# OBJECT-TRACKING USING MULTIPLE CONSTRAINTS

ISMAIL ONER SEBE

EE392J, DIGITAL VIDEO PROCESSING, WINTER 2002

Stanford University, EE Department

E-mail: iosebe@stanford.edu

## ABSTRACT

Object-tracking is one of the most popular areas of video processing; most of the methods so far are object-dependent and concentrates only on one constraint of the object. In this report, I tried to combine multiple constraints for a faster and more robust tracking algorithm. Dynamic use of multiple constraints is the ultimate goal of the project, for the specific video sequences of basketball game.

## INTRODUCTION

In the last decade object-tracking become very popular because of its applicability to daily problems and ease of production, e.g. surveillance cameras, adaptive traffic lights with object tracking, plane detection. The superiority of object-tracking to object recognition became apparent after the development in the video processing and motion estimation.

Although object-tracking using motion vectors is a very powerful method, it fails to give a robust and reliable answer all the time. There are additional methods for motion vectors in order to overcome some deficiencies of this method, such as regularization using smoothness constraint, multipoint neighborhood, and pel-recursive methods [1]. The methods of object recognition most of the underestimated and not used but in this report I try to combine these methods in order to use the power of both sides.

The report concentrates on the basketball sequences because the object recognition part of the algorithm uses feature detection, which is specific to the sequence. The

obvious choices for constraints are color information and edges. This study basically tries to combine color and edge constraints with the motion estimation criterion.

## CONSTRAINTS

This project tries to combine the color constraint and edge constraint with motion vectors, so let's look at these methods separately and in detail, to see if we can really use them to estimate the position and size of the ball. Here is a luminance of a sample frame from the sequence.

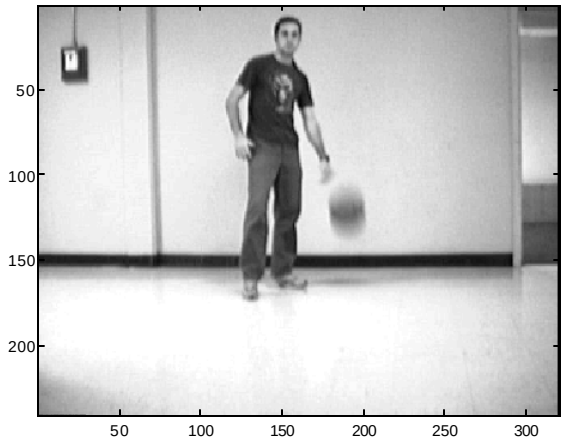



**Figure 1**

- Motion Vector Constraint

Block-matching algorithm is the most reliable and common motion vector estimation algorithm of all (very subjectively stated). Although it's the most common of all, it has some robustness problems. The problems most arise in the presence of noise and occlusions. As seen in the class, if the sequence has appreciable camera noise the block-matching algorithm starts to give spurious motion vectors. This phenomenon usually happens if the estimation is tried to be done in a smooth region.

If these spurious vectors are examined, it is easily can be seen that there is a strong relation between the vectors and the mean squared error. When the region that's examined has very similar values in a region, in the presence of camera noise these values change constantly. As you may see from Figure 1, the image quality is very low in terms of noise. So the normal matching criterion MSE is not enough to estimate the motion vectors.

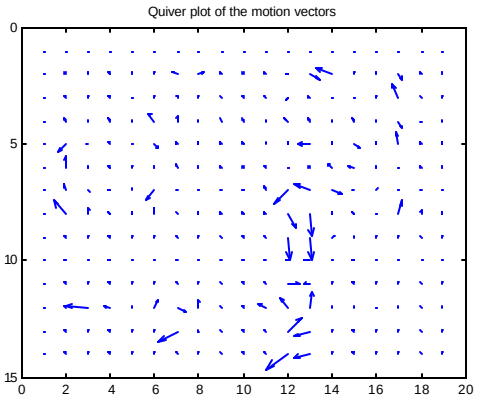
A simple but effective choice of matching criterion is a second term that will dominate in these situations and also will cease in the presence of motion. I propose to use a distance panelizing term as the second criterion. The ultimate form is as follows:

| S (x-x')2+ ?dist |
|---|

**Formula 1**

The result of the proposed matching criterion is as follows:

Criterion=MSE




Criterion = MSE+ ?dist

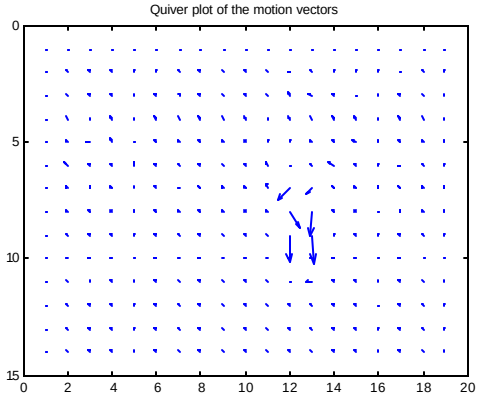


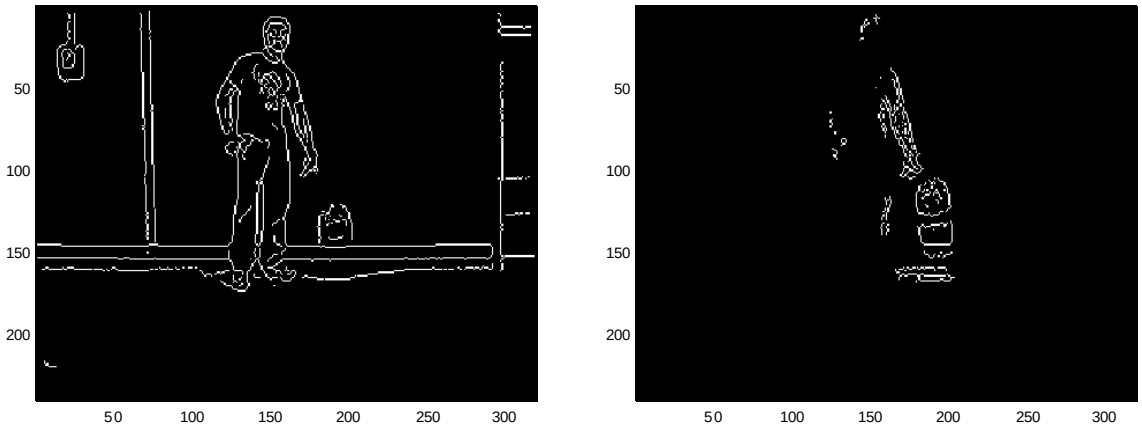

**Figure 2**

Motion vectors above are calculated using 16x16 window size with 32x32 pixel search area fro the whole image. The experimental value of the lamda is found to be 20. Actually it gives similar values for the range 10-50. If we look at this term it can be seen that even for the farthest point it gives an error bias of 2 out of 255 for each pixel. So if there is only noise this diminishes the noise but in the presence of the motion it is too small. Simple threshold for the MSE is tried too, but this scaling seems to work better empirically.

- Edge Constraint

Edge feature is one of the most popular object features in object recognition because of its power to represent a non-transparent object effectively with fewer values. Another reason is human eye is more sensitive to edges than the smooth areas. Because of the above reasons edge constraint is attempted to be used as one of the constraint.

As mentioned before the image quality is low, but this time in terms of motion blur. Because the ball is moving relatively fast the capture efficiency is very poor when the ball is moving. See figure 1. So the edge feature of the ball is not easy to calculate. Another problem is the images are not edge free. There are too many details around the balls trajectory including bar at the bottom of ball and the one playing with the ball. But these values can be eliminated by using absolute frame difference for edge detection. The results of both methods are given as follows:




Edge Detection to Current Frame  Edge Detection to Difference Frame

**Figure 3**

Canny edge detection with threshold value of 0.1 is used. Edge detection applied to only current frame suffers from the background detail and the player. The difference frame suffers from doubling of the ball because of using frame difference. As a result, edge constraint will not be used as one of the constraints in this project because of the above problems it has.

- Color Constraint

The images that are captured by the camera are converted from Avi format to YCrCb format. The power of this format is it has all the information that is needed and we do not need to convert the data into other color spaces. The luminance value is used for motion estimation and the all Y, Cr, and Cb is used for the color constraint. The color space is divided into inliers and outliers. The threshold value is the found with an algorithm that will be discussed in the following section. There band-pass are applied, one for each color vector. The outputs are "anded" in order to get a binary segmented image of the sequence. This part of the code is very fast because it only uses Matlab matrix operations. The output image mostly has the values that are needed, ball is mostly retrieved. But because the ball is smeared during capture, some of the pixels on the ball are rejected. Actually the position of the ball can't be determined by looking at the images (it is really hard even for humans, refer figure 1).

