Project - Computer Vision (EEN020)

Luca Modica

January 5, 2024

1 Introduction

The goal of this report is to describe and illustrate the results of the computer vision course project. In particular, the project consisted on creating a 3D reconstruction software: a non-sequential SfM (Structure from Motion) method was implemented to achieve this goal, starting from a dataset of images representing the same scene from multiple views. On a high-level explanation, the implemented algorithm follows the below steps in order to achieve this goal.

1. Compute descriptors and features of each image, using a SIFT algorithm.
2. For each image, the 2D points that found correspondences with all the other images in the dataset will be stored. These values will be important in the later steps.
3. For each pair of images $(i, i+1)$ the relative pose will be robustly estimated $[R_{i,i+1}, T_{i,i+1}]$. The inliers found in this step were saved for the final part of the pipeline.
4. Using the relative rotations of the previous step, the rotation averaging method was used to compute the absolute rotation R_i for each camera (image) i.
5. Using an initial pair of images (i_1, i_2) from the dataset with a reasonable baseline, the initial 3D points and their related relative pose were robustly estimated. The points used for the estimation, for both image i_1 and i_2 were the global mat
6. The initial 3D were then use for the translation registration method: for each image i the absolute translation vector T_i was robustly estimated. For this operation, the global matches between the image i and one of the image in the initial pair and the absolute rotation of the current camera i were used.
7. The absolute translation vectors were then refined using a Levenberg-Marquardt method.
8. For each pair of images $(i, i+1)$, the corresponding inliers and the related absolute cameras $(R_i, Ri+1)$ were used to triangulate the final 3D scene points. The results were then plotted alongside all the cameras of the dataset.

Considering the steps of the project above, the main focus of this report will be to illustrate the main key concepts of the implementations, the main issues aroused during the realization and the final results through the plots obtained in step 8.

2 Implementation

For the realization of the SfM pipeline, the programming language chosen was MATLAB for performance on matrices operations and ease of generating 3D plots. Parts of the code are also the results of the adaptations of routines written in previous assignments of the course, which provided part of necessary knowledge to realize the project. Moreover, *some algorithmic and code choices were developed in collaboration of the student Manousos Manouras*, especially for more detailed parts (thresholds tests, debugging and robust estimations adaptations).

Beyond the overall code project, down below the main key points of the pipeline will be described, with emphasis to new features that differ from the assignments' routines and some of their adaptations. The main points are in order of where they can be found in the pipeline, with some side features descriptions at the end.

Global matches

One of the first challenge of the project was to have points that, globally (that is, across all of the images in the dataset), have matches. This because they will then used alongside the initial 3D points taken from the initial pair of images to estimate the absolute vector for each cameras, representing the related correspondences.

To solve this problem, after running the SIFT algorithm, the following code will use the first image as reference to then find, for each image, the feature 2d points that can be also found in all the other ones.

```
%% Computing common correspondences between all images
disp("Computing common correspondences between all images...");
for i = 2:numImages
    matches{i-1} = vl_ubcmatch(descs{1}, descs{i});
end

common_indeces = matches{numImages-1}(1,:);
for i = 1:numImages-2
    temp_idx = ismember(common_indeces,matches{i}(1,:));
    common_indeces = common_indeces(1,temp_idx);
end

x_matches{1} = feats{1}(1:2,common_indeces);
x_matches{1} = [x_matches{1};ones(1,length(x_matches{1}))];

for i = 1:numImages-1
    temp_idx = ismember(matches{i}(1,:),common_indeces);
    matches_in_2nd_img{i} = matches{i}(2,temp_idx);
    xA{i} = feats{i+1}(1:2,matches_in_2nd_img{i});
    x_matches{i+1} = [xA{i};ones(1,size(xA{i},2))];
end
disp("done!");
%%
```

Listing 1: Code used to find, for each image, its point features in common with all the other images in the dataset.

Robust relative pose estimation

After setting up the project with also global correspondences for each image, the second important part of the the pipeline implementation was how to robustly estimate the relative pose for each adjacent pair of images. The solution was to implement a RANSAC algorithm (adapted from the E matrix estimation of previous assignments) that allows to estimate, in parallel, an essential matrix and an homography from a minimal sample of correspondences: 8 for E and 4 for H. This to take into account both planar and non-planar cases. The overall function, called `estimate_R_T_robust`, will return the following values:

- the best relative poses (R and T) between the essential matrix estimation and the homography estimation, based on which part found most inliers;
- the indices of the best inliers, used in the final triangulation step of the pipeline;
- the related best 3D points, used mainly for debug purposes.

We will now enter in more detailed part of the function implementation. Down below the code part of the RANSAC dedicated to the E matrix estimation is shown:

```
% **E estimation part**
if ~breakE
    % sample 8 correspondances, the minimum
    % required by the 8-point algorithm
    idxs = randperm(size(x1, 2), sE);
    x1s = x1(:, idxs);
    x2s = x2(:, idxs);

    % estimating E
    E = enforce_essential(reshape(estimate_F_DLT(x1s, x2s), [3 3]));

    % compute the errors to select the inliers
    inliers = ( compute_epipolar_errors(E, x1 , x2 ).^2 + ...
        compute_epipolar_errors(E', x2 , x1 ).^2 ...
        ) / 2 < epipolar_threshold^2;

    % check if we obtain a greater number of inliers
    % compared to the last matrix E estimation
    if sum(inliers) / length(x1) > epsE
        inlE = inliers;

        % extract R and T from E
        [P, X_E_curr] = find_best_P2(E, x1, x2, P1, inliers);
        % update the values if the essential matrix
        % is acceptable
        if P ~= zeros(3, 4)
            % update iter count and number of iters for E
            epsE = sum(inlE) / length(x1);
            itersE = ceil(log(1-a)/log(1-epsE^sE));

            X_E = X_E_curr;
            R_best_E = P(:, 1:3);
            T_best_E = P(:, 4);

            % check for loop exit condition, setting the
            % related flag in case of number of iteration
            % required reached
```

```
            if itersE + 5000 <= itersCount
                breakE = true;
            end
        end
    end
end
```

Listing 2: Essential matrix estimation part in the robust estimation of a relative pose.

