Caio A. K. Monteiro
Breno Siquara Azevedo

14/11/2018

Filtro de Partículas

O objetivo do relatório é fazer a análise e implementação do algoritmo de filtro de partículas apresentado em sala de aula. O trabalho será dividido em duas partes: Obtenção do centro de massa usando OpenCV e a implementação do filtro de partículas.

1. OpenCV para obter o centro de massa.

Existem várias abordagens para se detectar um objeto e seu centro de massa. Neste trabalho usaremos uma maneira mais simples mas que irá atender para demonstrar o filtro de partículas, que é o foco deste trabalho.

A abordagem que utilizaremos é detectar um objeto de cor uniforme em um espectro de cor definido previamente.

Imports necessários para o funcionamento do código. Uma observação necessária é o uso do "imutils" que nada mais é que uma série de implementações usando o OpenCV que fornecem ferramentas úteis. Um exemplo é a diminuição da dimensão do vídeo, fazendo com que seu processamento seja reduzido.

```
#Ajuda na passagem de argumentos para o programa
import argparse

#Ferramentas criadas usando OpenCV
import imutils

#importando o OpenCV
import cv2
```

É definida uma fronteira superior e inferior de cores (BGR) no modelo HSV.

```
#Fronteiras do espectro de cor no modelo HSV
corLower = (29, 86, 6)
corUpper = (64, 255, 255)
```

Após o vídeo ser aberto, o código entra em um loop infinito. Em cada interação um frame do vídeo é capturado para ser analisado. A condição de parada do loop é o frame da interação ser nulo, ou seja, o vídeo acabar.

```
#Lendo o video com OpenCV
video  = cv2.VideoCapture(args["video"])

while True:
    #Captura o frame atual
    cap, frame = video.read()

    #Verifica se esta no final do Video
    if frame is None:
        break
```

Primeiro o algoritmo borra a imagem para deixa la mais uniforme e sem ruídos. Entenda-se ruídos como uma imagem com muita informação. O próximo passo é alterar o frame para o modelo de cor HSV. Outras abordagens alteram para uma escala de cinza por exemplo. Com o frame no modelo de cor que precisamos, aplicamos uma máscara ao frame, no qual se tornará binário, ou seja, aonde estiver entre as fronteiras de cores previamente definidas vai ficar na cor branca, todo o resto fica preto. Com a imagem binária, nós precisamos reduzir as imperfeições e torná la mais uniforme. Para isso usamos duas funções que reduzem as "bolhas" e as "erosões".

```
#Borra a imagem para reduzir os ruidos e deixa-la uniforme
blurred = cv2.GaussianBlur(frame, (11, 11), 0)

#Converte o frame para o modelo de cor HSV
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

#Aplica uma mascara limitada pelas fronteiras, tornando o frame binario
mask = cv2.inRange(hsv, corLower, corUpper)

#Reduzindo as imperfeicoes removendo "bolhas" e "erosoes" do frame
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
```

Utilizaremos uma função do OpenCV para identificar contornos no frame.

```
#Identificando o contorno na mascara
contorno = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

#Garantir que vai ser compativel com opencv 2.4 e 3
contorno = contorno[0] if imutils.is_cv2() else contorno[1]
```

Após encontrarmos contorno, utilizaremos apenas o maior contorno identificado. Com isso, só vamos tratar o caso de apenas um objeto na imagem (o maior). Em seguida, identificamos o centro desse objeto e o raio até seu contorno.

```
#Verificando se pelo menos um contorno foi identificado
if len(contorno) > 0:

    #Identificando o maior contorno
    c = max(contorno, key=cv2.contourArea)

    #Identifica o menor delimitador
    ((x, y), raio) = cv2.minEnclosingCircle(c)

    #Identificando coordenadas do centro
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

    ####### Se quiser criar o contorno do OpenCV no objeto #######
```

É possível tratar mais de um objeto se feita algumas simples modificações no código. Mas para simplificar a análise, iremos utilizar apenas um objeto.

