2D TO 3D VIDEO CONVERSION WITH STATIC SCENE AND HORIZONTAL CAMERA MOTION

A Design Project Report

Presented to the Engineering Division of Graduate School

Of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering(Electrical)

By

Ling-Wei Lee

Project Advisor: Tsuhan Chen

Degree Date: May, 2011

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: 2D to 3D Video Conversion with Static Scene and Horizontal Camera Motion

Author: Ling-Wei Lee

Abstract:

Four automate methods to convert two-dimensional (2D) video to three-dimensional (3D) for video with only static scene and horizontal camera motion have been proposed in this project. Our approaches try to generate the paired view for each frame in the video using the pixels in the other frames. In two of the methods, we use structure from motion to calculate the camera centers for all the frames and where their paired view suppose to be, and synthesize those frames by interpolation using the two real frames next to them. We also compare these methods with the other two naïve approaches that choosing the constant N frame after each frame to be its pair or picking the real frame that is closet to the virtual pair to be the paired view. The methods that synthesize the paired view by interpolation output promising result under our assumption of static scene and horizontal camera motion.

Report Approved by
Project Advisor:_____Date: _____

I. INTRODUCTION

A. Summary

In recent years, consumer and film making industry have started to put more attention on 3D films and more 3D videos have been made using stereo cameras. However, most videos are still filmed in 2D, and making those 2D videos into 3D in the post processing stage often require retrieving the depth manually, which is costly and inefficient. Although stereovision and depth perception retrieving for images have been studied in the field of computer vision for a long time, there is still no standard method to automate 2D to 3D conversion process. In this project, we try to research on automate method to convert 2D films into 3D for videos with static scene and horizontal camera motion as an initial approach to the problem. We have explored four methods to solve the problem using classic computer vision technique such as structure from motion, optical flow, and block matching algorithm for motion estimation and depth retrieving. The basic idea behind all the methods in this project is to use the information in the other frames of the video to synthesize the paired view for each frame in the video so that in the output, each frame will contain two views constructed by the original and its synthesized pair for left and right eyes. We started this project by implementing the first method; in this method, we synthesize the paired view for each frame by copying the frame, which is a constant N after the target frame. One can image synthesizing the paired in this way should introduce lots of error in the output when the camera does not move in constant speed. However, this method is a base line to compare with the other methods in this project. In the second, we try to improve the first one by choosing the paired view based on location instead of time (constant Nth frame after). To do so, we use Bundler, a structure from motion algorithm, to estimate the camera parameters for each frame in the video, and therefore, we can calculate where the paired view for each frame supposed to be. With the location calculated, we will pick the frame in the video that is closest to the paired view to be the synthesized view. In the third method, we try to synthesize the paired view by interpolating the two frames that is closest to the calculated paired view location at the location of the paired view. We calculate the motion vector, by fixed sized block matching algorithm, between the two frames, and then weight the motion vector

according to the relative distance between the paired view location and the real frames, and finally, warp one of the real frames using the motion vector. In the fourth method, instead of interpolating the paired view using motion vector, we compute the optical flow between the two frames. Although the scope of this project is restricted to only videos with static scene and horizontal camera motion, we believe that this project can be the first step for future researches on 2D to 3D video conversion methods.

B. Design problem and issues

The motivation of this project is to explore ways to convert 2D video of any kind to 3D since most of the videos are still filmed in 2D and 3D videos are more interesting to watch. One can imagine that this is not a trivial task, since we miss the information for the third dimension in 2D video. Theoretically, such algorithm should perform perfect conversion if it some how retrieves the depth information for each frame in the video. Although this kind of problem has been research in the field of computer vision for a long time, there is still no standard way to solve the problem for every kind of camera motion and scene, which is often complicated in the movies. However, the problem can be further divided in to different sub-problems, since different kinds of scene have different difficulty to solve the problem. Scenes with moving objects are hard to solve, since it is hard to predict the motion of the object in relative to the camera. On the other hand, static scene seems to be approachable, since we can estimate the depth by the camera motion. Complicate camera motion also makes the difficulty higher because, we might not have enough information from the video to retrieve depth or synthesize views. Therefore, with the time constrain of this project, the scope of this project is only going to be the video with static scene and horizontal camera motion. The main issues under these assumptions become that how to synthesize the paired view for each frame in the video so that the viewer has the depth perception and does not feel discontinuity while watching the video. The requirements of the system are as follows:

- The algorithm should convert the 2D video of static scene and horizontal camera motion into 3D.
- The viewer should feel that the output is similar to the 3D video filmed by stereo camera.

- The output 3D video should represent content of the original 2D video without much distortion.

