REPORT
AUGMENTED REALITY PROJECT

BARTOLI SIMONE

CALIGIANA PAOLO

SUMMARY

1. INTRODUCTION

This work describes an Augmented Reality System to superimpose an artificial layer onto a recognized target object. The specific case study is the following: given an input video showing the well-known "Multiple View Geometry in Computer Vision" book, our goal is to draw a realistic image on it. Thus, the superimposition must be robust to orientation changes, as well as translation and bright changes.

The report is structured as follows: the next section illustrates the technologies and libraries used to develop the project; the third section describes the methodology applied and reports the main functions implemented in our code to reach the solution; the fourth and final section shows the obtained result including some consideration about the two different approaches to solve the problem.

2. TECHNOLOGY

2.1 PYTHON AND JUPYTER NOTEBOOK

The programming language used is Python, well suited to leverage the power of computer vision libraries. The chosen development environment is Jupiter Notebook.

2.2 OPENCV

OpenCV is the powerful python library and the main tool to develop Computer Vision applications. It has been used for the development of the whole project, to analyze the input video, recognize the target object and superimpose the Augmented Reality layer on it.

2.3 NUMPY

In order to process images, access the pixel values, and in general make mathematical operations between arrays, the Numpy library has been imported in the project.

2.4 MATPLOTLIB

This library has been used to plot images showing the results, step by step, of our algorithm.

3. PROJECT

The project task requires to develop two different solutions: the first one consists in matching the key points of the given reference frame and the following frames read from the video (F2R – Frame to Reference). The second one computes the matches only between successive frames (F2F – Frame to Frame).

There is a common set up for both, which consists in managing and preparing the first frame (reference frame). Indeed, in order to focus solely on the key points in the book, we have warped the reference frame with its mask to be able to detect just the key points belonging to the book. It is worth to highlight that, before this operation, the mask has been eroded in order to filter out the artefacts belonging to the background. After that, using the compute_kp() function (that we will explain later), we got the key points and computed the relative descriptors. Once obtained the information about the key-points and descriptors of the first frame, we were able to call the Frame_to_reference() and/or Frame_to_frame() functions.

Though the code of these two solutions are quite similar, they are discussed separately, to better describe them.

3.1 F2R – FRAME TO REFERENCE

This section illustrates the first solution, that implements the frame_to_reference() function. It makes use of a set of utility functions, described below.

3.1.1 ERODE(IMG,K)

The erode() function (see table 1) allows to make easier the calling of the erosion() function within the openCV library.

Input parameters:
• img : the image to be eroded • k: the parameter to set the kernel ((2 * k + 1) * (2 * k+1))
Output:
• img_eroded: the function returns the new eroded image

Table 1: input/output parameters of the erode function

3.1.2 COMPUTE_KP(IMG)

The compute_kp() function (see table 2) initializes the SIFT operator of the OpenCV library in order to detect the key-points within the input image and compute the relatives descriptors.

Input parameters:
• img: the image where to seek for the key-points
Output:
• img_kp: a new image with detected key-points directly drawn on it • kp: the list of detected key-points • des: the list of descriptors related with the detected key-points

Table 2: input/output parameters of the compute_kp function

3.1.3 PLOT_IMAGE(IMG, TITLE)

This function (see table 3) allows to ease the plotting process, defining once and for all, the parameters of the pyplot library.

Input parameters:
• img: the image to be plotted • title: the title explaining the image content

Table 3:input parameters of the plot_image function

3.1.4 HARRIS(IMG)

This function (see table 4) simply applies the Harris corner detector to the given image, in order to detect the four corners of the image. As we will see, this function is not fundamental for our project, it just helps us to define an enclosing rectangle that will surround the detected match. The four corners are sought within the corner points extracted by the Harris detector, using some useful functions provided by the OpenCV library.

Input parameters:
• img: the image where to seek for the edges
Output:
• corners: a list containing the four corners of the input image

Table 4: input/output parameters of the harris function

3.1.5 DRAW_AUGMENTED_IMAGE(IMAGE, IMAGE2PROJECT, WHITE_MASK, M)

The draw_augmented_image() (see table 5) function, is the one that takes care of superimposing the Augmented Reality layer on the image. Firstly, it uses the M matrix (representing the computed homography) to transform the image2project and the relative white_mask in order to let them lay on the same planar surface of the given image. Then, the artefacts are filtered out thanks to an erosion of the white_mask. Finally, the image2project is drawn onto the given image.