2. Aplicação do Filtro de Partículas

Utilizaremos o centro de massa encontrado anteriormente como parâmetro de entrada para o filtro de partículas. A função retorna uma lista dos filtros. Em seguida, iremos imprimir no frame, cada partícula, a centróide obtida com o OpenCV e o círculo que representa a média das partículas.

```
#Aplicando o filtro de particulas com relacao ao centro
lst_particulas = pf.filtro_de_particulas(center,lst_particulas)

#Printando cada particula
for part in lst_particulas:
    cv2.circle(frame, (int(part.pos_x),int(part.pos_y)), 5, (0, 255, 255), 2)

if raio > 10:
    #Printando o circulo calculado pela média das partículas
    cv2.circle(frame, pft.media_pos(lst_particulas), int(raio),(0, 255, 255), 2)
    #Printando a centroid obtida pelo OpenCV
    cv2.circle(frame, center, 5, (0, 0, 255), -1)
```

2.1 Filtro de Partículas - particle_filter.py

A primeira etapa do algoritmo é o *modelo de movimento,* que atribui para cada partícula um novo valor para theta, velocidade, posição X e posição Y. O valor de theta é calculado levando em conta o valor atual e uma "sujeira" adicionada a esse valor. O mesmo princípio é atribuído a velocidade. As novas posições da partícula são influenciadas pelo valor da posição atual, o seu deslocamento (Velocidade * Tempo) e o theta.

```
def motion_model(delta_time, particula):

    new_theta = particula.theta + np.random.normal(0.0,STD_DEV_T)

    new_velocity = particula.velocity + np.random.normal(0.0,STD_DEV_V)
    if (new_velocity > V_MAX):
        new_velocity = V_MAX
    elif (new_velocity < V_MIN):
        new_velocity = V_MIN

    new_x = particula.pos_x + (particula.velocity * delta_time * math.cos(particula.theta))
    new_y = particula.pos_y + (particula.velocity * delta_time * math.sin(particula.theta))

    particula.pos_x = int(new_x)
    particula.pos_y = int(new_y)
    particula.velocity = new_velocity
    particula.theta = new_theta

    return particula
```

A segunda etapa, chamada de *modelo de observação*, calcula o peso normalizado de cada partícula e atribui a ela esse valor. O cálculo do peso é influenciado pela distância da partícula ao centro do objeto.

```
def observation_model(lst_particulas,centro):
    sum_peso = 0.0

    #calculando o peso de cada particula
    for part in lst_particulas:
        w = pft.calc_peso(pft.calc_distancia(part,centro))
        part.weight = w
        sum_peso = sum_peso + w

    #Normalizando o peso de cada partiula
    for part in lst_particulas:
        part.weight = part.weight/sum_peso

    return lst_particulas
```

A última etapa do algoritmo, é a fase de *reamostragem.* Nessa fase, organizamos as partículas de forma que seu peso determine o "espaço" que ela ocupa em uma amostra. Em seguida, calculamos a média dos pesos para servir de referência de deslocamento na amostra. O próximo passo é fazer a reamostragem em si. Isso se dá selecionando inicialmente uma partícula aleatória na amostra como referência inicial e deslocamos a posição de referência de acordo com a média dos pesos calculado anteriormente. Esse processo se repete até que a nova amostra (que tem a mesma quantidade de partículas) esteja completa.

