3D RECONSTRUCTION.
COMPUTER VISION PROJECT

Boiko Oleksandr
COSI 2018

Input data: series of color and depth image pairs from 2 Kinect cameras (scene image pair and pairs for calibration)

1. Calibration of stereo setup

Calibration of stereo setup was performed during Lab 2 using Camera Calibration Toolbox.

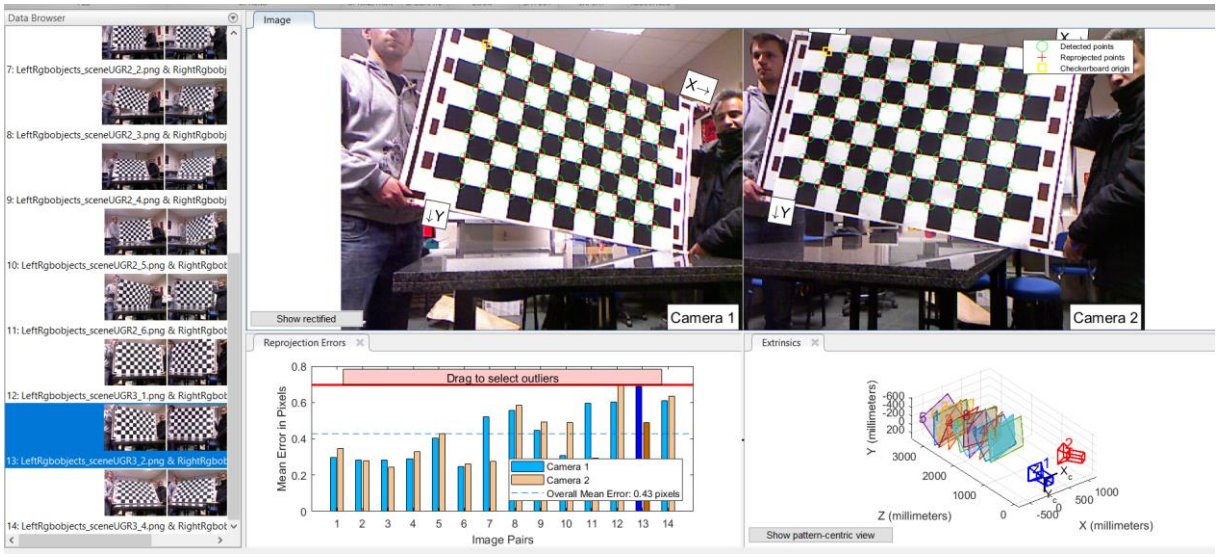
- Firstly, we chose corner points of the checkerboard
- Calibrated each camera separately (to get intrinsic parameters)
- Then calibrated one camera with respect of each other (to get extrinsic parameters).

```
% Intrinsic parameters of left camera:
%
% Focal Length:          fc_left = [ 525.91555   528.76660 ] ± [ 6.91756   7.27946 ]
% Principal point:       cc_left = [ 319.45571   273.24978 ] ± [ 4.97461   4.40869 ]
% Skew:             alpha_c_left = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
% Distortion:            kc_left = [ 0.18264   -0.26293   0.01044   0.00464  0.00000 ] ± [ 0.02498   0.08262   0.00407   0.00403  0.00000 ]
%
%
% Intrinsic parameters of right camera:
%
% Focal Length:          fc_right = [ 523.04244   525.87810 ] ± [ 7.25685   7.56044 ]
% Principal point:       cc_right = [ 312.21580   263.92612 ] ± [ 5.30806   3.87365 ]
% Skew:             alpha_c_right = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
% Distortion:            kc_right = [ 0.19469   -0.36418   0.00106   0.00209  0.00000 ] ± [ 0.02016   0.06069   0.00352   0.00418  0.00000 ]
%
%
% Extrinsic parameters (position of right camera wrt left camera):
%
% Rotation vector:             om = [ 0.01575   0.53349  -0.03561 ]
% Translation vector:           T = [ -1039.09504   -16.65551  425.33596 ]
```

Also, I additionally performed calibration using Stereo Calibration App (built-in in Computer Vision Toolbox)

There, I chose at first all image pairs and then noticed that some pairs give a big estimation error. Thus, such pairs I didn't take into account and recalibrated system until I got appropriate result.

This additional calibration was performed after I understood that first calibration results weren't precise enough (the estimation error was at least twice bigger and for scene recovery it played a big role).



So, finally an estimation error was 0.43 pixels which is quite good. Visually, the position of second camera corresponds a real scene.

