```
import cv2
from matplotlib import pyplot as plt
from tools import tracker
import numpy as np
```

```
%matplotlib inline
```

Particle Filter tracking

The filtering is handled by the tracker.track() function, here is how this works:

```
help(tracker.track)
```

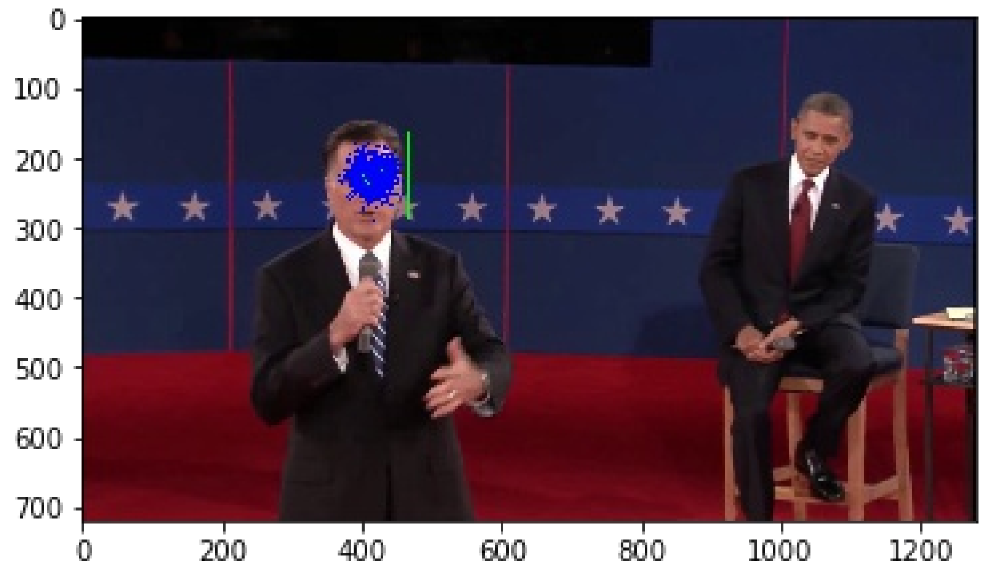
```
Help on function track in module tools.tracker:

track(input, w_cx, w_cy, w_w, w_h, N, sigma, alpha=None, output=None,
display=True, frames_to_save=None)
    apply particle filter to track something in a video
    :param input: the path string of the input video
    :param w_cx: position of the window center (x)
    :param w_cy: position of the window center (y)
    :param w_w: window width
    :param w_h: window height
    :param N: the amount of particles
    :param sigma: the smoothing factor
    :param alpha: the alpha factor used in appearance model update
    :param output: the basename of the output video, it will be extended with
params values
    :param display: boolean indicating whenever to display the frame as they
are beeing tracked
    :param frames_to_save: a list of frames to save, those will be saved as jpg
an returned as result
    :return: the dict of the saved frames
```

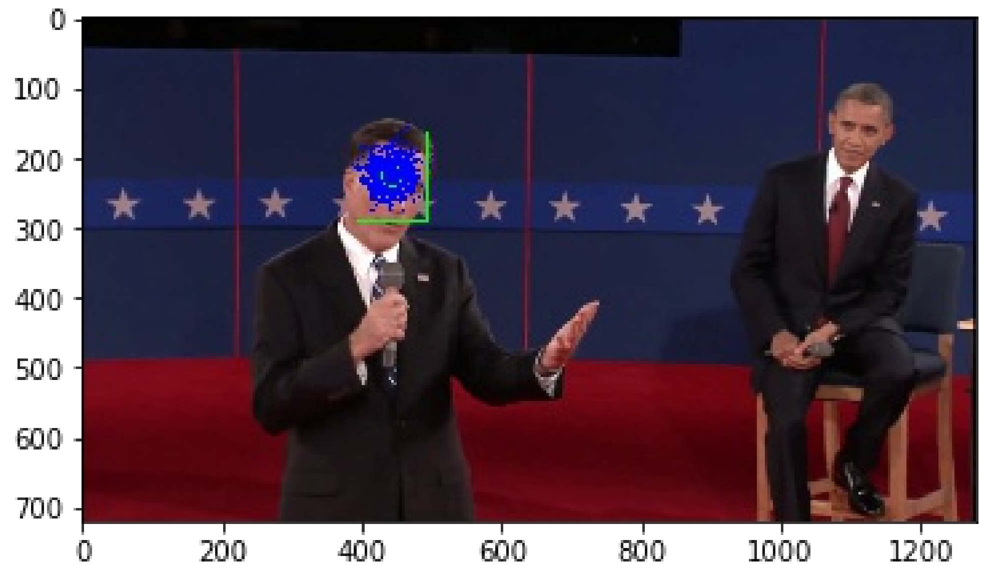
1.1

```
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 64, 51,
N=5000, sigma=1000, alpha=None, display=False, output='Q1', frames_to_save=[28,
84, 144])
```

