

THESIS/DISSERTATION TRANSMITTAL FORM

Name of Student Ying Liu

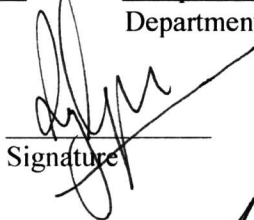
Title of Thesis A JAVA FRAMEWORK FOR OBJECT DETECTION AND TRACKING

We the undersigned members of the Committee advising this thesis/dissertation, have ascertained that in every respect it acceptably fulfills the final requirement for the degree of Master of Science in the Computer Information Science.

Specify degree

Department or School

Dr. Roy George
Major Advisor


Signature

Computer Information Science
Department or School


08-07-07
Date

Dr. Khalil A. Shujaee
Name


Signature

08-07-07
Date

Dr. Hsin-Chu Chen
Name


Signature

08/07/07
Date

As Chair of the Department of Computer Information Science, I have verified that this manuscript meets the School's/Department's standards of form and content governing theses or dissertations for the degree sought.

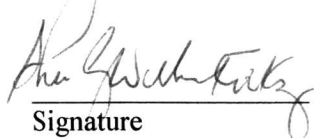
Dr. Roy George
Chair


Signature

08-07-07
Date

As Dean of the School of Arts and Sciences, I have verified that this manuscript meets the School's regulations governing the content and form of theses or dissertations.

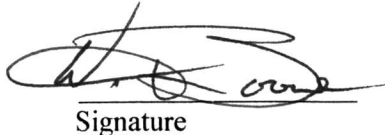
Dr. Shirley Williams-Kirksey
Dean


Signature

2/13/08
Date

As Dean of Graduate Studies, I have verified that this manuscript meets the University's regulations governing the content and form of theses and dissertations.

Dr. William Boone
Dean of Graduate Studies


Signature

3/10/2008
Date

ABSTRACT

A JAVA FRAMEWORK FOR OBJECT DETECTION AND TRACKING

Ying Liu M. ED. Northeast Normal University, China, 2001

Advisor: Professor Roy George

Thesis dated December 2007

Object detection and tracking is an important problem in the automated analysis of video. There have been numerous approaches and technological advances for object detection and tracking in the video analysis. As one of the most challenging and active research areas, more algorithms will be proposed in the future. As a consequence, there will be the demand for the capability to provide a system that can effectively collect, organize, group, document and implement these approaches.

The purpose of this thesis is to develop one uniform object detection and tracking framework, capable of detecting and tracking the multi-objects in the presence of occlusion. The object detection and tracking algorithms are classified into different categories and incorporated into the framework implemented in Java. The framework can adapt to different types, and different application domains, and be easy and convenient for developers to reuse. It also provides comprehensive descriptions of representative methods in each category and some examples to aspire to give developers or users, who require a tracker for a certain application, the ability to select the most suitable tracking algorithm for their particular needs.

A JAVA FRAMEWORK
FOR OBJECT DETECTION AND TRACKING

A DISSERTATION
SUBMITTED TO THE FACULTY OF CLARK ATLANTA UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF COMPUTER SCIENCE

BY
YING LIU

DEPARTMENT OF COMPUTER INFORMATION SCIENCE

ATLANTA, GEORGIA

December 2007

R x

T 88

© 2007

YING LIU

All Rights Reserved

CLARK ATLANTA UNIVERSITY THESIS OR DISSERTATION
DEPOSITED IN THE ROBERT W. WOODRUFF LIBRARY

STATEMENT OF UNDERSTANDING

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Clark Atlanta University, I agree that the Robert W. Woodruff Library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to quote from, to copy from, or to publish this thesis/dissertation may be granted by the author or, in her absence, the Dean of the School of Arts and Sciences at Clark Atlanta University. Such quoting, copying, or publication must be solely for scholarly purposes and must not involve potential financial gain. It is understood that any copying from or publication of this thesis/dissertation which involves potential financial gain will not be allowed without written permission from the author.



Signature of Author



Date

NOTICE TO BORROWERS

All theses and dissertations deposited in the Robert W. Woodruff Library must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis/dissertation is:

Name: Ying Liu

Street Address: 3365 Rae Place

City, State, and Zip: Lawrenceville, Georgia, 30044

The director of this thesis/dissertation is:

Professor: Roy George

Department: Computer Information Science

School: Arts and Sciences Clark Atlanta University

Office Telephone: (404) 880-6944

Users of this thesis/dissertation not regularly enrolled as students of the Atlanta University Center are required to attest accordance of the preceding stipulations by signing below. Libraries borrowing this thesis/dissertation for use of patrons are required to see that each user records here the information requested.

NAME OF USER	ADDRESS	DATE	TYPE OF USE
_____	_____	_____	_____
	_____		_____
_____	_____	_____	_____
	_____		_____

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Roy George, for his valuable guidance, encouragement, and patience throughout my course of study and thesis work.

I also extend my gratitude to Zhijian Liu and Bin Wu. Working in a group, learning from them and discussing my results with them is always pleasant and enjoyable.

I really would like to thank my parents and my sisters for all support during my study. I thank my husband, Jingxue Zhou for love and encouragement and our son Carl Zhou for always making us smile.

There is no doubt that without their tremendous encouragement and unfailing support, I would not have made it throughout my studies.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	ii
1. INTRODUCTION	1
1.1 Problems	2
1.2 ViReS.....	4
1.3 Contributions	4
1.4 Layout of the thesis.....	5
2. LITERATURE REVIEW	6
2.1 Video and Change Detection issues	6
2.1.1 Video Structure.....	6
2.1.2 Change detection	7
2.2 Object Representation issues	8
2.2.1 Objects can be represented by their shapes.	9
2.2.2 Objects can be represented by the appearance features.....	11
2.3 Object detection issues	13
2.3.1 Detection categories.....	13
2.3.2 Object detection algorithms.....	14
2.4 Object tracking issues	18
2.4.1 Feature selection for object tracking.....	18
2.4.2 Object tracking categories	20

TABLE OF CONTENTS

2.4.3 Object tracking algorithms	22
2.5 Software Frameworks	30
2.5.1 Why use framework?	30
2.5.2 How is a framework designing?	31
2.5.3 How to classify frameworks?	32
2.5.4 How to use a framework?	33
3. FRAMEWORK DESIGN	35
3.1 High level design	35
3.2 Change detection design	37
3.3 Object Detection design	38
3.3.1 Point Detector	39
3.3.2 Segmentation Detector	40
3.3.3 Background Subtraction Detector	40
3.3.4 Supervised Learning Detector	41
3.4 Object representation design	42
3.4.1 Shape Object	42
3.4.2 Appearance Object	43

TABLE OF CONTENTS

3.5 Object tracking design	44
3.5.1 Object Tracker	45
3.5.2 Point tracker	45
3.5.3 Kernel tracker	46
3.5.4 Silhouette tracker	47
3.6 Viewer Design	49
4. IMPLEMENTATION.....	50
4.1 Why use Java?	50
4.1.1 Interface versus abstract class.....	50
4.1.2 Java Media Framework (JMF).....	51
4.2 Implementation	52
4.2.1 Change detection implementation	52
4.2.2 Object representation implementation.....	57
4.2.3 Object Detector implementation.....	61
4.2.4 Object Tracker implementation	67
4.2.5 Viewer implementation	73

TABLE OF CONTENTS

5. EXPERIMENTS AND RESULTS.....	74
5.1 Change (Cut) detection experimental results.....	74
5.2 Object (Face) detection experimental results	76
5.3 Object (Particle Filter) tracking experimental results.....	77
5.4 Analysis for experimental results	79
6. CONCLUSION.....	80
6.1 Future Work.....	80
7. REFERENCES.....	82

LIST OF FIGURES

Figure 2.1	Video Structure	7
Figure 2.2	Object representations	11
Figure 2.3	Different tracking catogories.....	22
Figure 3.1	Overview of the framework	35
Figure 3.2	Class diagram of Change detection.....	38
Figure 3.3	Diagram of ObjectDetector Design.....	39
Figure 3.5	Diagram of SegDetector Design	40
Figure 3.6	Diagram of BGDetector Design.....	41
Figure 3.7	Diagram of SupDetector Design	41
Figure 3.8	Diagram of VideoObject representation design	42
Figure 3.9	Diagram of ShapeObject representation design	43
Figure 3.10	Diagram of appearance object representation design.....	44
Figure 3.11	Diagram of ObjectTrackter Design	45
Figure 3.12	Diagram of PointTracker Design	46
Figure 3.13	Diagram of KernelTracker Design	47
Figure 3.14	Diagram of SilhouetteTracker Design.....	48
Figure 3.15	Diagram of Viewer Design	49
Figure 4.1	Histogram based change detection algorithm	54
Figure 4.2	Measures for comparing histogram.....	55

LIST OF FIGURES

Figure 4.3	Sample codes for HistoPixelChgDetector.....	56
Figure 4.4	Sample Codes for VideoObject.....	58
Figure 4.5	Sample codes for ShapeObject.....	59
Figure 4.6	Sample codes for PrimitiveShapeObject.....	60
Figure 4.7	Sample codes for RectanglePrimitiveShapeObject.....	60
Figure 4.8	Sample codes for ObjectDetector.....	61
Figure 4.9	Cascaded structure.....	63
Figure 4.10	Block diagram of Adaboost algorithm	64
Figure 4.11	Procedure of Discrete AdaBoost	64
Figure 4.12	Sample Codes for AdaBoostSupDector	67
Figure 4.13	Sample codes for ObjectTracker	68
Figure 4.14	A Dynamic Filtering System.....	69
Figure 4.15	Block diagram for particle filter algorithm	70
Figure 4.16	Particle filter algorithm	70
Figure 4.17	Sample codes for ParticleFilterSatPointTracker	72
Figure 4.18	Sample code for Viewer.....	73
Figure 5.1	Shows change detection results for one video of flights.....	75
Figure 5.2	Change detection results for one news video	75
Figure 5.3	Shows change detection results for one video taken by one camera	76

LIST OF FIGURES

Figure 5.4	Show fade detection results for one video of flights.....	76
Figure 5.5	Shows Face detection results.....	77
Figure 5.6	Start tracking Sub1 on Frame 111, 113, 127.....	77
Figure 5.7	Tracking Sub2 Start on Frame 111 and End on frame114.....	78
Figure 5.8	Tracking Sub0 on Frame 126, 143, 170 in MVI_0078.avi	78
Figure 5.9	Tracking Sub0 on Frame 198-284.....	79

LIST OF TABLES

Table 2.1	Object Detection Categories.....	15
Table 2.2	Tracking Categories.....	22

CHAPTER 1

INTRODUCTION

Videos are sequences of images, each of which is called a frame. The contents of two consecutive frames are usually closely related. Visual content can be modeled through a hierarchy of abstractions: raw pixels, features, objects, and human concepts. At the lowest abstraction level, objects are simply aggregations of raw pixels from an image. The next higher abstraction for representing image is at the feature level. A feature is a distinguishing primitive characteristics or attribute of an image. These features are interpreted as objects and their attributes. One or more objects and relationships among them are supplied through the human level concepts. Although automatic detection and recognition, and tracking methods are available for certain objects and their attributes, their effectiveness is highly dependent on image complexity.

Object detection in videos involves verifying the presence of an object in image sequences and possibly locating it precisely for recognition. Object detection is a straightforward solution to the content-based video indexing and analysis. Object tracking monitors an object's spatial and temporal changes during a video sequence. Object tracking is one of the most challenging and active research areas in digital video processing and computer vision. Tracking over a time is difficult due to the complexity of both the scene and the objects to be tracked. The level of difficulty depends on how the object to be detected and tracked is defined. These two processes are

closely related because tracking usually starts with detecting objects, while detecting an object repeatedly in subsequent image sequence is often necessary to help and verify tracking.

The use of object detection and tracking is pertinent in the tasks of motion-based recognition, automated surveillance, video indexing, human-computer interaction, traffic monitoring, vehicle navigation, etc.

1.1 Problems

There are three key steps in video analysis: detection of interesting moving objects, tracking of such objects from frame to frame, and analysis of object tracks to recognize their properties and behaviors.

The motion of the object is relative to the background. It is difficult to detect moving object owing to the motion of the background. A common approach for object detection uses information in one single frame but temporal information from a sequence of frames can improve the precision of detecting.

Tracking objects can be complex due to loss of information caused by projection of the 3D world on a 2D image, noise in images, complex object motion, non-rigid or articulated nature of objects, partial and full object occlusions, complex object shapes, scene illumination changes, and real-time processing requirements. Tracking can be simplified by imposing constraints on the motion and/or appearance of objects. For example, almost all tracking algorithms assume that the object motion is smooth with no abrupt changes. One can further constrain the object motion to be of constant velocity or constant acceleration based on a *priori* information. Prior knowledge about the number

and the size of objects, or the object appearance and shape, can also be used to simplify the problem.

Several approaches for object detection and tracking have been proposed. These primarily differ from each other based on the way they approach the following questions: Which object representation is suitable for detection and tracking? Which image features should be used? How should the motion, appearance, and shape of the object be modeled?

The answers to these questions depend on the context/environment in which the detection and tracking are performed and the end use to which the detection and tracking information is being sought. Several detection and tracking methods have been proposed which attempt to answer these questions for different scenarios. Is there a simple way to understand, implement, and use them easily and fast? The answer is a framework.

A framework is a skeletal group of modules that may be tailored for building domain-specific applications, resulting in increased productivity. The implementation of a framework is an important part to be reused. Frameworks offer flexibility to developers and provide well-documented and easy to use solutions for common practices in the industry.

The goal of this thesis is to develop a uniform Java framework that groups tracking methods into broad categories and provides comprehensive descriptions of representative methods in each category. It provides developers and users the ability to select the most suitable tracking algorithm for their particular needs.

1.2 ViReS

The framework for Object Detection and Tracking is one of sub-frameworks for ViRes[1]. The ViReS (Video Index and Retrieval System) is a video analysis framework in Java. The system provides automatic indexing and querying based on visual contents. An important task in analyzing video content is to detect segment boundaries. A common approach for quick browsing is to detect changes that can be abrupt (Cut) or they can be gradual (Fade, Dissolve, Wipe). A video is divided into different shots by using one or more of the selected techniques. Object-based querying involves detection and tracking of moving objects and queries based on an example of the object provided or selected by the user.

1.3 Contributions

The work described here involves the design and implementation of a suite of Java interfaces, classes, and testing results for the object detection and tracking framework.

The major contributions are:

- proposing a Java object detection and tracking framework.
- designing and implementing the object detection and tracking framework.
- implementing histogram-based Cut detection, Fade Detection, Face Detection, Particle filter object tracking algorithms in Java as examples.
- performing experiments and tests.

1.4 Layout of the thesis

Some background on video and object tracking issues is presented in Chapter 2. The Framework design is presented in Chapter 3. The Chapter 4 describes the implementation of the framework and classes in Java in detail. The experiments and testing results are provided in Chapter 5. Chapter 6 concludes the thesis.

CHAPTER 2

LITERATURE REVIEW

Detection and tracking are two major research components in the analysis of computer vision. Detection locates potential boundaries and objects while tracking identifies and follows these objects as they move through images. In this section, we review the research issues in video, change detection, object representation, object detection and object tracking.

2.1 Video and Change Detection issues

Video is the most popular source of multimedia information. It combines all other media information, such as text, image, graphic, audio and etc., into a single data stream. Scene change detection is an effective method for segmenting a video sequence into significant shots and has been recognized as an important technology for video analysis, editing, indexing, and motion compensation.

2.1.1 Video Structure

Video in Figure 2.1 consists of a sequence of scenes [2]. A scene is defined as a collection of one or more adjoining shots that focus on one object or objects of interests. A shot is a sequence of frames captured by a single camera in a single continuous action in time and space. A frame is a still image which composes a complete video.

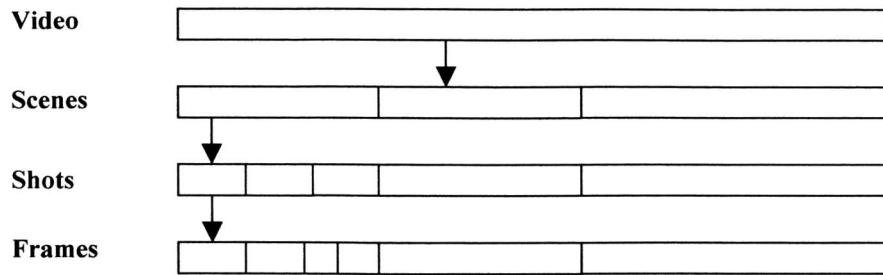


Figure 2.1 Video Structure

There are different types of transitions or boundaries between shots. A cut is an abrupt transition between two shots that occurs between two adjacent frames. A fade is a gradual change in brightness, either starting or ending with a black frame. A dissolve is similar to a fade except that it occurs between two shots. The images of the first shot get dimmer and those of the second shot get brighter until the second replaces the first. Other types of shot transitions include wipes and computer generated effects such as morphing.

2.1.2 Change detection

Change detection is to find shot boundary having transitions, and delimit the start and the end of the video shots.

1. Why Change detection?

- (1) Change detection is elementary core technology for constructing efficient content based video indexing and searching.
- (2) Change detection processes the video content in a more efficient way.

2. Change Detection Methods

The basic idea is to compute the differences between consecutive frames or groups of frames. Change detection methods vary in the way these differences are computed [3]:

(1) Pixel based: difference in subsequent frames of pixels at corresponding positions. It includes pixel-based analysis and histogram-based analysis.

- Pixel-based analysis:

Pixel Comparison counts the number of pixels changed from a frame to the next. It is less sensitivity to object motion and noisy.

- Histogram-based analysis:

Global histogram comparison produces relatively accurate results compared to others but local histogram comparisons produce the most accurate results compared to others.

(2) Block based: compare statistics of corresponding blocks. Block-based approaches use local characteristics to increase the robustness to camera and object movement.

(3) Frame based: summarize the whole frame into one measure and compare this with the same measure for the next frame.

(4) Object based: detection based on measures indicating how the objects change from frame to frame.