Compared to a default implementation of a RANSAC algorithm to robustly estimate a specific object, in this parallel version I applied a flag for each element estimated that contributed to the relative pose estimation (`breakE` for the essential matrix part and `breakH` for the homography). That flag will be set to `true` when the usual exit condition of RANSAC (reached a sufficient amount of iteration given in this case by `itersE`) is met. Due to space for improvement and heuristic from trials and errors, to the minimum number of iterations 5000 more iterations threshold was added (the same was applied to the H estimation case). Related to the essential matrix estimation part, the best R and T are retrieved from the best camera among that passed the cheirality test, among the 4 extracted from E: It will be considered only if the new estimated E is better in terms of outliers.

The homogrpahy estimation part, in terms of structure and relative pose choice, is very close to the implementation above. In this specific case, what will be considered after estimating H are the 2 essential matrices that can be obtained after the homography estiamtion.

```
% estimate H
 H = reshape(estimate_homography_DLT(x1s, x2s), [3, 3]);

 % extract R1, t1, R2 and t2 to then compute the
 % candidate E1 and E2 matrices
 [R1, t1, R2, t2] = homography_to_RT(H, x1, x2);
 E1 = enforce_essential(skew(1) * R1);
 E2 = enforce_essential(skew(t2) * R2);
```

Listing 3: Homography estimation and decomposition in the 2 essential matrices.

The check will be then the same as for the single essential matrix above: if either $E1$ or $E2$ has a better inliers estimation than previous E estimation from H, then the relative pose considered in this case will be the best between $E1$ and $E2$. Note that this check is completely independent from the single essential matrix directly estimated: results from H and E cannot be compared directly. For this reason, in the algorithm It will be kept track of one inlier rate (ϵ_E and ϵ_H) per estimated object in parallel.

After breaking out of the loop (for reaching the sufficient number of iteration for both E and H or for reaching the maximum number of iteration set), the algorithm will then return the best results from either E or H, based on who get the best overall results in terms of inliers rate.

Robust translation registration and refinement

After the rotation averaging step and estimating the first 3D points from the first pair of images (always using `estimate_R_T_robust`), another important part of the project is to robustly estimate the absolute translation T_i (for each camera i) from 2D-3D correspondences $x_i \iff X_i$. In this case, x_i represents the global matches for the image i, while X_i the initial 3D points. The implementation of the function for a robust estimation for T is shown below:

```
function best_T = estimate_T_robust(xs, Xs, R, inlier_threshold)
```

```
    num_iterations = 2000;
    best_T = [];
    best_inliers = [];
    best_eps = 0;

    for i = 1:num_iterations
        indices = randperm(size(xs, 2), 2);
        xs_sample = xs(:, indices);
        Xs_sample = Xs(:, indices);

        T = estimate_T_DLT(xs_sample, Xs_sample, R);
        proj = pflat([R, T] * Xs);
        inliers = ((sqrt(sum((xs - proj).^2, 1))) / 2) < inlier_threshold;

        eps = sum(inliers) / length(xs);

        % Update the best model if the current one has more inliers
        if eps > best_eps
            best_eps = eps;
            best_inliers = inliers;
            best_T = T;
        end
    end

    % Optional: Refine the pose with all inliers
    if isempty(best_inliers)
        % return a default value
        best_T = [0; 0; 1];
        warning(['RANSAC was unable to find a good pose with the given ' ...
            'inlier threshold']);
    end
end
```

Listing 4: Function to robustly estimate the absolute rotation T.

As It can be noticed, the implementation also follows a RANSAC estimation for a robust estimation, in this case with a fixed amount of iterations for an overall sufficient heuristic in the reference datasets. Other than the 2D-3D point correspondences, the function receive as parameter the absolute rotation of the same camera i and the used inlier threshold for RANSAC. R_i has 2 main uses.

- Setup the DLT matrix for the estimation of T_i using 2 point correspondences (2-point method). In particular, the implementation of this estimation follows one of the possible options to perform a transaltion registriation in a non-sequential SfM pipeline:

```
    function T = estimate_T_DLT(xs, Xs, R)

        xskew1 = skew(xs(:, 1));
        xskew2 = skew(xs(:, 2));
        A = [xskew1; xskew2];

        b = [-xskew1 * R * Xs(1:3, 1); -xskew2 * R * Xs(1:3, 2)];

        T = A \ b;
    end
```

Listing 5: The 2-point method used to estimate T, with a linear DLT solution.

- compute the reprojection error using the initial recontracted 3D points and an estimated T_i, in order to compute the number of inliers.

At the end of the function, a possible candidate solution will be returned if outliers are found. If this condition is not fullfilled, a fallback solution with will be returned with a warning.

