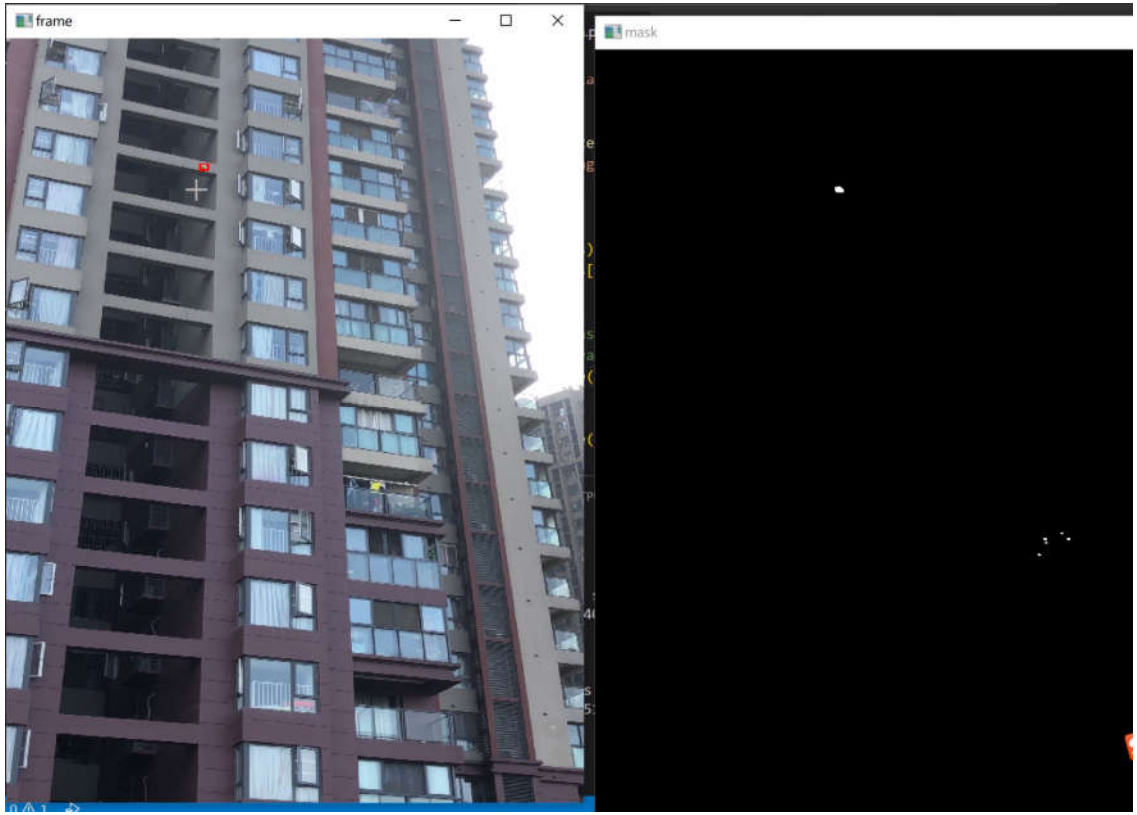
Python编程与人工智能实践



应用篇：高空坠物检测

于泓
鲁东大学
信息与电气工程学院
2023.1.22

整体模块

- 1、图像防抖
- 2、前景目标分割检测
- 3、运动目标跟踪（卡尔曼滤波）
- 4、系统融合（坠物视频录制）

图像防抖（视频数据预处理）

1、 选取一帧作为**基准帧**

2、 后继的**视频帧**与**基准帧，**
进行特征点匹配

3、 **根据匹配特征点**进行**仿射矩阵（M）**计算

4、 根据**仿射矩阵（M）**将视频帧向基准帧上进行映射

5、 裁边，得到最终的视频帧的输出

adjuster.py

```
import numpy as np
import cv2
import time


class Adjuster:
    def __init__(self, start_image, edge=(60, 20)):
        # determine if we are using OpenCV v3.X
        self.start_image = cv2.resize(start_image, (int(start_image.shape[1]/1), int(start_image.shape[0]/1)))
        self.edge = edge
        self.descriptor = cv2.ORB_create()
        self.matcher = cv2.DescriptorMatcher_create("BruteForce")
        (self.kps, self.features) = self.detectAndDescribe(self.start_image)
```

基准帧

裁边

特征点提取器

特征点匹配器

特征点

特征点
的描述子

对基准帧进行特征点检测

```python
def detectAndDescribe(self, image):
    # convert the image to grayscale
    if len(image.shape) > 2:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray = image

    # detect and extract features from the image

    (kps, features) = self.descriptor.detectAndCompute(image, None)

    # convert the keypoints from KeyPoint objects to NumPy
    # arrays
    kps = np.float32([kp.pt for kp in kps])

    # return a tuple of keypoints and features
    return (kps, features)
```

得到关键点
及其描述子

ORB全名为Oriented FAST and Rotated BRIEF，它采用改进的FAST关键点检测方法，使其具有方向性，并采用具有旋转不变性的BRIEF特征描述子。FAST和BRIEF都是非常快速的特征计算方法，因此ORB具有非同一般的性能优势。

OpenCV提取ORB特征并匹配 - 简书 (jianshu.com)

特征点匹配

```
def matchKeypoints(self, kpsA, kpsB, featuresA, featuresB,
                   ratio, reprojThresh):
    # compute the raw matches and initialize the list of actual
    # matches
    rawMatches = self.matcher.knnMatch(featuresA, featuresB, 2)
    matches = []

    # loop over the raw matches
    for m in rawMatches:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            matches.append((m[0].trainIdx, m[0].queryIdx))

    # computing a homography requires at least 4 matches
    if len(matches) > 4:
        # construct the two sets of points
        ptsA = np.float32([kpsA[i] for (_, i) in matches])
        ptsB = np.float32([kpsB[i] for (i, _) in matches])

        # compute the homography between the two sets of points
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
                                         reprojThresh)

        # return the matches along with the homograpy matrix
        # and status of each matched point
        return (matches, H, status)


    return None
```

