Development of Economic and Reliable Tool for Condition Survey and Middle Management of Asphalt Pavement

By

Mubarak Bakheet Al-Falahi
B.Sc Colorado State University
M.A.Sc United Arab Emirates University

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In
Civil Engineering

Ottawa-Carleton Institute of Civil and Environmental Engineering

Carleton University
Ottawa, Ontario

Abstract

Rehabilitation and maintenance of pavement are well established and costly operations. One of the major challenges of these operations is determining the optimum maintenance and repair schedule. In any given city, there are hundreds of miles of pavement constructed at different times and are in different states of deterioration. It is the job of municipal engineers to establish a cost effective schedule that prioritizes the repair or even the reconstruction of different segments of the roads. The schedule is usually based on an overall Pavement Condition Index (PCI). Typically, inspectors are sent out to observe the pavement conditions and conduct accurate measurements to be used in computing the PCI; however the cost associated with such inspection missions is often high.

This thesis proposes a novel approach to estimate a cost effective Pavement Indicator (PI) for the entire city (or any area of interest). The proposed approach exploits newly available miniature cameras, GPS, wireless networking and Digital Signal Processing to automatically and continually collect visual information about different segments of the road, and combines these images to establish a live map of the city roads where different colours correspond to an approximate estimate of a pavement indicator (PI). The proposed technique is not a replacement of the traditional inspection, but rather it is a tool to identify the sections that are in greater need of repair. The technique involves taking pictures of various sections of the road network using cameras mounted on public vehicles and transmitting these images to a processing centre. Each image is processed using image filtering techniques to produce an initial estimate of PI. The cumulative effect of these estimates produces regional estimates that become more

statistically accurate as time goes by, and the overall PI map is continually updated to maintain a global visual map of PI. The proposed pavement indicator (PI) map can then be linked to an optimization software package to determine the most cost effective road rehabilitation schedule.

This dissertation is dedicated to the memory of my father,

Bakheet Ghaith Bakheet Al-Falahi

ACKNOWLEDGEMENTS

First of all, I would like to profusely thank Allah for enabling me to achieve this research work.

I would like to express my thanks to my mother who prays and supports me until I completed my Ph.D.

I would like to thank Carleton University and NSERC for the support and assistance provided to me while working on this thesis. I would like to express my sincere gratitude to my supervisors Professor Abd El Halim Omar Abd El Halim and Professor Roshdy Hafez for their continuous guide, support and time during my Ph.D study. Also, I would like to express my many thanks to the members of the examination committee: Professor Carl Hass, Professor John Goldak, Professor Ata Khan, and Professor Yasser Hassan.

I would like to thank all the people who have helped me during the data collection. Special thanks are due to my friend who is like a brother to me; Dr. Ali Kassim who helped me since we met and treat me as part of his family.

Last but not the least, I would like to express my love and thanks to my wife and my beautiful kids for continuing support to finish my thesis and without them I could not reach that. Also, many thanks are due to my brothers and sisters for supporting me spiritually throughout writing this thesis and my life in general.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
3G	Third Generation of Mobile Telecommunication Technology
LTE	Long-Term Evolution Wireless Communication
AASHTO	American Association of State Highway and Transportation Officials
AC	Alternating Current
ADA	Automated Distress Analyzer software
ARAN	Automatic Road Analyzer
ARIA	Automated Road Image Analyzer
ASCII	File Formatted with a word processor &transmitted as binary file
ASTM	American Society of Testing Materials
CGH	Pavement Engineering Inc.
CI	Universal Cracking Indicator
CMOS	Complementary metal-oxide-semiconductor
DC	Direct Current
DHDV	Digital Highway Data Vehicle
DMI	Distance Measurement Instrument
D-Rate	Digital Distress Rating System
DS3	Dalsa Spyder 3 camera
FHWA	Federal Highway Administration

FPS	Frame Per Second
GB	Gigabit
GDP	Gross Domestic Product
GHz	Gigahertz, 10^9 Hz
GIS	Geographic Information System
GPS	Global Position System
HARRIS	Highways Agency Road Research Information System
ICC	International Cybernetics Corporation Company
IMS	Infrastructure Management Services Company
KDOT	Kansas Department of Transportation
KHz	Kilo Hertz, 10^3 Hz
K-NN	K-nearest Neighbor
LED	Light Emitting Diode
Long	Longitudinal cracking
LTPP	Long-Term Pavement Performance
MHM	MHM Association Pavement Company
MHz	Mega Hertz, 10^6 Hz
MLF	Multilayer Feed Forward
ODOT	Oregon Department of Transportation
PADIAS	Pavement Distress Analysis System
PAS 1	Automated Distress Measurement Device
Pasco`s 35mm	Traditional 35 millimeter Film
Pathview I	Pavement Semi-Automated Detection System

Pavetech	Pavement Technical Company
Pavetech VIV	Video Inspection Vehicle Manufacture by Pavetech Inc.
PAVUE	Pavement Data Acquisition System
PCES	Pavement Condition Evaluation Services
PCI	Pavement Condition Index
PI	Pavement Indicator
PMS	Pavement Management System
Roadview GD Plot	Semi-Automated Rating System
RST	Laser Road Surface Tester
SHRP	Strategic Highway Research Program
S-VHS	Super VHS, improved version of VHS video cassettes
Trans	Transverse Cracking
TxDOT	Texas Department of Transportation
V	Volts
VDOT	Virginia Department of Transportation
Waylink	Automated Data collection System
yd^2	Square Yard

CHAPTER 1: INTRODUCTION

1.1 Pavement Management and Condition Survey

Transportation systems are considered to be among the most critical and important components that contribute significantly to the national economy. For example, these systems account for nearly 11% of the USA Gross Domestic Product (GDP). Transportation systems provide links among nations, communities, businesses, industries, markets and consumers [1].

The highway system is considered to be a measure of the degree of advancement and standard of living of societies worldwide. In fact, the national road network in the United States is considered to be the most advanced and largest in the world and it consists of:

- Interstate highways – more than 77,000 km [2], [3].
- National highways system – almost 260,000 km [4], [5].
- Other roads and streets 6,545,839 km for local transportation [6].

The Canadian highway network is not as large as the American one, but it is also considered to be among the most developed in the world with more than 1,000,000 km of paved and unpaved roads [7], [8]. In order to manage such huge highway systems, governments and highway agencies need to rely on advanced and expensive management systems to protect their significant investments through maintaining the road networks to be in good service conditions. Over past decades, the Pavement Management System (PMS) has evolved to provide a major part of the techniques and methods of maintaining advanced transportation infrastructure. The PMS is defined as "a set of strategies at

various management levels to schedule pavement maintenance and rehabilitation at an adequate level of serviceability" [8], [9]. The PMS includes many tasks such as:

1. Inventory of pavement condition in classifying the status of the pavement into good, fair, and poor status.
2. Attaching an index number to indicate the importance (or urgency) of maintaining any specific road segment. The index used most often is the Pavement Condition Index (PCI); and
3. Scheduling maintenance of the road and repair the poor and fair pavements to improve the overall riding quality of the road network.

Figure 1.1 illustrates an example of pavement deterioration over time. This figure shows that the pavement naturally deteriorates over time due to traffic loading and environmental conditions. It also shows that early and timely treatments of highways pavements will significantly reduce the cost of repair, helps preserve the initial investment and extend the service life of the pavement [10], [11], [12].

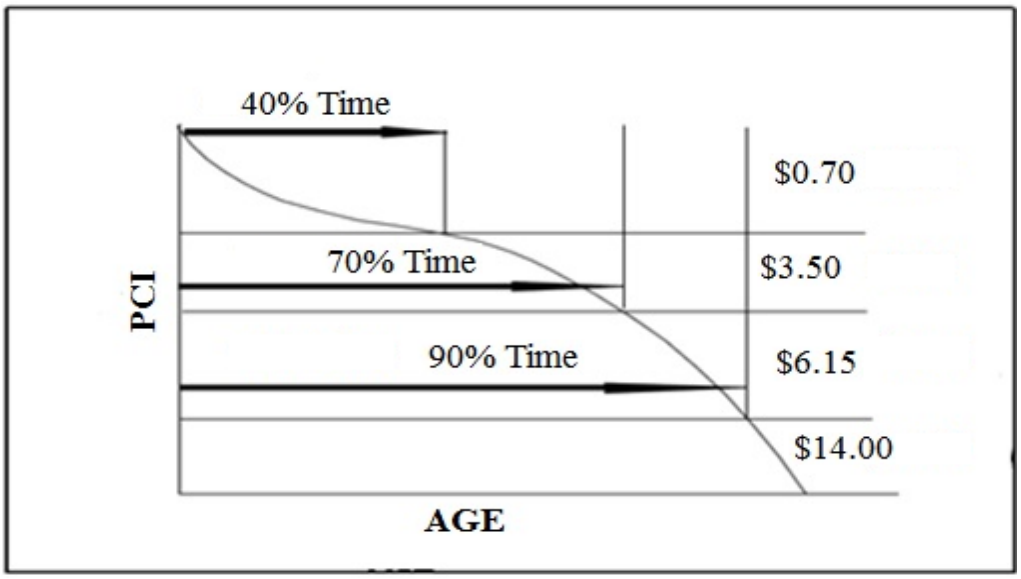


Figure 1.1: Pavement management system deterioration curve
(reproduced from [10], [11]).

The pavement condition survey is an essential step in the overall PMS processes. The results of such survey provide data and information which are used for analysis of the visual distress observed on the pavement surface such as cracking and/or surface deformation. Furthermore, it provides the information needed to evaluate the health of the road network and condition of any specific segment of the road. The PCI is a common figure of merit that quantifies performance and is used to objectively compare the status of various pavement segments over their relative service lives. The PCI is a numerical index ranging from 0 to 100 where 0 means very poor and 100 means excellent. However, roads are not allowed to deteriorate to a level of 0, but rather to a minimum acceptable level, depending on the importance of the road in question and the availability of funds [13], [14].

The Condition Survey is the process of collecting data to be used in computing the PCI [15], [16]. The data is used to determine the structural integrity, distress, skid

resistance and the overall riding quality of the pavement. Traditionally, maintenance schedules have relied on highly qualified technicians, field personnel and engineers. The raters are sent out to visually inspect sections of the road and determine their conditions, engaging in what is known as Windshield Survey. Sections that are judged to be of questionable quality would then receive more thorough inspection by using expensive mobile equipment. There are many drawbacks of this traditional method [17], [18], including:

1- The process is time consuming,

2- Highly qualified personnel are few and hard to train,

3- The cost of inspection trips is high and unsafe; and

4- The method can only inspect a small fraction of the road network [8].

Additionally, the traditional method should be allowed to continue to measure the pavement condition survey since there is no machine intelligence that can match the intuition of expert human force. Human technical experience is difficult to transfer from one person to another and decisions made by different experts using similar data often vary based on their individual experience. Moreover, present advanced automated equipment such as the Automatic Road Analyzer Unit (ARAN) as shown in Figure 1.2 and Figure 1.3 is considered expensive and; requires very highly trained crews and analysts, and the production of large volumes of data may not suit all levels of road municipalities and agencies. However, advanced technologies provide engineers and managers with another reliable approach that complements and enhances the old traditional approach. Based on the evidence provided above, there is a need to develop a

more economical and attainable technique that allows for medium size municipal road networks, and limited budgets to purchase sophisticated survey equipment such as the ARAN unit and the skills of highly trained experts. The development of such an intermediate approach is the focus of this research [19], [20].

