Advanced Vision - Assignment 2

Mihail Dunaev - s1567045

Amr Mashlah - s1574638

Introduction

In this work we present a method to register and fuse 16 range images and to extract a structures from the fused data, the images are taken by kinect depth sensor and contains colored and ranged data

Task 1 - Background plane extraction

We tried different approaches to extract the background. Using region growing algorithm with random starting point we were able to detect five regions then we consider the largest region as the floor plane and then we remove all points in this plane from the image. This method works fine but with high computational cost, since it has to deal with all points in the image.

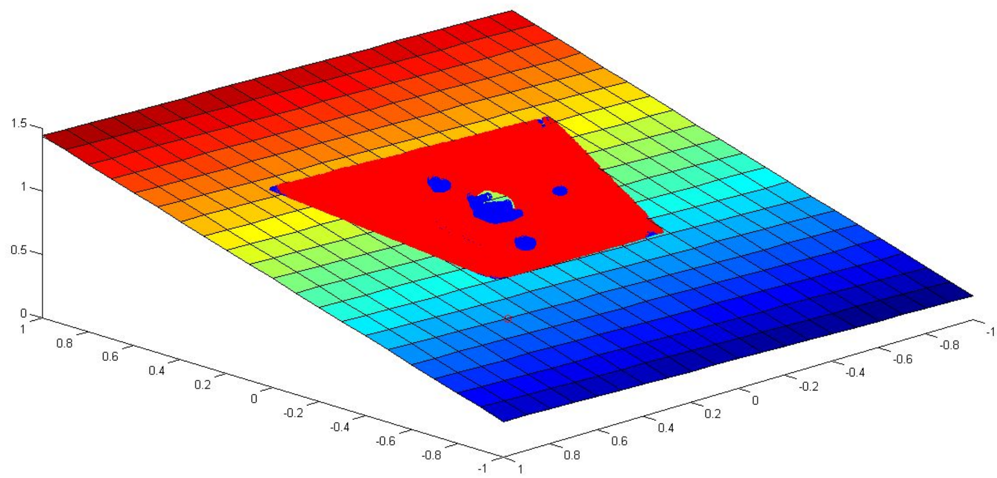


Figure: The extracted points belonging to the plane with red, everything else (the 4 objects and noise) with blue and the fitted plane with gradient colors.

Alternatively, we tried to identify starting point for the region growing algorithm. For each image we started with 4 points near each corner, at the following positions (100,100). (540,100), (100,380), (540,380) (image size is 480x640). We then tried to fit a plane through these 4 points using the provided function from the lecturer's code(fit_plane)

We used the plane extraction algorithm (as described in the lectures) to extract points close to the plane (we removed those closer than 0.007 to the plane, and those that were beyond the plane, since we are only interested in the objects on top of the plane). We obtained clean results with decent performance (~ 0.8 sec/view).

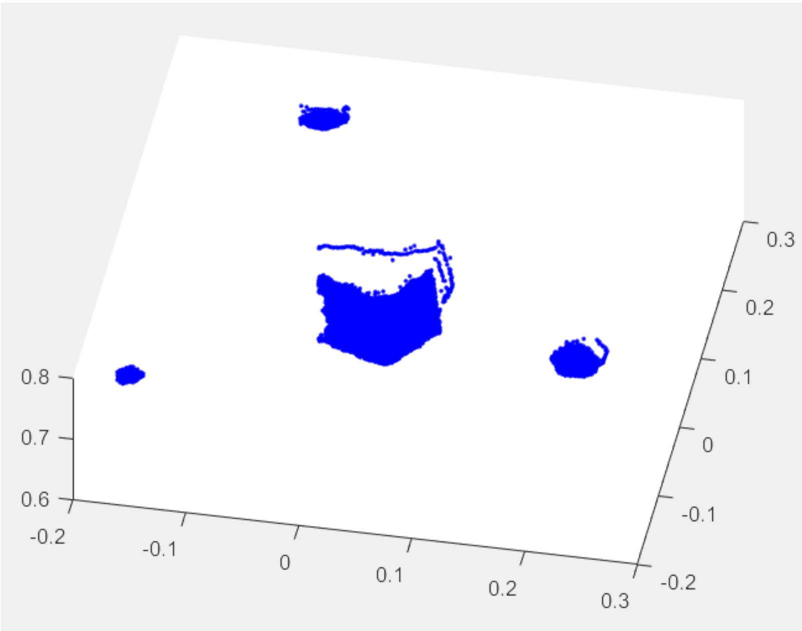


Figure: extracted points, after background plane removal, (in view 1).