2.2 Object Representation issues

An object is an artifact 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. An important task in object detection and tracking is object representation, i.e., how to model an object. The way of representing an object affects the successive algorithms of object detection and tracking.

A representation of the object that captures its characteristics is needed. Object representation may be classified into two categories: object-centered or shape-based and view-centered or appearance-based. A typical feature of shape-based approaches is the decomposition of objects in a set of primitive shapes such as points, boxes, silhouettes and blobs. The appearance-based (view-based) object representation that may include photometric as well as purely geometric information renounces the use of explicit object models. Instead, they base on a set of characteristic 2D views of the object. In both cases, the visual feature scope, color, texture, shape, or motion, cues and their combination can provide appropriate representations.

2.2.1 Objects can be represented by their shapes.

Objects can be represented by many kinds of shapes such as points, primitive geometric shapes, object silhouette and contour, articulated shape models, and skeletal models.

1. Points.

The object is represented by a point like centroid (Figure 2.2(a)) [4] or by a set of points (Figure 2.2(b)) [5]. In general, the point representation is suitable for tracking objects that occupy small regions in an image.

2. Primitive geometric shapes.

The object shape is represented by a rectangle, ellipse (Figure 2.2(c), (d)) [6], etc. Object motion for such representations is usually modeled by translation, affine, or projective (homography) transformation. Though primitive geometric shapes are more suitable for representing simple rigid objects, they are also used for tracking nonrigid objects.

3. Articulated shape models.

Articulated objects are composed of body parts that are held together with joints. The relationship between the parts is governed by kinematic motion models, for example, joint angle, etc. In order to represent an articulated object, one can model the constituent parts using cylinders or ellipses as shown in Figure 2.2(e).

4. Skeletal models.

Object skeleton can be extracted by applying medial axis transform to the object silhouette [7]. This model is commonly used as a shape representation for recognizing objects [8]. Skeleton representation can be used to model both articulated and rigid objects (see Figure 2.2(f)).

5. Object silhouette and contour.

Contour representation defines the boundary of an object (Figure 2.2(g), (h)). The region inside the contour is called the silhouette of the object (see Figure 2.2(i)). Silhouette and contour representations are suitable for tracking complex nonrigid shapes [9].

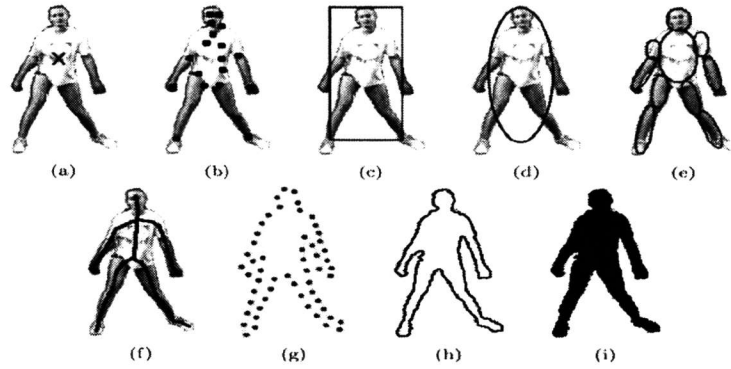


Figure 2.2 Object representations

(a) centroid, (b) multiple points, (c) rectangular patch, (d) elliptical patch, (e) part-based multiple patches, (f) object skeleton, (g) complete object contour, (h) control points on object contour, (i) object silhouette.

2.2.2 Objects can be represented by the appearance features.

An object can be represented by appearance features such as probability densities of the object appearance, shapes, templates, active appearance models, and multi-view appearance models.

1. Probability densities of object appearance.

The probability density estimates of the object appearance can either be parametric, such as Gaussian [10] and a mixture of Gaussians [11], or nonparametric, such as Parzen windows [12] and histograms [6]. 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).

2. Templates.

Templates are formed using simple geometric shapes or silhouettes [13]. 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.

3. Active appearance models.

Active appearance models are generated by simultaneously modeling the object shape and appearance [14]. In general, the object shape is defined by a set of landmarks. Similar to the contour-based representation, the landmarks can reside on the object boundary or, alternatively, they can reside inside the object region. For each landmark, an appearance vector is stored, which is in the form of color, texture, or gradient magnitude. Active appearance models require a training phase where both the shape and its associated appearance is learned from a set of samples using, for instance, the principal component analysis.

4. Multi-view appearance models.

These models encode different views of an object. One approach to represent the different object views is to generate a subspace from the given views. Subspace approaches, for example, Principal Component Analysis (PCA) and Independent Component Analysis (ICA), have been used for both shape and appearance representation [15], [16]. An alternative approach to learn the different views of an object is by training a set of classifiers, for example, support vector machines [17] or Bayesian networks [18]. A limitation of multiview appearance models is that the appearances in all views are required ahead of time.

2.3 Object detection issues

Object detection determines whether or not the object is present in an image, and, if present, determine the locations and sizes of each object. Reliable object detection systems are required as a front-end in numerous applications. For example, face detection is the first stage of many human computer interaction systems. Object detection deals with determining if an instance of a given class of objects (for examples cars, faces, etc.) is present or not in an image.

2.3.1 Detection categories

Successful object detection systems are based on the learning of object appearance using large collections of examples. The object detection systems that have been developed fall into one of 4 major categories: point detector, background subtraction, segmentation, and supervised learning.

1. Point Detector

Point detectors are used to find interest points in images which have an expressive texture. Interest points have been long used in the context of motion, stereo, and tracking problems. A desirable quality of an interest point is its invariance to changes in illumination and camera viewpoint.

2. Background Subtraction

Object detection can be achieved by building a representation of the scene called the background model and then finding deviations from the model for each incoming frame. Any significant change in an image region from the background model signifies a moving object. The pixels constituting the regions undergoing change are marked for

further processing. Usually, a connected component algorithm is applied to obtain connected regions corresponding to the objects. This process is referred to as the background subtraction.

3. Segmentation

The aim of image segmentation algorithms is to partition the image into perceptually similar regions. Every segmentation algorithm addresses two problems, the criteria for a good partition and the method for achieving efficient partitioning [19].

4. Supervised Learning

Object detection is performed by learning different object views automatically from a set of examples by means of a supervised learning mechanism. Learning of different object views waives the requirement of storing a complete set of templates. Given a set of learning examples, supervised learning methods generate a function that maps inputs to desired outputs. A standard formulation of supervised learning is the classification problem where the learner approximates the behavior of a function by generating an output in the form of either a continuous value, which is called regression, or a class label, which is called classification. In context of object detection, the learning examples are composed of pairs of object features and an associated object class where both of these quantities are manually defined.

2.3.2 Object detection algorithms

A common approach to object detection is to use information in a single frame. However, some object detection algorithms make use of the temporal information computed from a sequence of frames to reduce the number of false detections. This temporal information is usually in the form of frame differencing, which highlights

changing regions in consecutive frames. Given the object regions in the image, it is then the tracker's task to perform object correspondence from one frame to the next to generate the tracks. Table 2.1 shows 4 categories of object detection methods and representative works of each type.

Table 2.1: Object Detection Categories

Categories	Representative Work
Point detector	Moravec's detector [20], Harris detector [21], Scale Invariant Feature Transform [22]. Affine Invariant Point Detector [23].
Background Subtraction	Mixture of Gaussians[24], Eigenbackground[25], Wall flower [26], Dynamic texture background [27].
Segmentation	Mean-shift [28], Graph-cut [19], Active contours [29].
Supervised Learning	Support Vector Machines [30], Neural Networks [31], Adaptive Boosting [32].

1. Point Detector

In the literature, commonly used interest point detectors include Moravec's interest operator [20], Harris interest point detector [21], KLT detector [33], and SIFT detector [22]. To find interest points, Moravec's operator computes the variation of the image intensities in a patch in the horizontal, vertical, diagonal, and anti-diagonal directions and selects the minimum of the four variations as representative values for the window. Quantitatively both Harris and KLT emphasize the intensity variations using very similar measures. The only difference is the additional KLT criterion that enforces a predefined spatial distance between detected interest points. In order to introduce robust

detection of interest points under different transformations, Lowe [22] introduced the SIFT (Scale Invariant Feature Transform) method.

2. Background Subtraction

Background subtraction became popular following the work of Wren et al. [34]. In order to learn gradual changes in time, Wren et al. [34] propose modeling the color of each pixel of a stationary background with a single Gaussian. However, a single Gaussian is not a good model for outdoor scenes [35] since multiple colors can be observed at a certain location due to repetitive object motion, shadows, or reflectance. Stauffer and Grimson [24] use a mixture of Gaussians to model the pixel color.

An alternate approach for background subtraction is to represent the intensity variations of a pixel in an image sequence as discrete states corresponding to the events in the environment. Rittscher et al. [36] use Hidden Markov Models (HMM) to classify small blocks of an image as belonging to one of these three states. In the context of detecting light on and off events in a room, Stenger et al. [37] use HMMs for the background subtraction. The advantage of using HMMs is that certain events, which are hard to model correctly using unsupervised background modeling approaches, can be learned using training samples. Instead of modeling the variation of individual pixels, Oliver et al. [25] propose a holistic approach using the eigenspace decomposition. One limitation of the aforementioned approaches is that they require a static background. This limitation is addressed by Monnet et al. [27], and Zhong and Sclaroff [38]. Both of these methods are able to deal with time-varying background (e.g., the waves on the water, moving clouds, and escalators).

3. Segmentation

For the segmentation problem, Comaniciu and Meer [39] propose the mean-shift approach to find clusters in the joint spatial and color space. Shi and Malik [19] propose the normalized cut to overcome the over-segmentation problem. In their approach, the cut not only depends on the sum of edge weights in the cut, but also on the ratio of the total connection weights of nodes in each partition to all nodes of the graph. Object segmentation is achieved by evolving a closed contour to the object's boundary, such that the contour tightly encloses the object region. A contour is typically placed outside the object region and shrunk until the object boundary is encountered [40], [29].

4. Supervised Learning

Supervised learning methods usually require a large collection of samples from each object class. Additionally, this collection must be manually labeled. It is important to use a set of features that discriminate one class from the other. Once the features are selected, different appearances of an object can be learned by choosing a supervised learning approach. These learning approaches include, but are not limited to, neural networks [31], adaptive boosting [32], decision trees [41], and support vector machines [30].

Boosting is an iterative method of finding a very accurate classifier by combining many base classifiers, each of which may only be moderately accurate. Viola et al. [32] used the Adaboost framework to detect pedestrians. In their approach, perceptrons were chosen as the weak classifiers which are trained on image features extracted by a combination of spatial and temporal operators. As a classifier, Support Vector Machines (SVM) are used to cluster data into two classes by finding the maximum marginal

hyperplane that separates one class from the other. In the context of object detection, Papageorgiou et al. [30] use SVM for detecting pedestrians and faces in images.

2.4 Object tracking issues

Object detection and object tracking processes are closely related because tracking usually starts with detecting objects, while detecting an object repeatedly in subsequent image sequence is often necessary to help and verify tracking.

Object tracking is useful for computer vision, robotic navigation, surveillance, image understanding, image indexing and retrieval etc. The aim of object tracking is to analyze those images in sequence and locate area of interest in image. Reliability and precision are important for tracking system.

2.4.1 Feature selection for object tracking

Selecting the right features plays a critical role in object 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 such as color, edges, optical flow and texture are as follows.

1. Color.

The apparent color of an object is influenced primarily by two physical factors: the spectral power distribution of the illuminant and 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 [42]. Additionally, the RGB dimensions are highly correlated. In contrast, HSV (Hue, Saturation, Value) is an approximately uniform color space. However, these color spaces are sensitive to noise [43]. In summary, there is no last word on which color space is more efficient, therefore a variety of color spaces have been used in tracking.

2. 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. Because of its simplicity and accuracy, the most popular edge detection approach is the Canny Edge detector [44]. An evaluation of the edge detection algorithms is provided by Bowyer et al. [45].

3. Optical Flow.

Optical flow is a dense field of displacement vectors which defines the translation of each pixel in a region. It is computed using the brightness constraint, which assumes brightness constancy of corresponding pixels in consecutive frames [46]. Optical flow is commonly used as a feature in motion-based segmentation and tracking applications. Popular techniques for computing dense optical flow include methods by Horn and Schunck [46], Lucas and Kanade [47], Black and Anandan [48], and Szeliski and Coughlan [49].

4. Texture.

Texture is a measure of the intensity variation of a surface which quantifies properties such as smoothness and regularity. Compared to color, texture requires a processing step to generate the descriptors. There are various texture descriptors: Gray-Level Cooccurrence Matrices (GLCM's) [50], Law's texture measures [51], wavelets [52] (orthogonal bank of filters), and steerable pyramids [53]. Like edge features, the texture features are less sensitive to illumination changes compared to color.

Mostly features are chosen manually by the user depending on the application domain. Among all features, color is one of the most widely used features for tracking. Comaniciu et al. [6] use a color histogram to represent the object appearance. Despite its popularity, most color bands are sensitive to illumination variation. Hence in scenarios where this effect is inevitable, other features are incorporated to model object appearance. Cremers et al. [54] use optical flow as a feature for contour tracking. Jepson et al. [55] use steerable filter responses for tracking. Alternatively, a combination of these features is also utilized to improve the tracking performance.

2.4.2 Object tracking categories

The aim of an object tracker is to generate the trajectory of an object over time by locating its position in every frame of the video. The object tracker may also provide the complete region in the image that is occupied by the object at every time instant. The object tracking systems can be divided into point, kernel and silhouette tracking categories.

1. Point Tracking.

Objects detected in consecutive frames are represented by points, and the association of the points is based on the previous object state which can include object position and motion. This approach requires an external mechanism to detect the objects in every frame. An example of object correspondence is shown in Figure 2.3(a).

2. Kernel Tracking.

Kernel refers to the object shape and appearance. For example, the kernel can be a rectangular template or an elliptical shape with an associated histogram. Objects are tracked by computing the motion of the kernel in consecutive frames (Figure 2.3(b)). This motion is usually in the form of a parametric transformation such as translation, rotation, and affine.

3. Silhouette Tracking.

Tracking is performed by estimating the object region in each frame. Silhouette tracking methods use the information encoded inside the object region. This information can be in the form of appearance density and shape models which are usually in the form of edge maps. Given the object models, silhouettes are tracked by either shape matching or contour evolution (see Figure 2.3(c), (d)). Both of these methods can essentially be considered as object segmentation applied in the temporal domain using the priors generated from the previous frames.

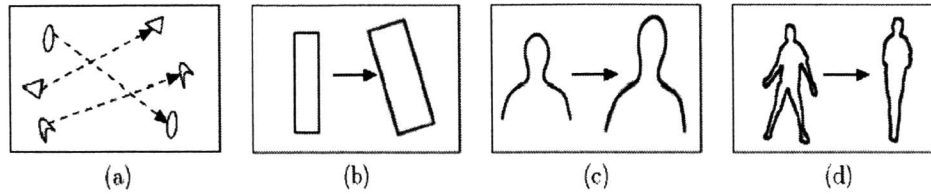


Figure 2.3 Different tracking categories

(a) Different tracking approaches. Multipoint correspondence, (b) parametric transformation of a rectangular patch, (c, d) Two examples of contour evolution.

2.4.3 Object tracking algorithms

We now briefly introduce the main tracking categories in Table 2.2, followed by a detailed section on each category.

Table 2.2 Tracking Categories

Categories	Representative Work
<i>Point Tracking</i>	
Deterministic methods	MGE tracker [56] GOA tracker [4]
Statistical methods	Kalman filter [57] JPDAF [58] PMHT [59]
<i>Kernel Tracking</i>	
Template and density based	Mean-shift [6] appearance models KLT [33] Layering [60]
Multi-view appearance models	Eigenttracking [16] SVM tracker [17]
<i>Silhouette Tracking</i>	
Contour evolution	State space models [61] Variational methods [62] Heuristic methods [63]
Matching shapes	Hausdorff [64] Hough transform [65] Histogram [66]

1. Point Tracking

Tracking can be formulated as the correspondence of detected objects represented by points across frames. Overall, point correspondence methods can be divided into two broad categories, namely, deterministic and statistical methods. The deterministic methods use qualitative motion heuristics [4] to constrain the correspondence problem. On the other hand, probabilistic methods explicitly take the object measurement and take uncertainties into account to establish correspondence.

(1) Deterministic Methods for Correspondence.

Deterministic methods for point correspondence define a cost of associating each object in frame $t - 1$ to a single object in frame t using a set of motion constraints. Salari and Sethi [56] handle occlusions, entries, or exits, in a modified greedy approach by first establishing correspondence for the detected points and then extending the tracking of the missing objects by adding a number of hypothetical points. Veenman et al. [4] extend the work of Sethi and Jain [67], and Rangarajan and Shah [68] by introducing the common motion constraint for correspondence. The common motion constraint provides a strong constraint for coherent tracking of points that lie on the same object; however, it is not suitable for points lying on isolated objects moving in different directions.

(2) Statistical Methods for Correspondence.

Measurements obtained from video sensors invariably contain noise. Moreover, the object motions can undergo random perturbations, for instance, in maneuvering vehicles. Statistical correspondence methods solve these tracking problems by taking the measurement and modeling uncertainties during object state estimation. The statistical correspondence methods use the state space approach to model the object properties such

as position, velocity, and acceleration. Measurements usually consist of the object position in the image obtained by a detection mechanism.

For a single object case, a Kalman filter is used to estimate the state of a linear system where the state is assumed to be distributed by a Gaussian. One limitation of the Kalman filter is the assumption that the state variables are normally distributed (Gaussian). Thus, the Kalman filter will give poor estimations of state variables that do not follow Gaussian distribution. This limitation can be overcome by using particle filtering [69]. The weights define the importance of a sample, that is, its observation frequency [61].