Figure 1.2: ARAN unit [21].

Figure 1.3: ARAN Imaging &Capturing system [21]**.**

There are several important tasks achieved when a pavement condition survey is monitored and performed. The following outcomes are identified [22], [23] as follows:

- Evaluate the current condition of the road at both project and network levels.

- Determine the rates of deterioration and its causes.

- Evaluate project`s future conditions.

- Evaluate and determine road maintenance needs.

- Identify and determine the cost of repairs.

- Schedule proper action plans through prioritization and optimization to meet budget constraints.

- Determine the effects of deferred maintenance on budget reductions through periodical monitoring and feedback.

- Schedule of future pavement maintenance and rehabilitation activities.

- Track performance of various pavement designs and materials and assessment their effectiveness.

Asphalt pavements are suffering day-to-day deterioration that subsequently manifests as surface distress of different types. This distress is the damage that can be visually observed on the pavement surface and monitored as their degrees of severity and density can grow over time. Condition surveys are often carried out to determine the type, severity, and density of surface distresses. Results of distress surveys can be influenced by factors such as pavement types, age, average daily traffic, axle loadings, drainage

characteristics, environmental conditions and the frequency of such surveys. Different time periods and surveying methods can be contingent on the type of road facility as well. For example, arterials might be inspected annually, collectors every two years, and residential streets every four years. Time and efforts that are spent on any pavement inspection are related to the condition of the pavement. For example, pavements that are in good condition need less repeated inspections than pavements that suffer high rates of deterioration. Distress surveys can be performed manually or by an automated scheme. There are several manuals that can be used to describe the distress types, severity, density or quantity, as well as how to measure the distress. For example, ASTM D6433-11 is a standard practice for roads and parking lots for pavement condition index surveys [24], SHRP-P-338 is distress identification manual for long term pavement performance project [25], and AASHTO-P44 is a standard practice for quantifying cracks in asphalt pavement surface [26]. There are many types of data necessary to be collected for monitoring the pavement condition. For example, [i] Identify roads which require no immediate maintenance, [ii] identify roads which need minor or routine maintenance, [iii] Identify roads which need preventive maintenance activities and [iv] Identify roads which require major rehabilitation or reconstruction [8] .

Collecting data and information to rate and diagnose the road conditions is considered to be the most critical and also expensive part of any PMS program. In general, there are two surveying that methods can be used in order to collect pavement condition data, [i] Manual surveys and [ii] Automated surveys. Manual survey has the disadvantage of being time consuming, hazardous, and subjective. Automated surveys, on the other hand, have the benefits of being accurate, timely, and also allow for real time

analysis. The present state of the art is somewhat divided between these two extremely different approaches. Both techniques are in use in most nations with an emphasis on the manual techniques in less developing nations while developed countries are technically and financially equipped to use the more advanced and sophisticated automated approach [27].

A vehicle is usually the device utilized to collect pavement condition data in automated surveys. The data includes rut depth, ride quality, texture, global positioning, orientation and numerous types of surface distress. Cracking is the most difficult type of data to detect and classify automatically. Examples of surface cracking types are: fatigue cracking, block cracking, edge cracking, longitudinal cracking and transverse cracking. There are new advanced technologies to detect these types of cracking by computer hardware, which includes image recognition techniques and classifying surface distresses automatically in an economical manner. The concept of fully automated pavement condition surveys is nearly realized by extensive research and technological advancements. However, there are some issues that have slowed the acceptance of automated data collection technology, especially in countries or regions that lack the technical expertise, and financial capabilities or both [28].

A second issue that added to the cost of automated pavement condition survey is the rapid change and advancement of technology that leads to increasing the amount of upgrades the system and maintenance of it. Finally, most nations' road networks are not as huge and advanced as those of USA and Canada. Subsequently, intermediate, economical and technical viable option is needed between the most advanced and the most traditional [29].

1.2 Problem Definition

The problem addressed in this thesis is automating the collection and processing of data used to evaluate pavement conditions. The aim is to produce an automated system that continuously monitors the status of paved roads and to use this data to establish an efficient rehabilitation and maintenance scheme. Such a system can save money and man power while at the same time providing an updated view of the road conditions throughout a city at all times.

1.3 Objectives and Scope of the Research

In general, the primary and main objective of this research is to develop an affordable simpler technique that will not require complex technology, will be suitable for middle-sized road networks, and will be economically viable. More specifically, the objectives of the research can be achieved through the following tasks:

- To develop an affordable automated data collection system that can give an initial assessment of the status of pavement.
- To develop an automated system that can classify the crack types, severities and densities within an acceptable level of accuracy.
- To build simple algorithms that can measure the crack in 2D imaging dimensions; and
- To provide data and information for future reliable modeling of the propagation of cracking of the pavement surface.

1.4 Research Contribution

This thesis developed a system to continuously monitor the road status and rank the priorities of required maintenance. The developed method uses recent advances in

miniature cameras, digital signal processing and wireless networks. The main contributions of the thesis are as follows: a prototype system mounted on a vehicle was developed to collect pavement image data and their location to schedule rehabilitation and maintenance. The automated software analysis was developed to detect the types, severities and densities of the cracks; and different statistical techniques were used to investigate the best weighted average speed to collect the pavement images.

The collected data are continuously fed to a live map of the city roads. This is a novel technique that was not reported in open literature. The “Live Map” is a very valuable tool to city planners and can be linked to an advanced maintenance scheduling scheme.

The vision of this research is to equip public vehicles with a compact monitoring system that continually photographs the road, processes the images and produces an estimate of road status. Locations and road pavement conditions estimation will be sent through wireless data networks to a central control station for rehabilitation priority ranking and scheduling.

The main contribution of this thesis is to introduce this vision and prove its viability by integrating different technologies into an advanced monitoring system. The continuity of the monitoring process would result in a "live map" of the road network quality index. This would provide a method for monitoring the changes of road status over months and years and gives us a way to evaluate the progression and rate of deterioration of the pavement.

1.5 Research Plan

The research plan includes:

(1) Data Collection and

(2) Image Processing and Classification.

The Data collection phase consists of identifying a road section to be tested send a vehicle equipped with data measurement set-up to collect images of the road pavement at many locations. The measurements were taken at different time of the day and at different vehicle speed.

The image processing phase is done on-site. Each image of the road segment was processed using specialized image processing software that identifies the number, type and severity of cracks that may exist in the segment. Each processed image is used to create a "record". The records have the following information:

- Location as determined by a GPS
- Time
- Speed (as determined by a Distance Measurement Instrument (DMI))
- Number of cracks
- Type of cracks
- Severity of the crack damage

The record is compressed and transmitted using a wireless device to a data collection center. The research plan is summarized in Figure 1.4.

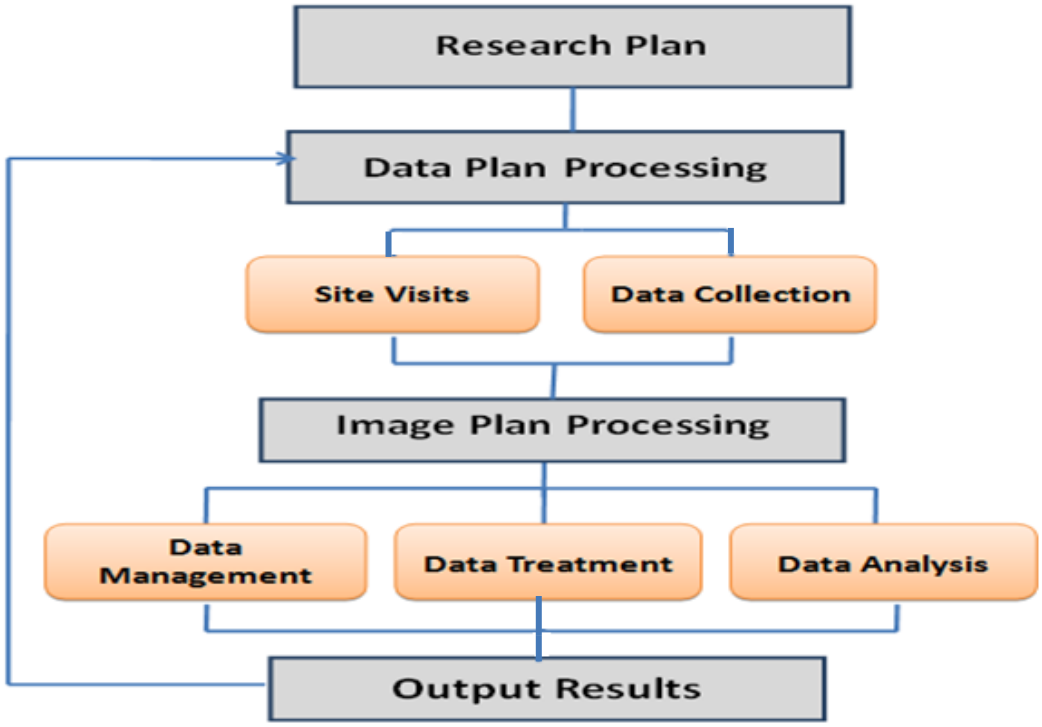


Figure 1.4: Research plans.

1.6 Thesis Organization

This thesis consists of five chapters that cover the main segments of this study. Chapter 1 is the introductory chapter and it presents the background of the proposed research, the problem definition, objectives, research plan and the thesis contribution.

Chapter 2 contains literature review of previous studies related to pavement management systems and methods used to process and analyze pavement images. The description of the prototype vehicle, components, and the developed software process and analysis of the pavement crack images is presented in Chapter 3. Chapter 4 proposes the use of different statistical analysis techniques to analyze the pavement data collected by the prototype vehicle and getting the best results of the average speed for collecting data. Finally, Chapter 5 summaries the major conclusions and recommendations for future work.

CHAPTER 2: LITERATURE REVIEW

This chapter is divided into in four main sections. The first section describes the three types of surveys; namely: manual, semi-automated and automated surveys. The chapter then proceeds to describe the distress detection theories in Section 2.2. This is followed by a section that describes the automated survey software. Finally, the last section contains a summary and a conclusion of the literature review.

2.1 Types of Survey

This section deals with the historical development of condition surveys, their importance within the PMS. Furthermore, this section studies the evolution of pavement condition survey and how it has changed from a fully manual survey, in the early fifties and sixties until the concept of PMS was formulated and presented in Canada and the USA in the late sixties, to a fully automated one in the present time. Therefore, data survey collection methods and techniques are classified as; manual, semi-automated, and fully automated survey methods and are discussed in the following sections.

2.1.1 Manual Surveys

Manual survey of pavement condition is defined as the method that is performed by humans and it consists of three procedures; these are:

(a) Walking while inspecting the pavement by eye,

(b) Windshield survey and

(c) Combination of the walking and windshield survey.

