Extracting and Modeling Useful Information from Videos for Supporting Continuous Queries

by

MANISH KUMAR ANNAPPA

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2016

Copyright © by MANISH KUMAR ANNAPPA 2016

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Sharma Chakravarthy for providing me an opportunity to work on this project and constantly motivating me and guiding me throughout this thesis work. I would like to thank Dr. Vassilis Athitsos for his constant guidance throughout this research work and taking time to serve in my thesis committee. Without their constant support and guidance this research work would not have been successful. I would also like to thank Dr. Ramez Elmasri for taking time to serve in my thesis committee.

I would like to extend my gratitude to all the people involved in this project, IT - Lab present members and alumni Abhishek Santra, Soumyava Das, Jay Bodra, Nandan Prakash, Damini Singh, Jasmine Varghese and Mayoor Arora for their constant support and guidance. Also, I would like to thank my family and friends for their constant support and encouragement throughout my academic career.

November 15, 2016

ABSTRACT

Extracting and Modeling Useful Information from Videos for Supporting Continuous Queries

MANISH KUMAR ANNAPPA, M.S.

The University of Texas at Arlington, 2016

Supervising Professor: Dr. Sharma Chakravarthy

Automating video stream processing for inferring situations of interest has been an ongoing challenge. This problem is currently exacerbated by the volume of surveillance/monitoring videos generated. Currently, manual or context-based customized techniques are used for this purpose. To the best to our knowledge the attempted work in this area use a custom query language to extract data and infer simple situations from the video streams, thus adding an additional overhead to learn their query language. Objective of the work in this thesis is to develop a framework that extracts data from video streams generating a data representation such that simple "what-if" kind of situations can be evaluated from the extracted data by posing queries in a Non-procedural manner (SQL or CQL without using any custom query languages). This thesis proposes ways to pre-process videos so as to extract the needed information from each frame. It elaborates on algorithms and experimental results for extracting objects, their features (location, bounding box, and feature vectors), and their identification across frames, along with converting all that information into an expressive data model. Pre-processing of video streams to extract queryable representation involves parameters and techniques that are context-based, that is, dependent on the type of video streams and the type of objects present in them. And lot of tuning of values or experiments is essential to choose right techniques or right values for these parameters. This thesis additionally, proposes starting values for such tuning or experiments, in order to reach the right values.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Big data analytics is about the ability to process and automate not only structured data but also disparate data types and their combinations. One such data type is a video stream. Significant progress has been made in our abilities to collect various types of data. Sensors have allowed us to capture numerical (and structured) data on a 24/7 basis. Now, thanks to the availability of mobile and inexpensive cameras and unmanned aerial vehicles, it is possible to collect large amounts of video data for security, surveillance, and other purposes [1]. However, automated processing (or even semi-automated processing where a human is assumed to be in the loop) of video data is still a long way to go notwithstanding the progress made in image processing and pattern recognition. Querying of video contents (for situation analysis) without a significant learning curve is critical for understanding/checking events in a video stream (e.g., person enters/leaves a building), spatial and temporal activities that can be inferred from a video (e.g., patterns of car movement from a house, duration of stay by an individual in a facility), and complex activities such as exchange of objects by people crossing each other, pickup of a person by a vehicle. This has given rise to an emphasis on video stream analysis for situation monitoring and reduce manual involvement in the processing of videos. Typically, video analysis surveillance is done manually by watching the video frames continuously for identifying the above-indicated kinds of events and anomalies. The manual approach is resource-intensive (especially human resource) and is also prone to errors. Some applications, such as security surveillance for home or commercial places like malls, parking lots etc.

may tolerate and can accept some degree of errors or approximations but mission-critical applications expect higher accuracy. Clearly, there is a need to automate video stream analysis to infer as many types of events as possible in an automated or semi-automated manner that significantly reduces (if not eliminates) human intervention. In order to accomplish this, we need an approach such as database querying where different types of queries can be asked and evaluated on the same data once the data is captured and represented in a desirable manner.

The long-term goal of our work is to be able to query situations continuously on live video streams in real time and make it as easy as querying a database. The work in this thesis is a preliminary step towards this goal. Video analysis automation is achieved in our work by making use of the best practices available currently in Computer Vision. It is beneficial to understand the present work in the field of automating video stream analysis to appreciate what is being proposed and how it is a major step towards complete automation as a long-term goal. Broadly, current body of work in automating video stream analysis can be divided into three categories: i) Video indexing and content-based retrieval: ([2], [3], [4]), ii) Human activity detection in video sequence: ([5], [6], [7]) and iii) Live video querying ([8], [9]). Briefly, *Video indexing and content-based retrieval* involves pre-processing the video content to extract important features like actors (through face identification), peculiar scene, colors etc. and indexing them for easy retrieval. Queries on these stored video data will make use of the features (information) extracted. Examples of such queries are "Return videos that contain scenes such as a man is walking on the meadow" [10] and "Find me a segment of the video where a party occurs" [2]. In these mentioned approaches, such query operations are accomplished by providing the query input as a video clip that contains the action to be searched. *Human activity detection* in video involves detection of predefined human actions by training detection models with video content that contains similar actions (similar to the supervised approach of machine learning). *Live video querying* involves real-time identification of objects of interest, extracting important features from each of the video frames so that this information can be converted to a queryable format.

The work in first two categories has extensive methods to pre-process video and convert them to a queryable format. However, they either target a specific class of queries (e.g. querying for human actions) or some of these frameworks expect the input as video clips. Also, the work in the Live video querying category uses either a custom query language or expects users to formulate queries pragmatically if some of the required operators are not present to express a situation (details are mentioned in Section 3). In contrast, the work in this thesis proposes video pre-processing methods that generates output by extracting generic information from the video to express simple "what-if" kind of situations (not specific actions, but related to generic object tracking) in a non-procedural manner (that is, not having to write code but to understand the semantics of the operator and use it meaningfully). As an example, in a video which captures people entering a mall (or a checkpoint) and exiting a mall (or a checkpoint), this thesis work focuses on evaluating queries such as Query 1: "How many distinct people are entering the mall throughout the day in each hour?" or Query 2: "Count average number of cars in a parking lot over an interval or calculate when cars are absent in a driveway" to learn travel patterns.

Although the long-term goal of this work is to be able to ask arbitrary queries (based on the expressiveness of the language supported) on live video streams in real-time, the initial steps are to identify methods to extract important and relevant information and generate a canonical queryable representation from available video streams. The framework proposed in this thesis can be extended for the live context once real-time processing requirements are clearly understood.

Details of the overall approach proposed in this work to query a Video Stream is explained in the next chapter (Chapter 2).

CHAPTER 2

Proposed Approach

Our ultimate goal in this work is to be able to evaluate a large class of queries on video streams. Querying video streams in real time requires handling additional requirements once we have demonstrated that general-purpose queries can be processed on the contents of videos represented in a suitable canonical form. Real-time processing of live videos will entail efficiency issues, use of main memory effectively, and the QoS (quality of service) requirements for a particular video in question. Stream processing (SP) and Complex event Processing (CEP) (supported in a DSMS or Data Stream Processing System in contrast to a DBMS or Data Base Management System) are already well established techniques for sensor data processing including the availability of many commercial systems. QoS issues, such as latency have been addressed through optimal scheduling and load shedding trade-offs. In general, Data Stream Management System (DSMS) and Stream Processing (SP) frameworks are applicable to process data streams from sensors in many applications (e.g., environmental monitoring, battlefield monitoring). Other continuous data generators such as web clicks, traffic data, network packets, mobile devices have also prompted processing and analysis of these streams for applications such as traffic congestion/accidents, network intrusion detection, and personalized marketing. Additionally, Complex event Processing (CEP) has come a long way to process continuous, varying input rate data and event streams from sensors and events generated by other applications.

Video can be viewed as a stream of frames generated as part of a video stream. It can be viewed as unbounded if it is an application that requires continuous monitoring.

Otherwise, it can be viewed as a bounded (albeit large at 30 frames per second) and queries can be processed on that portion of the video stream. Although the streaming nature is the same and some of the real-time requirements are similar, the types of queries and the kinds of processing required for querying video streams are very different. Hence, we can use the framework, but need to develop new or extended representations, new operators (including spatial and temporal ones) for querying and their semantics, as well as what and how much of information to extract to facilitate this. As we already have a stream processing system *MavEStream* developed at the IT Laboratory, UT Arlington, we plan on extending it for querying videos as needed. To elaborate, these frameworks cannot be used as is for video analysis due to the type of data representation used by them. Typically, Stream Processing systems consume the data streams for querying in the form of relational representation with defined schema and data types as textual, numerical and categorical. The work in this thesis proposes a video stream pre-processing framework to extract data from video streams and convert them to a canonical representation (an extended relational representation) so that it can be efficiently queried and evaluated by the Stream Processing (SP) system.

As a result of this thesis work we aim to generate a representation similar to Table 2.1 for the video stream (mall video) mentioned in Query 1 ("How many distinct people are entering the mall throughout the day in each hour?"). Certain aspects of the contents of a video, such as a bounding box, feature vector, object location require vector representation.

In Table 2.1 *ts* represents the time-stamp associated with video frames. *fid* represents the incremental id given for each frame of the video starting at 1. *glabel* represents the unique label given to each moving object present in the video. *b_box* represents the bounding box (the minimum rectangle that confines the outline of an

ts	fid	glabel	b_box	quadrant	fv
6:40:10	1	1	[240, 556, 24, 32]	top_right	$[fv1]$
6:40:10	1	2	[230, 700, 22, 31]	top_right	$[fv2]$
6:40:10	2	1	[242, 566, 24, 32]	top_right	$[fv3]$
6:40:10	2	2	[240, 708, 22, 31]	top_right	$[fv4]$
6:40:10	3	1	[244, 585, 24, 32]	top_right	$[fv5]$
6:40:10	3	2	[250, 726, 22, 31]	top_right	$[fv6]$

Table 2.1: VS1:Relational Representation of the video stream in Query 1

object) of an object. *quadrant* represents the position of the object in one of the four quadrants of the frame (assuming a static camera). *fv* is the feature vector (A vector that represents the unique information in an image. Additional details are given in the Chapter 5.). Note that this is an extended relational model as vectors are not supported in the traditional data model.

A simple query like "How many people entered the mall in the given video segment" can be answered by Structured Query Language (SQL) semantics as shown in Figure 2.1. A query like Query 1 ("How many distinct people are entering the mall throughout the day in each hour?") that needs the expression "count the number of people entering" to be evaluated for every 1 hour for the given video stream, cannot be evaluated using SQL semantics. This data pertaining to every 1 hour can be obtained by the usage of window semantics. Continuous Query Language (CQL) is an expressive SQL-based declarative language for registering continuous queries against streams and updatable relations [11]. It supports time based, tuple based and partition based types of sliding window semantics. Thus, continuous queries like Query 1 (the complete query is shown in the Chapter 6) can be expressed using CQL (the complete query is shown in the Chapter 6) semantics.

MavEStream has well-defined relational stream operators, windows of different types and supports standard SQL and CQL semantics. As mentioned above, the

```
SELECT     COUNT(DISTINCT vs1.glabel)
FROM       VS AS vs1
```

Figure 2.1: How many people entered the mall in the given video segment

Query 1 ("How many people are entering the mall throughout the day in each hour?") can be easily expressed using the operators present in Stream Processing (SQL and CQL) over the relational data model to be generated as a result of video stream pre-processing (Table 2.1). However, the standard semantics present in SQL and CQL may not be sufficient to express situations that are not as simple as Query 1. For example, let us consider this query [12] (Query 3) "Retrieve everyone in the video who has appeared in more than three consecutive frames". The corresponding SQL expression for Query 3 is given in Figure 2.2.

```
SELECT       DISTINCT vs1.glabel vs1.fid,
             vs2.fid, vs3.fid
FROM         VS AS vs1, VS AS vs2, VS AS vs3
WHERE        vs2.fid = vs1.fid + 1
             AND vs3.fid = vs2.fid + 1
             AND vs1.glabel = vs2.glabel
             AND vs2.glabel = vs3.glabel
ORDER BY     vs1.glabel ASC
```

Figure 2.2: Query 3. Retrieve all objects that appear in three consecutive frames

The query in Figure 2.2 is not only difficult to understand but the time taken to compute this query will be significantly more due to several SELF JOIN and Cartesian products. The query gets more complex when more consecutive frames are considered. Query 3 ("Retrieve everyone in the video who has appeared in more than three consecutive frames") is a continuous query that requires aggregation (counting) on an order dependent data (consecutive property is defined on the order of time stamp) with rolling window (every three consecutive frames) basis.

An extension to the SQL called AQuery has been proposed in [13], using which queries on order dependent data can be expressed and efficiently evaluated using the window semantics. AQuery supports order-dependent, size-preserving operators (running aggregates) and size-non-preserving (standard SQL aggregate functions like min,max,count etc.). The data model expected by AQuery is called an *arrable.* An arrable is obtained by applying GROUP BY operation on one of the columns in the target (on which query has to be run) relational representation and also specifying an ordering of tuples based on one or more of the columns. Thus, *arrable* representation can be adopted into our work as well to answer continuous queries that involve computation of moving aggregates. If we group by Table tab:data-representation on glabel (unique object label) assuming order on ts (time stamp), the resulting *arrable* representation would be as shown in Table 2.2.

ts	fid	glabel	b_box	quadrant	fv
[6:40:10:0001, 6:40:10:0003, 6:40:10:0005]	[1, 2, 3]	1	[[240, 556, 24, 32], [242, 566, 24, 32], [244, 585, 24, 32]	[top_right, top_right, top_right]	[[$fv1$], [$fv3$], [$fv5$]]
[6:40:10:0002, 6:40:10:0004, 6:40:10:0006]	[1, 2, 3]	2	[[230, 700, 22, 31], [240, 708, 22, 31], [250, 726, 22, 31]]	[top_right, top_right, top_right]	[[$fv2$], [$fv4$], [$fv6$]]

Table 2.2: Arrable Representation of VS1

Figure 2.3 shows the corresponding AQuery for Query 1 ("How many distinct people are entering the mall throughout the day in each hour?"). We can extend the AQuery operators by developing customized operators for our domain (AQuery was primarily targeted for financial class of application). *consecutive* is a new operator that operates on the frame-id (fid) vector (vector generated due to GROUP BY on glabel) and selects tuples if the condition is satisfied. In Query 3 ("Retrieve everyone in the video who has appeared in more than three consecutive frames") the condition is "more than three consecutive frames". The consecutive operator returns a frame-id (fid) vector if it finds three consecutive frames in it. Since the representation is ordered on fid vector, *consecutive* function was able to select fid vectors that have 3 consecutive frame-id. It is evident that the query in Figure 2.3 is simpler compared to the one in Figure 2.2 as they do not have any self-joins or Cartesian products. Thus, we plan to use the arrable representation for video processing and extend it if needed. Our first step is to incorporate the new representation and AQuery operators into MavEStream.

```
SELECT      glabel, fid
FROM        VS1
ASSUMING    ORDER fid
WHERE       consecutive(3, [fid])
GROUP BY    vs1.glabel ASC
```

Figure 2.3: AQuery for Query 3

As mentioned earlier, the work in this thesis focuses on extracting appropriate data from video streams, represent them in an appropriate model, and evaluate useful situational queries. Additionally, it identifies the types of data to be extracted from a video stream and state-of-the-art image processing techniques relevant for this extraction. The thesis also streamlines this extraction process by providing configurable parameters so end users need not worry about the extraction details or the image processing aspects. Even though stored videos are used in the experimentation of the thesis, the methods proposed are applicable to live video streams as well.

Let us consider Query 1 ("How many distinct people are entering the mall throughout the day in each hour?") and assume that the time interval during which the query to be answered is from 8am to 8pm (mall open hours) for every hour. Following are some of the relevant data that need to be gathered from the video stream and processed to answer the query:

1. Identify each person/object in every frame and label them such that the same person/object in different frames gets the same label.
2. The labeled people need to be filtered for frames between mall open hours for each hour which can be done using a window scheme used in continuous query processing.
3. The filtered result should be counted (aggregated) for the number of occurrences of unique labels.
4. Finally the above computation is repeated for each window of interest.

There are several Computer Vision techniques available to extract information indicated above from video streams. The intention of this work is to use the best available Computer Vision techniques to extract objects and related information and to output a relational representation for a given video streams. This thesis is not about extending or proposing new image processing techniques or algorithms. The first part of this thesis explains in detail the information to be extracted from the video streams and methods to extract them, demonstration of how situations are expressed using non-procedural query semantics and comparison of these expressions to some of the related work. Pre-processing of video streams to extract queryable representation involves parameters and techniques that are context-based, that is, dependent on the type of video streams and the type of objects present in them. And lot of tuning of values or experiments is essential to choose right techniques or right values for these parameters. The second part of this thesis focuses on hiding implementation details like object identification and object labeling and exposing these above mentioned parameters as configurable settings in the video stream pre-processing framework. This thesis additionally, proposes starting values for such tuning or experiments, in order to reach the right values.

2.1 **Contribution**

Following are the contribution of this thesis:

1. Evaluation and identification of the types of information to be extracted from a video.
2. Canonical representation of the information extracted to facilitate situation analysis through querying.
3. Application of the current computer vision techniques to Identify Objects and other information from video streams, Extract Objects from video frames using Background Subtraction methods.
4. Experimental evaluation of dimensionality reduction algorithms (Feature Vector Extraction) for different types of objects.
5. Formulation of Object Labeling (or Tracking) algorithm that will aid in answering situational queries related to object movement or presence.
6. Identification of limitations in the usage of these Computer Vision methods for the above steps.
7. Demonstration of usage of the generated data representation by evaluating few of the simple situational queries using SQL, CQL, and MavEStream semantics.
8. Comparison of the generated representation with few of the related work in this field.

9. Extending the CQL window semantics to answer more complex situational queries in video streams and their experimental evaluation.

2.2 Thesis Organization

Rest of the thesis is organized as follows. Chapter 3 discusses some of the related work in this field. Chapters 4 and 5 elaborates on object extraction, feature extraction and discusses experiments performed for identifying thresholds to compare human and car object types. Chapters 6 and 7 describe the algorithm used for labeling similar objects across frames and describes the proposed data model. Chapter 8 describes the key-parameters used in video pre-processing and the starting values recommended for these parameters. Section 9 includes conclusions and future work.

CHAPTER 3

RELATED WORK

The amount of work done in the literature on Querying/Processing Video Stream data is overwhelming as it spans image processing, pattern recognition, and storage, management, and processing of extracted video contents. Covering all of that is not only unnecessary but is also distracting from the details of this thesis. Hence, we have selected a few important and relevant work that is close to the work proposed in the thesis.