Partículas com pesos maiores tem a chance maior de ser selecionada para a reamostragem. Processo parecido com o que foi visto em algoritmos genéticos.

```
def resample(lst_particulas):
    #Definindo o vetor de distribuicao de probabilidade
    for i in range(NUM_PARTICULAS):
        #definindo as fronteiras de cada particula na distribuicao
        if i == 0:
            lst_particulas[i].resample_weights = (0,lst_particulas[i].weight)
        else:
            lst_particulas[i].resample_weights = (lst_particulas[i-1].resample_weights[1],lst_pa

    #Primeira particula selecionada aleatoriamente com um peso aleatorio de referencia
    referencia = random.random()

    #media dos pesos
    k = pft.media_pesos(lst_particulas)

    new_sample = []
    for i in range(NUM_PARTICULAS):
        #Operacao para tornar a lista circular
        if referencia > 1:
            referencia = referencia - 1
        #Funcao getParticle_byWeight para buscar a particula
        #Funcao copy para fazer uma copia da particular e adicionala a uma nova amostra
        nova_particula = pft.copy_particle(pft.getParticle_byWeight(referencia,lst_particulas))
        new_sample.append(nova_particula)
        referencia = referencia + k

    return new_sample
```

*Infelizmente a linha do "else" era muito grande e precisou ser cortada no print.

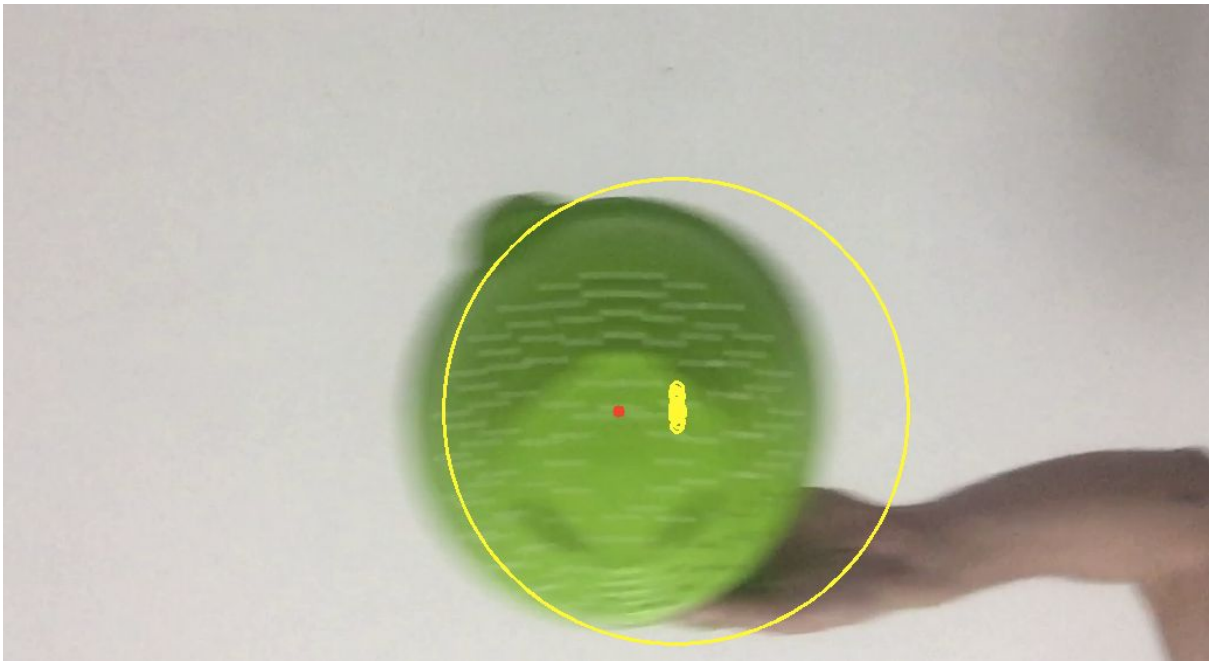
Com todos os passos definidos, basta fazer uma chamada da função de filtro de partículas para cada frame. A função nada mais é do que a chamada das etapas anteriores.

```
def filtro_de_particulas(centro,lst_particulas):
    #Modelo de Movimento
    for i in range(NUM_PARTICULAS):
        lst_particulas[i] = motion_model(DELTA_TIME,lst_particulas[i])

    #Modelo de Observacao
    lst_particulas = observation_model(lst_particulas,centro)

    #Re-amostragem
    lst_particulas = resample(lst_particulas)
    return lst_particulas
```