A sample output of the color constraint looks like as follows:

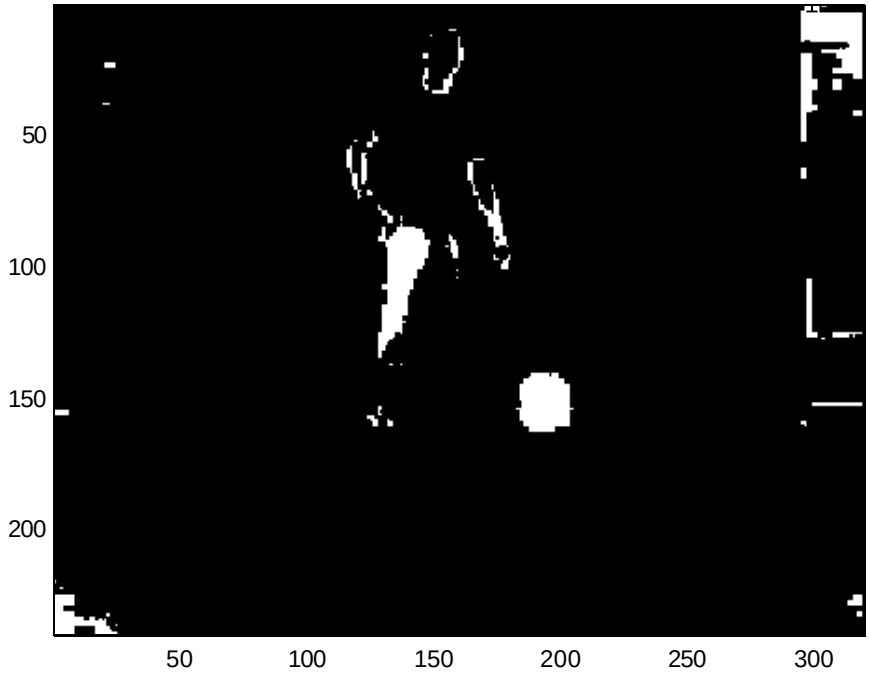



**Figure 4**

As you may see the proposed scheme works pretty good. But unfortunately not all the results are that good. Sometimes the ball is mostly rejected or the hand of the player (color properties are close) is included in the ball too; which results in an offset in the estimation of the balls position and radius.

## GENERAL PROCEDURE

Direct implementation of the above constraints will not give us a robust and fast algorithm. In this project, the constraints are used sequentially and the results of one affect the other in terms of where to make the search.

The motion estimation is the most computationally expensive constraint to calculate so we need to fit a window for that and decrease the number of calculations. The color constraint doesn't do anything by itself so a circle fit function is used in order to find the ball in this binary image. The search window is again adjusted using the result from the motion vector constraint.

Before going into the details of the general procedure, let's look at the simplified block diagram of the overall program.

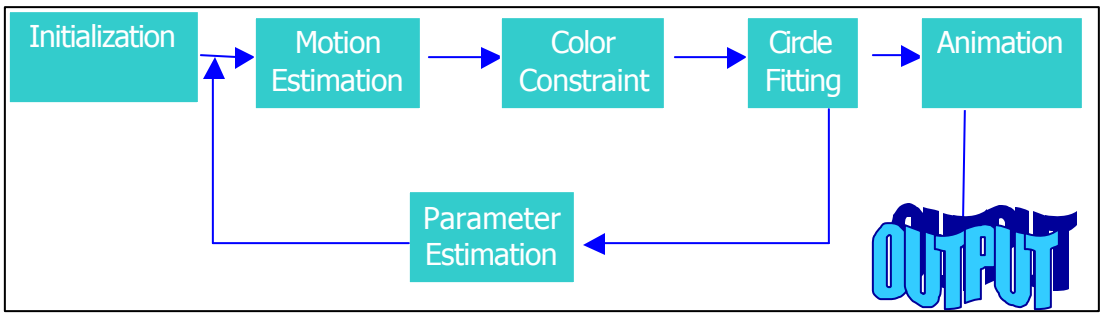



**Figure 5**

The program starts with initialization; this includes a pre-definition of the position of the ball (taken as input from mouse by means of clicking on the image), pre-definition of the ball's approximate color.

