华中科技大学

《计算机视觉》

基于 SIFT 和 KCF 的运动目标匹配与跟踪

院　　系	人工智能与自动化学院
专业班级	自实 1701、校交 1701
姓　　名	李星毅、曾文正
学　　号	U201712072、U201715853
指导教师	桑农、高常鑫

2020 年 5 月 20 日

目录

1. 摘要

本次编程实验我们从日常生活出发，运用计算机视觉算法解决实际问题。我们利用 SIFT 与 KCF 实现了运动目标的识别匹配与跟踪。该算法有着很广的应用，如无人机跟拍、导弹追踪目标、追踪人手实现人机交互等。我们的程序模拟了无人机跟拍的过程，此外，我们还开发了一款游戏，无须使用键盘和鼠标，利用摄像头追踪手的位置实现人在游戏中的操作。

2. 应用背景与任务描述

2.1 应用背景

目标检测匹配与跟踪算法有着很广的应用，如无人机自动跟拍、导弹追踪目标、追踪人手实现人机交互等。

2.2 任务描述

- 给定目标（以一张图片的方式方程告诉程序），要求程序能够根据所给目标的图片实时跟踪该目标在一段视频中的位置。
- 实现一个简单的飞机扫射游戏，计算机通过内置摄像头跟踪玩家的手，玩家通过移动手部，可以实现对游戏中的飞机的操控。

3. 问题分析

上述两个任务的核心都是需要在视频中检测出特定目标并跟踪。对于检测，需要把给定的目标图像与现在程序捕获到的一帧视频中的目标进行匹配。为了提高匹配的精度，并且让匹配有较好的鲁棒性，需要选择合适的特征。我们选择了 SIFT 特征，由于 SIFT 描述子的检测速度较慢，无法满足实时性，因此我们只在第一帧图像中使用了 SIFT 检测算法。我们假设目标在第一帧出现，那么当程序在第一帧检测出目标并框出后，由于我们的目的是跟踪这个特定目标，因此只要在第一帧检测到目标，剩下的任务就可以交给跟踪算法，而不需要每一帧都进行目标检测。我们选择了 KCF 作为跟踪算法，KCF 具有较好的精度和极好的实时性。通过跟踪得到结果后，进行显示等处理。对于处理视频，在视频每一帧中把框画出，在游戏中，根据跟踪得到的位置映射到游戏内部的操作。

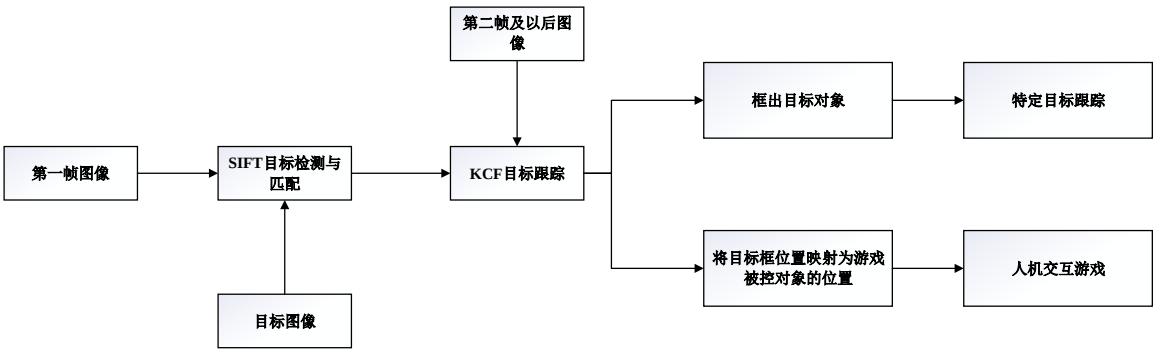

图 1 问题分析流程图

4. 算法原理

4.1目标检测与匹配

4.1.1 SIFT 综述

SIFT 的全称是 Scale Invariant Feature Transform，即尺度不变特征变换，由英属哥伦比亚大学教授 David G. Lowe 提出。SIFT 特征具有旋转、尺度、亮度变化不变性，是一种非常稳定的局部特征。SIFT 算法的实质是在不同的尺度空间上查找特征点，并计算出特征点的方向。SIFT 所查找到的特征点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

4.1.2 尺度空间极值检测

首先对图像做不同尺度的高斯模糊$L(x,y,\sigma)=G(x,y,\sigma)\otimes I(x,y)$以及对图像做降采样，构造高斯金字塔，然后利用同一组内图像做差$D(x,y,\sigma)=L(x,y,k\sigma)-L(x,y,\sigma)$构造高斯差分（DoG）金字塔。对 DoG 金字塔中的每一像素，在三维邻域（两维图像、一维尺度）内判断是否为极值点，可以确定极值点的位置及尺度，极值的大小反映了极值处的对比度（反差）。

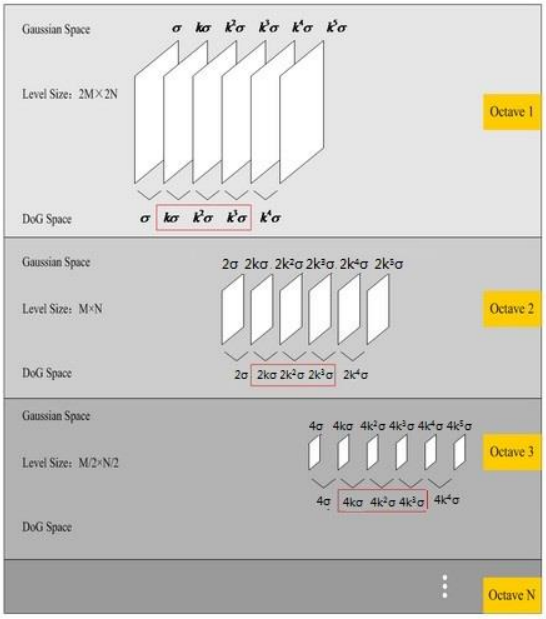

图 2 高斯金字塔示意图

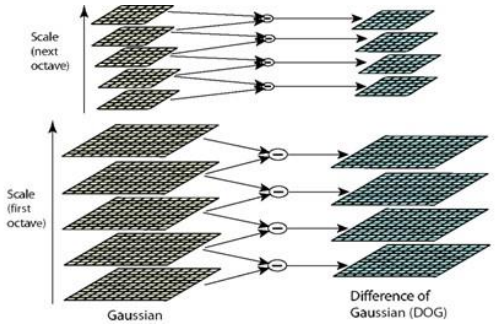

图 3 高斯差分金字塔的构建

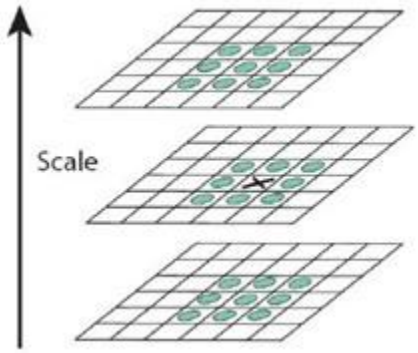

图 4 尺度空间上极值搜索

4.1.3 关键点定位

由于图像是离散化的，真正的极值点并不一定在整数点上，因此进行亚像素（子像素）精确定位：将原点移到极值点处，并将高斯差分在该极值点处展开，保留一阶项及二阶项，得到：