利用KNN进行初步筛选

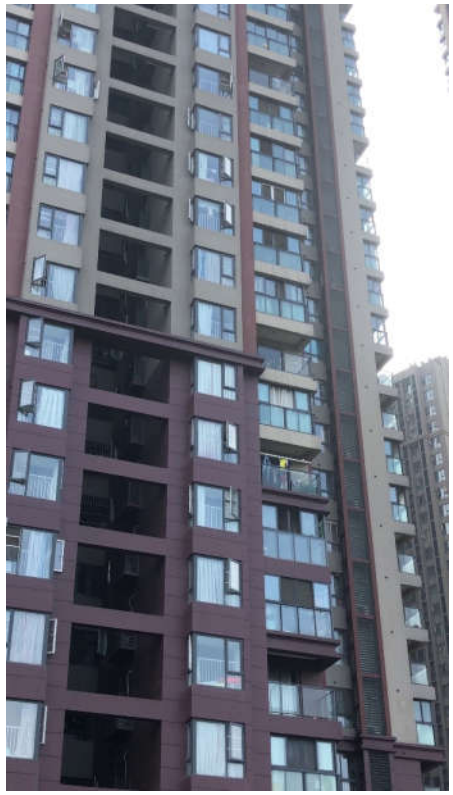
根据距离和方向，进一步筛选

利用RANSAC算法进行更加精细的筛选

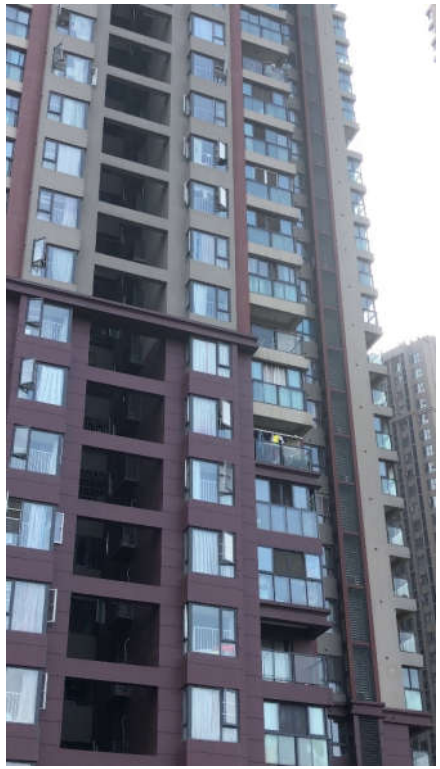
防抖主函数

```
def debouncing(self, image, ratio=0.7, reprojThresh=4.0, showMatches=False):
    image = cv2.resize(image, (int(image.shape[1]/1), int(image.shape[0]/1)))
    start = time.time()
    (kps, features) = self.detectAndDescribe(image)
    # print(f"take {time.time() - start} s")
    M = self.matchKeypoints(kps, self.kps, features, self.features, ratio, reprojThresh)

    if M is None:
        return None

    (matches, H, status) = M
    result = cv2.warpPerspective(image, H, (image.shape[1] + image.shape[1], image.shape[0] + image.shape[0]))

    result = result[int(self.edge[1]):int(image.shape[0] - self.edge[1]),
             int(self.edge[0]):int(image.shape[1] - self.edge[0])]


    return result
```

