天津大学

计算机视觉期中 Quiz

Meanshift 与 Camshift 目标跟踪

学　　院 智能与计算学部

专　　业 软件工程

学　　号 3020001267

姓　　名 王旭

目录

1 程序

1.1 Meanshift opencv 调库实现

A．程序片段

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include<iostream>
#include<cstdio>
using namespace std;
using namespace cv;
const int seed = 42;




int main()
{
    //打开视频文件
    VideoCapture cap;
    cap.open("D:\\study\\CV\\assignment3\\in5.mp4");

    if (!cap.isOpened())
    {
        cerr << "Failed to open video file." << endl;
        return -1;
    }

    Mat  frame;
    //输出结果
    int fps = cap.get(CAP_PROP_FPS);
    Size S = Size((int)cap.get(CAP_PROP_FRAME_WIDTH),
        (int)cap.get(CAP_PROP_FRAME_HEIGHT));
    VideoWriter writer("D:\\study\\CV\\assignment3\\out5.mp4", CAP_OPENCV_MJPEG, fps,
S, true);

    Mat roi;
    //读入一帧来选择要追踪的物体
    Rect rect;
```

```
cap >> frame;
rect = selectROI("选择ROI", frame);
roi = frame(rect);

//转换到hsv空间
Mat hsv_roi, mask;
cvtColor(roi, hsv_roi, COLOR_BGR2HSV);
//创建掩码,这个操作没意义只是为了满足calcHist的参数要求
inRange(hsv_roi, Scalar(0, 0, 0), Scalar(255, 255, 255), mask);


//计算直方图
float range_[] = { 0, 180 };
const float* range[] = { range_ };
Mat roi_hist;
int histSize[] = { 180 };
int channels[] = { 0,1,2 };
calcHist(&hsv_roi, 1, channels, mask, roi_hist, 1, histSize, range);

//归一化
normalize(roi_hist, roi_hist, 0, 255, NORM_MINMAX);


// 设置Mean Shift算法的终止条件
TermCriteria term_crit(TermCriteria::EPS | TermCriteria::COUNT, 10, 1);


while (true)
{
    cap >> frame;
    if (frame.empty())
    {
        break;
    }

    Mat hsv, dst;
    cvtColor(frame, hsv, COLOR_BGR2HSV);

    // 计算每一帧图像中的直方图的反投影
    calcBackProject(&hsv, 1, channels, roi_hist, dst, range);

    // 应用Mean Shift算法
    meanShift(dst, rect, term_crit);
```

```
        // 绘制矩形框标记目标物体
        rectangle(frame, rect, Scalar(0, 255, 0), 2);


        imshow("Video", frame);
        writer.write(frame);


        if (waitKey(20) == 27)
        {
            break;
        }
    }

    // 释放资源
    cap.release();
    writer.release();
    destroyAllWindows();
    return 0;


}
```

B．程序说明及原理解释

Meanshift 算法本质是就是一个均值偏移，随着迭代的进行，每次往当前窗口的质心移动，最终达到一个局部的最优解。当引入了一个核函数之后，上述过程是一个类似于反向传播梯度下降的过程，在 opencv 的 meanshift 实现中就是对一个核函数处理的 meanshift 向量进行一个求导，每次往极值点方向移动。

具体到目标跟踪，首先手动选取了一个 ROI 区域，程序首先对这个图像中的 ROI 区域转换到 HSV 空间,然后得到这个 ROI 区域的 H 通道的分布直方图，这个直方图代表了不同 H 分量出现的频率，对直方图做归一化后直方图代表了 ROI 区域不同 H 分量出现的概率。之所以选用 HSV 色彩空间的 H 分量来计算直方图是因为 HSV 色彩空间的 H 分量代表颜色，缓解了亮度对目标跟踪的影响。

对于视频中的每一帧图像，将当前帧图像 a 同样转换到 HSV 色彩空间。利

用其 H 分量，与 roi 区域的直方图做直方图反向投影。直方图方向投影会输出与输入图像（待搜索）同样大小的图像，只是这个图像的每一个像素值代表了输入图像上对应点属于给定直方图的概率。

直方图反向投影步骤如下，给定图像 frame，与直方图 roi_hist:

1. 从 frame 左上角开始，遍历全图所有像素
2. 以 frame 位于(x,y)处在 H 分量上的像素作为索引，取直方图中对应 H 值的概率，这个概率即为反投影结果位于(x,y)的值

在进行了直方图反向投影后，得到了图像各处属于 roi 的概率，然后在此概率空间上，使用 meanshift 以上一次收敛的结果作为初值，收敛到最相似区域。一次收敛完成即结束了一次目标跟踪。

整体程序流程描述如下:

1. 首先在图像上选定一个目标区域作为 roi
2. 计算 roi 的 HSV 色彩空间的直方图
3. 通过直方图反投影计算图像 a 中每个点与选定 roi 区域直方图的相似程度，使用 menashift 算法将选定区域区域沿着最为相似的区域移动，算法收敛时即完成了图像 a 中的目标跟踪。
4. 重复 3、4 步骤即完成了整个视频的目标跟踪

1.2 Camshift opencv 调库实现

A．程序片段

