CMP SCI 6420 **Image Processing and Computer Vision**
Assignment # 2 Due Date: February 18, 2021

Image Registration

Purpose

The purpose of this assignment is to apply and assess the effectiveness of both manual and automatic approaches to image registration.

Task

Image registration is the process of aligning image data in different images to a single coordinate system. The images may have been sensed using different sensors or at different times and the goal of image registration is to make sure that the objects in those images *overlap* each other, both in terms of position as well as scale. It is an essential process to enable comparison between different images of the same scene or objects.

You are given a set of two images. The first image is the target image while the second image is the template image. Your job is to resample the target image so that it is in the same coordinate system as that of the template image using image registration techniques.

Image registration can be performed manually or automatically. The manual registration requires a human to identify a set of points in the reference image as well as the corresponding points in the target image. Your program will use these points to compute an affine or projective homography between the two images. Next, you should use the computed homography to warp the target image to the coordinate space of the reference image. Lastly, display the images side-by-side to assess the quality of registration. Also use image subtraction to show the amount of translation and scaling in the resulting image compared to the input image.

Invoking the solution

Your solution will be invoked using the following command:

```
register [-h] [-M] [-e epsilon] [-m motion_type] [-o output_warp] [-w warp_img] \
image_file template_file [warp_file]
```

`register`	Name of your executable
`M`	Perform manual registration
`e epsilon`	ECC's convergence epsilon [default: 0.0001]
`m motion_type`	Type of motion (`translation`/`euclidean`/`affine`/`homography` [default: `affine`]
`o output_warp`	Output warp matrix filename [default: `out_warp.ecc`]
`w warp_img_file`	Warped image [default: `warped_ecc.jpg`]
`image_file`	Input image (to be warped/aligned)
`template_file`	Template image for alignment
`warp_file`	Input file containing warp matrix

The parameters enclosed in `[]` are optional. You are free to use long parameter names such as `--epsilon` for `-e`.

Suggested implementation steps

1. Parse the command line. You can create your own parser or use the class `CommandLineParser` provided by OpenCV. Each of the optional arguments may have a default value. If the user specifies the option `-h`, print a help message and exit. Otherwise, assign the suggested parameters from user inputs or default values.

2. Read the input image as color. Copy it into another image as grayscale and apply histogram equalization to enhance the edges. Read the template image as grayscale and apply histogram equalization to enhance edges.
3. If the user specifies manual registration, allow the user to specify corresponding points on image and template and use those to build the affine warp matrix.
4. Convert warp type from string to built-in enumerated type. For example, `affine` should be converted to `cv::MOTION_AFFINE`.
5. Create warp matrix as an identity matrix of size 2×3 [3×3 for motion type homography].
6. If a warp file is specified, read it into the warp matrix.
7. Use the function `cv::findTransformECC` to find the warp matrix (geometric transform) between the image and template in terms of the ECC criterion.
8. Use the warp matrix to compute the final warped image. For homography motion, use the function `cv::warpPerspective`, for other motion types, use `cv::warpAffine`. Use the original color image to perform transformation.
9. Display the transformed image.
10. Compute an error image by subtracting the grayscale value of transformed image from the template image. Find the maximum error ϵ by using `cv::minMaxLoc`. Scale the error image by taking its absolute value (use `cv::abs`) and multiplying by $255/\epsilon$. Display the error image as well.

Criteria for Success

You should be able to implement image registration using at least `cv::MOTION_AFFINE`. Transform the images using both manual and automatic computation of warp matrix. The manual computation will depend on specification of corresponding points using mouse clicks (left button click).

Grading

I'll use the following rubric to assess your submission.

1. *Overall submission; 30pts* Program compiles and upon reading, seems to be able to solve the assigned problem.
2. *Command line parsing; 10pts* Program is able to parse the command line appropriately, assigning defaults as needed; issues help if needed.
3. *Manual registration; 25pts* Program is able to compute manual registration appropriately.
4. *Automatic registration; 35pts* Program accurately computes the warp matrix using `cv::findTransformECC` and applies the warp to get the transformed image.

Submission

Submit an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.1 where *username* is your login name on delmar. This directory should be located in your `$HOME`. Once you are done with everything, *remove the executables and object files*, and issue the following commands:

```
% cd
% chmod 755 ~
% ~bhatias/bin/handin cs6420 1
% chmod 700 ~
```

Do not copy-paste these commands from the PDF; type in those commands.