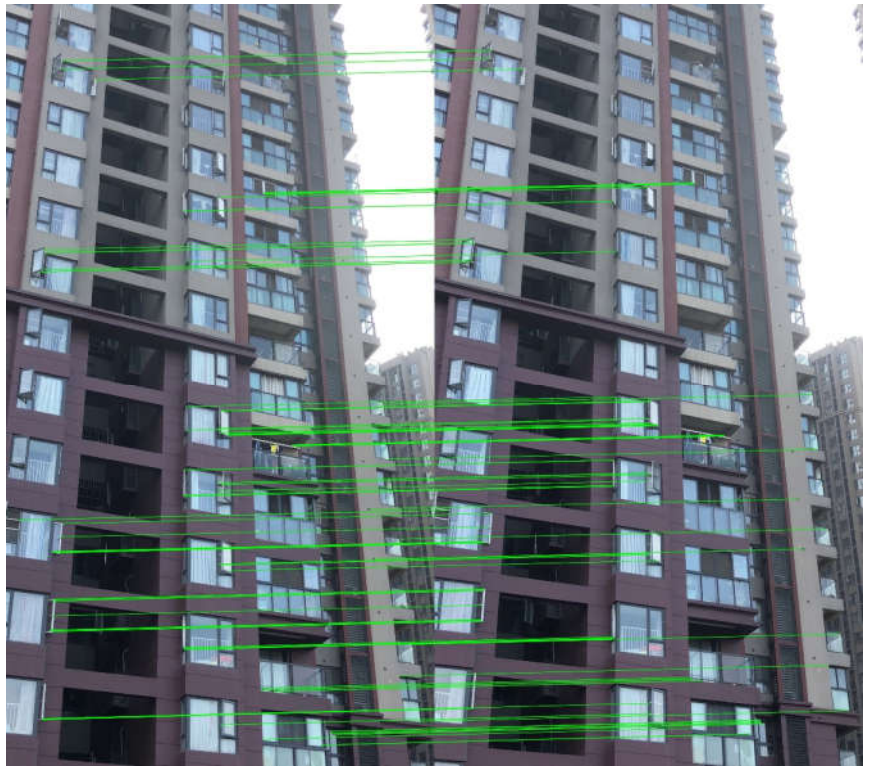
基准帧



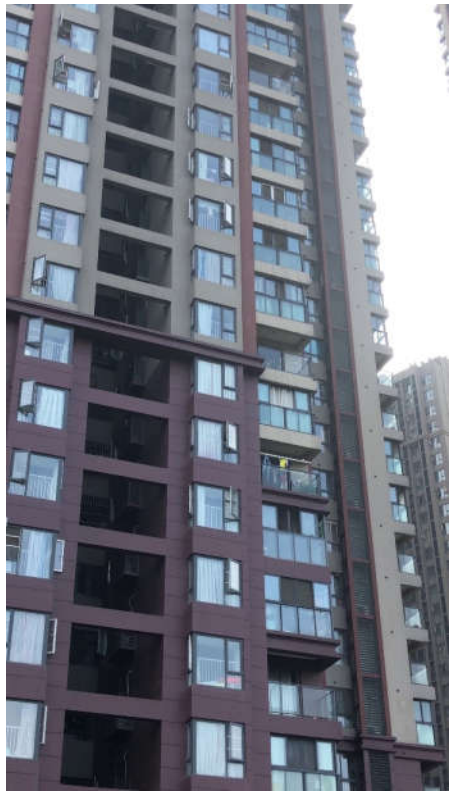
视频帧



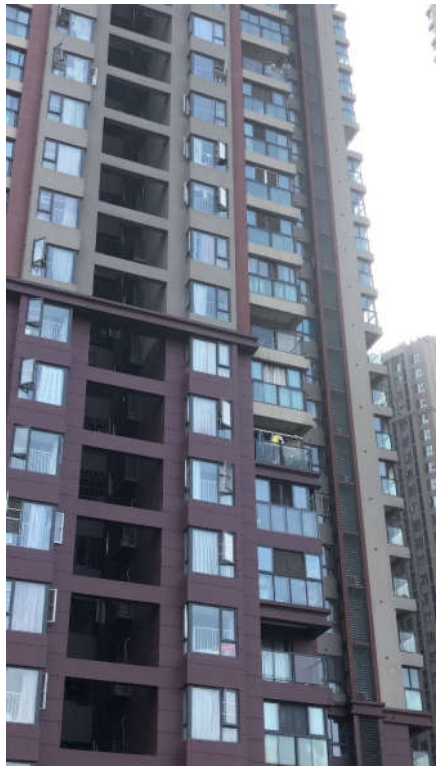
特征点匹配



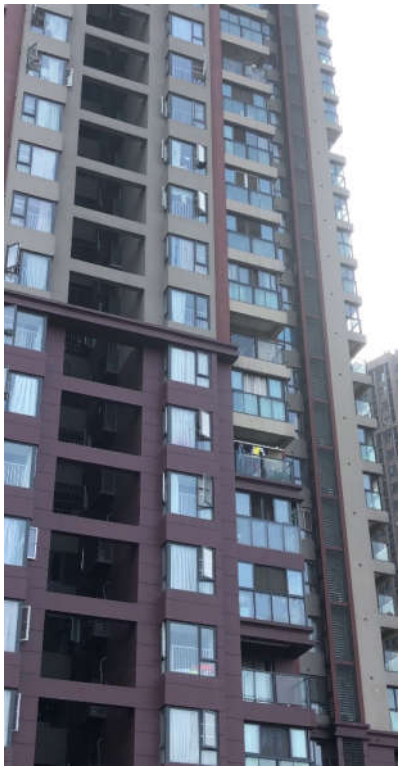
基准帧



视频帧



最终结果（裁边）



高空坠物检测（背景消除）

1 进行（背景消除）

2 进行形态学滤波

3 轮廓寻找

4 去除面积较小的轮廓

5 对较大的轮廓计算外接矩形

代码 knnDetector.py

```
import numpy as np
import cv2
import time

class knnDetector:
    def __init__(self, history, dist2Threshold, minArea):
        self.minArea = minArea
        self.detector = cv2.createBackgroundSubtractorKNN(history, dist2Threshold, False)
        self.kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
```

通过对背景像素进行建模来进行背景消除

背景建模所需要的的帧数

形态滤波的画笔

检测的阈值

```python
def detectOneFrame(self, frame):
    if frame is None:
        return None
    # 前景分离
    mask = self.detector.apply(frame)

    # 形态学滤波
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, self.kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_DILATE, self.kernel)

    # 轮廓检测
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    i = 0
    bboxs = []

    # 去掉面积过小的轮廓
    for c in contours:
        i += 1
        if cv2.contourArea(c) < self.minArea:
            continue
        # 获取外接矩形
        bboxs.append(cv2.boundingRect(c))

    return mask, bboxs
```

