Chapter 1

Particle Filter

The particle filter is a nonparametric filter, which approximate posteriors in partially observable controllable Markov chains with discrete time. The key idea is to represent the the posterior of the current state by a set of random state samples drawn from this posterior. The state in the Markov chain is not observable. Instead. one can measure z_t, which is a stochastic projection of the true state x_t according to the probability $p(z_t|x_t)$. Furthermore, the initial state x_0 is distributed according to some distribution $p(x_0)$.

The classical problem in partially observable Markov chains is to recover a posterior distribution over the state x_t at any time t, from all measurements $z_0, ..., z_t$. This posterior can be computed recursively. By Bayes' rule we know that:

$$p(x_t|z_{1:t}) = \frac{p(z_{1:t}|x_t)p(x_t)}{p(z_{1:t})}$$

Using the Markov assumption and the properties of conditional probability, we obtain:

$$\begin{aligned}
p(x_t|z_{1:t}) &= \frac{p(z_{1:t-1}, z_t|x_t)p(x_t)}{p(z_{1:t})} \\
&= \frac{p(z_t|x_t)p(z_{1:t-1}|x_t)p(x_t)}{p(z_{1:t-1}, z_t)} \\
&= \frac{p(z_t|x_t)p(z_{1:t-1}, x_t)}{p(z_{1:t-1}, z_t)} \\
&= \frac{p(z_t|x_t)p(x_t|z_{1:t-1})p(z_{1:t-1})}{p(z_t|z_{1:t-1})p(z_{1:t-1})} \\
&= \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \\
&\propto p(z_t|x_t)p(x_t|z_{1:t-1})
\end{aligned}$$

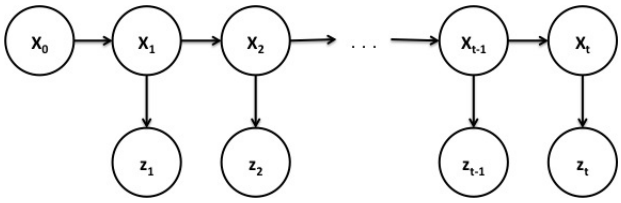


Figure 1.1: Hidden Markov Model for discrete states and measurements.

The posterior of the current state given the previous measurements $p(x_t|z_{1:t-1})$ is given by

$$\begin{aligned} p(x_t|z_{1:t-1}) &= \frac{p(x_t, z_{1:t-1})}{p(z_{1:t-1})} \\ &= \int \frac{p(x_{t-1}, x_t, z_{1:t-1})}{p(z_{1:t-1})} dx_{t-1} \\ &= \int \frac{p(x_t|x_{t-1}, z_{1:t-1}) p(x_{t-1}, z_{1:t-1})}{p(z_{1:t-1})} dx_{t-1} \end{aligned}$$

By the Markov assumption we get:

$$p(x_t|z_{1:t-1}) = \int \frac{p(x_t|x_{t-1}) p(x_{t-1}, z_{1:t-1})}{p(z_{1:t-1})} dx_{t-1}$$

Using the rules of conditional probability, we obtain:

$$\begin{aligned} p(x_t|z_{1:t-1}) &= \int \frac{p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) p(z_{1:t-1})}{p(z_{1:t-1})} dx_{t-1} \\ &= \int p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) dx_{t-1} \end{aligned}$$

Thus, the posterior $p(x_t|z_{1:t})$ can be expressed as follows:

$$p(x_t|z_{1:t}) \propto p(z_t|x_t) \int p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) dx_{t-1} \tag{1.1}$$

under the initial condition $p(x_0|z_0) = p(x_0)$. If the states and measurements are discrete, the Markov chain is equivalent to hidden Markov models and 1.1 can be computed exactly. In case that $p(x_0)$ is Gaussian and $p(x_t|x_{t-1})$ and $p(z_t|x_t)$ are linear in its arguments with added independent Gaussian noise, then 1.1 is equivalent to Kalman filter.

The main idea is to approximate the posterior of a set of sample states $\{x_t^{[i]}\}$, or particles, where i goes from 1 to M, the size of the particle filter. The particle filter algorithm consists of two steps:

- At time $t = 0$, draw M particles from $p(x_0)$. Call this set of particles X_0

- At time $t > 0$, generate a new particle $x_t^{[i]}$ for each particle $x_{t-1}^{[i]}$ by drawing from $p\left(x_t|x_{t-1}^{[i]}\right)$, then choose $x_t^{[i]}$ with replacement with probability proportional to $p(z_t|x_t^{[i]})$, this is called the "resampling" step. The resulting set of particles X_t is then used for the next state, and so on.

