Opencv——目标跟踪Tracker

OpenCV Tracker位于**OpenCV contrib modules**模块中，需要另行下载编译安装，**contrib版本一定要与opencv版本相匹配**，windows下的安装方法见OpenCV + OpenCV Contrib安装教程（windows）。库名称为**tracking.hpp**。

参考资料：https://blog.csdn.net/shujian_tianya/article/details/84558033

一、Introduction to OpenCV Tracker

首先介绍一下分类器。分类器的工作是将图像的矩形区域分为对象和背景。分类器将图像补丁作为输入，并返回0,1之间的数值，用来作为图像补丁包含对象的概率。当确定图像补丁是背景时，数值为0，补丁是对象时，分数为1。
在机器学习中，我们将“在线”用来指代在运行中训练的算法。离线分类器可能需要多个实例来训练分类器，但是在运行时通常使用非常少的实例训练在线分类器。
分类器通过馈送正（对象）和负（背景）示例来训练。如果你想建立一个分类器来检测猫，你训练它与成千上万包含猫的图像和数千个不包含猫的图像。这样分类器学习区分什么是猫和什么不是猫。你可以在这里了解有关图像分类的更多信息。在建立在线分类器时，我们没有成千上万的正负的例子。
让我们来看看不同的跟踪算法如何处理这个**在线训练**的问题。

BOOSTING Tracker
这个跟踪器是基于在线版本的AdaBoost——基于HAAR级联的人脸检测器内部使用的算法。该分类器需要在运行时使用对象的正反样本进行训练。以用户提供的初始bounding box (或其他对象检测算法提供的)作为对象的正样本，bounding box外的许多图像补丁作为背景。给定一个新的帧，在前一个位置邻域的每个像素上运行分类器，记录分类器的score。对象的新位置是score最大的位置。现在对于分类器我们多了一个正样本。随着更多的帧进入，分类器将使用这些额外的数据进行更新。
优点:没有。这个算法已经有10年的历史了，工作正常，但是我找不到一个很好的理由去使用它，尤其是当其他基于类似原理的高级跟踪器(MIL, KCF)可用时。
缺点:跟踪性能一般。它不能可靠地知道跟踪何时失败。

MIL Tracker
这个跟踪器在思想上与上面描述的BOOSTING跟踪器相似。最大的区别是，它不会只考虑对象的当前位置作为一个正样本，而是在当前位置附近的一个小邻域中寻找，以生成几个潜在的正样本。你可能会认为这是一个坏主意，因为在大多数“正样本”的例子中，对象不是居中的。
这就是多实例学习(MIL)的用得上的地方。在MIL中，你没有指定正样本和负样本，而是指定正面和负面的“包”。在正样本包中的图像集合并不都是正样本。相反，只有一个图像在positive bag需要是一个正样本!在我们的例子中，一个positive bag包含以对象当前位置为中心的patches，以及它周围一个小邻域的patches。即使被跟踪对象的当前位置不准确，当来自当前位置附近的样本被放入positive bag时，这个包很可能包含至少一个对象居中的图像。MIL project page为那些希望深入了解MIL跟踪器内部工作原理的人提供了更多信息。
优点：性能相当好。它不像BOOSTING漂移那么大，而且在部分遮挡的情况下，它能做合理的工作。如果您正在使用OpenCV 3.0，这可能是您可以使用的最好的跟踪器。但是如果您使用更高的版本，请考虑KCF。
缺点:追踪失败并不可靠。无法从完全遮挡中恢复。

KCF Tracker
KCF是kernefied Correlation Filters的缩写。此跟踪器基于前两个跟踪器中提出的思想。该跟踪器利用了MIL跟踪器中使用的多个正样本具有较大重叠区域这一事实。这些重叠的数据导致了一些很好的数学特性，这些特性被这个跟踪器利用，使跟踪速度更快，同时也更准确。
优点:准确性和速度都比MIL好，而且它比提振和MIL更有效地追踪失败，如果你使用的是OpenCV 3.1和

