Assignment 2: Photo tour

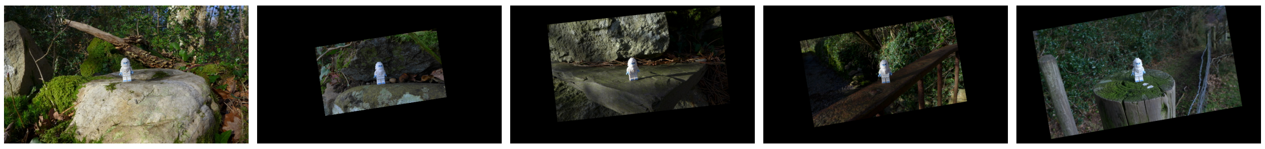

Figure 1: A series of photographs that were aligned to show the minifig in the same position.

Your task is to create a program that aligns a series of images containing a common object so that the images can be combined into an attractive animation in which one photograph blends into another. Figure 1 shows an example of a few images that were transformed so that the Lego minifig is at the same location in each image. Check also an example animation in Blackboard.

1 Specification

Your program will be executed from the command line as follows:

```
photo_tour [x1 y1 x2 y2] img1.jpg [img2.jpg] ...
```

If the first parameter starts with a number, the first four parameters should be interpreted as the upper-top and lower-bottom coordinates of the ROI in the first image (img1.jpg), which contains an object that needs to be matched across images.

If the coordinates are missing, the program should let the user select the ROI with a simple GUI.

The remaining parameters contain all the images that need to be aligned. They will be usually specified as `*.JPG` to select all images in a current directory. Note that you cannot use a "*" shell expansion in the QT Creator "Run — Arguments" field and you need to enter file names. The photo-tour uses the same order of images in the final video as the order specified in the command line.

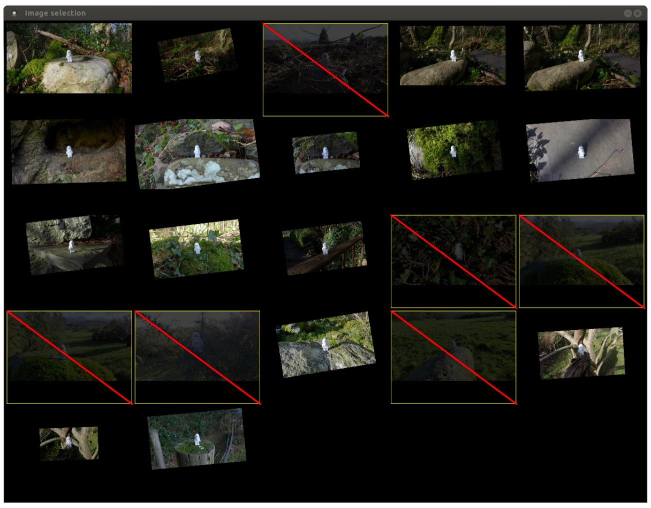

Figure 2: Image thumbnail and selection window. Crossed images are excluded from the result. The darker images could not be matched.

While processing, the program shows the progress by displaying the images that have been already aligned as thumbnails. Figure 2 shows an example of such a window. Because not every image can be aligned, the images in the window can be excluded from the photo tour by clicking on them. The excluded images are marked with yellow border and are crossed. The darker colour shows the images that could not be matched.

The application should offer the following options to output the results:

- When 'A' key is pressed, the program generates a video file `anim.avi` with an animated photo tour, in which the images smoothly transform and blend into each other.
- When 'I' key is pressed, the program generates a series of frames `frame_NN.jpg`, where `NN` is a number starting from 0. Each frame is transformed so that the object of interest is aligned to the position of this object in the first image. Figure 1 shows an example of such frames. The frames can be later merged into an animated GIF or similar with another application.
- When "Escape" is pressed, the program exits and no result is saved.

2 Features and components

Not all functionality must be implemented to get a mark for this assignment. Obviously, the more features are implemented, the higher the mark will be. The following sections explain how to implement each feature and how many points are awarded (up to the total of 100).

2.1 Basic alignment

Points: [25]
Difficulty: moderate / easy

This is a core algorithm, worth most of the mark, so it is important that you implement it first.

Your goal is to find a 2D transformation (3x3 matrix) that would map an object selected in the ROI in one image to the location of that object in another image. This will be achieved with feature point matching algorithm as follows:

1. Detect feature points in the ROI of the first image and everywhere in the second image.
2. Extract a descriptor for each feature point.
3. Find the best matches between feature points in both images.
4. (optionally) Prune the matches with one or both of the suggested methods (Sections 2.5 and 2.6).
5. Use the RANSAC algorithm to find a homography (projective transform) mapping ROI from one image to another. Optionally, you can use your own RANSAC restricted to similarity transformations to improve the results (Section 2.9)
6. Use the number of *inliners* returned by the RANSAC algorithm as the measure of the quality of the match. If there are too few inliners, the match is probably incorrect and needs to be ignored. You can automatically exclude such images from the results.

