doi:10.11887/j.cn.201903017 http://journal.nudt.edu.cn

雾霭图像实时处理算法加速策略*

崔 文[1],李 强[1],刘晓春[2],李 由[3]
(1. 中国人民解放军96941部队,北京 102208;2. 国防科技大学 空天科学学院,湖南 长沙 410073;
3. 中国航天员科研训练中心人因工程重点实验室,北京 100194)

摘 要:为仿真前视红外导引头实时匹配导引过程,完成目标实时跟踪性判断,针对限制对比度自适应直方图均衡和归一化互相关算法开展加速策略研究,提出基于统一计算设备架构(Compute Unified Device Architecture, CUDA)的雾霭图像实时处理加速方案,并进行设计参数寻优,得出线程块、线程网格设计参数存在最优尺寸,需实验测定的结论。实验结果表明,引入CUDA加速优化后,较未进行CUDA加速前时间指标提高5~20倍,能够满足目标跟踪实时性要求,可为红外导引头目标实时跟踪系统设计提供参考。

关键词:抗雾霭干扰;图像匹配;统一计算设备架构;加速策略
中图分类号:TP391.4 **文献标志码**:A **文章编号**:1001-2486(2019)03-106-06

Real-time image processing acceleration strategy in foggy weather condition

CUI Wen[1], *LI Qiang*[1], *LIU Xiaochun*[2], *LI You*[3]
(1. The PLA Unit 96941, Beijing 102208, China;
2. College of Aeronautics and Astronautics, National University of Defense Technology, Changsha 410073, China;
3. National Key Laboratory of Human Factors Engineering, Astronaut Research and Training Center of China, Beijing 100194, China)

Abstract: In order to simulate forward looking infrared seeker's matching guidance process, an acceleration strategy aiming at the contrast-limit adaptive histogram equalization and the normalization cross correlation algorithms was researched to realize real-time target tracking judgment. The real-time processing acceleration strategy of foggy images based on CUDA(compute unified device architecture) was proposed. The final real-time graph preprocessing acceleration schema implements two main design parameters' optimization, including blocksize and blockgrid. Experimations show that using CUDA acceleration schema, the work efficiency is 5~20 times faster than the original algorithm. The proposed acceleration schema can satisfy the real-time target tracking judgment and can provide references to researchers devoted to the related work.

Keywords: anti-fog interference; image match; compute unified device architecture; acceleration strategy

红外成像末制导[1]是精确制导武器的一种重要制导方式,采用前视红外末制导的导弹在进行攻击检查时可用图像匹配[2-8]实时推演判断识别点可跟踪性,以特定姿态仿真导弹导引至目标点的可行弹道。雾霭气象条件下,为保障命中精度,红外导引头一般需要对实时图进行预处理而后进行匹配运算。为仿真红外导引头实时匹配导引过程,完成目标实时跟踪性判断,本文针对限制对比度自适应直方图均衡(Contrast-Limit Adaptive Histogram Equalization, CLAHE[9])和归一化互相关(Normalization Cross Correlation, NCC)算法开展加速策略研究,提出基于统一计算设备架构(Compute Unified Device Architecture, CUDA)的雾霭图像实时处理加速方案,程序优化后,可实现目标可跟踪性实时判断。

1 软硬件设计考虑

在实时图像处理算法设计领域,学者们研究的热点集中在数字信号处理器(Digital Signal Processor, DSP)、现场可编程门阵列(Field Programmable Gate Array, FPGA)和多核中央处理器(Central Processing Unit, CPU)算法设计与应用。陈正刚等[10]研究了基于DSP与FPGA的视频跟踪系统硬件设计和NCC方法快速实现,将一帧图像的跟踪算法计算时间降至20 ms以内。肖汉等[11]采用图形处理器(Graphics Processing Unit, GPU)并行化方法研究图像匹配问题,获得7倍于CPU实现的图像匹配运算速度,虽然在影