When tracking multiple objects using Kalman or particle filters, one needs to deterministically associate the most likely measurement for a particular object to that object's state, that is, the correspondence problem needs to be solved before these filters can be applied. However, if the objects are close to each other, then there is always a chance that the correspondence is incorrect. An incorrectly associated measurement can cause the filter to fail to converge. There exist several statistical data association techniques to tackle this problem. Joint Probability Data Association Filtering (JPDAF) and Multiple Hypothesis Tracking (MHT) are two widely used techniques for data association. JPDAF is used by Chang and Aggarwal [70] to perform 3D structure reconstruction from a video sequence. Rasmussen and Hager [71] use a constrained JPDAF filter to track regions. The major limitation of the JPDAF algorithm is its inability to handle new objects entering the field of view (FOV) or already tracked objects exiting the FOV. Since the JPDAF algorithm performs data association of a fixed number of

objects tracked over two frames, serious errors can arise if there is a change in the number of objects.

MHT is an iterative algorithm where each iteration begins with a set of current track hypotheses. Note that MHT makes associations in a deterministic sense and exhaustively enumerates all possible associations. To reduce the computational load, Streit and Luginbuhl [59] propose a probabilistic MHT (PMHT) in which the associations are considered to be statistically independent random variables and thus there is no requirement for exhaustive enumeration of associations.

2. Kernel Tracking

Kernel tracking is performed by computing the motion of the object, which is represented by a primitive object region, from one frame to the next. We divide these tracking methods into two subcategories based on the appearance representation used, i.e., templates and density-based appearance models, and multi-view appearance models.

(1) Tracking Using Template and Density-Based Appearance Models.

Templates and density-based appearance models have been widely used because of their relative simplicity and low computational cost. We divide the trackers in this category into two subcategories based on whether the objects are tracked individually or jointly.

For tracking single objects, the most common approach in this category is template matching. A limitation of template matching is its high computation cost due to the brute force search. To reduce the computational cost, researchers usually limit the object search to the vicinity of its previous position. Comaniciu et al. [6] use a weighted histogram computed from a circular region to represent the object. Instead of performing

a brute force search for locating the object, they use the mean-shift procedure. An obvious advantage of the mean-shift tracker over the standard template matching is the elimination of a brute force search, and the computation of the translation of the object patch in a small number of iterations. However, mean-shift tracking requires that a portion of the object is inside the circular region upon initialization. Shi and Tomasi [33] proposed the KLT tracker which iteratively computes the translation of a region centered on an interest point.

For tracking multiple objects, modeling objects individually does not take into account the interaction between multiple objects and between objects and background during the course of tracking. Tao et al. [60] propose an object tracking method based on modeling the whole image, as a set of layers.

(2) Tracking Using Multi-view Appearance Models.

The objects may appear different from different views, and if the object view changes dramatically during tracking, the appearance model may no longer be valid, and the object track might be lost. To overcome this problem, different views of the object can be learned offline and used for tracking.

Black and Jepson [16] proposed a subspace-based approach, that is, eigenspace, to compute the affine transformation from the current image of the object to the image reconstructed using eigenvectors. In a similar vein, Avidan [17] used a Support Vector Machine (SVM) classifier for tracking. SVM is a general classification scheme that, given a set of positive and negative training examples, finds the best separating hyperplane between the two classes. One advantage of this approach is that knowledge

about background objects (negative examples that are not to be tracked) is explicitly incorporated in the tracker.

3. Silhouette Tracking

Objects may have complex shapes, for example, hands, head, and shoulders that cannot be well described by simple geometric shapes. Silhouette-based methods provide an accurate shape description for these objects. The goal of a silhouette-based object tracker is to find the object region in each frame by means of an object model generated using the previous frames. This model can be in the form of a color histogram, object edges or the object contour. We divide silhouette trackers into two categories, namely, shape matching and contour tracking. Shape matching approaches search for the object silhouette in the current frame. Contour tracking approaches, on the other hand, evolve an initial contour to its new position in the current frame by either using the state space models or direct minimization of some energy functional.

(1) Shape Matching.

Shape matching can be performed similar to tracking based on template matching, where an object silhouette and its associated model is searched in the current frame. In this approach, the silhouette is assumed to only translate from the current frame to the next, therefore nonrigid object motion is not explicitly handled.

The object model, which is usually in the form of an edge map, is reinitialized to handle appearance changes in every frame after the object is located. This update is required to overcome tracking problems related to viewpoint and lighting condition changes as well as nonrigid object motion. Huttenlocher et al. [64] performed shape matching using an edge-based representation. The authors used the Hausdorff distance to

construct a correlation surface from which the minimum is selected as the new object position. Kang et al. [66] used histograms of color and edges as the object models. In contrast to traditional histograms, they proposed generating histograms from concentric circles with various radii centered on a set of control points on a reference circle. The reference circle is chosen as the smallest circle encapsulating the object silhouette.

In contrast to looking for possible silhouette matches in consecutive frames, tracking silhouettes can be performed by computing the flow vectors for each pixel inside the silhouette such that the flow that is dominant over the entire silhouette is used to generate the silhouette trajectory. Following this observation, Sato and Aggarwal [65] proposed to generate object tracks by applying Hough transform in the velocity space to the object silhouettes in consecutive frames. In contrast to appearance-based matching of silhouettes, a motion-based matching of the object silhouettes is less sensitive to appearance variations, due to different object views

(2) Contour Tracking.

Contour tracking methods, in contrast to shape matching methods, iteratively evolve an initial contour in the previous frame to its new position in the current frame. This contour evolution requires that some part of the object in the current frame overlap with the object region in the previous frame. Tracking by evolving a contour can be performed using two different approaches. The first approach uses state space models to model the contour shape and motion. The second approach directly evolves the contour by minimizing the contour energy using direct minimization techniques such as gradient descent.

For tracking using state space models, the object's state is defined in terms of the shape and the motion parameters of the contour. The state is updated at each time instant such that the contour's a posteriori probability is maximized. The posterior probability depends on the prior state and the current likelihood which is usually defined in terms of the distance of the contour from observed edges. Isard and Blake [61] define the object state in terms of spline shape parameters and affine motion parameters. The measurements consist of image edges computed in the normal direction to the contour. This method represents the contours using explicit representation, for example, parametric spline. Explicit representations do not allow topology changes such as region split or merge [72] based on direct minimization of energy functional. These methods can use implicit representations and allow topology changes.

For tracking by direct minimization of contour energy functional, there is an analogy between the segmentation and the contour tracking methods in this category. Both the segmentation and tracking methods minimize the energy functional either through greedy methods or by gradient descent. The contour energy is defined in terms of temporal information in the form of either the temporal gradient (optical flow) [62], [73], [74], or appearance statistics generated from the object and the background regions [9], [63].

Bertalmio et al. [62] use the optical flow constraint to evolve the contour in consecutive frames. Similarly, Mansouri [73] also uses the optical flow constraint for contour tracking. In contrast to Bertalmio et al. [62] which computes the flow only on the object boundary, Cremers and Schnorr [74] also used the optical flow for contour evolution, and constraint such that an object can only have homogeneous flow vectors

inside the region. An alternative to using the optical flow is to exploit the consistency of the statistics computed inside and outside the object region from one frame to the next. This approach requires initialization of the contour in the current frame with its previous position. In this context, Ronfrad [63] defines the energy functional governing the contour evolution based on the piecewise stationary image models formulated as Ward distances.

2.5 Software Frameworks

In software development, a framework is a defined support structure in which another software project can be organized and developed. A framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project. A framework represents an architecture that models general relationships between domain's entities. It provides a structure and a methodology that extends or uses the domain's applications.

2.5.1 Why use framework?

A framework is a set of classes that embodies an abstract design for solutions to a family of related problems [75]. Framework designers focus on applicability to a certain set of problems, and on flexible best-practices embodied in software. A framework must be able to accommodate functionality in a number of essential areas, providing common behavior while allowing users and developers to customize behavior through configuration parameters and/or framework sub-classing.

Frameworks are designed with the intent of facilitating software development, by allowing designers and programmers to spend more time on meeting software

requirements rather than dealing with the more tedious low level details of providing a working system. There are the following advantages of a framework:

- (1) makes it easier to work with complex technologies.
- (2) ties together a bunch of discrete objects/components into something more useful.
- (3) forces the team to implement code in a way that promotes consistent coding, fewer bugs, and more flexible applications.
- (4) easily test and debug the code, even code that they didn't develop.

2.5.2 How is a framework designing?

The following factors are related to the general framework design:

- (1) Framework reusability: The main reason for building a framework rather than a single text categorization application is to increase reusability of design and implementation. Framework research literature provides guidelines on building application frameworks.
- (2) Modularity: The components' internal implementations should be able to change without affecting the other components.
- (3) Integration: The framework should be able to interface easily with existing categorization solutions, uniting many solutions under a common interface.
- (4) Rapid Application: Development Prototyping new applications should be very quick, with a minimum of custom code in each case. Custom code should generally implement new behaviors rather than new structures within the framework.

- (5) **Rapid Research Cycle:** Researchers should be able to quickly investigate new questions, using the framework as a starting point.
- (6) **Model Flexibility:** The framework structure should be flexible enough to accommodate the needs of many different categorization algorithms that may operate on different representations of the underlying data.
- (7) **Computational Efficiency:** The data sets involved can be quite large, so it is important to have a design and implementation that is efficient in memory, CPU time, and other practical measures such as the time it takes to load a categorizer from disk and generate a hypothesis.
- (8) **Separability:** Pieces of the framework should be usable in isolation for users that only need a feature selection package, a vector categorizer, etc. The most separable pieces of the framework should in many cases be completely separated and available under separate distribution, and used as a software dependency in our framework.

However, there are common complaints that using frameworks adds to "code bloat," and that a preponderance of competing and complementary frameworks means that one trades time spent on rote programming and designs for time spent on learning frameworks. Outside of computer applications, a framework can be considered as the processes and technologies used to solve a complex issue. It is the skeleton upon which various objects are integrated for a given solution.

2.5.3 How to classify frameworks?

Frameworks may be classified into 3 categories: wrappers, architectures, and methodologies.

- (1) Wrappers: A wrapper simplifies an interface to a technology, reduces or eliminates repetitive tasks, increases application flexibility through abstraction, and is often re-usable regardless of high level design considerations.
- (2) Architectures: An architecture manages a collection of discrete objects and implements a set of specific design elements.
- (3) Methodologies: A methodology enforces the adherence to a consistent design approach, decouples object dependencies, and is often re-usable regardless application requirements.

2.5.4 How to use a framework?

A software framework is a reusable design for a software system (or subsystem). This is expressed as a set of abstract classes and the way their instances collaborate for a specific type of software [76], [77]. All software frameworks are object-oriented designs. Although designs don't have to be implemented in an object-oriented language, they usually are. On the one hand, frozen spots define the overall architecture of a software system, that is to say its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework. On the other hand, hot spots represent those parts where the programmers using the framework add their own code to add the functionality specific to their own project.

A good framework in place allows the developers to spend more time concentrating on the business-specific problem at hand rather than on the plumbing code behind it. Also a Framework will limit the choices during development, so it increases productivity, specifically in big and complex systems.

A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It is a reusable design for all or part of a software system; a user interface framework only provides a design for the user interface of a system, while frameworks provide a design for the entire application. Large-scale reuse of object-oriented libraries requires frameworks. The framework provides a context for the components in the library to be reused.

CHAPTER 3

FRAMEWORK DESIGN

Videos are sequences of frames, displayed fast enough so that human eyes can perceive the continuity of its content. It is obvious that all image processing techniques can be applied to individual frames.

3.1 High level design

The framework (Figure 3.1) is based on the analysis of object detection and tracking domain. It consists of five components (sub-frameworks): Change Detection, Object Detection, Object Representation, Object Tracking and Viewer.

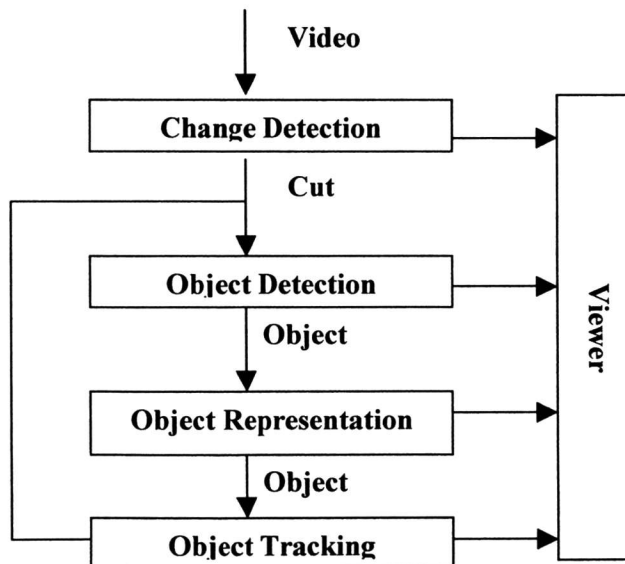


Figure 3.1 Overview of the framework

In this framework, a video is first segmented into semantic shots through change detection and representative frames may be saved into image files. Object detection locates objects of interest in a key frame using suitable object detection algorithms. The detected objects are represented by an object model, object tracking monitors the objects' spatial and temporal changes during a video sequence with an appropriate object tracking algorithm. Object detection and tracking are closely related because tracking starts with object detection, and detecting an object repeatedly in subsequent image sequence is often necessary to help and verify tracking.

The tasks of detecting the object and establishing correspondence between the object instances across frames can either be performed separately or jointly. In the first case, possible object regions in every frame are obtained by means of an object detection algorithm, and the tracker corresponds objects across frames. In the latter case, the object region and correspondence is jointly estimated by iteratively updating object location and region information obtained from previous frames.

In either tracking approach, the objects are represented using the shape and/or appearance models. The model selected to represent object shape limits the type of motion or deformation it can undergo. For example, if an object is represented as a point, then only a translational model can be used. In the case where a geometric shape representation like an ellipse is used for the object, parametric motion models like affine or projective transformations are appropriate. These representations can approximate the motion of rigid objects in the scene. For a non-rigid object, silhouette or contour is the most descriptive representation and both parametric and nonparametric models can be used to specify their motion.

The viewers are chosen to display the processes of object representation, detection and tracking.

3.2 Change detection design

The principal methodology of change detection is to extract one or more features from every frame of a video sequence, to compute the difference of features for consecutive frames, and to compare these differences to a given threshold. Each time the threshold is exceeded, a shot boundary is detected. It has a broadband spectrum of applications including video segmentation, where it forms a central unit of temporal segmentation that explores motion information.

ChgDetector (Figure 3.2) is designed as the root of the change detection hierarchy. It contains all the common attributes such as change lists and methods such as *detect()* for all types of change detections. All the change detection methods may be classified into 5 categories under **ChgDetector**: **PixelChgDetector** (pixel-based detection), **BlockChgDetector** (block-based detection), **FrameChgDetector** (frame-based detection), **ObjectChgDetector** (object-based detection), and **OtherChgDetector** (Hybrid-based detection). Each category is treated as one non-leaf node implemented as an interface/abstract class in the hierarchy tree. Each specific detection method could fall into one category and is implemented as one specific class. **PixelChgDetector** has two subcategories: **PixelBasedPixelChgDetector** (Pixel analysis detection) and **HistoBasedPixelChgDetector** (histogram analysis detection).

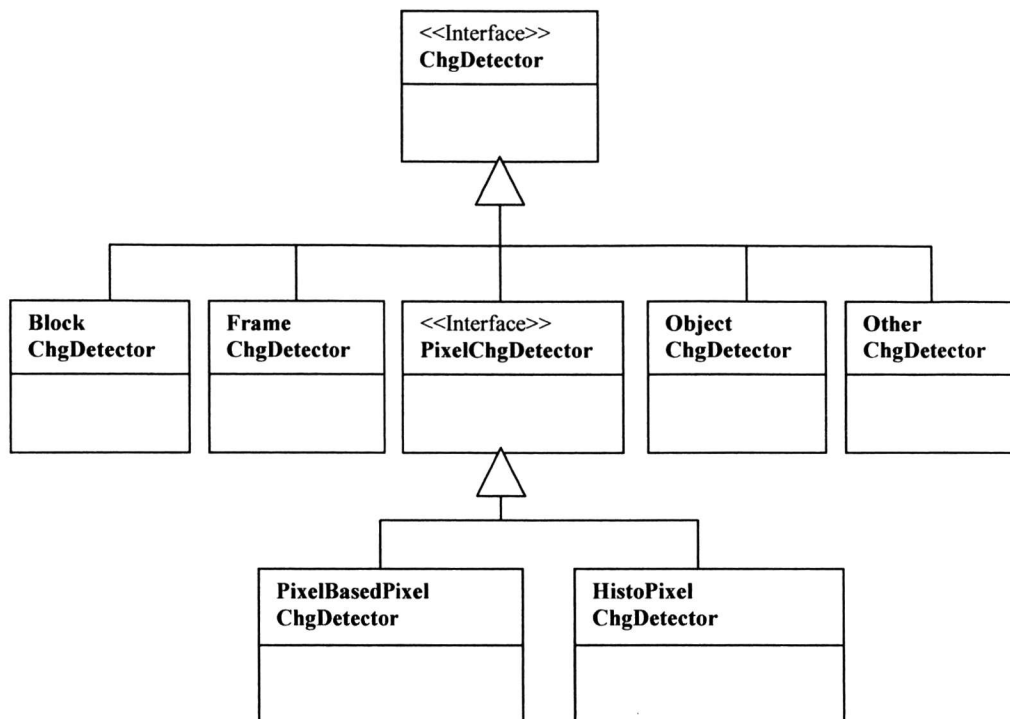


Figure 3.2 class diagram of Change detection

3.3 Object Detection design

Object detection methods (Figure 3.3) may be classified into the following categories: **PointerDetector** (point detection), **SegDetector** (segmentation detection), **BGDetector** (background subtraction detection), **SupDetector** (supervised learning Detection), and **OtherDetector** (other detection). Each detection algorithm should fall into one of these classes. **ObjectDetector** is designed as the root of the object detection hierarchy. It contains all the common attributes and methods such as *initialize()*, and *detect()* for all types of object detections.

