayImg,groundTruth1,max_value+addThreshval[…]
29.                 figure('WindowState', 'maximized');
30.                 segmentation_evaluation(groundTruth,BWmanual);
31.
32.                 fprintf("\nWould you like to improve segmentation results? [y/n]")
33.                 yesOrNo=input("Press ""y"" or ""n""  --->",'s');
34.
35.             end
```

Figure 26: A part of code that runs manual thresh. Segmentation and permits the user to modify some parameters

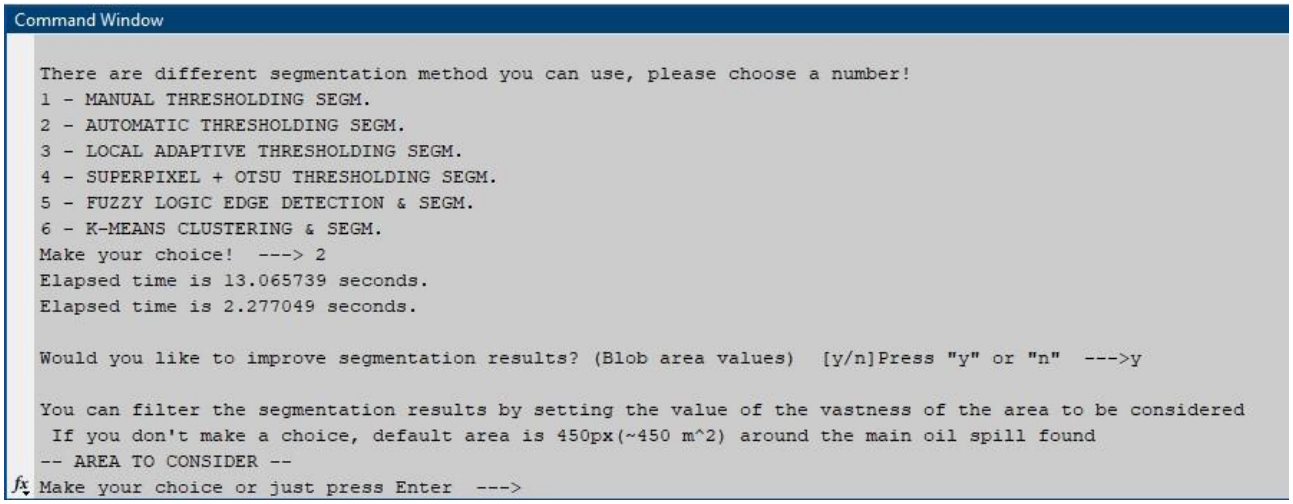

*Figure 27: The program asks the user **if** he wants to modify some parameters for segmentation and **how** to modify them*

Now that the code of the "main" program has been analyzed and explained, we will proceed on the analysis of the various used functions. Sometimes in the various snippets that will follow there will be the sign " **[…]** ", meaning that in the original file there is more code which functionalities are very easy to understand or which have been already explained. However, most part of the code has comments, so it is really easy to understand some parts that won't be analyzed here.

manual_thresholding.m

This function implements the manual thresholding segmentation method (4). It takes five inputs: the original grayscale image, the ground truth labelled image, the value the user wants to add or subtract to the previous computed threshold, the maximum area to consider a blob as an oil spill and the window size for the median filter.

Initially these input parameters have default values:

- Threshold value to add = 0.06 (experimentally verified as the best value)
- Area to consider a blob as oil spill = 45 (means 45 px^2 ≈ 450 m^2 (1))
- Median Filter window size = 3x3

After an operation of noise reduction using a median filter (**medfilt2** function), an operation of contrast enhancement is done (**histeq** function). Later, the modified image pops up to the user who clicks on some black points of the oil spill with **impixel** function to permit the program to memorize intensity values of the clicked pixels to find the threshold to recognize the oil spills. At this point the minimum and maximum values among the chosen pixels are computed to have a range of values to

use as threshold for segmentation (4). Some controls are applied in case user select only one point on the image making impossible to have both a minimum and maximum value, or to handle the case when the two values are equal. User can improve segmentation performance by adding or subtracting something to modify the threshold values used to recognize the oil spill, as the binary mask is created by selecting all the pixels that have an intensity value between the minimum and the maximum computed values. At the end some morphological operations are computed by using **imopen** and **imfill** functions to eliminate white isolated dots and to fill remaining blank areas.

After having created the oil spills binary mask, with the **bwlabel** function the image is divided in labels that will be used to extract all the features of the resulting blobs.

With **bwconncomp** function all the connected components in the image are counted, and with **regionprops** all blob's properties are extracted. In this work only area of the blob has been considered to make a discrimination between a real oil spill and other pixels that have the same gray intensity values. After having found all the blobs with an area greater than a set value, the final black and white mask is computed by using the **ismember** function that finds all the blobs with a specified index inside the matrix created by **labelmatrix** function containing all the labels counted before using **bwconncomp**.

At the end the resulting mask and the original image with overlapped mask are shown by calling the **visualizeImages** function that takes as input parameters the original image, the final mask computed, the labelled ground truth image and a string containing the segmentation method's name.

Also here **tic** and **toc** commands are used to compute the execution times.