The motion estimation is done only for a window which has a size of 80x80 pixels, where size of the blocks are 16x16, and has a search area of 32x32. I have used pixel accuracy because the motion vectors are extracted fast enough with windowing; but this procedure may become faster if sub-sampling was used.

The second constraint is color constraint and the whole image is used in order to threshold the image, because the operations are done at matrix comparisons and dividing the matrix itself costs more than applying the threshold to the whole image. The lower and upper threshold values are calculated recursively at the end of each frame, so the images are pretty good after this step.

The third step is the circle fitting, which uses the motion vector constraint result as a starting window and searches on a 30x30 window for the ball. Actually this window doesn't have to be this big most of the time; the motion vector estimates are pretty close to the actual values. Here are the statistical results of the motion vector estimation error with respect to the final result.

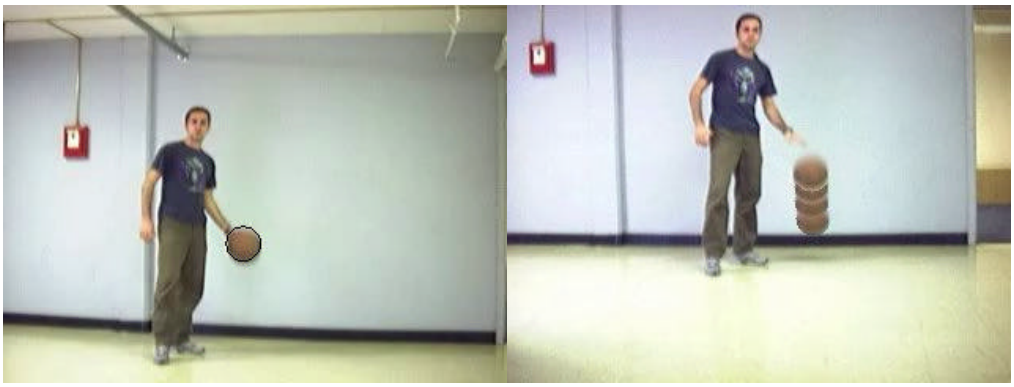
| | Horizontal | Vertical |
|---|---|---|
| Mean of absolute difference | 1.6 | 1.7 |
| Mean of difference | 0.82 | 0.51 |
| Variance of difference | 4.87 | 7.97 |
| Maximum difference | 12 | 13 |

**Table 1**

The variance in vertical dimension is more because of the more movement in the vertical dimension and the color estimate of the ball is not always correct but this shows why we need to use these methods together to get more robust algorithm. The circle fit function takes only the binary image from the color constraint and the window search and gives the position, radius and an image of the approximate ball. The radius of the ball may change so an expanding radius method is used in order to find the ball. With synthetic images these function is tested. Unfortunately, there was no video sequence to test this so it couldn't be shown as a result.

The last step is the parameter and window update. The balls estimate point and the estimate circle is used to extract balls color information. This is done by finding the histogram of the enclosed region by the ball and extracting the threshold values from these histograms. Because the point that is found is only an estimate and always has the probability of being false, the update function uses a window size of 3 in order to approximate the threshold values. Finite size is used in order to allow changes, like illumination. The weights are 0.5, 0.25 and 0.25 for the current, previous and the one before previous respectively.

An optional choice which is not directly related to the project is the animation part. There are two choices of animation which is circling the ball and making ghost balls in the motion trajectory. The examples of these are as follows:



Animation Type I Animation Type II

**Figure 6**

The first animation is really easy to do because all the information for the animation is already found. Only a circle is drawn on the image with the found radius to the estimate point.

The second animation needs more calculation and more data than the first one. As you may see from the figure 6, the original ball is not deformed and the previous balls are shown with respect to their time of appearance. The last ball is at the farthest back and then the third and then second and finally the current ball is on the top. This animation requires good estimate of the current plus all last three frames. The values of the ball are directly copied from the corresponding frames.

This kind of animations are done mostly with frame difference methods, as you may see in this method only the ball shows this property and all the other moving objects are not affected. The overlap of the estimate circles are the main source of animation. Refer to the appendix for the function "anim.m" for further details.

