École Polytechnique Montréal
Génie Informatique

INF6804 : Vision par ordinateur
Présenté à Guillaume-Alexandre Bilodeau

Laboratoire 3

Travail remis par :
XING, Jin Ling- #1915481
MICHÉA, Luc - #1921803

7 Avril 2019

Introduction

In this lab, we are asked to track multiple objects in a video sequence from boxes that surround the objects on the first frame. From those boxes, we have to deduce a description of the object, a model, that will be used to track the object during the rest of the video. In a first part we will identify what are the difficulties in this video sequence and try to find which method could work best for this video in the second part. Then, in the third part, we will implement the solution and do some tests, using those tests to see if we need to add some features in our method. In part four, we will do the results of our method on the video given, and analyse the results. Finally we will conclude on the project.

1 The video Itself and Difficulties

The video itself is a video of two cups that are moved from a table to another by someone. In this video we have to track the two cups as accurately and as long as possible. This video presents some hard features in video tracking. Those difficulties are expressed in the next sections :

1.1 Object Scale Change

At the beginning of the video sequence, the cups are left on a table, far from the camera. Once the person take the cups, the scale change a lot cause the person is doing some back and forth movement in front of the camera. Throughout the video, the scale continues to change with those back and forth. This movement, needs to be taken into account when we will choose our tracking method. Our tracking method should be able to change the scale of the object detected too. Over time would be a good fit.

1.2 Object Moving Fast

As a similar feature from the previous one, the objects scale are pretty big in pixels compared to their real life scale. And also the man's arm taking the cups move fast with them. It implies that the objects in the video cover a large amount of pixels between two frames, which means that our algorithm will need to be able to follow an object even if this object moves from a lot of pixels. Also, it will mean that because the movement of the object is not constant, our algorithm and method will have to be able to detect an object moving at different speed.

1.3 Object deformation

During the video sequence, more than once the object has some deformation, cause the object revolves in the 3D plan, but not on the picture plan. For example, a rotation of the cup may make the handle disappear for example, or the inside of the cup more, which may not be recognized as close enough from our model. So our method has to adapt itself to deformation. We will see the two ways used to solve this problem later.

1.4 Similar Object Affection

First of all, the two cups are really similar, and we don't want to associate one cup to the other one. The colors are pretty much the same. All white but the inside and a small label on the side of it. Once again we can't really base our model on the label for example because it can disappear with a rotation in the plan. Also, at the end of the video, the two cups are put aside some other coffee cups that are pretty similar. Our method will have to discriminate on something else than the shape of the object, and also be precise so that the two objects are not associated to the same cup for example or to one another when the person exchange them.

1.5 Occlusion

The occlusion is when the object or part of the object is hidden behind the other cups or behind any other objects. In this case, the occlusion occurs multiple times when the man takes the cup from the table to another table or when the man exchange the two cups. Our method needs to distinguish the pair of cups and track them even when the cup are partly occluded. Also our method needs to know when an object is occluded or not in the video anymore.

2 The method

2.1 Which method we should use

All the different difficulties presented in the last part gives us a lot of information o which method we should use for the problem. The first question we asked ourselves was which kind of model we want. Is a generative or a discriminative model better for our video ? In this lab, we choose the discriminative method. Why we don't choose the generative method is because there are some of the other cups. It means we not only want to track the cups which we want to track but also we want to detect that the similar cups are not the cups we want. The discrimintive method is more robust than generative method, that is also the reason we choose it.

We can't base our research on the edges. Even if the background and the object we want to detect are really different, and we don't have much of shadow changes, the cups interior doesn't have much edges. Too similar to other cups and the exterior edges are too simple (may recognize something else). Both of the cups are really white, compared to the background, which is good to distinguish them. Maybe we could choose a method based on histograms. Based on the explanation above, we would like to choose the method has a better performance on similar object affection problem.

We considered mean shift tracking based on histogram ? But it may have a weird behavior when the object is occluded or moves fast. Because the mean shift only adapts a small movement. We think particle filtering will be perfect for us to implement and test and we will also get better results. First, it's simple, fast and we can add some features really easily. Also it's really robust for scale change, translation and rotation problems and we have these challenges in the video sequence. But if here we use particle filtering method with histograms to implement and test on the given video sequence, we could also end up tracking other similar cups and be mix up the association when the person exchanges the two cups .