Checkerboard Square size was measured during Lab class: 0.11m*0.11m.

```
%calibration session. Data from Mat file

intr1 = [5.232990436019200e+02,0,0;
    0,5.251311849255515e+02,0;
    3.149598561462843e+02,2.624650685435734e+02,1];
focal1 = [  523.2990  525.1312];

intr2 = [518.295280009075,0,0;
    0,520.765343304700,0;
    312.685144618972,265.375512734611,1];
focal  = [5.182952800090750e+02,5.207653433047001e+02];

R_of_cam2 = [[0.863676805198133,-0.031294481288067,-0.503073584680866;
    0.048425076658019,0.998605690531141,0.021016345769168;
    0.501714448778796,-0.042512707273649,0.863988010105959];

T_of_cam2 = [-2.986067926449093e+02,6.603270651370639;65.734880067794560];

mean_error = 0.455866631153184;
```

Data from second calibration was used in the project can be found in data.mat file.

2. Image preprocessing

Here you can see images obtained from stereo setup:

Picture.

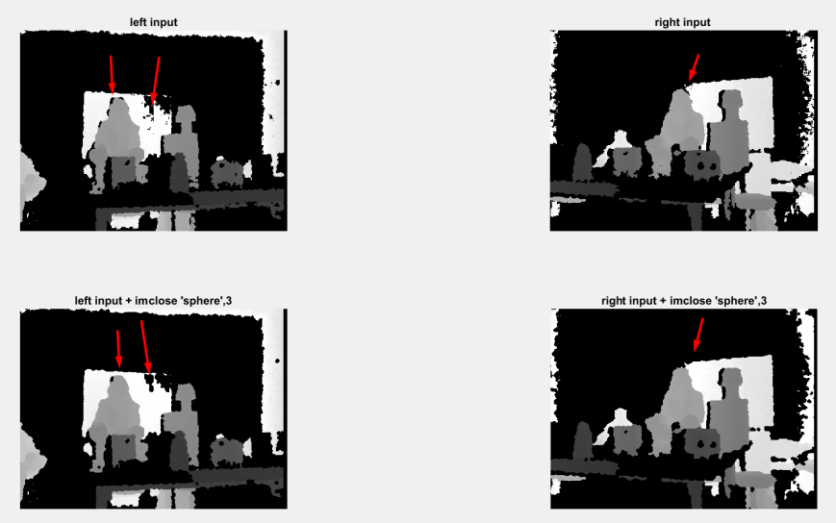
RGB Images obtained from left and right cameras respectively.

Picture. Depth images obtained from left and right cameras respectively.

We can notice that depth Images are quite noisy (especially near borders) and some regions are not filled as expected (regions near borders of objects, for example). Causes will be explained in Limitations chapter.

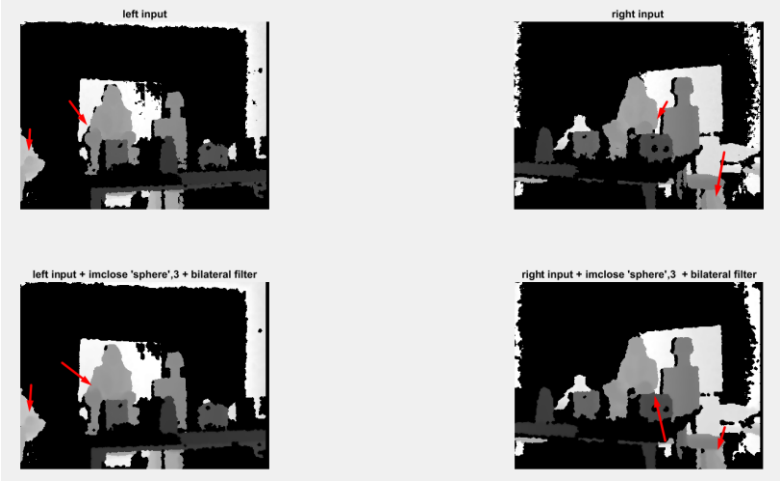
To preprocess images, I decided to apply morphological operations and filtering.

For morphological operations, first I tried just dilation – but then after rendering 3D scene I noticed that some additional parts which I don't need are shown (for example, Tanzima had white nimbus over her head from background). So, I decided to use closing – first, to fill the holes in depth image and then erode small outlier parts.

