Final Project
Physics Based Tracker
CS 5363
Dr. Olac Fuentes

I. Problem Description

We focused on the problem of occlusion on a predictable trajectory for a moving object. Particularly, we present a case of a soccer ball that moves diagonally through the screen. At the middle of the screen there lies a copy of the same soccer ball which acts as our worst possible case occlusion. Also, we present another case where we track a roomba moving in a frame without any expected route.

The reason for choosing the soccer ball problem lies on the fact that we attempted to perform the tracking task on the soccer ball video with the help of the trackers implemented as part of the OpenCV library but failed at successfully tracking the moving soccer ball after it was occluded by the stading soccer ball. Additionally, we develop our tracker to be more robust to handle situations like that of the moving roomba without any predictable trajectory in order to have a more robust mechanism to track objects in videos.

The trackers tested were BOOSTING, KCF, CSRT, MIL, TLD, MedianFlow, and MOSSE. The version of OpenCV used was 4.0.1 and the version used for OpenCV-Contrib was 4.5.1.

II. Algorithms Implemented

The main idea we implemented to try to solve the problem of interest was combining an appearance model based on salient point similarity between the tracked object and the frame where the tracking is performed. In addition, we included a motion model based on a linear approximation given two initial points selected by the user. The program starts by asking for two bounding boxes from the same object at two different frame counts. Also, the program requires the user to input a polygon mask of the object tracked as seen in the first frame of the video to be used as the object that is being searched-for in the frame by the appearance model. The coordinates of the bounding boxes provided by the user were used to create a bi-linear regression model.

Appearance Model

This model was created using gaussian distributions for each match in salient points between the tracked object and the current frame of the video, with their intensity being correspondent to the inverse of the hamming distance between the points.

Diagram 1: Appearance distribution of the frame

Initially, we created a polygon mask of the tracked object using ImageDraw's polygon function given the user's input points (preferably 6 or more). We used the mask to isolate the image of the object so that the salient point detector didn't take into consideration any points from the background, as seen in the image below.

Diagram 2: Isolated tracked object

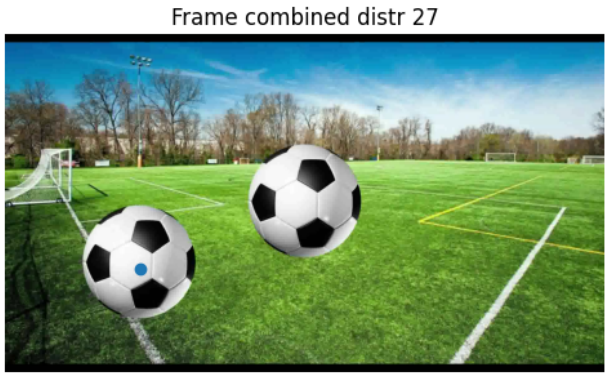
Through every frame in the video, we used the ORB (or SIFT) detector to find the salient points of the image, we used the brute-force matcher from OpenCV to find the matching points between the masked, first frame of the video and the frame that was analyzed. By using the gathered points and their corresponding hamming distance score we created the gaussian probability map for appearance, the map was composed of a sum of gaussian distributions with a size of 30% of the shorter edge of the image per every point. The “intensity”/importance of the match was scaled by applying a multiplication of the probability matrix built around the matching point times the inverse of the matching distance between the point of the masked-first-frame and then current frame. Using this approach, we prioritized the points with a smaller distance, as well as areas where many matches were found, in which we assumed the object is more likely to be located.

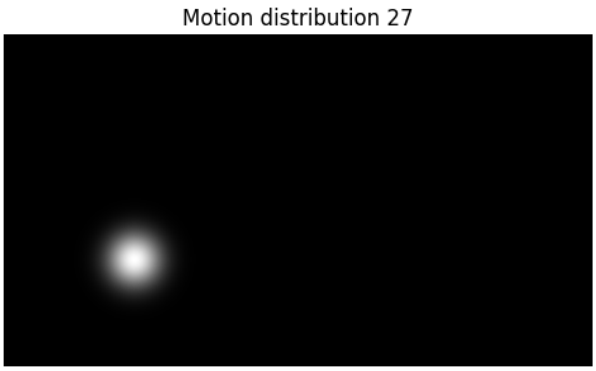
Motion Model

For the motion model we began with the assumption that the motion of the object could be modeled as an unchanged, linear trajectory. In order to model the movement of the object we asked the user for the input of the location of the object in two different moments of time. With those two locations at different moments of time we then proceeded to compute a bi-linear regression based on those two points. The first regression was in terms of the x-position in terms of time/frame. The second regression was in terms of the y-coordinate based on the calculated x-coordinate.