2.2 Particle Filter Tracking

Now that we explained why we chose this method, we will explain how it works. First of all, the model, the model is based on the histogram of the first box of the first picture. The two cups still have different colors, so they will be discriminated by this feature in the rest of the algorithm.
Then, in the next pictures, we create particles. Basically, particles are boxes around the previous one created at random. Both the position and the size of the new boxes are different and chose at random compared to the previous one, this property helps countering the fast movement or fast scale changes of the object and also the non continuous movement of the objects. We can see that in the next picture :

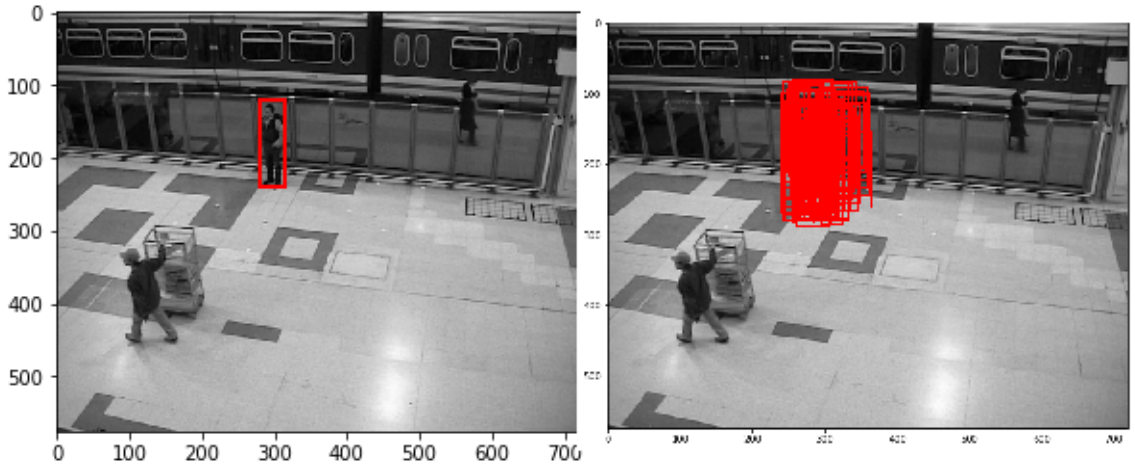

Fig. 1 – Example of particles created (left : first picture, right particles (in red))

In some applications, if the object doesn't change it scale too much and moves inside the video plan we may decide to remove the fact that the size is changed at random. It is not the case in our problem, so we decide to keep the scale

change.

Now that we have different particles created at random, we need to calculate their histograms and compare it with the histogram of the first box created. We use the Bhattacharyya metric to compare the histograms. The Bhattacharyya metric is a probabilistic metric, if a result is close to 1, it's more likely to be the same histogram ; if the result is close to 0, it's a different histogram. We can deduce from how common are both histograms if the object is or not included inside the particle and how much of the object is inside the particle. In the end, we choose the particle with the lowest distance, or the highest probability to contain the object completely. If we use a graph to represent the evaluation of the particles :

