# AUTONOMOUS OBJECT TRACKING OF A QUADROTOR USING COMPUTER VISION

**A PROJECT REPORT**

***Submitted by***


**S. SRI RAMANA (2010505055)**

**B.KOUSHIK (2010505021)**

**GOWTHAM MAHENDRAN (2010505010)**


***in partial fulfilment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

***in***

**ELECTRONICS AND INSTRUMENTATION ENGINEERING**



**DEPARTMENT OF INSTRUMENTATION ENGINEERING**

**MADRAS INSTITUTE OF TECHNOLOGY**

**ANNA UNIVERSITY: CHENNAI 600 044**


**APRIL 2014**

# ANNA UNIVERSITY: CHENNAI 600 044

## BONAFIDE CERTIFICATE

Certified that this report titled "**AUTONOMOUS OBJECT TRACKING OF A QUADROTOR"** is the bonafide work of **SRI RAMANA.S**(2010505055), **B.KOUSHIK**(2010505021) and **GOWTHAM MAHENDRAN**(2010505010) who carried out the project work under my supervision.

**SIGNATURE**
Dr.J.Prakash
**HEAD OF THE DEPARTMENT**
Professor,
Department of Instrumentation Engineering,
Madras Institute of Technology,
Anna University,
Chennai -600 044.

**SIGNATURE**
Dr.D.Manamalli
**GUIDE**
Associate Professor,
Department of Instrumentation Engineering,
Madras Institute of Technology,
Anna University,
Chennai -600 044.

## ACKNOWLEDGEMENT

We here place on record our sincere thanks to our respected Vice Chancellor, **Dr.M.Raja Ram** for all his efforts and administration in educating us in this premier institution. We take the opportunity to express our hearty thanks to **Dr.S.Thamirai Selvi**, Dean, Madras Institute of Technology.

We take privilege to extend our thanks to the Head of the Department of Instrumentation Engineering, **Dr.J.Prakash**, who has always been a constant source of inspiration and encouragement to us during the course.

We are indebted to our internal guide **Dr.D.Manamalli,** associate Professor, Department of Instrumentation Engineering for her commendable support.

We wish to convey our sincere thanks to all the teaching and non-teaching staffs of the Department of **Instrumentation Engineering** for their valuable suggestions and technical support rendered during the course of this project.

**S.SRI RAMANA**

**B.KOUSHIK**

**GOWTHAM MAHENDRAN**

## ABSTRACT

The main aim of the project is to realize an object tracking algorithm in an autonomous Unmanned Aerial Vehicles (UAV) that enables it to track ground moving object. This object tracking algorithm used in this project is a particle filter in combination with SURF feature detectors. The particle filter also known as condensation algorithm is an algorithm based on Bayesian inference that represent the pdf (Probability Density Function) as a set of particles and uses a colour histogram as the observation model of the reference image to be tracked. The SURF (Speeded Up Robust Features) is an improved version of SIFT(Shift Invariant Feature Transform).It applies mathematical operation to the reference image and stores the keypoints in the form of vectors. These keypoints also called features, are very distinct to any image. The SURF is rotation, scale, and illumination invariant. In our hybrid approach we use both the above mentioned algorithms. Normally, SURF keeps on tracking the object. But at times SURF can be detect many false positives. Our program checks if the object detected by SURF is the object to be tracked, if not then the particle filter takes over. The processor used here is an advanced digital signal processor from Blackfin BF561-EZ-kit.

## LIST OF FIGURES USED IN THIS PROJECT

**CHAPTERNo** **TITLE** **PAGE No.**

# CHAPTER 1

# INTRODUCTION

## 1.1 INTRODUCTION

The vision-based control of Unmanned Aerial Vehicles (UAVs) has become a very active field of research in the last decade. Vision indeed provides a cheap, passive and rich source of information, and low weight cameras can be embedded even on small-size flying UAVs. Until now, most of the efforts have been concentrated on developing vision-based control methods for autonomous take off, landing, stabilization and navigation, in which the visual information is usually obtained using a known model of a target or the environment, key images, or texture points for motion estimation or optical flow computation. To track arbitrary objects is a key ability for autonomous agents to fulfil many different tasks like surveillance, guiding or following as well as interacting with and learning from humans.

## 1.2 AIM AND OBJECTIVE

The aim of the project is to develop an object tracking algorithm that involves a combination of SURF feature detector and particle filter algorithm with the aim of porting the algorithm on-board a quadrotor. Both the algorithm has distinct desirable characters, by combining both algorithms we get robustness and the advantages of both the algorithms.

# CHAPTER 2

# RECENT TRENDS IN UAV OBJECT TRACKING

## 2.1 FACTORS LEADING TO SEVERAL APPROACHES

Many successful and accurate object tracking approaches have been proposed in recent years. However, many of them are not applicable for the tasks of mobile robots, because the domain violates several of the underlying assumptions. There is no static background and no fixed target appearance and the image quality can be bad due to insufficient illumination or glare. In some applications one cannot build a complex target model off-line, because the kind of object to track is not known in advance. Generally, one does not have a set of calibrated cameras for 3D reconstruction. And finally, the computational power is very limited because of small form factors and the available energy, but at the same time quick reactions are needed when interacting with a rapidly changing environment. There are several algorithms followed for object tracking and are discussed below.

## 2.2 ADAPTIVE ALGORITHM

The core of this method is a novel observation model and the way it is automatically adapted to a changing object and background appearance over time [2]. The model is integrated into the well known Condensation algorithm (SIR filter)for statistical inference, and it consists of a boosted ensemble of simple threshold classifiers built upon center-surround Haar-like features, which the filter continuously updates based on the images perceived. Optimizations and reasonable approximations to limit the computational costs were presented. Thus, the final algorithms are capable of processing video input in real time.

To experimentally investigate the gain of adapting the observation model, two different approaches with a non-adapting version of observation model were

compared: maintaining a single observation model for all particles, and maintaining individual observation models for each particle. In addition, experiments were conducted to compare system performances between the proposed algorithms and two other state of the art Condensation based tracking approaches.

## 2.3 LAYER SEGMENTATION APPROACH

In this approach [5], layer segmentation approach with background stabilization and post track refinement is combined to reliably detect small moving objects at the relatively low processing speed. A fast tracking algorithm that has been optimized for real-time application was employed. To classify vehicle and person from the detected objects, a (Histogram Of Oriented Gradient ) HOG based vehicle vs. person classifier is designed and integrated with the tracking post processing.

## 2.4 KALMAN FILTER BASED APPROACH

This approach[3] uses multiple independent object tracking algorithms as inputs to a single Kalman filter. A function for estimating each algorithm's error from related features is trained using linear regression. This error is used as the algorithm's measurement variance. With a dynamic measurement error covariance computed from these estimates, an overall object tracking filter that combines each algorithm's best-case behavior was produced while diminishing worst-case behavior. This filter is intended to be robust without being programmed with any environment-specific rules.

