关于mediapipe卡尔曼滤波算法的问题

1.请描述卡尔曼滤波算法的基本原理

卡尔曼滤波首先认定系统为线性系统，且噪声满足高斯分布，通过状态方程来描述物体的状态。卡尔曼滤波通过观测值和预测值来修正k时刻的状态，预测值受到k-1时刻的状态，以及外界噪声的影响，再通过一个权重系数H来确定修正后的状态，并不通过预测来更新调节H，从而得到物体最可能的真实状态。

卡尔曼滤波因为考虑到了观测噪声和过程噪声对于数据的影响，因此，以图像处理为例，经过卡尔曼滤波之后，相当于去除了图像中的噪声点，实现了图像的滤波处理。

从数学的角度来看，本质上，k卡尔曼滤波做了2件事:

(1)基于前一时间点的系统状态的概率分布给出当前时间点的系统状态的概率分布预测P1。

(2)基于当前时间点的观测量的概率分布Q给出另一组对当前时间点系统状态的概率分布预测P2 (这里需要注意，如果传感器能直接把系统状态都观测了，那么我们直接观测量的概率分布Q作为P2就好了，但是有可能传感器观测的物理量量并不是直接的系统状态，而是其他物理量，这时我们需要通过做一个线性变换来获得P2)。

现在，我们有两个关于当前时间点系统状态的概率分布的预测P1和P2，我们于是可以计算它们的联合概率分布，也就是为什么文章中要把两个概率分布乘到一起去。这个联合概率分布就是我们最终想要的更正过后的当前时间点的系统状态概率分布，有了它我们就可以进入下一个时间点继续迭代我们交替的预测-更正步骤了。

2.附件代码已实现卡尔曼滤波算法对单个2D关键点的实时平滑处理

参考代码如下:

```
import numpy as np
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt

mpHands = mp.solutions.hands
hands = mpHands.Hands(static_image_mode=False,
                      max_num_hands=2,
                      min_detection_confidence=0.5,
                      min_tracking_confidence=0.5)

# 定义x的初始状态 -- 需要修改，初始化为捕获手势的位置x坐标
x_mat = np.mat([[0, ], [0, ], [0, ], [0, ]])
# 定义初始状态协方差矩阵
p_mat = np.mat([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
# 定义初始化控制矩阵
b_mat = np.mat([[0.5, ], [1, ], [0.5, ], [1, ]])
# 定义状态转移矩阵，因为每秒钟采一次样，所以delta_t = 1
f_mat = np.mat([[1, 1, 0, 0], [0, 1, 0, 0], [0, 0, 1, 1], [0, 0, 0, 1]])
# 定义状态转移协方差矩阵，这里我们把协方差设置的很小，因为觉得状态转移矩阵准确度高
q_mat = np.mat([[0.3, 0, 0, 0], [0, 0.3, 0, 0], [0, 0, 0.3, 0], [0, 0, 0, 0.3]])
# 定义观测矩阵
h_mat = np.mat([1, 0, 1, 0])
```

```
# 定义观测噪声协方差
r_mat = np.mat([1])

video = cv2.VideoCapture(0)
first_frame_flag = True
while True:
    ret, frame = video.read()
    img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(img)
    h, w, c = img.shape
    if results.multi_hand_landmarks:
        for handLms in results.multi_hand_landmarks:
            for id, lm in enumerate(handLms.landmark):
                cx, cy = int(lm.x * w), int(lm.y * h)
                if id == 9:
                    if first_frame_flag:
                        x_mat[0, 0] = cx
                        x_mat[2, 0] = cy
                        first_frame_flag = False
                    else:
                        x_predict = f_mat * x_mat
                        p_predict = f_mat * p_mat * f_mat.T + q_mat
                        x_mat_before = x_mat
                        kalman = p_predict * h_mat.T / (h_mat * p_predict *
h_mat.T + r_mat)
                        # 出现问题
                        x_mat = x_predict + np.multiply(kalman, (np.mat(
                            [[cx, ], [cx - x_mat_before[0, 0], ], [cy, ], [cy -
x_mat_before[2, 0], ]]) - x_predict))
                        p_mat = (np.eye(x_mat.shape[0]) - kalman * h_mat) *
p_predict
                        noise_x = cx + int(np.random.normal(0, 1) * 10)
                        noise_y = cy + int(np.random.normal(0, 1) * 10)
                        cv2.circle(frame, (noise_x, noise_y), 6, (0, 0, 255),
cv2.FILLED)
                        cv2.circle(frame, (int(x_mat[0, 0]), int(x_mat[2, 0])),
3, (0, 255, 0), cv2.FILLED)
                    break
    cv2.imshow('video', frame)
    if cv2.waitKey(33) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
video.release()
```