背景消除

开闭运算

返回值 x,y,w,h

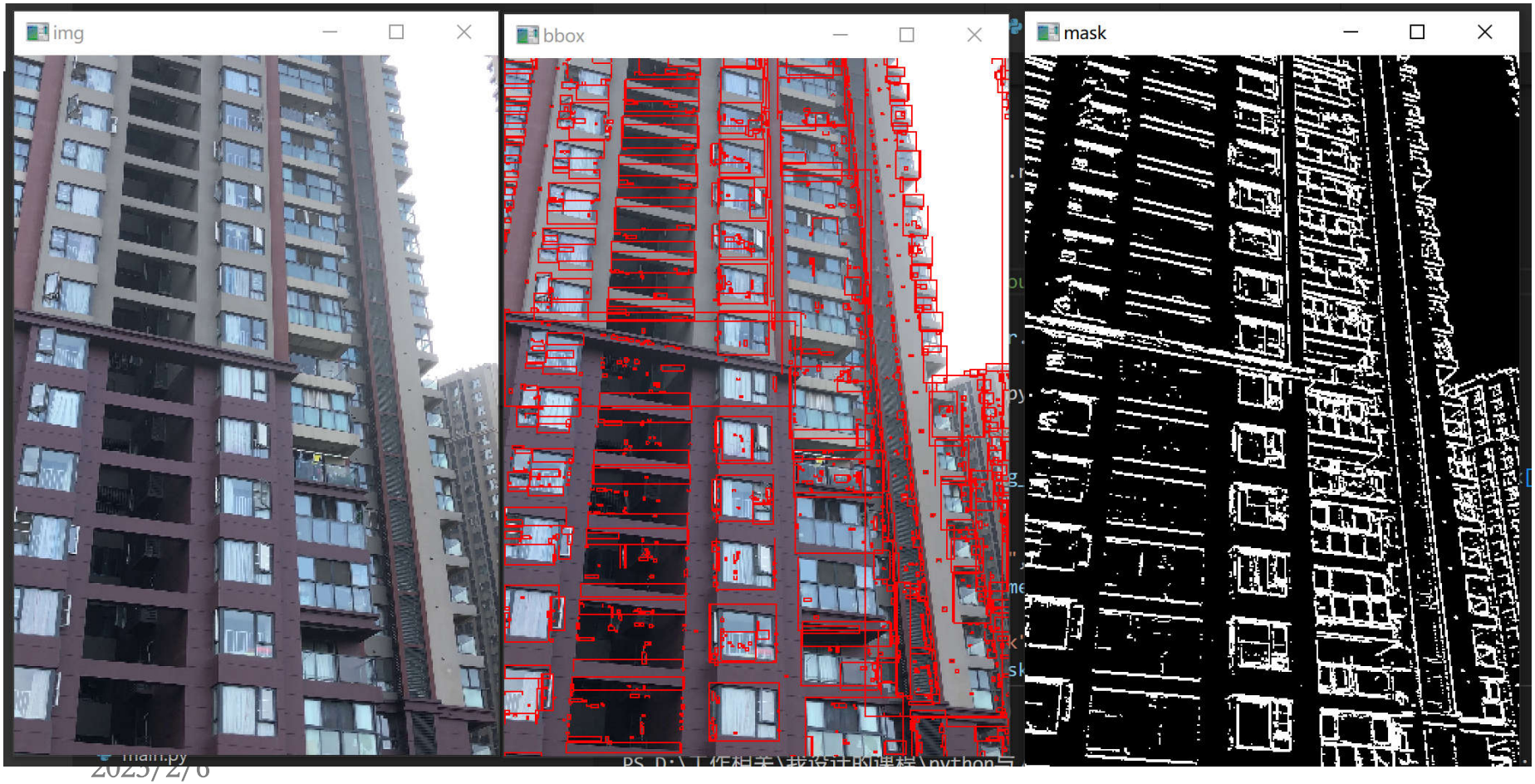
测试代码:

```
import cv2
from knnDetector import knnDetector
from adjuster import Adjuster

if __name__ == "__main__":


    history =500
    dist2Threshold = 400
    minArea =10

    detector = knnDetector(history, dist2Threshold, minArea)


    file_wav = "IMG_4550.MOV"
    capture = cv2.VideoCapture(file_wav)
    capture.set(cv2.CAP_PROP_POS_FRAMES, 200)
    fps = capture.get(cv2.CAP_PROP_FPS)

    # 加入防抖
    ret, frame = capture.read()
    adjust = Adjuster(frame, (120, 60))
```