Algorithm 1 shows the complete particle filter algorithm for certain state $t > 0$. Line 6 is important, because importance weights must be normalized in order to have a true probability density function. Line 10 and 11 correspond to the resampling step, where particles with small importance weight are less probable to be chosen than those particles with big importance weight.

1.1 Importance Sampling

In order to understand more deeply the behavior of the resampling step, we are going to explain the Importance Sampling method, which is used by the particle filter to obtain the new set of particles X_t. Suppose we want to obtain samples from a probability density function p. However, sampling from p directly may not be possible. Then we must generate a set of particles from other density function q, from which sampling is possible. This distribution is called the proposal distribution and must be greater than zero, when q is greater than zero, then $p(x) > 0$ implies $q(x) > 0$.

Suppose we want to estimate the expectation of a certain function f, which is given by

$$\mathbb{E}[f] = \int f(x)p(x)dx$$

Algorithm 1 Particle Filter Algorithm

1: Input: X_{t-1}, z_t
2: **for** $i = 1$ to M **do**
3: $\quad x_t^{[i]} \sim p\left(x_t|x_{t-1}^{[i]}\right)$
4: $\quad w_t^{[i]} = p\left(z_t|x_t^{[i]}\right)$
5: **end for**
6: **for** $i = 1$ to M **do**
7: $\quad w_t^{[i]} = w_t^{[i]} / \sum_{k=1}^{M} w_t^{[k]}$
8: **end for**
9: **for** $i = 1$ to M **do**
10: $\quad$ draw k with probability $w_t^{[k]}$
11: $\quad$ add $x_t^{[k]}$ to X_t
12: **end for**

The goal is to calculate this integral by sampling from the proposal distribution q, then, we can rewrite the last equation as follows:

$$\begin{aligned}\mathbb{E}[f] &= \int f(x)\frac{p(x)}{q(x)}q(x)dx \\ &\simeq \frac{1}{M}\sum_{i=1}^{M}\frac{p\left(x^{(i)}\right)}{q\left(x^{(i)}\right)}f\left(x^{(i)}\right)\end{aligned} \tag{1.2}$$

where each $x^{(i)}$ are samples from $q(x)$, and the quantities $r_i = p\left(x^{(i)}\right)/q\left(x^{(i)}\right)$ are known as importance weights. Let's recall that both p and q are normalized distributions such that the integral of those distributions sum up to one. Those distributions are defined by

$$\begin{aligned}p(x) &= \frac{\widetilde{p}(x)}{Z_p} \\ q(x) &= \frac{\widetilde{q}(x)}{Z_q}\end{aligned}$$

where Z_p and Z_q are the normalization constants for distributions p and q respectively. Often the normalization constant is unknown, and equation 1.2 can be rewritten by:

$$\begin{aligned}\mathbb{E}[f] &\simeq \frac{Z_q}{Z_p}\frac{1}{M}\sum_{i=1}^{M}\frac{\widetilde{p}\left(x^{(i)}\right)}{\widetilde{q}\left(x^{(i)}\right)}f\left(x^{(i)}\right) \\ &\simeq \frac{Z_q}{Z_p}\frac{1}{M}\sum_{i=1}^{M}\widetilde{r}_i f\left(x^{(i)}\right)\end{aligned} \tag{1.3}$$

The ratio Z_p/Z_q can be expressed as:

$$\begin{aligned}\frac{Z_p}{Z_q} &= \frac{1}{Z_q}\int \widetilde{p}(x)dx \\ &= \int \frac{\widetilde{p}(x)}{\widetilde{q}(x)}q(x)dx \\ &\simeq \frac{1}{M}\sum_{i=1}^{M}\widetilde{r}_i\end{aligned}$$

Then, we obtain

$$\mathbb{E}[f] \simeq \sum_{i=1}^{M} w_i f\left(x^{(i)}\right) \tag{1.4}$$

where we have defined

$$w_i = \frac{\widetilde{r}_i}{\sum_{j=1}^{M}\widetilde{r}_j}$$

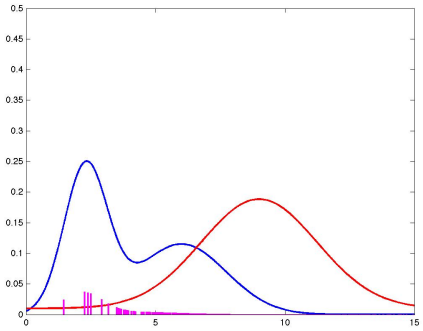


Figure 1.2: Illustration of Importance Sampling, the blue line represents the target distribution, and the red line represents the proposal distribution. The vertical lines are the normalized importance weights.

