**Predicting the trajectory of an Object with Kalman filter**

We will see a practical approach on how to use the **Kalman filter** to track and predict the trajectory of an object.

At first, I will show simple examples by drawing dots on the screen and having the trajectory predicted, and then we will see in reality how to predict the trajectory of a ball.

**Kalman filter** is an algorithm that takes measurements over time and creates a prediction of the next measurements. This is used in many fields such as sensors, GPS, to predict the position in case of signal loss for a few seconds and this is what we will also see in computer vision.

We will see the simplest possible implementation with **OpenCV (Computer vision)**

* **Kalman filter with dots on solid blue background**

First of all import **kalmanfilter.py(**algorithm sourced out online)and the **OpenCV** library

**from** kalmanfilter **import** KalmanFilter

**import** cv2

Now initialize Kalman filter and we try to insert values to get a prediction

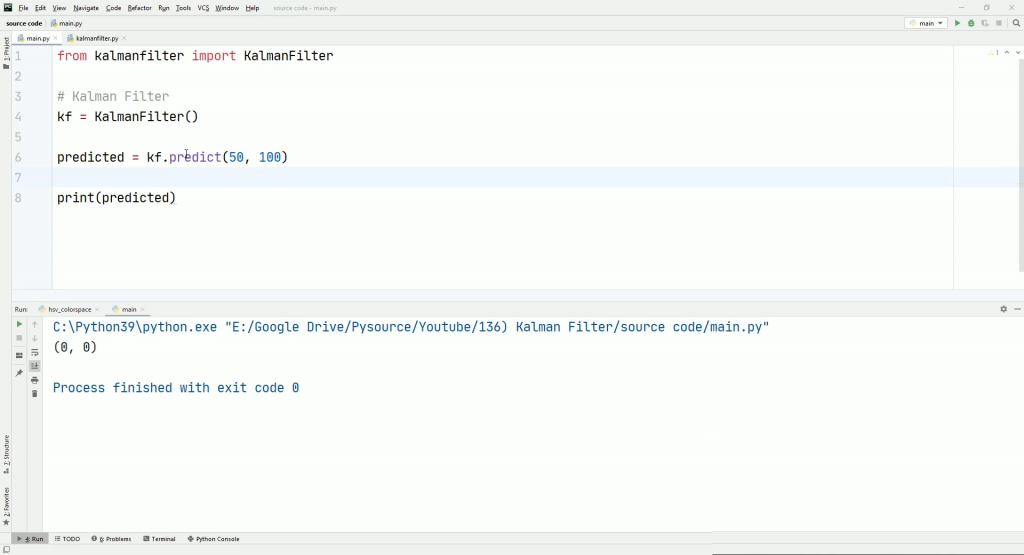
# Kalman Filter

kf = KalmanFilter()

predicted = kf.predict(50,50)

print(predict)

As you can see from the image below, if we print the result, we will get (0,0) because the Kalman filter analysis function needs more values to make a prediction.



**More precision with more values**

By inserting more position points you will get a more and more precise prediction. Here is an example of code to insert more points that will make our Kalman filter better.

# Kalman Filter

kf = KalmanFilter()

predicted = kf.predict(50,100)

predicted = kf.predict(100,100)

predicted = kf.predict(150,100)

predicted = kf.predict(200,100)

print(predict) # result (238,114)

**Simulate the movement of a ball**

To verify and put into practice the theory we have seen in the previous paragraph, we simulate the trajectory of a ball that goes from right to left. We can do everything with Opencv by specifying a series of points where the ball will be.