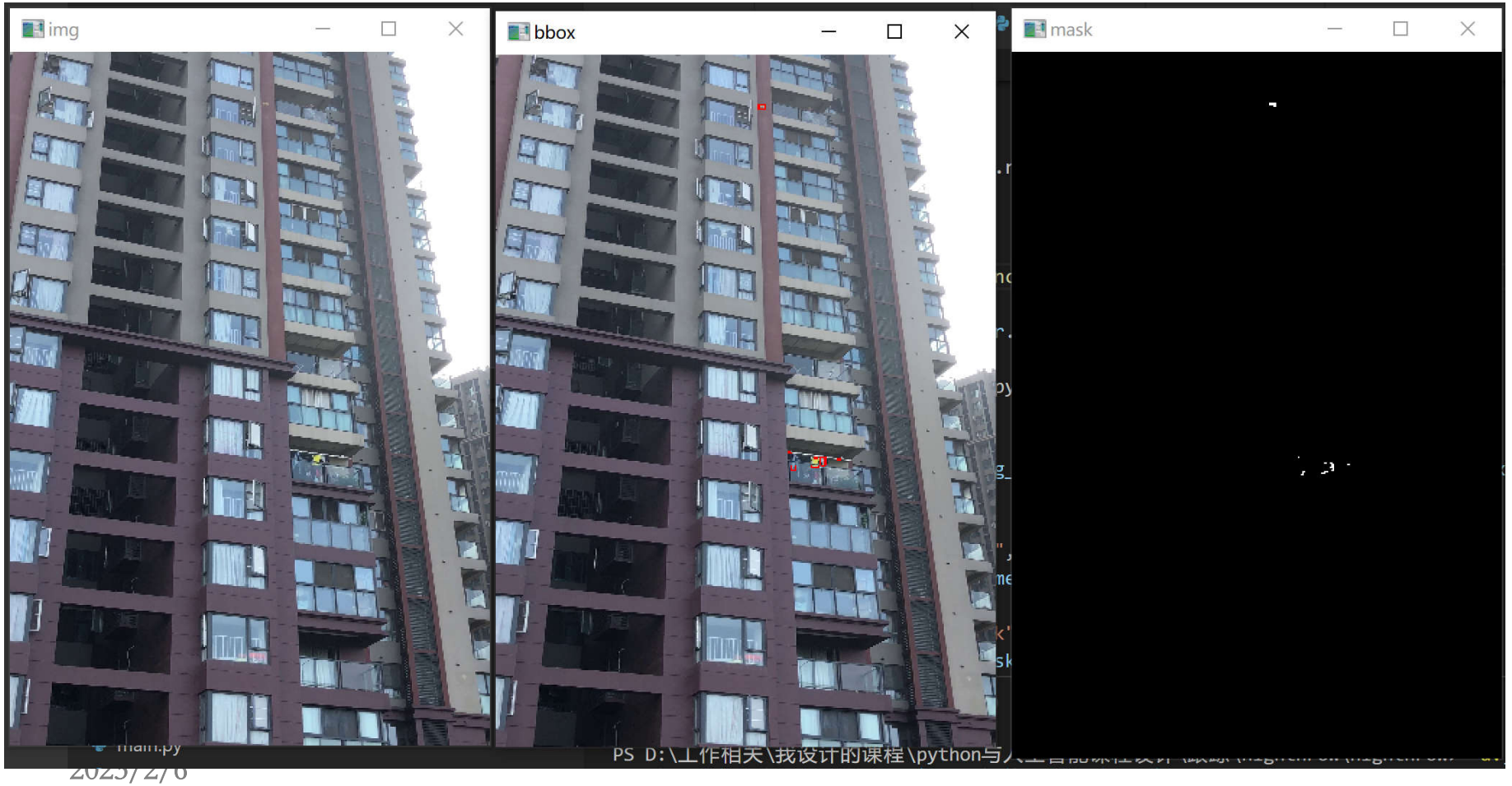
```
while True:

    ret, frame = capture.read()
    if frame is None:
        break

    frame = adjust.debouncing(frame)

    mask,bboxs = detector.detectOneFrame(frame)

    img_boxes = frame.copy()

    for box in bboxs:
        cv2.rectangle(img_boxes, (box[0], box[1]), (box[0]+box[2], box[1]+box[3]), (0, 0, 255), 6)

    # 显示
    cv2.namedWindow("img", cv2.WINDOW_NORMAL)
    cv2.imshow("img",frame)

    cv2.namedWindow("mask", cv2.WINDOW_NORMAL)
    cv2.imshow("mask",mask)

    cv2.namedWindow("bbox", cv2.WINDOW_NORMAL)
    cv2.imshow("bbox",img_boxes)

      # 按q退出
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

绘制矩形，左上，右下坐标

目标检测 没有防抖:



目标检测 加入防抖:



卡尔曼滤波跟踪过程

（1） 进行目标检测，获得**检测目标**

（2）对已存储的**跟踪目标**进行**预测**

利用卡尔曼滤波

（3）进行**检测目标**和**跟踪目标**的匹配

对能够和**检测目标**匹配的**跟踪目标，**利用**检测目标**更新**跟踪目标**

对没有匹配上的**检测目标，**设置为新的**跟踪目标**

（4）**跟踪目标** 每被跟踪上一次，跟踪计数+1

跟踪目标 没有被匹配，未匹配计数+1

（5）**跟踪计数较大**，且与**初始目标相比产生较大位移**，则进行输出

长时间未被匹配，跟踪目标删除

卡尔曼滤波跟踪

观测量 z_k ： x, y , s, r (中心点坐标，面积，宽高比)

预测量（状态量） ： x, y, s, r, vx, vy, vs, ax, ay

预测

$$\begin{cases} \hat{\mathbf{x}}_k = \mathbf{F}_k\hat{\mathbf{x}}_{k-1} + \mathbf{B}_k\vec{\mathbf{u}}_k \\ \mathbf{P}_k = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k \end{cases}$$

更新

$$\begin{cases} \mathbf{K}' = \mathbf{P}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \\ \hat{\mathbf{x}}_k' = \hat{\mathbf{x}}_k + \mathbf{K}'(\vec{\mathbf{z}_k} - \mathbf{H}_k\hat{\mathbf{x}}_k) \\ \mathbf{P}_k' = \mathbf{P}_k - \mathbf{K}'\mathbf{H}_k\mathbf{P}_k \end{cases}$$

需要预设的参数 F： 从z_k到 x_k的转换矩阵

H: 从x_k到 z_k的转换矩阵

R: z_k的协方差（变量不确定性）

P: z_k的协方差 (初始化)

Q: P的不确定性干扰

代码实现： kalmanFilter.py 构造了基本的卡尔曼滤波器，这里不做详细介绍

```
class KalmanBoxTracker(object):
    """
    This class represents the internal state of individual tracked objects observed as
    """
    count = 0

    def __init__(self, bbox):
        """
        Initialises a tracker using initial bounding box.
        """
        # define constant velocity model
        # 这里dim_x=9 分别为 x, y, s, r, vx, vy, vs, ax, ay (vs指的是面积变化的速率)
        # dim_z=4 分别为 x, y, s, r
        self.kf = KalmanFilter(dim_x=9, dim_z=4)
        self.kf.F = np.array(
            [[1, 0, 0, 0, 1, 0, 0, 0.5, 0],
             [0, 1, 0, 0, 0, 1, 0, 0, 0.5],
             [0, 0, 1, 0, 0, 0, 1, 0, 0],
             [0, 0, 0, 1, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 1, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 1, 0, 0, 1],
             [0, 0, 0, 0, 0, 0, 1, 0, 0],
             [0, 0, 0, 0, 0, 0, 0, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 0, 1]])
        self.kf.H = np.array(
            [[1, 0, 0, 0, 0, 0, 0, 0, 0],
             [0, 1, 0, 0, 0, 0, 0, 0, 0],
             [0, 0, 1, 0, 0, 0, 0, 0, 0],
             [0, 0, 0, 1, 0, 0, 0, 0, 0]])
```

构造一个跟踪器

跟踪目标的初始位置

)y 实现了滤波跟踪的具体步骤

$x = x_0 + vt + 0.5at^2$

F： 从z_k到 x_k的转换矩阵

H: 从x_k到 z_k的转换矩阵

