帧差法的运动检测在 RM 防御塔的应用

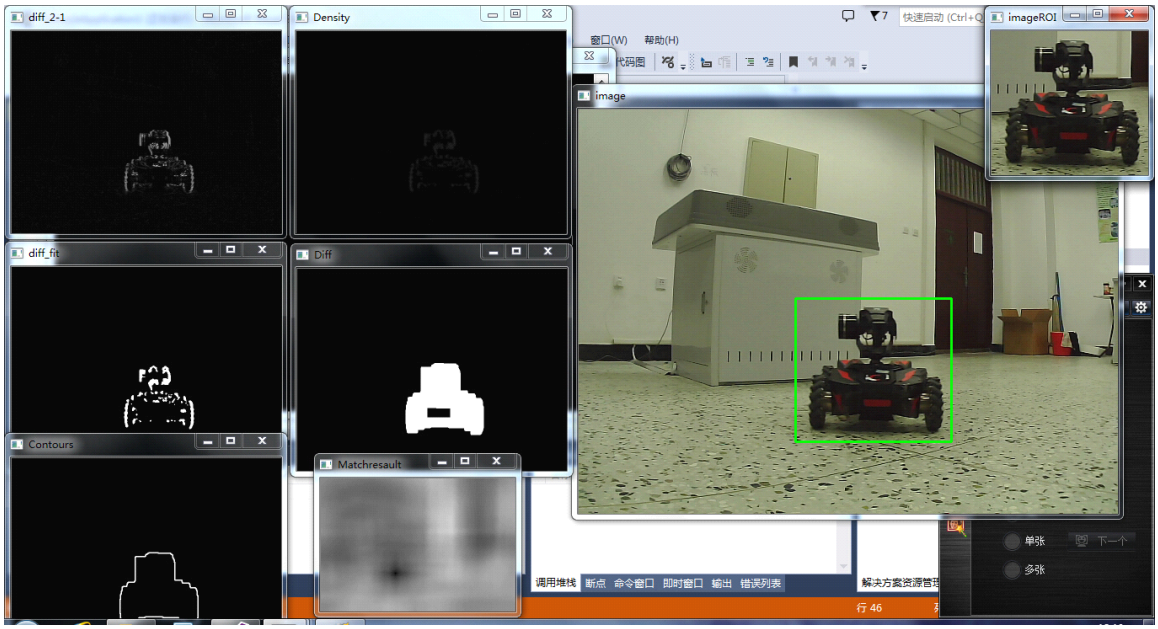
概述：

运动目标的检测是应用视觉研究领域的一个重要课题,也被看作是计算机视觉系统的一个重要能力,在智能监测以及军事、工业等领域都有着广泛的应用前景。

检测运动目标的方法主要有光流法和差图像法。一般来说,光流法的算法复杂,运算量较大,实时性和实用性较差。与之相反,差图像法的原理和算法比较简单,易于实现且实时性好,因而成为目前广泛应用的运动目标检测方法。

本程序采用帧差法进行运动物体提取。常用的帧差法通常为三帧差法，以相邻两帧的差分图像相与进行噪声的分离与细化运动物体的外轮廓，提高检测精度。但是三帧差法对于低速的物体，尤其是小物体在低速做径向运动的时候十分迟钝，效果很不理想。所以本程序采用两帧差法，以提高对小幅度移动物体的灵敏度。

以下是程序运行时的整体截图。



程序思路点拨：

首先，通过本帧和前一帧做差分在对差分图像取绝对值，可得出相对与背景的运动物体的外围轮廓。如图：

Code：

```
cv::cvtColor(image, imageTem_gray1, CV_BGR2GRAY); //颜色转换
cv::cvtColor(image_pre, imageTem_gray2, CV_BGR2GRAY);  //颜色转换
cv::absdiff(imageTem_gray1, imageTem_gray2, image_diff);  //做两帧差分
```

