TEMPLATE MATCHING

The given assignment was to develop a MATLAB program where the algorithm "Template Matching" is implemented. Given two images provided by a stereo vision system, the task of the program is to identify correspondent pixels in the right image, given target pixels defined in the left image.

ALGORITHM EXPLANATION

The idea behind the template matching algorithm is to define a fixed frame containing the target pixel in the left image. Then, a "sliding" frame with the same dimensions of the fixed frame is defined in the right image. This moving frame slides along the rows and the columns of the right image where we have to find the pixel corresponding to the target pixel in the left image. Since every single pixel of an image is coded as a vector of 3 values (RGB, for color images), it is possible to compare the values of the single pixels of the sliding frame with the values of the pixels of the fixed frame. Every position assumed by the sliding window within the right image correspond to an iteration of the algorithm: in every iteration, pixels of fixed and sliding frame are compared through a function called "similarity" or "distance". The three most used are:

1) Sum of squared difference
2) Sum of absolute difference
3) Maximum difference

All of them are used to compute a similarity value: at every iteration, the similarity value of the sliding frame in the current position is compared with the similarity value of the previous position. If the value is smaller than the previous, the position of the sliding frame is updated. At the end of the algorithm, the position of the sliding frame with the minor similarity value is returned. The found frame is the one corresponding to the fixed frame, in the right image!

For a stereo vision system with no vertical displacement between the two images, the correspondent sliding frame in the right image is moved towards left with respect to the fixed frame in the left image.

At the end, the corresponding pixel in the right image is found knowing its position within the sliding frame (the relative position of the target pixel within the fixed frame is the same also for the found frame in the right image).

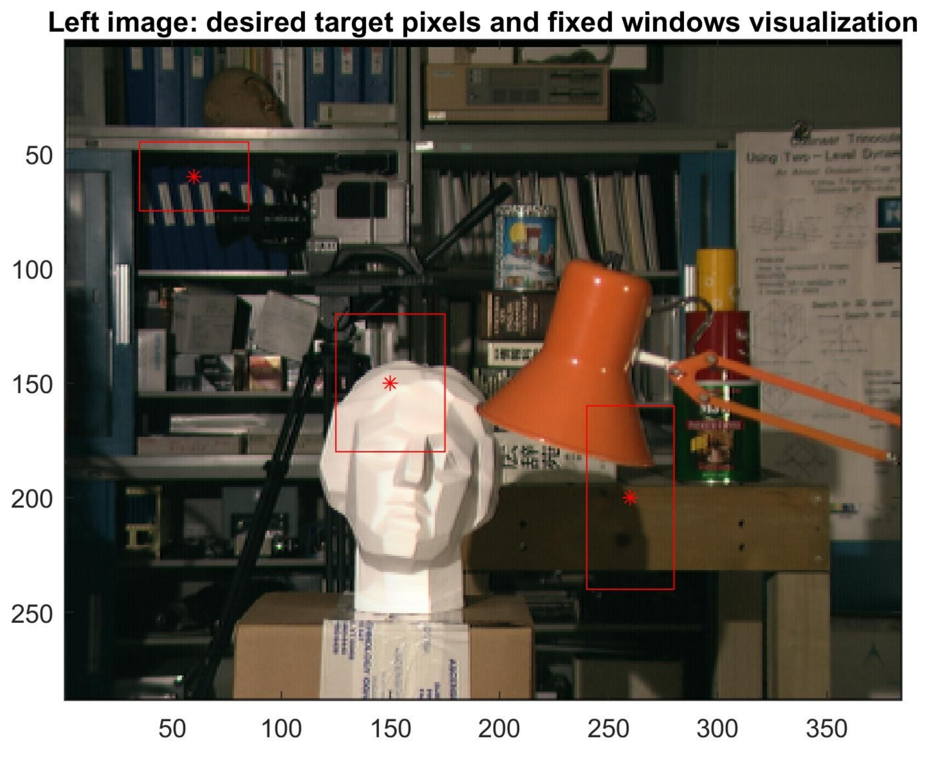
ALGORITHM IMPLEMENTATION – THE CODE

In the presented code, the template matching algorithm is realized. The program enables the user to find at the same time multiple target pixels.

Two arrays containing the x and y positions of the target pixels must be defined. After that, for each declared target pixel a fixed frame must be defined. In the code, it is possible to define a rectangular fixed frame, but for the sake of simplicity, the fixed frame is defined only by its base and height: automatically the program define the fixed frame in order to have the target pixel in its center. This has been done to be sure that the fixed frames contain the wanted target pixels. Notice that the program will automatically send an error message if a pixel or a frame dimension exceed the possible values.

Knowing the positions of target pixels and the positions of the correspondent fixed frames, trough a simple subtraction it is possible to define the relative position of the target pixels within the fixed frames. These “relative positions” are fundamental for the identification of the correspondent pixels in the right image.