For the case of the particles filters, the posterior over state t is given by

$$p(x_t|z_{1:t}) \propto p(z_t|x_t) \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1}$$

And the proposal distribution over the state t is defined as

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1}$$

then, the importance weight is given by the following expression

$$\begin{aligned}\widetilde{r} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\ &\propto \frac{p(z_t|x_t)\int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1}}{\int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1}} \\ &\propto p(z_t|x_t)\end{aligned}$$

1.2 Sampling Importance Resampling

The SIR algorithm consists of two steps. The first step is the importance sampling, generate M particles and then calculate the importance weights and normalize them. In the second step M particles are selected with replacement according to the normalized importance weights, this process is called the "resampling" step. Because the selection if made with replacement, the resulting set of particles can have duplicates. If some particles have a small weight, then those particles are less probably to be selected.

There are different methods that perform the resampling step, one of those is the method called "Low Variance Sampler", which is described in algorithm

Algorithm 2 Low Variance Sampler

Input: X_t, W_t
$\widetilde{X}_t = 0$
$r = rand\left(0, M^{-1}\right)$
$c = w_t^{[1]}$
$i = 1$
for $j = 1$ to M **do**
  $u = r + (j-1) \cdot M^{-1}$
  while u ¿ c **do**
    $i = i + 1$
    $c = c + w_t^{[i]}$
  end while
  add $x_t^{[i]}$ to $\widetilde{X}_t$
end for
return $\widetilde{X}_t$

2. The idea of this method is that instead of selecting samples independently, the selection involves a sequential stochastic process. First suppose that particles are located in a single row one after another, ant size of each particle is equal to its weight. Then we select a random number in the interval $[0, M^{-1}]$ and place mark in that location, after that, we continue increasing this number by M^{-1} and placing marks in every position. At the end, the selected particles will be those that have a mark on it, it can be that one particle have multiple marks, it means that it will be duplicates of that particle. Figure 1.3 illustrates the process of the Low Variance Sampler.

One advantage of the Low Variance Sampler is that if all particles have the same weight, all particles will be selected, then $\widetilde{X}_t$ will be equal to X_t, thing that doesn't happen with the independent random sampler. Another advantage is that the complexity of this algorithm is $O(M)$, which is faster than other algorithms. If we are dealing with a big number of particles, is always good to have algorithms with low cost in execution time, which makes the Low Variance Sampler a good option to work with.

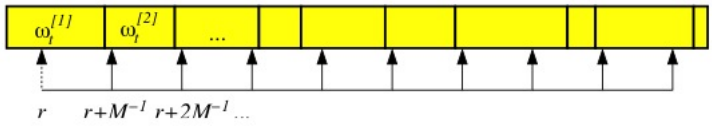


Figure 1.3: Illustration of the Low Variance Sampler. We choose a random number r and then we select particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, ..., M$.

1.3 Application: Robot Localization

Given a map of the environment where the robot lives, the goal is to estimate the robot localization according to the map and what the robot sense. The robot has four sensors. One in front, one in the back, one in the right side and another in the left side. These sensors are used to find the nearest obstruction using a straight line, like a laser. Sensor i perform a measure z_i at time t given by

$$z_t^{[i]} = D_i + \epsilon_i$$

where D_i is the distance to nearest obstacle and ϵ_i represent the noise of the measurement, which is drawn by a normal distribution with zero mean, and variance equal to v_s^2.

For this example, the Particle Filter algorithm for robot localization works as follows:

1. At the beginning ($t = 0$) we have no idea where the robot is located, then all the particles are initially drawn from an uniform distribution across the search space.

2. At time $t > 0$, the robot sense its surroundings, obtaining measurements $z_t = \{z_t^{[1]}, z_t^{[2]}, z_t^{[3]}, z_t^{[4]}\}$

3. If the new measurements differ from the previous measurements, then the particles locations are updated according to $p\left(x_t^{[i]} | x_{t-1}^{[i]}\right)$. It means that the new location of the particles depends only of their location in the previous state. For our case we used a normal distribution with mean in $x_{t-1}^{[i]}$ and variance v_m^2.