3.1 VIRAT (Video and Image Retrieval and Analysis Tool)

VIRAT is a video surveillance project funded by the Information Processing Technology Office (IPTO) of the Defense Advanced Research Projects Agency (DARPA) [7]. The goal of this project is to develop a software that goes through huge video datasets and alerts if a particular event of interest occurs thus reducing the burden on military operators who either manually have to go through the video contents using fast-forward approach or search using already assigned meta-data tags (through offline processing). An example of such interesting event is "find all of the footage where three or more people are standing together in a group" [14].

Following are the major categories of the events detected by VIRAT [7]:

1. *Single Person:* Digging, loitering, picking up, throwing, exploding/burning, carrying, shooting, launching, walking, limping, running, kicking, smoking, gesturing

2. *Person-to-Person:* Following, meeting, gathering, moving as a group, dispersing, shaking hands, kissing, exchanging objects, kicking, carrying an object together
3. *Person-to-Vehicle:* Driving, getting-in (out), loading (unloading), opening (closing) trunk, crawling under car, breaking window, shooting/launching, exploding/burning, dropping off, picking up
4. *Person-to-Facility:* Entering (exiting), standing, waiting at checkpoint, evading checkpoint, climbing atop, passing through gate, dropping off
5. *Vehicle:* Accelerating (decelerating), turning, stopping, overtaking/passing, exploding/burning, discharging, shooting, moving together, forming into convoys, maintaining distance
6. *Other:* convoy, parade, receiving line, troop formation, speaking to crowds, riding/leading animal, bicycling

VIRAT architecture is mainly based on indexed Data Store of video clips containing the actions to be detected that are similar to the once indicated above. VIRAT trains action detection modules with these archived video data and tries to detect actions listed above in the target video stream. In summary, VIRAT has focused on customized software to detect a large set of pre-determined set of interest to the project. It is not a general-purpose approach where the indexed information can be used to process arbitrary queries. Finally, it is an off-line approach to understand the contents of a video.

The work in this thesis contributes towards a larger goal which is different from that of VIRAT. The goal is to pre-process video streams to generate a rich canonical representation over which different types of queries can be posed. The motivation for this work is to move towards generalization from the customized way of processing videos. As will be shown in Section 6, the representation will allow us to process arbitrary and what if queries and has the further potential of being able to do this in real-time using the real-time aspects of continuous query processing systems. Identifying new event types is possible by addition new operators to the system rather than writing new customized code.

3.2 Live Video Database Management System (LVDBMS)

The components of LVDBMS can logically be grouped into four tiers, the lowest consisting of physical hardware (camera layer). Next is the spatial processing layer, then the stream processing layer and finally the client layer. Queries originate in the client layer and are pushed down to the stream processing layer and then to the spatial processing layer. Data originates in the camera layer and flows upwards. As it moves upwards it is transformed; from a stream of imagery in the lower layer to streams of sub-query evaluations to a stream of Boolean query evaluations [9].

The main intent of LVDMBS is to match objects in one video stream with objects in another stream using bipartite graph comparison. LVDBMS uses a query semantics called Live Video Query Language (LVQL) to express complex events formed by simple events connected by spatial or temporal operators. For example, a simple event could be a person (or more generally some object) appearing in a scene or moving in front of a desk (where the term scene refers to some portion of the real world that is observed by a camera and rendered into a sequence of frames in a video stream). For example, a complex event could be defined as a person first appearing in a scene and then, within some threshold of time, moving in front of a desk [9].

LVQL supports spatial operators like "Appear()", "North()", "Inside()", "Meet()" etc., temporal operators like "Before()", "Meets()" etc. and traditional Boolean operators like "and", "nor" etc. Additionally, LVDBMS supports the following three types of objects:

1. Static Objects: These objects are not automatically detected, but marked in the video streams by users using the LVDBMS GUI while forming LVQL queries. They are typically denoted as Appear(s1.12, 50), where s1.12 is the ID assigned to the marked static object and the Appear() operator will return true only if the marked static object s1.12 has bounding box more than 200 pixels.
2. Dynamic Objects: These type of objects are programmed to be automatically detected in LVDBMS and are denoted by an asterisk (* e.g. Appear(s1.*, 200)). Appear() operator will return true if there are any dynamic objects in the video stream that have bounding box more than 200 pixels.
3. Cross-Camera Dynamic Objects: These are the dynamic objects in one stream that are matched with another stream. They are generally denoted by #. For example, Before(Appear(c0.#),Appear(c1.#),60) returns true if an object in stream1 matches an object in stream2 after a time gap of at least 60 seconds.

LVDBMS differs from the work in this thesis, in the type of queries targeted, operators, type of data extracted etc. Following comparison highlights the differences between LVDBMS and the work in this thesis.

1. LVDBMS uses low-level query language LVQL and expects the users to build applications using LVQL to express situations in high-level query languages like CQL,SQL etc. Whereas, the work in this thesis aims to extract a representation from video streams using which users should be able to express situations using common non-procedural languages like CQL,SQL, without needing to build any application. We plan on adding appropriate operators for this domain to CQL and provide clear semantics, and integrate it with the rest of the query language so that they can be used in conjunction with others.
2. Most of the operators in LVQL like Appear(), North() etc return only the Boolean values limiting to queries that result in Boolean values using such

operators. The work in this thesis aims at a query language that is closed with respect to the representation allowing composition of any operator with any other available operator and includes aggregation and ordering as well.

3. LVDBMS does not facilitate queries that require tracking of unique dynamic objects within a single stream. Currently proposed operators (Appear()) in LVQL treat all the dynamic objects as same. In other words, the output of Appear() for dynamic object related queries does not contain unique labels assigned to each of the objects detected. Appear() translates to if any dynamic object has satisfied the given threshold or not. If unique objects have to be tracked using Appear(), the object to be tracked has to be marked manually using LVQL GUI before the query is executed. Whereas the work in this thesis aims to answer queries related to tracking unique objects within or across streams.
4. Finally, usage of custom operators introduces a significant learning curve for users to learn the new semantics. Whereas, the work in this thesis uses the standard semantics in CQL and/or SQL.

3.3 MedSMan: A Streaming Data Management System over Live Multimedia

MedSMan [8] presents a system that enables direct capture of media streams from various sensors and automatically generates meaningful feature streams that can be queried by a data stream processor.

MedSMan aims to answer queries like "Display the speakers video when he is answering the 4th question about multimedia" in a broadcast seminar event. MedSMan makes use of the multimedia streams from various devices such as presentation projector, speaker's microphone, video device capturing the podium, movement sensors etc. MedSMan uses the information from these Media Streams to inform users (who have subscribed to this broadcast event) when an event such as the above query occurs.

MedSMan lets the users create definitions of Media Streams at the Stream Processing layer via Media Stream Description Language (MSDL). A Media Stream is usually the output of a sensor device such as a video, audio, or motion sensor that produces a continuous or discrete signal, but typically cannot be directly used by a data stream processor. MedsMan continuously extracts the content-based data descriptors called features using Feature Generation Functions (FGFs) (defined in Feature Stream Description Language or FSDL from the Media Streams to create Feature Streams. These Feature Streams are used by the stream processors to query and return the corresponding segment from the Media Stream as the result. MedSMan uses custom query language called Media and Feature Stream Continuous Query Language (MF-CQL) to continuously query over live media. MF-CQL is CQL [11] extended with additional syntax and shortcuts to express the extended semantics beyond DSMS.

Although MedSMan is focused on answering situational queries with the aid of various sensor data, it can be used to answer queries on live video streams using MF-CQL query language. Following comparison highlights the difference between MedSMan and the work in this thesis.

1. MedSMan expects the user to define (schema) media streams by specifying various attributes that need to be extracted from sensors (including video capturing device). The work in this thesis automatically generates data representation or schema that is appropriate to express simple situational queries on live video streams.
2. MedSMan expects the users to use FSDL to process the media streams to generate Feature Streams. If a particular method to accomplish a task (for example,

track unique moving objects in a video stream) that processes Media Stream is not available in MedSMan, then it expects the user to define a new Feature Generation Function (FGF) to achieve this task pragmatically. Whereas, the work in thesis hides the processing of the video streams (object tracking algorithm) from users and users do not need to develop any new methods to express simple situations.

In summary, the proposed work, of which this thesis is a part of , is different from the relevant work in the literature. Our goal is to make querying of videos as general-purpose as possible and remove the learning curve present in most other approaches. This approach not only makes it easy to automate, also has the potential of adding real-time functionality to the framework (as has been done for sensor applications using stream processing). Furthermore, the use of CQL-like language affords optimizations by the system that are typically not possible or has to be done by the developed which is not ideal.

CHAPTER 4

Object Identification and Background Subtraction

Object Identification involves identifying the pixels in a frame that contributes to the object of interest. The object of interest in a video frame can be either a static object or a moving object. There are several methods available in Computer Vision to detect the pixels related to the object of interest in a video stream frame. Still object identification is mostly done by training a model using similar images of the object that needs to be identified [15]. As our current focus is on video streams for tracking objects that are moving, we have concentrated only on the identification of moving objects [1]. Before we delve into the details of *Object Identification*, following are the definitions of important fundamental components in *Object Identification*.

4.1 Definitions

Definition 4.1.1. Frame Representation: A video stream consists of continuous frames. Each frame is represented as an image in general. In our work, each frame is represented by the notion F_i, where 'i'denotes the sequence number (temporal order) of the frame in the video stream. Mathematically each of these frames (images) are represented by a matrix [m x n] where m, n represent the image dimension (or resolution) as width and height respectively. Every frame of a video stream will have the same dimension as the video stream. The matrix values represent the bit-depth (color-depth or intensity) of each of the pixels in the image. The possible range of

[1]For parking lot and other kinds of surveillance, it may be necessary to identify objects that do not move. This does not change the modeling and querying aspects of our work.

the bit-depth is from 1 to 48. For simplicity, we have used 8-bit depth in our work to represent image pixels.

The color images are represented by 3 dimension matrix with each planes corresponding to pixel intensity in the colors Red, Green and Blue. In contrast Grayscale images are 2 dimensional matrices whose pixel intensity varies between 0-255 (for 8-bit depth) with 0 representing a black pixel and 255 white. In general color images are process intensive to deal with so they are converted to Grayscale before performing any operations on them. Following is the formula to convert a color image to Grayscale.

$$G = 0.30 * I_R + 0.59 * I_G + 0.11 * I_B \tag{4.1}$$

G is the Grayscale converted image, I_R is the intensity of a pixel in the image for channel Red, I_G is the intensity of the same pixel for channel Green and I_B is the intensity of the same pixel for channel Blue. For further details about the image representation, please refer to [16].

Definition 4.1.2. Object Representation: Object of interest in a video frame can be anything like people, cars, animals, couch, door, fish etc. Once the object of interest is identified, it needs to be represented in a form such that it can be separated from the background using the information present in this representation. Following are the few available representation for representing objects as per the survey present in [17].

1. **Points:** Object can be represented by a point, which is the centroid of the object (Figure 4.1 (a)) or a set of points (Figure 4.1 (b)). This representation is more suitable when the object occupies a small region in the frame.

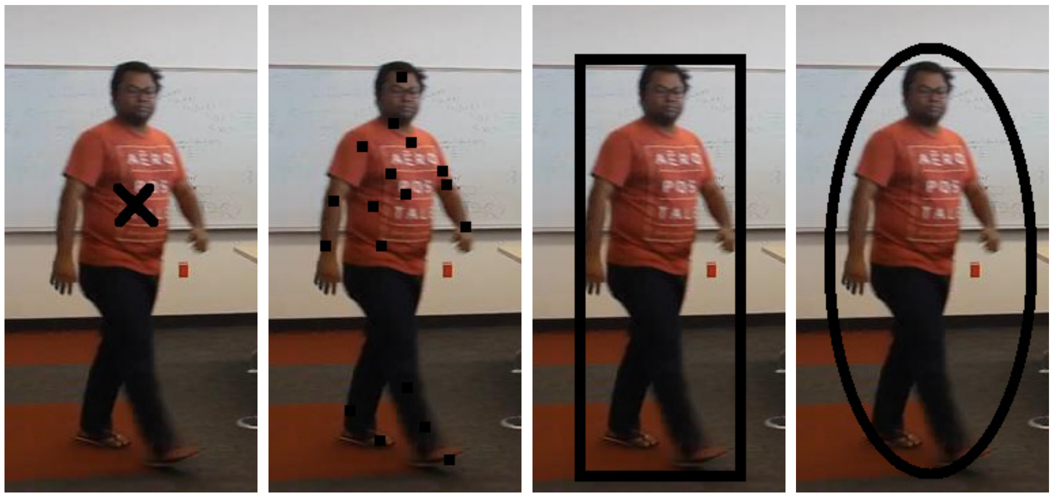


Figure 4.1: Figures from left to right represent various object representation (a) centroid, (b) set of points, (c) geometric shape - rectangle and (d) geometric shape - ellipse respectively

2. **Primitive geometric shapes:** Object is represented as a geometric shape that covers the skeleton of the object. Generally used shapes are rectangle (Figure 4.1 (c)), eclipse (Figure 4.1 (d)).
3. **Object silhouette and contour:** Contour representation is drawing the exact outline of an object (Figure 4.2 (e)). Silhouette is the area to be filled inside the outline of an object (Figure 4.2 (f)).
4. **Articulated shape models:** Articulated objects representation is representing each parts of the object as connected components (Figure 4.2 (g)).
5. **Skeletal models:** Object skeleton can be extracted by applying medial axis (Figure 4.2 (g)) transform to the object silhouette [Ballard and Brown 1982, Chap. 8].

Points representation can be used to track objects in video streams but they are not suitable if the object represented by a point has to be compared with another object for similarity (as the entire object pixels are needed for comparison). Similarity comparison is part of the work-flow in our algorithm (explained in the Chapter

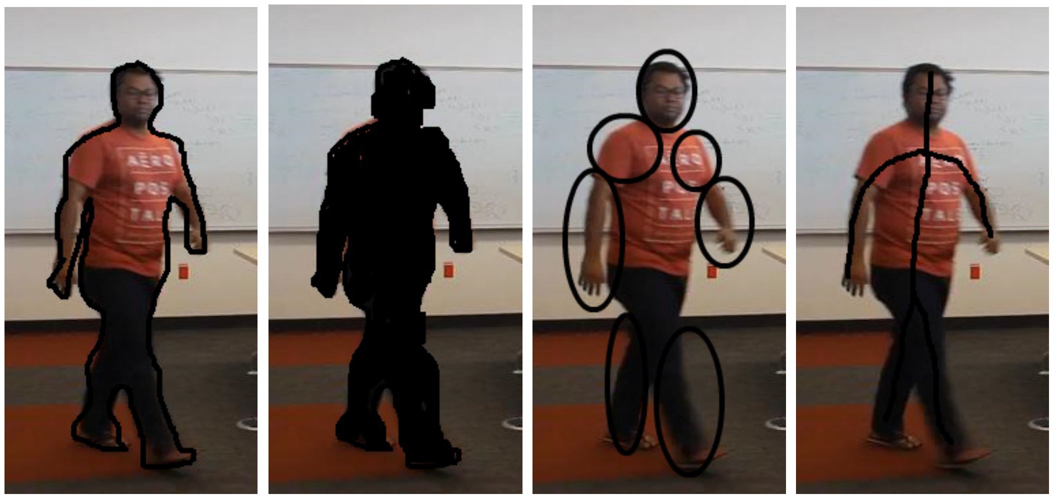


Figure 4.2: Figures from left to right represent various object representation (e) contour, (f) silhouette, (g) articulated and (h) medial axis respectively

6). We have chosen the *Primitive geometric shapes* representation with rectangle as the shape to denote an object of interest in our work. This representation is the simplest representation amongst all the available representations and is suitable to our requirements.

4.2 Moving Object Identification

A straightforward and most widely used method to identify moving objects in video streams is to calculate frame differences [16]. In frame differencing method, pixels related to moving objects in a video frame are obtained by selecting the minimum difference between the following two components. The first component is generated as a result of subtracting a frame F_i wherein the objects need to be identified with the frame F_{i1} that has lesser video time stamp compared to F_i. (need not necessarily be the previous frame.) Similarly, a second component is generated as a result of subtracting F_i from a frame F_{i+1} that has a greater video time stamp value. The difference calculated here is the absolute difference between grey scale converted images

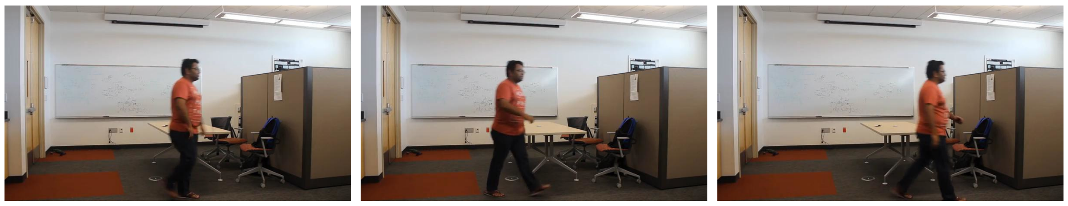


Figure 4.3: Object's position in frame 1030, 1010 and 1050, respectively from left to right

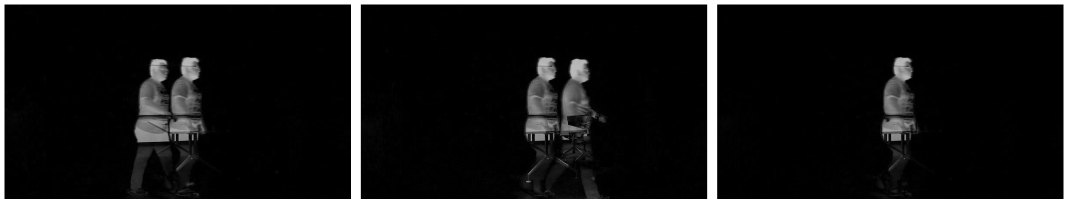


Figure 4.4: Figures from left to right represent the difference between frame 1030 and 1010, difference between frame 1030 and 1050 and motion of object in frame 1030 respectively.

of the frames to be subtracted. Equation 4.2 is the mathematical expression of the above mentioned method.

$$\begin{aligned} diff_1 &= |F_i - F_{i-1}| \\ diff_2 &= |F_i - F_{i+1}| \\ motion &= min(diff_1,\ diff_2) \end{aligned} \tag{4.2}$$

Figure 4.3 and 4.4 illustrate the moving object identification through frame differencing wherein a man is moving from left to right in the video. In Figure 4.3 first figure from left is the frame where the object need to be identified (F_i=1030), second figure from left is a frame that has lesser video time stamp (F_{i-1}=1010) compared to first and third figure from left is a frame that has greater video time stamp (F_{i-1}=1050) compared to first. In Figure 4.4 first figure from left is the pixels corresponding to the movements of the object in frame 1030 and 1020 ($diff_1$), second figure from left is the pixels corresponding to the movements of the object in frame

1030 and 1040 ($diff_2$) and third figure ($motion$) from left represents only the pixels corresponding to the moving object in frame 1030.

