Computer Vision Final Project

Benfatti Samuele, Donadi Ivano, Zuccante Anna

July 2021

1 Abstract

We worked on the final Computer Vision project in a group of three, thus we developed all three points of the instructions, i.e. boat detection, sea segmentation and boat tracking.

To tackle this problem we chose to keep Machine Learning to a minimum and to rely as much as possible upon "classical" CV methods, starting from low-level processing and building a complete pipeline which led us up to high-level processing.

From the very beginning, we knew this approach would probably not lead to state-of-the-art results; furthermore, the resulting algorithms would probably be quite application-specific w.r.t. implementation tricks and parameters choices. Nevertheless, we decided to do so, since we wanted to both deepen our understanding of the material seen in class and to somehow challenge ourselves.

Note: The instructions for the execution of our resulting programs are in section 4.

2 Method

2.1 General approach

Our main inspiration came from the bag-of-visual-words method, which gave us something to build upon. However, we soon moved away from the "standard" approach, building something much more personalized.

The general resulting pipeline is as follows:

1. Preprocess the image
2. Extract some image features
3. Classify the individual features (the only point which requires ML)
4. Extract some markers from clusters of features

5. Perform watershed segmentation from the given markers
6. Detect boats either from segmentation or directly from the markers
7. Perform post-processing as needed

Features are classified into sea, boat and background; as for the features themselves, we started by using SIFT, since it is a sort of "default" first-choice option in case of general images; it turned out to work reasonably well, thus we never explored other options.

Since our approach is based on semantic segmentation, we are not able to distinguish overlapping or very close boats; further panoptic segmentation techniques could be employed to enhance the results. It would have been interesting to try this kind of techniques given enough resources.

Thanks to both SIFT and our approach in general, we gained several robustnesses; in particular:

- Images without sea and/or boats are correctly analyzed given an adequate training set, since we did not put any prior on the distribution of the markers and we added quite robust noise-rejection techniques
- Boats can appear from any viewpoint, thanks to the usage of SIFT
- Thanks to both SIFT and multi-scale analysis, we reached reasonable scale-invariance

This will be clearer in the results section.

2.2 Ground truth and dataset construction

2.2.1 General

The datasets we used are the ones indicated in the project instructions, i.e. the MAR Venice dataset at

http://www.dis.uniroma1.it/ labrococo/MAR/classification.htm

and the Kaggle Boat Types Recognition dataset at

https://www.kaggle.com/clorichel/boat-types-recognition/version/1

In order to both build a training set and test our results we had to manually annotate several images. We required bounding boxes to quantify our performance w.r.t. detection, while binary masks were needed to asses segmentation quality. For training our machine learning module we technically needed labelled keypoints, but we could easily extract them directly from the images plus binary masks.

The fact that we trained on keypoints helped us to considerably reduce the number of needed images, since from a single sample we can obviously collect several keypoints. This led to a great speed up, since manual mask specification turned out to be extremely tedious and time-consuming.

Unfortunately, we still need a minimal amount of good-quality data to get reasonable results; while this was not a problem with Kaggle's dataset, it was a strong limitation for Venice's dataset. In particular, we could not simply merge the two datasets, since from a machine learning point of view they came from two really different probability distributions, i.e. it was almost impossible to generalize between the two; at the same time, we could not find online lots of images compatible with the ones of Venice, so we could not expand its dataset.

In the end, we had to include in the training set some images from the test set of Venice; while far from optimal, this was required to show that our approach works in principle, and that we just need some more data. To better show that the general pipeline is effective even without resorting to overfitting, we decided to keep Kaggle's data separated from the one of Venice, and to rigorously separate training and test data in the former case.

In both cases, more data and more training time would result in better performance, as shown by some tests we did.

Anyway, to speed up the data collection process we wrote two executables, which are better described in the following sections.

2.2.2 Boat bounding boxes

The program we wrote accepts as command-line parameter a file extension and a folder, from which it loads all the images with the given extension. Depending upon the options given in the command line the program will either:

- Display the bounding boxes already selected and saved to file
- Display the images without bounding boxes and enable the selection of boats using `cv::selectROI`
- Display all the images, possibly with the already selected bounding boxes, and let the user modify them

The selected ROIs are saved to a .TXT file with the same base name of the corresponding image, where each line corresponds to a boat and includes the upper left angle coordinates of its box, along with its width and height.

2.2.3 Segmentation masks

The program to select the binary masks accepts parameters similar to the previous one, but depending on the options it receives it will:

- Load only not-yet-annotated images and enable the user to paint the sea mask using the mouse, simultaneously visualizing it
- Do the same, but for the boats mask
- Once again the same, but loading also already labelled images in order to modify the masks

High precision turned out to be of vital importance, making the process particularly tedious; fortunately, background masks were not needed, since they could be extracted from the inversion of the union of the other two masks.

As for the file format of the labelling, we just saved the two binary masks as .PNG, with the same base name of the corresponding image, plus a "_mask" suffix for sea masks and a "_maskb" suffix for boat masks.

2.3 Machine Learning

For the classification of keypoints we tested in parallel three different methods:

- Artificial Neural Networks, implemented in C++ from scratch
- Decision Trees, once again implemented from scratch in C++
- k-means, exploiting the OpenCV implementation

The first method to give good results was k-means, so for limitations on time we concentrated on it, even tough the other methods proved to be promising.

The final resulting pipeline is, given a keypoint p to be classified:

1. Retrieve k (parameter) centroids for each of the three classes (e.g., $k = 2000$)
2. In general, take for each class the centroid closest to p
3. Consider the two best matches among the three classes
4. Classify the point according to the best match if and only if the ratio between the first and the second choice is under a parametric threshold; otherwise, discard p

Several implementation tricks were needed, like multitreadhing, approximated nearest neighbour via FLANN, an optimized version for real-time usage in video, and so on.

The greatest problem we faced, apart from the scarcity of training samples for Venice, was the lack of a sufficient number of keypoints in some images, in particular in overexposed and/or low resolution images (which once again are particularly abundant in the Venice dataset). We successfully tried to increase the number of detected keypoints using histogram equalization; unfortunately, we had to discard this idea, since the resulting descriptors were not consistent enough, due to the enhanced noise and pixellation of the images.

2.4 Segmentation and segmentation-based boat detection

To perform semantic segmentation, we start by performing SIFT on the whole image, classifying each keypoint as previously described. The image is then divided into cells; the number of cells along the larger side of the image is a parameter, while the number of cells along the shorter side is obtained via a proportion.

We thus obtain a 3D histogram, with two dimensions for the grid and the third one for the three categories (sea, boat, and background). Such histogram is filled by the keypoints, each one giving a vote to the cell and category it falls in. In the meanwhile, for each cell we calculate the center of mass of the corresponding keypoints, one for each category.

Cells are then labelled according to the category that received the most votes, as long as a minimum relative threshold is reached; we thus return to a 2D representation of data. Such representation is then converted into a set of three binary masks, one for each category; each mask is processed via morphological opening and closing (in this order) to improve stability and robustness to noise.

Particular care must be taken for the mask of the boats, where three cases are automatically considered:

- If large boats are detected via a threshold on the maximal contiguous area in the mask, we perform an additional 3×3 erosion, so as to avoid overflow of the boats from their contours during the segmentation phase
- Otherwise, we try to understand whether we have only noise or if, on the contrary, small boats are present:
  - We calculate the standard deviations σ_x and σ_y of the position of the cells suspected to be boats, along the corresponding directions
  - We then normalize such values, dividing by the size of the corresponding side of the image
  - If the standard deviations exceed a certain threshold and are well-conditioned, i.e. suspected boat cells are distributed isotropically along the image, then we assume that the mask is composed by noise
- If the mask is composed by noise, we just suppress it, thus enhancing the performance of the algorithm in absence of boats by reducing false positives
- Otherwise, we do not perform the morphological operators described previously, so that we do not erase small boats

At this point the user can choose whether or not to apply the bias of flat areas corresponding to background. This is useful when the background contains significant portions of clear sky or flat walls, where keypoints are scarce; on the other hand, if we have low-quality images with overexposed areas and/or flat regions induced by low resolution, we risk to mistake everything for background, so the following processing should be avoided.
If the bias is selected, then the following procedure is applied:

1. Compute the Laplacian of the image after Gaussian smoothing
2. Normalize the Laplacian between zero and one
3. Parametrically threshold it

4. Rescale it to the size of the grid, apply dilation and compute the negative
5. We have thus obtained a mask selecting flat regions
6. Create another mask by union and dilation of the grid channels, and take its negative
7. This selects unclassified regions; the background is added at the intersection of the two masks
8. Finally, slightly erode the sea mask, in order to avoid having overexpansion of the sea

At this point, independently of whether the bias was selected or not, we draw the markers to perform watershed segmentation; for each category mask:

- The grid is superimposed to the image
- For each non-empty cell:
  - If the cell contains a center of mass, then draw the marker at its position
  - Otherwise, put a marker towards the position of each non-empty neighbouring cell of an 8-neighbourhood; for example, put a marker in the upper-left corner if the upper-left neighbour is non-empty
- This helps avoiding overexpansion of segments due to cells created by morphological dilation of the grid

The order of precedence between the three labels is boats, sea and finally background.

We can, at last, apply watershed segmentation to the original image sharpened by subtracting the Laplacian of the Gaussian, starting from the markers we have just obtained. We also tried to perform segmentation on the histogram equalized grayscale image and on the distance transform of its edge map, but this led to worse results.

The segmentation results are saved via a three-channel mask, where boats are in the green channel, sea in the red one, and background in the blue one.

2.4.1 Boat detection from segmented image

After having segmented the image, we retrieve the green mask channel in order to perform boat detection via the following steps:

1. Morphologically close the mask
2. Extract bounding boxes from its contours
3. Merge overlapping bounding boxes, if a metric (area of the union over sum of the areas) is below a parametric threshold

4. Iteratively remove the rectangles with area below a parametric percentage of the mean area, until all rectangles have an area above the threshold of the current mean (e.g. 2%)

All the rectangles that survive this procedure are classified as individual boats, even if we know that we could have multiple boats inside a single bounding box due to the previously described semantic-segmentation limitation.

2.4.2 Parameters summary

To sum up, the non-static parameters of the pipeline so far are:

- The threshold of acceptance for k-means
- Whether or not to insert the bias over flat regions
- The threshold for the Laplacian to detect flat regions, if needed
- The number of grid cells along the major image side
- The percentage threshold over the area of the bounding boxes
- The threshold on the metric for the union of the bounding rectangles

2.5 Boat tracking and marker-based boat detection

Since we were already required to detect still boats in point one of the project, we thought that the key idea of the third point was not to simply find all the boats, but instead to focus on moving ones with real-time performance.