Task 2 - Sphere extraction

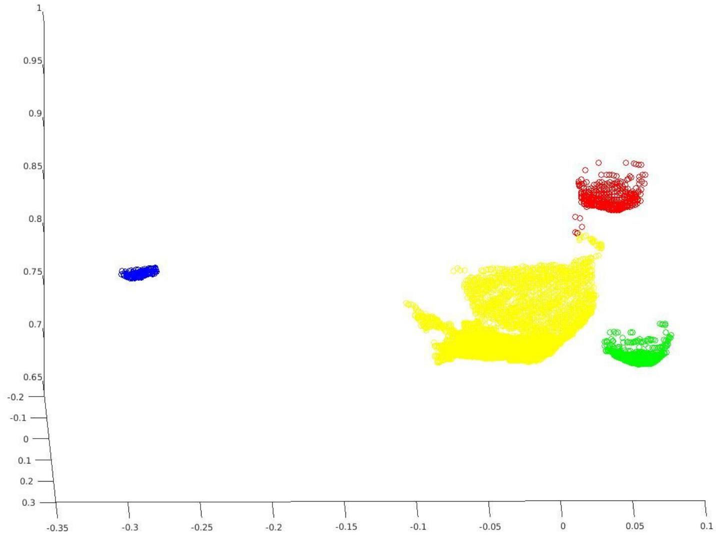
In the dataset there are three spheres left intentionally in all the views to help with inferring the angle of the view. In order to use this information we need to extract the spheres, calculate their centers and associate each one of them with different label.

K-means

We used k-means as a clustering algorithm, starting with 4 center points and ending up with 4 clustered regions.

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

Unfortunately the algorithm detects noise generated from the scattered points of the hidden side of the main object. This noise get clustered with one of the spheres causing a shift to the center when using *sphere_fit* function. That is because *sphere_fit* tries to fit a sphere that will have most of the points in the cluster on the surface of the sphere, and noise will give big errors in the fit.



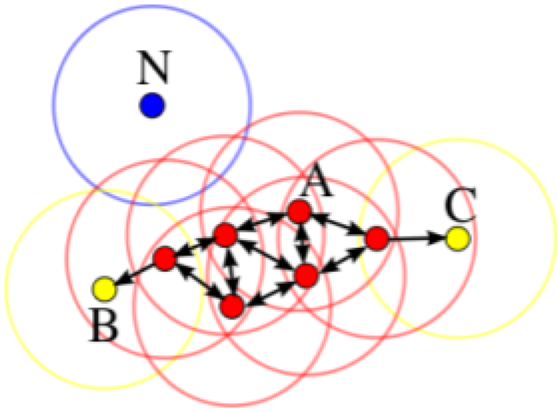
Results of clustering using k-means algorithm

This method worked for 11 out of 16 views (accuracy is 68.75%) and for the rest of 5 views it detects noise and sometimes it splits the main object into two clusters.

Density-based spatial clustering of applications with noise (DBSCAN)

To deal with the noise we found that DBSCAN is more robust to noise since it eliminates points that lie in low-density area.

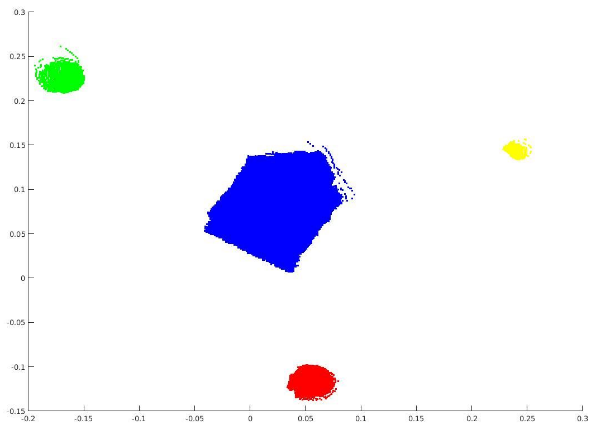
A cluster in DBSACN is formed based on a set of points that are mutually density-connected (the core of the cluster) and all points that are density-reachable from any point of the cluster[4].



DBSCAN algorithm [4]

In the figure red points represent the core of the cluster since they are all reachable from one another, while B and C are not core but they are still considered as part of the cluster since they are reachable from at least one point from the cluster within a certain distance. Whereas point N is considered out of the cluster since it is neither a core point nor reachable by other points in the cluster

Results:



Clustering result using DBSCAN algorithm

With dbscan we were able to detect/cluster the 4 objects correctly in each view. (accuracy 100%).