```
1.  function [BW,max_value] = manual_thresholding(img,ground,addValue,maxArea,medianFilter)
2.  tic
3.
4.  grayImg=medfilt2(img,[medianFilter medianFilter]);
5.  […]
6.  eqImg=histeq(grayImg,50);
7.
8.  %Manually choose pixels to use for thresholding:
9.  pixel_values=impixel(eqImg);
10.
11. min_value=min(pixel_values(:,1,1));
12. max_value=max(pixel_values(:,1,1));
13.
14. if (min_value==0 && max_value==0) || (min_value==max_value)
15.     max_value=min_value+addValue;
16. elseif max_value~=0
17.     max_value=max_value+addValue;
18. end
```

```
19.
20.
21. %Binary mask creation:
22. mask=eqImg>=min_value & eqImg<=max_value;
23.
24. %% Opening & Closing morphological operations to eliminate isolated white dots:
25. se=strel('disk',1);
26. mask= imopen(mask,se);
27.
28. %% "imfill" to fill blank areas
29. filledMask = imfill(mask, 'holes');
30.
31. %% SPOT FEATURES EXTRACTIONS:
32. %Label connected components in 2-D binary image (useful to use regionprops later)
33. [labeledImage, ~] = bwlabel(filledMask, 4);
34.
35. % Get all the blob properties.
36. cc=bwconncomp(filledMask);
37. stats = regionprops(labeledImage, img, 'all');
38.
39. %Find blobs with an area >45pixels^2 ~450m^2 in real world
40. idx=find([stats.Area]>maxArea);
41. BW=ismember(labelmatrix(cc),idx);
42.
43. %% Mask coloring and overlay on the original image
44. visualizeImages(img,BW,ground,'MANUAL THRESHOLDING');
45.
46. toc
47. end
```

Figure 28: manual_thresholding.m function

automatic_threshold.m

This function implements the automatic thresholding segmentation method (4). It is similar to that seen before. It takes four inputs: the original grayscale image, the ground truth labelled image, the maximum area to consider a blob as an oil spill and the window size for the median filter.

Initially these input parameters have default values:

- Area to consider a blob as oil spill = 45 (means 45 px^2 ≈ 450 m^2 (1))
- Median Filter window size = 3x3

After the same smoothing and image enhancement operations already seen before, the histogram of resulting image is analyzed. By using the **imhist** function, the bin locations and their counts are returned and used to determine the thresholds useful to recognize oil spills (4). At the end, there will be a range of values going from a minimum to a maximum value that will form our maximum and minimum thresholds to identify spills. These two values are computed from the histogram, taking the upper threshold T_{max} computed according to the accumulation gray intensity values of pixel histogram, while the lower threshold value T_{min} corresponds to the lowest accumulation point of pixel value in the histogram (4). Based on these values, the binary mask is computed using the

imbinarize function and then the same morphological operations already seen before are computed. Also the dark spot features extraction operation and the visualization of the resulting images has been done in the same way, in fact in *Figure29* only a part of code will be shown.