At the same time we pass all points to the function **kf.predict(x, y)**

**from** kalmanfilter **import** KalmanFilter

**import** cv2

# Kalman Filter

kf = KalmanFilter()

img = cv2.imread("blue\_background.webp")

ball\_positions = [(50, 100), (100, 100), (150, 100), (200, 100), (250, 100), (300, 100), (350, 100), (400, 100), (450, 100)]

**for** pt **in** ball2\_positions:

cv2.circle(img, pt, 15, (0, 20, 220), -1)

predicted = kf.predict(pt[0], pt[1])

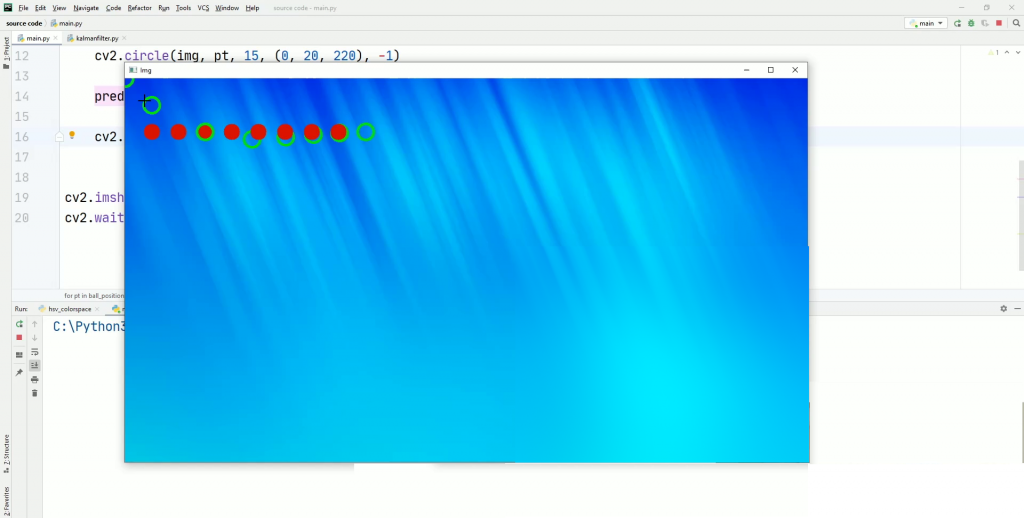
cv2.circle(img, predicted, 15, (20, 220, 0), 4)

cv2.imshow("Img", img)

cv2.waitKey(0)

By executing the code you can see from the image the solid red balls that go from left to right and the green circles that make the prediction of the next position.

As you can see at the alignment the first green circle is on the point with coordinate (0,0) because it does not have enough values to do the processing. Instead, the last green circle indicates, in an absolutely plausible way, the next possible position of the red ball.



Trying to add more points for prediction with this code

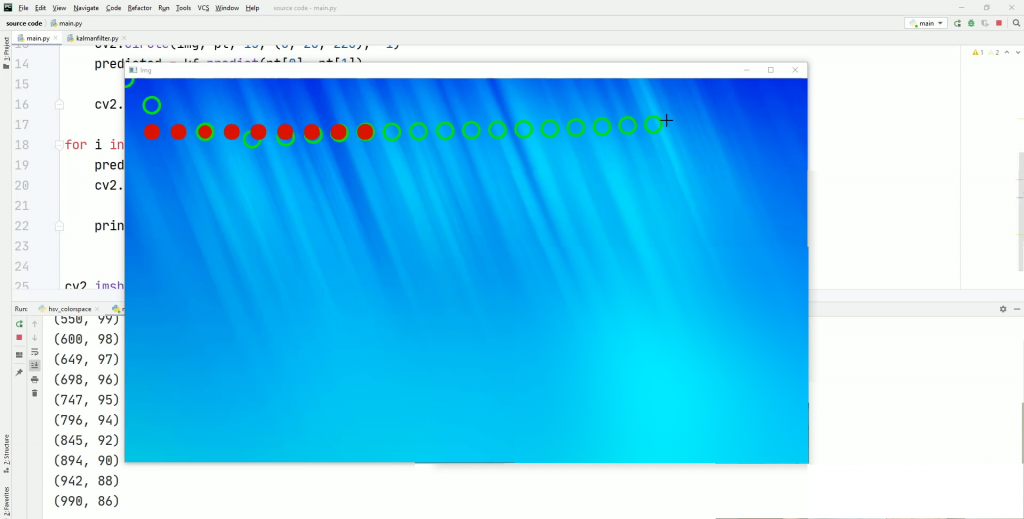
**for** i **in** range(10):

predicted = kf.predict(predicted[0], predicted[1])

