CMPE 264 – Project Assignment 2

You are to program an implementation of plane-sweeping multiview 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. 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

Part 2: Take the pictures

You will need to take three 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 all three images. For example, here are three images I took.



Rather than choosing such a messy scene like a did, I suggest a scene with one or a few objects in the foreground, and a distant background.

Deliverables:

[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

1. The three images you took

Part 3: Compute the essential matrix of all three camera pairs

For each image pair (1-2, 2-3, 1-3) you need to compute the corresponding essential matrix E. This requires selecting candidate matching points across the two images. The first thing you should do, however, is to "undistort" the images, in order to remove radial distortion. In OpenCV, you should use `getOptimalNewCameraMatrix` and undistort. In Matlab, use `undistortImage`.

To select candidate matching point pairs, you can proceed in one of two possible ways:

1. Select the points manually, by clicking on the images with the mouse. Here[4] is a primer of how to do it in Python and OpenCV. In C++ it will be similar, and in Matlab it is a piece of cake (you can use the function `cpselect`). At the end, you will have a list of point pairs (pairs of (x,y) coordinates). If you do this carefully, you can expect no false matches, although the point localization may not be very accurate. Make sure to select at least 15-20 matching point pairs.

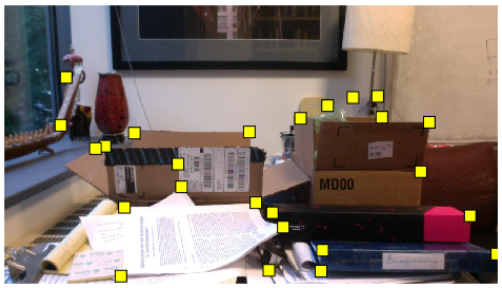
2. Select interesting points and candidate point matches automatically with a feature detector. For example, on OpenCV you could extract Surf features in each image with `SurfFeatureDetector`, obtain a descriptor for each feature using `SurfDescriptorExtractor`, then search for point pairs (one point per image) with similar descriptors using `FlannBasedMatcher`. At the end, you will have a list of possibly several hundred matching point pairs, many of which, however, will be incorrect (false matches or outliers).

Here are the matching points I selected by hand:

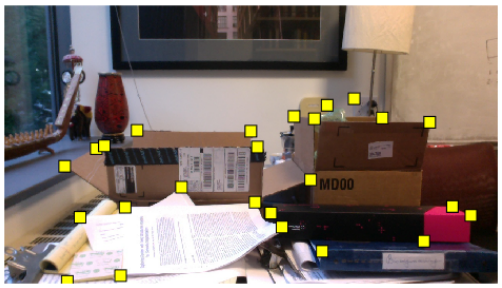
Images 1-2:

[4] http://www.pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/

Images 1-3:

Images 2-3:

From your candidate point matches you can estimate the essential matrix E using `findEssentialMat`. You could also first compute the fundamental matrix F using `findFundamentalMat`, then obtain the essential matrix E using the camera matrix K ($E = K^T F K$). I found that `findEssentialMat` gives a better result, as it uses an algorithm that forces the essential matrix to have equal positive eigenvalues. Note that these algorithms use random sampling, and thus the results may be different each time you run the code.

Note that (depending on the function and the parameters you use for computing E), the algorithm will return the set of "inliers". These are the matching point pairs that satisfy the epipolar constraint defined by the matrix E within the specified accuracy.

Given the essential matrix E, you can compute the epipolar lines for each image in the pair.