In the first figure from left in 4.4, pixels related to only the moving objects in frame 1020 and 1030 are visible as the background is static. In other words, as mentioned above the frames F_i and F_{i-1} are the gray scale converted images of the original frame and the pixel values in their image matrix will be between 0 and 255. Since the pixels corresponding to background in both these frames do not change, the resulting difference between these frames will have intensity value of 0 for the pixels related to the background and a value between 0-255 for the pixels corresponding to the moving objects . Thus the background will appear as black in the difference image $diff_1$. Similarly, the background in the resulting image $diff_2$ is black due to the same reasoning.

Third figure from left in Figure 4.4 shows only the pixels related to the movement of the object in frame 1030 due to the following reasons. In $diff_1$ the area occupied by object in frame 1020 and frame 1030 will have pixel intensity greater than 0. Similarly, in $diff_1$ the area occupied by object in frame 1040 and frame 1030 will have pixel intensity greater than 0. The common area amongst $diff_1$ and $diff_2$ that has pixel intensity greater than 0 belongs to just the object in frame 1030. Thus, the resultant image ($motion$) of minimum operation between $diff_1$ and $diff_2$ has intensity greater than 0 for only pixels related to movement of object in frame 1030 as the pixels related to the movement of objects in frame 1020 and 1040 are 0 in either of the images $diff_1$ or $diff_2$.

4.2.1 Background Subtraction

In order to separate the identified object from its background, it is essential to know the position coordinates of the object relative to the frame (considering left bottom corner of the frame as (0,0)) and the area occupied by the object in the frame. As discussed in the Section 4.1.2 we would be representing an object by a minimum rectangle that confines the outline of the object. This rectangle is called *bounding box*. A *bounding box* is represented as a vector of four values [x-coordinate, y-coordinate, length,width]. (x,y)-coordinates represent the position co-ordinates of the object, length and width represent the dimensions of the rectangle.

There are several algorithms to determine the area occupied by the object in the frame. One of the most used algorithm is *Connected Component Analysis* [15]. *Connected Component Analysis*[2] uniquely labels the connected regions. In the case of images, regions are denoted by collection of pixels. The resultant image (*motion*) of frame differencing contains only the pixels related to the object movement thus forming a region (connected component) that denotes the area occupied by the moving object in the frame. Thus, *Connected Component Analysis* can be applied on *motion* image to identify the region occupied by the moving object. But *Connected Component Analysis* cannot be directly applied on *motion* as it expects the elements forming a region to have similar values (same pixel intensity) whereas, *motion* is a Grayscale image and has the pixel intensity values between 0-255. Thus, in order to apply the *Connected Component Analysis* algorithm on *motion* to identify the region occupied by the moving object, all the pixels corresponding to the object movement have to be converted to the same intensity value.

[2]Connected Component Analysis algorithm can be either 4-connected or 8-connected. In 4-connected all the pixels on the 4 edges of the target pixel are neighbours. In 8-connected all the pixels on the 4 edges as well as the diagonal pixels are neighbours. Please refer to the book [16] for more details on it.

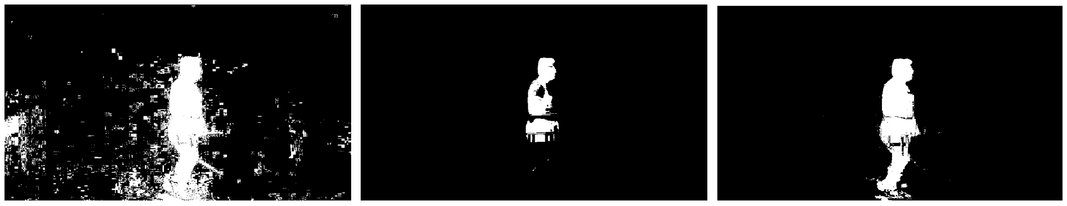


Figure 4.5: Figures from left to right represent the result of the filter for threshold values 10,100 and 30 respectively.

The result in the Equation 4.2 (*motion*) can be further filtered based on a threshold [3] to convert all the pixels related to moving object to the same intensity value. During the filtering process, all the pixels that have intensity value above the *threshold* are assigned an intensity value of 1 and rest as 0. Thus, the threshold value has to be chosen carefully so that almost all the pixels related to the moving objects get an intensity value as 1 and rest (background) of the pixels as 0. Additionally, this filtering process reduces background noise (minute background changes) present in *motion*. The threshold value is chosen through trial and error experiments. A low value of threshold may result in too much background noise (first figure from left in Figure 4.5 for threshold = 5) or a high value may remove some part of the region occupied by the object (second figure from left in Figure 4.5 for threshold = 80). The ideal threshold would be the one which reduces the background noise but retains all the area corresponding to the object movement (third figure from left in Figure 4.5 for threshold = 25). Additional details about choosing an appropriate value and recommended starting range of value for tuning the threshold for a given video is explained in the Section 7.2.1.

[3]The methods proposed in this thesis use several threshold values. These threshold values are calculated based on a random sample of the target video stream (the one on which the queries are run) or a video stream that is similar to the target video.

subsectionBounding Box Calculation The minimum rectangle ([x-coordinate, y-coordinate, length, width]) can be easily deduced for the labeled connected region (subset of pixels from the original frame) returned by *Connected Component Analysis* algorithm, by calculating the following components.

- x-coordinate: Lowest column index of the connected region.
- y-coordinate: Highest row index of the connected region.
- length: Difference between Highest row index and Lowest row index of the connected region.
- width: Difference between Highest column index and Lowest column index of the connected region.

The list of connected areas returned by *Connected Component Analysis* algorithm may have components corresponding to background noise (few of the noise remains even after filtering.) in addition to the components representing moving objects. But the connected area corresponding to the noise will be a lot smaller or bigger compared to that of the moving object. Our work eliminates the areas detected as noise through a min and max threshold value for the size (bounding box size) of the connected component detected. This threshold value should be calculated through experiments by observing the average size of the objects of interest in the video stream (or sample). This value varies for different video streams. Details on how to choose the min and max threshold for object size is given in the Section 7.2.3.

Once the bounding box of moving objects in a frame is calculated and most of the noise is eliminated, the detected moving objects can be extracted from the frame to separate from the background by superimposing (cropping) the bounding box coordinates on the original color image of the frame. The next chapter explains in detail about how to extract important features from these extracted object images for their comparison.

CHAPTER 5

Feature Extraction

To answer most of queries about object tracking in videos, it is necessary to classify (or group) identified moving objects in each frame as either a new object or an object that has appeared in the past frames. In other words, the same moving object in different frames must be given the same group label. This can be achieved by finding similarity between the object image (the cropped moving object image from a frame) to be labeled with the already labeled object images appeared in the past frames. Original color image data of the objects will be a three dimensional matrix. During labeling, all the object images appeared in the past frames may be required to be loaded into main memory for the comparison. This may result in huge amount of data in the main memory as each of these images are three dimensional matrices. In general, classification algorithms tend to give out less accurate results in case the amount of data is huge or is of high dimensionality. Such phenomenon is knows as "curse of dimensionality" [18]. There are several dimensionality reduction techniques to overcome this problem [19]. The dimensionality reduction techniques available for image data are called *feature extraction* algorithms. Feature extraction algorithms extract abstract data from the original image data by applying transformation on them. The resulting data has dimensionality reduced compared to original data by retaining only the important information that is sufficient enough to represent the object information. These important information are called *features*.

Features can be anything like peaks in a mountain image, doorway, corners, color of the objects in image etc. These features are called key-point features or interest points. Another class of important features are edges. Edges can be grouped into longer curves and straight line segments [16]. Once the important points (or information) are detected in an object image, each of these points are represented using descriptors. Collectively, all the descriptors extracted from an image are called *feature vectors*. There are several feature extraction algorithms available in Computer Vision that detect important features and extract descriptors from them. Few of such algorithms are *Histogram of an Image*, *Scale Invariant Feature Transform (SIFT) / Speeded Up Robust Features (SURF)*, *Hough Transform*, *Histogram of Oriented Gradients (HOG)* etc. These algorithms are applied based on the type of objects in the image. In this thesis, we deal with videos that contain moving objects of type humans or cars. A feature extraction algorithm that can be used to compare human images is not likely to be suitable for car images. The appropriate feature extraction algorithm for a given type of object need to be determined through experiments. Following are the experimental details conducted for identifying feature extraction algorithms suitable for each kind of object.

5.1 Feature Extraction Algorithm Selection

Following are the definition and details of few of the important components used in the experimentation. These definitions will be referred in rest of this report.

5.1.1 Definitions

Definition 5.1.1. Object Notation: Methods described in the Chapter 4 are used to identify and extract (crop) moving object images in every frame of the target video stream. Once the object images are extracted from frames, they are denoted as $Obj_{<i,j>}$. Where ‘j’is the incremental id given to every moving object detected (the first moving object detected in the video stream will have id as 1). ‘i’is the frame number in which this object was identified. Extracted (cropped) object image from its frame is a color image which is mathematically represented as three dimension matrix with each dimension corresponding to either red, green or blue color planes (Refer Definition 4.1.1 for more details on the color image mathematical representation)

Definition 5.1.2. Object Instances: The extracted object images of the same object in different frames are called *Object Instances*. All the object instances of an object belong to the same group/class (Each group denotes a unique moving object in the video stream.).

5.1.2 Details of the Feature Extraction Algorithm Used in the Experiments

5.1.2.1 Histogram:

Histogram of image is an example of statistical feature. Histogram of a Grayscale image (Assuming image has 8-bit depth intensity values.) will contain 256 bins with each bin corresponding to one of the possible intensity values between 0-255. Each bin in the histogram represents the count of the pixels in the image with that intensity value. Example of a Grayscale image and its corresponding histogram plot is given in the Figure 5.1 (x-axis represents the different bins (or intensity values) and the y-axis is the count of pixels with that intensity value in the image.). On the other hand color histogram consists of three histogram of 256 bins, for each of the color channel red, green and blue. An example of color image and its combined histogram of all the color channels is given in the Figure 5.2.

A straightforward method to calculate similarity between the color histograms of two different images is to calculate the distance between them. If H1 is the histogram matrix of an image *image1* and H2 is the histogram matrix of an image

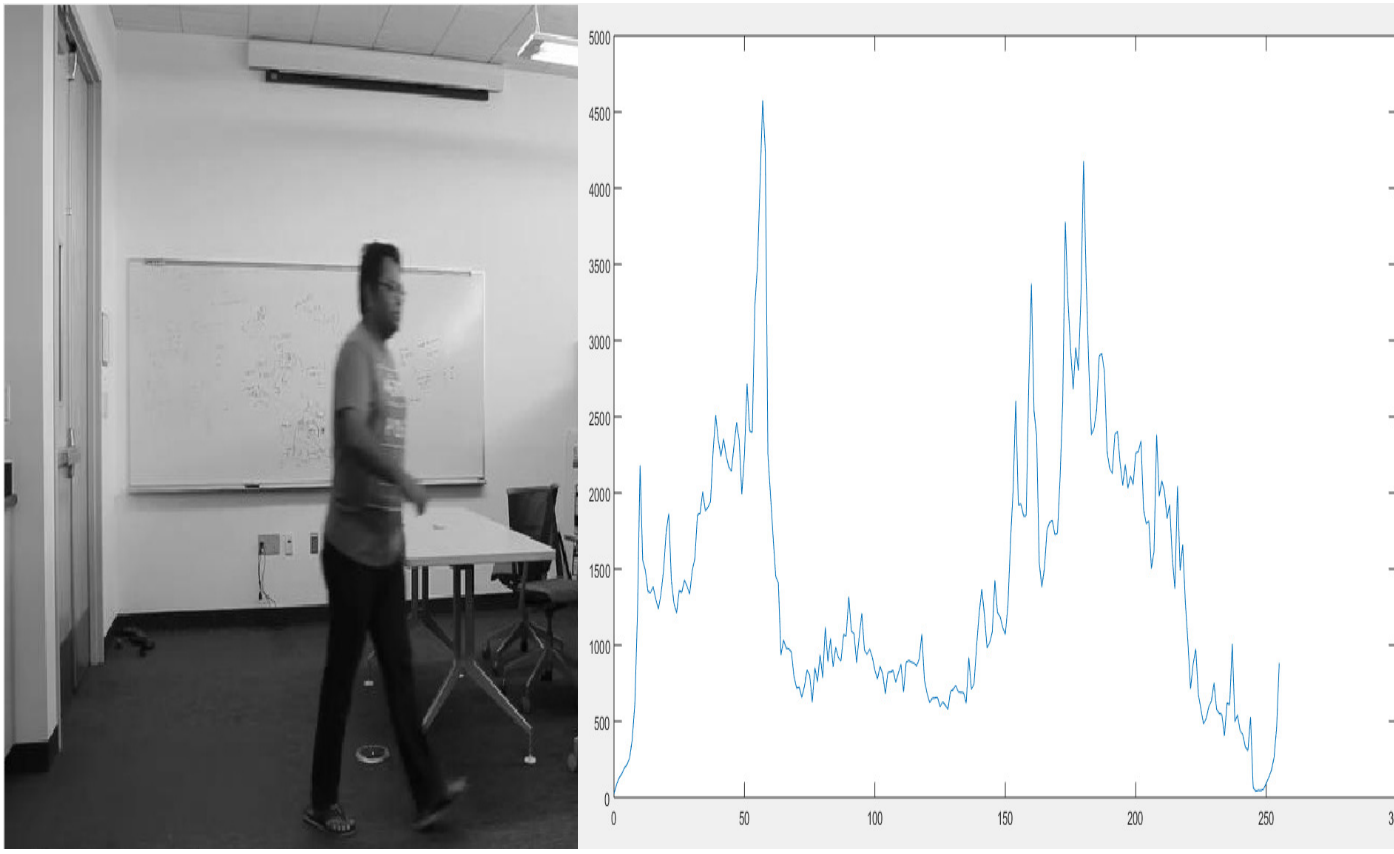


Figure 5.1: Figures from left to right represent the (a) Grayscale image and its (b) histogram respectively

image2 then the distance between them can be calculated as mentioned in Algorithm 1. *HistRed1*, *HistGreen1* and *HistBlue1* are the normalized histogram of the channel red, green and blue respectively in H1. Similarly, *HistRed2*, *HistGreen2* and *HistBlue2* are the normalized histogram of the channels green and Blue respectively in H2. *HRed*, *HGreen* and *HBlue* are the sum of squared difference between the each bin values in red, green and blue channel histogram of H1 and H2. And the final distance *H* is calculated using the *Luminance* color to Grayscale conversion formula in [20].

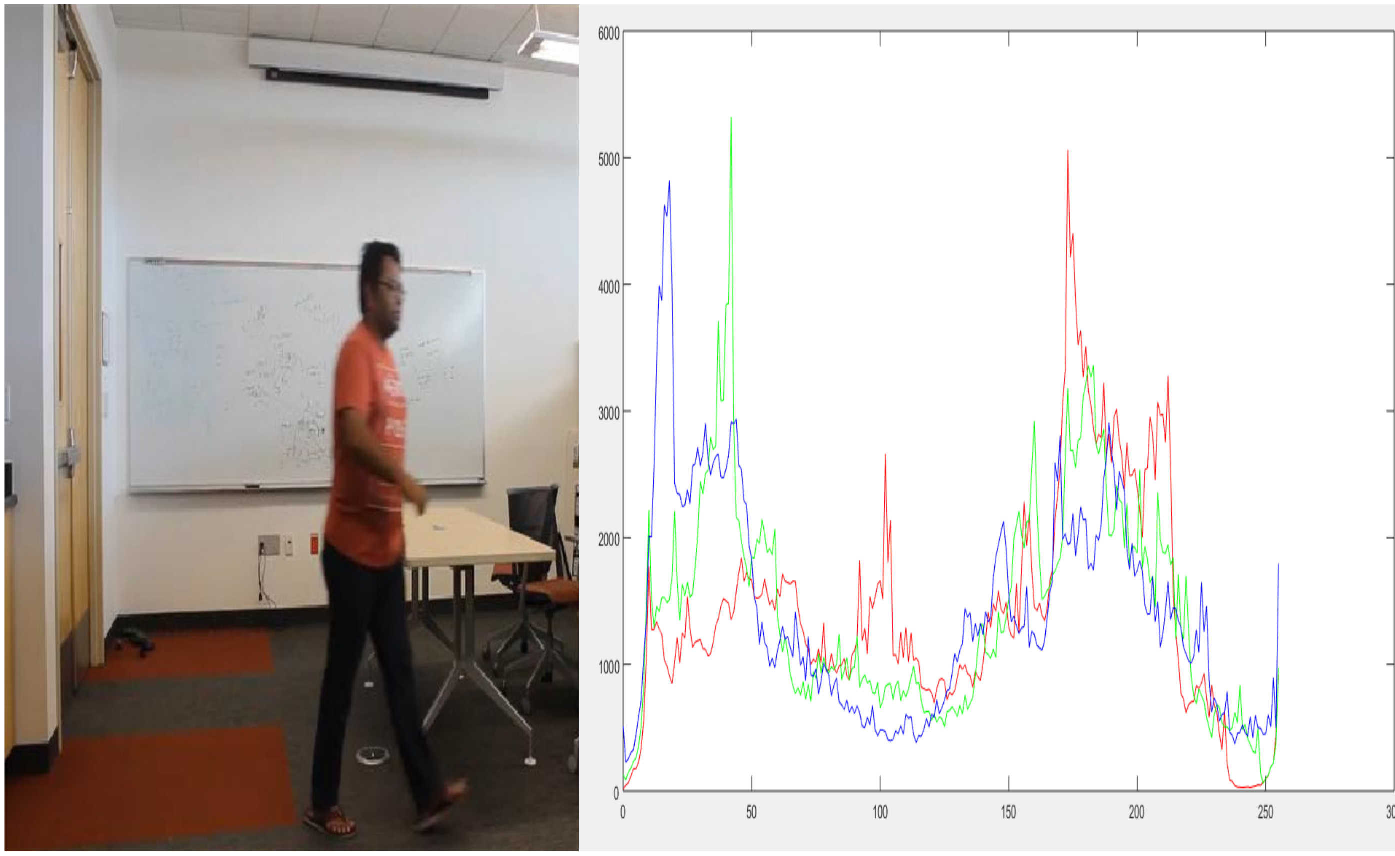


Figure 5.2: Figures from left to right represent (a) color image and its (b) histogram respectively

5.1.2.2 Scale Invariant Feature Transform (SIFT):

SIFT [21] algorithm is a *Feature Extraction* algorithms that detects important key-points (features) in the given image and provides the descriptors for each of these detected points (collectively all the descriptors extracted from an image are referred as feature vectors). The key-points detected can be edges, illumination points etc in the image. SIFT can detect image of an object varied in scale, illumination etc. by using the descriptors extracted from them. For example, SIFT can be used to detect a given object in a cluttered scene (the given object image is present in the target (cluttered) image in reduced scale.).