II. METHOD

Each frame in the 3D videos contains different views for left and right eyes. The two images are often synthesized together to generate anaglyph, which can be viewed using polarized 3D glasses so that each eye sees different views. Therefore, in order to generate the 3D video, the algorithm has to generate two views for each frame in the output. In this project, we implemented four methods to convert 2D videos of static scene and one dimensional camera motion to 3D. The general idea behind all the methods is that for each frame in the video, each method tries to create its stereo-pair frame, called virtual frame in this project, using data in other frames, where the original frame and the created frame are used to mimic the output of stereo camera. The difference between the methods implemented in this project is the way to synthesize virtual frames. Below are the outlines of each method:

A. Pick N frame after the original

In this method, virtual frames are generated by picking the N^{th} frame after each frame, where N is a parameter depends on the video input. For example, if we denote the frames in the image by 1, 2…n, and choose N to be 3, in the output video, frame 1 will have two views: the original frame 1 and frame 4. Frame 2 will be constructed by the original frame 2 and frame 5, and so on. After having the two views, we make the anaglyph using the two views as shown in figure 1. One can anticipant that this method will work well if the camera moves only in one direction and in constant speed, since the real distance between consecutive frames varies depends on the speed of camera motion This simple approach is considered the base line to compare with other methods.

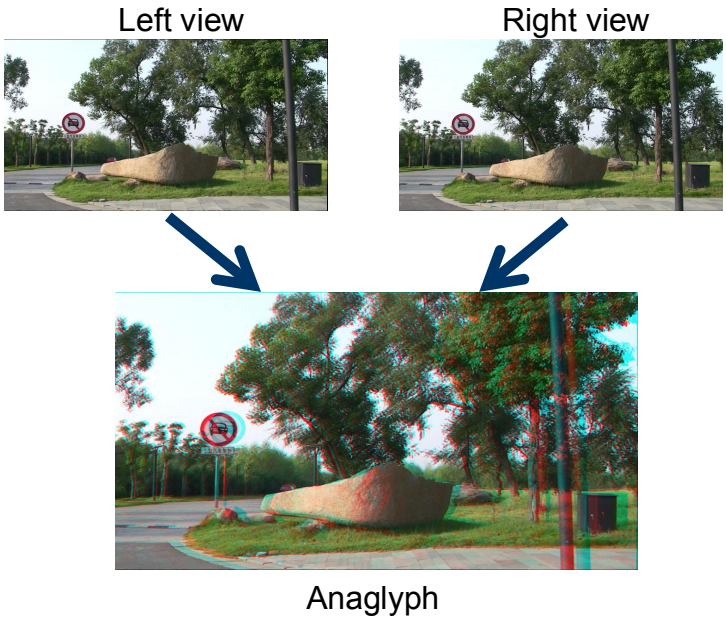

Figure 1, frames in 3D videos contains two views for each eye

B. Pick the frame that is the closet to the virtual frame location

Since the method that picks constant N frame after each frame has poor performance with camera moving in non-constant speed, we will improve the result by picking the frame based on the location of the virtual frames instead of time (constant N frames after). Structure from motion algorithm is applied here to estimate camera translation and rotation in order to calculate camera center location for each frame. The implementation uses Bundler to run structure from motion, which outputs rotation matrix and translation vector for each frame. [1] Given this information we can calculate the location of camera center for each frame by

$$-R^T \cdot t$$

where R is the 3 by 3 matrix represents the rotation of the camera and t is a 3 by 1 translation vector. Both R and t are estimated by Bundler.

After we know the camera center for each real frame, we can also calculate the camera center of its pair (virtual frame) with a parameter, which indicates the distance between

each real frame and its pair. The camera center of the virtual frame for each real frame can be calculated by:

$$-R^T \cdot (d - t)$$

where R and t are the same as above, and d is a 3 by 1 vector represents the relative distance in each axis between the two views. Since we only deal with one-dimensional movement here, d has only one non-zero entry; d is also the same for every frame.

Given the camera center of each frame and virtual frame, in this method, we synthesize the virtual frames simply by picking the real frame that is closest to the virtual frame. For each frame, we compute its distance to every other frame, and then search for the frame that has the minimum distance to the location of the paired view. Therefore, the two views for each frame in the output are the original frame, and the frame that has the shortest distance to the calculated location of the paired view.

C. Synthesize virtual frames with motion vector

Always using other real frames to as the virtual frames will return poor result if the camera is moving fast in relative to the sampling rate of the video because the actual location of the virtual is often in between the real frames. Therefore, it is reasonable to synthesize the virtual frame by interpolation using the two real frames that are in the two sides of the virtual frame. Here we will also use structure from motion to calculate the camera centers for each frame and its corresponding virtual frame. Given the locations of the virtual frame and the two real frames next to it, we interpolate the virtual frame by first, finding the motion vector between the two real frames with fix-sized block matching algorithm, and second, weight the motion vector according the relative location between the virtual frame and the real frames next to it, and finally, warp one of the real frame using the weighted motion vector.

