Advanced Vision
Assignment 3

Thorvaldur Helgason (s1237131)
Hywel Dunn-Davies (s1268263)

March 21, 2013

1 Introduction

This assignment involved analyzing 21 consecutive image frames captured with a Kinect sensor. These frames contained both RGB pixel intensity values and XYZ range values. The images showed many different angles of a cabinet containing an open book on top. Our goal was the following:

(i) Extract the three large visible planes of the filing cabinet and compute their plane equations

(ii) Extract the set of points belonging to the book

(iii) Compute the translation and rotation of the three planes between frames to register all views together

(iv) Fuse all views of the book together using the translation and rotation computed in the previous step

At last we evaluated our approach by creating a 3D fused view of the book, computing the determinant of the covariance matrix between the same point in all frames, and computed the average and standard deviation of surface normal angles which have been corrected in terms of rotation.

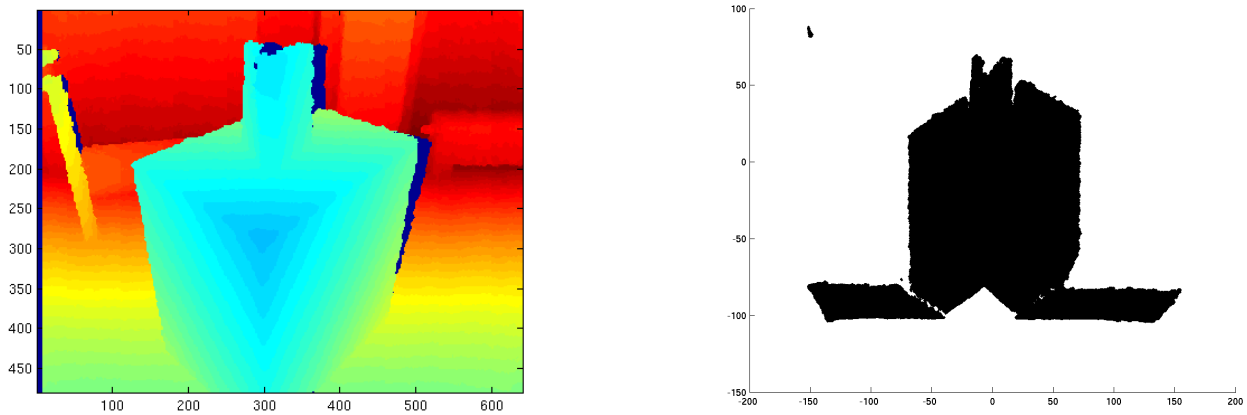
2 Methods

2.1 Pre-processing

Before extracting the planes, we performed the following pre-processing steps:

- We transformed the depth component of the supplied XYZ data from the Kinect sensor into a 3D point cloud representation similar to the representation used for system 6 from the lectures, to make it easier to perform region growing to extract the planes using the lecture example code
- We cleaned the data by:
  - removing all points from the data in which the Z co-ordinate (representing the depth of the point) was higher than a given threshold (we found that 1400 was an appropriate choice), as these were not relevant to the problem that we were trying to solve, since the only things we are interested are the cabinet and the book, which are in the foreground. and
  - removing all points from the data in which the XYZ co-ordinates were (0,0,0), as these points were found to interfere with the plane fitting
- We normalized the data by dividing all of the X,Y, and Z co-ordinate values by 5. This was found to have a favorable effect on the speed and accuracy of the region growing algorithm

Figure 1 illustrates the effect of our transformations. The raw XYZ depth data for frame 17 (which we used as our foundation frame) is shown in the picture on the left of the figure, and the transformed data is shown on the right. The matlab code to perform the pre-processing is shown in section 4.1 of this report.



(a) (b)

Figure 1: The raw depth image from the XYZ data (a) and the same data after the pre-processing steps had been carried out (b) for our foundation frame.

2.2 Plane Extraction

To perform the plane extraction, we used the region growing algorithm described in the lectures in the context of system 6. In general, we concentrated on tailoring the algorithm so that it worked quickly and effectively, rather than re-implementing functionality that was already present in the example code. Our main innovation was to use our knowledge of the problem to optimize the process of patch selection. Aside from that change, and the pre-processing described in the previous section, our main modifications to the example code consisted of tuning parameters of the region growing algorithm so that it worked well with the supplied data. We describe both the patch selection and region growing here. The matlab code to perform the plane extraction is shown in section 4.2 of this report.

2.2.1 Patch Selection

The approach that the example code takes to patch selection can be summarized by the following three step algorithm (implemented in the function *select_patch.m* in the example code:

1. Pick a seed point at random from the point cloud
2. Create a patch around that point by looping through each point and selecting the points whose euclidean distance from the seed point is below a certain threshold. If less than a certain number of points have been added to the patch when this process is complete, return to step 1
3. Fit a plane to the patch using the least squares approach implemented in *fitplane.m* in the example code. If the residual least squares fitting error is higher than a certain threshold, throw out the current patch and return to step 1 (as the patch probably includes points from more than one plane

An approach such as this one, which relies on seed points are chosen at random, has a number of drawbacks in the context of the current problem. Two major drawbacks are the following:

1. *It is slow.* The seed points may be on or close to edges, or in places where not enough neighbors exist to form a plane, quite often, meaning that the algorithm has to loop through all of the data

