MAJOR PROJECT REPORT

ON

OBJECT TRACKING USING PARTICLE FILTER

Submitted in the partial fulfilment
of the requirement for the degree

MASTER OF ENGINEERING
IN
ELECTRONICS &COMMUNICATION

(2009-2012)

Delhi College of Engineering

Guided By:
Mr. Rajesh Rohilla

Submitted By
Suman Yadav

CERTIFICATE

This is to certify that the project report entitled “Histogram Based Particle Filter” has been successfully completed by SUMAN YADAV (39124) in partial fulfilment of the requirement for the award of degree Master of Technology from Delhi College of Engineering for the academic year 2009 – 2012.

Mr. Rajesh Rohilla
Project Guide

ACKNOWLEDGEMENT

I am extremely greatful to Mr. Rajesh Rohilla from Department of Electronics & Communication for providing all the required resources for the successful completion of my Project. I also offer my sincere thanks with humility to Mr. Rajiv Kapoor for providing help whenever needed.

I am thankful for his valuable suggestions & guidance in the preparation of Thesis.

SUMAN YADAV

ABSTRACT

In this project, a particle filter is implemented in a color model based framework to track the moving object in outdoor environment. firstly the initialisation of samples is done in first frame by drawing them randomly on the screen or drawing them based on the region where the object is expected to appear.

Next the samples are predicted based on system model by propagating each sample based on this model. The samples are updated based on the observation model. In this report, we use color distribution of the object as the observation model. Then using the bhattacharya distance, the similarity between the color distribution of the target & the samples can be measured. Based on the bhattacharya distance weight of each sample is measured. The target state estimation is performed based on samples weight. The resampling is performed for the next sample iteration to generate a new sample set. During the resampling sample with a high weight are chosen leading to identical copies, while others with relatively low weights may be ignored & deleted.

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION:

OBJECT TRACKING

Object tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editining. Video tracking can be a time consuming process due to the amount of data that is contained in video.

Objective

The objective of video tracking is to associate target objects in consecutive video frames. The association can be especially difficult when the objects are moving fast relative to the frame rate. Another situation that increases the complexity of the problem is when the tracked object changes orientation over time. For these situations video tracking systems usually employ a motion model which describes how the image of the target might change for different possible motions of the object.

Examples of simple motion models are:

- When tracking planar objects, the motion model is a 2D transformation of an image of the object (e.g. the initial frame).
- When the target is a rigid 3D object, the motion model defines its aspect depending on its 3D position and orientation.

Algorithm

To perform video tracking an algorithm analyzes sequential video frames and outputs the movement of targets between the frames. There are a variety of algorithms, each having strengths and weaknesses. Considering the intended use is important when choosing which algorithm to use. There are two major components of a visual tracking system: target representation and localization , filtering and data association.

Target representation and localization is mostly a bottom-up process. These methods give a variety of tools for identifying the moving object. Locating and tracking the target object successfully is dependent on the algorithm. For example, using blob tracking is useful for identifying human movement because a person's profile changes dynamically. Typically the computational complexity for these algorithms is low. The following are some common target representation and localization algorithms:

Blob tracking: segmentation of object interior (for example blob detec bloction, k-based correlation or optical flow).
Kernel-based tracking (mean-shift tracking): an iterative localization procedure based on the maximization of a similarity measure (Bhattacharyya coefficient).

- Contour tracking: detection of object boundary (e.g. active contours or Condensation algorithm)

- Filtering and data association is mostly a top-down process, which involves incorporating prior information about the scene or object, dealing with object dynamics, and evaluation of different hypotheses. These methods allow the tracking of complex objects along with more complex object interaction like tracking objects moving behind obstructions.Additionally the complexity is increased if the video tracker (also named TV tracker or target tracker) is not mounted on rigid foundation (on-shore) but on a moving ship (off-shore), where typically

an inertial measurement system is used to pre-stabilize the video tracker to reduce the required dynamics and bandwidth of the camera system. The computational complexity for these algorithms is usually much higher. The following are some common filtering algorithms:

- Kalman filter: an optimal recursive Bayesian filter for linear functions subjected to Gaussian noise
- Particle filter: useful for sampling the underlying state-space distribution of nonlinear and non-Gaussian processes.

Three key steps are there in object tracking

- Detection of interesting moving objects
- Tracking of such objects from frame to frame
- Analysis of object tracks to recognize their behaviour

Application of object tracking :

- Motion based recognition, that is, human identification based on gait, automatic object detection, etc;
- Automated surveillance, that is, monitoring a scene to detect suspicious activities or unlikely events;
- Video indexing, that is, automatic annotation and retrieval of the videos in multimedia databases;
- Human-computer interaction, that is, gesture recognition, eye gaze tracking for data input to computers, etc.;
- Traffic monitoring, that is, real-time gathering of traffic statistics to direct traffic flow.
- Vehicle navigation that is, video-based path planning and obstacle avoidance capabilities.

Object representation

In a tracking scenario, an object can be defined as anything that is of interest for further analysis. For instance, boats on the sea, fish inside an aquarium, vehicles on a road, planes in the air, people walking on a road, or bubbles in the water are a set of objects that may be important to track in a specific domain. Objects can be represented by their shapes and appearances. In this section, we will first describe the object shape representations commonly employed for tracking and then address the joint shape and appearance representations.

Points: The object is represented by a point, that is, the centroid by a set of points. In general, the point representation is suitable for tracking objects that occupy small regions in an image.

Primitive geometric shapes: Object shape is represented by a rectangle, ellipse. Though primitive geometric shapes are more suitable for representing simple rigid objects, they are also used for tracking no rigid objects.

Object silhouette and contour: Contour representation defines the boundary of an object. The region inside the contour is called the silhouette of the object.

There are a number of ways to represent the appearance features of objects. Note that shape representations can also be combined with the appearance representations for tracking.

Some common appearance representations in the context of object tracking are:
Probability densities of object appearance*:* The probability density estimates of the object appearance can either be parametric, such as Gaussian

and a mixture of Gaussian or nonparametric, eg. histograms. The probability densities of object appearance features (color, texture) can be computed from the image regions specified by the shape models (interior region of an ellipse or a contour).

Templates*:* Templates are formed using simple geometric shapes. An advantage of a template is that it carries both spatial and appearance information. Templates, however, only encode the object appearance generated from a single view. Thus, they are only suitable for tracking objects whose poses do not vary considerably during the course of tracking.

Feature selection for tracking

Selecting the right features plays a critical role in tracking. In general, the most desirable property of a visual feature is its uniqueness so that the objects can be easily distinguished in the feature space. Feature selection is closely related to the object representation. For example, color is used as a feature for histogram-based appearance representations, while for contour-based representation, object edges are usually used as features. In general, many tracking algorithms use a combination of these features.

The details of common visual features are as follows.

Color*:* The apparent color of an object is influenced primarily by two physical factors, the spectral power distribution of the illuminated, the surface reflectance properties of the object. In image processing, the RGB (red, green, blue) color space is usually used to represent color. However, the RGB space is not a perceptually uniform color space, that is, the differences between the colors in the RGB space do not correspond to the color differences perceived by humans. Additionally, the RGB dimensions are highly correlated while HSV (Hue, Saturation, Value) is an approximately uniform color space. However, these color spaces are sensitive to noise

Edges: Object boundaries usually generate strong changes in image intensities. Edge detection is used to identify these changes. An important property of edges is that they are less sensitive to illumination changes compared to color features. Algorithms that track the boundary of the objects usually use edges as the representative feature.

CHAPTER 2

INTRODUCTION:

POSTERIOR PROBABILITY DISTRIBUTION

In Bayesian statistics, the **posterior probability** of a random event or an uncertain proposition is the conditional probability that is assigned after the relevant evidence is taken into account. Similarly, the **posterior probability distribution** is the distribution of an unknown quantity, treated as a random variable, conditional on the evidence obtained from an experiment or survey.

The posterior probability is the probability of the parameters θgiven the evidence x: $p(\theta / x)$.

It contrasts with the likelihood function, which is the probability of the evidence given the parameters: $p(x/\theta)$.

The two are related as follows:

Let us have a prior belief that the probability distribution function is $p(\theta)$ and observations X with the likelihood $p(x/\theta)$, then the posterior probability is defined as

$$p(\theta/x) = \frac{p(\theta)\,p(x/\theta)}{p(x)}$$

The posterior probability can be written in the memorable form as

$$\text{Posterior probability} \propto \text{Prior probability} \times \text{Likelihood}$$

Example

Suppose there is a mixed school having 60% boys and 40% girls as students. The girl students wear trousers or skirts in equal numbers; the boys all wear trousers. An observer sees a (random) student from a distance; all the

observer can see is that this student is wearing trousers. What is the probability this student is a girl? The correct answer can be computed using Bayes' theorem.

The event *A* is that the student observed is a girl, and the event *B* is that the student observed is wearing trousers. To compute P(*A*|*B*), we first need to know:

- P(*A*), or the probability that the student is a girl regardless of any other information. Since the observer sees a random student, meaning that all students have the same probability of being observed, and the
- percentage of girls among the students is 40%, this probability equals 0.4.
- P(*A*'), or the probability that the student is a boy regardless of any other information (*A*' is the complementary event to *A*). This is 60%, or 0.6.
- P(*B*|*A*), or the probability of the student wearing trousers given that the student is a girl. As they are as likely to wear skirts as trousers, this is 0.5.
- P(*B*|*A*'), or the probability of the student wearing trousers given that the student is a boy. This is given as 1.
- P(*B*), or the probability of a (randomly selected) student wearing trousers regardless of any other information. Since P(*B*) = P(*B*|*A*)P(*A*) + P(*B*|*A*')P(*A*') (law of total probability), this is 0.5×0.4 + 1×0.6 = 0.8.

Given all this information, the probability of the observer having spotted a girl given that the observed student is wearing trousers can be computed by substituting these values in the formula:

$$p(\ A|B) = \frac{p(B|A)\,p(A)}{p(B)}$$ = (0.5 *0.4)/0.8 = 0.25

Calculation

The posterior probability distribution of one random variable given the value of another can be calculated with Bayes' theorem by multiplying the prior

probability distribution by the likelihood function, and then dividing by the normalizing constant, as follows:

$$f_{x|y} = y(x)\frac{f_X(x)L_{x|y=y}(x)}{\int_{-\infty}^{\infty} f_X(x)L_{x|y=y}(x)}$$

gives the posterior probability density function for a random variable *X* given the data *Y* = *y*, where

- $f_X(x)$ is the prior density of *X*,
- $L_{X|Y=y}(x) = f_{Y|X=x}(y)$ is the likelihood function as a function of *x*,
- $\int_{-\infty}^{\infty} f_X(x)L_{x|y=y}(x)\,dx$ is the normalizing constant, and
- $f_{x|y} = y(x)$ is the posterior density of *X* given the data *Y* = *y*.

1.3 BAYESIAN ESTIMATION THEORY

Introduction

In estimation theory and decision theory, a **Bayes estimator** or a **Bayes action** is an estimator or decision rule that minimizes the posterior expected value of a loss function (i.e., the **posterior expected loss**). Equivalently, it maximizes the posterior expectation of a utility function. An alternative way of formulating an estimator within Bayesian statistics is Maximum a posteriori estimation.

