

实验报告

实验题目	姓名	学号
h.261运动矢量搜索算法	张桓	191220156

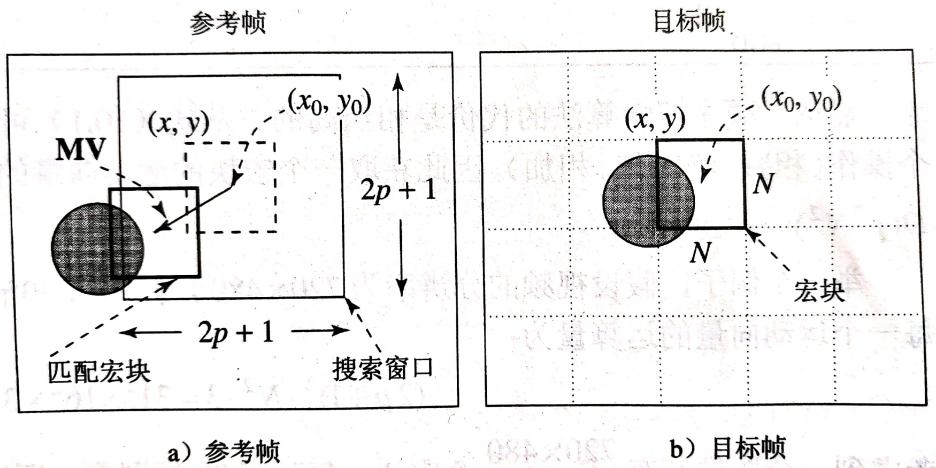
实验原理

基于运动补偿的视频压缩

一段视频可以看作在时间维度上顺序播放的一系列图像，由于视频的帧率比较高，因此连续帧的图像内容是很相似的，除非有移动比较快的物体。也就是说，视频在时间维度上存在信息冗余。

因此我们不必将每帧视频图像都作为一副新的图像进行编码，而是将当前帧和其他帧的差值进行编码。但是如果简单用整个图像的像素点作差的话，无法实现高压缩。由于帧间图像主要差别是图像内物体运动造成的，所以可以通过在这些帧里探测相应像素或区域的移动并测量它们的差值来“补偿”这些运动生成器。这就是基于运动补偿 (MC) 的压缩算法，算法主要分三步：

- 运动估计（运动向量的查找）
- 基于运动补偿的预测
- 预测误差的生成——差值



为了提高效率，我们把每张图像分为大小为 $N \times N$ 的宏块 (MB)，一般 N 取 16。当前帧称为目标帧，目标宏块由参考宏块预测生成，因此我们要在目标帧中的宏块和参考帧（前向帧或后向帧）中最相似的宏块寻找匹配。目标宏块和参考宏块间的位移就是运动向量MV。

搜索运动向量

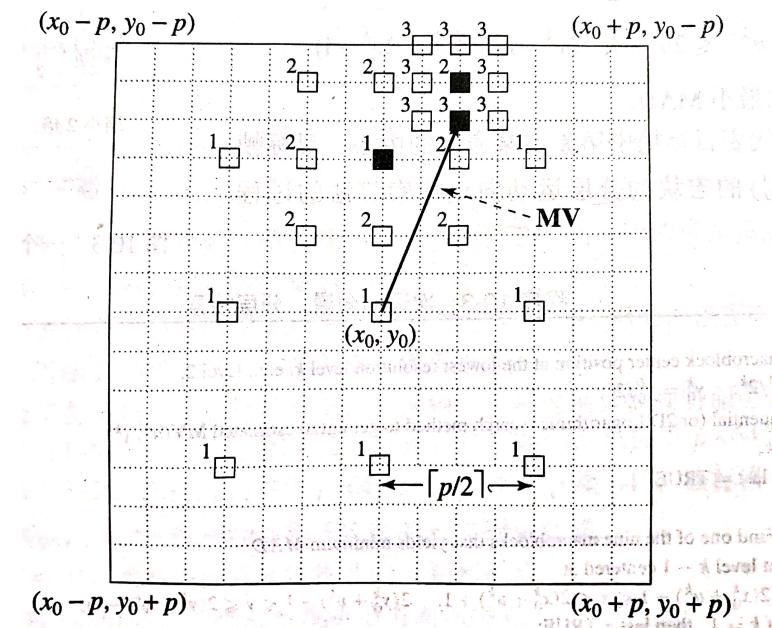
寻找运动向量的过程就是在参考帧中搜索最匹配的目标帧宏块的那个宏块，我们限制搜索位移的区域为 $[-p, p]$ 范围内。描述宏块间的匹配程度有多种方式，作业给出的代码是使用绝对误差和SAD，在搜索窗口内寻找最匹配的宏块方法有很多，最简单暴力的方法就是代码给出的全搜索。