The feature vector (cumulative feature descriptors) extracted from the images is a matrix of dimension [MxN] where M is the number of key points detected and N is the size of the each feature descriptor extracted. The number of key-points detected varies from image to image. In most of the implementations of SIFT, feature descriptor size (N) is 128. Each of these feature descriptor is orientation histogram created for 4x4 neighbour pixels of the key-point with 8 bins each. Which makes the each descriptor a vector of size 4x4x8 = 128 [22]

If two images are said to be similar if the the ratio between the number of SIFT feature descriptors that matched amongst the images to the total number of descriptors obtained from one of the above two images is greater than or equal to a threshold (Refer to the Section 5.1.4 for details on choosing threshold value.). If M1 is the number of feature descriptors in the first image, M2 is the number of feature descriptors in the second image and K is the number of feature descriptors that matched between the images, then the similarity value can be expressed as in the Equation 5.1. A feature descriptor D1 in first image matches with descriptor D2 in second image, if the distance between them multiplied by 1.5 is not greater than distance of D1 to all other descriptors in the second image (Please refer to [21] and [22] for additional details about SIFT descriptor comparison.). Figure 5.3 illustrates the matching of key-points between two images using SIFT.

$$Similarity = \frac{K}{M1} \tag{5.1}$$

5.1.3 **Feature Vector Extraction Algorithm Selection Experiments:**

The experiment involves the following steps.

1. Extracting moving objects from every frame of the video using the techniques mentioned in Chapter 4.

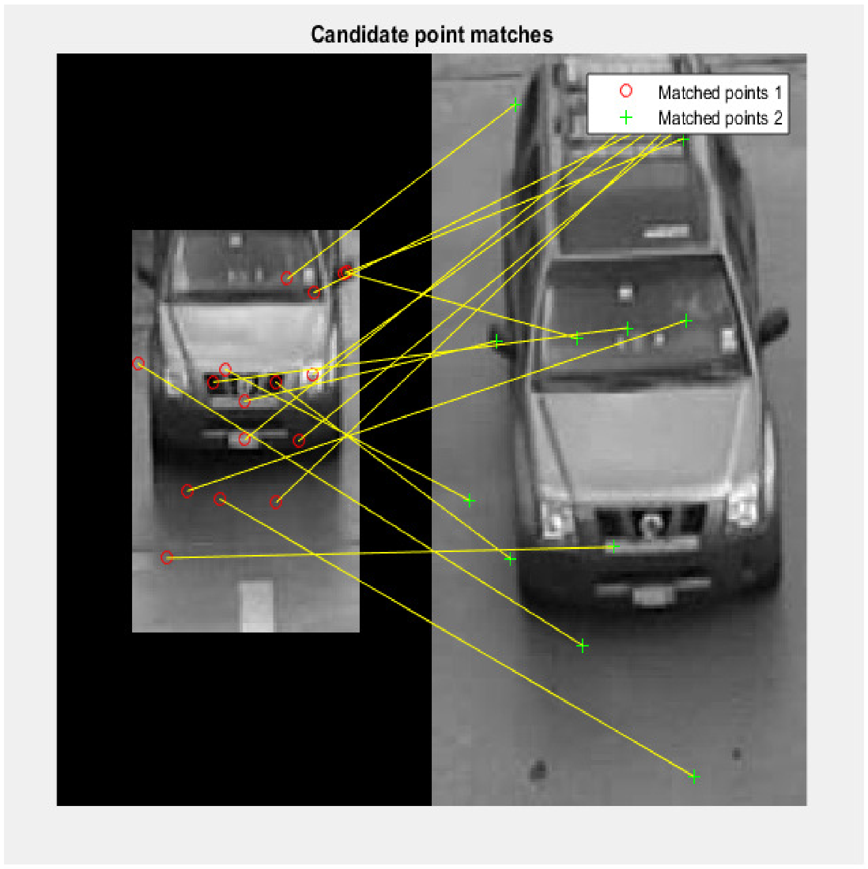


Figure 5.3: Illustration of SIFT key-points match between two images

2. Applying a feature extraction algorithm on each of these identified objects,
3. Comparing these feature vectors (instead of the original image data) to group object instances (*object instances* are images of the same object in different frames) together, and
4. Calculating the accuracy of matching.

For a given video with a specific type of objects (humans or cars) in it, suitable feature vector extraction algorithm is the one that gives better accuracy. Accuracy is calculated as shown in the Equation 5.2.

$$\frac{Number\ of\ objects\ grouped\ correctly\ using\ the\ feature\ vector\ comparison}{Total\ number\ of\ objects\ identified} \tag{5.2}$$

If $Obj_{<i,j>}$ is an object in frame F_j, Obj_1 is its group, FV_i is its feature vector and S is a set that represents different instance images of Obj_1 extracted from various frames. S_{FV} is a set that contains feature vectors corresponding to every object instance in set S. Then, in order to add $Obj_{<k,l>}$ found in frame F_l ($l¿i$) to set S, the distance between any one of the instances in S_{FV} and feature vector FV_k of object instance $Obj_{<k,l>}$ should be less than the threshold value.

Object $Obj_{<k,l>}$ in frame F_l is said to be correctly classified if the group label assigned to it is the same as the label in ground truth. In other words if the ground truth also says that the object instance $Obj_{<k,l>}$ belongs to the group Obj_1. Ground truth for the experiment is formed by scanning every frame in the video and assigning the same label to different instances of an object that are present in different frames.

5.1.4 **Threshold Calculation for Object Comparison**

As mentioned in the Section 5.1.3, threshold distance value decides if given two object images belong to the same group or not. Threshold value may vary based on the type of objects being compared or feature extraction algorithm being applied. Threshold value is calculated by experimenting with the range of possible distances (or ratio (0-1) in case of SIFT). Initially a random threshold distance is chosen and the experimentation steps under the Section 5.1.3 are followed to calculate the accuracy (Equation 5.2) of grouping (or classification) using this threshold. Similarly, experimentation steps are repeated for various values of threshold to see if the accuracy is improving. Final threshold value will be the value which gives the best accuracy.

Details of how to choose an appropriate initial range of values for threshold to reduce exhaustive experiments is mentioned in the Section 7.2.4.

5.1.5 Handling Induced Errors in Accuracy Calculation

In the process of accuracy calculation of the object classification as per Equation 5.2, if an object instance is mislabeled in a frame, that may lead to an incorrect assignment of labels to the instances of the same object in future consecutive frames. This is due to the fact that consecutive object instances of an object are most likely to have similar feature vector. For example, if object instance $Obj_{<k,l>}$ found in frame F_l was assigned an incorrect label, then the object instances that match $Obj_{<k,l>}$ in the consecutive future frames will also be assigned incorrect group labels. For such cases consecutive errors are ignored and considered as correct classification and only one error at frame F_l is counted towards accuracy calculation.

5.1.6 Description of Test Video Sets

Following are the descriptions of video sets used for experiments:

1. *Video 1*: Aerial view of people walking in an atrium (Video-1 in Figure 5.4). This is a sample video bundled with MATLAB installation. The human images extracted in this video will have *low resolution* as the video is captured from far.
2. *Video 2*: Ground view of people entering inside a room (Video-2 in Figure 5.4). This is a video recorded at our lab. The human images extracted in this video will have *higher resolution* as compared to Video-1.
3. *Video 3*: Ground view of cars moving on the road with camera placed far away from the road (Video-3 in Figure 5.4) [23]. The car images will have *low resolution* due to the far placement of the camera.

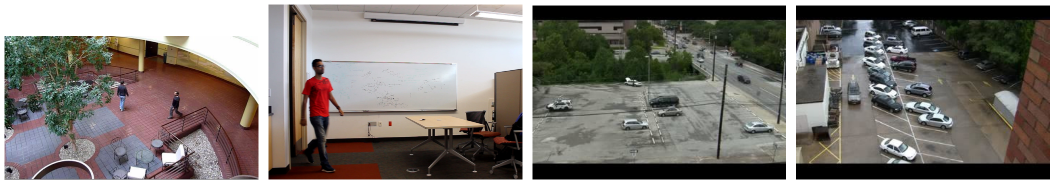


Figure 5.4: Test videos: Video-1, Video-2, Video-3 and Video-4 respectively from left to right

4. *Video 4*: Ground view of cars entering and leaving the parking lot (Video-4 in Figure 5.4) [24]. The car images in this video have *higher resolution* as compared to Video-3.

5.1.7 **Results of the Experiments on Feature Extraction Algorithms**

Various feature vectors were applied on the test videos to compare objects using threshold values and following are some of the conclusions of our experiments:

1. From Table 5.1, it is clear that SWIFT did not fare well for objects which do not have edges or pointy structure (like humans). The algorithm was not able to extract enough important feature points to perform object comparison of human objects. However SWIFT was able to successfully classify instances of cars.
2. SWIFT/SURF failed for objects which are smaller than 60x60 pixels (low resolution moving object images)
3. Color Histogram was able to successfully classify human object instances.

5.1.8 **Additional Feature Extraction**

Since our goal is to answer large class of queries, it is useful to extract additional information for the objects identified so that they can be used as needed. Additional information we have extracted are color of the object, size of the object (bounding box), location of the object relative to the video frame, direction of movement of the object etc.

Algorithm 1 Histogram Comparison

$$HistRed1 = \frac{H1[1][1] + H1[1][2] + ... + H1[1][256]}{256}$$
$$HistGreen1 = \frac{H1[2][1] + H1[2][2] + ... + H1[2][256]}{256}$$
$$HistBlue1 = \frac{H1[3][1] + H1[3][2] + ... + H1[3][256]}{256}$$

$$HistRed2 = \frac{H2[1][1] + H2[1][2] + ... + H2[1][256]}{256}$$
$$HistGreen2 = \frac{H2[2][1] + H2[2][2] + ... + H2[2][256]}{256}$$
$$HistBlue2 = \frac{H2[3][1] + H2[3][2] + ... + H2[3][256]}{256}$$

$$HRed = \sum_{i=1}^{256} |HistRed1[i] - HistRed2[i]|^2$$
$$HGreen = \sum_{i=1}^{256} |HistGreen1[i] - HistGreen2[i]|^2$$
$$HBlue = \sum_{i=1}^{256} |HistBlue1[i] - HistBlue2[i]|^2$$

$$H = 0.2989 * HRed + 0.5870 * HGreen + 0.1140 * HBlue$$

Video	Histogram	SIFT	Conclusion
Video-1: Low resolution people objects	98.07	N/A	SIFT was unable to extract enough points due to small image size of moving objects
Video-2: High resolution people objects	90.1	82.2	Histogram gave better accuracy for human type of objects comapred to SIFT. Thus, for human type of objects Histogram is better
Video-3: Low resolution car objects	skip (most of the cars have similar color and histogram cannot detect differences in shapes)	N/A	SIFT was unable to extract enough points due to small image size of moving objects
Video-4: High resolution car objects	skip (most of the cars have similar color and histogram cannot detect differences in shapes)	70.6	SIFT gave better accuracy for car type of objects as compared to histogram. Thus, for car type of objects SIFT is better

Table 5.1: VS1:Experimental results to determine which feature extraction algorithm is suitable for what type of objects

CHAPTER 6

Object Labeling Algorithm and Modeling Extracted Data

Labeling (or classification or grouping) techniques are applied for object tracking. In order to track an object throughout the video stream, every instance of this object must be identified and labeled same in the upcoming frames. Several algorithms, such as Bayesian classifier or k-NN classifier [25], are available for classifying a set of data (e.g. images, text etc.) into equivalent groups. This thesis has used a variation of the 1-NN classifier [25] to label object instances of a given object.

k-NN classifier in general is a non parametric classifier. k-NN is used to classify a given data item to one of the known classes. k-NN requires training data (which is essentially a set of data of which class labels are known) to build a classification model. Whenever k-NN has to assign a class label to a given item, it compares the similarity of the given item with the items present in the training data and assigns class label of the given item as the majority class label amongst the k items from training data that are similar to the given item. In 1-NN classifier, k is equal to one. The labeling algorithm used in this thesis is different from the standard 1-NN algorithm in the training data used. The labeling algorithm used in this thesis does not require any training data beforehand. The first moving object instance identified in the given video stream is given group label 1 and this becomes the training data to start with. This object instance will be added to the first group formed above. The second moving object instance, found after labeling the first instance, is compared with the first instance. If the second object instance is similar to the first one, then it is assigned the same group label as the first one. Else it is given a new incremental (if previous was 1 then the next will be 2) group label. This process is repeated for every moving object instances found thereafter. Thus, the classes (or groups) are determined on the fly unlike in 1-NN.

Tracking algorithm developed in this thesis uses a set of metrics to check the simialrity between object instances. The order followed by these metrics are illustrated in Algorithm 2 for the classification of object instance $Obj_{<i,j>}$ found in frame F_j. We first apply Metric 3 to label an object instance and if it fails (returns no group label), Metric 2 is used. Similarly, if Metric 2 fails, Metric 1 (exhaustive approach) is used. This order improves the efficiency of pre-processing of video frames.

Algorithm 2 Object Classification Algorithm

```
1: if $metric3(Obj_{<i,j>}) \neq NULL$ then
2:     $group(Obj_{<i,j>}) = metric3(Obj_{<i,j>})$
3: else
4:     if $metric2(Obj_{<i,j>}) \neq NULL$ then
5:         $group(Obj_{<i,j>}) = metric2(Obj_{<i,j>})$
6:     else
7:         $group(Obj_{<i,j>}) = metric1(Obj_{<i,j>})$
8:     end if
9: end if
```

6.1 Metric 1: Use equivalence classes for Similarity Comparison

This mtric states that an object instance identified in the current frame can be labeled by calculating the similarity between it and every object instance from the past frames that are already classified and assigned group labels. If any of the instances in past frames satisfies the threshold (Refer to the Section 5.1 for more details on threshold.) with the object instance being compared, then the object instance in the current frame is given the same group (or class) label as the instance that satisfied the threshold. Similarity measure used in this thesis to compare object instances is the Euclidean distance between the feature vectors extracted from object instances(Refer to the chapter 5 for more details on feature extraction and their comparison.).

Metric 1 is exhaustive; it compares a new object instance with all instances in the previously formed groups until a similar instance is found. Metric 2 improves efficiency by reducing the number of comparisons.

Let us assume $Obj_{<i,a>}$, $Obj_{<i,b>}$ and $Obj_{<i,c>}$ are the object instances identified in frame F_i and are already classified and given groups labels as Obj_1, Obj_2 and Obj_3 respectively. $Obj_{<j,d>}$, $Obj_{<j,e>}$ and $Obj_{<j,f>}$ are the object instances identified in frame F_j. If we assume S is a set of object instances that are already assigned with group labels, then the group label of $Obj_{<j,d>}$ is obtained by comparing it with with the object instances in S. If the object instance $Obj_{<j,d>}$ does not satisfy the threshold with any of the instances in the set S, then a new incremental group label will be assigned to the object. Algorithm representation of Metric 1 is shown in Algorithm 3.

6.2 Metric 2: Use closest frame first for Similarity Comparison

In most of the cases, consecutive instances of an object will have similar feature vectors. This metric uses this information to reduce the number of comparisons needed to label an object in the current frame by restricting the comparisons to just the object instances in the previous frame. This metric will fail if the object instance

Algorithm 3 Metric-1

```
1: for $group$ in $all\_group$ do
2:     for $instance$ in $List\_Of\_Instances(group)$ do
3:         if $Euclidean\_distance(instance, Obj_{<j,d>}) < threshold$ then
4:             $group(Obj_{<j,d>}) = group(instance)$
5:             $break$;
6:         end if
7:     end for
8: end for
9: if $group(Obj_{<j,d>}) =$ NULL then
10:     $group(Obj_{<j,d>}) = last\_group\_label + 1$
11: end if
```

to be compared is an instance of the object that has appeared newly in the video. If this metric fails, then Metric 1 can be used to label.

The algorithm in listing 4 illustrates assignment of group label for object instance $Obj_{<i+1,d>}$ found in frame F_{i+1} by comparing the similarity with the object instances of the previous frame F_i.

6.3 Metric 3: Use of Object Displacement Across Frames

This suggests an alternative to label an object instance by calculating the displacement of the object instance from the instances in previous frame. This eliminates the calculation of the Euclidean distance between the object instance feature vectors. Displacement of an object across continuous frames can be measured with the help of the bounding box. Bounding box is expressed as a group of four values (rectangle coordinates: x, y, width,height). x, y represent the position pixels of the object (left-

Algorithm 4 Metric-2

```
1: for $instance$ in $List\_Of\_Instances(frame - F_i)$ do
2:     if $Euclidean\_distance(instance, Obj_{<i+1,d>}) < threshold$ then
3:         $group(Obj_{<i+1,d>}) = group(instance)$
4:         $break;$
5:     end if
6: end for
7: if $group(Obj_{<j,d>})$ = NULL then
8:     $group(Obj_{<j,d>}) = metric1()$
9: end if
```

bottom corner of the rectangle) with respect to the frame. Displacement is calculated as the x-coordinate and y-coordinate difference between the object instances to be compared. The object instance to be compared is assigned the same group label as the object instance that is being compared with if both of the difference values are less than the threshold value.

In Metric 3 the threshold value is determined through experiments. Refer to the Section 7.2.5 for more details on the displacement threshold.

The algorithm- 5 explains how to determine the group label of $Obj_{<i+1,d>}$ by calculating bounding box displacement with object instances in the previous frame F_i. If $Obj_{<i+1,d>}$ does not satisfy the threshold value with any of the object instances in the previous frame, then Metric-2 is used.

6.4 Expressing similarity in terms of probabilistic measure

The above metrics can also be used to answer queries that involve comparing object instances in one video stream with another. But to answer some specific

Algorithm 5 Metric-3

```
1: for $instance$ in $List\_Of\_Instances(frame - F_i)$ do
2:     if $min(absolute(BoundingBox\_x - coordinate(Obj_{<}i + 1, d >)$
3:     $BoundingBox\_x - coordinate(instance)),$
4:     $absolute(BoundingBox_{y-coordinate}(Obj_{<}i + 1, d >)-$
5:     $BoundingBox_{y-coordinate}(instance))) < threshold$ then
6:         $group(Obj_{<i+1,d>}) = group(instance)$
7:         $break;$
8:     end if
9: end for
```

queries, similarity needs to be determined as a percentage. For example, to "Find everyone in video stream-2 who is dressed almost similar to person-1 (present in video stream-1)" an 80% confidence can be used. For these types of queries, expressing the similarity in terms of matched or not is not sufficient as it is expecting match in terms of percentage accuracy. This can be addressed by expressing similarity in terms of probability (or percentage).