A walking survey is conducted by a trained rater to measure the distresses according to road authority distress specifications. Examples of pavement condition forms observed in this procedure are shown in Figure 2.1. The walking rate fills in a form

that contains a description of severity and density of each distress identified on the pavement. [30]

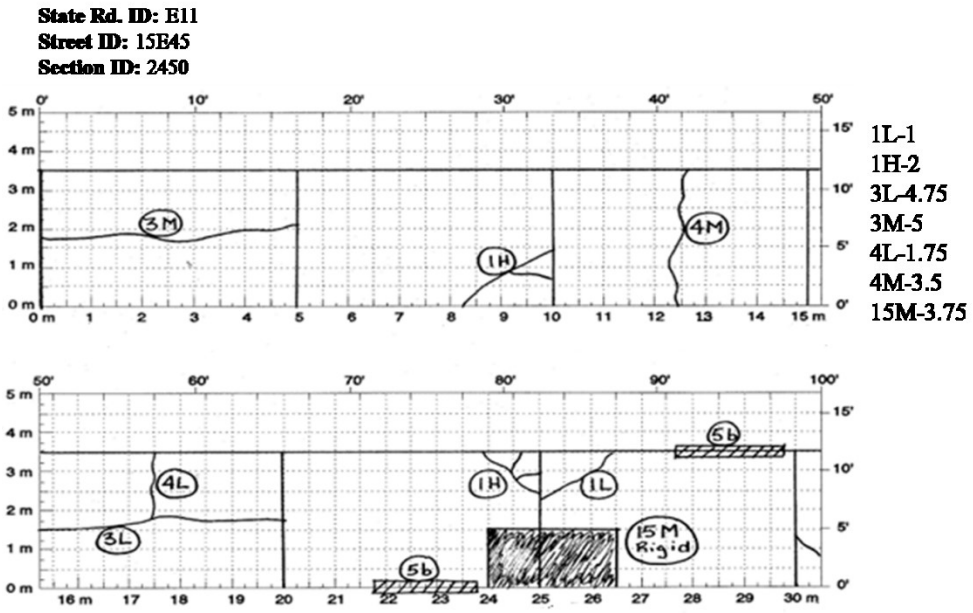


Figure 2.1: Forms of the rater record distress types, severities and densities (reproduced from [30]).

The most precise data collected on condition of the rated pavement depends mainly on the degree of training and the experience of raters. Because of the amount of walking in this type of survey, it is considered to be time consuming and therefore only a small fraction of the road networks can be studied this way. In order to save time, the pavement network is usually divided into representative segments where sample selections would be taken to represent the entire network. For example, surveying the first 100 m. of each kilometer can be considered as a proper representation of the entire pavement network. Under the manual survey method, sample selection depends on any one of the following options:

(i) Sampling at fixed distance intervals,

(ii) Making random selection, and

(iii) Having a rater picking up representative samples.

Random selection is not a very good option because random sampling could lead to the selection of road sections subjected to prior maintenance. As a result, random sampling may miss distresses that need to be identified and dealt with. In general, the sample size must be large enough to obtain a reliable assessment of the road conditions [8].

The other manual survey procedure is called a "Windshield survey". A windshield survey is defined as a manual survey to collect the pavement condition data by using a vehicle [31]. Vehicles can be driven along or on the shoulder of the road, and the surveyor rates the pavement through the windshield of the vehicle. This method covers more amounts of data in less time than the walking procedure; the quality of the data is less. Furthermore, with an average speed of travelling the human eye cannot distinguish specific types of distress that require closer eye contact or specific angle of eye projection, [32].

The third procedure for collecting pavement condition data is a combination of walking and windshield surveys. This compound procedure according to [33], [34] is considered a good procedure to inspect more pavement data and get a greater percentage of the network. In the study reported in [6], it was suggested that the compound survey would be satisfied only, if the same procedure were applied: [i] on every section in the network, and [ii] a random method is applied to select the sample where the walking survey will be performed.

In the Oregon Department of Transportation (ODOT) [35] conducted manual survey procedures by two raters trained in distress identification manuals. Data condition survey of the pavement road was collected and accomplished via a side window survey from a slow moving vehicle along the shoulder of the roadway. The safety of the entire operation is considered the most critical control for collecting the required data either by vehicle or walking surveys.

In general, the pavement condition data can be recorded using one of the following methods: [i] paper forms for later entry into the database or, [ii] by portable computer which can be held in hand or mounted in vehicle for riding survey. Pavement condition data can also be entered by the terminal keyboard or special keyboard on which distress types and severities have specially marked keys. Then the pavement condition data are transferred electronically to a computer in the engineer`s office. Additionally, there is a new clipboard to record data using a hand held microcomputer, or a pen-based-computer (electronic clipboard) that the rater writes and makes a check on the screen which transfers and records directly into computer which reduces data errors [36].

2.1.2 Semi-Automated Surveys

In [19], [37], there is a description of a semi-automated survey method which used a moving car to record pavement crack images by one or two cameras mounted in front or back of the moving vehicle. Pavement images were then transferred to the hard drive in order to identify and classify the distress types by the appropriate personnel. The raters were sitting at a workstation that uses image players that integrate distress rating, location reference software, as well as accessed image and database files. Pavement images were recorded on film, tape or digital cameras. In a workstation, the rater used

multiple image monitors to provide his/her perspective for location purpose and to define and determine distress types and classification. GPS was used to identify the locations of recorded images by considering real coordination's. Additionally, time and date were recorded using the GPS devices. Figure 2.2 shows a generic schematic of the semi-automated and automated pavement cracking analysis. Figure 2.3 shows the image analysis workstation. Table 2.1 summarizes the reported processing methods in use by the various agencies [19], [38].

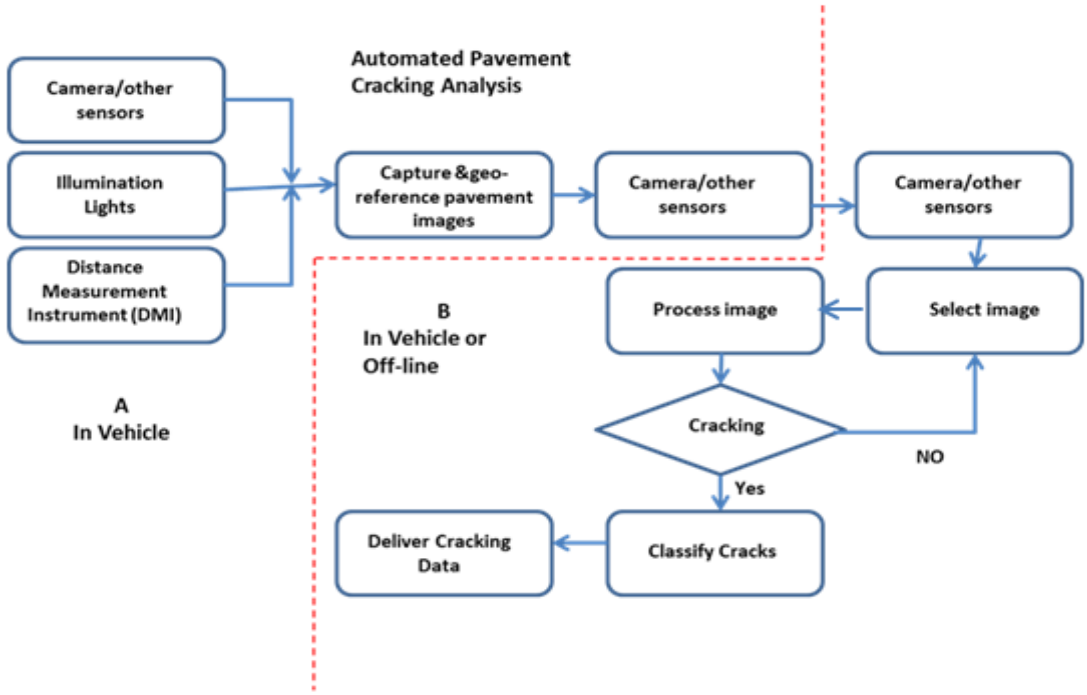


Figure 2.2: A generic schematic of the automated and semi-automated pavement cracking analysis (reproduced from [19], [38]).

Figure 2.3: Pavement data Image analysis workstation (reproduced from [38]).

Table 2.1: Image Processing Methods (reproduced from [38]).

Agency Type	Visual Survey	Semi-Automated Survey	Fully Automated Survey
State	16	1	7
Province	3	1	0
FHWA	1	------	1
Total	20	2	8

The Virginia Department of Transportation (VDOT) found that the pavement cracks are more visible during a semi-automated survey than walking or windshield survey. Also, patching that is well maintained may not be identified from the recorded images. These problems led VDOT to issue a pavement distress rating manual specific to videotape images. There are differences between semi-automated survey that use videotape and manual survey. The semi-automated survey can identify some distresses type and severities. There are two systems of detection of the pavement distresses used

by VDOT: [i] Pavetech, which provided a video tape system and [ii] Transportation management technologies, which used digital area and line scan images [39].

In a study by [40] various systems used for data collection, the first system was developed by a Japanese consortium and termed Komatsu in 1980s. This system consisted of a survey vehicle and board data processing unit in order to survey and measure surface cracking, surface deformation or rutting and longitudinal profile with an optimum resolution of 2048 x 2048 pixels. The vehicle operating speed was 10 km/h. The system was applied at night in order to ensure better control conditions such as avoiding heavy traffic and limiting the effects of daylight related shadows. The system represents an implementation of the most sophisticated hardware technologies at that time, but it failed to produce different types of surface cracking and only worked during the nighttime. Figure 2.4 shows Komatsu Survey Vehicle [41].

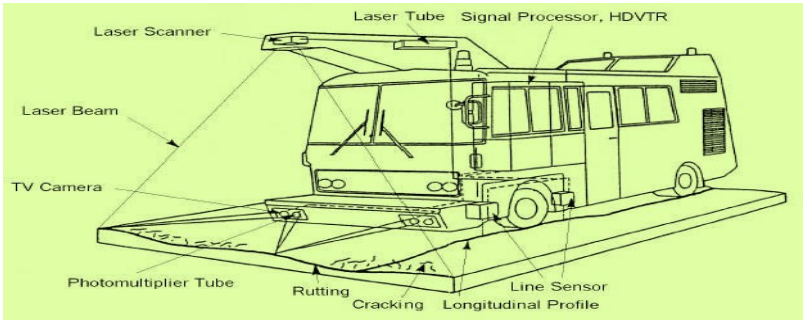


Figure 2.4: Komatsu survey vehicle (reproduced from [41]).

The second automotive system is called Pavement Condition Evaluation Services (PCES) Research Unit. The system was created by Earth Technology in California, starting in the late 1980s through to early 1990s. The system contained a line scan camera at 512 pixels resolutions and was firstly used by this unit to establish a fully automated

system. However, efforts were terminated due to poor performance of the technologies at the time of the development of the system. The used technique was not advanced enough for image recording and processing. Figure 2.5 shows the Pavement Condition Evaluation Services (PCES) Research Unit survey vehicle [38], [40].