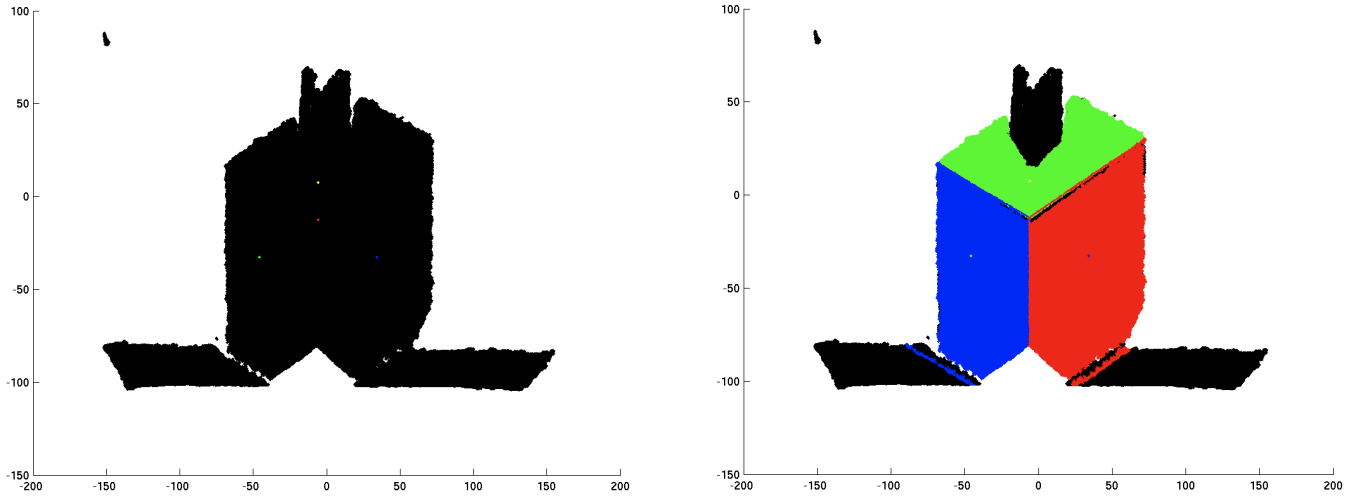
points multiple times before an appropriate patch is found. For an image with a large number of pixels, this takes a long time.

2. *It does not distinguish between planes.* The seed points may lie on any plane in the image, so the while the set of planes returned by the algorithm might contain the correct planes, the set is not ordered. As a result, it is necessary to perform the additional step of matching the planes to each other in the different images after the plane extraction has taken place

To overcome these drawbacks, we used a different approach, using our knowledge of the problem to approximate the position of appropriate seed points for the initial patches on each of the three planes. To do that we extracted the closest point in terms of Z in the range data, which was the corner of the cabinet, and then extracted three seed points with the following equations:

$$\begin{aligned} \text{rightPoint} &= (\text{cornerPoint(x)} + 40,\ \text{cornerPoint(y)} - 20,\ \text{cornerPoint(z)} + 30) \\ \text{leftPoint} &= (\text{cornerPoint(x)} - 40,\ \text{cornerPoint(y)} - 20,\ \text{cornerPoint(z)} + 30) \\ \text{topPoint} &= (\text{cornerPoint(x)},\ \text{cornerPoint(y)} + 20,\ \text{cornerPoint(z)} + 20) \end{aligned}$$

Here the rightPoint is our approximation for the seed point for the initial patch for the right side of the cabinet, leftPoint for the left side, and topPoint for the top half (see Figure 2a). By using these approximation points we found three seed points in the point cloud that were closest to the approximations, and each of them respectively turned out to be on the planes we wanted to extract. This meant that we could use them as out initial patches for region growing on each plane and not have to randomly select seed points for our initial patches. The patches we acquired using these points were computed using a distance tolerance of 5.



(a) (b)

Figure 2: Reduced point cloud data with the corner point (red) and the three approximation points for the three planes (a) and region growing with initialization on the closest patch to the points (b).

2.2.2 Region Growing

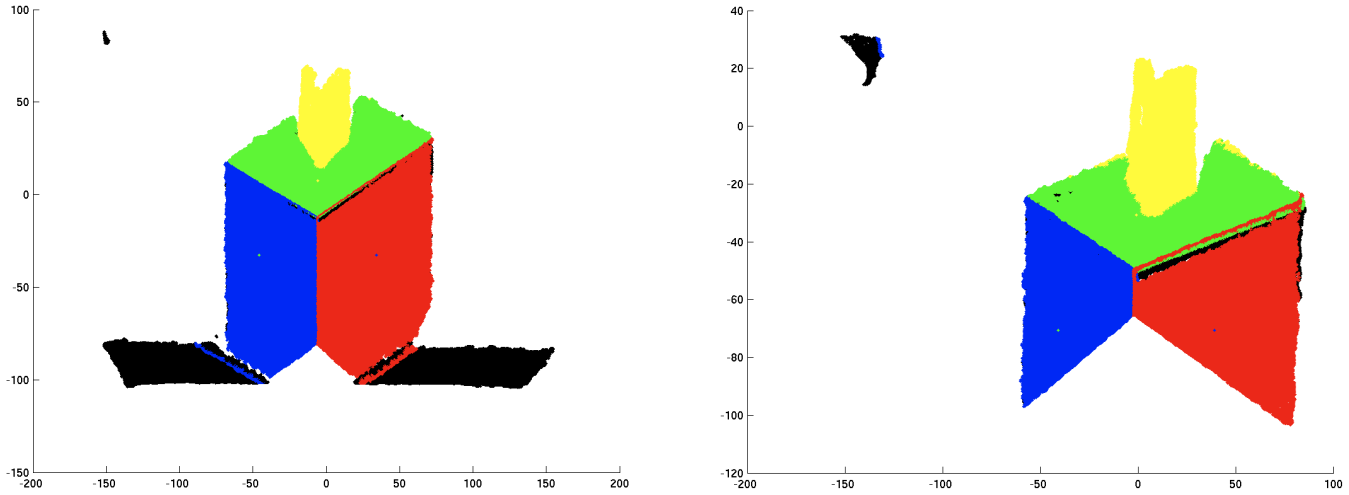
To perform the region growing itself, we used the example code from system 6 in the lectures, with the following settings for the relevant parameters:

- In *getallpoints.m*, we used a plane tolerance of 1 and point distance tolerance of 150. By having a low plane tolerance we acquired more accurate regions and with such a large point distance tolerance the growing only needed about 5 iterations

- In the main region growing loop, we specified that when the number of new points in a region exceeded the number of points already in the region plus 50 then a plane is (re)fitted to the region. The region growing stops when the least square fitting error of that plane is bigger than 30% of the number of new points.

2.3 Book Extraction

To extract the set of points that represented the book we used couple of thresholds in order to capture the book and reduce noise. First, we used the point cloud that can be seen in Figure 2a and subtracted the points belonging to the three planes. The, to eliminate other data points, such as outside the cabinet (see Figure 2b), we subtracted all points that were to the left and right of the topmost plane, and ones that were below the corner point. Finally, to eliminate noise on the topmost plane, we extracted only the points that had the distance ≥ 1 from the top plane. The result for our foundation frame, where the book extraction worked well, can be seen in Figure 3a.



(a) (b)

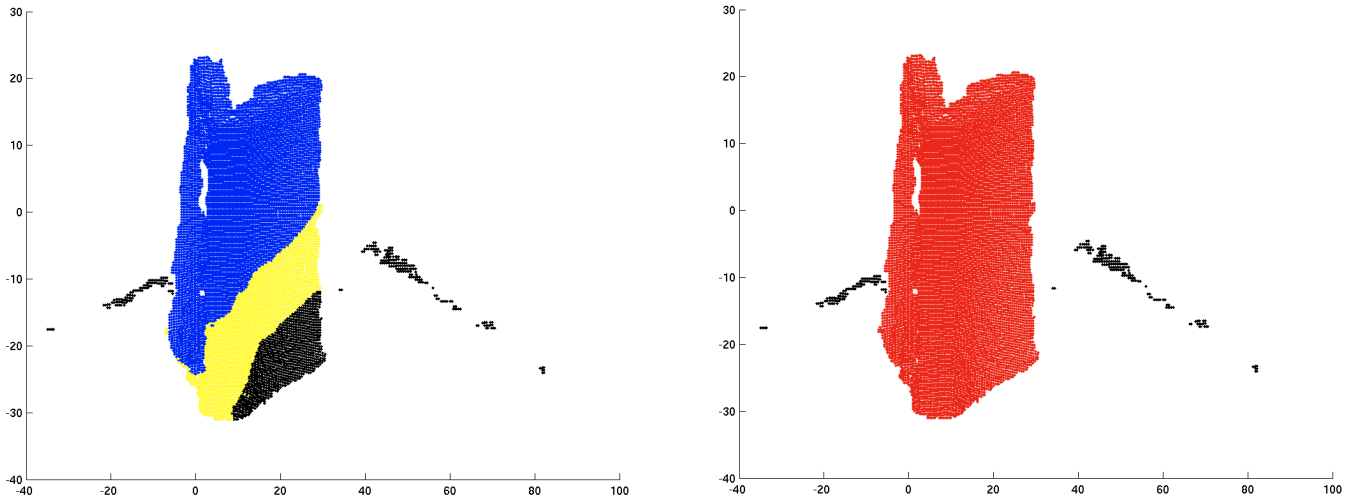
Figure 3: Book points for the foundation frame 17 (a) and frame 2 (b) displayed in yellow.

An example of a frame for which the book extraction worked less well (frame 2) is shown in figure 3b. In this figure it is apparent that there are a number of yellow points on the top plane that are not actually part of the book.

To fix this, we created a recursive algorithm that used a region growing approach to distinguish true book points from noise, based on the following two observations:

- The true book points form a coherent group, in which each point is close to some other point in the book, while the noise points are relatively spread out
- The highest point in the image is always a true book point, as the noise points are seen close to the cabinet

The algorithm defined a frontier set of points, initialized as a singleton set comprising the highest point. On each iteration the algorithm generated a new frontier set, comprising points that were within a set euclidean distance (12 in our case) from some member of the previous frontier set (see Figure 4a). The algorithm terminates when the new frontier set is empty. The resulting set of true book points is the union of all of the frontier sets, and is built incrementally as the algorithm runs. The result of running this algorithm on the noisy set of book points in frame 2 is shown in figure 4b, with the true book points shown in red, and the noise points in black.



(a) (b)

Figure 4: Extracted book points for frame 2 after an additional recursive algorithm for region growing has been applied to distinguish true book points from noise. In (a) blue points are region points and yellow are the frontier and in (b) the red points are the true book points

The matlab code to perform the book extraction is shown in section 4.3 of this report.

2.4 Plane Registration

As mentioned earlier in this report, for registering views we decided to use frame number 17 as our foundation frame. We used this frame for computing the rotation and translation between that frame and all the others. The plane registration step in the assignment consists of three stages:

1. Match large planes in the different frames to each other to identify which plane in each frame corresponds to each plane in the foundation frame
2. Estimate the rotation required to align each frame with the reference frame
3. Estimate the translation required to align each frame with the reference frame

Here we discuss each of these stages in turn. The matlab code to perform the plane registration is shown in section 4.4 of this report.