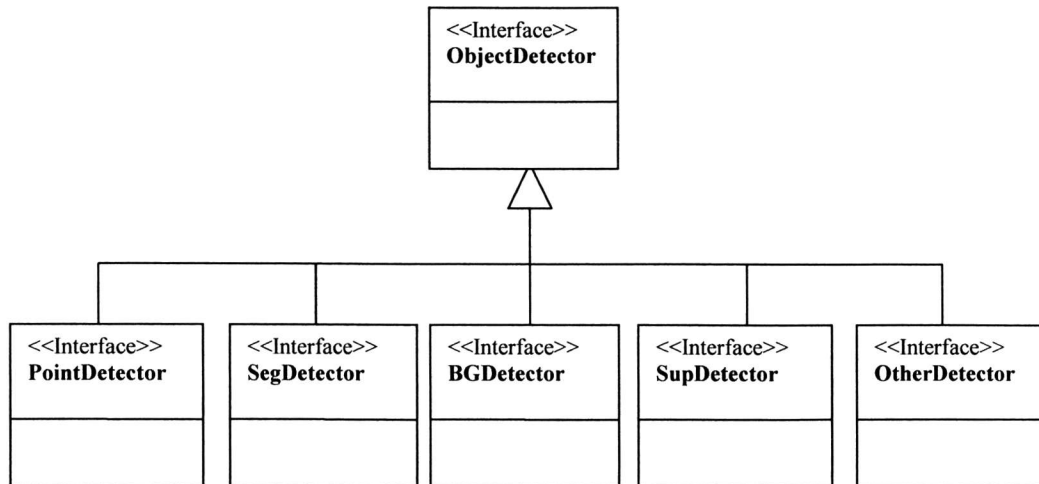


Figure 3.3 Diagram of ObjectDetector Design

3.3.1 Point Detector

PointDetector is a group of detection methods that uses interest points to detect objects. It may include **MoraVecPointDetector** (Mora Vec), **HarrisPointDetector** (Harris point), **SIFTPointDetector** (Scale Invariant Feature Transform), and **AIPointDetector** (Affine Invariant Point).

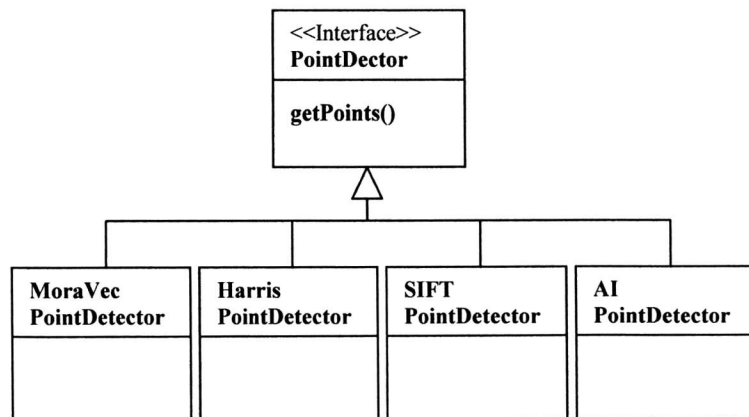


Figure 3.4 Diagram of PointDetector Design

3.3.2 Segmentation Detector

SegDetector (Figure 3.5) is a group of segmentation detection methods. It includes **MeanShiftSegDetector** (Mean-shift), **GraphCutSegDetector** (Graph-cut), and **ActiveContoursSegDetector** (Active contours) as sub classes.

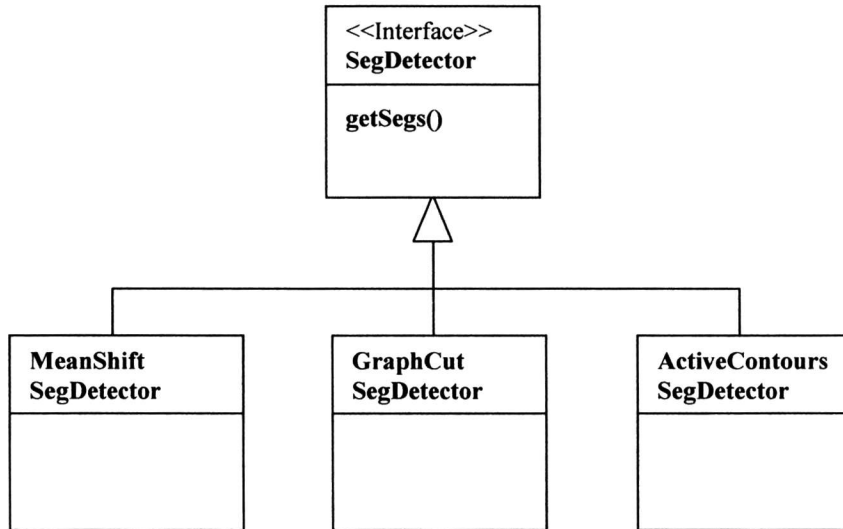


Figure 3.5 Diagram of SegDetector Design

3.3.3 Background Subtraction Detector

BGDetector, a category using background subtraction detection (Figure 3.6) includes **MixGaussiansBGDetector** (Mixture of Gaussians), **EigenBGDetector** (Eigen Background), **AllFlowerBGDetector** (All Flower Background), and **DynTextureBGDetector** (Dynamic Texture Background).

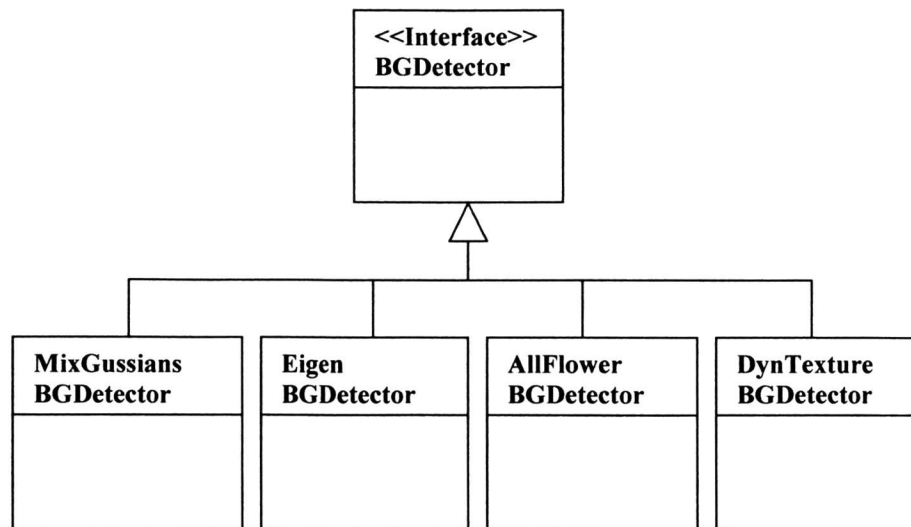


Figure 3.6 Diagram of BGDetector Design

3.3.4 Supervised Learning Detector

SupDetector representing supervised learning detector (Figure 3.7) includes **SVMSupDetector** (Support Vector Machine), **NeuralNetworksSupDetector** (Neural Networks), and **AdaBoostingSupDetector** (Adaptive Boosting) as subclasses.

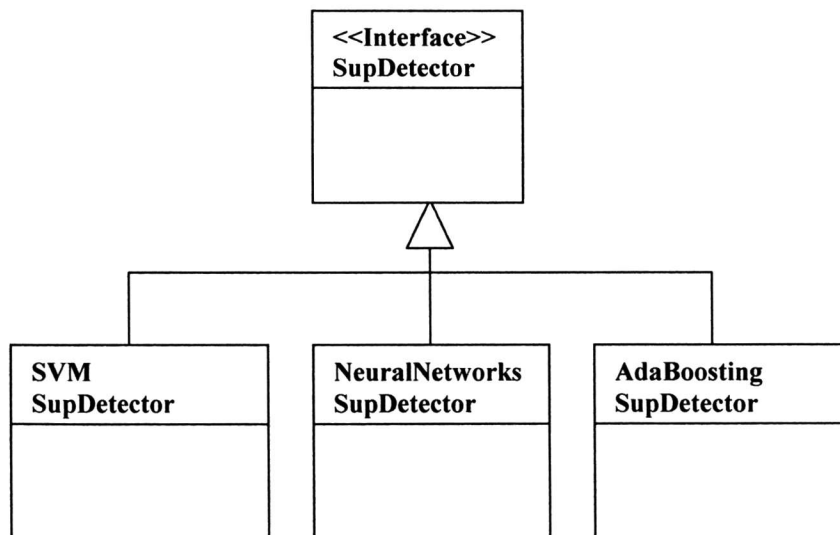


Figure 3.7 Diagram of SupDetector Design

3.4 Object representation design

VideoObject representation shown in Figure 3.8 can be classified into two categories: shape-based **ShapeObject** and appearance-based **AppearanceObject**, which are inherited from **VideoObject** class. **VideoObject** is implemented as a Java abstract class at the top level of object representation. **VideoObject** provides common attributes describing fundamental characteristics of Video and Object and common get/set accessor methods for each attribute.

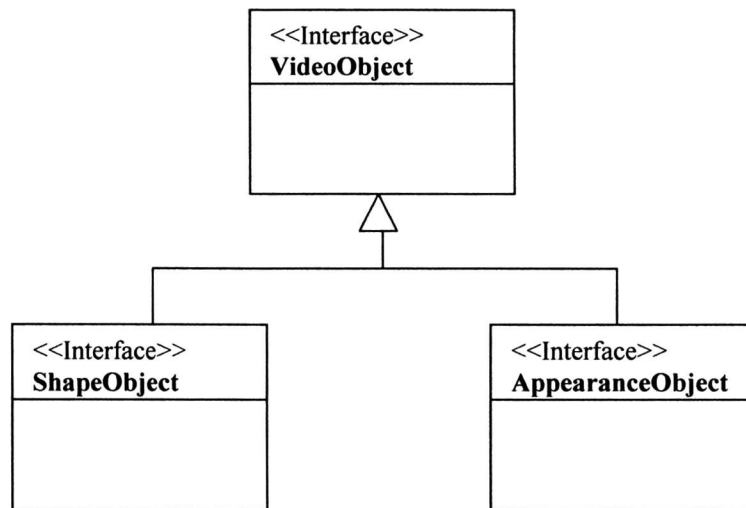


Figure 3.8 Diagram of VideoObject representation design

3.4.1 Shape Object

The **ShapeObject** interface (Figure 3.9) provides definitions for objects that represent some form of shape. **PointShapeObject**, **SilhouetteShapeObject**, **ArticulatedShapeObject** and **SkeletalShapeObject** classes are extended from **ShapeObject**. It is very common that primitive geometric shapes are used to represent objects. **PrimitiveShapeObject** (Primitive Geometrics), which is also inherited from **ShapeObject** interface, may be divided into more detailed shape classes such as **RectanglePrimitiveShapeObject**, **EclipsePrimitiveShapeObject**, and so on.

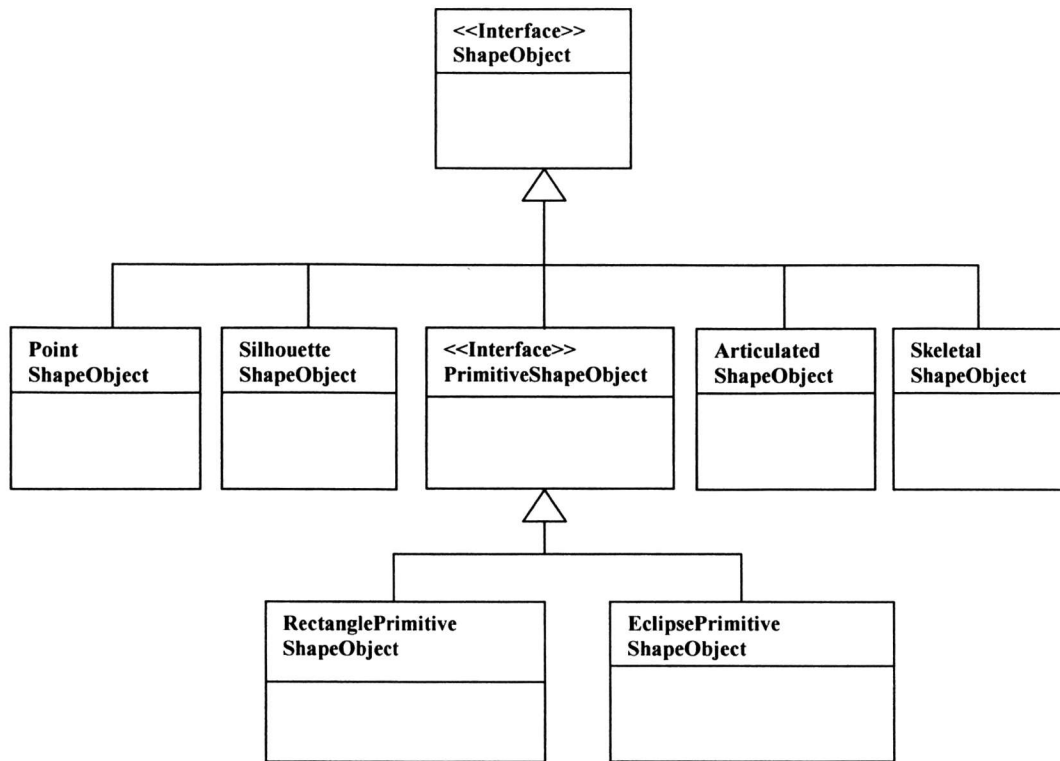


Figure 3.9 Diagram of ShapeObject representation design

3.4.2 Appearance Object

One of the limitations of primitive geometric shapes for object representation is that parts of the objects may be left outside of the defined shape while parts of the background may reside inside it. The phenomena can be observed for both the rigid objects (when the object poses changes) and nonrigid objects (when local motion results in changes in object appearance). In such cases, the object motion estimated by maximizing model similarity may not be correct. To overcome this limitation, one approach is to model the object appearance by probability density functions of color/texture and assign weights to the pixels residing inside the primitive shape based on the conditional probability of observed color/texture.

The **AppearanceObject** interface (Figure 3.10) provides definitions for objects that represent some form of appearance features. **PDFAppearanceObject** (Probability Densities based on object model), **TemplateAppearanceObject** (template appearance base object model), **ActiveAppearanceObject** (active appearance based object model), and **MultiviewAppearanceObject** (multi-view appearance based object model) classes inherited from **AppearanceObject**.

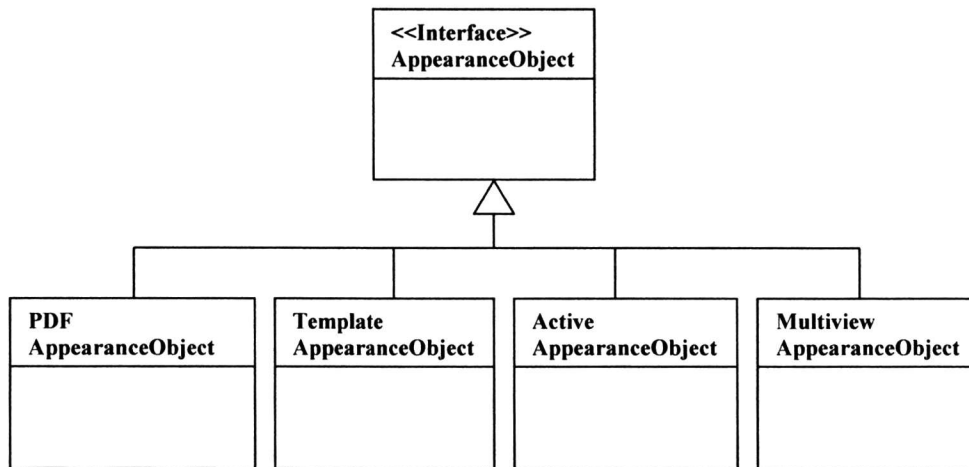


Figure 3.10 diagram of appearance object representation design

3.5 Object tracking design

All tracking methods require object detection at some point. For instance, point trackers require detection in every frame, whereas geometric region or contours-based trackers require detection only when the object first appears in the scene.

Object tracking methods can be classified into 4 categories: point tracking, kernel tracking, silhouette tracking, and other tracking. Each category may be implemented as an interface. Each category may have some subtypes that will also be treated as interfaces. Each specific object tracking method will be implemented as a class and classified into one type or category.

3.5.1 Object Tracker

ObjectTracker (Figure 3.11) is a top level interface that will be extended to 4 kinds of sub level trackers: **PointTracker**, **KernelTracker**, **SilhouetteTracker**, and **OtherTracker**.

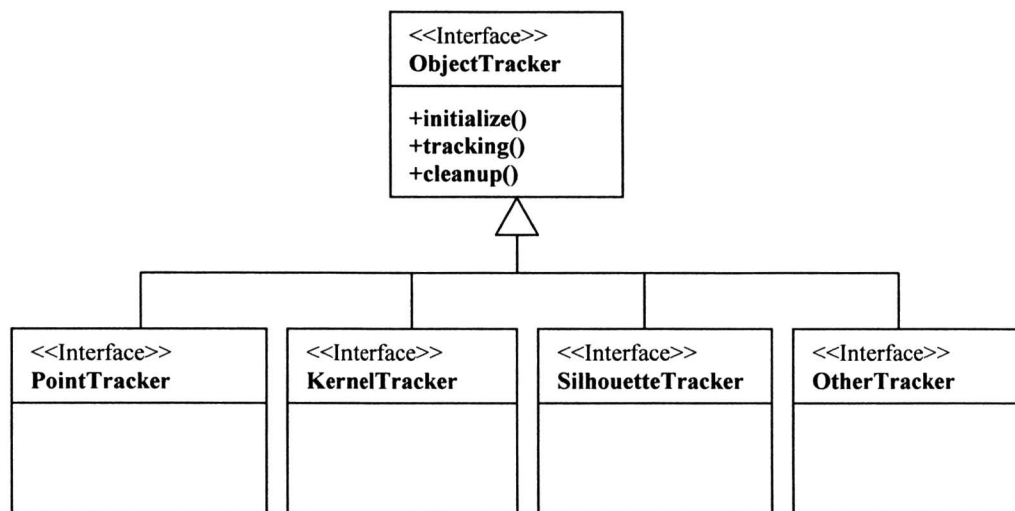


Figure 3.11 Diagram of ObjectTracker Design

3.5.2 Point tracker

Point trackers are suitable for tracking very small objects which can be represented by a single point (single point representation). Multiple points are needed to track larger objects. In the context of tracking objects using multiple points, automatic clustering of points that belong to the same object is an important problem. This is due to the need to distinguish between multiple objects and, between objects and background. Motion-based clustering or segmentation approaches usually assume that the points being tracked lie on rigid bodies in order to simplify the segmentation problem.