- 全搜索

遍历整个参考帧中大小为 $(2p + 1) \times (2p + 1)$ 的窗口，每个位置上的宏块计算匹配程度，找到最匹配的那个即**SAD最小的**。参考代码是螺旋遍历的，顺序不同但本质上还是全搜索，代价为 $(2p + 1)^2 \times N^2 \times 3$ 。

- 对数搜索

全搜索虽然效果好，但是代价太大。对数搜索的过程需要经过多次迭代，类似**折半查找**过程。如下图：



初始时，搜索窗口9个位置标记为“1”，作为搜索的起始位置。计算这9个位置宏块的匹配程度，选出SAD最小的那个作为**下一轮搜索区域**的中心，同时搜索步长减半，在图中标记为“2”，依此类推。

这个过程是启发式的，它假设图像内容具有一般意义上的连贯性——在搜索窗口中的图像不会随机变化，首先**粗略**找到一个匹配的位置，之后迭代在该基础上不断缩小搜索范围**精细**匹配结果。

代价为 $8 \times (\lceil \log_2 p \rceil + 1) + 1$ 。

实验方案

修改 Mot_est.cpp 文件中的 MotionEstimation 函数。将原来螺旋搜索的代码改成对数搜索的代码如下：

```

1  /* 对数搜索 */
2  int offset = ceil(sxy / 2.0);
3  int scx = x_curr + xoff, scy = y_curr + yoff; // 搜索窗口中心坐标
4  while (offset > 1) {
5      // 每一个offset下计算九个位置
6      i = scx - offset;
7      j = scy - offset;
8      for (k = 0; k < 8; k++) {
9          if (i >= ilow && i <= ihigh && j >= jlow && j <= jhigh) {
10             /* 16x16块的整象素MV */
11             ii = search_area + (i - ilow) + (j - jlow) * h_length;
12             sad = SAD_Macroblock(ii, act_block, h_length, Min_FRAME[0]);
13             if (sad < Min_FRAME[0])
14             {
15                 scx = i; // 更新下一步搜索窗口中心坐标

```

```

16         scy = j;
17         MV_FRAME[0].x = i - x_curr;
18         MV_FRAME[0].y = j - y_curr;
19         Min_FRAME[0] = sad;
20     }
21 }
22 if (k < 2) {
23     i += offset;
24 }
25 else if (k < 4) {
26     j += offset;
27 }
28 else if (k < 6) {
29     i -= offset;
30 }
31 else {
32     j -= offset;
33 }
34 }
35 offset = ceil(offset / 2.0);
36 }

```

其中变量 `scx` 和 `scy` 为搜索范围为 `offset` 下 **搜索窗口的中心坐标**。每个搜索窗口计算9个位置（实际为8个）的宏块 SAD，找出最小的那个作为下一轮的搜索窗口中心，直到 `offset <= 1` 搜索结束。

实验结果

- 全搜索

原来螺旋搜索耗时较大：

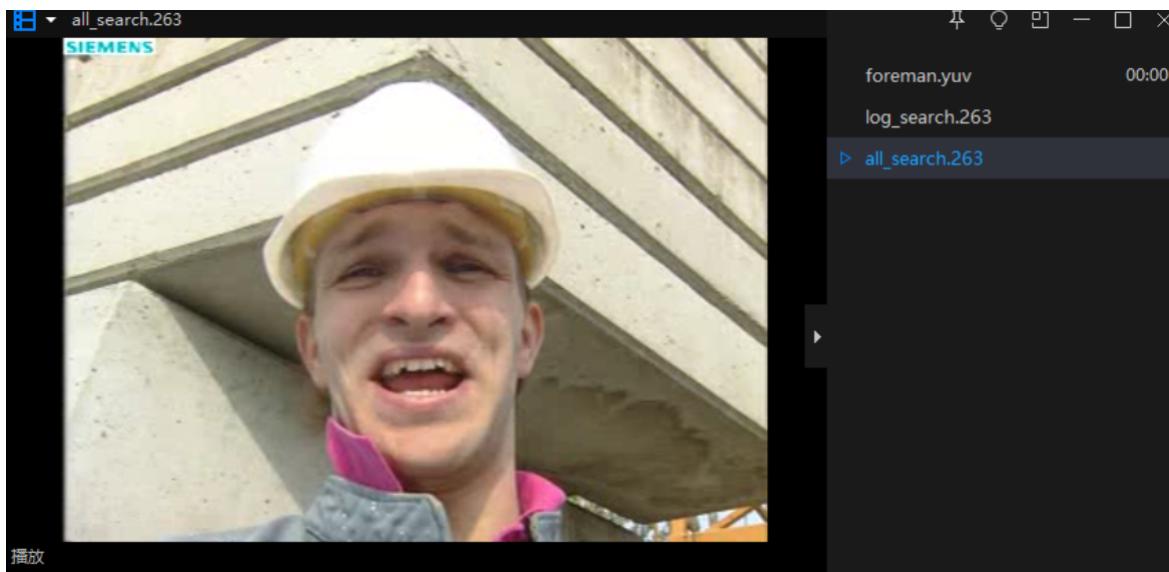
```

Starting to convering velocity file file.
Encoding format: CIF (352x288)
*****
**the total time is 33.694S
*****

