Detection, tracking and location of runways on visual approaches

Projeto realizado para o curso "EA979 - INTRODUÇÃO À COMPUTAÇÃO GRÁFICA E AO PROCESSAMENTO DE IMAGEM"

- Daniel Gonçalves Benvenutti 169448
- Matteo Di Fabio 264339
- Renan Sterle 176536

1 Introduction

1.1 Motivation

Both in military aviation as well as in civil aviation, visual flight rules (VFR) remain of great importance nowadays. Particularly in airports and terminals of reduced infrastructure, visual approach and landing procedures account for the majority of cases and such procedures rely on visibility conditions, visual navigational aids and personal performance of the pilot in command. Figure 1 depicts what a final visual approach looks like.

In any type of landing approach, the maintenance of a safe flight envelope is crucial. Therefore, during the final approach leg, parameters such as airspeed, altitude, descent rate, glide slope, ground clearance, as well as location in relation to the runway must be monitored and corrected.

While airspeed, altitude and descent rate are measure by the aircraft, the other parameters can only be determined with the use of external information. In the case of instrument flight rules (IFR), Instrument Landing Systems (ILS), Global Navigation Satellite System (GNSS), inertial navigation systems and radio beacons provide such information, whereas on VFR they are ultimately estimated visually by the pilot.

This project is the implementation of a system that:

1. Given a picture of an approach, identifies and isolates the runway in question, using image processing filtering and segmentation techniques;
2. Fits the runway image into the expected runway shape by applying perspective and affine transformations;
3. Determines the camera point of view in relation to the runway start. Alignment, lateral offset and distance information are the most important, while relative altitude is of secondary importance;

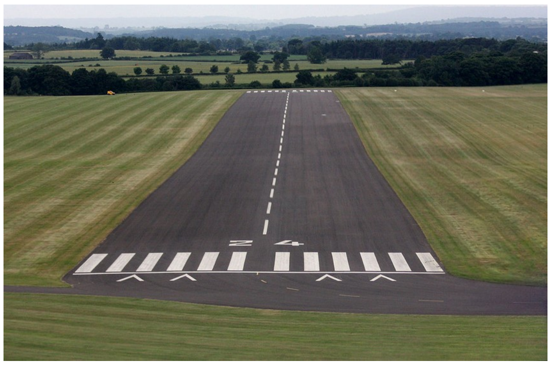


Figure 1: Approach view.

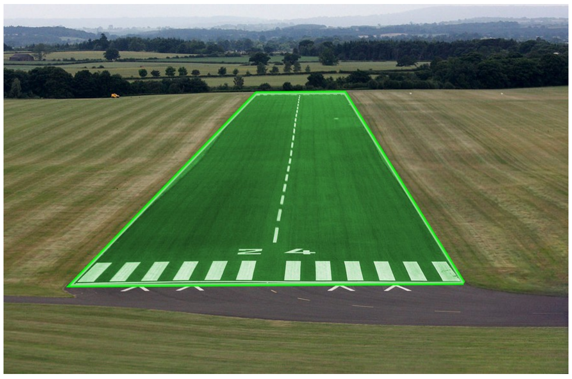


Figure 2: Segmented runway.

1.2 Theoretical Background

Most landing runways are approximately flat. Therefore, when photographed in two dimensions from a different plane in the 3D space, they can be encapsulated within a quadrilateral. This quadrilateral can be mapped to the expected rectangular 2D shape of the runway with perspective and affine transformations. Therefore, the coordinates of the camera can be determined in terms of those of the runway, and so can the relevant aeronautical parameters.

1.3 Development Plan

The project was developed in several independent fronts simultaneously with a final merge and constitution of the full pipeline. Tasks such as:

1. Finding the runway;
2. Isolating and correcting projection of runway;
3. Given a runway's bounding quadrilateral in the image, find finding camera's point of view;
4. Given camera's position, calculating the relevant parameters;

1.4 Data Source

As data sources, images of final approaches from the cockpit point of view where gathered from the internet. In addition, more standardized footage was generated on purpose and in controlled manner with the use of Microsoft Flight Simulator X.

2 Our Approach (The code/The processing)

Once the general idea of the project was defined, each member of the group was assigned one the the 3 following tasks and three independent modules where produced.

1) Identify runway;
2) Correct perspective;
3) Calculate camera origin;

The three modules are called by a main module and they interface with each other via a JSON file which is incrementally completed with the data generated by each step.

2.1 Identify runway

This modules uses both OpenCV and numpy frameworks, but no artificial intelligence. Given a picture of a runway on final approach, it’s task is to provide the coordinates of the four vertex of the runways bounding box.

The very ad hoc implementation is based on the following sequence of processing steps:

1. Contrast enhancement: The raw image has its contrast enhanced re-scaling its ABS data in order to make the runway more detectable. A constant positive Alpha and variable negative Beta are applied;
2. Bilateral Filtering: Two different settings of bilateral filtering are applied in parallel to the previous step. Both transformations have the purpose of making areas with similar colors more homogeneous, while keeping the runway edges sharp;
3. Canny Edge Detection: The algorithm is applied to the two results of the previous step and the high gradients of color and intensity are reflected into a black and white image corresponding to the edges identified;
4. Crop: The Canny algorithm in some situations falsely identifies the borders of the image as edges of interest. This is never the case and makes the processing more complicated, as they tend to be long as straight, as the runway should look like. Therefore, a margin of all the image’s borders is cropped to reduce ambiguity;
5. Dilation: The many edges identified by the Canny algorithm are in many cases duplicate of the same edges of interest. In other cases, one notable edge is not fully identified but is instead identified as various segments close to each other. The dilation process fuses these edges in spatial proximity and reduces the complexity of the analysis without significant losses of information. As a result, the general noise is reduced;
6. Hough Transformation: This step identifies segments of line among the dilated edges of the last step. These segments are described by the coordinates of their vertex and stored in a list;
7. Line filtering: Not all line segments are interesting. Particularly those which are vertical or with small inclination are of no interest. Therefore, a threshold filtering is done based on the inclination angle;
8. Line plotting: All filtered lines, from both processing branches, are plotted to a blank image. The result of this step is similar to the one intended by the dilation step. Line segments close to each other are merged, while segments that are in fact part of a greater segment are unified. Again the noise and ambiguity are reduced;
9. Hough Transformation: This time, the Hough transformation is applied for the plotted lines instead of the dilated edges, providing a smaller and more significant set of line segments;
10. Optimization: From all segments lines of the previous step, all vertex are extracted. For all vertex, in turn, two pairs of points are chosen such that the shadow of the segments of line