其中 image image_pre 分别是本帧图像 和 上一帧图像 image_diff 是差分图像，即时上图。

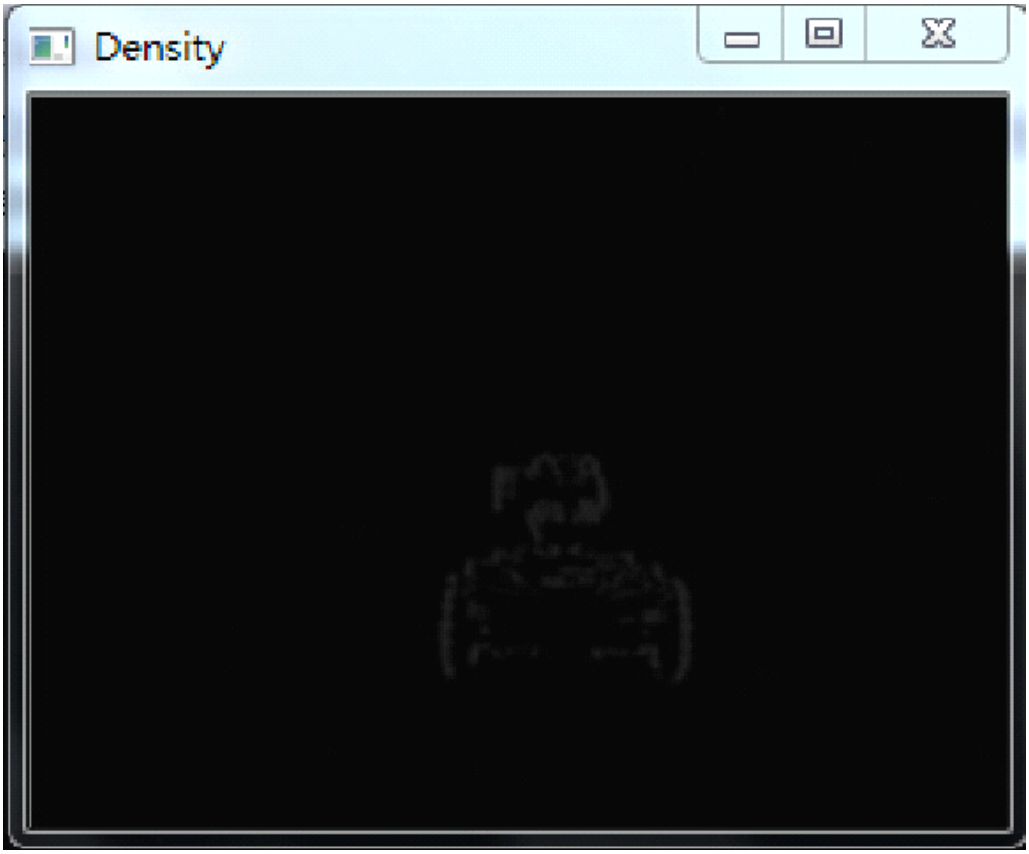
做差分时需要对灰度值做差分，故应首先进行颜色转换。

可以看出运动的战车轮廓被很好的提取出来，同时因为光照，传感器噪声等因素，在差分图像中存在存在着相当多的类似于椒盐躁点的噪声。这些噪声对于我们的目标提取造成了很大的阻碍，一个方法就是采

用中值滤波对本图处理，可以很好的抑制噪声，但是实测中值滤波的运算量过大，使得整个系统的实时性降低，得不偿失。

通过观察整幅图片，我们可以看到噪点的分布呈现均匀化，而战车轮廓附近的像素点非常集中且连续，换句话说，如果我们以 5*5 像素作为一个单位面积，对上图的白色像素统计密度，可以想象得出椒盐噪声所处的单位面积所对应的密度很低且比较平均，而战车轮廓处的白色像素所处位置的像素密度很高。故作者采用了将本幅图片转换到密度域下，二值化掉密度小的区域，即可得到没有噪点（或者极少噪点）的前景图像轮廓。

密度图像如图所示



Code：

```
cv::threshold(image_diff, image_diff, MotingDetectThreshold_Value, 255, cv::THRESH_BINARY);
PixelDensitySstatistic(image_diff);//统计像素密度
cv::threshold(image_Density, image_diff, 10, 255, cv::THRESH_BINARY); //密度图像二值化
```

