Assignment 4: Generate a Digital Surface Model

In this assignment, our aim is to create a 3D digital surface model from 3 aerial images captured by a DJI Phantom 4 UAV flying in approximately 50 meters above ground level (AGL). In order to do this, we need to perform the following

- Recover the camera-pose (position and orientation) using P3P for each of the three image using Ground Control Points which were measured by a GPS.
- Detect and match features between images in order to obtain 2D point correspondences
- Perform outlier rejection using RANSAC and the Fundamental Matrix
- Triangulate points to generate a 3D point cloud

The primary objective of this assignment is to use the algorithms and theory obtained from previous assignments to build a pipeline for generating a DSM from aerial images.

1 Use P3P to recover camera positions and orientation

The first step in the assignment is to recover the camera positions and orientations for each image using a number of ground control measurements performed with a GPS. In order to so you should use your previously developed p3p algorithm.

The intrinsic parameters of the camera and radial- and tangential distortion parameters can be imported by loading the file '**DJI_Phantom4_cam.mat**' located on DTU Learn.

For the spatial resection using P3P you should also undistort your image coordinates, as the camera is affected by radial and tangential distortion. This easily done with the MATLAB function undistortPoints:

```
[u,v]=undistortPoints([u_d v_d],cameraParams);
```

Where [u,v] is the undistorted pixel points, [u_d, v_d] is the original (distorted) pixel coordinates. cameraParams is a struct containing the radial- and tangential distortion coefficient.

In the following three tables, you are provided with 4 3D world points correspondences and 2D image coordinates. The three tables represent the individual images '100_0002_0067.jpg', '100_0002_0068.jpg' and '100_0002_0100.jpg' which all are captured from a DJI Phantom 4 drone.

PointID	UTM Coordinates	Image Coordinates (pixels)
1	[719970.70 6191955.16 69.59]	[1622 1321]
2	[719939.94 6191966.88 69.27]	[3745 1239]
3	[719978.34 6191972.15 71.10]	[1575 2536]
4	[719943.87 6191988.15 74.42]	[4143 2659]

Table 1: Ground Control Points for '100_0002_067.jpg'

PointID	UTM Coordinates	Image Coordinates (pixels)
1	[719971.93 6191944.48 68.99]	[1292 1309]
2	[719928.46 6191961.51 68.11]	[4193 1221]
3	[719985.46 6191966.53 71.24]	[1050 2917]
4	[719938.43 6191985.12 73.90]	[4328 2888]

Table 2: Ground Control Points for '100_0002_068.jpg'

PointID	UTM Coordinates	Image Coordinates (pixels)
1	[719921.16 6192001.99 76.57]	[949 785]
2	[719970.52 6191985.10 72.77]	[4457 730]
3	[719911.56 6191975.24 71.38]	[1139 2692]
4	[719958.46 6191950.13 67.70]	[4376 3000]

Table 3: Ground Control Points for '100_0002_100.jpg'

As a guide, there has been uploaded a MATLAB code skeleton which can help you get started 'PART1_CameraPoseEstimation_forStudents.m'. This code already contains the image and world coordinates for the first image (67). You are also provided with GPS-RTK positions from the UAV, which can help validate your results. If your camera center estimate comes within 50 cm of the GPS-RTK position your estimates should be correct. Repeat the P3P estimations for Image 68 (100_0002_068.jpg) and Image 100 (100_0002_100.jpg').

2 Feature Detection and Matching

In this part of the assignment we will work with detection, extraction and matching of features in order to generate the necessary image point correspondences required to do triangulation. We will use SURF features, which shares a lot of resemblance with SIFT features.

You are given a MATLAB script on DTU learn, 'Part2_FeatureDetectionandMatching_forStudents.m', which takes care of loading the images, detecting SURF features and performing similarity matching (based on the features descriptors). The feature matches however are not perfect and we need to eliminate the outliers using a RANSAC approach.

For the RANSAC routine, then it is designed to randomly pick 8 point matches for each iteration in order to calculate the Fundamental Matrix. The Fundamental Matrix is then used to evaluate a consensus for all the points, i.e. to veritfy if the epipolar constraint is fulfilled ($\mathbf{x}_2^T\mathbf{F}\mathbf{x}_1 = 0$). As we are not sure that the randomly picked points are actually inliers we would need to iterate a number of times and keep track of how many inliers we get in each iteration and finally select the model which maximized the number of inliers.

In order to implement the RANSAC routine, you should just provide your previously developed function for estimating the Fundamental Matrix as this constitutes our consensus model. Once the code is running you should experiment with some settings to get a feel for this type of algorithm. Try with different values for the number of iterations and observe how the number of inliers are varying between iterations. Try also to vary the number of randomly picked matches and see what effect that brings (remember that the 8-point algorithm easily can work with more than 8 matches). A screenshot of the RANSAC algorithm is seen below.

```
% Estimate Fundamental Matrix with RANSAC

RANSACiter = 100;               % Number of Iteration for the RANSAC routine
F=zeros(3,3,RANSACiter);        % Array for Fundamental Matrices
numInliers=zeros(1,RANSACiter); % Vector to show the number of inliers associated with each RANSAC iteration
L=length(features1);
threshold = 0.01;               % Consensus threshold
best_idx=[];                    % vector of indices for feature inliers

for i = 1:RANSACiter

    % Randomly select 8 matched points
    sampleFeatures = randperm(length(features1),8);

    % Provide your own code for Estimation of the Fundamental Matrix (from
    % Assignment 3)
    F(:,:,i) = EstimateFundamentalMatrix(valid_points1(sampleFeatures).Location',valid_points2(sampleFeatures).Location');

    % Compute the consensus
    residual = diag(abs([valid_points2(1:L).Location ones(L,1)]*F(:,:,i)*[valid_points1(1:L).Location ones(L,1)]'));

    %find number of points that are within the inlier threshold
    idx = (residual < threshold);
    numInlier=length(residual(idx));
    if (numInlier > max(numInliers))
        best_idx=idx;
    end
    numInliers(i)=numInlier;

end
```

