Homework 03

due May 6, 2020

Please submit your assignments through BlackBoard. To get you ready for your final project report, you need to use LaTeX to generate the PDF. Also, include all your code and a README that describes how your code can be run. We prefer running one script that generates all your results. Explicitly add any dependencies you may need to your submission.

1 Camera Calibration Toolbox [100 pts]

A popular tool to calibrate cameras is the MATLAB toolbox by Jean-Yves Bouget. It has a wide range of calibration options, including both single camera and stereo calibration. Given a set of images, the toolbox estimates calibration parameters (intrinsic and extrinsic) with minimal user interaction. In order to use this toolbox with the Kinect, there are three tasks that need to be done, **(1)** calibrating the RGB camera, **(2)** calibrating the IR (depth) camera, and **(3)** stereo calibrating both cameras together. To calibrate each individual camera, you are provided with a set of images of a calibration pattern in RGB and IR to feed to the calibration toolbox (refer to the images in the *Color* and *IR* directories). A step-by-step guide on how to use the toolbox for this task is available here. Once both sensors have been individually calibrated, the toolbox has an option for stereo calibrating them together following the instructions provided here.

1.1 RGB and IR Camera Calibration [20 pts]

The camera calibration toolbox works by looking at different images of the same planar calibration pattern. Figure 1 shows an example of corresponding images of the calibration pattern taken by the RGB and IR cameras. The toolbox provides a semi-automatic method to extract the grid corners. It requires the user to click on the 4 corners of the grid for every image, as well as information about the number of squares and their physical size. As pointed out in the online example, it is important that you follow the same clicking order on all the images of both cameras, *i.e.* you should **always** start by clicking at the same top-left corner and clicking the rest of the corners in either a clockwise or counter-clockwise manner. This is particularly important when doing the stereo calibration later. The checkerboard pattern is 9×7 squares in size (in the x and y directions respectively), and the physical edge size of each square is 27mm. Once all this information is input, the toolbox performs both a linear and non-linear calibration. The final calibration result is that of the nonlinear calibration method.

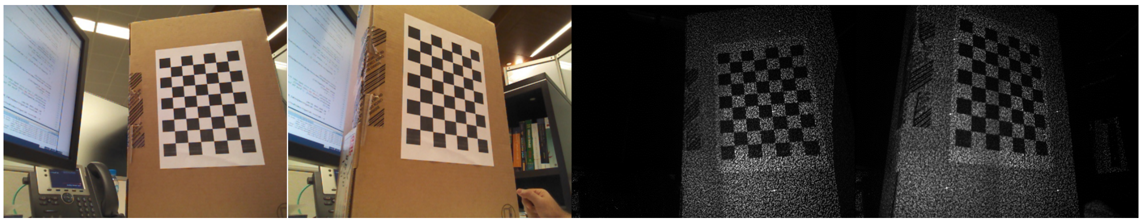

Figure 1: Calibration pattern imaged from two different views using the RGB and IR camera of the Kinect.

For this problem, you are provided with a set of 11 images for each camera (refer to the images in the *Color* and *IR* directories). You have to use the toolbox to calibrate each one separately.

Make sure the same clicking convention is used for each set of images. Save the output calibration files created by the toolbox in order to use them for the stereo calibration problem in Section 1.2. Report the following results:

(i) The intrinsic calibration matrix of the IR camera. Denote this matrix as $\mathcal{K}_{IR}$.

(ii) The intrinsic calibration matrix of the RGB camera. Denote this matrix as $\mathcal{K}_{RGB}$.

(iii) The average re-projection error in pixels for both calibrations using the toolbox function (smaller is better).

(iv) The *.mat* calibration result files for both cameras as output by the toolbox.

1.2 Stereo Camera Calibration [20 pts]

After you estimate the calibration parameters of each camera, you need to run the stereo calibration GUI. In this case, you simply provide the GUI with the calibration files created by the toolbox in Section 1.1. Erroneous results will occur if your clicking convention was different when you tagged the images from the two cameras, or if you started clicking on different top-left positions for every image.

To distinguish the two cameras, the toolbox uses the term *left* to represent quantities related to the *left* camera and *right* for the other one. The left camera determines the world coordinate system. In this problem, we take the IR camera of the Kinect to be the left camera. During stereo calibration, the toolbox performs another optimization step to refine the intrinsic parameters of both cameras. This final step will provide you with a new set of intrinsic parameters, all of which will be saved along with the stereo parameters ($\mathbf{R}$ and $\mathbf{T}$) in the final stereo calibration file. Remember that $\mathbf{R}$ and $\mathbf{T}$ determine how the camera coordinate system of the right (or RGB) camera can be transformed into the world coordinate system. Report the following results:

(i) The new intrinsic calibration matrices for both cameras, denoted as $\hat{\mathcal{K}}_{IR}$ and $\hat{\mathcal{K}}_{RGB}$.

(ii) The stereo calibration parameters (R and T).

