实验四 运动向量的预测

# 问题描述

利用matlab,导入YUV格式的视频，选择第一帧的某一位置为目标宏块，第2、3、5、10、20、50、100帧为参考帧，实现顺序搜索算法（采用绝对误差测量）对目标宏块进行运动向量的预测，再实现2D对数搜索算法（采用绝对误差测量）对目标宏块进行运动向量的预测。实验中自行设置搜索窗口的大小和9个参考点的位置，并对这两个参数对实验结果的影响进行分析。（视频的分辨率352\*288，共300帧。展示在不同参考帧上得到的结果并进行分析，分析不同的搜索窗口大小和参考点位置对实验性能的影响，解释顺序搜索和2D对数搜索得到的结果是否最优。）

# 问题分析

一段视频的时间冗余是比较显著的，利用这个特征，可以将当前帧与其他帧的差值进行编码，探测相应像素或区域的移动并测量它们的差值来“补偿”这些运动。采用该方法的视频压缩算法被称为基于运动补偿（MC）的压缩算法，算法一般有三步：（1）运动估计（2）基于运动补偿的预测（3）预测误差的生成——差值。本次实验，需要实现运动向量的搜索。

搜索的目标是找到一个向量(i,j)作为运动向量MV(u,v)，使得平均绝对误差(MAD)最小。（书本P201公式）

## 2.1顺序搜索

寻找运动向量最简单的方法是顺序搜索参考帧中整个(2p+1)(2p+1)大小的窗口。将该窗口中的每一个宏块逐个像素地和目标帧中的宏块进行比较，得到各自的MAD值，MAD最小的向量即为目标帧中宏块的运动向量。

## 2.2 2D对数搜索

对数搜索虽然不是最优方法， 通常却是非常有效的一个办法，且代价较低。在搜索窗口中只有9个位置被标记为1，它们作为基于MAD搜索的起始位置。当MAD最小值的位置确定后，将新的搜索区域中心移动到该位置，搜索的步长（偏移）减半。在下一次迭代中，9个新的位置为标记为2，依次类推。

计算复杂度为O(logp·N^2),与顺序搜索的O(p^2N^2)相比，有很大的改善。

# 算法分析与详细设计

## 3.1顺序搜索算法

在-p到p的窗口范围内搜索最小MAD的宏块进行运动向量的匹配。

|  |
| --- |
| Min\_MAD=LARGE\_NUMBER;  For i=-p to p  For j=-p to p  {  Cur\_MAD=MAD(i,j);  If cur\_MAD<min\_MAAD  {  Min\_MAD=cur\_MAD;  u=i;//get the coordinates for MV  v=j;  }  } |

Matlab实现：

|  |
| --- |
| function [ u,v ] = SequenceSearch(target,ref,p,N,x0,y0)  %SequenceSearch 顺序搜索  % 依次搜索宏块，找到全局中的最优解  %target为目标帧，ref为参考帧，p指示窗口大小(2\*p+1)，  %N指示宏块大小，x0,y0为宏块中心坐标  %u v为找到的匹配宏块中心坐标  %扩大尺寸，防止窗口越界  target=EnlargeSize(target,p);  ref=EnlargeSize(ref,p);  %全搜索算法  LARGE\_NUM=9999999999999;  min\_MAD=LARGE\_NUM;  cur\_MAD=0;%当前的平均绝对误差  x=x0-N/2;  y=y0-N/2;%目标宏块的左上角顶点坐标，作为原点  diff=zeros(N,N);%误差图  for i=-p:p  for j=-p:p%(i,j)表示子块在参考帧搜索的位置  cur\_MAD=sum(sum(target(x:x+N,y:y+N)-ref(x +i:x+i+N,y+j:y+j+N)))/(N^2);  if cur\_MAD<min\_MAD  min\_MAD=cur\_MAD;  u=i;  v=j;  end;  end;  end;  %绘制误差图  for i=1:N  for j=1:N  diff(i,j)=target(x+i,y+j)-ref(x+u+i,y+v+j);  end;  end;    figure,imshow(target,[]);  title('target frame');  figure,imshow(ref,[]);  title('reference frame');  frmdiff=target-ref;  figure,imshow(frmdiff,[]);  title('difference between two frames');  figure,imshow(diff,[]);  disp('顺序搜索 运动向量 ');disp(u);disp(v);  title('difference in the macroblock');  end |

## 3.2 2D对数搜索

每次选取9个位置进行对比，找到最小MAD所在的中心点，将搜索范围折半，继续搜索，直到偏置值为1，进行最后一次搜索，确定匹配宏块。

|  |
| --- |
| Offset=;  Specify 9 macroblocks within the search window in the Reference frame,  They are centered at (x0,y0) and separated by offset horizontally and/or vertically;  WHILE last!=TRUE  {  Find one of the 9 specified macroblocks that yields minimum MAD;  If offset=1 then last=TRUE;  Offset=;  Form a search region with the new offset and new center found;  } |