Suppose an unknown parameter θ is known to have a prior distribution π. Let $\delta = \delta(x)$ be an estimator of θ (based on some measurements *x*), and let L(θ, δ) be a loss function, such as squared error. The **Bayes risk** of δ is defined as $\mathrm{E}_{\Pi}\{L(\theta, \lambda)\}$, where the expectation is taken over the probability

distribution of θ: this defines the risk function as a function of δ. An estimator δ is said to be a *Bayes estimator* if it minimizes the Bayes risk among all estimators. Equivalently, the estimator which minimizes the posterior expected loss $E\{L(\theta,\delta)|x\}$ *for each x* also minimizes the Bayes risk and therefore is a Bayes estimator.

If the prior is improper then an estimator which minimizes the posterior expected loss *for each x* is called a **generalized Bayes estimator.**

Examples

Minimum mean square error estimation

The most common risk function used for Bayesian estimation is the mean square error (MSE), also called squared error risk. The MSE is defined by

MSE = $\mathrm{E}[(\theta'(x)-\theta)2]$

where the expectation is taken over the joint distribution of $\theta$and x.

Posterior mean

Using the MSE as risk, the Bayes estimate of the unknown parameter is simply the mean of the posterior distribution,

$$\theta'(x) = E[\theta|x] = \int \theta\pi(\theta|x)d\theta)$$

This is known as the *minimum mean square error* (MMSE) estimator. The Bayes risk, in this case, is the posterior variance.

Recursive Bayesian estimation theory

Recursive Bayesian estimation, also known as a **Bayes filter**, is a general probabilistic approach for estimating an unknown probability density function recursively over time using incoming measurements and a mathematical process model.

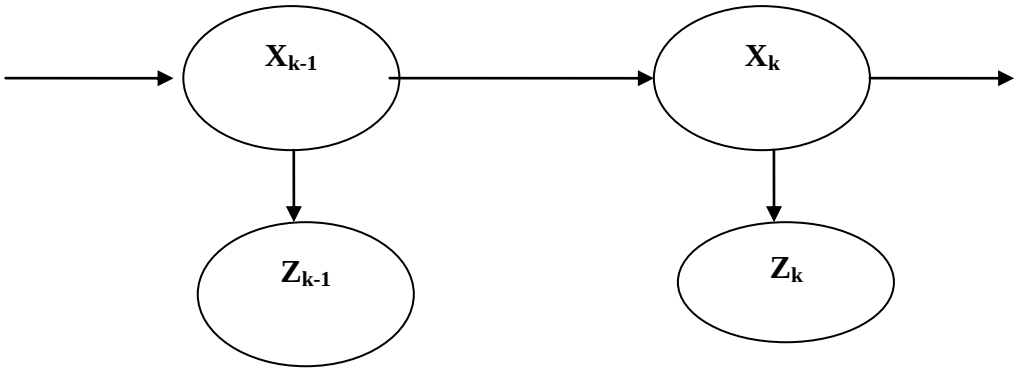
In Robotics

A Bayes filter is an algorithm used in computer science for calculating the probabilities of multiple beliefs to allow a robot to infer its position and orientation. Essentially, Bayes filters allow robots to continuously update their most likely position within a coordinate system, based on the most recently acquired sensor data. This is a recursive algorithm. It consists of two parts: prediction and innovation. If the variables are linear and normally distributed the Bayes filter becomes equal to the Kalman filter.

In a simple example, a robot moving throughout a grid may have several different sensors that provide it with information about its surroundings. The robot may start out with certainty that it is at position (0,0). However, as it moves farther and farther from its original position, the robot has continuously less certainty about its position; using a Bayes filter, a probability can be assigned to the robot's belief about its current position, and that probability can be continuously updated from additional sensor information.

MODEL

The true state x is assumed to be an unobserved Markov process, and the measurements z are the observed states of a Hidden Markov Model (HMM). The following picture presents a Bayesian Network of a HMM.



X_{k-1} **:** Previous State
X_k : Next State

Z_{k-1}: Measured Previos State
Z_k : Next Measured State

Because of the Markov assumption, the probability of the current true state given the immediately previous one is conditionally independent of the other earlier states.

$P(X_k/x_{k-1}, X_{k-2},\ldots\ldots X_0) = p(X_k/X_{k-1})$

Similarly, the measurement at the *k*-th timestep is dependent only upon the current state, so is conditionally independent of all other states given the current state.

$P(z_k/ X_k, X_{k-1}, X_{k-2},\ldots\ldots X_0) = p(X_k/X_{k-1})$

Using these assumptions the probability distribution over all states of the HMM can be written simply as:

$$P(X_0,\ldots\ldots,X_k,Z_1,\ldots\ldots Z_k) = p(X_0) \int_{i=1}^{k} p(z_i / x_i) p(x_i / x_{i-1})$$

However, when using the Kalman filter to estimate the state **x**, the probability distribution of interest is associated with the current states conditioned on the measurements up to the current timestep. (This is achieved by marginalising out the previous states and dividing by the probability of the measurement set.)

This leads to the *predict* and *update* steps of the Kalman filter written probabilistically. The probability distribution associated with the predicted state is the sum (integral) of the products of the probability distribution associated with the transition from the (*k* - 1)-th timestep to the *k*-th and the probability distribution associated with the previous state, over all possible x_{k-1}.

$$p(x_k / z_{k-1}) \quad = \int_{i=1}^{k} p(x_k / x_{k-1}) p(x_{k-1} / z_{k-1}) dx_{k-1}$$

The probability distribution of update is proportional to the product of the measurement likelihood and the predicted state.

$$p(x_k / z_k) \;=\; \frac{p(z_k / x_k) p(x_k / z_{k-1})}{p(z_k / z_{k-1})} \;=\; \alpha p(z_k / x_k) p(x_k / z_{k-1})$$

The denominator

$$p(x_k / z_{k-1}) \quad = \int p(x_k / x_k) p(x_k / z_{k-1}) dx_k$$

is constant relative to x, so we can always substitute it for a coefficient α, which can usually be ignored in practice. The numerator can be calculated and then simply normalized, since its integral must be unitary.

APPLICATION

- Kalman filter a recursive Bayesian filter for multivariate normal distributions
- Particle filter, a sequential Monte Carlo (SMC) based technique, which models the PDF using a set of discrete points
- Grid-based estimators, which subdivide the PDF into a discrete grid.

Seuential bayesian filtering

Sequential Bayesian filtering is the extension of the Bayesian estimation for the case when the observed value changes in time. It is a method to estimate the real value of an observed variable that evolves in time. It involves three steps.

Filtering

when we estimate the *current* value given past observations,

Smoothing

when estimating *past* values given present and past measures, and

Prediction

when estimating a probable *future* value.

.

CHAPTER 3

INTRODUCTION:

MONTE CARLO SIMULATION

Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to compute their results. Monte Carlo methods are often used in computer simulations of physical and mathematical systems. Monte Carlo simulation is a computerized mathematical technique that allows people to account for risk in quantitative analysis and decision making. The technique is used by professionals in such widely disparate fields as finance, project management, energy, manufacturing, engineering, research and development, insur2ance, oil & gas, transportation, and the environment.

Monte Carlo simulation furnishes the decision-maker with a range of possible outcomes and the probabilities they will occur for any choice of action. The technique was first used by scientists working on the atom bomb; it was named for Monte Carlo, the Monaco resort town renowned for its casinos. Since its introduction in World War II, Monte Carlo simulation has been used to model a variety of physical and conceptual systems.

Monte Carlo simulation is categorized as a **sampling method** because the inputs are randomly generated from probability distributions to simulate the process of sampling from an actual population. So, we try to choose a distribution for the inputs that most closely *matches data we already have*, or best represents our current state of knowledge. The data generated from the simulation can be represented as probability distributions (or histograms) or converted to error bars, reliability predictions, tolerance zones, and confidence intervals.

Application

They are widely used in mathematics, for example to evaluate multidimensional definite integrals with complicated boundary conditions

.They are used to model phenomena with significant uncertainty in inputs, such as the calculation of risk in business.
Monte Carlo methods vary, but tend to follow a particular pattern:

1. Define a domain of possible inputs.
2. Generate inputs randomly from a probability distribution over the domain.
3. Perform a deterministic computation on the inputs.
4. Aggregate the results.

For example, consider a circle inscribed in a unit square. Given that the circle and the square have a ratio of areas that is π/4, the value of π can be approximated using a Monte Carlo method:

1. Draw a square on the ground, then inscribe a circle within it.
2. Uniformly scatter some objects of uniform size (grains of rice or sand) over the square.
3. Count the number of objects inside the circle and the total number of objects.
4. The ratio of the two counts is an estimate of the ratio of the two areas, which is π/4. Multiply the result by 4 to estimate π.

In this procedure the domain of inputs is the square that circumscribes our circle. We generate random inputs by scattering grains over the square then perform a computation on each input (test whether it falls within the circle). Finally, we aggregate the results to obtain our final result, the approximation of π.

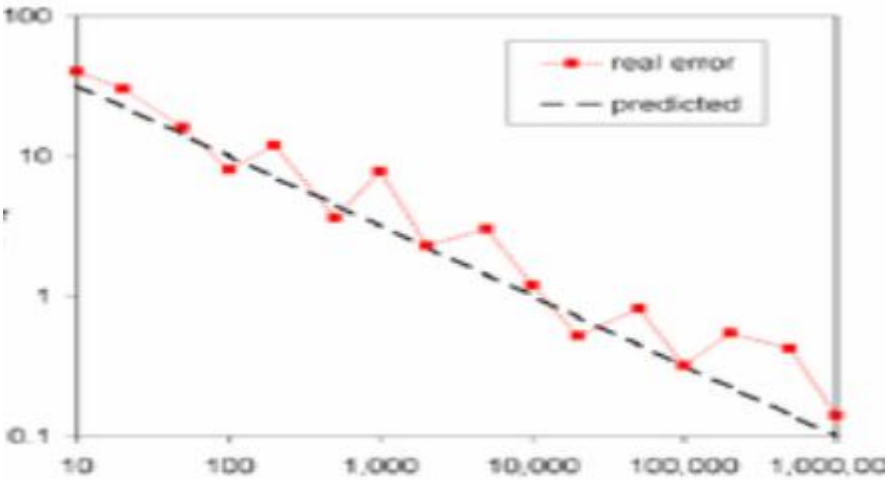
If grains are purposefully dropped into only the center of the circle, they are not uniformly distributed, so our approximation is poor. Second, there should be a large number of inputs. The approximation is generally poor if only a few grains are randomly dropped into the whole square. On average, the approximation improves as more grains are dropped.

Use in mathmatics

In general, Monte Carlo methods are used in mathematics to solve various problems by generating suitable random numbers and observing that fraction of the numbers that obeys some property or properties. The method is useful for obtaining numerical solutions to problems too complicated to solve analytically. The most common application of the Monte Carlo method is Monte Carlo integration.

Integration

Monte-Carlo integration works by comparing random points with the value of the function



X axis = Samples

Y axis = error(%)

Errors reduce by a factor of $1/\sqrt{N}$ where N is the no. of samples taken.

Deterministic numerical integration algorithms work well in a small number of dimensions, but encounter two problems when the functions have many variables. First, the number of function evaluations needed increases rapidly with the number of dimensions. For example, if 10 evaluations provide adequate accuracy in one dimension, then 10^{100} points are needed for 100

dimensions—far too many to be computed. This is called the curse of dimensionality. Second, the boundary of a multidimensional region may be very complicated, so it may not be feasible to reduce the problem to a series of nested one-dimensional integrals. 100 dimensions is by no means unusual, since in many physical problems, a "dimension" is equivalent to a degree of freedom.