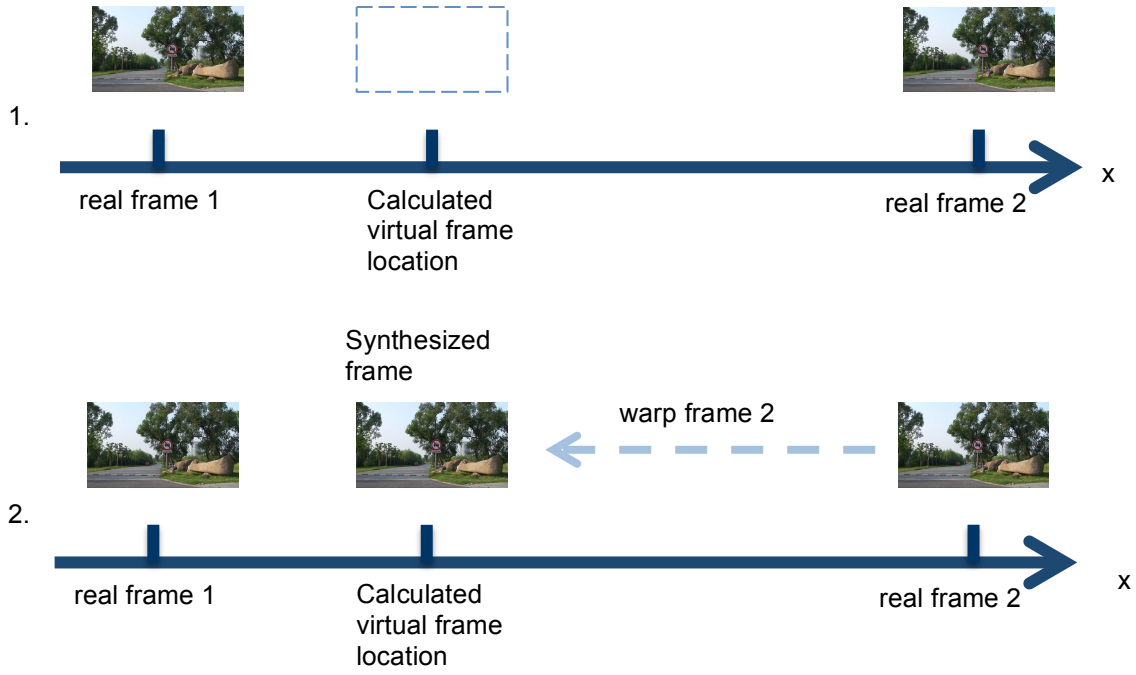

figure 2, synthesize the virtual frame by interpolation

The fixed-sized block matching algorithm is a method that compares the motions in two images and is easy to implement. Each image frame is divided into a fixed number of square blocks. For each block in the frame, a search is made in the reference frame over an area of the image, where the maximum search area is determined by a parameter. The search is to find the best matching block that gives the least error. In this project, we define the error to be the mean absolute difference which is much less computational expensive than other error scheme. The output of the fixed –sized block matching is often noisy and contains the blocky feature of the size of the input block size. Therefore, we also use median filter on the block map to reduce the noise, and then use Gaussian filter on the whole image to reduce the blockyness.

Figure 3, left: motion vector map, right: image synthesized using motion vector map

D. Synthesize virtual frames with optical flow

This method is similar to the one above, the only difference is that, here, we use optical flow instead of motion vector to estimate the motion between two real frames when synthesizing the virtual frame by interpolation. We adopt the implementation of optical flow from. [2]

Figure 4, left: optical flow map, right: image synthesized using optical flow

E. Stereo pair alignment

Although we assume only one-dimensional camera motion in this project, the input video often contain some motion in the vertical direction. One easy fix to make the result looks better is to align the pair (real frame and the synthesized virtual frame) by using SIFT feature points to determine the correspondence between the pairs, and then translate one frame so that the two frames are at the same height.

Figure 5, left: output 3D image before stereo pair alignment, right: after alignment

III. RESULTS

In this section, we will examine the result for each method under different conditions:

A. Pick N frame after the original

This method performs well with videos that have high sampling rate and move in constant speed as expected (see figure ()). This is because that in those videos the two views of each frame are apart by a constant in time (frame number), and under the condition of high sampling rate and constant speed, the two views are also apart by a constant in distance. Therefore the frame it choses represents the virtual frame well. On the other hand, if the camera does not move in constant speed, the distance between the views of two eyes will not be the same for different frames in the output as show in figure (). The outcome of this effect is that, the viewer will have distorted depth perception when viewing the 3D video, and if the error is too high, the viewer would feel that the video does not move smoothly.