* 收稿日期:2018-03-08
作者简介:崔文(1981—),女,河南驻马店人,工程师,博士研究生,E-mail:13426455682@139.com

像数据较大时,由于未对主机和设备之间的数据传输进行优化,实时度明显降低,但研究给后继者以启迪,推动着 GPU 并行化应用加速发展。宋骥等[12]改进优化了基于 GPU 并行计算的图像快速匹配,改进优化后的算法对图像尺寸有较强适应性。实时图像处理硬件设计方面,曾有研究人员[13-14]指出,对于数据处理能力要求很高的应用领域,GPU 具有 CPU 甚至高端 DSP 不可比拟的性能优势。以上研究表明,选用合适的硬件和软件实现对实时图像处理速度至关重要,是值得设计人员重点考虑的因素。基于逻辑门和触发器等数字电路进行并行任务处理的 FPGA,实时性很高,但造价昂贵、编程速度慢,并不适合做实时图像处理算法验证实验。鉴于上述研究成果和工程经验,本文初步确定采用 GPU + CPU 的硬件组成形式进行实验。

针对实时图像处理并行加速,有文献[15]指出,基于开源计算机视觉库 OpenCV 进行数字图像处理、模式识别、运动跟踪等多核 CPU 算法开发可以提升运算速度;文献[16]指出,采用高效的 GPU 并行计算算法,构建 CPU/GPU 异构计算平台已解决众多模拟计算加速问题;文献[17]指出,在目标跟踪点搜索及跟踪应用领域,基于 CUDA 并行程序设计的 GPU 编程加快了运算速度。也就是说,多核 CPU 并行算法、GPU 并行算法都可用于图像处理加速。更好更优的算法决定了更少的计算时间,是实现实时性的关键,也是本文要着重考虑的问题。

2　实时处理算法

2.1　问题描述

在进行加速策略研究之前,先对加速的对象——雾霭图像实时处理算法进行简单描述。该算法以 CLAHE 算法和模板图像匹配算法为核心要素,计算流程见图 1。基于分块处理的自适应直方图均衡算法可对雾霭天气条件下的图像进行去雾增强,并通过限制对比度阈值去除图像噪声影响,均衡效果较经典直方图优秀且不会过度放大噪声。图像匹配就是以“图”搜图,较为成熟的有基于模板的匹配和基于特征的匹配两种[1]。基于模板的匹配是从实时图中搜索比对出模板图,计算量大[18],对旋转、形变、遮挡比较敏感,但匹配准确度高,需要考虑硬件加速;基于特征的匹配是从实时图中搜索到特征量,计算量相对较小,对灰度、形变及遮挡的适应性较好,匹配精度较高。常用的图像匹配计算公式有绝对平均误差

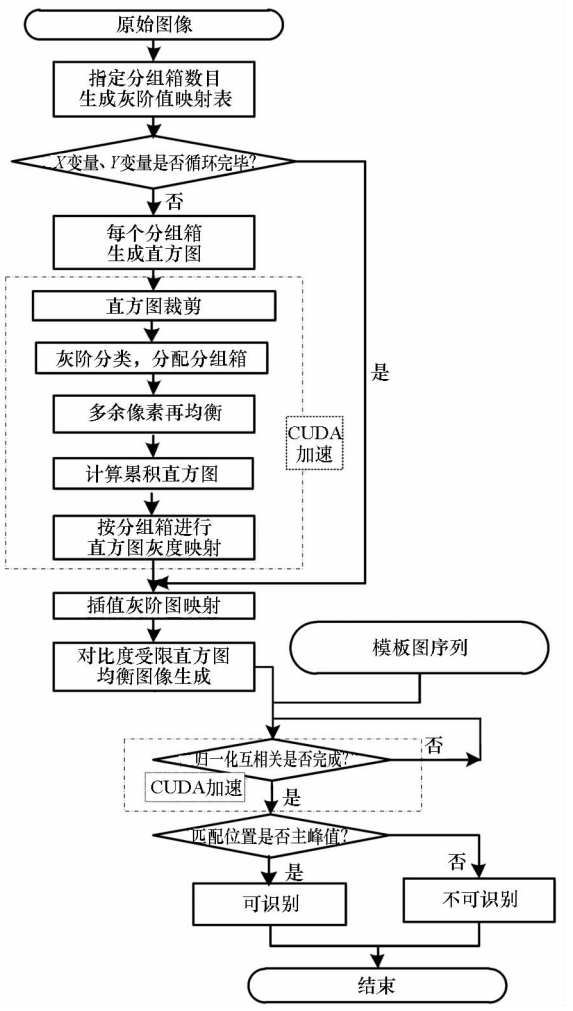

图 1　雾霭图像实时处理算法计算流程图

Fig. 1　Real-time foggy image processing flowchart

(Mean Absolute Deviation, MAD)、绝对差和(Sum of Absolute Differences, SAD)、最小均方误差(Mean Squared Error, MSE)和 NCC(如式(1)所示)。对于本文研究的目标实时跟踪性判断来说,需要考虑相互匹配的图像间亮度变化可能,NCC 能更好消除这种影响,准确度更高。