With the approximated x and y coordinates we assumed that such a point was the center of a gaussian distribution of motion probabilities within a predetermined size of the area in terms of percentage size of the image. Also, we radicalize the distribution of the gaussian by setting the sigma of the distribution to be > 30 in terms of magnitude.

Diagram 3: Motion distribution





In Diagram 3, the result of applying a sigma of magnitude 60 on an area of 60% of the smallest side of the image are shown.

Linear Combination

The linear combination of the models was based on a multiplication of the objects with some “probability noise” that expands throughout the entire models so that no point was totally discarded in case that any of the models is totally incorrect. The function has a parameter of noise per model, where we normalize the model probabilities as:

$$modelProbability = InitialProbability * (1 - noiseLevel) + noiseLevel$$

Then, we multiplied these newly gathered distributions to get our final combined model. This combination was more robust than the simple multiplication approach where once any model fails, it would invalidate the other as well.

Appearance Probability Enhancement

In order to better select the probabilities that are assigned based on the appearance model we decided to implement a scaling of the local gaussian distributions of probabilities based on the “intensity” or distances of the descriptors of the matches between every pair of points that ORB detects. The match that contains the least

distance receives a probability of 1. The rest of the matched points receive a score probability below 1 that “weighs” down the chances of selecting some of those areas as regions where the object was found.

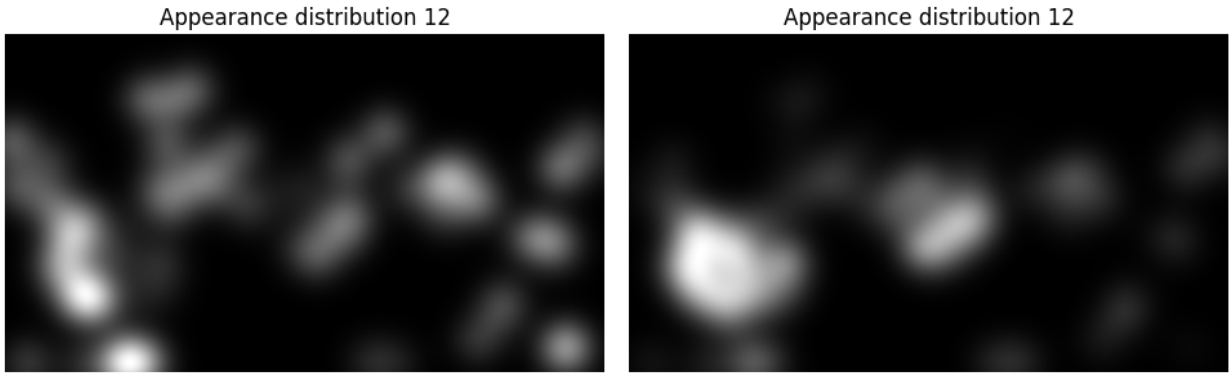


Diagram 4: Normal distribution of Probabilities without enhancements vs. enhanced distribution

Recall Enhancement

In an attempt to evolve the regression functions used for the motion model, we tried incorporating a mechanism where after *n* number of frames we would append a point into the lists (time, x_prediction, y_prediction) and recalculate said functions. After that, it would be reasonable to assign a lifetime for each point in the list by using the sliding windows method, in which we would enqueue a point to the list every *n* frame and dequeue it after the lifetime has expired. However, due to the variance in the appearance model gathered from the salient points, depending on the object, the appearance model might not be robust enough and would in turn confuse the motion model.