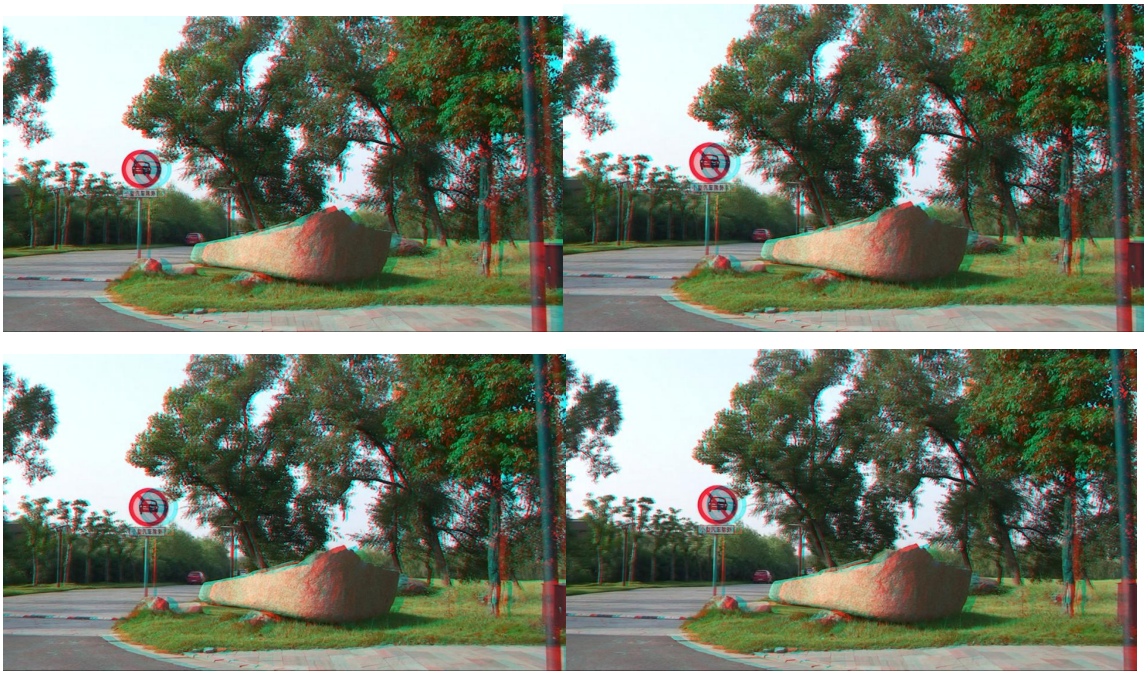

Figure 6, four consecutive frames in the output of method A using high sampling rate data set. Notice that the distance between the two views are the same for each frame.

Figure 7, four consecutive frames in the output of method A using lower sampling rate data set. Notice that the distance between the two views changes a lot for different frames frame.

B. Pick the frame that is the closet to the virtual frame location

This method performs well when the sampling rate is high, and even when the camera does not move in constant speed. However, when the sampling rate is low, it has the same problem as above, since the location of virtual frame is too far from the closest real frame.

C. Synthesize virtual frames with motion vector

In this method, distances between the two views in each frame in the output are approximately the same because of interpolation. This means the viewers feel that the camera motion is as smooth as the original 2D video. However, the motion vector map

calculated by block matching algorithm is often noisy as shown in figure (). Therefore, the viewer will sense some distortion of the structures in the scene in the output video.

D. Synthesize virtual frames with optical flow

This method has the best performance among all the methods implemented in this project. The distance between the two views are approximately the same for each frame and the motion map estimated by optical flow is not as noisy as the block matching algorithm output. If our assumption in this project is preserved, this method outputs promising results.

IV. CONCLUSION

In this project, we have implemented four methods to convert 2D videos to 3D for videos with only static scene and horizontal camera motion. The output in the first two approaches often has errors so that the viewer feels as if the video motion is not smooth. This is because in the output of first two methods, the distance between two views very too much. With interpolation using either motion vectors or optical flow this kind of error is minimized; however, the motion vector map calculated by block matching algorithm often contains lots of noise and unwanted blackness, so the views will see many artifacts in the output of the third method. The fourth method which use optical flow to interpolate the synthesized view outputs promising result if the input video is under the assumption of static scene and horizontal camera motion. Although the scope of this project is restricted to only videos with static scene and horizontal camera motion, we believe that this project can be the first step for future researches on 2D to 3D video conversion methods.

V. ACKNOWLEDGMENTS

We are grateful for advice and support from Professor Tsuhan Chen, Dr. Yao-Jen Chang, and Adarsh Kowdle at Advanced Multimedia Processing Laboratory, Cornell University.