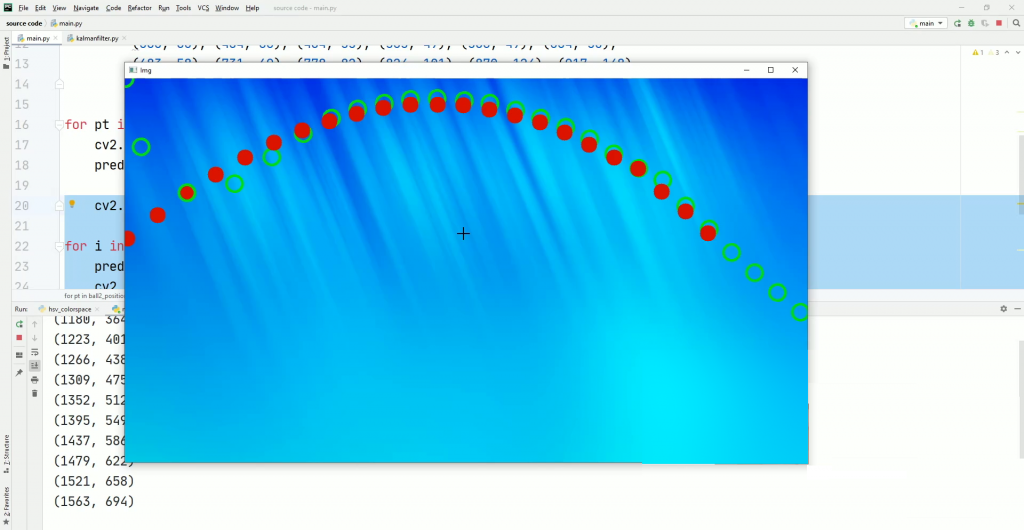
cv2.circle(img, predicted, 15, (20, 220, 0), 4)

print(predicted)

you can see how the Kalman filter manages to make a correct prediction. In the image below you can see other green circles that assume the position of the red ball



The same procedure is valid if I simulate the launch of a ball whose trajectory draws an arc. Again Kalman filter improves from time to time as new points are awarded. The one in the image below is the result.



* **Use kalman filter to predict the trajectory of real object**

**Detect the object**

The first step consists of object detection, in this case of an orange, identified with the colour recognition method

We import everything necessary and proceed with obtaining the frames from the video through a loop

**import** cv2

**from** *orange\_detector* **import** OrangeDetector

**from** *kalmanfilter* **import** KalmanFilter

cap = cv2.VideoCapture("orange.mp4")

# Load detector

od = OrangeDetector()

# Load Kalman filter to predict the trajectory

kf = KalmanFilter()

**while** **True**:

ret, frame = cap.read()

**if** ret **is** **False**:

**break**

Now We use the functions for our orange detector and we get the coordinates of 2 points. However, these correspond to the top-left point and the bottom right point of the object. With a small mathematical operation, we obtain the coordinates of the center.

orange\_bbox = od.detect(frame)

x, y, x2, y2 = orange\_bbox

cx = int((x + x2) / 2)

cy = int((y + y2) / 2)

Now we have what we need, we just need to add the Kalman filter function to predict the future position of the object.