$$R(x,y)=\frac{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}S_{x,y}(i,j)g(i,j)-\overline{S}_{x,y}\overline{g}}{\sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}S_{x,y}^{2}-\overline{g}^{2}}\sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}S_{x,y}^{2}-\overline{S}_{x,y}^{2}}} \tag{1}$$

式中:S 表示实时图;g 表示模板图;$\overline{g}$ 表示模板图中所有像素的灰度均值;$\overline{S}_{x,y}$表示实时图中以当前搜索位置为左上角点、模板图尺寸大小范围内图像灰度均值;$R(x,y)$代表实时图和模板图在某一位置匹配的相关系数;m、n 为模板图长宽,x、y 为搜索位置,$x\in[0,M-m+1]$,$y\in[0,N-n+1]$,M、N 为实时图长宽。$R(x_{\text{match}},y_{\text{match}})$达到最大值时,位置$(x_{\text{match}},y_{\text{match}})$为搜索到的匹配点。

要解决的问题就集中到如何在毫秒间完成图像匹配,即算法实时化加速问题。

2.2 加速方案一:OpenMP 多核 CPU 并行

OpenMP 多核 CPU 并行加速方案是采用多线程技术进行并行计算加速。特点是 CPU 所有的核共享一个内存,不同核执行不同线程,在同一内存的不同部分操作多数据,加速的目的是将各线程工作负载均衡分配,CPU 整体计算速度达到最高。常见的 For 循环语句采用图 2 方式实现代码并行化;不同代码段并行执行时,采用 OpenMP Section 语句(见图 3)实现。

```
#pragma omp parallel for schedule(dynamic)
    for(i=0;i<n;i++)
    {
    可并行部分代码;
    }
```

图 2 OpenMP 并行化 For 循环实现方式

Fig. 2 For loop's OpenMP parallelization

```
#pragma omp parallel sections
{
    # pragma omp section
    {
        可并行代码
    }
    # pragma omp section
    {
        可并行代码
    }
}
```

图 3 不同段并行执行的 OpenMP Section 语句

Fig. 3 Different section parallelization in OpenMP

2.3 加速方案二:高效 GPU 并行算法

GPU 并行加速方案是通过使用 NVIDIA 公司的 CUDA 编程模型实现"block 间粗粒度并行"和"thread 间细粒度并行"双层并行加速。

加速方案用 CUDA 实现时,由 CUDA 编程指定 CPU 指令和内存分配,GPU 完成具体并行计算。具体说就是,用 CPU 可编译执行的代码实现并行数据加载内存,数据映射到 GPU,然后 GPU 开展并行计算、共享存储器块内数据,定制运行的众线程数量,完成核函数(用_global_前缀标明)并行计算,完成后再把控制权交回 CPU,数据映射回 CPU,最终进行数据存取和文件读写,由 GPU 可编译执行的代码和 CPU 可编译执行代码共同完成全算法加速。加速算法实现时,要注意划分的线程块内 warp 数目合理,每个流处理器都可交替执行内部 warp,提高整体计算效率;也要根据 NVIDIA GPU 设备的流处理器个数,并行使用合理的线程块 block 数,提升流多处理器(Streaming Multiprocessor, SM)利用率。

2.4 CUDA 实现并行加速策略

GPU 算法开销常见的有调度开销、GPU 内核函数启动开销和内核函数之间数据传输开销。算法优化主要考虑循环融合和内核融合。循环融合是尽量将明显独立的循环进行归并,减少总体迭代的次数;内核融合是将数据重用、内存合并、线程间数据共享。雾霭图像实时处理算法加速策略中的归一化互相关部分,考虑申请共享内存进行 CUDA 并行实现,使得每个线程块 block、每个线程 thread 都能访问和操作这块共享内存。由于共享内存是基于存储体切换架构的,无论多少个线程发起操作,每个存储体每个周期只执行一次操作,即如果有线程束中的一个线程访问片上共享内存,所有的线程都能在这同一指令周期内同时执行访问共享内存,无须顺序访问,因而能有效减少图像匹配算法像元读取访问次数,降低数据访问开销。