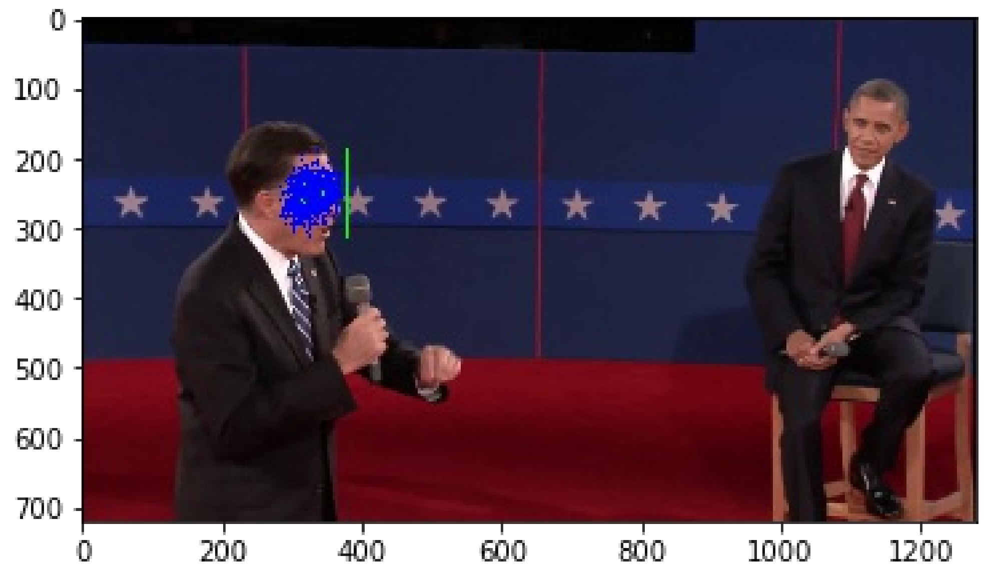
```
for frame in saved_frames.values():
    b, g, r = cv2.split(frame)
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```



0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

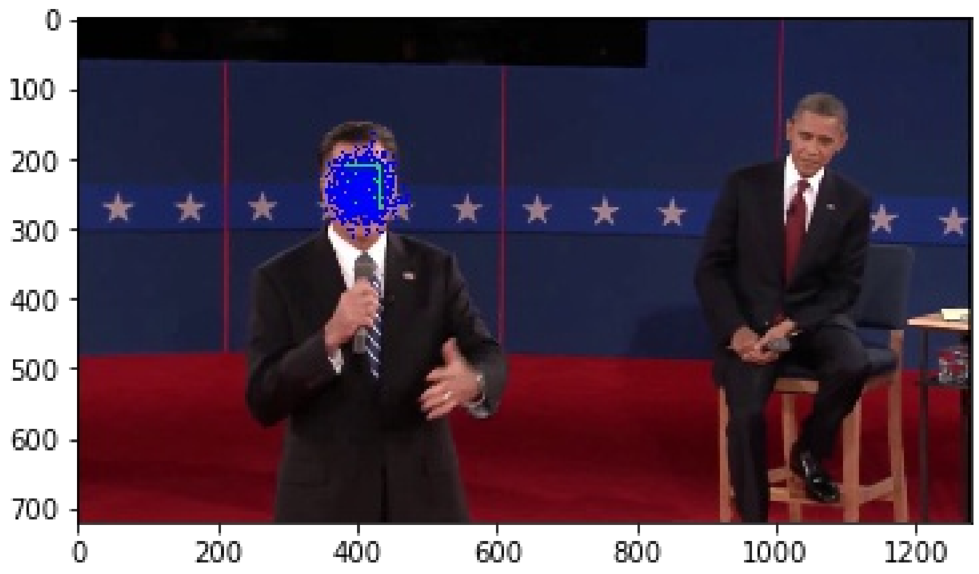


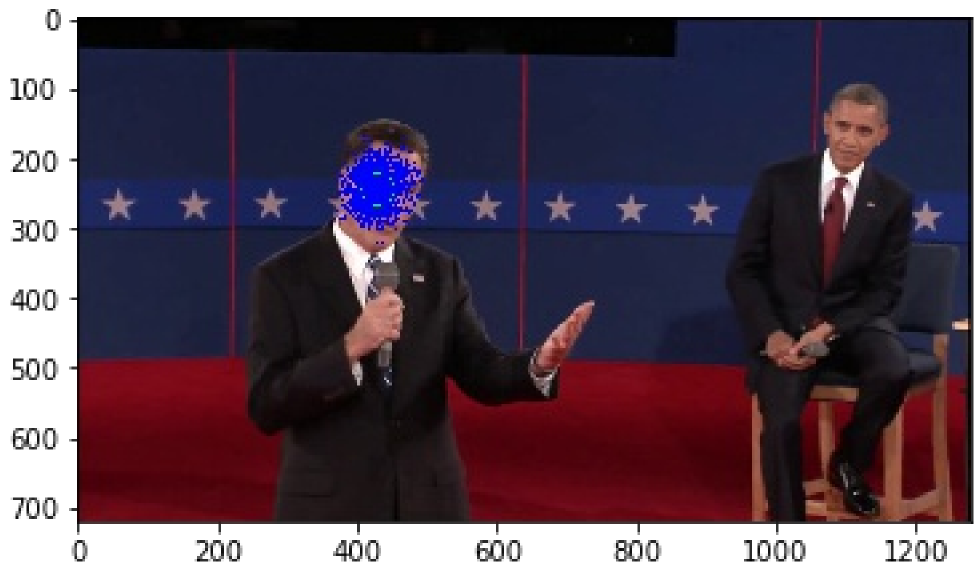
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

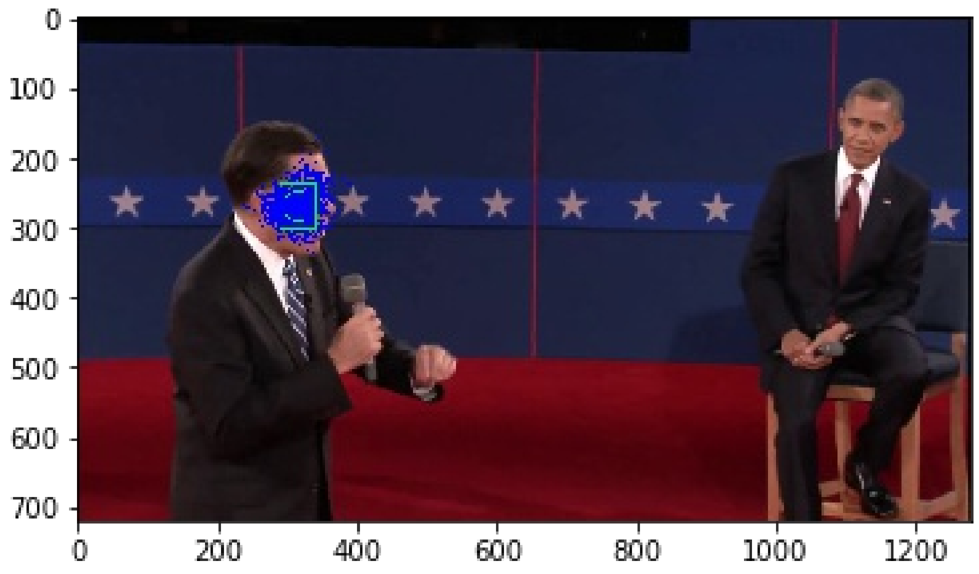
1.2

here are now the results for half and twice the size of the previous window:

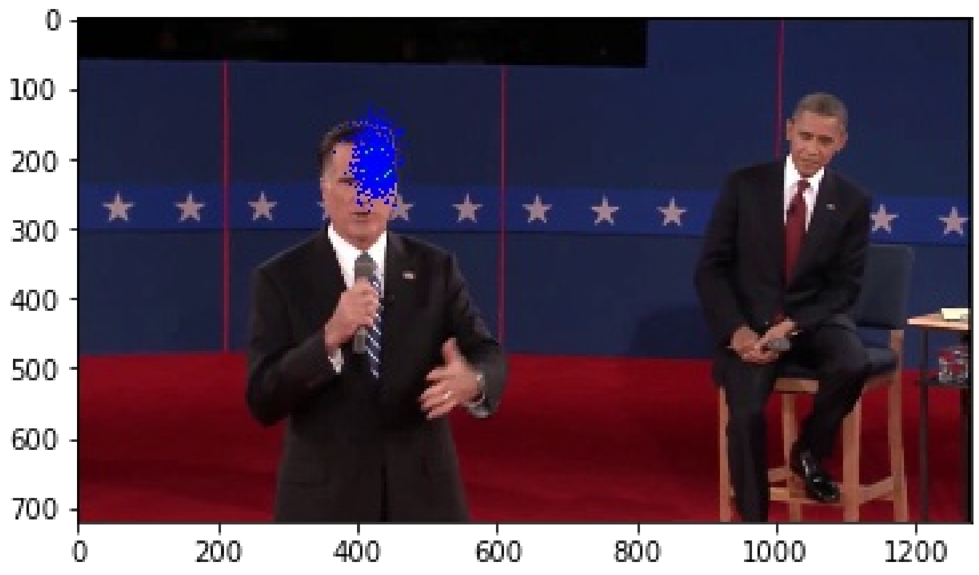
```
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 32, 25,
N=5000, sigma=1000, alpha=None, display=False, output='Q1', frames_to_save=[28,
84, 144])
for frame in saved_frames.values():
    b, g, r = cv2.split(frame)
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```