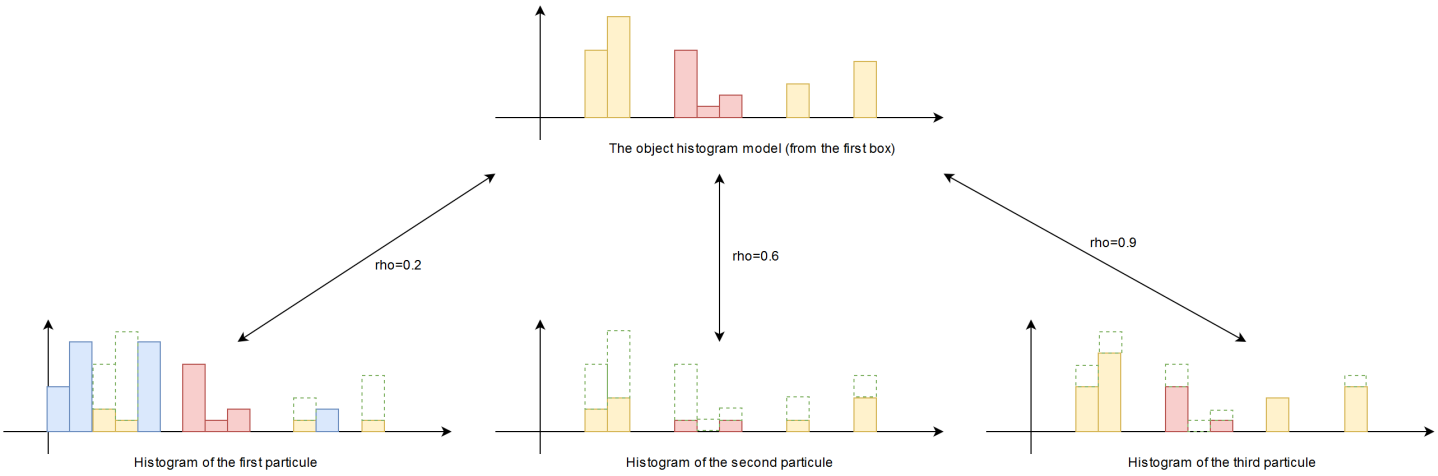

Fig. 2 – Example of histograms for 3 different particles

On the previous diagram we can see a lot of phenomenon in the tracking problems. First of all, in the histogram of the object, or the model taken for the rest of the tracking. We can see some yellow and some red levels. The yellow levels corresponds to the object itself, whereas the red objects represents the background of the object. In fact, we base our model on a rectangle around the object but in most of cases the object is not a rectangle hence some of the background is in the rectangle hence some of the background is evaluated in the model. Which means we can't really have a precise model because the background of the object may change during the tracking. We could have a really precise model if we isolate the object for our model, but it will take time to isolate the object's pixels in the beginning and we want a generic method that only requires a rectangle in the beginning that could be done really fast by a person compared to isolate every pixels of the object. Furthermore, to isolate the object, we should either choose to fill in the background in the rectangle with a color that is not represented in the object or find every pixels that belongs to the object if we want a strong model. Then we have three different cases of particles, that gives us a lot of information. In the first particle, we have some blue levels in the histogram that represents a different background. Also the green dots levels represents in all the diagrams the difference between the model and the particle levels. In this histogram, to understand the distance, we can sum up the area from both the green dots levels and the blue levels and we will obtain a value that will represent the distance. Then from this value we can create a probability that will represent if the histogram represents the same objects. Because the two histograms are really far from one another, we can guess that this particle doesn't contains much of the object, that it must contain a large part of the background and a background we didn't have before. On the second and third particle we can see a different phenomenon, this time the rectangle contains the object, but it contains different scale of the object. From the area of the green dots levels in both case, we can deduce that the third particle's histogram contains more pixels of the object than the second one. Now, if we represents the three particles in a graph we may obtain a result as follows :

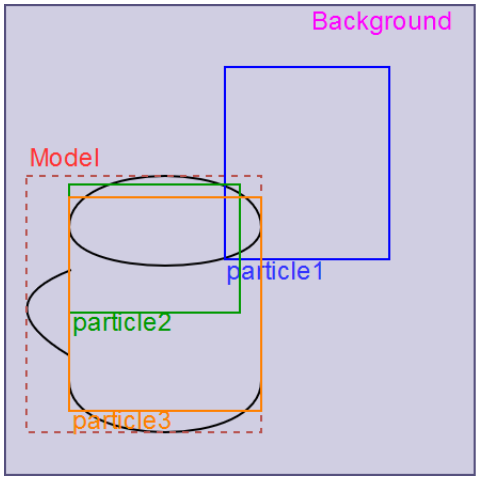

Fig. 3 – Representation of the 3 previous particles

Finally, in this case, we would have chosen the third particle because it represents the object better than the other particles.

Now our question is, what happens when the object is obstructed ? Well, at first we can think that if the object is only partially obstructed the best particle should contain a part of the object, so the result is not so bad. But what if the object is totally obstructed, what if we can't find the object in the picture anymore ? Well from our algorithm right now, we will still search in the picture and use the best particle's histogram. Which will be used in the following pictures to detect the next object. Thus, we may never find the object once again if we start to track somewhere else. So, we decided to change a little bit the algorithm, we added a minimum probability that allowed us to track nothing if no particles where close enough from our model. Basically, if the object is too occluded or if the object isn't in the picture anymore, we will detect nothing. The issue is that the object may appear once again later and might be occluded only on a few pictures, but might appear in a different place. If we are following a pedestrian walking and a truck parks itself in front of a pedestrian, the pedestrian will appear again, but once he passed the truck completely. We decided that if we didn't detect the object on a picture, we needed to search on a greater scale, but still from the last position the object was detected, it's a bit of a "spray and pray" technique, but because we do everything at random, with enough iteration we should with a bit of luck find the object once again.

Now, concerning the object deformation, we had to make a choice. The choice was about whether the object changes too much during the video, do we need to change the model too during the video. In tracking, the update of the model is a well known dilemma. Because, if you keep only one model and your object changes during the video and your algorithm is not flexible enough, you will loose the object. But, if your algorithm is too flexible your results will most likely be inconsistent or not precise. Now, if your object changes it's shape, you may decide to update the model. The problem is that when you update the model, you may start to track something else, and you have to choose when to update it. **To be more concise, the dilemma is : If you don't update your model you are pretty much sure you will loose the object really fast, if you update it you may track it for a longer period of time, but you may also start to track something else in the video**. In this algorithm, we decided to update the model, and we decided to update it on a fixed number of frames. We fix this number using some tests.