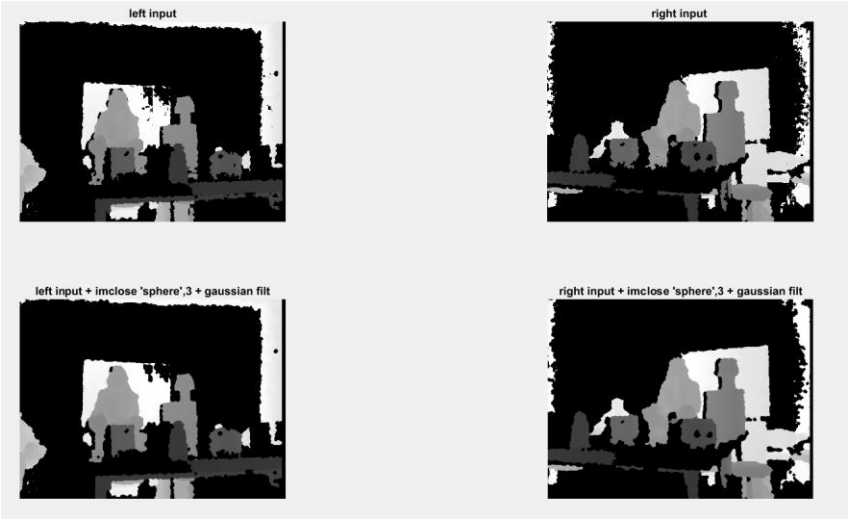


Picture. Example of morphological operation. Trade-off between restoring some object contours but saving holes in game cubes.

Next step was filtering – I tried two filters – bilateral [1] and gaussian filtering.
Limitations of bilateral filter: it can also interpret impulse noise spikes as forming an edge.
Limitations of Gaussian filtering: identical filtering for all data without considering the different saliency.
I tried different coefficients for both filters to get as good result as possible.

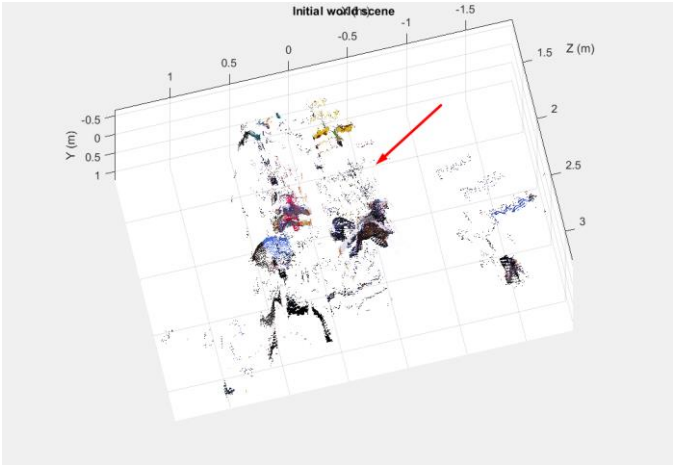


Picture. Example of mixing morphological operation + bilateral filtering.

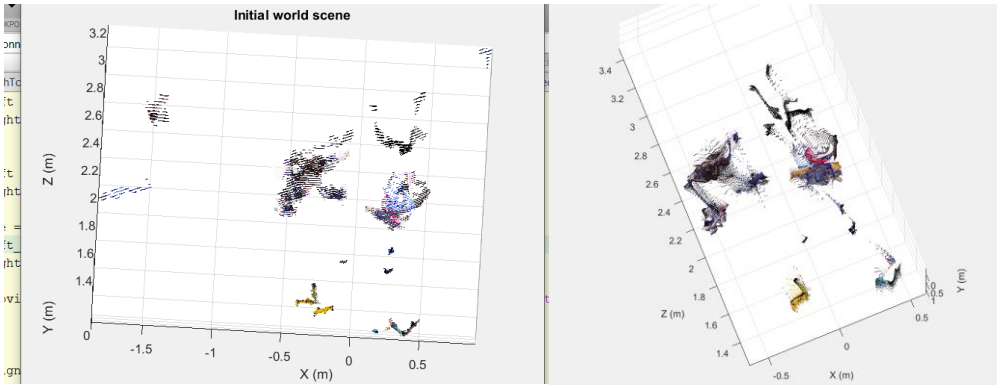


Picture. Example of mixing morphological operation + gaussian filtering.

Noise after applying gaussian filtering:



Noise after bilateral filtering:



To improve the result adaptive bilateral filtering can be used.

So, I decided to use morphological imerode + bilateral filtering, they produced the best result visually.

3. Calibrate the two depth cameras with respect of RGB data

Since depth and color information is captured by different sensors, the mapping between RGB and depth sensor is required.