Calculating the center of the spheres:

The spheres in the views are meant to be used as a reference to calculate the rotation and translation of each view. In order to exploit this information, we need to first extract the three spheres and calculate their centers of mass. Lastly, we need to label each sphere with the same label for all the views

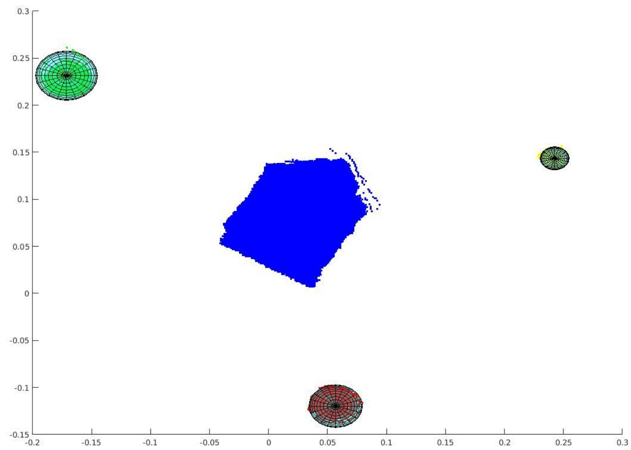
After clustering with DBSCAN, we took the four biggest regions as our objects (the cube and the three spheres). We considered the biggest one as the cube and the other three ones as the spheres. We used *sphere_fit* to fit spheres for these points and compute the centers. Next, we mapped the XYZ points in each sphere to an RGB value (from the RGB image). We did this to take the histogram for each sphere. We saved the histograms from first view and compared to those from other views, to correctly map the same spheres. We used the Bhattacharyya distance to compare the histograms. This method worked for all views. Once correctly mapped, we just needed to compute the centers for each sphere (from the few sphere points we had; we use sphere fit) and translate and rotate them such that they overlap the centers of spheres from the first view.

The algorithm for this task is described below:

- Cluster all the remaining XYZ points (after plane removal) with DBSCAN
- Take the biggest four regions
- Compute histogram for smallest three regions
- If it's the first image, save histograms
- If not, compare histograms with the ones from first image (previously saved)
- Use this (take smallest Bhattacharyya distance) to match spheres
- Compute translation and rotation between the two views

We tried to calculate the center of the points for each cluster to get the centers of the spheres but this method does not work since the 3D points are not equally captured around the sphere and due to the noise points that cause a shift in the center of mass. This shift can not be tolerated as we need an accurate centeres to calculate the rotation of the whole image

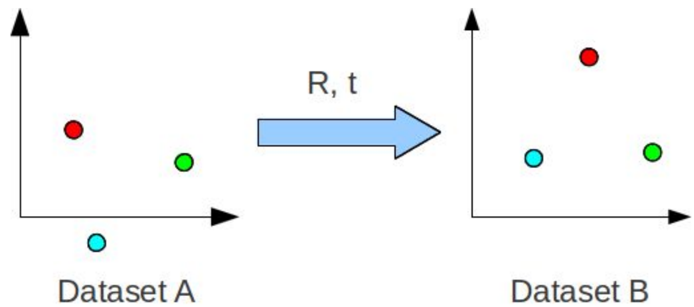
Alternatively, we used the sphere fit algorithm provided in the lectures' code. We fit a sphere to each cluster of points and then we calculate the center of this sphere



Results
By visualising the fitted spheres on top of the 3D points we can verify that this algorithm worked for all the views

Task 3 - Registration

Now that we have the centers of the three spheres in all views, we need to pair them with a baseline which is the first view in our case.



Calculating R, t for a dataset based on reference dataset [2]

We used rigid transform function from[5] to compute the translation and rotation matrix. We then applied these transformations to the main object points in each view, to get them to overlap with those from the first view and reconstruct the whole 3D model. The algorithm used is the following:

- Compute the center of the three points (centers for each sphere)

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$centroid_A = \frac{1}{N}\sum_{i=1}^{N} P_A^i$$

$$centroid_B = \frac{1}{N}\sum_{i=1}^{N} P_B^i$$

- Compute the covariance matrix (H) according to formula 2 [2]

$$H = \sum_{i=1}^{N}(P_A^i - centroid_A)(P_B^i - centroid_B)^T$$

$$[U, S, V] = SVD(H)$$

$$R = VU^T$$

Formula 2: SVD

- Use Singular Value Decomposition (SVD) on matrix H [U,S,V] = H
- Use U and V to compute the rotation matrix R = VU' in formula 2
- Compute translation matrix using formula:

$$t = -R \times centroid_A + centroid_B$$

After we rotated and translated the points from all the views, we needed to merge them in the same data structure. We used the built-in matlab function *union* to merge/fuse the values. A representation of the 3D model we obtained can be seen in Figure [8], where points from different views have different colors.