You will notice that very few combinations of feature detectors and descriptor extractors work well in this task. Experiment at least with the following feature detectors: `SURF`, `FAST`, `SIFT`, `GFTT`, and the following descriptor extractors: `SIFT`, `SURF`. It is essential that you use the function `drawMatches` to visually assess the quality of your matching and debug your code.

In the basic version of this algorithm you need to match the ROI of the first image with all other images. In a more advance version (Section 2.10) you will be matching all relevant pairs.

The functions and classes that you may find useful in this task: `FeatureDetector`, `DescriptorExtractor`, `FlannBasedMatcher`, `findHomography`, `drawMatches`.

2.2 Region of interest selection

Points: [5]
Difficulty: easy

Display an image in a window. Let the user click on two points (or press and drag) to selected a rectangle with the region of interest.

Use this selection to mark the object that needs to be matched across the images. For maximum mark, create a class with an easy-to-use interface.

Do not use such a selection window when debugging. Instead, enter the coordinates as command line arguments. If you use the manual selection for debugging, you will not get repetitive results and will waste time.

2.3 Your own image data set

Points: [5]
Difficulty: easy

Shoot at least 10 images with different background but containing the same object or person. You can use your mobile phone camera. Make sure that the object is photographed from approximately the same angle and does not differ too much in size. Otherwise matching will be very difficult. Resize the images so that the maximum resolution is 1024×768. Rename images `img_NN.jpg`, where `NN` starts at 00.

You will be asked to upload your images to Blackboard so that they can be shared with all other students for testing their algorithms.

2.4 Thumbnail preview and image selection window

Points: [10]
Difficulty: moderate / easy

Display a window with image thumbnails, as shown in Figure 2. The user can click on an image to select or deselect it from the final animation or frame sequence. There should be a margin (padding) between the thumbnails. The window should not exceed the resolution of your screen.

For maximum mark, create a class with at least the following methods:

- `void setImage(int index, Mat img)` - updates the image at the position `index` in the thumbnail preview window;
- `bool getSelection(int index)` - queries whether image `index` is selected;
- `void setSelection(int index, bool selected)` - set the image `index` to be selected or deselected.

The image `img` can be of any size and your class will be responsible for rescaling it to the thumbnail size.

Such a thumbnail window will be very useful for debugging so you are recommended to implement this feature.

2.5 Pruning with the Nearest-Neighbour-Distance-Ratio

Points: [5]
Difficulty: moderate

Implement the Nearest-Neighbour-Distance-Ratio (NNDR) criterion (introduced in the lecture) to eliminate dubious feature point matches.

2.6 Pruning with the Symmetry test

Points: [5]
Difficulty: moderate

Implement the symmetry test (introduced in the lecture) to eliminate dubious feature point matches.

2.7 Generate a set of aligned images

Points: [5]
Difficulty: easy / moderate

When 'I' key is pressed, your application will warp each selected image so that the object of interest is aligned to the first image and save them as `frame_NN.jpg`. Such frames can be later combined into animated GIF. This could be done with `ImageMagick`'s `convert` command line tool:

```
convert -delay 50 frame_?.png anim.gif
```

The functions and classes that you may find useful in this task: `warpPerspective`.

2.8 Generate a photo-tour video file

Points: [10]
Difficulty: moderate

When 'A' key is pressed, your application will produce a photo-tour video file. Each selected image will first alpha-blend to the next image in the sequence, so that the object of interest in both images is aligned. To achieve such an alignment, you need to transform the second image using the transformation found with the RANSAC algorithm (yours or from the OpenCV library). You will need to chain several transformations to get an image into the right position. Then, the second image will smoothly move to its original position. You can easily achieve such smooth movement by alpha-blending between the initial transformation matrix and the identity matrix. The entire animation for each pair of frames should take between 3 and 5 seconds at 30 frames-per-second frame rate. Check also an example animation in Blackboard.

The functions and classes that you may find useful in this task: `VideoWriter`, `warpPerspective`.

2.9 Own RANSAC for similarity transform

Points: [10]
Difficulty: moderate / hard

You will notice that using a full projective transform (homography) does not produce the best results. Some images will be tilted or sheered by rather unrealistic angles. However, since all images are taken from almost the same angle, it is usually sufficient to restrict the transformation (between images) to rotation, translation and scaling. These three operations form a similarity transform. Such restricted transformation has fewer degrees of freedom and requires fewer feature-point matches, making it more robust when not enough matches can be made. As a result, you will be able to match more images and the matches will be better.

OpenCV function `findHomography` does not allow to restrict the sought transformation to similarity so that the full homography matrix is always found. Therefore, for this task you need to implement your own RANSAC algorithm (explained in the lecture), which will allow only similarity transform between feature-points in both images.

The functions and classes that you may find useful in this task: `estimateRigidTransform`, `perspectiveTransform`.

2.10 Feature matching graph

Points: [20]
Difficulty: hard

Not every image can be matched to the ROI in the first image. However, every time you do a successful match, you also detect the object of interest in another image. If you could use that detected object in another image to again match against each problematic image, you could significantly increase the number of images that can be matched.