```
self.kf.R[2:, 2:] *= 10.
self.kf.P[4:, 4:] *= 1000.  # give high
self.kf.P *= 10.
self.kf.Q[-1, -1] *= 0.01
self.kf.Q[4:, 4:] *= 0.01

self.kf.x[:4] = convert_bbox_to_z(bbox)
self.time_since_update = 0
self.id = KalmanBoxTracker.count
KalmanBoxTracker.count += 1
self.history = []
self.hits = 0
self.hit_streak = 0
self.age = 0
self.org_box = bbox.copy()
self.is_throw = False
```

R: z_k的协方差（变量不确定性）

P: z_k的协方差 (初始化)

Q: P的不确定性干扰

被跟踪计数

未被跟踪计数

初始位置

是否为下坠物体

```
def convert_bbox_to_z(bbox):
    """
    Takes a bounding box in the form [x1,y1,x2,y2]
      [x,y,s,r] where x,y is the centre of the box
      the aspect ratio
    """
    w = bbox[2] - bbox[0]
    h = bbox[3] - bbox[1]
    x = bbox[0] + w / 2.
    y = bbox[1] + h / 2.
    s = w * h  # scale is just area
    r = w / float(h)
    return np.array([x, y, s, r]).reshape((4, 1))
```

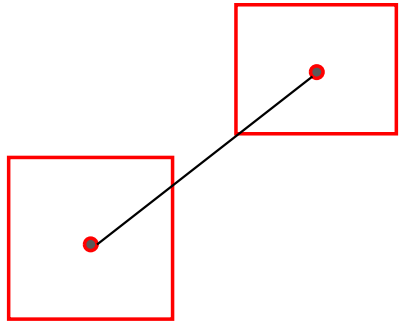
```
def update(self, bbox):
    """
    Updates the state vector with observed bbox.
    """
    self.time_since_update = 0
    self.history = []
    self.hits += 1
    self.hit_streak += 1
    self.kf.update(convert_bbox_to_z(bbox))
```

跟踪上，则调用 update函数

```
def predict(self):
    """
    Advances the state vector and returns the predicted bc
    """
    if ((self.kf.x[6] + self.kf.x[2]) <= 0):
        self.kf.x[6] *= 0.0
    self.kf.predict()
    self.age += 1
    if (self.time_since_update > 0):
        self.hit_streak = 0
    self.time_since_update += 1
    self.history.append(convert_x_to_bbox(self.kf.x))
    return self.history[-1]
```

没有跟踪上，调用 predict函数

```
def get_state(self):
    """
    Returns the current bounding box estimate.
    """
    bbox = convert_x_to_bbox(self.kf.x)[0]
    x = (bbox[0]+bbox[2])/2 - (self.org_box[0]+self.org_box[2])/2
    y = (bbox[1] + bbox[3]) / 2 - (self.org_box[1] + self.org_box[3]) / 2
    disdance = math.hypot(x, y)
    if disdance > 2 * (self.org_box[2]-self.org_box[0]+bbox[2]-bbox[0]) and \
        disdance > (self.org_box[3]-self.org_box[1]+bbox[3]-bbox[1]):
        self.is_throw = True

    return bbox, self.is_throw
```



移动距离足够大，判断下坠

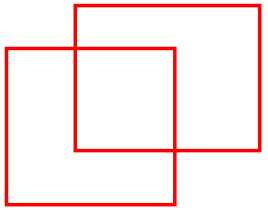
```
def convert_x_to_bbox(x, score=None):
    """
    Takes a bounding box in the centre form [x,y,s,r] and returns it in the form
      [x1,y1,x2,y2] where x1,y1 is the top left and x2,y2 is the bottom right
    """
    w = np.sqrt(x[2] * x[3])
    h = x[2] / w
    if (score == None):
        return np.array([x[0] - w / 2., x[1] - h / 2., x[0] + w / 2., x[1] + h / 2.]).reshape((1, 4))
    else:
        return np.array([x[0] - w / 2., x[1] - h / 2., x[0] + w / 2., x[1] + h / 2., score]).reshape((1, 5))
```

跟踪整体函数

未跟踪阈值

跟踪上阈值

检测目标与跟踪目标的匹配阈值



```
class Sort(object):
    def __init__(self, max_age=1, min_hits=3, iou_threshold=0.3):
        """
        Sets key parameters for SORT
        """
        self.max_age = max_age
        self.min_hits = min_hits
        self.iou_threshold = iou_threshold
        self.trackers = []
        self.frame_count = 0

  def update(self, dets=np.empty((0, 5))):

      self.frame_count += 1
      # get predicted locations from existing trackers.
      trks = np.zeros((len(self.trackers), 5))
      to_del = []
      ret = []
      # step1: predict
      for t, trk in enumerate(trks):
          pos = self.trackers[t].predict()[0]
          trk[:] = [pos[0], pos[1], pos[2], pos[3], 0]
          if np.any(np.isnan(pos)):
              to_del.append(t)
      trks = np.ma.compress_rows(np.ma.masked_invalid(trks))
      for t in reversed(to_del):
          self.trackers.pop(t)
```

对已跟踪的目标
先进性一次预测

```
# if detect or track failed
matched, unmatched_dets, unmatched_trks = associate_detections_to_trackers(dets, trks, self.iou_threshold)

# update matched trackers with assigned detections
for m in matched:
    self.trackers[m[1]].update(dets[m[0], :])
```

进行检测目标和跟踪目标的匹配

对已经匹配的跟踪目标进行更新

```
# create and initialise new trackers for unmatched detections
for i in unmatched_dets:
    trk = KalmanBoxTracker(dets[i, :])
    self.trackers.append(trk)
```

利用没有匹配的检测目标，构建新的跟踪目标