# CHAPTER 3

# OBJECT TRACKING

## 3.1 INTRODUCTION

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 (affine transformation or homography) 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.
- For video compression, key frames are divided into macroblocks. The motion model is a disruption of a key frame, where each macroblock is translated by a motion vector given by the motion parameters.
- The image of deformable objects can be covered with a mesh, the motion of the object is defined by the position of the nodes of the mesh.

## 3.2 TYPES OF ALGORITHM USED

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, as well as filtering and data association.

### 3.2.1 TARGET REPRESENTATION AND LOCALIZATION

It 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 detection, block-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)
- Visual **feature matching**: registration

### 3.2.2 FILTERING AND DATA ASSOCIATION:

It 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.

# CHAPTER 4

# COMPUTER VISION USING OPEN CV

## 4.1 INTRODUCTION

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

## 4.2 APPLICATIONS

Applications range from tasks such as industrial machine vision systems which, say, inspect bottles speeding by on a production line, to research into artificial intelligence and computers or robots that can comprehend the world around them. The computer vision and machine vision fields have significant overlap. Computer vision covers the core technology of automated image analysis which is used in many fields. Machine vision usually refers to a process of combining automated image analysis with other methods and technologies to provide automated inspection and robot guidance in industrial applications.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner.

As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

- Controlling processes, *e.g.*, an industrial robot;
- Navigation, *e.g.*, by an autonomous vehicle or mobile robot;
- Detecting events, *e.g.*, for visual surveillance or people counting;
- Organizing information, *e.g.*, for indexing databases of images and image sequences;
- Modelling objects or environments, *e.g.*, medical image analysis or topographical modelling.
- Interaction, *e.g.*, as the input to a device for computer-human interaction, and
- Automatic inspection, *e.g.*, in manufacturing applications.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, learning, indexing, motion estimation, and image restoration.

In most practical computer vision applications, the computers are pre-programmed to solve a particular task, but methods based on learning are now becoming increasingly common.

## 4.3 OPENCV

**OpenCV** (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance

Primitives on the system, it will use these proprietary optimized routines to accelerate it.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Ego motion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and Recognition
- Stereo sis Stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

## 4.4 CONTENTS OF MACHINE LEARNING LIBRARY

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting (meta-algorithm)
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbour algorithm
- Naive Bayes classifier

- Artificial neural networks
- Random forest
- Support vector machine (SVM)

## 4.5 PROGRAMMING LANGUAGE USED IN OPENCV

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are now full interfaces in Python, Java and MATLAB/OCTAVE (as of version 2.5). The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Ch, Ruby have been developed to encourage adoption by a wider audience.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

# CHAPTER 5

# PARTICLE FILTER USING BAYESIAN INFERENCE

## 5.1 BAYESIAN INFERENCE

Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a "likelihood function" derived from a probability model for the data to be observed. Bayesian inference computes the posterior probability according to Bayes' rule:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

Where

- $\mid$ denotes a conditional probability; more specifically, it means *given*.
- $H$ stands for any *hypothesis* whose probability may be affected by data (called *evidence* below). Often there are competing hypotheses, from which one chooses the most probable.
- The *evidence* $E$ corresponds to new data that were not used in computing the prior probability.
- $P(H)$, the *prior probability*, is the probability of $H$ *before* $E$ is observed. This indicates one's previous estimate of the probability that a hypothesis is true, before gaining the current evidence.
- $P(H \mid E)$, the *posterior probability*, is the probability of $H$ *given* $E$, i.e., *after* $E$ is observed. This tells us what we want to know: the probability of a hypothesis *given* the observed evidence.
- $P(E \mid H)$ is the probability of observing $E$ *given* $H$. As a function of *E* with *H* fixed, this is the *likelihood*. The likelihood function should **not** be confused with *P*(*H* | *E*) as a function of *H* rather than of *E*. It indicates the compatibility of the evidence with the given hypothesis.
- $P(E)$ is sometimes termed the marginal likelihood or "model evidence". This factor is the same for all possible hypotheses being considered.

(This can be seen by the fact that the hypothesis $H$ does not appear anywhere in the symbol, unlike for all the other factors.) This means that this factor does not enter into determining the relative probabilities of different hypotheses.

Note that, for different values of $H$, only the factors $P(H)$ and $P(E \mid H)$ affect the value of $P(H \mid E)$. As both of these factors appear in the numerator, the posterior probability is proportional to both. In words:

- (more exactly) The posterior probability of a hypothesis is determined by a combination of the inherent likeliness of a hypothesis (the prior) and the compatibility of the observed evidence with the hypothesis (the likelihood).
- (more concisely) Posterior is proportional to likelihood times prior.

Note that Bayes' rule can also be written as follows:

$$P(H \mid E) = \frac{P(E \mid H)}{P(E)} \cdot P(H)$$

where the factor $\frac{P(E|H)}{P(E)}$ represents the impact of $E$ on the probability of $H$.

## 5.2 RECURSIVE BAYES FILTER:

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.

### 5.2.1 MODE

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).

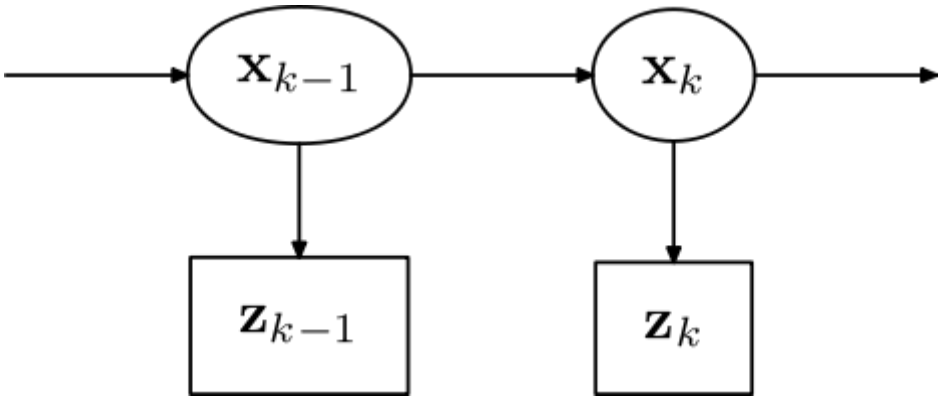



Fig 5.1: Bayesian Network of HMM

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.