III. Testing Instructions

User Input Description

Initially, the user will be prompted to select two boundary boxes and a series of points pertaining to a polygonal boundary of the object:

- The first boundary box pertains to the position of the object at frame 0
- The second boundary box pertains to the position of the object at frame 5. Using the two boxes, we will compute the regression function for the motion model.
- The series of points will be used to compute a polygonal mask to isolate the tracked object from the background and avoid the salient point detector to identify them as relevant.

Function Call

The method has multiple parameters depending on the functionality desired by the user:

- vid_name: name of the video file to be processed
- bbox: coordinates corresponding to the first boundary box, if left blank it will prompt the user to generate new ones
- bbox2: coordinates corresponding to the second boundary box, if left blank it will prompt the user to generate new ones
- poly_points: list of points corresponding to the polygonal mask of the tracked object, if left blank it will prompt the user to generate new ones
- gauss_percentage: percentage of the shorter edge of the image to be used as a size for the gaussian distributions in the models
- write_video: boolean value that signals if the video will be stored
- motion_diff: percentage of certainty for the motion model
- appearance_diff: percentage of certainty for the appearance model
- display_lines: generates images pertaining to the matches in the detected salient points of the frame and the tracked object initial image
- display_motion: displays the images pertaining to the motion probability model
- display_appearance: displays the images pertaining to the appearance probability model
- display_combined: displays the images pertaining to the combined probability model.

- display_tracked: displays the images of the frames with a blue point signaling the where the tracked object is found
- limit: amount of frames to be computed, -1 signals the complete video.
- detector: has two choices, either ‘orb’ or ‘sift’ pertaining to the salient point detectors with the same name
- ransac: option to enable RANSAC, however this will not work when we can expect two similar objects in the same video.
- use_weighted_apperance: enables the weighted appearance enhancement

IV. Experimental Results & Discussion

Our results were somewhat positive, we indeed solved the problem of tracking the object even when it is occluded by a secondary, similarly looking soccer ball. The proof of this can be found in the video soccer_ball.mp4. But as a summary, the tracker gets slightly confused when the two soccer balls are aligned. As the soccer ball with a velocity continues moving diagonally through the frame our tracker predicts points that are close to the edge where the two balls are overlapping. This confusion emanates from the gaussian used to distribute probabilities of the probabilities on the pairs of points.

We have concluded that our methodology solves a simple tracking problem that involves a hard-to-recover-from occlusion obstacle. On the other tracking scenarios, like that of a Roomba moving around in a room with random changes of direction could was not always tracked satisfactorily with the proposed tracker implementation. To observe the results of tracking the Roomba please watch the roomba.mp4 video. As seen in roomba.mp4 it can be seen that the Roomba is indeed tracked mostly because the apperance model has successful matches, not because the motion model does a

successful prediction. This means that the appearance model also is “powerful” enough to correct the predictions of the model on its own. Which represents the capacity of the model to be robust against individual fails of either the appearance and motion.

Having two different models which can act as the source of correct predictions allowed to create a simple yet powerful tracker. Our approach is, of course, not perfect but provides a moderately robust scheme for tracking moving targets on videos on an offline prediction setup.

V. Conclusion

We conclude that our scheme is successful at tracking different objects on different tracking environments. We also showed that our problem of interest that could not be tracked by the trackers in OpenCV can be solved by applying the tracking techniques that we define in this work. Overall, our project was successful to this limited scope.

VI. Future Steps

Even though most modern trackers use deep learning models for motion and appearance, if someone were to develop further on the approach we took, we consider there are some areas of improvement such as:

- Using segmentation with the second boundary box to avoid asking the user for a polygonal mask.

- Automatic tuning of parameters of the detectors to avoid irrelevant points depending on the type of video being fed.

- Stabilizing the appearance model prediction so that the sliding windows approach can be more effective and properly update the regression functions.

We can appreciate that most of these shortcomings come from the initial idea of using salient point detectors such as ORB, SURF and SIFT.