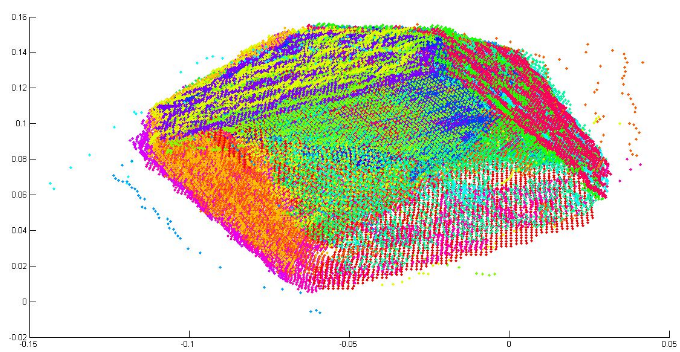


Figure 8: Registration of points from different views, with a different color for each view.

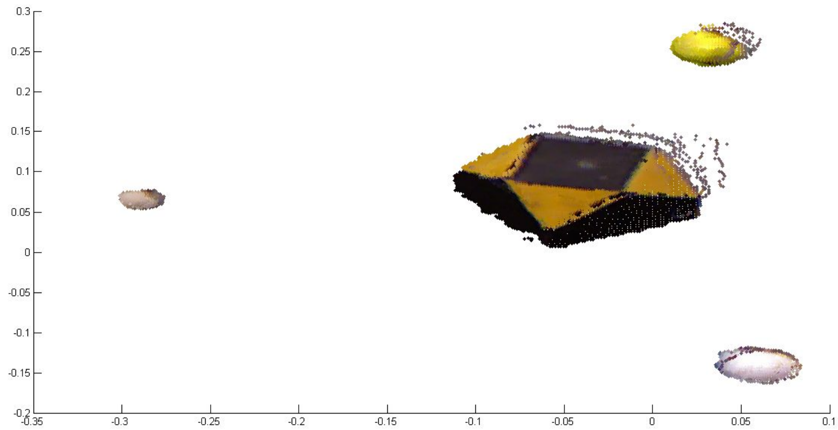


Figure 9: Registration from different views, including the sphere points, with colors from original RGB image.

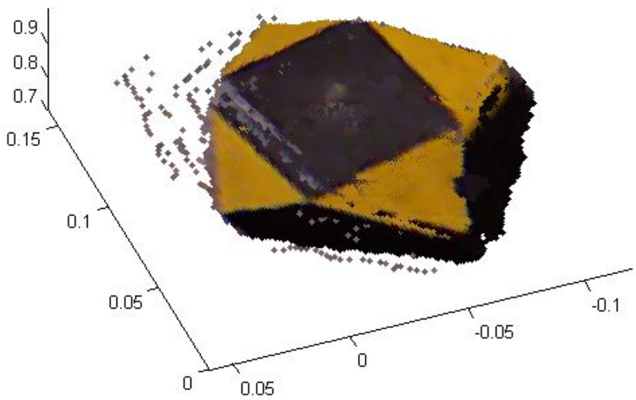


Figure 10: Registration of points from different views, for the main object, with colors from original image

Task 4 - Object plane extraction

To extract the 9 planes of the object, we used the same algorithm used for task 1 (background plane removal). This time we started with some random point. We then looked at all the points inside a sphere of radius 0.01 from that point and used these as our starting points for the plane fitting algorithm. Using a sphere will guarantee us that the plane will have the right orientation (there will be more points across the plane direction).

We then added all the points from the image that are within 0.01 distance to the plane and other points already close to the plane. If more than 50 points are added, the plane is refitted. This goes on until either the plane has a bad fit, or less than 50 points are added. Finally, if more than 1000 points were extracted for this plane, then we save the plane equation and move on to the next plane. We considered that less than 1000 points would be just noise.