```
function [maschera,BW] = automatic_threshold(img,ground,maxArea,medFilter)
tic
%% Image Smoothing operation with median filter:
[…]
%% Contrast enhancement (Histogram equalization):
[…]
%% AUTOMATIC THRESHOLDING:
[counts,binLocations]=imhist(eqImg);
A=[counts,binLocations];
%% Finding two threshold values:
valueMax= max(max([counts,binLocations]));
valueMin= min(min([counts,binLocations]));
[row column]= find(A==valueMax,1,'first');
%Maximum threshold value:
Tmax=A(row,2);
[r c]= find(A==valueMin,1,'first');
%Minimum threshold value:
Tmin=A(r,2);

%If Tmin=0 means that there are no pixel with that value, so you have to use only Tmax
if Tmin==0
    mask=imbinarize(eqImg,Tmax);
else
    mask=imbinarize(eqImg,Tmin:Tmax);
end
mask=~mask;
%% Opening & Closing morphological operations to eliminate isolated white dots:
[…]
 %% SPOT FEATURES EXTRACTIONS:
[…]
%% Mask coloring and overlay on the original image
[…]
toc
end
```

Figure 29: automatic_threshold.m function

local_threshold.m

This function implements the local adaptive thresholding segmentation method (9). It takes five inputs: the original grayscale image, the ground truth labelled image, the value the user wants to use as window size for the Wiener noise removal filter, the threshold to use for the unsharpening mask and the window size for the Gaussian filter.

Initially these input parameters have default values:

- Wiener Filter window size = 5x5 (experimentally verified as good value)
- Unsharpening threshold value = 0.5 (experimentally verified as good value)
- Gaussian Filter window size = 5x5

Image enhancement operations on the original grayscale image consists in noise removal trough Wiener filter by using the **wiener2** function and enhancing the contrast using an unsharping mask with the **imsharpen** function. Later, a gaussian filter is applied on the resulting image with the **imgaussfilt** function. To improve segmentation results, user can modify parameters related to these three functions. He can modify window sizes of Wiener's and Gaussian filters and the unsharping mask's threshold that represents the minimum contrast required for a pixel to be considered an edge pixel, allowing sharpening only in some regions. The local adaptive threshold has been computed using the **adaptthresh** function, using as statistics to compute the threshold the gaussian weighted mean in pixel neighbourood by setting the parameter '*Statistic*' as *'gaussian'*. Other parameters used are: *'sensitivity'* (set to *1*) of the algorithm to classify a pixel as foreground based on the threshold value computed, '*NeighborhoodSize'* set at *41*, indicating the size of neighborhood used to compute local statistic around each pixel, and *'ForegroundPolarity'* set at *'bright'* used to determine which pixels are considered foreground pixels (setting 'bright' means that our foreground is brighter than the background). All these parameter's values have been experimentally verified as good values for our goal. At the end, some morphological operations already seen, have been computed to have a better segmented binary mask. Results are always shown using the afore mentioned **visualizeImages** function (*Figure30*).

```
function [filledMask]= local_threshold(img,ground,noiseFilter,SharpThresh,GaussianFilter)

tic
%% IMAGE ENHANCEMENT
noiseRemoved=wiener2(img,[noiseFilter noiseFilter]);
sharpenedImg=imsharpen(noiseRemoved,'Radius',1.5,'Amount',1.5,'Threshold',SharpThresh);
%% Gaussian Filtering
Iblur=imgaussfilt(sharpenedImg,'FilterSize',GaussianFilter);