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

其中$\mathbf{x} = (x, y, \sigma)^T$。

令$D'(\mathbf{x}) = 0$，得：

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2}\frac{\partial D}{\partial \mathbf{x}}$$

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}}$$

去除对比度较低的不稳定极值点：利用$\hat{\mathbf{x}}$处的高斯差分值$D(\hat{\mathbf{x}})$，设对比度阈值为T，则当$|D(\hat{\mathbf{x}})| < T$时，去除极值点$\hat{\mathbf{x}}$。

去除边缘点：由于 DoG 算子对边缘点也很敏感，在边缘处也有很大的响应值。利用 Hessian 矩阵来处理边缘点。在极值点处，利用$D(\hat{\mathbf{x}})$计算 Hessian 矩阵

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$$

$D(\hat{\mathbf{x}})$的主曲率，在边缘梯度的方向上比较大，而在沿着边缘的方向上则较小，且与H的特征值成正比。利用矩阵H两个特征值的和等于H的迹，两个特征值的积等于H的行列式，有

$$\frac{Tr^2(H)}{det(H)} = \frac{(r+1)^2}{r}$$

其中r为H的大特征值与小特征值之比。

设关于r的阈值为T_r，则当$\frac{Tr^2(H)}{det(H)} > \frac{(T_r+1)^2}{T_r}$时，判断该极值点为边缘点，并去除。

4.1.4 获取关键点主方向

根据关键点的尺度，在高斯金字塔对应图像的关键点位置处的局部区域计算梯度方向直方图。直方图峰值对应得方向即为关键点的主方向。

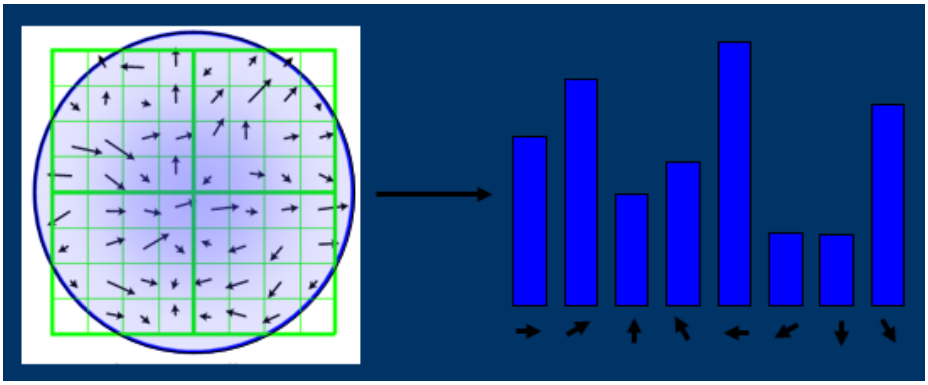

图 5 关键点主方向的获取

4.1.5 关键点描述

首先确定用于计算关键点描述的局部区域，然后根据关键点主方向变换坐标系，在局部区域内对每个像素点求其梯度幅值和方向，生成方向直方图，最后为克服光照影响，将描述子向量归一化。

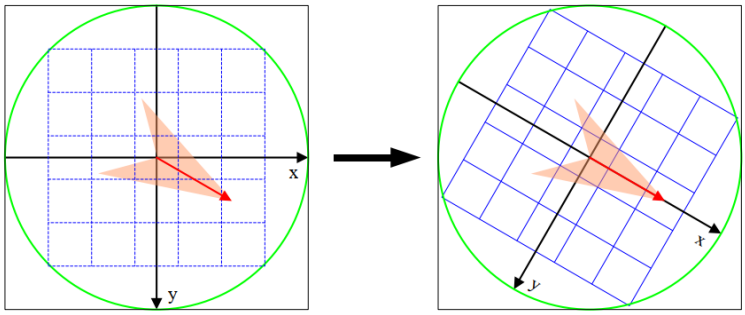

图 6 根据关键点主方向变换坐标系

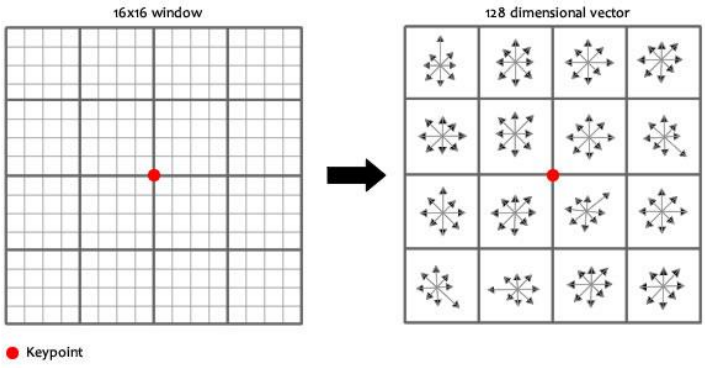

图 7 在局部区域内对每个像素点求其梯度幅值和方向，生成方向直方图

4.2目标跟踪

4.2.1 KCF 综述与总体实现流程图

KCF 为核化相关滤波器。算法使用目标扩展后区域的循环矩阵采集正负样本，利用岭回归训练检测器，并用核方法将线性空间映射到更高维的非线性空间。核方法与循环矩阵的性质让运算极大地简化，满足了实时性的要求。主要思想为根据当前帧的信息（提取特征）和之前帧的特征训练出一个相关滤波器，然后与新输入的帧进行相关性计算，得到的置信图就是预测的跟踪结果，显然，得分最高的那个点（或者块）就是最可能的跟踪结果。

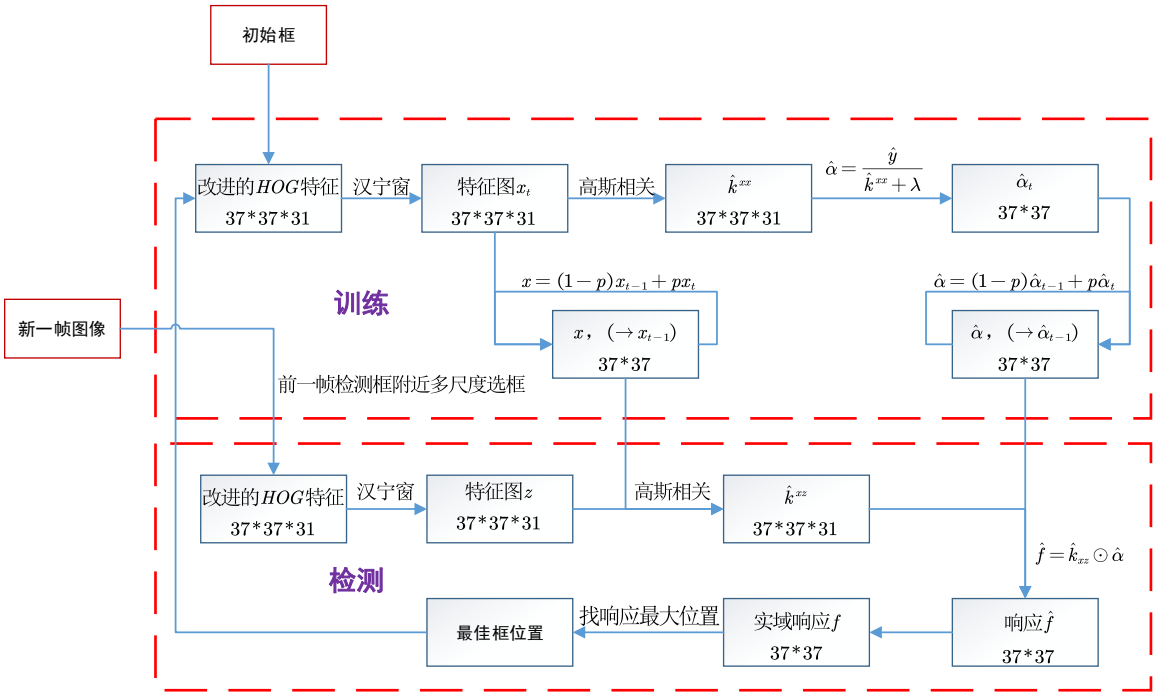

图 8 KCF 总体实现流程图

4.2.2 岭回归

设训练样本集(x_i,y_i)，样本及标签都为列向量，那么其线性回归函数$f(x_i)=w^Tx_i$，w是一个列向量，表示权重系数，可通过最小二乘法求解，写成矩阵形式为：