Monte Carlo methods provide a way out of this exponential increase in computation time. As long as the function in question is reasonably well-behaved, it can be estimated by randomly selecting points in 100-dimensional space, and taking some kind of average of the function values at these points. By the law of large numbers, this method displays $1/\sqrt{N}$ convergence—i.e., quadrupling the number of sampled points halves the error, regardless of the number of dimensions.

A refinement of this method, known as importance sampling in statistics, involves sampling the points randomly, but more frequently where the integrand is large. To do this precisely one would have to already know the integral, but one can approximate the integral by an integral of a similar function or use adaptive routines such as Stratified sampling, recursive stratified sampling, adaptive umbrella sampling or the VEGAS algorithm.

A similar approach, the quasi-Monte Carlo method, uses low-discrepancy sequences. These sequences "fill" the area better and sample the most important points more frequently, so quasi-Monte Carlo methods can often converge on the integral more quickly.

Variance Reduction

In mathematics, more specifically in the theory of Monte Carlo methods, **variance reduction** is a procedure used to increase the precision of the estimates that can be obtained for a given number of iterations. Every output random variable from the simulation is associated with a variance which limits

the precision of the simulation results. In order to make a simulation statistically efficient, i.e., to obtain a greater precision and smaller confidence intervals for the output random variable of interest, variance reduction techniques can be used. The main ones are: Common random numbers, antithetic variates, control variates, importance sampling and stratified sampling. Under these headings are a variety of specialized techniques; for example particle transport simulations make extensive use of "weight windows" and "splitting/Russian roulette" techniques, which is a form of importance sampling.

Common Random Numbers (CRN)

The common random numbers variance reduction technique is a popular and useful variance reduction technique which applies when we are comparing two or more alternative configurations (of a system) instead of investigating a single configuration. CRN has also been called *Correlated sampling*, *Matched streams* or *Matched pairs*.

CRN requires synchronization of the random number streams, which ensures that in addition to using the same random numbers to simulate all configurations, a specific random number used for a specific purpose in one configuration is used for exactly the same purpose in all other configurations. For example, in queueing theory, if we are comparing two different configurations of tellers in a bank, we would want the (random) time of arrival of the *N*th customer to be generated using the same draw from a random number stream for both configurations.

Underlying principle of the CRN technique

Suppose X_{1j} and X_{2j} are the observations from the first and second configurations on the *j*th independent replication.

We want to estimate

$$\xi = E(X_{1j}) - E(X_{2j}) = \mu_1 - \mu_2$$

If we perform *n* replications of each configuration and let

$$Z_j = X_{1j} - X_{2j} \quad \text{for j=1,2,….,n,}$$

then $E(Z_j) = \xi$ and $Z(n) = \Sigma Z_j / n$ is an unbiased estimator of ξ.

And since the Z_j's are independent identically distributed random variables,

$$\text{Var[Z(n)]} = Var(Z_j)/n$$

In case of independent sampling, i.e., no common random numbers used then $\text{Cov}(X_{1j}, X_{2j}) = 0$. But if we succeed to induce an element of positive correlation between X_1 and X_2 such that $\text{Cov}(X_{1j}, X_{2j}) > 0$, it can be seen from the equation above that the variance is reduced.

It can also be observed that if the CRN induces a negative correlation, i.e., $\text{Cov}(X_{1j}, X_{2j}) < 0$, this technique can actually backfire, where the variance is increased and not decreased (as intended).

IMPORTANCE SAMPLING

Importance sampling is a variance reduction technique that can be used in the Monte Carlo method. The idea behind importance sampling is that certain values of the input random variables in a simulation have more impact on the parameter being estimated than others. If these "important" values are emphasized by sampling more frequently, then the estimator variance can be reduced. Hence, the basic methodology in importance sampling is to choose a distribution which "encourages" the important values. This use of "biased" distributions will result in a biased estimator if it is applied directly in the simulation. However, the simulation outputs are weighted to correct for the use of the biased distribution, and this ensures that the new importance sampling estimator is unbiased. The weight is given by the likelihood ratio, The fundamental issue in implementing importance sampling simulation is the

choice of the biased distribution which encourages the important regions of the input variables. Choosing or designing a good biased distribution is the "art" of importance sampling. The rewards for a good distribution can be huge run-time savings; the penalty for a bad distribution can be longer run times than for a general Monte Carlo simulation without importance sampling.

Basic theory

More formally, let X : $\Omega \to R$ be a random variable in some probability space (Ω, F, P). We wish to estimate the expected value of *X* under *P*. If we have random samples $x_1, \ldots, x_n$, generated according to *P*, then an empirical estimate of **E**[*X;P*] is

$$\hat{E}_n[X;P] = \frac{1}{n}\sum_{i=1}^{n} x_i$$

The basic idea of importance sampling is to change the probability *P* so that the estimation of **E**[*X;P*] is easier. Choose a random variable $L \geq 0$ such that **E***[L;P]=1* and that *P*-almost everywhere $L(\omega) \neq 0$. The variate *L* defines another probability $P^{(L)} = L\,P$that satisfies

$$\text{E[X;P]} = E[\frac{X}{L}; P^{(L)}]$$

The variable *X/L* will thus be sampled under $P^{(L)}$ to estimate $\mathbf{E}[X;P]$as above. This procedure will improve the estimation when

$$E \setminus Var[\frac{X}{L}; P^{(L)}] \; < \; \text{Var [X;P]}$$

Another case of interest is when *X/L* is easier to sample under $P^{(L)}$ than *X* under *P*.

When *X* is of constant sign over Ω, the best variable *L* would clearly be

L* = $\frac{X}{E[X;P]} \geq 0$, so that *X/L** is the searched constant **E***[X;P]* and a single sample under $P^{(L^*)}$ suffices to give its value. Unfortunately we cannot take that choice, because **E***[X;P]* is precisely the value we are looking for! However this theoretical best case *L** gives us an insight into what importance sampling does:

$$\forall a \in R, P^{(L^*)}(X \in [a; a+da]) = \int_{\omega \in \{X \in [a;a+da]\}} \frac{X(\omega) dP(\omega)}{E[X;P]} = \frac{1}{E[X;P]}$$

to the right, a P(X $\in$ [a; a+da]) is one of the infinitesimal elements that sum up to **E***[X;P]* = $\int_{a \to -\infty}^{a \to \infty} aP(X \in [a; a+da])$

therefore, a good probability change $P^{(L)}$ in importance sampling will redistribute the law of *X* so that its samples' frequencies are sorted directly according to their weights in E*[X;P]*. Hence the name "importance sampling."Note that whenever $P$is the uniform distribution and $\Omega = \mathbb{R}$, we are just estimating the integral of the real function X:R$\to$R so the method can also be used for estimating simple integral.

Application to probabilistic inference

Such methods are frequently used to estimate posterior densities or expectations in state and/or parameter estimation problems in probabilistic models that are too hard to treat analytically, for example in Bayesian networks.

Mathematical approach

Consider estimating by simulation the probability $p_t$of an event $X \geq t$, where $X$is a random variable with distribution$F$ and probability density function f(x) = F'(x), where prime denotes derivative. A K-length independent and identically distributed (i.i.d.) sequence $X_i$is generated from the distribution F, and the number $k_t$of random variables that lie above the

threshold t are counted. The random variable k_t is characterized by the Binomial distribution

$$P(k_t = k) = \binom{K}{k} p_t^k (1-p_t)^{K-k} \qquad k = 0,1,\ldots\ldots K$$

One can show that $E\left[k_t / K\right] = p_t$, and $\text{var}\left[k_t / K\right] = p_t (1- p_t)/K$, so in the limit $K \to \infty$ we are able to obtain p_t. Note that the variance is low if $p_t \approx 1$. Importance sampling is concerned with the determination and use of an alternate density function f_* (for X), usually referred to as a biasing density, for the simulation experiment. This density allows the event $X \geq t$ to occur more frequently, so the sequence lengths Kgets smaller for a given estimator variance. Alternatively, for a given K, use of the biasing density results in a variance smaller than that of the conventional Monte Carlo estimate. From the definition of p_t, we can introduce f_* as below.

$$p_t = E\left[1(X \geq t)\right]$$

$$= \int 1\left(X \geq t \frac{f(x)}{f_*(x)} f_*(x) dx \right)$$

$$= E_x\left[1(X \geq t) W(x)\right]$$

where

$$W(\circ) \equiv \frac{f(x)}{f_*(x)}$$

is a likelihood ratio and is referred to as the weighting function. The last equality in the above equation motivates the estimator.

$$\hat{p}_t = \frac{1}{K} \sum_{i=1}^{K} 1(X_i \geq t) W(X_i), \qquad X_i \sim f_*$$

This is the importance sampling estimator of p_t and is unbiased. That is, the estimation procedure is to generate i.i.d. samples from f_* and for each sample which exceeds t, the estimate is incremented by the weight W evaluated at the sample value. The results are averaged over K trials. The variance of the importance sampling estimator is easily shown to be

$$\mathrm{var}_* = \frac{1}{K}\mathrm{var}_*\left[1(X \geq t)W(x)\right]$$

$$= \frac{1}{K}\left\{\mathrm{E}_*\left[1(X \geq t)^2 W^2(X)\right] - p^2{}_t\right\}$$

$$= \frac{1}{K}\left\{\mathrm{E}\left[1(X \geq t)W(x)\right] - p^2{}_t\right\}$$

Now, the importance sampling problem then focuses on finding a biasing density f_* such that the variance of the importance sampling estimator is less than the variance of the general Monte Carlo estimate. For some biasing density function, which minimizes the variance, and under certain conditions reduces it to zero, it is called an optimal biasing density function.

Conventional biasing methods

Although there are many kinds of biasing methods, the following two methods are most widely used in the applications of importance sampling.

Scaling

Shifting probability mass into the event region $X \geq t$ by positive scaling of the random variable X with a number greater than unity has the effect of increasing the variance (mean also) of the density function. This results in a heavier tail of the density, leading to an increase in the event probability. Scaling is probably one of the earliest biasing methods known and has been

extensively used in practice. It is simple to implement and usually provides conservative simulation gains as compared to other methods.

In importance sampling by scaling, the simulation density is chosen as the density function of the scaled random variable aX, where usually $a > 1$for tail probability estimation. By transformation,

$$f_*(x) = \frac{1}{a} f\left(\frac{x}{a}\right)$$

and the weighting function is

$$W(x) = a \frac{f(x)}{f\left(\frac{x}{a}\right)}$$

While scaling shifts probability mass into the desired event region, it also pushes mass into the complementary region $X < t$which is undesirable. If X is a sum of $n$random variables, the spreading of mass takes place in an n dimensional space. The consequence of this is a decreasing importance sampling gain for increasing n, and is called the dimensionality effect.

Translation

Another simple and effective biasing technique employs translation of the density function (and hence random variable) to place much of its probability mass in the rare event region. Translation does not suffer from a dimensionality effect and has been successfully used in several applications relating to simulation of digital communication systems. It often provides better simulation gains than scaling. In biasing by translation, the simulation density is given by

$$f_*(x) = f(x-c), c\rangle 0$$

where c is the amount of shift and is to be chosen to minimize the variance of the importance sampling estimator.

Effects of system complexity

- The fundamental problem with importance sampling is that designing good biased distributions becomes more complicated as the system complexity increases. Complex systems are the systems with long memory since complex processing of a few inputs is much easier to handle.