VI. REFERENCES

[1] Noah Snavely, Steven M. Seitz, Richard Szeliski. "Modeling the World from Internet Photo Collections". *International Journal of Computer Vision*, 2007.

[2] C. Liu. **"**Beyond Pixels: Exploring New Representations and Applications for Motion Analysis"**.** *Doctoral Thesis*. Massachusetts Institute of Technology. May 2009.

[3] Guofeng Zhang, Jiaya Jia, Tien-Tsin Wong and Hujun Bao. "Consistent Depth Maps Recovery from a Video Sequence." *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(6):974-988, 2009.

VII. APPENDIX

to3d_constant.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
% 2d to 3d conversion using constant method
%
%   change following field for input
%
%   im_dir - the directory of image set%
%   out_dir - directory to save the output images and data
%   constant - the constant N after
%   number - number of frames
%
%
% Ling-Wei Lee
% 05-19-2011
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

im_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/road';
out_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/out2_of';
constant = 3 ;
number=141;

im_list = dir([im_dir '/*.jpg']);
n=number-constant;


for i =1:n
    tempstr1 = sprintf('%s/%s%04d.jpg', im_dir, im_list(n).name(1:end-
8),i-1);
    tempstr2 = sprintf('%s/%s%04d.jpg', im_dir, im_list(n).name(1:end-
8),i+constant-1);
    tempstr3 = sprintf('%s/out_%04d.jpg', out_dir, i-1);
    anaglyph(tempstr1,tempstr2,tempstr3)
end
```

to3d_mindist.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
% 2d to 3d conversion using closest distance method
%
%   change following field for input
%
%   im_dir - the directory of image set%
%   out_dir - directory to save the output images and data
%   datafile - outout of bundler (bundle.out)%
%   number - number of frames
%
%
% Ling-Wei Lee
% 05-19-2011
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

im_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/road';
out_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/out2_of';
datafile = '/Users/ling-weilee/workspace/mexOpticalFlow/cameras.txt';
number=141;


fbundle = fopen(datafile);
im_list = dir([im_dir '/*.jpg']);
data = {};
b_temp = textscan(fbundle, '%f %f %f',5, 'headerLines', 2);




camcenter = zeros(number,3);
vcamcenter = zeros(number,3);

for n=1:number


   data{n,1} = i;
   %tmpstr=sprintf('%s/%s', im_dir, im_list(n).name);
   %img = imread(tmpstr);
   %h = size(img,1); w = size(img,2);
   data{n,2} = [b_temp{1}(2:end-1) b_temp{2}(2:end-1) b_temp{3}(2:end-
1)]; %rotation
   data{n,3} = [b_temp{1}(end); b_temp{2}(end); b_temp{3}(end)];
%translation
   data{n,4} = [-b_temp{1}(1),0,0.5*w;0,b_temp{1}(1),0.5*h;0,0,1 ];%K
matrix
   data{n,5} = -data{n,2}'*data{n,3}; %camera center
   data{n,6} = data{n,2}'*([relative_dist; 0; 0] - data{n,3}); %camera
center of pair
   data{n,8} = sqrt(sum((data{n,6}-data{n,5}).^2)); %dist btw pair
```

```
   b_temp = textscan(fbundle, '%f %f %f',5);

   camcenter(n,:) = data{n,5}';
   vcamcenter(n,:) = data{n,6}';



end




n=n-5;

for i=1:n
   mdis = sqrt(sum((data{i,6}-data{i,5}).^2));
   pairid = i;

   for j = 1:number
       m =  sqrt(sum((data{i,6}-data{j,5}).^2));
       if m<mdis
            mdis = m;
            pairid = j;
       end

   end

   data{i,7} = pairid;
end




for i =1:n
    tempstr1 = sprintf('%s/%s', im_dir, im_list(i).name);
    tempstr2 = sprintf('%s/%s', im_dir, im_list(data{i,7}).name);
    tempstr3 = sprintf('%s/out_%s.jpg', out_dir, im_list(i).name(end-
7:end-4));
    anaglyph(tempstr1,tempstr2,tempstr3)
end
```

to3d_bma.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
% 2d to 3d conversion using block matching
%
%   change following field for input
%   im_dir - the directory of image set
%   pair_dir - the directory to save the synthesized image pairs
%   out_dir - directory to save the output images and data
%   datafile - outout of bundler (bundle.out)
%   relative_dist - the baseline of the distance between pairs
%   number - number of frames
%
%
% Ling-Wei Lee
% 05-19-2011
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%




im_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/road';
pair_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/pair2_of';
out_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/out2_of';
datafile = '/Users/ling-weilee/workspace/mexOpticalFlow/cameras.txt';
relative_dist = 3 ;
number=141;


fbundle = fopen(datafile);
im_list = dir([im_dir '/*.jpg']);
data = {};
b_temp = textscan(fbundle, '%f %f %f',5, 'headerLines', 2);







camcenter = zeros(number,3);
vcamcenter = zeros(number,3);

for n=1:number


   data{n,1} = i;
   %tmpstr=sprintf('%s/%s', im_dir, im_list(n).name);
   %img = imread(tmpstr);
   %h = size(img,1); w = size(img,2);
   data{n,2} = [b_temp{1}(2:end-1) b_temp{2}(2:end-1) b_temp{3}(2:end-
1)]; %rotation
```

