3D Scene Reconstruction from Point Cloud Data

David Strömbäck
s1874170

Xingyu Jin
s1836694

1 Introduction

The emergence of low-cost RGB-D cameras has led to major developments and state-of-the-art results in the area of 3D scene reconstruction from RGB-D data [1]. 3D scene reconstruction has applications in a wide range of disciplines, such as in virtual reality content creation, constructing 3D models of historical sites, and for creating interactive visualisations of real-life scenes.

In this work we demonstrate the typical RGB-D reconstruction workflow on the problem of reconstructing a 3D model of a portion of the inside of an office from a set of 40 3D point clouds. An Intel RealSense depth sensor was used to capture a sequence of 40 consecutive 640×480 frames of point cloud data. Each point cloud has its own coordinate system, with the camera positioned at the origin. Each point in the point clouds has six associated values, corresponding to its RGB value and XYZ-location. The objective is to fuse all the frames into one coordinate system in order to create a complete 3D model of the back end of the office.

The report is structured in a sequential manner, with each section describing a stage of the reconstruction pipeline. We begin by outlining the preprocessing steps in Section 2, before describing the approach used to estimate the 3D transformations between frames in Section 3. Section 4 details how we fuse all the frames into one coordinate system, and Section 5 evaluates and discusses the results. MATLAB code for our implementation can be found in Appendix A.

2 Preprocessing

Before we can estimate the transformations between frames, it is important to remove noise and outliers from our point cloud data. Additionally, we must identify the relevant points to use for the transformation and fusing stages. In this section we describe the methods used for preprocessing and discuss their effectiveness. Figure 3 displays an example of the 3D points that are removed in the preprocessing stage.

The portion of the office that we are modelling has two windows, so irrelevant points from the outside were also recorded when the RGB-D data was captured, as shown in Figure 1. These points are not helpful when reconstructing the office so these were removed. By visualising and inspecting the point cloud data, we identified that simple thresholding on the point cloud z-values could be used to remove the points corresponding to the outside. A threshold value of 4 metres was chosen. This was an effective approach that removed the majority of outside points, however, in some frames a few of the inside points were also removed. To avoid this problem, a custom threshold value could be carefully chosen for each frame, instead of using the same threshold value for all frames.

Another challenge that we faced was that there was an undesired person in one of our frames. By inspecting the point cloud data of the concerned frame we identified that thresholding could again be used to remove the unwanted points. Since the person was standing in the foreground and on the right side of the frame, we removed all points closer than 2 metres to the camera and on the right side of the frame. This method removed the person entirely from the frame, as can be seen in Figure 2.

Point cloud data often contains points that are in the empty space between foreground objects and the background, so called flying pixels. To remove flying pixels, we implemented a method that for each point checks how many of the points in its surrounding 11×11 neighbourhood are within a

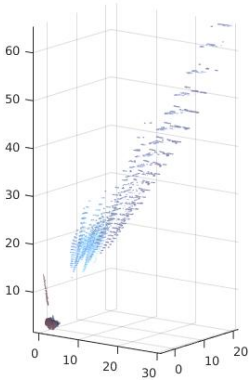
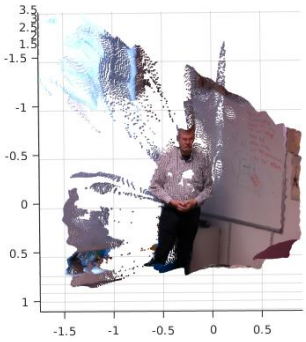
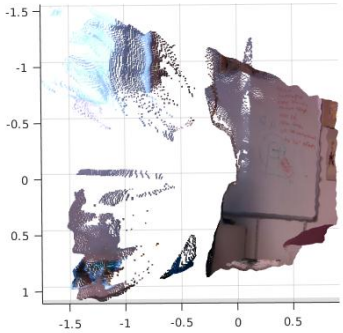


Figure 1: Point cloud data for frame 19, which can clearly be seen to contain many distant 3D points corresponding to points from outside the window.

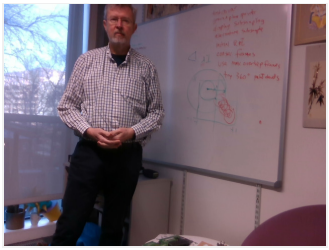


(a) Before removing the person.



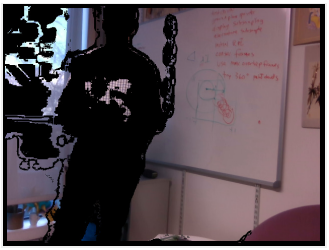
(b) After removing the person.

Figure 2: 3D point cloud for frame 27 before and after removing the person in the scene.