One way to calculate the probability percentage is through Bayesian classifier. If *score* is the Euclidean distance between the feature vectors of the object instances being compared, then the probability model can be expressed as:

$$P(correct/score) = [P(score/correct) * P(correct)]/P(score) \tag{6.1}$$

P(correct/score) is the probability of the instances being compared are same (i.e. they belong to the same object), given the score. P(score/correct) is the ratio of number of times the comparison resulted in this score and was also classified as *match*, to the total number of times this score has occurred throughout the video segment as result of object comparison. P(correct) is the ratio between the number times the comparison resulted in a *match* to the total number of comparisons in the video segment. P(score) is the ratio of the total number of times the comparison has resulted in this score to total comparisons in the video segment. The video segment can be any sample in the video streams being compared and the *match* is determined as the comparison that results in a Euclidean distance lower than threshold. The values obtained for above probabilistic parameters using the video sample can be extrapolated to process rest of the video stream, as well as any video with similar background and object setup.

6.5 Modeling Extracted Data

Table 6.1 is the snapshot of the pre-processing output for the Video 1 in Figure 5.4. *fid* represents the incremental id given for each frame of the video starting at 1. *oid* is the unique object instance id given to every object instance detected in the video stream. If a given instance does not match with any instances in the past frames in Metric 1, then this object instance generates a new group and the group label given same as the object instance id. *glabel* represents the group label given for object instances after classification. *b_box* represents the bounding box of an object. *fv* is either a 3-by-256 vector (256 bins correspond to the pixel intensity range of 0-255, with one row for each channel of RGB) representing color histogram or a M-by-N vector corresponding to the feature vector of M important points identified, with each feature vector of length N representing SIFT. Note that this is an extended relational model as vectors are not supported in the traditional data model. The feature vectors are stored as object files for each of the tuples in the relational representation. A map between the an object instance id (every tuple has an object instance in it) and its corresponding feature vector. While processing a tuple, the feature vector corresponding to the object instance present in the tuple can be obtained by indexing (oid) on the map.

fid	oid	glabel	b_box	fv
79	73	1	[386 ,139 ,250 ,311]	[$fv1$]
80	74	1	[386 ,137 ,249 ,313]	[$fv2$]
81	75	1	[393 ,136 ,247 ,314]	[$fv3$]
82	76	76	[394 ,280 ,128 ,22]	[$fv4$]
83	77	76	[487 ,384 ,148 ,66]	[$fv5$]
84	78	76	[500 ,136 ,135 ,314]	[$fv6$]
85	79	76	[521 ,135 ,124 ,315]	[$fv7$]
86	80	76	[529 ,135 ,109 ,315]	[$fv8$]
87	81	76	[542 ,135 ,120 ,315]	[$fv9$]
88	82	76	[560 ,135 ,112 ,315]	[$fv10$]

Table 6.1: VS1:Relational Representation of the video stream Video 1

Action ‘$q1'$ on
Appear(c0.*,250)

Table 6.2: LVQL Query

The Video 1 in in Figure 5.4 captures side-view of people entering through the door one by one. It can be compared to the video stream which captures people entering a mall. Thus, Query 1(“How many distinct people are entering the mall throughout the day in each hour?”) can be evaluated on the data extracted from video stream Video 1 in Figure 5.4.

The Table 6.2 shows a partial LVQL query for expressing Query 1(“How many distinct people are entering the mall throughout the day in each hour?”). LVQL detects (using Appear() operator) all the dynamic objects in the video stream (c0) that are above 250 (assuming each human object is at least 250 pixels in size) and since all the humans in the video will have similar size, it does not identify different people uniquely. In order to identify and track people uniquely in LVQL, they need to be marked and each of the marked objects needs to be mentioned explicitly in the query using unique labels (like c2.s565b46). This entails manual intervention by users

SELECT	COUNT(DISTINCT glabel)
AS	Total_No_Of_People
FROM	vs1 [Range 12 hours Slide 1 hour]

Table 6.3: CQL Query

to *identify* objects. Alternatively, unique object tracking in LVQL can be achieved by writing code/program and developing new operators (certainly not *non-procedural.*)

On the other hand, Table 6.3 shows the *complete* CQL query for expressing Query 1 ("How many distinct people are entering the mall throughout the day in each hour?"). The rolling window specification used in the query ([*Range* 12 *hours slide* 1 *hour*]) goes over the rows in the Table 6.1 and groups them based on the glabel and outputs the aggregated count of unique glabel every one hour over the specified interval 8 AM - 8PM. Note that, the representation in the Table 6.1 has the time-stamp in terms of frame number, as the video stream used to extract this representation did not contain the meta-data related to the original time-stamp of the video stream. If such meta-data information is available, it can be included in the extracted data. However for the representation in Table 6.1, window definition in CQL will use the frame numbers for measuring time. Each second of frame has 29 frames, thus an hour long of window will have data corresponding to 104400 frames.

The query in Table 6.3 is not only simpler and easy to understand as compared to the one in Table 6.2, but grouping, counting, and aggregation are part of the language and no additional code is needed. The output of the query in Table 6.3 for the snapshot window shown in Table 6.1 will be 2 as there are only two unique labels in the glabel column. This computation will be done for each window. The query is answered based on the semantics of operators available in CQL and the users do not need to learn any new operators and do not require manual intervention to mark moving objects. Other queries can be expressed similarly.

Thus, it is clear from the above details that the representation created from the pre-processing techniques in this thesis can be used to evaluate simple situations using standard Non-procedural query languages like SQL and CQL.

CHAPTER 7

Object Labeling Algorithm Optimization and Experimental Analysis

in the case of failure of Metric 3 and 2, in order to assign group label to an object in the current frame, the object classification algorithm explained in the Chapter 6 compares the object instance to be labeled with every object instances from the past frames till a similar instance is found. Although this may not affect the processing time of the algorithm for a short video stream, it will increase the overall processing time of the algorithm for long videos. This chapter explains few of the methods proposed in this thesis to reduce the number of comparisons for Metric 1. The details of the methods are decsribed in the Section 7.1. Additionlly, previous chapters explained in detail about the various steps involved in pre-processing video streams to extract queryable representation from them. Lot of the parameters involved in these steps (like threshold value for object matching, what feature extraction algorithm to use for a type of object, minimum / maximum object size etc.) are determined through experiments. These parameters are referred as *key parameters* in this thesis. Varying the value of these parameters will have direct impact on the accuracy of the object classification algorithm. Goal of video pre-processing is to arrive at (tune) ideal values to these parameters so that maximum accuracy for object classification and tracking is achieved. But it is not feasible to try all possible ranges of values for these parameters to arrive at an ideal value. Section 7.2 in this chapter lists such essential parameters and proposes techniques to determine range of near ideal values for these parameters. Thus, reducing the effort in experimenting to find arrive at the ideal values. The details about choosing starting range of values for experiments of tuning key-parameters are explained in the Section 7.2.

7.1 Optimization of Object Classification Algorithm

This thesis has proposed two methods *Frame History Length* and *Selecting Group Representatives* for reducing the number of object instances in the object label groups. These methods are explained in the detail in the Sections 7.1.1 and 7.1.2.

7.1.1 Frame History Length

This method restricts the number of object instances to be compared to in Metric 3 to the ones from just K past frames instead of all the past frames. These K past frames are defined as *frame history length*. For example, if an object in frame N has to be classified and K is the frame history length, then for Metric 3 the object instance to be labeled should be compared with only the object instances in every group that belong to the frames from N-K to N.

If the video stream being pre-processed has objects going out of the video boundary and comping back later in time and when the object returns back if the previous object instances are removed due to them falling beyond frame history length range, the returned object will be labeled as a new object, reducing the object classification algorithm accuracy. Thus, a larger value of frame history length should be chosen such that it can retain the object instances of the previous appearance of an object that has returned. Thus this method to reduce number of instances in the object label group is not feasible.

7.1.2 Selecting Group Representatives

During the exhaustive comparison in Metric 3, if the object instance (let us assume it is found in frame F_i) being compared does not match with an object instance of a group Obj_k , then it may not match with any instances of group Obj_k that are derived from few immediate consecutive frames after or before F_i. Object instances derived from immediate consecutive frames after or before F_i and the object instance in F_i will have similar feature vector. Thus, it is not necessary to retain every object instance in a group. Only a few object instances can be retained per group throughout the course of the video stream as representatives. This will reduce the number of comparison involved in Metric 3. The method to choose representative instances is explained below.

Every time a group label is to be assigned to an object instance in the current frame, this object instance is compared (in the descending order of frame number or video time stamp) with the object instances already present in the group. If it matches with one of the already present object instances then it will be dropped. Else it will be retained in the group. The overall workflow of the object labeling algorithm after integrating 'selecting group representative'optimization is shown in the Figure 7.1.

7.1.3 Experimental Results and Analysis

The results and the analysis of the experiments conducted to quantify the improvements in number of similarity comparison involved in object labeling after introducing *selecting group representative* optimization are given in the Table 7.1

These experiments are run with object labeling work flow such that whenever an object instance need to be labeled, metric 3 is initially applied to be determine its group label. If the metric 3 fails, then metric 2 is used to determine the label. If

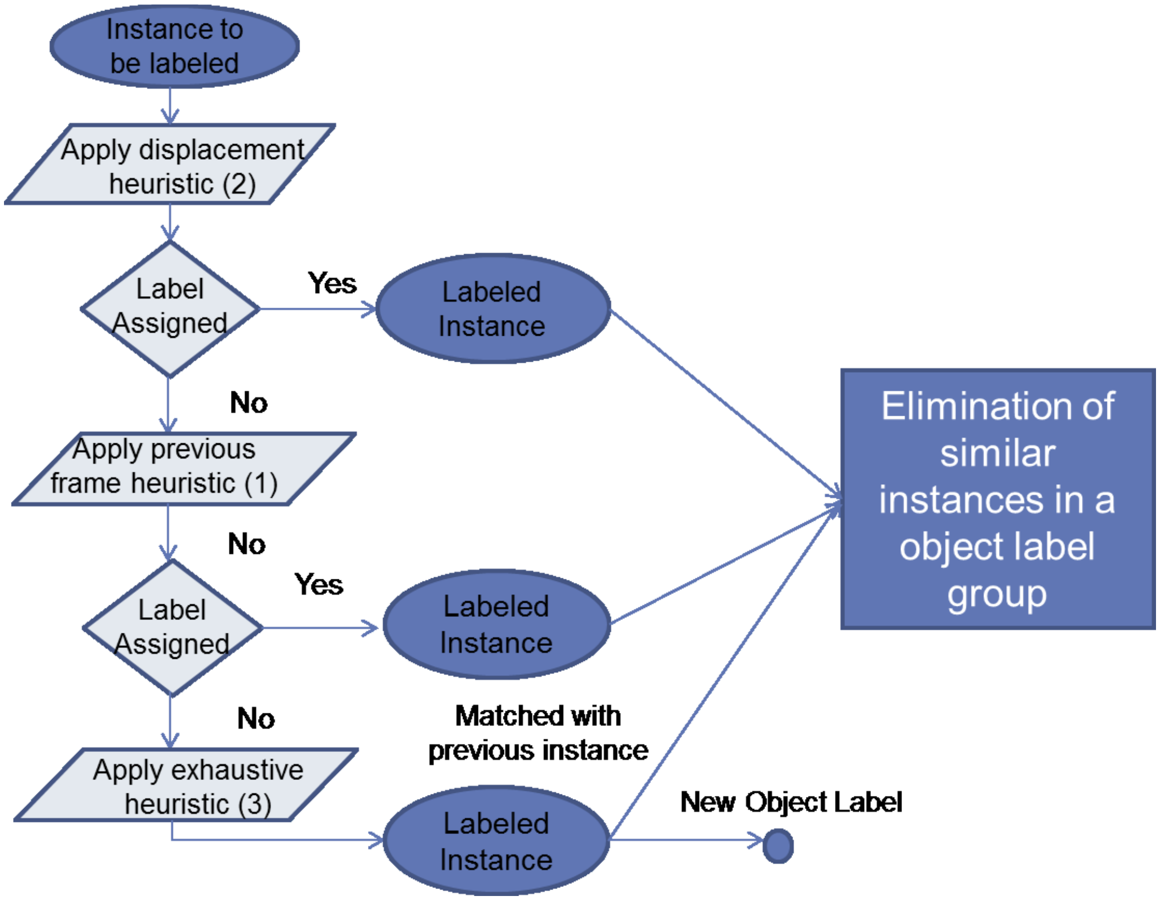

Instance to be labeled
Apply displacement heuristic (2)
Label Assigned
Yes
Labeled Instance
No
Apply previous frame heuristic (1)
No
Yes
Label Assigned
Labeled Instance
No
Matched with previous instance
Apply exhaustive heuristic (3)
Labeled Instance
New Object Label
Elimination of similar instances in a object label group

Figure 7.1: Object Labeling Algorithm Optimization Workflow.

metric 2 fails, then metric 1 is used to determine the object group label. Whenever an object instance is assigned a label, it is either added to the group label it belongs or it is discarded. The decision is based on the methods explained in Section 7.1.2. The videos used in theses experiments are Video 2 and 4 in Figure 5.4 that captures the people entering a room and cars entering parking lot respectively. The *OFF* value in *Optimization* column represents the execution of object labeling without the *selecting group representative* optimization. Similarly the *ON* value represents the execution of object labeling with the *selecting group representative* optimization.

Video Stream	Improvement	Number of Comparisons	Improvement in number of comparisons	Accuracy
Video 1 (People enter a room)	OFF	27044	N/A	96.99
Video 1 (People enter a room)	ON	6970	74.22%	96.8
Video 2 (Parking lot)	OFF	313	N/A	99.82
Video 2 (Parking lot)	ON	139	55.59%	99.82

Table 7.1: Effect of 'selecting group representative'optimization object labeling algorithm

7.2 Key Parameters for Video Pre-Processing Framework

Previous chapters explained in detail about the various steps involved in pre-processing video streams to extract queryable representation from them. Lot of the parameters involved in these steps (like threshold value for object matching, what feature extraction algorithm to use for a type of object, minimum / maximum object size etc.) are determined through experiments. These parameters are referred as *key parameters* in this thesis. Variation of these parameters will have direct impact on the accuracy of the object classification algorithm. Goal of video pre-processing is to arrive at (tune) ideal values to these parameters so that maximum accuracy for object classification and tracking is achieved. But it is not feasible to try all possible ranges of values for these parameters to arrive at an ideal value. Section 7.2 in this chapter lists such essential parameters and proposes techniques to determine range of possible values for these parameters that are near to the ideal value. Thus, reducing the effort in experimenting to find arrive at the ideal values.

Following are the key parameters in video stream pre-processing work-flow that need experimentation to determine ideal values for them. These parameters are identified and parametrized as input configurations (Refer the Chapter 8 for more details.)

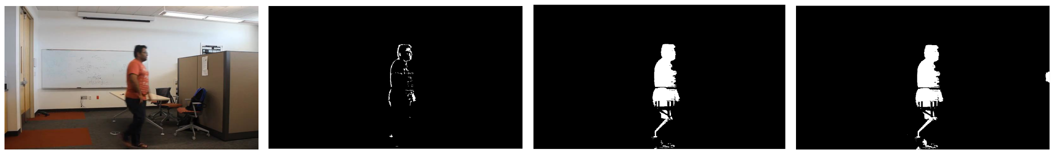


Figure 7.2: Figures from left to right represent the frame 1024, motion of the object in frame 1024 for offset 1, 20 and 124 respectively.

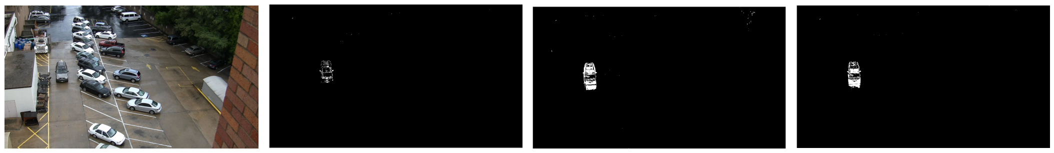


Figure 7.3: Figures from left to right represent the frame 303, motion of the object in frame 303 for offset 1, 20 and 124 respectively.

in the video pre-processing framework built as part of this thesis, so that effect of varying (tuning) values can be easily observed.

7.2.1 **Frame Difference Offset**

As mentioned in the Section 4.2 moving objects in a frame F_i is identified by frame differencing with a frame (F_{i-1}) that has lesser video time-stamp value compared to F_i, as well as differencing with a frame (F_{i+1}) that has greater video time-stamp value compared to F_i. *Frame difference offset* is defined (Mathematically represented as in the Equation 7.1.) as the number of frames between F_{i-1} and F_i or F_{i+1} and F_i.

$$offset = abs(F_i - F_{i-1}) \vee abs(F_i - F_{i+1}) \tag{7.1}$$

Figure second, third and fourth from left to right in Figure 7.2 show the effect of offset value in detecting moving objects (using the method mentioned in Section 4.2.1) in frame 1024 of a video where a person is walking from left to right (Video 1 in Figure 5.4). It can be observed that the movement pixels detected when offset is 1 does not cover all the pixels related the person's body skeleton. This is due to minimal movement of the whole body part across a single frame (from frame 1023 to 1024 or frame 1024 to 1025). Thus, a very low value of offset may not capture the complete skeleton of the moving object. Even though there is not much difference between the movement pixels captured by offset value 20 and 124, a very high offset value like 124 may not be suitable. A high value of offset will result in more waiting period in the starting of object detection for a video stream. In other words, for an offset value of 124, the video pre-processor has to wait till first 123 frames are captured to start detecting moving objects. So an intermediate value like 20 would be ideal.

Similarly, figures second, third and fourth from left to right in Figure 7.3 show the effect of offset value in detecting objects in frame 303 of a video where a car is entering the parking lot (Video 2 in Figure 5.4). However, in case of this video a very low offset is still able to capture the complete skeleton (motion pixels sufficient enough to draw skeleton). This is due to the adequate movement of the car across a single frame. Thus, if an object is moving fast, lower value of offset will still give good results.

Following are the conclusions drawn from the above experiments. The below mentioned values can be proposed to the users (attempting video pre-processing) as recommended values to start with for experimenting with offset value to reach an ideal value.

1. The near ideal value for difference offset depends on the speed of the object in the video stream. For a slow moving object like human walking, a good starting range of values for the offset would be 20 to 30. For fast moving objects like cars, a good starting range of values for the offset would be 3 to 10.