```
   data{n,3} = [b_temp{1}(end); b_temp{2}(end); b_temp{3}(end)];
%translation
   data{n,4} = [-b_temp{1}(1),0,0.5*w;0,b_temp{1}(1),0.5*h;0,0,1 ];%K
matrix
   data{n,5} = -data{n,2}'*data{n,3}; %camera center
   data{n,6} = data{n,2}'*([relative_dist; 0; 0] - data{n,3}); %camera
center of pair
   data{n,8} = sqrt(sum((data{n,6}-data{n,5}).^2)); %dist btw pair

   b_temp = textscan(fbundle, '%f %f %f',5);

   camcenter(n,:) = data{n,5}';
   vcamcenter(n,:) = data{n,6}';



end




n=n-5;

for i=1:n
   mdis = sqrt((data{i,6}(1)-data{i,5}(1)).^2);
   weight = 0.5;
   pairid = [i i+1 weight];

   for j = 1:number
       m =  sqrt(((data{i,6}(1)-data{j,5}(1)).^2));
       if m<mdis
            mdis = m;

            if sqrt(((data{i,6}(1)-data{j+1,5}(1)).^2))
>sqrt(((data{j,5}(1)-data{j+1,5}(1)).^2))

                weight = sqrt(((data{i,6}(1)-data{j,5}(1)).^2)) /
sqrt(((data{j,5}(1)-data{j-1,5}(1)).^2));


                pairid = [j-1 j weight];

            else

                weight = sqrt(((data{i,6}(1)-data{j+1,5}(1)).^2)) /
sqrt(((data{j,5}(1)-data{j+1,5}(1)).^2));


                pairid = [j j+1 weight];

            end
       end

   end
```

```
   data{i,7} = pairid;

   intercamcenter(i,:) = [data{pairid(2),5}(1)-(data{pairid(2),5}(1)-
data{pairid(1),5}(1))*weight,
                          data{pairid(2),5}(2)-(data{pairid(2),5}(2)-
data{pairid(1),5}(2))*weight,
                          data{pairid(2),5}(3)-(data{pairid(2),5}(3)-
data{pairid(1),5}(3))*weight];
end


fig=figure(1);
set(gca,'position',[0 0 1 1],'units','normalized')
hold on
scatter3(camcenter(:,1),camcenter(:,2),camcenter(:,3),8,'blue','filled'
);
scatter3(vcamcenter(:,1),vcamcenter(:,2),vcamcenter(:,3),8,'green','fil
led');
scatter3(intercamcenter(:,1),intercamcenter(:,2),intercamcenter(:,3),8,
'red','filled');

for i=1:size(camcenter,1)
   text(camcenter(i,1),camcenter(i,2),camcenter(i,3),['\color{blue}'
int2str(i)])

text(vcamcenter(i,1),vcamcenter(i,2),vcamcenter(i,3),['\color{green}'
int2str(i)])


end

for i=1:size(intercamcenter,1)

text(intercamcenter(i,1),intercamcenter(i,2),intercamcenter(i,3),['\col
or{red}' int2str(i)])

end


hold off
grid on
axis equal
xlabel('x')
ylabel('y')
zlabel('z')
tmpstr=sprintf('%s/location.m', out_dir);
saveas(fig,tmpstr)


for i=1:n
   i
   r=data{i,7}(1);
   l=data{i,7}(2);
```

```
   im_list(i).name
   im_list(r).name
   im_list(l).name

   tmpstr1=sprintf('%s/%s', im_dir, im_list(r).name);
   tmpstr2=sprintf('%s/%s', im_dir, im_list(l).name);
   im1= imread(tmpstr1);
   im2= imread(tmpstr2);
   [d, vy, vx, v, h] = bma(im1, im2, 16, 32);

   tmpstr6=sprintf('%s/mv_%s.mat',out_dir, im_list(i).name(end-7:end-
4));
   save(tmpstr6,'vx','vy','r','l');

