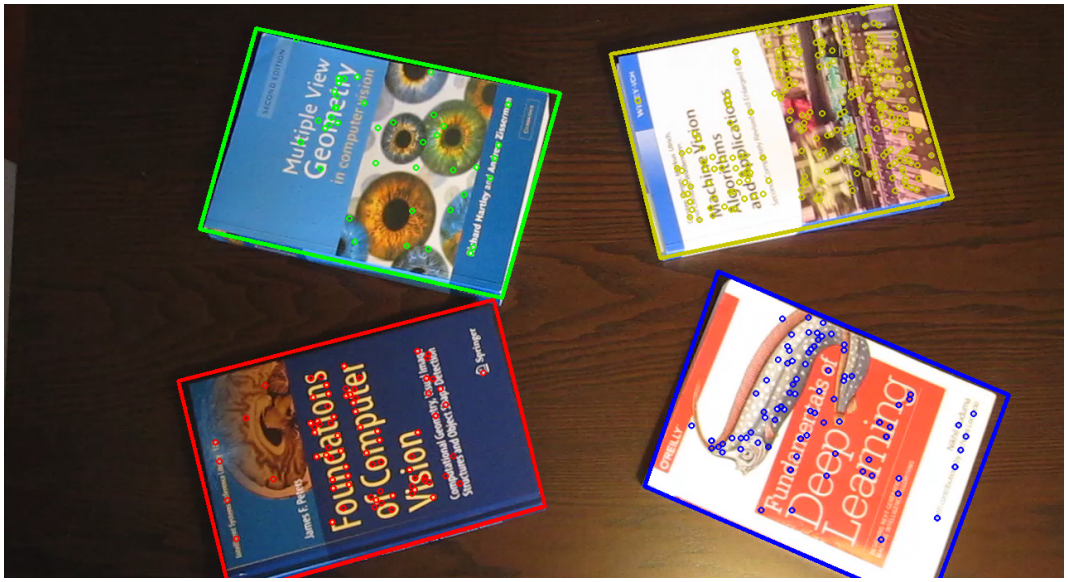
Object tracking

Viaggi Alessandro

Mautone Alberto

The code starts with the uploading of all the pics of single books given, thanks to the *glob(...)* function (**NB**: path need to be changed with the one with the template pictures on your PC) putting them into an **array** called fn. I define some **variables** that I'll use later. Most of them are used for the **books tracking** in the video, while other are needed to guarantee the best **matching** between every single book and the corresponding object in the video. We also define an int equal to 0 called "count" that allows the program to distinguish whenever tha elaboration of the first frame has ended, so it can go on tracking books on the video. Follows the capture of the first frame of the video (allowed by the statement (if(count==0)), and an algorithm for object matching: using **SIFT** features we extract keypoints and descriptor and we filter all the obtained matches first basing on the **Lowe's ratio test**, and then using **inliners** given by the mask obtained thanks to the *findHomography(...)* function. At first we tried to extract keypoints using **ORB** features, but the result we got was not the one we were expecting: in fact some books did not match properly with the corrisponding book on the video. Going on we detect **object corners** first putting them in a vector called "obj corners" and then we just fill it with the corners of every **single-book image**. We also define a scene corners vector that's gonna be filled later on thanks to the *perspectiveTrasform(...)* function, that also uses the **homography matrix H** computed before. At this point we have all we need to draw a **rectangle** around each located book, and we do that thanks to the *line(...)* function, that takes as parameter 4 point that actually are the corners of each book on the first frame of the video (apart of the first parameter which is the Mat we're drawing on). After showing what we've got until now, the program waits for an input from the user to go on detecting every book in the whole video.



All four books detected in the first frame of the video.

At this point we have to track each book in the video while frames keep going on: the count value changes to 1, so the if statement (if(count==0)) is not valid anymore, so we can get into the second part of the code, where the "*else*" starts. After a check that the frame is not an empty Mat, we define some useful variables: most of them are vector of vector since we have to track 4 books at **the same time**, and we want to store every information in a single variable. The tracking starts (for each of the books) with the *calcOpticalFlowPyrLK(...)* function that actually calculates the flow between 2 image and allows to track the movement of an object by implementing the **Pyramid Lukas Kanade optical flow**. Again we need *perspectiveTrasform(...)* (so we compute homography matrix in a previous step) to get all the right corners and, after an **inliners check** we draw again each rectangle around each object, once again thanks to the *line(...)* function. After updating some variables (changing previous keypoints to next keypoints and previous corners to next corners for the *calcOpticalFlowPyrLK(...)* function) for the following round of the for cycle, we show what we'got in a window called "*Result*" and after waiting 1ms we pass to the next frame and we repeat the previous steps to get the new rectangles on the **new frame**. The program ends when the last frame is processed, so the following one in an **empty Mat**, so the condition (*if(!frame.empty()*) is not valid anymore. Hence the program returns "*Press any key to close*" to the user, and ends as soon as the user actually presses any key on the keyboard.