Overall, **PointTracker** (point tracker) (Figure 3.12) can be divided into two broad categories: **DetPointTracker** (deterministic point tracker) and **StaPointTracker**

(Statistical point tracker). MCE, GOA and other deterministic methods may be implemented as classes under **DetPointTracker**, whereas **ParticleFilter**, **KalmanFilter**, **JPDAF**, **PMHT** and other statistical methods (as classes) are representatives of **StaPointTracker**.

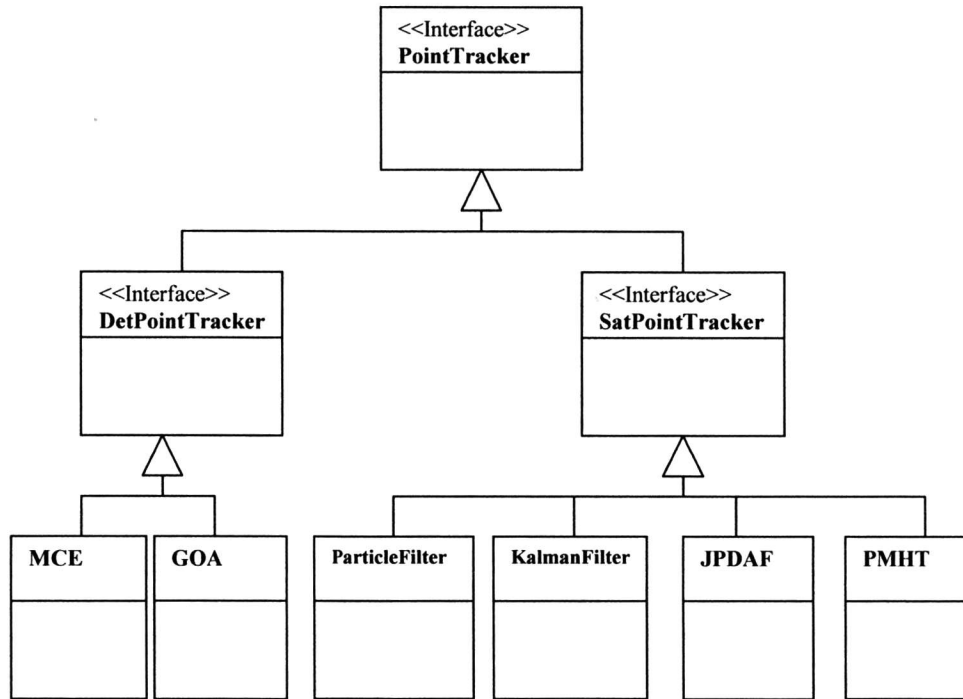


Figure 3.12 Diagram of PointTracker Design

3.5.3 Kernel tracker

The goal of kernel tracking is to estimate the object motion. With the region-based object representation, computed motion implicitly defines the object region as well as the object orientation in the next frame since, for each point of the object in the current frame, its location in the next frame can be determined using the estimated motion model.

As a group of kernel tracking methods, **KernelTracker** (Figure 3.13) that extends **ObjectTracker** has two sub categories of trackers: **MultiViewKernelTracker** and **TemplateKernelTracker**. **SVM**, **EigenTracking** and other multiview based kernel

classes (methods) are examples for **MultiViewKernelTracker**. **MeanShift**, **KLT**, **layering** and other template based kernel classes (methods) are implemented as sub classes under **TemplateKernelTracker**.

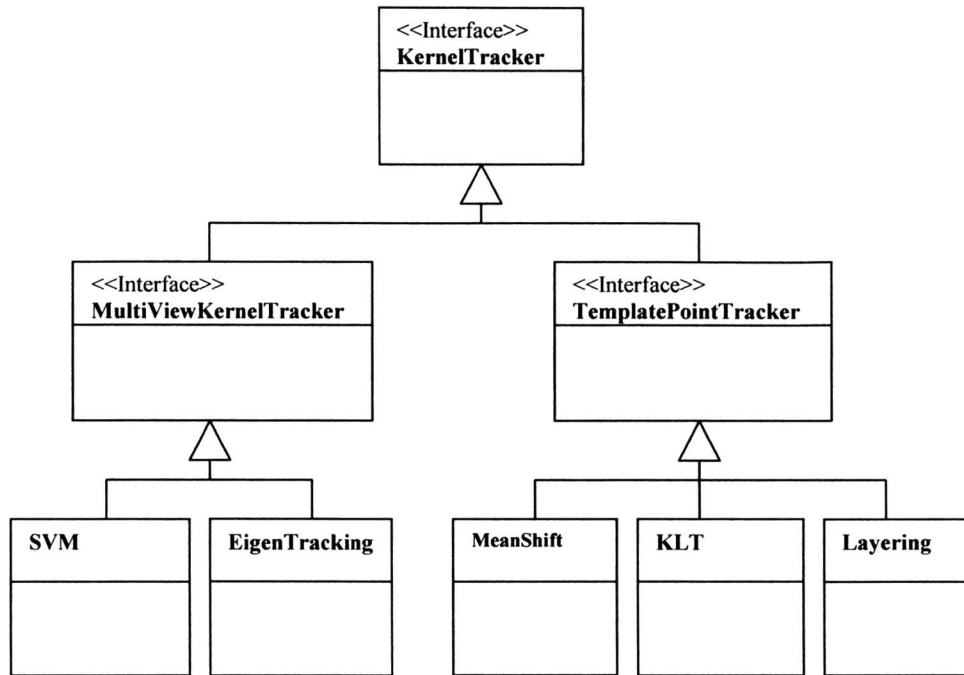


Figure 3.13 Diagram of KernelTracker Design

3.5.4 Silhouette tracker

Silhouette tracking is employed when tracking of the complete region of an object is required. The advantage of tracking silhouettes is their flexibility to handle a large variety of object shapes. Silhouettes may be represented in different ways. The most common silhouette representation is in the form of a binary indicator function, which marks the object region by ones and the nonobject regions by zeros. For contour-based methods, the silhouette is represented either explicitly or implicitly. Explicit representation defines the boundary of the silhouette by a set of control points. Implicit

representation defines the silhouette by means of a function defined on a grid. The most common implicit contour representation is the level sets representation.

SilhouetteTracker as a collection of silhouette tracking methods (Figure 3.14) that extends **ObjectTracker** is divided into two sub categories: **ContourSilhouetteTracker** and **AppearanceMatchSilhouetteTracker**. StateSpace model and Heuristic tracking algorithms using active contour are organized under **ContourSilhouetteTracker**, while Hausdorff, Hough Transform and Histogram tracking methods are implemented as sub classes under **AppearanceMatchSilhouetteTracker** since they use appearance matching for tracking.

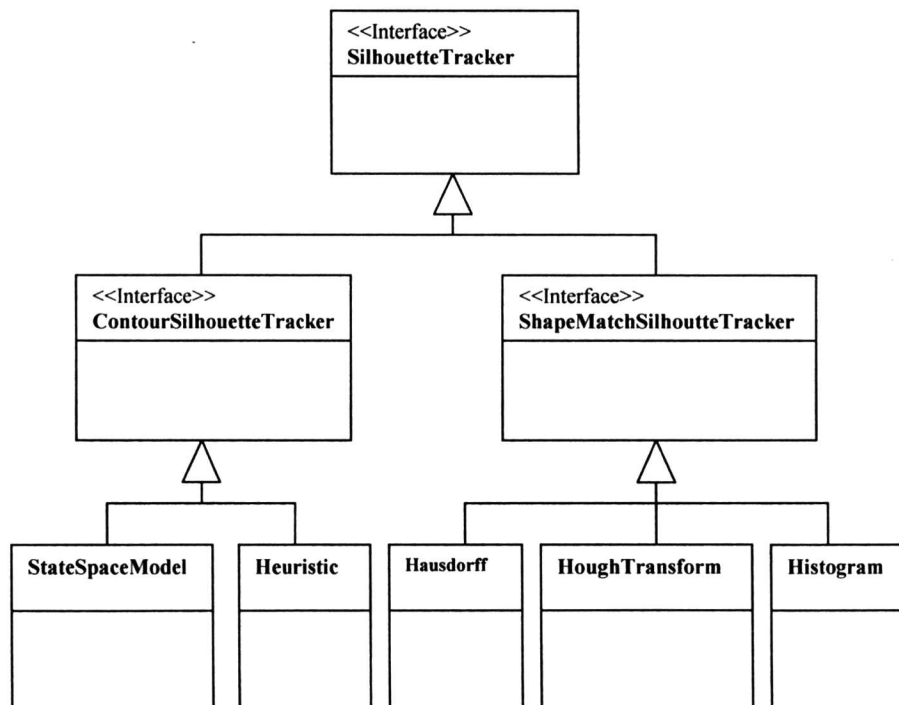


Figure 3.14 Diagram of SilhouetteTracker Design

3.6 Viewer Design

The viewer is designed to display or show videos, objects, change detecting, object detection or object tracking processes and results. The viewer (Figure 3.15) includes 4 sub types of **ObjectViewer**, **ObjDetectionViewer**, **ChgDetectionViewer** and **ObjTrackingViewer**. **ObjectViewer** is for object representations. **ChgDetectionViewer** (Change detection viewer) is designed to view change or fade detection results. **ObjDetectionViewer** and **ObjTrackingViewer** as Object detection and tracking viewers are to display the detection and tracking results.

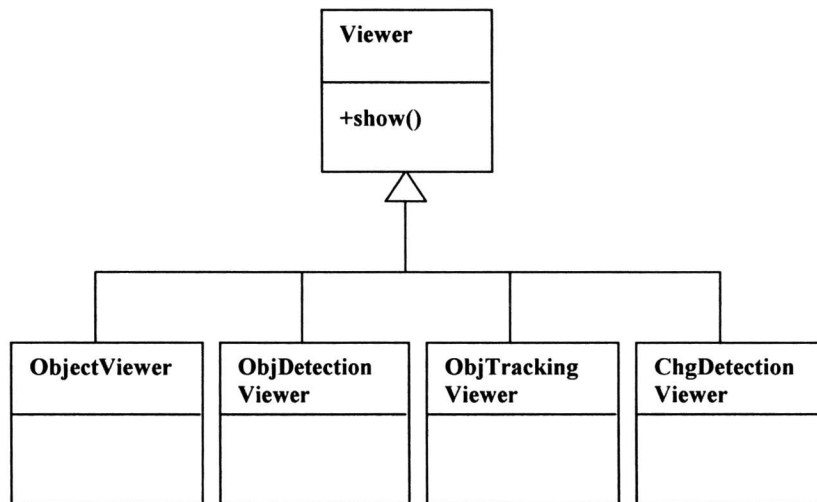


Figure 3.15 Diagram of Viewer Design

CHAPTER 4

IMPLEMENTATION

The framework provides implementation of several different object detection and tracking algorithms as case studies to demonstrate how the system works based on video data with object models. These algorithms are implemented as classes in Java and classified into one category as interface to supply common behavior for specific detection and tracking algorithm classes. A user interface called viewer displays and to visualize how the detection and tracking algorithms work for a given video, and for a given starting and ending path point.

4.1 Why use Java?

A natural choice for programming framework is Java, a true object-oriented language since it has portability across all platforms, and has ability to interact with Internet technologies, permit its fast dissemination and evolution, and simplify memory management.

4.1.1 Interface versus abstract class

An interface is a collection of abstract methods. It is used when certain classes may have one or a few behaviors in common, but are fundamentally different.

An abstract class is one that cannot be instantiated. It is used when several different classes have a lot of behavior in common, and in fact only differ slightly from each other.

Since abstract classes provide default behavior, they are excellent candidates for application frameworks. Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for non-leaf classes in class hierarchies.

4.1.2 Java Media Framework (JMF)

The Java Media Framework (JMF) [78] is a Java Library that enables audio, video and other time-based media to be added to Java applications and applets. This package, which can capture, playback, stream, and transcode multiple media formats, extends the Java Platform, Standard Edition (Java SE) and allows development of cross-platform multimedia applications. JMF handles time-based media, i.e., media which changes with respect to time. The JMF architecture is organized into three stages: input stage, processing stage and output stage.

During the input stage, data is read from a source and passed in buffers to the processing stage. The input stage may consist of reading data from a local capture device (such as a webcam or TV capture card), a file on disk or video stream from the network.

The processing stage consists of a number of codecs and effects designed to modify the data stream to one suitable for output. These codecs may perform functions such as compressing or decompressing the audio to a different format, adding a watermark of some kind, cleaning up noise or applying an effect to the stream (such as echo to the audio).

Once the processing stage has applied its transformations to the stream, it passes the information to the output stage. The output stage may take the stream and pass it to a file on disk, output it to the local video display or transmit it over the network.

4.2 Implementation

The framework API is designed as a set of Java abstract classes. Developers can choose to implement their own "vision" of plug-ins and Framework runtime behavior. "Standard" or default implementations are provided by the framework so developers can start using the framework quickly and easily.

In this thesis, some typical methods such as histogram-based change detection, adaptive boost face detection, and color-based particle filter tracking are implemented as components of framework.

4.2.1 Change detection implementation

Histogram based change detection is an efficient ways to detect changes in videos, and it will be implemented as an example of the change detection component in the framework implementation.

1. Histogram based change detection

The histogram based change detection algorithm counts the number of the pixels changed, and the change is declared if the percentage of the total number of pixels changed exceeds a certain threshold. The change between the two frames can be detected by comparing the differences in intensity values of corresponding pixels in the two

frames. Histogram difference is used as the first metric since it is less sensitive to object motion than other metrics and it can combine many existing metrics.

In this framework, a fade detection algorithm in [79] is implemented in Java. Among the types of shot transitions (cut, fade, dissolve) considered, the cut is an abrupt change from one shot to another and can be seen as the shortest distance between two shots. A better cut detection method is proposed and implemented in Java based on the combination of the methods in [80], [3] and [81].

(1) Cut detection algorithm

As Figure 4.1 shows, this cut detection algorithm starts extracting the frame i and computing the difference Di between the considered frame and the previous $i-d$ frame, which is compared against current threshold Th . If Di is greater than threshold Th , a ratio for detecting flash effects is calculated. When this ratio is greater than threshold $Tflash$, a flash is detected. Otherwise, a change is found. Finally, the current window variance is calculated in order to test whether the data is suitable for updating the value of threshold Th , and in that case a recalculation is needed. Ratio $R = Di/Ds$ where Ds is the difference between the W frames preceding the current frame and the W ones after it.

In cut detection, sequential search is too expensive for practical use. A strategy to compare every other d^{th} frame can speed up a conventional method based on color histograms significantly. This technique can skip unnecessary comparison. If the current comparison indicates that the two frames are in two different shots, we scan backward to look for all the boundaries in the reverse direction of d frames. Once the boundary has been determined, we can scan forward again using the same procedure. Even the regular skip distance = 2 scheme can reduce the cost almost in half.

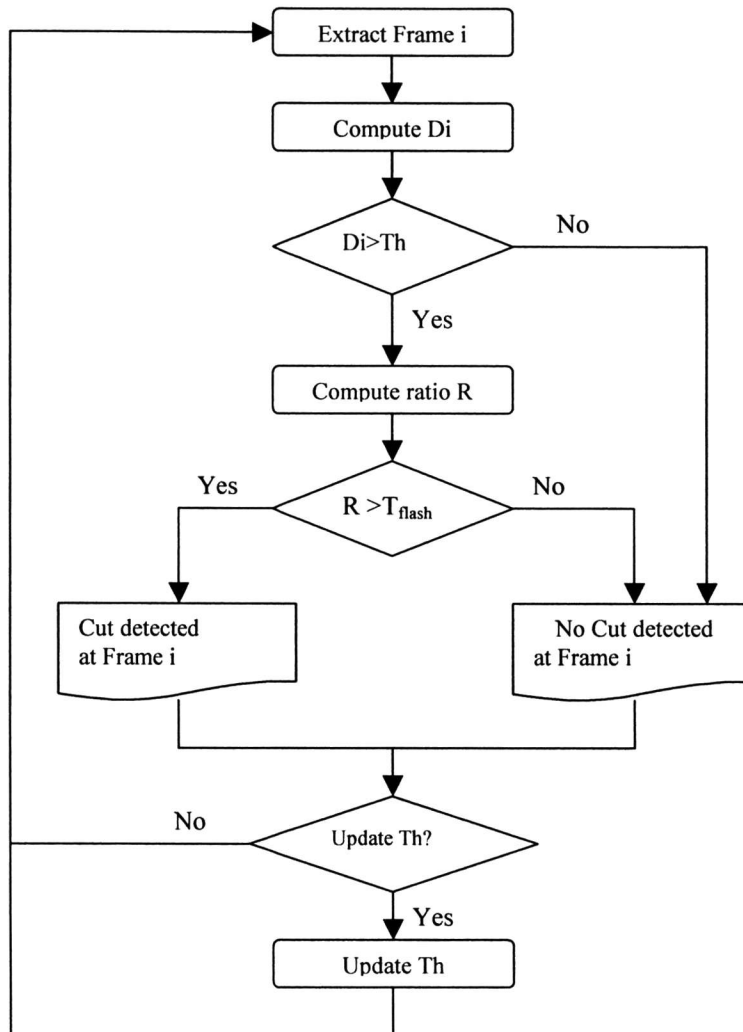


Figure 4.1 Histogram based change detection algorithm

To compute the histogram difference between two frames, there are 3 models (Figure 4.2) for users' options.

$$D_{L1}(i, i+1) = \sum_{j=1}^N |H_i(j) - H_{i+1}(j)|$$

$$D_{L2}(i, i+1) = \sum_{j=1}^N \sqrt{H_i^2(j) - H_{i+1}^2(j)}$$

$$D_{Cos}(i, i+1) = 1 - \frac{\sum_{j=1}^N H_i(j) \cdot H_{i+1}(j)}{\sqrt{\sum_{j=1}^N H_i^2(j)} \cdot \sqrt{\sum_{j=1}^N H_{i+1}^2(j)}}$$

$H_i(j)$ is the histogram value for the gray level j (color bin j) in the frame i ;
 n is total number of gray level(bins);
 $D(i, i+1)$ is a distance measure between Frame i and $i+1$, where many metrics can be used such as: *L1* distance (sum of absolute differences),
L2 distance (Euclid distance) and
Cos (Cosine distance)

Figure 4.2 Measures for comparing histogram

(2) Threshold selection

Threshold selection is the first important problem when comparing changes between two frames. Most of the existing methods use global pre-defined thresholds, or simple local window based adaptive threshold.

a). Global threshold is definitely not efficient since the video property could change dramatically when content changes, and it is often impossible to find a universal optimal threshold across any video segments.