Matlab实现：

|  |
| --- |
| function [ u,v ] = LogSearch( target,ref,p,N,x0,y0 )  %LogSearch 2D对数搜索函数  % 进行2D对数搜索，每次选取9个位置进行对比  %扩大尺寸，防止窗口越界  target=EnlargeSize(target,p);  ref=EnlargeSize(ref,p);  %2D对数搜索  LARGE\_NUM=9999999999999;  min\_MAD=LARGE\_NUM;  cur\_MAD=0;%当前的平均绝对误差  x=x0-N/2;  y=y0-N/2;%目标宏块的左上角顶点坐标，作为原点  diff=zeros(N,N);%误差图  offset=ceil(p/2);  %9个点的位置自行定义。  %这里以target里目标宏块的中心位置作为中心扩展开来  %cp0 cp1 cp2  %cp3 cp4 cp5  %cp6 cp7 cp8  cp=cell(1,9);  cp0.val=ref(x0-offset,y0-offset);cp0.x=x0-offset;cp0.y=y0-offset;  cp1.val=ref(x0,y0-offset);cp1.x=x0;cp1.y=y0-offset;  cp2.val=ref(x0+offset,y0-offset);cp2.x=x0+offset;cp2.y=y0-offset;  cp3.val=ref(x0-offset,y0);cp3.x=x0-offset;cp3.y=y0;  cp4.val=ref(x0,y0);cp4.x=x0;cp4.y=y0;  cp5.val=ref(x0+offset,y0);cp5.x=x0+offset;cp5.y=y0;  cp6.val=ref(x0-offset,y0+offset);cp6.x=x0-offset;cp6.y=y0+offset;  cp7.val=ref(x0,y0+offset);cp7.x=x0;cp7.y=y0+offset;  cp8.val=ref(x0+offset,y0+offset);cp8.x=x0+offset;cp8.y=y0+offset;  cp{1}=cp0;  cp{2}=cp1;  cp{3}=cp2;  cp{4}=cp3;  cp{5}=cp4;  cp{6}=cp5;  cp{7}=cp6;  cp{8}=cp7;  cp{9}=cp8;  last=0;  while(last~=1)  for i=1:9  cur\_MAD=sum(sum(target(x:x+N,y:y+N)-ref(cp{i}.x-N/2:cp{i}.x+N/2,cp{i}.y-N/2:cp{i}.y+N/2)))/(N^2);  if cur\_MAD<min\_MAD  min\_MAD=cur\_MAD;  u=cp{i}.x;  v=cp{i}.y;  end;  end;  if offset==1  last=1;  end;  offset=ceil(offset/2);  cp{1}.x=u-offset;cp{1}.y=v-offset;  cp{2}.x=u;cp{2}.y=v-offset;  cp{3}.x=u+offset;cp{3}.y=v-offset;  cp{4}.x=u-offset;cp{4}.y=v;  cp{5}.x=u;cp{5}.y=v;  cp{6}.x=u+offset;cp{6}.y=v;  cp{7}.x=u-offset;cp{7}.y=v+offset;  cp{8}.x=u;cp{8}.y=v+offset;  cp{9}.x=u+offset;cp{9}.y=v+offset;    end;  %绘制误差图  for i=1:N  for j=1:N  diff(i,j)=target(x+i,y+j)-ref(u-N/2+i,v-N/2+j);  end;  end;    figure,imshow(target,[]);  title('target frame');  figure,imshow(ref,[]);  title('reference frame');  frmdiff=target-ref;  figure,imshow(frmdiff,[]);  title('difference between two frames');  figure,imshow(diff,[]);  disp('对数搜索 运动向量 ');disp(u-x0);disp(v-y0);  title('difference in the macroblock');    end |

## 3.3主体程序

读取视频文件，提取指定帧。

首先查询了matlab的VideoReader方法，得到的obj结构体有如下元素：

|  |
| --- |
| Name - -视频文件名  Path – 视频文件路径  Duration – 视频的总时长（秒）  FrameRate - -视频帧速（帧/秒）  NumberOfFrames – 视频的总帧数  Height – 视频帧的高度  Width – 视频帧的宽度  BitsPerPixel – 视频帧每个像素的数据长度（比特）  VideoFormat – 视频的类型, 如 ‘RGB24’.  Tag – 视频对象的标识符，默认为空字符串”  Type – 视频对象的类名，默认为’VideoReader’. |