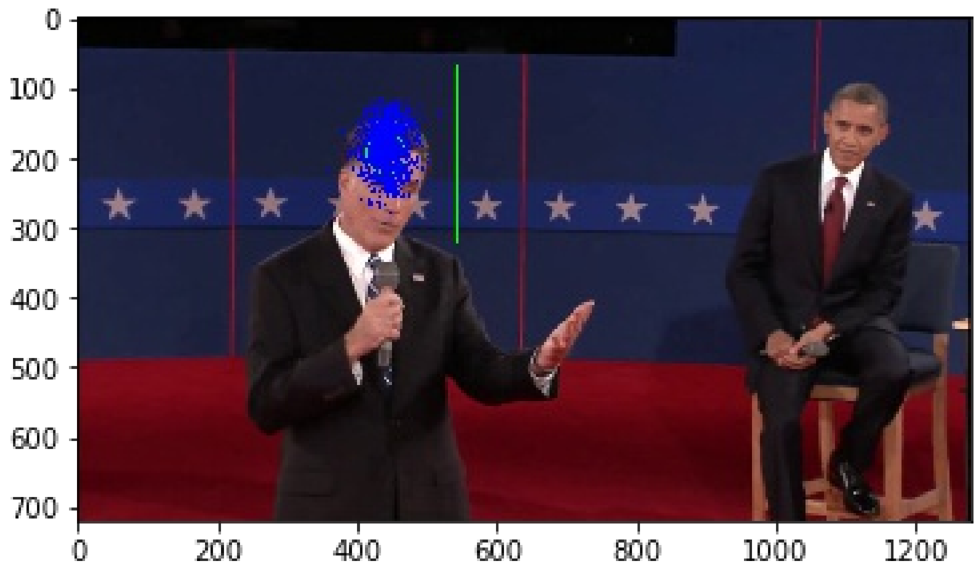


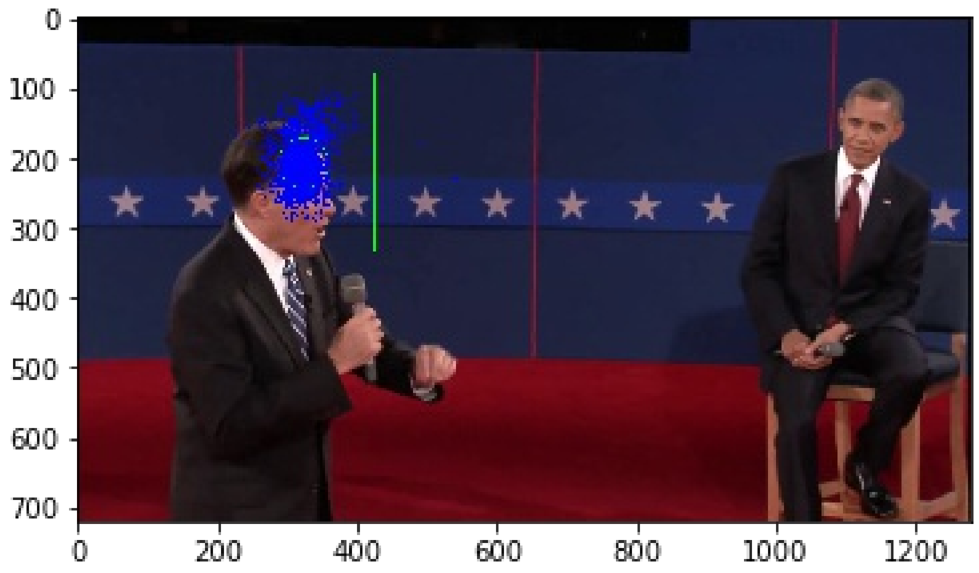




```
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 128, 102,
N=5000, sigma=1000, alpha=None, display=False, output='Q1', frames_to_save=[28,
84, 144])
for frame in saved_frames.values():
    b, g, r = cv2.split(frame)
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```







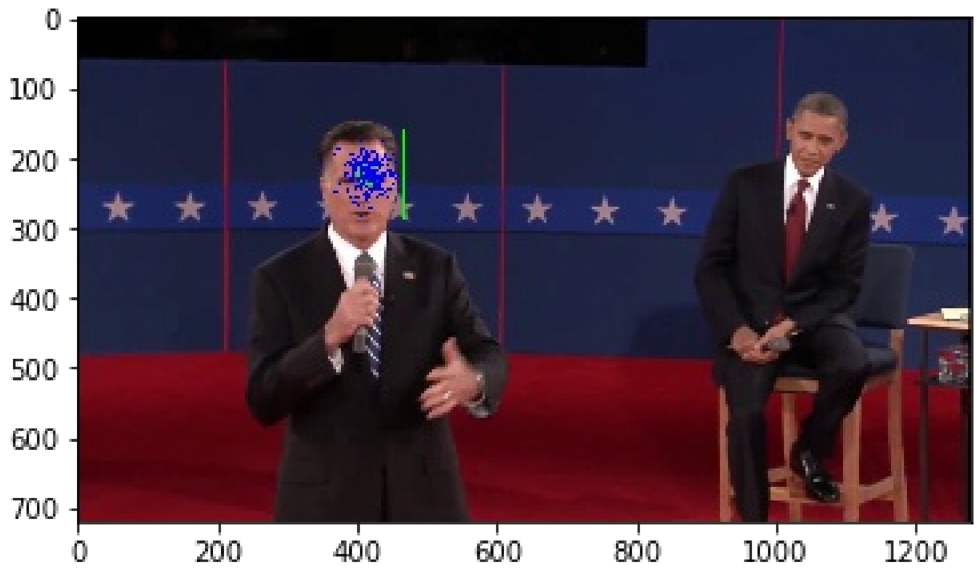
As we can see, small windows allows to track some objects more accurately, but it can lead to some confusion: the skin color of his hand, can confuse the filter has it has a low MSE too. Finally small windows are less affected by the border of the image.

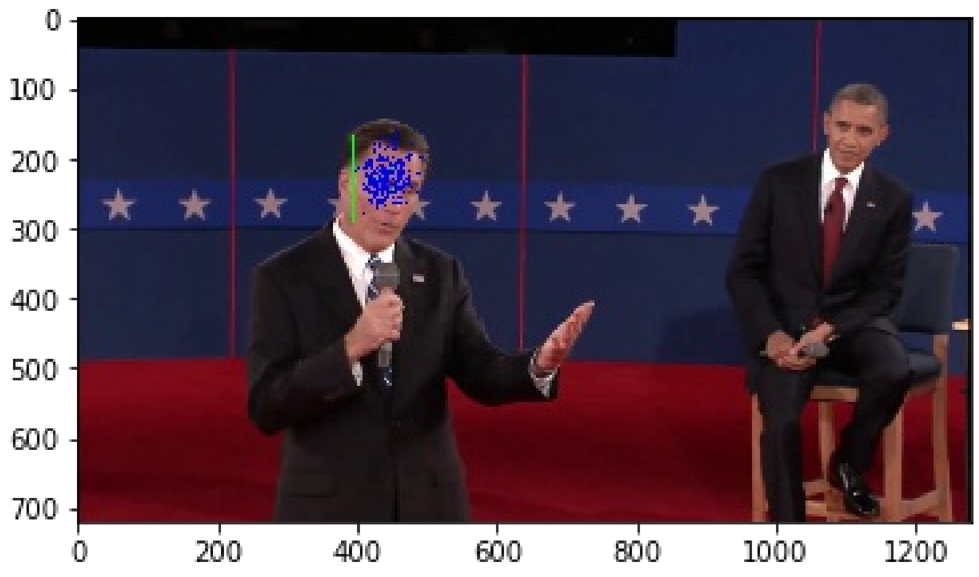
1.3:

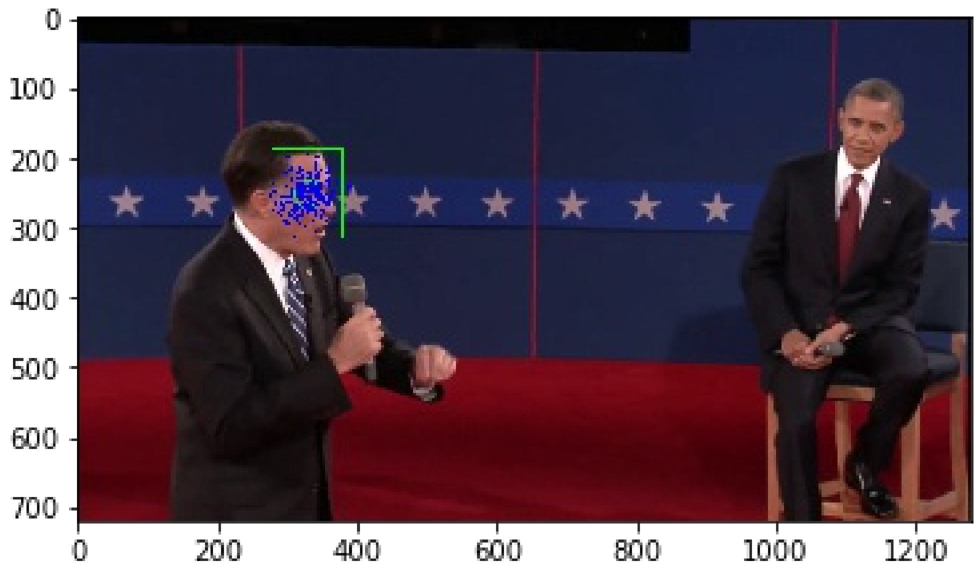
The *sigma* param correspond to the smoothness param: it allows to less penalize suboptimal MSE. By increasing the values, we expect the particles to spread more easily. At the opposite we would expect lower values to make particle stick more to the optima. As i'm workin with SSE instead of MSE, the value of sigma need to be quite big to match the high score values.

```python
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 64, 51,
N=1000, sigma=1000, alpha=None, display=False, output='Q1', frames_to_save=[28,
84, 144])
for i in [28, 84, 144]:
    b, g, r = cv2.split(saved_frames[i])
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```

```
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 64, 51,
N=1000, sigma=5000, alpha=None, display=False, output='Q1', frames_to_save=[28,
84, 144])
for i in [28, 84, 144]:
    b, g, r = cv2.split(saved_frames[i])
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```

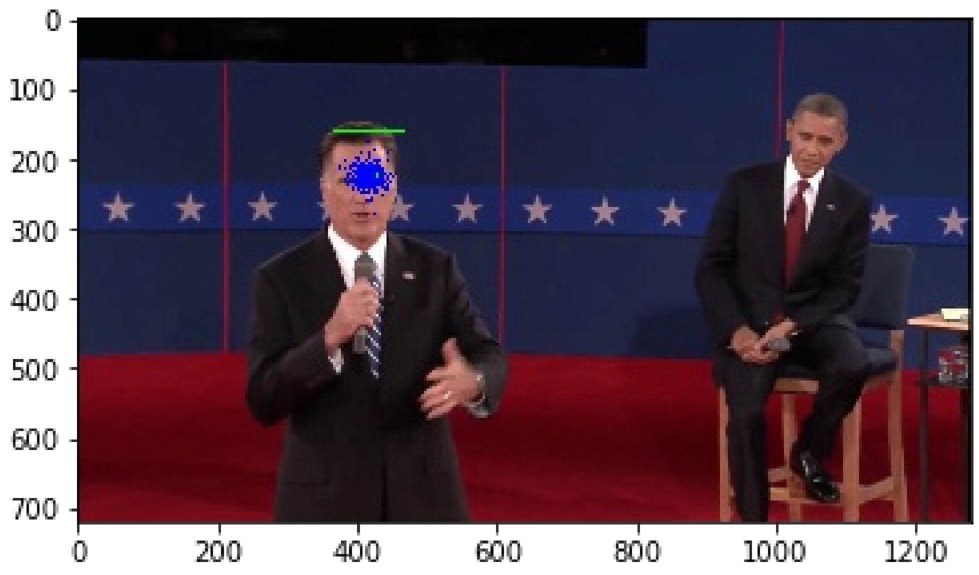


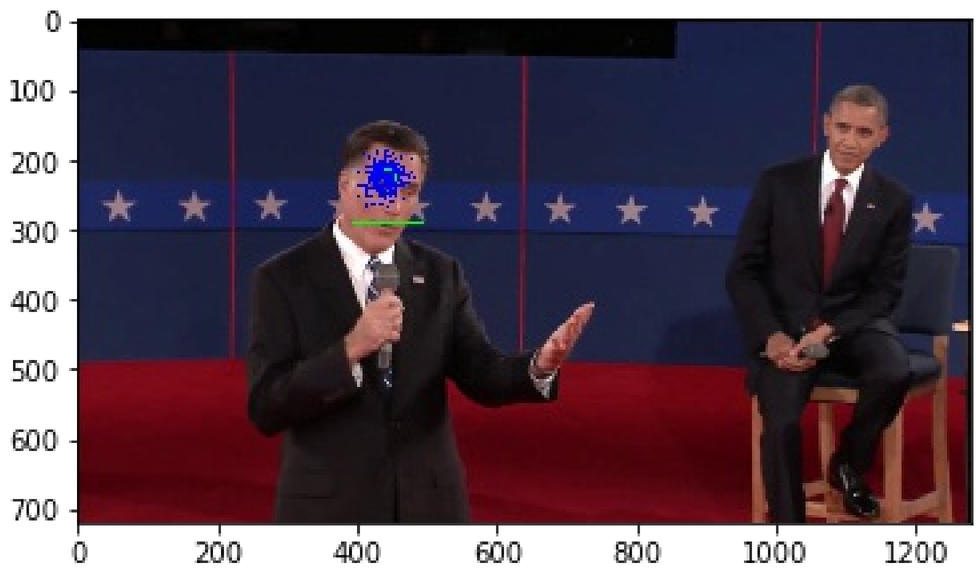


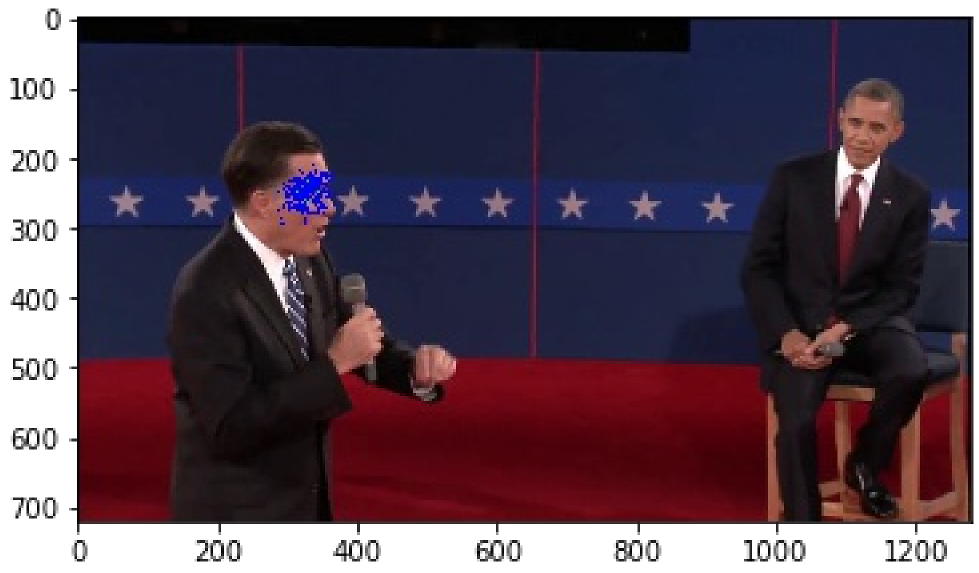


1.4 as i compute the SSE over the entire image using the cv2.matchTemplate function, i could afford using a lot of particles. Here are examples with fewer particles to see the impact.

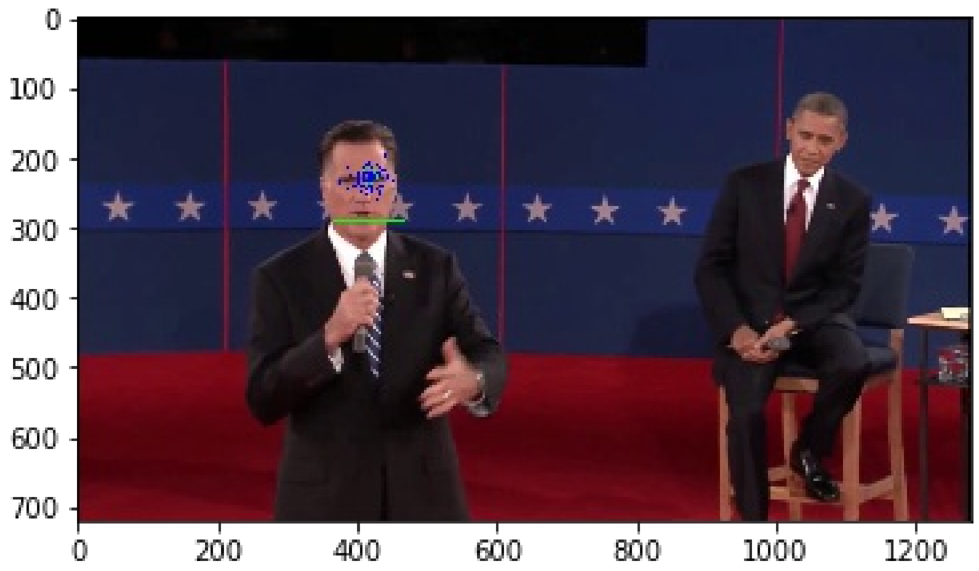
```
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 64, 51,
N=1000, sigma=5000, alpha=None, display=False, output='Q1', frames_to_save=[28,
84, 144])
for frame in saved_frames.values():
    b, g, r = cv2.split(frame)
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```