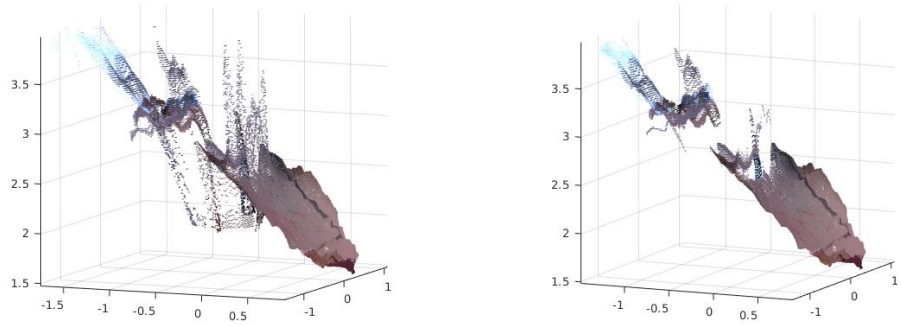
(a) Original RGB image.

(b) Binary mask.



(c) Preprocessed RGB image.

Figure 3: Binary mask of the pixels corresponding to the removed 3D points in the pre-processing stage for frame 27. The reason for the hole in the mask in the center region of the person's chest is that there was no point cloud data for that region in the dataset, as can be seen in Figure 2a.



(a) Before flying pixel removal.

(b) After flying pixel removal.

Figure 4: The pre-processing step of removing flying pixels is demonstrated on frame 27, which was a frame that contained many flying pixels due to it originally containing a person in the foreground.

15 centimetre Euclidean distance. If the number of nearby points is less than a chosen threshold, we class the point as a flying pixel and remove it. Through trial and error we selected 100 as the threshold for the minimum number of nearby points required in order to keep a point. This function is sensitive to the parameters, and a balance between removing too many points and removing all the flying pixels needs to be found. Removing flying pixels is the most computationally expensive stage of our reconstruction pipeline, with each frame taking an average of 20 seconds to be processed. We also explored MATLABs *pcdenoise* function which works in a similar way to our function, however, we did not notice any quality or speed improvements compared to our method. Figure 4 demonstrates the results of our flying pixel removal function.

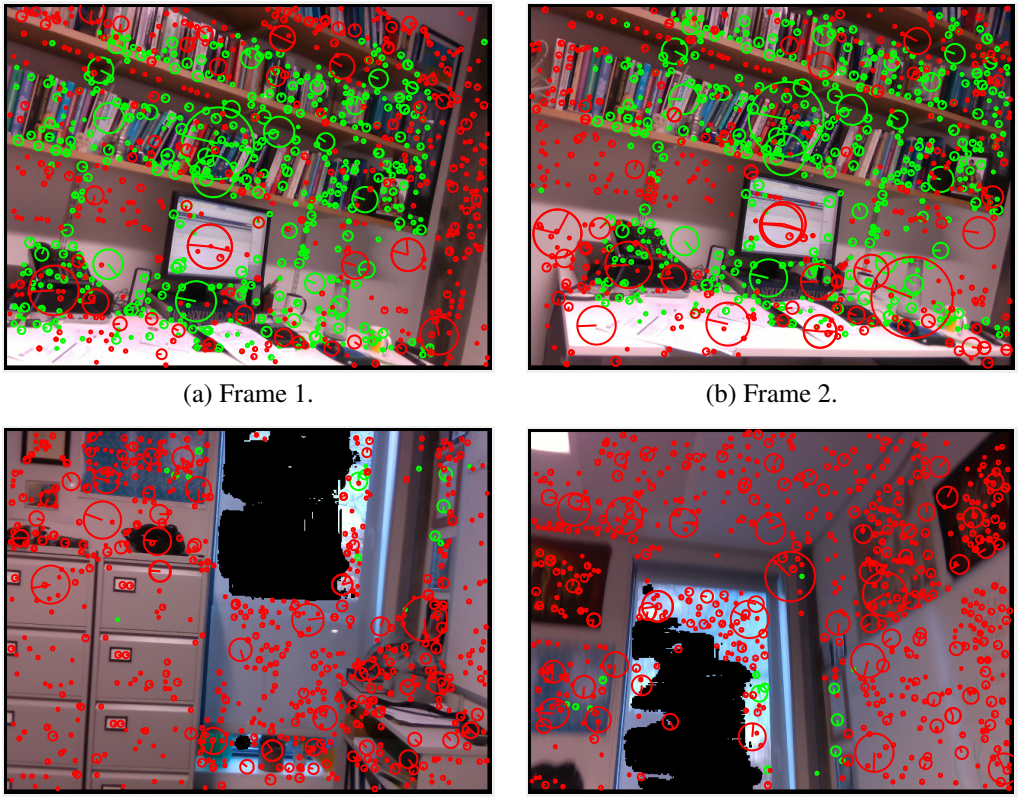
Lens distortions and the limitations of image processing algorithms at handling image boundaries can result in distorted points at the edges of images. These noisy points can cause problems further down the reconstruction pipeline, so in order to prevent this, we removed all points corresponding to a five pixel wide border around the edge of the image.