$$w=\min_w\|Xw-y\|^2+\lambda\|w\|^2$$

其中$X=[x_1,x_2,...,x_n]^T$，即每行表示一个样本，y 是列向量，表示每个样本对应的标签，λ是正则化参数，防止过拟合。

令上式对 w 求偏导，让导数等于 0，可以求得：

$$w = (X^T X + \lambda I)^{-1} X^T y$$

上式是在实数域的解，在该算法中需要在傅里叶域内进行计算，涉及复数，所以把解改写为复数域形式：

$$w = (X^H X + \lambda I)^{-1} X^H y$$

其中X^H表示共轭转置矩阵，即$X^H = (X^*)^T$。

4.2.3 循环移位与循环矩阵

一般来说，正样本容易获取，而好的负样本在整个跟踪过程中不好人为获取，但负样本往往又对分类器的性能起重要作用。KCF 利用循环移位得到跟多的负样本，这些负样本具有代表性，对分类器的性能有很多帮助，并且循环矩阵自身有很好的特性，可以极大地加速运算。

一维情况下：设$x = [x_1, x_2, ..., x_n]^T$，

$$X = C(x) = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ x_n & x_1 & x_2 & \dots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \dots & x_{n-2} \\ \dots & \dots & \dots & \dots & \dots \\ x_2 & x_3 & x_4 & \dots & x_1 \end{bmatrix}$$

在二维情况下，效果是这样的：

图 9 图像竖直方向循环移位示例

通过循环偏移得到的图像在边界处并不是很平滑，消除这种现象的方式就是通过图像乘以一个汉宁窗来降低边缘图像的权重。

所有的循环矩阵都能通过离散傅里叶变换（DFT）在傅式空间实现对角化，如下式：

$$X = F \,\mathrm{diag}(\hat{x}) F^H$$

其中 F 是离散傅里叶矩阵，是常量，DFT 是个线性操作，$\hat{x} = \mathcal{F}(x) = \sqrt{n}\,Fx$ 表示 DFT 变换结果，是个对角阵。

循环矩阵的计算可以直接把所有的样本都转换为对角矩阵进行处理，因为在循环矩阵对样本进行处理的时候，样本并不是真实存在的样本，存在的只是虚拟的样本，可以直接利用循环矩阵所特有的特性，直接把样本矩阵转换为对角矩阵进行计算，这样可以大幅度加快矩阵之间的计算，因为对角矩阵的运算只需要计算对角线上非零元素的值即可。

4.2.4 引入核函数提速

将线性转为非线性：$w=\sum_i \alpha_i \varphi(x_i)$，$\varphi(x_i)$把特征映射到了高维空间。

$\varphi^T(x)\varphi(x')=k(x,x')$类似于核空间变量的求协方差，结果存在核矩阵 K 中，K 是所有训练样本的核相关矩阵：

$$K_{ij}=k(x_i,x_j)$$

需要优化的目标函数变为：

$$\min_{\alpha} \|\varphi^T(X)\varphi(X)\alpha-y\|^2+\lambda\|\varphi^T(X)\alpha\|^2$$

引入核方法，得到非线性回归函数如下：

$$f(z)=w^Tz=\sum_{i=1}^{n}\alpha_i k(z,x_i)$$

核空间的岭回归的解为：

$$\alpha=(K+\lambda I)^{-1}y$$

进一步利用循环矩阵的对角化化简：

$$\begin{aligned}\alpha&=\left(F\,\mathrm{diag}\left(\hat{k}^{xx}\right)F^H+\lambda FIF^H\right)^{-1}y\\&=\left(F\,\mathrm{diag}\left(\hat{k}^{xx}+\lambda\right)F^H\right)^{-1}y\\&=F\,\mathrm{diag}\left(\frac{1}{\hat{k}^{xx}+\lambda}\right)F^Hy\\&=C\left(\mathscr{F}^{-1}\left(\frac{1}{\hat{k}^{xx}+\lambda}\right)\right)y\end{aligned}$$

左右两边同时做 DFT 得：

$$\hat{\alpha}=\left(\frac{1}{\hat{k}^{xx}+\lambda}\right)^{*}\odot\hat{y}$$

k^{xx}是核相关矩阵的第一行。对于k_i^{xx}与k_{N-i}^{xx}，两者都是同一个向量和自身移位结果进行运算。而所有涉及到的核函数都只和位移的绝对值有关，所以$k_i^{xx}=k_{N-i}^{xx}$，k^{xx}具有对称性。对称向量的傅里叶变换为实数。所以：

$$\hat{\alpha}=\frac{\hat{y}}{\hat{k}^{xx}+\lambda}$$

每次训练得到的参数受以往训练得到的参数的影响，有一个加权的过程，即

$$\hat{\alpha}_t=(1-p)\hat{\alpha}_{t-1}+p\hat{\alpha}_t$$

4.2.5 特征提取

使用了改进的 HOG 特征：

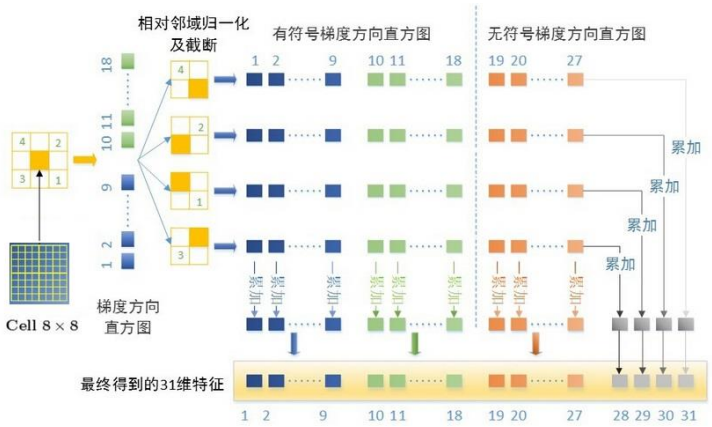

图 10 改进的 HOG 特征

在 HOG 的基础上去掉了 Block，考虑包含每个 Cell 的四个邻域，并进行归一化和截断。将特征分为对方向敏感的（360/2=18 维）和不敏感的（180/2=9 维），如果考虑 4 邻域则共有 4×(9+18)=108 维特征，显然计算量过大，并且特征信息有过多冗余。可与 HOG 特征提取一样，对这 4×27 维特征行、列累加，可得到 4+27=31 维特征。