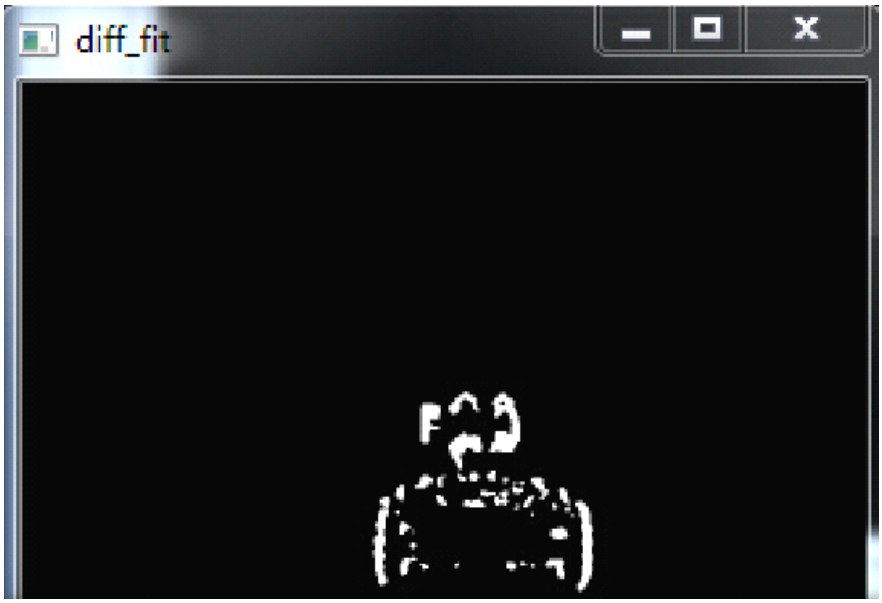
首先对差分图像做二值化，再进行密度统计。密度统计结果为上图，对密度统计的结果进行二值化后的结果如下图。

给出密度统计函数：

```
void DefensiveTower::PixelDensitySstatistic(cv::Mat imgbin)
{
    cv::Mat image_bin;
    imgbin.copyTo(image_bin);
    cv::threshold(image_bin, image_bin, 10, 1, CV_8U);
    cv::Mat Kernel(Density_Element_Size, Density_Element_Size, CV_8U, cv::Scalar(1));
    cv::filter2D(image_bin, image_Density, image_bin.depth(), Kernel);
}
```