3 Estimating the 3D Transformation

To estimate the 3D transformation between frames, we leverage the fact that frames in our dataset were captured consecutively, and that there are a lot of common points between consecutive frames. To find corresponding points across consecutive frames, SIFT features [2] were extracted from the non-mask regions of the RGB images for each frame, and their descriptors were matched across frames. To ensure that weak or ambiguous matches were ignored, matches were only kept if the Euclidean distance between descriptors, $D1$ and $D2$, multiplied by a threshold was less than the Euclidean distance between $D1$ and all other descriptors. Higher threshold values ensures that the returned matches are more reliable, however, it may also result in too few matches being returned. We used a threshold of 5, unless only 10 or less matches were returned, in which case we used a lower threshold value of 2.25. Knowing which pixels in each pair of 2D intensity images correspond to the same point enabled us to derive which 3D points match across frames. Figure 5 displays two pairs of consecutive frames and the detected SIFT features.

Given a set of matched 3D points between two point clouds there are a number of algorithms that can be used to estimate the transformation required to transform one of the point clouds into the coordinate system of the other point cloud. We used a least squares fitting approach [3] to estimate the transformations, as outlined in Section 2.1 in [4].

Despite enforcing more stringent requirements for descriptors to be classed as matches, erroneous matches still occur. These erroneous matches can cause the resulting transformation estimates to be very inaccurate. To tackle this issue and to improve the robustness of our reconstruction pipeline we use the RANSAC algorithm [5]. We estimate the transformation using four randomly sampled pairs of matched 3D points. The estimated rotation and translation matrices are used to transform all matched 3D points into one coordinate system. If a chosen threshold percentage of matched



(a) Frame 1. (b) Frame 2.

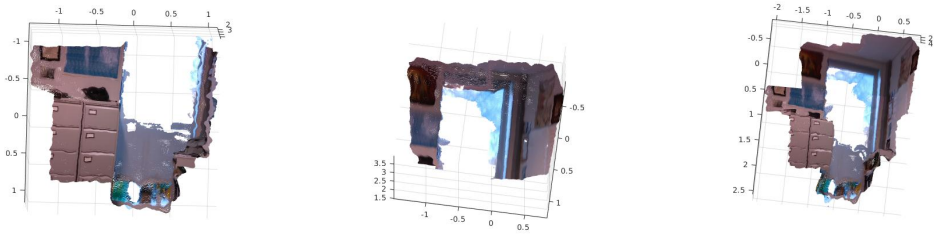
(c) Frame 24. (d) Frame 25.

Figure 5: Examples of SIFT feature matches between consecutive frames. Green circles denote matched features, and red circles denote unmatched features. Frame 1 and frame 2 contain many good matches, and it is easy to find an accurate transformation between the two frames. However, frame 24 and frame 25 have very few correctly matched SIFT features and it was only by using a variation of RANSAC with moving thresholds that a good transformation estimate was found.

points are within a chosen threshold Euclidean distance of each other, the four matched 3D point pairs used to estimate the transformation are added to a list containing good matches. We set the initial threshold values for the RANSAC algorithm such that a minimum of 90% of matched points need to be within a 10 centimetre Euclidean distance of each other. Every 10,000 iterations the number of good points found so far are counted, and if this number is less than 4, then the percentage threshold is dynamically reduced by 10%, and otherwise the method finishes and returns the list of good matches. This process is repeated until the threshold percentage is decreased to 20%, at which point the threshold distance is incremented by 5 centimetres every 10,000 iterations, until at least 4 good matched points have been found, or until the maximum specified number of iterations have been performed. This approach was able to handle the varying number of erroneous matches between frame pairs, and resulted in very good transformation estimates. Figure 6 displays two consecutive point cloud frames and their merged point cloud after using the transformation estimate to transform one point cloud into the coordinate system of the other. Our implementation took on average just over a second to estimate the transformation for each frame, totalling 50 seconds for all 40 frames, when run using a standard laptop.

4 Fusing points from all the frames

In the previous step, we have successfully estimated the transformation between every pair of consecutive frames using SIFT features and the RANSAC algorithm. Given these transformation estimations, we can then fuse all of the points into a single 3D coordinate system. The performance of 3D fusion strongly depends on the accuracy of the reference frame transformation.



(a) Frame 24 (b) Frame 25. (c) Merged frames.

Figure 6: Example of two consecutive point clouds being transformed to the same coordinate system, and being merged together into one point cloud.

Given a translation matrix T_1 (3×1) and a rotation matrix R_1 (3×3) which are estimated from the matched SIFT features between the first frame, F_1, and the second frame, F_2, two-frame fusion is implemented by mapping all the 3D points in F_1 into the coordinate system of F_2 using,

$$transformedPoints = R_1 * F_1 + T_1, \tag{1}$$

where F_1 is a $(3 \times N)$ matrix, with N being the number of points in F_1. The transformed 3D points of F_1 are then added into the list of cloud points in F_2. Once a single transformation has been completed, the merged point clouds are regarded as a single new frame F_2'. This new point cloud can then be merged together with the next frame, F_3 by using the same approach outlined in Equation 1. This process is repeated until all the frames have been merged into one single coordinate system. Equation 2 details how the iterations are performed.

$$T_{t,t+N} = T_{t,t+1} * T_{t+1,t+2} * ... * T_{t+N-1,t+N} \tag{2}$$

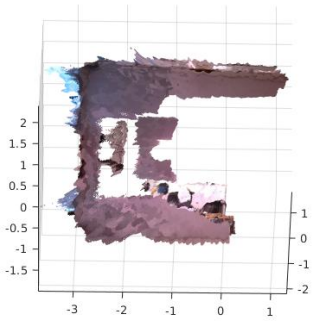
where T stands for the transformation (including rotation and translation), t represents the t-th frame, and N is the total number of frames to fuse.