3 The implementation and test

3.1 The base of the algorithm

We base our algorithm on the Particle Filter Tracking method given in class. First of all the algorithm given in class allows you to track only one object. In our algorithm we will have to track multiple objects, so we will have first to track the objects one by one and then associate them to the right object. The global algorithm to track objects using a simple Particle Filter Tracking is the following :

```
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
import random as ran
```

At first we import every library we will need in the rest of the algorithm.Then we define some functions :

```
def CompareParticles(hist1, hist2):
    """
    Define a weight of comparison between a particule's histogram and the model's histogram.

    Steps : Basicaly we calculate the distance between two histograms.
    (bhattacharya so it is reprensented as a probability)
    Then we return a value between 0 and 1.
    The higher the value is, the more probable the histogram is the same.
    """
    return 1 - cv2.compareHist(hist1, hist2, cv2.HISTCMP_BHATTACHARYYA)

def CalcHisto(image, bbox):
    """
    Calculate the Histogram of a picture in a certain area (defined by a rectangle).

    Steps : We define the region of interest and the mask.
    Then we calculate the histogram.
    """
    roi = (bbox[0], bbox[1], bbox[0]+bbox[2], bbox[1]+bbox[3])
    mask = np.zeros((image.shape[0],image.shape[1]), np.uint8)
    cv2.rectangle(mask,(roi[0],roi[1]),(roi[2],roi[3]),255,-1,8,0);
    return cv2.calcHist([image],[0],mask,[64],[0,256])

def GenererParticules(Particules, nbparticules, mvt = 20, scl = 20):
    """
    Generate the particules, at random around the object, at random for the scale of the rectangle.

    Commentary : We removed one of the mode, each time we change the bbox to the previous one.
    """
    NouvParticules = Particules
    for i in range(1,nbparticules):
        part = [(Particules[0][0]+ran.randint(-mvt,mvt), Particules[0][1]+ran.randint(-mvt,mvt),
    Particules[0][2]+ran.randint(-scl,scl), Particules[0][3]+ran.randint(-scl,scl))]
        NouvParticules = NouvParticules + part
    return NouvParticules
```

The first function **CompareParticles** calculates the weight of the particles, basically it's a function that compares two histograms, one from the particule and the other one from the model itself. The second function **CalcHisto** calculates the histogram of a picture, inside a rectangle (particle or model). And the last function **GenererParticules** is used to create the different particles at random around the first box or around the best previous box (using ran.choices).

```
# We import the pictures
image1 = cv2.imread('C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/2012/dataset/baseline/
    PETS2006/input/in000216.jpg',cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/2012/dataset/baseline/
    PETS2006/input/in000219.jpg',cv2.IMREAD_GRAYSCALE)

# We chose which one of the three person we will follow
bbox = (279, 120, 36, 120) #box that represents the object in the beginning of the algorithm
#bbox = (83, 300, 70, 160)
#bbox = (547, 90, 45, 100)

nbpart = 80  #Number of particules (rectangles created at random)
mouvement = 40  #Max mouvement on X and Y (in pixels)
modele = CalcHisto(image1, bbox) #Histogram we will use for the comparison (histogram of the inside
    of the rectangle)
particules= GenererParticules([bbox],nbpart, mouvement) #We generate the particules

#We calculate the weight of each particules compared to the original picture.
poids =[]
for p in particules:
```

```
    candidat = CalcHisto(image2, p)
    dist = CompareParticles(modele, candidat)
    poids.append(dist)

#We calculate which rectangle corresponds to the best weight and we plot it
p = particules[poids.index(max(poids))]
fig,ax = plt.subplots(1)
ax.imshow(image2,cmap = plt.get_cmap('gray'))
rect = patches.Rectangle((p[0],p[1]),p[2],p[3],linewidth=2,edgecolor='r',facecolor='none')
ax.add_patch(rect)
plt.show()
```

Then we import every pictures, we define the rectangle, we define the number of particles, the max movement and the model. Then we generate the particles, we calculate the weight for each particles. Finally we find which particle is the best one.

3.2 First modifications

Because we have a problem with a large scale of pictures, first we need to import all the pictures and do a for loop on the pictures. We also need to do a for loop on each object and import them. Then, we just modified the **GenererParticules** function so that it will only have the first part (first if), after each iterations on the pictures, we change the rectangles of the objects with the new one detected. Each of those modifications can be coded as follows :