*密度函数通过一个 2D 滤波器改写而成，其本质是一个 5*5 的元素全是 1 的矩阵对被统计图片的卷积。*

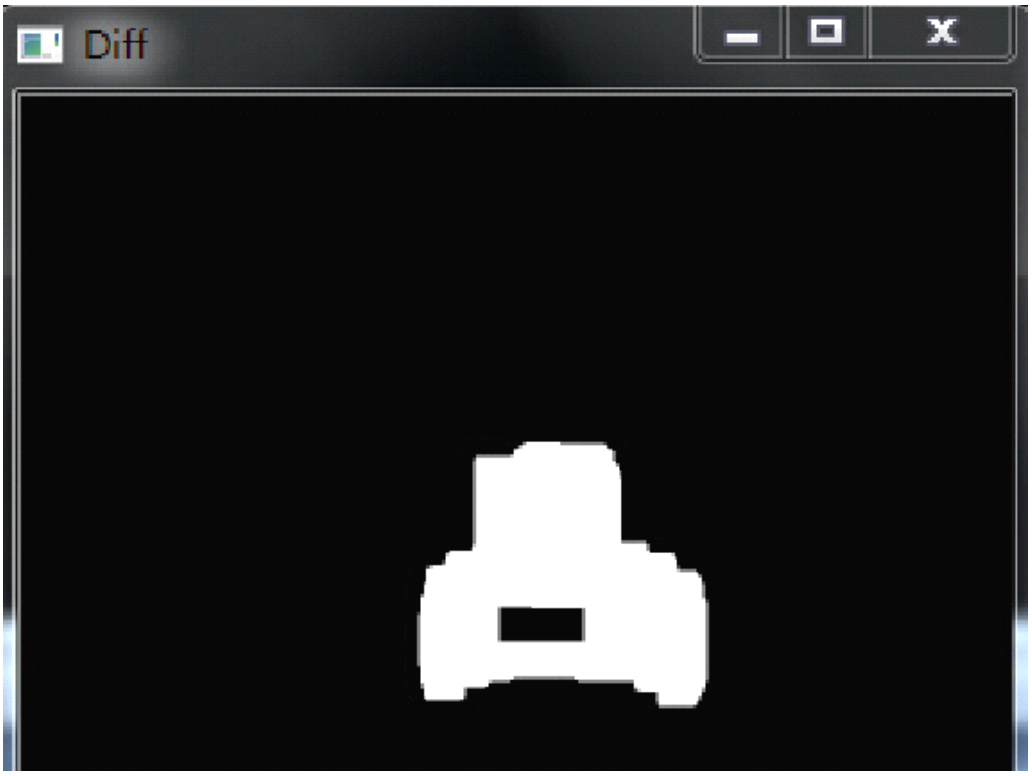
密度图像二值化后的结果如图：



可以看到，通过密度二值化后的图像对椒盐躁点有了很好的抑制，效果非常理想。

之后对移除噪声后的图片进行简单的图像形态学运算，就可得到图像的运动区域，以便下一步进行连通区域的查找。

膨胀 20 次外加腐蚀 10 次的效果：

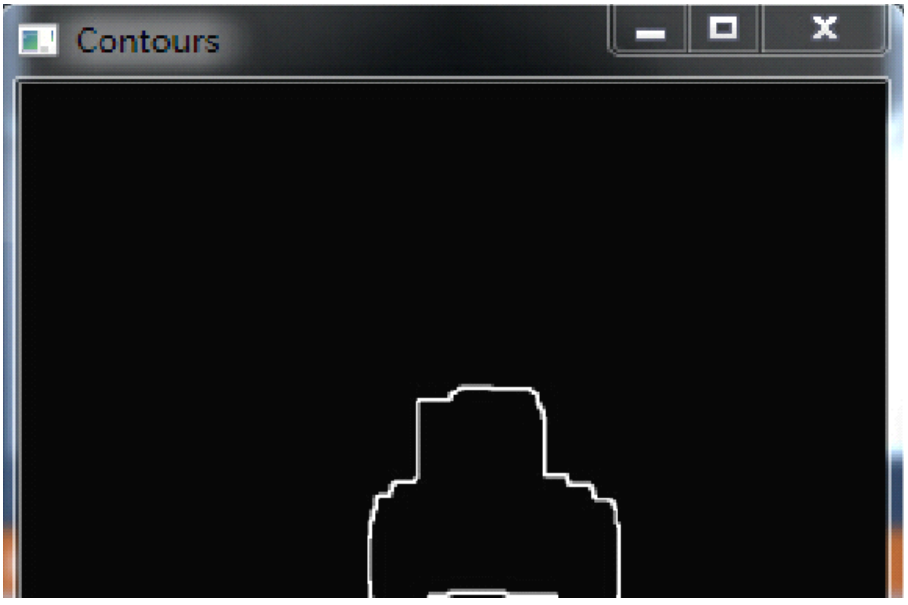


Code：

```
cv::dilate(image_diff, image_diff, cv::Mat(), cv::Point(-1, -1), 20);//结果差分图像进行膨胀
cv::erode(image_diff, image_diff, cv::Mat(), cv::Point(-1, -1), 10);//结果差分图像进行膨胀
```