2. A high offset value will give good results but may increase the waiting period in the starting of object detection for a video stream.

7.2.2 Frame Hop-size

By default during video pre-processing every frame in the video stream is processed to check for any moving objects in them. For a video stream that captures a scene that has same moving objects staying for long time in the camera view, it is not necessary to process every frame as the same object will be repeated from many number of frames. For example, aerial view of a parking lot that has cars coming in or going out, but any car that starts moving will be in the vicinity of the camera for at least three minutes (assuming a parking lot that has long entry or exit road). So, skipping every two minutes worth of frames may not lead to loosing any new objects as the cars will be moving for at least three minutes. The number of frames that can be skipped without processing is called *hop size*. Hop size depends on the type of the video and objects and the users are expected to come up with appropriate values by observing small segment of the video stream to be pre-processed.

7.2.3 Upper and Lower Bound for Object Size

The output of the methods mentioned in the Section 4.2 may detect some of the noise as moving objects even after refining the results based on a threshold value. The object detection output for one of the frames in Video 1 in Figure 5.4 is illustrated at Figure 7.4. It can be observed from the output that there are many other tiny bounding boxes (black rectangles) apart from the bounding box of the object of interest (person). Similarly, due to the movement of the camera a large portion background may be detected as part of the object of interest as shown by the huge rectangle surrounding the object of interest (person) in the Figure 7.5[1]. These additionally detected objects need to be eliminated before the object labeling as they will result in new object group (labeled as new groups), thus decreasing the object labeling algorithm accuracy.

Unwanted object instances in frames can be eliminated by discarding all moving objects detected that have bounding boxes with size below a minimum threshold and above a maximum threshold. Minimum threshold will filter all the small noises that are detected as moving objects. Similarly maximum threshold will discard objects with large portion of background (created due to background movement). These threshold values for a video stream can be determined as follows.

1. Select a small segment in the video stream (to be pre-processed) that contains an object of interest (person or car) moving from one end of the video to the other in the path followed by all (or most) objects of interest. In case of Video 1 in Figure 5.4, it would be a video segment that captures a person appearing from left border (entering from door) and walking till the right border of the video and disappearing. Similarly, in case of Video 2 in Figure 5.4 a car moving from top border of video (entering the parking lot) to bottom border (till the car disappears from the camera vicinity). This will cover the variation of size of an object in the most common path in the video stream.
2. Apply the methods in Section 4.2.1 and crop moving objects in every frame in the video segment. Note down the size of the object of interest in every frame.

[1]Ignoring these instances with large portion as background may not result in loosing out any objects of interest as the instance of this object without large portion of background will appear in future frame and it will be labelled correctly.

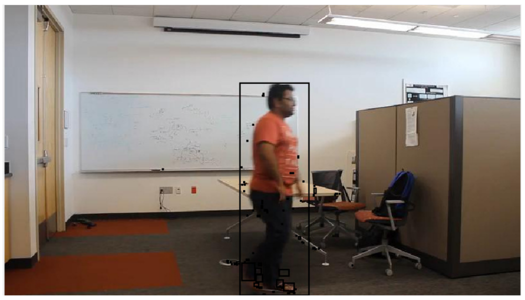


Figure 7.4: Illustration of need for minimum object size

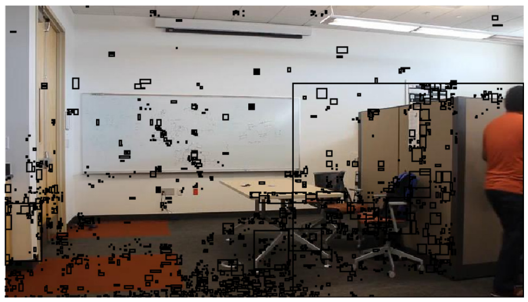


Figure 7.5: Illustration of need for maximum object size

3. Minimum threshold will be the minimum size of the object in the whole of this video segment. Similarly, maximum threshold will be the maximum size of the object.
4. In our experiments, observed minimum size of the human object for videos like Video 1 in Figure 5.4 is 60x100 pixels and the maximum size is 115x280 pixels. Similarly, for parking lot videos like Video 2 in Figure 5.4, the minimum object size is 30x60 and maximum size is 110x80.

7.2.4 **Object Match Threshold**

Object match threshold values decide if two given object instances belong to same object or not. Details of the application of object match threshold is explained in the Section 5.1.4. Threshold value depends on a lot of factors like type of video, type of objects present in the video streams, feature extraction algorithm being used etc. Following are the proposed guideline to choose a starting range of threshold value to arrive at a near ideal value.

1. Choose a short segment in the video stream being preprocessed such that it has the movement of two different objects. For example, if the video is similar to

Video 1 in Figure 5.4, select the video segment such that it contains at least two people walking from left end of the video to right end completely.

2. Apply the algorithms mentioned in the Chapter 4 and 5 on whole of this video segment to extract moving object instances (cropped images of objects) and their feature vectors in every frame.

3. Choose a random object instance (instance 1) of the first object, then choose an object instance (instance 2) of the second object such that position of the second object in the video in instance 2 is same as the position of the first object in the instance 1. This will assure that both the instances have same (or similar) background and the change is only in the actual object. For example, Figure 7.6 is the random object instance of the first object in Video 1 in Figure 5.4 and Figure 7.7 is the object instance of another object in the same video in same location (same background).

4. Apply the appropriate feature extraction algorithm on the object instances Figure 7.6 and Figure 7.7. And calculate the distance between the feature vectors. Refer the Section 5.1 for details on applying feature vector algorithm and calculating distance on them.

5. The distance obtained in the above step will be the object match threshold for the given video stream. All the object instances that have similarity distances between them fall below this threshold will belong to same object. The distance between Figure 7.6 and Figure 7.7 came out to be 0.0036. Thus 0.0036 is the starting value for object match threshold value for Video 1 in Figure 5.4. In order to verify this value, we calculated the distance between the object instances of the first object (Figure 7.7 and Figure 7.8). We received a value of 0.0013, which is less than threshold. Thus, it can be said that the object instances Figure 7.7 and Figure 7.8 belong to the same object group.

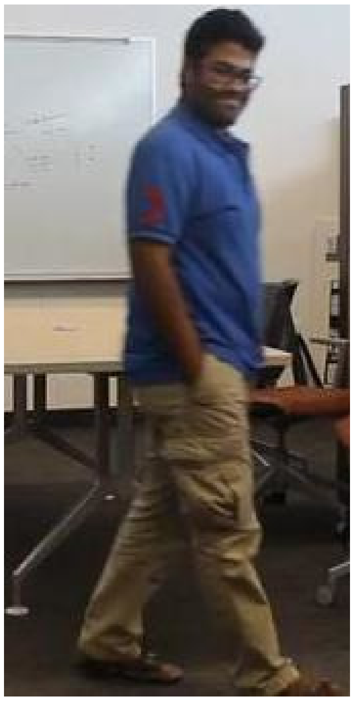


Figure 7.6: Object match threshold sample-1

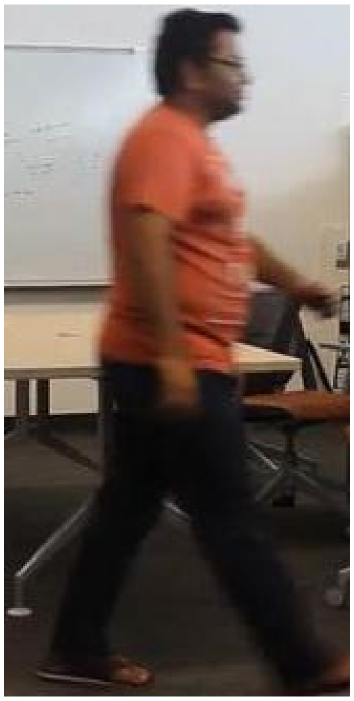


Figure 7.7: Object match threshold sample-2

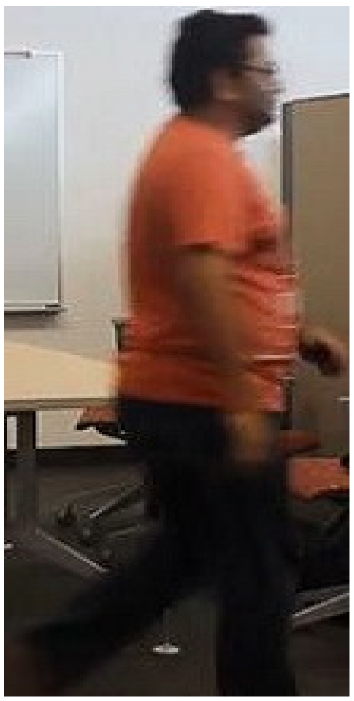


Figure 7.8: Object match threshold sample-3

6. If there are more than 2 moving objects in the video segment, then repeat steps 2-4 for each pair of objects and choose the minimum distance as the threshold.

7.2.5 Object Displacement Threshold

Object displacement threshold is the maximum number of pixels by which an object can move from its previous frame. This is used in Metric 3 of the object classification algorithm used in the Chapter 6. Displacement of an object can be measured using bounding box. Refer Chapter 6 for more details on measuring the displacement. Near ideal value for this threshold can be determined by applying the method in Section 4.2.1 to extract bounding boxes of object instances of an object on at least 10 continuous sample frames of given video stream. The displacement (x-coordinate and y-coordinate) of this object across every two subsequent continuous frames are calculated for the sample number of frames. And these values are averaged for each of x-coordinate and y-coordinate separately. This average value is chosen as the object displacement threshold. For example, the Algorithm 6 shows the calculation of x-coordinate ($threshold_x$) and y-coordinate ($threshold_y$) object displacement threshold for a sample of 15 frames in Video 1 in Figure 5.4. $Obj_{<2,j>}$ (j¿=1000 and j¡=1014) represents object instances of group label $Object_2$ in Video 1 in Figure 5.4. $bb_<2, j, x - cooridinate >$ is the x-coordinate of bounding box of object instance $Obj_{<2,j>}$. Similarly, $bb_<2, j, y - cooridinate >$ is the y-coordinate of bounding box of object instance $Obj_{<2,j>}$

Algorithm 6 Object Displacement Threshold

$$thresohld_x = \frac{\sum_{j=1000}^{1014} |bb_<2, j, x - cooridinate > -bb_<2, j+1, x - cooridinate >|}{15}$$

$$thresohld_y = \frac{\sum_{j=1000}^{1014} |bb_<2, j, y - cooridinate > -bb_<2, j+1, y - cooridinate >|}{15}$$

While testing this threshold on different video sets mentioned earlier, it was found that the displacement (x or y) of objects is generally within 50 pixels for consecutive frames. Hence the displacement threshold is fixed to the value 50.

This chapter has explained in detail about the key-parameters available in video stream pre-processing that used in video pre-processing framework. The details of the implementation of this framework is given in the Chapter 8.

CHAPTER 8

Video Pre-Processing Framework Implementation

The end-to-end pre-processing workflow (object identification, object labeling, output data model etc.) is implemented as a 'video stream pre-processing framework'. The implementation details like object identification, object extraction, feature extraction and object labeling is hidden from the user who attempts pre-process video stream using this framework. The framework can be visualized (Figure 8.1) as a blackbox that hides the implementation from the user but exposes the key-parameters mentioned in the Section 7.2 as configurations. The overall file directory organization of the framework is shown in the Figure 8.2. The main directory contains sub directories 'input'which contains input video files and configuration file, 'output'which contains the directories that store the frame image, the cropped object images and the output file containing relational model and 'source'contains the source code files. The framework is implemented using MATLAB 2015b. The following section explain in detail about the various data structures used and the overall work-flow of the framework.

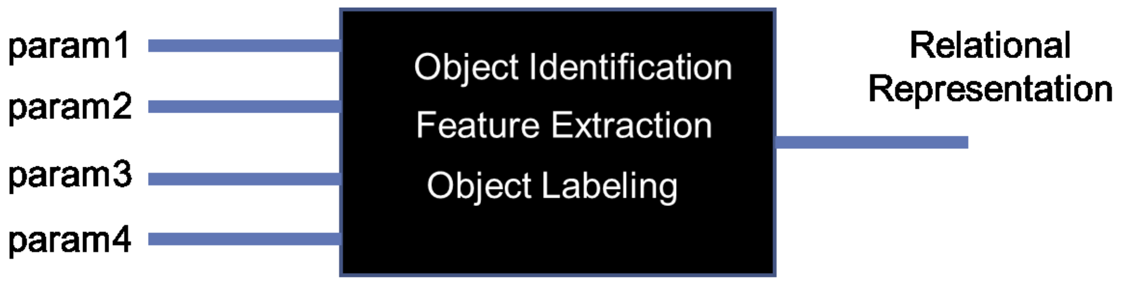
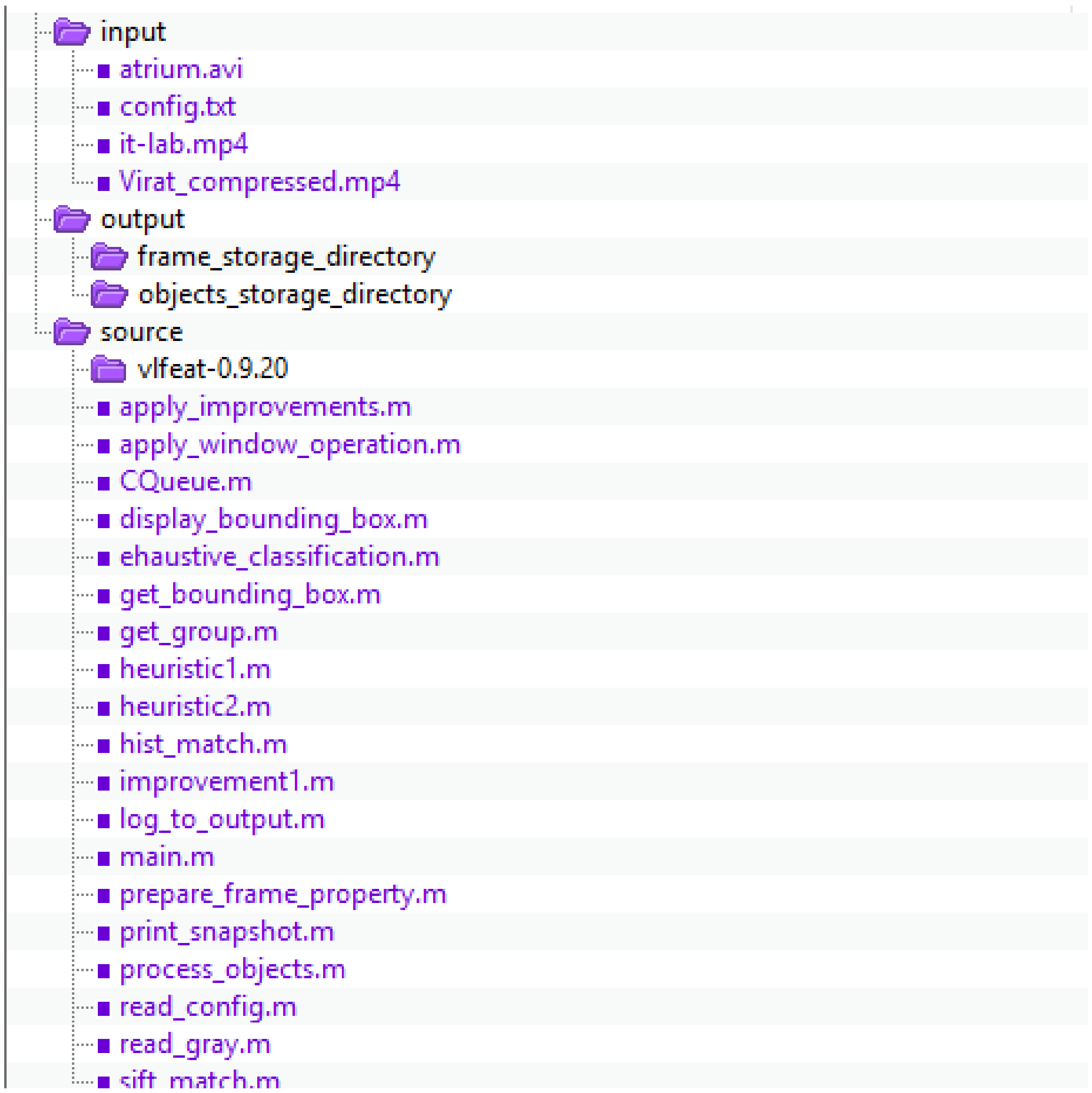


Figure 8.1: Video pre-processing framework as a blackbox



```
input
    atrium.avi
    config.txt
    it-lab.mp4
    Virat_compressed.mp4
output
    frame_storage_directory
    objects_storage_directory
source
    vlfeat-0.9.20
    apply_improvements.m
    apply_window_operation.m
    CQueue.m
    display_bounding_box.m
    ehaustive_classification.m
    get_bounding_box.m
    get_group.m
    heuristic1.m
    heuristic2.m
    hist_match.m
    improvement1.m
    log_to_output.m
    main.m
    prepare_frame_property.m
    print_snapshot.m
    process_objects.m
    read_config.m
    read_gray.m
    sift_match.m
```

Figure 8.2: Video pre-processing framework directory structure organization

8.1 Details of the Data Structures used

The implementation uses the following data structures to store the data that are carried across different functions and read/edited by these functions.

1. **frame_properties_map:** This is a map with key *frame_id* (each frame in the video stream is given an incremental id with starting id as 1) and value is a

structure that contains members i) path of this frame image stored on the disk and ii) list of *object_id* of the moving objects detected in this frame.

2. **objects_properties_map:** This is a map with key *object_id* (each object instance in the video stream is given an incremental id with starting id as 1) and value is a structure that contains the members i) path of this moving object image (cropped) stored on disk and ii) bounding box of this object (rectangle co-ordinates).
3. **group_objects_map:** This is a map with key *group_id* (output as *glabel* in Table 6.1) and value is a map that has key as object_id of the moving object instance that belong to this group label and the value as the frame_id that they appear in.
4. **universal_struct:** This is a structure that contains the above mentioned data structures as members along with few others members. The members in this structures are: i) frame_properties_map, ii) objects_properties_map, iii) group_objects_map, iv) group_count (total number of moving object instances detected so far in the video stream), v) comparison_count (total number of feature vector comparison done so far in the video stream for object instance labeling).
5. **config:** This structure is used to hold the various configuration values set by the user.

8.2 Video pre-processing framework implementation workflow

Figure 8.3 shows the overall workflow of the implementation of video pre-processing framework. This section goes through each of the steps in detail.