```
#include <opencv2/opencv.hpp>
```

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include<iostream>
#include<cstdio>
using namespace std;
using namespace cv;
const int seed = 42;

int main()
{

    VideoCapture cap;
    cap.open("D:\\study\\CV\\assignment3\\in5.mp4");

    if (!cap.isOpened())
    {
        cerr << "Failed to open video file." << endl;
        return -1;
    }

    Mat  frame;
    int fps = cap.get(CAP_PROP_FPS);
    Size S = Size((int)cap.get(CAP_PROP_FRAME_WIDTH),
        (int)cap.get(CAP_PROP_FRAME_HEIGHT));
    VideoWriter writer("D:\\study\\CV\\assignment3\\out5.mp4", CAP_OPENCV_MJPEG, fps,
S, true);

    cv::namedWindow("Object Tracker");
    cap >> frame; // 读取一帧
    cv::Rect track_window = cv::selectROI("Object Tracker", frame);

    Mat roi = frame(track_window);
    Mat hsv_roi,mask;
    cvtColor(roi, hsv_roi, COLOR_BGR2HSV);
    inRange(hsv_roi, Scalar(0, 60, 32), Scalar(180, 255, 255), mask);
    float range_[] = { 0, 180 };
    const float* range[] = { range_ };
    Mat roi_hist;
    int histSize[] = { 180 };
    int channels[] = { 0,1,2 };
    calcHist(&hsv_roi, 1, channels, mask, roi_hist, 1, histSize, range);
    normalize(roi_hist, roi_hist, 0, 255, NORM_MINMAX);
```

```
    TermCriteria term_crit(TermCriteria::EPS | TermCriteria::COUNT, 10, 1);




    while (true)
    {
        cap >> frame;
        if (frame.empty())
        {
            break;
        }

        Mat hsv, dst;
        cvtColor(frame, hsv, COLOR_BGR2HSV);
        calcBackProject(&hsv, 1, channels, roi_hist, dst, range);
        RotatedRect rot_rect = CamShift(dst, track_window, term_crit);

        Point2f points[4];
        rot_rect.points(points);
        for (int i = 0; i < 4; i++)
            line(frame, points[i], points[(i + 1) % 4], 255, 2);

        imshow("Video", frame);
        writer.write(frame);


        if (waitKey(20) == 27)
        {
            break;
        }
    }

    // 释放资源
    cap.release();
    writer.release();
    destroyAllWindows();
    return 0;


}
```

B．程序说明及原理解释

Camshift 与 meanshift 进行目标跟踪的程序整体上一致，只是寻找最相似区域时调用的函数是 cv::CamShift 而不是 cv::meanShift。camshift 算法全称“continuously adaptive mean-shift”，是对 meanshift 算法的改进算法，可随着跟踪目标的大小变化实时调整搜索窗口的大小。Camshift 同样使用 meanshift 去迭代收敛到最相似的区域，一旦 meanshift 收敛，随后会更新窗口大小，并且旋转矩形窗口到一个合适的位置。

1.3 Opencv 相关接口介绍

A．selectROI

```
cv::Rect selectROI(
        const cv::Mat& img,
        bool showCrosshair = true,
        bool fromCenter = false
);
```

该接口会弹出一个新的窗口，在窗口上用鼠标拖动选择感兴趣区域。选择完成后，该接口会返回一个 cv::Rect 对象，表示选择的矩形区域的左上角坐标和宽高。

B．cvtColor

```
void cv::cvtColor(
        InputArray src,
        OutputArray dst,
        int code,
        int dstCn = 0
);
```