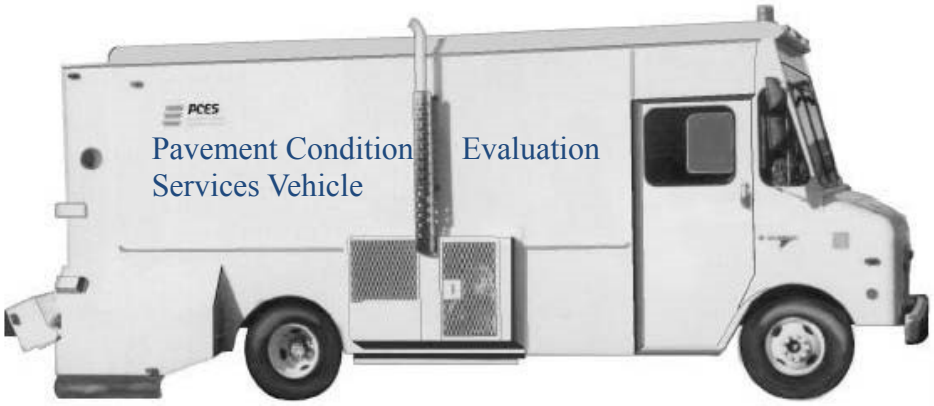


Figure 2.5: The PCES survey vehicle (reproduced from [40]).

The third type or system is the Swedish Pavue Pavement Data Acquisition System (PAVUE). The system was available for use in North America in the late 1980s and early 1990s. The system included four cameras with proprietary lighting system and four S-VHS videocassette recorders. This system was upgraded by adding its facilities to a laser road surface tester van. The Laser road surface tester (RST) is a multi-function testing vehicle that is used by Infrastructure Management Services (IMS) in North America and was developed in Sweden. The Laser RST consists of video cameras, accelerometers, laser sensors, a distance measuring instrument and a computer system. The system utilizes laser technology to determine pavement distresses, define the road profile, measure surface roughness, measure the rut depth, and quantify macro texture of the pavement surface in question. A subsystem that consists of a set of specially built processor boards is installed in a cabinet at the office as shown in Figure 2.6. The

subsystem is designed to analyze the data obtained from the recorded videotapes in analog format. The Pavue system also has a resolution of 1400 pixels per line at operating speeds of 10 to 90 km/h which can detect the presence of 2.5 mm crack size. As more advanced camera and board processors technologies became available on the market in North America, new systems, were developed and are in use [42].

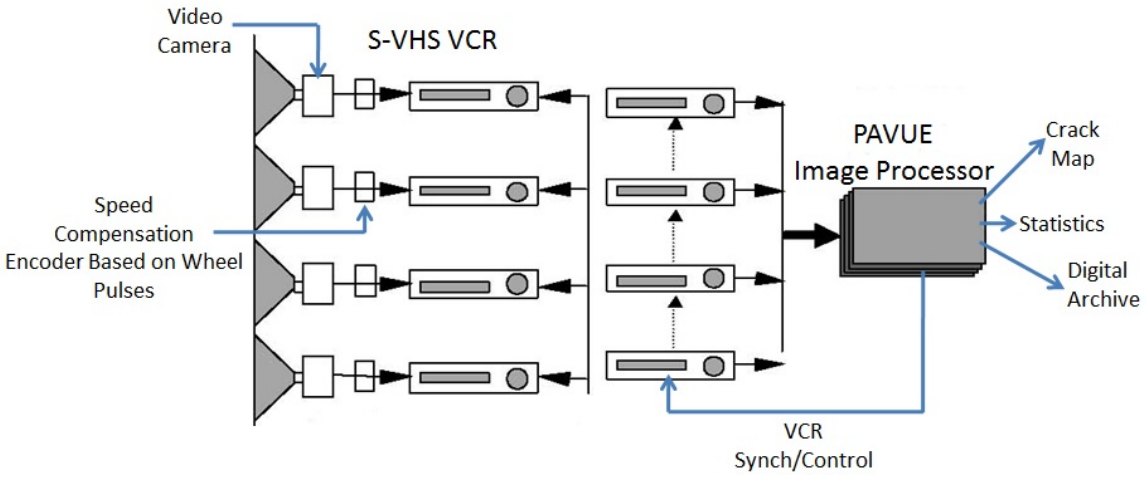


Figure 2.6: Data Flow in IMS's PAVUE System (reproduced from [42]).

Another semi-automated pavement-distress analysis system was established by Lockheed-Martin in Arizona in 1995 which had digitized images from Pasco`s 35 mm film [40].

There are several systems that have special capabilities for identifying the severity and density of pavement distress types. The main differences among the systems are in the software's features and the level of human interaction. For example, pavement distress analysis system (PADIAS) designed and produced by CGH engineering in Mechanicsburg, Pennsylvania, was used and applied to Long Term Pavement

Performance (LTPP) program projects. The system makes use of digitized images by which a trained operator can analyze the distress data files and convert them to ASCII format before downloading into the database. The system is built into the LTPP distress data base that researchers and engineers can use and modify to meet their needs [43].

Another system is presented by pathway services in Tulsa, Oklahoma in 1996 known by the commercial name of Path View I and it is equipped with a video and sensor playback system. The system includes six videotape players and monitors and example samples of each distress mode showing its severity levels to maintain consistency between evaluators. After analysis, database is created and a detailed distress features is included [44].

D-Rate digital distress rating system is another semi-automated rating system. In this system, the operator can utilize the location, classification, severity, and subsequently determine the density of a specific distress type from digital images. Then the collected data are fed to the computer to calculate both the lateral and longitudinal coordinates of distresses and the area or length of the determined distress. Also, the system can define the wheel path with respect to the reported distresses [45].

Roadview GD Plot is another type of semi-automated system that was developed to calculate the 3D digital imaging processes. The system is considered not fully automated because there are no features allowing distress summaries to be developed and reported [46].

Another type of semi-automated system that can be used to identify transverse cracking is used by the Kansas Department of Transportation (KDOT), International

Cybernetics Corporation (ICC) digital imaging and distress measurement analysis software is the main components of this system. The system consists of digital line scan camera of 2048 pixels and a computerized controller and illumination system allowing the recording of surface cracks up to 1.0 mm while travelling at speed of approximately 100 km/h. A manual backup method is used to measure crack severity and density from Kansas department of transportation`s (KDOT) algorithms [47].

The Department of Civil and Environmental Engineering at the University of Catania in Italy in 2006 presented a low cost image acquisition and analysis system. This system was used manually to identify cracks, patches and potholes. The system could carry out its designed functions under normal traffic conditions or speeds of 25-30 km/h and cracks identification width of 3.0 mm or wider. The system consisted of a video camera with CMOS sensor (Complementary Metal-Oxide-Semiconductor are a technology circuits used for the image sensor in the camera) at resolution of 1288x1032 pixels, frame digital process, 16 MHz acquisition frequencies at 9.3 fps. This system is supported by software used to allow image acquisition to be synchronized to the GPS survey by special external trigger sent by the GPS. The results showed that the cracks, patches and potholes can be identified automatically [48].

As can be noted from the above discussion semi-automated systems have been developed and utilized to perform condition surveys of pavement surfaces for more than two decades. However, there are several common features governing these systems which inhibit their use in countries and regions with small size road networks, limited trained manpower and/or limited financial supports.

2.1.3 Automated Surveys

In contrast to the manual and semi-automated techniques, automated methods are equipped with a camera which can be used to perform pavement crack surveys without interrupting the normal traffic. Subsequently, software packages can automatically analyze pavement crack images and identify cracking patterns present on the pavement surface by type, severity and density. The new development of the automation crack detection has adopted an advanced technology of video imaging and pattern recognition. This new technology has resulted in the full automation of video imaging and pattern recognition of pavement crack. Therefore, eliminate any possibility of human bias or error. Table 2.2 shows the results of comparing the relative costs and characteristics of various image media types.

Table 2.2: Image media comparison (reproduced from [49]).

Media	Relative Cost	Lines of Resolution
35 mm Film	High	1700-3500+
VHS	Low	250+
¾ inch Video Tape	Low	340+
S-VHS	Low	400+

There are many automated systems used by various departments of transportation across the world such as: ARAN, the Pavetech VIV unit, MHM associates ARIA system, Pavedex`s PAS-1 device, the Australian road evaluation vehicle, Highways Agency Road Research Information System (HARRIS) [50] and the videocomp trailer use the videos to record pavement images [51].

The automated road analyzer (ARAN) vehicle is one of the most reliable automated systems in the world today. It is a high speed, multi-functional and diverse road data acquisition vehicle. The unit has capabilities to measure pavement condition and distress while travelling at reasonable speeds. The vehicle is equipped with various data collection subsystems that can collect a variety of data and road related information. One of the data collected is digital images of the pavement surface. The system collects its data through two high resolution monochromatic cameras attached to the rear of a vehicle that scans pavement surface with strobe lights synchronized with the cameras. The two cameras are synchronized and the software overlaps and stitches two images in real time. WiseCrax is automated crack detection software that can detect crack length, width, area, and orientation and classify them according to type, severity and density. The output is crack maps and summary statistics. Digital Rating (D-rate) is the system that allows the rater to identify distresses and measure distress quantities. The rater uses a mouse to choose certain distress type with abilities which allow users to assign labels and pre-defined distress types as point, linear or area measured prompting software to allow dot, line or box drawing as applicable with allowing user to review or edit at any time. Figure 2.7 shows the ARAN automated vehicle [52].

Figure 2.7: ARAN automated vehicle (reproduced from [52]).

Another system is known as waylink (digital highway data vehicle). The system can provide images in real-time of surface cracks as small as having a width of 1 mm and can use two different manuals: AASHTO and Universal cracking indicator and Texas Department of Transportation (TxDOT) methods. The system is manufactured by waylink and consists of [26], [53], [54], [55]:

- Two lasers to generate line lights;
- Two digital lines scan cameras which are attached to a precision beam connected to the rear of a digital highway data vehicle (DHDV) with 13 ft. transverse coverage of the pavement surface;
- Automated Distress Analyzer (ADA) software package for image analysis of pavement surface distresses; and
- The vehicle is equipped with on-board computers, intercomputer communications techniques, multi computer and multi control processing unit based parallel computing and capabilities to generate multimedia database. Figure 2.8 shows the DHDV vehicle.

Figure 2.8: DHDV vehicle (reproduced from [54]).

2.1.4 Comparison between Manual and Automated Survey Techniques

Pavement condition data survey using manual methods is a labor intensive and unsafe processes when performed on major roads and highways. The availability of new and innovative technologies has provided safer, less labor intensive, more cost effective and time efficient automated techniques. The development of the new technology allowing full automation of pavement crack surveys reduced the dependency on the traditional manual methods and resulted in wide adoption of the automated detection and classification of pavement surface distresses. The technology advancement in computer hardware and imaging recognition techniques has offered better opportunities and wide choices for developing cost effective and more reliable surveys. Clearly, such developments improved the process of evaluating the pavement surface condition, supported making more rational management decisions, and ultimately led to maintaining the national road networks at affordable costs for countries such as Canada and USA.

In a study by [56], a comparison between manual and automated survey was conducted and concluded that the manual survey demands intensive labor and subjects its manpower to unsafe situations in contrast to the less labor intensive, faster and safer automated surveys. Furthermore, the results of the comparison study suggested that while the distress type and density were logical regardless of the method followed, the selected technique influences the measured severity. The analysis showed that the automated system was much more capable to address lower severity values. There was an average of 3.2 with a maximum discrepancy of 5 in the calculated values of the PCI of manual and automated surveys. An interesting observation was that the costs of using either manual

or automated methods to perform the distress survey on a 405,000 yd² pavement areas were similar.