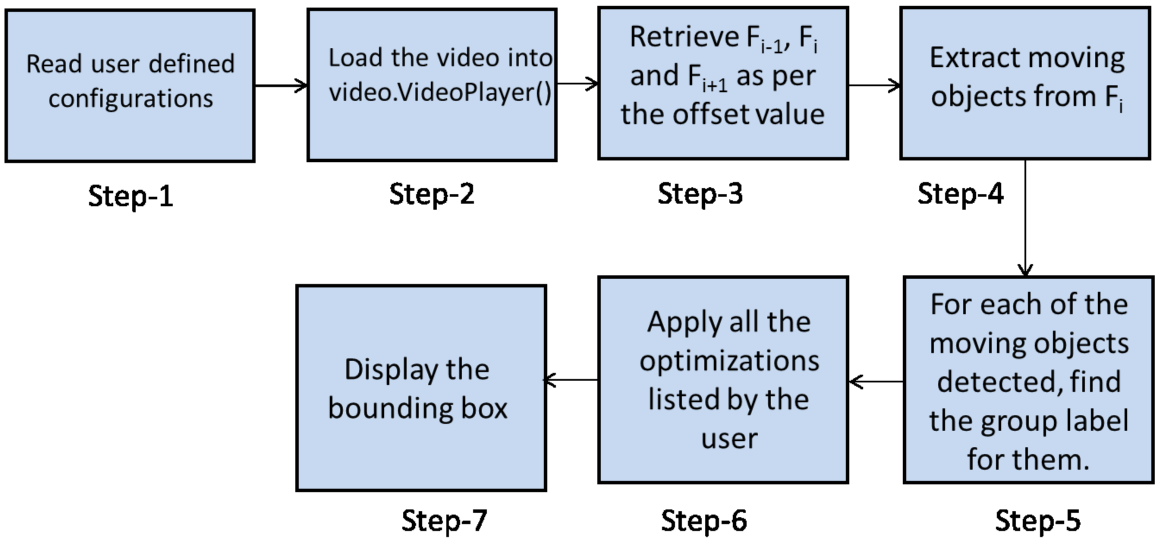
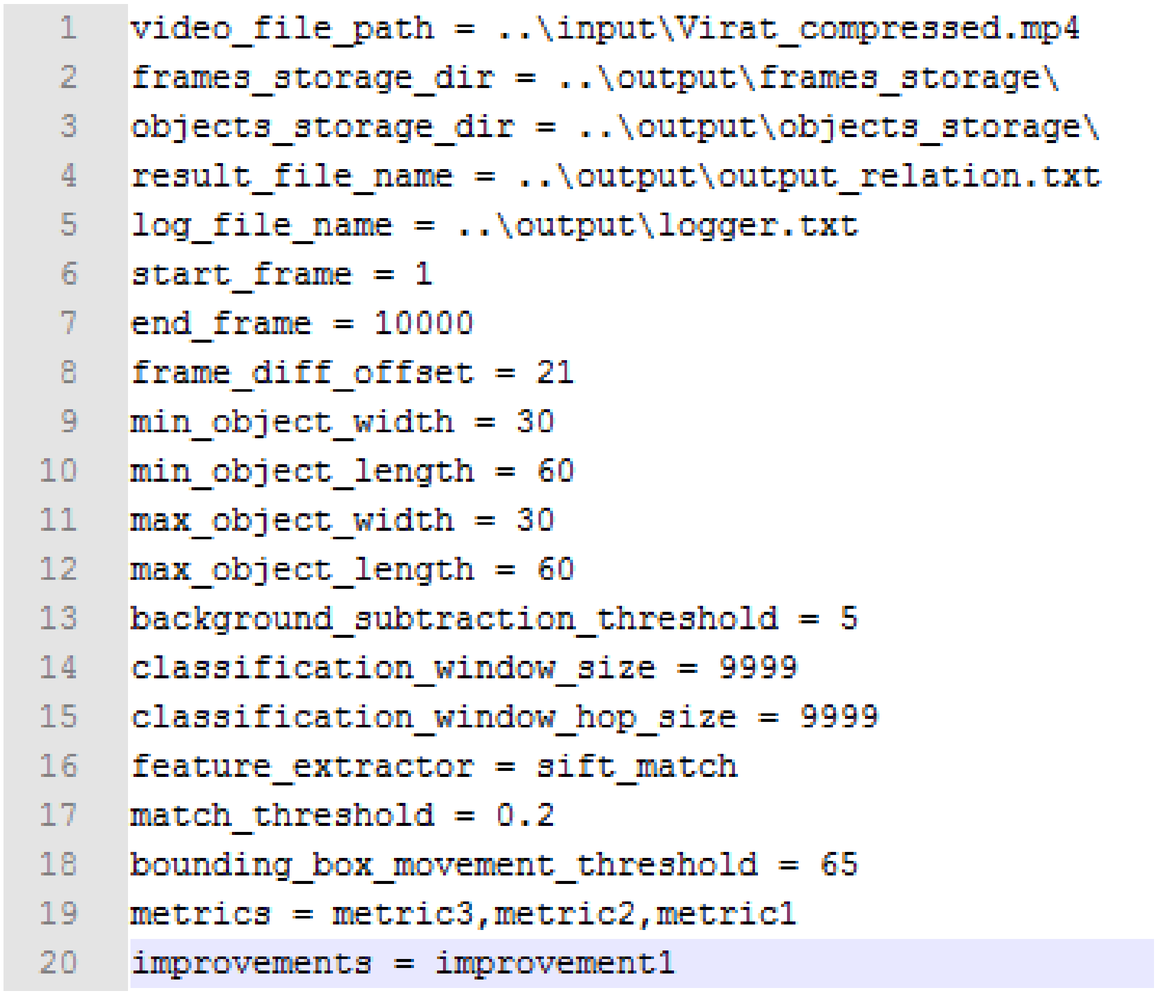

Read user defined configurations
Step-1
Load the video into video.VideoPlayer()
Step-2
Retrieve F_{i-1}, F_i and F_{i+1} as per the offset value
Step-3
Extract moving objects from F_i
Step-4
For each of the moving objects detected, find the group label for them.
Step-5
Apply all the optimizations listed by the user
Step-6
Display the bounding box
Step-7

Figure 8.3: Video Pre-processing Framework Workflow.

8.2.0.1 Step-1: Read user defined configurations

User defined configurations are recorded in a text file called ‘config.txt’present inside the directory ‘input’. The content of the configuration files are shown in the Figure 8.4. Following are the list of configurations available and the description of each of these parameters.

1. *video_file_path:* The complete (absolute path or relative path with respect to the ‘source’directory) file path of the input video file.
2. *frames_storage_dir:* Directory where the complete frame images are stored. It should be a sub-directory inside ‘output’directory.
3. *objects_storage_dir:* Directory where the cropped object images are stored so that they can be referred in various steps of the pre-processing workflow.
4. *result_file_name:* Complete path (absolute path or relative path with respect to the ‘source’directory) of the output file where the relational data model will be written to.



```
video_file_path = ..\input\Virat_compressed.mp4
frames_storage_dir = ..\output\frames_storage\
objects_storage_dir = ..\output\objects_storage\
result_file_name = ..\output\output_relation.txt
log_file_name = ..\output\logger.txt
start_frame = 1
end_frame = 10000
frame_diff_offset = 21
min_object_width = 30
min_object_length = 60
max_object_width = 30
max_object_length = 60
background_subtraction_threshold = 5
classification_window_size = 9999
classification_window_hop_size = 9999
feature_extractor = sift_match
match_threshold = 0.2
bounding_box_movement_threshold = 65
metrics = metric3,metric2,metric1
improvements = improvement1
```

Figure 8.4: Video Pre-processing Framework configuration File.

5. *log_file_name:* Complete path (absolute path or relative path with respect to the ‘source’directory) of the logger file that contains details of every step the framework goes through, statistics like number of comparisons, time taken, memory consumed etc.
6. *start_frame:* Frame number in the video from where the pre-processing should start with.

7. *end_frame:* The end frame number in the video where the pre-processing should stop.
8. *frame_diff_offset:* Value for frame offset size (Section 7.2.1).
9. *min_object_width, min_object_length, max_object_width and max_object_length:* Minimum and maximum object sizes used to filter noises in moving object detection (Section 7.2.3).
10. *background_subtraction_threshold:* Threshold value used to convert the grey-scale image containing moving object pixels to binary (Section 4.2.1).
11. *classification_window_size:* Frame history length value (Section 7.1.1)
12. *classification_window_hop_size:* Frame hop-size value (Section 7.2.2)
13. *feature_extractor:* Feature extraction algorithm to be used for object labeling (Chapter 5).
14. *match_threshold:* Threshold value that decides if two given object instances (cropped object images) are similar or not (Section 7.2.4).
15. *bounding_box_movement_threshold:* Value by which an object can move from one frame to other (Section 7.2.5).
16. *metrics:* Various metrics used in object labeling algorithm (Chapter 6) and the order to use them. In the Figure 8.4 'metric2'will be applied first, if that fails then metric1 is applied. Metric mentioned in Section 6.1 which is exhaustive need not be specified explicitly. If the specified metric in this configuration fail, framework will fall back to the metric1 (Section 6.1).
17. *imporvements:* Improvements are the optimization done on the object labeling algorithm as mentioned in the Section 7.1. In Figure 8.4 'improvement1'corresponds to the improvement 'selecting group representatives'(Section 7.1.2).

8.2.0.2 **Step-2: Load Video Into MATLAB**

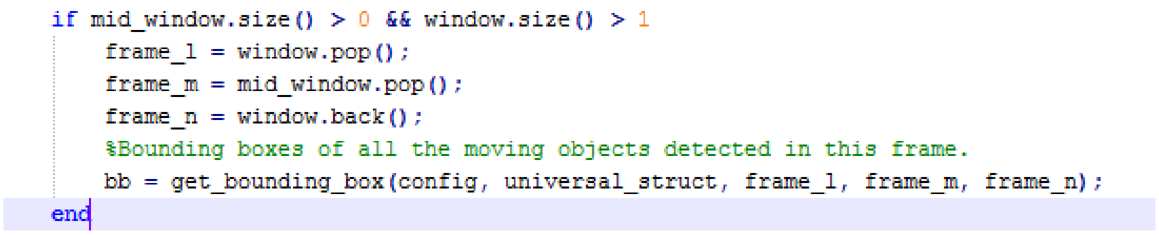
MATLAB Computer Vision Toolbox API vision.VideoFileReader() and property vision.VideoPlayer is used to load the video file and process it frame by frame. Each frame is extracted from the object created by vision.VideoFileReader() using the step() function.

8.2.0.3 **Step-3: Retrieve frames according to the frame difference offset value**

In order to achieve frame differencing for a frame F_i, a frame (frame_diff_offset - F_i) and (F_i + frame_diff_offset) need to be selected. Thus frames from (frame_diff_offset - F_i) to (F_i + frame_diff_offset) need to be collected before finding moving object in F_i. This is achieved using the queue definition (CQueue.m - copy this file to the 'source'dirctory) in [26]. Whenever a frame is extracted it is pushed to the queue 'window'(a queue of size frame difference offset value) and whenever (frame difference offset value)/2 frames are collected the subsequent frames are pushed to 'window_mid'(a queue of half of frame difference offset value) as well as 'window'. While processing F_i, the frame from beginning of the queue 'window'(F_{i-1}), frame from beginning of 'window_mid'(F_i) and frame from end of 'window'(F_{i+1}) are extracted. Code snippet corresponding to retrieving frames from queues for frame differencing is captured in Figure 8.5.

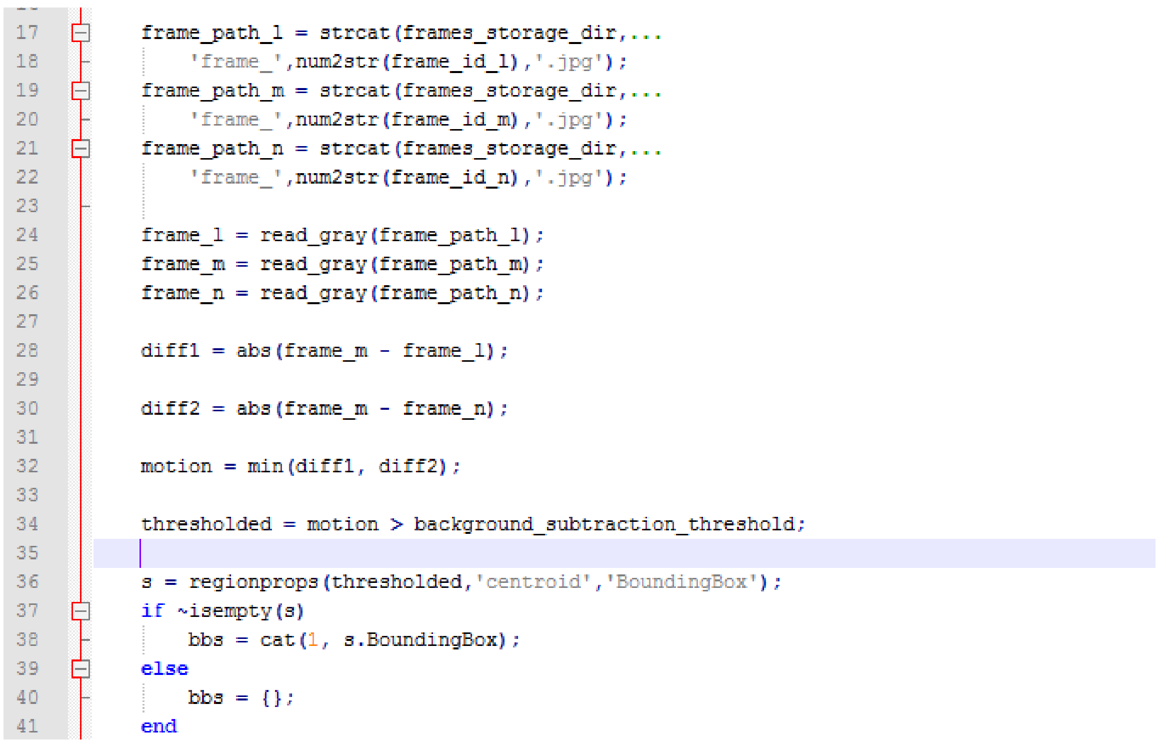
8.2.0.4 **Step-4: Extract moving objects from background**

The moving object image identified can be cropped calculating the bounding box co-ordinates as explained in the Section 4.2.1. MATLAB provides an API regionprops() that identifies connected areas and returns bounding boxes of these



```
if mid_window.size() > 0 && window.size() > 1
    frame_l = window.pop();
    frame_m = mid_window.pop();
    frame_n = window.back();
    %Bounding boxes of all the moving objects detected in this frame.
    bb = get_bounding_box(config, universal_struct, frame_l, frame_m, frame_n);
end
```

Figure 8.5: Code snippet capturing extracting frames for frame differencing from queues.

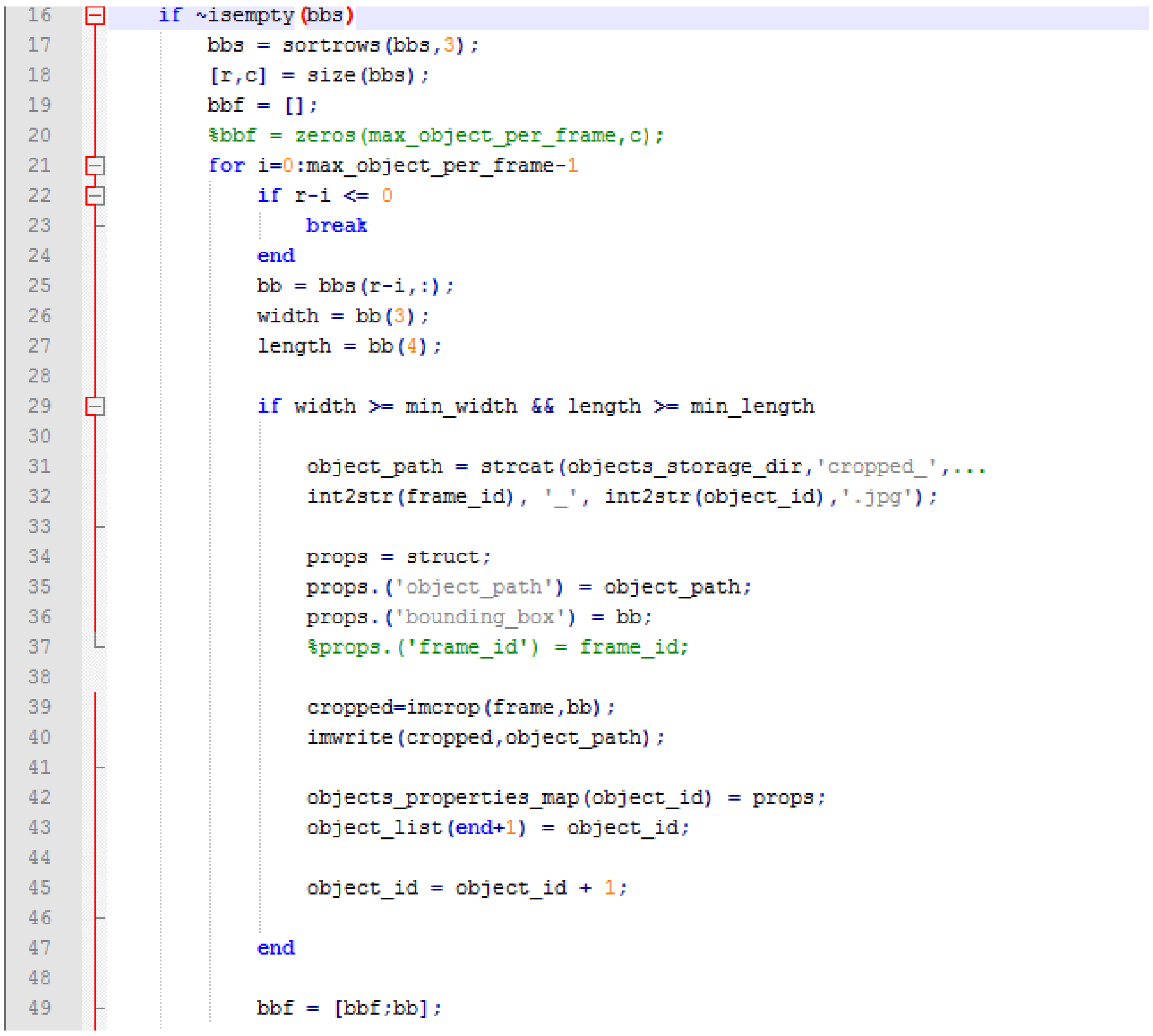


```
frame_path_l = strcat(frames_storage_dir,...
    'frame_',num2str(frame_id_l),'.jpg');
frame_path_m = strcat(frames_storage_dir,...
    'frame_',num2str(frame_id_m),'.jpg');
frame_path_n = strcat(frames_storage_dir,...
    'frame_',num2str(frame_id_n),'.jpg');

frame_l = read_gray(frame_path_l);
frame_m = read_gray(frame_path_m);
frame_n = read_gray(frame_path_n);

diff1 = abs(frame_m - frame_l);

diff2 = abs(frame_m - frame_n);

motion = min(diff1, diff2);

thresholded = motion > background_subtraction_threshold;

s = regionprops(thresholded,'centroid','BoundingBox');
if ~isempty(s)
    bbs = cat(1, s.BoundingBox);
else
    bbs = {};
end
```

Figure 8.6: Code snippet capturing frame differencing for object identification.

areas in the resultant difference image (motion) in the Equation 4.2. The file ‘get_bounding_box.m’captures the implementation details mentioned above (Figure 8.6).

