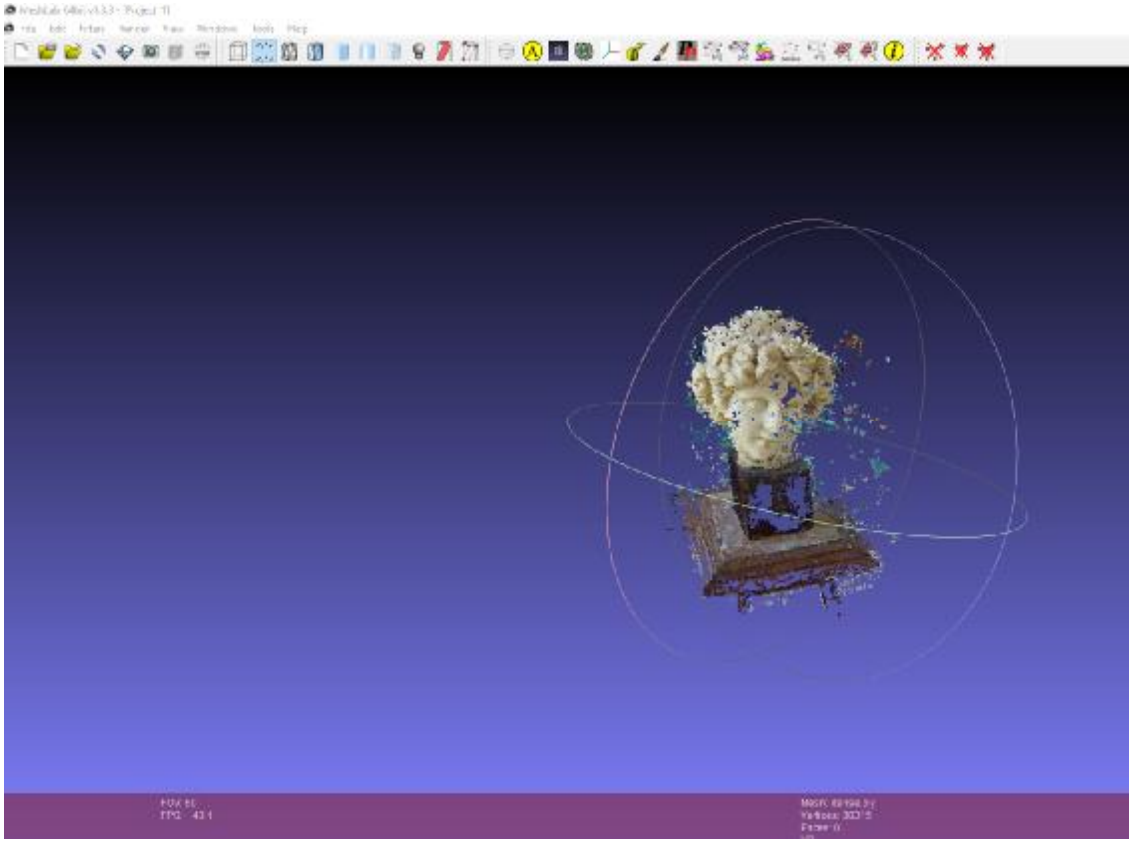
SFMedu and Stereo Reconstruction

Baoshi Sun (WatID 20625524)

Experiment Environment

- Computer: Asus M32BF (CPU AMD A10-7800, 12G Memory), Windows 10
- MATLAB R2015b
- MeshLab_64bit v1.3.3
- Digital Camera: iPhone 5S rear camera

Problem 1



Problem 2

Major Steps for SfM and dense matching:

1. Initialization: clear working space, append lib paths and set motion images (hard coded 5 jpg files); Try to retrieve focal length from EXIF dictionary. Unfortunately, there is no entry found. So the program just sets the focal length to 719.5459, which may be got from other source by the developer.
2. SIFT matching: loop through all 5 frames and perform the following operations

2.1. call 'match2viewSIFT' function to perform SIFT matching of current image and its proceeding image, return the matched key points pairs

2.2. estimate the fundamental matrix and the essential matrix

2.3. Calculate the locations of matched pairs (decomposition and triangulation)

2.4. call 'bundleAdjustment' function for global optimization, which employs Levenberg-Marquardt algorithm in this case

2.5. After this step, we get 4 sets of matched graphs
3. Iteratively merge graphs and product a refined graph
3.1. merge graphs
3.2. re-triangulation
3.3. bundle adjustment
3.4. remove outliers
3.5. bundle adjustment again
4. According to the parameter of the merge graph, adjust the focal length if necessary
5. Perform dense matching by calling 'denseMatch' function on each matched graph and corresponding frames
6. Dense reconstruction: step through each frame and perform the following operations
6.1. Pick Mot points of current frame and its proceeding frame from the merged graph
6.2. Using nonlinear method to estimate the 3D point from image matches and camera matrices
6.3. Reconstruct the merged graph using the 3D points information
7. There is blurb of code to visualize the dense point cloud. But it won't execute. Instead, the result of reconstruction is stored into a point cloud file, namely dense.ply.