```
i = len(self.trackers)
for trk in reversed(self.trackers):
    bbox, is_throw = trk.get_state()
    if is_throw and (trk.time_since_update < 1) and (trk.hit_streak >= self.min_hits or self.frame_count <= self.min_hits):
        ret.append(np.concatenate((bbox, [trk.id + 1])).reshape(1, -1))  # +1 as MOT benchmark requires positive
    i -= 1
    # remove dead tracklet
    if (trk.time_since_update > self.max_age):
        self.trackers.pop(i)
if (len(ret) > 0):
    return np.concatenate(ret)
return np.empty((0, 5))
```

对跟踪器的状态进行筛选，判定为下坠且跟踪时间较长的，输出跟踪框

长时间没有跟踪上的跟踪对象删除

```python
def associate_detections_to_trackers(detections, trackers, iou_threshold=0.3):
    """
    Assigns detections to tracked object (both represented as bounding boxes)
    Returns 3 lists of matches, unmatched_detections and unmatched_trackers
    """
    if (len(trackers) == 0):
        return np.empty((0, 2), dtype=int), np.arange(len(detections)), np.empty((0, 5), dtype=int)

    iou_matrix = iou_batch(detections, trackers)
```

```python
def iou_batch(bb_test, bb_gt):
    """
    From SORT: Computes IUO between two bboxes in the form [l,t,w,h]
    每一行代表一个跟踪框，每一列代表一个检测框，那么每个坐标的意义就是 跟踪框y与检测框x的IOU
    """
    bb_gt = np.expand_dims(bb_gt, 0)
    bb_test = np.expand_dims(bb_test, 1)
    # 这里用到了 maximum 的广播属性，从 44*1 × 1*56 广播到 44*56
    xx1 = np.maximum(bb_test[..., 0], bb_gt[..., 0])
    yy1 = np.maximum(bb_test[..., 1], bb_gt[..., 1])
    xx2 = np.minimum(bb_test[..., 2], bb_gt[..., 2])
    yy2 = np.minimum(bb_test[..., 3], bb_gt[..., 3])
    w = np.maximum(0., xx2 - xx1)
    h = np.maximum(0., yy2 - yy1)
    wh = w * h
    o = wh / ((bb_test[..., 2] - bb_test[..., 0]) * (bb_test[..., 3] - bb_test[..., 1])
              + (bb_gt[..., 2] - bb_gt[..., 0]) * (bb_gt[..., 3] - bb_gt[..., 1]) - wh)
    return (o)
```

```
if min(iou_matrix.shape) > 0:
    # 将大于阈值的置为1，小于阈值的置为0
    a = (iou_matrix > iou_threshold).astype(np.int32)
    # 如果每个跟踪框只与一个检测框IOU大于阈值，每个检测框只与一个跟踪框IOU大于阈值，
    #则认为跟踪唯一，大于阈值的跟踪框都是正确的
    if a.sum(1).max() == 1 and a.sum(0).max() == 1:
        matched_indices = np.stack(np.where(a), axis=1)
    # 否则需要使用线性任务指派算法，将每个检测框与跟踪框以最小代价匹配起来
    #，这也是为什么 iou_matrix 需要乘以 -1 的原因
    else:
        matched_indices = linear_assignment(-iou_matrix)
else:
    matched_indices = np.empty(shape=(0, 2))

unmatched_detections = []
for d, det in enumerate(detections):
    if (d not in matched_indices[:, 0]):
        unmatched_detections.append(d)
unmatched_trackers = []
for t, trk in enumerate(trackers):
    if (t not in matched_indices[:, 1]):
        unmatched_trackers.append(t)

 matches = []
 for m in matched_indices:
     if (iou_matrix[m[0], m[1]] < iou_threshold):
         unmatched_detections.append(m[0])
         unmatched_trackers.append(m[1])
     else:
         matches.append(m.reshape(1, 2))
 if (len(matches) == 0):
     matches = np.empty((0, 2), dtype=int)
 else:
     matches = np.concatenate(matches, axis=0)

 return matches, np.array(unmatched_detections), np.array(unmatched_trackers)
```

进行匹配

差的也被匹配了

进一步筛选

```python
def linear_assignment(cost_matrix):
    try:
        import lap
        _, x, y = lap.lapjv(cost_matrix, extend_cost=True)
        return np.array([[y[i], i] for i in x if i >= 0])  #
    except ImportError:
        from scipy.optimize import linear_sum_assignment
        x, y = linear_sum_assignment(cost_matrix)
        return np.array(list(zip(x, y)))
```

线性任务分配（匈牙利匹配法）

$$C(i,j)=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} p & q & r & s \end{array} \\ \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix} \end{array}$$

Workers = {a,b,c,d}
Jobs = {p,q,r,s}

An arbitrary assignment
A = {(a,q),(b,s),(c,r),(d,p)}

Total cost = 23

lapjv算法是一种最佳任务分配方法，可以应用的地方很多。需要输入一个分数方阵，最终获得一列最佳分配数值。如n 个数值，要实现其最佳的配对，那么配对就需要根据n*n的一个分数方阵来计算，以总体最小代价实现任务分配，每一个数值不会重复分配

代码整合

```
import cv2
import numpy as np
from knnDetector import knnDetector
from sort import Sort
from adjuster import Adjuster
if __name__ == "__main__":

    path = "IMG_4550.MOV"
    capture = cv2.VideoCapture(path)

    # 构造检测器
    detector = knnDetector(history=500, dist2Threshold=400, minArea=10)

    cv2.namedWindow("frame", cv2.WINDOW_NORMAL)
    cv2.namedWindow("mask", cv2.WINDOW_NORMAL)

