In [1]:

```
import numpy as np
import cv2
import time
```

1 Particle Filter Tracking

1.1 Implement the particle filter and run it on the pres debate.avi clip. You should begin by attempting to track Romney's face. Tweak the parameters including window size until you can get the tracker to follow his face faithfully (5-15 pixels) up until he turns his face significantly. Run the tracker and save the video frames 28, 84, and 144 with the visualizations overlaid.

Output: The code, the 3 image frames with overlaid visualizations, and the image patch used for tracking.

In [2]:

```
def read_bounding_box(filename):
    file = open(filename, 'r')
    for i in file:
        xl, yl, w, h = i.split('  ')
        return int(float(xl)), int(float(yl)), int(float(w)), int(float(h))
```

In [3]:

```
def prediction(X, sigma_d):
    mean = np.array([0, 0])
    sigma = np.array([[sigma_d, 0], [0, sigma_d]])
    d = np.random.multivariate_normal(mean, sigma, 1)
    new_X = X + d
    return new_X[0]
```

In [48]:

```
def measurement(X, template, image, sigma_MSE):
    win_h, win_w = template.shape
    x = int(X[1])
    y = int(X[0])
    image_part = image[y - int(win_h/2) : y + int(win_h/2+0.5), x - int(win_w/2)
: x + int(win_w/2+0.5)]
    if (template.shape != image_part.shape):
        return 0
    diff = np.matrix(template, dtype=np.float64) - np.matrix(image_part, dtype=n
p.float64)
    MSE = 1/(win_h*win_w) * np.sum(np.multiply(diff, diff))
    p = np.exp((-1)* MSE / (2 * (sigma_MSE ** 2)))
    return p
```

In [5]:

```
def update(particles, weights, template, image_t, sigma_d, sigma_MSE):
    new_particles = []
    new_weights = []
    weight_sum = 0
    for i in range(len(particles)):
        p_index = np.random.choice(range(len(particles)), p=weights)
        sample_p = particles[p_index]

        # Sample from prediction
        new_p = prediction(sample_p, sigma_d)
        # weight by measurement
        w = measurement(new_p, template, image_t, sigma_MSE)
        new_weights.append(w)
        weight_sum += w
        new_particles.append(new_p)

    for i in range(len(particles)):
        new_weights[i] = new_weights[i]/weight_sum
    return new_particles, new_weights
```

In [6]:

```
def particle_filter(video_file='pres_debate.avi', txt_file='pres_debate.txt', wi
ndow_scaling=1, nb_particle=10,\
                    sigma_d=10, sigma_MSE=10, frame_output=[28, 84, 144], id_='1
-1',\
                    all_frames=False, video_output='pres_debate_tracking'):
    """
    video_file: source video file
    txt_file: source bounding rectangle file
    window_scaling: for scaling different size of window
    nb_particle: number of particles initialize
    sigma_d: sigma in prediction model
    sigma_MSE: sigma for MSE calculation in updating weight
    frame_output: special frame output, list with numbers
    id: video and images' id
    all_frames: if True, output all the frames in a file named with id_
    video_output: vido output file name. If False, no video output
    """
    pres_debate = cv2.VideoCapture(video_file)
    count = 0

    rect_x, rect_y, rect_w, rect_h = read_bounding_box(txt_file)
    init_particle_center = np.array((int(rect_y + rect_h/2), int(rect_x + rect_w
/2)))

    weights = np.ones((nb_particle)) / nb_particle

    particles = np.random.multinomial(100, [1/2.] * 2, size=nb_particle)
    particles = particles - 50
    for idx in range(len(particles)):
        particles[idx] = particles[idx] + init_particle_center
    ret, frame = pres_debate.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    template = gray[rect_y : rect_y + rect_h, rect_x : rect_x + rect_w]
    if window_scaling != 1:
        template = cv2.resize(template, (0, 0), template, fx=window_scaling, fy=
window_scaling)
    win_h, win_w = template.shape

    count += 1
    for p in particles:
        frame = cv2.circle(frame, (int(p[1]), int(p[0])), 1, (255, 0, 0), 1)
    if all_frames:
        cv2.imwrite('./{}/0.png'.format(id_), frame)

    if video_output:
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        out = cv2.VideoWriter("{}-{}-p{}-mse{}-win{}.avi".format(video_output, i
d_, nb_particle, sigma_MSE, window_scaling),\
                              fourcc, 20.0, (frame.shape[1], frame.shape[0]))
        frame = cv2.flip(frame, 1)
        out.write(frame)

    start_time = time.time()

    while(pres_debate.isOpened()):
        ret, frame = pres_debate.read()
        if not ret:
            break
```