As end step of this novel feature, each estimated translation vector T_i will be then refined using a Levemberg-Marquardt method. The implementations of the method follow closely the one implemented in an earlier assignment of the course, from the linearization of the erorr to the value update; the only change added is to the Jacobian always to linearize the reprojection error, in this case with respect of T.

```
function [r, J] = linearize_reproj_error_wrt_T(R, T, x, X)
    J = [];
    for i = 1:length(X)
        Xi = X(1:3, i);
        df1dt1 = -1 / (R(3, :) * Xi + T(3));
        df1dt2 = 0;
        df1dt3 = 0;
        df2dt1 = 0;
        df2dt2 = -1 / (R(3, :) * Xi + T(3));
        df2dt3 = 0;

        Ji = [
            df1dt1, df1dt2, df1dt3;
            df2dt1, df2dt2, df2dt3
        ];
        J = [J; Ji];
    end

    [~, r] = compute_reprojection_error([R, T], x, X);
    r = reshape(r, 2 * length(X), 1);
end
```

Listing 6: Function to linearize the reprojection error with respect of T.

Other features

During the project realization, few other new side features were implemented to achieve the desired result.

- *Function to filter far 3D points.* The function, in particular, use a DBSCAN algorithm to filter points that are not part of a cluster, thus points that can be considered as outliers. In the project, the name of the function is `filter_far_3d_points`.
- *Debug mode.* During the realization of the pipeline, a simple debug mode was implemented to pre-load computed features during the main steps of the algorithm, such as the SIFT features of each image. If it's activated (by setting the related `debug_mode` boolean flag to `true` in the main function), the software will look for pre-computed features in specific main steps. Moreover, more plots will be generated to let the developer check the results of the pipeline in its specific parts, such as during the the relative poses estimations. Also, the debug mode will set a fixed random seed for stochastic operations (such as random permutations).

3 Unexpected problems and issues

During the pipeline realization, few problems and unexpected issues were encountered. Some of them, unfortunately, can be still encountered also due to the stochastic nature of the RANSAC algorithm. Despite this, it was still possible to show at least partial final results for each provided dataset.

On a high level explanations, one the main unexpected issues of the project is related to some over-skewed 3D point clouds during the final triangulations step. This can be due to cases with to few inliers, which can lead to some poor estimations of the absolute translation vector; this alongside to far outliers that sometimes were not correctly removed.

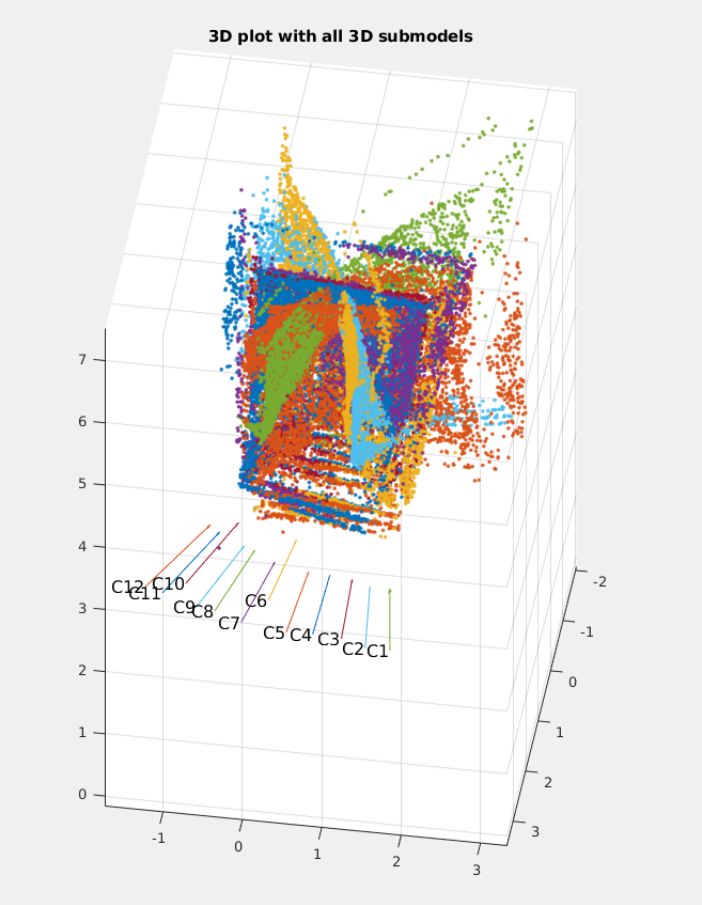

Figure 1: Case of 3D point clouds that are over-skewed or distorted.

4 Results

Given the key points of the implementation and and the unexpected issues found, down below the the final resulting 3D plots with the estimated cameras are displayed for the dataset 3-9.

Due to persistent problems, for some dataset some 3D point clouds from the final triangulations were removed and some result are approximated also with earlier version of the code. This to show, despite the issues encountered, the current achievable results of the implemented project.