The result of the final 3D fusion is very good, as is demonstrated in Figure 7, which is in large part due to the accurate transformation estimation algorithm in Section 3. The final 3D model gives a complete reconstruction of the office without noticeable mismatches, and with each frame merged seamlessly. From the top viewpoint, it is clear that the pixels outside the windows are almost entirely cleaned up. The undesired person in front of the right wall is removed, and the majority of undesired flying pixels have also been removed.

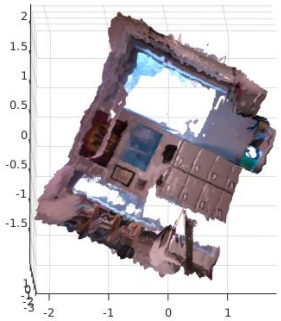
5 Evaluation

This section describes the metric that is used to evaluate the quality of the final model as well as the implementation of the evaluation method. As mentioned in Section 4, the final model is made up of the left, right and end walls together with a part of the ceiling. In theory, the left and right walls of the room should be parallel to each other, which means that the angle between the normal of the left and right wall is expected to be close to $0°$. The closer to $0°$ the angle is, the better the 3D reconstruction is deemed to be. Similarly, the wall at the end of the room should be perpendicular to the side walls (i.e. the angles are expected to be close to $90°$).

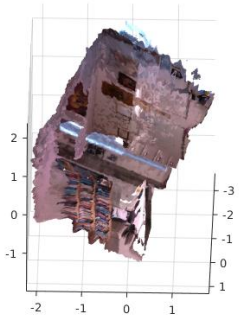
To calculate the angle between two walls, one has to extract the left, right, and end wall planes in advance. The least squares fitting approach [6] can be used to compute the plane fit solution, and is similar to the method used to estimate the 3D transformations in Section 3. The solution makes each of the data points approximately satisfy the plane equation (i.e. the dot product between the data points and the surface normal is approximately zero). In order to obtain more accurate and robust plane fits, we used the M-estimator SAmple Consensus (MSAC) algorithm [7] which is a variant of the RANSAC algorithm [5]. Using the same sampling strategy as RANSAC to generate putative solutions, MSAC chooses the solution that maximizes the likelihood rather than just the number of inliers [7]. In MATLAB, this plane fitting approach was implemented by the $pcfitplane$ function, which fits a plane to a point cloud, and returns a geometrical model that describes the plane. The $pcfitplane$ function has a parameter that specifies the maximum allowable distance that an inlier point can have to the fitted plane. A high maximum allowable distance makes finding a plane fit easier



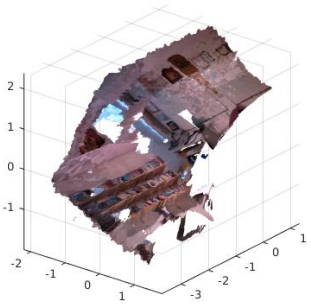
(a) 3D model from top view.



(b) 3D model for end wall.



(c) 3D model for left wall.



(d) 3D model for right wall.

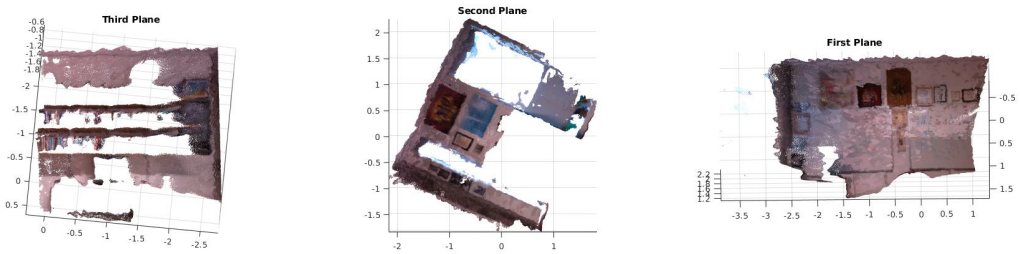
Figure 7: The result of 3D fusion. In order to fully visualize the 3D model, different viewpoints are displayed.

but also results in a thicker plane, which makes the normal less accurate because the plane contains other points that are close to the wall but are not exactly the wall. The maximum allowable distance was set to 0.1 after fine-tuning. As this method gives the best plane fitting solution considering all the 3D points in the point cloud, we extract the first plane and then remove those points in the first plane from the register list before fitting another plane on the remaining cloud points. This plane fitting approach performs rather well as shown in Figure 8. The left wall has some holes which correspond to the parts hidden by foreground objects such as the books, the shelf, the desk, and the computer screen. The holes in the plane for the end wall are due to the windows and the cabinet. The extracted plane of the right wall has a hole in it corresponding to the removed person. Due to the random sampling strategy, the algorithm cannot guarantee to always return the three best plane fits. However, from inspection we have seen that in most cases a good fit is found.