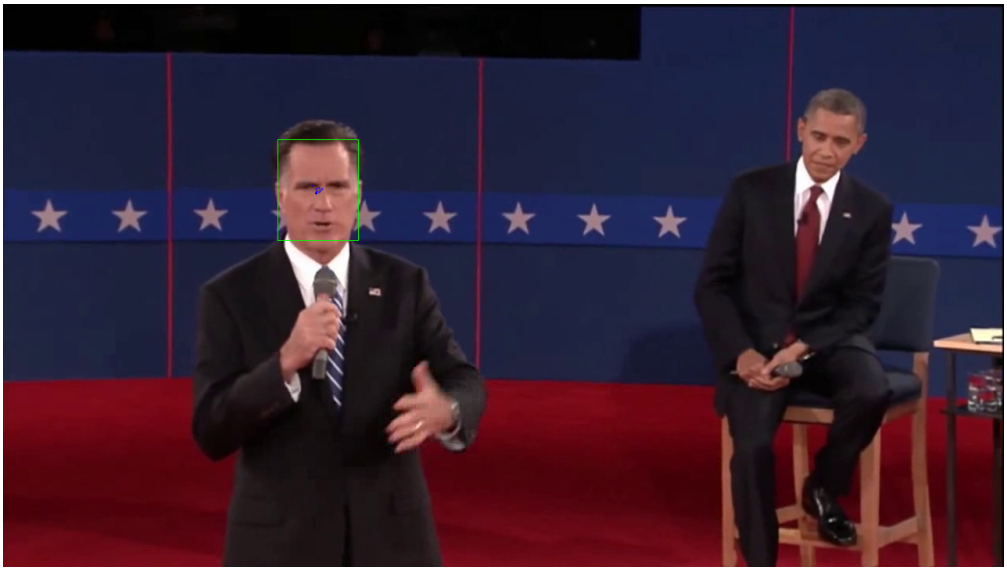
```
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        particles, weights = update(particles, weights, template, gray, sigma_d,
sigma_MSE)

        mean_p = np.mean(particles, axis=0)
        mean_rect_LT = [int(mean_p[1] - win_w/2), int(mean_p[0] - win_h/2)]
        mean_rect_RB = [int(mean_p[1] + win_w/2), int(mean_p[0] + win_h/2)]

        for p in particles:
            frame = cv2.circle(frame, (int(p[1]), int(p[0])), 1, (255, 0, 0), 1)
        frame = cv2.rectangle(frame, (mean_rect_LT[0], mean_rect_LT[1]), (mean_r
ect_RB[0], mean_rect_RB[1]),\
                              (0, 255, 0), 1)
        if all_frames:
            cv2.imwrite('./{}/{}.png'.format(id_, count), frame)
        if count in frame_output:
            cv2.imwrite('ps6-{}-{}-p{}-mse{}-win{}.png'.format(id_, count, nb_pa
rticle, sigma_MSE, window_scaling),\
                        frame)
        count += 1
        if video_output:
            frame = cv2.flip(frame, 1)
            out.write(frame)
        #break
    pres_debate.release()
    elapsed = time.time() - start_time
    print("Elapsed: ", elapsed)
```

In [7]:

```
particle_filter()
```

```
Elapsed:  4.953596353530884
```

10 Particles; sigmaMSE=10; sigma d=10:



1.2 Experiment with different dimensions for the window image patch you are trying to track. Decrease the window size until the performance of the tracker degrades significantly. Try significantly larger windows than what worked in 1.1. Discuss the trade-offs of window size and what makes some image patches work better than others for tracking.

Output: Discussion in the pdf. Indicate 2-3 advantages of larger window size and 2-3 advantages of smaller window size.

In [17]:

```
particle_filter(window_scaling=3, id_='1-2')
```

```
Elapsed:  9.193111658096313
```

Discussion:

I tried to resize the template from 0.3 to 3 times. All the trackers can be tracking the face or part of the face. I think it's because, in this video, the face we are tracking has a distinctive gray-scale color with its background. So if the particle can find the right brightness pattern, it can retrieve the skin and take it as the face.

Advantages of larger window size:

1. Can keep the full face in the detection frame.
2. When there is a change in the face, can still catch a global pattern

Advantages of smaller window size:

1. Running significantly faster than the larger window size.
2. Lower the effect by background

1.3 Adjust the σMSE parameter to higher and lower values and run the tracker. Discuss how changing this parameter alters the results and attempt to explain why.

Output: Discussion in the pdf.

In [14]:

```
particle_filter(sigma_MSE=50, id_='1-3')
```

```
Elapsed:  5.097007751464844
```

Discussion:

I tried with σMSE=3, 10, 15, 20, 50

The particle filter still works well with lower σMSE values (the lowest possible I tried was 3), but the performance starts to degrade after σMSE=20. After the face changes its formation, the particle cannot find the right position; even the face is distinctive with the background. I think it's because a large σMSE makes every possibility similar. So it hardly selects the best particles into the next frame.

Discussion:

1.4 Try and optimize the number of particles needed to track the target. Discuss the trade-offs of using a larger number of particles to represent the distribution.

Output: Optimized particle number and discussion in the pdf.

In [11]:

```
particle_filter(nb_particle=50, id_='1-4')
```

```
Elapsed:  12.591731071472168
```

Discussion:

Running time: 50 particles ~12s 10 particles ~5s 5 particles ~4s

A larger number of particles takes more time to run. But it is more sensitive to the small motion of the face. With 50 particles, we can see the rectangular frame moves with the face motion. A small number of particles run faster but more imprecise.