该接口按照参数 code 指定的转换样式将 src 转换，输出到 dst。在本实验使

用的是 cv::COLOR_BGR2HSV，表示将 RGB 图像转为 HSV 图像

C．calcHist

```
void cv::calcHist(
    const Mat* images,
    int nimages,
    const int* channels,
    InputArray mask,
    OutputArray hist,
    int dims,
    const int* histSize,
    const float** ranges,
    bool uniform = true,
    bool accumulate = false
);
```

该接口 images 输入可以是一个或者多个图像, nimages 指定输入图像的数量, channels 指定使用哪些通道来计算直方图，只有在 mask 区域内的像素才会用于计算，hist 是输出，dims 是直方图的维度通常为 1（表示一个向量），histSize 表示直方图每个维度的桶数（将图像值离散地划分到对于范围），ranges 表示每个维度直方图值的范围。

D．calcBackProject

```
void cv::calcBackProject(
    const Mat* images,
    int nimages,
    const int* channels,
    InputArray hist,
    OutputArray backProject,
    const float** ranges,
    double scale = 1,
```

bool uniform = true
);

Images、nimages、channels 参数含义与直方图计算类似不再赘述，hist 表示输入的直方图，backProject 是输出的反向投影图像，和输入图像有相同的尺寸和深度。

E . meanShift

```
int cv::meanShift(
    InputArray probImage,
    Rect& window,
    TermCriteria criteria
);
```

probImage 是输入的反向投影图像，window 表示目标所在矩形初值，既是输入也是输出，函数运行后，表示找到的目标区域位置大小，criteria 是算法停止的条件。

G. CamShift

```
RotatedRect cv::CamShift(
    InputArray probImage,
    Rect& window,
    TermCriteria criteria
);
```

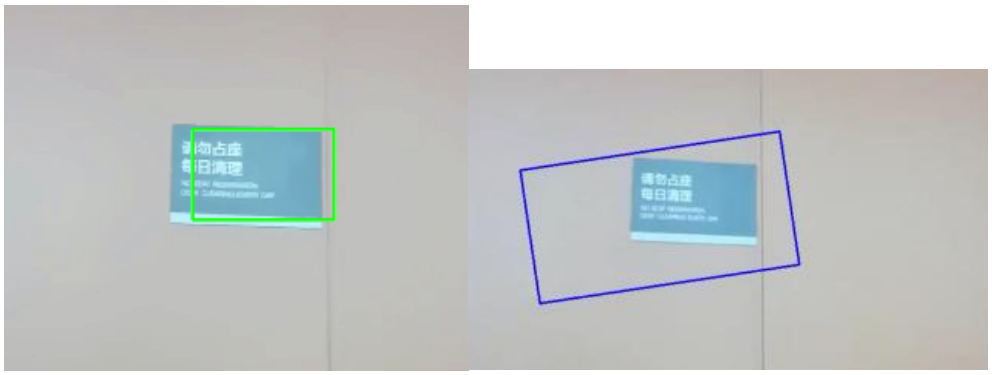
与 meanshift 类似，只是返回值是一个 RotatedRect，表示找到的目标区域的位置、大小和旋转角度。（因为 camshift 会调整窗口大小、角度，所以不能用一个 rect 的类型来表示了）

2 结果与讨论

2.1 实验结果及效果异同

Meanshift 的框变化较为稳定，但是大小固定，对于大小变化的物体效果不好。

Camshaft 能够动态的改变框的大小，但是没有那么稳定，实测效果感觉不如 meanshift。



如图，左图为 meanshift，有图为 camshift。当物体速度较快时，camshift 追踪效果明显差于 meanshift。

如图，左图为 meanshift，有图为 camshift。当物体变大时，meanshift 由于大小固定，无法跟踪，而 camshift 则可以继续跟踪物体

2.2 Bonus：camshift 如何实现动态改变搜索框大小

Camshaft 就是在 meanshift 收敛后，进一步对矩形进行调优，改变其大小与

旋转角度。

在 opencv 中，camshift 源码如下：

```
cv::RotatedRect cv::CamShift( InputArray _probImage, Rect& window,
                              TermCriteria criteria )
{
    CV_INSTRUMENT_REGION();

    const int TOLERANCE = 10;
    Size size;
    Mat mat;
    UMat umat;
    bool isUMat = _probImage.isUMat();

    if (isUMat)
        umat = _probImage.getUMat(), size = umat.size();
    else
        mat = _probImage.getMat(), size = mat.size();

    meanShift( _probImage, window, criteria );
```