The results of several studies comparing between the manual and automated surveys are summarized below [56], [57], [58]:

- The results from manual and automated surveys are similar when surveyed cracks are not extensive;
- No significant differences were reported when the cracking severities are in the range of small to moderate;
- When universal cracking indicator (CI) values are adopted to measure the cracking distress, automated survey resulted in a higher values than manual survey; and
- The differences between the two methods appear to increase when the tasks and efforts required performing the condition surveys intensified.

2.2 Cracking Detection Theories

The previous review showed that the most important factors determining the reliability and cost effectiveness of the type of condition surveys are the size of the survey and the severity of the distress type. It is reasonable to assume that for small road networks with light traffic and stable moderate environmental conditions the selection of manual survey is a better choice. On the other hand, for large sized road networks with heavy traffic and severe climatic conditions automated or semi-automated surveys equipped with advanced high speed facilities are the most reliable selection. However, for medium size networks more economical and safer techniques will be needed. In order to maintain the quality of the pavement surface and provide the public a good riding quality,

pavement distress identification is the most important part in the highway maintenance practice. Pavement cracking and rutting are the two most recognized surface distress modes and maintenance plans depend very much on identifying these two types of distress.

Cracking distress is considered the main distress which draws more attention in applying image recognition processing during automated surveys. In general, surface cracks in pavement images are disconnected and not clear because of difficulties associated with the image recognition which leads to an inability to identify the cracks correctly.

The procedure of classification pavement distress images is performed by segmenting the image into blocks and determines the distress in every block. The results of this step provide potential for the development of a set of models for addressing the distress types of longitudinal, transverse, block, alligator and plain cracking. Figure 2.9 shows images of a block and an alligator crack (after background subtraction). In addition Figure 2.10 shows the result of the probabilistic segmentation [59].

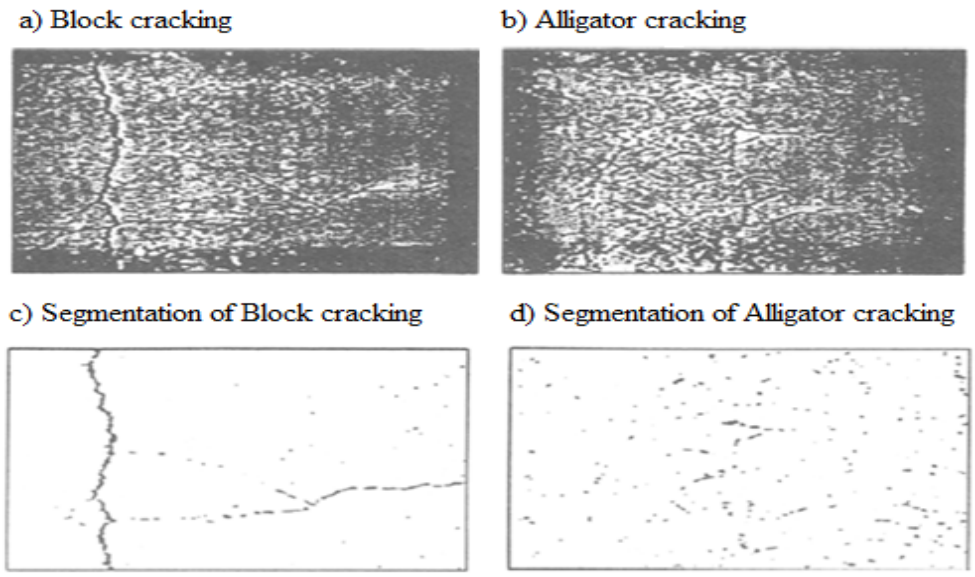


Figure 2.9: (a) Image of block cracking; (b) Image of alligator cracking; (c) Segmentation of block cracking image; (d) Segmentation of alligator cracking image (reproduced from [59]).

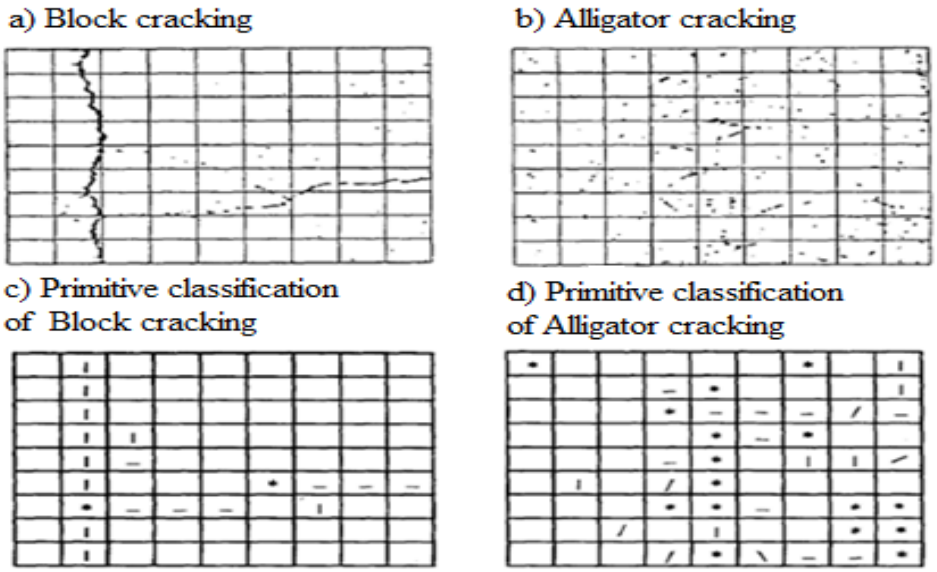


Figure 2.10: Segmented image superimposed by 9 x 9 grids: (a) Block cracking; (b)Alligator cracking; (c) Primitive classification of block cracking Image; (d) Primitive Classification of Alligator Cracking Image (reproduced from [59]).

In another study by [60], video images of asphalt concrete pavement surfaces were used to compare between the evaluation of traditional and neural network classifiers. In order to detect cracks bayes classifier and the k-nearest neighbor (k-NN) decision rule are used in this study. The neural classifiers are the multilayer feed forward (MLF) neural network classifier and two stage piece wise linear neural network classifier. The researcher used an evaluation methodology in order to check the performance of bayes classifier and (k-NN) in detection and classification of crack segments in the images. The results show that the neural-network classifiers performed slightly better than the other classifiers on the test data set with careful parameters selected and extensive empirical training performed. Figure 2.11 shows the stages of image segmentation; feature extraction; decomposition of the image into tiles and identification of tiles with cracking; integration of the results and classification of the type of cracking in each image; and computation of the severities and extents of cracking detected in each image [61].

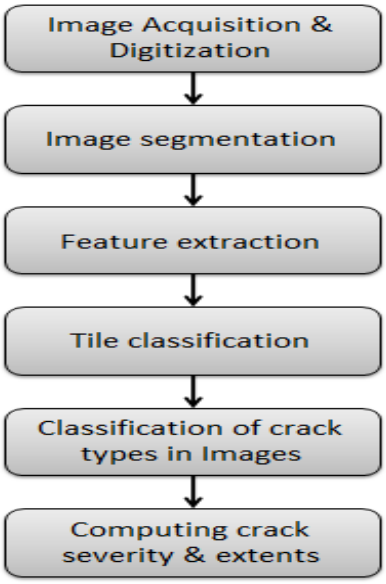


Figure 2.11: Stages of video-image-based automated pavement surface distress evaluation (reproduced from [61]).

2.3 Automated Survey Software

Asset management system such as pavement management system supports the engineer with the tools necessary to accomplish cost-effective management of a roadway network. There are many pavement management software packages developed by various pavement engineers and researchers that are capable of identifying pavement distress types, severities and density of distress as well as analyzing pavement data to detect pavement distress types, severities and densities. Analyzed data can be stored in readable format or in GIS map forms that show the ranking of collected data according to certain manuals and guidelines.

One of these systems is Uni-analyze software that can analyze the collected pavement images. Uni-analyze software measures the pavement distress types, density and severity of cracking manually and automatically. The software is used to analyze various types of pavement surface distresses with digital image processing techniques which can be used with new the SHRP-LTPP distress identification manual [25] and the AASHTO provisional standards [26] to be recalled from a computer screen. Then, a color coded digital map presented by the system which analyzes pavement condition data after processing the distresses. The system is cost-effective and user friendly to set up for maintenance and rehabilitation scheduling. Also, the system is capable of measuring the quality assurance for automated crack detection systems and validating the Unianalyze system and other systems as well. Two steps were used to develop the software; image segmentation and cracks classification. Image segmentation is applied to divide the pavement image into specified girds to take care of lighting condition in pavement surfaces and removing the noise in pavement image by noise filter. Statistical properties

of pavement images are used to reduce the errors caused by distinctive edges of any white line and distinguishing the cracked girds from background girds done by crack detection girds. Three different crack classification manuals standards were used (SHRP-LTPP [25], AASHTO [26] and Unified cracking index [62]) and the software can implement another standard upon the need of transportation agencies. The rater can measure the distress using a mouse on a computer screen by tracing a line along a crack. Figure 2.12 shows a sample of the unified crack index and 2.13 shows the example of a computer screen, which displays the analyzed result on a digital map [63].

Unianalyze cracking software

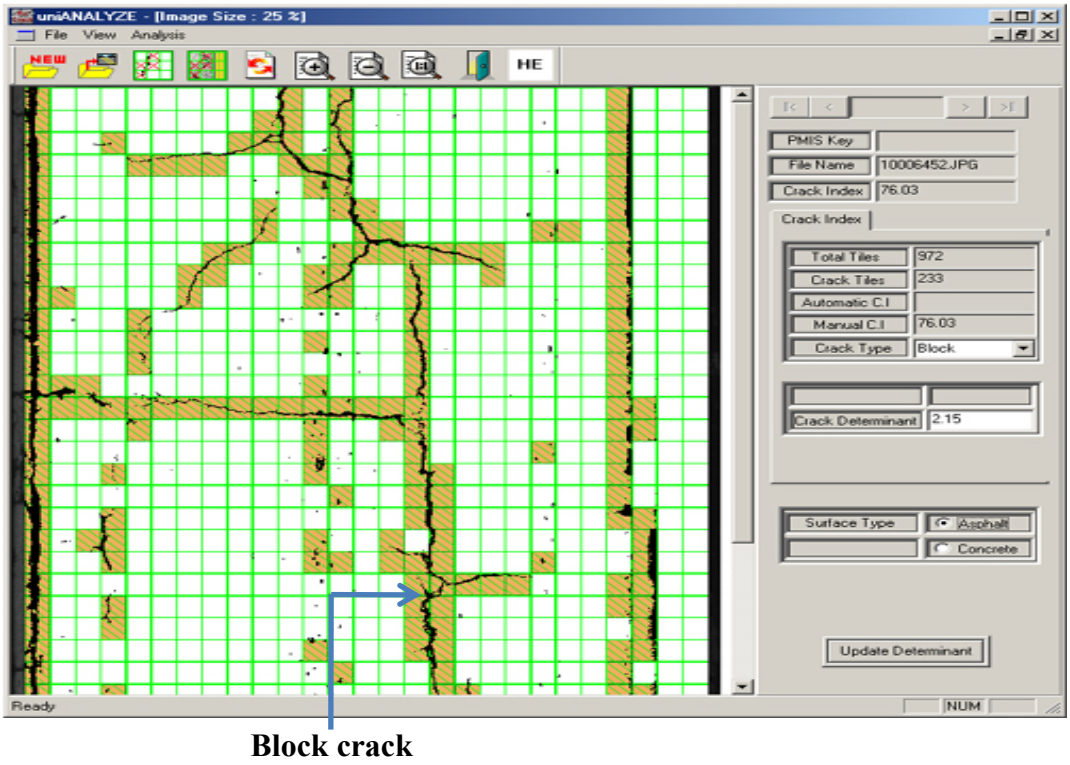


Figure 2.12: The unified crack index analysis window displaying a processed unified crack index of 73.03% (100 - 233cracked grids out of 972 total grids) (reproduced from [63]).