In principle, the importance sampling ideas remain the same in these situations, but the design becomes much harder. A successful approach to combat this problem is essentially breaking down a simulation into several smaller, more sharply defined subproblems. Then importance sampling strategies are used to target each of the simpler subproblems. Examples of techniques to break the simulation down are conditioning and error-event simulation (EES) and regenerative simulation.

Evaluation of importance sampling

In order to identify successful importance sampling techniques, it is useful to be able to quantify the run-time savings due to the use of the importance sampling approach. The performance measure commonly used is $\sigma^2_{MC}/\sigma^2_{IS}$, and this can be interpreted as the speed-up factor by which the importance sampling estimator achieves the same precision as the MC estimator. This has to be computed empirically since the estimator variances are not likely to be analytically possible when their mean is intractable. efficiency.

Variance cost function

Variance is not the only possible cost function for a simulation, and other cost functions, such as the mean absolute deviation, are used in various statistical applications. Nevertheless, the variance is the primary cost function

addressed in the literature, probably due to the use of variances in confidence intervals and in the performance measure $\sigma^2_{MC} / \sigma^2_{IS}$.

An associated issue is the fact that the ratio $\sigma^2_{MC} / \sigma^2_{IS}$. overestimates the run-time savings due to importance sampling since it does not include the extra computing time required to compute the weight function. Hence, some people evaluate the net run-time improvement by various means. Perhaps a more serious overhead to importance sampling is the time taken to devise and program the technique and analytically derive the desired weight function.

CHAPTER 4

INTRODUCTION:

PARTICLE FILTER

Particle filter, also known as a sequential Monte Carlo method (SMC), is a sophisticated model estimation technique based on simulation. Particle filters are usually used to estimate Bayesian models in which the latent variables are connected in a Markov chain similar to a hidden Markov model (HMM), but typically where the state space of the latent variables is continuous rather than discrete, and not sufficiently restricted to make exact inference tractable (as, for example, in a linear dynamical system, where the state space of the latent variables is restricted to Gaussian distributions and hence exact inference can be done efficiently using a Kalman filter). In the context of HMMs and related models, "filtering" refers to determining the distribution of a latent variable at a specific time, given all observations up to that time; particle filters are so named because they allow for approximate "filtering" (in the sense just given) using a set of "particles" (differently weighted samples of the distribution).

A dynamic system can be modeled with two equations:

3.1 State Transition or Evolution Equation

$X_k = f_k(x_{k-1}, u_{k-1}, v_{k-1})$

$f(\cdot, \cdot, \cdot)$ = evolution function (possible non-linear)

x_k, x_{k-1} = current and previous state

v_{k-1} = state noise (usually not Gaussian)

u_{k-1} = known input

Measurement Equation

$z_k = h_k(x_k, u_k, n_k)$

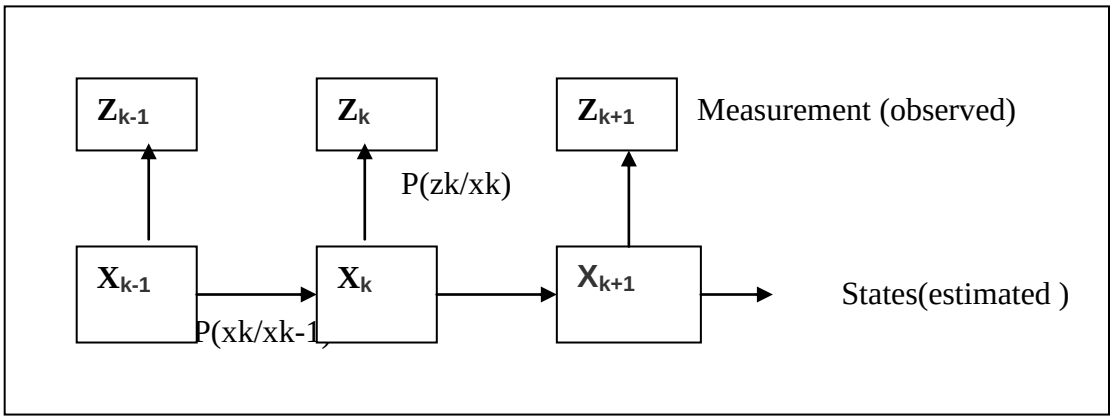
$h(\cdot, \cdot, \cdot)$ = measurement function (possible non-linear)

z_k = measurement

x_k = state

n_k = measurement noise (usually not Gaussian)

u_k = known input



X_k = current state

X_{k-1} = previous state

X_{k+1} = next state

Z_k = corresponding observed state

$P(x_k/x_{k-1})$ = state transition probability for k>0

$P(z_k/x_k)$ = observation are conditionally dependent

Bayesian Filters

Estimating the Posterior

Bayesian approach: We attempt to construct the posterior pdf of the state given all measurements. It can be termed a complete solution to the estimation

Problem because all available information is used; from the pdf, an optimal estimate can theoretically be found for any criterion in detail: We seek estimates of X_k based on all available measurements up to time k (abbreviated as $Z_{1:k}$) by constructing the posterior $p(X_k|Z_{1:k})$.

Assumption: initial state pdf (prior) $p(\mathbf{X}0)$ is given

The Use of Knowing the Posterior

Let $f_k : IR^{(k+1)\times nx}$ IR be any arbitrary (integrable) function that can depend on all components of the state **x** on the whole trajectory in state-space.

Examples: This function can be an estimator for the current state or for future observations.
Then we can compute its expectation using

$$E[\, f_k(\mathbf{X}_{0:k})] = \int f(\mathbf{X}_{0:k}) p(\mathbf{X}_{0:k}|\mathbf{Z}_{1:k}) d\mathbf{x}_{0:k}$$

MMSE estimate of state: X' = $E[\mathbf{X}_k]$.

Recursive Filters

recursive filters (i.e. sequential update of previous estimate) means batch processing (computation with all data in one step). It is not only faster: allow on-line processing of data. It has lower storage costs, rapid adaption to changing signals characteristics)

It essentially consist of two steps:

Prediction step: $p(\mathbf{X}_{k-1}|\mathbf{Z}_{1:k-1}) = p(\mathbf{X}_k|\, \mathbf{Z}_{1:k-1})$
(Usually deforms / translates / spreads state PDF due to noise)

Update step: $p(\mathbf{X}_k|\, \mathbf{Z}_{1:k-1})$, $\mathbf{Z}_{1:k} = p(\mathbf{X}_k|\, \mathbf{Z}_{1:k})$
(Combines likelihood of current measurement with predicted state; usually concentrates state PDF)

General Prediction-Update Framework

Assume that PDF $p(\mathbf{X}_{k-1}|\, \mathbf{Z}_{1:k-1})$ is available at time $k-1$.

Prediction step:

$$p(\mathbf{X}_k|\, \mathbf{Z}_{1:k-1}) = \int p(\mathbf{X}_k|\, \mathbf{X}_{k-1}) p(\mathbf{X}_{k-1}|\, \mathbf{Z}_{1:k-1}) d\, \mathbf{x}_{k-1} \quad (1)$$

This is the prior of the state $\mathbf{X}_k$ at time k without knowledge of the measurement $\mathbf{Z}k$, i.e. the probability given only previous measurements.

Update step: (compute posterior pdf from predicted prior pdf and new measurement)

$p(\mathbf{X}_k|\ \mathbf{Z}_{1:k}) = p(\mathbf{Z}_k|\ \mathbf{X}_k)p(\mathbf{X}_k|\ \mathbf{Z}_{1:k-1})/\ p(\mathbf{Z}_k|\ \mathbf{Z}_{1:k-1})$ (2)

Let us prove formula (2) (just in order to train calculations
with joint and conditional probabilities. . .)

$p(\mathbf{X}_k|\ \mathbf{Z}_{1:k})$

$= p(\mathbf{z}1{:}k|\ \mathbf{X}_k)p(\mathbf{X}_k)/p(\mathbf{Z}_{1:k})$

$=p(\mathbf{Z}_{1:k-1},\ \mathbf{Z}_{1:k-1}|\ \mathbf{X}_k)p(\mathbf{X}_k)/p(\mathbf{z}k,\mathbf{z}1{:}k-1)$

$= p(\mathbf{Z}_{1:k}|\ \mathbf{Z}_{1:k-1},\ \mathbf{X}_k)p(\mathbf{Z}_{1:k-1}|\ \mathbf{X}_k)p(\mathbf{X}_k)\ /p(\mathbf{Z}_{1:k}|\ \mathbf{Z}_{1:k-1})p(\mathbf{Z}_{1:k-1}))$

$= p(\mathbf{Z}_{1:k}|\ \mathbf{Z}_{1:k-1},\ \mathbf{X}_k)p(\mathbf{X}_k|\ \mathbf{Z}_{1:k-1})p(\mathbf{Z}_{1:k-1})p(\mathbf{X}_k)\ /p(\mathbf{Z}_{1:k}|\ \mathbf{Z}_{1:k-1})p(\mathbf{Z}_{1:k-1})p(\mathbf{X}_k)$

$=p(\mathbf{Z}_{1:k}|\ \mathbf{X}_k)p(\mathbf{X}_k|\ \mathbf{Z}_{1:k-1})/\ p(\mathbf{Z}_{1:k}|\ \mathbf{Z}_{1:k-1})$

(independence of observations; cancelling out terms)

The Structure of the Update Equation

$$p(\mathbf{x}_k|\ \mathbf{z}_{1:k}) = p(\mathbf{z}_{1:k}|\ \mathbf{x}_k)\ \cdot\ p(\mathbf{x}_k|\ \mathbf{z}_{1:k-1})/\ p(\mathbf{z}_{1:k}|\ \mathbf{z}_{1:k-1})$$

posterior = likelihood · prior/ evidence

- Prior: given by prediction equation
- Likelihood: given by observation model
- Evidence: the normalizing constant in the denominator

$p(\mathbf{Z}_{1:k}|\ \mathbf{Z}_{1:k-1}) = \int p(\mathbf{Z}_{1:k}|\ \mathbf{X}_k)p(\mathbf{X}_k|\ \mathbf{Z}_{1:k-1})d\ \mathbf{X}_k$

This theoretically allows an optimal Bayesian solution (in the sense of computing the posterior pdf).

NOTE

If we cannot solve the integrals required for a Bayesian recursive filter analytically. we represent the posterior probabilities by a set of randomly chosen weighted samples in particle filter.

Sequential Importance Sampling

Sequential importance resampling (SIR), the original particle filtering algorithm is a very commonly used particle filtering algorithm, which approximates the filtering distribution $p\left(x_k \left| y_{0,\ldots\ldots\ldots,} y_k \right.\right)$by a weighted set of P particles

$\left\{\left(w_k^{(L)}, x_k^{(L)}\right): L \in \{1,\ldots\ldots\ldots,P\}\right\}$

The *importance weights* $W_k^{(L)}$ are approximations to the relative posterior probabilities (or densities) of the particles such that $\sum_{L=1}^{P} W_k^{(L)} = 1.$

SIR is a sequential (i.e., recursive) version of importance sampling. As in importance sampling, the expectation of a function $f(\cdot)$can be approximated k

$$\int f(x_k) p\left(x_k \left| y_{0,\ldots\ldots\ldots,} y_k \right.\right) dx_k \approx \sum_{L=1}^{P} W^{(L)} f\left(x_k^{(L)}\right)$$

choice of the *proposal distribution*

$$\Pi\left(x_k \left| x_{0:k-1}, y_{0:k}\right.\right)$$

The *optimal proposal distribution* is given as the *target distribution*

$$\Pi\left(x_k \left| x_{0:k-1}, y_{0:k}\right.\right) = p\left(x_k \left| x_{k-1}, y_k\right.\right)$$

However, the transition prior is often used as importance function, since it is easier to draw particles (or samples) and perform subsequent importance weight calculations:

$$\Pi\left(x_k \middle| x_{0:k-1}, y_{0:k}\right) = p\left(x_k \middle| x_{k-1}\right)$$

Resampling is used to avoid the problem of degeneracy of the algorithm, that is, avoiding the situation that all but one of the importance weights are close to zero. The performance of the algorithm can be also affected by proper choice of resampling method.