4. Compute the importance weights of each particle with $p(z_t | x_t^{[i]})$. The importance weights r_i are obtained with

$$r_i = \prod_{j=1}^{4} q_j(s_{j,i})$$

   where $s_{j,i}$ is the distance of particle i to the nearest obstacle obtained by sensor j, and $q_j(\cdot)$ is a normal distribution with mean equal to $z_t^{[j]}$ and variance equal to v_w^2.

5. Normalize the importance weights to obtain w_i, then

$$w_i = \frac{r_i}{\sum_{j=1}^{M} r_j}$$

6. Obtain the new set of particles using the Low Variance Sampler.

(a) after 0 iterations (b) after 3 iterations

(c) after 18 iterations (d) after 33 iterations

Figure 1.4: Illustration of Particle Filter for a robot localization problem. The blue points represent the particles, the red circle represent the robot location what the algorithm try to estimate. The red lines that come from the robot location represent the measurements.

Steps 2 trough 6 are repeated in every state. Figure 1.4 illustrates how the behavior of the particle filter. As we can see at the beginning all the particles are distributed across the entire search space, this because we have no idea where the robot is. Then after three iterations, we have only few clusters of particles located, each one of this clusters represent an estimate of the robot location. After 18 iterations only two clusters survive, but at the end only one survives, this means that the particles converged to the actual robot location.

1.4 Application: Face Tracking

In this application what we want to do is to track a specific face given a set of measurements, in this case, the frames of some video. The particles are rectangles of size equal to the image face we are trying to track. At time $t = 0$ the particles are distributed uniformly because we don't know the location of the face we are looking for. According to the algorithm 1 we need to compute the new position of the particles at time t depending on their position at time $t - 1$. For this we

Figure 1.5: Target image for tracking face application.

use the following probabilities:

$$p(x_t^{[i]}|x_{t-1}^{[i]}) = x_{t-1}^{[i]} + vel_x^{[i]} + \epsilon_x$$
$$p(y_t^{[i]}|y_{t-1}^{[i]}) = y_{t-1}^{[i]} + vel_y^{[i]} + \epsilon_y$$

where $\epsilon_x = \mathcal{N}(0, \sigma_x^2)$ and $\epsilon_y = \mathcal{N}(0, \sigma_y^2)$. At time $t = 0$, the velocity of each one of the particles in any direction is zero, but when $t > 0$ we have:

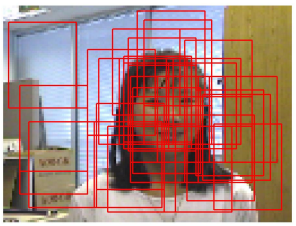
$$vel_x^{[i]} = x_t^{[i]} - x_{t-1}^{[i]}$$
$$vel_y^{[i]} = y_t^{[i]} - y_{t-1}^{[i]}$$

The next step is to compute the importance weights of each one of the particles. In order to do this we use a color histogram, which consists on counting how many times each color appears in the image. A histogram with k intervals means that the histogram has k distinct colors. For example, suppose we have $k = 3$, and a RGB image, where colors can take values from 0 to 255. The first possible values for the color histogram are the following:

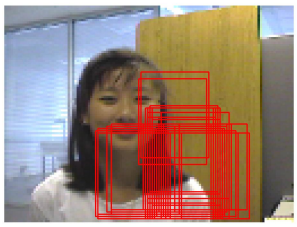
$0 \leq r < 85$	$0 \leq g < 85$	$0 \leq b < 85$
$0 \leq r < 85$	$0 \leq g < 85$	$85 \leq b < 170$
$0 \leq r < 85$	$0 \leq g < 85$	$170 \leq b$
$0 \leq r < 85$	$85 \leq g < 170$	$0 \leq b < 85$
$0 \leq r < 85$	$85 \leq g < 170$	$85 \leq b < 170$
$0 \leq r < 85$	$85 \leq g < 170$	$170 \leq b$
$0 \leq r < 85$	$170 \leq g$	$0 \leq b < 85$
$0 \leq r < 85$	$170 \leq g$	$85 \leq b < 170$
$0 \leq r < 85$	$170 \leq g$	$170 \leq b$

Then we have a total of k^3 possible values. The method of histogram intersection calculates the distance similarity between the image we are trying to find (target image) and the image enclosed by particle p (particle image), a distance of 0 means identical images. Thus, the more distinct the images are, the greater the magnitude of the distance. The distance is calculated for histograms with k intervals as:

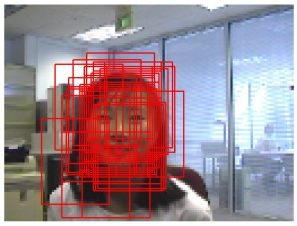
$$d_{t,p} = 1 - \frac{\sum_{i=0}^{k^3-1} \min(h_t[i], h_p[i])}{\min(|h_t|, |h_p|)} + \epsilon_s$$



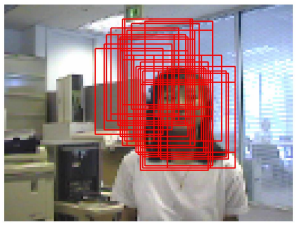
(a) after 1 iterations



(b) after 21 iterations



(c) after 41 iterations



(d) after 61 iterations

Figure 1.6: Particle Filter for face tracking

where

$$|h_x| = \sum_{i=0}^{k^3-1} h_x[i]$$

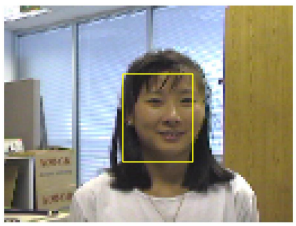
$h_t[i]$ and $h_p[i]$ represent the value of the i value of the target image's histogram and particle image's histogram, respectively. The unnormalized importance weight of a given particle p is given by the term:

$$\tilde{w}_p = \frac{\sum_{i=0}^{k^3-1} \min(h_t[i], h_p[i])}{\min(|h_t|, |h_p|)}$$

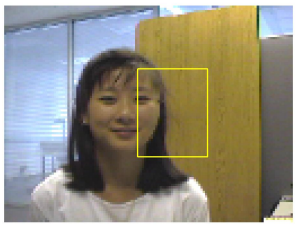
The value of ϵ_s represent an error in the measurement, and is obtained by a normal distribution with mean 0 and standard deviation of 0.01.

Once obtained the importance weights, we proceed to the resampling step, for this we apply the Low-Variance Sampler algorithm 2 without any modification. Figure 1.6 shows the behavior of the particles in different iterations of the algorithm.

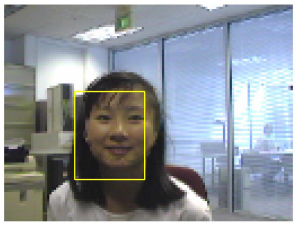
As we can see in figure 1.6, at the beginning we have no idea where the target image is located, so the particles are distributed uniformly over all the search space. In image 1.6b the algorithm is a little bit confused, this because the



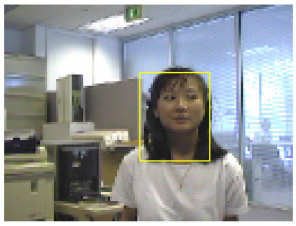
(a) after 1 iterations



(b) after 21 iterations



(c) after 41 iterations



(d) after 61 iterations

Figure 1.7: Particle Filter for face tracking

histogram intersection method is not so robust, for example, there can be two images with the same histogram but represent two entirely different images, but because the frequency of the color are the same they will be reported as the same image. In iterations 41 and 61 the particles converged correctly to the target image. Figure 1.7 shows the mean of the particles in different iterations. It is expected that if particles converged around the target image, the mean will be very close to the target image.

1.4.1 Differential Evolution

The goal is to build a hybrid algorithm that use the benefits of Particle Filters and Differential Evolution. Before the particles are updated and before the importance weight are calculates, is necessary to run few generations of the DE algorithm. The initial population are the same particles. This will arise to a new set of possible solutions which is expected to have a better fitness value than the one of the initial population. The fitness function of each individual is computed in the same way we compute the importance weight of particles, using the color histogram intersection method. The DE algorithm is shown in algorithm 3.