Input parameters:
• image: the image where to superimpose the augmented image • image2project: the augmented image to be drawn • white_mask: the white mask of the image2project helpful for the superimposition • M: the homography matrix
Output:
• warped: the obtained augmented image

Table 5: input/output parameters of the draw_augmented_image function

3.1.6 DRAW_ENCLOSING_RECTANGLE(IMAGE, M)

After the book detection within a frame of the video stream, it is possible to use this function to draw an enclosing rectangle around it. Without this operation, it would not be possible to finely appreciate the differences between the two solutions (F2R and F2F): one of these is more precise during the object tracking from different angles and the superimposition of the rectangular outline helps to show it. The latter is built in three steps, as follows: firstly, the harris() function extracts the white_mask corners of the reference image (see table 6); then, the M homography is applied to the vertices and, finally, the rectangle is drawn onto the given image.

Input parameters:
• Image: the image where to draw the enclosing rectangle • M: the homography
Output:
• Img_rectangle: the image containing the enclosing rectangle

Table 6: input/output parameters of the draw_enclosing_rectangle function

3.1.7 FRAME_TO_REFERENCE(VIDEO_PATH, IMG_REF, REF_KP, REF_DES)

The frame_to_reference() function (see table 7) uses the above described functions to realize each step of the algorithm. The base idea is to process each frame of the video stream, looking for the key points and matching them with the given ones of the reference frame. This operation is carried out by a brute force method (the knnMatch() function of the OpenCV library).After that, the result is filtered by a threshold, discarding most of the bad matches and keeping just the good ones (the threshold used is T = 0.75). Thus, if the number of good matches is over a defined lower bound (in our case 30 good matches), they are used to compute the homography. The latter is then passed as an input to the draw_augmented_image() function to superimpose the augmented layer.

Input parameters:
• video_path: the path name of the video stream • img_ref: the reference image with the plotted key points • ref_kp: the list of the key points within the reference image • ref_des: the list of descriptors related to the key points

Table 7: input parameters of the frame_to_reference function

3.2 F2F – FRAME TO FRAME

This second solution is quite similar to the first one. Indeed, it uses the same utility functions to realize the several steps of the algorithm. However, thanks to the introduction of the update_reference_image() function, we are able to detect matches and compute homographies between successive frames.

3.2.1 UPDATE_REFERENCE_IMAGE(IMG, IMG_MASK, M)

This function allows to update the reference frame, replacing it with the current read frame. After that, it transforms the reference image mask with the M homography (see table 8) and computes again its key points and its descriptors. It is worth to mention that we could have used the already computed descriptors in the previous matching phase, however, since we would like to match only key points belonging to the book, we have decided to recompute them, with an obvious increase of computational effort.

Input parameters:
• img: the current frame that must replace the reference image • img_mask: the white mask of the reference image • M: the homography
Output:
• query_img: the updated reference image • ref_kp: the list of key points within the new reference image • ref_des: the list of descriptors related to the key points

Table 8: input/output parameters of the update_reference_image function

3.2.2 FRAME_TO_FRAME(VIDEO_PATH, IMG_REF, REF_KP, REF_DES, IMG_MASK)

The frame_to_frame() function (see table 9), differs from the previous one (F2R) in the introduction of the update_reference_image() function and the managing of the M homography matrix. The latter has to be applied to the augmented image for its projection, introducing, in this solution, a greater complexity thanbefore. In fact, the given augmented image lays on the same plane of the reference image, so, if in the first case we compute, for each frame, the homography with respect to the reference frame (which is also applicable to the

augmented image), in this second case we have the homography with respect to the previous frame, which is not applicable to the augmented image. To solve the problem, our solution takes advantage of the linearity property of the homography matrix to update, step by step, the current homography by a dot product between the previous one and that just computed. In doing so, it is possible to pass this new homography matrix M to the draw_augmented_image() function to superimpose the augmented image with the correct transformation. Finally, the update_reference_image() function is called to update the reference data and the loop starts again till the end of the video stream.

Input parameters:
• video_path: the path name of the video stream • img_ref: the reference image with key points plotted • ref_kp: the list of key points within the reference image • ref_des: the list of descriptors related to the key points • img_mask: the white mask of the reference image

Table 9: input parameters of the frame_to_frame function

4. CONCLUSION

This last section contains some consideration about the reached results by applying the two different solutions: F2R and F2F.