本文并行加速优化的一个重要设计参数是线程块大小。因优化原理主要是考虑最大化的数据重用进行内核融合,假设模板图按 $BLOCKSIZE \times BLOCKSIZE$ 大小进行分块,实时图按 $2BLOCKSIZE \times 2BLOCKSIZE$ 大小分块(见图 4),利用线程索引、线程块索引后,可定位图像每个像素点数据在实时图中的唯一位置为[(线程块 Y 方向大小×线程块 Y 方向索引+块内线程索引)×实时图宽度+线程块 X 方向大小×线程块 X 方向索引+块内线程索引],标识为位置 ID。而线程块索引又由线程网格 grid 的维数、各维度位置索引确定,这样每个像素点数据与线程间建立对应关系,又能保证同一个线程块不必太大,避免每个线程运算前都将模板图全部像素重新读入到 GPU 设备进行作业。并行加速策略优化主要针对图像匹配归一化相关计算中计算量大的 $\sum_{i=1}^{m}\sum_{j=1}^{n} S_{x,y}(i,j)g(i,j)$ 进行,使用 GPU 内核函数在同一指令周期内同时执行 $S_{x,y}(i,j)g(i,j)$,先整幅图像数据同步,后块内数据同步,快速完成累加运算。除此之外,算法还综合运用了循环融合,将 $\sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} S_{x,y}^2 - \bar{g}^2}$ 、$\sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} S_{x,y}^2 - \bar{S}_{x,y}^2}$ 的计算采用循环融合统一考虑,尽可能降低总迭

代次数,进一步减低算法开销。

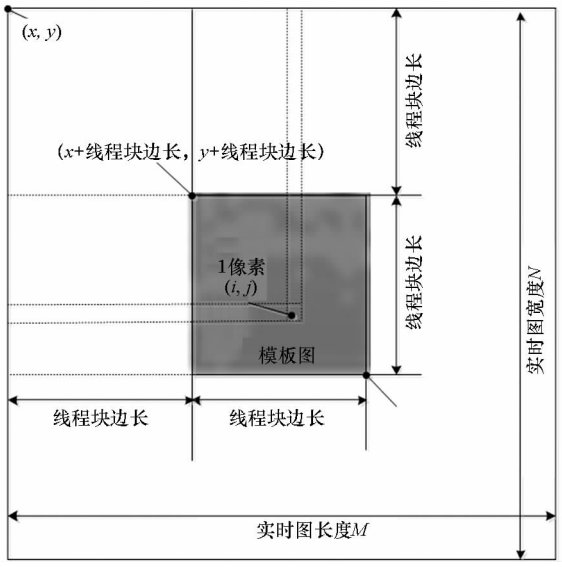

图 4　实时图内 block 划分

Fig. 4　Block division in real-time figure

需要说明的是,匹配循环次数也对算法效率产生影响。由于匹配循环次数等于[(实时图宽度 - 模板图宽度)/ 线程块 *Y* 方向大小,(实时图长度 - 模板图长度)/ 线程块 *X* 方向大小],匹配循环次数越少, CUDA 内核函数 1 次可使用核心数越多,算法开销也越小。当内核函数计算需被拆分成多次,无法一次完成时,加速效果将大大降低。以上性能加速均基于线程块大小的平方与匹配循环次数的乘积小于 GPU 设备标称核心数目。实验使用的 GPU 单个线程块 block 管理的最大线程数量为 1024, *BLOCKSIZE* 必须小于等于 32。固定模板图为 64 像素 ×64 像素,模板图线程块大小 *BLOCKSIZE* 可从典型值 8、16、32 中取一种;固定模板图为 32 像素 ×32 像素,模板图线程块大小 *BLOCKSIZE* 可从典型值 8、16 中取一种。又鉴于实验所用 GPU 设备标称核心数目,线程块大小、匹配循环次数最好按表 1 所列配对方式选取,具体的实验结果参见下一部分内容。

