**HW3: Particle Filter Tracking**

Student Name: Ben Leibovicz ID1: 315488171

Student Name: Avraham Yosovitch ID2: 314756297

**Part A: General Questions**

**1.** The formula is based on Bayes theorem. We will denote the measurement as Z and the state vector as X, at a given moment or a history of moments. The likelihood is the probability of Z given X. The prior is the probability of X. The posterior is the probability of X given Z. Hence, according to Bayes theorem:

The formula relates to the Kalman filter, as the posterior state vector is obtained from the prior state vector within two steps: update and project. The relation between the posterior state vector and the prior state vector is according to the formula, and it is expressed by Kalman gain. The Kalman Gain is a proportion that takes into consideration the measurement noise and the system transfer matrix (incorporate data), to decide whether to use Lucas-Kanade or not. Then, during the project stage, deterministic drift is applied on the posterior state vector to obtain the posterior state vector for the next time step.

The formula also relates to particle filters. The particle filter is like the Kalman filter; however, it does not assume a Gaussian distribution for the posterior distribution. Based on the Markov assumption and conditional independence, the formula is used in a recursive manner such that the posterior distribution is:

And that the filter distribution is: .

**2.** For the measurement step in the particle filter, we used the histogram of the two patches (the patch around the new particle and the original patch of the object).

**a.** According to the condensation algorithm, in the measurement step we measure the weight of a particle by calculating the likelihood that the current state vector is equal to the predicted state for the particle. In our particle filter, we compute a histogram within a patch around each particle and compare it to the patch around the object based on the initial state vector. The weight of the particle is the Bhattacharyya distance between the two histograms. We use these histograms to score the patches since they hold relevant information used for tracking the movement of an object. Each value of the histogram holds the percentage of a 4-bit RGB value for each pixel in the area around a particle as defined by its state vector. Thus, the maximal weight calculated from the histograms matches the object we are tracking.

The pros of the method are: 1) uses a relatively small histogram vector of 4096 values, 2) good at tracking, 3) works for every given input state vector, 4) the likelihood of the measurements given the predicted state vector is well defined.

The cons of the method are: 1) inaccurate, as data is lost by converting 8-bit RGB values into 4-bit RGB values, 2) will fail if there are several objects that look like each other in the video (in this case, tracking based on mean shift clustering may work better).

**b.** Instead of calculating the histograms and the Bhattacharyya distance, it is offered to calculate the sum square difference (SSD) between the two patches. This method won’t work. One of the requirements to apply SSD is that the window size for each particle is the same since we must calculate the square difference between each two pixels. This already limits a predicted state vector which is something not wanted in a particle filter. Even if we are willing to force the window size to be the same for every particle, we also must consider how the object has changed between each frame. If the object is a person that is running, then it is expected that the SSD will be consistently large. Hence, it will be difficult to determine the likelihood that the predicted state vector is the correct one.

**c.** We have thought of another method for comparing between the two patches. This method is based on a computer vision algorithm called SIFT (scale-invariant feature transform). The algorithm detects feature points of the object (such as the object’s corners). The object is recognized in a new image by comparing each feature from the new image and matching features based on Euclidean distance of their feature vectors. Subsets of feature points that agree on the object, its location, scale, and orientation in the new image are identified to filter good matches. Determination of consistent cluster is done using a hash table implementation of Hough transform, and then outliers are discarded. Finally, the probability that a particular set of features indicates the presence of an object is computed. We can call the probability obtained from the SIFT algorithm as the weight of the particle. This method is much more accurate than calculating the distance between two histograms.

Pros: 1) this method provides specific information on the object using feature points and thus is more accurate at recognizing the object. 2) can identify objects even among clutter and partial occlusion, since the SIFT feature descriptor is invariant to uniform scaling, orientation, illumination changes and partially invariant to affine distortion.

Cons: 1) computationally expensive – as it requires much more steps than creating a histogram and computing distances, 2) the features extracted from the object must be detectable under changes to image scale, noise, and illuminations (requires high contrast regions such as corners), 3) the relative position between the features shouldn’t change from one image to another (for example, locating features in flexible objects).

**שאלה 3**

מסנן החלקיקים שמימשנו בתרגיל לא יצליח לעקוב אחרי אובייקט עםscale משתנה בין תמונות. זאת מכיוון שמלבן העקיבה בעל רוחב וגובה קבועים, ובסופו של דבר האלגוריתם ייכשל במקרה הזה, זאת מכיוון שהשוואת ההיסטוגרמות לא תביא לתוצאות מדוייקות אם מספר הפיקסלים לא יהיה זהה.

אבל עדיין אפשר לבצע התאמות למסנן החלקיקים כמו שמתואר במאמר הבא : <https://webee.technion.ac.il/people/nakhmani/Nakhmani,%20Tannenbaum_2008_Scale-invariant%20visual%20tracking%20by%20particle%20filtering.pdf>

ואז להצליח לעקוב בעזרת מסנן חלקיקים אחרי אובייקט עם scale משתנה.

תחת ההנחה שהscale לא משתנה בפתאומיות, אפשר להניח שהscale בין תמונות הוא תהליך מרקובי ואפשר לשערך אותו כחלק מהפילטר, ואז עקיבה אחרי אובייקט עם scale משתנה תצליח.

מסנן החלקיקים מניח שהמצלמה במקום קבוע שאינו משתנה. לדוגמה באלגוריתם שמימשנו בתרגיל, שיערוך המצב שלנו היה על ידי הוספת מהירות מהframe הקודם, ללא התייחסות לתנועת המצלמה. במצב זה, מלבן העקיבה יהיה במקום שונה ממיקום האובייקט, והפילטר ייכשל.

אבל אפשר לגרום לפילטר להצליח לעקוב גם כאשר המצלמה בתנועה על ידי שילובו עם אלגוריתם שמחשב בנוסף את תנועת המצלמה, כמו שמצויין במאמר הבא : <https://www.researchgate.net/publication/263630375_Object_Tracking_Using_Particle_Filters_in_Moving_Camera>

במקרה הזה, לפני צעד של הפילטר, נקזז את תנועת המצלמה ואז נוכל להשוות בין שני ההיסטוגרמות של הframes בצורה נכונה.

**שאלה 4**

במידה ולא נעדכן את הtemplate ההתחלתי האלגוריתם ייאבד עקיבה ברגע שהאובייקט יהיה בתנאי תאורה שונים או בזווית שונה למצלמה.

במאמר הבא : <http://www.jsoftware.us/vol8/jsw0805-13.pdf>הציעו פתרון לבעיה.

אופציה ראשונה היא לעדכן את הtemplate בכל צעד של הפילטר לפי ההתאמה הכי טובה של הframe הקודם. החיסרון בדרך הזאת היא שבסוף יש שגיאות מצטברות בtemplate שייגרמו להתבדרות שלו.

האלגוריתם שהציעו במאמר לפתור את הבעיה משתמש בdynamic template , שבו בכל צעד מחשבים את הtemplate החדש לפי המשקול הבא :

כאשר *הוא ה*template *החדש, הוא ה*template *ההתחלתי ו* הוא הtemplate לפי ההתאמה הכי טובה לפי הframe הקודם.

את הפרמטר *נקבע לפי* Coefficient Bhattacharyya שהוא המרחק בין ההתפלגויות של הtemplate ההתחלתי והtemplate החדש.

כאשר נשתמש באלגוריתם הנ"ל, נוכל לפתור את הבעיה של איבוד עקיבה בעקבות template התחלתי ששינה את צורתו המקורית.

**Part B: Particle Filter Algorithm**