    # 需要预测成功5次才返回预测框，IOU最少0.1
    sort = Sort(max_age=3, min_hits=5, iou_threshold=0.1)

    ret, frame = capture.read()
    adjust = Adjuster(frame, (120, 60))
```

```
while True:
    ret, frame = capture.read()
    if frame is None:
        break
    # 防抖
    frame = adjust.debouncing(frame)

    # 目标检测
    mask, bboxs = detector.detectOneFrame(frame)

    # 检测到目标
    if bboxs != []:
        # x,y,w,h -> x1,y1,x2,y2
        bboxs = np.array(bboxs)
        bboxs[:, 2:4] += bboxs[:, 0:2]
        # 目标跟踪
        trackBox = sort.update(bboxs)
    else:
        # 没有检测到目标，直接预测
        trackBox = sort.update()

    # 绘制检测框
    for bbox in trackBox:
        bbox = [int(bbox[i]) for i in range(5)]
        cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (0, 0, 255), 6)
        cv2.putText(frame, str(bbox[4]), (bbox[0], bbox[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255))
    cv2.imshow("mask", mask)
    cv2.imshow("frame", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()
```

系统整合（坠物录制）

当检测到下坠物体时录像

设置**5个状态** state =0 空 state =1 开始录制 state =2 正在录制 state = 3 录制等待 state =4 结束录制

检测标志位 tracked=True 发现跟踪物体
检测物体消失后的时间 n_wait

在主循环内，构建状态机，实现5种状态之间的切换

根据当前状态，进行视频图像创建、视频帧写入、视频录制结束等一系列操作

代码实现

```
import cv2
import numpy as np
from knnDetector import knnDetector
from sort import Sort
import adjuster
from collections import deque
from datetime import datetime
import os
stat_dic= {0:'null',
           1: 'cap_start',
           2: 'caping',
           3:'cap_waite',
           4: 'cap_stop'
           }
# 视频保存
def save_video(datas,fps):
    # 图片保存
    currentDateAndTime = datetime.now()
    floder_name = currentDateAndTime.strftime("%Y-%m-%d")
    video_name = currentDateAndTime.strftime("%Y-%m-%d-%H-%M-%S")+".avi"
    os.makedirs(os.path.join('videos',floder_name),exist_ok=True)
    wav_save = os.path.join('videos',floder_name,video_name)
    fourcc = cv2.VideoWriter_fourcc(*'DIVX')  #XVID
    h,w,_ = datas[0].shape
    out = cv2.VideoWriter(wav_save, fourcc, fps, (w,h))
    for d in datas:
        out.write(d)
    return out
```

起始视频的缓存

主函数

```
def det_cap(path,b_adjust=False):
    capture = cv2.VideoCapture(path)
    capture.set(cv2.CAP_PROP_POS_FRAMES, 200)
    fps = capture.get(cv2.CAP_PROP_FPS)

    # 定义检测器
    detector = knnDetector(500, 400, 10)

    # 定义目标跟踪器
    sort = Sort(3, 5, 0.1)


    # 定义标志位
    flag = False

    # 读取一帧，作为背景模板
    ret, frame = capture.read()

    # 对图像进行预处理进行简单的裁剪
    adjust = adjuster.Adjuster(frame, (120, 60))
    state = 0
    state_pass = -1
    n_wait= 0
    n_frame =0
```

```
# 定义存储前端
catch_front= deque(maxlen=20)
video_save =[]
while True:
    tracked = False

    ret, frame = capture.read()
    if frame is None:
        break
    n_frame = n_frame + 1
    # 将当前帧和存储的背景帧进行对齐
    if b_adjust:
        frame = adjust.debouncing(frame)
    catch_front.append(frame)

    # 进行目标检测
    mask, bboxs = detector.detectOneFrame(frame)

    if bboxs != []:
        bboxs = np.array(bboxs)
        bboxs[:, 2:4] += bboxs[:, 0:2]

        trackBox = sort.update(bboxs)
    else:
        # test
        trackBox = sort.update()
```

```
# 有跟踪目标则进行绘图
for bbox in trackBox:

    bbox = [int(bbox[i]) for i in range(5)]
    cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (0, 0, 255), 6)
    cv2.putText(frame, str(bbox[4]), (bbox[0], bbox[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255))
    tracked = True
# 如果没有开始录制,但检测到跟踪目标则，开始进行录制
if state ==0 and tracked == True:
    state = 1
    n_wait =0
# 如果开始录制，检测到跟踪目标，则进入正在录制状态
elif state ==1 and  tracked == True:
    state = 2
    n_wait =0

# 正在录制状态下，没有跟踪到目标则进入等待状态
elif state ==2 and tracked == False:
    state =3
    n_wait = n_wait+1

# 等待状态下，且没有跟踪到目标，等待时间 +1
elif state ==3 and tracked == False:
    n_wait = n_wait+1
    # 如果没有跟踪上的时间过长则停止跟踪
    if n_wait >30:
        state = 4

# 如果等待状态下 又跟踪到目标，则进入正在录制状态
elif state ==3 and tracked == True:
    state =2
    n_wait =0

elif state == 4 and tracked == False:
    state =0
```

状态切换

设置**5个状态**

state =0 空

state =1 开始录制

state =2 正在录制

state = 3 录制等待

state =4 结束录制

```
if not state_pass == state:
    print("%d %s"%(n_frame,stat_dic[state]))

state_pass = state


if state == 1:
    video_save = [d for d in catch_front]
    out = save_video(video_save,fps)
elif state ==2 or state == 3:
    out.write(frame)
elif state ==4:
    print("保存视频")
    out.release()


cv2.imshow("frame", frame)

 # 按q退出
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

状态变化，进行状态打印

根据状态的不同
进行视频录制

```
if __name__ == "__main__":
    det_cap('IMG_4550.MOV',True)
```

主函数

视频输出