%% LOCAL THRESHOLDING:
tresh=adaptthresh(Iblur,1,'NeighborhoodSize',41,...
    'ForegroundPolarity','bright','Statistic', 'gaussian');
mask=imbinarize(Iblur,tresh);
mask=~mask;

%% Opening & Closing morphological operation to eliminate isolated white dots:
[…]

%% Mask coloring and overlay on the original image
[…]

toc
end
```

Figure 30: local_threshold.m function

superpixel.m

This function implements the superpixel approach for image segmentation (7). It takes four inputs: the original grayscale image, the ground truth labelled image, the value the user wants to use as number of superpixels in which to divide the original image and the value of the distance from the greatest oil spill the user wants to use to recognize other dark points as oil spills to improve the segmentation.

Initially these input parameters have default values:

- Superpixels number to divide original image = 25.000 (experimentally verified as good value)
- Distance from greatest spill to consider a dark point an oil spill = 450px (means ≈ 450 m (1))

The original grayscale image is divided in a number "*spNumber*" of superpixels using the **superpixels** function. Using the function **label2idx** it has been possible to assign an index to every superpixel in the image and so the mean intensity value for every of them has been computed. With the **graythresh** function, the Otsu's threshold is computed for the resulting image and then some adjustments, experimentally verified as good, are done. With the **imbinarize** function the binary mask is computed and thanks to **regionprops** all dark spot's features are extracted. From all the resulting dark spots, only the one with the largest area is considered (using **bwarefilt** function) and from its centroid the two main blobs' (if they exists) distances are computed (in the overall image only three main blobs are considered because it has been seen that this value gives good segmentation results at the end). Only those blobs which have a shorter distance than the default value (450px) or than the user's chosen distance, are considered as oil spills in the final mask. At the end always the function **visualizeImages** has been used to show final results (*Figure31*).

```
1.  function [BW2]= superpixel(img,ground,spNumber, minDist)
2.
3.  tic
4.  %Superpixel image creation and show: 25.000 superpixel are created:
5.  [L,NumLabels]=superpixels(img,spNumber);
6.  BW = boundarymask(L);
7.  outputImage = zeros(size(img),'like',img);
8.  idx = label2idx(L);
9.  %The mean intensity value is computed for each superpixel(to apply Otsu method)
10. for labelVal = 1:NumLabels
11.     Idx = idx{labelVal};
12.     outputImage(Idx) = mean(img(Idx));
13. end
14. […]
15. %Otsu thresholding applied:
16. level=graythresh(outputImage);
17. if level>0.5
18.     level=level-0.25;
```

```
else
    level=level-0.15;
end
%Image mask creation using the computed Otsu threshold:
binary=imbinarize(outputImage,level);
binary=~binary;

%% DARK SPOT FEATURE EXTRACTION
%DISTANCE COMPUTATION FROM THE CENTRAL OIL SPILL
props = regionprops(binary, 'all');
allAreas=[props.Area];
%Sort resulting blobs by area and choose only 3 blobs with maximum area
sortedAreas=sort(allAreas);
maxAreaBlob=max(sortedAreas);
if size(sortedAreas,2)>=3
    low=sortedAreas(size(sortedAreas,2)-2);
    binary=bwareafilt(binary,[low maxAreaBlob]);
else
    low=0;
    binary=bwareafilt(binary,[low maxAreaBlob]);
end
%Now you have to find the Centroid of the spill with the maxArea:
blobIndex=find(allAreas==maxAreaBlob);
blobIndexCentroid=props(blobIndex).Centroid;

%Centroid computation for all the remaining blobs
measurements = regionprops(binary, 'Centroid');
allCentroids=[measurements.Centroid];
centroidX=allCentroids(1:2:end)';
centroidY=allCentroids(2:2:end)';
centr=[centroidX centroidY];

%Compute distances between the principal oil spill and all other blobs
%Only spills with a distance <450px (~450mt) from the principal spill are shown
distanceMatrix = pdist2(blobIndexCentroid,centr);
%Default value of minDist=450;
binary=bwlabel(binary);
ind=find(distanceMatrix<=minDist);