使用高斯核，则

$$k^{xx'}=\exp\left(-\frac{1}{\sigma^2}\left(\|x\|^2+\|x'\|^2-2\mathscr{F}^{-1}\left(\sum_{i=1}^{31}\hat{x}_i^*\odot\hat{x}'_i\right)\right)\right)$$

4.2.6 快速检测

训练样本和测试样本的核相关矩阵为$K^z=C(k^{xz})$。k^{xz}是对称向量。

$$f(z) = K^z\alpha = C(k^{xz})\alpha = C\left(\mathcal{F}^{-1}\left(\hat{k}^{xz}\right)\right)\alpha$$
$$\mathcal{F}(f(z)) = \left(\hat{k}^{xz}\right)^* \odot \hat{\alpha} = \hat{k}^{xz} \odot \hat{\alpha}$$

由此得到：

$$\hat{f}(z) = \hat{k}^{xz} \odot \hat{\alpha}$$

即检测的响应通过向量的离散傅里叶变换和训练参数进行哈达玛积计算得到。然后找到响应最大的位置，就是目标框位置。

5. 程序流程

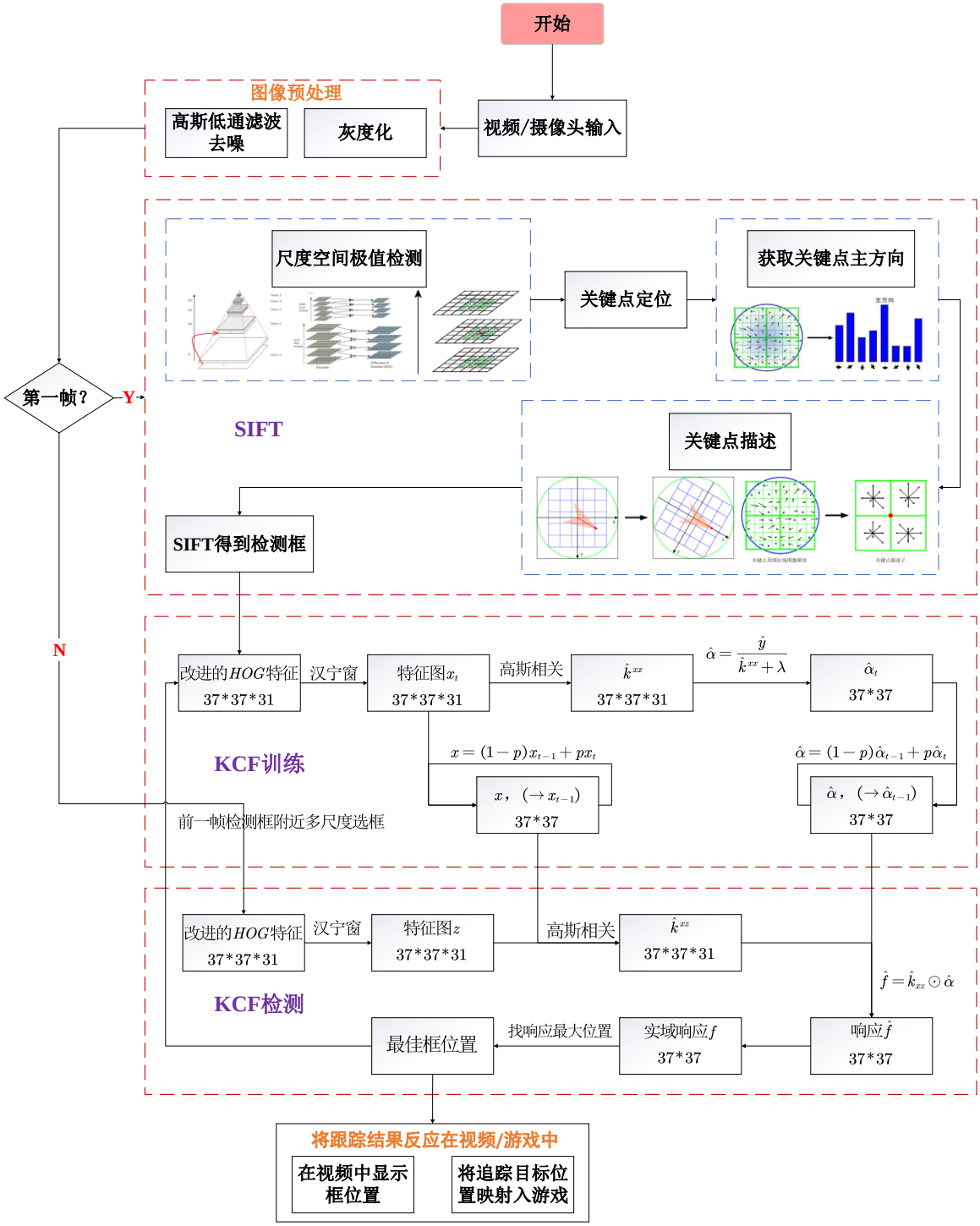

图 11 程序流程图

6. 程序使用说明

6.1 运行环境

- 操作系统：Windows、macOS、Linux 均可；
- 编程语言：Python；
- 需要的库：cv2、PyQt5、numpy、pygame、numba 等。

6.2 运行指南

1. 进入工程目录 cv_homework 下；
2. 在 IDE（如 PyCharm 等）中直接运行或在终端内输入命令 python main.py；
3. 在 GUI 中有两个按钮，左边是特定目标追踪，右边是人机交互游戏《外星人入侵》；
4. 点击左边按钮，进入特定目标跟踪界面，先打开需要跟踪的目标图像，再打开测试视频，等待视频加载完成后，可点击“开始检测并跟踪”，就开始检测匹配与跟踪目标，最后可点击“退出”进行退出；
5. 点击右边按钮，进入人机交互游戏《外星人入侵》配置窗口，选择一张有明显纹理特征的标识图像（如有纹理特征的手套等），玩家需在手上进行佩戴，准备完毕后，将标识对准摄像头，点击“开始配置”，直至游戏界面弹出，便可点击“Play”开始游戏。

