基本图像处理算法的优化过程研究

徐启航[1] 游安清[1,2] 马 社[1] 崔云俊[1]
(中国工程物理研究院应用电子学研究所 绵阳 621900)[1]
(高功率微波技术国防科技重点实验室 绵阳 621900)[2]

摘 要 针对实时、高效的图像处理任务的实现,以视频图像中基于模板匹配的运动目标开环跟踪算法为例,对其基于 Matlab 原型算法的跟踪性能进行评估,具体介绍了对此算法的多级优化过程。从 Matlab 原型算法开始,主要从以下两方面进行优化:在提高实时处理速度方面,采用 C 语言提速、乘法提速、Release 提速、合并运算、CUDA 架构提速等 10 级以上的优化策略;在提高正确率方面,采用简单的多模板策略。测试结果表明,算法速度提高了 200 多倍,最终达到 30Hz 的实时处理速度,并且大幅提高了跟踪正确率。

关键词 模板匹配,提速,多模板策略,目标跟踪,图像处理,CUDA 架构
中图法分类号 TP751.1 **文献标识码** A

Study on Optimizations of Basic Image Processing Algorithm

XU Qi-hang[1] YOU An-qing[1,2] MA She[1] CUI Yun-jun[1]
(Institute of Applied Electronics, China Academy of Engineering Physics, Mianyang 621900, China)[1]
(National Key Laboratory of Science and Technology on HPM Technology, Mianyang 621900, China)[2]

Abstract To provide useful references for the implementation of real-time and excellent image processing task, taking the open loop tracking algorithm based on template matching in video image as an example, the tracking performance of the algorithm based on MATLAB prototype algorithm was evaluated, and the multi-level optimization process was presented. Starting with an MATLAB prototype algorithm, we began to optimize mainly in the two aspects as below. In improving real-time processing speed, more than 10 levels of optimizations are applied to speed up the algorithm, including C language speeding, multiplication speeding, release speeding, merge operation, CUDA speeding and so on. In terms of improving the correct rate, simple multi-pattern strategy is used. Testing results indicate that the algorithm reaches performance of 30Hz real-time image process and tracking rate of the algorithm is also promoted greatly.

Keywords Pattern match, Speed promotion, Multi-pattern strategy, Targets tracking, Image processing, CUDA architecture

1 概述

图像跟踪是光电探测领域的一个常见议题。国内外大量研究机构和科研院校对视频图像中的运动目标的检测与跟踪开展了广泛研究[1-4]。国内代表性的单位包括清华大学、西安交通大学、哈尔滨工业大学、中科院自动化研究所、华中理工大学、成都光电所等。一个良好的运动目标跟踪系统往往要依赖一台复杂的光电经纬仪和一套实时的 DSP 图像处理系统[5]。对于装备应用领域,这些都不是问题;但做基础研究的实验室却不一定具备充足的硬件条件,仅仅可利用一台计算机及少量的硬件资源,只具备一种算法研究或仿真验证实验初期的简单条件。在这种条件下,实现实时、高效的图像处理算法并不容易,需要从算法的每个环节上下功夫,才能逐渐缓慢地提高算法的速度与效率。近年来,Nvidia 公司推出的 CUDA 构架的高速显卡为基于上位机的实时图像处理提供了一种可能的途径。灵活掌握和综合运用各种优化策略,可以将一个图像处理算法的速度提高数十至上百倍。本文正是基于以上考虑,从一个 Matlab 原型算法开始逐级优化,最终达到实时的性能。

2 算法原型

本文以一个基本的目标跟踪算法——模板匹配法为出发点[6],首先利用 Maltab 内置矩阵运算和图像处理函数的优点,快速实现原理算法,算法描述如下:

1)在序列视频图像的第 1 帧中人为指定一个大小为 a 的运动目标,并以指定位置为中心提取子图像当作目标模板 pPattern;