We thus decided to focus on changing areas of the video, detected as follows:

1. Every n frames, where n is a parameter (e.g. $n = 3$), a frame is buffered
2. This way, we can perform movement detection between frames that are more spaced in time; this leads to better precision (slower movements are detectable) and faster execution of code
3. We thus got two successive images; both are converted to grayscale and blurred with a Gaussian filter
4. The absolute difference between the two frames is taken
5. Such difference is thresholded; we initially tried using Otsu's method, but it gave erroneous results when there were no changes between frames. We thus resorted to a static threshold, which performed way better
6. The thresholded mask is then dilated, so as to have a margin around the areas where to search
7. Finally ROIs are extracted from the contours of the mask and filtered by area

This way, we can detect SIFT features only inside the ROIs, obtaining a great speedup. We initially tried by just passing the obtained mask as a parameter to the OpenCV function, but it later turned out that this way the time complexity was not reduced w.r.t. scanning the whole image. We hence manually cropped the frame, this time resulting in the desired speedup.

We then proceed to classify the detected keypoints via an optimized version of the previously described k-means classifier; ROIs are hence classified as boat or not-boat by directly checking the ratio of boat keypoints over total detected keypoints. We draw the so-found bounding boxes on the frame, using green for the boats and red for the ones which are not yet accepted.

Finally we track detected boats using Lucas-Kanade; to avoid as much as possible the presence of multiple boxes for a single boat, we dynamically merge newly found bounding boxes if they overlap enough, according to the metric area of the intersection over area of the smaller box.

Several other secondary implementation tricks are present, along with a series of parameters. Such thresholds are statically defined inside the program and are tailored for noise reduction in the given videos; generally speaking, it would certainly be an option to have such parameters as command line or configuration file options.

3 Individual contributions

3.1 Ideas

Most of the important ideas and implementation tricks were found by brainstorming on Zoom or via messages throughout the day, so it is generally difficult to clearly assign specific ideas to individual components of the group.

3.2 Implementation

Anna was the main contributor to the code for the tracking subproblem, in addition to implementing some of the preprocessing utilities.

Ivano implemented the k-means encapsulation class, togheter with half of the mask-painting utility program and the core of the segmentation class.

Samuele contributed with most of the machine learning code that was not included in the final release, plus some general utilities, the program for bounding box annotation and the remaining half of the mask-painting utility.

The rest of the segmentation class, including several implementation tricks important for the segmentation task, where developed by Ivano and Samuele during pair-programming sessions.

3.3 Test and performance measurements

Samuele and Ivano concentrated on testing the images, while Anna focused on videos, since performance measures where a key component of pipeline polish-

ing, with some implementation tricks deriving from the observation of common errors. The final test on both images and videos were performed collectively.

3.4 General considerations

The tedious task of ground truth construction was evenly distributed among the group. Similarly, the report was written collectively by thinking together and taking turns at writing. Generally, we managed to evenly distribute working hours, with about 50 to 60 hours each.

As for the management of the project as a whole, we mostly relied upon Zoom calls, a dedicated Telegram group and, most importantly, GitHub. We initially tried to estimate the amount of workload via Project Management techniques, but we greatly underestimated the actual requirements. This made us better appreciate the difficulties that Computer Vision practitioners have to face.

As for the suggested collaboration with other groups for the building of the ground truth, it unfortunately proved to be unfeasible due to very busy exam schedules, i.e. we already had quite an hard time organizing between ourselves, so it was not possible to wait for other groups.

4 Executables

We developed three command line utilities to showcase our work:

- bseadetector: to build the datasets, perform k-means clustering and compute the performance metrics on the test set
- image_classifier: to process test images without having any ground truth as reference
- video_track: to perform boat tracking in videos and to save the results

4.1 bseadetector

Usage:

```
./bseadetector dir format params bdataset bcentroids [k]
```

where:

- `dir`: directory containing the images to classify. This directory must also contain the .TXT file for the bounding boxes, and the two .PNG masks for the boats and sea. Those files must follow the naming convention described previously
- `format`: expression for the images format ("*.JPG", "*.PNG", ...)
- `params`: path to the .TXT params file (containing "*params*" in the filename)

- `bdataset`: boolean, if true it builds a keypoint dataset from the input images
- `bcentroids`: boolean, if true it performs k-means on the keypoint dataset
- `[k]`: if '`bcentroids`' is true, it specifies the number k of clusters

This utility has three working modes which can be activated by the above command line options. If '`bdataset`' is selected, the program will build the keypoint dataset, by saving each keypoint descriptor along with a one-hot encoding of its label. If '`bcentroids`' is selected, the keypoint dataset, divided by label, will be clustered using k-means and the resulting centroids will be saved; if the two previous options are not selected this program will try to load the relative information from memory. After having obtained the centroids, it will proceed with the segmentation and boat detection of all the images in the input directory, displaying the results one image at a time, while outputting the IoU and pixel accuracy metrics. The parameters for the segmentation and detection tasks are read from the relative input file, which must have a row for each parameter in this order:

- Whether or not to insert the bias over flat regions
- The number of grid cells along the major image side
- The threshold for the Laplacian to detect flat regions
- The percentage threshold over the area of the bounding boxes
- The threshold on the metric for the union of the bounding rectangles
- The threshold of acceptance for k-means

4.1.1 image_classifier

Usage:

```
./image_classifier image_path params centroids_path
```

where

- `image_path`: the full path to the image to analyze
- `params`: path to the .TXT params file (containing "*params*" in the filename)
- `centroids_path`: path to a directory where the directory `kmclassifier` of the centroids is located

This program simply performs sea segmentation and boat detection on the input image without requiring its ground truth. For this reason it will not output the performance metrics but it will just show the visual results. The parameters file must follow the same conventions as above.

4.1.2 video_track

Usage:

```
./video_track video_path centroids_path
```

where

- `video_path`: path to the video file to analyze
- `centroids_path`: path to a directory where the directory `kmclassifier` of the centroids is located

This program loads the video and performs boat tracking in each frame as described in the boat tracking section. On exit, this program saves the processed video up to that point to a new file with the same name as the original plus a "_TrackVid.mp4" suffix.

4.1.3 Tested systems

We tested all our programs on Windows 10 using the "mingw64" suite and on MacOS Catalina using CLion. The C++ standard used is C++11, and the project is built using CMake. The version of OpenCV used is 4.5.1. In case of any problem please feel free to contact us.

5 Metrics

As instructed, we evaluated sea segmentation using pixel accuracy, boat detection using IoU and boat tracking by visually evaluating the results.

5.1 Pixel accuracy

We divided the image in two classes: sea and not-sea (i.e. boats + background), and we evaluated the pixel accuracy by finding the percentage of total pixels correctly classified between those two classes in the image.

5.2 IoU

Since there in no unique association of true bounding boxes with estimated bounding boxes, we performed best-first matching: we iterate over all the boxes and each time choose the couple $(true\_box, estimated\_box)$ that maximize the IoU. Such value is added to a list and both boxes are excluded from their sets. Whenever one of the two sets becomes empty or the maximum IoU becomes zero, we count all remaining estimated boxes as false positives and all remaining true boxes as false negatives.

To each image we then associate all the computed IoUs and the count of false positives and false negatives.

6 Results

In general, we find the qualitative visual results to be quite satisfying. For what concerns the mean IoU metric, we reach (according to the resources provided in the instructions) above the good-match threshold (0.5) in both test sets. Pixel accuracy is quite good too, reaching on average above 75% in both sets.

The full results for all test images are in appendix A, while here we briefly discuss some of the most interesting results.

6.1 Boat detection and sea segmentation

6.1.1 Test set

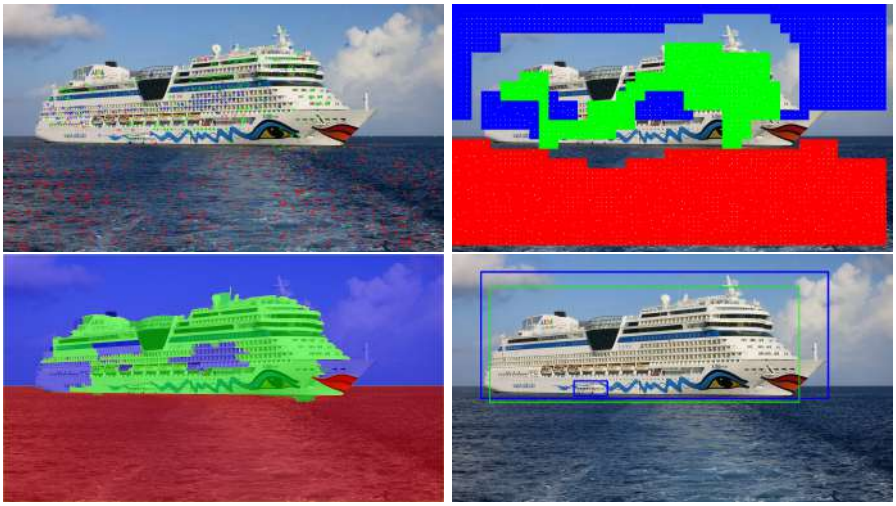

Figure 1: image 2712423

This is an example of good boat detection. We can see that some windows get classified as background but most of the boat keypoints get labeled correctly. Since most of the boat shape is detected the resulting bounding box (green) is quite close to the ground-truth one (blue). In this image we can notice how performing semantic segmentation groups together boats that are one over the other. In this case the small boat at the bottom produces a false negative.The sea instead, is labeled almost perfectly, helped by the added belief that smooth areas correspond to background labels.

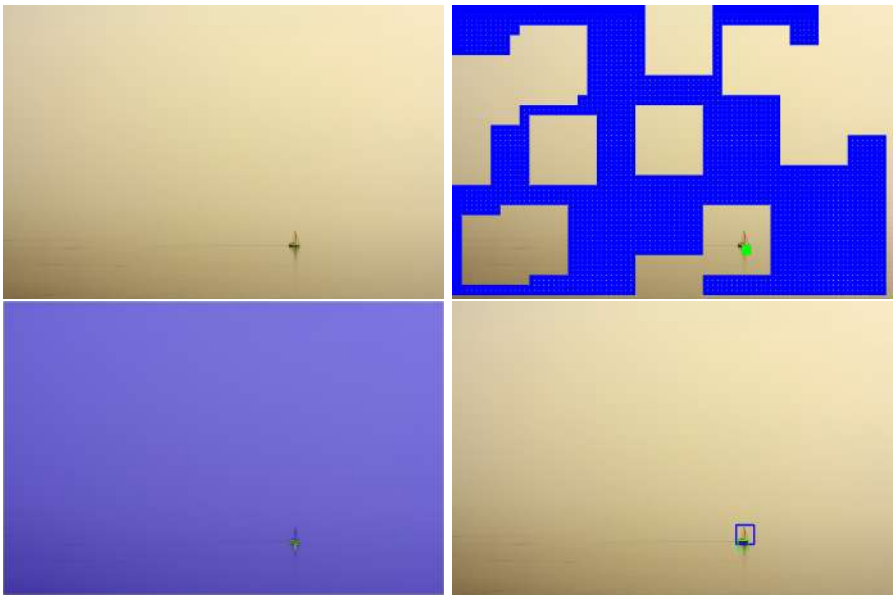

Figure 2: image 1819696

This image is quite problematic because it is very smooth and so it has very few keypoints. Hilariously, the system "correctly" detects the reflected image of the boat and not the boat itself. Since there are almost no keypoints or rough surfaces at all, most of the grid is filled with blue cells.