A single step of sequential importance resampling is as follows:

- For L = 1,…….,P draw samples from the *proposal distribution*
  $x_k^{(L)} \approx \Pi\left(x_k \middle| x^{(L)}_{0:k-1}, y_{0:k}\right)$

- For L = 1,…….,P update the importance weights up to a normalizing constant:

$$W_k^{(L)} = W_{k-1}^{(L)} \frac{p\left(y_k \middle| x_k^{(L)}\right) p\left(x_k^{(L)} \middle| x_{k-1}^{(L)}\right)}{\Pi\left(x_k^{(L)} \middle| x_{0:k-1}^{(L)}, y_{0:k}\right)}$$

  Note that when we use the transition prior as the importance function, $\Pi\left(x_k^{(L)} \middle| x_{0:k-1}^{(L)}, y_{0:k}\right) = p\left(x_k^{(L)} \middle| x_{k-1}^{(L)}\right)$, this simplifies to the following :

  $W'^{(L)}_k = W_{k-1}^{(L)}\ p\left(y_k \middle| x_k^{(L)}\right)$

- For L = 1,……..,P compute the normalized importance weights:

$$W_k^{(L)} = \frac{W'^{(L)}_k}{\sum_{j=1}^{P} W'^{(L)}_k}$$

- Compute an estimate of the effective nu mber of particles as

$$\hat{N}_{eff} = \frac{1}{\sum_{L=1}^{P} \left(W_k^{(L)}\right)^2}$$

- If the effective number of particles is less than a given threshold

$\hat{N}_{eff} < N_{thr}$ then perform resampling:

a) Draw P particles from the current particle set with probabilities proportional to their weights. Replace the current particle set with this new one.

b) For $L = 1, \ldots, P$ set $W_k^{(L)} = 1/P$.

The term Sampling Importance Resampling is also sometimes used when referring to SIR filters.

CHAPTER 4

INTRODUCTION:

RGB COLOR MODEL

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors.

RGB is a device-dependent color model: different devices detect or reproduce a given RGB value differently, since the color elements (such as phosphors or dyes) and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time. Thus an RGB value does not define the same *color* across devices without some kind of color management.

Typical RGB input devices are color TV and video cameras, image scanners, and digital cameras. Typical RGB output devices are TV sets of various technologies (CRT, LCD, plasma, etc.), computer and mobile phone displays, video projectors, multicolor LED displays, and large screens such as JumboTron, etc. Color printers, on the other hand, are not RGB devices, but subtractive color devices (typically CMYK color model).

Addictive primary color

To form a color with RGB, three colored light beams (one red, one green, and one blue) must be superimposed (for example by emission from a black

screen, or by reflection from a white screen). Each of the three beams is called a component of that color, and each of them can have an arbitrary intensity, from fully off to fully on, in the mixture.

The RGB color model is additive in the sense that the three light beams are added together, and their light spectra add, wavelength for wavelength, to make the final color's spectrum.

Zero intensity for each component gives the darkest color (no light, considered the black), and full intensity of each gives a white; the quality of this white depends on the nature of the primary light sources, but if they are properly balanced, the result is a neutral white matching the system's white point. When the intensities for all the components are the same, the result is a shade of gray, darker or lighter depending on the intensity. When the intensities are different, the result is a colorized hue, more or less saturated depending on the difference of the strongest and weakest of the intensities of the primary colors employed.

When one of the components has the strongest intensity, the color is a hue near this primary color (reddish, greenish, or bluish), and when two components have the same strongest intensity, then the color is a hue of a *secondary* color (a shade of cyan, magenta or yellow). A secondary color is formed by the sum of two primary colors of equal intensity: cyan is green+blue, magenta is red+blue, and yellow is red+green. Every secondary color is the complement of one primary color; when a primary and its complementary secondary color are added together, the result is white: cyan complements red, magenta complements green, and yellow complements blue.

The RGB color model itself does not define what is meant by *red*, *green*, and *blue* colorimetrically, and so the results of mixing them are not specified as absolute, but relative to the primary colors. When the exact chromaticities of the red, green, and blue primaries are defined, the color model then becomes an absolute color space, such as sRGB or Adobe RGB.

Numerical Representation

A color in the RGB color model is described by indicating how much of each of the red, green, and blue is included. The color is expressed as an RGB triplet (*r*,*g*,*b*), each component of which can vary from zero to a defined maximum value. If all the components are at zero the result is black; if all are at maximum, the result is the brightest representable white.

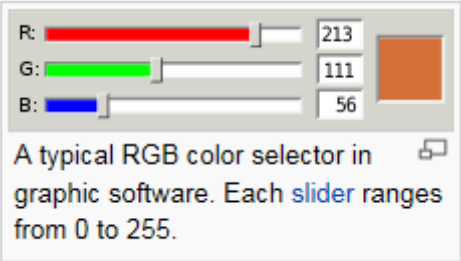
These ranges may be quantified in several different ways:

- From 0 to 1, with any fractional value in between. This representation is used in theoretical analyses, and in systems that use floating-point representations.
- Each color component value can also be written as a percentage, from 0% to 100%.
- In computers, the component values are often stored as integer numbers in the range 0 to 255, the range that a single 8-bit byte can offer. These are often represented as either decimal or hexadecimal numbers.
- High-end digital image equipment are often able to deal with larger integer ranges for each primary color, such as 0..1023 (10 bits), 0..65535 (16 bits) or even larger, by extending the 24-bits (three 8-bit values) to 32-bit, 48-bit, or 64-bit units (more or less independent from the particular computer's word size).

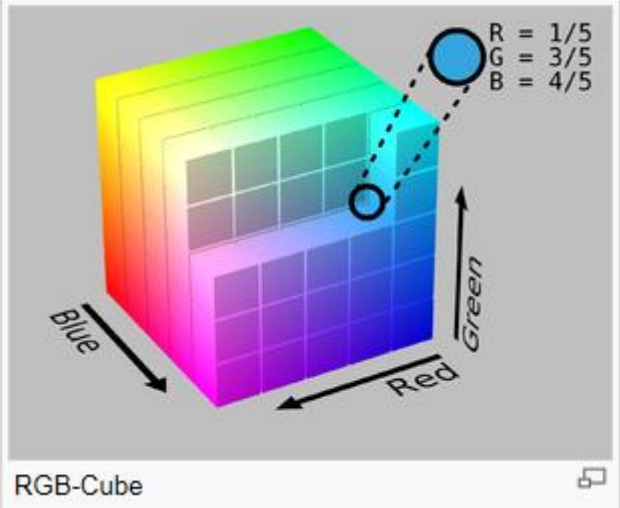
For example, brightest saturated **red** is written in the different RGB notations as:

Notation	RGB triplet
Arithmetic	(1.0, 0.0, 0.0)
Percentage	(100%, 0%, 0%)
Digital 8-bit per channel	(255, 0, 0) or sometimes #FF0000 (hexadecimal)
Digital 16-bit per channel	(65535, 0, 0)

In many environments, the component values within the ranges are not managed as linear (that is, the numbers are nonlinearly related to the intensities that they represent), as in digital cameras and TV broadcasting and receiving due to gamma correction, for example. Linear and nonlinear transformations are often dealt with via digital image processing. Representations with only 8 bits per component are considered sufficient if gamma encoding is used.



A typical RGB color selector in graphic software. Each slider ranges from 0 to 255.



RGB-Cube

Object tracking using Particle filter

The particle filter is a sequential Monte Carlo method used for Bayes filtering Point mass, or particles, with corresponding weights are used to form an approximation of a probability density function (PDF). The particles are propagated over time by Monte Carlo simulation to obtain new particles and weights (usually as new information are received), hence forming a series of PDF approximations over time. Conceptually, a particle filter-based tracker maintains a probability distribution over the state (location, scale, etc.) of the object being tracked. Particle filters represent this distribution as a set of weighted samples, or particles. Each particle represents a possible instantiation of the state of the object. In other words, each particle is a guess representing one possible location of the object being tracked. The set of particles contains more weight at locations where the object being tracked is more likely to be. This weighted distribution is propagated through time using a set of equations known as the Bayesian filtering equations, and we can determine the trajectory of the tracked object.
As a Bayesian operator Particle filter has two main steps:

- Prediction is done by propagating the sample based on system model.
- The update step is done by measuring the weights of sample based on the observation model.

The implementation of the particle filter can be described as follows

Particle initialization

Starting with a weighted set of samples at k-1 $\left\{X^{i}_{k-1}, \Pi^{i}_{k-1}, i = 1:N\right\}$ approximately distributed according to $p(X_{k-1}/y_{1:k-1})$ as initial distribution p(x0), new samples are generated from a suitable proposal distribution, which may depend on the previous state & the new measurement.

Prediction step

Using the probabilistic system transition model $p(\mathbf{X}_k / \mathbf{X}_{k-1})$, the particle are predicted at time k. it is done by propagating each particle based on the transition or system model.

$$X_{k+1} = f_k(\mathbf{X}_k, W_k) = p(\mathbf{X}_k / \mathbf{X}_{k-1})$$

Update step

To maintain a consistent sample, the new importance weights are set to

$$\Pi_k^i = \Pi_{k-1}^i \frac{p\left(y_k \middle| x_k^i\right) p\left(x_k^i \middle| x_{k-1}^i\right)}{q\left(x_k \middle| x_{1:k-1}, y_{1:k}\right)}$$

It is done by measuring the likelihood of each sample based on the observation model.