2.4.1 Plane Matching

Thanks to our approach to selecting the seed points for region growing, the region growing algorithm output the three visible planes of the cabinet in the same order for each frame. As a result plane matching was trivial for us, as we knew that for each frame the first plane was the right side of the cabinet, the second plane was the top, and the third plane was the left side of the cabinet.

2.4.2 Rotation Estimation

To estimate the rotation matrices that would bring each frame into the same orientation as the foundation frame, we chose one of the large planes and estimated the angle of rotation based on the surface normal of that plane in each frame and the surface normal of that plane in the foundation frame. We performed the following steps:

1. Calculate the surface normals for the two planes being compared
2. Take the cross product of the two surface normals to give a vector that is orthogonal to both, and provides an appropriate axis of rotation

3. Calculate the angle of rotation about that axis in radians

4. Use this angle to calculate the rotation matrix

2.4.3 Translation Estimation

The basic idea behind the translation estimation was that if the co-ordinates of a chosen reference point in each point cloud were subtracted from the co-ordinates of every point in the point cloud, this would result in an image centered at the reference point. Given that the reference point is at the same location in every image, doing this to each image should result in each image being aligned.

We experimented with a number of approaches to estimating the appropriate translation between the frames, based on a number of different possible reference points. These included the closest point in the image, the average point from one of the points on the plane, and the average of the rotated book points. While using the closest point would have given us the best fit in the evaluation, it proved to be less effective than using the average of the rotated book points in providing an accurate 3D representation of the book during the book point fusion stage. We therefore decided to use the average of the rotated book points in our final solution.

2.5 Book Fusion

The final task was to apply the transformations described above to each image, giving a final 3D point cloud corresponding to the fused book points. To do this we performed the following steps:

1. Apply the correct rotation matrix to each book point cloud by multiplying the transpose of the book point cloud (in which each (x,y,z) point is represented by a column) by the rotation matrix

2. Translate each book point cloud by subtracting the co-ordinates of its mean point from the co-ordinates of every point

3. Combine the individual point clouds to create the overall image, after multiplying each by the scaling factor so that the points are at the same scale as the original images

The matlab code to perform the book point fusion is shown in section 4.5 of this report.

3 Evaluation

Three methods of evaluation were requested for this assignment. These were:

1. Produce an image of the book after each dataset is merged to produce 21 incrementally denser views of the book

2. Compute the determinant of the covariance matrix for the corner of the cabinet closest to the viewer in each translated image to assess how closely registered the point sets are

3. Compute the angle between the surface normal of all of the upward facing planes from the 21 images and that of the foundation image and report the mean and standard deviation of the angles

The matlab code to perform the evaluation is shown in section 4.6 of this report. The results are described in the following sections.

3.1 Images of the book after each dataset is merged

We produced images of the book after each dataset was merged. These images are shown in figures 5 and 6.