connecting such pairs maximizes the coverage of the plotted lines from step 8. Hence, the maximization of the shadow translates itself into the modeling of the runway edges.

11. Sorting: The points obtained need to be sorted in order to be used in the next stages of the pipeline. Therefore, they are converted to polar coordinates referenced to the mean of them, and sorted by the angle. Lastly, they are again converted to Cartesian coordinates;

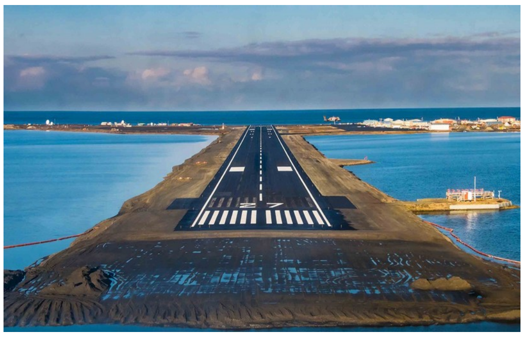


Figure 1: Raw Image

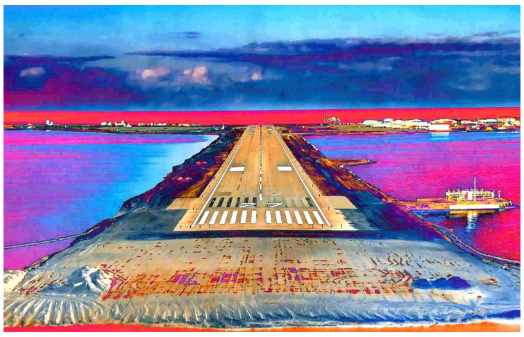


Figure 2: Contrast Enhacement

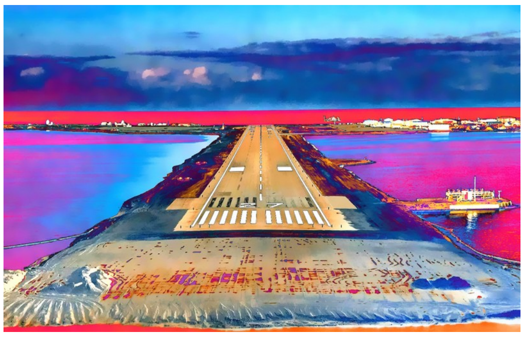


Figure 3: Bilaterial Filtering 1

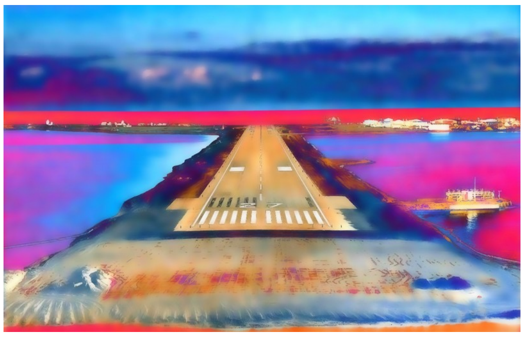


Figure 4: Bilateral Filtering 2

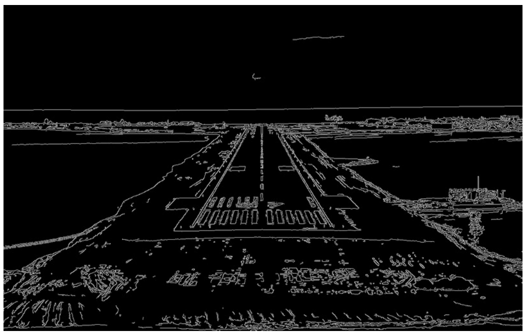


Figure 5: Canny 1

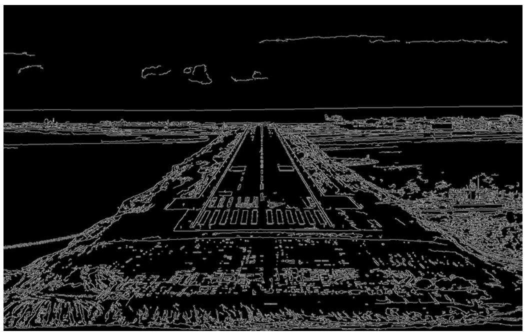


Figure 6: Canny 2

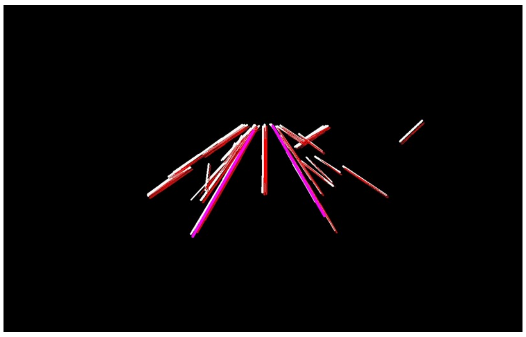


Figure 7: Optimization

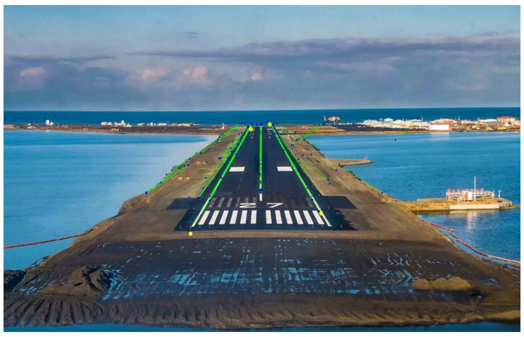


Figure 8: Result

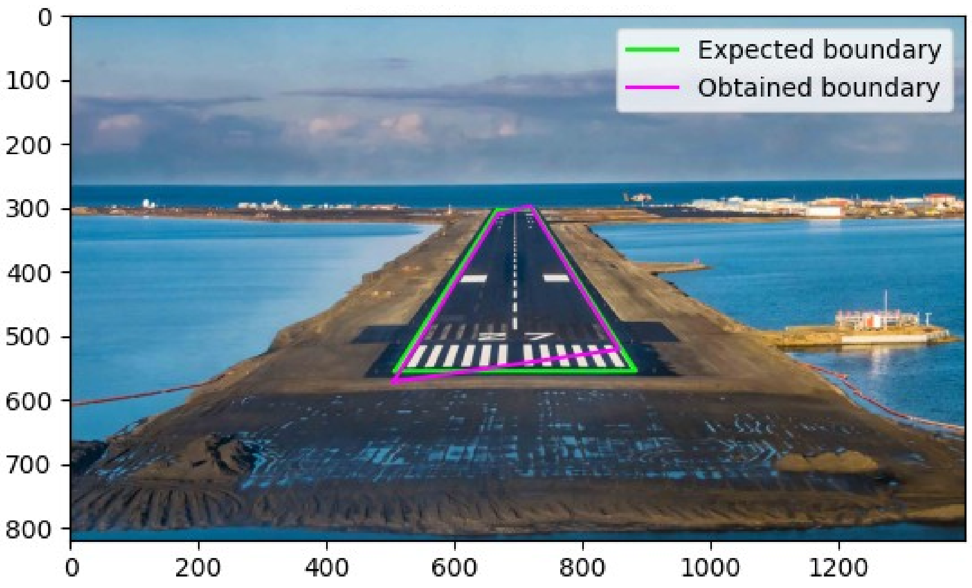


Figure 9: Result Comparison

2.2 Correct perspective

This modules uses both Pillow and numpy frameworks to correct the perspective of the runway given its vertex and expected dimension.

It first rotates the image in order to make the runway's center line vertical. Then, the center offset is determined as the average of the upper and lower start and end lines. Following, a shear transformation makes the lower start line horizontal;