Algorithm 3 DE Algorithm

Input: Initial population X, C_r and F
$t \leftarrow 0$
while $t <$ maximum number of iterations **do**
  for $i = 1$ to M **do**
    Select r_1, r_2 and r_3 such that $r_1 \neq r_2 \neq r_3 \neq i$
    $u \leftarrow floor(D \cdot rand(0,1))$
    for $j = 1$ to D **do**
      if $rand(0,1) \leq C_r$ or $j = u$ **then**
        $z_j^{(i)} \leftarrow x_j^{(r1)} + F \cdot \left(x_j^{(r2)} - x_j^{(r3)}\right)$
      else
        $z_j^{(i)} \leftarrow x_j^{(i)}$
      end if
    end for
  end for
  for $i = 1$ to M **do**
    if $f\left(Z^{(i)}\right) \geq f\left(X^{(i)}\right)$ **then**
      $X^{(i)} = Z^{(i)}$
    end if
  end for
  return X
end while

1.4.2 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) is an algorithm developed by Kennedy and Eberhart in 1995. This algorithm is motivated from the simulation of social behavior of bird flocking, where each potential solution, called particles, have certain velocity that allow it to "flow" trough the search space.

In the classical PSO, each particle has two state variables, its current position and its current velocity. It is also equipped with a small memory comprising its previous best position. On each iteration the velocity and position of each particle are updated using the following equations:

$$v_i^{(t+1)} = w \cdot v_i^{(t)} + C_1 \cdot \varphi_1 \cdot \left(p_i^{(t)} - x_i^{(t)}\right) + C_2 \cdot \varphi_2 \cdot \left(g^{(t)} - x_i^{(t)}\right) \tag{1.5}$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \tag{1.6}$$

where $x_i^{(t)}$ and $v_i^{(t)}$ is the position and velocity of particle i at time t, $p_i^{(t)}$ is the best particular best of particle i, and $g^{(t)}$ is the global best until time t. The factor w is called the inertia factor. Venter and Sobeiski termed C_1 as "self-confidence" and C_2 as "swarm confidence". Since C_1 has a contribution towards its own experience, an C_2 has a contribution towards motion of the particles in

Algorithm 4 PSO Algorithm

Initialize the location $x^{(0)}$, and velocity $v^{(0)}$ of the particles.
$t \leftarrow 0$
while $t <$ maximum number of iterations **do**
  for $i = 1$ to M **do**
    $v_i^{(t+1)} \leftarrow w \cdot v_i^{(t)} + C_1 \cdot \varphi_1 \cdot \left(p_i^{(t)} - x_i^{(t)}\right) + C_2 \cdot \varphi_2 \cdot \left(g^{(t)} - x_i^{(t)}\right)$
    $x_i^{(t+1)} \leftarrow x_i^{(t)} + v_i^{(t+1)}$
    if $f\left(x_i^{(t+1)}\right) > f\left(p_i^{(t)}\right)$ **then**
      $p_i^{(t+1)} \leftarrow x_i^{(t+1)}$
      if $f\left(x_i^{(t+1)}\right) > f\left(g^{(t)}\right)$ **then**
        $g^{(t+1)} \leftarrow x_i^{(t+1)}$
      else
        $g^{(t+1)} \leftarrow g^{(t)}$
      end if
    else
      $p_i^{(t+1)} \leftarrow p_i^{(t)}$
    end if
  end for
end while

global direction. φ_1 and φ_2 are uniform random numbers within the interval $[0, 1]$. The PSO algorithm is shown in 4.

The PSO-DV Algorithm

The PSO-DV (PSO - Differential Velocity) algorithm uses a differential operator in the velocity update scheme. For this, two particles, x_j and x_k , are chosen randomly and instead of moving to the individual best location, there is a contribution proportional to the differential between these two particles, and unlike the PSO scheme, a particle is actually shifted to a new location only if the new location yields a better fitness value.

The d^{th} velocity component of the target particle i is updated as

$$v_{id}^{(t+1)} = \begin{cases} w \cdot v_{id}^{(t)} + \beta \cdot \delta_d + C_2 \cdot \varphi_2 \cdot \left(g_d^{(t)} - x_{id}^{(t)}\right) & if\ rand_d(0,1) \leq Cr \\ v_{id}^{(t)} & otherwise \end{cases}$$

where $\delta = x_j^{(t)} - x_k^{(t)}$, and β is a scale factor in $[0, 1]$. The PSO-DV algorithm is described in 5.