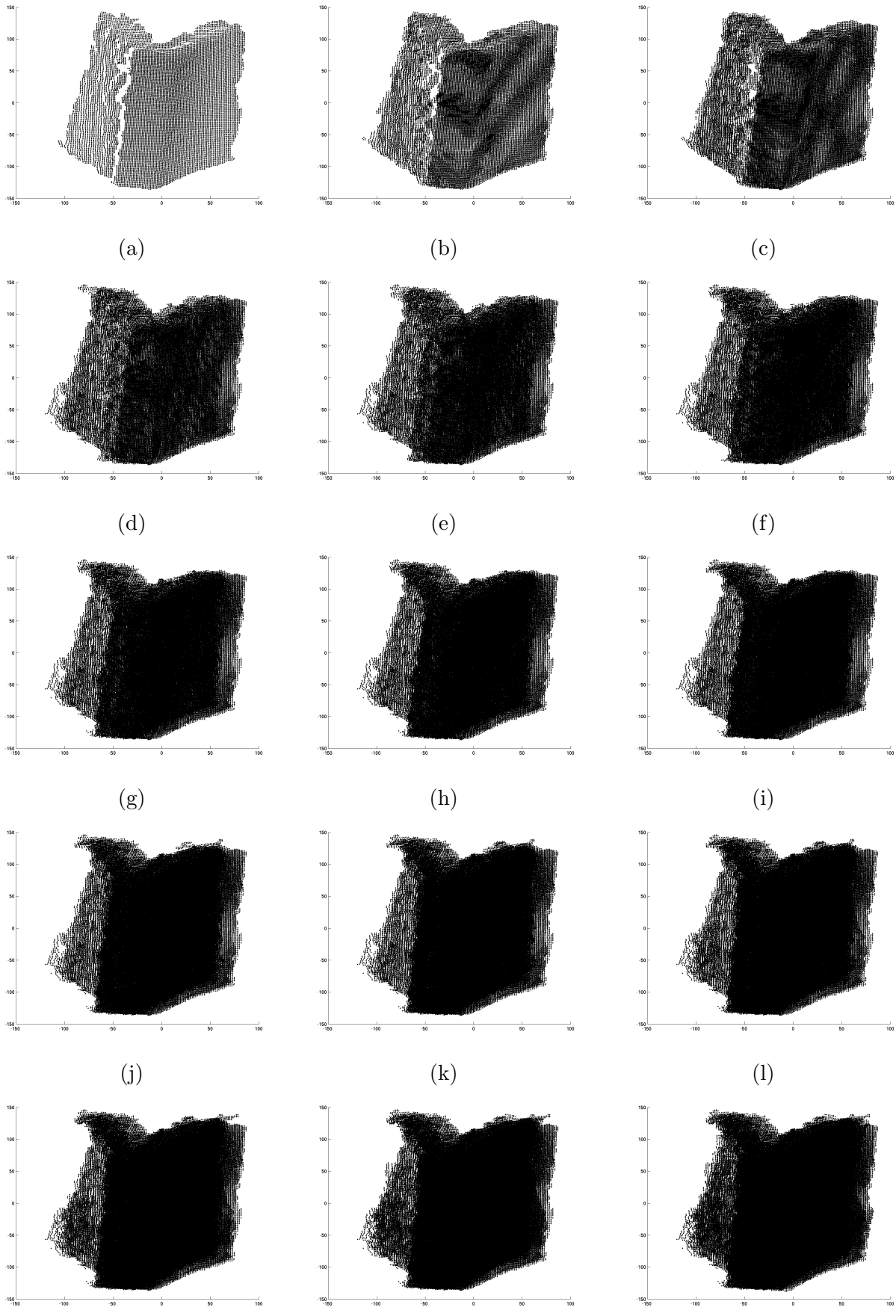
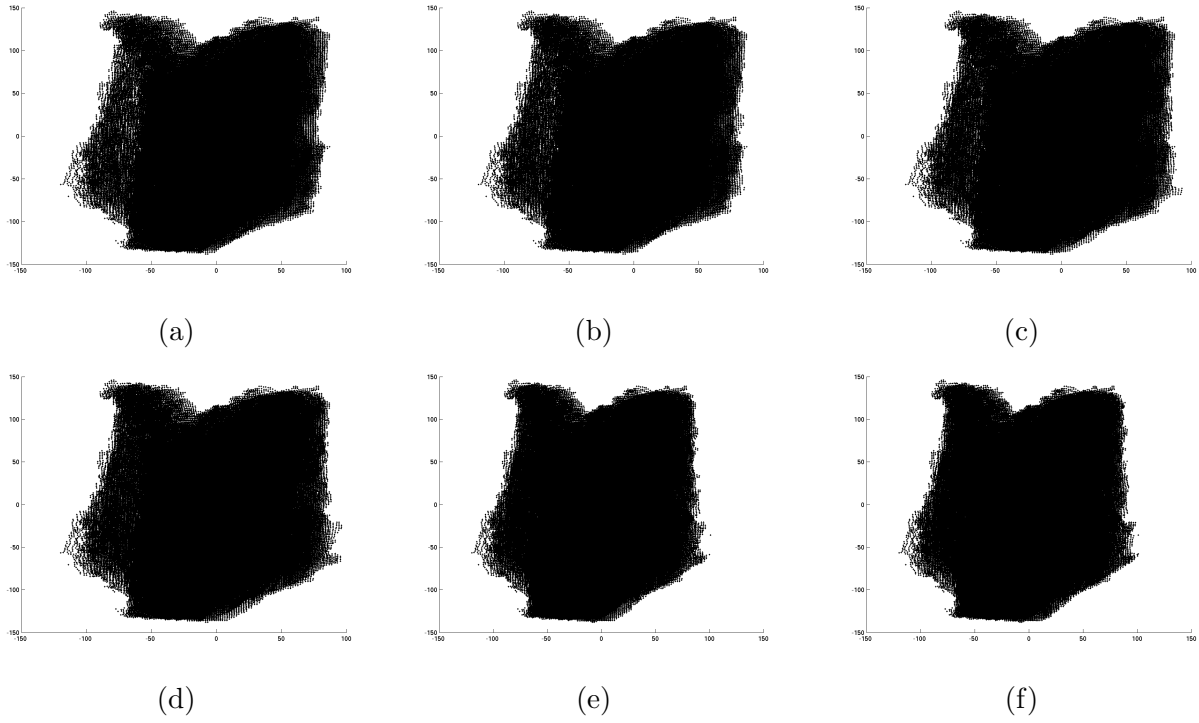


Figure 5: Images of the book points progressively merging the first 15 frames. Image (a) is just frame 1, image (b) is frames 1 and 2, and so on. Image (o) is the result of merging images 1 to 15.



(a) (b) (c)

(d) (e) (f)

Figure 6: Continuation of the images of the book points progressively merging the last 6 frames to the existing point cloud. Image (a) is the result of merging frames 1 to 16, image (b) is frames 1 to 17, and so on. Image (f) is the result of merging all 21 images.

3.2 The determinant of the covariance matrix for the position of the corner of the cabinet closest to the viewer

We computed the covariance matrix and took its determinant. The value for the determinant was $5.9359e-202$.

3.3 The mean and standard deviation of the angles between the surface normals of all of the upward facing planes

We computed the angles between the surface normals of the upwards facing plane of our foundation frame and the others. The average angle between the vectors was 0.0756 and the standard deviation was 0.0599.

4 Code

4.1 Pre-Processing

```
% preProcessData: function to pre-process the raw range data from a frame,
% creating a 3D point cloud that can be used with the region growing algorithm

% Input:
% frame: a single frame from the input data

% Output:
% pointCloud: a preprocessed point cloud representing the frame

function pointCloud = preProcessData(frame)
```

```
    % Convert the frame into a 3D point cloud
    xyzFrame = frame.XYZ(:,:,:);
    xFrame = xyzFrame(:,:,1);
    xFrame = xFrame(:);
    yFrame = xyzFrame(:,:,2);
    yFrame = yFrame(:);
    zFrame = xyzFrame(:,:,3);
    zFrame = zFrame(:);
    pointCloud = [xFrame yFrame zFrame];

    % Clean and normalise the point cloud
    % - Remove all points further away than the depth threshold
    % - Remove all points whose co-ordinates are (0,0,0)
    % - Divide the points by 5
    depthThreshold = 1400;
    backgroundRowsToIgnore = find(pointCloud(:,3)>depthThreshold);
    zeroRowsToIgnore = find(abs(pointCloud(:,1)) + ...
        abs(pointCloud(:,2)) + abs(pointCloud(:,3))==0);
    index = true(1,size(pointCloud,1));
    index(backgroundRowsToIgnore') = false;
    index(zeroRowsToIgnore') = false;
    pointCloud = pointCloud(index,:)./5;

end
```