Figure 3: image 53122

This image is affected by some noise in the detection phase, resulting in small boats being detected at random points in the image. Since the actual boats have a feeble detection themselves, we cannot perform a round of morphological opening to get rid of the noise. Luckily, since the noise is surrounded by dense regions of the other labels, their segments end up having a very small area compared to the boat's segment and so they get filtered out. Unfortunately, the second boat is lost in the process, since its marker falls in a small window.

Figure 4: image 336718

This image has few keypoints to begin with, and the few water ones that are present get misclassified, resulting in no red cell in the grid. The boat markers, on the other hand, fall perfectly in the boats' main bodies, resulting in a quite good segmentation. However, since the resulting bounding boxes are drawn around the main bodies of the boats, while the ground truth boxes also contain the boats' tops, the IoU for all three boats is around 0.3.

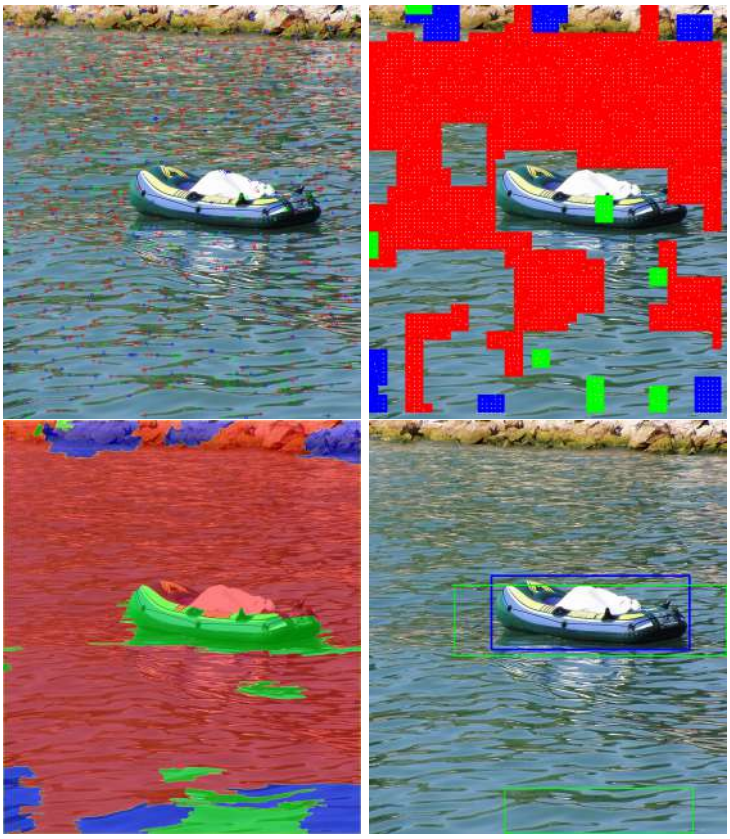

Figure 5: image 199811

The water in this picture is quite unique, compared to the others in both the test and training set (Kaggle). This results in the keypoint classification being very noisy, but fortunately most of the noise gets suppressed by employing our grid system. Some small boat segments are still present in the water, but most of them get area-filtered and so we end up with just one false positive.

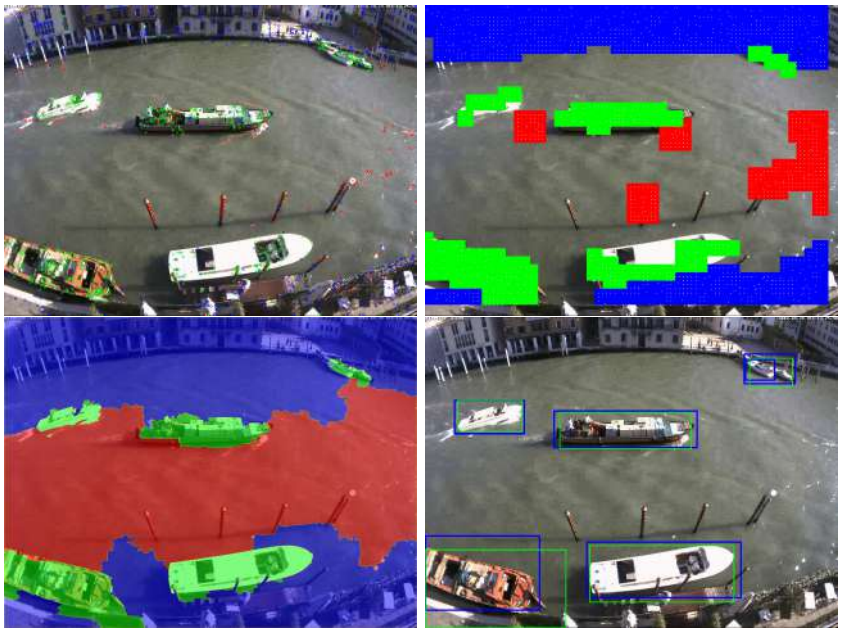

Figure 6: image 04

Here we can see a problem that is present in many images from the Venice test set: the lack of keypoints in the water, due to the poor quality of the images. In this particular image, the boats' markers fall quite well inside the boats, allowing the few water-labeled keypoints to expand between the boats. This is not the case for the background in the top of the image, which eats-up a good portion of the upper part of the canal.

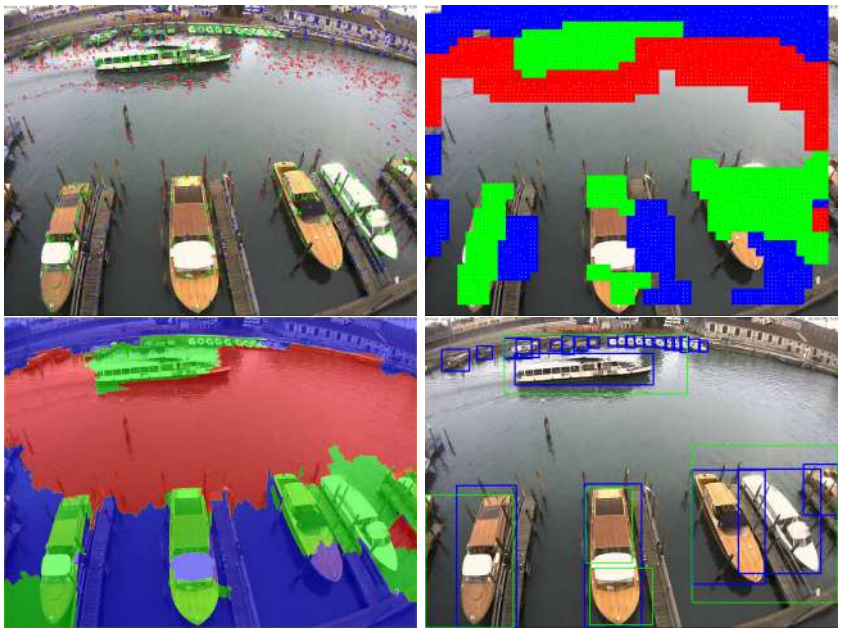

Figure 7: image 06

Here we can see a quite (metric-wise) disastrous consequence of using a semantic segmentation technique: a row of very small and very close together boats get merged into one. Then, since they are also quite close to a big boat with only flat water in between, it all becomes just one big happy boat. To a lesser extent this also happens to the two boats in the bottom right. This particular image ends up having an impressive record of 21 false negatives.

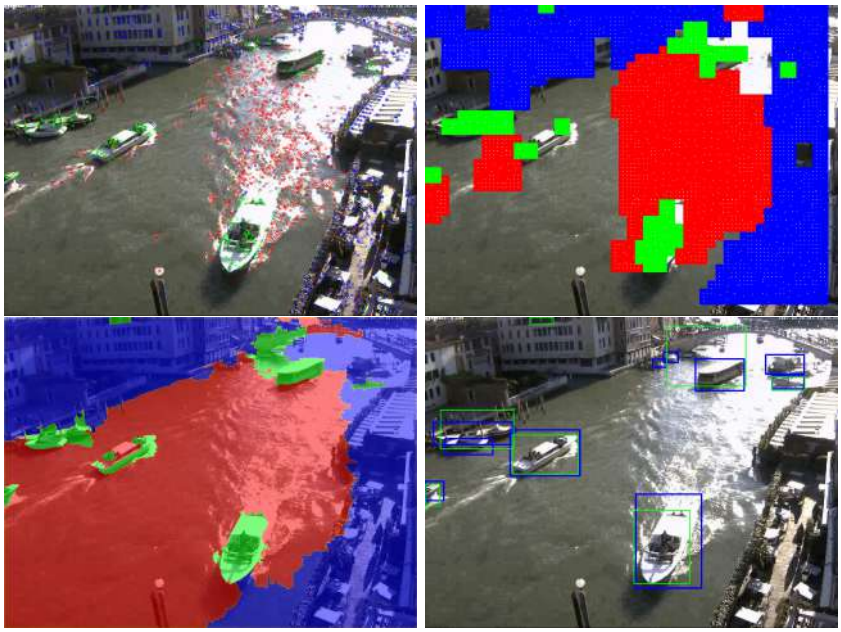

Figure 8: image 08

Here we have both good and not so good results: thanks to our grid and marker-placement techniques the boats get segmented quite well, leaving room for the sea to expand between them. On the other hand, a pair of windows in the top left part of the image gets detected as a boat. By visual inspection, we can see that they actually do resemble a boat, if enough fantasy is employed.