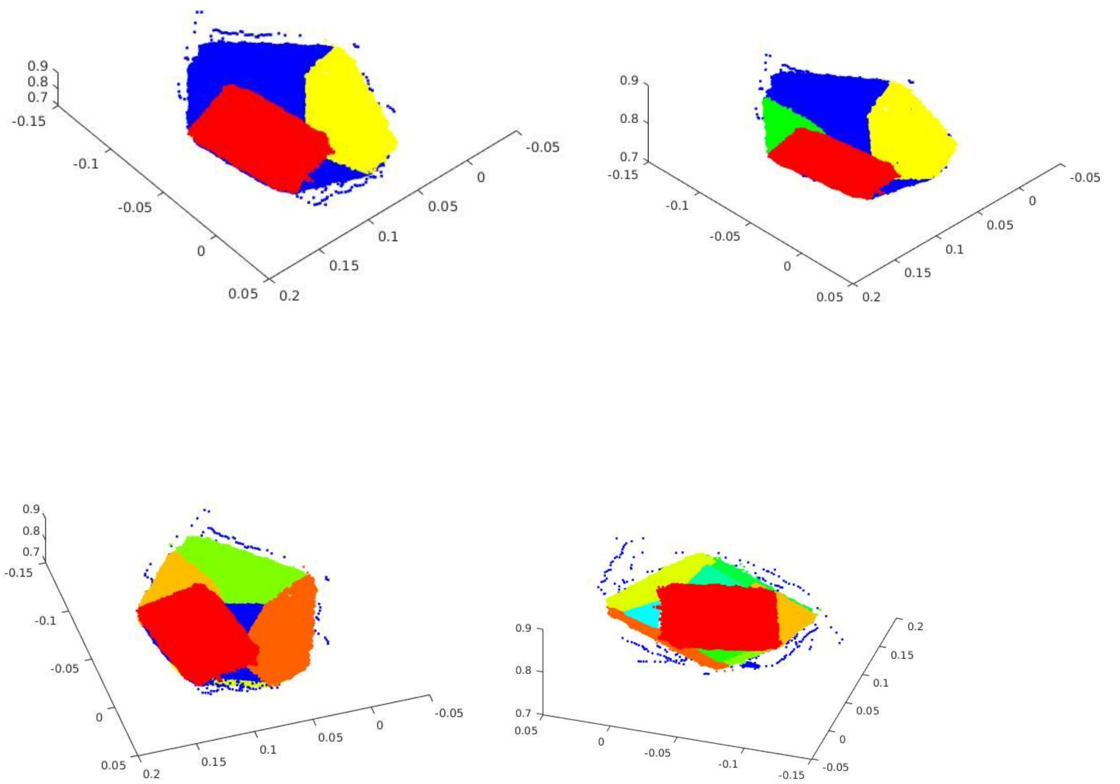


Figure: Plane extraction stages: in first image, 2 planes have been successfully extracted; in the second image 3 planes have been successfully extracted etc. In the last image all the planes have been successfully extracted.

References:

[1] http://www.mathworks.com/matlabcentral/fileexchange/24616-kmeans-clustering

[2] http://nghiaho.com/?page_id=671

[3] https://en.wikipedia.org/wiki/Rigid_transformation

[4] https://en.wikipedia.org/wiki/DBSCAN

[5] http://nghiaho.com/uploads/code/rigid_transform_3D.m

[6] http://www.mathworks.com/matlabcentral/fileexchange/53842-dbscan

Code Appendix:

Main program code:

```
% load data
load('av_pcl.mat');

% some initializations
A = zeros(3,3);
B = zeros(3,3);

% iterate through images
nimgs = 16;
cmap = hsv(nimgs);
for i = 1:nimgs
   img = pcl_cell{i};

   % rgb and depth
   img_rgb = img(:,:,1:3);
   img_depth = img(:,:,4:6);

   % reshape XYZ image as 1D array (col major)
   img_size = size(img_depth,1) * size(img_depth,2);
   img_depth_array = reshape(img_depth,[img_size,3]);

   % plane = plane_pts, objects = obj_pts

   % get 4 points for plane
   plane_start_pts = zeros(4,3);
   plane_start_pts(1,:) = img_depth_array(99*480 + 100,:);
   plane_start_pts(2,:) = img_depth_array(540*480 + 100,:);
   plane_start_pts(3,:) = img_depth_array(99*480 + 381,:);
   plane_start_pts(4,:) = img_depth_array(99*480 + 100,:);

   % remove pts from objects data
   obj_pts = img_depth_array;
   obj_pts(540*480 + 381,:) = [];
   obj_pts(540*480 + 100,:) = [];
   obj_pts(99*480 + 381,:) = [];
   obj_pts(540*480 + 381,:) = [];
```

```
% fit plane
[plane, r] = fitplane(plane_start_pts);
if r >= 0.1
   fprintf('Could not find fit plane from the given points\n');
   return;
end

% get all the points from the plane
plane_pts = zeros(img_size,3);
plane_pts(1:4,:) = plane_start_pts;

% transfer points from obj_pts to plane_pts
eps_plane = 0.01;
j = 1;
num_pts = 4;
tmp_size = img_size - 4;
while true
   if j > tmp_size
      break;
   end
   pnt = obj_pts(j,:);

   % first check if point is beyond the plane
   z_plane = (-plane(4) - plane(1) * pnt(1) - plane(2) * pnt(2))/plane(3);
   if pnt(3) > z_plane
      num_pts = num_pts + 1;
      plane_pts(num_pts,:) = obj_pts(j,:);
      obj_pts(j,:) = [0,0,0];
      j = j + 1;
      continue;
   end

   % check if within certain distance (eps)
   dist_plane = abs([obj_pts(j,:) 1]*plane);
   if dist_plane < eps_plane
      num_pts = num_pts + 1;
      plane_pts(num_pts,:) = obj_pts(j,:);
      obj_pts(j,:) = [0,0,0];
   end
   j = j + 1;
end
```

```
% get rif of the 0 elements
cnt = 0;
for j = 1:size(obj_pts,1)
   if (obj_pts(j,1) == 0) && (obj_pts(j,2) == 0) && ...
         (obj_pts(j,3) == 0)
      continue;
   end
   cnt = cnt + 1;
end

new_obj_pts = zeros(cnt,3);
colors = zeros(cnt,3);
cnt = 0;
for j = 1:size(obj_pts,1)
   if (obj_pts(j,1) == 0) && (obj_pts(j,2) == 0) && ...
         (obj_pts(j,3) == 0)
      continue;
   end
   cnt = cnt + 1;
   new_obj_pts(cnt,:) = obj_pts(j,:);
   [x,y] = map_index_to_pos(j);
   colors(cnt,:) = img_rgb(y,x,:);
end

% cluster regions using dbscan
[la, type] = dbscan(new_obj_pts,5,[]);

% find max
max_la = -1;
sz = size(new_obj_pts,1);

for j = 1:sz
   if la(j) > max_la
      max_la = la(j);
   end
end

% count how many for each cluster
cnts = zeros(max_la,1);
for j = 1:sz
   if la(j) > 0
      cnts(la(j)) = cnts(la(j)) + 1;
```

```
    end
end

% sort descending and get first 4 clusters
[~,is] = sort(cnts,'descend');

obj1 = zeros(cnts(is(1)),3);
colors_obj1 = zeros(cnts(is(1)),3);
obj2 = zeros(cnts(is(2)),3);
obj3 = zeros(cnts(is(3)),3);
obj4 = zeros(cnts(is(4)),3);

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
cnt4 = 0;

for j = 1:sz
    if la(j) == is(1)
        cnt1 = cnt1 + 1;
        obj1(cnt1,:) = new_obj_pts(j,:);
        colors_obj1(cnt1,:) = colors(j,:);
    end
    if la(j) == is(2)
        cnt2 = cnt2 + 1;
        obj2(cnt2,:) = new_obj_pts(j,:);
    end
    if la(j) == is(3)
        cnt3 = cnt3 + 1;
        obj3(cnt3,:) = new_obj_pts(j,:);
    end
    if la(j) == is(4)
        cnt4 = cnt4 + 1;
        obj4(cnt4,:) = new_obj_pts(j,:);
    end
end

% fit sphere for each ball & get center
[cobj2,robj2] = sphereFit(obj2);
[cobj3,robj3] = sphereFit(obj3);
[cobj4,robj4] = sphereFit(obj4);
```

```
% go back to 2D image to compute histogram
lbls = zeros(img_size,1);

% look for occurence of first element from obj2
[~, locb] = my_ismember(obj2(1,:),obj_pts);
lbls(locb) = 2;

cnt = size(obj2,1);
for k = 2:cnt
    while (locb < img_size)
        locb = locb + 1;
        if obj_pts(locb,1) ~= obj2(k,1)
            continue;
        end
        if obj_pts(locb,2) ~= obj2(k,2)
            continue;
        end
        if obj_pts(locb,3) ~= obj2(k,3)