Read(obj,i)可以获取该视频对象的第i帧。

但是该方法不支持YUV格式的视频，搜索后得知，必须自己编写读取YUV格式视频的函数。

打开文件后，计算一帧图像中的总图像个数，通过fseek函数定位到视频中，分别创建Y U V分量，将每一帧的对应数值存放进分量中。

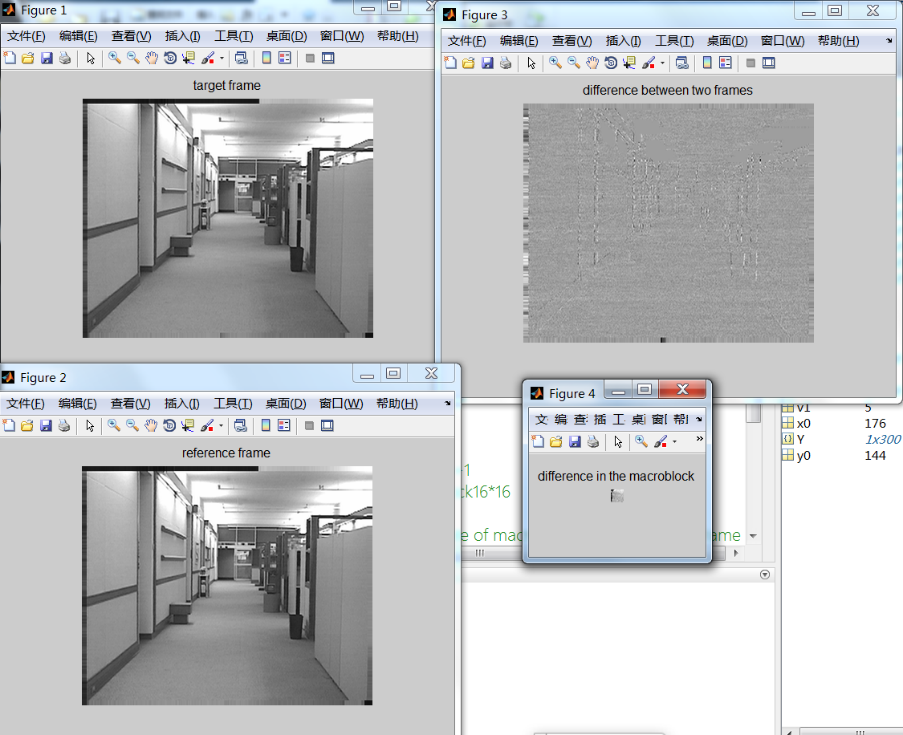
|  |
| --- |
| function [ Y,U,V ] = yuv\_import(filename,dims,numfrm,startfrm)  %yuv\_import 读取YUV格式的视频  % 读入YUV视频分别获得Y U V分量的数据  %filename为文件路径，dims为图像的分辨率大小[width height]，numfrm为帧数  fid=fopen(filename,'r');  if(fid<0)  error('File does not exist!');  end;  Yd=zeros(dims(1),dims(2));  UVd=zeros(dims(1)/2,dims(2)/2);    frelem=numel(Yd)+2\*numel(UVd);%一帧图像总的像素个数  %if we have the 'starting frame'  if(nargin==4)  fseek(fid,startfrm\*frelem,0);  end;  Y=cell(1,numfrm);  U=cell(1,numfrm);  V=cell(1,numfrm);  for i=1:numfrm  Yd=fread(fid,[dims(1) dims(2)],'uint8');  Y{i}=Yd';  UVd=fread(fid,[dims(1)/2 dims(2)/2],'uint8');  U{i}=UVd';  UVd=fread(fid,[dims(1)/2 dims(2)/2],'uint8');  V{i}=UVd';  end;  end |

# 实验结果分析

## 4.1 顺序搜索算法

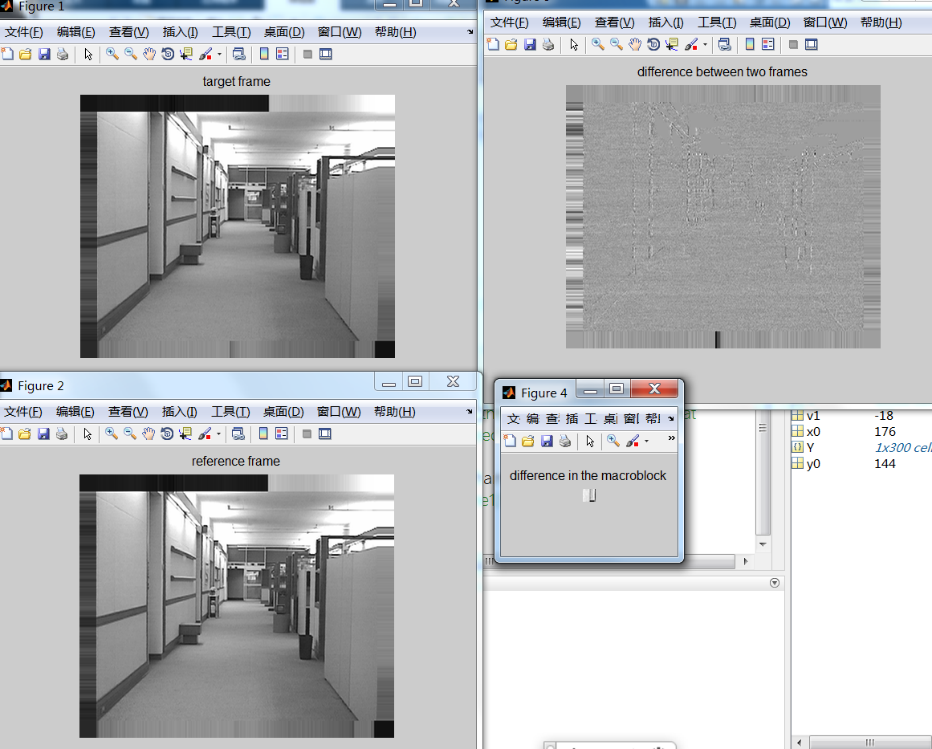
|  |  |  |  |
| --- | --- | --- | --- |
| 参考帧 | 窗口大小参数P | 宏块大小N | 所得运动向量 |
| 2 | 5 | 16 | 5 5 |
| 2 | 20 | 16 | -6 -18 |
| 2 | 5 | 32 | 5 5 |
| 10 | 5 | 16 | 5 5 |
| 100 | 5 | 16 | 5 1 |
| 3 | 5 | 16 | 5 5 |
| 5 | 5 | 16 | 5 5 |
| 20 | 5 | 16 | 5 5 |
| 50 | 5 | 16 | 5 1 |