One expression of global threshold: $Th = \mu + \alpha \delta$ where μ is the mean of the frame-to-frame differences. δ is the standard deviation. α should be between 5 and 6 when the histogram comparison metrics is used. Di is the histogram difference between frame i and $i-1$.

$$\sigma = \sqrt{\frac{\sum_{i=1}^N (Di - \mu)^2}{N}} \quad u = \frac{\sum_{i=1}^N Di}{N} \quad (1)$$

b). The local window based adaptive threshold selection method also has its limitation because in some situation the local statistic are ‘polluted’ by strong noises such as big motions and flashlight. An expression of dynamic threshold Th :

$$Th = weight \frac{\sum_{i=j-w}^{i+w} D(i)}{2w+1} \quad (2)$$

where W is the number of difference values taken into account of the left and right local neighbor windows, i is the frame under consideration and weight is a gain factor. Therefore, the threshold is updated for each processed frame.

(3) **HistoPixelChgDetector** class

HistoPixelChgDetector (Figure 4.3) extends from **PixelChgDetector** inherited from **ChgDetector**, so it implements *detect()* as the most important function described above. This class is implemented based on **HistogramEqualizer** that is a class to calculate histogram for one image.

```
public class HistoPixelChgDetector extends PixelChgDetector
{
    private HistogramEqualizer histoEqual;
    public HistoPixelChgDetector()
    {
        histoEqual = new HistogramEqualizer();
        changesList = new LinkedList();
    }
    public Detector() {
        ... ..
    }
    ... ..
}
```

Figure 4.3 Sample codes for HistoPixelChgDetector

4.2.2 Object representation implementation

VideoObject in Figure 4.4 is implemented as a Java abstract class at the top level of object representation. **VideoObject** provides common attributes describing fundamental characteristics of Video and Object and common get/set accessor methods for each attribute.

```

package videoobject;
import java.awt.*;
import java.awt.image.*;
public class VideoObject extends Object
{
    protected String objectType = "VideoObject";
    protected BufferedImage backgroundImage;
    protected Rectangle regionOfInterest;
    protected String videoFileName;
    protected int frameNo;
    public VideoObject(){}
    public void setObjectType(String objType){
        objectType = objType;
    }
    public String getObjectType(){
        return objectType;
    }
    public void setBackgroundImage(BufferedImage bgImg){
        backgroundImage = bgImg;
    }
    public BufferedImage getBackgroundImage(){
        return backgroundImage;
    }
    public void setRegionOfInterest(Rectangle regOfInterest){
        regionOfInterest = regOfInterest;
    }
    public Rectangle getRegionOfInterest(){
        return regionOfInterest;
    }
    public void setVideoFileName(String vFileName){
        videoFileName = vFileName;
    }
    public String getVideoFileName(){
        return videoFileName;
    }
    public void setFrameNo(int fNo){
        frameNo = fNo;
    }
    public int getFrameNo(){
        return frameNo;
    }
    public String toString(){
        return objectType;
    }
};
} //end of class

```

Figure 4.4 Sample Codes for VideoObject

1. ShapeObject

ShapeObject (Figure 4.5) is a subclass of **VideoObject** and provides common attributes and methods for the shaped objects but **AppearanceObject**, the other subclass of **VideoObject**, contains more appearance features for non-shaped objects.

```
import java.awt.*;
public abstract class ShapeObject extends VideoObject
{
    protected Rectangle bounds;
    public void setBounds(Rectangle bounds){
        this.bounds = bounds;
    }
    public Rectangle getBounds(){
        return bounds;
    }
    public String toString(){
        objectType = "ShapeObject";
        return objectType;
    };
    .....
}
```

Figure 4.5 Sample codes for ShapeObject

2. PrimitiveShapeObject and RectanglePrimitiveShapeObject

PrimitiveShapeObject is shown in Figure 4.6, **SilhouetteShapeObject**, **ArticulatedShapeObject**, and **SkeletalShapeObject** are sub classes extended from **ShapeObject**. **PrimitiveShapeObject** contains common attributes and methods for all types of Primitive Geometrics such as Rectangle (**RectanglePrimitiveShapeObject**) and Eclipse.

```

import java.awt.*;

public abstract class PrimitiveShapeObject extends ShapeObject
{
    protected int model;
    public void setModel(int model){
        this.model = model;
    }
    public int getModel(){
        return model;
    }
    public String toString(){
        return "PrimitiveShapeObject";
    };
    .....
}

```

Figure 4.6 Sample codes for PrimitiveShapeObject

We take **RectanglePrimitiveShapeObject** (Figure 4.7) as one example of object representation implementation.

```

public class RectanglePrimitiveShapeObject extends PrimitiveShapeObject
{
    protected Rectangle x;
    public RectanglePrimitiveShapeObject(int model){
        objectType ="RectanglePrimitiveShapeObject";
    }
    public RectanglePrimitiveShapeObject(){
        objectType ="RectanglePrimitiveShapeObject";
    }
    public void setRect(Rectangle x){
        this.x = x;
    }
    public Rectangle getRect(){
        return x;
    }
    public String toString(){
        return "RectanglePrimitiveShapeObject";
    };
    .....
}

```

Figure 4.7 Sample codes for RectanglePrimitiveShapeObject

4.2.3 Object Detector implementation

ObjectDetector (Figure 4.8) is implemented as a Java abstract class at the top level of object detection. **ObjectDetector** provides common attributes such as `detectedObjects` describing fundamental characteristics of Object detection and common methods such as *initialize()*, *detect()*, *analyze()* and *cleanup()* and get/set accessor methods for each attribute.

```
public abstract class ObjectDetector
{
    protected String detectedType = "ObjectDetector";
    protected LinkedList detectedObjects;
    public void setDetectedObject(LinkedList detectedObjs){
        detectedObjects= detectedObjs;
    }
    public LinkedList getDetectedObjects(){
        return detectedObjects;
    }
    public boolean initialize(){
        return true;
    }
    public boolean detect(){
        return true;
    }
    public boolean analyze(){
        return true;
    }
    public boolean cleanup(){
        return true;
    }
    public String getDetectedType(){
        return detectedType;
    }
    ... ..
}
```

Figure 4.8 Sample codes for ObjectDetector

Point detection, Segmentation detection, Background Subtraction detection, Supervised Learning Detection are implemented as **PointDetector**, **SegDetector**, **BGDetector**, and **SupDetector** abstract classes extended from **ObjectDetector**. They all contain common attributes and methods for all types of sub types of object detection.

We take **AdaBoostingSupDetector** inherited from **SupDetector** as one example of our framework implementation

1. **AdaBoostingSupDetector** (Face detection)

Object detection, and in particular, face detection is an important element of various computer vision areas, such as image retrieval, shot detection, video surveillance, etc. The goal is to find an object of a pre-defined class in a static image or video frame. Sometimes this task can be accomplished by extracting certain image features, such as edges, color regions, textures, contours, etc. and then using some heuristics to find configurations and/or combinations of those features specific to the object of interest.

We would like to take face detection (**AdaBoostingSupDetector**) as one examples of our framework implementation.

(1) Rapid object detection system

Viola and Jones [82] introduce a rapid object detection system. This system introduces integral image which allows fast computation of features, uses AdaBoost [83] to train efficient classifiers, and introduces a cascaded structure which can reject non-face images quickly. The system can achieve high detection rate with small number of false positives.

(2) Adaptive Boost algorithm for face detection

AdaBoost is an algorithm for constructing strong classifier as a linear combination of “weak” classifiers. To extend Viola’s research, Lienhart et al. [84] introduce a novel feature set which is designed for detecting in-plane rotation faces, present analyses among the different boosting algorithms (Discrete, Real and Gentle

AdaBoost), and compare the performance between stumps and Regression Tree (CART) and also analyze the effect of sizes of training data.

The face detection component is a cascaded structure. The structure is cascaded by strong classifiers. Each strong classifier consists of several weak classifiers. And a weak classifier consists of a weight and a feature with thresholds.

The structure (Figure 4.9) is cascaded by strong classifiers (circles “1, 2, 3 ...”). In the cascaded structure, each strong classifier rejects numbers of non-face images. This is efficient for detecting large numbers of sub-images. In the early stages, most of the non-face images are rejected; only few numbers of non-faces (hard samples) are needed to be processed in the late stages.

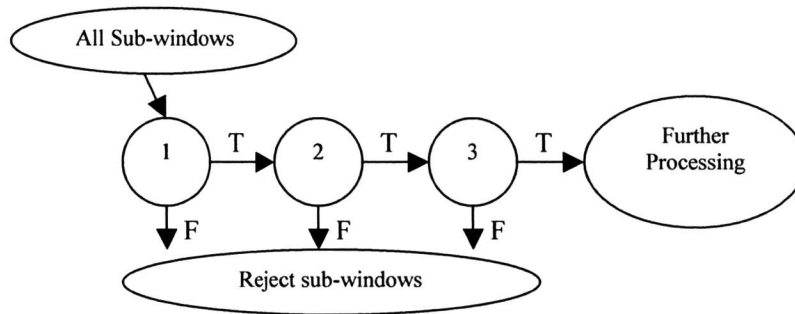


Figure 4.9: Cascaded structure

The AdaBoost algorithm shown in Figure 4.10 takes as input a training set $(x_1, y_1), \dots, (x_m, y_m)$ where each x_i belongs to some instance space X , and each label y_i is in some label set $Y = \{-1, 1\}$. Boosting calls given weak classifiers in a series of rounds $t = 1, \dots, T$. The weight distribution on training example i on round t is denoted by $W_t(i)$. Initially, all weights are set equally, but on each round, the weights W_t are updated according to the classification results of h_t . The weights of examples misclassified by h_t are increased while those of the correctly classified examples are decreased. The strong

classifier is a weighted majority vote of the T weak classifiers where α_t is the weight assigned to h_t .

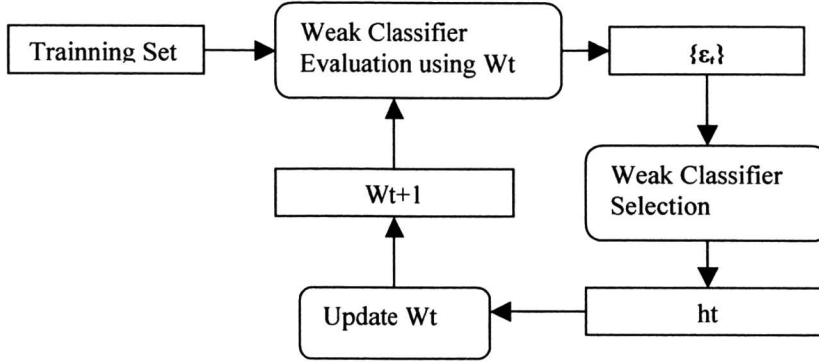


Figure 4.10 Block diagram of Adaboost algorithm

The following (Figure 4.11) is the pseudocode for the AdaBoost algorithm .

1. Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = \{-1, 1\}$ for negative and positive examples respectively.
2. Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2n}$, where m and n are the number of positive samples and the number of negative samples respectively.
3. For $t = 1, \dots, T$ (maximum number of weak classifier):
 - a. Normalize the weights, $w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$, so that w_t can represent the probability distribution.
 - b. For each feature, j , train a classifier h_j which is restricted to use a single feature. The error is evaluated with respect to w_t , $\epsilon_i = \sum_i w_i (h_j(x_i) \neq y_i)$.
 - c. Choose the classifier h_t , which has a lowest error ϵ_t .
 - d. Update the weights of samples: $w_{t+1,i} = w_{t,i} (\beta^{e_{t,i}})$,
 where $e_{t,i} = \begin{cases} 1, & h_t(x_i) \neq y_i \\ 0, & h_t(x_i) = y_i \end{cases}$ and $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$.
4. The final strong classifier is:

$$h(x) = \begin{cases} 1, & \sum_{t=1}^T \alpha_t h_t(x) \geq \sum_{t=1}^T \alpha_t \\ 0, & \text{otherwise} \end{cases}$$
 where $\alpha_t = \frac{1}{2} \log \frac{1}{\beta_t}$.

Figure 4.11 Procedure of Discrete AdaBoost

(3) AdaBoostSupDetector class

This method in Figure uses simple Haar-like features (so called because they are computed similar to the coefficients in Haar wavelet transforms) and a cascade of boosted tree classifiers as a statistical model.

The Open Source Computer Vision Library (OpenCV) [85] is a free, open source collection of computer vision routines geared mainly towards human-computer interaction, robotics, security, and other vision applications where the lighting and context of use cannot be controlled. OpenCV is not geared towards factory machine vision where the environmental conditions can be controlled and one generally knows what one is looking for, although there is a large overlap.

OpenCV provides low-level and high-level APIs for face/object detection. A low-level API allows users to check an individual location within the image by using the classifier cascade to find whether it contains a face or not. Helper functions calculate integral images and scale the cascade to a different face size (by scaling the coordinates of all rectangles of Haar-like features).

Jni2opencv, a Java wrapper of the face detector of OpenCV, is used to implement Ada Boost methods in **AdaBoostSupDetector** class (Figure 4.12).

```

import java.awt.*;
import java.awt.image.*;
import java.awt.geom.*;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import flanagan.analysis.Regression;

public class AdaBoostSupDetector extends SupDetector {
    String imageFile = "temp.jpg"; // "lena.jpg"
    String []detectedImage;
    Picture pic;
    BufferedImage image; // the rasterized image
    int frameNo;
    String videoFileName;
    String cascade;
    private JNIOpenCV myJNIOpenCV;

    private JFrame f; // on-screen view
    Graphics2D g2d;

    public AdaBoostSupDetector() {
        myJNIOpenCV = new JNIOpenCV();
        cascade = "haarcascade_frontalface_alt.xml";
    }
    public AdaBoostSupDetector(String vFileName,int fNo)
    {
        videoFileName = vFileName;
        frameNo = fNo;
        myJNIOpenCV = new JNIOpenCV();
        cascade = "haarcascade_frontalface_alt.xml";
    }
    public void setVideoFileName(String vFileName)
    {
        videoFileName = vFileName;
    }
    public String getVideoFileName(){
        return videoFileName;
    }
    public void setFrameNo(int fNo){
        frameNo = fNo;
    }
    public int getFrameNo(){
        return frameNo;
    }
    public VideoObject[] getDetectedObjects(){
        return detectedObjects;
    }
    private void setDetectedObjects(String imageFile,int[] detectedFaces){
        ... ..
    }
}

```

```

// view on-screen, creating new frame if necessary
public void show() {
    ... ..
}

boolean detect() {

    System.out.println("start to run detect()...");

    int[] detectedFaces = myJNIOpenCV.detectFace(40, 40, cascade, imageFile);
    System.out.println("The size of detectedFaces is "+detectedFaces.length);
    if(detectedFaces.length==0) return false;
    int numFaces = detectedFaces.length / 4;
    String []detected = new String[numFaces];
    setDetectedObjects(imageFile, detectedFaces);
    ... ..
    return true;
}

... ..
}
class JNIOpenCV {
    static {
        System.loadLibrary("JNI2OpenCV");
    }
    public native int[] detectFace(int minFaceWidth, int minFaceHeight, String cascade, String
filename);
}

```

Figure 4.12 Sample Codes for AdaBoostSupDector

4.2.4 Object Tracker implementation

ObjectTracker in Figure 4.13 is implemented as a Java abstract class at the top level of object tracking. **ObjectTracker** provides common attributes describing fundamental characteristics of object tracking and common get/set accessor methods such as *initialize()*, *tracking()*, *analyze()* and *cleanup()*.

```

public abstract class ObjectTracker
{
    protected String trackingType = "ObjectTracker";
    public boolean initialize(){
        return true;
    }
    public boolean tracking(){
        return true;
    }
    public boolean analyze(){
        return true;
    }
    public boolean cleanup(){
        return true;
    }

    public String getTrackingType(){
        return trackingType;
    }
    ... ..
}

```

Figure 4.13 Sample codes for ObjectTracker

Point tracking, kernel tracking, silhouette tracking are implemented as **PointTracker**, **KernelTracker**, and **SilhouetteTracker** abstract classes extended from **ObjectTracker**. They all contain common attributes and methods for all types of sub types of object tracking.

We take **ParticleFilterSatPointTracker** inherited from **SatPointTracker** extending **PointTracker** as one example of our framework implementation.

1. ParticleFilterSatPointTracker (Particle filter)

Conceptually, a particle filtering algorithm maintains a probability distribution over the state of the system it is monitoring the state of the object being tracked. In most cases, non-linearity and non-Gaussianity in the object's motion and likelihood models yields an intractable filtering distribution.

(1) Particle filter

Particle filtering overcomes this intractability by representing the distribution as a set of weighted samples, or particles. Each particle represents a possible instantiation of the state of the system. In other words, each particle describes 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. We can thus determine the most probable state of the object by finding the location in the particle filtering distribution with the highest weight.

