Ahmedabad University

CSE 541 - Computer Vision

Weekly Report 08_04_2023

Team: - Pixel Pioneers

Group Member Details

AU2040170	Kathan Bhavsar
AU2040117	Maulik Ranadive
AU2040183	Nand Patel
AU2040185	Arsh Mansuri

- Prof Mehul Raval

Tasks Performed in the week

For this week we perfected the code for Kalman filter and generalized the Kalman filter class for multi object detection. The code for multi object filtering is achieved by assigning all the detected entities a unique ids which could be used as the index to assign a different Kalman filter to each entity. We were also facing the problem of having multiple bounding boxes for a single object. We solved that problem by performing the Non-maximal Suppression to remove unnecessary bounding boxes and only keeping one bounding box per entity. Each of the final detected bounding boxes are assigned a unique ID using the euclidean distance method. It is basically done by finding the distance between the centers of all the bounding boxes detected and based on the distance calculated if the objects are classified the same or not. We only employed this method for testing Kalman for multi object and we are not planning to use this method for final since our main problem is prediction in occlusion and thus, in highly dense traffic or public areas it could happen that multiple objects are close to each other than the parameters decided by the tracker to differentiate between them. We formed a dictionary in python and used those unique ids as index for the Kalman objects for each object. Code for the above discussed problem is given below.

```
import cv2
import numpy as np
from tracker import *
from Kalman2 import KalmanFilter



model =
cv2.dnn.readNetFromDarknet('F:/CV/Kalman_Test/yolov4_config/yolov4.cfg',
'F:/CV/Kalman_Test/yolov4_weights/yolov4.weights')

dt = 1.0/30.0
track = EuclideanDistTracker()
with open('classes.names', 'r') as f:
    classes = [line.strip() for line in f.readlines()]
```

```python
u = np.array([[0]])
std_acc = 1.0  # standard deviation of acceleration
std_meas = 1.0  # standard deviation of measurement noise

input_size = (416, 416)
confidence_threshold = 0.5
kalman = {}

cap = cv2.VideoCapture('F:/CV/Kalman_Test/input/multiple_person.webm')


frame_rate = int(cap.get(cv2.CAP_PROP_FPS))
frame_size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))

output_video = cv2.VideoWriter('F:/CV/Kalman_Test/output/output_6.mp4v',
cv2.VideoWriter_fourcc(*'mp4v'), frame_rate, frame_size)


while True:

    ret, frame = cap.read()

    if not ret:
        break

    blob = cv2.dnn.blobFromImage(frame, scalefactor=1/255, size=input_size,
swapRB=True, crop=False)


    model.setInput(blob)
    outputs = model.forward(model.getUnconnectedOutLayersNames())
    boxes = []
    confidence_ = []
    class_id_ = []
    detection_output = []

    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > confidence_threshold:
                center_x = int(detection[0] * frame.shape[1])
                center_y = int(detection[1] * frame.shape[0])
                bbox_width = int(detection[2] * frame.shape[1])
                bbox_height = int(detection[3] * frame.shape[0])

                boxes.append([center_x,center_y,bbox_width,bbox_height])
                confidence_.append(confidence)
                class_id_.append(class_id)


detection_output.append([center_x,center_y,bbox_width,bbox_height,confidence,cl
ass_id])
                # print([center_x,center_y,bbox_width,bbox_height])


    det = []
    nms_threshold = 0.7
    indices = cv2.dnn.NMSBoxes(boxes, confidence_, confidence_threshold,
nms_threshold)
    if len(indices)>0:
        for i in indices.flatten():
            box = boxes[i]
            x,y,w,h = box
            if box != None:
                # det.append([x,y,w,h])
                det.append([x,y,w,h])
                print([x,y,w,h])



        if(len(det)>0):
            boxes_id = track.update(det)
            font = cv2.FONT_HERSHEY_SIMPLEX
            for box_id in boxes_id:
                x,y,w,h,id = box_id
                x_pred = x
                y_pred = y
                if(id not in kalman):
                    #  dt, u, std_acc, std_meas
                    kalman[id] = KalmanFilter(dt,u,std_acc,std_meas)
                    x_est = kalman[id].get_state()
                    if x_est is None:
                        x_est = np.array([[x], [y], [0], [0]])
                        kalman[id].set_state(x_est)
                    else:
                        # Predict and update using the measurement
                        kalman[id].predict()
                        z = np.array([[x], [y]])
                        kalman[id].update(z)
                        x_est = kalman[id].get_state()
                        x_pred = x_est[0][0]
                        y_pred = x_est[2][0]
                else:
                    x_est = kalman[id].get_state()
                    if x_est is None:
                        x_est = np.array([[x], [y], [0], [0]])
                        kalman[id].set_state(x_est)
                    else:
                        # Predict and update using the measurement
                        kalman[id].predict()
                        z = np.array([[x], [y]])
                        kalman[id].update(z)
                        x_est = kalman[id].get_state()
                        x_pred = x_est[0][0]
                        y_pred = x_est[2][0]

                cv2.putText(frame,str(id),(int(x - w/2),int(y -
h/2)-15),cv2.FONT_HERSHEY_PLAIN,1,(0,255,255),2)
                cv2.rectangle(frame, (int(x - w/2), int(y - h/2)), (int(x +
w/2), int(y + h/2)), (0, 255, 255), 3)
                cv2.rectangle(frame, (int(x_pred - w/2), int(y_pred - h/2)),
(int(x_pred + w/2), int(y_pred + h/2)), (0, 255, 0), 3)


    cv2.imshow('Output', frame)

    # Press 'q' to exit the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


cap.release()
cv2.destroyAllWindows()
```