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{x}_{k-2}, \ldots, \mathbf{x}_0) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$$

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.

$$p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{x}_{k-1}, \ldots, \mathbf{x}_0) = p(\mathbf{z}_k|\mathbf{x}_k)$$

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

$$p(\mathbf{x}_0, \ldots, \mathbf{x}_k, \mathbf{z}_1, \ldots, \mathbf{z}_k) = p(\mathbf{x}_0) \prod_{i=1}^{k} p(\mathbf{z}_i|\mathbf{x}_i) p(\mathbf{x}_i|\mathbf{x}_{i-1}).$$

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 $x_{k-1}$.

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

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

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} = \alpha\, p(\mathbf{z}_k|\mathbf{x}_k) p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$$

The denominator

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

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

## 5.3 SEQUENTIAL 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.

The method is named:

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

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

**Prediction,** when estimating a probable *future* value given the present and the past measures.

The notion of Sequential Bayesian filtering is extensively used in control and robotics.

## 5.4 SEQUENTIAL MONTE CARLO METHODS

**Particle filters** or **Sequential Monte Carlo** (SMC) methods are a set of on-line posterior density estimation algorithms that estimate the posterior density of the state-space by directly implementing the Bayesian recursion equations. SMC methods use a grid-based approach, and use a set of particles to represent the posterior density. These filtering methods make no restrictive assumption about the dynamics of the state-space or the density function. SMC methods provide a well-established methodology for generating samples from the required distribution without requiring assumptions about the state-space model or the state distributions. The state-space model can be non-linear and the initial state and noise distributions can take any form required. However, these methods do not perform well when applied to high-dimensional systems. SMC methods implement the Bayesian recursion equations directly by using an ensemble based approach. The samples from the distribution are represented by a set of particles; each particle has a weight assigned to it that

represents the probability of that particle being sampled from the probability density function.

Weight disparity leading to weight collapse is a common issue encountered in these filtering algorithms; however it can be mitigated by including a re-sampling step before the weights become too uneven. In the re-sampling step, the particles with negligible weights are replaced by new particles in the proximity of the particles with higher weights.

## 5.5 SEQUENTIAL IMPORTANCE RESAMPLING (SIR)

### 5.5.1 ALGORITHM

*Sequential importance re-sampling (SIR),* the original particle filtering algorithm (Gordon et al. 1993), is a very commonly used particle filtering algorithm, which approximates the filtering distribution $p(x_k|y_0, \ldots, y_k)$ by a weighted set of P particles

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

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 as a weighted average

$$\int f(x_k)p(x_k|y_0, \ldots, y_k)dx_k \approx \sum_{L=1}^{P} w_k^{(L)} f(x_k^{(L)}).$$

For a finite set of particles, the algorithm performance is dependent on the choice of the *proposal distribution*

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

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

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

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

$$\pi(x_k|x_{0:k-1}, y_{0:k}) = p(x_k|x_{k-1}).$$

*Sequential Importance Re-sampling* (SIR) filters with transition prior probability distribution as importance function are commonly known as bootstrap filter and condensation algorithm.

*Re-sampling* 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 re-sampling method. The *stratified sampling* proposed by Kitagawa (1996) is optimal in terms of variance.

### 5.5.2 PROCEDURE FOR SEQUENTIAL RESAMPLING

The steps involved in sequential re-sampling are as follows:

1) For $L = 1, \ldots, P$ draw samples from the *proposal distribution*

$x_k^{(L)} \sim \pi(x_k|x_{0:k-1}^{(L)}, y_{0:k})$

2) For $L = 1, \ldots, P$ update the importance weights up to a normalizing constant:

$$\hat{w}_k^{(L)} = w_{k-1}^{(L)} \frac{p(y_k|x_k^{(L)})p(x_k^{(L)}|x_{k-1}^{(L)})}{\pi(x_k^{(L)}|x_{0:k-1}^{(L)}, y_{0:k})}.$$

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

$$\hat{w}_k^{(L)} = w_{k-1}^{(L)} p(y_k|x_k^{(L)}),$$

3) For $L = 1, \ldots, P$ compute the normalized importance weights:

$$w_k^{(L)} = \frac{\hat{w}_k^{(L)}}{\sum_{J=1}^{P} \hat{w}_k^{(J)}}$$

4) Compute an estimate of the effective number of particles as

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

5) 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 6

# ABOUT THE PROJECT WORK

The project involves implementation of an object tracking algorithm in a quad rotor with an on-board camera. The algorithm used in this project is a combination of SURF (Speeded Up Robust Features) feature detectors and Condensation algorithm also called a Particle Filter.

## 6.1 SURF FEATURE DETECTOR

This is a method for object detection that is scale and rotation invariant, robust, fast and most importantly, can work with a single training image! Speeded-Up Robust Features, SURF in short.

A short description of what SURF does is:

- Find interest points in the image using Hessian matrices
- Determine the orientation of these points
- Use basic Haar wavelets in a suitably oriented square region around the interest points to find intensity gradients in the X and Y directions. As the square region is divided into 16 squares for this, and each such sub-square yields 4 features, the SURF descriptor for every interest point is 64 dimensional.
- The 4 features are sums of gradient changes.

## 6.2 MATCHING STRATEGY

A 64 dimensional descriptor for every key point is useless without a method of deciding whether a query descriptor matches a training descriptor or not. For matching, we use the *K nearest neighbour search*. A KNN search basically computes the 'distance' between a query descriptor and all of the

training descriptors, and returns the K pairs with lowest distance. Here we will keep K=2.

Hence we now have 2 pairs of matches for each query descriptor. So basically the KNN search gave us 2 matches to every query descriptor. What is important is how we decide which of all these matches ‘good matches’ are after all. The strategy we employ is (each match has a 'distance' associated with it) .
For every query descriptor,
If distance(match1) < 0.8 * distance (match2), match1 is a good match otherwise discard both match1 and match2 as false matches.