## FURTHER IMPROVEMENTS

The initialization of the procedure is done by user input. The whole search for the image for the ball is tried but this procedure doesn't always converge to the desired point in the approximate ball color values and shows weakness in other aspects like non-recoverability. Motion vector initialization is tried but when there are other things in the scene this is not a very good approximate either. So the solution for the initialization remains as an open question. But if the failure of the overall is somehow not a problem than one of these weak procedures can be used.

The occlusion of the ball results in bad approximates but maybe a limit to the match can be put and this can be detected. Because this is the same question of the initialization problem, there is no answer can be given to this for the time being. The above procedures are tried for this methods but their reliability is very low and may converge random points.

Multiple ball detection can be supported, but this question can be solved very easily with an object oriented programming language. If the ball was defined as an object that can move and if this ball detection method is written as a function then the number of balls will not be a problem at all.

The aim of this project is try to come up with a fast but robust method, rather than trying to answer fancy question which may ort may not result in correct solution all the time.

# CONCLUSION

This project tries to combine video processing algorithms with image processing algorithms in order to get more robust and faster tracking algorithm. The proposed methods so far lacks either in robustness to noise and resolution or too slow to work in real-time.

The proposed method works really fast even in the presence of error when compared to the other conventional methods. The code is written in Matlab and each frame takes around 10 seconds without additional animation and it takes around 17 seconds with animation. The same machine makes a full 320x240 block based motion vector estimation (that is told in motion vector constraint) in 8 seconds. This shows that this method can be applied in real time if it was written in C or C++ which are capable of doing iterative loops much faster than Matlab.

Motion vectors are very powerful when they are combined with another more robust method like color matching. The details of the cross usage of these constraints can be found in the "Constraints" part of the report.

This project directly shows a daily life application of the proposed method, ghost ball animation, e.g. think this is applied to basketball games in NBA; it will be more interesting to watch games. The animations may be varied and modified by the need, too.

As a result using multiple constraints for object-tracking is a very powerful, robust and faster way when compared to the conventional methods.

# REFERENCES


1. Video Processing and Communications; Wang Y., Ostermann J., Zhang Y., 2002
2. Machine Vision; Jain R., Kasturi R., Schunk B., 1995
3. Elliptical Head Tracking Using Intensity Gradients and Color Histograms; Birchfield S., IEEE Conference on Computer Vision and Pattern Recognition, June 1998

# APPENDICES

## Main.m

```
function main(videoin,videoout)
%This Function tracks the ball in an basketball video sequence
%Multiple constraints are used for the estimation of the ball
%These constraints are motion vectors and the color information.

%definitions
blocks=16;
mvsize=256/blocks;
rows = 240;   %image size
cols = 320;
fid = fopen(videoin,'r');
fid2= fopen(videoout,'w');
imm1=ones(240,320);
imm2=ones(240,320);
imm3=ones(240,320);
last=250;
%differnece between the estimated mv and the output result
xdif=zeros(last,1);
ydif=zeros(last,1);
%Position of the ball
xpos=zeros(last,1);
ypos=zeros(last,1);
%The color values
YUV=[105 112 142];
var=[50 8 10];
YUV1=YUV;
var1=var;
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %read very first image first image
   [y1,u1,v1,count]=myread(fid,rows,cols);
   figure;
   imagesc(y1);
   colormap(gray);
   [xin,yin]=ginput(1)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   finish=0;
   for i=1:last
      i
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %read second image
   [y2,u2,v2,count]=myread(fid,rows,cols);

   %check end of file
   if count==0
      finish=1;
      i=last+2;
   end

   if finish==0
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
   %Find the motion vectors
   [x,y]=mv22(y1,y2,blocks,[yin xin],2);
```

```
    %The estimate motion
    transx=sum(sum(x.*(abs(x)>2)))/(sum(sum(abs(x)>2))+1e-20);
    transy=sum(sum(y.*(abs(y)>2)))/(sum(sum(abs(y)>2))+1e-20);
    xx=round(xin+transx);
    yy=round(yin+transy);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    imout=color_constraint2(y2,u2,v2,YUV,var,round(x)',round(y)');
    rmax=11;
    %The window comes from the mv22 the estimate is pretty accurate so small
window
    lim=round([xin+transx-15 xin+transx+15;yin+transy-15 yin+transy+15]);
    %Fit the ball to the image
    [imm,immf,xin,yin,r]=fit2(imout,rmax,lim);
    %Find the color information
    [YUV3,var3]=color_update(y2,u2,v2,xin,yin,r);
    xpos(i)=xin;
    ypos(i)=yin;
    xdif(i)=xin-xx;
    ydif(i)=yin-yy;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %color update
    if i==1
       YUV=(YUV1+YUV3)/2;
       YUV2=YUV3;
       var=(var1+var3)/2;       var2=var3;
    else
       YUV=(1/2*YUV2+1/4*YUV1+1/4*YUV3);
       var=(1/2*var2+1/4*var1+1/4*var3);
       YUV1=YUV2;
       YUV2=YUV3;
       var1=var2;
       var2=var3;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Create animation
    if i==1
       y3=y2;
       u3=u2;
       v3=v2;
       imm3=immf;
       y_3=y2;
       u_3=u2;
       v_3=v2;
    elseif i==2
       y3=y2;
       u3=u2;
       v3=v2;
       imm2=immf;
       y_2=y2;
       u_2=u2;
       v_2=v2;
    elseif i==3
       y3=y2;
       u3=u2;
       v3=v2;
       imm1=immf;
       y_1=y2;
```

```
      u_1=u2;
      v_1=v2;
   else
      [y3 u3
v3]=anim(y2,u2,v2,immf,y_1,u_1,v_1,imm1,y_2,u_2,v_2,imm2,y_3,u_3,v_3,imm3,imm);
      imm3=imm2;
      imm2=imm1;
      imm1=immf;
      y_3=y_2;
      u_3=u_2;
      v_3=v_2;

      y_2=y_1;
      u_2=u_1;
      v_2=v_1;

      y_1=y2;
      u_1=u2;
      v_1=v2;
   end
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %Write the output to disk
   mywrite(fid2,y3,u3,v3);
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   end
   %at the end switch the second image as one
   y1=y2;
   u1=u2;
   v1=v2;

end
fclose(fid);
fclose(fid2);
```