Code for the Kalman Filter:

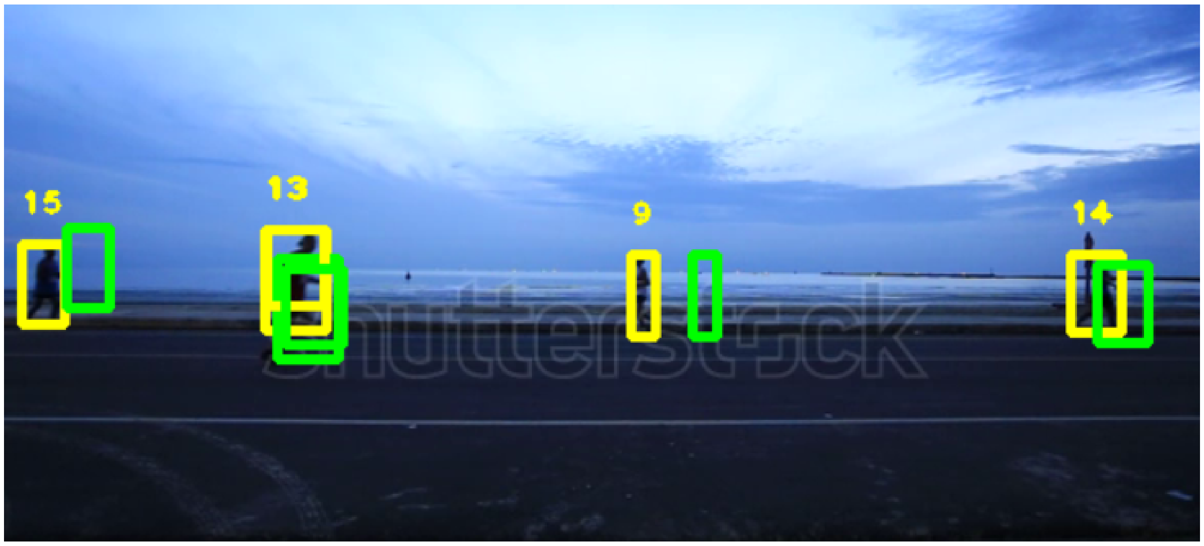
```
import numpy as np

class KalmanFilter:
    def __init__(self, dt, u, std_acc, std_meas):
        # State transition matrix
        self.F = np.array([[1, dt, 0, 0],
                           [0, 1, 0, 0],
                           [0, 0, 1, dt],
                           [0, 0, 0, 1]])

        # Control input matrix
        self.B = np.array([[(dt**2)/2],
                           [dt],
                           [(dt**2)/2],
                           [dt]])

        # Process noise covariance matrix
        self.Q = np.array([[(dt**4)/4, (dt**3)/2, 0, 0],
                           [(dt**3)/2, dt**2, 0, 0],
                           [0, 0, (dt**4)/4, (dt**3)/2],
                           [0, 0, (dt**3)/2, dt**2]]) * std_acc**2

        # Measurement matrix
        self.H = np.array([[1, 0, 0, 0],
                           [0, 0, 1, 0]])

        # Measurement noise covariance matrix
        self.R = np.array([[std_meas**2, 0],
                           [0, std_meas**2]])

        # State vector
        self.x = None

        # State covariance matrix
        self.P = np.eye(4)

        self.u = u
        # self.reset()
```

```
def reset(self):
    self.x = np.matrix([[0],
                        [0],
                        [0],
                        [0]])
    self.P = np.eye(4)
    # Control input

def predict(self):
    # Predict the state vector and covariance matrix


    if(np.any(self.x != None)):
        self.x = self.F.dot(self.x) + self.B.dot(self.u)
        self.P = (self.F.dot(self.P)).dot(self.F.T) + self.Q

def update(self, z):


    Y = z - self.H.dot(self.x)
    S = (self.H.dot(self.P)).dot(self.H.T) + self.R
    K = (self.P.dot(self.H.T)).dot(np.linalg.inv(S))
    self.x = self.x + K.dot(Y)
    self.P = (np.eye(4) - K.dot(self.H)).dot(self.P)

def get_state(self):
    return self.x

def set_state(self,x):
    self.x = x
```