- Importation of the init file and initialisation of the boxes :

```
init = open("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/2012/dataset/baseline/PETS2006/init.txt","r")
#init = open("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/TP3_data(final)_COPY/init.txt","r")
#init = open("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/TP3_data(final)/init.txt","r")

bboxes_init=[]
for line in init:
    B=line.split()
    bboxes_init.append([int(B[2]),int(B[4]),int(B[3])-int(B[2]),int(B[5])-int(B[4])])

N_obj=len(bboxes_init)
## We chose which one of the three person we will follow
#bbox = (279, 120, 36, 120)
##bbox = (83, 300, 70, 160)
##bbox = (547, 90, 45, 100)
image1=imlist[0]
fig,ax = plt.subplots(1)
ax.imshow(image1,cmap = plt.get_cmap('gray'))
for i in range(N_obj):
    bbox=bboxes_init[i]
    rect = patches.Rectangle((bbox[0],bbox[1]),bbox[2],bbox[3],linewidth=2,edgecolor='r',facecolor='none')
    result_table.append([1,i+1,bbox[0],bbox[0]+bbox[2],bbox[1],bbox[1]+bbox[3]])
    ax.add_patch(rect)
plt.show()

init.close()
```

- The importation of every pictures

```
filenames = [img for img in glob.glob("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/2012/dataset/baseline/PETS2006/input/*.jpg")]
#filenames = [img for img in glob.glob("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/TP3_data(final)_COPY/frame/*.jpg")]
#filenames = [img for img in glob.glob("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/TP3_data(final)/frame/*.jpg")]

filenames.sort() # ADD THIS LINE

imlist = []
for img in filenames:
    n= cv2.imread(img,cv2.IMREAD_GRAYSCALE)
    imlist.append(n)

```

```
12
13 #parameters of images
14 im = imlist[0] #first image
15 height = np.size(im, 0)
16 width = np.size(im, 1)
17 #RGB = np.size(im, 2)
18 N = len(imlist)
```

- The two loops, one on the pictures, one on the object :

```
1 ##############################################################################
2 ####    Loop on the pictures of the file
3 ##############################################################################
4
5 for i in range(1,N):
6     imagetrack=imlist[i]
7     detected = []
8
9 ##############################################################################
10 ####    Loop on the objects we need to track
11 ##############################################################################
12
13     for o in range(N_obj):
14
15         bbox=bboxes[o]
```

Once all of this is done, we use the basic algorithm described before.

3.3 Objects disappearance

The first characteristic that we are confronted to is the fact that some objects may disappear, from the video framing or because it's occluded. We were confronted to this problem because we used the PETS2006 challenge and tried to track two people from the frame 216. But, two of those three people move outside the video framing. And then, we still detect some rectangles that are not the object, just because we always choose the best particle, but a particle with a weight of 0.2 doesn't really look like our object in the first place. So we added a parameter and a condition as follows when we calculate the best rectangle :

```
1 #We calculate which rectangle corresponds to the best weight
2         p = particules[poids.index(max(poids))]
3         if ((max(poids)>=w_lim)and(p[2]>edge_lim and p[3]>edge_lim)):
4             detected.append([p[0],p[1],p[2],p[3]])
5             mvt_list[o]=mouvement
6             scl_list[o]=scale
7         else:
8             mvt_list[o]+=deltamvt
9             scl_list[o]+=deltascl
```

w_lim corresponds to the lowest weight we can accept for the particle. It's a parameter we have to set depending on other parameters such as the number of particles. If the number of particles is too low we can't set it close to 0.8 or 0.9 because it may be hard to be flexible enough. Also, we discovered another phenomenon, which is that we can have a pretty high weight on some really thin but big rectangles. so we decided to place another condition, **edge_lim** is the number of pixels minimum whith which we can accept a rectangle to be the object detected. Finally, if we detect a rectangle it's placed in the list **detected** that will contain every rectangles detected. If we didn't detect any good rectangles for one object, it means we lost the object, so we need to search for a larger scale at the next picture that's why we add **deltamvt** and **deltascl** to the scale and movement for this object. Also, if we detect the object once again, we decide to have the basic movement and scale changes for the next pictures.

3.4 Association

In this part we want to associate each rectangle detected with their correct object. The first part (paragraph in the code) is to calculate each weight for every rectangles compared with every objects. The second part is to associate each objects, going from the best association to the worst. The algorithm is as follow ;

```
    #We calculate the comparison between every objects detected
    association=[]
    indexleft=[]
    poids_compare=[]
    for o in range(N_obj):
        poids = []
        for j in range(len(detected)):
            candidat = CalcHisto(imagetrack,detected[j])
            dist = CompareParticles(modele[o], candidat)
            poids.append(dist)
        poids_compare.append(poids)
        #we set the values that will be used for the association
        association.append([])
        indexleft.append(o)


    #We do the association of each objects.
    poids=[]
    for k in range (len(detected)):
        index1=poids_compare.index(max(poids_compare))
        poids=poids_compare.pop(index1)
        index2=poids.index(max(poids))
        index1=indexleft.pop(index1)
        association[index1]=detected.pop(index2)
        for j in range (len(poids_compare)) :
            poids_compare[j].pop(index2)
```