Mv22.m

```
function [x,y]=mv22(Y1,Y2,blocks,point,window)
%This function finds the motion vectors between two images
%The motion vectors are extracted for only the window
%construct the empty motion vector
motion=zeros(size(Y1,1)/blocks,size(Y1,2)/blocks,2);
%Construct the window
limit=zeros(2,2);
%point is in the form of (y,x)
limit(1,1)=floor(point(1)/blocks)-window;
limit(1,2)=ceil(point(1)/blocks)+window;
limit(2,1)=floor(point(2)/blocks)-window;
limit(2,2)=ceil(point(2)/blocks)+window;
%The search is confined into the window size
if limit(1,1)<2
   limit(1,1)=2;
end
if limit(2,1)<2
   limit(2,1)=2;
end
```

```
if limit(1,2)>size(Y1,1)/blocks-2
   limit(1,2)=size(Y1,1)/blocks-2;
end
if limit(2,2)>size(Y1,2)/blocks-2
   limit(2,2)=size(Y1,2)/blocks-2;
end
%Over the image with the step size of blocks*blocks
SAD=zeros(2*blocks,2*blocks);
for i=limit(1,1):limit(1,2)
   for j=limit(2,1):limit(2,2)
      %Make the initializiations
      min_x=0;
      min_y=0;
      min=100000;
      A=Y2(i*blocks+1:i*blocks+blocks,j*blocks+1:j*blocks+blocks);
      for k=-(blocks-1):blocks
         for l=-(blocks-1):blocks
            y=i*blocks+k;
            x=j*blocks+l;
            %Split the desired part out of the images
            B=Y1(y:y+(blocks-1),x:x+(blocks-1));
            %Calculate the absolute error
            %The total estimation error + smoothness + distance measure
            SAD(k+blocks,l+blocks)=sum(sum(abs(A-B)))+20*((k-1)^2+(l-
1)^2);
         end
      end
      %write the motion vector to the motion matrix
      [val,min_y]=min(SAD);
      [val,min_x]=min(val);
      motion(i,j,1)=-(min_y(min_x)-blocks)+1;
      motion(i,j,2)=-(min_x-blocks)+1;
   end
end
%make the flip action and plot the MVs
x=motion(:,:,2);
y=motion(:,:,1);
```