VII. Appendix

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import sys
from scipy import signal
from scipy.interpolate import interp1d as ip1d
from sklearn.linear_model import LinearRegression
from PIL import Image, ImageDraw
from matplotlib.figure import Figure
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas

def region_points(shape,reg_rows,reg_cols, points, distances):
    centsums = np.zeros((shape[0]+reg_rows, shape[1]+reg_cols))
    magnitudes = (-distances + np.amax(distances) + 1)

    off_top = np.ceil(reg_rows/2).astype(int)
    off_bottom = np.floor(reg_rows/2).astype(int)
    off_left = np.ceil(reg_cols/2).astype(int)
    off_right = np.floor(reg_cols/2).astype(int)

    points = points.astype(int)
    for i in range(len(points)):
        centsums[off_top+points[i,1], off_left+points[i,0]] = magnitudes[i]
```

```
    centsums = np.cumsum(np.cumsum(centsums,axis=0),axis=1)

    for i in range(centsums.shape[0]-off_bottom, centsums.shape[0]):
        centsums[i] = centsums[-(off_bottom+1)]
    for i in range(centsums.shape[1]-off_right, centsums.shape[1]):
        centsums[:,i] = centsums[:,-(off_right+1)]

    point_probs = centsums[reg_rows:,reg_cols:] - centsums[reg_rows:,:-reg_cols] - centsums[:-
reg_rows,reg_cols:] + centsums[:-reg_rows,:-reg_cols]
    return point_probs/np.amax(point_probs)


def distance(p0,p1):
    return np.sqrt(np.power(p0[0]-p1[0],2) + np.power(p0[1]-p1[1],2))

def loop(points):
    while True:
        p = np.asarray(plt.ginput(n=2), dtype=int)
        if distance(p[0],p[1]) < 5:
            break
        points.append(p)
        plt.plot(p[:,0],p[:,1],marker = '.',color='cyan')
        plt.pause(0.01)
        plt.show()

def gaussian_filter(size, sigma):
    d = ((np.arange(size) - (size-1)/2)**2).reshape(-1,1)
    f = np.exp(-(d+d.T)/2/sigma/sigma)
    return f/np.sum(f)

def correlate_scipy(image, filt):
    r,c = filt.shape
    if(len(image.shape) == 2):
        return signal.correlate2d(image, filt,mode='same')
    else:
        new_image = np.zeros((image.shape[0], image.shape[1], 3))
        for i in range(3):
            new_image[:,:,i] = signal.correlate2d(image[:,:,i], filt,mode='same')
        return new_image

def show_image(image,title='',save_im=False,filename=None):
    # Display image in new window
    fig, ax = plt.subplots()
    ax.imshow(image,cmap='gray')
    ax.axis('off')
    ax.set_title(title)
```

```
    if save_im:
        if filename==None:
            filename=title
        fig.savefig(filename+'.jpg',bbox_inches='tight', pad_inches=0.1)
    return fig, ax

def approximate(x,y):
    return ip1d(x,y)

def select_matches_ransac(pts0, pts1, dist):
    H, mask = cv2.findHomography(pts0.reshape(-1,1,2), pts1.reshape(-1,1,2), cv2.RANSAC,10.0)
    choice = np.where(mask.reshape(-1) ==1)[0]
    return pts0[choice], pts1[choice], dist[choice]

def get_total_gauss(shape, c, r, filt=np.array([]),percentage=20, sigma=30):
    total_filter = np.zeros(shape[:2])
    if not filt.any():
        percentage = percentage
        filt = gaussian_filter(min(shape[0], shape[1])//(100/percentage), sigma)

    m_targ = [0, shape[0], 0, shape[1]]
    m_offs = [r-np.floor(filt.shape[0]/2).astype(int), r+np.ceil(filt.shape[0]/2).astype(int),
            c-np.floor(filt.shape[1]/2).astype(int), c+np.ceil(filt.shape[1]/2).astype(int)]
    diffs = [-min(0, m_offs[0]), -min(0, m_targ[1]-m_offs[1]), -min(0, m_offs[2]), -min(0, m_targ[3]-
m_offs[3])]
    filt = filt[diffs[0]:filt.shape[0]-diffs[1], diffs[2]:filt.shape[1]-diffs[3]]

    total_filter[m_offs[0]+diffs[0]: m_offs[1]-diffs[1], m_offs[2]+diffs[2]: m_offs[3]-diffs[3]] = filt
    total_filter = total_filter/np.amax(total_filter)
    return total_filter