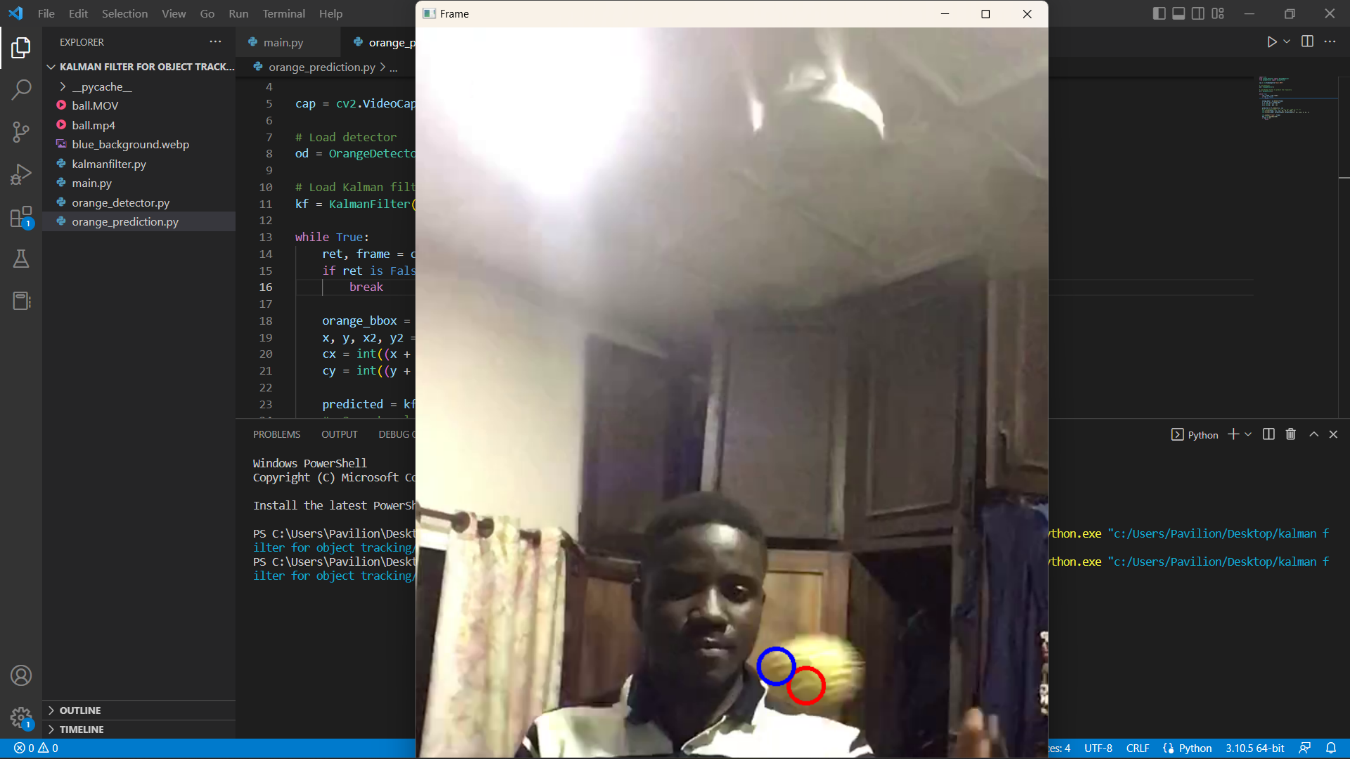
predicted = kf.predict(cx, cy)

#cv2.rectangle(frame, (x, y), (x2, y2), (255, 0, 0), 4)

cv2.circle(frame, (cx, cy), 20, (0, 0, 255), 4)

cv2.circle(frame, (predicted[0], predicted[1]), 20, (255, 0, 0), 4)

Showing a small red circle for the current position and a blue circle for prediction through Kalman filter, this is the result extrapolated from a single frame



OTHER APPLICATIONS

* In the 1960s, the Kalman filter was applied to navigation for the Apollo Project, which required estimates of the trajectories of manned spacecraft going to the Moon and back. With the lives of the astronauts at stake, it was essential that the Kalman filter be proven effective and reliable before it could be used
* Kalman filter is also used in more complex tracking algorithms such as **SORT and Deep SORT** and this allows its use in real scenarios