            continue;
        end
        break;
    end
    lbls(locb) = 2;
end

% look for occurence of first element from obj3
[~, locb] = my_ismember(obj3(1,:),obj_pts);
lbls(locb) = 3;

cnt = size(obj3,1);
for k = 2:cnt
    while (locb < img_size)
        locb = locb + 1;
        if obj_pts(locb,1) ~= obj3(k,1)
            continue;
        end
        if obj_pts(locb,2) ~= obj3(k,2)
            continue;
        end
        if obj_pts(locb,3) ~= obj3(k,3)
            continue;
        end
```

```
        break;
    end
    lbls(locb) = 3;
end

% look for occurence of first element from obj3
[~, locb] = my_ismember(obj4(1,:),obj_pts);
lbls(locb) = 4;

cnt = size(obj4,1);
for k = 2:cnt
    while (locb < img_size)
        locb = locb + 1;
        if obj_pts(locb,1) ~= obj4(k,1)
            continue;
        end
        if obj_pts(locb,2) ~= obj4(k,2)
            continue;
        end
        if obj_pts(locb,3) ~= obj4(k,3)
            continue;
        end
        break;
    end
    lbls(locb) = 4;
end

% compute histogram for each ball
% obj2
obj2_sz = size(obj2,1);
histr2 = zeros(1, obj2_sz);
histg2 = zeros(1, obj2_sz);
histb2 = zeros(1, obj2_sz);

% obj3
obj3_sz = size(obj3,1);
histr3 = zeros(1, obj3_sz);
histg3 = zeros(1, obj3_sz);
histb3 = zeros(1, obj3_sz);

% obj4
obj4_sz = size(obj4,1);
```

```
histr4 = zeros(1, obj4_sz);
histg4 = zeros(1, obj4_sz);
histb4 = zeros(1, obj4_sz);

k2 = 1;
k3 = 1;
k4 = 1;
for j = 1:img_size
   if lbls(j) == 2
      [x,y] = map_index_to_pos(j);
      histr2(k2) = img_rgb(y,x,1);
      histg2(k2) = img_rgb(y,x,2);
      histb2(k2) = img_rgb(y,x,3);
      k2 = k2 + 1;
   end
   if lbls(j) == 3
      [x,y] = map_index_to_pos(j);
      histr3(k3) = img_rgb(y,x,1);
      histg3(k3) = img_rgb(y,x,2);
      histb3(k3) = img_rgb(y,x,3);
      k3 = k3 + 1;
   end
   if lbls(j) == 4
      [x,y] = map_index_to_pos(j);
      histr4(k4) = img_rgb(y,x,1);
      histg4(k4) = img_rgb(y,x,2);
      histb4(k4) = img_rgb(y,x,3);
      k4 = k4 + 1;
   end
end

% bins
edges = zeros(256,1);
for k = 1 : 256;
   edges(k) = k-1;
end

% one normalised vector

% obj2
histr2 = histc(histr2,edges)';
histr2 = histr2 / obj2_sz;
```

```
histg2 = histc(histg2,edges)';
histg2 = histg2 / obj2_sz;
histb2 = histc(histb2,edges)';
histb2 = histb2 / obj2_sz;

histv2 = [histr2' histg2' histb2'];
histv2 = histv2 / 3;

% obj3
histr3 = histc(histr3,edges)';
histr3 = histr3 / obj3_sz;
histg3 = histc(histg3,edges)';
histg3 = histg3 / obj3_sz;
histb3 = histc(histb3,edges)';
histb3 = histb3 / obj3_sz;

histv3 = [histr3' histg3' histb3'];
histv3 = histv3 / 3;

% obj4
histr4 = histc(histr4,edges)';
histr4 = histr4 / obj4_sz;
histg4 = histc(histg4,edges)';
histg4 = histg4 / obj4_sz;
histb4 = histc(histb4,edges)';
histb4 = histb4 / obj4_sz;

histv4 = [histr4' histg4' histb4'];
histv4 = histv4 / 3;

% if first image, save histograms
if i == 1
   A = [cobj2; cobj3; cobj4];
   base_hist_2 = histv2;
   base_hist_3 = histv3;
   base_hist_4 = histv4;
   base_obj_pts = obj1;
   continue;
end

% find out which object is which (2,3,4 from first image),
% put them in the same order
```

```
% obj2 from first image
bd2 = bhattacharyya(base_hist_2, histv2);
bd3 = bhattacharyya(base_hist_2, histv3);
bd4 = bhattacharyya(base_hist_2, histv4);
if bd2 < bd3
    if bd2 < bd4
        B(1,:) = cobj2;
    else
        B(1,:) = cobj4;
    end
else
    if bd3 < bd4
        B(1,:) = cobj3;
    else
        B(1,:) = cobj4;
    end
end

% obj3 from first image
bd2 = bhattacharyya(base_hist_3, histv2);
bd3 = bhattacharyya(base_hist_3, histv3);
bd4 = bhattacharyya(base_hist_3, histv4);
if bd2 < bd3
    if bd2 < bd4
        B(2,:) = cobj2;
    else
        B(2,:) = cobj4;
    end
else
    if bd3 < bd4
        B(2,:) = cobj3;
    else
        B(2,:) = cobj4;
    end
end

% obj4 from first image
bd2 = bhattacharyya(base_hist_4, histv2);
bd3 = bhattacharyya(base_hist_4, histv3);
bd4 = bhattacharyya(base_hist_4, histv4);
if bd2 < bd3
    if bd2 < bd4
```

```
            B(3,:) = cobj2;
        else
            B(3,:) = cobj4;
        end
    else
        if bd3 < bd4
            B(3,:) = cobj3;