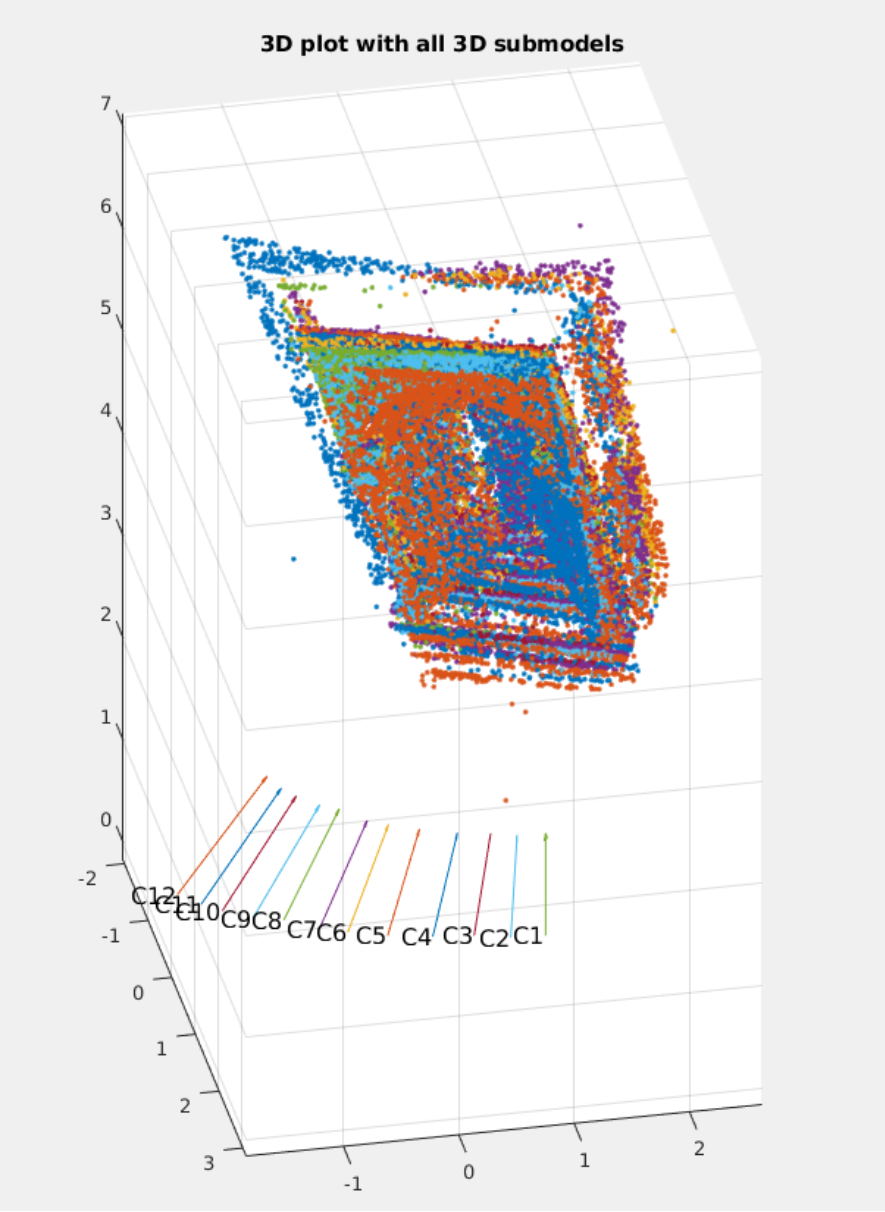

Figure 2: 3D reconstruction from the dataset 3.