4.1 RESULTS OF F2R SOLUTION

The F2R solution guarantees a truly remarkable result, with a realistic superimposition of the augmented image. To give a clear example of the algorithm in action and show its robustness (to translations, rotations and rapid brightness changes) over time, we have sampled two different moments of its execution. The first one is the computation of the 101 frame, while, the second one is that of the 201 frame.
In the first sampling (Fig.1) we have plotted the 30 best matches between the key points in the reference image and the key points in the current frame. It is noticeable that some errors occur during the matching of corresponding descriptors (see red circles in the image below).

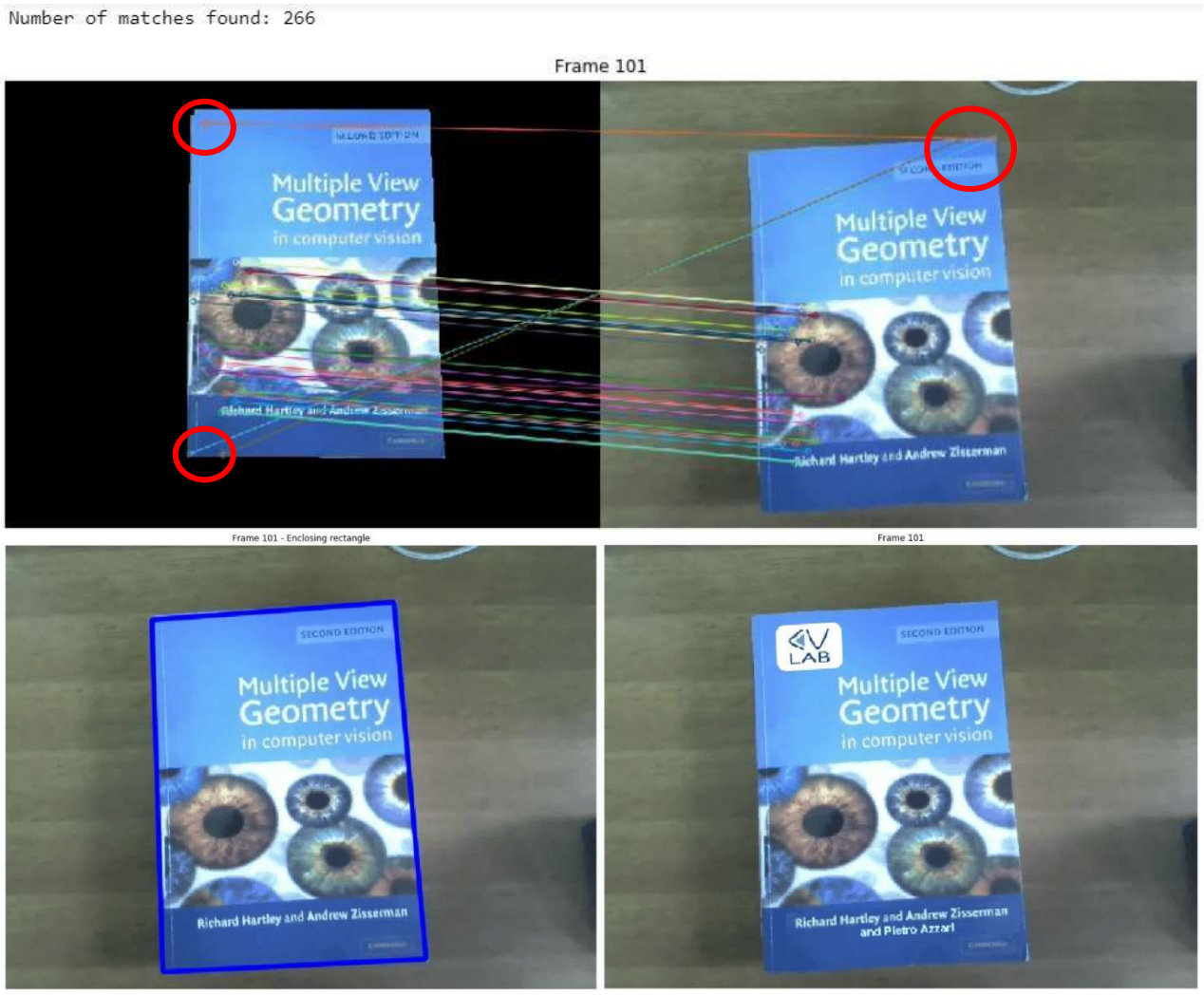

Fig 1: plot of the F2R algorithm execution at frame 101

However, the second sampling (Fig. 2) shows that the computed homography works pretty well and the enclosing rectangle surrounds almost perfectly the book.