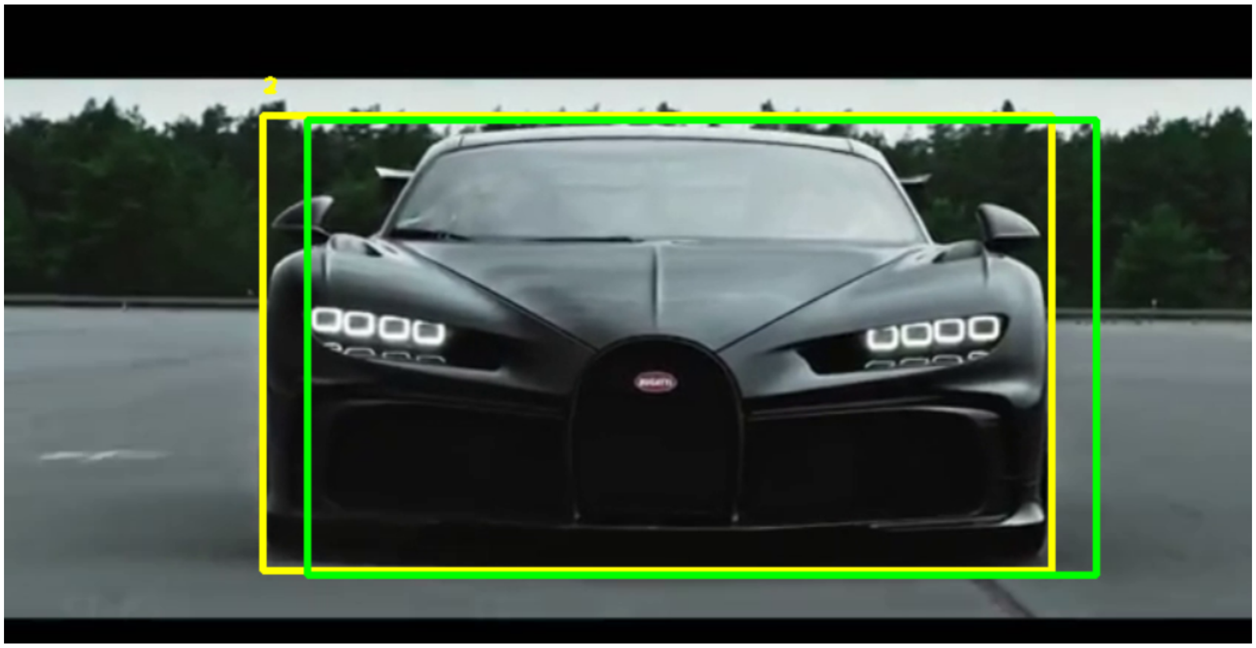
Code for the Tracker:

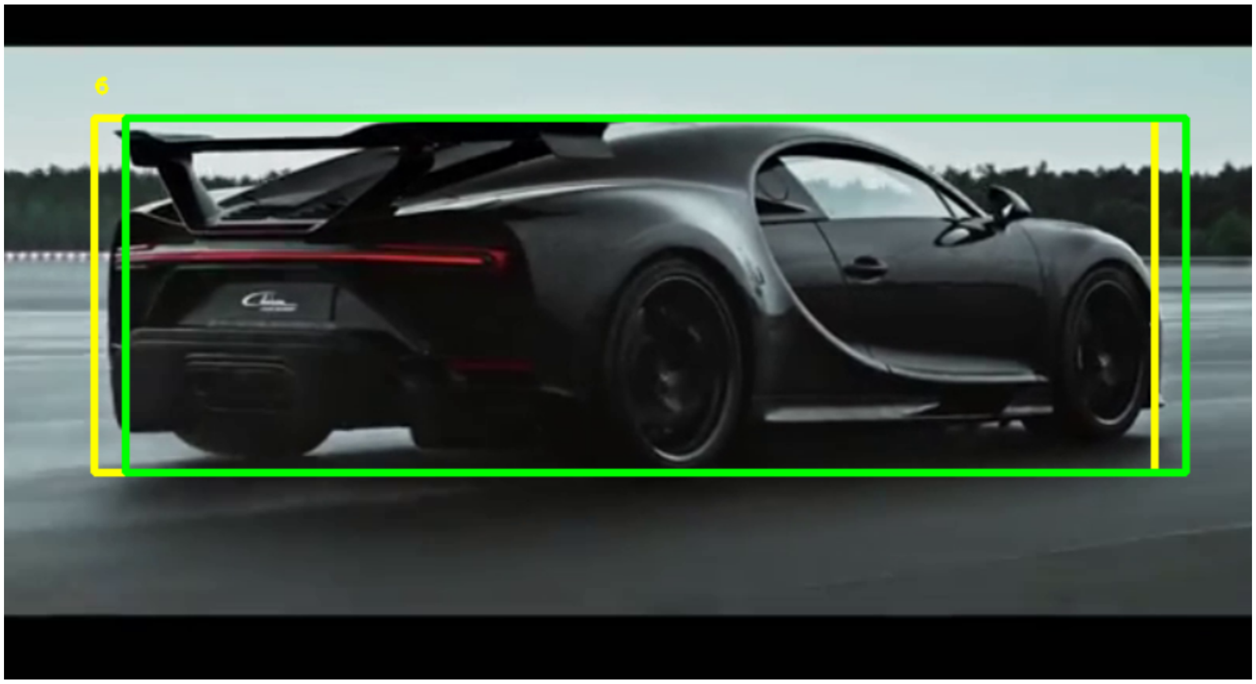
```python
import math


class EuclideanDistTracker:
    def __init__(self):
        # Store the center positions of the objects
        self.center_points = {}
        # Keep the count of the IDs
        # each time a new object id detected, the count will increase by one
        self.id_count = 0


    def update(self, objects_rect):
        # Objects boxes and ids
        objects_bbs_ids = []

        # Get center point of new object
        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            # Find out if that object was detected already
            same_object_detected = False
            for id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 25:
                    self.center_points[id] = (cx, cy)
                    print(self.center_points)
                    objects_bbs_ids.append([x, y, w, h, id])
                    same_object_detected = True
                    break

            # New object is detected we assign the ID to that object
            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
                objects_bbs_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1
        # Clean the dictionary by center points to remove IDS not used anymore
        new_center_points = {}
        for obj_bb_id in objects_bbs_ids:
            _, _, _, _, object_id = obj_bb_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center

        # Update dictionary with IDs not used removed
        self.center_points = new_center_points.copy()
        return objects_bbs_ids
```