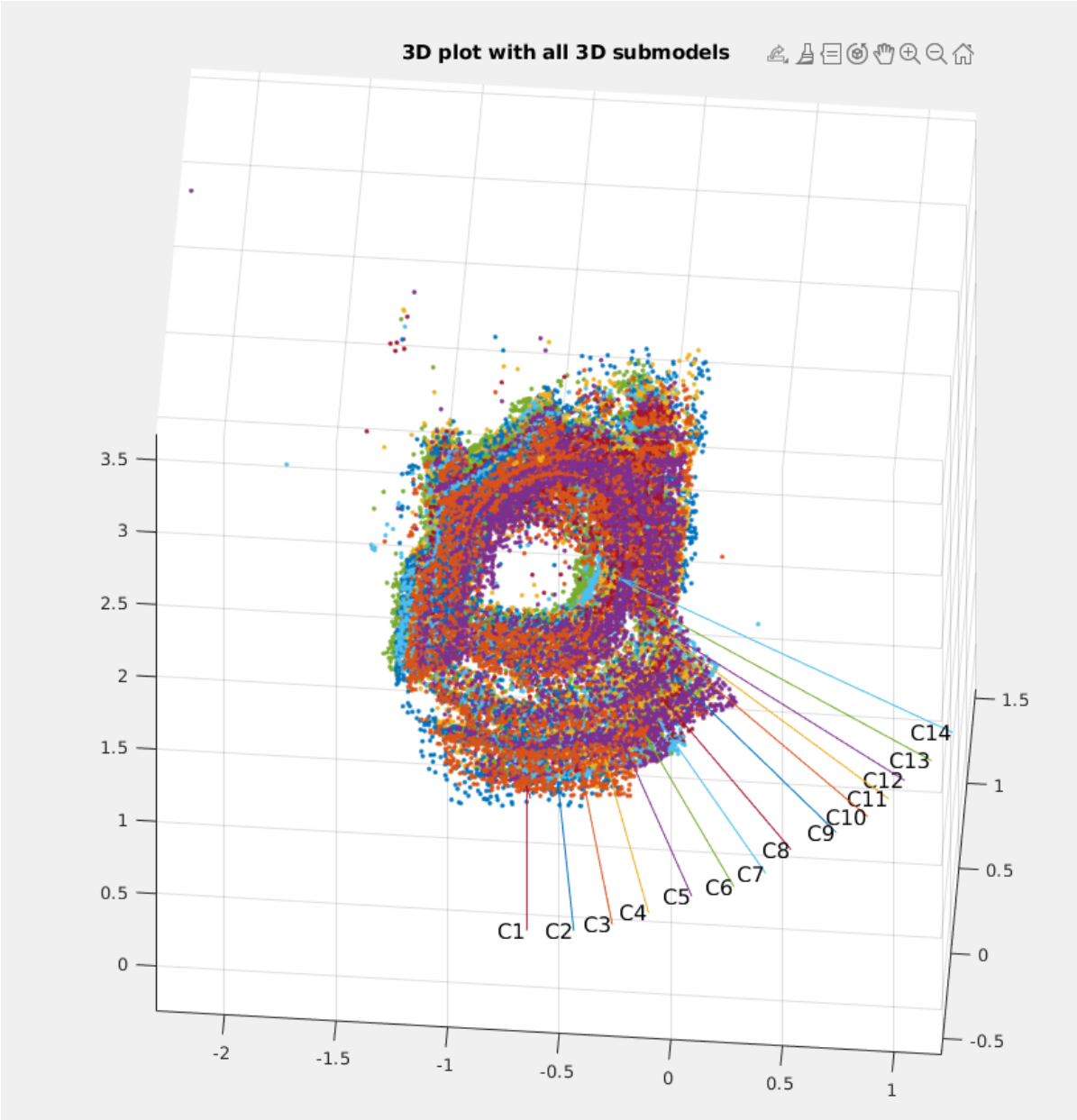

Figure 3: 3D reconstruction from the dataset 4.

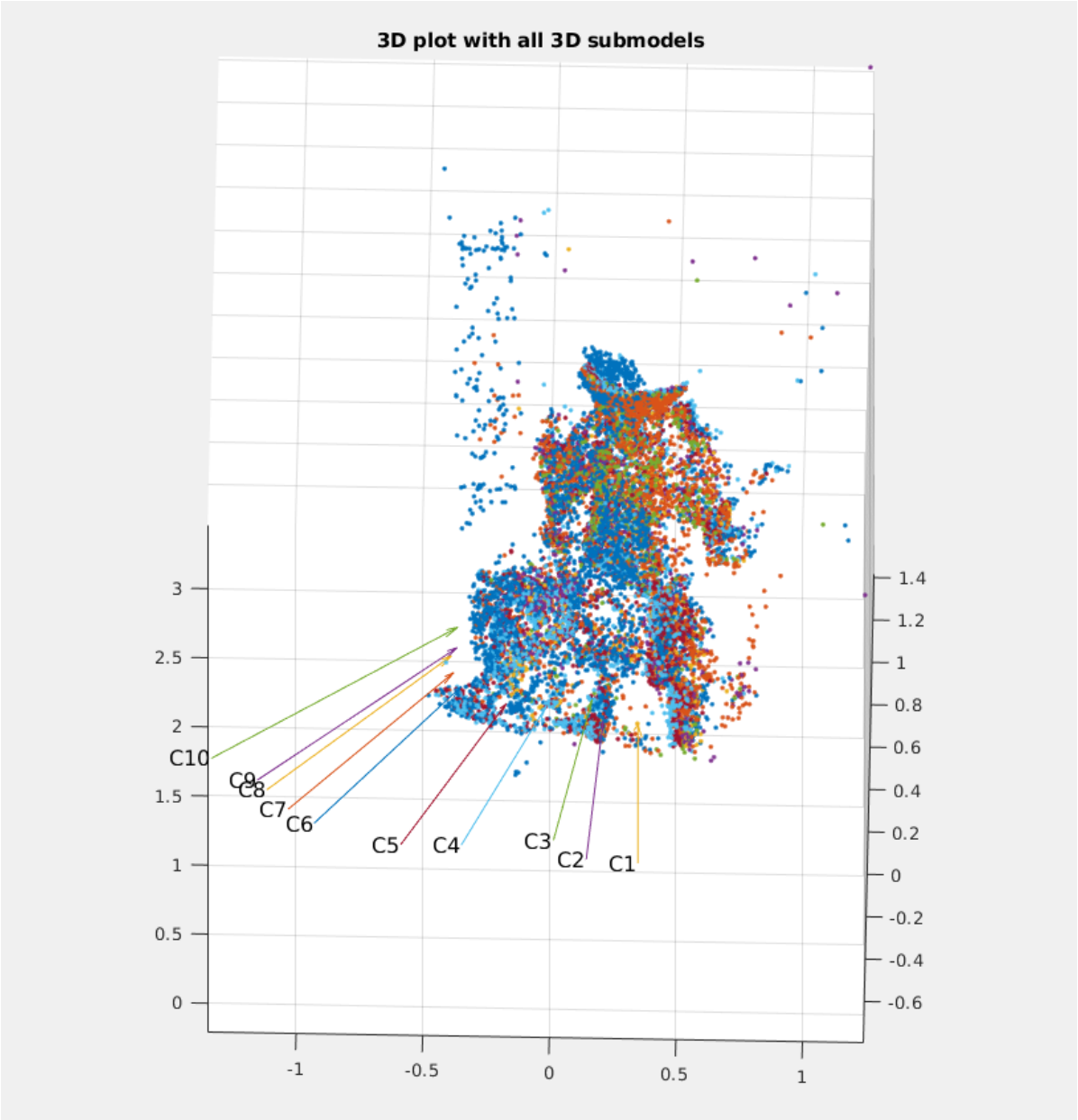

Figure 4: 3D reconstruction from the dataset 5.