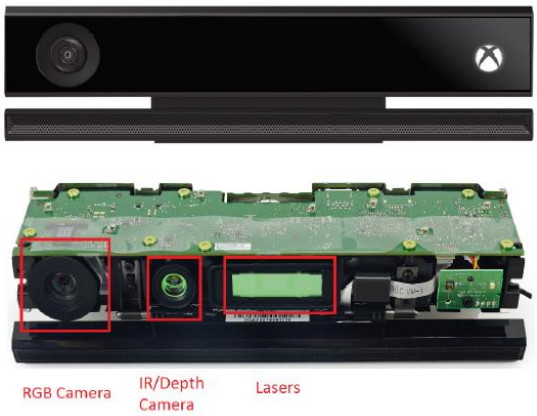
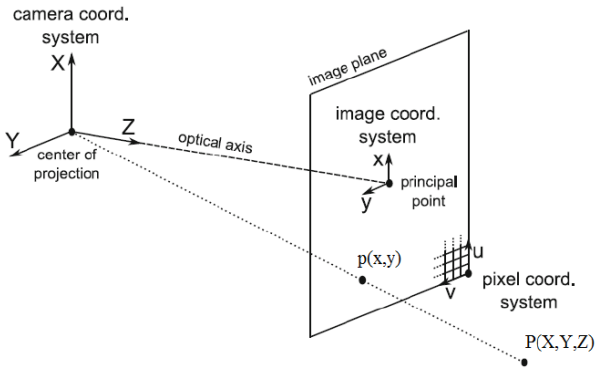


Image. Kinect setup

For this step, data from Kinect Parameters.m file was used.



First of all, depth image has uint16 data type which was converted to double. Next step – data there is represented in mm (for ex., 4000) so I divided it by 1000 to get values in m.

The principal point distance from the pixel coordinate origin is also required, let its coordinates be (x0; y0).

Point coordinates:

$$u = k_u(x + x_0)$$

$$v = k_v(y + y_0)$$

Camera intrinsic matrix K consists of: In pixels these represent focal length (u; v) and coordinates of the principal point (u0; v0).

$$\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The parameters are known:

```
% Kinect Depth camera parameters
fx_d =  5.7616540758591043e+02;
fy_d = 5.7375619782082447e+02;
cx_d = 3.2442516903961865e+02;
cy_d = 2.3584766381177013e+02;
```

To transform 2D depth to 3D depth camera coordinates image:

$$ZK^{-1}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

We can solve the matrix equation and then:

Z – depth value, u, v – pixel number for rows and columns respectively.

X,Y,Z – camera coordinates.

For the intrinsic parameters we assume that pixel has size 1:1 (but it's not exactly square pixel).

So, first we create a 3d point cloud (left and right depth images separately), then we do alignment for left and right Kinect setup.

```
% convert depth image to 3d point clouds
pcloud_left = zeros(rows,cols,3);
pcloud_right = zeros(rows,cols,3);

xgrid = ones(rows,1)*(1:cols) - cx_d;
ygrid = (1:rows)'*ones(1,cols) - cy_d;

pcloud_left(:,:,1) = xgrid.*depth_l/fx_d;
pcloud_left(:,:,2) = ygrid.*depth_l/fy_d;
pcloud_left(:,:,3) = depth_l;
```

```
% left Kinect alignment of Depth camera

for i = 1:rows
    for j = 1:cols
        pcloud_align_Left(i,j,1:3)=R_extr_cam*[pcloud_left(i,j,1); pcloud_left(i,j,2); pcloud_left(i,j,3)]+T_extr_cam;
    end
end
```

Alignment of depth and color sensor. Rigid transformation

R_extr_cam, T_extr_cam – data from from Kinect Parameters.m

4. Align left and right Kinects using Extrinsic calibration data

```
% right Kinect alignment with left Kinect
for i = 1:rows
    for j = 1:cols
        Right_align(i,j,1:3)=(R)*[pcloud_align_Right(i,j,1); pcloud_align_Right(i,j,2); pcloud_align_Right(i,j,3)]+T';
    end
end
```

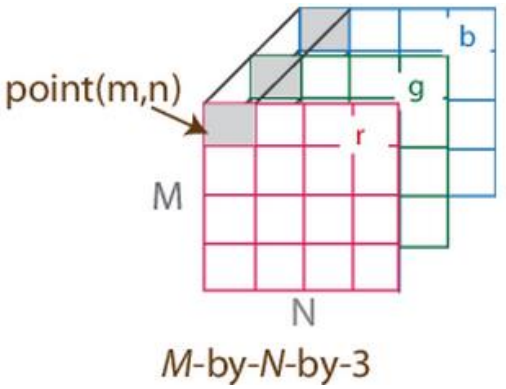
R,T – parameters estimated during camera calibration.