7. 实验过程与结果

7.1 运行结果

图 12 欢迎界面

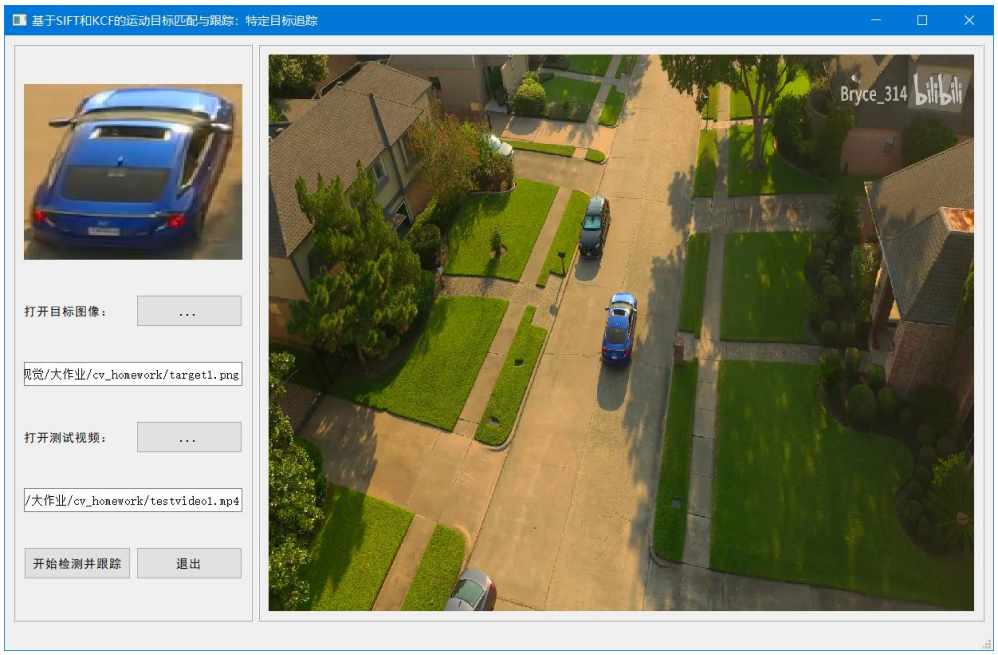

图 13 特定目标跟踪界面

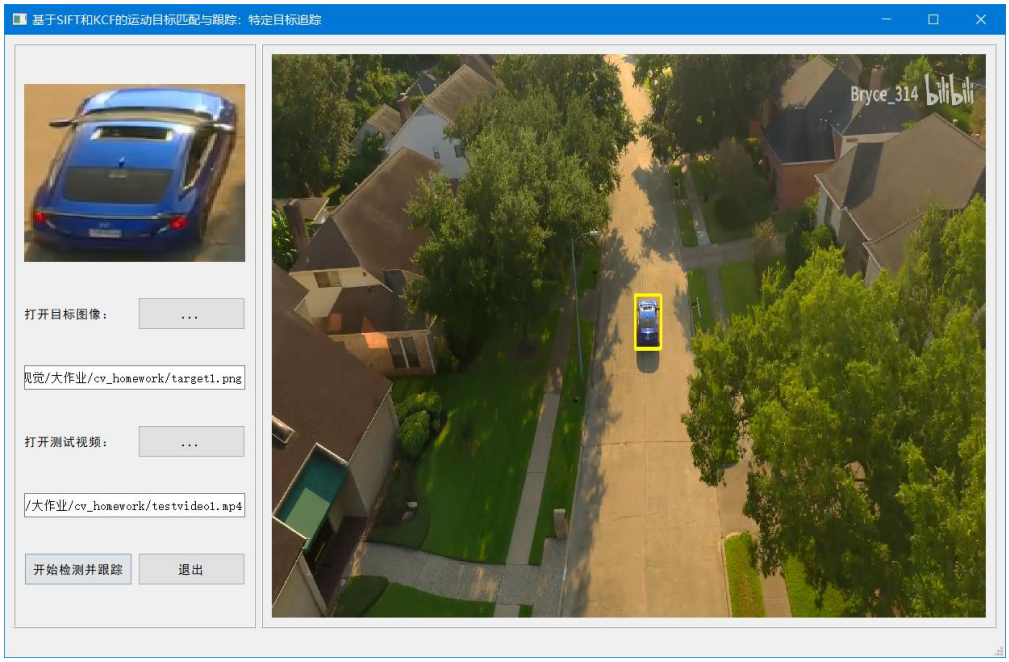

图 14 特定目标跟踪界面

图 15 人机交互游戏配置窗口

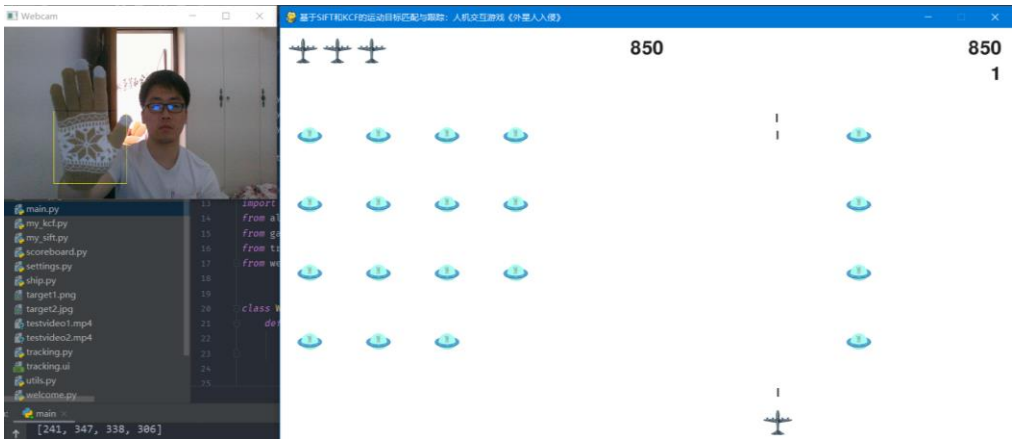

图 16 人机交互游戏《外星人入侵》界面

7.2 与 OpenCV 库对比

1. SIFT

图 17 和图 18 是自己实现的 SIFT，图 19 和图 20 是 OpenCV 的 SIFT。二者的特征点几乎一致，如果有区别也是浮点数的影响；自己实现的 SIFT 比 OpenCV 的要慢上好几倍甚至几十倍，推测的原因是，SIFT 算法本来计算量就大，实时性不好，再加上使用 Python 就更加慢了。而 OpenCV 是调用 C++的实现，相对来讲速度稍微能够接受一些。

