Lab report 7 Computer Vision - Matteo De Gobbi

For this laboratory I wrote a class called ImageStitcher, this class defines:
1. A constructor that takes as input:
   - A string containing the path to the directory from which we want to read the images to stitch.
   - The string containing the search format for the file extension to look for in the directory (e.g. “*.bmp”).
   - The angle movement from one photo to the next.

The constructor reads all the images with the specified extension in the directory provided, applies a cylindrical projection and stores the images in a std::vector private variable of the class.
2. A stitch_two_images private member function that takes two input images and stitches them together and stores the result in a third image. There is also an offset parameter that is used when this function is called repeatedly to stitch multiple images together.
To stitch two images together we extract the SIFT features in the two images and match them as in the previous lab using a matcher provided by opencv followed by Lowe’s ratio test to keep only the good matches.
Then we use the findHomography function but in this lab we also use the mask parameter which tells us which of the keypoints matches are considered inliers in the homography. With this mask we can easily compute how much the second image is translated compared to the first one:

```
double translation = 0;
int count = 0;
for (int i = 0; i < mask.size(); ++i) {
  if (mask[i] == 1) {
    translation += good_keypoints1[i].x - good_keypoints2[i].x;
    ++count;
  }
}
translation /= count;
```

Then we can stitch the images by copying the first image in the output image and then copying the second image translated to the right by the amount we just computed. This is done using the operator () that the Mat class overloads and allows us to select an area of the image to modify:

```
int i_tr = (int)translation;
Mat first_half =
    stitched(Range::all(), Range(col_offset, col_offset + im1.cols));
Mat second_half = stitched(
    Range::all(), Range(col_offset + i_tr, col_offset + im1.cols + i_tr));

im1.copyTo(first_half);
im2.copyTo(second_half);
```

3. To stitch all the images together we call the stitch_two_images function repeatedly and by increasing the offset each time we can fill the output image from left to right:

```
Mat ImageStitcher::stitch() {
  int rows = imgs[0].rows;
  int cols = 0;
  for (int i = 0; i < imgs.size(); ++i) {
    cols += imgs[i].cols;
  }
  Mat stitched = Mat(rows, cols, imgs[0].type());
  int col_offset = 0;
  for (int i = 0; i < imgs.size() - 1; ++i) {
    col_offset += stitch_two_images(imgs[i], imgs[i + 1], stitched, col_offset);
  }
  return stitched;
}
```

This offset is increased each time by the amount of pixel the second image is translated with respect to the first one.

In the main function we use argv to obtain the directory name, the angle and (optionally) the file extension of the images.
In the case the file format is not provided the program assumes the images are in png format.

```
std::string glob_path("*.");
if (argc < 3) {
  cout << "Error, provide path to the directory, the angle and format if "
          "it's not png"
       << endl;
  return -1;
} else if (argc < 4) {
  glob_path.append("png");
} else {
  glob_path.append(argv[3]);
}
ImageStitcher stitcher{argv[1], glob_path, strtod(argv[2], nullptr)};
```

Results:

For the dolomites images this is the result:

If we zoom in we can see how there are lines in the sky where the images are stitched:

Also there is a mistake on the right were there is a sharp jump in the snow due to a bad estimation in the translation between the previous and current image.

The kitchen image is very well stitched and the only imperfections we can see are the stitching lines.

Also the lab image is stitched correctly: