Multiple Camera Tracking

Helmy Eltoukhy and Khaled Salama
Stanford image sensors group
Electrical Engineering Department, Stanford University

Tracking of humans or objects within a scene has been studied extensively. We present a multiple camera system for object tracking. The system employs uncalibrated cameras and depends on the motion-tracking algorithm to achieve both point correspondence and image registration. The system is capable of switching between different cameras to achieve the best tracking conditions of the object. A simple communication scheme is established between the cameras in order to achieve both robust tracking and occlusion prevention.

1. Introduction

Recently multiple sensor environment gained interest within academia. The problem of having multiple sensors covering a relatively small environment and interacting with each other is of huge interest due to great number of problems that arise which spans different areas of research. Those sensor networks provide redundant information due to their large number and their overlap that can be used as a means to implement robust algorithms or to achieve better performance. Those sensors can be biological, chemical, temperature, pressure, audio, image or video sensors. They should be able to communicate with each other wirelessly so as to minimized the infra structure needed for their deployment.

Object tracking in a video sequence has been studied extensively for surveillance purposes. The ability to detect an object and then later tracking is of great interest in many applications like security, missile tracking and rescue operations. The traditional tracking algorithms are designed either to track a single object within the field of view of the camera or to detect multiple objects within the same scene and both cases depends on using a single camera. Recently there has been some work in multiple camera environments, in which an array of cameras are used to image the same scene. The main application for such a system is for 3D modeling of objects like in light fields. It was also

used to track multiple objects in the scene such that each camera is dedicated to an object. Only recently people started to consider using multiple cameras to track a single object in the scene. This has the advantage of having more information about the object being tracked.

In this project we present a system in which two cameras at 180 degrees are used to track an object in the scene. This provides more robust tracking since the camera system is capable of switching between the 2 camera view for better tracking of the object if it is outside the field of view of one of them or if it is occluded by an object. Ideally the system consists of three main parts:

a) Object tracking: In which each camera is tracking the object independently and producing an estimate of its position with some kind of an error measure
b) Real time communication: each camera should be able to send the position and error information to a central node to be processed
c) Point correspondence: a central node should be able to achieve point correspondence between the two cameras so as to confirm the position of the object and based on the make some decision on what action to take. This is done by building a model and using calibrated points to estimate the parameters

In reality we had to simplify the system so as to be easily done with the class project period. The system currently looks like the following:

a) Object tracking: a two-camera system is used to capture 2 sequences of the scene that are later are processed independently to track the object. The output of this stage is a matrix which includes the estimated object position and an error measurement for each scene within the sequence
b) Communication: The matrices obtained from the previous stage are operated in sequence in order to simulate the communication part between cameras and central node
c) Point correspondence: After trying different models for the scene it was apparent that the accuracy of this registration problem is really hard and it can not be trusted to give accurate results. So we switched to a dynamic real-time point correspondence that depends on the tracking algorithm. And based on that we can access the exact location of the object and determine which view is better to see.

2. Optical Flow and Feature Tracking

The brightness constancy assumption is vital to the successful implementation of correlation or gradient-based optical flow estimation algorithms, i.e., $\psi(\mathbf{x}_{k+1}, t+\Delta t) = \psi(\mathbf{x}_k, t)$, where $\psi(\mathbf{x}, t)$ is the frame intensity at location $\mathbf{x}$ and time t. Accordingly, all methods discussed herein make ample use of this assumption. . First of all, any thriving feature tracking algorithm must be predicated upon a reliable optical flow estimation algorithm which is simple enough to be implemented at 30 frames per second or higher. Such stringent requirements preclude the use of computationally intensive methods such as that of Fleet and Jepson. Furthermore, since it is well-known that gradient-based methods, such as Lucas-Kanade, are fairly accurate when applied to subpixel optical flow estimation, as well as computationally tractable, a logical first step is to explore the feature tracking scheme proposed by Shi and Tomasi.

2.1 Shi and Tomasi Feature Tracking

This algorithm employed the use of Lucas-Kanade on carefully chosen "corner" points. Intuitively, it is clear that good features constitute those with large spatial gradients in two orthogonal directions. Since Lucas-Kanade involves solving the optical flow equation iteratively assuming the displacement is characterized by constant velocity as given by,

$$\begin{bmatrix} \left(\frac{\partial s}{\partial x}\right)^2 & \frac{\partial s}{\partial y}\cdot\frac{\partial s}{\partial x} \\ \frac{\partial s}{\partial x}\cdot\frac{\partial s}{\partial y} & \left(\frac{\partial s}{\partial y}\right)^2 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \frac{\partial s}{\partial t}\cdot\frac{\partial s}{\partial x} \\ \frac{\partial s}{\partial t}\cdot\frac{\partial s}{\partial y} \end{bmatrix},$$

the two by two spatial gradient matrix can be used to determine the quality of each possible corner point, where the gradients are summed across an n x n block. Tomasi, et al. suggests that a reasonable criterion for feature selection is for the minimum eigenvalue of the spatial gradient matrix to be no less than some λ. This ensures that the matrix is well conditioned and above the noise level of the image so that its inverse does not unreasonably amplify possible noise in a certain critical direction, i.e., there are sufficient

spatial gradients in two orthogonal directions. Once these features are located, the Lucas-Kanade optical flow algorithm can be applied.

2.2 Using Lucas Kanade for optical flow estimation

Although ostensibly straightforward, the equation given above can be set up in a myriad of different ways. For instance, the number of taps for both the spatial and gradient derivatives, the windowing function for the spatial gradient and the block size are all unspecified. Rather than reinventing the wheel, we refer to a close variant of the implementation used by Fleet, et al. Thus, the spatial derivative vector used is 1/12[-1 8 0 -8 1]. This filter is used in each dimension and acts upon the sum of both images in order increase the accuracy of the solution through smoothing of the spatial gradients. Furthermore, in addition to using a simple 2 tap temporal derivative filter, spatio-temporal Gaussian smoothing is applied initially to enhance the accuracy of the velocity estimation by removing possibly "distracting" high frequency components.

2.3 Object Tracking via Lucas Kanade

The previous two results were then combined in order to track objects of interest in the scene. A synthetic sequence of a moving black cross on a white background was generated so that groundtruth data would be available. First, user initialization was required in order to manually indicate the object to be tracked. Then the n x n blocks in the region of interest which met the minimum eigenvalue criterion would be chosen as current corner points to be tracked. The motion vectors for each of these corner points could then be calculated using Lucas-Kanade. Our particular implementation used an average (rounded to the nearest pixel) of the motion estimates of all the feature points. Corner blocks are updated using the calculated, rounded motion vectors and those that fail to meet the minimum eigenvalue criterion are culled out. Plots of calculated motion of the synthetic moving object and the associated groundtruth motions are shown below.

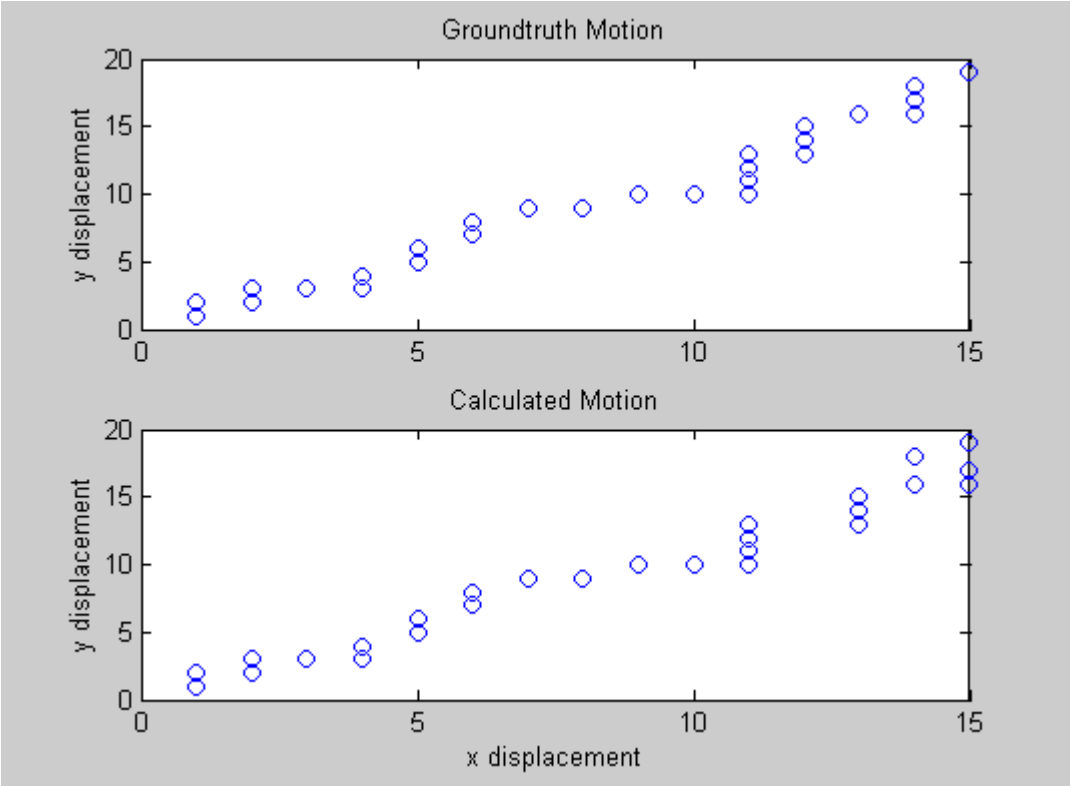


Figure 1 Comparison of groundtruth motion versus estimated motion using Lucas Kanade optical flow estimation. Note the errors in the latter portion of the sequence.

As can be seen from the figure, the estimated motion matches perfectly with the actual motion in the scene for about the first 25 frames. However, because there is no periodical corroboration with the original manually selected region of interest, invariably drift sets in and the features that are tracked are no longer those corresponding to the desired object. Indeed, Shi and Tomasi propose affinely mapping the object back to some stored template to eschew the onset of drift. Furthermore, Lucas-Kanade optical flow estimation, although producing accurate angular estimates of the displacements typically exhibits large errors in displacement magnitudes if iteration towards convergence is not performed at each point. Since what is desired is not absolute accuracy, but a quick and reasonable estimate of the object's location with some indication of the tracking confidence, Lucas-Kanade although useful for high-speed, accurate optical flow may not be the proper choice for demonstrating multi-camera feature tracking. Hence, robustness will be provided by the multiplicity of viewing angles, rather than the complexity or accuracy of each viewer's optical flow estimation.

2.4 Block-Based Motion Estimation

Since sub-pixel optical flow is not required and the interframe motions encountered may well span several pixels, block-based motion estimation should be sufficient for reasonable object tracking. Given an accurate window size, a sum of squared differences (SSD) error criterion applied to sufficiently sized blocks can satisfy the desired aforementioned requirements for multi-camera object tracking. The basic interframe motion estimation error is given by,

$$SSD_{error} = \frac{1}{n_1 n_2} \sum_{(x,y)\in\beta} (s_c(x+m, y+n, t+\Delta t) - s_c(x,y,t))^2$$

where n_1 and n_2 are the width and height of the block, m and n are the components of the motion vectors, and the corresponding pixel values in the ith and (i+1)th frames are differenced and squared. The difficulty in block-based motion estimation is ensuring that the current window is sized appropriately for the tracked object, otherwise if it is too large, the SSD error will be overwhelmed by background information, while an undersized window can easily drift and lose the object completely when inundated by a sea of diminutive spatial gradients.

2.5 Adaptive Window Sizing

Several techniques were developed and tested for reliable adaptive sizing of the object tracking window to deal with difficulties arising from motion perpendicular to the camera's sensor plane. Two of the tested techniques are described below.

2.5.1 Four Corner Method

This technique involves dividing the initially rectangular region of interest into four quadrants. Motion vectors can then be calculated for each quadrant independently. If the object is simply translating, the four quadrants will ideally have identical motion vectors and the window size will be unchanged. However, if the camera is zooming in and the object appears to be growing, the motion vectors will be divergent and cause the window size to increase. The window size can be, in effect, any quadrilateral that is approximated by piecewise constant segments, i.e., any four rectangles with a common central point. Although this method is appealing at first glance, problems can arise when

any one of the four windows slips off its assigned region. If this occurs, the window grows uncontrollably large and fails to reasonable track the object (see figure below).

This method can be improved with the addition of a fifth region, a central, rectangular region of interest. Furthermore, the four corner regions would only be responsible for the sizing of the central window rather than its translatory motion. This is accomplished by continually dividing the central window into four quadrants. Five motion vectors are calculated at each iteration, one for the main central window and the others for the four quadrants of the window. This eliminates unwanted drift by any of the four quadrants and allows for robust motion estimation using the highly accurate central window. In general, this technique fared better than the first, yet it was still subject to the perils of drift in extended sequences due to the fact that errors can continually accumulate without bound.

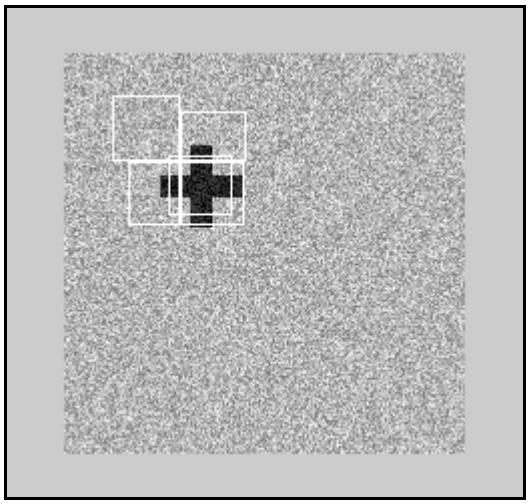


Figure 2 Variant of the Four Corner adaptive window sizing technique. The upper left quadrant has lost the object completely.

2.5.2 Corrleation-Based Adaptive Window Sizing

To overcome error accumulation, some sort of feedback is necessary that continually forces the window to converge back to some master object aspect ratio. In order to implement such overriding control of the window size, the original region of interest which is manually specified in the initialization stage can be stored for future

comparison. Instead of dividing the region of interest into multiple regions, the entire window will be used for block-based motion estimation with a block size equal to the window size. Thus, only one motion vector is calculated at each iteration. Furthermore, once the motion vector is calculated, the current window size is progressively varied and successively compared to a resized version of the stored object image. The window then assumes the size of the comparison with the smallest SSD. This correlative approach actually worked so well that instead of a purely frame differencing comparison for estimation of the motion vectors, a weighted portion of the correlative (with the stored image) error is added to the standard interframe SSD. Hence the error calculated now becomes,

$$SSD_{error} = \frac{1}{n_1 n_2}\left[a_1 \sum_{(x,y)\in\beta} (s_c(x+m, y+n, t+\Delta t) - s_c(x,y,t))^2 + \right.$$

$$\left. (1-a_1) \sum_{(x,y)\in\beta} (s_c(x+m, y+n, t+\Delta t) - s'_c(x_0, y_0, 0))^2 \right]$$

where a_1 is the weight given to each portion of the combined SSD error (interframe and correlative) and $s_c'(x_0,y_0,0)$ is the resized stored template image. Some restrictions were necessarily added in order to prevent uncontrolled convergence to unreasonable window sizes. Foremost was the restriction of minimum window size since optical flow estimation can become unreliable with decreasing dimensions of the comparison block. It is this last correlative method that is used with the multi-view feature tracking algorithms to be discussed in later sections.

2.6 Occlusion Detection

In order for any multi-view algorithms to be successful, they must have some real-time indication of the algorithm's object tracking reliability. Since a variant of a SSD error criterion is used, this can to first order give a reasonable assessment of each camera's tracking confidence. However, if a single camera's algorithm is to assess whether or not occlusion has occurred, it must have some more sophisticated technique of evaluating its SSD error. Unfortunately, a simple, invariant error threshold cannot be used since it is not apparent at any point what value of error is too high. This is due to the fact that various changes in object appearance, size and shape as well as varying

backgrounds can heavily influence the SSD error value. Thus, tracking may still remain accurate even when errors become larger than they were initially. To deal with such issues, a simple moving average of minimum correlative SSD errors of the last k (currently 5) frames can be calculated and compared with the current minimum correlative error. If the current minimum error is some c (where c>1) times greater than the moving average, then occlusion is said to have occurred. Accordingly, occlusion detection becomes adaptive to the current state of detection quality rather than imposing some arbitrary, absolute limit.

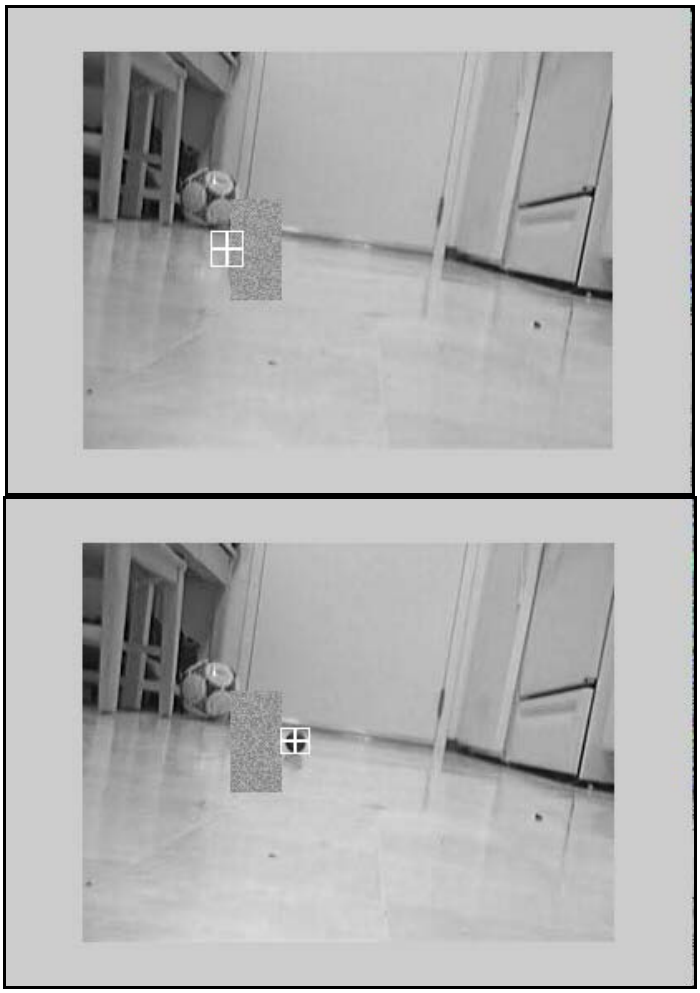


Figure 3 A ball has rolled behind the synthetically added noisy box. The window remains fixed at the point of last motion detection since occlusion has occurred. In the bottom frame, the object tracker has redetected the ball and resumes normal operation.

2.7 Object Redetection

Finally, the last issue of concern in single-camera object tracking, is that of object redetection once occlusion has occurred. Although in its most general form, this can be akin to initial object detection, which is a difficult problem in its own right, the problem can be slightly simplified by constraining the search area of the detection to the neighborhood in which the object was last detected. Furthermore, it can be further assumed that the object (once it reappears) will be of similar size to its last known dimension. These simplifications allow the development of a heuristic detection algorithm that works reasonably well in a variety of scenes. First, once occlusion has occurred, the object tracker shifts gears and becomes a simple motion detector whereby the tracking window tracks the largest temporal pixel intensity change in an enlarged search area. If there are no intensity changes that are some value times greater than the noise level, the window remains stationary until either motion or the object is detected. The object is said to have been detected if the current correlative SSD is some value d (where $d < 1$) times the moving average. Once redetection occurs, the object tracker returns to normal and standard object tracking resumes (see figures above).

3. Camera\Scene Calibration

Two different models of the point correspondence between the camera views were considered. The first is an affine model in which 3 points are the minimum number of points which are needed to estimate the parameters in the scene. In order to achieve better accuracy more pixels were used and least means squares was used to estimate the 6 parameters. The second model used was a prespective model in which the minimum number of pixels needed is 4. Again we used more pixels distributed along the whole field of view. It was clear that the results achieve d by those models were not sufficient and a better way was need to achieve the point correspondence.

3.1 Dynamic Point Correspondence

In this model we depend on the fact that there is no need to achieve accurate point correspondence in the whole scene but rather along the motion trajectory or the object.

Based on that and on the fact that the first few frames are very accurate estimate of the object, we perform the following algorithm and a block diagram is attached:

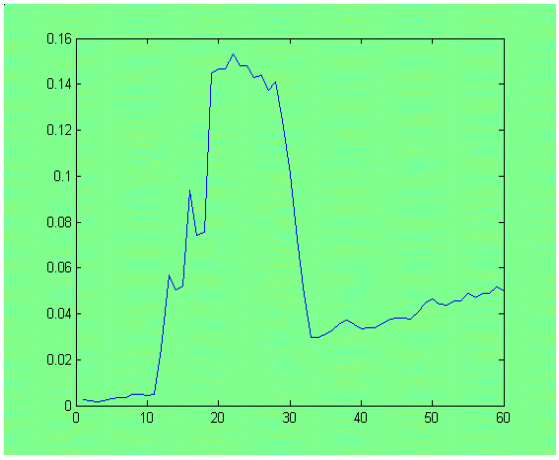
a) use an affine model to achieve correspondence between the 2 frames. Estimate the model parameters from the first 3 frames
b) check for each frame if the error is smaller than a certain threshold. If yes then add this new point to the model and resolve for the parameters using least mean squares. If the error is large then add this point to a temporary set that is used to have other estimate of the parameters
c) For the next frame check for the point correspondence using the temporary set and main set. The one with smaller error is used as the main transformation model parameters set
d) Go to step b until tracking ends

4. Camera-Central Node Communication

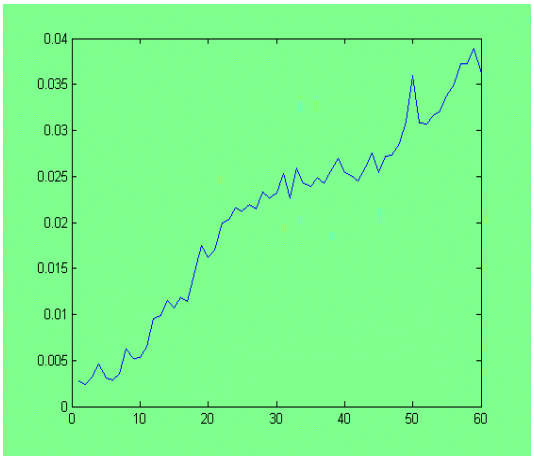
A very simple communication simulation and detection scheme is implemented that works fine. In this stage we go through the results of each tracking step frame by frame. If the error exceeds a certain criteria we disregard this camera and use the other camera. If both cameras have high error we indicate the inability to track. The criteria is very flexible and we used the moving average error in the previous frames, the percentage with respect to the maximum error recorded. It seems that such criteria is sufficient enough to make a decision on which camera to use. As an exmple, here are the error plots and figures for the 2 cameras at which occlusion occurs for one of the cameras and not for the other

(a)

(b)



(c)



(d)

Figure 4: a) view from first camera, b) view of the second camera, c) tracking error estimate for first camera, d) tracking error estimate for second camera

5. Future work

There are some simple extensions to this work which will render it very effective and useful:

1) Use an SDK to communicate with the camera directly so that we can achieve real time operation and not just record and operate on sequences
2) Provide the cameras or the laptops they are attached to with a wireless card to achieve this real time communication mechanism

3) Explore more robust tracking algorithms and check for simple fast versions
4) Refine the decision making step by making a more complicated stochastic model of the error so as to make the decision making more robust

References

1 S. Dockstader and M. Tekalp. Multiple camera tracking of interacting and occluded human motion. http://www.ece.rochester.edu/~dockstad

2. O. Javed, S. khan, Z. Rasheed and M. shah. Camera Handoff: Tracking in multiple uncalibrated stationary camera

3. F. pedersini, A. Sarti and S. Tubaro. Multi-camera parameter tracking. Proceesing IEE on Image signal processing, Vol 148, No1, 70-77, 2001

4. T. Shen and C. Menq.Automatic Camera Calibration for a Multiple sensor integrated coordinate measurement System. IEEE transactions on robotics and automation, Vol 17, No 4, 2001

5. J. L. Barron, S. S. Beauchemin, and D. J. Fleet. On optical flow. In AIICSR 94: Proc. 6th Int. Conf. on Artificial Intelligence and Information Control Systems of Robots, pages 3--14, Bratislava, Slovakia, September 1994.

6. J. Shi and C. Tomasi. Good features to track. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94), pages 593--600, IEEE Computer Society, Seattle, Washington, June 1994.

7. Tomasi, C. and Kanade, T. 1991. Detection and tracking of point features. Tech. Rept. CMU-CS-91132. Pittsburgh:Carnegie Mellon University School of Computer Science.

8. R. Szeliski, S. B. Kang, and H.-Y. Shum. A parallel feature tracker for extended image sequences. In IEEE International Symposium on Computer Vision, Coral Gables, Florida, November 1995.

9. B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In Proceedings IJCAI, pages 674--679, Vancouver, Canada, 1981.

10. David Prewer and Les Kitchen. **"**A fast simple edge-based visual tracker." *Technical Report 97/20,* September 1997.

Division of the work:

Helmy:

1) **implemented lucas and kanade for optical flow**
2) **implemented block matching algorithm**
3) **implemented the tracking algorithm**
4) **implemented occlusion detection and object resizing**

khaled:

1) **implement the dynamic point correspondence**
2) **implemented the synthetic test sequences**
3) **acquired the various video clips and impelemted the test benche environment**

MATLAB Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test of Shi,Tomasi and Kanade image tracking -- EE392J Final Project                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numb=40; % number of frames
motion=zeros(2,numb);

%generate random motion
for m=1:numb,
   motion(1,m) = round(rand(1,1));
   motion(2,m) = round(rand(1,1));
end
motionsum = cumsum(motion,2);
sigma=5; %standard deviation of smoothing mask
image1=ones(200,200);
image1(45:55, 30:70)=0;
image1(30:70,45:55)=0;
 hmsk = 1/12*[-1 8 0 -8 1]; %derivative filters
 vmsk = 1/12*[-1; 8; 0; -8; 1];
 block=16+2;
 mvs=zeros(2,numb);
smsk = fspecial('gaussian', ceil(3*sigma), sigma); %smoothing mask
points= getFeatures(image1, block-2,block-2,.6) %get good block features
centroid = round(sum(points,2)/size(points,2)+block/2)

for m=1:numb,

   image2=ones(200,200);
image2((45:55)+motionsum(1,m), (30:70)+motionsum(2,m))=0; %generate next frame
image2((30:70)+motionsum(1,m), (45:55)+motionsum(2,m))=0;
```

```
vs=zeros(2,size(points,2));
%iterate over all feature blocks and calculate motion vectors
for n=1:size(points,2),
   if n <= size(points,2),
   curr = image1(points(1,n)+(1:block)-1,points(2,n)+(1:block)-1);
     next =  image2(points(1,n)+(1:block)-1,points(2,n)+(1:block)-1);

    %solve for gradients
     smpic= conv2(.5*curr+.5*next, smsk, 'same'); % smooth sum of frames
     xgrad = conv2(smpic, hmsk, 'valid');
     xgrad =xgrad(3:(block-2),:);
      ygrad = conv2(smpic, vmsk, 'valid');
      ygrad =ygrad(:, 3:(block-2));
      temp = sum(sum(xgrad.*ygrad));
      A = [sum(sum(xgrad.*xgrad)) temp; temp sum(sum(ygrad.*ygrad))];
      if min(eig(A)) > .01,
      tg = conv2(next-curr,smsk,'same');
      tg = tg(2:(block-3),2:(block-3));
      B = sum([(tg(:).*xgrad(:))'; (tg(:).*ygrad(:))'],2);
      % Solve for the velocity
      v = A \ (-B);
      vs(1,n) = 1*v(2)+0*vy;
      vs(2,n) = 1*v(1)+0*vx;
      %pointstest= getFeatures(image1, block-2,block-2,.6);
      points(1,n) = points(1,n) + round(2*vs(1,n));
      points(2,n) = points(2,n) + round(2*vs(2,n));
   else
      atemp = points(:,1:(size(points(:,2))-1));
      points = atemp;
      n=n-1;
   end
```

```
end
    end
    image1=image2;
    %store motion vectors
    mvs(1,m) = round(sum(round(2*vs(1,:)))/size(vs,2));
    mvs(2,m) = round(sum(round(2*vs(2,:)))/size(vs,2));
end
%plot estimated motion versus actual motion
subplot(2,1,1);
plot(motionsum(1,:), motionsum(2,:),'o');
subplot(2,1,2);
plot(cumsum(mvs(1,:)),cumsum(mvs(2,:)),'o');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [points, val] = getFeatures(image, sizem, sizen, thresh);
% Extract good tracking features with minimum eigenvalue greater
% than thresh and window size sizem by sizen, a la Tomasi and Kanade.
hmsk = [-.5 0 .5];
vmsk = [-.5; 0; .5];
points=[];
val=[];
sizem=sizem+2;
sizen=sizen+2;
sigma = sqrt(sizem*sizen)/4;
smsk = fspecial('gaussian', ceil(6*sigma), sigma);


for m = 1:4:(size(image,1)-sizem+1),
    for n = 1:4:(size(image,2)-sizen+1),
        curr = image(m:(m+sizem-1),n:(n+sizen-1));
```

```
        xgrad = conv2(curr, hmsk, 'valid');
        xgrad =xgrad(2:(sizem-1),:);

        ygrad = conv2(curr, vmsk, 'valid');
        ygrad =ygrad(:, 2:(sizen-1));

        temp = sum(sum(xgrad.*ygrad));
        mat=[sum(sum(xgrad.*xgrad)) temp; temp sum(sum(ygrad.*ygrad))];
        minlamb=min(eig(mat));

        if minlamb > (.5*thresh*max(image(:))^2*sqrt(sizem*sizen))
            vect = [m;n];
            points = [points vect];
            val = [val minlamb];
         end
      end
 end

%%%%%%%%%%%%%%%%%%%
%% generation of synthetic test sequences
%%%%%%%%%%%%%%%%%%%%
clear all;
close all;

frame=0.3+0.5*rand(100,100);

x=zeros(7,7);
rcam_offset_x=30;
rcam_offset_y=20;
rcam_width=60;
rcam_height=60;
```

```
lcam_offset_x=10;
lcam_offset_y=20;
lcam_width=60;
lcam_height=60;
xold=50;
yold=50;
image=frame;
image(xold-4:xold+2,yold-4:yold+2)=x;
scene=zeros(100,100,200);
left_camera=ones(lcam_width,lcam_height,200);
right_camera=ones(rcam_width,rcam_height,200);
scene(:,:,1)=image;
left_camera(:,:,1)=rot90(image(lcam_offset_y:lcam_offset_y+lcam_height-
1,lcam_offset_x:lcam_offset_x+lcam_width-1));
right_camera(:,:,1)=flipud(fliplr(rot90(image(rcam_offset_y:rcam_offset_y+rcam_height
-1,rcam_offset_x:rcam_offset_x+rcam_width-1))));
%imshow(image)
k=1;
%% generate random motion for a box
for i=1:20
   par=6*(rand(1,2)-0.5);
 for j=1:10
     image=frame;
     xpos=round(xold+par(1));
     ypos=round(yold+par(2));

    if (xpos>20)&(xpos<80)&(ypos>20)&(ypos<80)
       image(xpos-4:xpos+2,ypos-4:ypos+2)=x;
    else
        break;
    end
```

```
    xold=xpos;
    yold=ypos;
    scene(:,:,k)=image;
left_camera(:,:,k)=rot90(image(lcam_offset_y:lcam_offset_y+lcam_height-
1,lcam_offset_x:lcam_offset_x+lcam_width-1));
right_camera(:,:,k)=flipud(fliplr(rot90(image(rcam_offset_y:rcam_offset_y+rcam_height
-1,rcam_offset_x:rcam_offset_x+rcam_width-1))));

    k=k+1;

    %subplot(3,1,1)
    %imshow(image);

  end
end
%%%%%
%Camera point correspondance detection
%%%%%%
clear all;
close all;
first_camera=zeros(240,320);
second_camera=zeros(240,320);
for i=1:10
tmp=aviread('video4',i);
tmp1=rgb2ntsc(double(tmp.cdata));
first_camera=first_camera+tmp1(:,:,1);
tmp=aviread('video5',i);
tmp1=rgb2ntsc(double(tmp.cdata));
second_camera=second_camera+tmp1(:,:,1);
end
first_camera=255*first_camera/max(first_camera(:));
```

```
second_camera=255*second_camera/max(second_camera(:));

subplot(2,1,1)
imshow(first_camera,[])
subplot(2,1,2)
imshow(second_camera,[])
[inx,iny]=ginput(8);
new_x=inx(2:2:8);
new_y=iny(2:2:8);
old_x=inx(1:2:8);
old_y=iny(1:2:8);

A=[old_x(1) old_y(1) 1 0 0 0 -1*old_x(1)*new_x(1) -1*old_y(1)*new_x(1);
   0 0 0 old_x(1) old_y(1) 1 -1*old_x(1)*new_y(1) -1*old_y(1)*new_y(1);

   old_x(2) old_y(2) 1 0 0 0 -1*old_x(2)*new_x(2) -1*old_y(2)*new_x(2);
   0 0 0 old_x(2) old_y(2) 1 -1*old_x(2)*new_y(2) -1*old_y(2)*new_y(2);

   old_x(3) old_y(3) 1 0 0 0 -1*old_x(3)*new_x(3) -1*old_y(3)*new_x(3);
   0 0 0 old_x(3) old_y(3) 1 -1*old_x(3)*new_y(1) -1*old_y(3)*new_y(3);

   old_x(4) old_y(4) 1 0 0 0 -1*old_x(4)*new_x(4) -1*old_y(4)*new_x(4);
   0 0 0 old_x(4) old_y(4) 1 -1*old_x(4)*new_y(4) -1*old_y(4)*new_y(4);
];
parameters=pinv(A)*[new_x(1);new_y(1);new_x(2);new_y(2);new_x(3);new_y(3);new_
x(4);new_y(4);];
```