可以看到效果十分明显，运动区域被完全覆盖。

查找连通区并进行绘制连通区外轮廓：

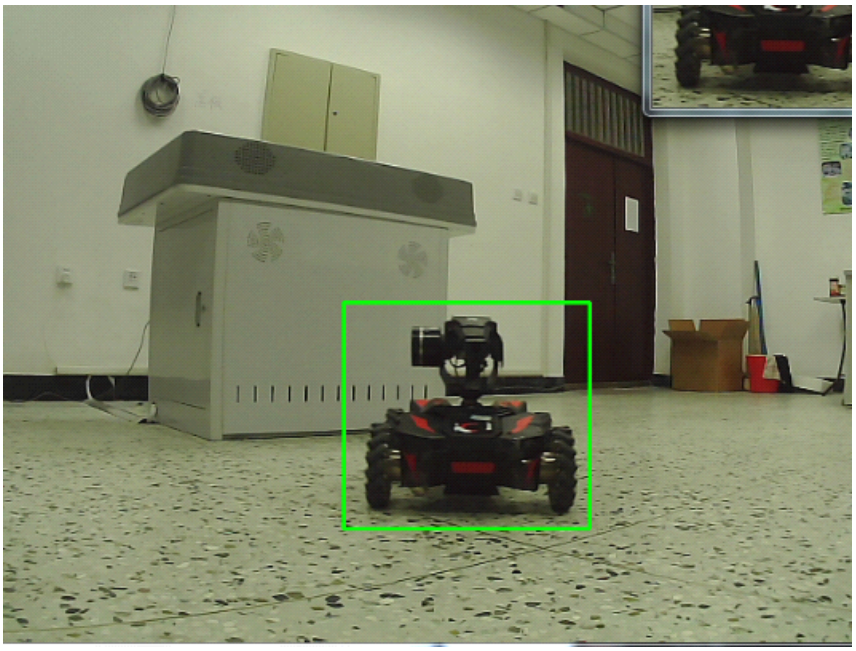


Code：

```
// 扫描连通区域
image_diff.copyTo(imageTem_diff);
std::vector<std::vector<cv::Point>> contours;
cv::findContours(imageTem_diff, //寻找连通区域
    contours, // a vector of contours
    CV_RETR_EXTERNAL, // retrieve the external contours
    CV_CHAIN_APPROX_NONE); // retrieve all pixels of each contour

// 挑选连通区域
int cmin = 300;  // minimum contour length
int cmax = 2000; // maximum contour length
int tem_erase_num = 0;
std::vector<std::vector<cv::Point>>::
    const_iterator itc = contours.begin();
while (itc != contours.end()) {

    if (itc->size() < cmin || itc->size() > cmax)
    {
        itc = contours.erase(itc);
        tem_erase_num++;
    }

    else
        ++itc;
}
```

提取连通区域并且通过连通区域的周长去除连通区。详情参见 Opencv2 计算机视觉编程 P160

最后将此轮廓为参数查找最小包络矩形，可得到运动目标的位置。



Code：

```
Loc_Target = cv::boundingRect(cv::Mat(contours[0]));
```

boundingRect 可以返回连通区域的最小包络矩形。详细介绍参见 Opencv2 计算机视觉编程 P164。

最后以此矩形的位置与大小提取目标外观，以供开发员参考。

写在最后

在实践中，如果你仅仅按照上述的方法进行了程序设计，会发现在物体由运动到停止的过程中，会识别出很多零散区域，造成运动跟踪紊乱，在战车停止后造成丢失目标，甚至即使是已经完成目标识别后，摄像头有请问震动使发生目标丢失的现象。

针对于在目标在运动到停止的过程中所出现的零散识别现象可以通过限制上次和本次目标识别矩形的大小位置变化速度来解决。

Code：

```
if ((State&(STATE_FramdiffLocking|STATE_MatchLocking)))//如果上次目标锁定
{
    if (                                                    //本次与上次的位置信息差别很大
        abs(Loc_Target_pre.height - Loc_Target.height)>30 ||  // 高度变化大于阈值
        abs(Loc_Target_pre.width - Loc_Target.width)>30 ||    // 宽度变化大于阈值
        abs((Loc_Target_pre.x - Loc_Target.x)>50) ||          // x变化大于阈值
        abs((Loc_Target_pre.y - Loc_Target.y)>50)             // y变化大于阈值
        )
    {
        tem_TargetGet = 0;//检测不成功
    }
}
```

```
if (((double)Loc_Target.height / (double)Loc_Target.width) < 0.67 ||  //如果长宽比不为 1：2
    ((double)Loc_Target.height / (double)Loc_Target.width) >1.5)
{
    tem_TargetGet = 0;
}

if (tem_erase_num > 2)  //如果剔除的小连通区域大于2个 认为是在抖动
{
    tem_TargetGet = 0;
    tem_Delay = 1;
}
if (tem_Delay) //抖动后等待图像稳定
{
    tem_Delay--;
    tem_TargetGet = 0;
}
```

以上条件是程序的部分截取，其中涉及了一些下一章要讲的内容。希望大家继续期待。

而对于识别目标后，目标停止运动的跟踪则是通过另一种模版匹配的方法进行。模版匹配的方式可以对已经识别的物体进行最优匹配，并且没有对摄像头严格稳定的要求，是我下一篇文章所要介绍的东西，也希望各位能够继续关注我们视觉组的研究进展，提出宝贵意见，大家一起探讨学习，共同进步。

当然，在我写完下一章的内容后，支持的人数超过 100 人次，会将演示程序的工程（vs2015）和所有函数分享给大家。

侯小猴

东北林业大学 大学生创新实验室