1.5 Run your tracker on noisy debate.avi and see what happens. Tune your parameters so that the cluster is able to latch back onto his face after the noise disappears. Include varying σMSE. Report how the particles respond to increasing and decreasing noise. Save the video frames 14, 32, and 46 with the visualizations overlaid.

Output: The code, the 3 image frames with overlaid visualizations, and discussion in the pdf.

In [20]:

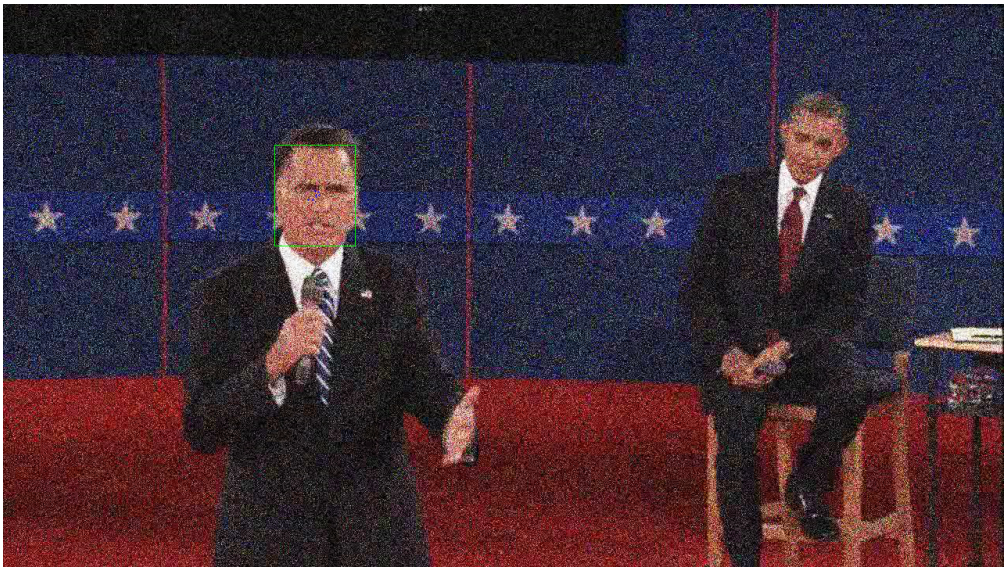
```
particle_filter(video_file='noisy_debate.avi', txt_file='noisy_debate.txt', wind
ow_scaling=1, nb_particle=10,\
                sigma_d=10, sigma_MSE=10, id_='1-5', frame_output = [14, 32, 46
],\
                video_output='noisy_debate_tracking')
```