        else
            B(3,:) = cobj4;
        end
    end

    % rotate and translate
    [R,t] = rigid_transform_3D(B,A);
    transf_obj1 = R * obj1' + repmat(t, 1, size(obj1,1));
    transf_obj1 = transf_obj1';

    % add to the base obj points
    base_obj_pts = union(base_obj_pts, transf_obj1, 'rows');
end

eps = 0.01;
eps_plane = 0.01;
start_num_pts = 100;

aux_obj_pts = base_obj_pts;

% base_obj_pts --> tmp_plane_pts
num_planes = 0;
planes = zeros(4,9);

figure;
hold on;

while num_planes < 9

    sz = size(aux_obj_pts,1);
    % make sure we have enough pts for a plane
    if sz < 1000
        break;
    end
```

```
tmp_plane_pts = zeros(sz,3);

% get random point from the plane
pnt_index = floor(rand(1,1) * sz);
if pnt_index < 1
   pnt_index = 1;
end
pnt = aux_obj_pts(pnt_index,:);

% add it to plane
tmp_plane_pts(1,:) = pnt;
aux_obj_pts(pnt_index,:) = [];
sz = sz - 1;

% get 9 other pts close by
% get all the points within eps distance
num_pts = 1;
i = 1;
while true
   if i > sz
      break;
   end
   dist = norm(aux_obj_pts(i,:) - pnt);
   if dist < eps
      num_pts = num_pts + 1;
      tmp_plane_pts(num_pts,:) = aux_obj_pts(i,:);
      aux_obj_pts(i,:) = [];
      sz = sz - 1;
      continue;
   end
   i = i + 1;
end

% get rid of noise
if num_pts < 100
   continue;
end

first = true;
first_r = 0;
tmp_plane = zeros(4,1);
while true
```

```
added = 0;

% try to fit a plane
[tmp_plane, r] = fitplane(tmp_plane_pts(1:num_pts,:));
if first == true
   first_r = r;
   first = false;
end
if r >= 0.1
   % just noise
   break;
end

% add all the pts within eps_plane and eps to other pts
for i = 1:size(aux_obj_pts,1)
   pnt = aux_obj_pts(i,:);
   dist_plane = abs([pnt 1]*tmp_plane);
   if dist_plane < eps_plane
      for k = num_pts:-1:1
         dist = norm(tmp_plane_pts(k,:) - pnt);
         if dist < eps
            added = added + 1;
            num_pts = num_pts + 1;
            tmp_plane_pts(num_pts,:) = pnt;
            aux_obj_pts(i,:) = [-10,-10,-10];
            break;
         end
      end
   end
end

% get rid of the -10 values from base_obj_pts
cnt = 0;
tmp_bop = zeros(sz - num_pts,3);
for i = 1:size(aux_obj_pts,1)
   if aux_obj_pts(i,:) == [-10,-10,-10]
      continue;
   end
   cnt = cnt + 1;
   tmp_bop(cnt,:) = aux_obj_pts(i,:);
end
```

```
        aux_obj_pts = tmp_bop;
        % if we got less than 50 new points, no point to go on
        if added < 50
            break;
        end
    end

    % if we got less than 5k pts, not a plane
    if num_pts < 1000
        continue;
    end

    num_planes = num_planes + 1;
    planes(:,num_planes) = tmp_plane;

    % compute z value (plane) for each point (or just 100 of them)
    plane_pts = zeros(size(tmp_plane_pts,1),3);
    for k = 1:size(tmp_plane_pts,1)
        x = tmp_plane_pts(k,1);
        y = tmp_plane_pts(k,2);
        plane_pts(k,1) = x;
        plane_pts(k,2) = y;
            plane_pts(k,3) = (-planes(4,num_planes) - planes(1,num_planes) * x -
planes(2,num_planes) * y)/planes(3,num_planes);
    end

    % plot plane with a diff color
    clf;
    plot3(aux_obj_pts(:,1),aux_obj_pts(:,2),aux_obj_pts(:,3),'.b');
    hold on;
    plot3(plane_pts(:,1),plane_pts(:,2),plane_pts(:,3),'.', 'Color', cmap(num_planes,:));
    while true
        k = waitforbuttonpress;
        if k ~= 0
            break;
        end
    end
end
```

Mapping index to position (width and height) function:

```
function [i,j] = map_index_to_pos(index)
  j = mod(index,480);
  i = (index - j)/480 + 1;
```

Map position (width and height) to index in one large array function:

```
function [index] = map_pos_to_index(i,j)
  index = 480 * (j - 1) + i;
```

Check if elem is a member of array (array is multidimensional):

```
function [lia,locb] = my_ismember(elem, array)
    sz = size(array,1);
    d = size(array,2);

    lia = 0;
    locb = 0;
    for i = 1:sz
        dif = false;
        for j = 1:d
            if elem(j) ~= array(i,j)
                dif = true;
                break;
            end
        end
        if dif == false
            lia = 1;
            locb = i;
            break;
        end
    end
end
```