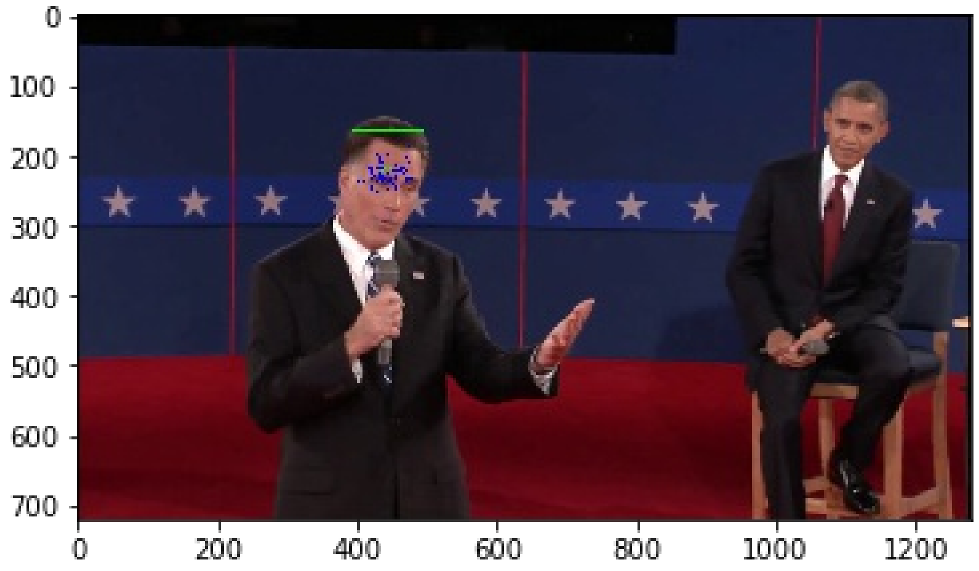


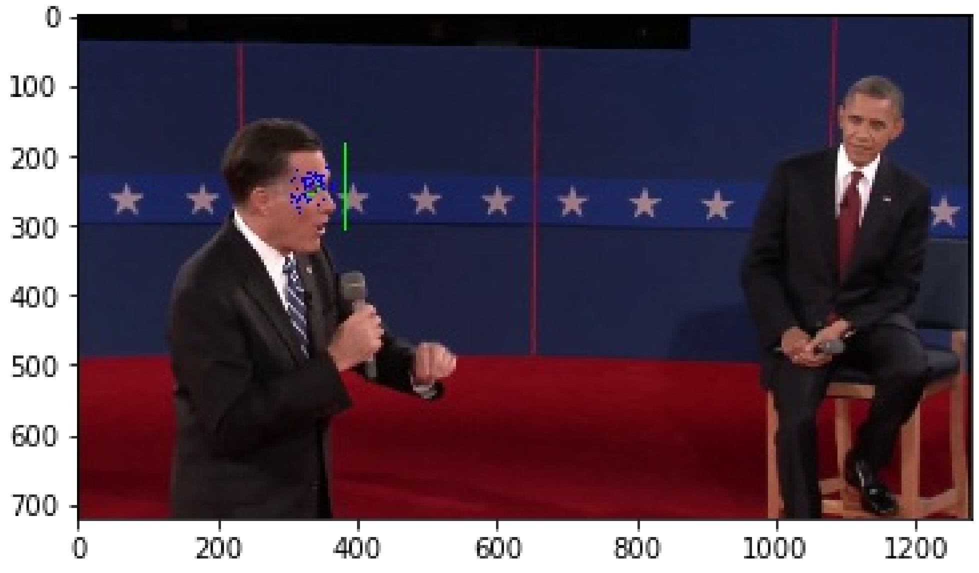




```
saved_frames = tracker.track('inputs/pres_debate.avi', 239, 371, 64, 51, N=200,
sigma=5000, alpha=None, display=False, output='Q1', frames_to_save=[28, 84,
144])
for frame in saved_frames.values():
    b, g, r = cv2.split(frame)
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

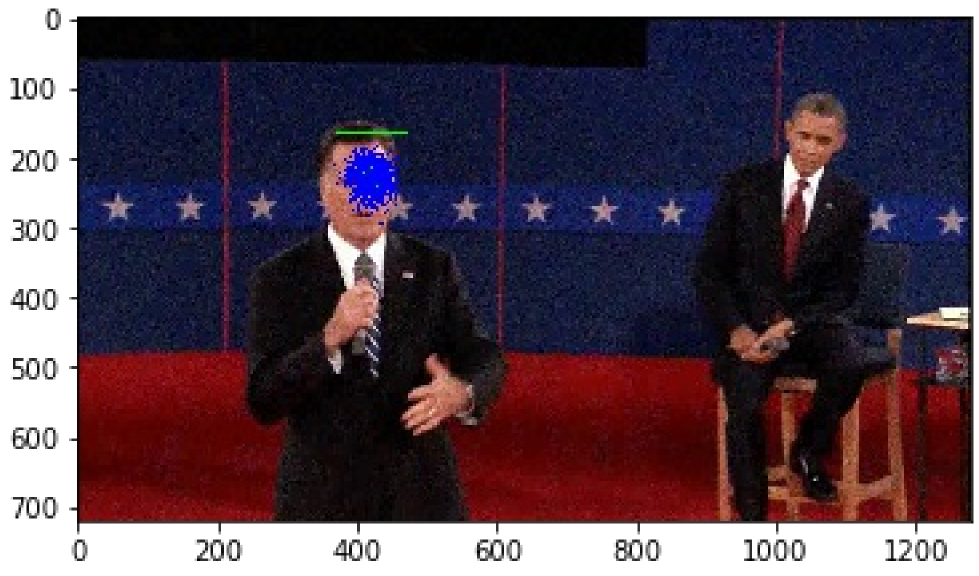

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

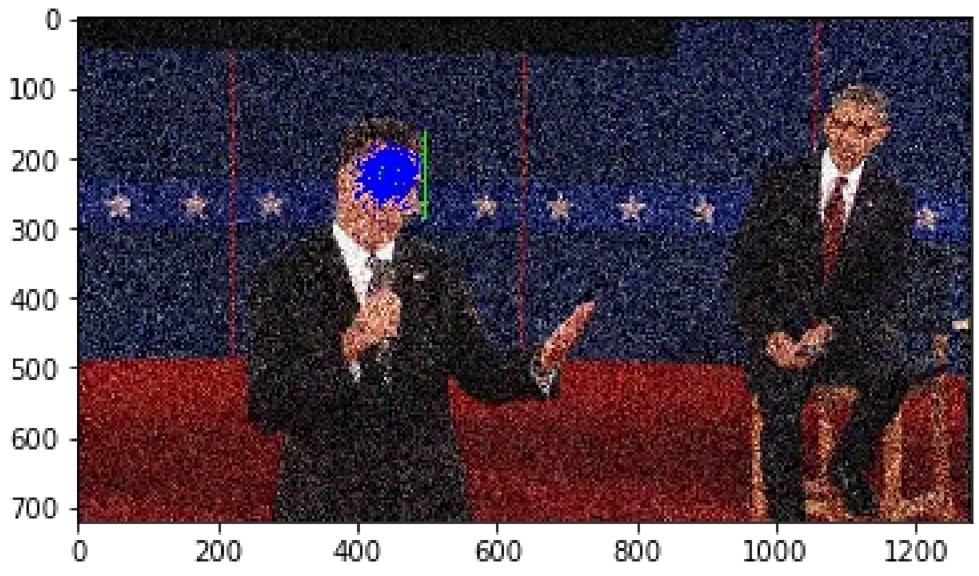

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

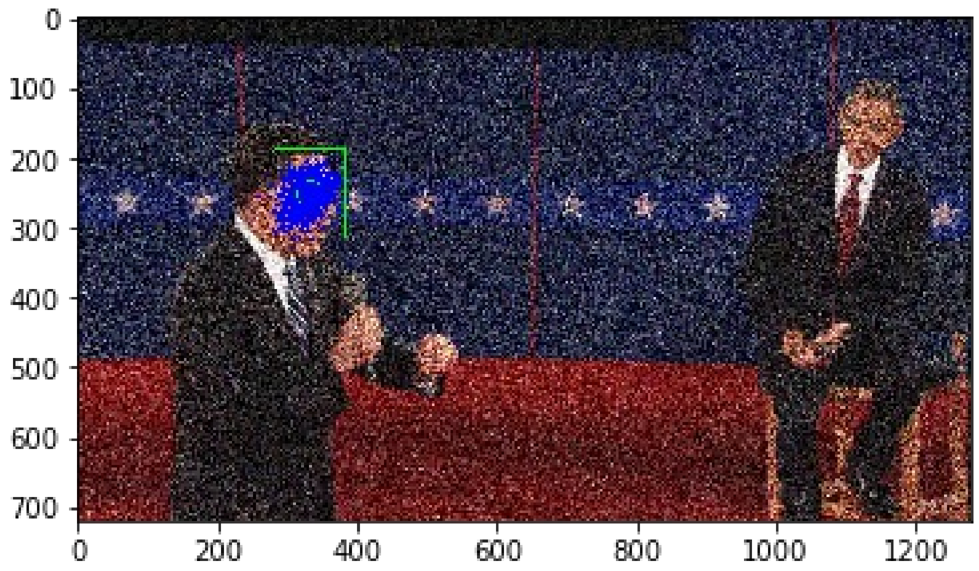
Here we can see that even with a few particles the results are still consistents, altought the windows tends to be a bit more shaky. The computation time is better (on the resampling phase, as the SSE is computed over the entire image)

1.5

```
saved_frames = tracker.track('inputs/noisy_debate.avi', 239, 371, 64, 51,