Given the extracted planes for the left, right, and end walls, and their normals, we compute the angles between each plane pair: $88^\circ \pm 2$ (right-end), $87.5^\circ \pm 2.5$ (left-end), $2^\circ \pm 2$ (right-left). The best results we got are 89.7215°, 89.9660° and 0.3533°. These results are very good, especially for the right-end pair which has an almost perfect 90° angle.

In addition, to compute the separation between the left and the right wall, we simply compute the mean across all the points in each wall as the centre-of-mass, before calculating the Euclidean distance between the two points which is approximately 2.866-2.880 meters depending on the run. This value represents the approximate distance between the left and the right wall in the office.

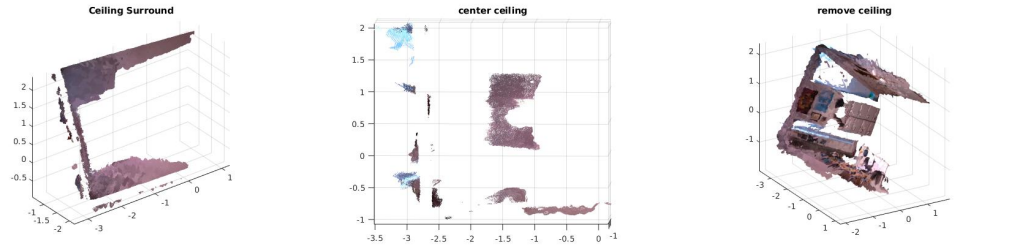
Another task that is required is to remove the ceiling points. As the end, left, and right walls have already been extracted, we continue to apply our plane fitting approach to the remaining cloud points, and add another constraint that the normal of the plane is expected to be perpendicular to the end, left and right walls. We set a loose but reasonable threshold (60°) for the angles. If the angles are all greater than the threshold, the newly extracted plane is regarded as the ceiling. However, we find that not all the ceiling points are located at the same height. Specifically, the center ceiling points are



(a) Left Wall (b) End Wall (c) Right Wall

Figure 8: The three wall planes extracted from the 3D model. The left wall has some holes which correspond to the parts hidden by foreground objects such as the books, the shelf, the desk, and the computer screen. The holes in the plane for the end wall are due to the windows and the cabinet. The extracted plane of the right wall has a hole in it corresponding to the removed person.

located at a lower level than the surrounding ceiling points (see Figure 7a). Therefore, two planes need to be fit to the ceiling and removed before all the ceiling points are completely removed. The final visualization result is shown in Figure 9.



(a) Ceiling Plane 1 (b) Ceiling Plane 2 (c) 3D model with ceiling removed

Figure 9: Demonstration of our ceiling removal approach using plane fitting.

6 Conclusion

In this work we reconstruct a 3D model of the back end of an office by fusing 3D point clouds from 40 frames captured by an Intel RealSense depth sensor. Our implementation is fairly efficient, taking on average 850 seconds to complete all stages of the 3D reconstruction pipeline, including evaluation. The time required can be greatly reduced by removing the flying-pixel removal function, which results in a marginally worse 3D reconstruction, but speeds up the reconstruction pipeline such that it only takes on average 80 seconds to complete, using a standard laptop.

In the pre-processing stage, we remove the pixels outside the windows, and remove an undesired person in the scene by thresholding the Z coordinate. Furthermore we remove the majority of flying pixels by eliminating the 3D points which do not have enough neighboring pixels within a specified distance. The issue of removing flying pixels is computationally expensive. We tackle this by only checking a 11×11 neighbourhood rather than checking the distance to all points in the point cloud. Instead of implementing an element-wise loop, we take advantage of the matrix operation of MATLAB to further speed up the process, achieving an overall speedup from around 10 minutes per frame using the naive approach to 20 seconds per frame using our implementation.

We utilized SIFT features and RANSAC to estimate the transformation between two consecutive frames. However, the estimation between some frame pairs was erroneous due to very few correctly matched SIFT features being found. This often happened when there was a big jump between two consecutively captured frames. We explored using the Iterative Closest Point algorithm, and using alternative features, such as SURF, however we were still unable to find good transformation estimates for the more challenging frame pairs. Finally, we designed a variant of the RANSAC algorithm

which uses moving thresholds, which obtains accurate 3D transformation estimates even for the frame pairs with very few correctly matched points. This enabled us to create a 3D reconstruction that is visually coherent and quantitatively sound. We evaluated the quality of our final model by measuring the angles between each wall and comparing them to their real-world values. The best results we obtained were 89.7215° for the right-end wall pair, 89.9660° for the left-end wall pair, and 0.3533° for the left-right wall pair, which are close to the theoretical ideal solutions.

References

[1] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nie, Reinhard Klein, and Andreas Kolb. State of the Art on 3D Reconstruction with RGB-D Cameras. *Comput. Graph. Forum*, 37:625–652, 2018.