参考帧为第二帧，选取中央的宏块，p=5,N=16,用顺序搜索算法得到的结果是：

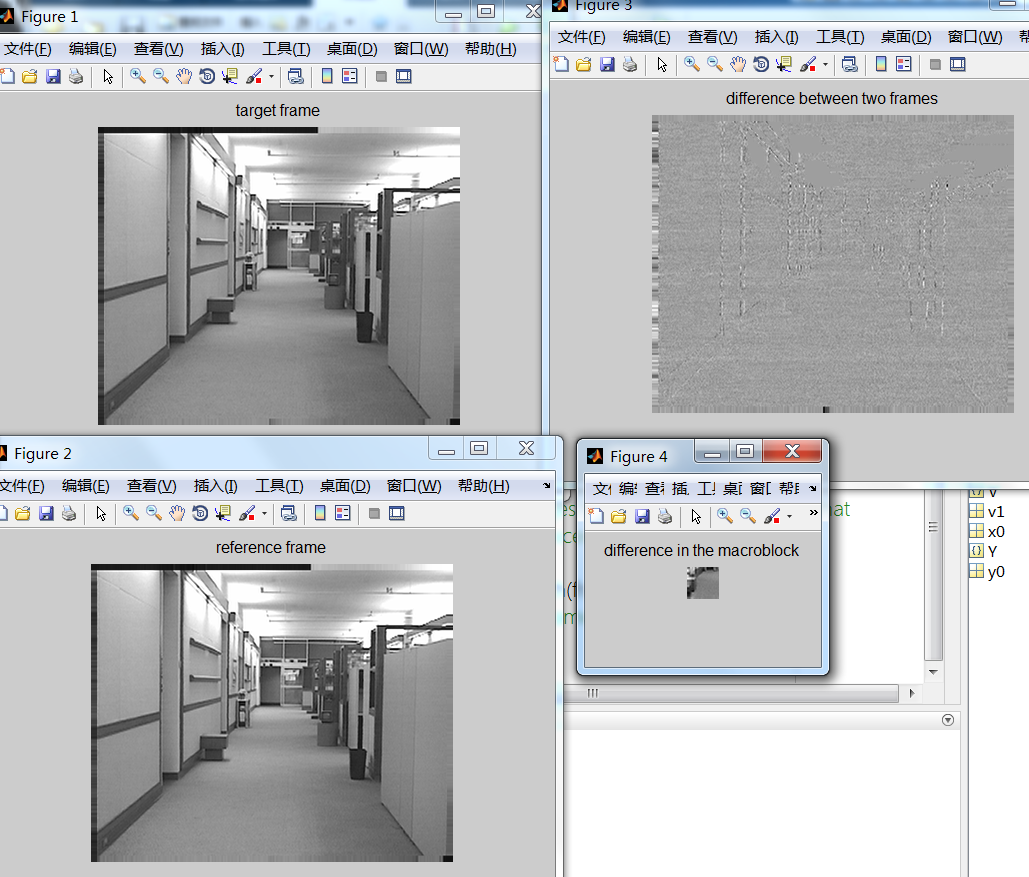


运动向量为(5,5)。

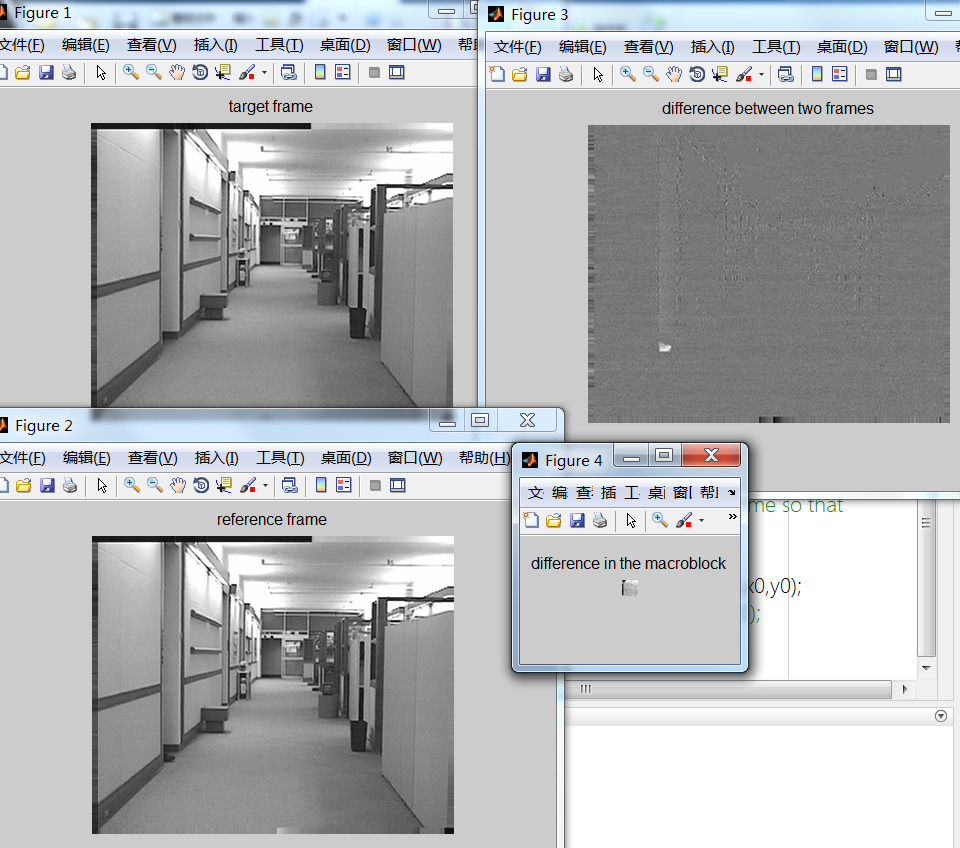