3.请将卡尔曼滤波算法扩展到mediapipe全身2D关键点并总结实验效果

```
import numpy as np
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt
import copy
```

```
mp_holistic = mp.solutions.holistic
holistic = mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5)

import cv2
import mediapipe as mp

# mp.solutions.drawing_utils用于绘制
mp_drawing = mp.solutions.drawing_utils

# 参数：1、颜色，2、线条粗细，3、点的半径
DrawingSpec_point1 = mp_drawing.DrawingSpec((255, 0, 0), 2, 2)
DrawingSpec_line1 = mp_drawing.DrawingSpec((255, 0,0), 1, 1)
DrawingSpec_point2 = mp_drawing.DrawingSpec((0,0, 255), 2, 2)
DrawingSpec_line2 = mp_drawing.DrawingSpec((0, 0, 255), 1, 1)
l = [[]*i for i in range(33)]
# mp.solutions.pose，是人的骨架
mp_pose = mp.solutions.pose
# mpHands = mp.solutions.hands
# hands = mpHands.Hands(static_image_mode=False,
#                       max_num_hands=2,
#                       min_detection_confidence=0.5,
#                       min_tracking_confidence=0.5)

# # 定义x的初始状态 -- 需要修改，初始化为捕获手势的位置x坐标
# x_mat = np.mat([[0, ], [0, ], [0, ], [0, ]])
# # 定义初始状态协方差矩阵
# p_mat = np.mat([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
# # 定义初始化控制矩阵
# b_mat = np.mat([[0.5, ], [1, ], [0.5, ], [1, ]])
# # 定义状态转移矩阵，因为每秒钟采一次样，所以delta_t = 1
# f_mat = np.mat([[1, 1, 0, 0], [0, 1, 0, 0], [0, 0, 1, 1], [0, 0, 0, 1]])
# # 定义状态转移协方差矩阵，这里我们把协方差设置的很小，因为觉得状态转移矩阵准确度高
# q_mat = np.mat([[0.3, 0, 0, 0], [0, 0.3, 0, 0], [0, 0, 0.3, 0], [0, 0, 0,
0.3]])
# # 定义观测矩阵
# h_mat = np.mat([1, 0, 1, 0])
# # 定义观测噪声协方差
# r_mat = np.mat([1])
#
# video = cv2.VideoCapture(0)
# first_frame_flag = True
# 定义x的初始状态 -- 需要修改，初始化为捕获手势的位置x坐标
epoch = 0
# 定义初始状态协方差矩阵
p_mat = np.mat([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
# 定义初始化控制矩阵
b_mat = np.mat([[0.5, ], [1, ], [0.5, ], [1, ]])
# 定义状态转移矩阵，因为每秒钟采一次样，所以delta_t = 1
f_mat = np.mat([[1, 1, 0, 0], [0, 1, 0, 0], [0, 0, 1, 1], [0, 0, 0, 1]])
# 定义状态转移协方差矩阵，这里我们把协方差设置的很小，因为觉得状态转移矩阵准确度高
q_mat = np.mat([[0.3, 0, 0, 0], [0, 0.3, 0, 0], [0, 0, 0.3, 0], [0, 0, 0, 0.3]])
# 定义观测矩阵
h_mat = np.mat([1, 0, 1, 0])
# 定义观测噪声协方差
```

```
r_mat = np.mat([1])

class karman:
    def __init__(self,id,x_mat,visited):
        self.id = id
        self.x_mat = x_mat
        self.visited = visited

for i in range(33):
    x_mat = np.mat([[0, ], [0, ], [0, ], [0, ]])
    l[i] = karman(i,x_mat,False)

def kalman_filter(x,cx,cy):
    global p_mat,b_mat,f_mat,q_mat,h_mat,r_mat
    if (x.visited==False):
        x_mat = np.mat([[0, ], [0, ], [0, ], [0, ]])
        x_mat[0, 0] = cx
        x_mat[2, 0] = cy
        x.x_mat = x_mat
        x.visited = True
    else:
        x_mat = x.x_mat
        x_predict = f_mat * x_mat
        p_predict = f_mat * p_mat * f_mat.T + q_mat
        x_mat_before = x_mat
        kalman = p_predict * h_mat.T / (h_mat * p_predict * h_mat.T + r_mat)
        # 出现问题
        x_mat = x_predict + np.multiply(kalman, (np.mat(
            [[cx, ], [cx - x_mat_before[0, 0], ], [cy, ], [cy - x_mat_before[2,
0], ]]) - x_predict))
        p_mat = (np.eye(x_mat.shape[0]) - kalman * h_mat) * p_predict
        # noise_x = cx + int(np.random.normal(0, 1) * 5)
        # noise_y = cy + int(np.random.normal(0, 1) * 5)
        x.x_mat = x_mat

video = cv2.VideoCapture(0)
while True:
    ret, frame = video.read()
    img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = holistic.process(img)
    h, w, c = img.shape
    if (results.pose_landmarks):
        # for poseLms in results.pose_landmarks:
        for id, lm in enumerate(results.pose_landmarks.landmark):
            cx, cy = int(lm.x * w), int(lm.y * h)
            if id in [i for i in range(33)]:
                noise_x = cx + int(np.random.normal(0, 1) * 3)
                noise_y = cy + int(np.random.normal(0, 1) * 3)
                lm.x = noise_x/w
                lm.y = noise_y/h
        mp_drawing.draw_landmarks(frame, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS, DrawingSpec_point1, DrawingSpec_line1)
        # mp_drawing.draw_landmarks(frame, plm2, mp_holistic.POSE_CONNECTIONS,
DrawingSpec_point2, DrawingSpec_line2)
```

```
        for id, lm in enumerate(results.pose_landmarks.landmark):
            cx, cy = int(lm.x * w), int(lm.y * h)
            if id in [i for i in range(33)]:
                # noise_x = cx + int(np.random.normal(0, 1) * 5)
                # noise_y = cy + int(np.random.normal(0, 1) * 5)
                # cv2.circle(frame, (noise_x, noise_y), 5, (0, 0, 255),
cv2.FILLED)
                x_mat = l[id].x_mat
                kalman_filter(l[id],cx,cy)
                x_mat = l[id].x_mat
                lm.x = int(x_mat[0, 0]) / w
                lm.y = int(x_mat[2, 0]) / h
                # plm2 = copy.deepcopy(results.pose_landmarks)
                # cv2.circle(frame, (int(x_mat[0, 0]), int(x_mat[2, 0])), 3, (0,
255, 0), cv2.FILLED)
        mp_drawing.draw_landmarks(frame, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS, DrawingSpec_point2, DrawingSpec_line2)

    cv2.imshow('video', frame)
    if cv2.waitKey(33) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
video.release()
```