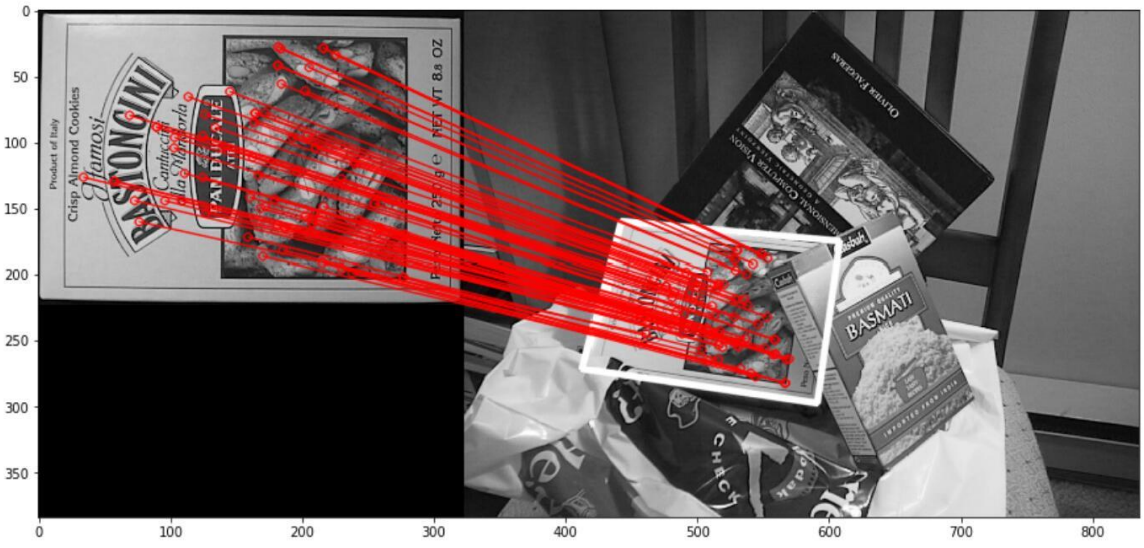

图 17 自己实现的 SIFT 算法

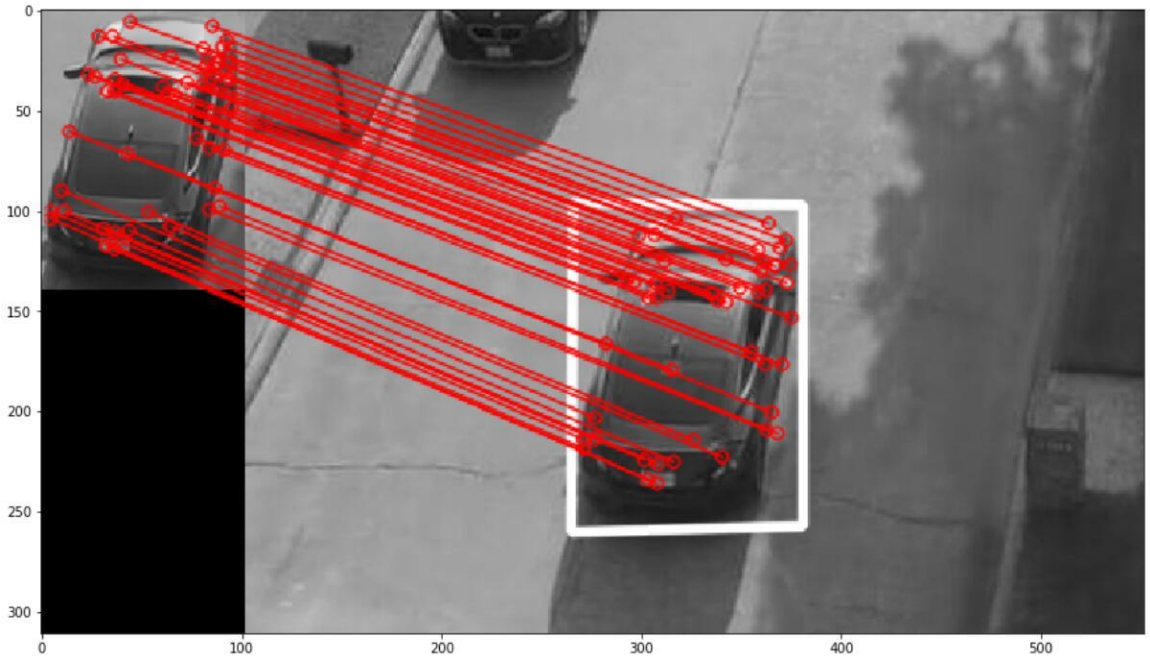

图 18 自己实现的 SIFT 算法

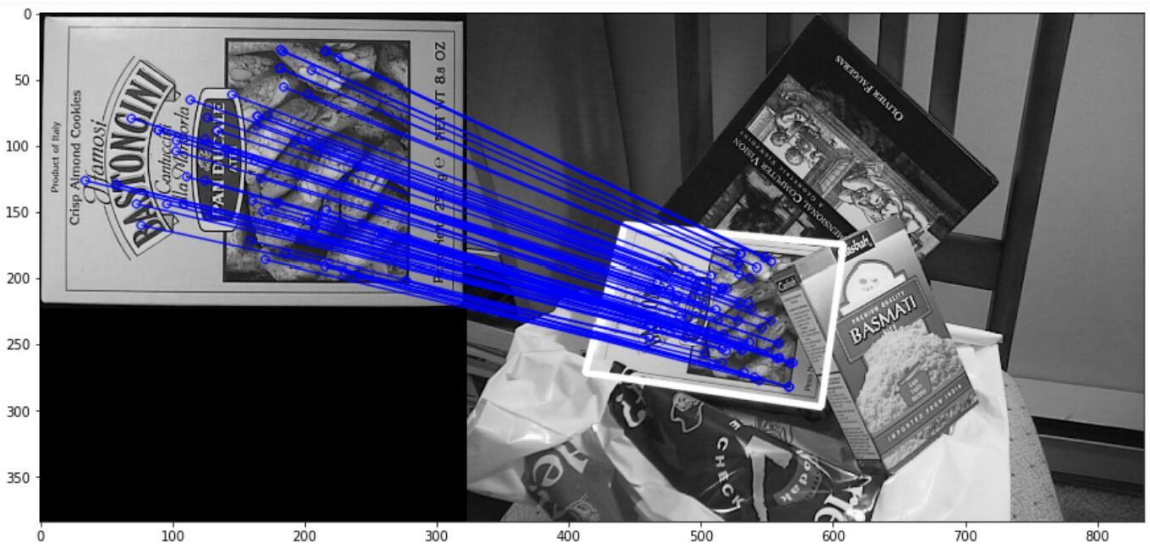

图 19 OpenCV 库的 SIFT 算法

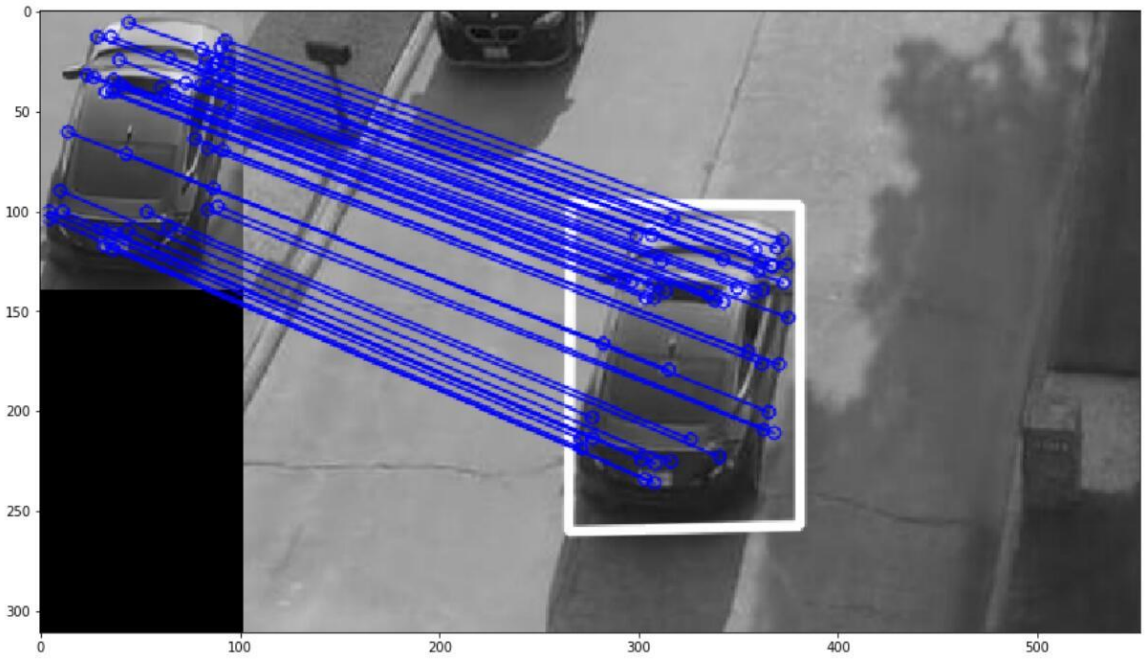

图 20 OpenCV 库的 SIFT 算法

2. KCF

图 21 和图 22 是我们自己实现的 KCF 算法的效果，而图 23 和图 24 是 OpenCV 库的 KCF 算法。可以看到，跟踪的初始阶段，效果差别不大，OpenCV 的 KCF 略优，而越到后面，随着误差积累越来越多，OpenCV 的 KCF 开始出现框脱离目标的情况，而我们自己实现的 KCF 则一直框住目标，即使目标姿态变化（如图中进行拐弯），也依然能够不脱离目标。