N=5000, sigma=5000, alpha=None, display=False, output='Q15', frames_to_save=
[28, 84, 144])
for frame in saved_frames.values():
    b, g, r = cv2.split(frame)
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```







As we have many particles, the face is still tracked, but during the noisy perdiods, the particles tends to spread out, uniformly in all the directions. During this period the window is not moving, but the spreading of the particles allow to quickly reframe the face.

2 Appearance model update:

Here is the hand tracking, with model update enabled:

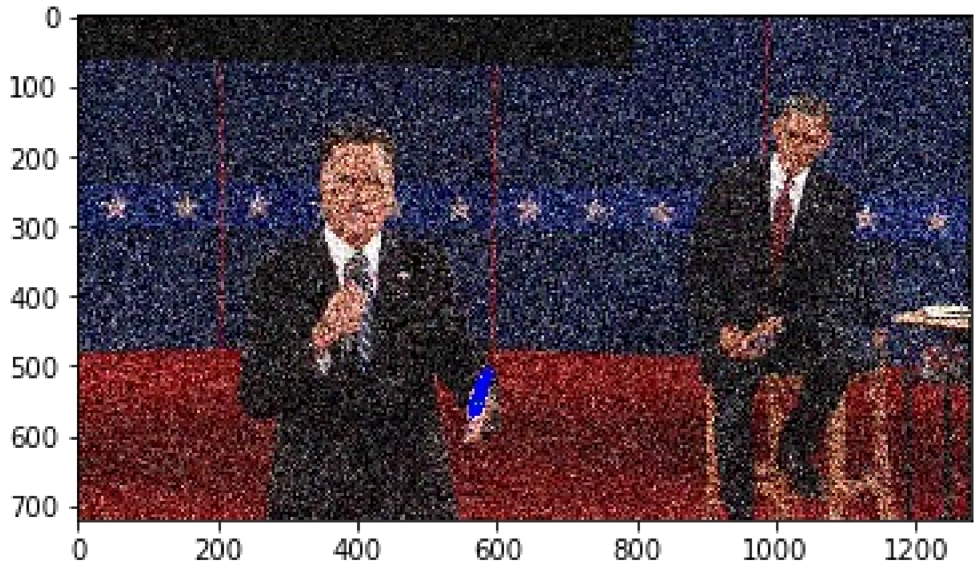
```
saved_frames = tracker.track('inputs/pres_debate.avi', 433, 570, 60, 60,
N=5000, sigma=5000, alpha=0.5, display=False, output='Q2', frames_to_save=[15,
50, 140])
for i in [15, 50, 140]:
    b, g, r = cv2.split(saved_frames[i])
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
for i in [10015, 10050, 10140]:
    plt.imshow(saved_frames[i], plt.gray())
    plt.show()
```