Deliverables:

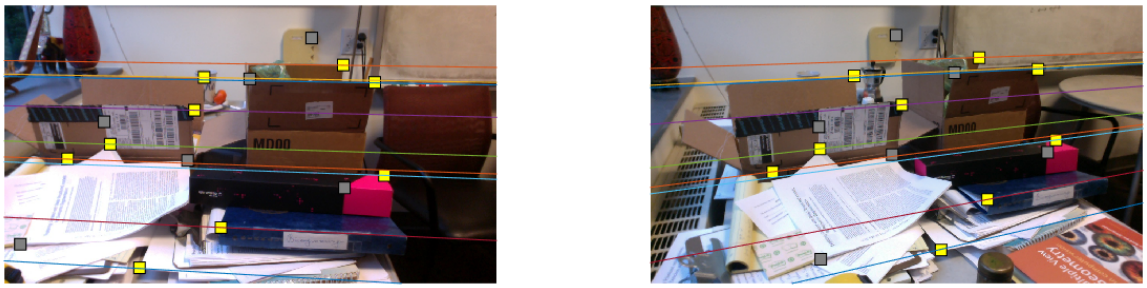
1. The matching points you selected for each image pair (shown on the images). Show the "inlier" and "outlier" points with different colors.

2. For each image pair, show the conjugate epipolar lines for the inliers (also show the inliers in the image)

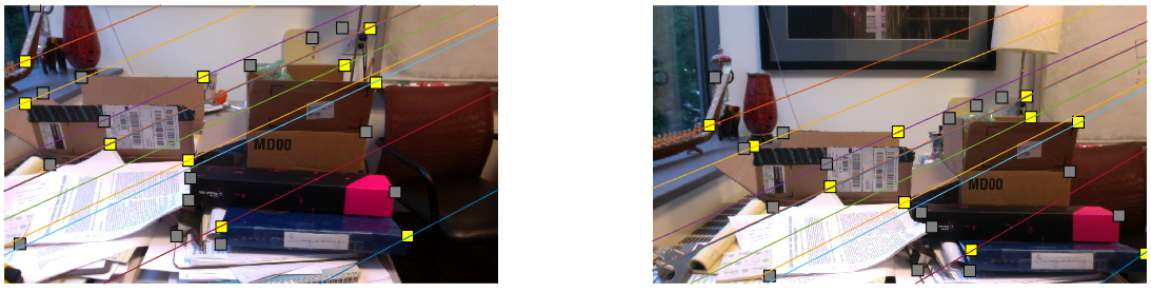
3. The essential matrices you found for the three images

Here are the conjugate line pairs I found. The outliers are shown as grey squares.

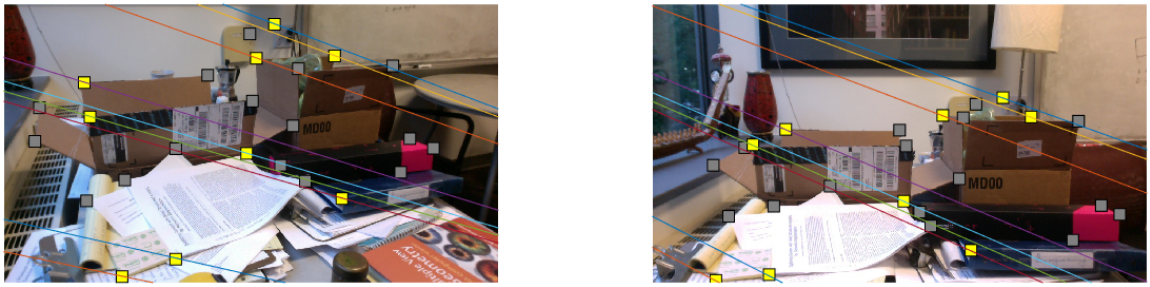
Images 1-2:



Images 1-3:



Images 2-3:



Part 4: Find the extrinsic parameters

Note: the description below applies to each image pair. You are supposed to perform these computations on each one of three image pairs.

Given the essential matrix E, you will need to find the rotation matrix R_L^R and the translation vector $\mathbf{r}^R$. Remember that there are two rotation matrices possible, and that the translation vector is defined up to a scale factor and up to a sign. You will need to find a pair (R_L^R,$\mathbf{r}^R$) that is consistent, meaning that the depths measured of all inliers need to be positive.

Note: sometimes, due to numerical errors, the essential matrix you computed may not result in a proper rotation matrix ($R^TR=I$, $\det(R)=1$). You should make sure that both

matrices U and V in the SVD of E ($E = U \Sigma V^T$) have determinant equal to 1. If one of the two has determinant equal to -1, then you should use -E instead of E in the SVD decomposition. (You can easily see that this will make both determinants of U and V positive, resulting in a consistent rotation matrix.) If both determinants of U and V are negative, you may need to re-compute E.

