Lab #9

3D Reconstruction (17)

In this lab, we will take two images of a scene, perform rectification using detected SURF features, compute a disparity map, and reconstruct a 3D point cloud.

1. Take two images of a scene from slightly different positions while trying to maintain as close to the same orientation as possible for the camera (i.e., try not to rotate the camera). The scene should have as much texture as possible (disparity will not work well for things like blank walls).

2. Load the camera parameters from the calibration procedure in lab 8 using `load`. **(1)**

```
load('cameraParametersFile.mat')
```

3. Remove lens distortion using undistortImage. **(1)**

```
imgLeft = undistortImage(imgLeft, cameraParams);
```

4. View the images using `imshow`.

5. Convert the images to grayscale. **(1)**

6. Detect and extract SURF features in each image. **(1)**

7. Match the features and generate a new list of matched points from the detected points. **(2)**

8. Estimate the fundamental matrix using `estimateFundamentalMatrix`. **(1)**

```
[F, inliers] = ...
estimateFundamentalMatrix(matchPoints_L,matchPoints_R);
```

9. Generate a new list of inlier points from the matched points. **(1)**

10. Compute the homography transformation matrices for each image using `estimateUncalibratedRectification`. **(1)**

```
[H_L,H_R] = ...
estimateUncalibratedRectification(F,inlierPoints_L,...
inlierPoints_R, size(imgGray_L));
```

11. Convert the homography transformation matrices into transform objects using `projective2d`. **(1)**

```
tform_L = projective2d(H_L);
tform_R = projective2d(H_R);
```

12. Rectify the images using `imwarp` to apply the homography transformations. **(1)**

```
imgRectified_L = imwarp(imgLeft,tform_L,'OutputView',...
imref2d(size(imgGray_L)));
imgRectified_R = imwarp(imgRight,tform_R,'OutputView',...
imref2d(size(imgGray_L)));
```

13. Convert the rectified images to grayscale. **(1)**

14. Compute the disparity map using `disparityBM`. Note that the range of disparity values and block size depend on the image resolution and the translation/rotation between camera poses. The values below can be used for 4K images (e.g., typical smartphone images). **(1)**

```
disparityRange = [-256 256];
blockSize = 65;
disparityMap = disparityBM(imgRectifiedGray_L,imgRectifiedGray_R,...
'DisparityRange',disparityRange,'BlockSize',blockSize,...
'UniquenessThreshold',0,'ContrastThreshold',0.9)
```

15. Convert the disparity map to a depth map using the relationship between disparity and depth. Note that the focal length *f* can be extracted from the intrinsic camera matrix in the camera parameters object but the baseline distance between cameras is unknown (you can assume a value of *b* = 1). Also note that the minimum disparity should be 0 but due to camera rotation and resulting imperfection in rectification you may need to add some offset to the disparity map. **(1)**

```
disparityOffset = 32;
f = cameraParameters.IntrinsicMatrix(1,1);
b = 1;
depthMap = b*f./(disparityMap + disparityOffset);
```

16. To build a 3D point cloud from the depth map, compute homogeneous coordinates for a grid of pixel coordinates using the intrinsic camera matrix, then multiply by the depth map. **(1)**

```
width = size(imgLeft,2);
height = size(imgLeft,1);
[U,V] = meshgrid(0:(width-1),0:(height-1));
C = cameraParameters.IntrinsicMatrix';
XY1 = (inv(C)*[U(:) V(:) ones(numel(U))]')';
xyz = XY1.*depthMap(:);
```

17. Compile a matching list of the RGB colour values from the left image. **(2)**

```
imgLeftRed = imgLeft(:,:,1);
imgLeftGreen = imgLeft(:,:,2);
imgLeftBlue = imgLeft(:,:,3);
rgb = [imgLeftRed(:) imgLeftGreen(:) imgLeftBlue(:)];
```

18. Plot the coloured 3D point cloud in a new figure window.

```
scatter3(xyz(:,1),xyz(:,2),xyz(:,3),1,rgb);
```