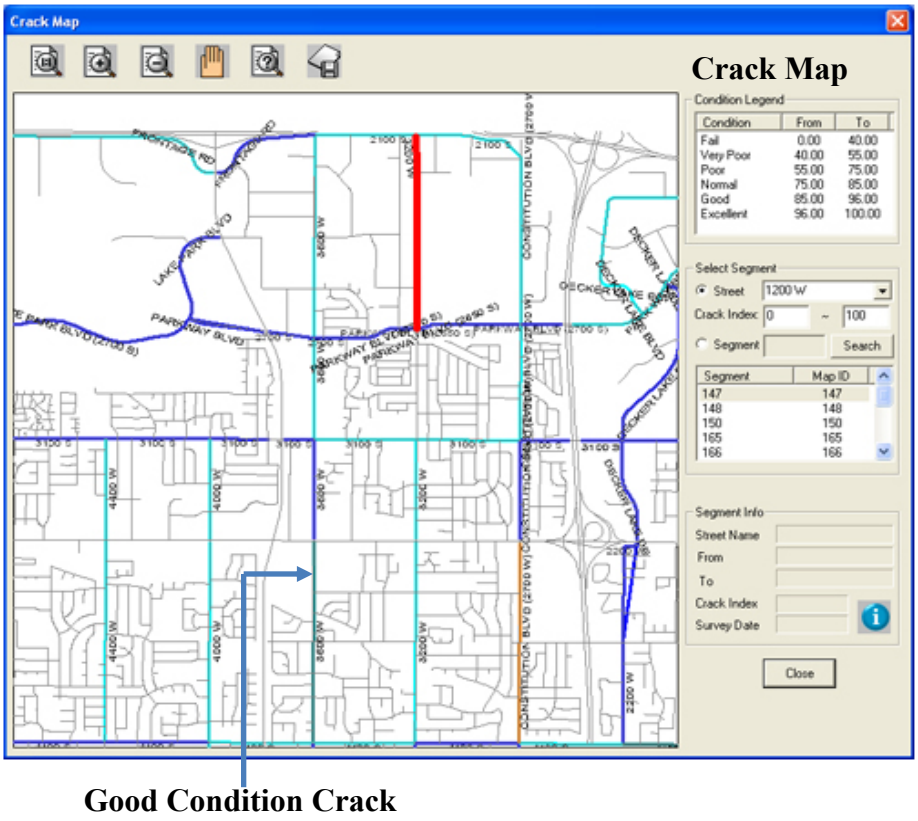


Figure 2.13: Display of processed result on a digital map (reproduced from [63]).

Collected pavement distress images analyzed by Automated Distress Analyzer (ADA) can detect longitudinal, transverse, block and alligator cracks. The length, direction and widths of flexible pavement were calculated. The ADA processed the images in real-time with the vehicle speed up to 100 km/h with the capability of detecting cracks of 1 mm and wider. The ADA can be used as post-processing. The ADA provides manual processing of distress data using manual rating tools with availability of measuring and defining criterion based on the distress identification manual for long term pavement performance program (LTPP) and PCI methods. The ADA can be integrated with GPS data. Figure 2.14 and Figure 2.15 show ADA examples of longitudinal and alligator crack analysis [64].

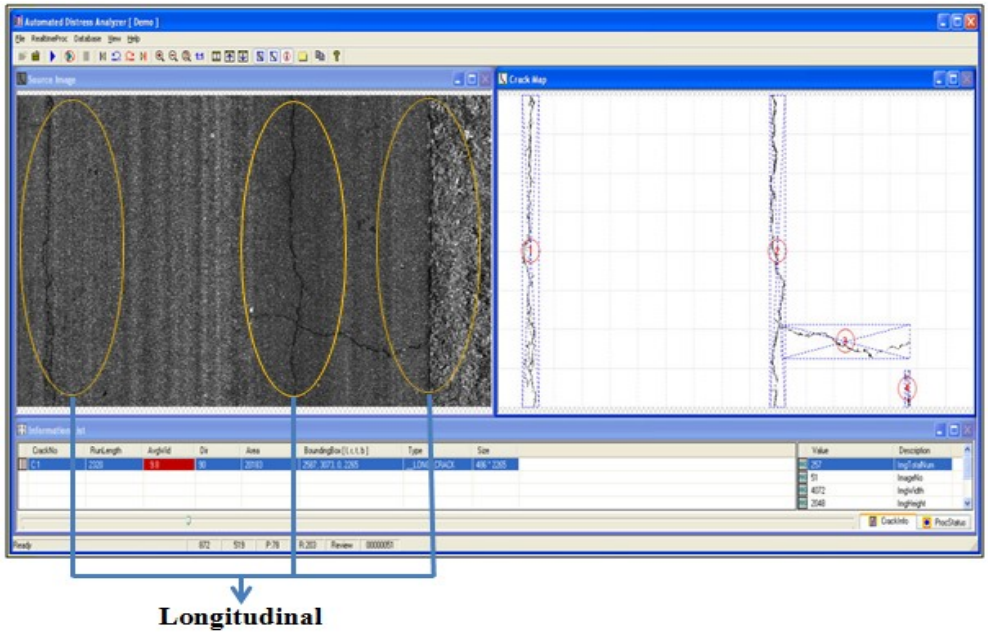


Figure 2.14: ADA example of longitudinal crack analysis (reproduced from [64]).

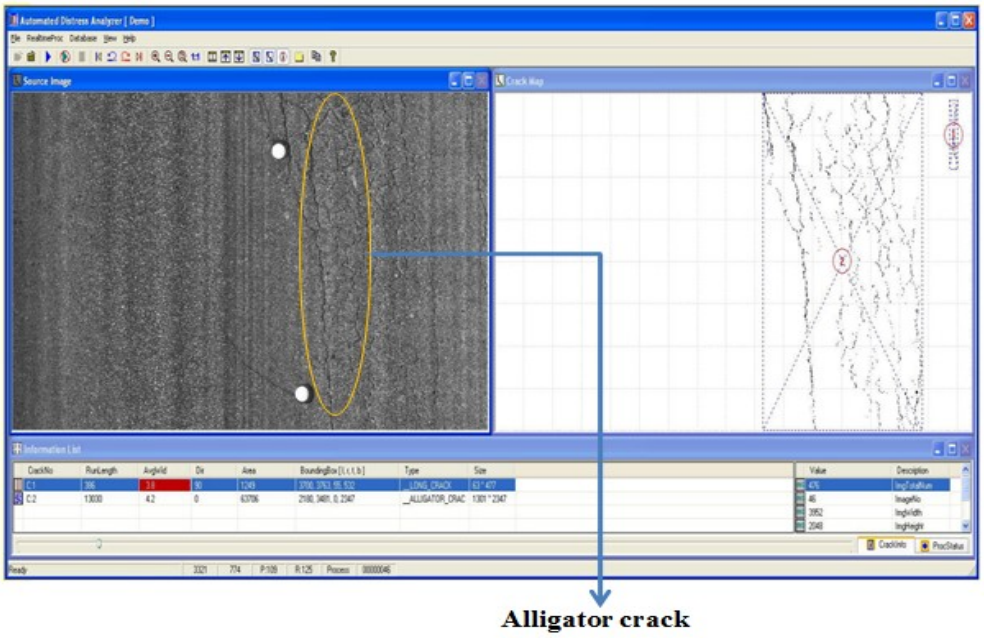


Figure 2.15: ADA example of alligator crack analysis (reproduced from [64]).

One of the most commonly used advanced automated systems for performing distress surveys is the Automated Road Analyzer (ARAN). The system is developed by European research teams in Sweden and is equipped with two high resolution monochromatic cameras with strobe lights synchronized with the cameras to provide the require lighting. The left and right cameras are synchronized and software properly overlaps and stitches the two images in real-time. The system covers the pavement area of 4.9ft.x14ft and has the ability to take images of cracks with width from 2 to 3 mm at speed of 80 km/h. The system can display the detected cracks automatically (longitudinal, transverse, block and alligator cracks) in a window and the detected cracks are overlaid on the pavement image. The crack type, severity, density and location and maps of cracks can be produced and printed with the available option of adjusting the architectural design to suit distress criteria. The second method can be evaluating the collected image which is called Digital Rating. The Digital Rating allows the rater to identify individual distresses and measure quantities by using the images of allowing dot, line or box drawing to assign labels and predefined distress types. The data analysis can be reviewed and edited any time. The system can define the location of distress by GPS and distance measuring instrument. Figure 2.16 shows the wisecrack screen analysis and image snapshot [52].

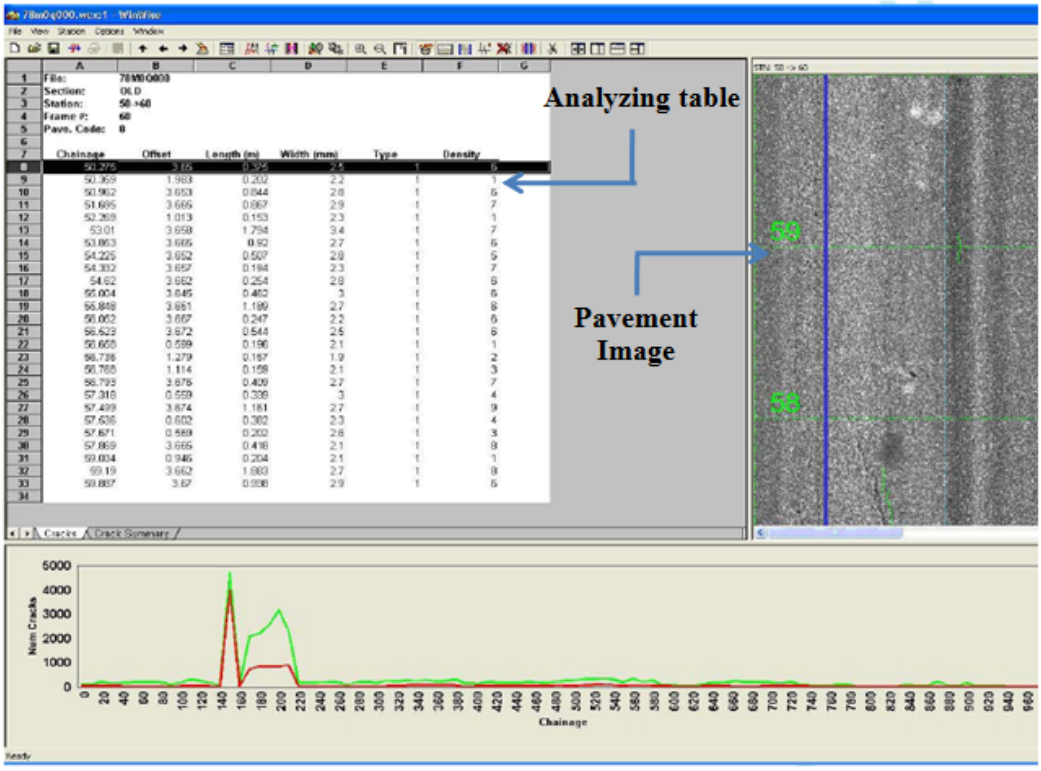


Figure 2.16: Wisecrack analysis screen (reproduced from [52]).