(iii) The *.mat* file containing the stereo calibration results from the toolbox.

1.3 Colored 3D Point Cloud [60 pts]

Given the calibration parameters of the RGB and IR cameras of the Kinect, and given a corresponding pair of color and depth images ($\mathbf{I}_C$ and $\mathbf{I}_D$), you can compute a 3D point cloud representing the scene, with color information superimposed. Figure 2 shows an example of an RGB-D pair with its computed colored point cloud.

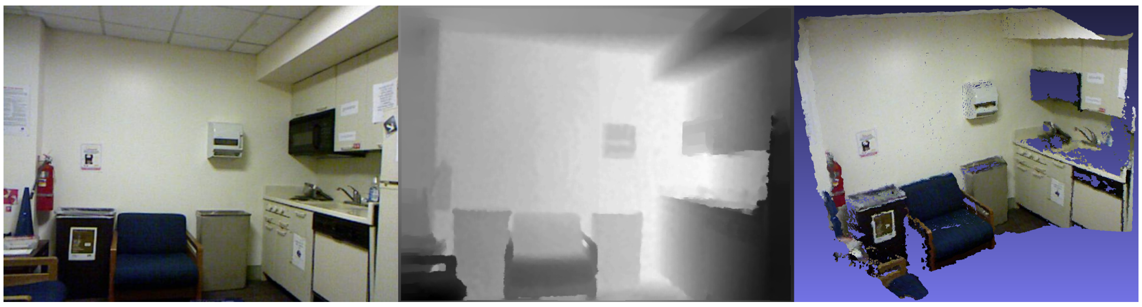

Figure 2: From left to right we can see the RGB image ($\mathbf{I}_C$) with its corresponding depth image ($\mathbf{I}_D$), as well as the 3D point cloud with color superimposed. The 3D point cloud can viewed using an open source 3D viewer called Meshlab.

Given the calibration parameters estimated in the previous problem, you are required to compute a colored point cloud for the RGB-D image pair provided in the homework data *rgbd_pair.mat*. To do this, you need to find a mapping between every pixel in the depth image to one pixel in the RGB image.

(i) Formulate a method to compute the 3D coordinates of points in the world coordinate system (or IR camera coordinate system) that project unto pixels in the IR image, *i.e.* find a function $\mathbf{X} = \mathbf{f}(\mathbf{x}_{IR})$ with $\mathbf{x}_{IR}$ being a pixel in the IR image. Note that the z value of the 3D point $\mathbf{X}$ that projects unto an IR pixel $\mathbf{x}_{IR}$ is the depth value at $\mathbf{x}_{IR}$, or simply $\mathbf{I}_D(\mathbf{x}_{IR})$. In fact, this is what makes the Kinect sensor so popular, as it allows you to measure the z component directly. Describe and analyze your method. Generate the 3D points corresponding to all the pixels in the IR image $\mathbf{I}_D$.

(ii) Once the 3D points have been computed, write MATLAB code to backproject them unto the RGB image plane. In other words, given a 3D point $\mathbf{X}$, compute the pixel coordinates of its projection $\mathbf{x}_{RGB}$ using the camera calibration matrix of the RGB image. You will need to round to the nearest pixel. Remember that your 3D points were computed in the world coordinate system and so $\mathbf{R}$ and $\mathbf{T}$ must be incorporated in the projection.

(iii) In the above two steps, you have found a set of 3D points describing a scene and have projected them unto the RGB image. In this way, the color of a 3D point $\mathbf{X}$ can be determined as the RGB color value of its 2D projection $\mathbf{x}_{RGB}$. Write MATLAB code to compute a 3D colored point cloud using the results of the first two parts.

In order to show your results, create on OBJ point cloud file to visualize in Meshlab (or any other 3D viewer of your choice). An OBJ point cloud file contains rows of ASCII data of the format *v x y z r g b*. Here, each 3D point is defined as a vertex, where v is the counter for that 3D point, *(x y z)* are the 3D coordinates of the point, and *(r g b)* its corresponding RGB colors normalized to lie in $[0, 1]$ (*i.e.* divide by the maximum value 255). The final OBJ file must contain a number of rows equal to the number of 3D points you computed in the previous step. You can import this file into Meshlab and visualize the point cloud as done in Figure 2. Provide the full code as well as several snapshots of your point cloud at different views using Meshlab.

(iv) Assume a virtual RGB camera exists with the same intrinsic parameters $\hat{\mathcal{K}}_{RGB}$ as the Kinect RGB camera, but different extrinsic parameters. The world coordinate system is still coincident with the camera coordinate system of the IR camera. Write a MATLAB function that takes as input new extrinsic parameters and that generates an RGB image from the estimated 3D colored point cloud. In other words, what kind of image of the 3D scene would the virtual camera *see*? Apply the function using the extrinsic parameters provided in (*rgbd_pair.mat*), namely *R_virtual* and *T_virtual.* Show the resulting virtual camera RGB image. Unseen pixels should be set to black.