表 1　主要设计参数配对表

Tab. 1　Main design parameters pairing table

线程块大小/像素	匹配循环次数
8 ×8	2 ×2 或 4 ×4
16 ×16	2 ×2 或 4 ×4
32 ×32	2 ×2 或 4 ×4

CLAHE 部分的 CUDA 优化加速是分别对直方图生成、直方图裁剪、直方图再均衡环节进行,将图像各个分组箱分块内的像元灰阶数、分组箱分块内超出限定灰度阈值范围的灰阶数目使用共享存储器实现数据共享,实现图像像素点 For 循环累加计数运算 CUDA 并行化,达到降低算法开销、优化加速的目的。

3　结果与分析

针对上述两种加速方案开展对比实验。针对图 5(a)所示的雾霭图像进行去雾处理,处理后的图像增强效果见图 5(b)。近处楼房、远处楼房、天空均得到对比度增强,去雾效果显著。进行方案一加速和方案二加速后,图像匹配归一化算法部分 GPU 加速后的性能指标、OpenMP 多核 CPU 加速后的性能指标分别见表 2、表 3;CLAHE 算法

(a) 原始雾霭图像

(a) Origin foggy image

(b) 去雾霭效果图

(b) Foggy image processing effect show

图 5　雾霭图像处理效果

Fig. 5　Foggy image processing effect

部分 GPU 加速后性能指标、OpenMP 多核 CPU 加速后性能指标分别见表4、表5。

表2 NCC 算法 GPU 加速后性能指标

Tab.2 Performance indicators of NCC algorithm with GPU acceleration

图像大小（实时图与模板图）/像素	线程块大小/像素	循环次数	未使用加速算法实现/ms	CUDA 并行算法实现/ms	加速比
64×64 与 32×32	8×8	4×4	114.82	5.57	20.61
64×64 与 32×32	16×16	2×2	114.82	6.95	16.52
64×64 与 32×32	32×32	1×1	114.82	16.08	7.14
64×64 与 32×32	13×13	3×3	114.82	83.81	1.37
128×128 与 64×64	8×8	8×8	517.30	55.42	9.33
128×128 与 64×64	16×16	4×4	517.30	70.93	7.29
128×128 与 64×64	32×32	2×2	517.30	88.52	5.84
128×128 与 64×64	13×13	5×5	517.30	278.56	1.86

表3 NCC 算法 OpenMP 多核 CPU 加速后性能指标

Tab.3 Performance indicators of NCC algorithm with multicore CPU OpenMP acceleration

图像大小（实时图与模板图）/像素	未使用加速算法实现/ms	OpenMP 多核 CPU 并行/ms	加速比
64×64 与 32×32	114.82	84.43	1.36
128×128 与 64×64	517.30	404.14	1.28

表4 CLAHE 算法 GPU 加速后性能指标

Tab.4 Performance indicators of CLAHE with GPU acceleration

图像大小/像素	线程块大小/像素	循环次数	未使用加速算法实现/ms	CUDA 并行算法实现/ms	加速比
64×64	16×16	4×4	29.30	4.11	7.13
64×64	8×8	8×8	29.30	3.81	7.69
64×64	32×32	2×2	29.30	3.68	7.96
64×64	64×64	1×1	29.30	2.98	9.83
128×128	8×8	16×16	31.78	4.00	7.95
128×128	16×16	8×8	31.78	3.95	8.05
128×128	32×32	4×4	31.78	4.55	6.98
128×128	64×64	2×2	31.78	4.06	7.83

表5 CLAHE 算法 OpenMP 多核 CPU 加速后性能指标

Tab.5 Performance indicators of CLAHE with multicore CPU OpenMP acceleration

图像大小/像素	未使用加速算法实现/ms	OpenMP 多核 CPU 并行/ms	CPU 核数	加速比
64×64	29.30	16.74	2	1.75

查看表2中加速比为1.37和1.86的两个数据发现，选用的线程块尺寸大小不合适（图像大小不能整除线程块尺寸大小）会造成加速比过低，导致效果接近表3中 OpenMP 多核 CPU 加速效果。为提高加速比，在 GPU CUDA 加速实现策略中避免选择此种线程块尺寸。剔除两数据后的表2，图像匹配 GPU 优化加速后较未使用加速策略前计算时间缩短至 4.85% ~ 17.11%，计算性能提高 5 ~ 20 倍，加速效果理想。对比表4、表5可以发现，CLAHE 算法 GPU 优化加速后时间可缩短至 10.17% ~ 14.32%，计算性能提高 6 ~ 9 倍。分析表2中图像大小与加速比关系可发现，使用 CUDA 并行实现 NCC 加速时，选定线程块大小后，存在实时图与模板图最优尺寸配对，使加速比达到最大。