Problem 3

What is the assumption for the principal points in this system? Under what condition will this assumption violated?

In this system, the principal points are assumed to be at the center of the images (px=py=0) or with minimum offset, so that the images are not nearly cropped. This will ensure the amount of overlap of the two adjacent frames. As long as we get enough matched features, the assumption can be held violated.

Problem 4

Field	Data	How data is organized	Purpose
.f	focal length	Single value, copied from the original frame parameter	To form intrinsic matrix
.Mot	Motion data	3 by 4 [R | t] matrices for each frame	To form extrinsic matrix
.Str	Structure of 3D points	3 by N matrix, N = the number of 3D points	To store 3D point cloud
.ObsVal	Value of observations	2 by matrix, N = the number of observations. For matched pairs, the matrix is 4 by N.	To calculate 3D positions
.ObsIdx	Index of KxN for N points observed by K cameras	Sparse storage	To calculate 3D positions

Why we need these 5 fields?

The 5 fields are parameters to reproject the matched points and apply triangulation to get their 3D positions.

Problem 5

Five pictures of a toy car were taken with iPhone 5s rear camera. The focal length extracted from EXIF is 2720. To keep the stability of camera, I put the toy on the center of a round plate, and moved the camera along the edge of the plate at each shot.

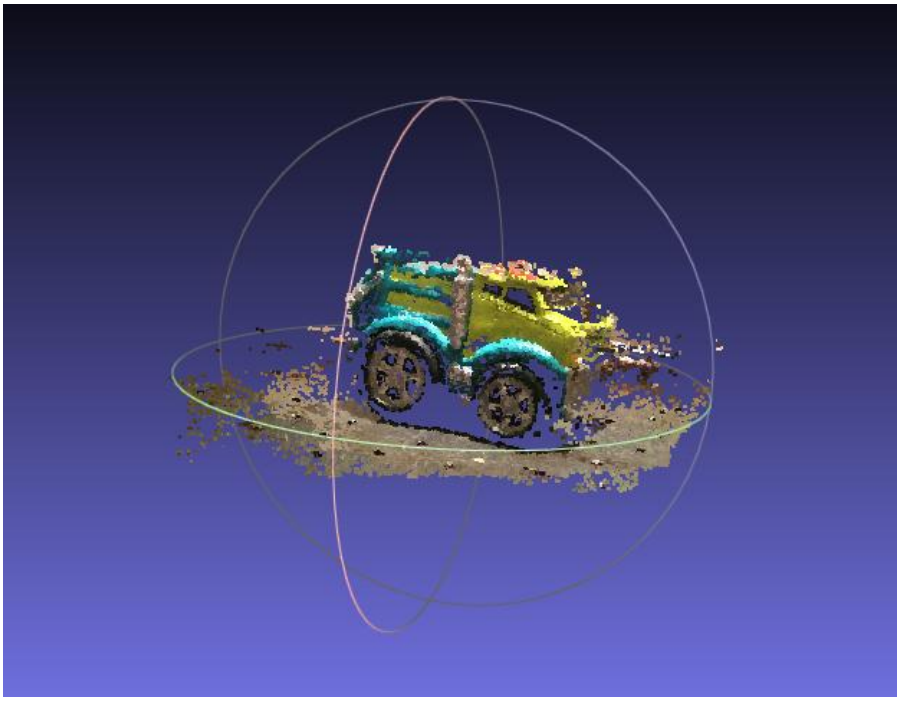
Besides the stability of picturing, I think the brightness also matters. So I turned on the pendant lighting to provide consistent background illumination, the flashlight of the camera was disabled. In order to produce more matched features between two adjacent images, the movement of camera was limited within 10 degree and I managed to keep the principle point at the center of images.

I think the most unstable step should be merging graphs, because the two graphs have different extrinsic matrix [R | t], which could introduce large reprojection errors and result in rejection of many outliers.