BW2=ismember(binary,ind);
%% Mask coloring and overlay on the original image
[…]
toc
end
```

Figure 31: superpixel.m function

fuzzy_edgeDetect.m

This function implements the fuzzy edge detection approach for image segmentation (3). It takes three inputs: the original grayscale image, the ground truth labelled image, the value the user wants to use for Lee filter's window size.

Initially this input parameters have default values:

- Lee Filter window size = 5

Image enhancement operations done on the original image regards the use of Lee's Filter to eliminate speckles, done using the **lee_filter** function (15), and the use of **histeq** function to enhance the contrast. Then, in the following lines of the code the Sobel operator is used to compute edge detection together with a FIS (Fuzzy Inference System) and membership function. Defining the FIS, the values of the triangular membership function, needed to detect or not an edge, have been set manually after lots of trials, experimentally seeing that they on average give good results. This time, the program doesn't ask the user to modify these values because it would be very stressful to manually modify eight more parameters, but these values can be easily modified, changing them directly in the code.

Then, a grayscale image with all the highlighted found edges is created. The edges have a gray intensity value of 0.8, that is used to create the binary mask using the **imbinarize** function. At the end, some morphological operations already seen, are computed to have a better segmented mask. Results are always shown using the afore mentioned **visualizeImages** function (*Figure32*).

```
1.  function [BW]=fuzzy_edgeDetect(img,ground,filterSize)
2.  %% IMAGE ENHANCEMENT
3.  leeFiltered=lee_filter(img,filterSize);    %Default window size used: 5
4.  eqImg=histeq(leeFiltered);
5.  %Calculate the image gradient along the x-axis and y-axis, because The fuzzy logic edge-
    %detection algorithm relies on the image gradient to locate breaks in uniform regions.
6.  %To do this, Sobel operator has been used:
7.  Gx = [1 0 -1; 2 0 -2; 1 0 -1];
8.  Gy = Gx';
9.  Ix = conv2(eqImg,Gx,'same'); %convolution of two 2D matrices
10. Iy = conv2(eqImg,Gy,'same');
11. %% Definition of a FIS(Fuzzy Inference System) for Edge Detection
12. edgeFIS = mamfis('Name','edgeDetection');
13. %Specification of image gradient, Ix and Iy, as the input for edgeFIS:
14. edgeFIS = addInput(edgeFIS,[-4 4],'Name','Ix');
15. edgeFIS = addInput(edgeFIS,[-4 4],'Name','Iy');
16. […]
17. %Triangular membership function parameters for Iout
18. wa = 0.4;           %start of the triangle
19. wb = 1;            %peak of the triangle
20. […]
21. %% FIS evaluation phase:
22. Ieval = zeros(size(eqImg));
23. for ii = 1:size(eqImg,1)
24. […]
25. end
26. […]
27. %% Binary Mask creation from the resulting edges
28. T=0.8;
29. mask=imbinarize(Ieval,T);
30. %% Opening & Closing morphological operations
31. […]
32. %% Mask coloring and overlay on the original image
33. […]
34. end
```

Figure 32: fuzzy_edgeDetect.m function

kmeansSegment.m

This function implements the k-means clustering approach for image segmentation (5). It takes five inputs: the original grayscale image, the ground truth labelled image, the window size for Gaussian filter, the value for the number of clusters the user wants to use for his segmentation and the number of blobs he wants to extract from the cluster that probably contains the oil spill to create the final segmentation mask.

Initially this input parameters have default values:

- Gaussian filter window size = 3
- Number of clusters to use to segment the image = 5 (experimentally verified as good value)
- Number of blobs to extract from the "probably" oil spill cluster = 1

Image enhancement operations on the original grayscale image consists in image smoothing trough the **imgaussfilt** function, where the user can also modify the window size used for the Gaussian filter. Then, k-means clustering algorithm is computed using the **kmeans** function. Experimentally has been seen that the algorithm works well by passing to the function some parameters like: '*Replicates*' set at '*5*' and '*MaxIter*' set at '*350*'. *Replicates* stands for the number of times to repeat clustering using new initial cluster centroid positions, while *MaxIter* stands for the maximum number of iterations to compute to reach a solution.

At the end, the centroids of all clusters will be the mean gray intensity value of a specific class, so to identify the oil spill in the image, we need to find the darkest class. This is done using the **min** function, while the extraction of the resulting blobs is done through the **bwareafilter** function that extract a certain number of blobs told by the user. Results are always shown using the afore mentioned **visualizeImages** function (*Figure33*).

