

**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 3](#_Toc130469963)

[1.1 Meanshift opencv调库实现 3](#_Toc130469964)

[1.2 Camshift opencv调库实现 6](#_Toc130469965)

[1.3 Opencv相关接口介绍 9](#_Toc130469966)

[2 结果与讨论 11](#_Toc130469967)

[2.1 实验结果及效果异同 12](#_Toc130469968)

[2.2 Bonus：camshift如何实现动态改变搜索框大小 12](#_Toc130469969)

# 程序

## 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

## 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。

## Opencv相关接口介绍

1. **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。

1. **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。

1. **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

);

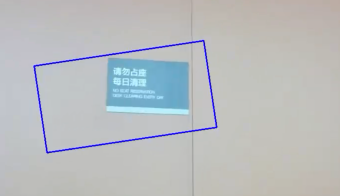
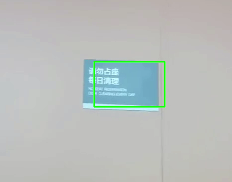
与meanshift类似，只是返回值是一个 RotatedRect，表示找到的目标区域的位置、大小和旋转角度。（因为camshift会调整窗口大小、角度，所以不能用一个rect的类型来表示了）

# 结果与讨论

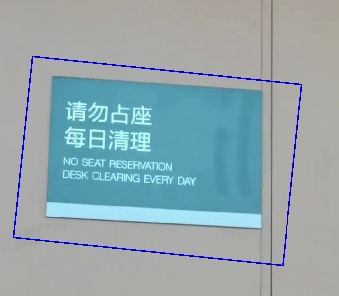
## 实验结果及效果异同

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。

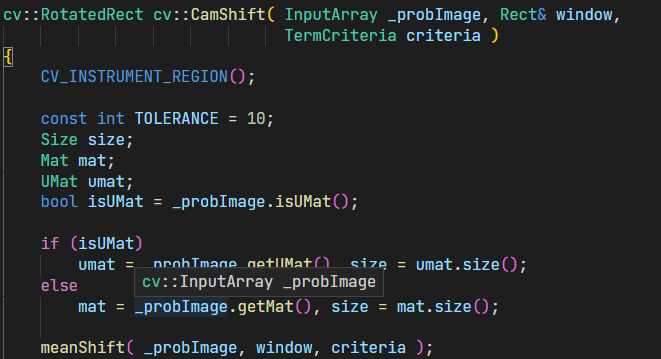


如图，左图为meanshift，有图为camshift。当物体变大时，meanshift由于大小固定，无法跟踪，而camshift则可以继续跟踪物体

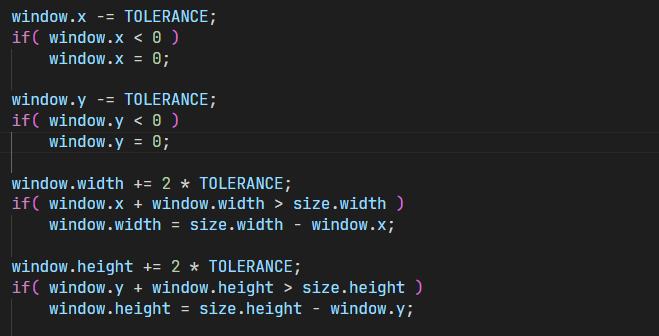
## Bonus：camshift如何实现动态改变搜索框大小

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：



这一部分对图像进行了meanshift之后，开始调整搜索框大小、角度



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)