def display_control_lines(im0,im1,pts0,pts1,point,clr_str = 'rgbycmwk',title='A'):
    canvas_shape = (max(im0.shape[0],im1.shape[0]),im0.shape[1]+im1.shape[1],3)
    canvas = np.zeros(canvas_shape,dtype=type(im0[0,0,0]))
    canvas[:im0.shape[0],:im0.shape[1]] = im0
    canvas[:im1.shape[0],im0.shape[1]:canvas.shape[1]]= im1
    fig, ax = plt.subplots(figsize=(8, 4))
    fig.suptitle(title)
    ax.imshow(canvas)
    ax.axis('off')
    pts2 = pts1+np.array([im0.shape[1],0])
    for i in range(pts0.shape[0]):
        ax.plot([pts0[i,0],pts2[i,0]],[pts0[i,1],pts2[i,1]],color=clr_str[i%len(clr_str)],linewidth=1.0)
    fig.suptitle('Point correpondences', fontsize=16)
    plt.scatter(point[0],point[1])
```

```
def get_descriptors(frame, mode='orb'):
    if mode == 'sift':
        sift = cv2.SIFT_create()
        keypoints, descriptors = sift.detectAndCompute(cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY),None)
    else:
        orb = cv2.ORB_create()
        keypoints, descriptors = orb.detectAndCompute(frame, mask=None)
    return keypoints, descriptors

def match_descriptors(keypoints1, descriptors1, keypoints2, descriptors2, ransac=False):
    matcher = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
    matches = matcher.match(descriptors1,descriptors2)
    # matches[x].queryIdx is src, trainIdx is dest

    dist = np.array([m.distance for m in matches])
    ds = np.argsort(dist)
    ind1 = np.array([m.queryIdx for m in matches])
    ind2 = np.array([m.trainIdx for m in matches])
    keypoints1 = np.array([p.pt for p in keypoints1])
    keypoints2 = np.array([p.pt for p in keypoints2])
    keypoints1 = keypoints1[ind1]
    keypoints2 = keypoints2[ind2]

    if ransac:
        keypoints1, keypoints2, dist = select_matches_ransac(keypoints1, keypoints2, dist)
    return keypoints1, keypoints2, ds, dist

def txy_regressions(t_pred, x_pred, y_pred):
    f_y = LinearRegression().fit(x_pred.reshape(-1, 1),y_pred) #Approximation f(x) = y
    f_x = LinearRegression().fit(t_pred.reshape(-1, 1),x_pred) #Approximation f(t) = x
    return f_x, f_y

def get_gaussian_ditribution_frame(frame, keypoints, filt=None, percentage=30, sigma=30,
distances=[]):
    if not filt:
        filt = gaussian_filter(min(frame.shape[0], frame.shape[1])//(100/percentage), sigma)
    keypoints = keypoints.astype(int)
    distances = (-distances + np.amax(distances) + 1)
    if len(distances) > 0:
        filters = [get_total_gauss(frame.shape, keypoints[i,0], keypoints[i,1],
                                   filt) * distances[i] for i in range(len(keypoints))]
    else:
        filters = [get_total_gauss(frame.shape, k[0], k[1], filt) for k in keypoints]
    distribution = np.sum(filters, axis=0)
    distribution = distribution/np.amax(distribution)
```

```
    return distribution

def track(vid_name, bbox=np.array([]), bbox2=np.array([]), poly_points=[],
          gauss_percentage=20, write_video=False, motion_diff=0.95, appearance_diff=0.95,
          display_lines=False, display_motion=False, display_appearance=False,
          display_combined=False, display_tracked=True, limit=-1, detector='orb',
          ransac=False, use_weighted_apperance=False):

    path = './videos/'
    video = cv2.VideoCapture(path+vid_name)
    fps = video.get(cv2.CAP_PROP_FPS)
    print("frames per second: " + str(fps))