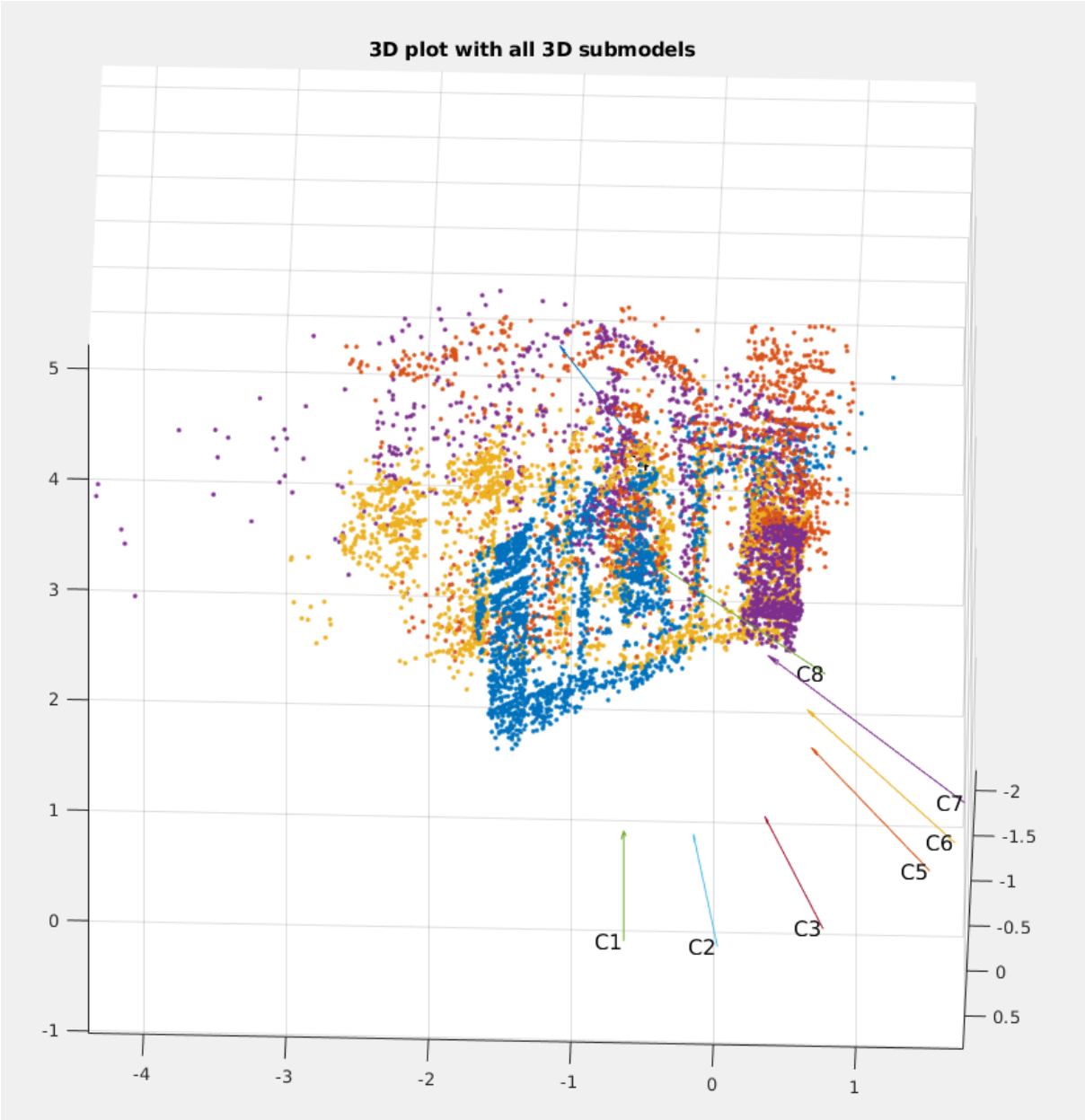

Figure 5: 3D reconstruction from the dataset 6.