```

但是理论上讲显式效果比较好，因为是全部搜索找到最匹配的那个。

输出显式：

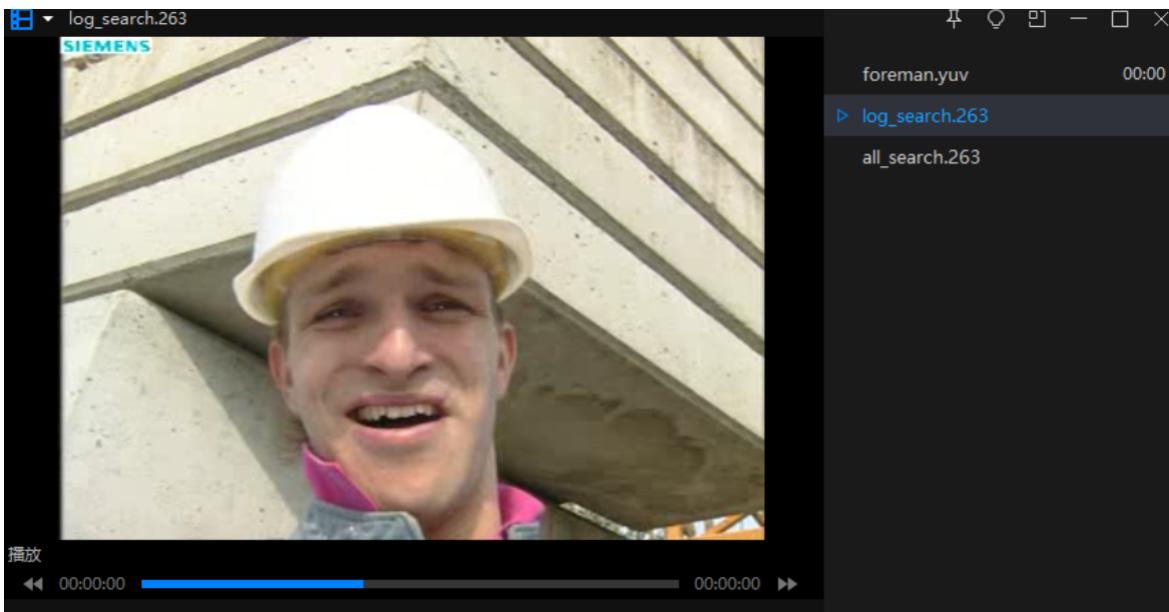


- 对数搜索

改成对数搜索后，程序执行时间显著变少，比全搜索快了接近5倍

```
Starting to convering velocity file file.  
Encoding format: CIF (352x288)  
*****  
**the total time is 7.673S  
*****
```

输出显式：



实际观看起来对数搜索生成的视频和全搜索生成的视频几乎没有区别，效果还是非常好的，并且程序耗时要短得多。

实验小结

- 心得体会

用运动补偿来进行视频压缩的思想本身就很妙，因为考虑到视频中图像在时间上的**连贯性**。而寻找最优匹配宏块的方法，从一开始的暴力遍历，到之后的对数搜索，乃至层次搜索，体现了对图像内容**连贯性**的考虑。

正是由于连贯性，对数搜索或层次搜索最开始的几次迭代寻找出的大致位置是正确的，也就是“方向对了”，剩下的迭代则可以看作是优化的过程，找到更加精细的匹配位置。

- 建议

代码的注释比较少，尤其是函数的参数没有详细说明，只介绍了函数功能，因此实验时很多时间都花费在了理解代码，推测参数意义上，可以考虑对一些参数做注释。

【参考文献】

多媒体技术教程[M].机械工业出版社，2007：200-205