5. Generate Point Clouds for left and right Kinects. Color pixel + Depth

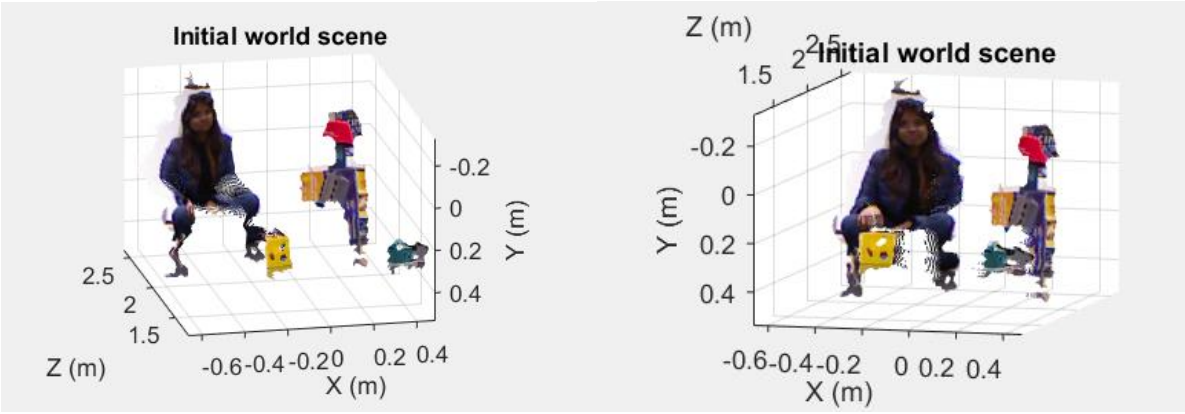
```
ptCloudLeft = pointCloud(pcloud_align_Left,'Color',rgb_l);
ptCloudRight = pointCloud(Right_align,'Color',rgb);
```

Output of cloud point:

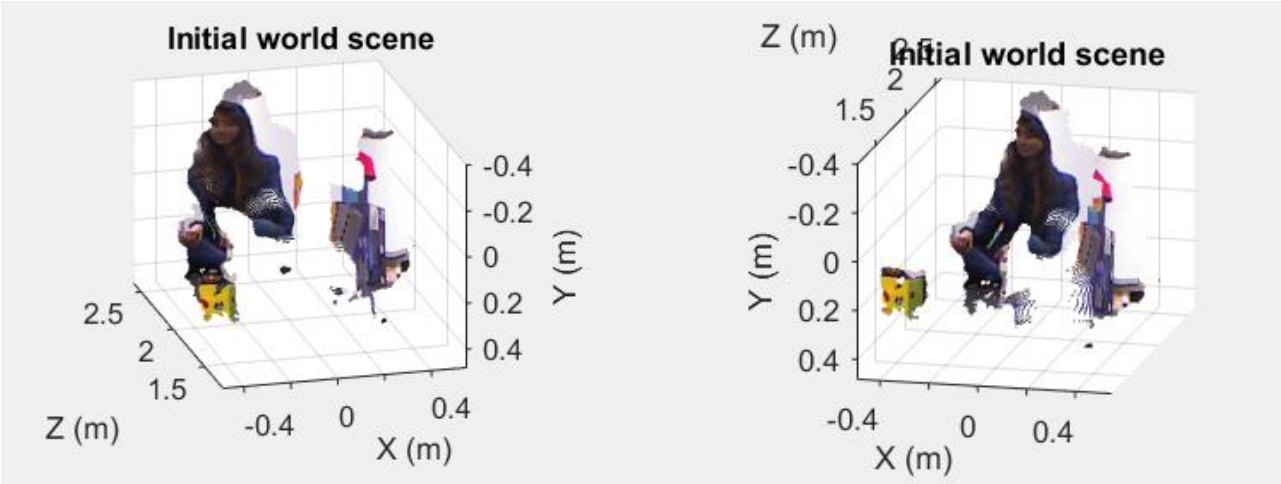
1) Coordinates specified by xyzPoints
2) Point cloud color specified as the comma-separated pair of 'Color' and an 480-by640-by-3 matrix containing RGB values for each point.



Left Point Cloud:



Right Point Cloud:

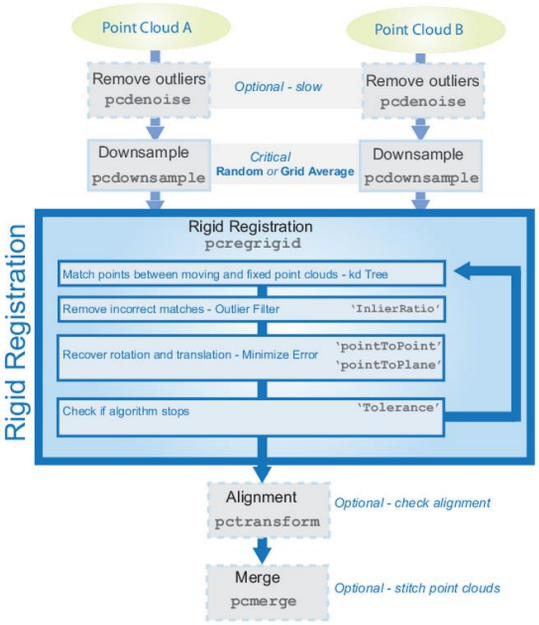


6. Point Cloud Registration Workflow

Firstly, it seems that after all the transformations dine before cloud points should be perfectly aligned which is not the case. I think that it's because we have a lot of ambiguities during capturing.