[2] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[3] K. S. Arun, Thomas S. Huang, and Steven D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9:698–700, 1987.

[4] Adele Lorusso, David W. Eggert, and Robert B. Fisher. A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations. In *BMVC*, 1995.

[5] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[6] V Schomaker, J Waser, RE Marsh, , and G Bergman. To fit a plane or a line to a set of points by least squares. *Acta crystallographica*, 12(8):600–604, 1959.

[7] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding*, 78(1):138–156, 2000.

[8] A. Vedaldi and B. Fulkerson. VLFeat: An Open and Portable Library of Computer Vision Algorithms. `http://www.vlfeat.org/`, 2008.

Appendices

A MATLAB Project Code

The open source computer vision library VLFeat [8] is needed to run the code. The function *imag2d.m* provided by the Advanced Vision course organisers was also used.

Function 1: 3D Reconstruction Pipeline

```
%% Load the training data
office = load('office1.mat');
office = office.pcl_train;
%% Load the test data
% office = load('office2.mat');
% office = office.pcl_test;

%% 3.1 Preprocessing and SIFT feature extraction.
processed_frames = {};
x_dim = 640;
y_dim = 480;
max_point_depth = 4;
edge_pixels_to_remove = 5;
edge_indices = remove_edges(edge_pixels_to_remove, x_dim, y_dim);
for i = 1:length(office)
    rgb = office{i}.Color;
    point = office{i}.Location;
    ind_to_delete = find(point(:,3) > max_point_depth); % Remove distant
        points.
    if i == 27
        person_indices = find(point(:,3) < 2 & point(:,1) < 0);
        ind_to_delete = [ind_to_delete; person_indices]; % Remove person from
             frame 27.
    end
    ind_to_delete = [ind_to_delete; edge_indices];

    point(ind_to_delete, :) = NaN;
    rgb(ind_to_delete, :) = NaN;

    flying_pix_ind = remove_flying_pixels(point, 0.15, 100, x_dim);
    point(flying_pix_ind, :) = NaN;
    rgb(flying_pix_ind, :) = NaN;
    ind_to_delete = [ind_to_delete; flying_pix_ind];

    pc = pointCloud(point, 'Color', rgb);
    mask = plot_binary_mask(ind_to_delete, x_dim, y_dim);

    new_rgb = imag2d(rgb, false);
    old_rgb = imag2d(office{i}.Color, false);

    %% Extract SIFT features
    [f,d] = extract_sift(old_rgb, mask);

    %% Store preprocessed data
    processed_frames{i, 1} = new_rgb;
    processed_frames{i, 2} = pc;
    processed_frames{i, 3} = f;
    processed_frames{i, 4} = d;
end

```

```
%% Match 3D points in frame i and frame i+1, using matched SIFT features.
for i = 1:(length(processed_frames)-1)
    [matches, scores] = vl_ubcmatch(processed_frames{i, 4}, processed_frames{
        i+1, 4}, 5);
    if size(matches(2)) <= 10
        [matches, scores] = vl_ubcmatch(processed_frames{i, 4},
            processed_frames{i+1, 4}, 2.25);
    end
    matchedSiftPoints1 = processed_frames{i, 3}(:, matches(1,:));
    matchedSiftPoints2 = processed_frames{i+1, 3}(:, matches(2,:));

    f1_match_ind = convert_2d_ind_to_linear_ind(matchedSiftPoints1(1:2, :),
        x_dim);
    f2_match_ind = convert_2d_ind_to_linear_ind(matchedSiftPoints2(1:2, :),
        x_dim);

    processed_frames{i, 5} = matchedSiftPoints1;
    processed_frames{i, 6} = f1_match_ind;
    processed_frames{i+1, 7} = matchedSiftPoints2;
    processed_frames{i+1, 8} = f2_match_ind;

    %% Visualisation of matched SIFT feature points in frame i and i+1.
    %figure(1); subplot(1,2,1); imshow(processed_frames{i, 1});
    %f1_m = vl_plotframe(matchedSiftPoints1);
    %set(f1_m,'color','g','linewidth', 2);
    %f1_no_match = processed_frames{i, 3}(:, setdiff(1:end, matches(1,:)));
    %f1_no_m = vl_plotframe(f1_no_match);
    %set(f1_no_m,'color','r','linewidth', 2);

    %subplot(1,2,2); imshow(processed_frames{i+1, 1});
    %f2_m = vl_plotframe(matchedSiftPoints2);
    %set(f2_m,'color','g','linewidth', 2);
    %f2_no_match = processed_frames{i+1, 3}(:, setdiff(1:end, matches(2,:)));
    %f2_no_m = vl_plotframe(f2_no_match);
    %set(f2_no_m,'color','r','linewidth', 2);

    %pause
end

%% Find 3D transformation from frame i to frame i+1.
for i = 1:(length(processed_frames)-1)
    M = processed_frames{i, 2}.Location(processed_frames{i, 6}, :);
    D = processed_frames{i+1, 2}.Location(processed_frames{i+1, 8}, :);
    [M, D] = ransac(M, D, 100000, 4, 0.1, 0.9);
    [R, T] = get_transformation(M, D);
    processed_frames{i, 9} = R;
    processed_frames{i, 10} = T;

    %% Visualisation
    %figure(1); pcshow(processed_frames{i, 2});
    %figure(2); pcshow(processed_frames{i+1, 2});
    %merged_pc = fuse_frames(processed_frames(i:i+1, :));
    %figure(3); pcshow(merged_pc);
    %pause
end

%% Fuse frames
fused_pc = fuse_frames(processed_frames);
figure(1); pcshow(fused_pc);
```

```
%% Evaluation
[plane1_pc, plane2_pc, plane3_pc] = fit_plane(fused_pc);
distance = separation(plane1_pc, plane2_pc);
```