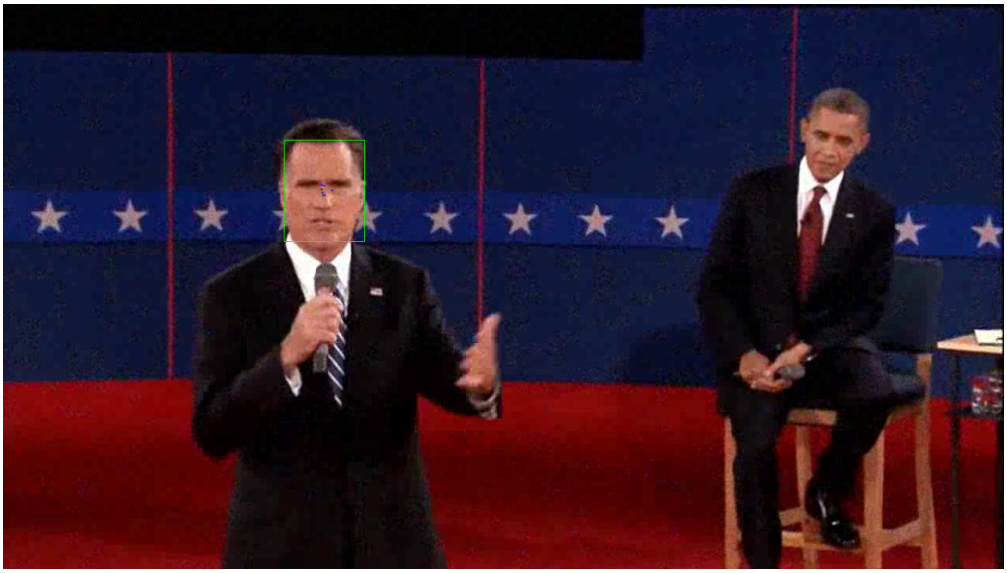
```
Elapsed:  14.896376848220825
```

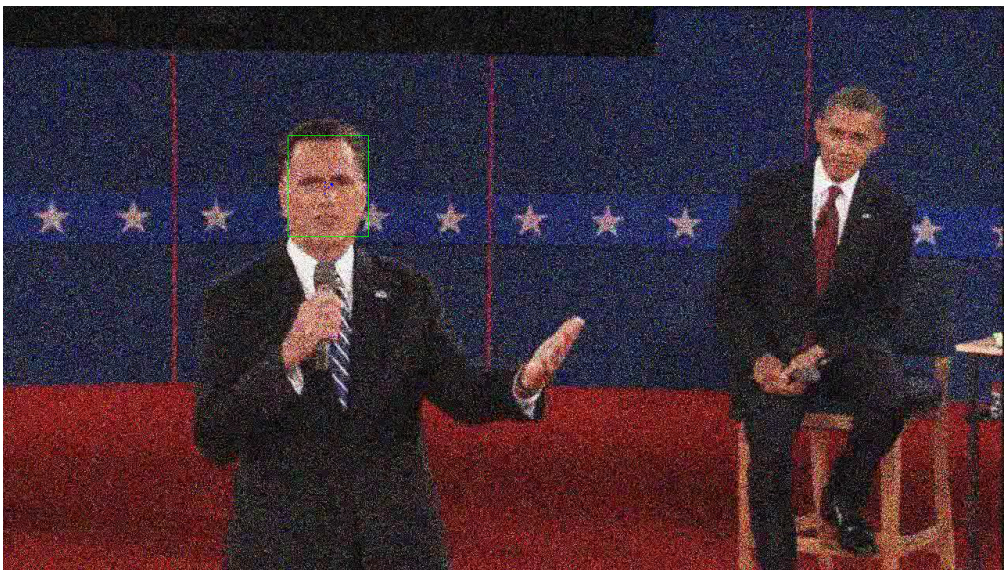
Discussion:

5 particles do not work with the noisy version video, but it works with the no-noise one. Because of the noise, there might be some pattern that has the same brightness as the face. Small particles number cannot ensure a particle that finds the right face position but not the noise. 7 Particles can already track the face.

7 Particles:







## 2 Appearance Model Update

2.1 Implement the appearance model update. Run the tracker on pres_debate.avi and adjust parameters until you can track Romney's hand up to frame 140. Run the tracker and save the video frames 15, 50, and 140 with the visualizations overlaid.

Output: The code, the 3 image frames with overlaid visualizations, and the image patch used for tracking.

In [21]:

```
def update_template(particles, weights, template, image_t, sigma_d, sigma_MSE):
    new_particles = []
    new_weights = []
    weight_sum = 0
    for i in range(len(particles)):
        p_index = np.random.choice(range(len(particles)), p=weights)
        sample_p = particles[p_index]

        # Sample from prediction
        new_p = prediction(sample_p, sigma_d)
        # weight by measurement
        w = measurement(new_p, template, image_t, sigma_MSE)
        new_weights.append(w)
        weight_sum += w
        new_particles.append(new_p)

    for i in range(len(particles)):
        new_weights[i] = new_weights[i]/weight_sum
    best_template_yx = new_particles[np.argmax(new_weights)]
    best_template_yx[0] = int(best_template_yx[0])
    best_template_yx[1] = int(best_template_yx[1])
    return new_particles, new_weights, best_template_yx
```

In [27]:

```
def particle_filter_appearance_model(video_file='pres_debate.avi', nb_particle=10, alpha=0.5, origin_temp=False,\
                                     sigma_d=10, sigma_MSE=10, frame_output=[15, 50, 140],id_='2-1',\
                                     all_frames=False, video_output='pres_debate_hand_tracking'):
    pres_debate = cv2.VideoCapture(video_file)
    count = 0

    rect_x, rect_y, rect_w, rect_h = 533, 383, 74, 105
    init_particle_center = np.array((int(rect_y + rect_h/2), int(rect_x + rect_w/2)))

    weights = np.ones((nb_particle)) / nb_particle

    particles = np.random.multinomial(100, [1/2.] * 2, size=nb_particle)
    particles = particles - 50
    for idx in range(len(particles)):
        particles[idx] = particles[idx] + init_particle_center
    ret, frame = pres_debate.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    template = gray[rect_y :rect_y + rect_h, rect_x : rect_x + rect_w]
    origin_template = template
    win_h, win_w = template.shape
    cv2.imwrite('ps6-{}-template.png'.format(id_), template)

    count += 1
    for p in particles:
        frame = cv2.circle(frame, (int(p[1]), int(p[0])), 1, (255, 0, 0), 1)
    frame = cv2.rectangle(frame, (rect_x, rect_y), (rect_x+rect_w, rect_y+rect_h),(0, 255, 0), 1)
    if all_frames:
        cv2.imwrite('./{}/0.png'.format(id_), frame)

    if video_output:
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        out = cv2.VideoWriter("{}-{}-p{}-mse{}-d{}-a{}.avi".format(video_output, id_, nb_particle,\
                                                                   sigma_MSE, sigma_d, alpha),\
                              fourcc, 20.0, (frame.shape[1], frame.shape[0]))
        frame = cv2.flip(frame, 1)
        out.write(frame)

    start_time = time.time()

    while(pres_debate.isOpened()):
        ret, frame = pres_debate.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        particles, weights, template_yx = update_template(particles, weights, template, gray, sigma_d, sigma_MSE)

        mean_p = np.mean(particles, axis=0)
        mean_rect_LT = [int(mean_p[1] - win_w/2), int(mean_p[0] - win_h/2)]
        mean_rect_RB = [int(mean_p[1] + win_w/2), int(mean_p[0] + win_h/2)]
```

```
        # Update template
        best_window = gray[int(template_yx[0] - int(template.shape[0]/2)) : int(
template_yx[0] +\

int(template.shape[0]/2+0.5)),\
                           int(template_yx[1] - int(template.shape[1]/2)) : int(
template_yx[1] +\

int(template.shape[1]/2+0.5))]
        template = alpha * np.matrix(template, dtype=np.float64) + (1 - alpha) *
np.matrix(best_window, dtype=\

np.float64)

        for p in particles:
            frame = cv2.circle(frame, (int(p[1]), int(p[0])), 1, (255, 0, 0), 1)
        frame = cv2.rectangle(frame, (mean_rect_LT[0], mean_rect_LT[1]), (mean_r
ect_RB[0], mean_rect_RB[1]),\
                              (0, 255, 0), 1)
        if all_frames:
            cv2.imwrite('./{}/{}.png'.format(id_, count), frame)
        if count in frame_output:
            cv2.imwrite('ps6-{}-{}-p{}-mse{}-d{}-a{}.png'.format(id_, count, nb_
particle, sigma_MSE, sigma_d, alpha),\
                        frame)
        count += 1
        if video_output:
            frame = cv2.flip(frame, 1)
            out.write(frame)
        #break
    pres_debate.release()
    elapsed = time.time() - start_time
    print("Elapsed: ", elapsed)
```