So, to align points method of Point Cloud Registration was used [].

General information about method:



For evaluation of estimated transformation is done by root mean squared error of the Euclidean distance between the aligned point clouds.

Performed steps:

1. First operation – try to reduce noise by pcdenoise function which removes outliers from Noisy Point Cloud.

```
ptCloudLeft = pcdenoise(ptCloudLeft);
ptCloudRight = pcdenoise(ptCloudRight);
```

This operation showed quite good result, but I've noticed that it reduces a huge amount points, good ones as well. So, I decided to filter cloud points by another method described below.

At first, I decided to choose only some region where the most important part is – desk with objects and Tanzima. So, I filtered depth map from 1.2m to 3.5m:

But it stil gave me a big RMSE value (approx. rmse = 0.46) and the result of reconstruction was still quite bad.

So, next step – I selected only central part and distance from 2 m to 3 m – to choose Tanzima to perform calibration, it's easier to evaluate the result of reconstruction.

Selection of central region (setting pixel range):

```
dep(:,1:100) = nan;
dep_l(:,1:100) = nan;
dep(:,380:640) = nan;
dep_l(:,340:640) = nan;


dep(1:80,:) = nan;
dep_l(1:80,:) = nan;
dep(300:480,:) = nan;
dep_l(300:480,:) = nan;
```

Selection of depth:

```
depth(depth == 0) = nan;
depth(depth > 2.9) = nan;
depth(depth < 2) = nan;
```

Result:

After this step, choosing appropriate number of points and method the rmse decreased to 0.02 which is quite good.

2. Next step is down sampling of points and registration of cloud point from right camera regarding to left one. Sampling is done to reduce the number of points and there are different variations of down sampling.

First, I tried to use random down sampling for both clouds.

Random downsample method, specified as the character vector, 'random'. This method is more efficient than the 'gridAverage' downsample method, especially when it is applied before point cloud registration.

Percentage of input, specified as a positive scalar in the range [0, 1]. The percentage input specifies the portion of the input for the function to return.

Then I tried nonuniform grid sample method, specified as the character vector 'nonuniformGridSample'. The best use of this method is to apply it as a preprocessing step to the pcregrigid function for point cloud registration, when you use the 'pointToPlane' metric. When you use the 'nonuniformGridSample' algorithm, the normals are computed on the original data prior to downsampling. The downsampled output preserves more accurate normals.

Metric	Point Cloud A Downsample Method	Point Cloud B Downsample Method
'pointToPoint'	'random'	'random'
	'gridAverage'	'gridAverage'
'pointToPlane'	'gridAverage'	'gridAverage'
	'random'	'nonuniformGridSample'

The result of cloud point rigid transformation estimation strongly depends on number of points chosen.

[tform,movingReg,rmse] = pcregrigid(ptCloudRight_samp,ptCloudLeft_samp,'Extrapolate',true,'Metric','PointToPoint','Tolerance',[0.001, 0.0009]);

tform – estimated rigid transform matrix
movingReg – aligned clouds (left + transformed right)
rmse – root minus square error between projected and original points

In our case point-to-plane method was proposed as the best one. By the way, practically point-to-point random downsampling worked much better. In first case, rmse was equal to 0.07-0.20, while in second case it was reduced to 0.02-0.05 by choosing proper percentage of points. 50 percent of points showed the best result. Each time the algorithm chooses different points, that's why from iteration to iteration rmse can be different. Remind that transformation was performed for different variations of depth images (for all scene; for partially reduced scene by choosing special distances (from 1.5 m to 4 m, from 2 m to 3.5 m ect); for central part with Tanzima, depth range – 2m – 2.9m)

Extrapolation specified as the comma-separated pair consisting of 'Extrapolate' and the boolean true or false. When you set this property to true, the function adds an extrapolation step that traces out a path in the registration state space. Setting this property to true can reduce the number of iterations to converge.

'Tolerance' — Tolerance between consecutive ICP iterations [0.01, 0.009] (default) | 2-element vector

I changed tolerance to get more precise result. [0.001, 0.0009];

3. Next step to show the result: there are 2 choices, we can show **movingReg**, which is quite nice, or we can apply transformation using **tform** matrix to initial right point cloud.