Function 2: Pre-processing function that returns the indices of the points in the point cloud corresponding to the edge pixels of its associated 2D image.

```
function pc_indices = remove_edges(num_pixels_to_remove, x_dim, y_dim)
    pc_indices = [];
    count = 0;
    for i = 1:y_dim
        for j = 1:x_dim
            count = count + 1;
            if i <= num_pixels_to_remove | i > y_dim-num_pixels_to_remove | j
                 <= num_pixels_to_remove | j > x_dim - num_pixels_to_remove
                pc_indices = [pc_indices; count];
            end
        end
    end
end
```

Function 3: Pre-processing function that returns the indices corresponding to the flying pixels in the input point cloud.

```
function pc_indices = remove_flying_pixels(pc, distance, number, x_dim)
    pc_indices = [];
    neighbor = 5;
    for i = 1:length(pc)
        total_distances = [];
        if sum(isnan(pc(i,:)))==0
            for mmm = i-x_dim*neighbor : x_dim : i+x_dim*neighbor
                distances = sum((pc(i,:) - pc(mmm-neighbor : mmm+neighbor, :)
                    ).^2, 2).^0.5;
                total_distances = [total_distances ; distances];
            end
            if sum(total_distances < distance) < number
                pc_indices = [pc_indices; i];
            end
        end
    end
end
```

Function 4: Returns and plots the binary mask to show which pixels are removed and which ones are kept.

```
function mask = plot_binary_mask(ind_to_remove, x_dim, y_dim)
    mask = ones(x_dim*y_dim, 1);
    mask(ind_to_remove) = 0;
    mask = reshape(mask, [x_dim, y_dim])';
    %figure(1); imshow(mask);
    %pause
end
```

Function 5: Extracts SIFT features and returns the feature locations and corresponding descriptors.

```
function [f,d] = extract_sift(rgb, mask)
    I = single(rgb2gray(im2double(rgb)));

```

```
    [f,d] = vl_sift(I);

    to_keep = [];
    for i = 1:size(f,2)
        if mask(uint64(f(2,i)), uint16(f(1,i))) == 1
            to_keep = [to_keep; i];
        end
    end
    f = f(:,to_keep);
    d = d(:,to_keep);
end
```

Function 6: Converts the index of a 2D image to a linear index which is used in the point cloud data structure.

```
function linear_ind = convert_2d_ind_to_linear_ind(ind, x_dim)
    linear_ind = [];
    for i = 1:size(ind, 2)
        linear_ind = [linear_ind; x_dim*(uint64(ind(2,i))-1) + uint64(ind(1,i
            ))];
    end
end
```

Function 7: RANSAC with moving thresholds.

```
function [new_M, new_D] = ransac(M, D, max_iter, samples, threshold_dist,
    threshold_percent)
    new_M = [];
    new_D = [];
    i = 0;
    while i < max_iter
        i = i+1;
        subset_ind = randperm(length(M), samples);
        M_s = M(subset_ind, :);
        D_s = D(subset_ind, :);
        [R, t] = get_transformation(M_s, D_s);
        M_t = R*M' + t;
        euclid_dist = sum((M_t - D').^2, 1).^0.5;
        if sum(euclid_dist < threshold_dist) > threshold_percent*length(M)
            new_M = [new_M; M_s];
            new_D = [new_D; D_s];
        end
        if mod(i, 10000) == 0
            [new_M,ia,~] = unique(new_M, 'rows', 'stable');
            new_D = new_D(ia, :);
            if size(new_M, 1) > 4
                i = max_iter + 1;
            else
                threshold_percent = threshold_percent-0.1;
                if threshold_percent <= 0.2
                    threshold_percent = 0.2;
                    threshold_dist = threshold_dist + 0.05;
                end
            end
        end
    end
end
```

Function 8: Estimates the transformation between two sets of matched 3D points.

```
function [R, T] = get_transformation(M, D)
    M_nan = isnan(M);
    D_nan = isnan(D);
    ind = find(M_nan(:,1)==0 & D_nan(:,1)==0);
    M = M(ind, :);
    D = D(ind, :);
    m = mean(M, 1);
    d = mean(D, 1);
    M = M - m;
    D = D - d;
    H = M'*D;
    [U,~,V] = svd(H);
    R = V*U';
    if det(R) > -1.0001 && det(R) < -0.9999
        V_new = V;
        V_new(:, 3) = V(:, 3)*-1;
        R = V_new*U';
    end
    T = d'-R*m';
end
```