Once you have a pair $(R_L^R, \mathbf{r}^R)$, you should compute the 3-D locations of the inlier with respect to the first image in the pair, and project these 3D points onto the second image. You should then compute the distances between these re-projected points and the actual points inlier points in the second image in the pair. If these distances are too large, you may want to re-compute E and start again.

Deliverables: For each image pair:

1. Write down the matrix R_L^R and translation vector $\mathbf{r}^R$ you found.

2. Write the list of depths computed for all inliers (with respect to the first camera in the pair)

3. Show, in the second image of the pair, the original location of the inliers, and the location of the re-projected points (use different colors for these two sets of points). If you did things correctly, they should overlap nicely.

4. Compute the root mean squared reprojection error.

Part 5: Rescale the translation vector

Let $\mathbf{r}_{12}^2$, $\mathbf{r}_{13}^3$, $\mathbf{r}_{23}^3$ be the three translation vectors you found, and R_1^2, R_1^3, R_2^3 the rotation matrices, for the pairs image1-image2, image1-image3, and image2-image3. Note the change in notation: let camera i be the "left" camera and j be the "right" camera. Then, $\mathbf{r}^R$ is now renamed as $\mathbf{r}_{ji}^j$ (left camera seen by the right camera, expressed in reference of the right camera) while $\mathbf{r}^L$ becomes $\mathbf{r}_{ij}^i$. Also note that $\mathbf{r}_{ij}^i$, which is the vector from camera i to camera j expressed in terms of the reference frame of camera i, is equal to $-\mathbf{r}_{ji}^i$. Finally, remember that $\mathbf{r}_{ij}^i = R_k^i \mathbf{r}_{ij}^k = (R_i^k)^T \mathbf{r}_{ij}^k$ (keep in mind that these are vectors, not fixed points in space – no translation term.)

Make sure that $\mathbf{r}_{12}^2$, $\mathbf{r}_{13}^3$, $\mathbf{r}_{23}^3$ all have unit norm. Remember that we only know the direction, not the length of these translation vectors. Our next step is to find a common scale factor for these vectors.

The idea is the following. Consider the following vectors, expressed without reference to a particular reference system: $\vec{r}_{12}, \vec{r}_{12}, \vec{r}_{31}$, where vector $\vec{r}_{ij}$ joins cameras i and j. It should be clear that $\vec{r}_{12} + \vec{r}_{23} + \vec{r}_{31} = 0$, as these three vectors form the sides of a triangle. Rewrite this as $\vec{r}_{12} \cdot \|\vec{r}_{12}\|/\|\vec{r}_{12}\| + \vec{r}_{23} \cdot \|\vec{r}_{23}\|/\|\vec{r}_{23}\| + \vec{r}_{31} \cdot \|\vec{r}_{31}\|/\|\vec{r}_{31}\| = 0$. Divide by $\|\vec{r}_{12}\|$ to obtain: $\vec{r}_{12}/\|\vec{r}_{12}\| + \vec{r}_{23}/\|\vec{r}_{23}\| \cdot \|\vec{r}_{23}\|/\|\vec{r}_{12}\| + \vec{r}_{31}/\|\vec{r}_{31}\| \cdot \|\vec{r}_{31}\|/\|\vec{r}_{12}\|$ $= \vec{r}'_{12} + \beta \cdot \vec{r}'_{23} + \gamma \cdot \vec{r}'_{31} = 0$ where $\vec{r}'_{ij} = \vec{r}_{ij}/\|\vec{r}_{ij}\|$ is the unit-norm version of $\vec{r}_{ij}$, and

$\beta = \|\vec{r}_{23}\|/\|\vec{r}_{12}\|$ and $\gamma = \|\vec{r}_{31}\|/\|\vec{r}_{12}\|$ are the lengths of $\vec{r}_{23}$ and $\vec{r}_{31}$ expressed in terms of the length of $\vec{r}_{12}$.