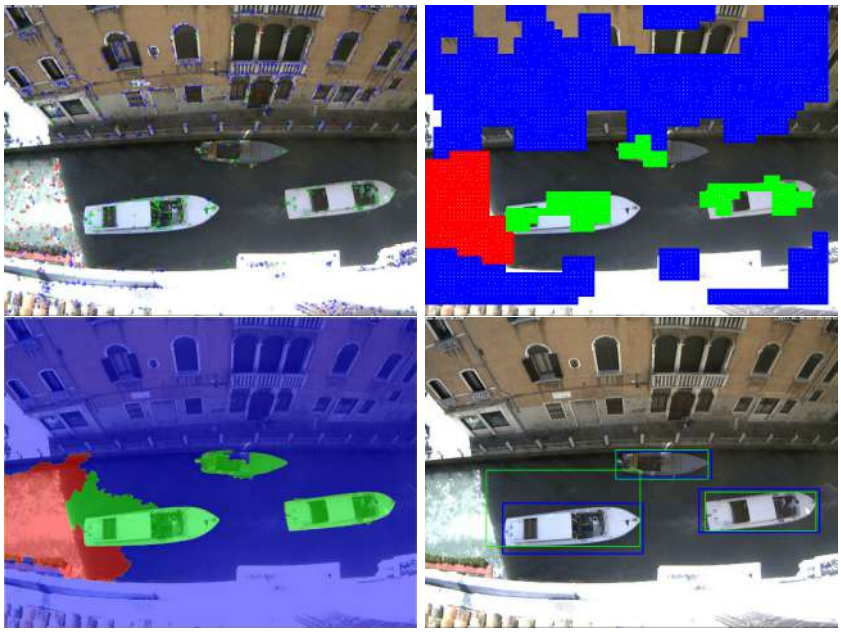

Figure 9: image 11

As a final consideration for the test-set images, here we give another example of an image where boats are detected and segmented very well, but the lack of water keypoints due to poor image quality does not allow for similar precision for what regard sea segmentation.

6.1.2 Additional test images

These images were downloaded from the top results of Google images, along with some images taken with our phones. Since their quality resembles the one of the Kaggle dataset, we used the centroids extracted from the Kaggle training-set. Since they come from different sources different parameters were used for each one.

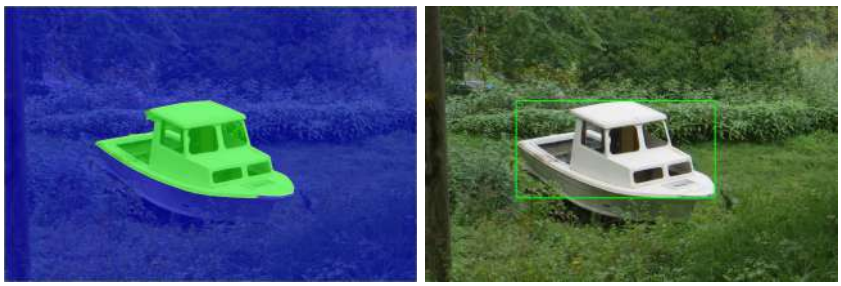

Figure 10: Boat without sea: watershed - bounding boxes

In this figure we can see that the boat gets correctly recognized, and no pixel is labeled as sea. To get this detection quality we needed to set a strict threshold on the confidence of the prediction.

Figure 11: Boat on sea: grid - bounding boxes

Here is another generic picture of a boat on the sea to show detection performance on images from other sources.

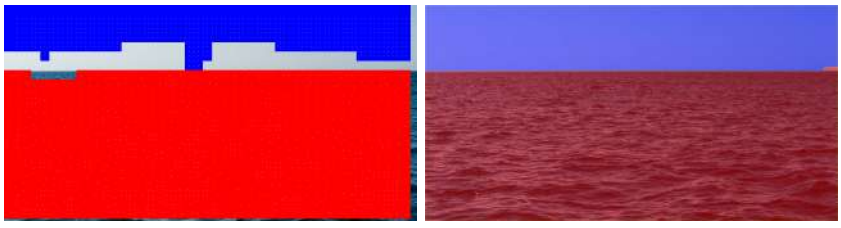

Figure 12: Sea without boats: grid - watershed

In this image we can see that no pixel gets labeled as boats, and that the

sea and the sky are both accurately labeled. Since this image was quite similar to the ones from Kaggle, it worked with the same set of parameters.

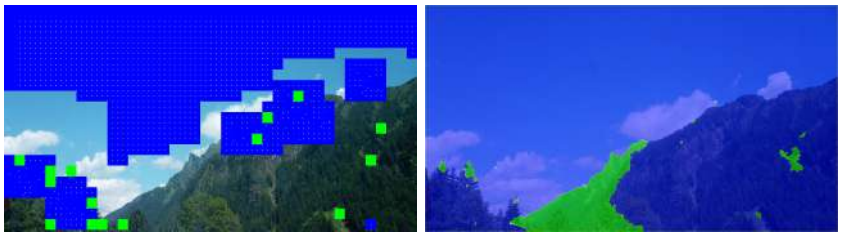

Figure 13: Mountains: grid - watershed

Here is the result of running the segmentation on the image of a mountain with the same parameters used for the kaggle dataset. We can see that there are some noisy detections of boats and, since they are mainly distributed along one direction, they do not get suppressed by the noise detection system.

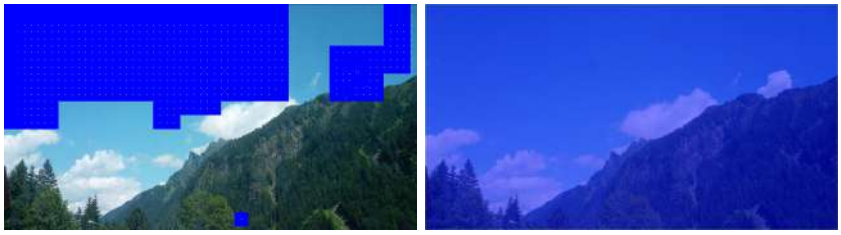

Figure 14: Mountains: grid - watershed

This is the same image as before but with a stricter threshold on the classification confidence. Now the noisy detections do not pass the threshold and the image is labeled correctly.

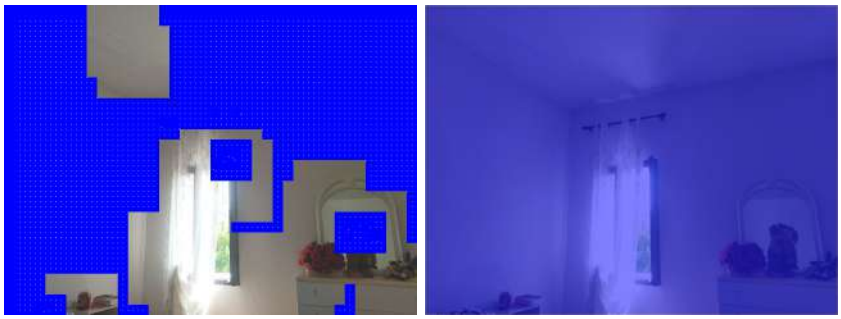

Figure 15: Empty room: grid - watershed

As a last example we took a picture of an empty room in one of our homes and it gets correctly labeled with the default parameters for the kaggle dataset.

6.2 Boat tracking

In the following video frames, overall, we can see how our implementation quite successfully detects moving boats but not still boats, as we stated in the implementation's description of video tracking.

Furthermore, we can appreciate the presence of both green bounding boxes, highlighting the presence of a boat, and red bounding boxes, which are possible boats not yet accepted.

In the second video, some boats passing through the top right area are not easily detected, since it is overexposed and it's difficult to find keypoints in it.

Figure 16: Video 1

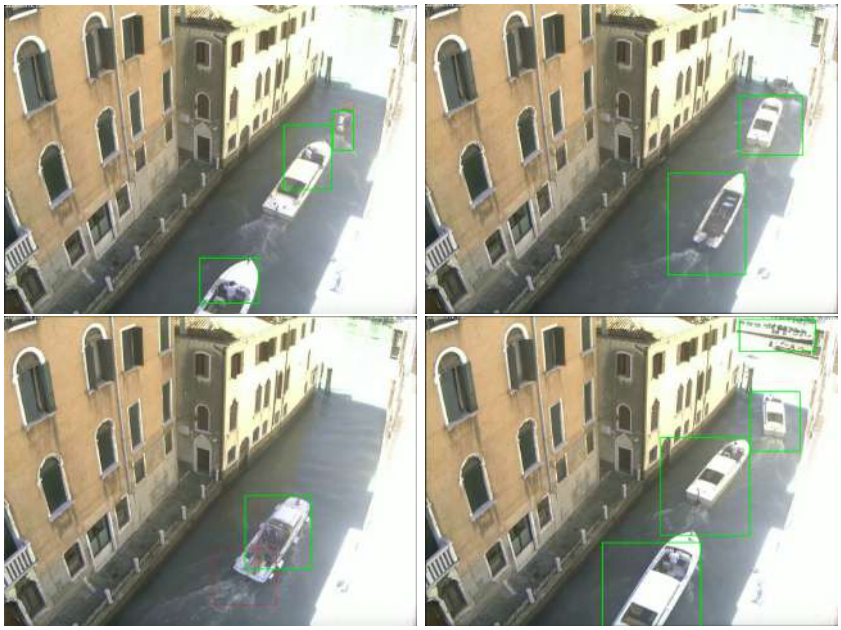

Figure 17: Video 2

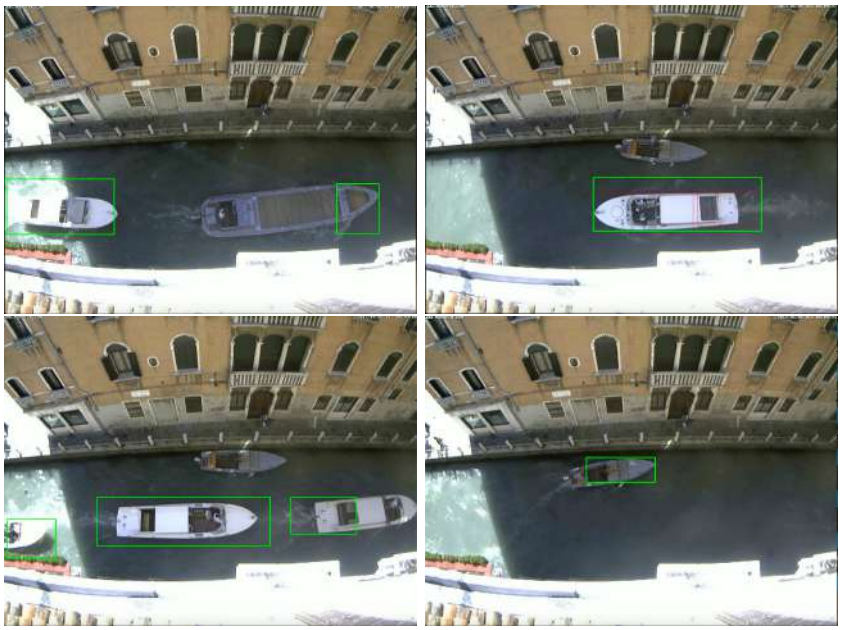

Figure 18: Video 3

7 Conclusions

All in all, we are pretty satisfied with the results. We understand that they are not "state of the art" but we were surprised by the accuracy that can be reached employing only "classical" computer vision techniques. The main drawback of our approach is the presence of several adjustable parameters, but it can't be helped since most of the classical tools we built upon are themselves parameter-dependent. For this reason we believe that our approach is best suited for an application-specific context, where parameters can be tuned for a restricted set of images.