   tmpstr7=sprintf('%s/flowmap_%s.jpg', out_dir, im_list(i).name(end-
7:end-4));
   flow(:,:,1) = vx;
   flow(:,:,2) = vy;
   imflow = flowToColor(flow);
   imwrite(imflow,tmpstr7);

   pairim=warpFLColor(im1,im2,vx.*data{i,7}(3),vy.*data{i,7}(3));

   tmpstr3=sprintf('%s/pair_%s.jpg', pair_dir, im_list(i).name(end-
7:end-4));
   imwrite(pairim, tmpstr3);



   tmpstr4=sprintf('%s/%s', im_dir, im_list(i).name);
   tmpstr5=sprintf('%s/out_%s.jpg', out_dir, im_list(i).name(end-7:end-
4));
   anaglyph(tmpstr4,tmpstr3,tmpstr5)
end
```

to3d_of.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
% 2d to 3d conversion using optical flow
%
%   change following field for input
%   im_dir - the directory of image set
%   pair_dir - the directory to save the synthesized image pairs
%   out_dir - directory to save the output images and data
%   datafile - outout of bundler (bundle.out)
%   relative_dist - the baseline of the distance between pairs
%   number - number of frames
%
%
% Ling-Wei Lee
% 05-19-2011
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

im_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/road';
pair_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/pair2_of';
out_dir = '/Users/ling-weilee/workspace/mexOpticalFlow/out2_of';
datafile = '/Users/ling-weilee/workspace/mexOpticalFlow/cameras.txt';
relative_dist = 3 ;
number=141;


fbundle = fopen(datafile);
im_list = dir([im_dir '/*.jpg']);
data = {};
b_temp = textscan(fbundle, '%f %f %f',5, 'headerLines', 2);




camcenter = zeros(number,3);
vcamcenter = zeros(number,3);

for n=1:number


   data{n,1} = i;
   %tmpstr=sprintf('%s/%s', im_dir, im_list(n).name);
   %img = imread(tmpstr);
   %h = size(img,1); w = size(img,2);
   data{n,2} = [b_temp{1}(2:end-1) b_temp{2}(2:end-1) b_temp{3}(2:end-
1)]; %rotation
   data{n,3} = [b_temp{1}(end); b_temp{2}(end); b_temp{3}(end)];
%translation
   data{n,4} = [-b_temp{1}(1),0,0.5*w;0,b_temp{1}(1),0.5*h;0,0,1 ];%K
matrix
   data{n,5} = -data{n,2}'*data{n,3}; %camera center
```

```
   data{n,6} = data{n,2}'*([relative_dist; 0; 0] - data{n,3}); %camera
center of pair
   data{n,8} = sqrt(sum((data{n,6}-data{n,5}).^2)); %dist btw pair

   b_temp = textscan(fbundle, '%f %f %f',5);

   camcenter(n,:) = data{n,5}';
   vcamcenter(n,:) = data{n,6}';



end




n=n-5;

for i=1:n
   mdis = sqrt((data{i,6}(1)-data{i,5}(1)).^2);
   weight = 0.5;
   pairid = [i i+1 weight];

   for j = 1:number
       m =  sqrt(((data{i,6}(1)-data{j,5}(1)).^2));
       if m<mdis
            mdis = m;

            if sqrt(((data{i,6}(1)-data{j+1,5}(1)).^2))
>sqrt(((data{j,5}(1)-data{j+1,5}(1)).^2))

                weight = sqrt(((data{i,6}(1)-data{j,5}(1)).^2)) /
sqrt(((data{j,5}(1)-data{j-1,5}(1)).^2));


                pairid = [j-1 j weight];

            else

                weight = sqrt(((data{i,6}(1)-data{j+1,5}(1)).^2)) /
sqrt(((data{j,5}(1)-data{j+1,5}(1)).^2));


                pairid = [j j+1 weight];

            end
       end

   end

   data{i,7} = pairid;

   intercamcenter(i,:) = [data{pairid(2),5}(1)-(data{pairid(2),5}(1)-
data{pairid(1),5}(1))*weight,
```

```
                          data{pairid(2),5}(2)-(data{pairid(2),5}(2)-
data{pairid(1),5}(2))*weight,
                          data{pairid(2),5}(3)-(data{pairid(2),5}(3)-
data{pairid(1),5}(3))*weight];
end



fig=figure(1);
set(gca,'position',[0 0 1 1],'units','normalized')
hold on
scatter3(camcenter(:,1),camcenter(:,2),camcenter(:,3),8,'blue','filled'
);
scatter3(vcamcenter(:,1),vcamcenter(:,2),vcamcenter(:,3),8,'green','fil
led');
scatter3(intercamcenter(:,1),intercamcenter(:,2),intercamcenter(:,3),8,
'red','filled');

for i=1:size(camcenter,1)
   text(camcenter(i,1),camcenter(i,2),camcenter(i,3),['\color{blue}'
int2str(i)])

text(vcamcenter(i,1),vcamcenter(i,2),vcamcenter(i,3),['\color{green}'
int2str(i)])



end

for i=1:size(intercamcenter,1)

text(intercamcenter(i,1),intercamcenter(i,2),intercamcenter(i,3),['\col
or{red}' int2str(i)])

end



hold off
grid on
axis equal
xlabel('x')
ylabel('y')
zlabel('z')
tmpstr=sprintf('%s/location.m', out_dir);
saveas(fig,tmpstr)



for i=1:n
   i
   r=data{i,7}(1);
   l=data{i,7}(2);



   im_list(i).name
   im_list(r).name
   im_list(l).name
```

```
   tmpstr1=sprintf('%s/%s', im_dir, im_list(r).name);
   tmpstr2=sprintf('%s/%s', im_dir, im_list(l).name);
   im1= imread(tmpstr1);
   im2= imread(tmpstr2);
   [vx, vy, warpI2]=Coarse2FineTwoFrames(im1,im2);

   tmpstr6=sprintf('%s/flow_%s.mat',out_dir, im_list(i).name(end-7:end-
4));
   save(tmpstr6,'vx','vy','r','l');

   tmpstr7=sprintf('%s/flowmap_%s.jpg', out_dir, im_list(i).name(end-
7:end-4));
   flow(:,:,1) = vx;
   flow(:,:,2) = vy;
   imflow = flowToColor(flow);
   imwrite(imflow,tmpstr7);

   pairim=warpFLColor(im1,im2,vx.*data{i,7}(3),vy.*data{i,7}(3));

   tmpstr3=sprintf('%s/pair_%s.jpg', pair_dir, im_list(i).name(end-
7:end-4));
   imwrite(pairim, tmpstr3);



   tmpstr4=sprintf('%s/%s', im_dir, im_list(i).name);
   tmpstr5=sprintf('%s/out_%s.jpg', out_dir, im_list(i).name(end-7:end-
4));
   anaglyph(tmpstr4,tmpstr3,tmpstr5)
end
```