File ‘process_objects.m’contains the implementation of filtering the noise based on the minimum and maximum size (Section 7.2.3) of the detected areas. Also it



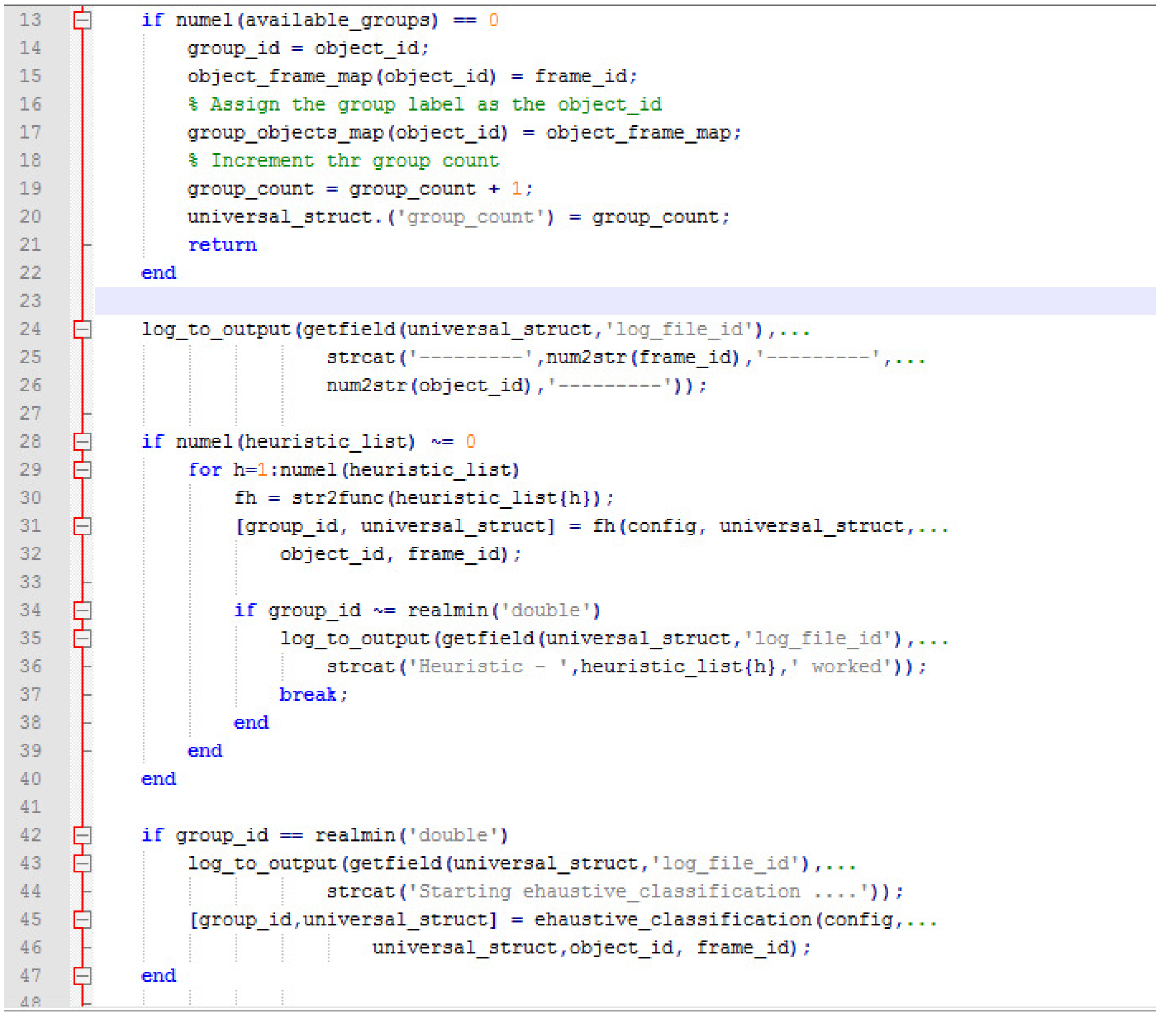
```
    if ~isempty(bbs)
        bbs = sortrows(bbs,3);
        [r,c] = size(bbs);
        bbf = [];
        %bbf = zeros(max_object_per_frame,c);
        for i=0:max_object_per_frame-1
            if r-i <= 0
                break
            end
            bb = bbs(r-i,:);
            width = bb(3);
            length = bb(4);

            if width >= min_width && length >= min_length

                object_path = strcat(objects_storage_dir,'cropped_',...
                int2str(frame_id), '_', int2str(object_id),'.jpg');

                props = struct;
                props.('object_path') = object_path;
                props.('bounding_box') = bb;
                %props.('frame_id') = frame_id;

                cropped=imcrop(frame,bb);
                imwrite(cropped,object_path);

                objects_properties_map(object_id) = props;
                object_list(end+1) = object_id;

                object_id = object_id + 1;

            end

            bbf = [bbf;bb];
```

Figure 8.7: Code snippet capturing moving object extraction from background.

contains the implementation of imposing the bounding box of moving objects on the original image to crop and save the object image on disk. Figure 8.7 contains the code snippet from the file 'process_objects.m'.

8.2.0.5 **Step-5: Object labeling**

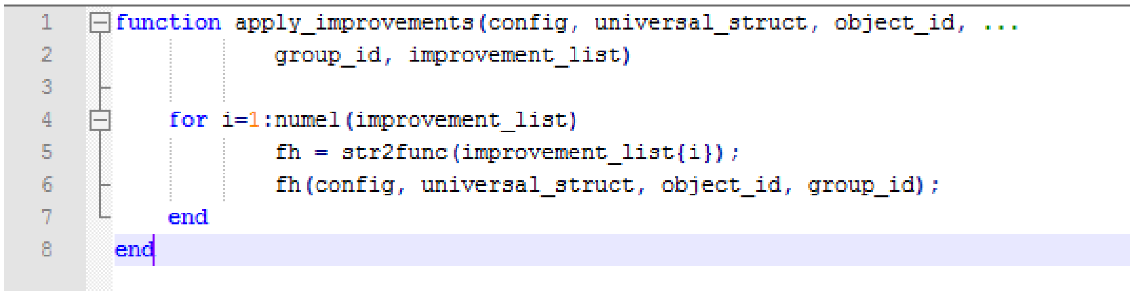
For every moving object identified in step-4, object labeling subroutine is called to determine its group label. 'get_group.m'file contains the implementation details



```
if numel(available_groups) == 0
    group_id = object_id;
    object_frame_map(object_id) = frame_id;
    % Assign the group label as the object_id
    group_objects_map(object_id) = object_frame_map;
    % Increment thr group count
    group_count = group_count + 1;
    universal_struct.('group_count') = group_count;
    return
end

log_to_output(getfield(universal_struct,'log_file_id'),...
            strcat('---------',num2str(frame_id),'---------',...
            num2str(object_id),'---------'));

if numel(heuristic_list) ~= 0
    for h=1:numel(heuristic_list)
        fh = str2func(heuristic_list{h});
        [group_id, universal_struct] = fh(config, universal_struct,...
            object_id, frame_id);

        if group_id ~= realmin('double')
            log_to_output(getfield(universal_struct,'log_file_id'),...
                strcat('Heuristic - ',heuristic_list{h},' worked'));
            break;
        end
    end
end

if group_id == realmin('double')
    log_to_output(getfield(universal_struct,'log_file_id'),...
            strcat('Starting ehaustive_classification ....'));
    [group_id,universal_struct] = ehaustive_classification(config,...
                universal_struct,object_id, frame_id);
end
```

Figure 8.8: Code snippet capturing assigning group label to moving objects.

of the determining group label. It calls every metric mentioned in the configuration ‘metrics’one by one in the order to determine the group label and if the group label is not determined by the mentioned metrics, it falls back to the exhaustive metric (Section 6.1). Code snippet in the Figure 8.8 shows the implementation of the assigning group label. ‘available_groups’is the list of available group labels, ‘heuristic_list’is the list of metrics and ‘ehaustive_classification.m’contains the implementation of Section 6.1.



```
function apply_improvements(config, universal_struct, object_id, ...
            group_id, improvement_list)

    for i=1:numel(improvement_list)
            fh = str2func(improvement_list{i});
            fh(config, universal_struct, object_id, group_id);
    end
end
```

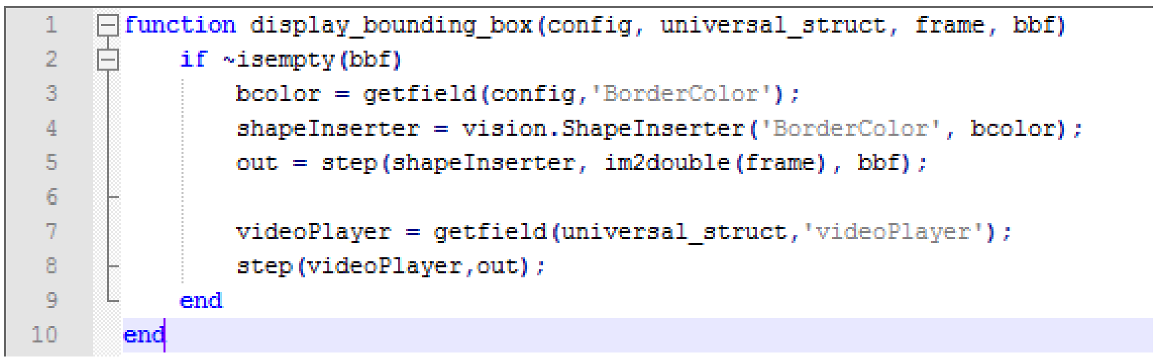
Figure 8.9: Code snippet capturing applying improvements.

8.2.0.6 Step-6: Apply optimizations for object labeling algorithm

Once the object label is determined for a moving object instance in a frame, it can be either added to its group or it is discarded (Section 7.1). File 'apply_improvements.m'reads the improvements list from the 'improvements'configuration and applies them one by one. Code snippet in Figure 8.9 shows the implementation of applying improvements.

8.2.0.7 Step-7: Display bounding box

Every-time bounding box is determined for a moving object, it has to be displayed in the video player of MATLAB. File 'display_bounding_box.m'contains the implementation of inserting rectangle shape around the moving object in the video player. Figure shows the code snippet corresponding to inserting shape.



```
function display_bounding_box(config, universal_struct, frame, bbf)
    if ~isempty(bbf)
        bcolor = getfield(config,'BorderColor');
        shapeInserter = vision.ShapeInserter('BorderColor', bcolor);
        out = step(shapeInserter, im2double(frame), bbf);

        videoPlayer = getfield(universal_struct,'videoPlayer');
        step(videoPlayer,out);
    end
end
```

Figure 8.10: Code snippet capturing displaying bounding box for moving objects.

CHAPTER 9

Conclusion and Future Work

This thesis was able to successfully demonstrate that generic information related to moving objects can be extracted from video streams and scenarios related to object tracking can be evaluated using SQL and CQL semantics on the extracted data (modeled in relational representation). This thesis has developed a video stream pre-processing framework using which a given video stream containing either human or car type of objects can be pre-processed to extract relational data model. As part of this framework, key-parameters involved in video pre-processing are identified and these key-parameters are implemented as tunable configuration in the framework implementation. Additionally, this thesis has recommended starting value ranges for these key parameters so that the users using this framework does not have to run exhaustive experiments to arrive at ideal values for these key parameters.

This thesis has identified few limitations in the methods used for video pre-processing. Some of the limitations identified are i) histogram cannot differentiate people wearing similar colored clothing, ii) frame differencing cannot identify two separate objects if they are too close to each other and iii) frame differencing cannot identify static objects in the video stream. As future work we would like to address these limitations by adapting alternative Computer Vision methods. Additionally, we would like to extend (arrable) the data model and extend the query language to express more situational queries.

REFERENCES

[1] A. Cordova, L. D. Millard, L. Menthe, R. A. Guffey, and C. Rhodes, "Motion Imagery Processing and Exploitation (MIPE)," *Santa Monica, CA: Rand Corporation*, 2013. [Online]. Available: http://www.rand.org/pubs/research_reports/RR154.html

[2] E. Hwang and V. S. Subrahmanian, "Querying video libraries," *J. Visual Communication and Image Representation*, vol. 7, no. 1, pp. 44–60, 1996. [Online]. Available: http://dx.doi.org/10.1006/jvci.1996.0005

[3] I. Ide, *Video Querying.* Boston, MA: Springer US, 2009, pp. 3292–3296. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-39940-9_1030

[4] M. Flickner, H. S. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: The QBIC system," *IEEE Computer*, vol. 28, no. 9, pp. 23–32, 1995. [Online]. Available: http://dx.doi.org/10.1109/2.410146

[5] F. Pla, P. C. Ribeiro, J. Santos-Victor, and A. Bernardino, "Extracting motion features for visual human activity representation," in *Pattern Recognition and Image Analysis, Second Iberian Conference, IbPRIA 2005, Estoril, Portugal, June 7-9, 2005, Proceedings, Part I*, 2005, pp. 537–544. [Online]. Available: http://dx.doi.org/10.1007/11492429_65

[6] W. Niu, J. Long, D. Han, and Y. Wang, "Human activity detection and recognition for video surveillance," in *Proceedings of the 2004 IEEE International Conference on Multimedia and Expo, ICME 2004, 27-30 June 2004, Taipei, Taiwan*, 2004, pp. 719–722.

[7] (2008) Virat. [Online]. Available: https://en.wikipedia.org/wiki/VIRAT

[8] B. Liu, A. Gupta, and R. C. Jain, “Medsman: a streaming data management system over live multimedia,” in *Proceedings of the 13th ACM International Conference on Multimedia, Singapore, November 6-11, 2005*, 2005, pp. 171–180. [Online]. Available: http://doi.acm.org/10.1145/1101149.1101174

[9] A. J. Aved, “Scene Understanding For Real Time Processing Of Queries Over Big Data Streaming Video,” *Ph.D. Dissertation, UCF Orlando, Florida*, 2013.

[10] X. Liu, Y. Zhuang, and Y. Pan, “A new approach to retrieve video by example video clip,” in *Proceedings of the 7th ACM International Conference on Multimedia ’99, Orlando, FL, USA, October 30 - November 5, 1999, Part 2.*, 1999, pp. 41–44. [Online]. Available: http://doi.acm.org/10.1145/319878.319889

[11] A. Arasu, S. Babu, and J. Widom, “The CQL continuous query language: semantic foundations and query execution,” *VLDB J.*, vol. 15, no. 2, pp. 121–142, 2006. [Online]. Available: http://dx.doi.org/10.1007/s00778-004-0147-z

[12] S. Chakravarthy, A. Aved, S. Shirvani, M. Annappa, and E. Blasch, “Adapting stream processing framework for video analysis,” in *Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014*, 2015, pp. 2648–2657. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.05.372

[13] A. Lerner and D. E. Shasha, “Aquery: Query language for ordered data, optimization techniques, and experiments,” in *VLDB*, 2003, pp. 345–356. [Online]. Available: http://www.vldb.org/conf/2003/papers/S11P03.pdf

[14] “Broad agency announcement, video and image retrieval and analysis tool (virat),” DARPA INFORMATION PROCESSING TECHNIQUES OFFICE

(IPTO), Tech. Rep. BAA 08-20, 03 March 2008. [Online]. Available: https://www.fbo.gov/utils/view?id=32f2382440cfb57d2695171885acab57

[15] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *In Workshop on Statistical Learning in Computer Vision, ECCV*, 2004, pp. 1–22.

[16] R. Szeliski, *Computer Vision - Algorithms and Applications*, ser. Texts in Computer Science. Springer, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-1-84882-935-0

[17] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, 2006. [Online]. Available: http://doi.acm.org/10.1145/1177352.1177355

[18] R. E. Bellman, *The Bellman Continuum: A Collection of the Works of Richard E. Bellman.* World Scientific Publishing Company Incorporated, 1986.

[19] C. O. S. Sorzano, J. Vargas, and A. D. Pascual-Montano, "A survey of dimensionality reduction techniques," *CoRR*, vol. abs/1403.2877, 2014. [Online]. Available: http://arxiv.org/abs/1403.2877

[20] C. Kanan and G. W. Cottrell, "Color-to-grayscale: Does the method matter in image recognition?" *PLoS ONE*, vol. 7, no. 1, pp. 1–7, 01 2012. [Online]. Available: http://dx.doi.org/10.1371/journal.pone.0029740

[21] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[22] G. Bradski, *Dr. Dobb's Journal of Software Tools*, 2000.

[23] S. Oh, A. Hoogs, A. G. A. Perera, N. P. Cuntoor, C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, H. Lee, L. S. Davis, E. Swears, X. Wang, Q. Ji, K. K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen,

A. Torralba, B. Song, A. Fong, A. K. Roy-Chowdhury, and M. Desai, "A large-scale benchmark dataset for event recognition in surveillance video," in *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, 2011, pp. 3153–3160. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995586

[24] MathWorks, *Image Processing Toolbox MATLAB R2016a Sample Video - atrium.avi.* MathWorks, 2016. [Online]. Available: http://www.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html

[25] J. Han, *Data Mining: Concepts and Techniques.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[26] Z. Zhang. (2011) list,queue,stack. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/28922-list--queue--stack

BIOGRAPHICAL STATEMENT

Manish Kumar Annappa was born in Kundapura, Karnataka, India. He received his Bachelor of Engineering degree in Information Science and Engineering from PES Institute of Technology, Bangalore, India in May 2010. There after he worked as a Software Engineer with Adobe Systems India Pvt Ltd, Bangalore from July 2010 till July 2014. In the Fall of 2014, he started his graduate studies in Computer Science and Engineering at The University of Texas, Arlington. He worked as an ASE intern at MathWorks Inc, Natick, MA during Fall - 2015. He received his Master of Science in Computer Science and Engineering from The University of Texas at Arlington, in December 2016. His research interests include computer vision, stream processing and data mining.