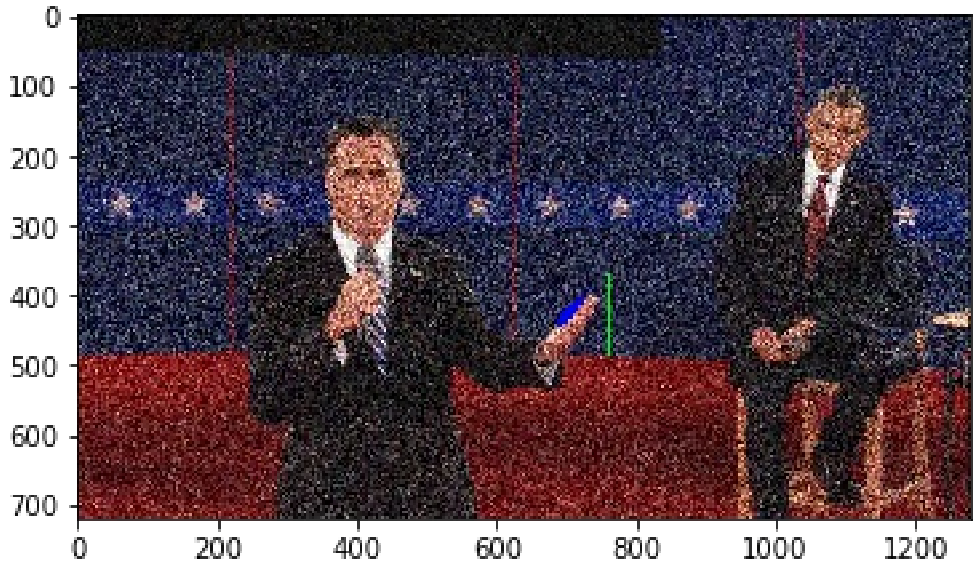
2.2

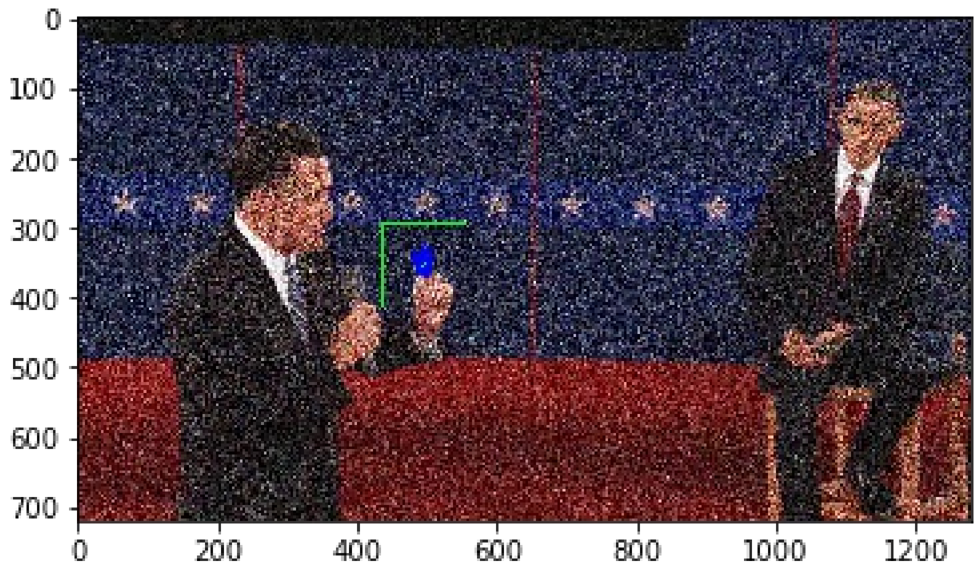
Here the particle filter still work up to frame 140, but the noise force the paticles to be as accurate as possible, this is because i'm using hte max to get update the window. So we need the particle to do not spread, because the more spread they are, the more probable it is to have a false positive max. As we can see the alpha value make the window update very quick so it the matching window takes is noisy, which wouldn't have been the case if alpha->1. But this is a tradeoff as the shape of the hand change quickly.

```
saved_frames = tracker.track('inputs/noisy_debate.avi', 433, 570, 60, 60,
N=5000, sigma=1000, alpha=0.5, display=False, output='Q22', frames_to_save=[15,
50, 140])
```

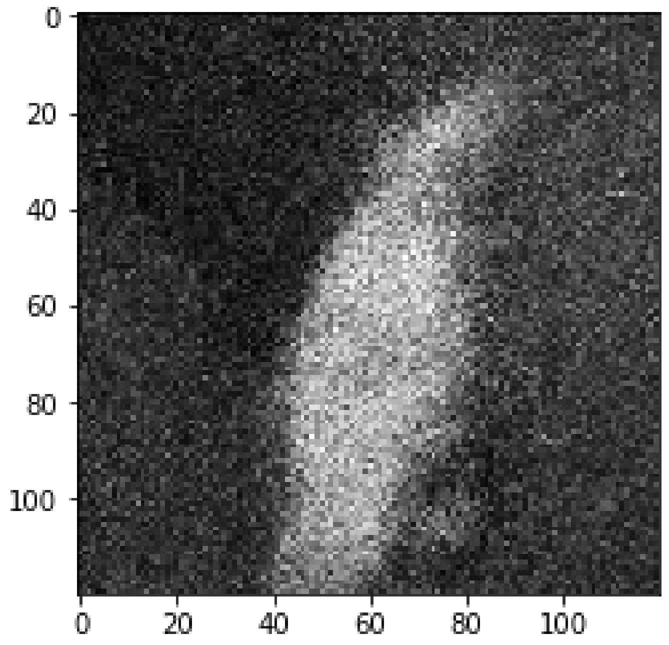
```
for i in [15, 50, 140]:
    b, g, r = cv2.split(saved_frames[i])
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
for i in [10015, 10050, 10140]:
    plt.imshow(saved_frames[i], plt.gray())
    plt.show()
```