In [28]:

```
particle_filter_appearance_model(nb_particle=50, sigma_d=120, sigma_MSE=10, alph
a=0.5, video_output=\
                                 'pres_debate_hand_tracking', origin_temp=False)
```

```
Elapsed:  10.899605989456177
```

particle30

particle30

particle30

2.2 Try running the tracker on noisy debate.avi. Adjust the parameters until you are able to track the hand all the way to frame 140. Indicate what parameters you had to change to get this to work on the noisy video and discuss why this would be the case.

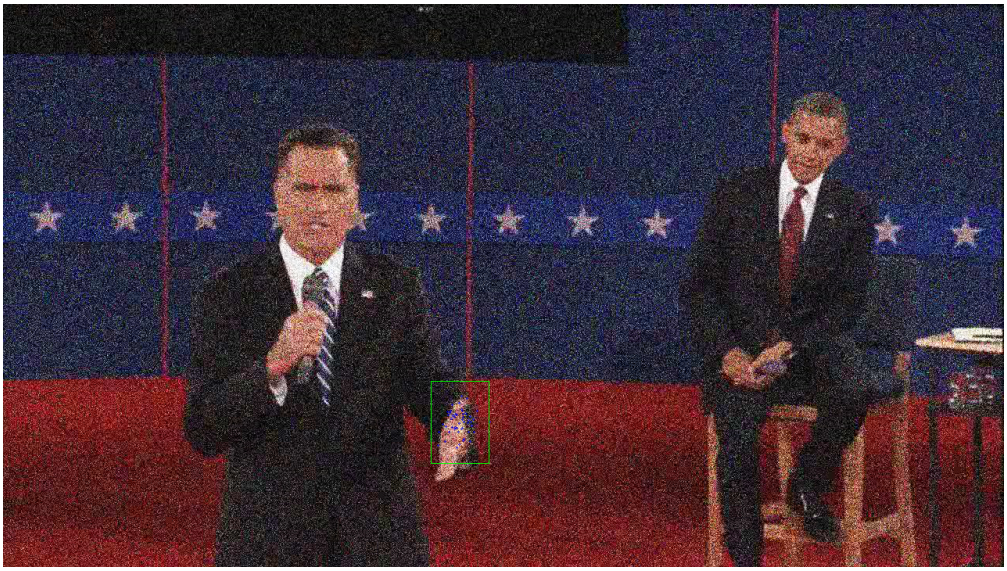
Output: The code, the 3 image frames with overlaid visualizations, the discussion in the pdf, and the image patch used for tracking.

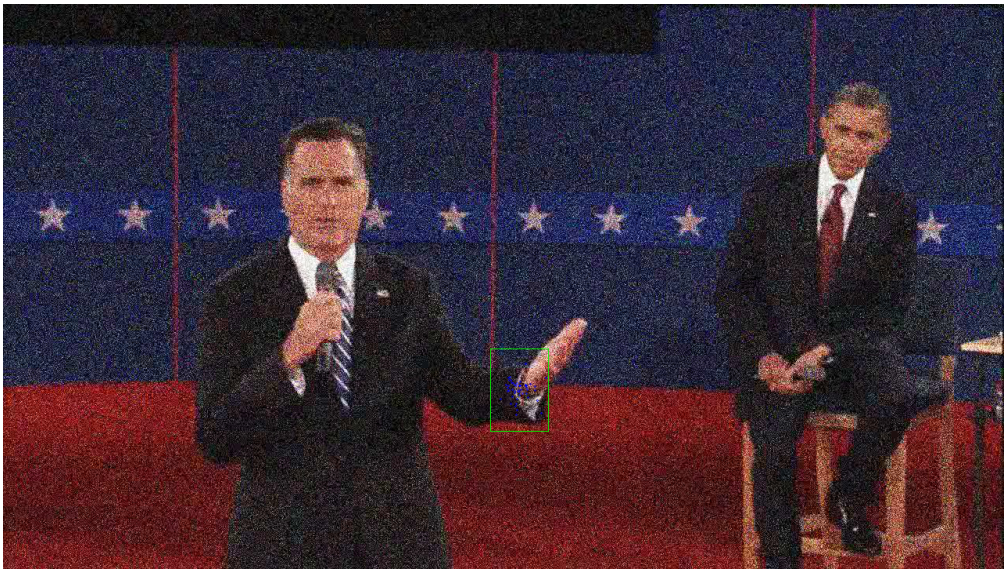
In [29]:

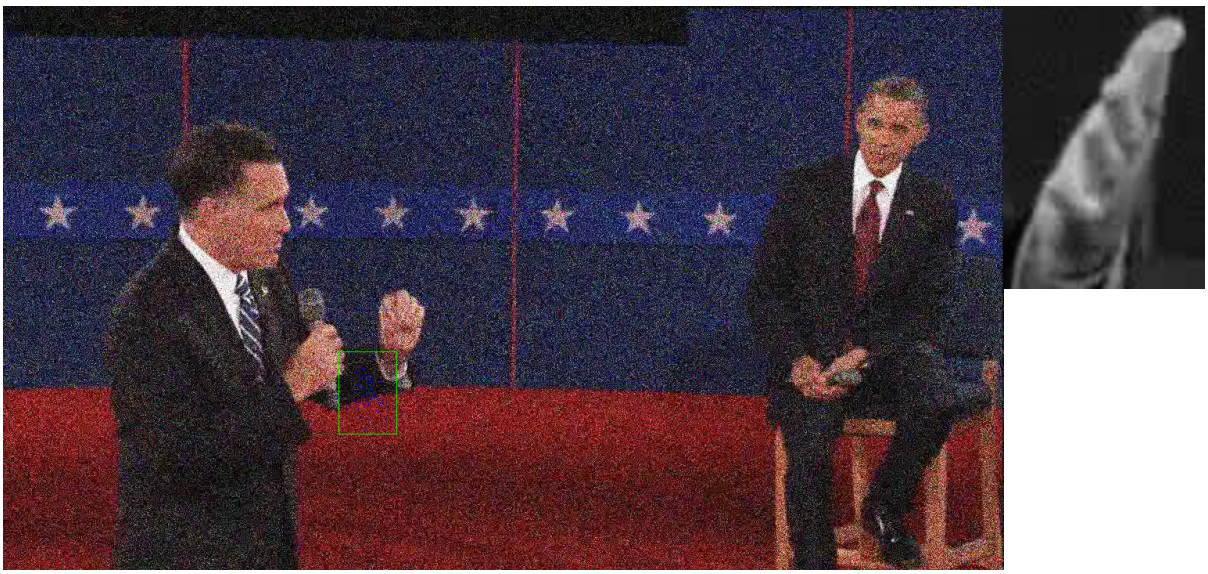
```
particle_filter_appearance_model(video_file='noisy_debate.avi', nb_particle=50,
alpha=0.3,\
                                 sigma_d=150, sigma_MSE=10, frame_output=[15
, 50, 140],id_='2-2',\
                                 all_frames=False, video_output='noisy_debat
e_hand_tracking')
```

```
Elapsed:  19.87918782234192
```

Discussion: I tried to use lower alpha and larger sigma for prediction stage.







## 3 Incorporating More Dynamics

3.1 Run the tracker and save the video frames 40, 100, and 240 with the visualizations overlaid. You will receive partial credit if you can show the tracking size estimate (illustrate this with the rectangle outline) up to frame 100. You will receive full credit if you can reliably track all the way to the end of the street and deal gracefully with the occlusions (reasonable tracking at frame 240).

Output: The code, the 3 image frames with overlaid visualizations, and the image patch used for tracking.

In [76]:

```
def scaling_template(template, image, mean_p, scaling_factor, sigma_MSE):
    win_h, win_w = template.shape

    new_win_h = int(win_h * scaling_factor)
    new_win_w = int(win_w * scaling_factor)

    template_reshaped = cv2.resize(template, (new_win_w, new_win_h))

    x = int(mean_p[1])
    y = int(mean_p[0])
    image_part = image[y - int(new_win_h/2) : y + int(new_win_h/2+0.5), x - int(
new_win_w/2) : x + int(new_win_w/2+0.5)]
    if template_reshaped.shape != image_part.shape:
        return None, 0
    diff = np.matrix(template_reshaped, dtype=np.float64) - np.matrix(image_part
, dtype=np.float64)
    MSE = 1/(new_win_h * new_win_w) * np.sum(np.multiply(diff, diff))
    return template_reshaped, MSE
```

In [98]:

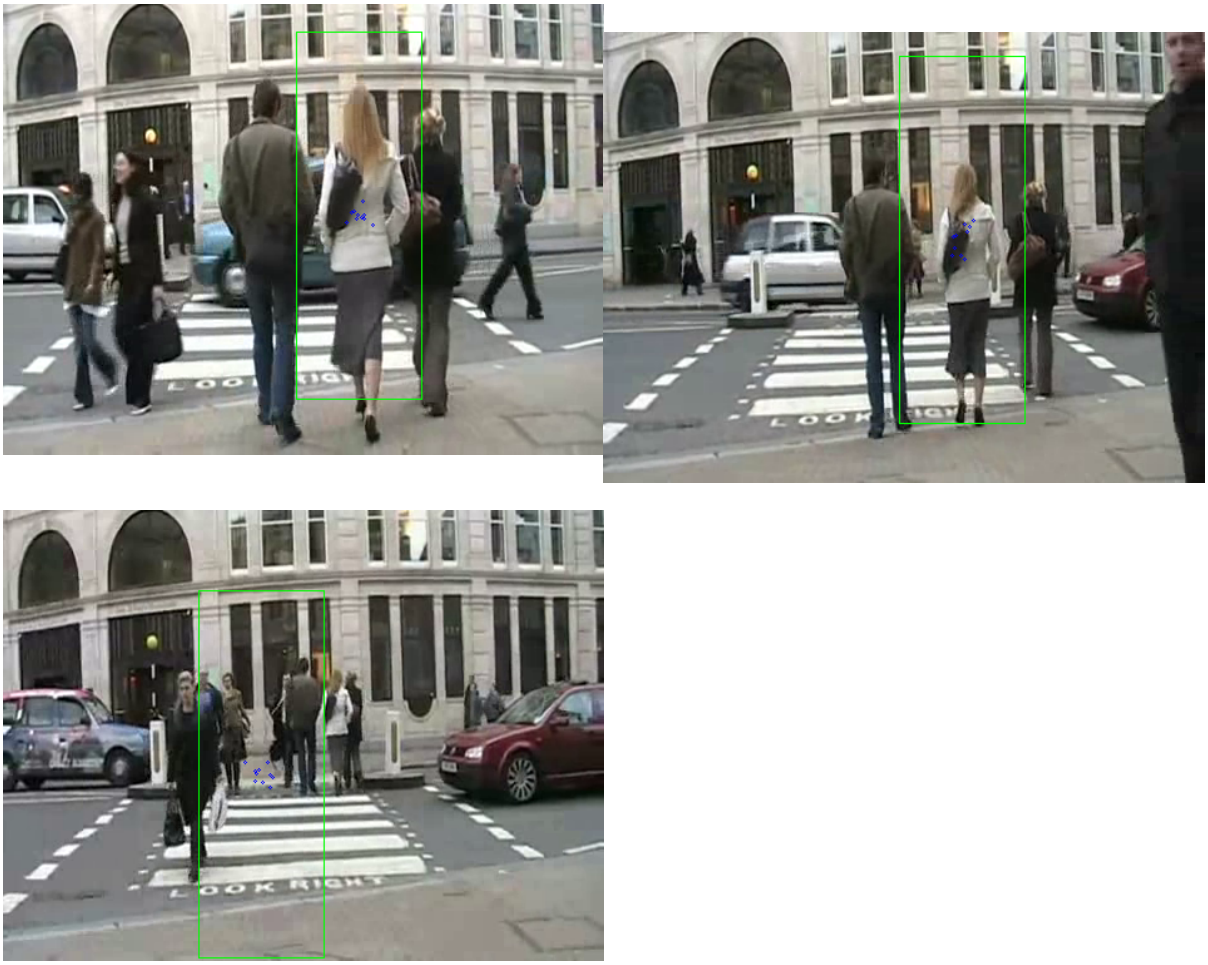
```
def particle_filter_dynamic_window(video_file='pedestrians.avi', txt_file='pedes
trians.txt', nb_particle=10,\
                    sigma_d=30, sigma_MSE=10, frame_output=[40, 100, 240], id_=
'3-1', alpha=0.5,\
                    all_frames=False, video_output='pedestrians_tracking'):
    pres_debate = cv2.VideoCapture(video_file)
    count = 0

    rect_x, rect_y, rect_w, rect_h = read_bounding_box(txt_file)
    init_particle_center = np.array((int(rect_y + rect_h/2), int(rect_x + rect_w
/2)))

    weights = np.ones((nb_particle)) / nb_particle

    particles = np.random.multinomial(100, [1/2.] * 2, size=nb_particle)
    particles = particles - 50
    for idx in range(len(particles)):
        particles[idx] = particles[idx] + init_particle_center
    ret, frame = pres_debate.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    template = gray[rect_y : rect_y + rect_h, rect_x : rect_x + rect_w]
    win_h, win_w = template.shape

    count += 1
    for p in particles:
        frame = cv2.circle(frame, (int(p[1]), int(p[0])), 1, (255, 0, 0), 1)
    frame = cv2.rectangle(frame, (rect_x, rect_y), (rect_x+rect_w, rect_y+rect_h
),(0, 255, 0), 1)
    if all_frames:
        cv2.imwrite('./{}/0.png'.format(id_), frame)

    if video_output:
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        out = cv2.VideoWriter("{}-{}.avi".format(video_output, id_), fourcc, 20.
0, (frame.shape[1], frame.shape[0]))
        frame = cv2.flip(frame, 1)
        out.write(frame)

    start_time = time.time()

    while(pres_debate.isOpened()):
        ret, frame = pres_debate.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        particles, weights = update(particles, weights, template, gray, sigma_d,
sigma_MSE)

        mean_p = np.mean(particles, axis=0)

        # Update template size
        scaling_list = np.arange(0.999, 1.0, 0.0001)
        min_MSE = np.inf
        for s in scaling_list:
            new_template, MSE = scaling_template(template=template, image=gray,
mean_p=mean_p,\
                                                 scaling_factor=s, sigma_MSE=si
gma_MSE)
```

```
            if MSE < np.inf:
                best_scaling = s
        #print(best_scaling)
        #scaling = (1 - alpha) * 1 + alpha * best_scaling
        template = cv2.resize(template, (0, 0), template, fx=best_scaling, fy=be
st_scaling)
        win_h, win_w = template.shape

        mean_rect_LT = [int(mean_p[1] - win_w/2), int(mean_p[0] - win_h/2)]
        mean_rect_RB = [int(mean_p[1] + win_w/2), int(mean_p[0] + win_h/2)]

        for p in particles:
            frame = cv2.circle(frame, (int(p[1]), int(p[0])), 1, (255, 0, 0), 1)
        frame = cv2.rectangle(frame, (mean_rect_LT[0], mean_rect_LT[1]), (mean_r
ect_RB[0], mean_rect_RB[1]),\
                              (0, 255, 0), 1)

        if all_frames:
            cv2.imwrite('./{}/{}.png'.format(id_, count), frame)
        if count in frame_output:
            cv2.imwrite('ps6-{}-{}.png'.format(id_, count), frame)
        count += 1
        if video_output:
            frame = cv2.flip(frame, 1)
            out.write(frame)
        #break
    pres_debate.release()
    elapsed = time.time() - start_time
    print("Elapsed: ", elapsed)
```

In [99]:

```
particle_filter_dynamic_window(nb_particle=10)
```

```
Elapsed:  6.397747993469238
```



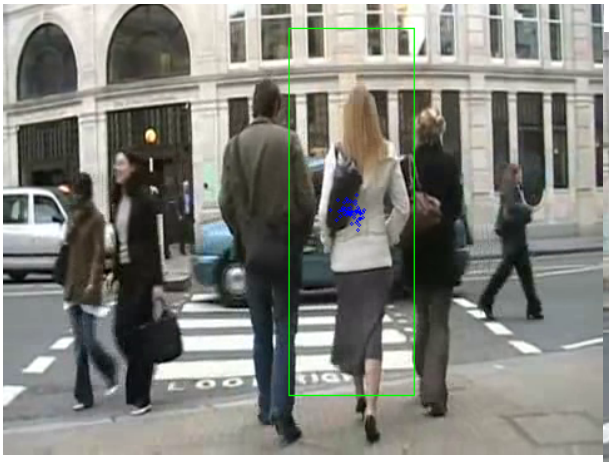

3.2 Try and optimize the number of particles needed to track the model in this video. Compare that to the number you found in problem 1.4. Why is this number different?

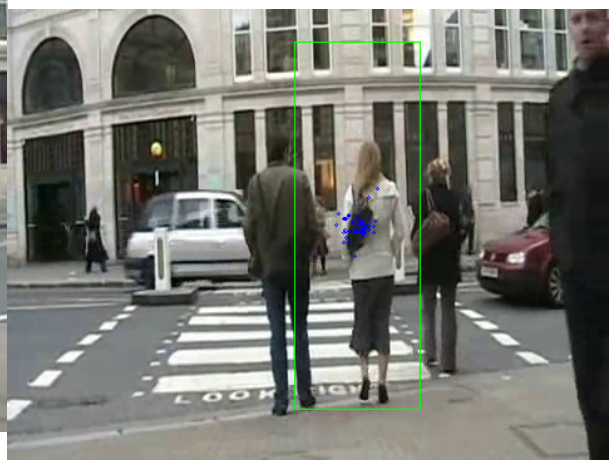
Output: The number of particles you found to be optimal and discussion in the pdf.

In [100]:

```
particle_filter_dynamic_window(nb_particle=50, id_="3-2")
```

```
Elapsed:  18.088554859161377
```






Discussion:

In 3-1, I use 10 particles. Then I tried 30, 50, 100 particles. But did not improve the performance. When there is a man who passed through the screen in front, the tracker could not find the woman anymore, even if I increased the sigma in the prediction stage, which did not improve. Maybe a better way is to introduce a motion model for the woman's previous movement and predict her next step on it.