4.2 Plane Extraction

4.2.1 planeExtraction.m

```
% planeExtraction: code to find the largest planes for an image
% adapted from doall.m in the system 6 example code

% Input:
% pointCloud: the point cloud for the image

% Output:
% pointCloud: the original point cloud
% planeEq: the plane equation for each plane
% planePoints: the points in each plane
% closestPoints: the closest points in the image

function [pointCloud planeEq planePoints closestPoints] = ...
    planeExtraction(pointCloud)

planePoints = {};       % Points on each individual plane
bookPoints = {};        % Points on the book
closestPoints = zeros(3); % Closest point in each image, i.e. the corner

plot3(pointCloud(:,1),pointCloud(:,2),pointCloud(:,3),'k.')

closestPoints = getClosestPoint(pointCloud);
plot3(closestPoints(1),closestPoints(2),3000,'r.')

[NPts,W] = size(pointCloud);
```

```
planeEq = zeros(3,4);

remaining = pointCloud;

rightPoint = [closestPoints(1)+40 ...
    closestPoints(2)-20 closestPoints(3)+30];
leftPoint = [closestPoints(1)-40 ...
    closestPoints(2)-20 closestPoints(3)+30];
topPoint = [closestPoints(1) ...
    closestPoints(2)+20 closestPoints(3)+20];

patchPoints = [rightPoint; leftPoint; topPoint]
plot3(rightPoint(:,1),rightPoint(:,2),3000,'b.')
plot3(leftPoint(:,1),leftPoint(:,2),3000,'g.')
plot3(topPoint(:,1),topPoint(:,2),3000,'y.')

for i = 1 : 3
    planePatch = getClosestPointToPoint(pointCloud,patchPoints(i,:))

    % select a random small surface patch from the remaining points
    [oldlist,plane] = select_patches(remaining,planePatch);

    plot3(oldlist(:,1),oldlist(:,2),oldlist(:,3),'y.')

    % grow patch

    growthCycles=0;

    stillgrowing = 1;

    while stillgrowing

        growthCycles = growthCycles+1

        stillgrowing = 0; %-- until we find a bad fit we keep growing

        [newlist,remaining] = ...
            getallpoints(plane,oldlist,remaining,NPts);

        [NewL,W] = size(newlist);
        [OldL,W] = size(oldlist);

        if i == 1
            plot3(newlist(:,1),newlist(:,2),newlist(:,3),'r.')
            save1=newlist;
        elseif i==2
            plot3(newlist(:,1),newlist(:,2),newlist(:,3),'b.')
            save2=newlist;
        else
            plot3(newlist(:,1),newlist(:,2),newlist(:,3),'g.')
            save3=newlist;
        end
```

```
        pause(0.01)

        if NewL > OldL + 50
            ['refitting plane']

            % refit plane
            [newplane,fit] = fitplane(newlist);
            planeEq(i,:) = newplane';
            fitThreshold = 0.3;

            if fit > fitThreshold*NewL    % bad fit - stop growing
                break
            end

            stillgrowing = 1;
            oldlist = newlist;
            plane = newplane;
        end
    end

    ['***************Segmentation Completed']

    planePoints{i} = oldlist;
end
```

4.2.2 getClosestPoint.m

```
% gets the closest point in the image
function meanClosestPoint = getClosestPoint(pointCloud)
minDepth = min(pointCloud(:,3))
minDepthIndex = find(pointCloud(:,3)==minDepth)
closestPoints = pointCloud(minDepthIndex,:)
meanClosestPoint = mean(closestPoints)
```

4.2.3 getClosestPointToPoint.m

```
% gets the closest point in the point cloud to the point given as input
function closestPoint = getClosestPointToPoint(pointCloud, point)
distancesFromPoint = [];
l = length(pointCloud(:,1));
dist = abs(pointCloud(:,1)-point(1)) + abs(pointCloud(:,2)-point(2)) ...
    + abs(pointCloud(:,3)-point(3));
closestIndex = find(dist==min(dist));
closestPoint = pointCloud(closestIndex(1),:);
```

4.2.4 Other

Here we also used the following code from the course:

- *select_patches.m* - Slightly modified version of *select_patch.m* where the exhaustive search for points is removed and instead only the three approximated plane patches are used.
- *getallpoints.m* - Tolerance variables modified.
- *fitplane.m* - Not modified.