Algorithm 5 PSO-DV Algorithm

Initialize the location $x^{(0)}$, and velocity $v^{(0)}$ of the particles.
$t \leftarrow 0$
while $t <$ maximum number of iterations **do**
  for $i = 1$ to M **do**
    Select j and k such that $j \neq k \neq i$
    $\delta \leftarrow x_j^{(t)} - x_k^{(t)}$
    for $d = 1$ to D **do**
      if $rand_d(0,1) \leq Cr$ **then**
        $v_{id}^{(t+1)} \leftarrow w \cdot v_{id}^{(t)} + \beta \cdot \delta_d + C_2 \cdot \varphi_2 \cdot \left(g_d^{(t)} - x_{id}^{(t)}\right)$
      else
        $v_{id}^{(t+1)} \leftarrow v_{id}^{(t)}$
      end if
    end for
    $y \leftarrow x_i^{(t)} + v_i^{(t+1)}$
    if $f(y) > f\left(x_i^{(t)}\right)$ **then**
      $x_i^{(t+1)} \leftarrow y$
      if $f\left(x_i^{(t+1)}\right) > f\left(g^{(t)}\right)$ **then**
        $g^{(t+1)} \leftarrow x_i^{(t+1)}$
      else
        $g^{(t+1)} \leftarrow g^{(t)}$
      end if
    else
      $x_i^{(t+1)} \leftarrow x_i^{(t)}$
    end if
  end for
end while

1.4.3 Particle Filter + DE/PSO

After generated a new set of particles using DE/PSO, we update the original set of particles using Particle Filter and we are going to keep the best N particles, where $N < M$, it means, the N particles where bigger importance weight. The rest $M - N$ particles are replaced with the best $M - N$ particles created by DE/PSO. Once we have the new set of particles mix of the particle filter and DE/PSO we proceed to do the resampling using the Low-Variance Sampler method described before. The process is repeated in the next generation until convergence. Algorithm **??** shows the hybrid particle filter algorithm.

One thing that must be considered when using the hybrid algorithm is that this algorithm sacrifices exploration with precision, because the particles with lower weights are removed from the set that goes to the low-variance sampler,

Algorithm 6 Hybrid PF + DE/PSO Algorithm

Input: X_{t-1}, z_t, C_r and F
$X_{DE} \leftarrow DE(X_{t-1}, C_r, F)$
for $i = 1$ to M **do**
  $x_t^{[i]} \sim p\left(x_t | x_{t-1}^{[i]}\right)$
  $w_t^{[i]} = p\left(z_t | x_t^{[i]}\right)$
end for
for $i = 1$ to M **do**
  $w_t^{[i]} = w_t^{[i]} / \sum_{k=1}^{M} w_t^{[k]}$
end for
Sort x_t in ascending order according to their weights.
Sort X_{DE} in descending order according to their weights.
$X_t^{[N+1:M]} \leftarrow X_{DE}^{[N+1:M]}$
for $i = 1$ to M **do**
  draw k with probability $w_t^{[k]}$
  add $x_t^{[k]}$ to X_t
end for