I found that the process of checking view angle in 'removeOutlierPts.m' rejected many matched points and led to the failure of reconstruction. After I skipped this code section, the result looked fair.

In order to take good pictures, one should keep the following rule of thumb in mind:

1. Choose proper object which contains sufficient features
2. Keep the camera steady
3. Put the object in the center of pictures
4. Try to maintain the same distance between the camera and the object for each picture



Problem 6

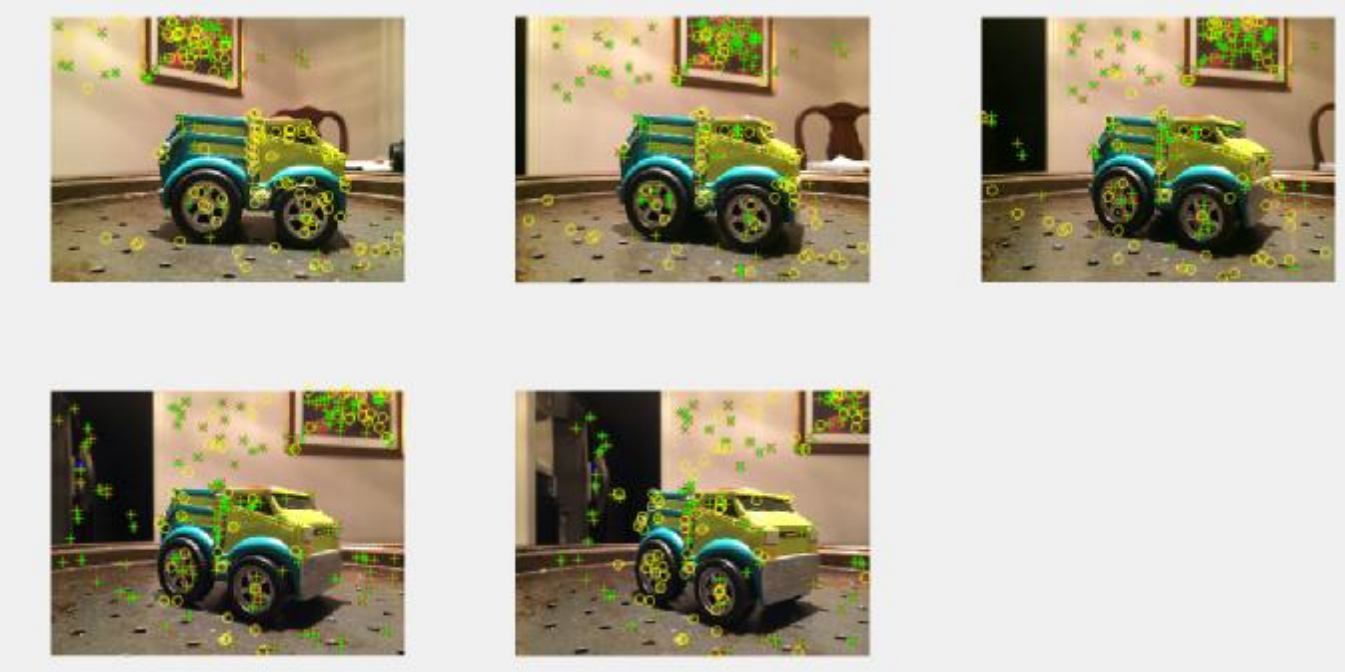
```
function printReprojectionError(graph)
% SFMedu: Structrue From Motion for Education Purpose
% This function is added by Baoshi Sun as a part of assignment work
```

```
% Print out the current reprojection error
% convert from Rt matrix to AngleAxis
nCam=length(graph.frames);
Mot = zeros(3,2,nCam);
for camera=1:nCam
    Mot(:,1,camera) = RotationMatrix2AngleAxis(graph.Mot(:,1:3,camera));
    Mot(:,2,camera) = graph.Mot(:,4,camera);
end

Str = graph.Str;
f = graph.f;

% assume px, py=0
px = 0;
py = 0;
residuals = reprojectionResidual(graph.ObsIdx, graph.ObsVal, px, py, f, Mot, Str);
fprintf('Reprojection error = %f\n', 2*sqrt(sum(residuals.^2)/length(residuals)));
```