以上，我建议用这个来做大部分的应用。
缺点:不能从完全遮挡中恢复。OpenCV 3.0中没有实现。

TLD Tracker
TLD代表跟踪、学习和检测。顾名思义,这个追踪分解长期跟踪任务分为三部分——(短期)跟踪、学习、和检测。从作者的论文，“跟踪器跟踪对象从一帧到另一帧。检测器定位到目前为止观察到的所有外观，并在必要时纠正跟踪器。学习估计检测器的错误，并对其进行更新，以避免以后出现这些错误。“这个跟踪器的输出往往有点跳跃。例如，如果您正在跟踪一个行人，并且场景中有其他行人，那么这个跟踪器有时可以临时跟踪与您打算跟踪的行人不同的行人。在积极的一面，这个轨迹似乎在更大的范围内跟踪一个物体，运动和遮挡。如果你有一个视频序列，其中对象隐藏在另一个对象后面，这个跟踪器可能是一个不错的选择。
优点:在多帧的遮挡下效果最好。此外，跟踪规模变化最好。
缺点:大量的假阳性使它几乎无法使用。

MEDIANFLOW Tracker
在内部，该跟踪器在时间上同时在向前和向后两个方向跟踪对象，并测量这两个轨迹之间的差异。最小化这种向前向后的误差可以使他们可靠地检测跟踪失败，并在视频序列中选择可靠的轨迹。在我的测试中，我发现当运动是可预测的并且很小的时候，这种跟踪器工作得最好。与其他跟踪器不同的是，即使在跟踪时，跟踪器也会一直运行。
优点:出色的报告跟踪失败。当运动是可预测的并且没有遮挡时，这种方法非常有效。
缺点:在大动作下失败。

GOTURN tracker
在跟踪器类中的所有跟踪算法中，这是唯一一个基于卷积神经网络(CNN)的算法。从OpenCV文档中，我们知道它是“对视点变化、光照变化和变形的健壮性”。但它不能很好地处理遮挡。GOTURN是一个基于CNN的跟踪器，使用caffe模型进行跟踪。Caffe模型和原型文本文件必须出现在代码所在的目录中。这些文件也可以从opencv_extra存储库下载，在使用前连接并提取。GOTURN目标跟踪算法已经移植到OpenCV。

MOSSE tracker
最小平方误差输出和(MOSSE)使用自适应相关的目标跟踪，产生稳定的相关滤波器时，初始化使用一帧。MOSSE跟踪器对于光线、比例、姿态和非刚性变形的变化是鲁棒的。它还根据峰值旁瓣比检测遮挡，这使得跟踪器能够在对象重新出现时暂停并恢复到它停止的位置。MOSSE跟踪器也运行在更高的fps (450 fps甚至更多)。此外，它也非常容易实现，与其他复杂的跟踪器一样准确，而且速度快得多。但是，在性能方面，它落后于基于深度学习的跟踪器。

CSRT tracker
在带通道和空间可靠性的判别相关滤波器(DCF-CSR)中，我们使用空间可靠性图将滤波器支持调整到从帧中选择区域的部分进行跟踪。这确保了选定区域的放大和定位，并改进了对非矩形区域或对象的跟踪。它只使用两个标准特性(生猪和颜色名称)。它的fps也相对较低(25 fps)，但对目标跟踪具有较高的精度。