3.5 Update of the model

Finally we implement the part about the model update, for that we use a function and one part in the loops. The function is the following :

```
def UpdateModel(org_modele,modele, association, image, mouvement, scale):
    """
    Function with the aim to update the appearence of the object.

    Steps : At first we try to find the best rectangle and then we update.
    """
    particules= GenererParticules([association],15*nbpart, 5*mouvement,2*scale) #We generate the
    particules

    #We calculate the weight of each particules compared to the original picture.
    poids = []
    poids2 = []
    for p in particules:
        candidat = CalcHisto(image, p)
        dist = CompareParticles(modele, candidat)
        dist2= CompareParticles(org_modele,candidat)
        poids.append(dist)
        poids2.append(dist2)
    #We calculate which rectangle corresponds to the best weight and we plot it
    p = particules[poids.index(max(poids))]
    p2 = particules[poids2.index(max(poids2))]
    if ((poids.index(max(poids))!=poids2.index(max(poids2)))and((max(poids2)>=0.6)and(p2[2]>edge_lim
     and p2[3]>edge_lim))):
        association=p2
        modele=CalcHisto(image,p2)
    else :
        if ((max(poids)>=0.8)and(p[2]>edge_lim and p[3]>edge_lim)):
            association=p
            modele=CalcHisto(image, p)
    return (modele,association)
```

Basically, this function only do the same but with a lot more particles. Then it tries to find the best particle once again. We compare the particle both with the previous model and the first model. Then we chose the particle with the order of priority on the first model. Finally, we update both the associated rectangle and the model. In the main algorithm, we do the next steps for each frames :

```
    #We update the model and we associate bboxes to the association done.
    upd_nb+=1
    for o in range(len(association)):
        if (len(association[o])!=0):
            if(upd_nb>=upd_lim):
                upd_nb=0
                (modele[o],association[o])=UpdateModel(org_modele[o],modele[o],association[o],
    imagetrack,mouvement,scale)
            bboxes[o]=association[o]
            result_table.append([i+1,o+1,association[o][0],association[o][0]+association[o][2],
    association[o][1],association[o][1]+association[o][3]])
        else:
            result_table.append([i+1,o+1,None,None,None,None])
```

If the number of frame without any update is big enough, we update the model. We also only change the bboxes that were indeed detected and associated. If we didn't detect any rectangle, then the results would be filled with NoneTypes.

4 Results and analysis

First, we set the parameters of the method using a couple of tests to see the results and change them accordingly. We obtain the following parameters :

```
nbpart = 150  #Number of particules (rectangles created at random)
mouvement = 150  #Max mouvement on X and Y (in pixels)
scale = 20 #Max change of scale (in pixels)
#dist_lim = 340 #Limit on the distance between two rectangles detected (in pixels)
w_lim=0.70 #Limit in the weight, if a rectangle is below this the rectangle is not believed to be
    the object
edge_lim=50 #Limit of the length
upd_nb=0
upd_lim=30 #every 30 pictures, we update the model.
deltamvt=60 #if the object disapears, we can move more.
deltascl=30 #if the object disapears, the rectangle scale can change more.
```

We will analyze the results according to the video sequence order with respect of the different difficulties. In all the video sequence, the tracker sometimes gets lost for different reasons. It is partly because of our parameters (when w_lim equals to 0.7, we remove a lot of possible detection) and partly because the method cannot adapt to some situations, like similar object affection.

We can easily see that the method adapts to scale change well and has a good performance on occlusion, we will also see another scale change result at the end of this chapter. When the man is moving the right side cup, the cup becomes bigger and the tracker works well when the scale changed. The man took the cup on the left side, he rotated this cup and occluded part of the cup's body by his hand, the tracker can still find the top of this cup. Because another cup went near this rotated and occluded cup and moved away after, the other cup got lost by the tracker. We can see those three situations in order in the next graph :

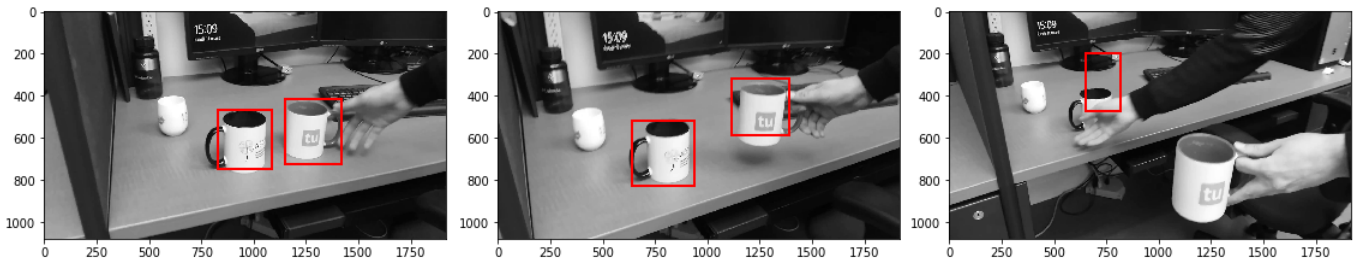

Fig. 4 – Result 1 of the particle filtering

On the right side of the next picture, We found an interesting phenomenon, when the two pair of cups are near each other, the tracker of the right side cup is attracted by the left side cup, which means the particle filtering method