Problem 7



```
function visualizeReprojection(graph, frames)
% SFMedu: Structrue From Motion for Education Purpose
% This function is added by Baoshi Sun as a part of assignment work
% Draw 3D keypoint point cloud proejected onto each images, as well as
% their observerd location.
figure
nCol = 3;
nRow = floor((frames.length - 1) / nCol + 1);

for frame=1:frames.length

    subplot(nRow, nCol, frame);
    image=imresize(imread(frames.images{frame}),frames.imsize(1:2));
    axis normal; axis off;
```

```
    % draw image
    imshow(image);

    % draw keypoints
    X = f2K(graph.f) * transformPtsByRt(graph.Str,graph.Mot(:,:,frame));
    xy = X(1:2,:) ./ X([3 3],:);    % 3D estimated location
    selector = find(graph.ObsIdx(frame,:)~=0);
    unselector = find(graph.ObsIdx(frame,:)==0);
    xyProjected = xy(:,selector);   % 3D estimated projected location on current
image
    xyNoProjected = xy(:,unselector);   % 3D estimated projected location not on
current image
    xyObs = graph.ObsVal(:,graph.ObsIdx(frame,selector));   % Observed location on
current image
    oxyObs = zeros(2,size(selector,2));
    oxyObs(1,:)= size(image,2)/2 - xyObs(1,:);
    oxyObs(2,:)= size(image,1)/2 - xyObs(2,:);
    oxyProjected = zeros(2,size(selector,2));
    oxyProjected(1,:)= size(image,2)/2 - xyProjected(1,:);
    oxyProjected(2,:)= size(image,1)/2 - xyProjected(2,:);
    oxyNoProjected = zeros(2,size(unselector,2));
    oxyNoProjected(1,:)= size(image,2)/2 - xyNoProjected(1,:);
    oxyNoProjected(2,:)= size(image,1)/2 - xyNoProjected(2,:);
    hold on;
    plot(oxyObs(1,:), oxyObs(2,:), 'rx');
    plot(oxyProjected(1,:), oxyProjected(2,:), 'g+');
    plot(oxyNoProjected(1,:), oxyNoProjected(2,:), 'yo');
    for ln=1:size(selector)
        line([oxyObs(1,ln) oxyProjected(1,ln)], [oxyObs(2,ln) oxyProjected(2,ln)],
'Color', 'b', 'LineStyle', '-');
    end
end
```

Problem 8

The lsqnonlin function is used to solve nonlinear lease-squares problems. The objective function we are optimizing for is to minimize the reprojection error. The equation can be expressed as:

$$\min \sum_i \sum_j \left(\tilde{x}_i^j - K[R_i|t_i]X^j\right)^2$$

Levenberg-Marquardt algorithm is suitable to the problems that are underdetermined (fewer equations than dimensions). However, since Levenberg-Marquardt algorithm needs to calculate partial derivatives, the computational cost will be very high if the problem has a big number of valuables. In this case, we can choose Powerll algorithm.

Problem 9

$$\min \sum_i (\tilde{x_i} - K[R_i|t_i]X)^2$$

```
% adjust motion [for homework]% !!! fill in your code here (Line 34)
```

```
[vec,resnorm,residuals,exitflag] = lsqnonlin(@(x)
reprojectionResidual(graph.ObsIdx,graph.ObsVal,px,py,f,x,Str), Mot(:),[],[],options);
```

Problem 10

$$\min \sum_j \left(\tilde{x}^j - K[R|t]X^j\right)^2$$

```
% adjust structure [for homework]% !!! fill in your code here (Line 30)
[vec,resnorm,residuals,exitflag] = lsqnonlin(@(x)
reprojectionResidual(graph.ObsIdx,graph.ObsVal,px,py,f,Mot,x), Str(:),[],[],options);
fprintf('error = %f\n', 2*sqrt(resnorm/length(residuals)));
```

Considering the large number of points, when this method is used to do triangulation, the computation cost could be very high.

Problem 11

It is doable to bypass the estimation of essential matrix and do the bundle adjustment directly if the scale of reconstruction is not large. However, we should notice the computation cost if there are many keypoints and frames.

If we can control the pose of camera and try to keep the angle and distance towards the object consistent, the estimation of essential matrix could be skipped. In this case, we can assume the rotation and transformation of each picture is almost the same. We just estimate one essential matrix and use it in all following triangulation. Then the bundle adjustment process will take care of the already suppressed residuals.

Furthermore, we can use a standard item, such as a unit cube, as the reconstruction object. Since we already know the actual length of each edge, a little bit geometric calculation can solve the triangulation and graph merging questions without essential matrix.