To implement this feature, you need to consider your task as a graph problem. Each node in the graph is an image. Each edge in the graph is a transformation that maps an object of interest in one image (node) to the object of interest in another image (node). The basic matching (Section 2.1) attempts to find edges from the first image to all other images. However, if one of the edges (transformations) cannot be found, the corresponding image (say j) fails and needs to be eliminated from the results.

However, there is more than one way to reach a problematic node j in a graph. Once you find a transformation from node 1 to node k, you can extract ROI from image k and match it with a problematic node j.

This can also be used to improve matches. The quality of each match (edge) is determined by the number of inliners found by the RANSAC algorithm. Let us say that the edge $E(1, 2)$ (from node 1 to 2) contains 5 inliners. But you also found an edge $E(1, 3)$ with 20 inliners and the edge $E(3, 2)$ with 20 inliners. Clearly, the quality of chained transformations $E(1, 3)$ and $E(3, 2)$ will be better than a single transformation $E(1, 2)$.

To implement this feature, create a graph structure with nodes and edges. Add edges to the graph by matching the relevant nodes (graph traversal). Then implement Dijkstra shortest path algorithm to find the best transformation from node A to node B.

The functions and classes that you may find useful in this task: `std::priority_queue`, `boundingRect`, `convertPointsFromHomogeneous`.

3 Milestones

To ensure that you make a steady progress through the assignment, you will need to submit a partial solution before the following deadlines:

- (28 Feb): Basic feature matching AND (ROI selection OR Thumbnail preview - at least one of the two)
- (21 Mar): Your own data-set AND generate a set of aligned images
- (4 April): At least one of the pruning methods AND generate a photo-tour video

Submit only source files (.cpp and .h) without the rest of the project, in particular without the compiled objects and executables. You can submit the files as separate attachments.

The milestone submission is to ensure that you make a steady progress and put effort into the assignment. You will not be penalised if the required functionality does not work as expected but there is sufficient evidence that you put your effort to implement it. **5% will be subtracted from the final mark for each milestone that was missed.** This means that if your initial mark for the assignment is 60% but you missed two milestones, the final mark will be reduced to 50%.

4 Final submission

Final submission deadline: 2 May 2014 (before the lab session).

Submit to Blackboard a .ZIP archive with a complete QtCreator project (all sources). You need to be able to demonstrate the software and answer questions about the code. "0" mark will be awarded for not showing up for the demonstration session.

5 Help

5.1 Feature matching in OpenCV

OpenCV provides several classes for detecting, describing and matching feature points. They are all the subclasses of `FeatureDetector`, `DescriptorExtractor`, and `DescriptorMatcher`. These classes provide a common interface, but the actual algorithms are implemented in subclasses, such as `SurfDescriptorExtractor`. The advantage of this approach is that you can experiment with different algorithms just by replacing the class you instantiate without the need to change the rest of your code. Experiment with different algorithms to choose the one that provides the best trade-off between robustness and speed.

The documentation of the feature matching function can be found at: `http://opencv.itseez.com/modules/features2d/doc/features2d.html`. You may also want to check the tutorials at `http://opencv.itseez.com/doc/tutorials/features2d/table_of_content_features2d/table_of_content_features2d.html`.

5.2 Nearest Neighbor Distance Ratio

There is no OpenCV function that would match features based on the NNDR, but the distance ratio can be computed efficiently using `DescriptorMatcher::knnMatch` and asking for 2 best (nearest) matches.

5.3 Bidirectional matching

The bidirectional matching can eliminate a large number of incorrectly matched features. The idea is that the NNDR should be above a certain threshold not only for the feature matched from the first image to the second image, but also for the same pair of features matched from the second to the first image. The pseudo-code for the bidirectional matching can be written as follows:

```
m_12 := matches from img1 to img2 for NNDR <= threshold
m_21 := matches from img2 to img1 for NNDR <= threshold
m_sym := empty list of matches; // contains all symmetric matches
for each match m_a in m_12
{
   for each match m_b in m_21
   {
      if( m_a.from_feature == m_b.to_feature and m_b.from_feature == m_a.to_feature )
      {
          add match m_a to m_sym;
      }
   }
}
```

5.4 Homography

The function for finding the transformation between two images based on a set of feature points is `findHomography`. Run it with the `CV_RANSAC` option. More information in the lecture.

5.5 Warping images

The images can be warped using a projective transformation using the function `warpPerspective`.

5.6 Visualizing matches

It is quite important to visualize the results of your matching to spot potential problems and also to fine-tune the parameters. The OpenCV function that is useful for that purpose is:

```
drawMatches( img1, keypoints1, img2, keypoints2, matches, res_img,
   Scalar(255,255,255));
```

But remember to disable such visualisation in your final program.

5.7 Missing function on linking

For this task you will need function from several OpenCV modules. Make sure that you link your program with the following libraries:

```
-lopencv_core -lopencv_highgui -lopencv_imgproc -lopencv_features2d
-lopencv_calib3d -lopencv_flann -lopencv_nonfree -lopencv_video
```

5.8 Video encoding on OSX

The current version of the libary does not seem to work with the native OSX video encoding libraries. To make it work, it needs to be compiled with the `ffmpeg` support. If you encounter this issue, report it on the forum or during the lab session.