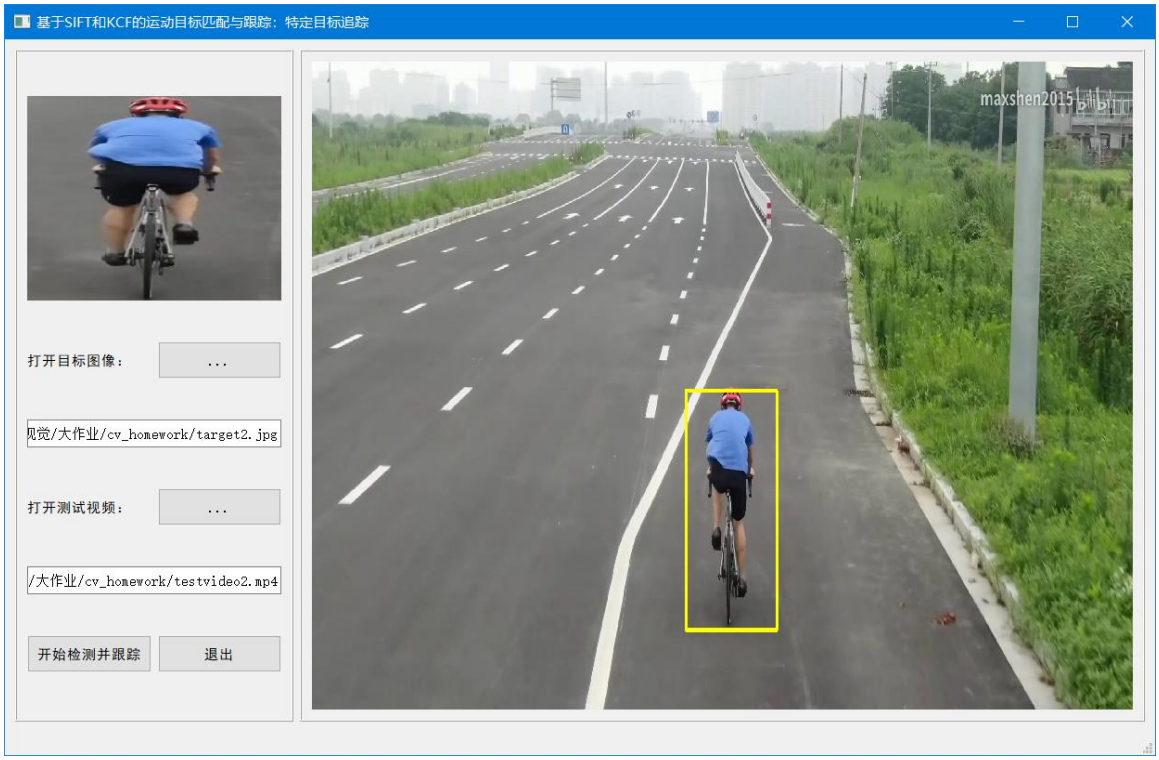

图 21 自己实现的 KCF 算法

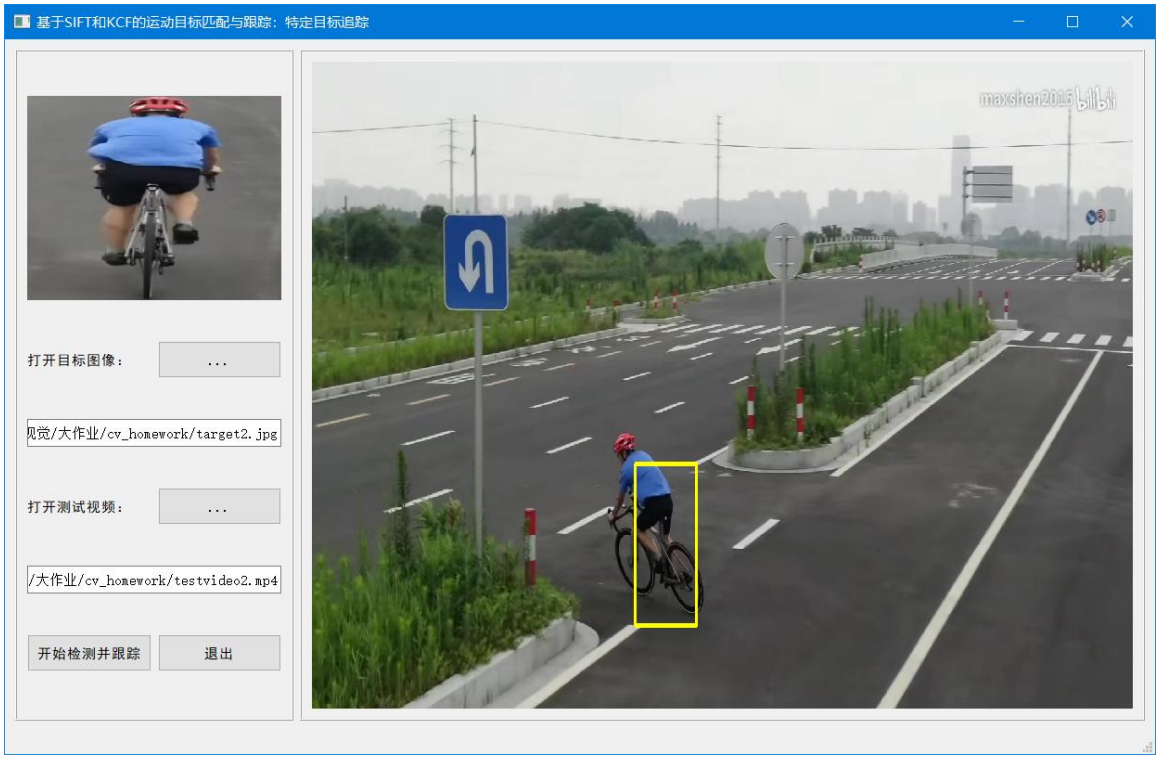

图 22 自己实现的 KCF 算法，目标姿态改变依然能框出

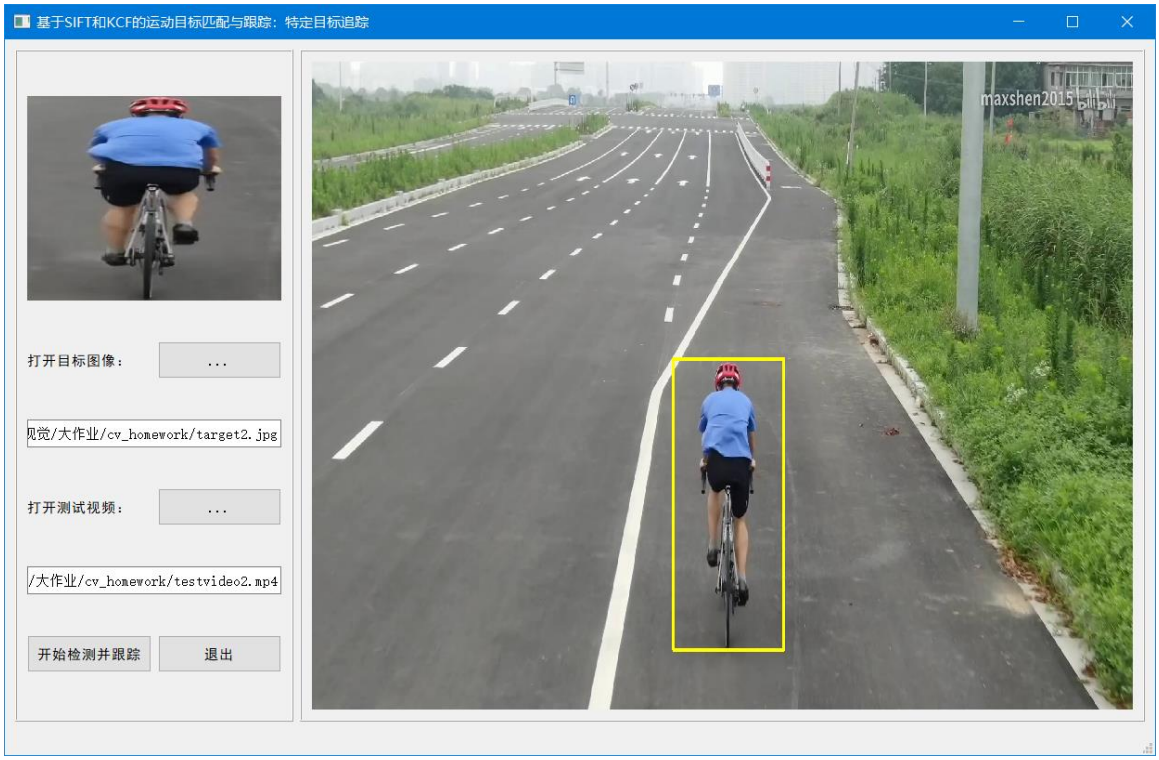

图 23 OpenCV 库的 KCF 算法，刚开始挺稳定

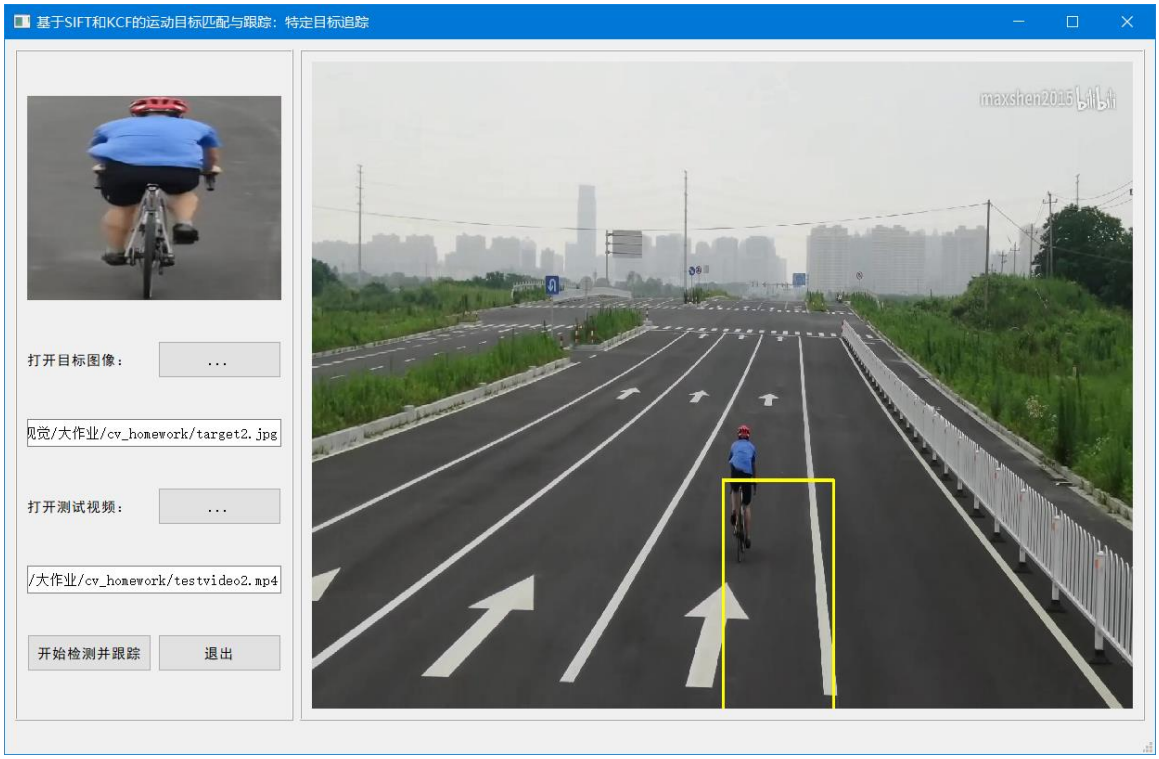

图 24 OpenCV 库的 KCF 算法，随着误差积累越来越多，到最后直接脱离目标

8. 模型优点与缺点

8.1 优点

- 我们从实际应用背景出发，设计的算法能够真正应用于现实，有很广泛的应用场合。
- 我们设计了合理的算法框架，完整地实现了预期任务，可以对给定视频中的特定目标进行跟踪，也开发了一款人机交互游戏。考虑了实时性、准确性、鲁棒性和便于操作性等因素，程序的运行效果良好。
- 我们自己实现了算法并与库函数进行了对比，除了 SIFT 的速度慢于库函数，其他方面效果相近，对于 KCF 效果甚至比库函数更理想。
- 我们的程序可以跟踪各种指定的目标，有很强的灵活性。只需一张图片指明目标即可，不需要花时间、也不需要大量训练样本训练分类器。