4.请验证mediapipe自带的伪3D关键点检测与2D关键点检测的x，y坐标是否一致?

一致。3D关键点就多出了一个深度值的坐标。

5.尝试将卡尔曼滤波算法扩展到mediapipe伪3D关键点的平滑处理中，并总结实验效果

可以防止关键点的剧烈抖动，但效果一般。

6.尝试将针对3D点关键点检测平滑处理的卡尔曼滤波算法封装成函数接口

```
class karman:
    def __init__(self,id,x_mat,visited):
        self.id = id
        self.x_mat = x_mat
        self.visited = visited

def kalman_filter(x,cx,cy):
    global p_mat,b_mat,f_mat,q_mat,h_mat,r_mat
    if (x.visited==False):
        x_mat = np.mat([[0, ], [0, ], [0, ], [0, ]])
        x_mat[0, 0] = cx
        x_mat[2, 0] = cy
        x.x_mat = x_mat
        x.visited = True
    else:
        x_mat = x.x_mat
        x_predict = f_mat * x_mat
        p_predict = f_mat * p_mat * f_mat.T + q_mat
```

```
        x_mat_before = x_mat
        kalman = p_predict * h_mat.T / (h_mat * p_predict * h_mat.T + r_mat)
        # 出现问题
        x_mat = x_predict + np.multiply(kalman, (np.mat(
            [[cx, ], [cx - x_mat_before[0, 0], ], [cy, ], [cy - x_mat_before[2,
0], ]]) - x_predict))
        p_mat = (np.eye(x_mat.shape[0]) - kalman * h_mat) * p_predict
        # noise_x = cx + int(np.random.normal(0, 1) * 5)
        # noise_y = cy + int(np.random.normal(0, 1) * 5)
        x.x_mat = x_mat
```

7.了解卡尔曼滤波算法在计算机视觉领域中的其他应用

防抖动。

附件：

https://blog.csdn.net/qq_42500340/article/details/124476348