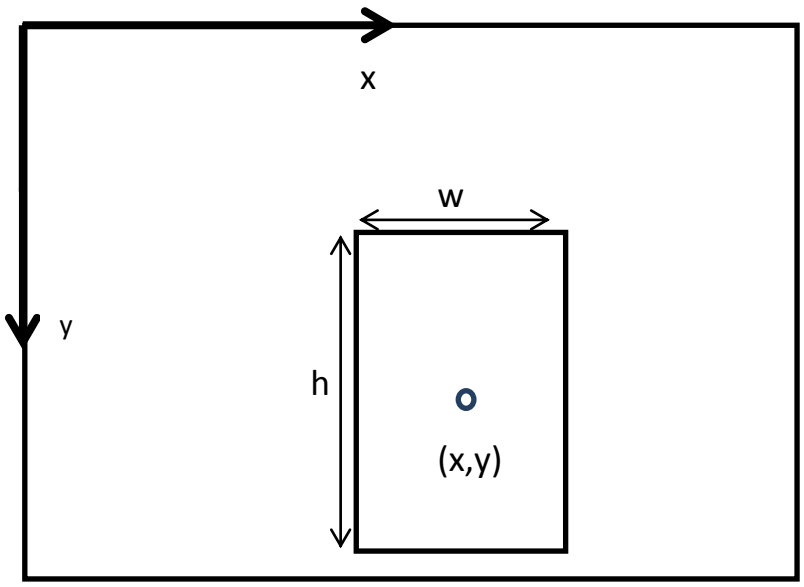
## 6.3 REAL-TIME IMPLEMENTATION

- Take an image of the object you want to detect and extract SURF descriptors for it. It is important for this image to contain *only* the object and to be free from any harsh lighting.
- Now do the same for every frame coming from your camera.
- Employ the matching strategy to match descriptors from every frame with the descriptors of the object and find out the 'good matches'.
- Create a window with the object image on one side and the video playing on the other side. Draw good matches between the two.
- To get a bounding box around a detected object, using the good matches, find a homography that transforms points from the object image to the video frame. Using this homography , transform the 4 corners of the object image. Consider these 4 transformed points as vertices and draw a box in the video frame.

## 6.4 THE CONDENSATION ALGORITHM

### 6.4.1 THE OBJECT STATE

We have employed a first order motion model in order to predict the future target position in the next few frames of the sequence and avoid the exhaustive search over the entire frame. The state vector has therefore been defined so as to include the position and size of the rectangular bounding box of the target object in the image plane, as well as its velocity and changes in size. In particular, the state space for our tracker is defined over vectors of the form as shown in Fig 6.1:




$X_k = (x, y, w, h, x', y', w', h')$

Fig 6.1: State space representation of the tracker

Here as defined in the picture the (x,y) represents the position of the object in the image frame with a bounding box of width w and height h. These form the first four term of the equation and the other four terms form the

derivative for the four terms i.e.) x', y', w', h' denote the rate of change of x, y, w, h respectively.

### 6.4.2 THE OBSERVATION MODEL

The observation model used in this project is the colour histogram of the reference object to be tracked. The histogram of the bounding box of dimensions w and h centred on (x,y) is considered.

### 6.4.3 INITIALISING THE POINTS

Let initially the target position is known, N points are taken around it with Gaussian distribution. All of these points have equal weights assigned to each other so that the total sums up to one hence weights are initialised to 1/N.

### 6.4.4 MEASUREMENTS UPDATE EQUATION

Now for each and every point compute the histogram and calculate its distance from the reference histogram. Let $q^* = \{q_u^*\}_{u=1,2,\ldots m.}$ denote the reference histogram, determined in the tracking initialisation step. Then, the tracking process aims to find in each frame k, the candidate state

$X_k$ which histogram $q(x_k)$ is the "closest" to the reference histogram $q^*$. To achieve this, a correlation criterion in the histogram space is provided by Bhattacharyya coefficient:

$$\varrho(x_k) = \varrho(q^*; q(x_k)) = \Sigma\sqrt{(q_u^* q_u)}$$

A candidate state $x_k$ for the object in frame k is then compared to the reference object using the Bhattacharyya distance:

$$d(x_k) = d(q^*, q(x_k)) = \sqrt{(1-\varrho(x_k))}$$

The likelihood function is derived from the distance defined in equation. The spatial information is yielded by computing $n^{th}$ histograms in the $n^{th}$ parts as in.

Formally, the likelihood of a state $x_k$ is then defined by:

$p(z_k \mid x_k) \alpha \exp(-\lambda d_m^2(x_k))$

where $\lambda$ is a constant parameter tuned empirically (a typical value is $\lambda = 20$).For initialisation, the particles are sampled on a Gaussian distribution around the initial known position. The weights are assigned the value of the likelihood and then normalised such that the sum of all the weights of the particles add up to 1.

### 6.4.5 RE-SAMPLING

The N particle states are resampled with replacement and the probability of a particle getting resampled is directly proportional to the weight of the particle. Hence, this way particles with higher weights are sampled for next generation while particles with lesser weights get discarded. Each particle is again assigned a weight of 1/N.

### 6.4.6 STATE UPDATE

The velocity is incremented depending on the distance of current centre of mass of the particles. The increment of velocity is directly proportional to the distance of the centre of mass. As a consequence if the center of mass is more similar to the reference image then the particles are propagated only to a certain amount else they are propagated far out. The position vector is updated by adding the current velocity vectors.

Then the cycle of measurement update, re-sampling is repeated at any instant the centre of mass of the particles gives the current position of the object.

## 6.5 THE ALGORITHM

1. In our project the algorithm is a combination of the above mentioned algorithm.
2. Initially the colour histogram in HSV colour space of the object is stored and the SURF features for the reference object are stored.
3. Then once the video frames starts pouring in the SURF, detects the object and draws a bounding box around the object.
4. First we detect whether the bounding rectangle is of correct size and whether it is a rectangle and calculate the colour histogram of the center of the bounding box.
5. If all of the above conditions hold then the all the particles of particle filter is given the centre of the bounding box and all its weights are initialised to one.
6. If any of the above condition fails then the particle filter automatically takes over and starts tracking the object using the algorithm mentioned above.
7. Once the SURF starts producing correct transformations we resume tracking using SURF.

## 6.6 IMPLEMENTATION AND RESULTS

In this project the on-board processor is an ADSP BF561-EZ KIT board. The camera is attached to the board and the board runs on a visual DSP kernel. We use visual DSP IDE for programming and we use openCV library for the computer vision primitives.

The ADSP bf561 is a dual core processor with cores named as core A and core B. It is interfaced with the input video device and an output video device using a video decoder and an encoder through the PPI (Parallel Peripheral Interface) 0 and 1.

Our algorithm is situated in core A as it gets the input from the PPI0 processes the frame and gives it back via the output frame.

The Fig 6.2 shows a ball being tracked by the particle filter. As it can be seen, the blue particles are the particles evolved from the previous generation of re sampled particles, using the given dynamic model and, after performing measurement update , the particles are weighed based on the observation model, Only the green particles that have a higher weight are chosen with higher probability in the re-sampling step. This clearly illustrates random weighted sampling. The location of image at any instant is the weighted mean of the particles and by finding the mean we get the center of the image. A bounding box is drawn around the object using the center of mass.

Fig 6.2: Object tracked by the Particle Filter

The Fig 6.3 illustrates the SURF feature detection algorithm. The SURF automatically draws a bounding box around the detected object from the detected features that are indicated by blue circles.

In Fig 6.4 partial or total occlusion to the target object occurs and when the occlusion is released in Fig 6.5 the particle filter resumes tracking the object. It can be seen that although the blue dot is dense around the lower portion of the image and sparse in the object location the particle filter accurately samples the particle near the object that has more weight indicated by the green dot.