透过数据本身，进一步分析可得到：线程块大小设计对加速性能有很大影响，这进一步验证了优化设计的原理；当图像大小是线程块尺寸大小的整数倍时，相比随意的线程块大小尺寸，计算时间更短，这是因为减少了总迭代次数，降低了算法开销；同时线程块越小，GPU 使用的核心数越多，加速性能越明显。对于图像匹配 NCC 算法，实时图为 64 像素 ×64 像素、模板图为 32 像素 ×32 像素、线程块大小为 8 像素 ×8 像素时，CUDA 使用核心数为 4 ×4 ×8 = 128 < 384（GPU 核心数），并且中间成果数据融合效率最高，因而加速性能最高。当实时图为 128 像素 ×128 像素、模板图为 64 像素 ×64 像素、线程块大小为 8 像素 ×8 像素时，CUDA 使用核心数为 8 ×8 ×8 = 512 > 384（GPU 核心数），内核函数计算被拆分成多次计算，加速性能有所减弱，但因分块形式对于数据融合较为高效，也能最终获得 9.33 倍的加速效果。

4 结论

1）充分利用共享寄存器的 CUDA 加速算法设计，可以降低 GPU 设备读取数据次数，实现并行线程间数据共享，实现算法加速。单幅图像 CUDA 加速算法线程块尺寸设计对加速性能有较大影响，当模板图图像大小是线程块尺寸大小整数倍时，相比随意的线程块大小尺寸，计算时间更短；对于性能指标固定的 GPU，选定线程块大小后，能实时处理的实时图和模板图存在最优尺寸大小，可通过仿真实验方法进行测试标定。

2）从总体时间性能指标方面来看，基于 CUDA 的雾霭图像 GPU 并行处理算法计算速度较未进行 CUDA 加速优化前提高 5 ~ 20 倍，且

GPU 使用的核心数越多,加速性能越明显;64 像素 ×64 像素实时图与 32 像素 ×32 像素模板图匹配的计算时间仅为 5.57 ms,达到实时仿真计算的要求,能够用于目标实时跟踪性判断检测。

3)在解决单幅图像并行处理需求方面,使用 OpenMP 实现多核 CPU 并行加速效果不如使用 GPU CUDA 加速明显,OpenMP 算法开支时间较大,实时性较差。相比之下,OpenMP 多核 CPU 并行更适合多幅图像同时进行处理,减少总体作业开支时间。

参考文献(References)

[1] 李言俊, 张科. 景象匹配与目标识别技术[M]. 西安: 西北工业大学出版社, 2009: 46 – 47.
LI Yanjun, ZHANG Ke. Scene matching and target recognition technology[M]. Xi′an: Western Polytechnical University Press, 2009: 46 – 47. (in Chinese)

[2] 王永仲. 现代军用光学技术[M]. 北京: 科学出版社, 2004: 189 – 202.
WANG Yongzhong. Modern military optics technology[M]. Beijing: Science Press, 2004: 189 – 202. (in Chinese)

[3] 郑南宁. 计算机视觉与模式识别[M], 北京: 国防工业出版社, 1998: 99 – 103.
ZHENG Nanning. Computer vision and pattern recognition[M]. Beijing: National Defense Industry Press, 1998: 99 – 103. (in Chinese)

[4] 章毓晋. 图像分割[M]. 北京: 科学出版社, 2001: 135 – 136.
ZHANG Yujin. Image segmentation[M]. Beijing: Science Press, 2001: 135 – 136. (in Chinese)

[5] 贾云得. 机器视觉[M]. 北京: 科学出版社, 2000: 55 – 57.
JIA Yunde. Machine vision[M]. Beijing: Science Press, 2000: 55 – 57. (in Chinese)

[6] 卢福刚. 红外图像目标识别与跟踪方法研究[D]. 西安: 西北工业大学, 1998.
LU Fugang. Infrared image target recognition and tracking method research[D]. Xi′an: Western Polytechnical University Press, 1998. (in Chinese)

[7] 杨亚军. 红外图像处理与目标跟踪研究[D]. 西安: 西北工业大学, 1998.
YANG Yajun. Infrared image process and target tracking method research[D]. Xi′an: Western Polytechnical University Press, 1998. (in Chinese)

[8] Chakraborty A P, de Sabyasachi T S, Mainak C A. Grey-scale template image matching by area based matching techniques[C]//Proceedings of Kolkata: IEMCON, 2011.