bma.m

```
function  [d, vvector, hvector, v, h] = bma(frame, frame_s, blocksize,
displacement)

m= size(frame,1);
n= size(frame,2);

a = floor(m/blocksize);
b = floor(n/blocksize);

v=zeros(a,b);
h=zeros(a,b);


hvector = zeros(m,n);
vvector = zeros(m,n);
d = zeros(m,n);


for i = 1:a
    for j = 1:b


        ind = [(i-1)*blocksize+1 (j-1)*blocksize+1];

        m1 = frame(ind(1):ind(1)+blocksize-1,ind(2):ind(2)+blocksize-
1,: );
        m2 = frame_s(ind(1):ind(1)+blocksize-1,ind(2):ind(2)+blocksize-
1,: );



        %err = sum(reshape(mean(mean((double(m1) -
double(m2)).^2,2),1),[1,3]));
        err = sum(sum(sum(abs(double(m1)-double(m2)))));
        mind = [ind(1) ind(2)];

        for s = -displacement:1:displacement
            for k = -displacement:1:displacement

                m2 = zeros(blocksize, blocksize);
                up = ind(1)+s;
                down = ind(1)+s+blocksize-1;
                left = ind(2)+k;
                right = ind(2)+k+blocksize-1;

                if (up>1 && down <m && left>1 && right <n)

                    m2 = frame_s(up:down,left:right ,:);

                    tmp = sum(sum(sum(abs(double(m1)-double(m2)))));
                    %tmp = sum(reshape(mean(mean((double(m1) -
```

```
double(m2)).^2,2),1),[1,3]));

                    if tmp<err
                        err = tmp;
                        mind = [up left];
                    end

                end

            end

        end




        ind = [(i-1)*blocksize+1 (j-1)*blocksize+1];


        v(i,j) = mind(1)-ind(1);

        h(i,j) = mind(2)-ind(2);


    end
end

 v=medfilt2(v);
 h=medfilt2(h);

for i = 1:a
    for j = 1:b


        ind = [(i-1)*blocksize+1 (j-1)*blocksize+1];

        vvector(ind(1):ind(1)+blocksize-1,ind(2):ind(2)+blocksize-1 )
=...
                            v(i,j);

        hvector(ind(1):ind(1)+blocksize-1,ind(2):ind(2)+blocksize-1 )
=...
                            h(i,j);



    end
end


H=fspecial('gaussian',blocksize,blocksize/2);
vvector=imfilter(vvector,H,'replicate');
hvector=imfilter(hvector,H,'replicate');
```

```
d = (vvector.^2+hvector.^2).^1/2;
```