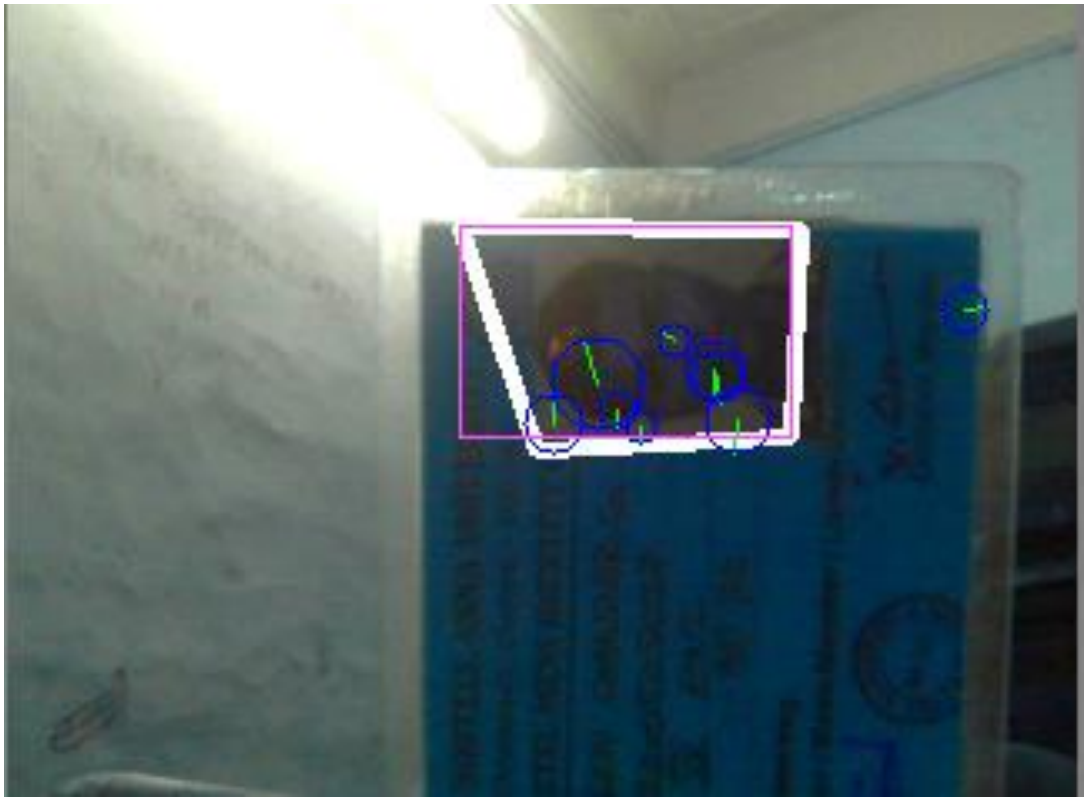


Fig 6.3: Object being tracked by SURF feature descriptor

Fig 6.4 : Total occlusion of the object

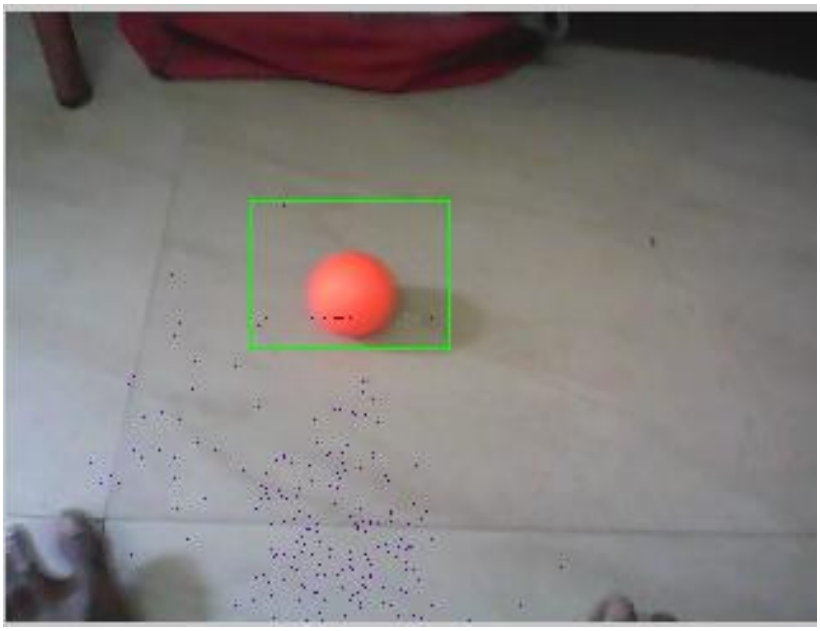


Fig 6.5: Retracking of the object by the Particle Filter

The openCV implementation of the program can be found in the appendix. The first program ‘testapp.h’ is the header file for our program that declares the various variables and functions necessary for our implementation. The second program ‘testapp.c’ is the main source code for our implementation. It contains the definition of our functions declared in ‘testapp.h’. The main.cpp is the main function of the core A processor in which the ‘processvideo()’ function contains the call to our algorithm.

# CHAPTER 7

# CONCLUSION

## 7.1 OUTCOME OF THE PROJECT

An object tracking algorithm using a combined approach of SURF and particle filter is completed. This algorithm is scale, rotation, illumination invariant, robust against occlusion. The algorithm is realized in the blackfin ADSP bf561 EZ KIT and successfully installed in UAV.

## 7.2 FUTURE SCOPE

- The particle filter can further be enhanced by having an adaptive observation model.
- We can combine a learning algorithms such as Self Organising Maps(SOMs) to learn every particular feature of the target image and hence leading to robust tracking.
- Another approach is to create an ensemble of classifiers by combining all the approaches mentioned so far and build a very robust tracker.

  [Note: we must always bear in mind the computational complexity involved in any algorithm and based on the situation we should tweak the algorithm to get the desired performance.]

# REFERENCES


1. Afef Salhi and Ameni Yengui Ammoussi, 'Object tracking system using Camshift , Meanshift and Kalman filter ', *World Academy of Science, Engineering and Technology 64 2012*

2. C´eline Teuli`ere, Laurent Eck, Eric Marchand, 'Chasing a moving target from a flying UAV',*(IROS), IEEE/RSJ – 2011*

3. Dominik A. Klein, Dirk Schulz, Simone Frintrop, and Armin , 'Adaptive Real-Time Video-Tracking for Arbitrary Objects'*(IROS),IEEE/RSJ – 2010*

4. Gavin Coelho B.S, 'An Object tracking autonomous quadrotor for real-time recognition', *UNIVERSITY OF NORTH TEXAS, May 2012.*

.