Regarding the team work, overall, we enjoyed it and no problems arose; furthermore some great ideas came from each member, which probably wouldn't have been possible working individually.

To conclude, some improvements for future work we thought about are the following:

- Usage of a bigger dataset, to check how high accuracy can get
- Test of other features transform, like ORB, SURF etc.
- Further development of other machine learning techniques, like NN, SVM etc.
- More robust tracking of the boats using better parameters

As a general feedback, in case it is of interest for next years, we found the total workload (project plus exam) to be quite substantial for the 9 CFU of the course; nevertheless, the project turned out to be really interesting and challenging (in a positive sense); furthermore, it enabled us to explore lot of the techniques seen in the course, from low-level processing to mid and high level algorithms.

A Full Results

A.1 Metrics

Kaggle Dataset				
Image ID	Pixel Accuracy %	IoU	False positives	False Negatives
199811	83.17	0.603	1	0
144796	97.95	0.587	0	2
2878	55.67	0.265	0	0
84588	88.42	0.508	0	0
2733061	90.11	0.868	0	1
1819696	79.97	0.106	0	0
2712423	99.16	0.765	0	1
53122	85.46	0.594	0	1
336718	49.46	0.429 0.376 0.277	0	0
2573453	98.13	0.632	0	0
mean	82.75	0.501	0.1	0.5

Venice Dataset				
Image ID	Pixel Accuracy %	IoU	False positives	False Negatives
00	83.12	0.462 0.340	0	7
01	89.09	0.110	0	3
02	51.49	0.617 0.305	0	0
03	80.22	0.697 0.667	1	0
04	77.45	0.877 0.850 0.815 0.661 0.549	0	1
05	76.21	0.872 0.626 0.333 0.285	0	6
06	76.56	0.596 0.431 0.403 0.377	1	21
07	73.67	0.736 0.649 0.393	1	4
08	90.60	0.805 0.804 0.622 0.544 0.538 0.273	1	4
09	59.76	0.546 0.241	0	3
10	69.82	0.745 0.101	0	4
11	77.86	0.890 0.822 0.463	0	0
mean	75.49	0.557	0.33	4.4

A.2 Images

The whole already-processed, full-resolution images, together with the videos (in the folder `Tracking_videos`), can be found **at this link**. All the following images are presented in this order:

- In the upper left corner, we have the classified keypoints
- In the upper right corner, we have the mask for the grid
- In the lower left corner, we have the result of watershed
- In the lower right corner, we have the resulting bounding boxes

In all this images, blue stands for background, red for sea and green for boats. White dots on the grid mask represents marker coordinates.