2)从第 2 帧开始(至第 n 帧),在全图 pI 中搜索与目标模

本文受国防微波重点实验室基金(2015HPM-11)资助。
徐启航(1989-),男,硕士生,主要研究方向为图像处理和目标跟踪,E-mail:xuqihang000@163.com;游安清(1975-),男,博士,副研究员,主要研究方向为计算机视觉;马 社(1978-),男,硕士,副研究员,主要研究方向为力学仿真设计;崔云俊(1975-),男,博士,副研究员,主要研究方向为计算机视觉。

板累差最小的子图像 pSubI 作为对目标的匹配，输出此最佳匹配的子图像的位置即为第 2 帧(第 n 帧)中对目标的跟踪位置；

3)为了减小成像过程中光照强度随时间变化而产生的影响，对目标模板 pPattern 和子图像 pSubI 都进行去均值处理。

据此算法，得到匹配算法原型的伪代码：

```
minDiff=MAX_NUMB;// 预置最小累差为一大值
for y=1:mageHeight-a{ // 遍历图像的各行
  for x=1:imageWidth-a{ // 遍历图像的各列
    for yy=1:a{ // 对每个遍历位置
      for xx=1:a{ pSubI[yy*a+xx]=pI[(y+yy)*imageWidth+
      (x+xx));}} // 取与模板等大的子图像
    sum=0;// 子图像内灰度和预置 0
    for yy=1:a{ // 遍历子图像的各行
      for xx=1:a{ // 遍历子图像的各列
        sum+=pSubI[yy*a+xx];}}// 计算子图像内的灰度和
    mean=sum/(a*a);// 计算子图像内的灰度均值
    for yy=1:a{ // 遍历子图像的各行
      for xx=1:a{ // 遍历子图像的各列
        pSubI[yy*a+xx]=mean;}} // 对子图像去均值
    diff=0; // 子图像与模板的灰度累差预置 0
    for yy=1:a{ // 遍历子图像的各行
      for xx=1:a{ // 遍历子图像的各列
        diff+=fabs(pSubI[yy*a+xx]-pPatern[yy*a+xx]);}}
              // 计算子图像与模板图像的灰度累差
      if minDiff>diff
        // 过滤出最小灰度累差及其对应的子图像位置
          { minDiff=diff;
            xStar=x;
            yStar=y;}
}}output xStar,yStar
// 输出最小累差对应的子图像位置作为对目标的匹配
```

根据上面的算法原型，用 Matlab 编程实现后，对于一个 640×480×121 帧的视频图像序列，在首帧中指定一个大小为 21×21 的目标后，在后续帧中得到的跟踪性能是：图像的处理速度为 8s/帧；跟踪正确率为 73%(120 帧中正确跟踪 87 帧，即 87/120)。任务示意图如图 1 所示。

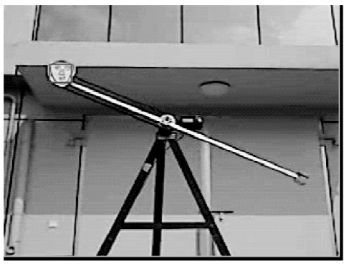

图 1 基于模板匹配的开环目标跟踪

3 逐级优化

3.1 第一级优化：使用高效的开发语言

显然，基于 Matlab 适于矩阵运算且内置大量图像处理函数的优势，原型算法可能得到很快验证。但是，由于 Matlab 平台是一种解释性脚本语言，本就不是为实时应用而开发的，因此其运行效率比较低，8s/帧的处理速度离实时性要求尚远。

于是，在用 Matlab 验证原型算法基本可行后，改用 VC 开发平台重新进行实现。经编译后，得到的跟踪正确率与 Matlab 版完全相同，但图像处理速度提高到 2.8s/帧，相对提速 2.9 倍。