Function 9: Fuses the point clouds of all the frames into a single coordinate system using the estimated transformations.

```
function fused_pc = fuse_frames(frames)
    fused_pc = frames{1, 2};
    for i = 1:(size(frames, 1)-1)
        i
        R = frames{i, 9};
        T = frames{i, 10};
        transformed_points = R*fused_pc.Location' + T;
        fused_pc = pointCloud(transformed_points', 'Color', fused_pc.Color);
        fused_pc = pcmerge(fused_pc, frames{i+1, 2}, 0.01);
    end
end
```

Function 10: Fits planes to each wall and to the ceiling, and computes the angles between each wall.

```
function [plane1, plane2, plane3] = fit_plane(fused_pc)
    maxDistance = 0.1;
    %referenceVector = [0,0,1];
    %maxAngularDistance = 5;


    [model1,inlierIndices,outlierIndices] = pcfitplane(fused_pc,maxDistance);
    plane1 = select(fused_pc,inlierIndices);
    remainPtCloud = select(fused_pc,outlierIndices);

    [model2,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,
        maxDistance);
    plane2 = select(remainPtCloud,inlierIndices);
    remainPtCloud = select(remainPtCloud,outlierIndices);


    [model3,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,
        maxDistance);
    plane3 = select(remainPtCloud,inlierIndices);
    remainPtCloud = select(remainPtCloud,outlierIndices);
```

```
19
20      [model4,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,
            maxDistance);
21      plane4 = select(remainPtCloud,inlierIndices);
22      remainPtCloud = select(remainPtCloud,outlierIndices);
23
24      find_ceiling = (rad2deg(acos(abs(model4.Normal*model1.Normal'))) > 60 & (
            rad2deg(acos(abs(model4.Normal*model2.Normal')))) > 60 & (rad2deg(acos
            (abs(model4.Normal*model3.Normal')))) > 60)
25      model_ceiling = model4;
26      plane_ceiling = plane4;
27      if ~find_ceiling
28          [model5,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,
                maxDistance);
29          plane5 = select(remainPtCloud,inlierIndices);
30          remainPtCloud = select(remainPtCloud,outlierIndices);
31          model_ceiling = model5;
32          plane_ceiling = plane5;
33      end
34
35
36      [model_floor,inlierIndices,outlierIndices] = pcfitplane(remainPtCloud,
            maxDistance, model_ceiling.Normal);
37      plane_floor = select(remainPtCloud,inlierIndices);
38      remainPtCloud = select(remainPtCloud,outlierIndices);
39
40      [model_center_ceiling,inlierIndices,outlierIndices] = pcfitplane(
            remainPtCloud,maxDistance, model_ceiling.Normal);
41      plane_center_ceiling = select(remainPtCloud,inlierIndices);
42      remainPtCloud = select(remainPtCloud,outlierIndices);
43
44      figure(2)
45      pcshow(plane1)
46      title('First Plane')
47      figure(3)
48      pcshow(plane2)
49      title('Second Plane')
50      figure(4)
51      pcshow(plane3)
52      title('Third Plane')
53      if find_ceiling
54          figure(5)
55          pcshow(plane_ceiling)
56          title('Ceiling Surround')
57      else
58          figure(5)
59          pcshow(plane5)
60          title('plane5')
61      end
62
63      figure(6)
64      pcshow(plane_center_ceiling)
65      title('center ceiling')
66
67      pc_rm_ceiling = pcmerge(remainPtCloud, plane1, 0.01);
68      pc_rm_ceiling = pcmerge(pc_rm_ceiling, plane2, 0.01);
69      pc_rm_ceiling = pcmerge(pc_rm_ceiling, plane3, 0.01);
70      if ~find_ceiling
71          pc_rm_ceiling = pcmerge(pc_rm_ceiling, plane4, 0.01);
```

```
    end
    pc_rm_ceiling = pcmerge(pc_rm_ceiling, plane_floor, 0.01);
    figure(7)
    pcshow(pc_rm_ceiling)
    title('remove ceiling')

% visualization of models
%     hold on
%     plot(model1)
%     hold on
%     plot(model2)
%     hold on
%     plot(model3)

    eval_12 = abs(model1.Normal * model2.Normal');
    angle_12 = rad2deg(acos(eval_12))
    eval_23 = abs(model2.Normal * model3.Normal');
    angle_23 = rad2deg(acos(eval_23))
    eval_13 = abs(model1.Normal * model3.Normal');
    angle_13 = rad2deg(acos(eval_13))
end
```

Function 11: Computes the distance between the left wall and the right wall.

```
function distance = separation(plane1_pc,plane2_pc)
    center_plane1 = [mean(plane1_pc.Location(:,1)), mean(plane1_pc.Location
        (:,2)), mean(plane1_pc.Location(:,3))]
    center_plane2 = [mean(plane2_pc.Location(:,1)), mean(plane2_pc.Location
        (:,2)), mean(plane2_pc.Location(:,3))]
    distance = norm(center_plane1 - center_plane2)
end
```