p=20,N=16，得出运动向量为(-6,-18)：



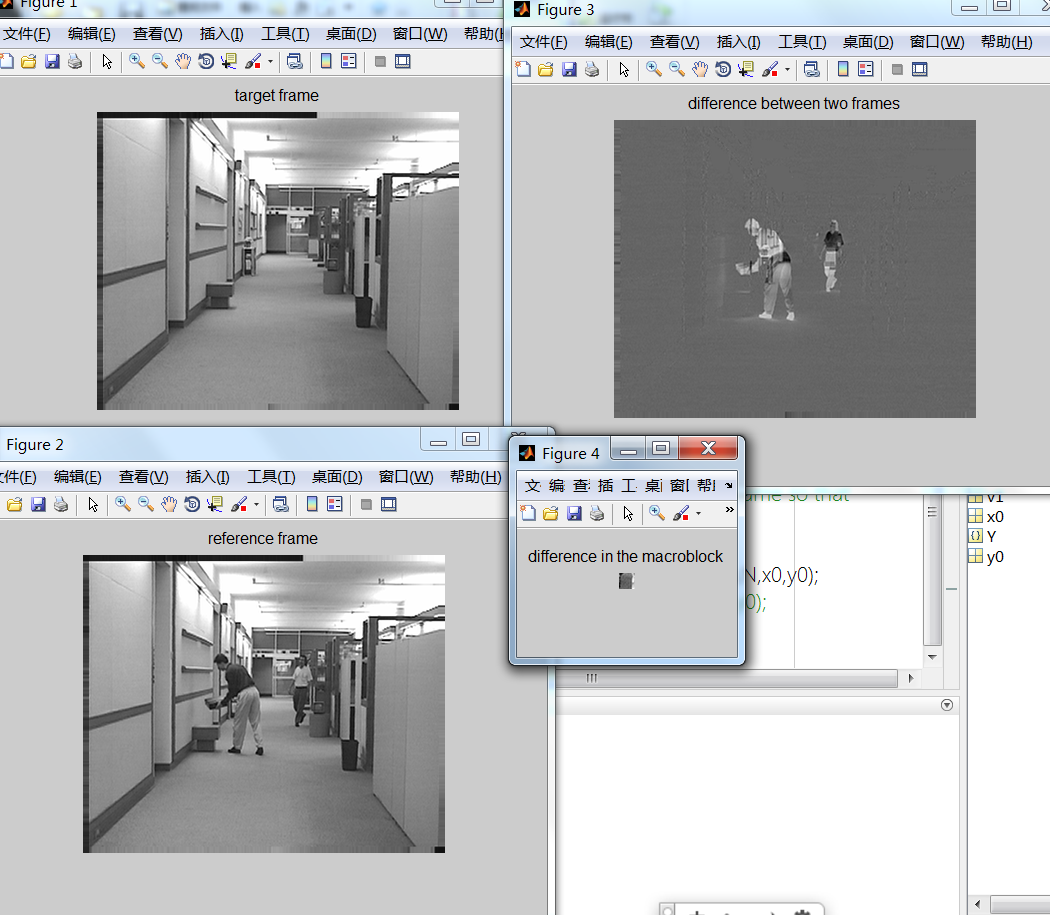
p=5,N=32,得到运动向量为（5,5）:



参考帧为第10帧，p=5,N=16,运动向量为(5,5)：



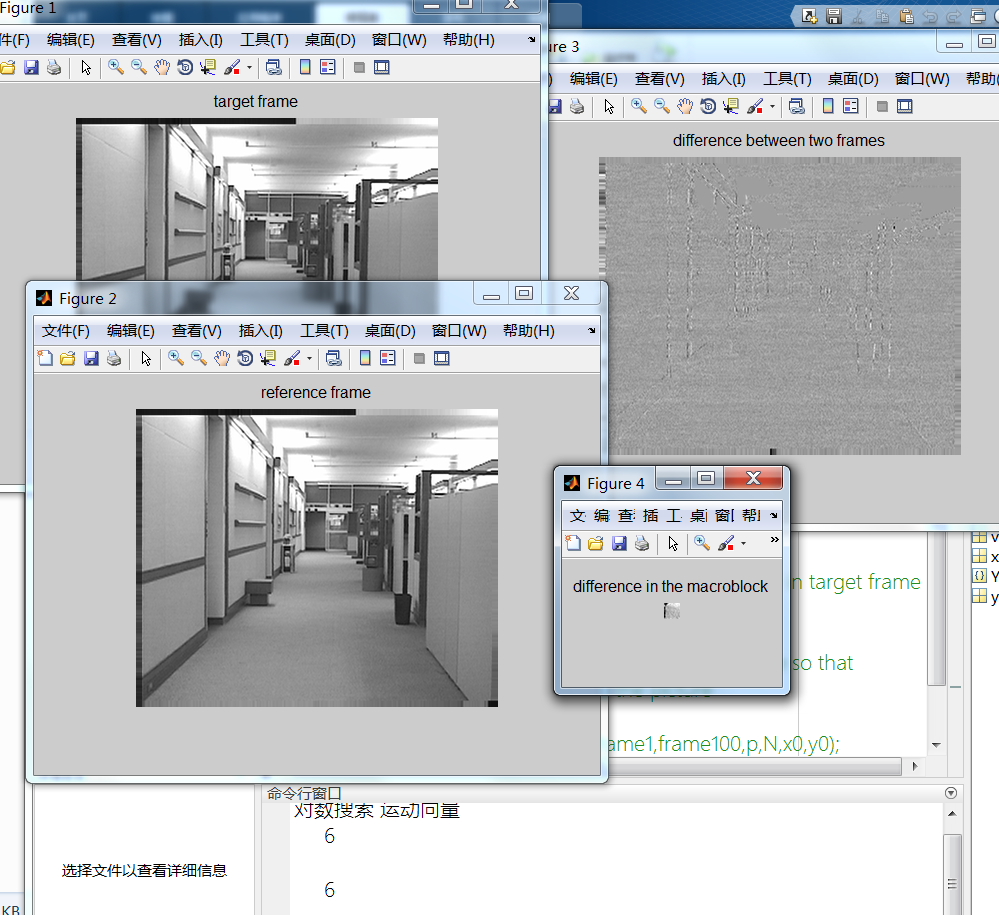
参考帧为第100帧，运动向量为(5,1)：



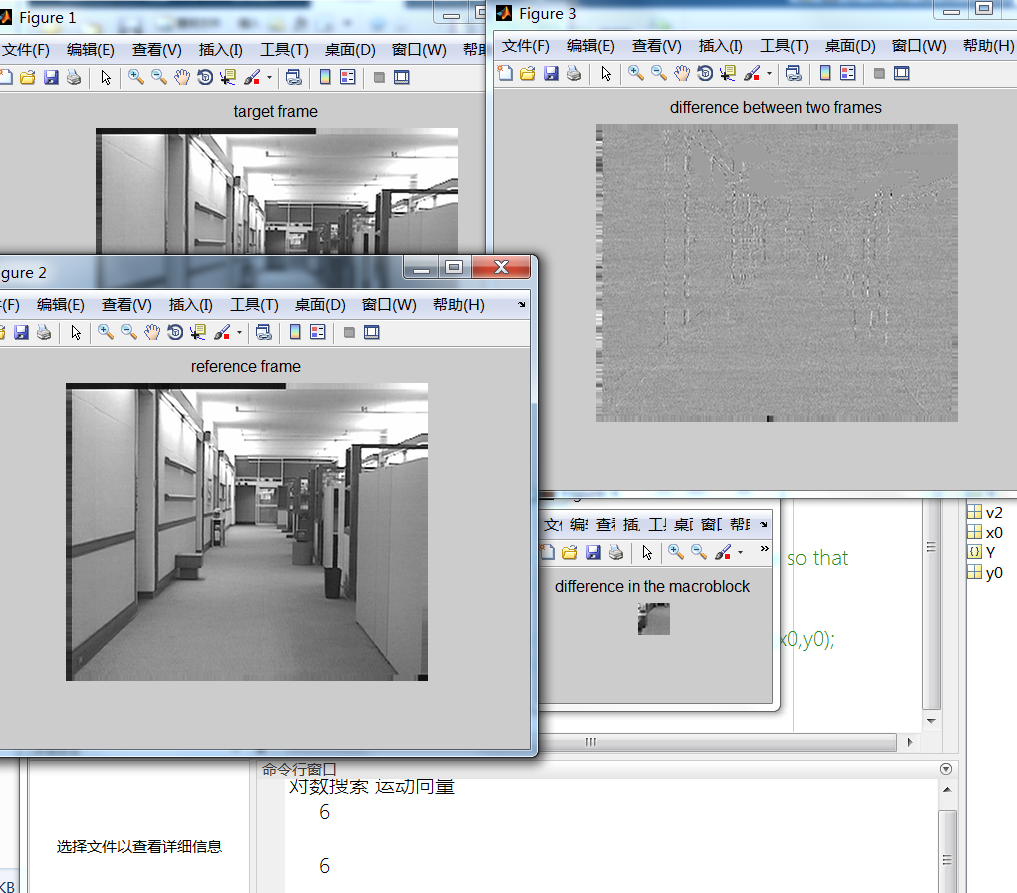
## 4.2 2D对数搜索

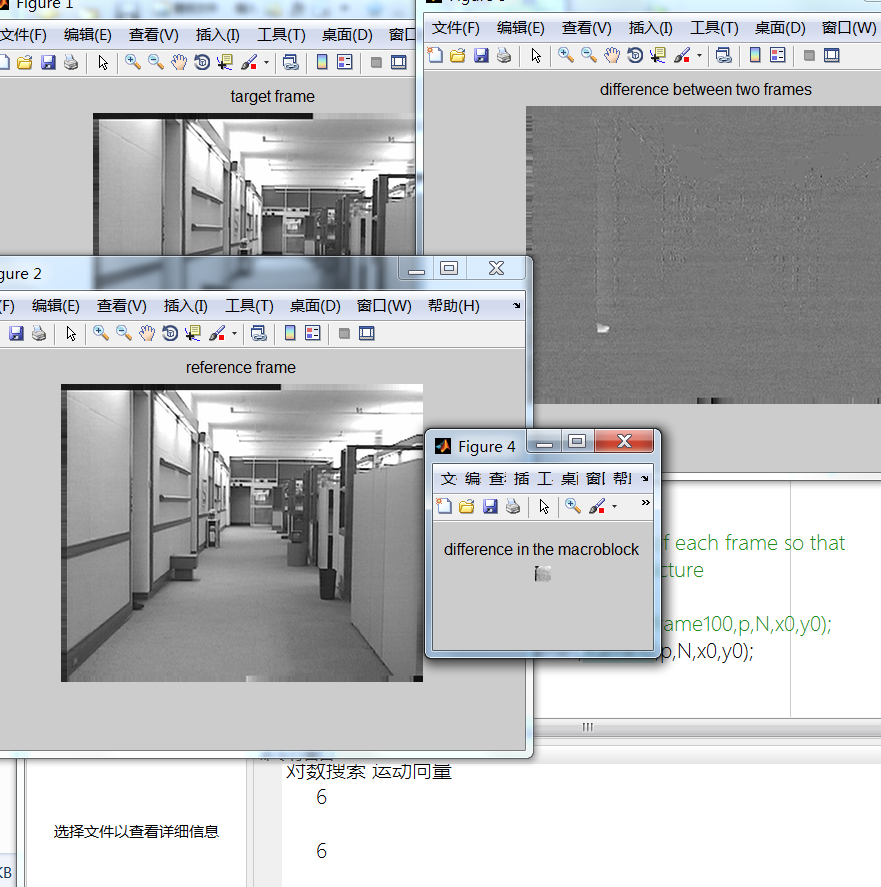
|  |  |  |  |
| --- | --- | --- | --- |
| 参考帧 | 窗口大小参数P | 宏块大小N | 所得运动向量 |
| 2 | 5 | 16 | 6 6 |
| 2 | 20 | 16 | -6 -18 |
| 2 | 5 | 32 | 6 6 |
| 10 | 5 | 16 | 6 6 |
| 100 | 5 | 16 | 6 0 |
| 3 | 5 | 16 | 6 6 |
| 5 | 5 | 16 | 6 6 |
| 20 | 5 | 16 | 6 6 |
| 50 | 5 | 16 | 6 1 |
| 2 | 10 | 16 | 11 11 |
| 2 | 50 | 16 | -26 13 |