The next step is a perspective transformation that makes the upper end line parallel and aligned to the lower start line. After a crop, the image of the runway is a rectangle. Finally, a resize transformation brings it to the correct aspect ratio. For demonstration, the aspect ratio is artificially wider at a ratio of 2,5 to make the visualization easier.

Following the same example of the runway identification section, the following is obtained:

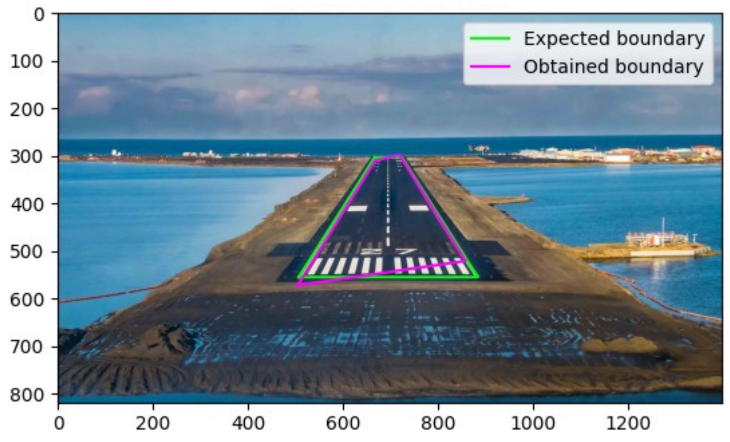


Figure 10: Runway identification result

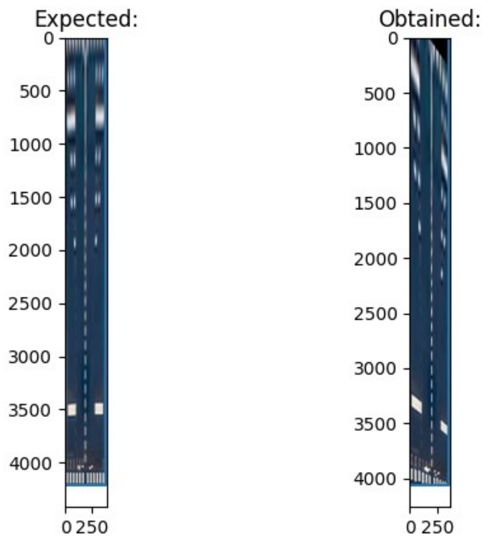


Figure 11: Perspective correction result

2.3 Calculate camera origin

The position of the airplane is found using the runway's four vertex. The implementation uses a routine from OpenCV called calibrateCamera that also executes the function SolvePnP. In order to find the aircraft position, calibrateCamera takes several points with their coordinates in the image as well as in the real world. It is even possible to use several different images.

Hence, a calibration matrix (intrinsic matrix) is obtained regarding the camera in question. If many images are available, one can use this matrix in the equation for solve PnP and find, together with the above mentioned points, the matrices tvecs and rvecs of an image. Those are the extrinsic factors and also the translation and rotation vectors of the image and can be used to convert coordinates from the image space to the real world space.