The difference between methods: first – everything is aligned and transformed automatically, but applied to the point clouds used for estimation, which can be downsampled – means we can less points after transformation.

Second: we can apply transform to initial clouds and then merge; transformation will be found downsampled clouds, but output cloud will be merged from input clouds(left and right transformed) and then we get more points, but the problem is that we have more noise. Examples attached below:

```
ptCloudAligned = pctransform(ptCloudRight,tform);
ptCloudScene = pcmerge(ptCloudLeft, ptCloudAligned, mergeSize);
```

Rmse - 0.0316

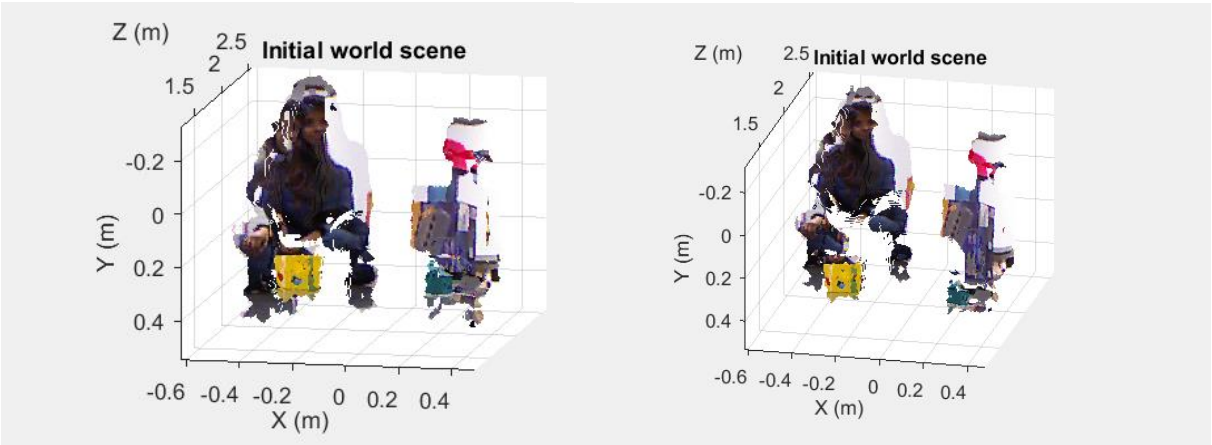


Image. 1st method – no noise. Performed with initial point clouds without denoising and downsamling

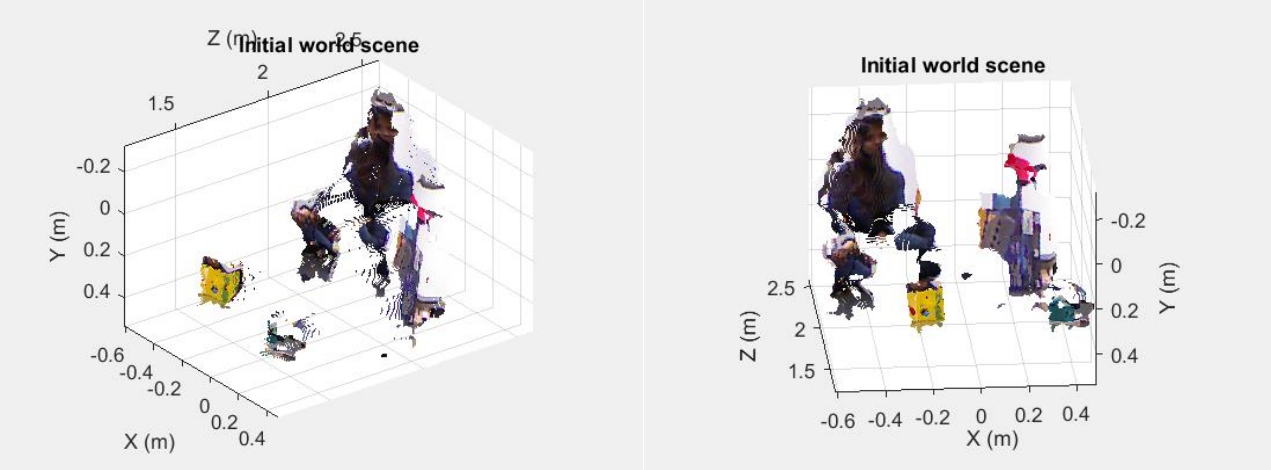
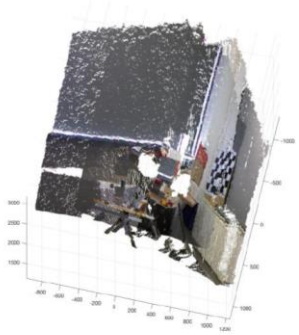


Image. 2nd with applying transform to initial point cloud and merging them. Noise is noticeable

Another way is trying to align Point clouds by advanced Point Cloud Registration or by manual stitching as was done in example from pdf:



3D reconstruction using left and right views after **manual alignment** of right images, note that there is another alignment problem but this time between the left and the right view.