Our goal is to find β and γ. The first thing we should do is to express the unit-norm vectors $\vec{r}'_{12}, \vec{r}'_{23}, \vec{r}'_{31}$ in terms of a common reference frame. For example, let's choose the reference frame of camera 1. (You could choose any camera as a reference.) Then: $\vec{r}'_{12} \equiv \mathbf{r}_{12}^{1}$, $\vec{r}'_{23} \equiv \mathbf{r}_{23}^{1}$, $\vec{r}'_{31} \equiv \mathbf{r}_{31}^{1}$. (Make sure to correctly convert your data, $\mathbf{r}_{12}^{2}$, $\mathbf{r}_{13}^{3}$, $\mathbf{r}_{23}^{3}$, to these quantities.)

In an ideal world, we would have $\mathbf{r}_{12}^{1} + \beta \cdot \mathbf{r}_{23}^{1} + \gamma \cdot \mathbf{r}_{31}^{1} = 0$. In practice, due to error/noise, this may not be possible for any values of β and γ. Thus, we will try to minimize the squared norm of the residual $\|\mathbf{r}_{12}^{1} + \beta \cdot \mathbf{r}_{23}^{1} + \gamma \cdot \mathbf{r}_{31}^{1}\|^2$. You should be able to do this yourself (hint: write $\|\mathbf{r}_{12}^{1} + \beta \cdot \mathbf{r}_{23}^{1} + \gamma \cdot \mathbf{r}_{31}^{1}\|^2$ as $(\mathbf{r}_{12}^{1} + \beta \cdot \mathbf{r}_{23}^{1} + \gamma \cdot \mathbf{r}_{31}^{1})^T \cdot (\mathbf{r}_{12}^{1} + \beta \cdot \mathbf{r}_{23}^{1} + \gamma \cdot \mathbf{r}_{31}^{1})$, which will allow you to express this as a quadratic function of β and γ. Setting the derivatives of this quadratic form with respect to β and γ to 0 will give you two linear constraints on β and γ.)

At this point, you should rescale $\mathbf{r}_{13}^{3}$ to $\gamma \cdot \mathbf{r}_{13}^{3}$ so that it approximates the actual baseline between camera 1 and 3, expressed in units of $\mathbf{r}_{12}^{1}$. Likewise, you should rescale $\mathbf{r}_{23}^{3}$ to $\beta \cdot \mathbf{r}_{23}^{3}$

Deliverables:

1. Write down the values for β and γ that you found.

2. Write down the values of $\|\mathbf{r}_{12}^{1} + \mathbf{r}_{23}^{1} + \mathbf{r}_{31}^{1}\|$ and $\|\mathbf{r}_{12}^{1} + \beta \cdot \mathbf{r}_{23}^{1} + \gamma \cdot \mathbf{r}_{31}^{1}\|$ you found (the second one should be smaller than the first one if you did things right).

Part 6: Plane-sweeping 2-views stereo

Now consider the pairs image1-image2 and image1-image3. The following applies to each one of these two pairs.

Select a set of planes $\{(\mathbf{n}^1, d_i)\}$ defined in terms of the first camera, such that the planes are all orthogonal to the first camera's optical axis. Make sure that $\mathbf{n}^1 = (0,0,-1)$. The set of distances $\{d_i\}$ of the planes should span some "useful" interval. I suggest choosing $d_i = f/i$ where f is the camera's focal length. Remember that these distances are measured in units of baseline $\mathbf{r}_{12}^{1}$. You could make the index i span from 1 ($d_i = f$) to a number less than equal to f ($d_i = 1$).

Now that you have the set of sweeping planes, for each such plane $(\mathbf{n}^1, d_i)$ you should find the homography that warps the second image in the pair such that points belonging to that plane project onto the same pixel in the two images. You can then proceed as described in the "Plane-sweeping stereo - non-rectified cameras" slides. To warp the second image, 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)

At the end, you will obtain a "distance map" referred to the first camera, in units of $\|\mathbf{r}_{12}^{1}\|$.