测试代码:

```
#include <opencv2/core/utility.hpp>
#include <opencv2/tracking.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>
#include <cstring>
using namespace std;
using namespace cv;
int main(int argc, char** argv) {
    // show help
    if (argc < 2) {
        cout <<
            " Usage: tracker <video_name>\n"
```

```
            " examples:\n"
            " example_tracking_kcf Bolt/img/%04d.jpg\n"
            " example_tracking_kcf faceocc2.webm\n"
            << endl;
        return 0;
    }
    // declares all required variables
    Rect2d roi;
    Mat frame;
    // create a tracker object
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerBoosting::create();
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerMIL::create();
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerKCF::create();
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerGOTURN::create();
    cv::Ptr<cv::Tracker> tracker = cv::TrackerCSRT::create();
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerMedianFlow::create();
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerMOSSE::create();
    //cv::Ptr<cv::Tracker> tracker = cv::TrackerTLD::create();
    // set input video
    std::string video = argv[1];
    VideoCapture cap(video);
    // get bounding box
    cap >> frame;
    roi = selectROI("tracker", frame);
    //quit if ROI was not selected
    if (roi.width == 0 || roi.height == 0)
        return 0;
    // initialize the tracker
    tracker->init(frame, roi);
    // perform the tracking process
    printf("Start the tracking process, press ESC to quit.\n");
    for (;; ) {
        // get frame from the video
        cap >> frame;
        // stop the program if no more images
        if (frame.rows == 0 || frame.cols == 0)
            break;
        // update the tracking result
        tracker->update(frame, roi);
        // draw the tracked object
        rectangle(frame, roi, Scalar(255, 0, 0), 2, 1);
        // show image with the tracked object
        imshow("tracker", frame);
        //quit on ESC button
        if (waitKey(1) == 27)break;
    }
    return 0;
}
```

经过本人的一番测试，发现与其他几种相比较CSRT Tracker这个跟踪效果最好，即使在遮挡的情况下，但是有一缺点就是处理速度慢，慢到你能很明显看出来是一帧一帧场景的播放。其他几个速度快的精度差，精度好的速度慢。

二、Using MultiTracker

目标：创建一个MultiTracker对象
使用MultiTracker对象同时跟踪几个目标

```
/*----------------------------------------------
 * Usage:
 * example_tracking_multitracker <video_name> [algorithm]
 *
 * example:
 * example_tracking_multitracker Bolt/img/%04d.jpg
 * example_tracking_multitracker faceocc2.webm KCF
 *--------------------------------------------------*/

#include <opencv2/core/utility.hpp>
#include <opencv2/tracking.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>
#include <cstring>
#include <ctime>
#include "samples_utility.hpp"

using namespace std;
using namespace cv;

int main( int argc, char** argv ){
  // show help
  if(argc<2){
    cout<<
      " Usage: example_tracking_multitracker <video_name> [algorithm]\n"
      " examples:\n"
      " example_tracking_multitracker Bolt/img/%04d.jpg\n"
      " example_tracking_multitracker faceocc2.webm MEDIANFLOW\n"
      << endl;
    return 0;
  }

  // set the default tracking algorithm
  std::string trackingAlg = "KCF";

  // set the tracking algorithm from parameter
  if(argc>2)
    trackingAlg = argv[2];

  // create the tracker
  MultiTracker trackers;

  // container of the tracked objects
  vector<Rect2d> objects;

  // set input video
  std::string video = argv[1];
  VideoCapture cap(video);

  Mat frame;

  // get bounding box
  cap >> frame;
  vector<Rect> ROIs;
  selectROIs("tracker",frame,ROIs);
```

```
  //quit when the tracked object(s) is not provided
  if(ROIs.size()<1)
    return 0;

  // initialize the tracker
  std::vector<Ptr<Tracker> > algorithms;
  for (size_t i = 0; i < ROIs.size(); i++)
  {
      algorithms.push_back(createTrackerByName(trackingAlg));
      objects.push_back(ROIs[i]);
    }

  trackers.add(algorithms,frame,objects);

  // do the tracking
  printf("Start the tracking process, press ESC to quit.\n");
  for ( ;; ){
    // get frame from the video
    cap >> frame;

    // stop the program if no more images
    if(frame.rows==0 || frame.cols==0)
      break;

    //update the tracking result
    trackers.update(frame);

    // draw the tracked object
    for(unsigned i=0;i<trackers.getObjects().size();i++)
      rectangle( frame, trackers.getObjects()[i], Scalar( 255, 0, 0 ), 2, 1 );

    // show image with the tracked object
    imshow("tracker",frame);

    //quit on ESC button
    if(waitKey(1)==27)break;
  }

}
```