After this initial procedure, the program displays the left image with, in red, the defined target pixels and fixed frames, and opens a figure for every declared target pixel.



At this point, every defined pixel is treated independently, and so a for loop iterates as many times as the number of the pixels to define.

The following step is the initialization of the sliding frame: by default, it starts at the top-left corner of the right image and it moves from left to right and from up to down.

Also, the variable `lowest_similarityscore` is initialized as `inf`, so that for sure this variable will be updated during the first iteration of the algorithm.

The core of the program is represented by 5 “for loop”s. A brief explanation: the exterior loop (`m` variable) is needed to make the sliding window move along the rows of the right image; the next loop (`n` variable). For both these two loops, the upper value of `m, n` variables is determined by the subtraction between the y and x dimension of the right image and the y and x dimension of the sliding frame.

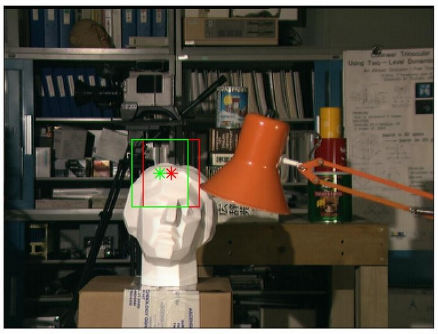
The next 2 loops (i,j), are needed to compare every single pixel in the sliding window with the pixels in the fixed window. Pixel(i,j) is the pixel in position i,j in the frame.

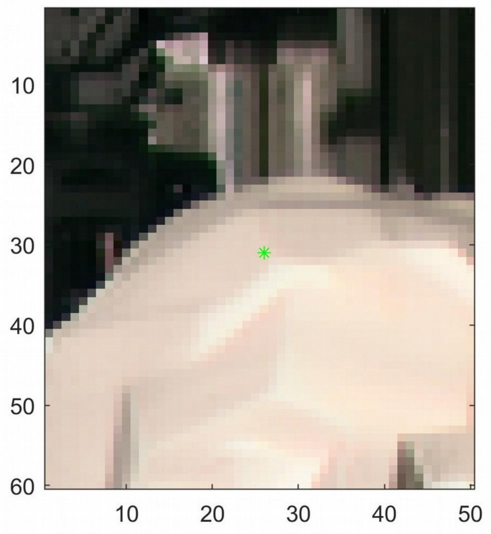
The inner loop makes three iterations, and is needed to make the comparison between the three RGB values associated to the pixel. The values are stored in 3-dimensional array.

Before passing to another position of the sliding frame, once the comparison between pixels of the frames has been done, the similarity score is computed as the sum (or the max value) of the `value(i,j,l)` representing the similarity between the pixels. The lowest the value, the most similar the sliding frame is to the fixed frame. For this reason, the overall similarity value assigned to the sliding frame in the generic position (`m,n`) is compared with the previous value: if the new value is less then the previous, `lowest_similarityscore` is updated with the current value and current m,n positions of the frame is stored in the `s_pos, r_pos` variables.

At the end of the overall algorithm, s_pos and r_pos contains the coordinate of the best found sliding frame. Knowing the dimensions of the sliding frame (same of the fixed one), the sliding frame is univocally identified and the pixel corresponding to target pixel is also determined knowing its relative coordinate (same of the fixed frame) in the best frame.

At the end, for each pixel, a figure containing the right image with the found pixel (in green) and the corresponding best window is displayed. It has been also added, in red, the rectangle corresponding to the position of the fixed frame, in the left image: in this way you can visually evaluate the effective displacement between the two images.