As the final task in this part of the assignment you should extend the code, such that it also calculates feature matches for Image 2 and 3. You should end up with matching point correspondences between all three views, i.e. you should only store the points which are matched for all three viewpoints. Remember to store the results in a .mat file in order to easily import the points for the triangulation scripts.

```
save('pointcorrespondences.mat','features1', 'valid_points1','features2',...
    'valid_points2','features3', 'valid_points3');
```

3 Triangulation

After successful completion of part 1 and 2, we should be able to triangulate the points in order to generate a Digital Surface Model. Similarly, to part 1 and 2, you are provided with a MATLAB script 'Part3_Triangulation_forStudents.m' to help guide you through the process.

In this script we will perform triangulation based on both 2-view correspondences and 3-view correspondences. The triangulated points will be exported/written to individual .txt file for both cases. The requirement for the script to run is that you have a functional triangulation function (from assignment 3).

Once the files with 3D points has been generated you should inspect the points using the software CloudCompare. This software is free to use and allows you to easily inspect 3D point clouds.

CC can be fetched from: https://www.danielgm.net/cc/. Make sure to select the latest stable release and download to whatever platform you are running (windows/mac/linux).

Once you have CC installed, you should simply just open the program and go to ‘File’->’Open’ and navigate to your points file generated from 2-view triangulation. You would encounter a window for importing data and here you should indicate that we use ‘,’ as a separator for the generated files and press ‘Apply all’. Press ok to translate the coordinates into a more narrow range.

Now you should see a point cloud in the main window. Hereafter you should repeat the above process and import the points obtained from 3-view triangulation.

Once both point clouds are loaded you can distinguish them by selecting different colors. Mark one of the clouds in the left side pane and go to ‘Edit’->’Colors’->’Colorize’ and select an appropriate color. Make a preliminary assessment between the two clouds, i.e. which one seems to be smoothest and with smallest variations? Reflect on your findings.

For a reference, you are also provided with 3D points for the entire drone-survey in ‘points_agisoft.txt’. In the image below you can see how this point cloud compare with the MATLAB generated cloud (green mesh) using 3-view triangulation.

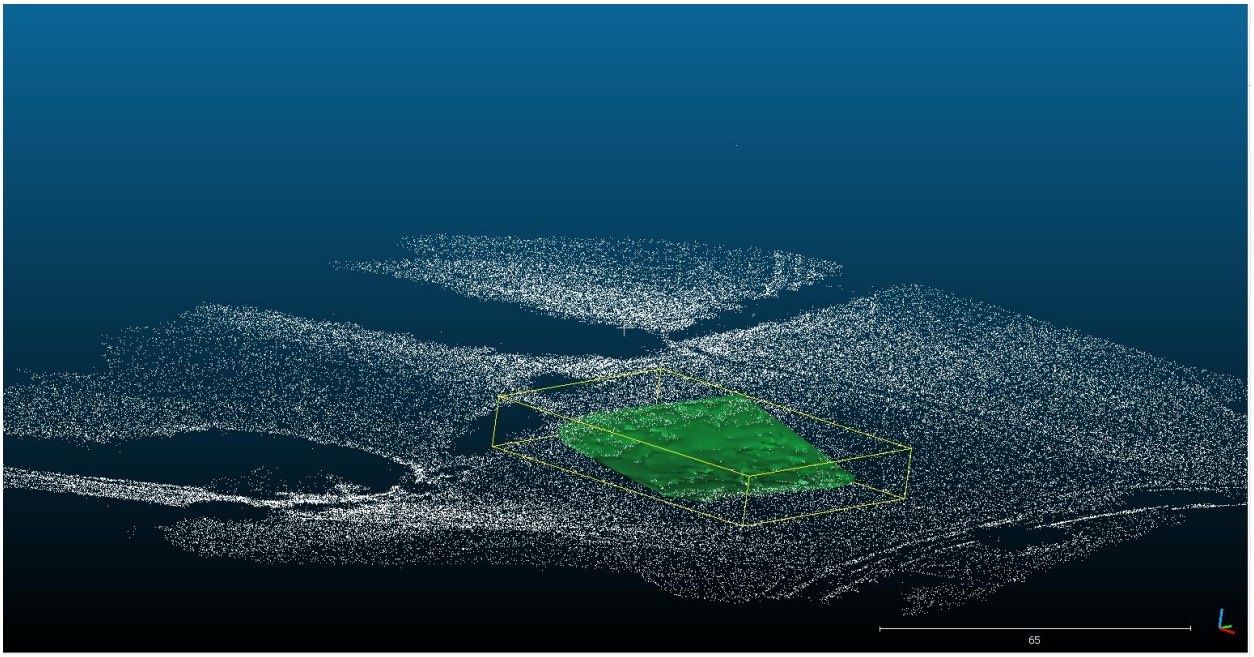

Figure 1: Digital Surface Model (DSM) from Agisoft overlayed with solution using 3 aerial images in MATLAB (Green) in CloudCompare.