cv::InputArray _probImage

这一部分对图像进行了 meanshift 之后，开始调整搜索框大小、角度

```
window.x -= TOLERANCE;
if( window.x < 0 )
    window.x = 0;

window.y -= TOLERANCE;
if( window.y < 0 )
    window.y = 0;

window.width += 2 * TOLERANCE;
if( window.x + window.width > size.width )
    window.width = size.width - window.x;

window.height += 2 * TOLERANCE;
if( window.y + window.height > size.height )
    window.height = size.height - window.y;
```

对 meanshift 收敛的 window 的左上角坐标、窗口长宽做一定调整，以适应大小的变化。

后面就是一些看不懂的过程了，涉及到一些复杂的数学推导，需要去阅读 camshift 的论文才能搞懂，将后续代码的截图贴上。未来有时间可以深究。

论文：Computer Vision Face Tracking For Use in a Perceptual User Interface [archive]

erreur modèle {{Lien archive}} : renseignez un paramètre « |titre= », Intel Technology Journal, No. Q2. (1998)

```
// Calculating moments in new center mass
Moments m = isUMat ? moments(umat(window)) : moments(mat(window));

double m00 = m.m00, m10 = m.m10, m01 = m.m01;
double mu11 = m.mu11, mu20 = m.mu20, mu02 = m.mu02;

if( fabs(m00) < DBL_EPSILON )
    return RotatedRect();

double inv_m00 = 1. / m00;
int xc = cvRound( m10 * inv_m00 + window.x );
int yc = cvRound( m01 * inv_m00 + window.y );
double a = mu20 * inv_m00, b = mu11 * inv_m00, c = mu02 * inv_m00;
```

```
// Calculating width & height
double square = std::sqrt( 4 * b * b + (a - c) * (a - c) );
```

```
// Calculating orientation
double theta = atan2( 2 * b, a - c + square );
```

```
// Calculating width & length of figure
double cs = cos( theta );
double sn = sin( theta );

double rotate_a = cs * cs * mu20 + 2 * cs * sn * mu11 + sn * sn * mu02;
double rotate_c = sn * sn * mu20 - 2 * cs * sn * mu11 + cs * cs * mu02;
rotate_a = std::max(0.0, rotate_a);  // avoid negative result due calculation numeric errors
rotate_c = std::max(0.0, rotate_c);  // avoid negative result due calculation numeric errors
double length = std::sqrt( rotate_a * inv_m00 ) * 4;
double width = std::sqrt( rotate_c * inv_m00 ) * 4;
```

```
// In case, when tetta is 0 or 1.57... the Length & Width may be exchanged
if( length < width )
{
    std::swap( length, width );
    std::swap( cs, sn );
    theta = CV_PI*0.5 - theta;
}
```

```
// Saving results
int _xc = cvRound( xc );
int _yc = cvRound( yc );

int t0 = cvRound( fabs( length * cs ));
int t1 = cvRound( fabs( width * sn ));

t0 = MAX( t0, t1 ) + 2;
window.width = MIN( t0, (size.width - _xc) * 2 );

t0 = cvRound( fabs( length * sn ));
t1 = cvRound( fabs( width * cs ));

t0 = MAX( t0, t1 ) + 2;
window.height = MIN( t0, (size.height - _yc) * 2 );

window.x = MAX( 0, _xc - window.width / 2 );
window.y = MAX( 0, _yc - window.height / 2 );

window.width = MIN( size.width - window.x, window.width );
window.height = MIN( size.height - window.y, window.height );
```

```
RotatedRect box;
box.size.height = (float)length;
box.size.width = (float)width;
box.angle = (float)((CV_PI*0.5+theta)*180./CV_PI);
while(box.angle < 0)
    box.angle += 360;
while(box.angle >= 360)
    box.angle -= 360;
if(box.angle >= 180)
    box.angle -= 180;
box.center = Point2f( window.x + window.width*0.5f, window.y + window.height*0.5f);

return box;
```