- 在算法方面 KCF 可以做到实时目标跟踪，精度和鲁棒性也很强，在性能极其普通的电脑上也能够有十分流畅的运行效果。
- 我们的算法可以真正做到跟踪某特定目标，一旦在第一帧中检测到目标，若在后面帧中出现同类或相似目标，算法不会被干扰。

8.2 缺点

- 算法对遮挡敏感，若遮挡严重会导致后面的跟踪全部错误。
- 缺乏纹理的目标难以被检测，不过这可以通过人为预先给定初始跟踪框来解决。
- SIFT 实时性差，进行检测匹配还是需要时间，因此可以感受到第一帧的小小停顿。也正因为此，我们假设第一帧一定要检测到目标，我们只对第一帧进行检测，后面只利用跟踪算法来保证程序的实时性。

9. 展望与改进空间

- SIFT 算法可以使用 ORB 算法替代，以提高实时性。
- 将单目标检测匹配与跟踪拓展为多目标检测匹配与跟踪。
- 提高检测算法的实时性后，就可以让其不只在第一帧进行，目的是让后续帧出现的目标也可以适用此框架。

10. 成员分工

- 李星毅：SIFT 实现（实现已完成，即 my_sift.py，但是由于自己实现的 SIFT 速度慢，为了不牺牲实时性，因此最后采用 OpenCV 提供的 SIFT 进行实验，速度能够接受）、实验报告、成员代码整理合并；
- 曾文正：KCF 实现、GUI 界面编写、实验报告、成员代码整理合并。