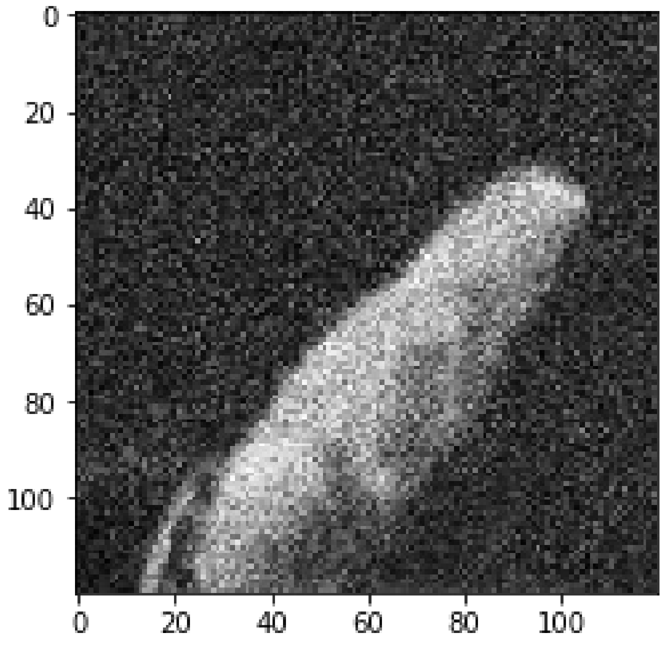


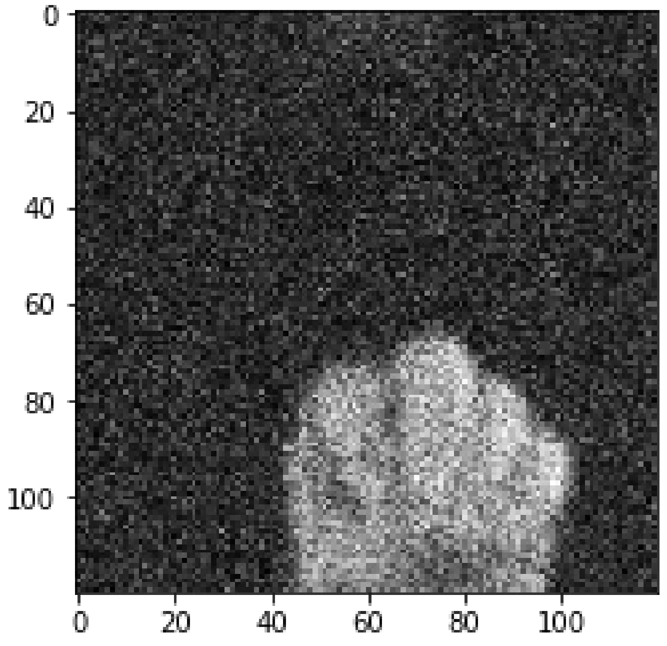


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



0
20
40
60
80
100
0
20
40
60
80
100





3 Incoroporating more dynamics

3.1

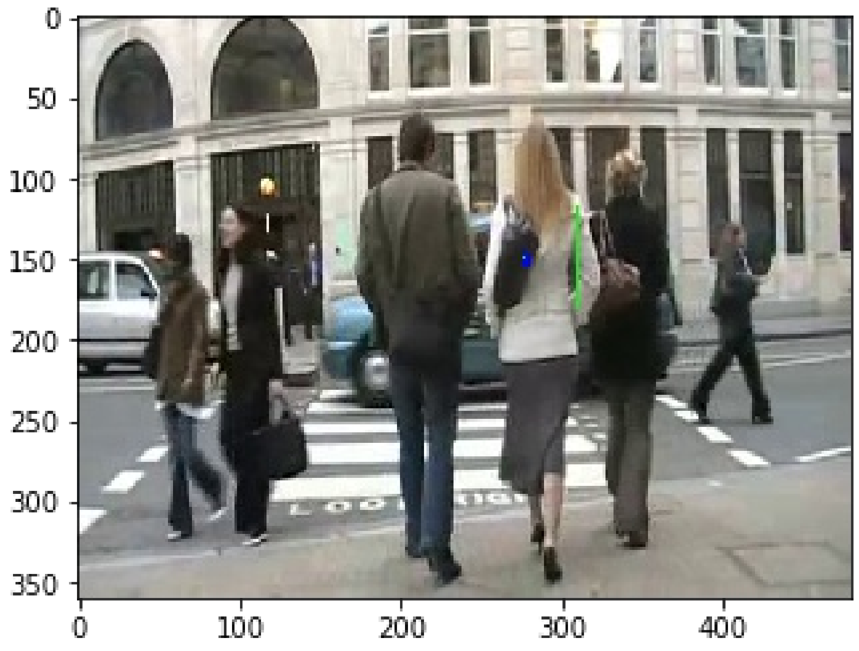
There is two main porblems to deal with:
.1 changing size of the object to match
.2 occlusions

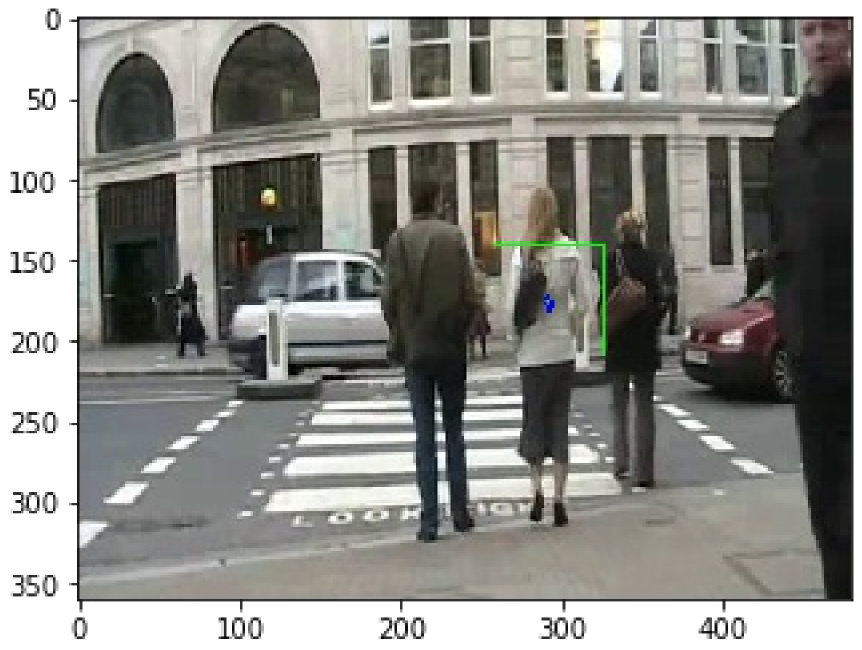
For the first problem, a new dimension has been added to each particle, in this case i choosed to discretize the size, as im using the cv2.matchTemplate function, having a continous scale factor would have required many changes in the code

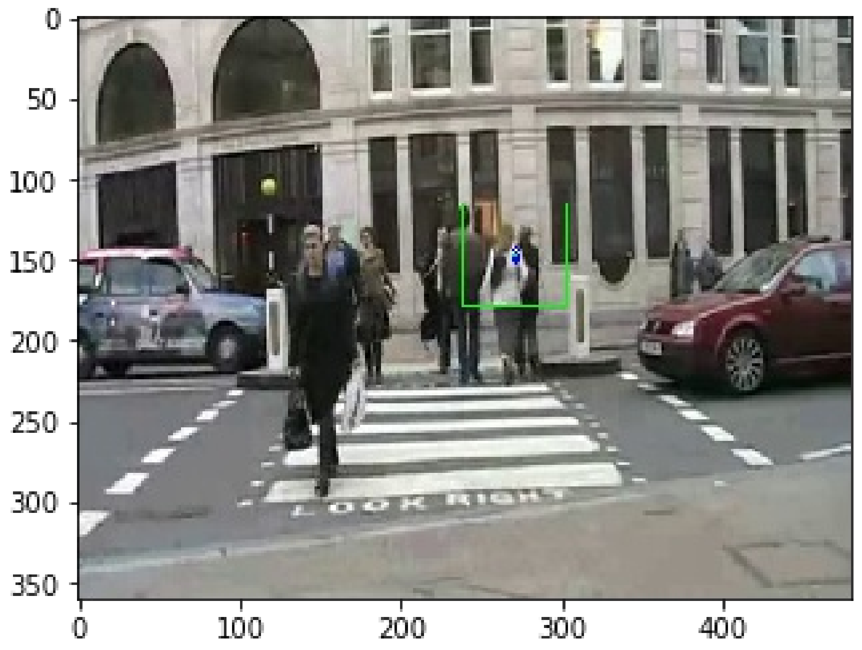
For the second one, i've added a *frozen* mode, which, when enabled, remove the update part, this prevent the windows matching to be lost (because of the IIR). This mode enable if the best match score is lower than a threshold.

```
saved_frames = tracker.track('inputs/pedestrians.avi',
                             140, 240, 50, 50,
                             N=1000, sigma=1000, alpha=0.9,
scale_bins=np.arange(1.0, 0.6, -0.05),
                             display=False, output='PS6_3_1', frames_to_save=
[40, 100, 240])
```

```
for i in [40, 100, 240]:
    b, g, r = cv2.split(saved_frames[i])
    plt.imshow(cv2.merge([r, g, b]))
    plt.show()
```







3.2

As there is a new dimension, the number of particles must be increased respective to the number of bins. In this case the initial amount of particles was already quite big so the was no real need to have more than 1000 particles.