4.3 Book Extraction

4.3.1 bookExtraction.m

```
% bookExtraction: function to extract the book points from
% the point clouds

% Input: The point clouds, plane equations, plane points, and
% closest points from each image

% Output: A cell array containing the cleaned book points for
% each frame

function cleanedBookPoints = ...
        bookExtraction(pointCloud,planeEq,planePoints,closestPoints)

% subtract points in R that are on planes
allPoints = [planePoints{1};planePoints{2};planePoints{3}];
[~,planeRowsToIgnore] = ismember(allPoints,pointCloud,'rows');
planeRowsToIgnore(find(planeRowsToIgnore > 0));

planeIndex = true(1,size(pointCloud,1));
planeIndex(planeRowsToIgnore') = false;
reducedPointCloud = pointCloud(planeIndex,:,:);

% subtract points too low / to left / to right of planes
minXPoint = min(planePoints{3}(:,1));
maxXPoint = max(planePoints{3}(:,1));
bookPoints = reducedPointCloud(find( ...
    (reducedPointCloud(:,1) > minXPoint) & ...
    (reducedPointCloud(:,1) < maxXPoint) & ...
    (reducedPointCloud(:,2) > closestPoints(2))),:);

% Get points above top plane
[abovePoints aboveIndices] = ...
    getPointsAbovePlane(bookPoints,planeEq(3,:));
bookPoints = bookPoints(aboveIndices,:);
plot3(bookPoints(:,1), bookPoints(:,2),bookPoints(:,3),'y.');
pause(0.01);

% clean up the bookpoints
cleanedBookPoints = getCleanedBookPoints(bookPoints);
```

4.3.2 getPointsAbovePlane.m

```
% getPointsAbovePlane: function to find the points within
% a certain distance of the specified plane

function [points indices] = getPointsAbovePlane(R,plane)
R = [R ones(size(R,1),1)];
distanceFromPlane = R*plane';
distanceTolerance = -1;
indices = find(distanceFromPlane <= distanceTolerance);
points = R(indices);
```

4.3.3 getCleanedBookPoints.m

```
% getCleanedBookPoints: function to find the true book points

% Input:
% bookPoints: the point cloud of potential book points with noise

% Output:
% cleanedBookPoints: the cleaned point cloud

function cleanedBookPoints = getCleanedBookPoints(bookPoints)

cleanedBookPoints = {};

% step through each book point cloud filtering out non-book points
for i = 1:21

fig = figure(1);
clf
hold on
plot3(bookPoints{i}(:,1),bookPoints{i}(:,2),bookPoints{i}(:,3),'k.')

% Calculate the highest book point
% this will be the seed point
% the idea is that we're resonably sure that this will be a true book point
highestBookPoint = ...
        bookPoints{i}(find(bookPoints{i}(:,2)==max(bookPoints{i}(:,2))),:);

% Find the true book points
regionGrowingInput{1} = highestBookPoint;
regionGrowingInput{2} = [];
regionGrowingInput{3} = bookPoints{i};
regionGrowingInput{4} = 1;

cleaningOutput = growCleanedBookPoints(regionGrowingInput);
cleanedBookPoints{i} = cleaningOutput{2};

% plot the results
plot3(cleanedBookPoints{i}(:,1),...
        cleanedBookPoints{i}(:,2),cleanedBookPoints{i}(:,3),'r.')
pause(1);

end

end
```

4.3.4 growCleanedBookPoints.m

```
% growCleanedBookPoints: recursive function to find true book points

% Inputs:
% progress, a cell array containing:
% frontier points, cleaned points, remaining points and should continue
```

```
function output = growCleanedBookPoints(progress)

frontierPoints = progress{1};
cleanedPoints = progress{2};
remainingPoints = progress{3};
shouldContinue = progress{4};

X = frontierPoints;
Y = remainingPoints;

% next 5 lines are from Piotr Dollar's toolbox
% http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html
% The code calculates the pairwise squared euclidean distance between the rows
m = size(X,1);
n = size(Y,1);
XX = sum(X.*X,2);
YY = sum(Y'.*Y',1);
D = XX(:,ones(1,n)) + YY(ones(1,m),:) - 2*X*Y';

% calculate the euclidean distances
euclideanDistances = sqrt(D);

% create the list of new frontier points
distanceThreshold = 12;
newFrontierPoints = [];
for r = 1:size(euclideanDistances,1) % for each frontier point

distantPoints = find(euclideanDistances(r,:)>distanceThreshold);
index = true(1,size(remainingPoints,1));
index(distantPoints') = false;

if(isempty(newFrontierPoints))
        newFrontierPoints = remainingPoints(index,:);
else
        newFrontierPoints = ...
                union(newFrontierPoints,remainingPoints(index,:),'rows');
end

end

% update the variables
progress{1} = newFrontierPoints;
progress{2} = [cleanedPoints;frontierPoints];
progress{3} = setdiff(remainingPoints,newFrontierPoints,'rows');
progress{4} = 1-isempty(newFrontierPoints);

fig = figure(1);
plot3(newFrontierPoints(:,1), ...
        newFrontierPoints(:,2),newFrontierPoints(:,3),'y.');
plot3(frontierPoints(:,1), frontierPoints(:,2),frontierPoints(:,3),'b.');

pause(0.1);
```

```
if(progress{4}==1)

        progress = growCleanedBookPoints(progress);

end

output = progress;

end
```

4.4 Plane Registration

4.4.1 performPlaneRegistration.m

```
% performPlaneRegistration: function to transform the book points in
% the different images so that they are aligned

% Inputs:
% bookPoints: a cell array containing a point cloud for each book
% planes: a cell array containing the plane equations from each image

function transformedBookPoints = performPlaneRegistration(bookPoints,planes)

% get the rotation matrices
[angles rotationMatrices] = findRotationMatrices(planes);

% apply the rotation to each book point cloud
rotatedBookPoints = {};
for i = 1:21
        rotatedBookPoint = (rotationMatrices{i}*cleanedBookPoints{i}')';
        rotatedBookPoints{i} = rotatedBookPoint;
end

% find the mean bookpoint in each frame
meanBookPoints = {};
for i = 1:21
        meanBookPoint = mean(rotatedBookPoints{i});
        meanBookPoints{i} = meanBookPoint;
end

% apply the translation to each rotated book point cloud
transformedBookPoints = {};
for i = 1:21
        transformedBookPoints{i} = ...
                rotatedBookPoints{i}- ...
                repmat(meanBookPoints{i},size(rotatedBookPoints{i},1),1);
end

end
```