    # Exit if video not opened.
    if not video.isOpened():
        print("Could not open video")
        sys.exit()

    # Read first frame.
    ok, frame = video.read()
    if not ok:
        print('Cannot read video file')
        sys.exit()

    if write_video:
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter('roomba.mp4',fourcc, 5.0, (600,400))

    if len(bbox) == 0:
        # Define an initial bounding box
        bbox = cv2.selectROI(frame, False)
        print("boundary box1: "+ str(bbox))

    delta_frames = 5
    for i in range(delta_frames):
        ok, frame = video.read()

    if len(bbox2) == 0:
        # GET BOUNDARY BOX AT SECOND MOMENT
        bbox2 = cv2.selectROI(frame, False)
        print("boundary box2: "+ str(bbox2))

    prediction_times = [0,delta_frames]
    prediction_points_x = [bbox[0]+bbox[2]/2,bbox2[0]+bbox2[2]/2,]
    prediction_points_y = [bbox[1]+bbox[3]/2,bbox2[1]+bbox2[3]/2,]
```

```
    # Initial approximation function with center of bboxes
    t_pred = np.array(prediction_times) # moments of time
    x_pred = np.array(prediction_points_x)
    y_pred = np.array(prediction_points_y)
    # print(x_pred,y_pred)
    f_x, f_y = txy_regressions(t_pred, x_pred, y_pred)

    # box_frame = frame[bbox2[1]:bbox[1]+bbox[3], bbox2[0]:bbox[0]+bbox[2]]
    # show_image(box_frame, title='box_frame')

    # _,frame = video.read()
    # x_pred = f_x.predict(np.array([10]).reshape(-1, 1)).astype(int)
    # y_pred = f_y.predict(x_pred.reshape(-1, 1)).astype(int)
    # print(f"New center C:{x_pred} R:{y_pred}")

    if len(poly_points) == 0:
        plt.imshow(frame[:,:,::-1])
        plt.show()
        p = []
        loop(p)

        poly_points = []
        for pair in p:
            poly_points.append(tuple(pair[0]))
            poly_points.append(tuple(pair[1]))
        print(poly_points)

    # Get the isolated source object from the polygon
    img = Image.new('L', (frame.shape[1],frame.shape[0]), 0)
    ImageDraw.Draw(img).polygon(poly_points, outline=1, fill=1)
    mask = np.array(img)
    masked_rgb = np.zeros(frame.shape)
    masked_rgb[:,:,0],masked_rgb[:,:,1],masked_rgb[:,:,2] =
frame[:,:,0]*mask,frame[:,:,1]*mask,frame[:,:,2]*mask
    print('MaskedRGB Shape: '+str(masked_rgb.shape))


    # descriptors of the trackedobject
    kpboxframe, desboxframe = get_descriptors(masked_rgb.astype(np.uint8), mode=detector)
    sel = 10

    ok, frame = video.read()
    count = delta_frames
    while(ok):

        x_pred = f_x.predict(np.array([count]).reshape(-1, 1))
```

```
        y_pred = f_y.predict(x_pred.reshape(-1, 1))

        print(count,x_pred,y_pred)

        kpframe, desframe = get_descriptors(frame[:,:,::-1], mode=detector)
        match_frame, match_box, ds, dist = match_descriptors(kpframe, desframe,
                                            kpboxframe, desboxframe,
                                            ransac=ransac)
        if display_lines:
            display_control_lines(frame[:,:,::-1],masked_rgb, match_frame[ds[:sel]],
                            match_box[ds[:sel]], np.array([x_pred[0],y_pred[0]]))
            plt.show()

        if ransac:
            appearance_distribution = get_gaussian_ditribution_frame(frame, match_frame,
percentage=60, sigma=60)
        else:
            if not use_weighted_apperance:
                appearance_distribution = get_gaussian_ditribution_frame(frame, match_frame[ds[:sel]],
percentage=60, sigma=60)
            else:
                appearance_distribution = get_gaussian_ditribution_frame(frame, match_frame,
percentage=60, sigma=60, distances=dist)
        if display_appearance:
            show_image(appearance_distribution, title='Appearance distribution '+str(count))
            plt.show()