3.2 第二级优化：用合并访问代替逐点访问

在计算机底层操作机制中，合并访问是指对具有一定排列规律和对齐状态的多字节内存空间进行一次性访问，这比逐字节访问速度快。原型算法中 Line4－Line6 是从全图像 pI 中逐像素取出大小为 $a\times a$ 的子图像 pSubI，其中用到了二重循环，而多重循环往往是算法中很耗时的环节。由于内循环其实是对内存中一段连续的像素值进行取值，因此可以改用 memcpy()函数取代内循环(即 Line5 行)，一次性地从全图中拷贝 a 个像素的值赋给子图像 pSubI，而不是逐个像素赋值。代码如下：

```
Line4:for yy=1:a{
Line5:  memcpy(pSubI,pI,a*sizeof(float));
Line6:  pSubI+=a;pI+=imageWidth;}
```

经此改进后，每帧处理时间减少 0.1s。

3.3 第三级优化：用加法取代乘法

在计算机的底层指令中，完成一次加法需要 1 个时钟周期，完成一次乘法需要 4 个时针周期。因此，将乘法运算改为加法运算实现，是一种常用的提速手段。仔细观察前面的算法原型，Line6，Line10，Line14，Line18 因受各自之外的循环控制，都有大量的乘法寻址运算。因为这些地址具有一定的连续性，所以完全可以用地址递增来代替地址计算。以 Line10 的乘法寻址为例，可以将 Line8－Line10 改为如下的加法寻址：

```
  p=pSubI;// 设置一个指针指向子图像首地址
  Line8:for yy=1:a{ // 遍历子图像的各行
  Line9:    for xx=1:a{ // 遍历子图像的各列
  Line10:         sum+=*p++;}}
// 计算子图像内的灰度和，并递增子图像取址指针
```

对原型算法中的 Line6，Line10，Line14，Line18 都做此改进后，算法运行时间减少 0.3s。

3.4 第四级优化：循环降维

众所周知，循环跳转是影响速度的一个常见环节。在原型算法中存在多处二重循环，如 Line2&3，Line4&5，Line8&9，Line12&13，Line16&17，尤其是后 4 对，都在用二重循环进行连续地址上的操作，这完全是可以降维的。以 Line12&13 为例，可以将它们改为：

```
Line12:for i=1:a*a{
          // 遍历子图像的各个像素
Line13:     pSubI[i]-=mean;}
// 对子图像去均值
```

经此优化后，算法运行时间减少 0.1s。

3.5 第五级优化：用自定义宏代替简单的系统函数

在 VC 中，用求“绝对值函数 fabs()”、“求最小值函数 min()”、“求最大值函数 max()”等一些系统函数来完成一些简单或基本的数学运算。由于系统函数要对参与运算的各种情况考虑周全(整数的、小数的、浮点的、双精度的…都要兼容)，导致其实现过程比较罗嗦，很多环节其实并非用户所需

要，造成了不必要的费时；同时，这些函数都是用链接库调用来实现的。一个常用的解决办法是：自定义一些宏，用最简短的语句实现这些系统函数的功能。比如，求绝对值的 fabs() 函数可以用下面的宏来代替：

```
#define FABS(x) ((x)>=0?(x):(-x))
```

注意：宏定义必须要用足括号，如果将上面的宏定义改为“#define FABS(x) x>=0?x:-x”或“#define FABS(x) (x>=0?x:-x)”，则不能保证一定会得到 x 的绝对值。

用自定义宏代替原型算法中的系统函数 fabs() 后，算法运行时间减少 0.7s，速度提升幅度较大。至此，图像处理算法的速度已经提高到 1.6s/帧，相对提速 1.8（即 2.8/1.6）倍。

3.6　第六级优化：合并运算

这里的“合并运算”是指在一次循环中做尽量多的事情，以减少多次循环。在前面的原型算法中，Line12－Line14 和 Line16－Line18 分别实现了子图像去均值和子图像与模板图像的累差，这两次循环可以通过在去均值的同时做累差的方法合并成一次。代码如下：

```
diff=0;pS=pSub;pP=pPattern;
Line12:  for i=1:a*a{
           // 遍历子图像和模板图像的各个像素
Line13:    diff+=FABS(*pS++-*pP++);}
```