colorseperator.m

```
function imout=color_seperator2(Y,U,V,YUV,var)
% imin is sizey x sizex x 3 (YUV image)
% YUV is a vector for desired YUV value
% Var is the allowed variance of the color
map1=zeros(size(Y,1),size(Y,2));
map2=zeros(size(Y,1)/2,size(Y,2)/2);
map=zeros(size(Y,1),size(Y,2));

YUV=double(YUV);
%Y value
map1=Y>(YUV(1)-var(1));
map1=and(map1,Y<(YUV(1)+var(1)));

%U value
map2=U>(YUV(2)-var(2));
```

```
map2=and(map2,U<(YUV(2)+var(2)));
%figure;
%imagesc(map2);
%colormap(gray);

%V value
map2=and(map2,V>(YUV(3)-var(3)));
map2=and(map2,V<(YUV(3)+var(3)));

for i=1:size(Y,1)
   for j=1:size(Y,2)
      ind1=round(i/2);
      ind2=round(j/2);
      if and(map2(ind1,ind2),map1(i,j))
         map(i,j)=1;
      end
   end
end

imout=map;
```

Fit2.m

```
function [im2,im1,best_x,best_y,best_r]=fit2(im,rmax,limit)
%This function tries to fit an circle to the image windowed with
"limit"

%limit is on the form of [xmin xmax;ymin ymax]
best=0;
best_r=0;
best_den=0;
best_x=100;
best_y=100;
for j=limit(2,1):limit(2,2)
   for i=limit(1,1):limit(1,2)
      nonzero=1;
      %for every posible r value
      for r=11:rmax
         if nonzero==1
            %search for the best fit
            match=0;
            for k=-r:r
               for l=-r:r
                  if (k^2+l^2)<=r^2
                     if and((j+k)<size(im,1),(i+l)<size(im,2))
                        match=match+double(im(j+k,i+l));
                     else
                        match=0;
                     end
                  end
               end
            end
            %convert to density
            density=match/pi/(r)^2;
            if match>best%density<0.6
```

```
                best_den=density;
                best=match;
                best_y=j;
                best_x=i;
                best_r=r;
             %else
             %  nonzero=0;
             end
          end
       end
    end
end
best_x;
best_y;
best_r;

im2=zeros(size(im,1),size(im,2));
for j=-best_r:best_r
   for i=-best_r:best_r
      if (i^2+j^2)<=best_r^2
         im2(j+best_y,i+best_x)=1;
      end
   end
end
im1=im2;
%find the outher shell
im2=double(edge(im2,'sobel'));
%insert the image
im2=(im2==0);
```

Color_update.m

```
function [YUV,var]=color_update(Y,U,V,xx,yy,r)
%This function estimates the color properties of the ball
%Uses the current image and estimate position and radius
Y1=zeros(1,4*r^2);
U1=zeros(1,4*r^2);
V1=zeros(1,4*r^2);
var=zeros(3,1);
k=0;
%Extract thye values of image if they are in the circle
for i=yy-r:yy+r
   for j=xx-r:xx+r
      if ((i-yy)^2+(j-xx)^2)<=r^2
         k=k+1;
         ind1=round(i/2);
         ind2=round(j/2);
         Y1(k)=Y(i,j);
         U1(k)=U(ind1,ind2);
         V1(k)=V(ind1,ind2);
      end
   end
end
%Clip the data
Y1=Y1(1:size(Y1,2)-sum(Y1==0));
```

```
U1=U1(1:size(Y1,2)-sum(Y1==0));
V1=V1(1:size(Y1,2)-sum(Y1==0));
%Find statistical data such as mean
errorY=sort(abs(Y1-mean(Y1)));
errorU=sort(abs(U1-mean(U1)));
errorV=sort(abs(V1-mean(V1)));
%Assign the values
YUV=[mean(Y1) mean(U1) mean(V1)];
var=[errorY(round(size(errorY,2)*2/3))
errorU(round(size(errorU,2)*2/3)) errorV(round(size(errorV,2)*2/3))];
```

## Myread.m

```
function [y,u,v,count]=myread(fid,rows,cols)
%Read the next frame
   [temp,count] = fread(fid,[cols,rows],'uchar');
   y = temp';
   [temp2,count1] = fread(fid,[cols/2,rows/2],'uchar');
   u = temp2';
   [temp2,count1] = fread(fid,[cols/2,rows/2],'uchar');
   v = temp2';
```

## Mywrite.m

```
function mywrite(fid,y,u,v)
%Write the next frame
   fwrite(fid,y','uchar');
   fwrite(fid,u','uchar');
   fwrite(fid,v','uchar');
```