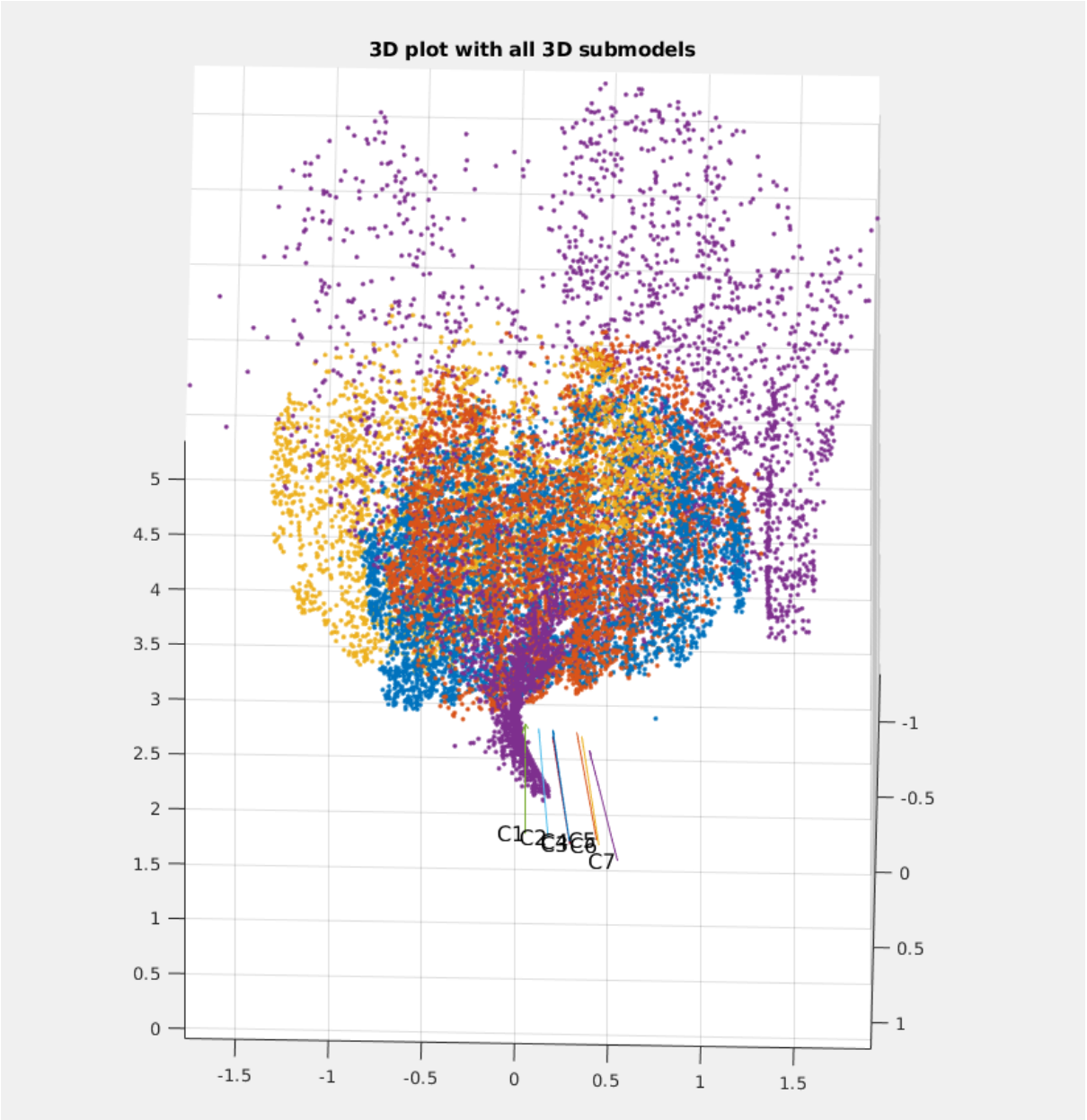

Figure 6: 3D reconstruction from the dataset 7.