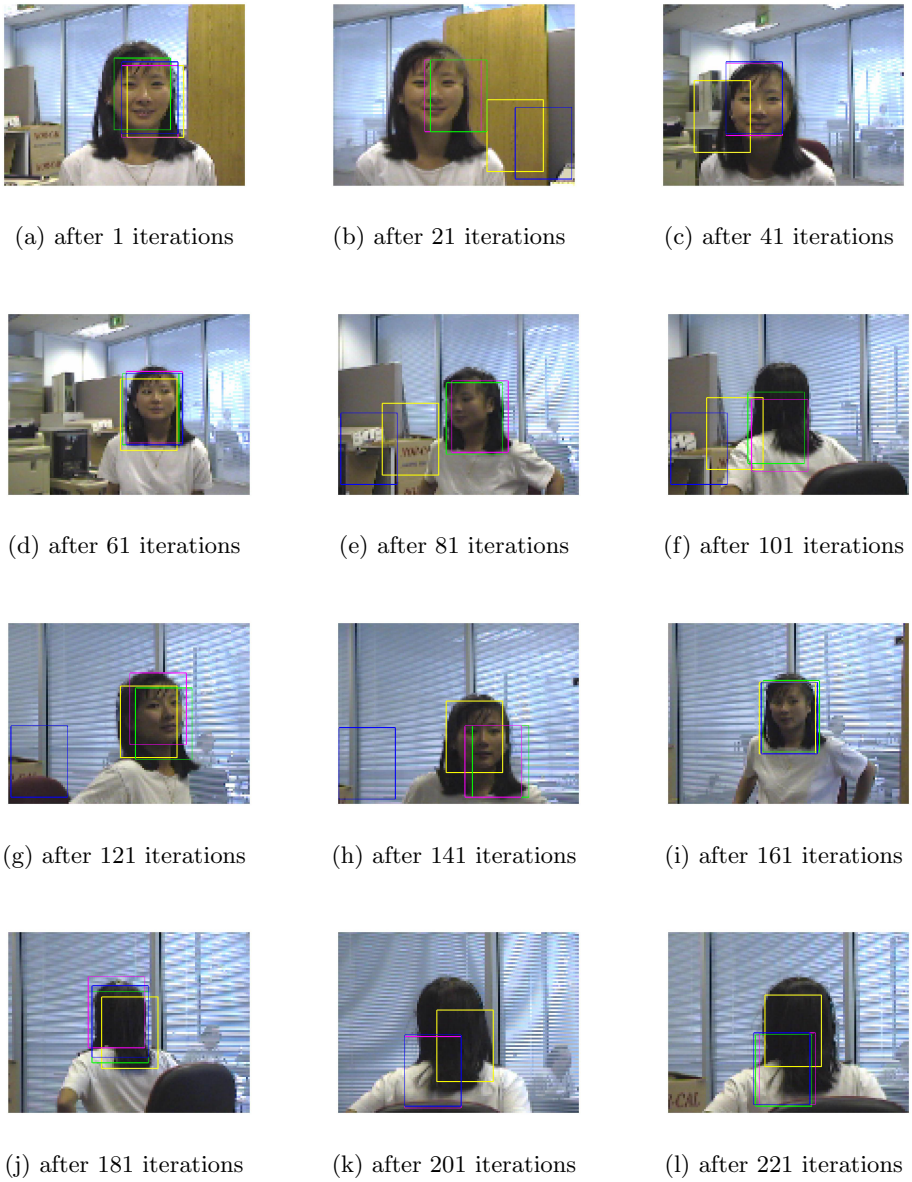
and can occur that the resulting particles get concentrated in a small region, which would be undesirable, specially in the first iterations of the algorithm. For that reason is important to have some variance in move and measurements in the particles, because this can help us to do a better exploration of the search space.

Figure 1.8 shows the mean of the particles in different iterations using the Hybrid PF + DE/PSO algorithm and the PF algorithm. the result thrown by the hybrid algorithm are better than the results obtained using PF alone.

One disadvantage of the hybrid algorithm is its execution time, is computationally more expensive because of the computation required in the DE/PSO algorithm. For this example we used $M = 52$, $N = M/4$, $k = 3$, that represents the number of colors for the histogram method, $C_r = 0.8$ and $F = 0.75$. We run the DE algorithm 5 generations to obtain X_{DE}. In the case of the PSO algorithm, we used $C_1 = 2, C_2 = 2$ and $w = 0.9$. For the $PSO - DV$ we defined $C_r = 0.8$ and $\beta = 0.7$.

As conclusion we can say that PF algorithm is a very powerful tool to estimate states through time, it is very easy to implement and can be used in different applications, such as robot localization (Google self-driving car) and object tracking. Is important to mention that are commonly used in low-dimensional problems, but there are variants of particle filters that have provided solutions for high-dimensional problems. Here we tried to optimize the result of PF by using evolutionary algorithms (DE, PSO), the result obtained was more precise,

but it is computationally more expensive.



(a) after 1 iterations (b) after 21 iterations (c) after 41 iterations

(d) after 61 iterations (e) after 81 iterations (f) after 101 iterations

(g) after 121 iterations (h) after 141 iterations (i) after 161 iterations

(j) after 181 iterations (k) after 201 iterations (l) after 221 iterations

Figure 1.8: PF (yellow), PF+DE (green), PF+PSO (pink) and PF+PSO-DV (blue), in a face tracking application

Bibliography

[1] de Lima, L. and Krohling, R. (2011) *Hybrid Particle Filter with Differential Evolution for Image Tracking*. Online Conference on Soft Computing in Industrial Applications.

[2] Doucet, A., de Freitas, N. and Smith, A. (2010) *Sequential Monte Carlo Methods in Practice*. Springer.

[3] Bishop. C. (2009) *Pattern Recognition and Machine Learning*. Chapter 8. Probabilistic Graphical Models. Springer.

[4] Swagatam, D., Ajith, A. and Amit, K. (2008) *Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives*. Studies in Computational Intelligence (SCI).

[5] Thrun, S., Leiserson, C., Burgard, W. and Fox, D. (2006) *Probabilistic Robotics*. The MIT Press.

[6] Thrun, S. (2002) *Particle Filters in Robotics*. In Proceedings of Uncertainty in AI (UAI) 2002.