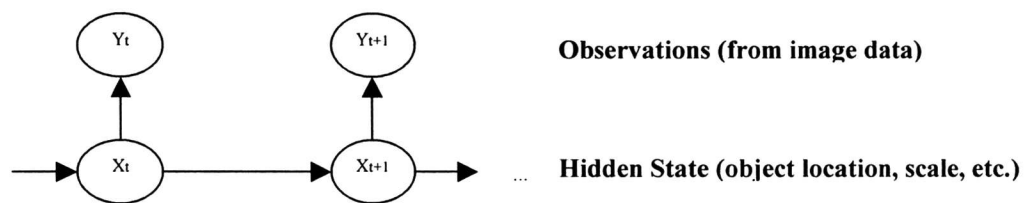
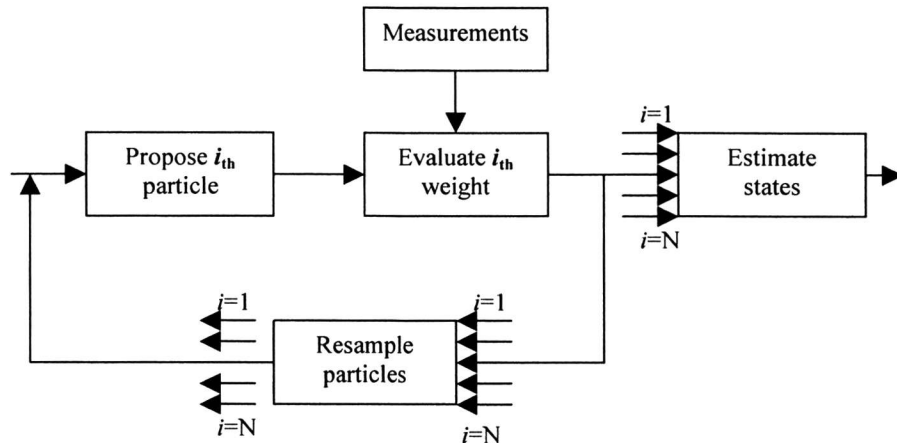


Figure 4.14: A Dynamic Filtering System

Filtering (Figure 4.14) is the problem of sequentially estimating the states (parameters or hidden variables) of a system as a set of observations.

(2) Particle filter algorithm

**Figure 4.15 Block diagram for particle filter algorithm**

Block diagram (Figure 4.15) shows the major computational steps in a sampling importance resampling (SIR) particle filter algorithm. The particles are initialized randomly. At each time step, the algorithm uses transition model to predict the next new state for each particle and use observation model to assign a weight to each particle, and then creates a new set of equally weighted particles by sampling the distribution of the weighted particles produced in the previous step.

The following is the pseudocode for the particle filter algorithm (Figure 4.16):

1. Initialization: For $i = 1$ to N , initialize the particles. Let $t = 0$.
2. For $i = 1$ to N , evaluate the importance weights according to the likelihood in Transition Model and normalize weights
3. PF measurement update: Resample N particles with replacement according to Resampling
4. PF time update according to Observation model
5. Set $t = t + 1$ and continue from step 2.

Figure 4.16 Particle filter algorithm

Particle filter algorithm performs forward inference using the Bayesian filtering distribution

$$\underbrace{P(x_t | y_{1:t})}_{\text{Current Object State}} = \underbrace{\sigma}_{\text{Observation Model}} \underbrace{P(y_t | x_t)}_{\text{Transition Model}} \int \underbrace{x_{t-1} P(x_t | x_{t-1})}_{\text{Previous Object State}} P(x_{t-1} | y_{1:t-1}) dx_{t-1} \quad (3)$$

Transition Model $P(x_t | x_{t-1})$: Specifies how objects move between frames. We use a second-order, auto-regressive dynamical model:

$$X_t - E(X) = A_2(X_{t-2} - E(X)) + A_1(X_{t-1} - E(X)) + B_0 W_t \quad (4)$$

This predicts the next state based on the previous two plus some noise (W_t). $E(X)$ is mean of X .

Observation Model $P(y_t | x_t)$: Specifies the likelihood of an object being in a specific state (i.e. at a specific location). Likelihood is based on a distance metric D between histograms h_0 and $h(x_t)$:

$$p(y_t | x_t) \propto e^{-\lambda D^2[h_0, h(x_t)]} \quad (5)$$

Resampling: To prevent degeneracy of weights, before time step $t+1$, particle set is re-sampled according to particle weights (Produces a new set of unweighted particles).

(3) ParticleFilterSatPointTracker class

In **ParticleFilterSatPointTracker** (Figure 4.17) class, we need to implement *initialize()*, and *tracking()* inherited from its parents' classes. For Particle Filter, there are many particular methods such as Observation for this particle filter that need to be implemented as well.

```

....
public class ParticleFilterSatPointTracker extends SatPointTracker implements Runnable
{
    protected int particleNumber;
    protected int objectNumber;
    protected Particle[] particles;
    protected VideoObject[] videoObjects;
    protected String videoFileName;
    public static final int INVALID_INDEX = -1;
    protected static final int PARTICLE_MAX = 500;
    protected static final int PARTICLE_MIN = 50;
    protected static final int OBJECT_NUM = 1;
    private static final int FORMAT_SIZE = 512;
    private int IMAGE_WIDTH = 512;//, 512;
    private int IMAGE_HEIGHT = 512;//358;
    static final int NH = 10;
    static final int NS = 10;
    static final int NV = 10;
    protected boolean isGaussian = true;
    protected int displayMode = 0;
    protected int beginFrame=0;
    protected int endFrame;
    protected int totalFrames;
    private int current =0;

    private boolean isDone;
    private static final int FRAME_SIZE = 20;
    Observation observation;
    protected LinkedList trackedImages;
    Particle[] resample(Particle[] particles)
    {
        .....
    }
    boolean initialize(BufferedImage bkImg)
    {
        .....
    }
    private Particle transition( Particle p,int w,int h)
    {
        .....
    }
    public boolean tracking()
    {
        .....
    }
    .....
}

```

Figure 4.17 Sample codes for ParticleFilterSatPointTracker

4.2.5 Viewer implementation

Viewer in Figure 4.18 is a top Java class in the viewer hierarchy. It extends from `JFrame` class and contains all the common attributes and methods for all types of viewers. Each type of viewer is implemented as a Java class to view or test detection or tracking results.

```
class Viewer extends javax.swing.JFrame implements Runnable, ActionListener,
TreeSelectionListener{

    private File videoFile; // The video selected in the selectFileDialog
    private MediaPlayer mediaPlayer = new javax.media.bean.playerbean.MediaPlayer();

    // used for waiting until the player has started
    private Object waitSync = new Object();
    private boolean stateTransitionOK = true;

    private FramePositioningControl fpc;
    private int totalFrames = FramePositioningControl.FRAME_UNKNOWN;

    DefaultMutableTreeNode root;
    private JTree cutTree;
    private javax.swing.JButton loadFileButton;

    private javax.swing.JFrame selectFileDialog;
    private javax.swing.JFileChooser selector;

    Container contentPane = getContentPane();
    private java.awt.Container resultContainer = new Container();
    private java.awt.Container leftContainer = new Container();
    private java.awt.Container rightContainer = new Container();
    ... ..
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Viewer().creatAndShowGUI();
            }
        });
    }

    public void run() {
        creatAndShowGUI();
    }
    public void creatAndShowGUI() {
        /** Init the components in the frame */
    }
    ... ..
}
```

Figure 4.18 Sample code for Viewer

CHAPTER 5

EXPERIMENTS AND RESULTS

Testing is a major consideration in the development and maintenance of a framework. We tested our framework on Pentium-4 1.6GHz and 512MB memory PC. In order to evaluate the performance and scalability of our approach, the test clips in our experiment are extracted from different videos recorded from digital camera and live broadcast television. The videos are compressed in AVI and MPEG-2 standard with the frame rate of 25-30fps.

5.1 Change (Cut) detection experimental results

Figure 5.1 (a) shows some experimental results of the changes detection procedure without skipping any frame for a10strafe.avi. This video contains 498 frames captured by camera. The key frame is in the fourth detected segments. In the right part of screen, result tree shows detected segments with the range of frame numbers. Figure 5.1 (b) shows comparison between the frame after changing and the frame before changing for one of changes after change detection procedure for a10strafe.avi. The two images show the previous frame and current frame when one cut happened.

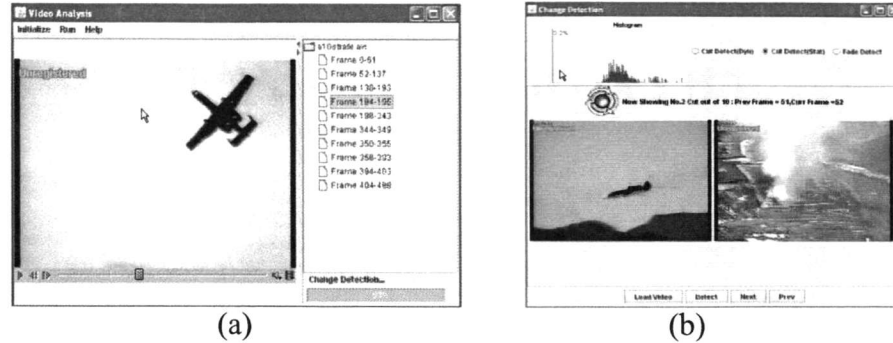


Figure 5.1 shows change detection results for one video of flights

The change detection with a skipping distance of 30 is tested on another video clip from TV that contains a large number of over 2300 frames. Figure 5.2 (a) shows the segment result tree and the key frame for the selected segment. Figure 5.2 (b) shows a change between Frame 1577 and 1578 in key Segment 11.

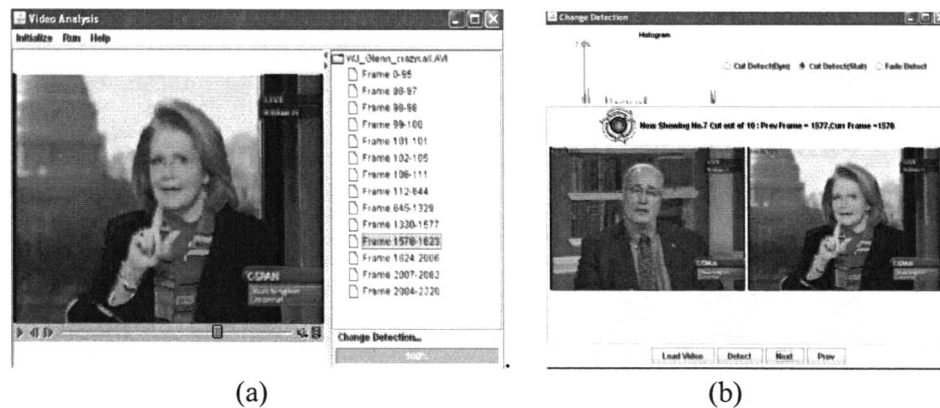


Figure 5.2 Change detection results for one news video

The change detection with a skipping distance of 3 is tested on a video taken by a camera in the street. This video has over 300 frames. Figure 5.3 (a) shows segment result tree of 4 segments detected, and the key frame of Segment 3. Figure 5.3 (b) shows a cut between Frame 73 and Frame 74 in Segment 3.

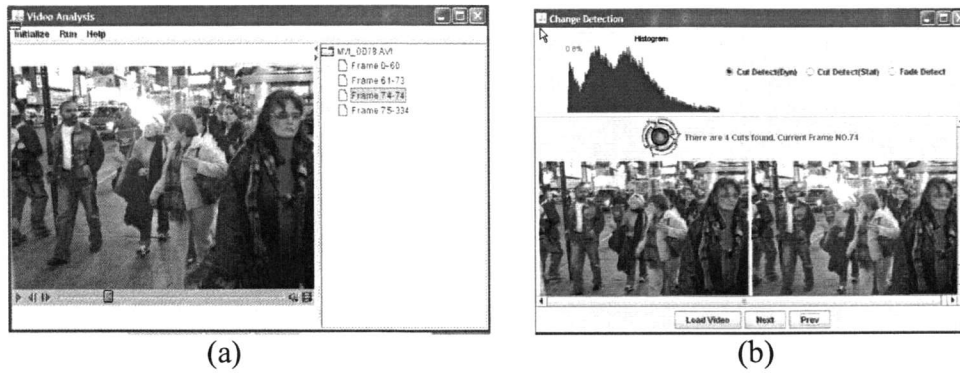


Figure 5.3 shows change detection results for one video taken by one camera

Figure 5.4(a) shows a fade in results detected of Frame 65 -75 . Figure 5.4 (b), (c) show a fade out results of Frame 271-311.

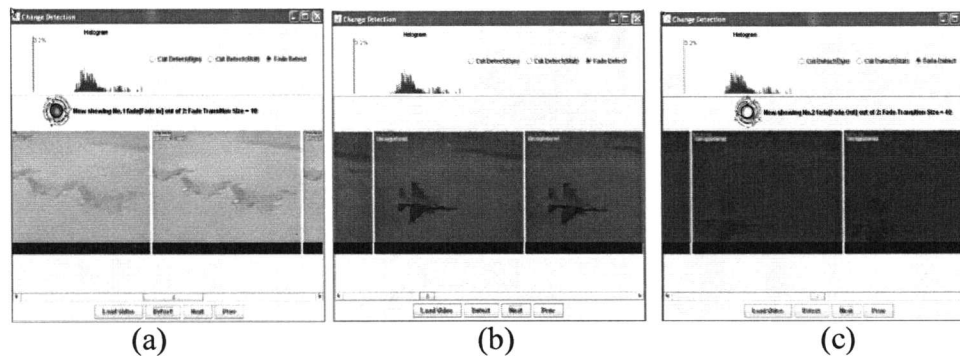


Figure 5.4 show fade detection results for one video of flights

5.2 Object (Face) detection experimental results

Figure 5.5 shows the face detection results. Figure 5.5(a) are face detection results on Frame 111 of MVI_0006.avi created by one family camera. In the image, detected faces are marked with red rectangle. There are two good results: Sub 1 and Sub2 detected out of 4 detected faces. Another testing case is shown in Figure 5.5(b). There are two good results detected on Frame 120 of MVI_0078.avi: Sub 0 and Sub1 detected out of 2 detected objects.

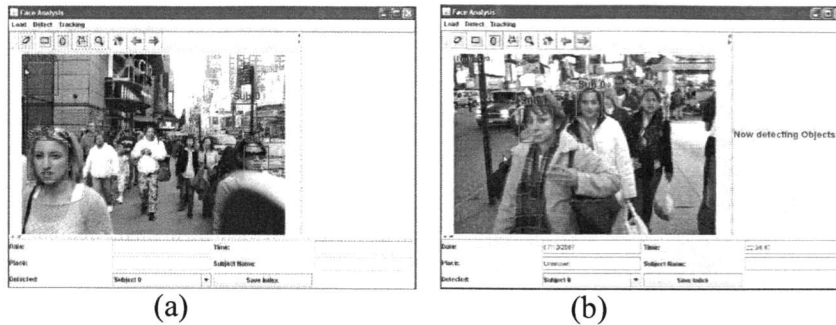


Figure 5.5 shows Face detection results

5.3 Object (Particle Filter) tracking experimental results

The video contained the object to be tracked on an uncluttered background. All the tracked objects (faces) are detected by object detection procedures. Tracked faces are represented with blue rectangle. The most likely are marked with a red rectangle. The particles are marked with red points. A sampling of output images for tracking the face of one girl (labeled Sub1) detected by the face detector on Frame 111 of MVI_0078.avi in Figure 5.6 These picture generated by the object tracker viewer is provided to demonstrate its performance. The images presented were generated with $N = 100$ particles. The tracking procedure lasts more than 25 frames and then this face is out of track.

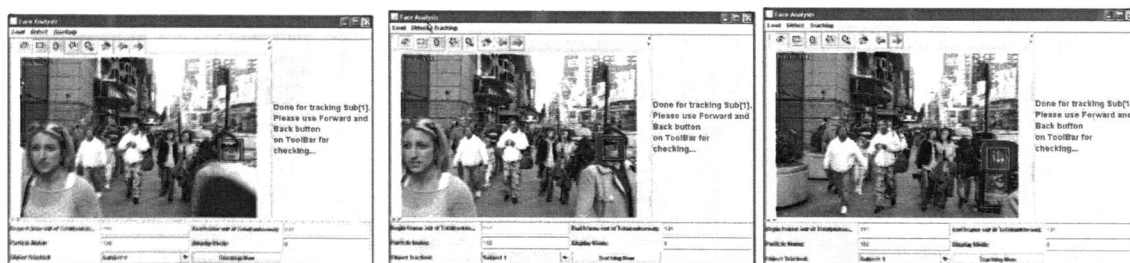


Figure 5.6 Start tracking Sub1 on Frame 111, 113, 127

Another sampling of output images for tracking the face of the other woman labeled Sub2 in Video MVI_0006.avi shown in Figure 5.7 generated by the tracker is provided to demonstrate its performance. The face is out of track after 4 frames from the beginning of tracking.

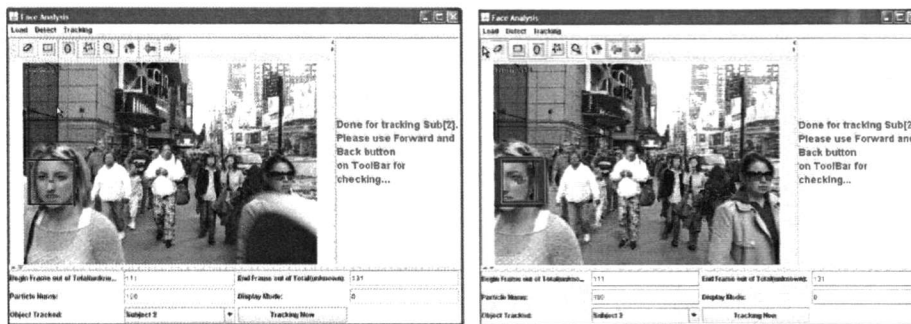


Figure 5.7 tracking Sub2 Start on Frame 111 and End on frame114

One bad tracking example shown in the Figure 5.8 starts tracking the face labeled Sub0 from Frame 120 of Video MVI_0078.avi. Because of noise, the tracker can not be kept track of the object very well in some frames, although it can stop when that person walks out of the view.