But I think that it's not good to align it manually – because it will be useful only for current scene.

Limitations of depth camera:

1) **Systematic distance error:** Approximations in the sinusoidal signal shape lead to some systematic depth error measurements in Kinect sensor. The magnitude of the systematic error has been shown to be relatively small, in the order of 1-2 m
2) **Depth inhomogeneity:** At object boundaries pixels can have inconsistent depth values. This is because some of the light reflected is obtained from object but some from the background, mixing that information together can produce so called flying pixels which show object boundaries at incorrect coordinates.
3) **Multi-path effects:** Since time of flight measuring principle relies on capturing light reflected back from the object surface, it can happen that light does not travel directly from illumination unit to object and back to sensor but instead takes indirect path being reflected from several surfaces. When such info is captured by sensor the depth measurement combines several waves and gives incorrect results. For very reflective surfaces, if it is positioned at relatively flat angle towards camera, no light might be reflected back and there is no depth info about that surface
4) **Semitransparent and scattering media:** When surface, such as glass, only reflects part of the light directly and some other part is reflected from within the object, there is additional phase shift and the depth measurement is incorrect. For example, we had a bottle which was transparent – it wasn't detected properly.
5) **It cannot handle reflections well**. If object is transparent and the same wave gets partially reflected from object surface and partially reflected from background, the resulting depth image does not correspond to the true

situation (ex. – table) light has to reflect back to sensor for Kinect to register it these objects might not appear in depth image at all or have very distorted measurements

6) **Additionally, the method doesn't work with fully symmetrical objects**. If the depth info is identical in two frames, then ICP algorithm converges already with initial transformation guess (identity transformation) and the frames are not aligned correctly.

According to our experiments and analysis, we find that the gross error of depth data of Kinect sensor can be categorized into two types. The first will appear when the edges of object surface are scanned, and the second will come when some objects that are not well reflective are scanned – table, for example.

References

[1] C. Tomasi and R. Manduchi. 1998. Bilateral Filtering for Gray and Color Images. In Proceedings of the Sixth International Conference on Computer Vision (ICCV '98). IEEE Computer Society, Washington, DC, USA, 839-.

[2] Chen, Y. and G. Medioni. "Object Modelling by Registration of Multiple Range Images." *Image Vision Computing*. Butterworth-Heinemann . Vol. 10, Issue 3, April 1992, pp. 145-155.

[3] Besl, Paul J., N. D. McKay. "A Method for Registration of 3-D Shapes." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Los Alamitos, CA: IEEE Computer Society. Vol. 14, Issue 2, 1992, pp. 239-256.

[4] Diana-Margarita Córdova-Esparza, Juan R. Terven, Hugo Jiménez-Hernández, Alberto Vázquez-Cervantes, Ana-Marcela Herrera-Navarro, Alfonso Ramírez-Pedraza ''Multiple Kinect V2 Calibration''

[5] ''A Post-Rectification Approach of Depth Images of Kinect v2 for 3D Reconstruction of Indoor Scenes'' Jichao Jiao 1,* ID , Libin Yuan 1 ID , Weihua Tang 2, Zhongliang Deng 1 and Qi Wu 1

[6] ''3D Surface Reconstruction Based on Kinect Sensor'' Song Tiangang, Lyu Zhou, Ding Xinyang, and Wan Yi

[7] Lembit Valgma ''3D reconstruction using Kinect v2 camera''

Also:

- Microsoft Kinect V2 & Windows adapter for PC.
- Acquisition Using Kinect for Windows Hardware http://es.mathworks.com/help/imaq/acquisition-using-kinect-for-windows-hardware.html?requestedDomain=www.mathworks.com
- MATLAB and Simulink http://es.mathworks.com/hardware-support/kinect-windows.html?requestedDomain=www.mathworks.com . Microsoft Kinect for Windows Support from Image Acquisition Toolbox.
- Simulink Support for Kinect http://www.mathworks.com/matlabcentral/fileexchange/32318-simulink-support-for-kinect
- Kinect 2 Interface for Matlab (http://www.mathworks.com/matlabcentral/fileexchange/53439-kinect-2-interface-for-matlab)
- https://pdfs.semanticscholar.org/presentation/a4d7/9c1590fcb5feada4a5ebec4530e1bc74ef70.pdf
- https://www.mathworks.com/matlabcentral/fileexchange/12191-bilateral-filtering