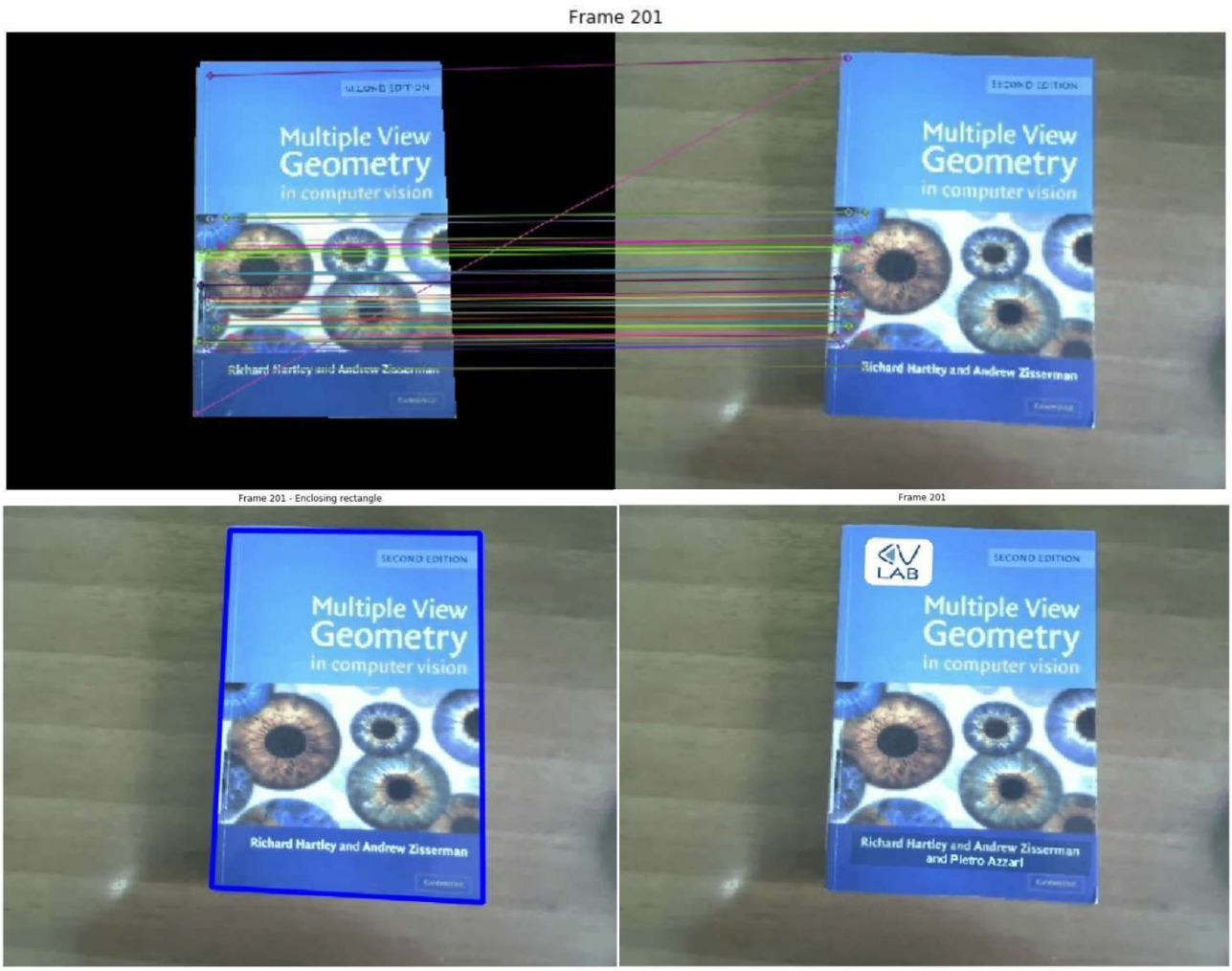

Fig 2: plot of the F2R algorithm execution at frame 201

In conclusion, 100 frames later the algorithm is robust to changes and the computed homography allows to superimpose a realistic augmented image.

4.2 RESULTS OF F2F SOLUTION

Unfortunately, with the F2F approach we have not achieved results as good as with the F2R approach. The problem shows up after the processing of many frames, and it goes worse and worse toward the end of the video stream. This behaviour is due to the computation, frame by frame, of the dot product between the previous homography and the new one. In fact, every time an homography is computed, we assume a small error. Even if this is not considerable in the first frames, going farther with the execution, the sum of each single error contributes to mislead the homography from the correct transformation. Moreover, this solution is slower than the previous one, due to the double computation of the key points for each frame. This problem could be easily overcome if we didn't want to focus only on those key points belonging to the book.