Next, we make an operation with these vectors accordingly to this article on Wikipedia: https://en.wikipedia.org/wiki/Camera_resectioning#Extrinsic_parameters.

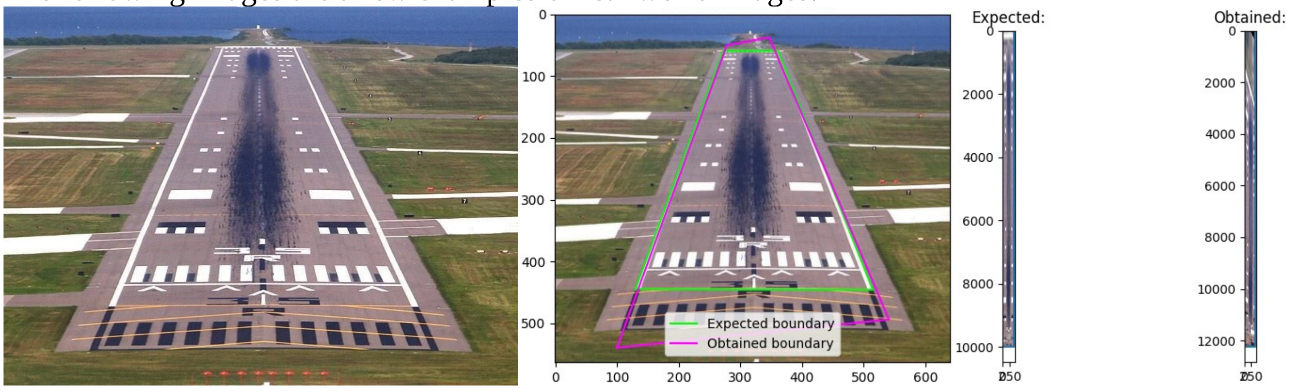
$$C = -R^{-1}T = -R^{T}T$$

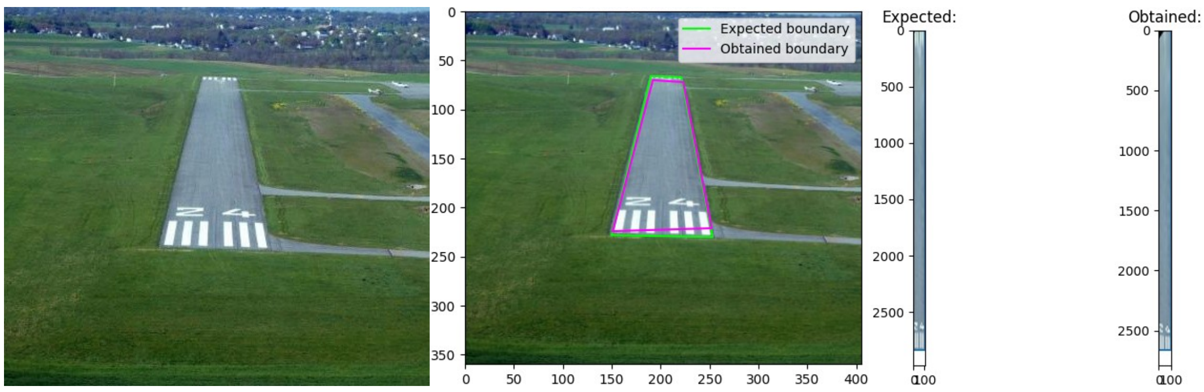
Where C is the position of the camera in real world's coordinates, R is the rotation matrix obtained from the rotation vector with OpenCV's rotate function and T is the translation vector.

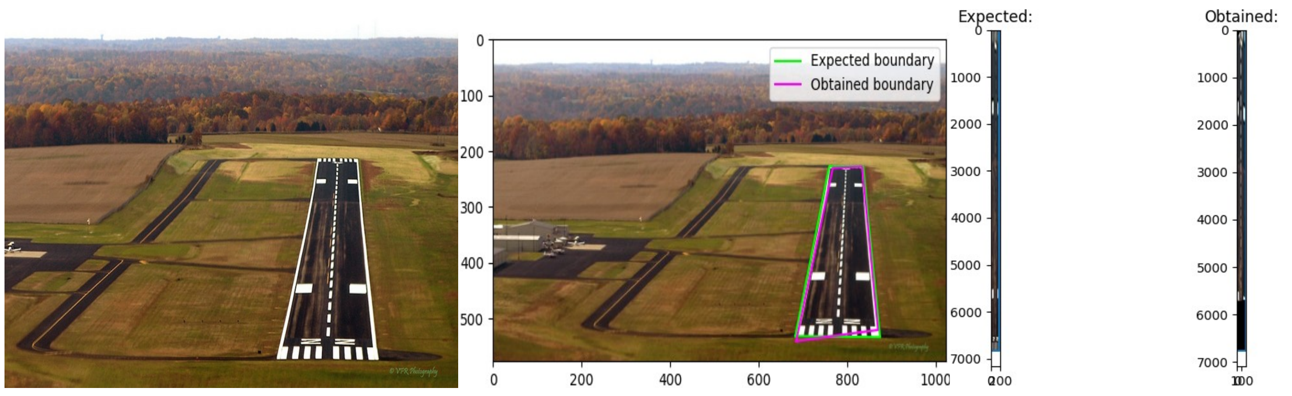
The quality of the values can be improved when various images are used together to obtain the camera calibration matrix. We chose, nevertheless, not to implement something in that sense due to limitation of test images, which are taken with several different cameras.

