Ahmedabad
University

CSE 541 - Computer Vision

Weekly Report 02_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

Outcomes of the Tasks Performed

1. **More Training of the YOLOv4 for object detection**

   For this time we train over the yolov4 model with our custom dataset which we created using an image labeling tool. We took around 700 labeled images to train the model. We also generate the coordinates for each bounding box for every image in a separate text file. File with containing the coordinates of the bounding box is helpful for the Kalman filter to predict that object.

2. **Kalman Code for Predicting Velocity and Position of the detected objects.**

   To understand the Kalman filter we researched so many sources from YouTube, research papers, and many more, but finally, we came across a YouTube video that explained the basic principle and steps such as prediction and update used to predict the future state of the detected object and also update the current state of the object.
   The Kalman filter is used by us to estimate the position and velocity of detected objects for future frames (in case of occlusion). We have not assumed a constant velocity model we have taken into consideration of acceleration of the detected object. The prediction will be done with the help of Measurements provided by the yolo and it involves the center of the bounding box in x and y and the previous velocity (initialized to zero) of the object in both directions based on that future values will be predicted.

We also wrote the code for Kalman Filter for both 1D model.

1. **1D model:**

```
# Offsets
iX = 0
iV = 1
NUMVARS = iV + 1

class KF:
  def __init__(self,initial_x: float,initial_v: float,accel_variance: float) ->
None:
    # Mean of State GRV
    self._x = np.zeros(NUMVARS)
    self._x[iX] = initial_x
    self._x[iV] = initial_v


    # self._x = np.array([initial_x, initial_v])
    self._accel_variance = accel_variance

    # Covariance of the State GRV
    self._P = np.eye(NUMVARS)



  def predict(self,dt: float) -> None:
    # F = np.array([[1,dt],[0,1]])
    F = np.eye(NUMVARS)
    F[iX][iV] = dt
    # G = np.array([0.5 * dt**2,dt]).reshape((2,1))
    G = np.zeros((2,1))
    G[iX] = 0.5 * dt**2
    G[iV] = dt

    # x = F*x
    new_x = F.dot(self._x)
    # P = F*P*F.T + G*G.T*a
    new_P = F.dot(self._P).dot(F.T) + G.dot(G.T)*self._accel_variance

    self._x = new_x
    self._P = new_P
```

```python
def update(self,meas_value: float, meas_variance: float)->None:
  # y = z - H*x
  # S = H*P*H.T + R
  # K = P*H.T*S^(-1)
  # x = x + K*y
  # P = (I - K*H)*P
  z = np.array([meas_value])
  R = np.array([meas_variance])
  H = np.array([1, 0]).reshape((1,2))

  y = z - H.dot(self._x)
  S = H.dot(self._P).dot(H.T) + R

  K = self._P.dot(H.T).dot(np.linalg.inv(S))

  new_x = self._x + K.dot(y)
  new_P = (np.eye(2) - K.dot(H)).dot(self._P)

  self._P = new_P
  self._x = new_x


@property
def cov(self) -> np.array:
  return self._P


@property
def mean(self)->np.array:
  return self._x

@property
def pos(self) -> float:
  return self._x[iX]

@property
def vel(self) -> float:
  return self._x[iV]
```

Code to Plot the predicted velocity and position with time (Since YOLO and Kalman are not connected here I have used normal for loop and its iteration represent the time steps)

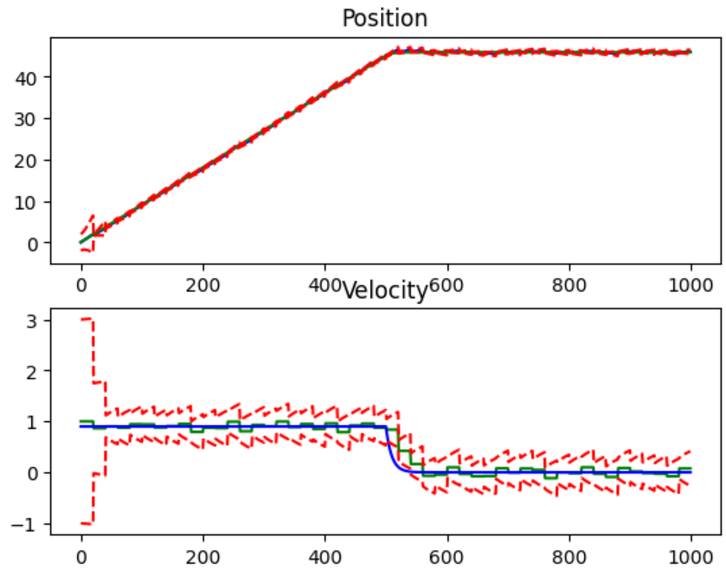
```
# Change of position and velocity with time
from matplotlib import pyplot as plt
plt.ion()
plt.figure()

x = 0.0 #Initial Position
v = 1.0 # Initial Velocity
a = 0.1 #Initial Acc

real_x = 0.0
real_v = 0.9
meas_variance = 0.1**2
real_xs = []
real_vs = []

kf2 = KF(initial_x = x,initial_v = v,accel_variance = a)

dt = 0.1
Steps = 1000
MEAS_EVERY_STEP = 20

means = []
covs = []

for i in range(Steps):
  # decreasing the speed after 500 steps since its constant acc kalman filter
model
  if i>500:
    real_v *= 0.9
  covs.append(kf2.cov)
  means.append(kf2.mean)

  real_x = real_x + dt*real_v
  kf2.predict(dt=dt)

  if(i!=0 and i%20 == 0):
    kf2.update(meas_value = real_x + np.random.randn()*np.sqrt(meas_variance),
meas_variance=meas_variance)

  real_xs.append(real_x)
```

```
  real_vs.append(real_v)

plt.subplot(2,1,1)
plt.title("Position") # Increasing Position.
plt.plot([me[0] for me in means],'b')
plt.plot(real_xs,'g')
plt.plot([me[0] - 2*np.sqrt(cov[0,0]) for me, cov in zip(means,covs)],'r--')
plt.plot([me[0] + 2*np.sqrt(cov[0,0]) for me, cov in zip(means,covs)],'r--')

plt.subplot(2,1,2)
plt.title("Velocity") # constant velocity.
plt.plot([me[1] for me in means],'g')
plt.plot(real_vs,'b')
plt.plot([me[1] - 2*np.sqrt(cov[1,1]) for me, cov in zip(means,covs)],'r--')
plt.plot([me[1] + 2*np.sqrt(cov[1,1]) for me, cov in zip(means,covs)],'r--')
```

Output of object detection using YOLOv4 on the video file link:

1. **https://drive.google.com/file/d/1-3pLObyNeIbfobjZTIKRBffzo1FMBMjk/view?usp=sharing**
2. **https://drive.google.com/file/d/1-LJQ_u9UmJ6De5ew5F1Qr4esnRTG8tbC/view?usp=sharing**

Output of the Kalman filter for dummy Velocity and position.



Tasks to be performed in upcoming week

- For the next week we are planning to integrate the Kalman filter with YOLOv4 and train a CNN for feature extraction to develop the Appearance vector for solving identity switching problems.

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. *Understand & Code a Kalman Filter [Part 1 Design]*. (2019, November 7). YouTube. Retrieved April 2, 2023, from https://youtu.be/TEKPcyBwEH8