5. Jiangjian Xiao, Chang jiang Yang, Feng Han, and Hui Cheng, 'Vehicle and Person Tracking in UAV Videos', *Classification of Events, Activities, and Relationships Evaluation and Workshop, Baltimore, 2007.*

6. Alberto Del Bimbo, Fabrizio Dini, 'Particle filter–based visual tracking with a first order dynamic model and uncertainty adaptation'.

# APPENDIX

## APPENDIX A

## PROGRAM

*Testapp.h*

```
#pragma once
#include "opencv.hpp"
#include "surf-algo\surflib.h"
#define in 50
#define N 50

static  IplImage *img,*pix2,*pix3;
static  IpVec refpts,hipts[in];
static  IpVec pts,ppts;static IpPairVec match;
static  int refkey[10][2];
static  int h,w,t,tg;
static  double hist1[256 * 3],hist2[256 * 3],refhist[256*3], obj[8], updt[N][9];
static  unsigned char *pix1;


float compkey(int refkey[][2],unsigned int numrefkey,int key[][2],unsigned int
numkey);
void init(unsigned char *img,double *refhist,IpVec *refpts,double updt[][9],int
w,int h);
float comphist(double *hist1,double *hist2);
void crhist(unsigned char *img,double *hist,int w,int h,double obj[4]);
void resample(unsigned char *img,double updt[][9],double *pconf,int w,int
h,int t);
void evolve(unsigned char *,double updt[][9],double *refhist,int,int);
void msrmtupdate(unsigned char *img,double updt[][9], IpVec *refpts,double
*refhist,int w,int h);
void exobj(double updt[][9],double *obj);
void rep(IplImage *img,double obj[4],int w,int h);
bool isrect(CvPoint p[]);
```

