CMPE 264 – Project Assignment 2

You are to program an implementation of plane-sweeping 2-view stereo. We covered the theory in class. Here are all of your project components:

Part 1: Camera calibration (intrinsic parameters)

You can use any camera for this project. If you are using a camera with a zoom, you will need to lock it at a certain zoom level. (I advise against using a zoom lens for this project.) Your first step is to calibrate your camera, that is, to find the intrinsic parameters matrix (K) and the radial distortion coefficients. You can follow the tutorial on calibration[1] on OpenCV (see also here[2]), or `cameraCalibrator` in Matlab. The OpenCV functions used are:

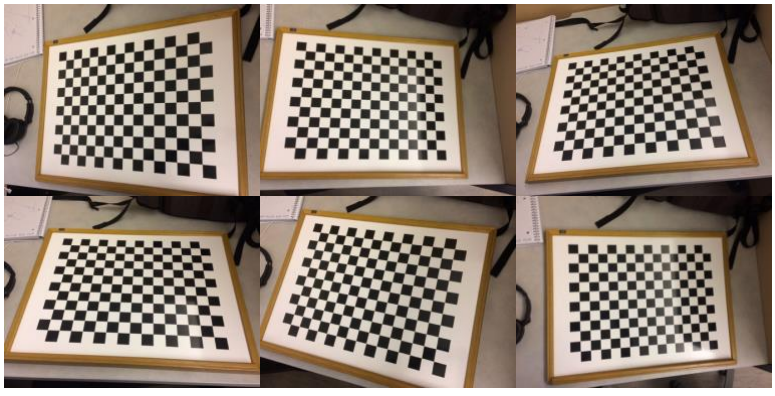
```
cv2.findChessboardCorners   # finds chessboard corners
cv2.cornerSubPix            # refines corner position
cv2.calibrateCamera         # camera calibration
```

You will need to print a chessboard pattern[3], glue it to a flat surface, and take multiple (at least 10) pictures of it from different viewing angles (important). Then process the images (follow the tutorial) to obtain your calibration parameters. It is important that you check the quality of calibration by computing the mean square error (again, see the tutorial). The mean square reprojection error should definitely be less than 1 pixel, and possibly less than 0.5 pixels. If it isn't, take more pictures and try again.

Deliverables:

1. The pictures you took of the chessboard pattern
2. The intrinsic matrix K
3. The radial distortion coefficients
4. The reprojection mean square error

Here are sample calibration images:



[1] http://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

[2] http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[3] http://docs.opencv.org/2.4/_downloads/pattern.png

Part 2: Take the pictures

You will need to take two pictures of a scene with objects at different distances. The pictures need to be taken from different viewpoints, and with different camera orientations. You don't want to move the camera too much – just enough that there is appreciable parallax. Make sure that:

- There is a substantial part of the scene visible in both images.
- There is enough "texture" in the images, to enable stereo matching.
- There are objects at different distance from the camera.
- The maximum distance of objects visible in the image is finite (take the picture indoors).

Here is a sample image pair:

Deliverables:

1. The images you took

Part 3: Compute the relative camera pose $R_L^R, \boldsymbol{r}^R$

You will use pre-packaged OpenCV (or Matlab) functions for this. Tutorials for OpenCV are available here and here.

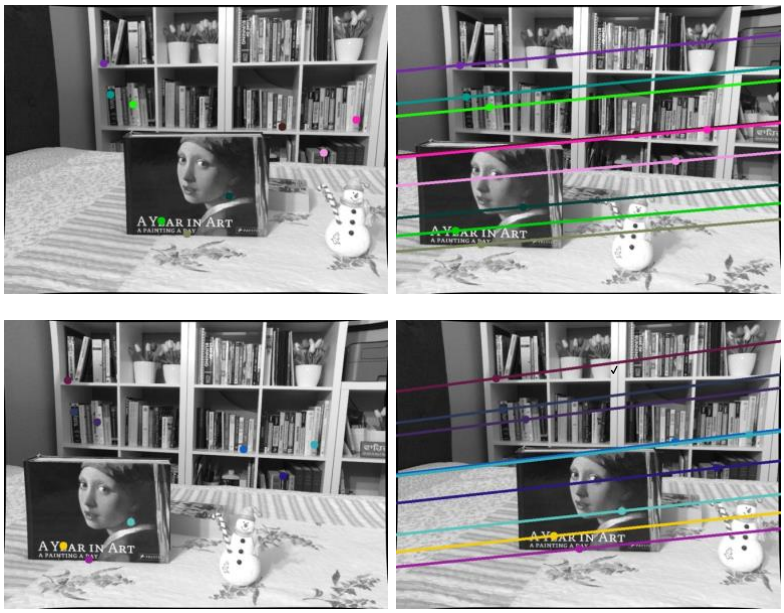
Here are the different components, with relevant OpenCV functions:

- Undistort the images using the parameters you found during calibration
  `undistortPoints()`
- Create "feature points" in each image:
  ```
  sift = cv2.xfeatures2d.SIFT_create()
  sift.detectAndCompute()
  ```
- Match the feature points across the two images:
  ```
  cv2.FlannBasedMatcher()
  flann.knnMatch()
  ```

Here are the feature points matches for the images above:

- Compute the fundamental matrix F and find some epipolar lines (note: although this is not strictly necessary for our purpose, you still are expected to do it):
  `findFundamentalMat()`
  `computeCorrespondEpilines()`

  Here is a sample result with a few epipolar lines for the first and the second image:



- Compute the essential matrix E:
  `findEssentialMat()`

- Decompose the essential matrix into $R_L^R, \boldsymbol{r}^R$
  `decomposeEssentialMat()`

- Calculate the depth of the matching points:
  `triangulatePoints()`

  At this point, you have a set of 3-D points, obtained by triangulating the feature

points that you matched across images.

- Show the re-projected points. This means that you re-project the 3-D points you found onto the first image. If depth was computed correctly, they should re-project exactly on the same position as the original features. Due to errors, however, you may find that some points re-project on slightly different locations. Here is an example (green: feature points; blue: re-projected points):

Deliverables:

1. Show a few epipolar lines on the images.
2. Write down the matrix $R_L^R, \boldsymbol{r}^R$ you found.
3. Show the re-projected feature points on the first image

Part 4: Plane-sweeping stereo

Select a set of planes $\{\boldsymbol{n}^L, d_i\}$ defined in the reference frame of the first camera, such that the planes are all orthogonal to the first camera's optical axis (use $\boldsymbol{n}^L = (0,0,-1)$). Choose a set of **N=20 equispaced distances** $\{d_i\}$ spanning the interval between the minimum (d_{min}) and the maximum (d_{max}) depths that you found when you triangulated the feature points. (Remember that depth is defined in terms of baseline units.)

For each plane $(\boldsymbol{n}^L, d_i)$:

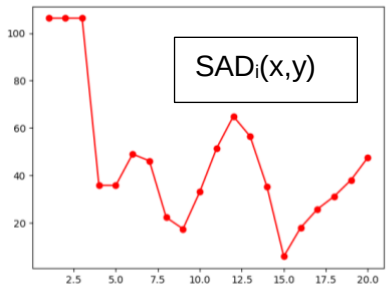
- Find the homography that warps the second image in the pair such that any point belonging to that plane projects onto the same pixel in the two images (the first image and the warped second image).
- Warp the second image using this homography. You can use the function `warpPerspective` in OpenCV or `imwarp` in Matlab (note: Matlab translates the image so that it fits within the window; you may need to use `imtranslate` to make sure that the warped image is correctly registered with the first image).

  Here are images showing the warped second image overimposed on the first image for three different planes:

See here for an animation with all 20 planes.

- Compute the pixel-by-pixel absolute difference between the warped second image and the first image.
- Run a block filter (`blockfilter()` in OpenCV) of suitable size (e.g. 15x15 pixels)

At this point you will have N=20 images that represent the block-filtered differences {$SAD_i(x,y)$}. For each pixel (x,y), you should find the index *i* (corresponding to a plane with distance d_i) that gives the lowest value of $SAD_i(x,y)$. Here is plot of $SAD_i(x,y)$ as a function of *i* for a generic pixel (x,y) (marked in the image on the left):



In this case, the depth assigned pixel (x,y) should be 15 (in units of baseline length).

After you assign a depth for each pixel, you can display the computed depth image:

Deliverables:

1. The values d_{min}, d_{max} you are considering for depth (in units of baseline length)
2. The N=20 warped second images, one warped image per plane
3. The resulting depth image, shown in greyscale. Make sure that white pixels correspond to d_{max}, while black pixels correspond to d_{min}.

What I am expecting from you

You will need to submit:

- A report that explains carefully all that you did and that contains all deliverables
- Your code – all project zipped. Harsimran needs to be able to compile and run the code. Include:
    - A README file describing:
        - OS, language, libraries used
        - Detailed instructions on how to run your code
    - Scripts that generate all of the graphs and images you are showing in your report.
    - If you think it will help us better understand your code or how to run it, you could also include relevant screenshots.
    - **Please use relative path names.**

You will be evaluated on:

- Correctness of your implementation of the project and results
- Quality of your report (including quality of your writing, images/plots included, and clarity)
- If your code does not compile/run as expected, you will be assessed a penalty