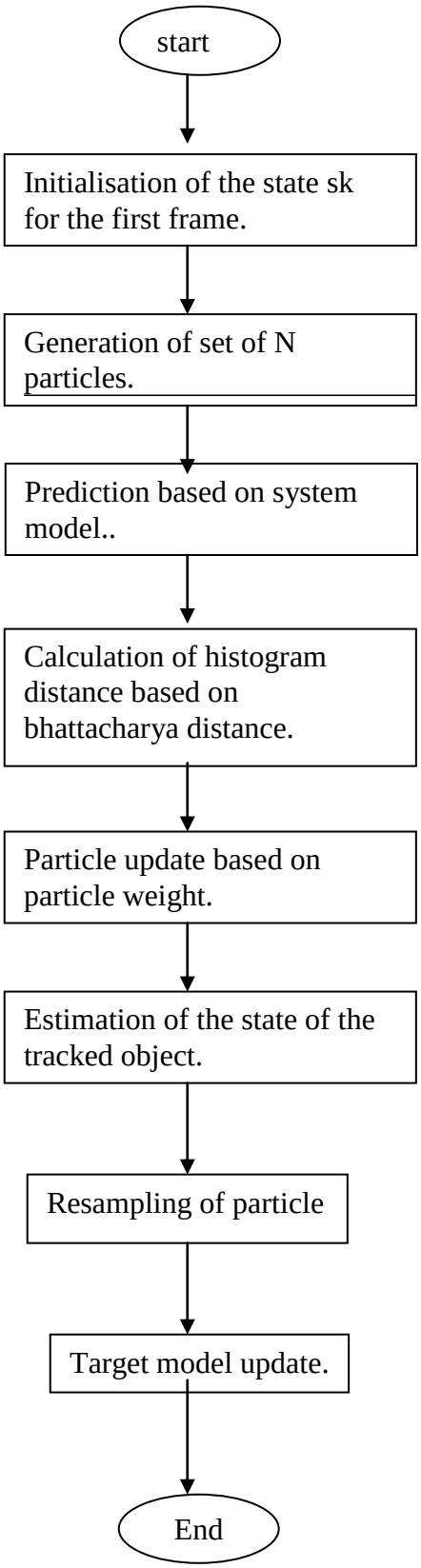
Resample

This step is performed to generate a new sample set according to their weights for the next iteration. The resample step will decrease the no. of samples with low weights & will increase the no. of high weight sample. The new particle set is resampled using normalized weights as probabilities. This sample set represents the posterior at time k, $p\left(x_k \middle| y_{1:k}\right)$.

Then the expectations can be approximated as

$$\mathrm{E}_p\left(X_k \middle| y_{1:k}\right) = \sum_{i=1}^{N} \Pi^i{}_k \, x_k^i$$

Flow of procedure



Color Histogram

A color histogram is a representation of the distribution of colors in an image. For digital images, a color histogram represents the number of pixels that have colors in each of a fixed list of color ranges, that span the image's color space, the set of all possible colors.

The color histogram can be built for any kind of color space, although the term is more often used for three-dimensional spaces like RGB or HSV. For monochromatic images, the term **intensity histogram** may be used instead. For multi-spectral images, where each pixel is represented by an arbitrary number of measurements (for example, beyond the three measurements in RGB), the color histogram is *N*-dimensional, with N being the number of measurements taken. Each measurement has its own wavelength range of the light spectrum, some of which may be outside the visible spectrum.

If the set of possible color values is sufficiently small, each of those colors may be placed on a range by itself; then the histogram is merely the count of pixels that 0have each possible color. Most often, the space is divided into an appropriate number of ranges, often arranged as a regular grid, each containing many similar color values. The color histogram may also be represented and displayed as a smooth function defined over the color space that approximates the pixel counts.

A histogram of an image is produced first by discretization of the colors in the image into a number of bins, and counting the number of image pixels in each bin. For example, a Red–Blue chromaticity histogram can be formed by first normalizing color pixel values by dividing RGB values by R+G+B, then quantizing the normalized R and B coordinates into N bins each. A two-dimensional histogram of Red-Blue chromaticity divide in to four bins (*N*=4) might yield a histogram that looks like this table:

		red			
		0-63	64-127	128-191	192-255
blue	0-63	43	78	18	0
	64-127	45	67	33	2
	128-191	127	58	25	8
	192-255	140	47	47	13

HISTOGRAM EQUALISATION

This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

The method is useful in images with backgrounds and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed. A key advantage of the method is that it is a fairly straightforward technique and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered. The calculation is not computationally intensive. A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal.

Consider a discrete grayscale image {*x*} and let n_i be the number of occurrences of gray level *i*. The probability of an occurrence of a pixel of level *i* in the image is

$$p_x(i) = p(x = i) = \frac{n_i}{L} \qquad 0< i< L$$

L being the total number of gray levels in the image, *n* being the total number of pixels in the image, and $p_x(i)$ being in fact the image's histogram for pixel value i, normalized to [0,1].

Let us also define the *cumulative distribution function* corresponding to p_x as

$$cdf_x = \sum_{j=0}^{i} p_x(j)$$

which is also the image's accumulated normalized histogram.

We would like to create a transformation of the form $y = T(x)$ to produce a new image {*y*}, such that its CDF will be linearized across the value range, i.e.

cdf_x (i) = ik

for some constant *K*. The properties of the CDF allow us to perform such a transform ; it is defined as

y = T(x) = cdf_x (x)

Notice that the T maps the levels into the range [0,1]. In order to map the values back into their original range, the following simple transformation needs to be applied on the result:

$$y' = y \bullet (\max\{x\} - \min\{x\}) + \min\{x\}$$

eg. There is a 8 bit gray scale image (0-256) with its intensity level shown

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

determine its histogram & pixel count that have 0 value has been removed.

Value	Count	Value	Count	Value	Count	Value	Count	Value	Count
52	1	64	2	72	1	85	2	113	1
55	3	65	3	73	2	87	1	122	1
58	2	66	2	75	1	88	1	126	1
59	3	67	1	76	1	90	1	144	1
60	1	68	5	77	1	94	1	154	1
61	4	69	3	78	1	104	2		
62	1	70	4	79	2	106	1		
63	2	71	2	83	1	109	1		

now determine the cumulative histogram

Value	cdf	Value	cdf	Value	cdf	Value	cdf	Value	cdf
52	1	64	19	72	40	85	51	113	60
55	4	65	22	73	42	87	52	122	61
58	6	66	24	75	43	88	53	126	62
59	9	67	25	76	44	90	54	144	63
60	10	68	30	77	45	94	55	154	64
61	14	69	33	78	46	104	57		
62	15	70	37	79	48	106	58		
63	17	71	39	83	49	109	59		

This cdf shows that the minimum value in the subimage is 52 and the maximum value is 154. The cdf of 64 for value 154 coincides with the number of pixels in the image. The cdf must be normalized to $[0, 255]$. The general histogram equalization formula is:

$$h(v) = round\left(\frac{cdf(v) - cdf_{\min}}{(M \times N) - cdf_{\min}} \times (L-1) \right)$$

Where cdf_{min} is the minimum value of the cumulative distribution function (in this case 1), M × N gives the image's number of pixels (for the example above 64, where M is width and N the height) and L is the number of grey levels used (in most cases, like this one, 256). The equalization formula for this particular example is:

$$h(v) = round\left(\frac{cdf(v)-1}{63} \times 255\right)$$

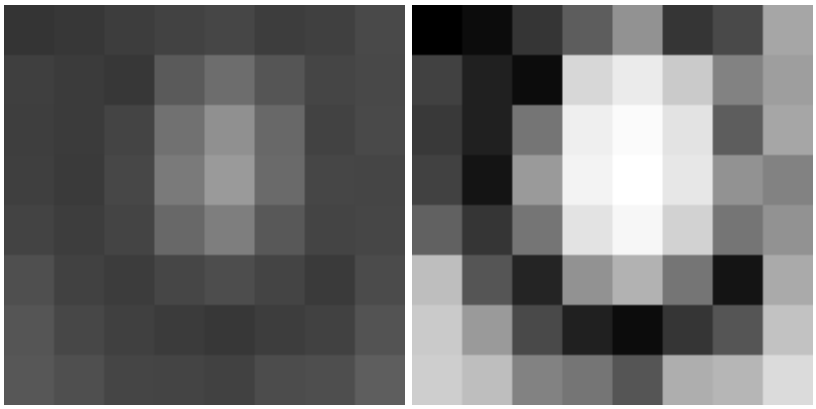
For example, the cdf of 78 is 46. (The value of 78 is used in the bottom row of the 7th column.) The normalized value becomes

$$h(78) = round\left(\frac{46-1}{63} \times 255\right) = round(0.714286 \times 255) = 182$$

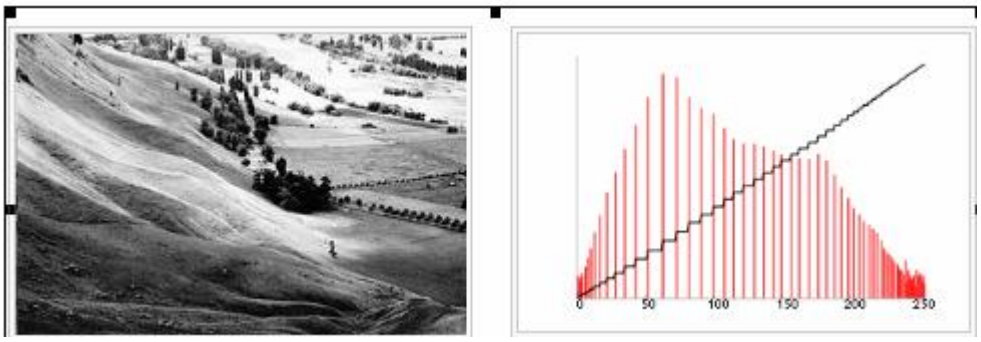
Once this is done then the values of the equalized image are directly taken from the normalized cdf to yield the equalized values:

$$\begin{bmatrix} 0 & 12 & 53 & 93 & 146 & 53 & 73 & 166 \\ 65 & 32 & 12 & 215 & 235 & 202 & 130 & 158 \\ 57 & 32 & 117 & 239 & 251 & 227 & 93 & 166 \\ 65 & 20 & 154 & 243 & 255 & 231 & 146 & 130 \\ 97 & 53 & 117 & 227 & 247 & 210 & 117 & 146 \\ 190 & 85 & 36 & 146 & 178 & 117 & 20 & 170 \\ 202 & 154 & 73 & 32 & 12 & 53 & 85 & 194 \\ 206 & 190 & 130 & 117 & 85 & 174 & 182 & 219 \end{bmatrix}$$

Notice that the minimum value (52) is now 0 and the maximum value (154) is now 255.



Original Equalized



(a) (b)

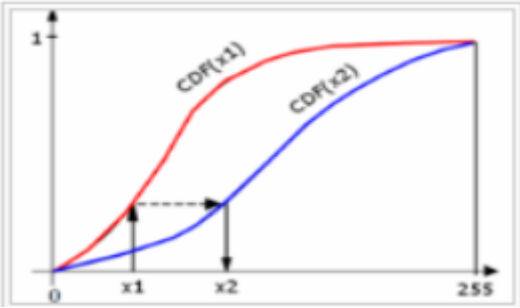
(a) = After histogram equalisation

(b) = Corresponding histogram(red) & cumulative histogram(black)

Histogram Matching

Histogram matching is a method in image processing of color adjustment of two images using the image histograms.
It is possible to use histogram matching to balance detector responses as a relative detector calibration technique. It can be used to normalize two images, when the images were acquired at the same local illumination (such as shadows) over the same location, but by different sensors, atmospheric conditions or global illumination.
Algorithm

Given two images, the reference and the adjusted images, we compute their histograms. Following, we calculate the cumulative functions of the two images' histograms - $F_1()$for the reference image and $F_2()$for the target image. Then for each gray level $G_1 \in [0, 255]$, we find the gray level $G_2$for which $F_1(G_1) = F_2(G_2)$, and this is the result of histogram matching function: $M(G_1) = G_2$. Finally, we apply the function $M()$on each pixel of the reference image.



This is an example of histogram matching.

Observation Model

The observation model is used to measure the observation likelihood of the samples. Many observation models have been built for particle filtering tracking. One of them is a contour based appearance template (Isard & Black, 1998). The tracker based on a contour template gives an accurate description of the targets but performs poorly in clutter and is generally time-consuming. The initialization of the system is relatively difficult and tedious. In contrast, color-based trackers are faster and more robust, where the color histogram is typically used to model the targets to combat the partial occlusion, and non-rigidity.