```
1.  function [maschera,BW]=kmeansSegment(img,ground,filter,numClusters,numBlobs)
2.
3.  tic
4.  %% IMAGE ENHANCEMENT
5.  grayImg=imgaussfilt(img,'FilterSize',filter);
6.
7.  %% K-MEANS CLUSTERING
8.  [rows, columns] = size(grayImg);
9.  numberOfClusters = numClusters;
10.
11. % Convert to column vector
12. grayLevels = double(grayImg(:));
13. [clusterIndexes, clusterCenters] = kmeans(grayLevels, numberOfClusters,'Replicates', 3,...
14.     'MaxIter',350);
15. labeledImage = reshape(clusterIndexes, rows, columns);
16.
17. […]
```

```
18.
19. %% OIL SPILL DETECTION PHASE:
20. % We will assume the oil spill is the darkest class (between the clusters)
21. % The cluster center will be the mean gray level of the class.. so we now try to find the
    darkest class
22. [minValue, indexOfMaxValue] = min(clusterCenters);
23. % Get pixels that are labeled as the oil spill
24. oilSpill = labeledImage == indexOfMaxValue;
25. % Extract the largest blobs;
26. blobsToExtract=numBlobs;
27. oilSpill=bwareafilt(oilSpill, blobsToExtract);
28. % Fill holes.
29. oilSpill=imfill(oilSpill, 'holes');
30. %% Mask coloring and overlay on the original image
31. […]
32. toc
33. end
```

Figure 33: kmeansSegment.m function

land_mask.m

This function implements the automatic threshold approach for image segmentation (6) when the user wants to analyze images containing both sea and lands. It takes five inputs: the original grayscale image, the ground truth labelled image, the window size for Wiener filter, the value the user wants to use for the minimum area of a dark spot to consider it an oil spill and the value the user wants to use as a threshold to recognize land in the image.

Initially this input parameters have default values:

- Wiener filter window size = 5x5
- Area to consider a blob as oil spill = 45 (means 45 px^2 ≈ 450 m^2 (1))
- Threshold value to recognize land = 0.5

Image enhancement operations consist in using a Wiener filter with the **wiener2** function, an unsharpening mask with the **imsharpen** function and a Gaussian filter with the **imgaussfilt** function.

On the resulting image, land mask is computed considering all the pixels that have an intensity value greater than the fixed threshold. To have a better final mask the usual morphological operations are computed using **imclose** and **imopen** functions.

To compute the oil spill segmentation the **automatic_threshold_for_land** function is called. This last function is almost the same as the **automatic_threshold** function already explained above, but for some differences useful to show together land mask computed before and oil spill mask.

Final results are shown by calling the **visualizeImages_for_land** function. Also this function is almost the same as the already used **visualizeImages** function, but for some modification useful to show together in the same image both land and oil spills' masks (*Figure34*).