经此改进后，算法运行时间减少 0.3s。

3.7　第七级优化：使用序贯机制

这里的“序贯机制”是指逐像素累计计算子图像与模板图像的灰度差值之和时，一旦发现累差超过了已有的最小累差，则立即放弃此子图像，不必对其尚未累计的像素再做计算（因为再计算，只会使累差继续增大，这样的子图像不可能是累差最小的子图像）。这一措施的主要目的是防止大量徒劳无益的计算，对于与模板图像差异很大的子图像，这样处理的效果很明显。

经此改进后，前面的算法运行时间再次减少 0.3s。至此，图像处理速度约为 1.0s/帧。

3.8　第八级优化：用 Release 版代替 Debug 版

普通的 VC 程序都可以编译成调试版（Debug 版）和发布版（Release 版）。Debug 版由于使用了 Windows 的动态链接库，因此运行时经常需要动态调用计算机硬盘中关于开发平台的 DLL 文件（即 VC 语言的相关安装文件），这会导致一定的耗时。而 Release 版则是将用户程序需要用到的这些动态链接库文件一并编译进了用户的可执行程序，这样不仅可以脱离开发环境而运行，而且可以省掉许多动态调用所致的耗时。对于前面的算法，经编译成 Release 版后，其处理速度提高到 0.4s/帧，相对提速 2.5（即 1.0/0.4）倍。在不采取进一步根本性措施的情况下，这个速度基本是上位机 CPU 运算能力下能达到的最高速度了。

3.9　第九级优化：用 GPU 运算代替 CPU 运算

Nvidia 公司推出的 CUDA 构架的 GPU 显卡，为基于上位机的高速运算提供了一种良好的技术途径[7]，使得不需要借助复杂的 DSP 硬件电路系统就实现可观的算法提速成为可能。CUDA 实现这种提升的主要原因在于它将一个显卡内存划分成很多个 block，每个 block 又被划分成很多个 thread，在遵循访问冲突和总存储容量限制的原则下，这些线程可以并行执行，同时完成各自的任务。由于总 thread 数十分庞大（如 Nvidia 的 G740 显卡，thread 总数为 2147483647*1024），因此只要正确运用，算法的并行程度将相当可观。不过，使用 CUDA 技术实现算法提速时，除了要遵守 CUDA 的一些基本规定和原则外，原来在 CPU 上采用的一些优化策略可能会被放弃，比如递增寻址、序贯最小等，因为这些策略必须串行执行，不适合用 CUDA 实现。

对于前面的匹配跟踪算法，经用 VS 编程，在一块 G740 显卡上基本的 CUDA 技术实现本文“第五级优化”后的 CPU 版本，处理速度提高到 0.18s/帧，相对提速 8（即 1.6/0.18）倍。

3.10　第十级优化：在 CUDA 中用纹理存储器代替全局存储器

CUDA 编程的关键环节有几项：1）正确分配 block 数量和 thread 数量，防止存储器冲突；2）小心规划程序的并行，最大限度地避免线程间的交叉访问；3）灵活使用各种存储器。其中，存储器分为全局存储器、纹理存储器、共享存储器、常数存储器、寄存器等，它们各有特点：全局存储器容量大，但没有缓存机制，访问速度慢；其余存储器访问速度快，但共享存储器、常数存储器和寄存器的数量有限，一般不能用来存储大幅面图像。因此，一个良好的选择是使用纹理存储器来进行图像处理，它既有缓存机制，能快速访问，又有很大容量，最适合存储二维图像[8-9]。选择好存储器后，在线程实现上还要避免线程间的交叉。如果线程间有太多的共享存储器，则不得不采用强制的线程同步函数，这会大大降低线程间的并行性，使表面上看起来并行的多个线程实际上是在串行或交替执行，发挥不并行的优势。