[9] 张璞, 王英, 王苏苏. 基于 CLAHE 变换的低对比度图像增强改进算法[J]. 青岛大学学报(工程技术版), 2011, 26(4): 57 – 60.
ZHANG Pu, WANG Ying, WANG Susu. Low contrast image enhancement algorithms based on the improved CLAHE transform[J]. Journal of Qingdao University(Engineering & Technology Edition), 2011, 26(4): 57 – 60. (in Chinese)

[10] 陈正纲, 袁晓辉. 基于 DSP 与 FPGA 的视频跟踪系统设计[C]//江苏省自动化学会学术年会论文集, 2008: 182 – 186.
CHEN Zhenggang, YUAN Xiaohui. Video tracking system design based on DSP and FPGA[C]//Jiangsu Province Automation Academic Essays, 2008: 182 – 186. (in Chinese)

[11] 肖汉, 张祖勋. 基于 GPGPU 的并行影像匹配算法[J]. 测绘学报, 2010, 39(1): 46 – 51.
XIAO Han, ZHANG Zuxun. Parallel image matching algorithm based on GPGPU[J]. Acta Geodaetica et Cartographica Sinica, 2010, 39(1): 46 – 51. (in Chinese)

[12] 宋骥, 周松涛. 基于 GPU 并行计算的图像快速匹配[J]. 湖北民族学院学报(自然科学版), 2011, 29(3): 306 – 310.
SONG Ji, ZHOU Songtao. Fast image matching based on GPU parallel computing[J]. Joural of Hubei Institute for Nationalities(Natural Sciences), 2011, 29(3): 306 – 310. (in Chinese)

[13] 王俊, 张玉玺, 杨彬. DSP/FPGA 嵌入式实时处理技术及应用[M]. 北京: 电子工业出版社, 2015: 1 – 11.
WANG Jun, ZHANG Yuxi, YANG Bin. Embedded real-time processing technology and application based on DSP and FPGA[M]. Beijing: Electronic Industry Press, 2015: 1 – 11. (in Chinese)

[14] 刘枫, 李桦, 田正雨, 等. 基于 MPI + CUDA 的异构并行可压缩流求解器[J]. 国防科技大学学报, 2014, 36(1): 6 – 10.
LIU Feng, LI Hua, TIAN Zhengyu, et al. Heterogeneous parallel compressible flow solver based on MPI + CUDA[J]. Journal of National University of Defense Technology, 2014, 36(1): 6 – 10. (in Chinese)

[15] Bradski G, Kaehler A. 学习 OpenCV[M]. 于仕琪, 刘瑞祯, 译. 北京: 清华大学出版社, 2009: 25 – 28.
Bradski G, Kaehler A. Learning OpenCV[M]. Translated by YU Shiqi, LIU Ruizhen. Beijing: Tsinghua University Press, 2009: 25 – 28. (in Chinese)

[16] 卢风顺, 宋群强, 银福康, 等. CPU/GPU 协同并行计算研究综述[J]. 计算机科学, 2011, 38(3): 5 – 9.
LU Fengshun, SONG Qunqiang, YIN Fukang, et al. Survey of CPU/GPU synergetic parallel computing[J]. Computer Science, 2011, 38(3): 5 – 9. (in Chinese)

[17] Lin L, Bar-Shalom Y, Kirubarajan T. Tracking labeling and PHD filter for multitarget tracking[J]. IEEE Transactions on Aerospace and Electronic Systems, 2006, 42(3): 778 – 795.

[18] 李卓, 邱慧娟. 基于相关系数的快速图像匹配研究[J]. 北京理工大学学报, 2007, 27(11): 998 – 1000.
LI Zhuo, QIU Huijuan. Fast image matching based on correlation coefficient[J]. Transactions of Beijing Institute of Technology, 2007, 27(11): 998 – 1000. (in Chinese)