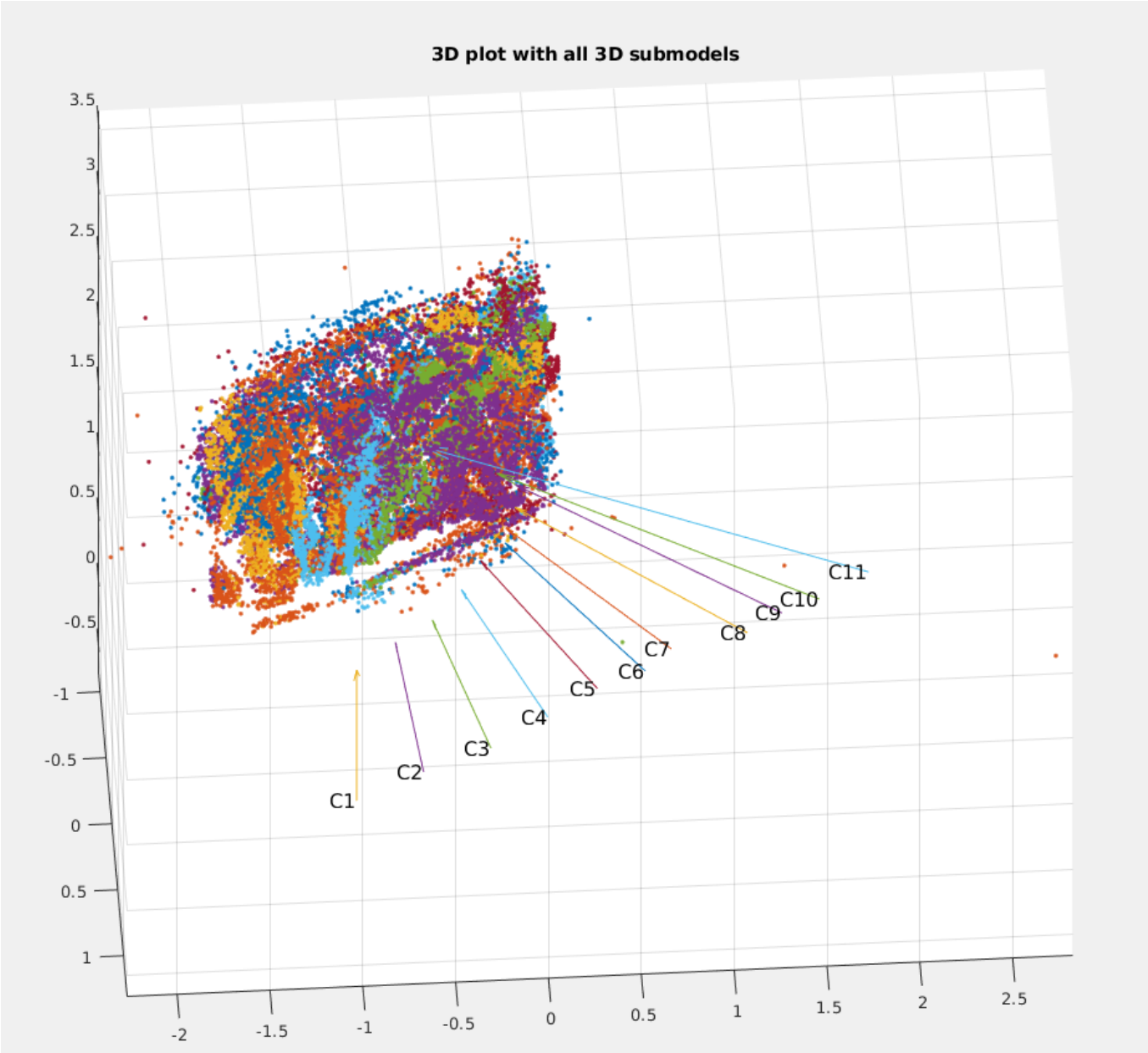

Figure 7: 3D reconstruction from the dataset 8.

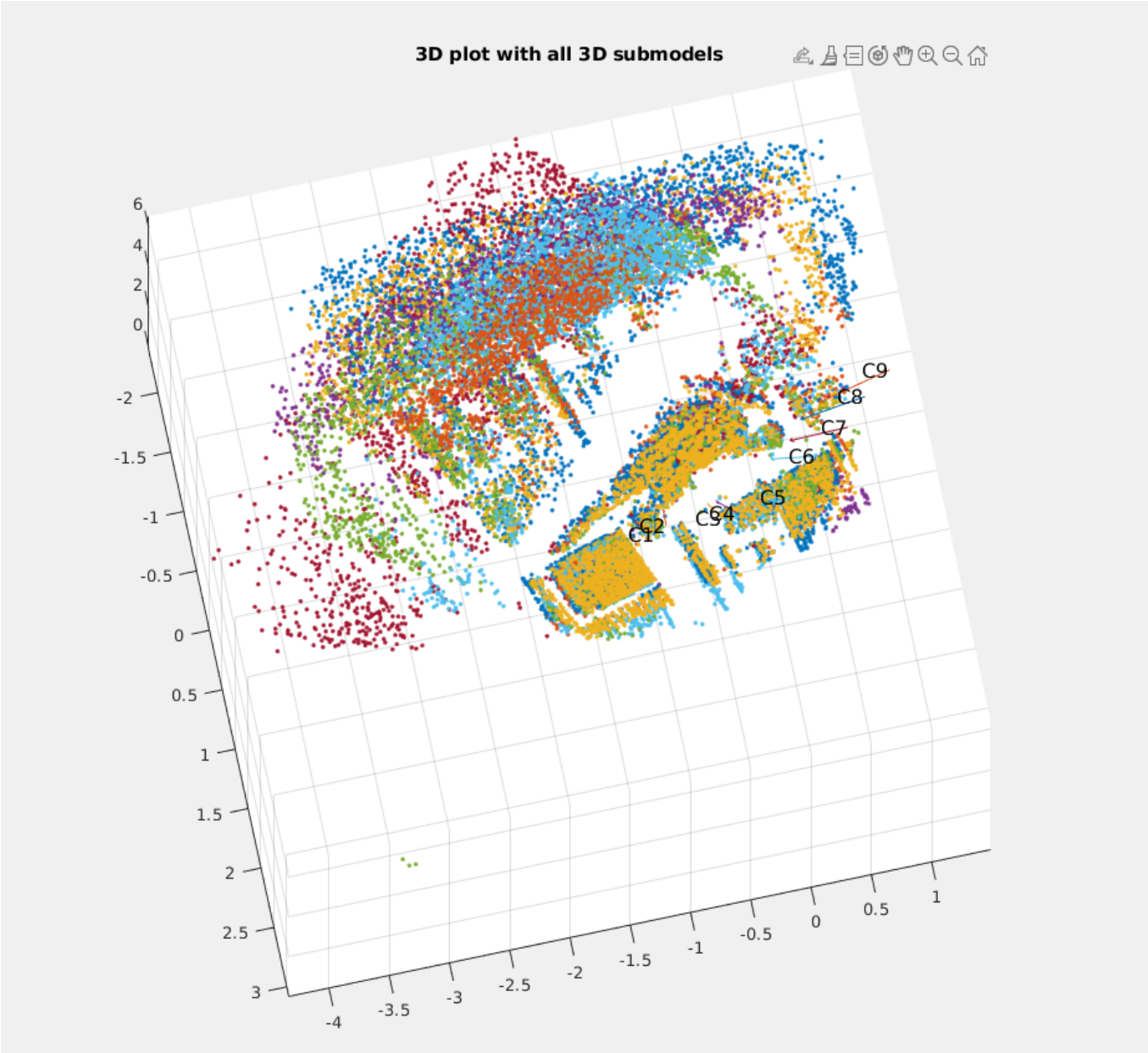

Figure 8: 3D reconstruction from the dataset 9.