In this article, the observation model is made based on color information of the target obtained by building the color histogram in the RGB color space. This section describes how the color features is modeled in a rectangular region *R*, where *R* can be a region surrounding the object to be tracked or region surrounding one of the hypothetical regions. A color histogram is commonly used for object tracking because they are robust to partial occlusion, rotation and scale invariant. They are also flexible in the types of object that they can be used to track, including rigid and non-rigid object.

The color distribution is expressed by an *m*-bins histogram, whose components are normalized so that its sum of all bins equals one. For a region *R* in an image, given a set of *n* samples in *R*, denoted by **X** = {*xi*, *i* =1, 2, … , *n*} ∈ *R*, the *m*-bins color histogram *H*(*R*) = {*hj*}, (*j* = 1, 2, . . . ,*m*) can be obtained by assigning each pixel *xi* to a bin, by the following equation:

$$h_j = \frac{1}{n}\sum_{x_{i\in x}} \delta_j\left[b(x_i)\right]$$

Here *b*(*xi*) is the bin index where the color component at *xi* falls into, and *δ* is the Kronecker delta function. To increase the reliability of the target model,

smaller weight are assigned to the pixels that are further away from region center by employing a weighting function

$$g(r) = \begin{cases} 1 - r^2 \\ 0 \end{cases}$$

$1-r^2$ when r<1

0 otherwise

r is the distance from the centre of the region.

Using this weight, the color histogram

$p_y = \{p_y^{(u)}\}$ u = 1,……..,m at location y is calculated as

$$p_y^{(u)} = f \sum_{j=1}^{I} g\left(\frac{\|y - x_j\|}{a}\right) \delta\left[h(x_j) - u\right]$$

where *m* is number of bins, *I* is the number of pixels in the region *R*, x_j is the position of pixels in the region *R*, δ is the Kronecker delta function, *a* is the normalization factor, *f* is the scaling factor to ensures that $\left(\sum_{u=1}^{m} p_y^{u=1} = 1\right)$ *g*(.) is weighting function, respectively. Fig. 7 shows an example of target histogram at time step *k*. In subsequent frames, at every time *k*, there are *N* particles that represent *N* hypothetical states need to be evaluated. The observation likelihood model is used to assign a weight associated to a specific particle (new observation) depending on how similar the model histogram *q* and the histogram *p*(*xt*) of object in the region described by the *i*th particle *xki* are.

The similarity between two color histograms can be calculated using Bhattacharya distance $d = 1 - \rho[p,q]$, where $\rho[p,q] = \sum_{u=1}^{m} p^{(u)} q^{(u)}$ . . Similar histogram will have a small Bhattacharya distance which corresponds to high sample weight. Based on the Bhattacharya distance, the weight $\pi(i)$ of the sample state $x(i)$ is calculated as,

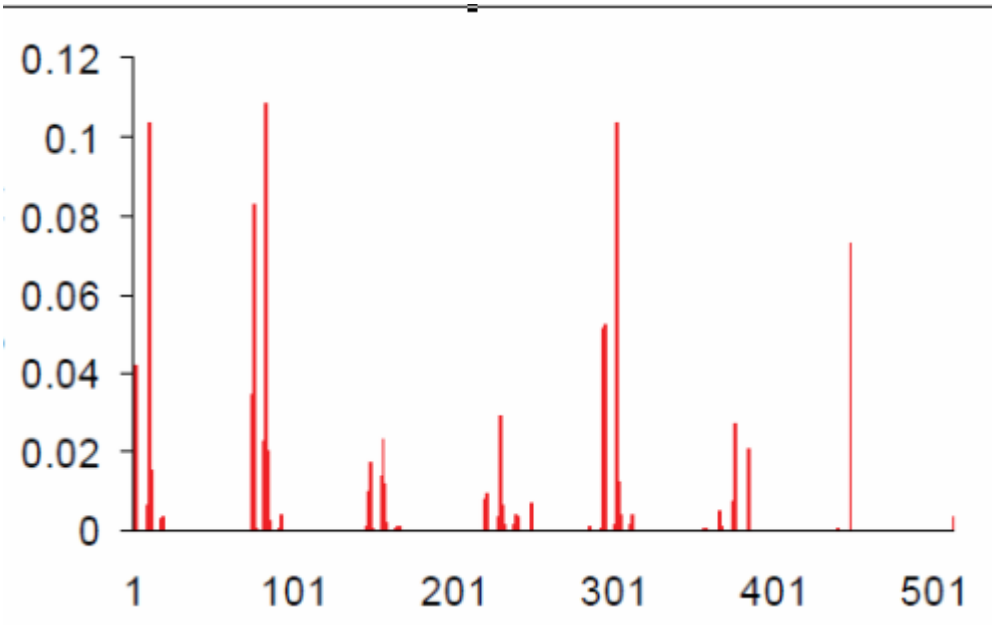
$$\Pi^{(i)} = \frac{1}{\sqrt{2\Pi\sigma}} \exp\left(\frac{d^2}{2\sigma^2} \right)$$

$$= \frac{1}{\sqrt{2\Pi\sigma}} \exp\left(- \frac{\left(1 - \rho\left[p\left(x^{(i)}\right), q\right]\right)}{2\sigma^2} \right)$$

Where $p(x(i))$ and q are the color histogram of the samples and target model, respectively.

During the resample step, samples with a high weight may be chosen several times leading to identical copies, while others with relatively low weights may be ignored.

Target object at time k



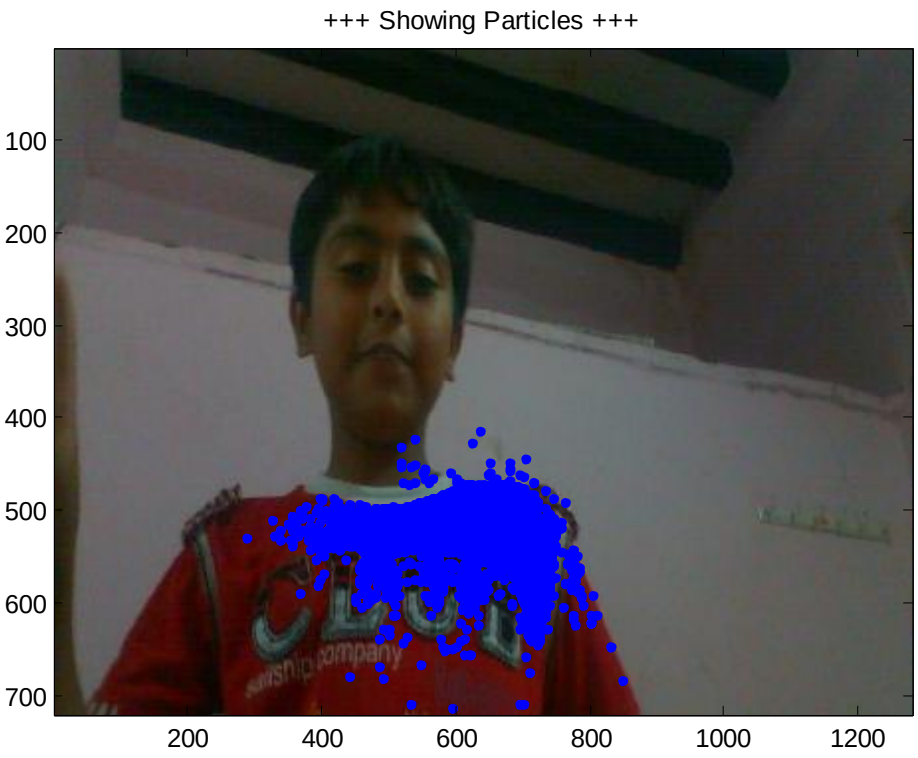
X axis = Bin number

Y axis = histogram at (t0)

Histogram of target

Fig.7 An example of color histogram distribution of a target model at time *k*

RESULTS OF CODING



X axis = Vedio.Width (1280 in pixels)

Y axis = Vedio.Height (720 in pixels)

40000 samples are trcking the red color which the boy is wearing given by

Xrgb_tgt = [255 0 0]

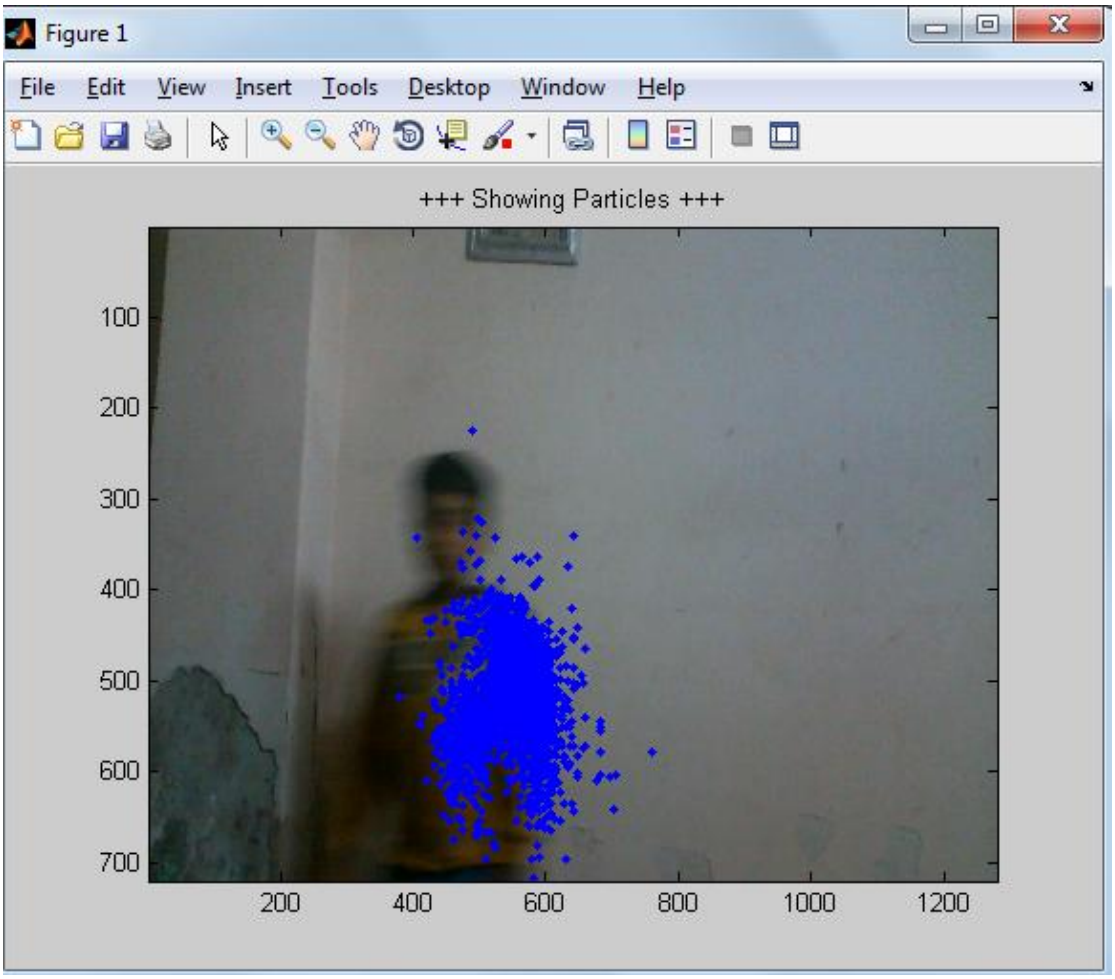
Red = 255

Blue = 0

Green = 0

Similarly for a boy wearing multi colour t-shirt eg.(yellow + Black) & we have given code for
[255 165 0]