Figure 5.8 Tracking Sub0 on Frame 126, 143, 170 in MVI_0078.avi

Another good example is shown as below: we loaded one video of hockey players that contains over 500 frames and marked one player labeled Sub0 on Frame 198 with red rectangle for tracking. Figure 5.9 show tracking Sub0 procedures using display mode

1 that only display the most likely particle. This player can be tracked very well during over 100 frames.

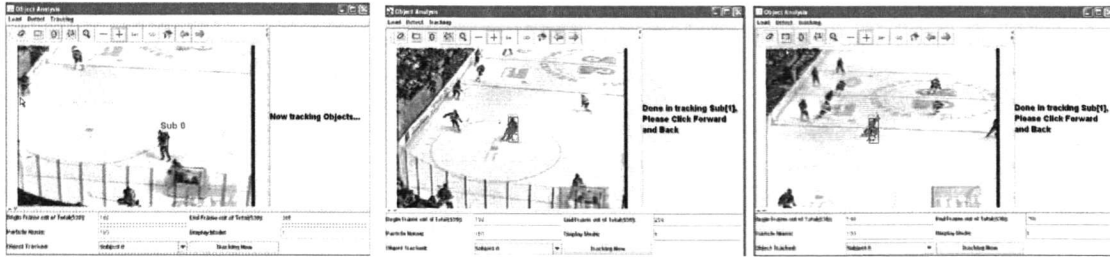


Figure 5.9 Tracking Sub0 on Frame 198-284

5.4 Analysis for experimental results

From our testing experiment results, we can see our framework works very well. Change Detection can help us to improve the performance of object detection and tracking. The faces can be detected by the detection procedures. The detected objects (faces) can be tracking by our particle filter tracking.

CHAPTER 6

CONCLUSION

In this thesis we describe how to design and implement object detection and tracking framework by using Java programming.

The object detection and tracking framework is a set of classes that embodies an abstract design for solutions to a family of object detection and tracking algorithms. It provides a set of the reusable object detection and tracking architecture designs. Several typical detection and tracking algorithms such as histogram-based change detection, adaboost object detection, particle filter object tracking are implemented as case studies in this framework. The framework is able to accommodate functionality in a number of object tracking and detection, to provide common behavior while allowing users and developers to customize behavior through configuration parameters and/or framework sub-classing.

6.1 Future Work

Our ultimate goal is to build a tracking system to handle any type of video tracking and detection method in a real video system. An important issue in video is to identify which algorithm is best for what kind of video data. This following issue needs to be addressed according to the classes suggested in the design in the future.

The case study is effective for generalizing using the framework. The purpose of our case studies is to provide clear and well-documented examples which demonstrate

or point to ways of how the framework. The linkage of issues across different sections of information within and between case studies and other data sources will be a key feature. The long term aim is to disseminate good practice in the sustainable management of framework. The information in the case studies will also be used as learning or teaching tools and for framework development. More detailed help and documents can help users to understand how the framework works.

Recognition of objects in video can offer significant benefits to video analysis. The problem in object recognition is to determine which, if any, of a given set of objects appear in a given image or image sequence. Object recognition is an important stage for pattern recognition. It can help the framework to extract objects from video frames faster. This component will be added to the framework before object detection component for extracting objects of interest.

Some algorithms are implemented in Java in our framework and more algorithms are needed to be implemented in Java and we see that it has a potential to handle and wrapper the existing algorithms in C++/C, MATLAB, and other programming languages. More efficient algorithms deserve further investigations, and we hope to see the design extended to cover them as well.

Based on our framework implementation, some algorithms may be combined to improve the performance or effectiveness of them. Or some new algorithms may be proposed through this framework development.

CHAPTER 7

REFERENCES

- [1] R. George, “Spatio-Temporal querying in video databases”, *ARL Final Report*, WINF: 04-02-0036, 2005
- [2] J. S. Boreczky and L. A. Rowe “Comparison of video shot boundary detection techniques”. *Journal of Electronic Imaging*, JEI(5), No.2, pp.122-128, April 1996.
- [3] O. D. Robles, P. Toharia, A. Rodríguez and L. Pastor., “Automatic Video Cut Detection using Adaptive Thresholds”. *4th International Conference on Visualization, imaging, and image processing*, VIIP, September 2004.
- [4] C. Veenman, M. Reinders, and E. Backer, “Resolving motion correspondence for densely moving points”, *IEEE Trans. Patt. Analy. Mach. Intell.* 23, 1, pp.54–72, 2001.
- [5] D. Serby, S. Koller-Meier, and L. V. Gool, “Probabilistic object tracking using multiple features”, *IEEE International Conference of Pattern Recognition (ICPR)*, pp.184–187, 2004.
- [6] D. Comaniciu, V. Ramesh, and P. Andmeer, “Kernel-based object tracking”, *IEEE Trans. Patt. Analy. Mach. Intell.* 25, pp.564–575, 2003.
- [7] D. Ballard, and C. Brown, “Computer Vision”, *Prentice-Hall, Englewood Cliffs, NJ*, 1982.
- [8] A. Ali, and J. Aggarwal, “Segmentation and recognition of continuous human activity”, *IEEE Workshop on Detection and Recognition of Events in Video*, pp.28–35, 2001.
- [9] A. Yilmaz, X. Li, and M. Shah, “Contour based object tracking with occlusion handling in video acquired using mobile cameras”, *IEEE Trans. Patt. Analy. Mach. Intell.* 26, 11, pp.1531–1536, 2004.
- [10] S. Zhu, and A. Yuille, “Region competition: unifying snakes, region growing, and bayes/mdl for multiband image segmentation”, *IEEE Trans. Patt. Analy. Mach. Intell.* 18, 9, pp.884–900, 1996.
- [11] N. Paragios, and R. Deriche, “Geodesic active regions and level set methods for supervised texture segmentation”, *Int. J. Comput. Vision* 46, 3, pp.223–247, 2002.

- [12] A. Elgammal, R. Duraiswami, D. Harwood, and L. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance", *Proceedings of IEEE 90*, 7, pp.1151–1163, 2002.
- [13] P. Fieguth, and D. Terzopoulos, "Color-based tracking of heads and other mobile objects at video frame rates", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.21–27, 1997.
- [14] G. Edwards, C. Taylor, and T. Cootes, "Interpreting face images using active appearance models", *International Conference on Face and Gesture Recognition*, pp.300–305, 1998.
- [15] B. Mughadam, and A. Pentland, "Probabilistic visual learning for object representation", *IEEE Trans. Patt. Analy. Mach. Intell.* 19, 7, pp.696–710, 1997.
- [16] M. Black, and A. Jepson, "Eigentracking: Robust matching and tracking of articulated objects using a view-based representation", *Int. J. Comput. Vision* 26, 1, pp.63–84, 1998.
- [17] S. Avidan, "Support vector tracking", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.184–191, 2001
- [18] S. Park, and J. K. Aggarwal, "A hierarchical bayesian network for event recognition of human actions and interactions", *Multimed. Syst.* 10, 2, pp.164–179, 2004.
- [19] J. Shi, and J. Malik, "Normalized cuts and image segmentation", *IEEE Trans. Patt. Analy. Mach. Intell.* 22, 8, pp.888–905, 2000.
- [20] H. Moravec, "Visual mapping by a robot rover", *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp.598–600, 1979.
- [21] C. Harris, and M. Stephens, "A combined corner and edge detector", *4th Alvey Vision Conference*, pp.147–151, 1988.
- [22] D. Lowe, "Distinctive image features from scale-invariant keypoints", *Int. J. Comput. Vision* 60, 2, pp.91–110, 2004.
- [23] K. Mikolajczyk, and C. Schmid, "An affine invariant interest point detector", *European Conference on Computer Vision (ECCV)*. Vol. 1, pp.128–142, 2002.
- [24] C. Stauffer, and W. Grimson, "Learning patterns of activity using real time tracking", *IEEE Trans. Patt. Analy. Mach. Intell.* 22, 8, pp.747–767, 2000.
- [25] N. Oliver, B. Rosario, and A. Pentland, "A bayesian computer vision system for modeling human interactions", *IEEE Trans. Patt. Analy. Mach. Intell.* 22, 8, pp.831–843, 2000.

- [26] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practices of background maintenance", *IEEE International Conference on Computer Vision (ICCV)*, pp.255–261, 1999.
- [27] A. Monnet, A. Mittal, N. Paragios, and V. Pamesh, "Background modeling and subtraction of dynamic scenes", *IEEE International Conference on Computer Vision (ICCV)*, pp.1305–1312, 2003.
- [28] D. Comaniciu, and P. Meer, "Mean shift analysis and applications", *IEEE International Conference on Computer Vision (ICCV)*. Vol. 2, pp.1197–1203, 1999.
- [29] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours". In *IEEE International Conference on Computer Vision (ICCV)*, pp.694–699, 1995.
- [30] C. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection", *IEEE International Conference on Computer Vision (ICCV)*, pp.555–562, 1998.
- [31] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection", *IEEE Trans. Patt. Analy. Mach. Intell.* 20, 1, pp.23–38, 1998.
- [32] P. Viola, M. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance", *IEEE International Conference on Computer Vision (ICCV)*, pp.734–741, 2003.
- [33] J. Shi, and C. Tomasi, "Good features to track", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.593–600, 1994.
- [34] C. Wren, A. Azarbayejani, and A. Pentland, "Pfinder: Real-time tracking of the human body", *IEEE Trans. Patt. Analy. Mach. Intell.* 19, 7, pp.780–785, 1997.
- [35] X. Gao, T. Boulton, F. Coetzee, and V. Ramesh, "Error analysis of background adaption", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.503–510, 2000.
- [36] J. Rittscher, J. Kato, S. Joga, and A. Blake, "A probabilistic background model for tracking", *European Conference on Computer Vision (ECCV)*, Vol. 2, pp.336–350, 2000.
- [37] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee, and J. Buhmann, "Topology free hidden markov models: Application to background modeling", *IEEE International Conference on Computer Vision (ICCV)*, pp.294–301, 2001.
- [38] J. Zhong, and S. Sclaroff, "Segmenting foreground objects from a dynamic textured background via a robust kalman filter", *IEEE International Conference on Computer Vision (ICCV)*, pp.44–50, 2003.

- [39] D. Comaniciu, and P. Meer, "Mean shift: A robust approach toward feature space analysis", *IEEE Trans. Patt. Analy. Mach. Intell.* 24, 5, pp.603–619, 2002.
- [40] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models", *Int. J. Comput. Vision* 1, pp.321–332, 1988.
- [41] L. Grewe, and A. Kak, "Interactive learning of a multi-attribute hash table classifier for fast object recognition", *Comput. Vision Image Understand.* 61, 3, pp.387–416, 1995.
- [42] G. Paschos, "Perceptually uniform color spaces for color texture analysis: an empirical evaluation", *IEEE Trans. Image Process.* 10, pp.932–937, 2001.
- [43] K. Y. Song, J. Kittler, and M. Petrou, "Defect detection in random color textures", *Israel Verj. Cap. J.* 14, 9, pp.667–683, 1996.
- [44] J. Canny, "A computational approach to edge detection", *IEEE Trans. Patt. Analy. Mach. Intell.* 8, 6, pp.679–698, 1986.
- [45] K. Bowyer, C. Kranenburg, and S. Dougherty, "Edge detector evaluation using empirical roc curve", *Comput. Vision Image Understand.* 10, pp.77–103, 2001.
- [46] B. Horn, and B. Schunk, "Determining optical flow", *Artific. Intell.* 17, pp.185–203, 1981.
- [47] B. D. Lucas, and T. Kanade, "An iterative image registration technique with an application to stereo vision", *International Joint Conference on Artificial Intelligence*, pp.674–679, 1981.
- [48] M. Black, and P. Anandan, "The robust estimation of multiple motions: Parametric and piecewise smooth flow fields" *Comput. Vision Image Understand.* 63, 1, pp.75–104. 1996.
- [49] R. Szeliski, and J. Coughlan, "Spline-based image registration", *Int. J. Comput. Vision* 16, 1-3, pp.185–203, 1997.
- [50] R. Haralick, B. Shanmugam, and I. Dinstein, "Textural features for image classification", *IEEE Trans. Syst. Man Cybern.* 33, 3, pp.610–622, 1973.
- [51] K. Laws, "Textured image segmentation", *PhD thesis, Electrical Engineering, University of Southern California.* 1980.
- [52] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation", *IEEE Trans. Patt. Analy. Mach. Intell.* 11, 7, pp.674–693, 1989.

- [53] H. Greenspan, S. Belongie, R. Goodman, P. Perona, S. Rakshit, and C. Anderson, "Overcomplete steerable pyramid filters and rotation invariance", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.222–228, 1994.
- [54] D. Cremers, T. Kohlberger, C. Schnörr, "Nonlinear shape statistics in mumford shah based segmentation", *European Conference on Computer Vision*, pp.93–108, 2002
- [55] A. Jepson, D. Fleet, and T. Elmaraghi, "Robust online appearance models for visual tracking", *IEEE Trans. Patt. Analy. Mach. Intell.* 25, 10, pp.1296–1311, 2003.
- [56] V. Salari, and I. K. Sethi, "Feature point correspondence in the presence of occlusion", *IEEE Trans. Patt. Analy. Mach. Intell.* 12, 1, pp.87–91, 1990.
- [57] T. Broida, and R. Chellappa, "Estimation of object motion parameters from noisy images", *IEEE Trans. Patt. Analy. Mach. Intell.* 8, 1, pp.90–99, 1986.
- [58] Y. Bar-Shalom, and T. Foreman, "Tracking and Data Association", *Academic Press Inc.*, 1988.
- [59] R. L. Streit, and T. E. Luginbuhl, "Maximum likelihood method for probabilistic multi-hypothesis tracking", *Proceedings of the International Society for Optical Engineering (SPIE.)* vol. 2235, pp.394–405, 1994.
- [60] H. Tao, H. Sawhney, and R. Kumar, "Object tracking with bayesian estimation of dynamic layer representations", *IEEE Trans. Patt. Analy. Mach. Intell.* 24, 1, pp.75–89, 2002.
- [61] M. Isard, and A. Blake, "Condensation - conditional density propagation for visual tracking", *Int. J. Comput. Vision* 29, 1, pp.5–28, 1998.
- [62] M. Bertalmio, G. Sapiro, and G. Randall, "Morphing active contours", *IEEE Trans. Patt. Analy. Mach. Intell.* 22, 7, pp.733–737, 2000.
- [63] R. Ronfard, "Region based strategies for active contour models", *Int. J. Comput. Vision* 13, 2, pp.229–251, 1994.
- [64] D. Huttenlocher, J. Noh, and W. Rucklidge, "Tracking nonrigid objects in complex scenes", *IEEE International Conference on Computer Vision (ICCV)*, pp.93–101, 1993.
- [65] K. Sato, and J. Aggarwal, "Temporal spatio-velocity transform and its application to tracking and interaction", *Comput. Vision Image Understand.* 96, 2, pp.100–128, 2004.

- [66] J. Kang, I. Cohenand, G. Medioni, "Object reacquisition using geometric invariant appearance model", *International Conference on Pattern Recongnition (ICPR)*, pp.759–762, 2004.
- [67] I. Sethi, and R. Jain, "Finding trajectories of feature points in a monocular image sequence", *IEEE Trans. Patt. Analy. Mach. Intell.* 9, 1, pp.56–73, 1987.
- [68] K. Rangarajan, and M. Shah, "Establishing motion correspondence", *Conference Vision Graphies Image Process* 54, 1, pp.56–73, 1991.
- [69] H. Tanizaki, "Non-gaussian state-space modeling of nonstationary time series", *J. Amer. Statist. Assoc.* 82, pp.1032–1063, 1987.
- [70] Y. L. Chang, and J. K. Aggarwal, "3d structure reconstruction from an ego motion sequence using statistical estimation and detection theory", *Workshop on Visual Motion*, pp.268–273, 1991.
- [71] C. Rasmussen, and G. Hager, "Probabilistic data association methods for tracking complex visual objects", *IEEE Trans. Patt. Analy. Mach. Intell.* 23, 6, pp.560–576, 2001.
- [72] J. Sethian, "Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics Computer Vision and Material Sciences", *Cambridge University Press*, 1999.
- [73] A. Mansouri, "Region tracking via level set pdes without motion computation", *IEEE Trans. Patt. Analy. Mach. Intell.* 24, 7, pp.947–961, 2002.
- [74] D. Cremers, and C. Schnorr, "Statistical Appearance knowledge in variational motion segmentation", *Image and Vision Computing*, 21(1), pp.77–86, 2003.
- [75] Mohamed E. Fayad, E. Johnson Ralph and Douglas C. Schmidt, "Building Application Frameworks", *John Wiley & Sons*, 1999.
- [76] R. E. Johnson, and B. Foote. "Designing reusable classes", *Journal of Object-Oriented Programming* 1(2), pp.22-35, 1988
- [77] P. L. Deutsch, "Design reuse and frameworks in the Smalltalk-80 system", *Software reusability, volume II: applications and experience. MA, Addison-Wesley*, pp.57-71, 1989.
- [78] <http://java.sun.com/products/java-media/jmf/2.0/jmf20-08-guide.pdf>
- [79] W. A. C. Fernando, C. N. Canaharajah, D. R. Bull "Fade-In and Fade-Out Detection in Video Sequences Using Histograms", *Proc. ISCAS 2000 IEEE International Symposium on Circuits and System, IV, Geneva, Switzerland*, pp.709-712, 2000.

- [80] C. Y. Low, Q. Tian, and H. Zhang. "An automatic news video parsing, indexing and browsing system". In *Proc. of ACM Int'l Conf. on Multimedia, Boston, MA*, pp.425-426, November 1996
- [81] A. Kien, Jung-Hwan. Hua, "Detecting video shot boundaries up to 16 times faster." *ACM Multimedia 2000*, pp.385-387, 2000
- [82] P. Viola, and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition*, pp.511–518, December 2001.
- [83] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, 55(1), pp.119-139, 1997
- [84] R. Lienhart, A. Kuranov, and V. Pisarevesky, "Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection," *MRL Technical Report, Intel Labs*, 2002
- [85] Open Source Computer Vision Library:
<http://www.intel.com/research/mrl/research/opencv>