**APPENDIX B**

*Testapp.c*

```
#include "testApp.h"

float compkey(int refkey[][2],unsigned int numrefkey,int key[][2],unsigned int
numkey);
void init(unsigned char *img,double *refhist,IpVec *refpts,double updt[][9],int
w,int h);
float comphist(double *hist1,double *hist2);
void crhist(unsigned char *img,double *hist,int w,int h,double obj[4]);
void resample(unsigned char *img,double updt[][9],double *pconf,int w,int
h,int t);
void evolve(unsigned char *,double updt[][9],double *refhist,int,int);
void msrmtupdate(unsigned char *img,double updt[][9], IpVec *refpts,double
*refhist,int w,int h);
void exobj(double updt[][9],double *obj);
void rep(IplImage *img,double obj[4],int w,int h);
bool isrect(CvPoint p[]);

float err(float x)
{
        float t= 1/(1+.5 *abs(x));
        float tu=t*exp(-x*x-1.26551+1.000023*t+.37409*t*t*t-.186288 *
pow(t,4)+.27886*pow(t,5)-1.135203*pow(t,6));
        if(x)
                return (1-tu);
        else
                return(tu-1);

}


void evolve(unsigned char *img,double updt[][9],double *refhist,int w,int h)
```

```
{
	double obj[8],hist[257*10],dist;
	exobj(updt,obj);
	IplImage *im,*im2;
	im2=cvCreateImage( cvSize(w,h), IPL_DEPTH_8U, 3);
	memcpy(im2->imageData,img,im2->imageSize);
	im=cvCloneImage(im2);
	cvCvtColor(im2,im, CV_RGB2HSV);
	crhist((unsigned char *)im->imageData,hist,w,h,obj);
	dist=(comphist(refhist,hist));
	for(int i=0;i<N;i++)
	{
		updt[i][4]+=(rand()%((int)(60*dist))-(int)(30*dist));
		updt[i][5]+=(rand()%((int)(60*dist))-(int)(30*dist));
		updt[i][6]+=(rand()%((int)(60*dist))-(int)(30*dist));

		updt[i][0]+=updt[i][4];
		updt[i][1]+=updt[i][5];
		updt[i][2]+=updt[i][6];
		updt[i][3]=(updt[i][2]*60)/80;

			if((updt[i][0]<0 || updt[i][0]>320))
			{
				updt[i][0]=w/2;updt[i][4]=0;
			}
			if((updt[i][1]<0 || updt[i][1]>240))
			{	updt[i][1]=h/2;updt[i][5]=0;
			}
		if((updt[i][2]<=40 || updt[i][3]<=40))
			{
				/*updt[i][2]=80;
				updt[i][3]=60;
				updt[i][6]=updt[i][7]=0;*/
			}

	}
}
```

```
bool isrect(CvPoint p[])
{

	if(abs(sqrt(pow((float)p[0].x-p[2].x,2)+pow((float)p[0].y-p[2].y,2)) -
	sqrt(pow((float)p[1].x-p[3].x,2)+pow((float)p[1].y-p[3].y,2)))<9 )
		return 1;
	else
		return 0;

}
void msrmtupdate(unsigned char *img,double updt[][9],IpVec *refpts,double
*refhist,int w,int h)
{
	long double tot=0,dist2=0;
	double hist[257 * 10]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
	static double ptr[4],p;
	int key[10][2];
	IpVec pts,tpts;
	IplImage *cl,*tmp2;
	cl=cvCreateImage(cvSize(w,h),IPL_DEPTH_8U,3);
	memcpy(cl->imageData,(img),cl->imageSize);

	tmp2=cvCloneImage(cl);
	surfDetDes(cl,tpts ,false, 5, 4, 2, 0.00004f);
	IpPairVec matches;
	IpVec ipts, ref_ipts;
	CvPoint src_corners[4] = {cvPoint(0,0), cvPoint(80,0), cvPoint(80, 60),
cvPoint(0, 60)};
	CvPoint dst_corners[4];
	getMatches(tpts,*refpts,matches);

	int tt=0;
	tt=translateCorners(matches, src_corners, dst_corners);
	if (translateCorners(matches, src_corners, dst_corners))
   {
    // Draw box around object
    for(int i = 0; i < 4; i++ )
```

```
    {
      CvPoint r1 = dst_corners[i%4];
      CvPoint r2 = dst_corners[(i+1)%4];
      cvLine( cl, cvPoint(r1.x, r1.y),  cvPoint(r2.x, r2.y), cvScalar(255,255,255),
3 );
    }


    for (unsigned int i = 0; i < matches.size(); ++i)
      drawIpoint(cl, matches[i].first);
  }

        CvPoint cpt;
        cpt.x=((dst_corners[0].x)+(dst_corners[2].x))/2;
        cpt.y=((dst_corners[0].y)+(dst_corners[2].y))/2;
        p++;

                if(tt)
                {
                if((abs(ptr[2]-abs(dst_corners[0].x-dst_corners[1].x))>=30 ||
                abs(ptr[3]-abs(dst_corners[0].y-dst_corners[3].y))>=30 ||
                !isrect(dst_corners)) && p>3 )
                {
                        tt=0;
                }
                else
                {
                cvCvtColor(tmp2,cl ,CV_RGB2HSV);
                ptr[0]=cpt.x;ptr[1]=cpt.y;ptr[2]=abs(dst_corners[0].x-
                st_corners[1].x);ptr[3]=abs(dst_corners[0].y-dst_corners[3].y);
                crhist((unsigned char *)cl->imageData,hist,w,h,ptr);
                 dist2=.1*(double)exp(-2*pow(comphist(hist,refhist),2));
                }
                }
        for(int i=0;i<N;i++)
        {
                if(tt && dist2>.05  )
```

```
        {
        updt[i][0]=cpt.x;
        updt[i][1]=cpt.y;
        updt[i][2]=ptr[2];
        updt[i][3]=ptr[3];
        updt[i][4]=1;
        updt[i][5]=1;
        updt[i][8]=1;
        tot++;
        }
        else
           {

        double pt[4];
        for(int k=0;k<4;k++)
        {
                pt[k]=updt[i][k];
        }

        cvCvtColor(tmp2,cl, CV_RGB2HSV);
         crhist((unsigned  char *)cl->imageData,hist,w,h,pt);
        dist2=.1*(double)exp(-100*pow(comphist(hist,refhist),2));
        updt[i][8]=dist2;
        tot+=updt[i][8];

        }
        }

        for(int i=0;i<N;i++)

        {
        updt[i][8]/=(double)tot;
        }
}
void exobj(double updt[][9],double *obj)
{
for(int i=0;i<8;i++)
```

```
{
	obj[i]=0;
for(int j=0;j<N;j++)
{
	obj[i]+=(double)updt[j][8]*updt[j][i];

}
}

}


void init(unsigned char *img,double *refhist,IpVec *refpts,double updt[][9],int w,int h)
{
	double obj[8]={0};
	unsigned char pix[60*80*3]={0,0};

	obj[0]=w/2;obj[1]=h/2;obj[2]=80;obj[3]=60;obj[4]=obj[5]=1;obj[6]=obj[7]=0;
	int x=obj[0],y=obj[1],wt=obj[2],ht=obj[3];
	int p=0,l=0;

	IpVec pt;
	IplImage *im;
	im=cvCreateImage(cvSize(w,h),IPL_DEPTH_8U,3);
	memcpy(im->imageData,img,im->imageSize);
	surfDetDes(im,pt,false,5,4,2,0.00004f);

	 p=0;
	 for(int k=0;k<pt.size();k++)
	 {
		if((pt.at(k).x>=(w/2-wt/2)) && (pt.at(k).x<=(w/2+wt/2)) && (pt.at(k).y>=(h/2-ht/2)) && (pt.at(k).y<=(h/2+ht/2)) )
	{
			Ipoint tmp;
			pt.at(k).x-=(w/2-wt/2);
```

```
                    pt.at(k).y-=(h/2-ht/2);
                    (*refpts).push_back(pt.at(k));

                }
        }
        for(int i=0;i<N;i++)
        {

                updt[i][0]=obj[0]+(rand()%60-30);
                updt[i][1]=obj[1]+(rand()%60-30);
                updt[i][2]=obj[2];
                updt[i][3]=obj[3];
                updt[i][4]=obj[4];
                updt[i][5]=obj[5];
                updt[i][6]=obj[6];
                updt[i][7]=obj[7];
                updt[i][8]=(double)1/N;

        img[3*(int)(w*updt[i][1]+updt[i][0])]=0;img[3*(int)(w*updt[i][1]+updt[i
][0])+1]=255;img[3*(int)(w*updt[i][1]+updt[i][0])+1]=0;
        }
}

void resample(unsigned char *img,double updt[][9],double *pconf,int w,int
h,int t)
{
        long double obj[8]={0},tot=0;
        double temp[N][10];

        temp[0][9]=0;
        for(int i=0;i<N;i++)
        {
                for(int j=0;j<9;j++)
                {
                        temp[i][j]=updt[i][j];
                }
                if (i>0)
```

```
        temp[i][9]=temp[i-1][9]+updt[i][8];
        tot+=updt[i][8];
}

for(int i=0;i<N;i++)
{
        double tmp=rand()%100+1;
        tmp=tmp/100;
        updt[i][1]=temp[N-1][1];
        updt[i][0]=temp[N-1][0];
        updt[i][4]=temp[N-1][4];
        updt[i][5]=temp[N-1][5];
        updt[i][8]=1/N;
        int rt=0,lt=N,key=(rt+lt)/2;
        while(rt!=lt+1 && lt!=rt+1)
        {
        key=(rt+lt)/2;
        if(temp[key][9]==tmp)
        {
                break;
        }
        else if(temp[key][9]>tmp)
        {
                lt=key;
        }
        else if(temp[key][9]<tmp)
        {
                rt=key;
        }
        }

        updt[i][1]=temp[key][1];
        updt[i][0]=temp[key][0];
        updt[i][3]=temp[key][3];
        updt[i][2]=temp[key][2];
        updt[i][4]=temp[key][4];
        updt[i][5]=temp[key][5];
```

```
            updt[i][6]=temp[key][6];
            updt[i][7]=temp[key][7];
            updt[i][8]=(double)1/N;

        img[3*(int)(w*updt[i][1]+updt[i][0])]=0;img[3*(int)(w*updt[i][1]+updt[i
][0])+1]=255;img[3*(int)(w*updt[i][1]+updt[i][0])+1]=0;
        }
}


void crhist(unsigned char *img,double *hist,int w,int h,double obj[4])
{
        long int i=0;
        int x,y,wt,ht,tot=0;
         x=obj[0];y=obj[1];wt=obj[2];ht=obj[3];
        double mr=0,mg=0,mb=0;
        for(i=0;i<=(257*4);i++)
        {

            hist[i]=0;
        }

        for(i=w*(y-ht/2);i<w*h && i<w*(y+ht/2) && i>0; i++)
        {

            if(i%w > x-wt/2 && i%w < x+wt/2)
            {

                tot++;
            int     r=img[3*i],g=img[3*i+1],b=img[3*i+2];
            hist[r]++;
            hist[256+g]++;
            hist[256*2+b]++;

            }

        }
```

```
        mr=mg=mb=wt*(ht-1);
        for(i=0;i<256;i++)
        {
                hist[i]/=mr;
                hist[256+i]/=mg;
                hist[256*2+i]/=mb;
        }



}

void rep(IplImage *img,double obj[4],int w,int h)
{
                int x1,y1,wt,ht,tot=0;
                CvPoint  p1,p2,p3,p4,c;

         x1=obj[0];y1=obj[1];wt=obj[2];ht=obj[3];
         p1.x=x1-wt/2;p1.y=y1-ht/2;p2.x=x1-
wt/2;p2.y=y1+ht/2;p3.x=x1+wt/2;p3.y=y1-ht/2;p4.x=x1+wt/2;p4.y=y1+ht/2;
         CvMat im;
         cvRectangle(img,p1,p4,cvScalar(0,255,0,100),1,8,0);



}




float comphist(double *hist1,double *hist2)
{
        double dist[3]={0.0,0.0,0.0};
        for(int i=0;i<256;i++)
        {

                dist[0]+=sqrt(hist1[i]*hist2[i]);
```

```
            dist[1]+=sqrt(hist1[256+i]*hist2[256+i]);
            dist[2]+=sqrt(hist1[256*2+i]*hist2[256*2+i]);

        }

        dist[0]=sqrt(abs(1-dist[0]));
        dist[1]=sqrt(abs(1-dist[1]));
        dist[2]=sqrt(abs(1-dist[2]));

        return((dist[0]+dist[1]+dist[2])/3);

}
```