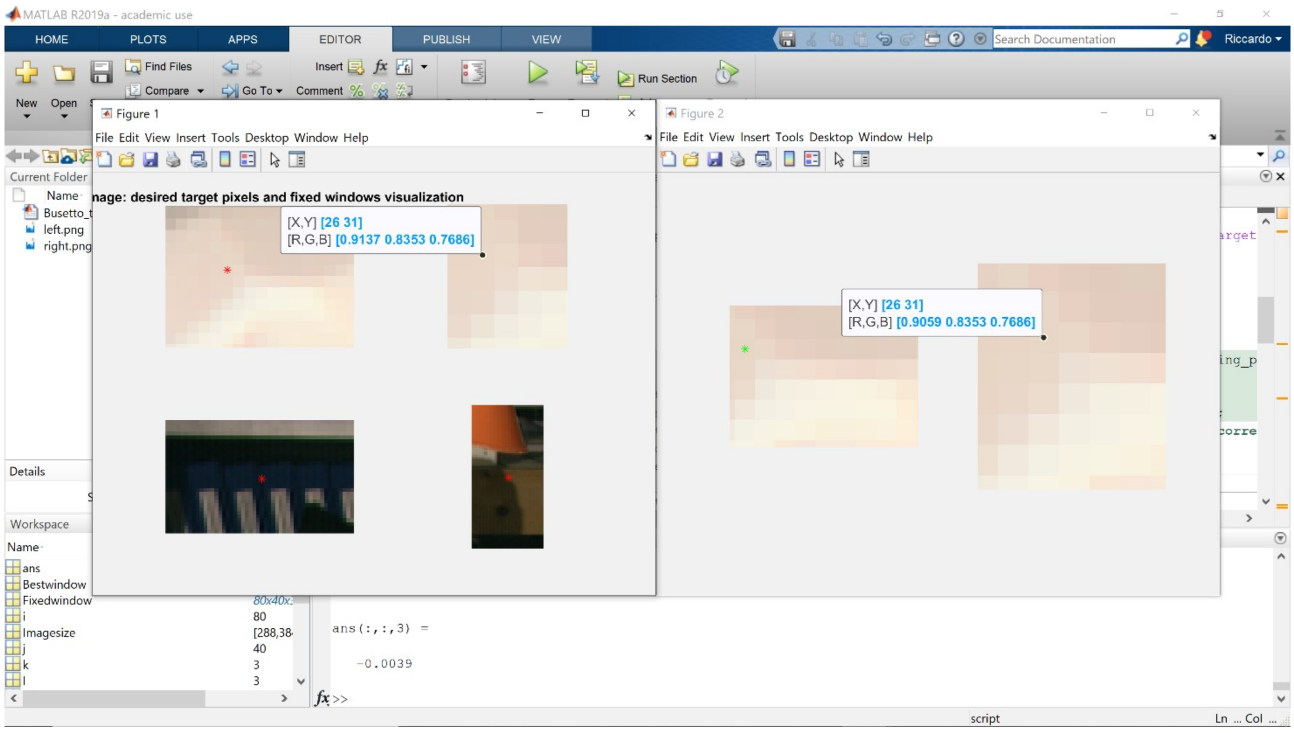




OBTAINED RESULTS

In this section, overall performances of the algorithm are analyzed. The algorithm has been tested with different values for the pixels and for the windows dimensions (always manually inserted) and it works fine, because in a reasonable computing time, it returns the correct pixel values (the computing complexity increases proportionally to the number of pixels to be find).

However, in the code have been implemented some “checks” to evaluate the correspondence between find pixels and target pixels. As a matter of fact, subtracting found pixels values from target pixels values, if there is perfect match the result should be a column vector of 3 zeros. This values are not all perfectly zeros because there are some small errors related to computational precision of the software itself. However, pixels have been manually compared trough visual inspection and they match. Example:



nage: desired target pixels and fixed windows visualization

```
[X,Y] [26 31]
[R,G,B] [0.9137 0.8353 0.7686]
```

```
[X,Y] [26 31]
[R,G,B] [0.9059 0.8353 0.7686]
```

```
Name
ans
Bestwindow
Fixedwindow      80x40x
i                80
Imagesize        [288,38
j                40
k                3
l                3
```

```
ans(:,:,3) =

  -0.0039

fx >>
```

Some notes on the code:

1) Fuction `im2double` as been used to obtain better precision for the computing of the pixel values and their comparison
2) functions `imshow` and `imagesc` as been used to display 3D array in the corresponding colored images. `Imagesc` has been used for its property of having scaled reference axes and display the image with adjusted proportions even if they are made of few pixels.
3) The `rectangle()` function has been used to display windows dimensions.

FURTHER ANALYSIS

1) As aforementioned, there are three commons way to determine the similarity score. In the code, there are the commands to use all of the three typologies of functions, and some tests have been run in order to compare the performance of the three. Results are summarized in the following table:

	PRECISION	COMPUTING TIME
SUM OF SQUARED DIFFERENCE	HIGH	HIGH
SUM OF ABSOLUTE DIFFERENCE	HIGH (but less than ssd)	MEDIUM
MAXIMUM DIFFERENCE	LOW	LOW

   While ssd and abs difference have proved to be very robust and in rare occasions fail to find the target pixel (despite a heavier computational effort), maximum difference is not always able to correctly evaluate the corresponding pixel, especially if the target pixel is in a dark area of the image. Its precision is not reliable.

2) Dimensions of the windows strongly affects the overall computational time required by the algorithm. Reason is simple to understand: the bigger the windows, the lowest the number of position of the window to evaluate within the right image. If for example we select a window with horizontal and vertical dimensions equal to (horizontal-1) and (vertical-1) dimensions of the overall image, the possible positions of this frame within the image are only four! However, increasing too much windows dimensions means reduce the precision of the algorithm because similarity scores between sliding frames will be very close.