在对上述几个关键环节都作了精心设计后，上节的 CUDA 版程序被提速到 0.015s/帧，相对提速 12（即 0.18/0.015）倍。至此，它基本达到实时性的要求。

3.11　第十一级优化：用模板更新策略提高对目标变化的适应性

到目前为止，前面的所有优化策略都是为了提高算法的速度，而且最终也达到了实时性的期望。现将重点转到提高跟踪正确率上。

第 1 节“算法原型”中已经提到，对于给定的视频图像和目标，跟踪正确率约为 73%。出现误跟的一个主要原因是：随着时间的推移，目标的形态和姿态都会发生一定的变化，而且这种变化随着时间越积越多，以至于一段时间后，目标的实时形态与首帧中指定的模板目标形态已经差异很大，这时如果仍用原来的模板进行目标匹配跟踪，则自然会出现误配。

鉴于此，从第 2 帧起，用每帧中匹配得到的目标子图像来更新之前的目标模板，以用于下一帧的继续匹配。这一策略似乎简便而完美，但实际效果却出乎意料：对于前面所给定的视频图像序列，120 帧的跟踪图像中正确跟踪的只有 74 帧，跟踪正确率约为 62%。

仔细分析，不难发现这种简单的模板更新策略的问题所在：一旦跟踪过程中某一帧出现了误配，后面的帧就都会以这个误配的子图像为模板一直误配下去。

3.12　第十二级优化：用多模板策略提高跟踪正确率

为克服或缓解上节的困境，可以采用很多补充策略，本文

采用一种简单易实现的策略:多模板策略[10-11]。其原理是:除了首帧指定一个目标模板外,将其后目标变化还不大的几帧图像(比如第 2—5 帧)中提取出来的目标也作为模板,这样就形成了多个模板,再往后的各帧图像中匹配目标时,就将各适配的子图像与所有模板都进行累差,总累差最小的子图像才作为对目标的适配,并用此子图像更新模板组中的最后一个模板,然后进入下一帧的再搜索、再匹配。这样做最大的优点是:如果跟踪过程中有一两帧误配,不会导致这种误配的负作用被延续下去,因为它们最多形成一两个错误的模板,在总模板中点的比例并不大,在后续的匹配中,模板组中正确的模板继续起着主导作用来引导算法去搜索正确的适配子图像。这相当于是在上节简单的模板更新策略上增加了一套稳定机制,从而提高了模板的鲁棒性,防止了模板的突变。这与人脑中记忆与遗忘这一对相辅相成的矛盾机制类似:既逐渐遗忘太久远的目标模样,吸纳目标的新模样,又保证不对目标的历史模样完全失忆。

当然,引入多模板机制后,自然也会增加一些时间代价,算法速度会略微变慢,因为要对多个模板进行累差。

采用这种策略后,对于前面给定的视频图像和首帧指定的目标,跟踪率便提高到了 100%,即 120 帧图像都实现了正确跟踪,处理速度为 0.03s/帧,相当于 33Hz,依然满足实时性要求。

至此,对于给定的原型算法和给定的视频图像,优化结束。各级速度优化的策略和效果汇总如表 1 所列。

表 1 速度优化策略汇总表

算法版本	序号	优化策略	每帧处理时间/s	提速能力/s	版本间提速比	总提速时间
V1	—	算法原型	8.0			
V2	1	用 VC 编程	2.8		2.9	2.9
V3	2	合并访问		0.1		
	3	乘法代替加法		0.3		
	4	循环降维		0.1		
	5	宏定义代替系统函数	1.6	0.7	1.8	5.2
V4	6	合并运算		0.3		
	7	使用序贯机制	1.0	0.3	1.6	
V5	8	Release 版本代替 Debug	0.4		2.5	
V6	9	GPU 代替 CPU	0.18		8.8(相对于 V3 版)	44
V7	10	在 CUDA 中用纹理存储器代替全局存储器	0.015		12	530
进一步策略		1. 在 CPU 层面的代码中采用 OpenMP 技术 2. 使用目标位置预测机制(提速 10 倍以上)				