Deliverables:

1. The resulting distance maps for the two pairs image1-image2 and image1-image3, referred to the first image in the pair. Make sure to use $\gamma \cdot \mathbf{r}_{13}^3$ for the baseline of image1-image3, so that the distance maps have the same unit.

These are the results I obtained. Since my scene is pretty hollow (the max distance of visible objects was probably no more than 5 times the baseline), I "saturated" the values of depth to 5 for better visualization. Note that there are many mistakes in textureless areas.

Image1-image2:

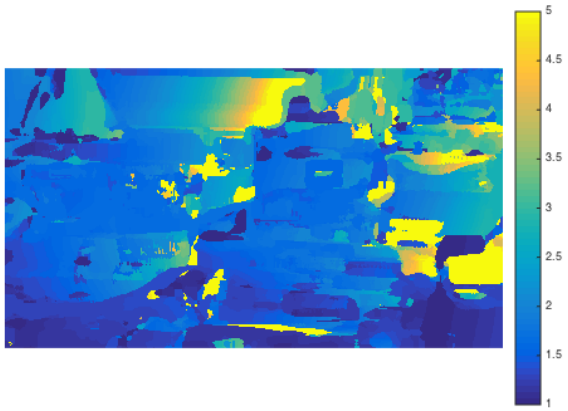
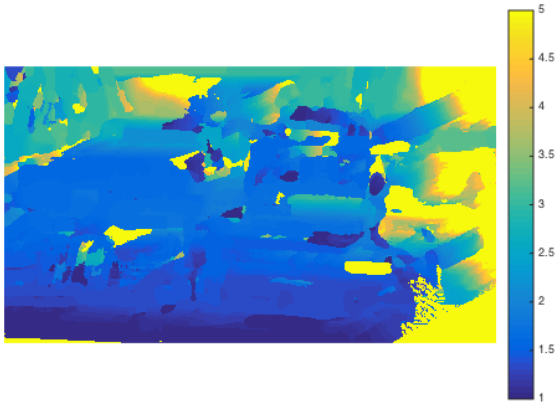


Image1-image3:



Part 7: Plane-sweeping 3-view stereo

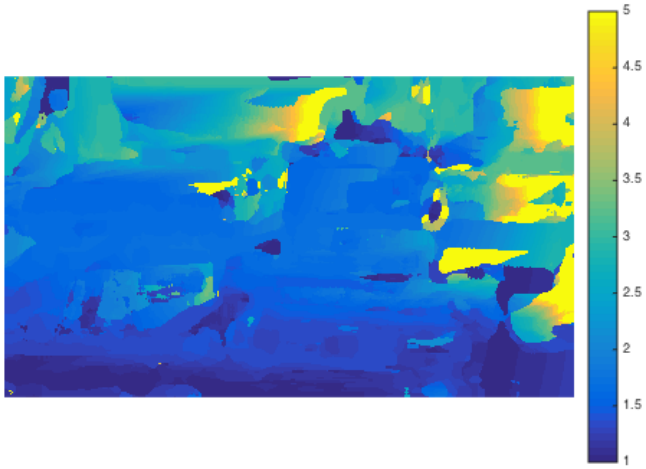
At this point, you have all the ingredients to compute a distance map using a full 3-view plane-sweeping stereo system. Referring to the slide "Plane-sweeping stereo - multiple cameras", you can re-use the arrays $SAD_i^{1-2}(x,y)$ and $SAD_i^{1-3}(x,y)$ from Part 6 too compute $SAD_i^{1-2-3}(x,y) = SAD_i^{1-2}(x,y) + SAD_i^{1-3}(x,y)$, from which you can compute the distance map.

Deliverables:

1. The resulting distance map.

This is what I obtained:

Image1-image2-image3:



What I am expecting from you

You will need to submit:

- A report that explains carefully all that you did. The more detail the better. Make sure to produce all deliverables indicated.

- Your code – all project zipped. Siyang 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 the sata/images that you are showing in your report. You and Siyang must be able to reproduce those

    - If you think it will help us better understand your code or how to run it, you could also include relevant screenshots

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