2.4 Summary and Conclusions

This chapter presented an overview of the types and evolution of three types of distress condition survey. Traditionally, detection of pavement distresses and their classification is easily determined and reported by trained humans. However, with increasing amounts of paved roads, growing traffic volumes and speeds coupled with higher risks associated with the safety of manually performed surveys the pavement industry focused on other approaches which took advantage of new technologies and advanced development in software capabilities. Semi-automated and automated condition surveys can remedy most of the problems associated with traditional techniques, but the available technologies and systems are very costly to purchase,

operate and maintain especially when the road network is fairly large or there is shortage of highly trained personnel.

This research attempts to fill the gap between the traditional time consuming methods and their labor intensive characteristics and the newly developed methods rely on fully automated systems. It is therefore proposed herein to select a system which will provide the following capabilities:

(1) Develop a detecting system utilizing the available image processing technologies;

(2) Develop software packages to analyze the collected images; and

(3) Provide a mobile system with economic day-to-day automated capabilities.

The proposed system will be suitable for networks with small to medium size road networks and municipalities which lack large sums of money to invest in acquiring complex and costly systems similar to the ARAN vehicle. The details of the research plan are discussed in the next chapter.

CHAPTER 3: DEVELOPMENT OF A PAVEMENT CRACK MEASUREMENT SYSTEM

3.1 Introduction

The discussion and results of the literature review demonstrated that there is a need to develop a simple effective automated technique that is reliable and affordable for providing daily data and information related to pavement crack surveys. Also, it is important that the developed technique will require minimal manpower and can run without interruption to regular traffic operations. In order to meet these requirements and achieve the main objective of this research, it was proposed to develop a novel and innovative approach for collect and process data related to the status of the road conditions. The followed approach was to create a live map of a city where each section is ranked on a quality scale, which is termed referred to here as “Pavement Indicator". The term “live” is important to emphasize as the indicator will be continually updated by an automated method. The basic steps involved in creating the live map are the following:

(1) Public transit and/or service cars (e.g. buses) would be equipped with a system consisting of cameras and a wireless communication device. The cars will take snap shots at randomly in many locations throughout the day and transmit the images to a central office;

(2) Each image will be analyzed to determine the approximate conditions of the section it depicts. The conditions are indicated by a single Pavement Indicator (PI). It is understood that this index is not very accurate due to the use of inexpensive equipment;

(3) The images which are collected at a central office are combined in a statistical fashion to give approximate PI's to all locations; and

(4) Although each image may produce a noisy index, the cumulative effect of many snap-shots in the same location or overlapped locations, will improve the accuracy of the PI estimates over time, and the different sections within the area of interest are then ranked based on their quality conditions. The images with the lowest PI (poor pavement indicator) will be signaled out and revisited for more accurate assessment and subsequently help making proper maintenance decisions. The research plan is shown in Figure 3.1.

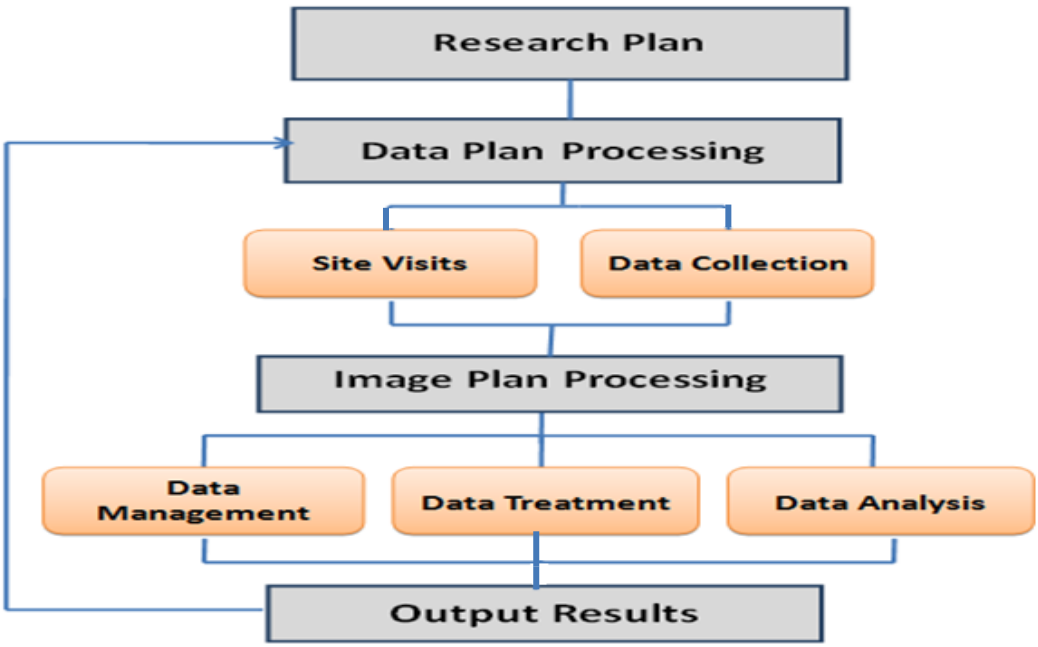


Figure 3.1: Research plans.

In this chapter, four main sections are described. The first section includes the system components description. The second section deals with the image processing. The third section includes the image analysis. The last section describes the software development steps.

3.2 System Components

In order to collect data automatically, an automated measurement system was mounted (workstation) on a prototype vehicle. A picture of the automated measurement system is shown in Figure 3.2. The automated measurement system consists of:

- Two cameras (cam#1 and cam#2).
- Illumination lights.
- Global Positioning System (GPS).
- Distance measurement instrument (DMI).
- Power supply and power distribution system (inventor).
- Computer with ”image processing software”
- Wireless devices

The description of each component is provided in the following sections.

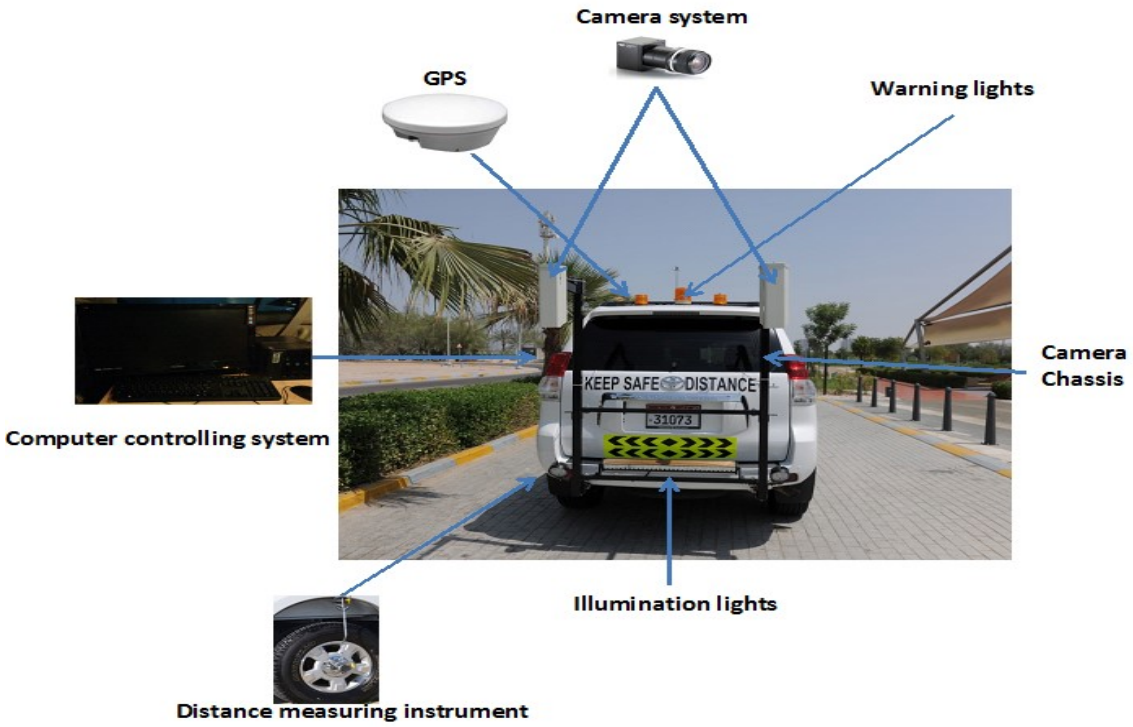


Figure 3.2: Automated data collection vehicle.

Figure 3.3 illustrates the overall measurement systems.

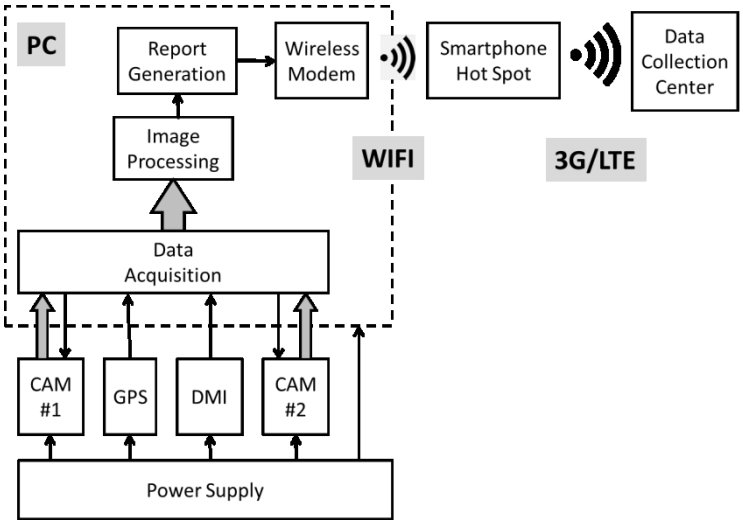


Figure 3.3: Measurement Setup.