```
function [oilThresh,BWautomatic]=land_mask(img,ground,areaToExplore,landThreshold,medFilt)

tic
%% IMAGE ENHANCEMENT
noiseRemoved=wiener2(img,[medFilt medFilt]);
sharpenedImg=imsharpen(noiseRemoved,'Radius',1.5,'Amount',1.5,'Threshold',0.5);
Iblur=imgaussfilt(sharpenedImg,'FilterSize',5);
%% LAND MASK:
mask=Iblur>landThreshold;
%% Closing & Opening morphological operations
mask=imclose(mask,[0 1 0;1 1 1;0 1 0]);          %    |0 1 0|
mask=imopen(mask,[0 1 0;1 1 1;0 1 0]);           %  S=|1 1 1|
                                                 %    |0 1 0|
%% Call to oil spill segmentation function to show oil & land segmentation together:
[oilThresh,BWautomatic]=automatic_threshold_for_land(img,ground,areaToExplore,medFilt);
%% Call to function that shows mask and images
visualizeImages_for_land(img,mask,ground,oilThresh,'AUTOMATIC THRESH. OIL + LAND
SEGMENTATION');
%Used for the segmentation_evaluation function:
BWautomatic=or(mask,BWautomatic);
toc
end
```

Figure 34: land_mask.m function

visualizeImages.m

This function is used to show to the user the results of the segmentation method he had chosen. It takes four inputs: the original grayscale image, the ground truth labelled image, the resulting segmentation mask computed using one of the afore mentioned methods and a string containing the used method's name to show it as title of the resulting images.

First a new blank image having same size of the original image is created to act as a background, and on this is superimposed the cyan color oil spill mask (like the color used in the ground truth image). Then, the resulting mask and the ground truth are shown together to the user to be visually compared.

Later, another image is computed: the original grayscale image with the overlapped computed mask to give the user a better view of the results (*Figure35*).

```
function visualizeImages(img,mask,ground,stringMethod)
%Visualization GROUND TRUTH & FINAL MASK
background=zeros(size(img),'double');
maschera=imoverlay(background,mask,'cyan');
figure('WindowState', 'maximized');
```

```
6.  subplot(1,4,[1 2]),imshow(ground),title('Ground Truth');
7.  subplot(1,4,[3 4]),imshow(maschera),title(sprintf('%s : MASK',stringMethod));
8.  %Visualization ORIGINAL IMAGE with overlapped the FINAL MASK
9.  figure('WindowState', 'maximized');
10. imshow(img), title(sprintf('%s: Original image with overlapped mask',stringMethod))
11. hold on;
12. overlappedImg=imshow(maschera);
13. overlappedImg.AlphaData=0.35;
14. end
```

Figure 35: visualizeImages.m function

segmentation_evaluation.m

This function is used to compute all the evaluation metrics already mentioned in *Section 3* like Jaccard index, Dice index and BF Score, useful to evaluate segmentation results.

This function is very simple: all these indices are computed by using MATLAB's **jaccard**, **dice** and **bfscore** functions that takes in input only two parameters, the ground truth labelled image and the segmented mask produced. At the end, all these computed values and their relative images are shown to the user to give a visual representation of the segmentation method's goodness (*Figure36*).

```
1.  function segmentation_evaluation(groundTruth,segmentationMask)
2.  tic
3.  %% 1st metric: JACCARD SIMILARITY
4.  similarityJaccard = jaccard(groundTruth,segmentationMask);
5.  subplot(4,4,[1,2,5,6])
6.  imshowpair(groundTruth,segmentationMask)
7.  title(['Jaccard Index = ' num2str(similarityJaccard)])
8.  %% 2nd metric: DICE SIMILARITY
9.  similarityDice = dice(groundTruth,segmentationMask);
10. subplot(4,4,[3,4,7,8])
11. imshowpair(groundTruth,segmentationMask)
12. title(['Dice Index = ' num2str(similarityDice)])
13. %% 3rd metric: BF SCORE
14. similarityBFScore = bfscore(groundTruth,segmentationMask);
15. subplot(4,4,[10,11,14,15])
16. imshowpair(groundTruth,segmentationMask)
17. title(['BF Score = ' num2str(similarityBFScore)])
18. toc
19. end
```

Figure 36: segmentation_evaluation.m function

kmeansSegment_for_land.m* and *visualizeImages_for_land.m

These last functions are almost the same as the relative function already mentioned above, but for some differences useful to show land and oil spill masks together.