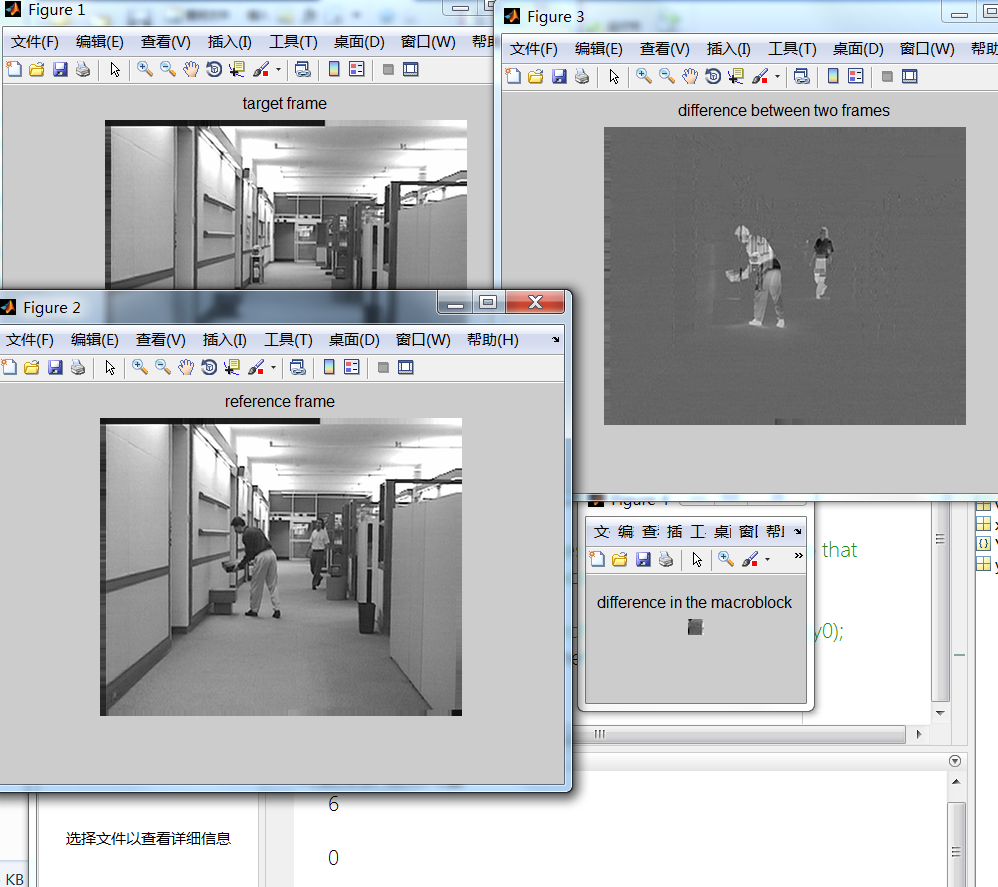
对应的图像显示：











## 4.3综合分析

通过多组实验数据的比较，可以发现，宏块大小对运动向量的预测结果影响较小，对运算速度产生一定的影响，宏块扩大后运算量增大，速度减慢。而窗口大小对运动向量的预测结果影响是较大的，不同的窗口大小会导致预测的运动向量发生较大幅度的改变，这可能是因为相较于宏块大小，窗口过大时，较为单一的画面会引导算法找到不正确的匹配宏块，窗口过小时，可能会找不到匹配的宏块，而选取了其他相似的宏块。

对于2D对数搜索算法，点的选取对实验结果有一定影响，如果选取的点很密集，范围很小，可能会导致较差的结果，找不到正确的匹配宏块、从而得到错误的运动向量。选取的点分布较散时，能够通过折半查找迅速找到更接近匹配宏块的区域，这是因为区域之间的亮度值具有一定的连续性。

顺序搜索是一种全局搜索，得到的结果是最优的，但是运算速度很慢，随着宏块、窗口大小的增大，运算量会迅速增大，代价极高。而2D对数搜索虽然未必得到最优结果，性能却较好，能够在较短的时间内找到与目标宏块近似匹配的宏块。

# 五、实验总结

通过本次实验，了解了YUV格式视频的读取与处理，深入理解了两种运动向量搜索的算法，并进行了运动向量的预测。提取出Y通道的灰度值进行宏块的匹配与运动向量的预测，因为灰度值能够基本代表帧中像素的情况，从而反映运动趋势。通过改变窗口大小、宏块大小、特征点的选取等参数，对实验结果进行了一定的比对分析，得到了以下结论：（1）宏块大小对于运动向量的搜索结果产生的影响较小，对运算速度会造成一定影响。（2）窗口大小对于运动向量的顺序搜索结果产生的影响较大，时间开销增大的同时、精确度也降低；对于对数搜索时间方面的影响不大，但是精确度也降低。（3）点的位置的选取对于对数搜索的实验结果有一定影响，当选取的点过于密集时，可能会产生错误的匹配结果。

另外，虽然Y通道的亮度值基本可以代表图像特征，也可以对UV通道进行运动向量的搜索，因为颜色也是重要的图像特征之一。搜索方法是类似的，只是针对不同采样的视频，UV通道的数据量不同。