A.2.1 Kaggle

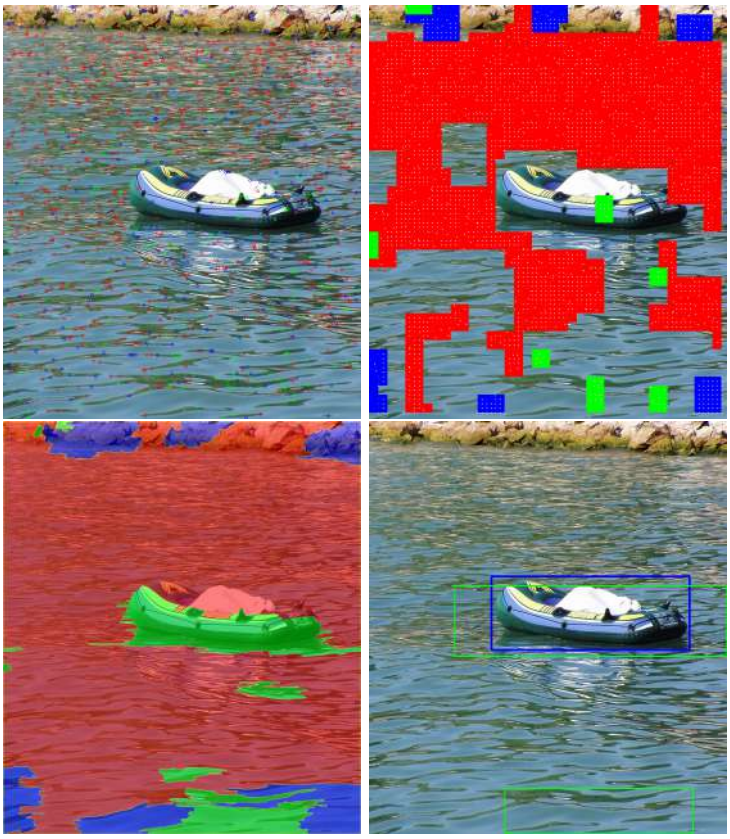

Figure 19: image 199811

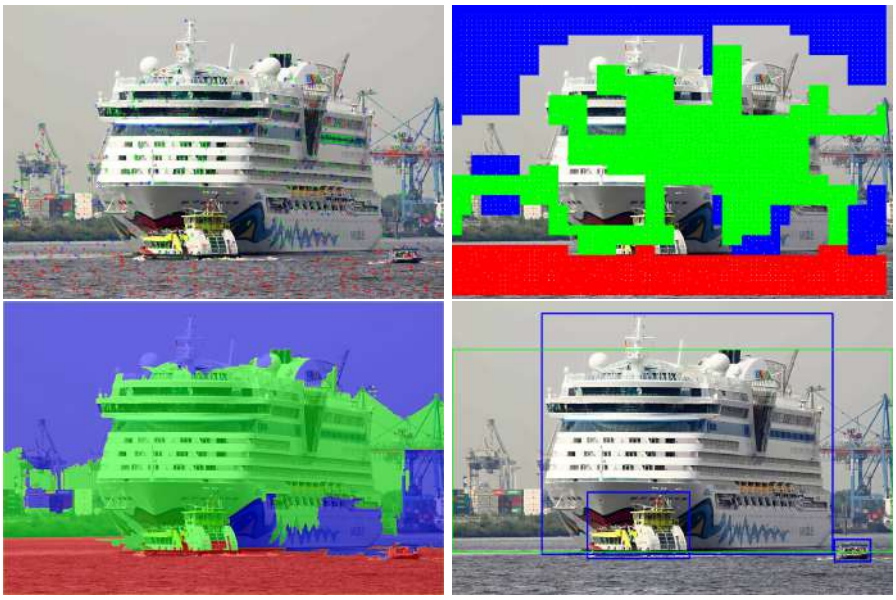

Figure 20: image 144796

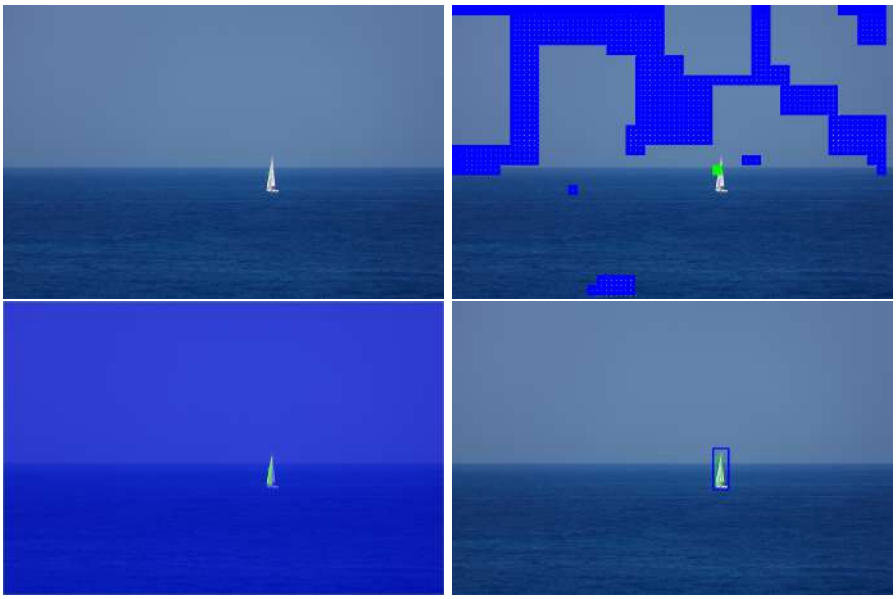

Figure 21: image 2878

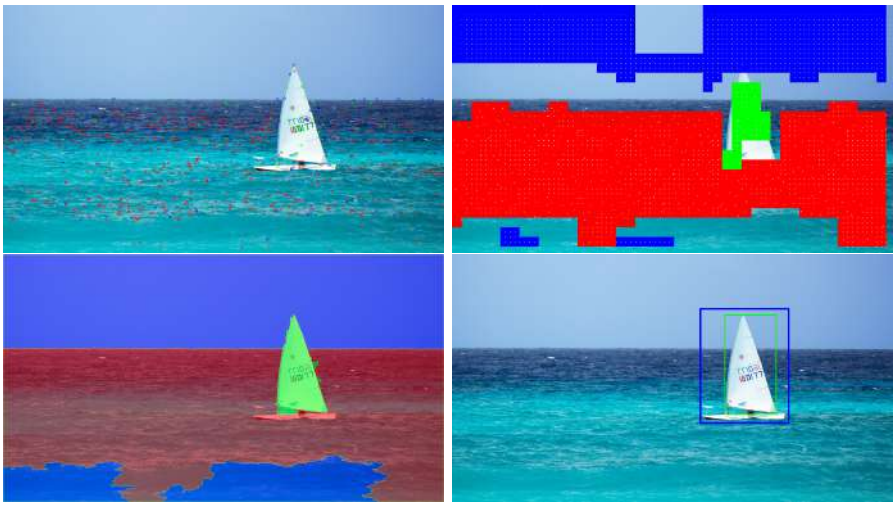

Figure 22: image 84588

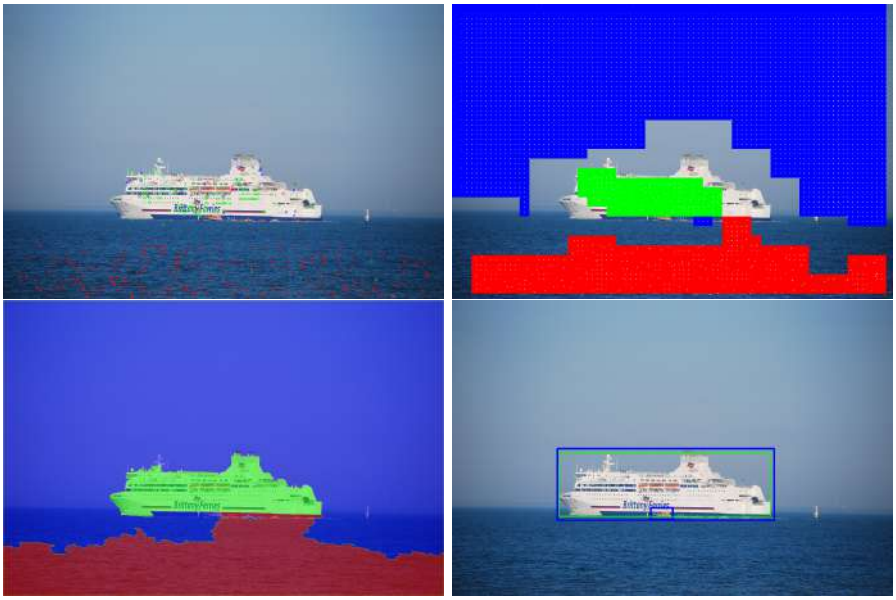

Figure 23: image 2733061

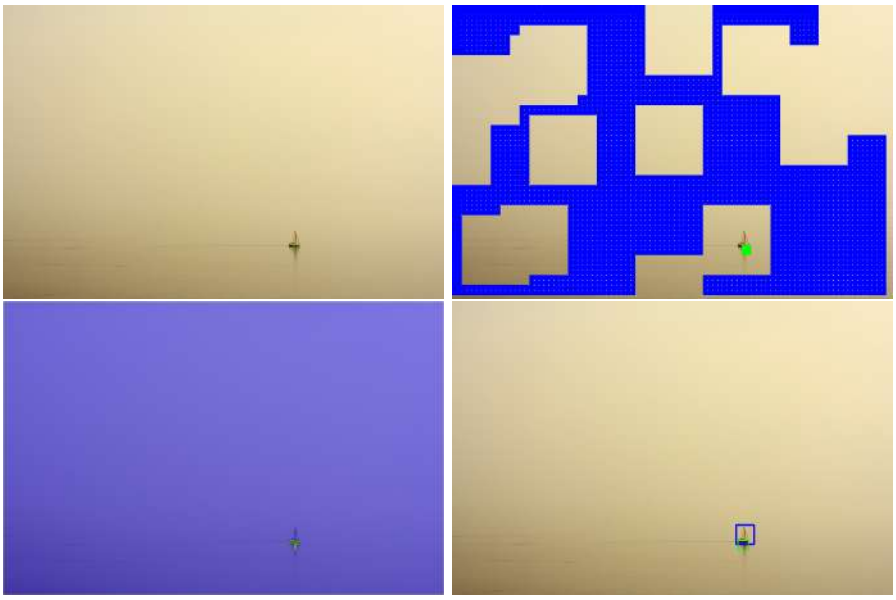

Figure 24: image 1819696

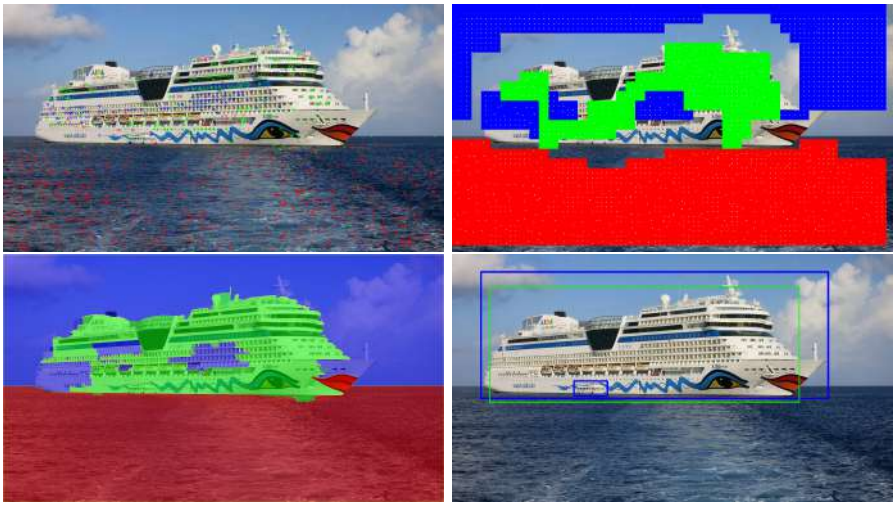

Figure 25: image 2712423

Figure 26: image 53122