4.4.2 findRotationMatrices.m

```
% findRotationMatrices: find the rotation matrices for the planes
```

```
% Inputs:
% planes: the plane list

% Output:
% angles: angles between the surface normals
% rotationMatrices: rotation matrices for the planes

function [angles rotationMatrices] = findRotationMatrices(planes)

% the top plane from image number 17 is our reference plane
p2 = planes(1,:,17);

% get the angle and rotation matrix for each frame
angles = {};
rotationMatrices = {};

for i = 1:21

p1 = planes(1,:,i);

% calculate normal vectors n1 and n2 for planes p1 and p2
n1 = [p1(1) p1(2) p1(3)]';
n2 = [p2(1) p2(2) p2(3)]';

% the axis of rotation is the cross product of the two vectors
% (cross product is orthogonal to both vectors)
axis = cross(n1,n2);

% the normals are normalised, so the angle in radians is
angle = acos(dot(n1,n2));

% then we define the rotation matrix using makehgtform
R = makehgtform('axisrotate',axis,angle);

% if it's the reference frame we just use the identity matrix for the rotation,
% otherwise use R
if (i==17)
        rotationMatrices{i} = eye(3);
        angles{i} = 0;
else
        rotationMatrices{i} = R(1:3,1:3);
        angles{i} = angle;
end

end

end
```

4.4.3 getMeanBookPoints.m

```
% getMeanBookPoints: returns a point whose co-ordinates are
% the mean of the co-ordinates of all the book points in each image
```

```
% Input:
% bookPoints: a set of book points for each image

% Output:
% meanBookPoints: a cell array containing the mean book point
% for each image

function meanBookPoints = getMeanBookPoints(bookPoints)

        % find the mean bookpoint in each frame
        meanBookPoints = {};
        for i = 1:21
                meanBookPoint = mean(bookPoints{i});
                meanBookPoints{i} = meanBookPoint;
        end

end
```

4.5 Book Fusion

4.5.1 performBookFusion.m

```
% performBookFusion: function to perform book point fusion and plot the results

% Input:
% transformedBookPoints: a cell array containing the transformed book points
% produced by the plane registration algorithm

% Output:
% allBookPoints: a point cloud containing all of the book points,
% scaled appropriately


function allBookPoints = performBookFusion(transformedBookPoints)

% apply the scaling to each transformed book point cloud
scaledBookPoints = {};
for i = 1:21
        scaledBookPoints{i} = transformedBookPoints{i}.*5;
end

% plot the results
fig = figure(1);
clf
hold on

% put all the bookpoints together
allBookPoints = [];
for i = 1:21
        allBookPoints = [allBookPoints;scaledBookPoints{i}];
        plot3(allBookPoints(:,1),allBookPoints(:,2),allBookPoints(:,3),'k.')
        saveas(fig, ...
                strcat('../../Images/FusedBookPoints/3D/Book', ...
```

```
                int2str(i),'.png'));
end

end
```

4.6 Evaluation

4.6.1 performEvaluation.m

```
% performEvaluation: function to evaluate the algorithm

% Input:
% transformedBookPoints: the transformed but unscaled book points
% closestPoints: the closest point in each image

% Output:
% output: a cell array containing:
% the covariance matrix of the corner point positions
% the determinant of the covariance matrix
% the mean of the surface normal angles
% the standard deviation of the surface normal angles

function output = performEvaluation(transformedBookPoints,closestPoints)

% 1. plot the book points as the points from each image are added

% apply the scaling to each transformed book point cloud
scaledBookPoints = {};
for i = 1:21
        scaledBookPoints{i} = transformedBookPoints{i}.*5;
end

% plot the results
fig = figure(1);
clf
hold on

% put all the bookpoints together
allBookPoints = [];
for i = 1:21
        allBookPoints = [allBookPoints;scaledBookPoints{i}];
        plot3(allBookPoints(:,1),allBookPoints(:,2),allBookPoints(:,3),'k.')
        saveas(fig, ...
                strcat('../../Images/FusedBookPoints/3D/Book', ...
                int2str(i),'.png'));
end

% 2. compute the determinant of the covariance matrix

% get the mean of all of the book points for each image
meanBookPoints = getMeanBookPoints(bookPoints);

% go through each of the frames applying the mean book point translation
% to the closest point
```

```
translatedPoints = {};
for i = 1:21
        translatedPoints{i} = closestPoints(i,:)-meanBookPoints{i};
end

% subtract the closest point from the foundation image from the others
foundationPoint = translatedPoints{17};
subtractedPoints = [];
for i = [1:16 18:21] % skip the foundation image
        subtractedPoints = [subtractedPoints;scaledPoints{i}-foundationPoint];
end

% Compute the evaluation statistics
covarianceMatrix = cov(subtractedPoints')
determinantOfCovarianceMatrix = det(covarianceMatrix)

output = {};
output{1} = covarianceMatrix;
output{2} = determinantOfCovarianceMatrix;

% 3. compute the mean and standard deviation of the surface normal angles
[angles rotationMatrices] = findRotationMatrices(planes);
angles = [angles(1:16) angles(18:21)];
angleMean = sum(angles) / length(angles);
squaredMeanDiff = (angles-mean).^2;
stdDevAngle = sqrt(sum(squaredMeanDiff)/length(angles));

output{3} = angleMean;
output{4} = stdDevAngle;


end
```