4 还可进一步优化的策略

众所周知,当图像幅面增大或算法复杂度增大时,算法的实时性会逐渐下降。如果这些原因导致前文的算法不再满足实时性要求,则可以进一步考虑更多的速度优化策略。这里列举两个供参考:

(1)在 CPU 层面的代码中采用 OpenMP 技术[12]。OpenMP 是专为多核处理器系统设计的一种并行运行技术,它可以将一段循环均分成几部分,且每部分在一个处理器核上并行执行。这样,对于这段循环,4 核处理器的计算机上就可以提速 4 倍,8 核提高 8 倍,…,依此类推。

(2)使用目标位置预测机制[13]。前文的原型算法中,每帧图像匹配都是在整个图像中搜索对目标模板的最佳匹配,其实,当跟踪趋于稳定的时候,基本可以预测目标在下帧图像中的大致范围,此范围一般只有全图的 1/10 或者更小,仅在此小范围内搜索对目标的最佳匹配,则可以避免做大量的无用功,耗时也就大大减少,速度提升可达 10 倍以上。

需要说明的是:所有提速策略的综合效果并不等于各个策略提速能力的简单乘积,因为不同提速策略作用于算法代码中不同的部分,有些策略之间甚至是互斥的。总的效果是所有代码提速情况的综合表现,必小于各个策略的提速能力之积。

结束语 本文针对基于模板匹配的目标跟踪算法和给定的视频图像,从一个 Matlab 原型算法开始,经过 10 级速度优化和 2 级正确率优化,将图像处理速度从 8s/帧提高到了 0.03s/帧,跟踪正确率从 73%提高到了 100%;详细论述了每一步优化的原理与实现方法,并给出了进一步优化的方向,为基于实验室简单条件的实时图像处理任务提供了一种参考。后续工作的重点将从基于视频图像的开环跟踪转向针对实际场景和伺服机构的闭环跟踪。

参 考 文 献

[1] 焦波. 面向智能视频监控的运动目标检测与跟踪方法研究[D]. 长沙:国防科技大学,2009.

[2] 徐治非. 视频监控中运动目标检测与跟踪方法研究[D]. 上海:上海交通大学,2009.

[3] 傅荟璇. 运动目标识别与光电跟踪定位技术研究[D]. 哈尔滨:哈尔滨工程大学,2009.

[4] 郭伟. 复杂背景下红外目标检测与跟踪[D]. 西安:西安电子科技大学,2008.

[5] 李位星,范瑞霞. 基于 DSP 的运动目标跟踪系统[J]. 自动化技术与应用,2004,23(4):46-49.

[6] 梁路宏,艾海舟,肖习攀,等. 基于模板匹配与支持矢量机的人脸检测[J]. 计算机学报,2002,25(1):22-29.

[7] CUDA Reference Manual(Version 2.0)[M]. Santa Clara, Nvidia Corporation,2008.

[8] CUDA Programming Guid(Version 2.0)[M]. Santa Clara, Nvidia Corporation,2008.

[9] 邓浩,杨菲,潘旭东,等. 利用 CUDA 实现上位机的实时目标跟踪[J]. 信息与电子工程,2010,8(3):368-371.

[10] 邵平,杨路明,黄海滨,等. 基于积分图像的快速模板匹配[J]. 计算机科学,2006,33(12):225-229.

[11] 魏国剑,侯志强,李武,等. 融合光流检测与模板匹配的目标跟踪算法[J]. 计算机应用研究,2014(11):3498-3501.

[12] 武亮,孙秦. 有限元的 OpenMp 并行计算技术[J]. 航空计算技术,2013,43(5):56-60.

[13] ISARD M,BLAEK A. CONDENSATION—Conditional Density Propagation for Visual Tracking[J]. International Journal of Computer Vision,1998,29(1):5-28.