Figure 27: image 336718

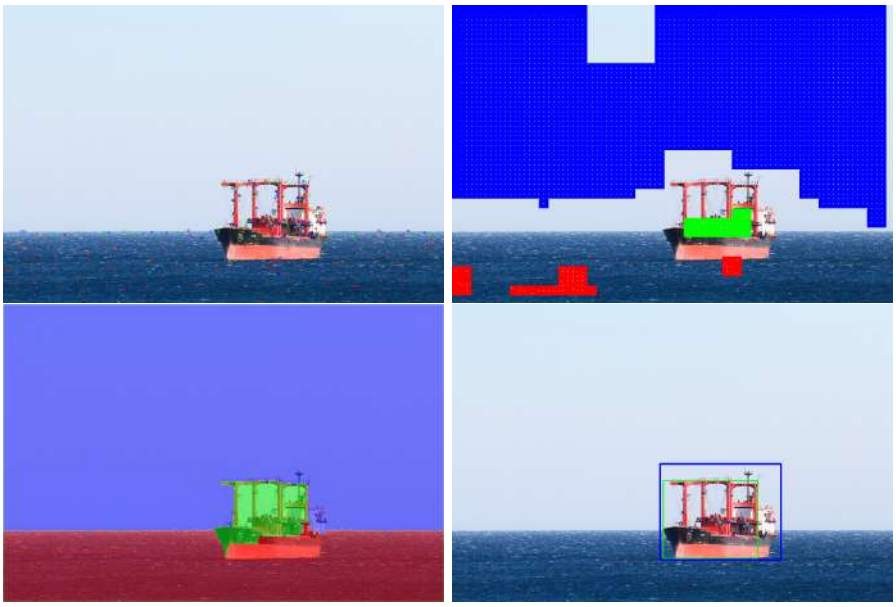

Figure 28: image 2573453

A.2.2 Venice

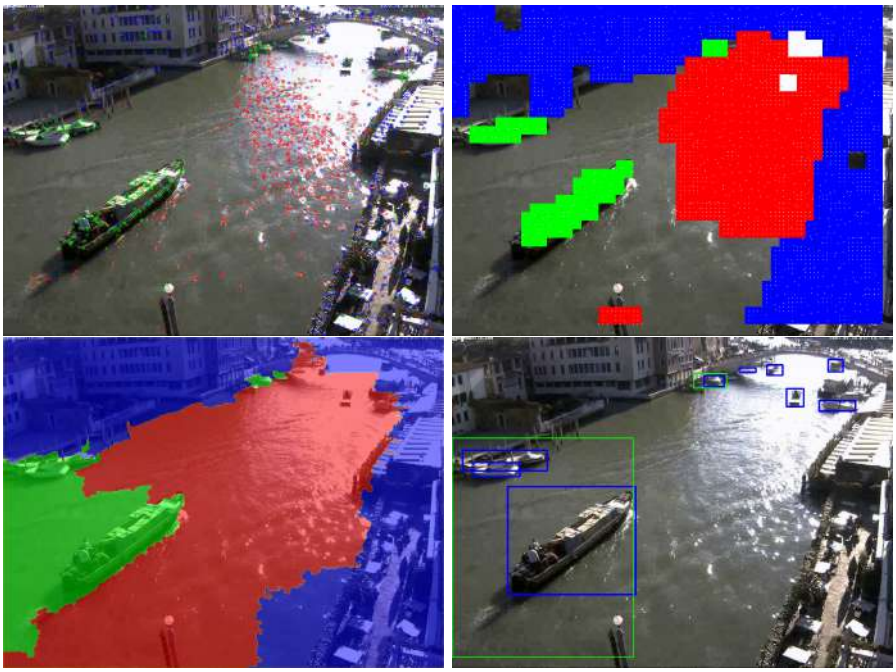

Figure 29: image 00

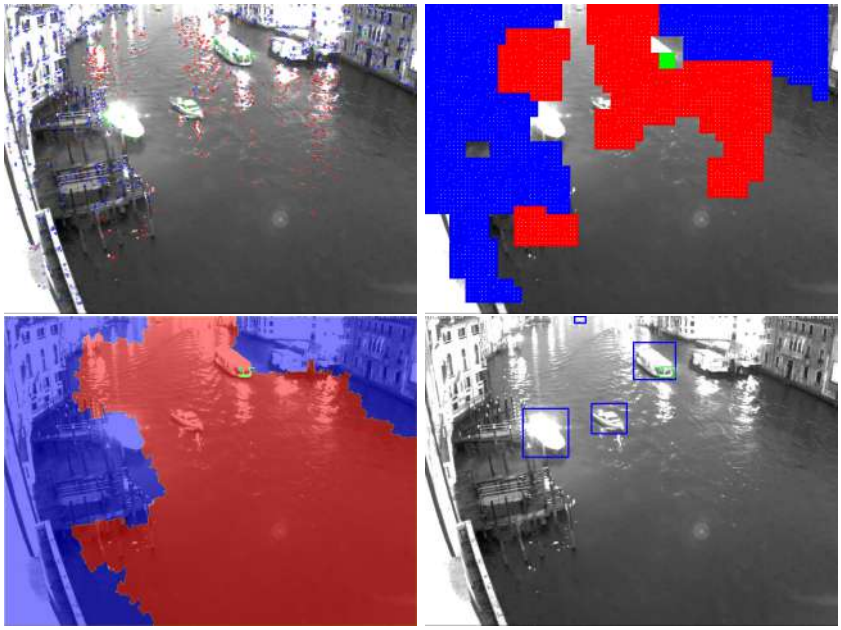

Figure 30: image 01

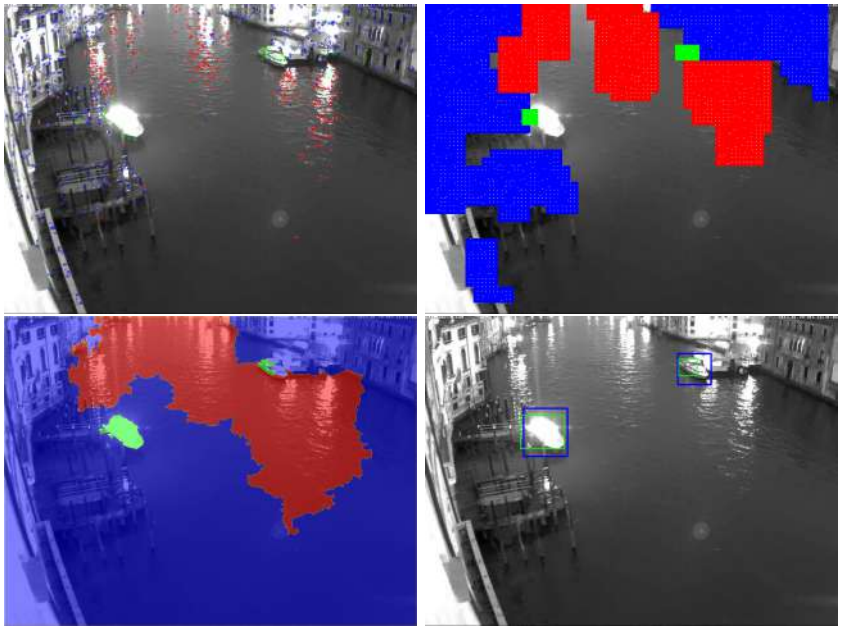

Figure 31: image 02

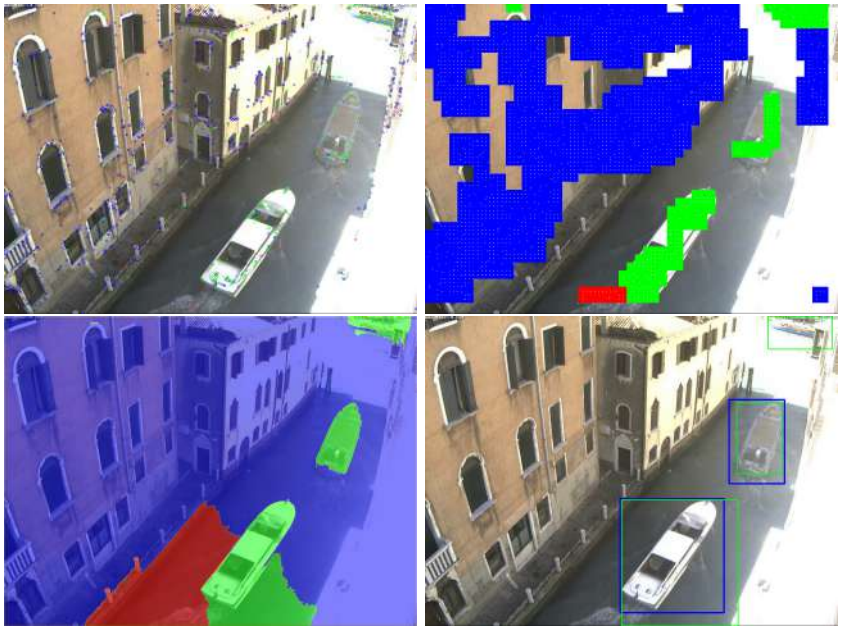

Figure 32: image 03

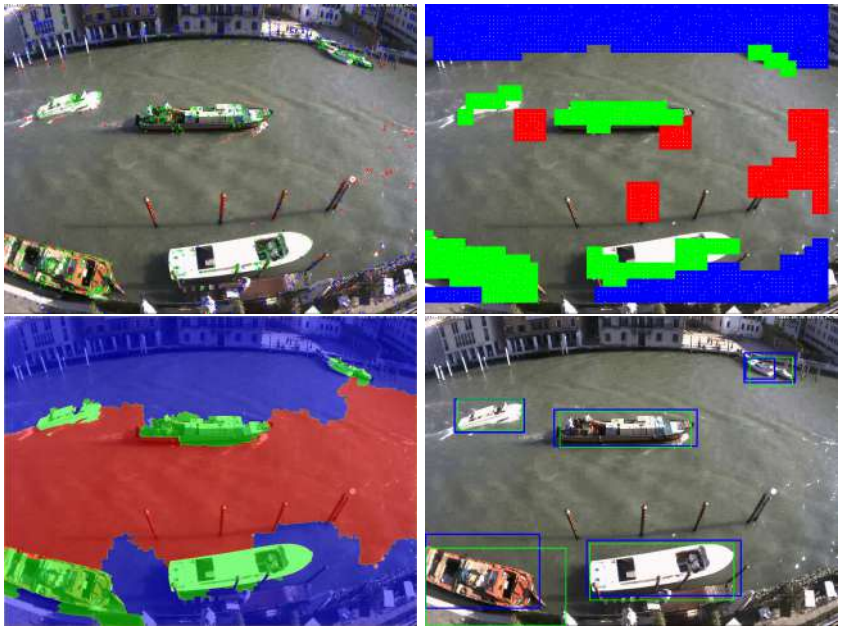

Figure 33: image 04

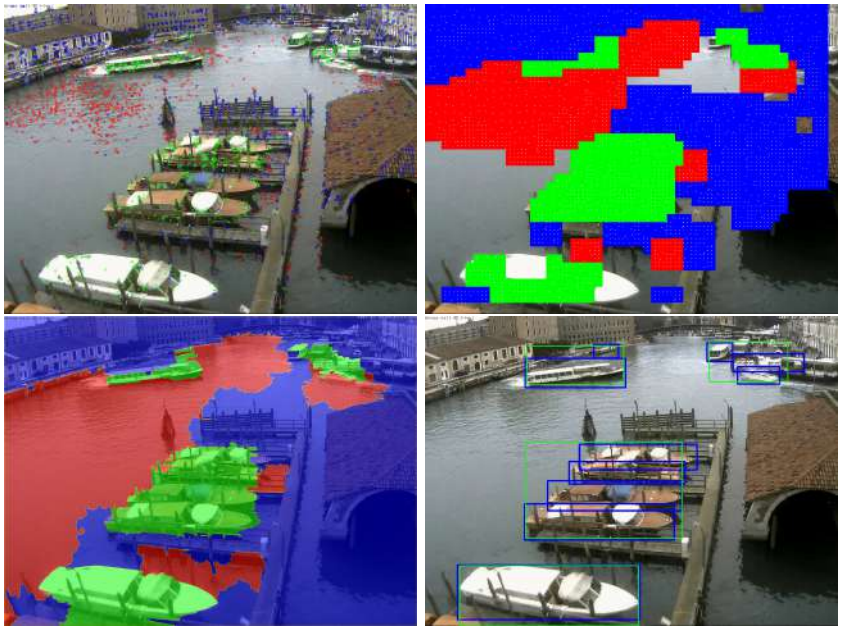

Figure 34: image 05

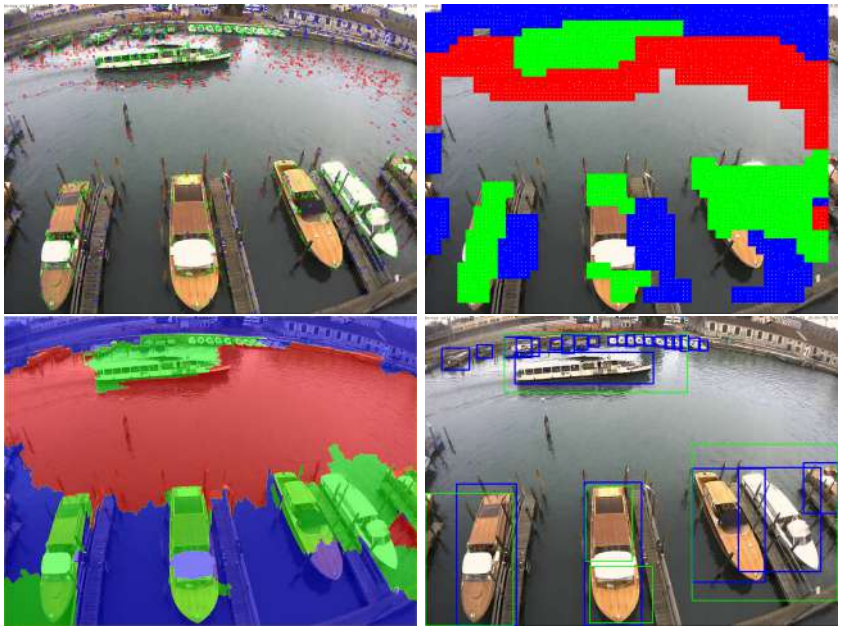