In order to make a proper comparison with the previous solution, also in this case, we have sampled two different moment in the algorithm execution. The first one is the same as before, the computation of the 101 frame, shown in the image below (Fig.3).

```
Number of matches found: 321
```

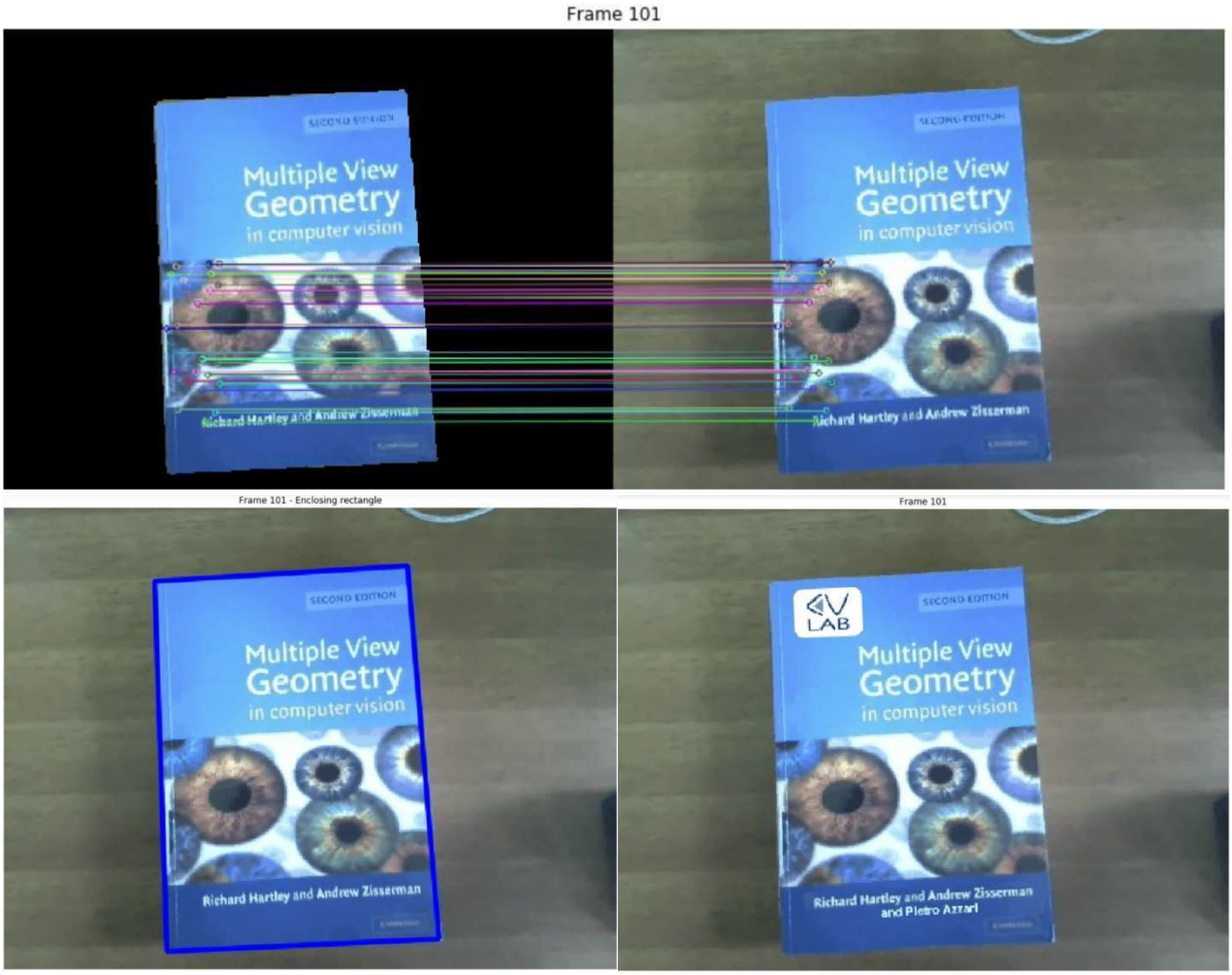

Fig 3: plot of the F2F algorithm execution at frame 101

It is noticeable that the number of good detected key points with this algorithm is greater (321) than before with the F2R algorithm (288). Moreover, within the first 30 good matches, no errors occur.
To better appreciate the homography error previously described, in this case, the second sampling has been made later at the 301 frame (Fig. 4), instead of 201 (as we have done for the F2R algorithm).

```
Number of matches found: 322
```

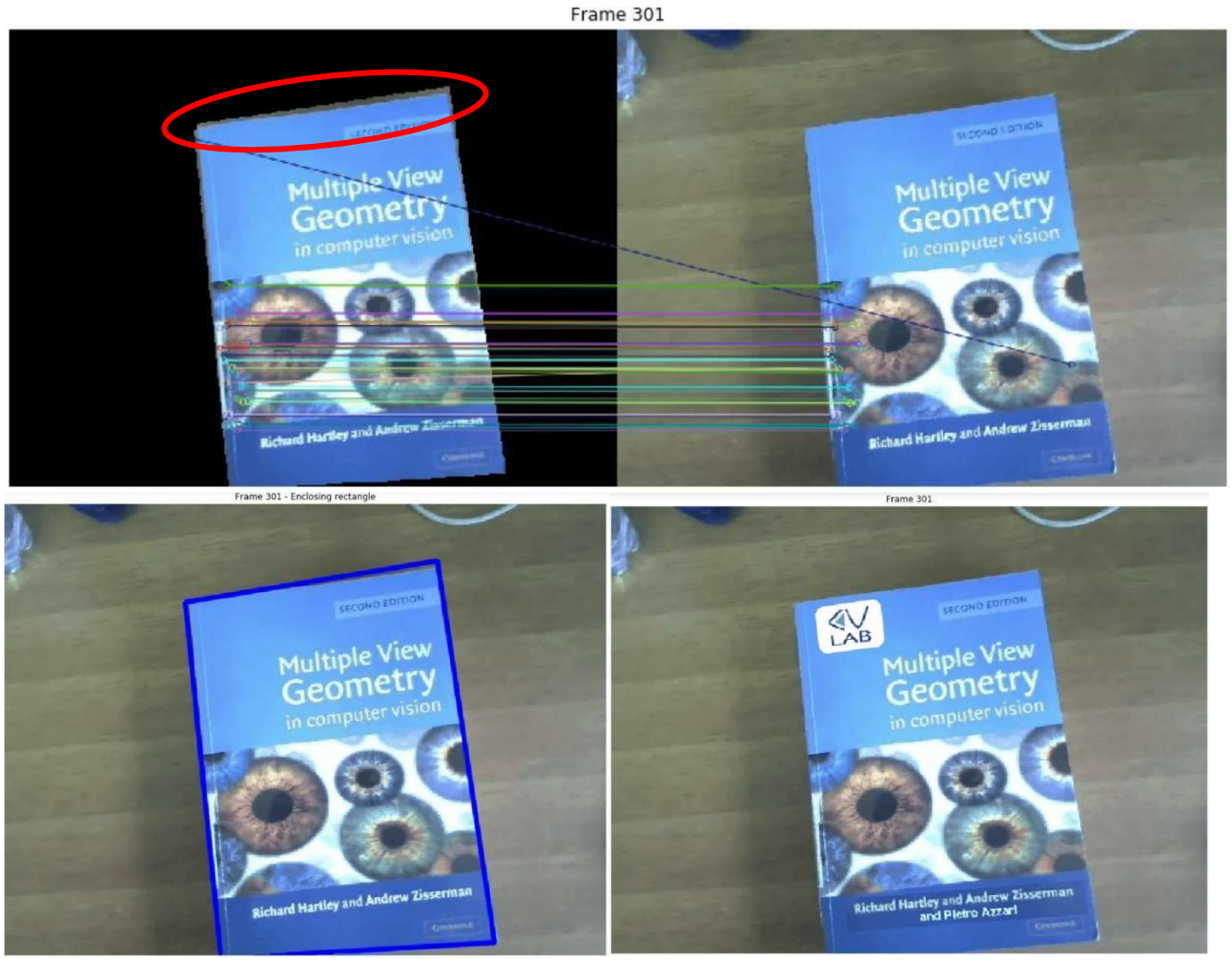

Fig 4: plot of the F2F algorithm execution at frame 301

The error can be clearly seen both in the drawing representing the matches (within the red circle) and in the drawing with the plot of the enclosing rectangle. Despite this error, the result is still acceptable when it comes to superimpose the augmented image. However, if the video stream had been longer, the result would have been worse.