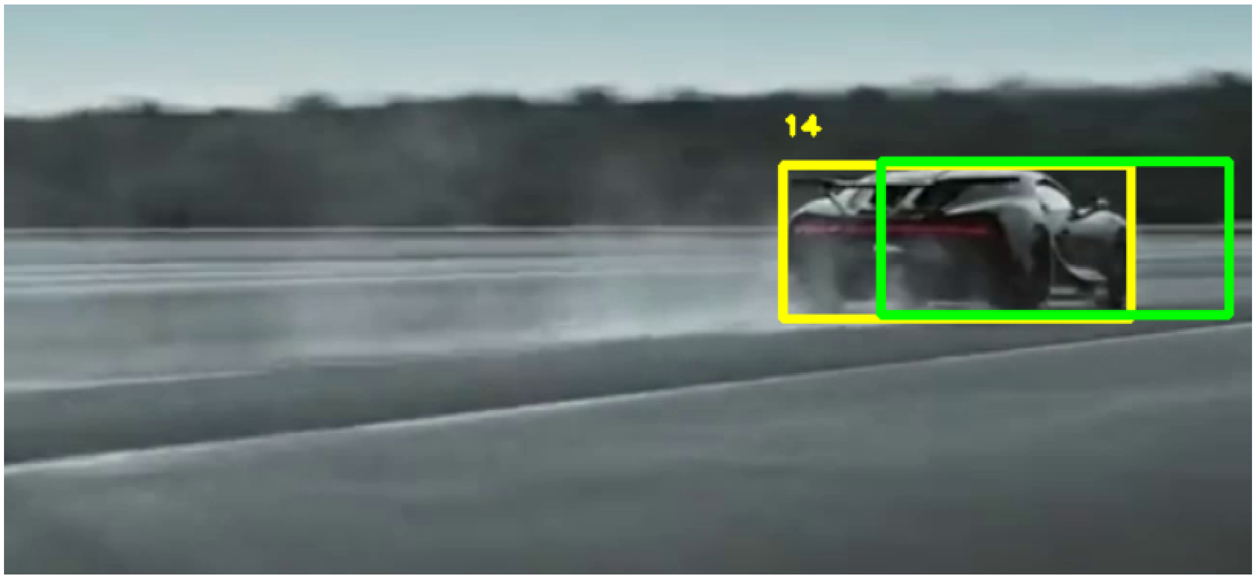
Outcomes of the Tasks Performed



Green Bounding boxes represents the Kalman Prediction and Yellow bounding box represents the Yolov4's detection and number above bounding box are the object's unique ids.


2


6



Output of object detection using YOLOv4 on the video file link:
https://drive.google.com/file/d/1-3pLObyNeIbfobjZTIKRBffzo1FMBMjk/view?usp=sharing

Output of Object detection with Kalman Filter on the video:
https://drive.google.com/file/d/1At1oGzZiE1VJKy5Jxvjq8VZzPpQ28bxU/view?usp=sharing

Tasks to be performed in the upcoming week.

We are planning to use the Convolution neural network to assign a unique id to all the detected objects rather than religion on a Euclidean distance system and this is where the appearance vector part of DeepSort comes in thus solving the problem of identity switching.

References

1. T. (2023, February 2). *TRAIN A CUSTOM YOLOv4 OBJECT DETECTOR (Using Google Colab)*. Medium. https://medium.com/analytics-vidhya/train-a-custom-yolov4-object-detector-using-google-colab-61a659d4868
2. *Home*. (n.d.).GitHub. Retrieved March 28, 2023, from https://github.com/heartexlabs/labelImg.git
3. (*Real-Time YOLOv4 Object Detection on Webcam in Google Colab | Images and Video*, 2020)
4. (*Object Tracking With Opencv and Python*, 2021) https://youtu.be/O3b8IVF93jU