is puzzled by really similar objects and worked bad on similar object affection. Therefore, at the beginning of video, the performance is not really good, because the tracker can only find one of the cups and miss-associate them more than once. We also talked about similar object affection on the previous paragraph. On the left side and middle side of the picture, we found that the handle of the cups disappeared, the tracker can still find one of the cup, so it proves that particle filtering method has not a bad performance on object deformation. We can see those three situations in order in the next graph :

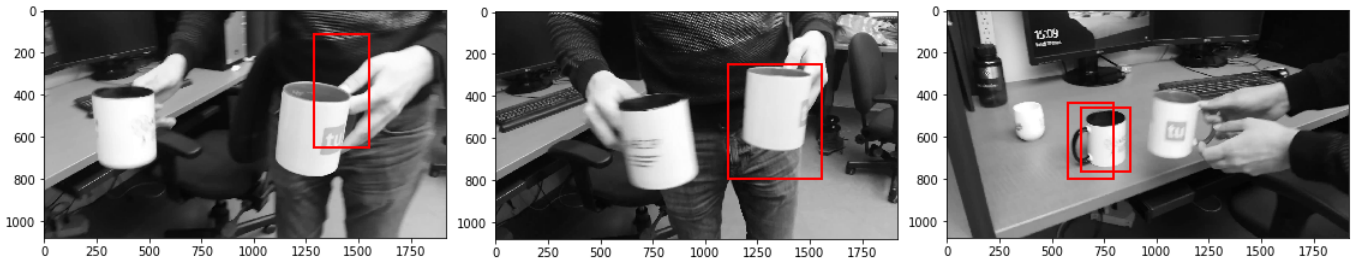

Fig. 5 – Result 2 of the particle filtering

We were happy to find out that when the tracker lost the object, it can find the object later. On the middle side of the next graph, the tracker found part of the left cup although this cup was a bit of blurry in the previous frame (we could see it from the left side of the next pictures). When one cup is totally hidden by another cup, only one cup can be partly detected by tracker and hidden cup's tracker totally got lost. But it cannot prove that particle filtering method works bad at occlusion problem, it's impossible to track a totally hidden object.

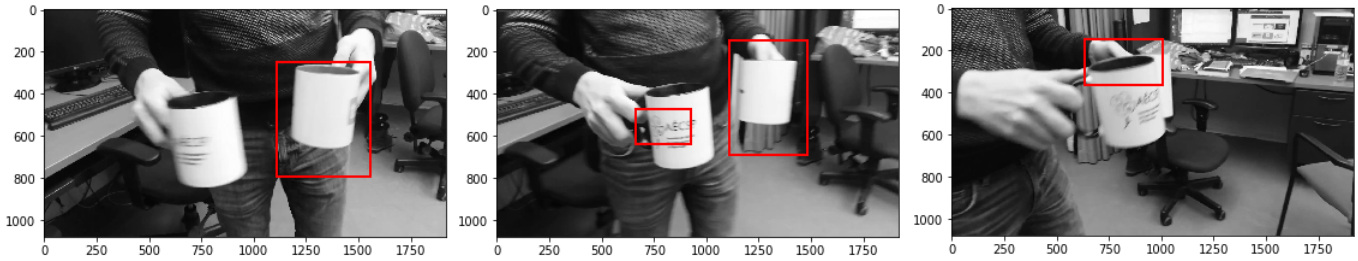

Fig. 6 – Result 3 of the particle filtering

The tracker gets totally lost after one cup hides the other one, it mistook the man's hand as object, maybe it is because the gray-color of the man's hand is similar with cup's gray-color. We set the **deltamvt** as 60 and **deltascl** as 30, in case if the object disappears, the rectangle scale can change more to find objects again. When the tracker tried move around the man's hand, finally we can see that it found one object on the right side of result. Also, this part is challenging because as we discussed previously we need to update our model accordingly. If it happens we update our model while the tracker is on the hands of the person, then in all the next pictures we will track the hands and not the cups anymore.