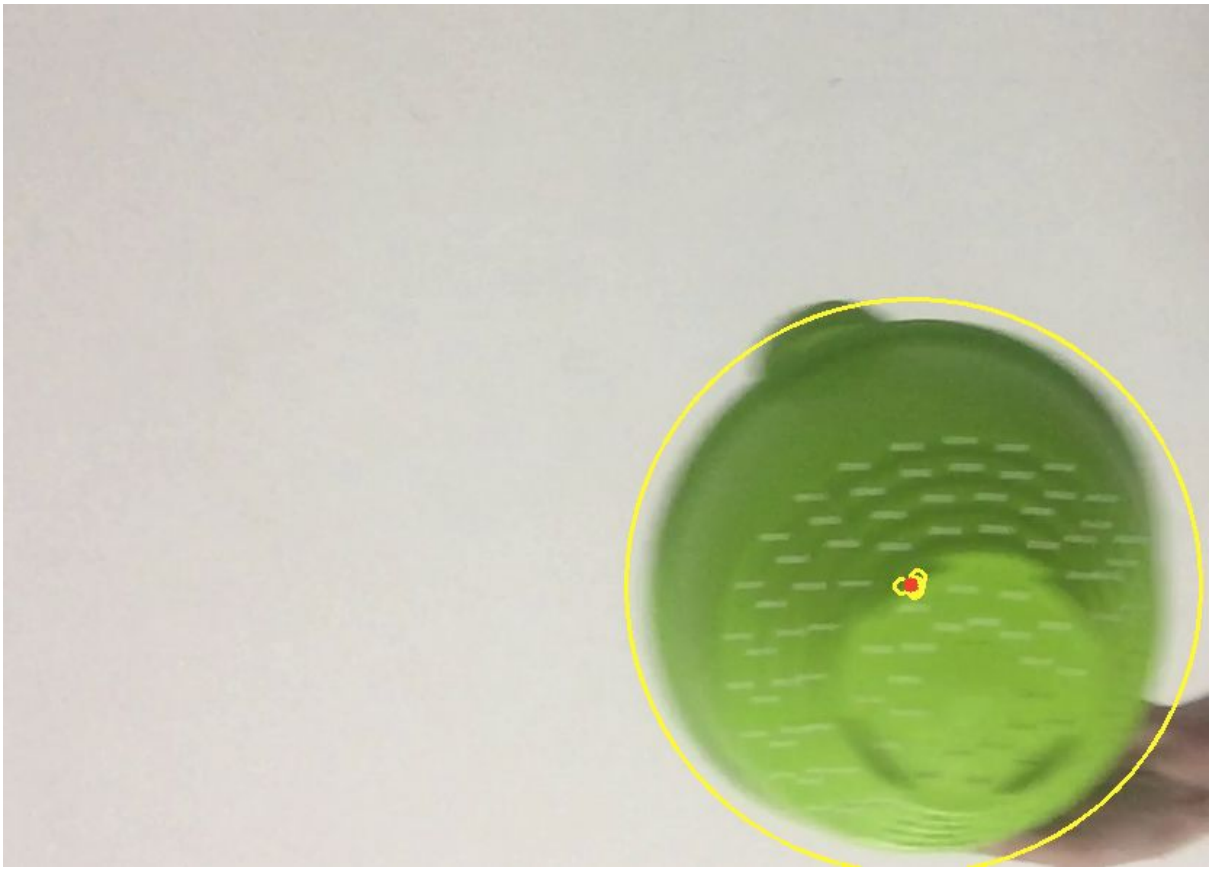
Foi criado um arquivo *pf_tools.py* que possui as funções de apoio às etapas do filtro de partículas. E também um arquivo *particle.py* aonde tem a definição da classe da partícula.

Para rodar o algoritmo: python center.py --video bola.mov

Algoritmo em funcionamento:



Com movimentos bruscos as partículas tentam acompanhar o centro.



Movimentos mais uniformes fazem com que a acurácia seja maior.