Figure 35: image 06

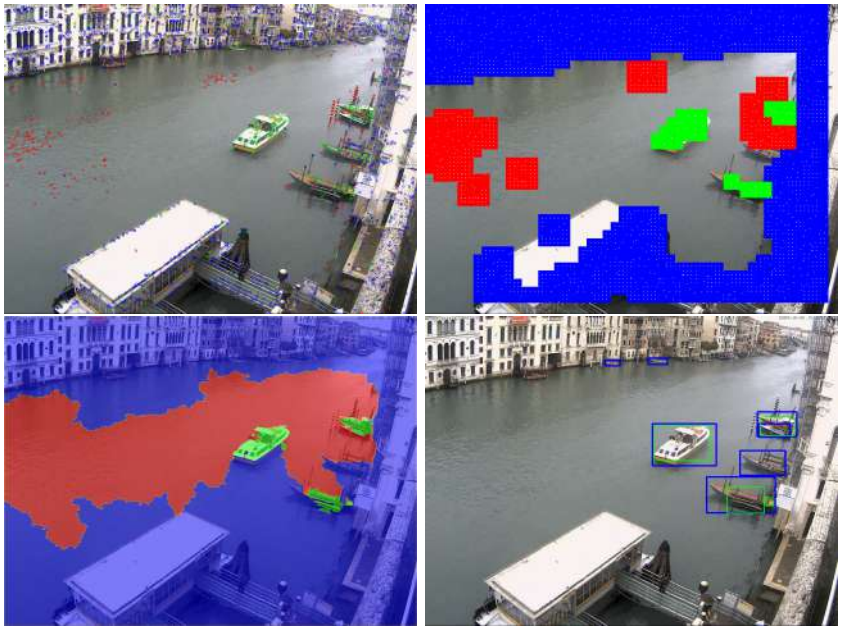

Figure 36: image 07

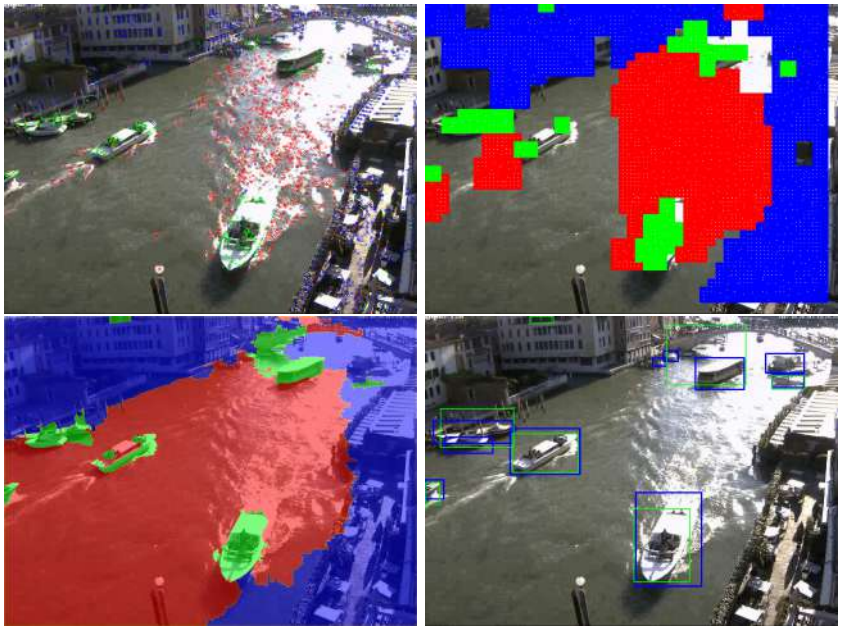

Figure 37: image 08

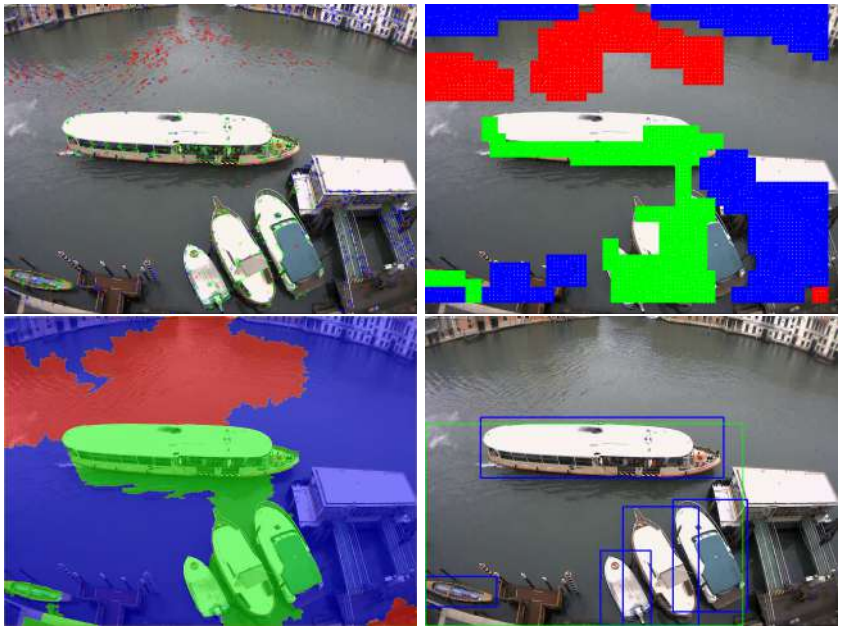

Figure 38: image 09

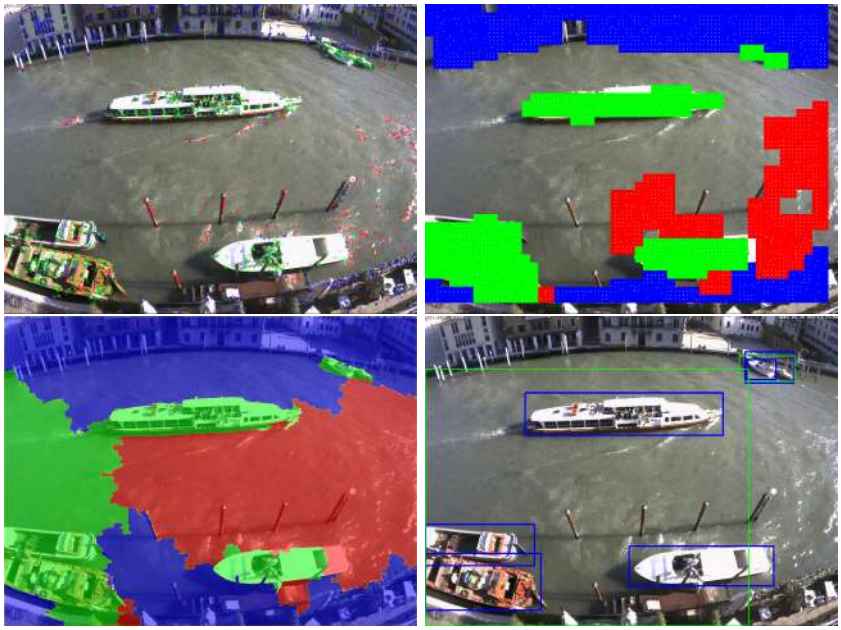

Figure 39: image 10

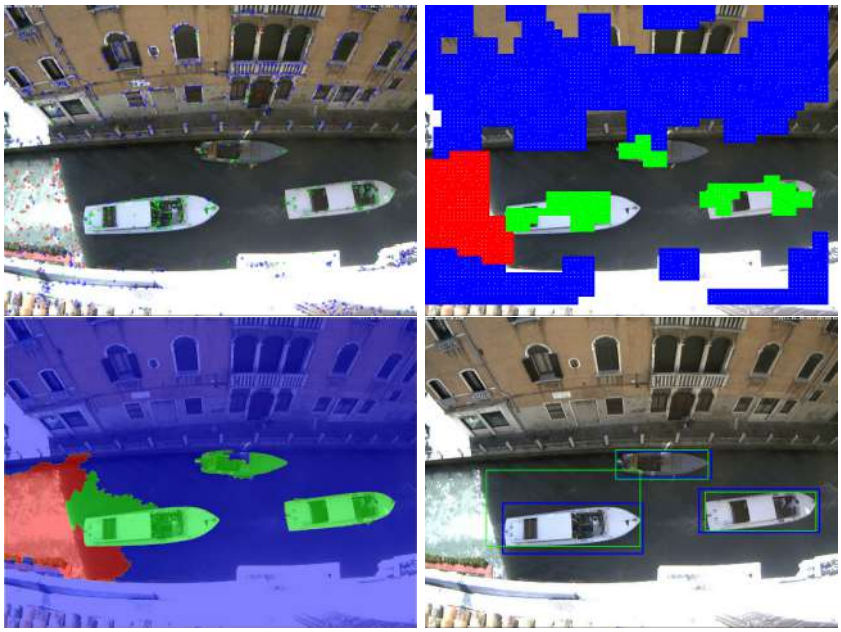

Figure 40: image 11

A.2.3 Videos frames

Figure 41: Video 1

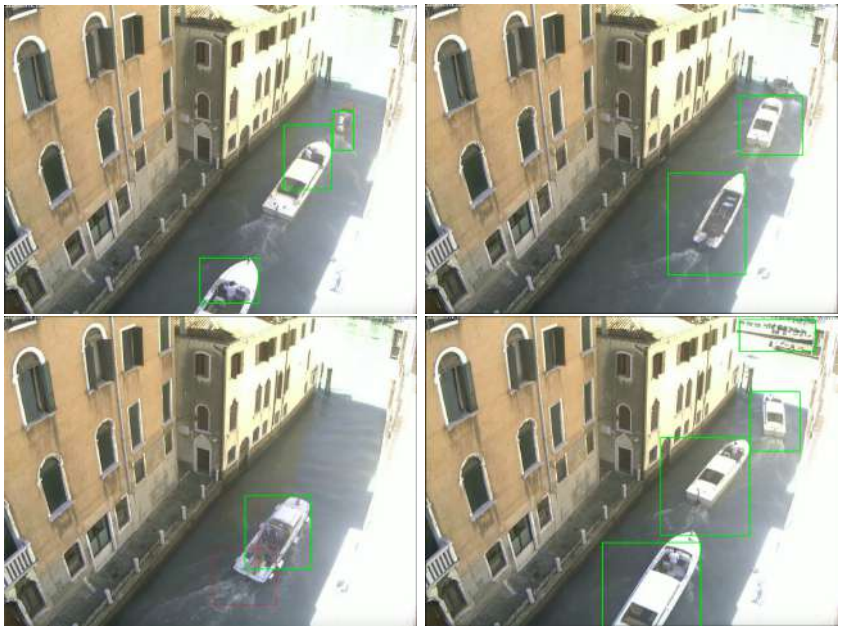

Figure 42: Video 2

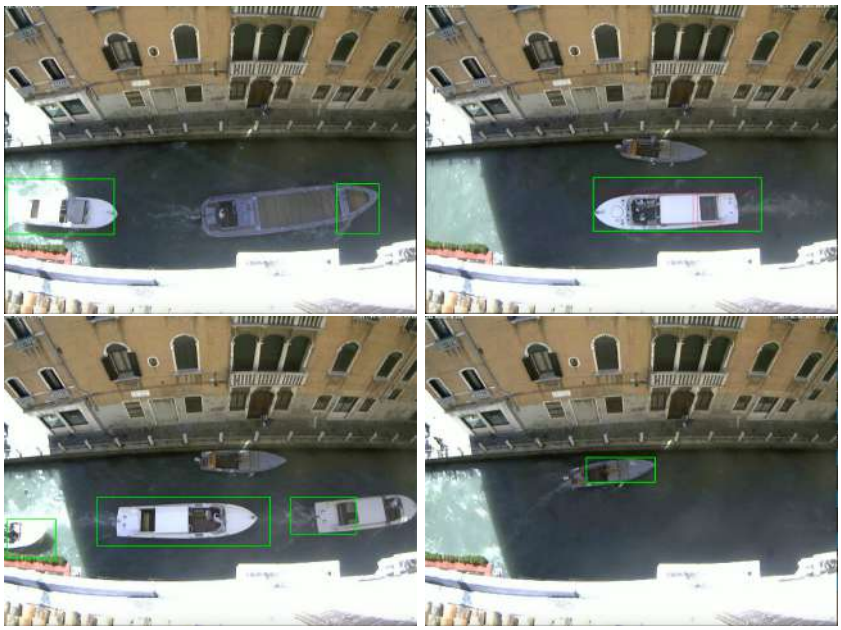

Figure 43: Video 3