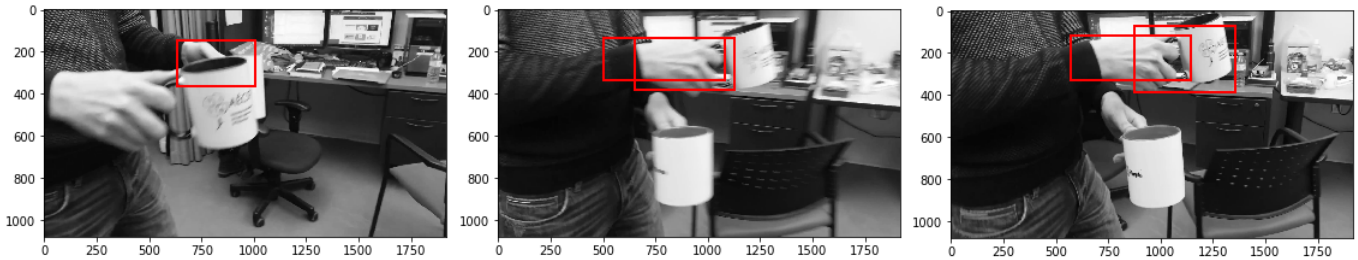

Fig. 7 – Result 4 of the particle filtering

When the cup were placed on the white table, the tracker almost got lost. The background color is similar as cup's color and there has some other coffee cups, the coffee cups and the white table disturbed the tracker to detect objects. The left side of the next graphs shows that the tracker detected one cup and the other tracker detected background. On the middle of the next graph, the tracker got lost, it is either puzzled by other cups or the change of scale is too fast for the tracker to adapt itself. We can see pretty much the same phenomenon and some occlusion problems in the right side picture. We can see all those phenomenons in the next picture :

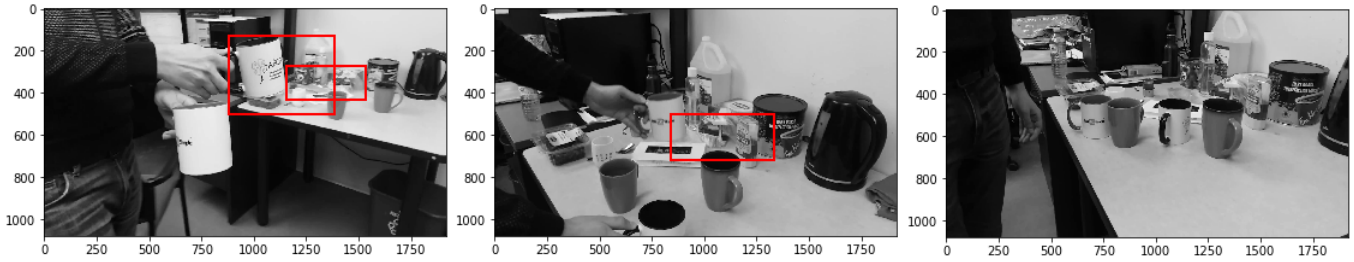

Fig. 8 – Result 5 of the particle filtering

Finally, we are pleasantly surprised that most of the time the Particle Filtering Tracker works well even in hard conditions even if sometimes it's only partially on the object, most of the time the results are encouraging. One hard sequence can be seen in the next pictures :

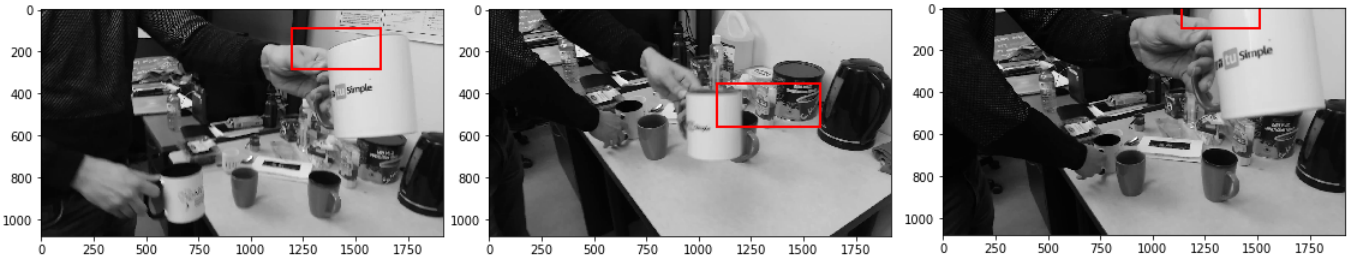

Fig. 9 – Result 6 of the particle filtering

Conclusion

To conclude, we can say that our tracking method is pretty effective, the objects were not tracked on every frames but the results were pretty good. We still have some problems, maybe we update the model too much, maybe the update being triggered by a number of frames. We also think that we could solve the Similar Object Affectation problem using RGB and not grayscales, in grayscale more than one object refers to the same thing. But if we use RGB the object might be more sensitive to light changing and the weight might be pretty different for each color.