        # regsums = region_points(frame.shape, bbox2[0], bbox2[1], match_frame, dist)
        # show_image(regsums, title='Regsum distribution '+str(count))
        # plt.show()

        motion_distribution = get_total_gauss(frame.shape, x_pred[0].astype(int),
y_pred[0].astype(int), percentage=60, sigma=60)
        if display_motion:
            show_image(motion_distribution, title='Motion distribution '+str(count))
            plt.show()

        combined_distribution = ((appearance_distribution*appearance_diff)+(1-
appearance_diff))*((motion_distribution*motion_diff)+(1-motion_diff))
        # combined_distribution = appearance_distribution * motion_distribution
        x_max, y_max = np.unravel_index(np.argmax(combined_distribution, axis=None),
combined_distribution.shape)

        if count%(delta_frames+90) == 0:
            prediction_times.append(count)
            prediction_points_x.append(x_max)
```

```
            prediction_points_y.append(y_max)
            t_pred = np.array(prediction_times) # moments of time
            x_pred = np.array(prediction_points_x)
            y_pred = np.array(prediction_points_y)
            f_x, f_y = txy_regressions(t_pred, x_pred, y_pred)

        if display_combined:
            show_image(combined_distribution, title='Combined '+str(count))
            plt.scatter(y_max, x_max)
            plt.show()

        if display_tracked:
            show_image(frame[:,:,::-1], title='Frame combined distr '+str(count))
            plt.scatter(y_max, x_max)
            plt.show()

        count+=1
        ok, frame = video.read()

        if write_video:
            f,a = plt.subplots(2,2)
            f.suptitle(f"Tracking frame {count}")
            a[0,0].imshow(frame[:,:,::-1])
            a[0,0].scatter(y_max, x_max)
            a[0,0].title.set_text('Frame')
            a[0,1].imshow(combined_distribution,cmap='gray')
            a[0,1].title.set_text('Combined')
            a[1,0].imshow(appearance_distribution,cmap='gray')
            a[1,0].title.set_text('Appearance')
            a[1,1].imshow(motion_distribution,cmap='gray')
            a[1,1].title.set_text('Motion Distribution')
            plt.subplots_adjust(wspace=0.5,hspace=0.5)
            plt.show()

            canvas = FigureCanvas(f)
            canvas.draw()
            size = f.canvas.get_width_height()
            readings_frame = (np.frombuffer(f.canvas.tostring_rgb(),
dtype=np.uint8).reshape((size[1],size[0],3)))
            print(type(readings_frame[0,0,0]))
            cv2.imshow("Tracking", readings_frame[:,:,::-1])
            out.write(readings_frame[:,:,::-1])

        if count == limit:
            break
```

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

    video.release()
    if write_video:
        out.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    plt.close('all')

    # Soccer Ball
    bbox = np.array([93, 599, 364, 369])
    bbox2 = np.array([124, 590, 358, 368])
    poly_points = [(111, 770), (134, 677), (181, 630), (251, 584), (340, 584),
                   (421, 630), (460, 688), (475, 750), (464, 843), (417, 905),
                   (359, 944), (262, 948), (200, 909), (134, 847)]
    print("socccer ball bounce")
    track('socc_bounce.mp4', display_tracked=False, bbox=bbox, bbox2=bbox2,
poly_points=poly_points,
          display_lines=False, display_motion=False, display_appearance=1,
          display_combined=False, limit=-1, detector='sift', ransac=False,
          use_weighted_apperance=True, write_video=True)

    # Roomba
    bbox = np.array([172, 359, 124, 73])
    bbox2 = np.array([178, 357, 115, 71])
    poly_points = [(160, 364), (199, 352), (214, 350), (252, 349),
                   (272, 358), (285, 395), (279, 405), (218, 434),
                   (202, 437), (160, 439)]

    track('tesvor1.avi', display_tracked=False, bbox=bbox, bbox2=bbox2, poly_points=poly_points,
          display_lines=False, display_motion=False, display_appearance=0,
          display_combined=False, limit=-1, detector='sift', ransac=False,
          use_weighted_apperance=True, write_video=True)
```

Statement of Academic Honesty:

“We certify that this project is entirely our own work. We wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. We also certify that we did not share my code or report or provided inappropriate assistance to any student in the class.”