particles will detect only yellow colour in t shirt

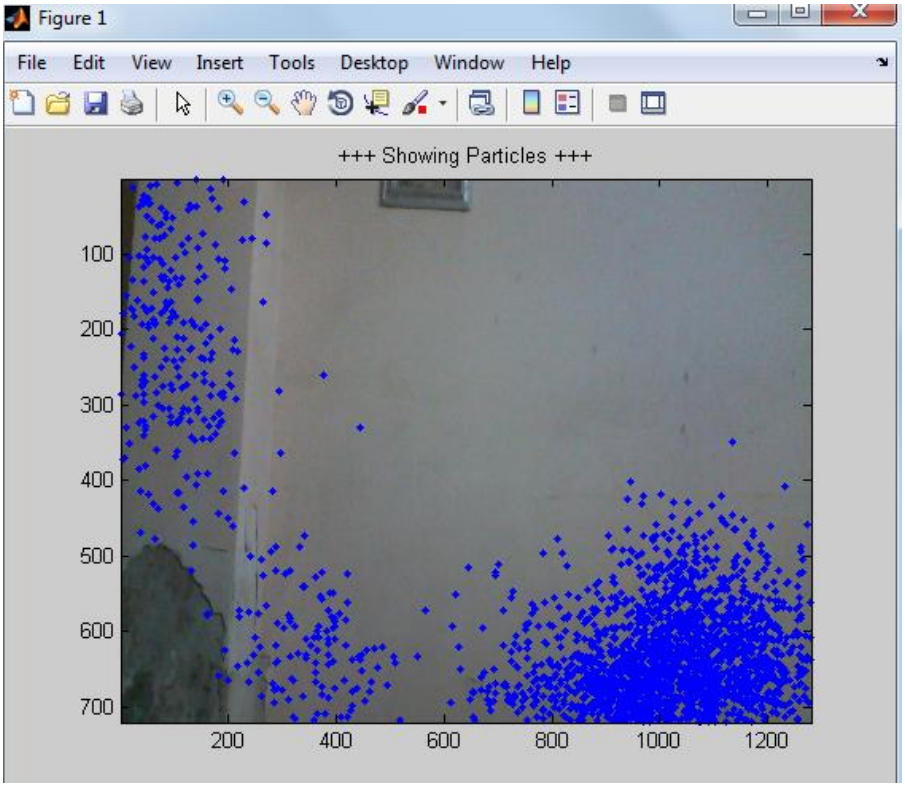


This code can track any color defined by color codec formed from RGB color space.

Video Parameters: 30.00 frames per second, RGB24 1280x720.
668 total video frames available.

Vedio at different parameters:

Initially when the particles are distributed on the entire screen.



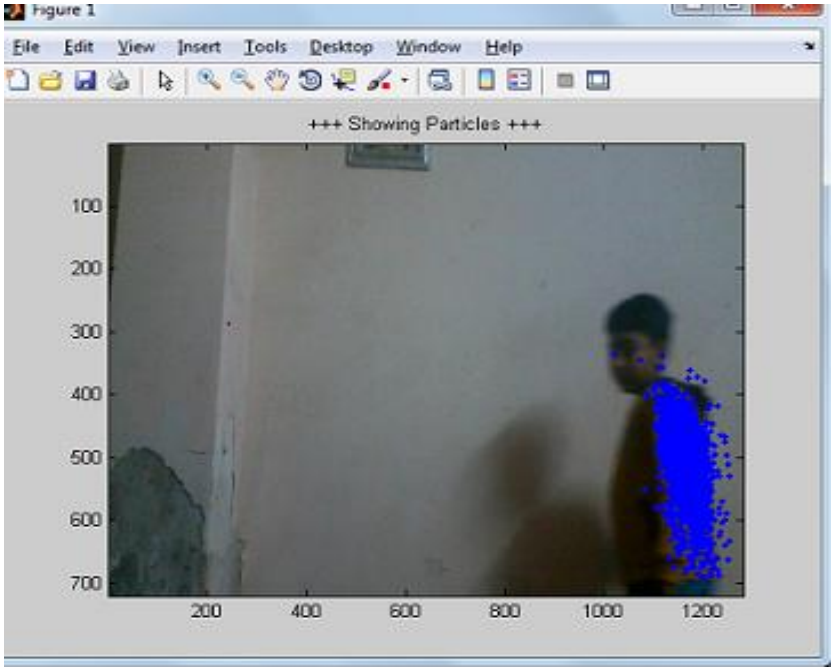


Figure 1
File Edit View Insert Tools Desktop Window Help
+++ Showing Particles +++
100
200
300
400
500
600
700
200
400
600
800
1000
1200

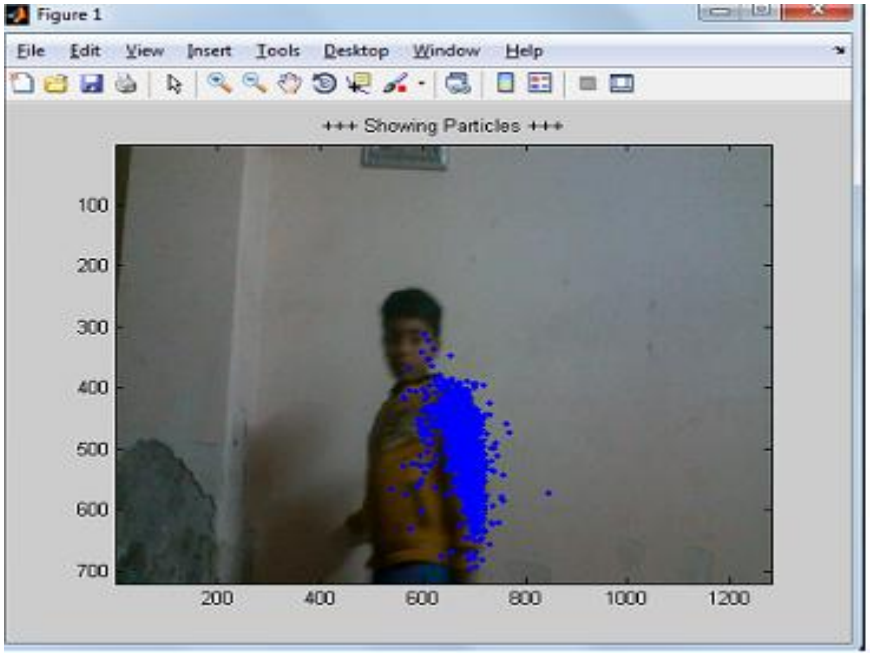


Figure 1
File Edit View Insert Tools Desktop Window Help
+++ Showing Particles +++
100
200
300
400
500
600
700
200
400
600
800
1000
1200

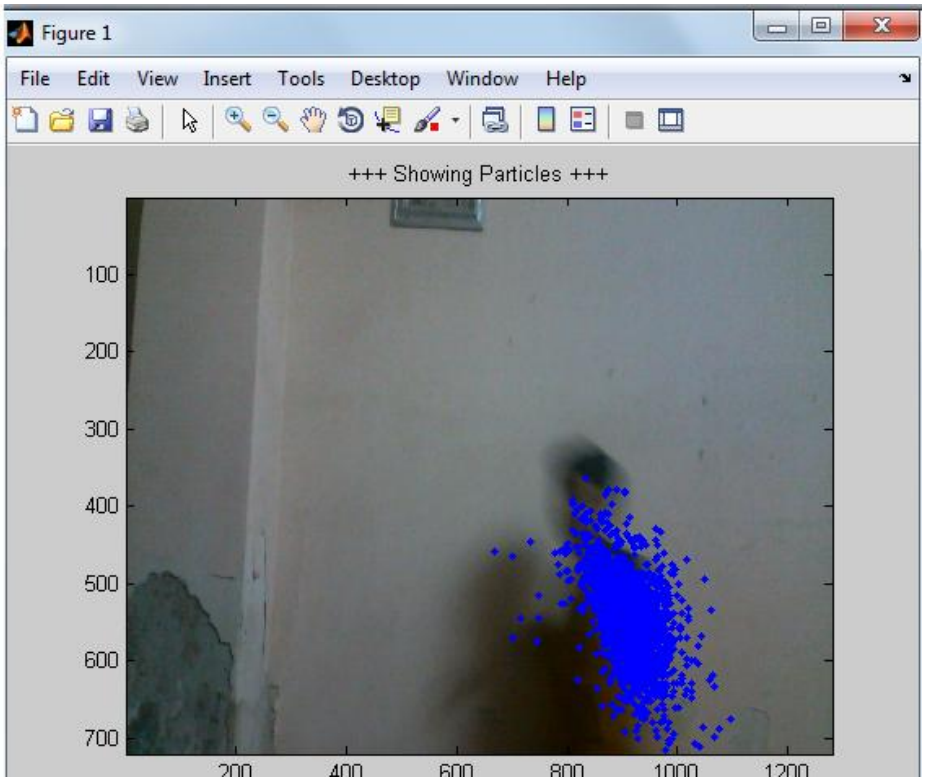


Figure 1
File Edit View Insert Tools Desktop Window Help
+++ Showing Particles +++
100
200
300
400
500
600
700
200
400
600
800
1000
1200

Conclusion

This method can track any moving object in conjunction with color information in both known & unknown initial position of the object & in the presence of occlusion with static object. The robust color likelihood is used to evaluate samples properly assosciated to the target which represents high appearance variability. We rely on bhattacharya coefficient between target & sample histogramto perform this task. Model updating is carried to update the object in presence of appearance change.

Furthermore the performance of the objects tracking can be improved in several ways such as adding the background modelling information in the calculation of likelihood, detection of appearing object & extend the method to track the multiple object in the presence of object occlusion & so on. Objects can be tracked withother techniques also like features , texture, motion based recognition, even with shapes & so on.

Reference

- IEEE parer based on “Particle Filter-Based Object Tracking Using Adaptive Histogram “ 2011 by M. Fotouhi*, A. R. Gholami**, and S. Kasaei*** Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

- IEEE paper based on “Object tracking by the mean-shift of regional color distribution combined with the particle-filter algorithm” Koichiro Deguchi, Oki Kawanaka, and Takayuki Okatan Graduate School of Information Sciences, Tohoku University Aoba-campus 01, Sendai 980-8579 Japan.

- *“On sequential Monte Carlo sampling methods for Bayesian filtering” by*

  ARNAUD DOUCET, SIMON GODSILL and CHRISTOPHE ANDRIEU *Signal Processing Group, Department of Engineering, University of Cambridge, Trumpington Street, CB2 1PZ Cambridge, UK* ad2@eng.cam.ac.uk, sjg@eng.cam.ac.uk, ca226@eng.cam.ac.uk Received July 1998 and accepted August 1999

- “Color-based object recognition Theo Gevers*, Arnold W.M. Smeulders *ISIS, Faculty of WINS, University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands* Received 22 December 1997; received for publication 4 February 1998

- “Object Tracking Based on Color Information” Employing Particle Filter Algorithm 69 Budi Sugandi, Hyoungseop Kim, Joo Kooi Tan and Seiji Ishikawa