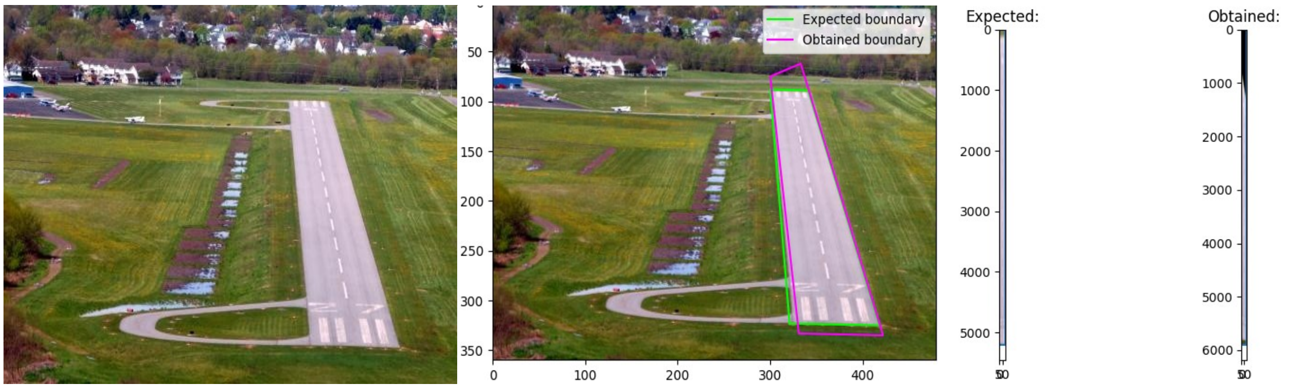
3 Results

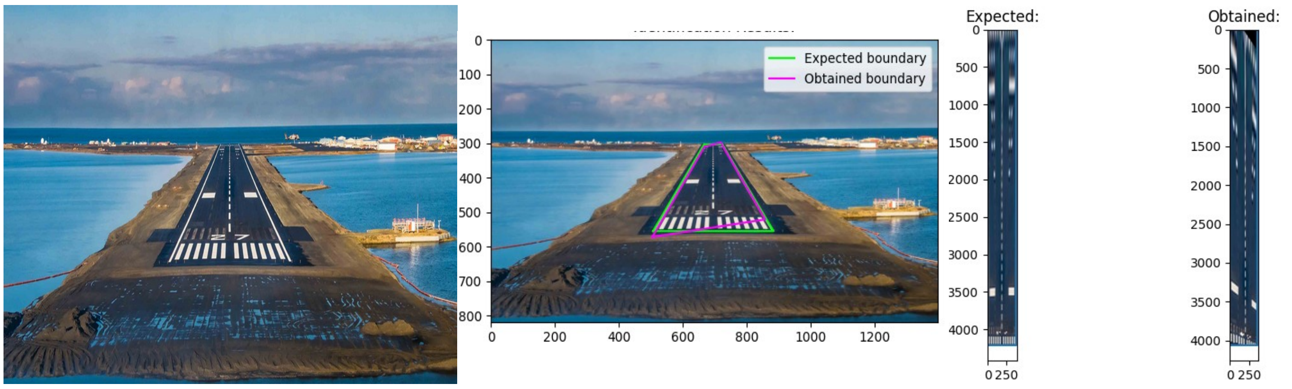
The following images are a few examples of real world images:

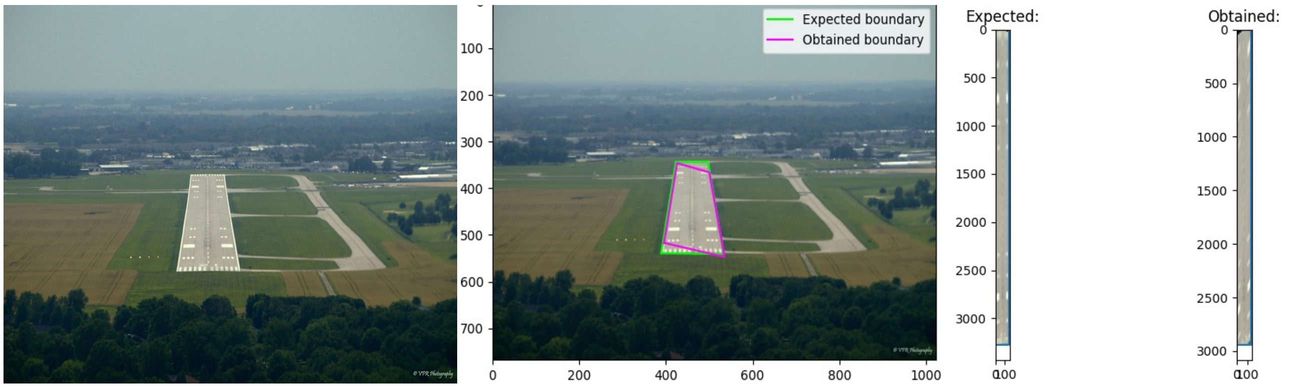


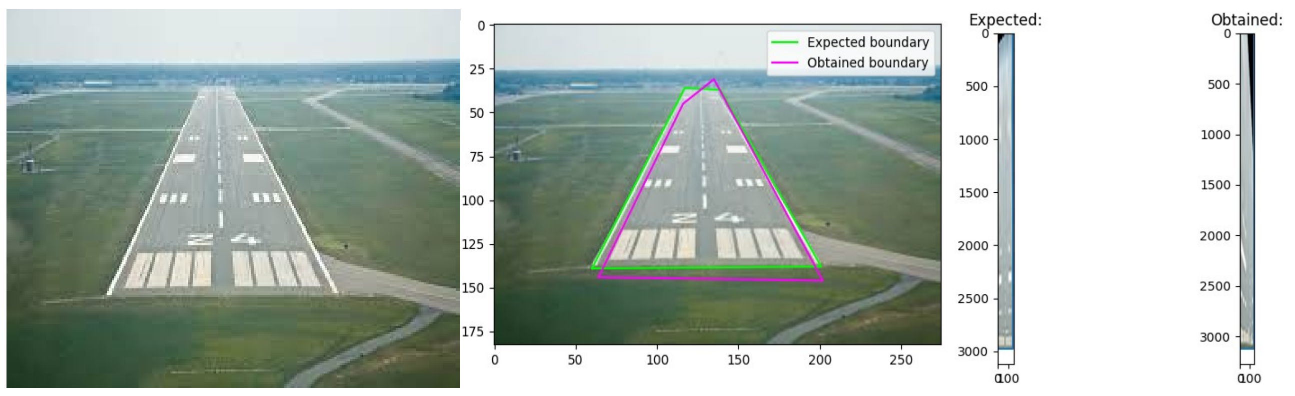


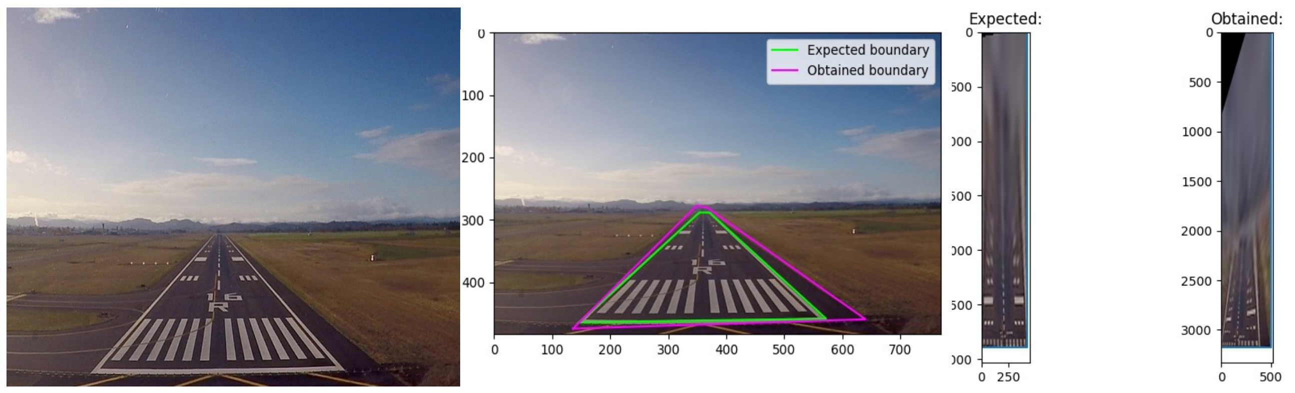

Expected boundary
Obtained boundary
Expected:
Obtained:

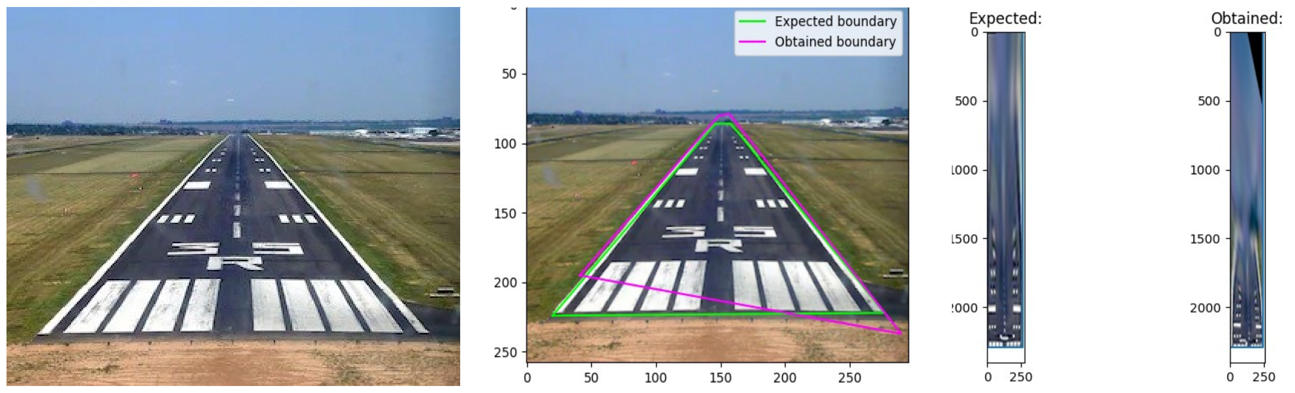

Expected boundary
Obtained boundary
Expected:
Obtained:

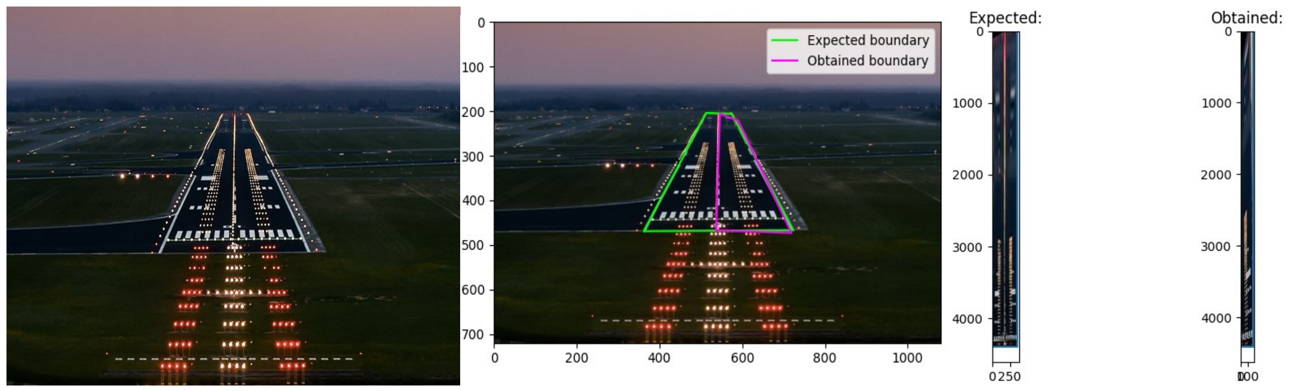

Expected boundary
Obtained boundary
Expected:
Obtained:

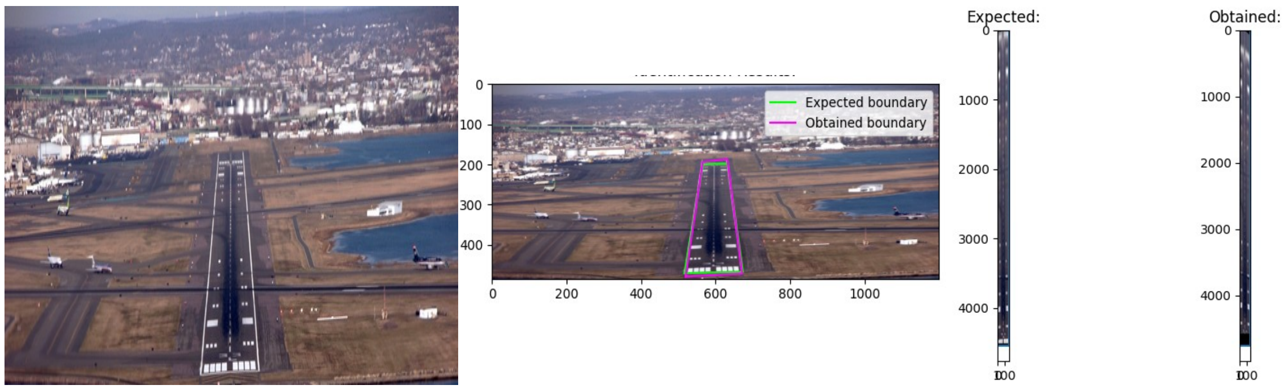

Expected boundary
Obtained boundary
Expected:
Obtained:

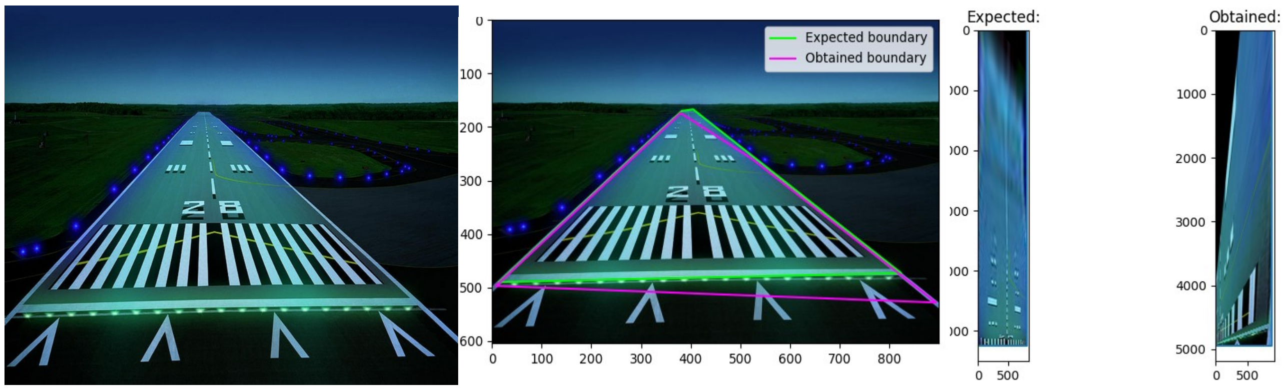

Expected boundary
Obtained boundary
Expected:
Obtained:


Expected boundary
Obtained boundary
Expected:
Obtained:

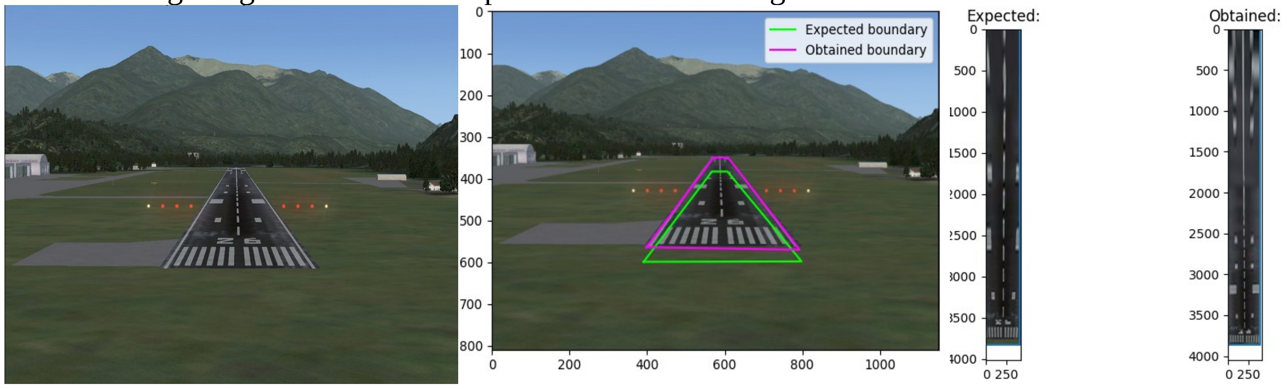

Expected boundary
Obtained boundary
Expected:
Obtained:

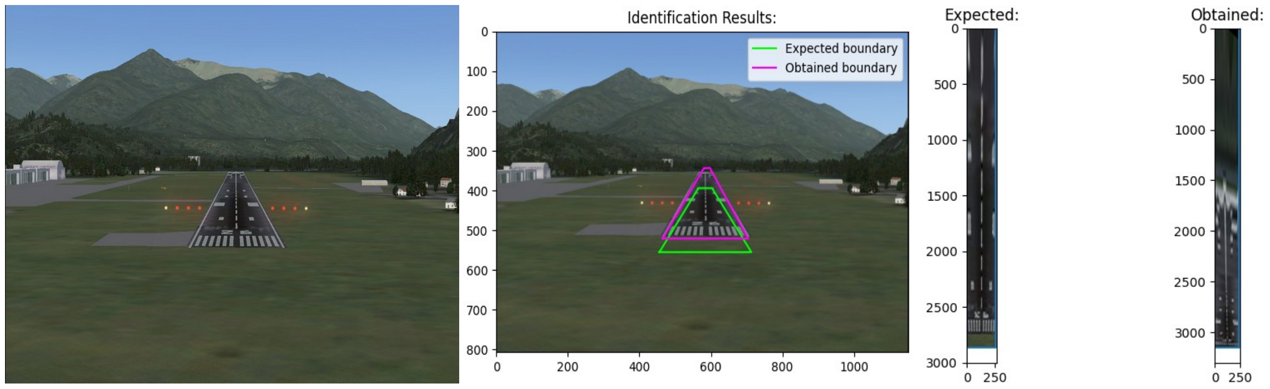

Expected boundary
Obtained boundary
Expected:
Obtained:

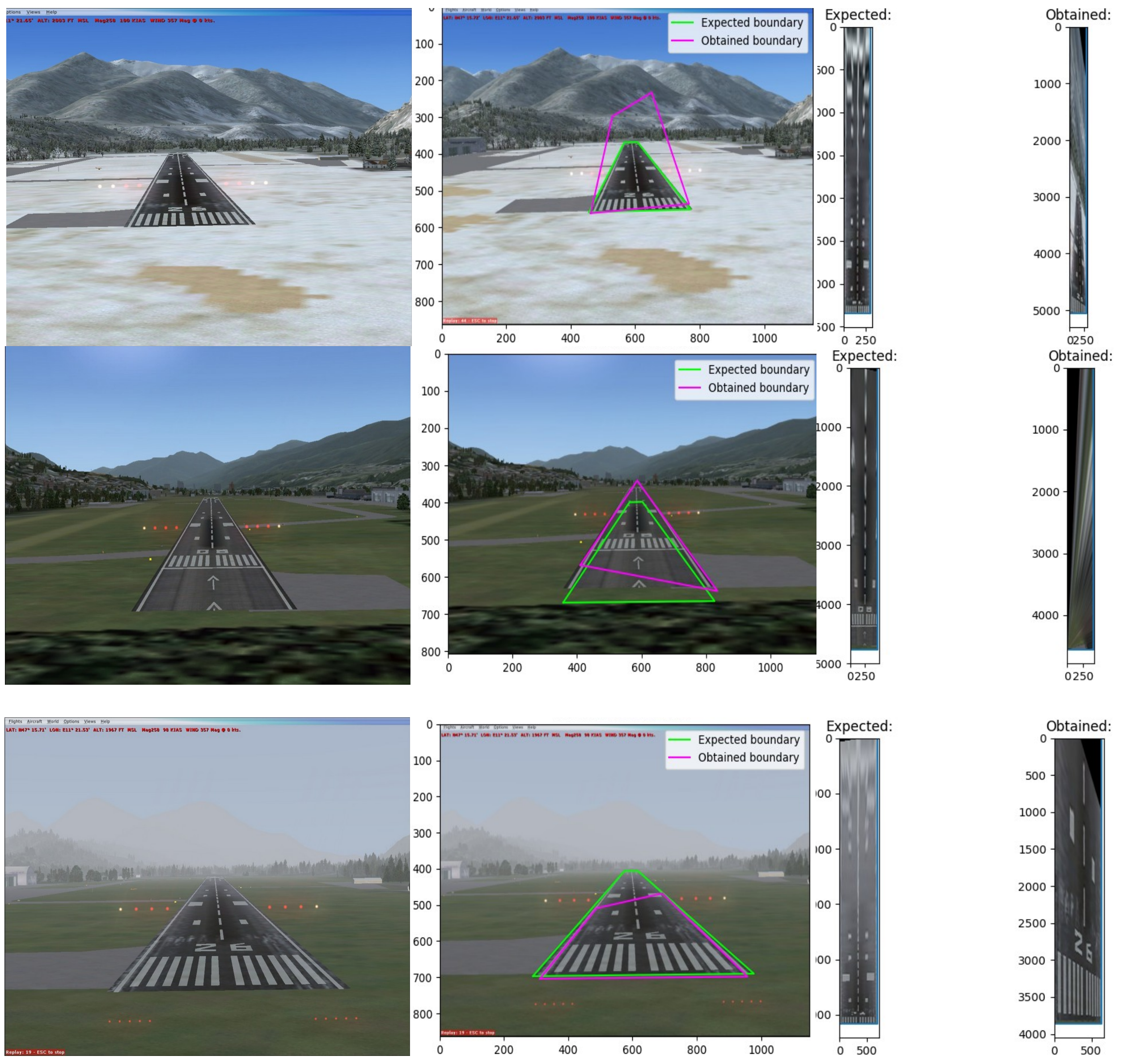




The following images are a few examples of simulation images:







4 Conclusion

Given the limitations of the tools and techniques used in face of the complexity of the problem, the results are quite satisfying. The implementation is able to detect the runway several different scenarios of environment, lighting, morphology of runway, proximity and so on. The correction of perspective works as well, but is notably very sensitive to the precision of the runway detection. Also, in some cases of the real world test images it is possible to notice how non-flat runways affect negatively the result.

One surprising conclusion is that the system does not work as well for the simulated test cases. Although the scenarios are more controlled and homogeneous, the reduced amount of noise and distortions from the real world makes the analysis more complex, ambiguous and prone to failure. We believe a big factor that contributes to this phenomenon is that the simulation images tend to have many more straight lines, as in the horizon, taxiways, buildings, contours and landscape.