**APPENDIX-C**

*Main.cpp*

```
#include "main.h"
#include "testapp.h"

// set up DMA descriptors (one for each frame, then repeat)
// small descriptor model, only start address needs to be fetched

tDMA_descriptor DMA_PPI0_first = {&DMA_PPI0_second, sFrame0};
tDMA_descriptor DMA_PPI0_second =      {&DMA_PPI0_third , sFrame1};
tDMA_descriptor DMA_PPI0_third =       {&DMA_PPI0_fourth, sFrame2};
tDMA_descriptor DMA_PPI0_fourth =      {&DMA_PPI0_first , sFrame3};


volatile int current_in_Frame  = -1;            // 0, 1, 2 or 3  ... indicates the last
frame that was received COMPLETELY

bool Set_PACK32 = false;
bool Set_Entire_Field = false;


// User program
void main() {

        // unblock Core B if dual core operation is desired
#ifndef RUN_ON_SINGLE_CORE
        *pSICA_SYSCR &= 0xFFDF; // clear bit 5 to unlock
#endif

        // set Clocks
        Set_PLL( (short)(CORECLK/CLKIN), (short)(CORECLK/SYSCLK));
                        // sets Core and System Clocks to the values defined in
system.h
```

```
	// initialise SDRAM
	InitSDRAM();
	*pTC_PER = 0x0770;			// set DMA traffic control register
to favour unidirectional transfers to SDRAM

	// initialise Video Encoder ADV7179
	Reset_ADV7179();

	// initialise Video Decoder ADV7183
	Reset_ADV7183();


	// initialise PPI0 and associated DMA channel for Video IN
	current_in_Frame  = -1;				// no frames received yet
	semaphore_frames_received = false;   // do not start output stream yet
	semaphore_frames_completed = false;	// do not start output stream
yet

	Set_Entire_Field = false;
#ifdef ENTIRE_FIELD_MODE
	Set_Entire_Field = true;
#endif

	Set_PACK32 = false;
#ifdef PACK_32
	Set_PACK32 = true;
#endif



	InitPPI0(Set_Entire_Field, Set_PACK32, &DMA_PPI0_first,
PIXEL_PER_LINE, LINES_PER_FRAME);



	// initialise Interrupts
	InitInterrupts_coreA();
```

```
	// enable transfers
	EnablePPI0();

w=PIXEL_PER_LINE,h=LINES_PER_FRAME,t=0;

	// main loop, just wait for interrupts
	while(1) {
	//	idle();		//	do nothing
		// check for PPI framing error
		if (*pPPI0_STATUS & FT_ERR)
		{
			// error occurred -- clear error and restart video transfer
			*pPPI0_STATUS &= ~FT_ERR;
			semaphoreResetVideo = true;

			while(semaphoreResetVideo);	// wait for core B to reset
video

			DisablePPI0();
			current_in_Frame  = -1;		// no frames received yet
			semaphore_frames_received = false;

			InitPPI0(Set_Entire_Field, Set_PACK32,
&DMA_PPI0_first, PIXEL_PER_LINE, LINES_PER_FRAME);

			EnablePPI0();
		}

		if (semaphore_frames_received)
		{
			semaphore_frames_received = false;
			//Call process Video with the currentFrame -1 (the frame just
filled)
			ProcessVideo((current_in_Frame-1) % 4);
		}
```

```
        }       // while(1)


}               // main

void ProcessVideo(int FrameToProcess)
{
        pix2=cvCreateImage( cvSize(w,h), IPL_DEPTH_8U, 3);
        switch(FrameToProcess)
        {
case 0:memcpy(pix2->imageData,(const void *)sFrame0,pix2->imageSize);
break;
case 1:memcpy(pix2->imageData,(const void *)sFrame1,pix2->imageSize);
break;
case 2:memcpy(pix2->imageData,(const void *)sFrame2,pix2->imageSize);
break;
case 3:memcpy(pix2->imageData,(const void *)sFrame3,pix2->imageSize);
        }



                pix3=cvCloneImage(pix2);


        if(t==0)

        {
                (refpts).clear();

cvCvtColor(pix3,pix2, CV_RGB2HSV);

crhist((unsigned char *)pix2->imageData,refhist,w,h,obj);

obj[0]=w/2;obj[1]=h/2;obj[2]=80;obj[3]=60;obj[4]=obj[5]=1;obj[6]=obj[7]=0;

init((unsigned char *)pix3->imageData,refhist,&refpts,updt,w,h);
```

```
obj[0]=w/2;obj[1]=h/2;obj[2]=80;obj[3]=60;obj[4]=obj[5]=1;obj[6]=obj[7]=0;

rep(pix3,obj,w,h);

}
if(t>0)
{
evolve((unsigned char *)pix3->imageData,updt,refhist,w,h);
msrmtupdate((unsigned char *)pix3->imageData,updt,&refpts,refhist,w,h);
 exobj(updt,obj);
rep(pix3,obj,w,h);
resample((unsigned char *)pix3->imageData,updt,0,w,h,t);
}

switch(FrameToProcess)
        {
case 0: memcpy((void *)sFrame0,pix3->imageData,pix2->imageSize);break;
 case 1:memcpy((void *)sFrame1,pix3->imageData,pix2->imageSize);break;
case 2:memcpy((void *)sFrame2,pix3->imageData,pix2->imageSize);break;
case 3:memcpy((void *)sFrame3,pix3->imageData,pix2->imageSize);
        }
        //Do processing in here
        //Set Semaphore to indicate that the frame is complete
        semaphore_frames_completed = true;
}
```