3.2.1 Camera Image System

In choosing a camera suitable for the objectives of this research it was important to define the image properties. The most common width of an Abu-Dhabi standard lane is 3.75 meters. In order, to ensure covering the standard lane width of 3.75 meters, 4 meters wide was selected for this research project. Since two cameras were used to cover the lane, each camera had a beam-width that covers two meters on the ground. It was also decided that the basic ground segment covered in one measurement is 4 x 10 meters. The 10 meter measurement was a reasonable compromise between very short and very long segments. Since the cameras were mounted on the back of a car or a bus at fairly low height above ground, a 4x10 image be produced using a single shot. Therefore, it was decided to choose cameras that could take a sequence of images while the car was travelling at a certain speed and then stitch these images together to produce the required image. A third consideration was the resolution and shutter speed of the cameras. The system was being designed such that images will be snapped while the car is travelling at a speed ranging from 20 to 100 km/hr. This requires that each image of the sequence of images be snapped with very fast strong exposure before the car displacement due to motion affecting the quality of the image pixels. There were two types of cameras:

(1) Line Scan Camera: that captures the image line by line very quickly. Each line consists of a fixed number of pixels. The software that comes with the camera can link up the lines together to form the complete image; and

(2) Area Scan Camera: that captures the whole image at once as a matrix of pixels. This is the most widely used kind of camera and is more suitable for

nearly stationary objects since capturing the entire area all at once requires that the shutter stays open longer than the a line scan camera.

For the reasons mentioned above, the selected camera was a model Dalsa Spyder 3 (DS3) which is a line scan camera with resolution of 2048 or 4096 pixels per line. Other properties of the DS3 include:

- Total data transfer rate of 80 MHz,
- Maximum line rate of 36 kHz,
- Operations temperature is from 0 to 65°C; and
- 14 μm pixel sizes.

Two cameras were used and mounted outside the back of the prototype vehicle. The camera image system has capability of withstanding shock, vibrations and environmental impacts. Figure 3.4 shows the camera image system position mounted on the prototype vehicle.

a) b)

Figure 3.4: Camera mounting system.

3.2.2 The Illumination lights

Illumination lights were added to the entire system with the two cameras in order to improve illumination of the sections during the survey. The illumination lights help with producing picture with consistent quality regardless of the natural lighting of the day and the lights also minimize the difference between photos taken at night and those which were taken during the day. The illumination lights system is a Light Emitting Diode (LED). Figure 3.5 shows the position of LED connected with the prototype vehicle.

Figure 3.5: Illumination system for automated data collection.

3.2.3 GPS System

A Global Positioning System (GPS) was used in order to determine the coordinate's location of the section that was captured by the cameras. The GPS provided a reference number and coordinates to each captured photo. The reference number was used to manage all captured photos. The coordinates include latitude and longitude of the

section of interest. Figure 3.6 shows the position of the GPS connected with the prototype vehicle.

Figure 3.6: Global Positioning System (GPS).

3.2.4 Wireless and Computer System

The set up included a regular PC. One of the computer expansion slots has a data capturing card. The card was connected to the cameras, the GPS and the DMI. The captured images were passed to the Digital Signal Processing software (written in C++) [65], to process each image and extract the required road quality parameters. The computer was also equipped with a WIFI card used to establish connection with a central office. The wireless link is set up using a hand-held smart phone that offers a "personal hot spot". The smart phone was connected to the Internet using 3G/LTE facilities

provided by a service provider. Then the smart phone functions as a wireless router inside the vehicle and allowed the PC to connect to the Internet. All processed data and images too were also transferred to a central processing location through this arrangement. Figure 3.7 shows the wireless device used in the prototype vehicle.

Figure 3.7: Prototype vehicle wireless device.

The computer used was a Dell T1650 workstation which has Intel Xeon 3.7 GHz with four cores processors and 32 GB memory system was used and connected with the cameras and encoder by network switch. Figure 3.8 shows the computer used to control and store the automated data collection system in the prototype vehicle.

Figure 3.8: Computer controlling system.

3.2.5 Power Supply

The entire experimental setup was powered by a "Terminator Inverter" that converted 12V DC battery power to 240V AC (maximum of 3000 watt) household power. The Terminator Inverter series is an advanced line of AC power system with many safety features built in with low voltage alarm, auto shutoff as well as overload and short circuit protection. Figure 3.9 shows a photo of the used power supply, while Figure 3.10 shows the schematic diagram that illustrates the power system connections.

Figure 3.9: Terminator inventor power supply.

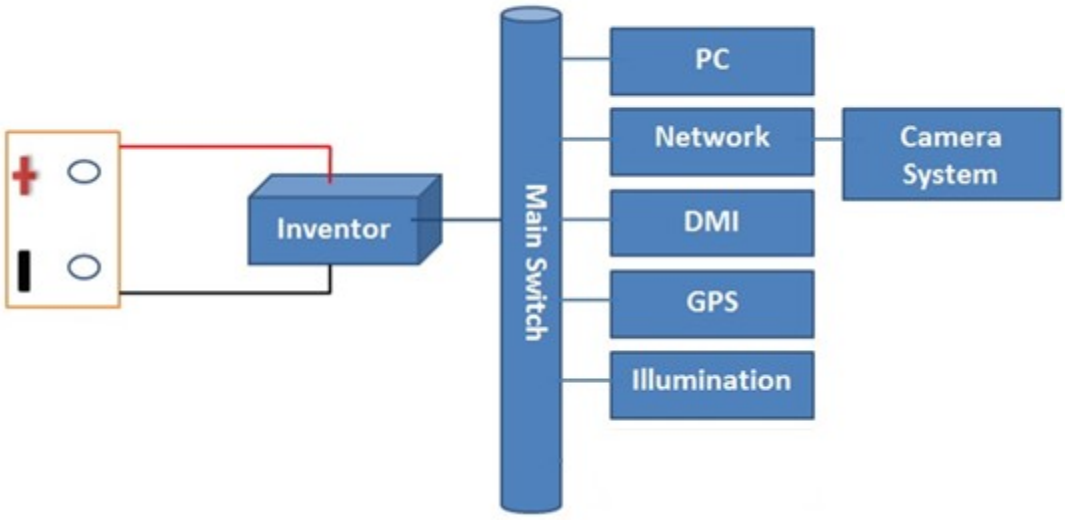


Figure 3.10: Power system connections.

3.2.6 Distance Measurement Instrument

The distance measurement instrument (DMI) is an accurate and reliable device which measures the distance and the speed. For this project, it also tracked GPS information on a second by second basis. The DMI provided pulse every tire revolution and this pulse train was processed to trigger both cameras to take the picture and at the same time pick the reading up from the GPS. Figure 3.11 is a picture of the physical DMI used in the set up.

Figure 3.11: Distance Measurement Instrument.

The speed sensor was installed at the wheel and it was connected to the dashboard console unit. Figure 3.12 shows the distance measurement instrument speed sensor installed in the prototype vehicle.

Figure 3.12: Distance measurement instrument speed sensor.

The details of the DMI (as reported in the instrument manual) are shown below in Figure 3.13.

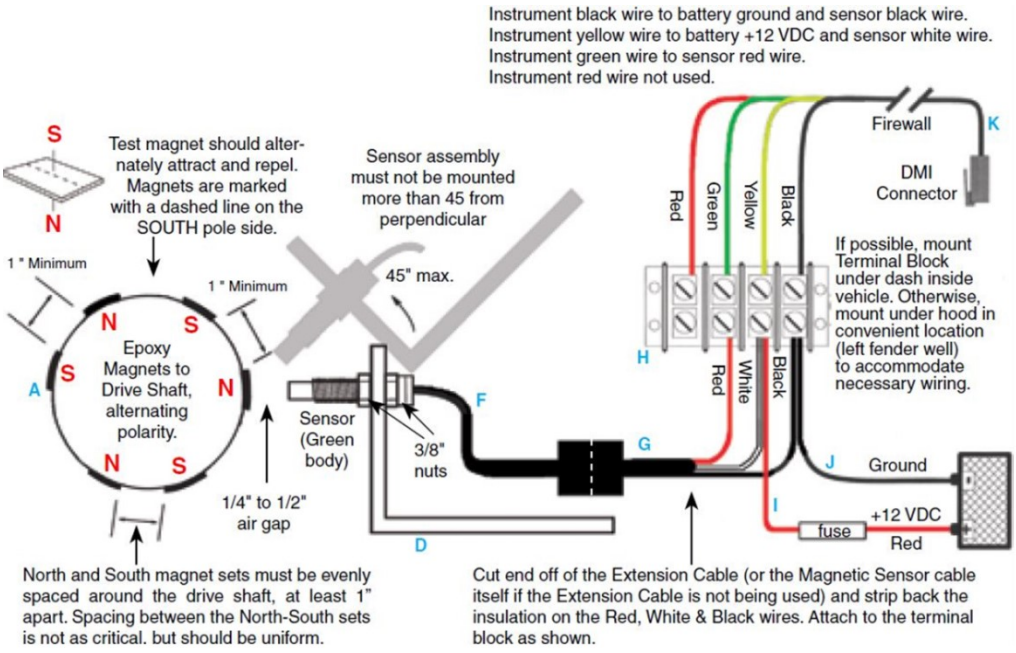


Figure 3.13: The DMI details.

3.2.7 Prototype Vehicle

The vehicle that was used for assembling the prototype system for collecting the data is shown in Figure 3.14. This vehicle is owned and operated by the city of Abu Dhabi Municipality. The prototype system has been designed and modified to be mounted on any vehicle type (Public service vehicles).

Figure 3.14: Prototype vehicle.

3.3 Image Processing

In this section, the detailed operation and processing system are presented.

3.3.1 Height Camera Calibration

Several trials were made in order to select an appropriate camera position that could cover a road section of interest without including any useless data. The height of the two

cameras was adjusted in order to avoid any interference with the captured photos and covered the entire width. Figure 3.15 shows a schematic diagram that illustrates the different attempted camera positions and the selected one. Vehicle processes includes several steps such as triggering both cameras, processing each image and compressing data. In the vehicle, the processed data was transmitted through the use of 3G or LTE wireless device and transferred to central office for final image recognition.

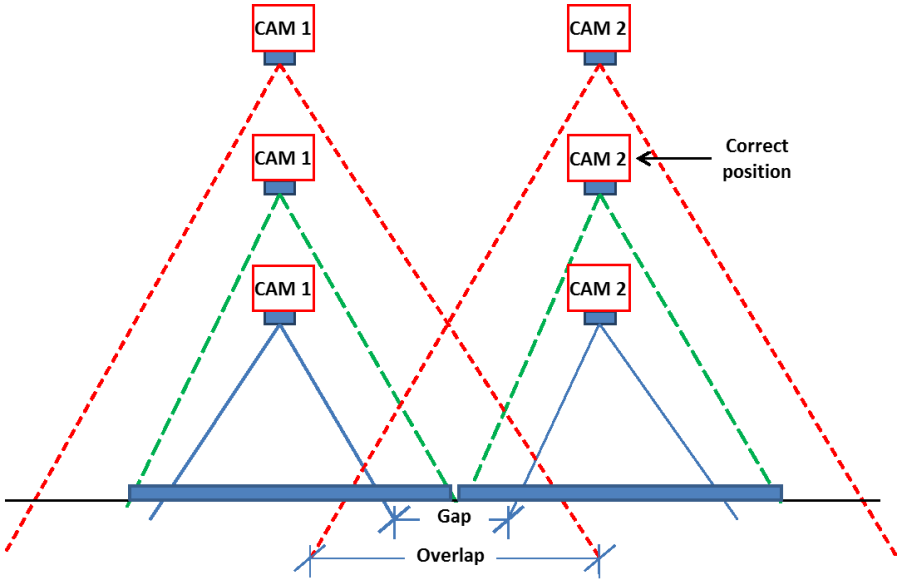


Figure 3.15: Schematic diagram of camera height adjustment.

Consider a road section (say a few hundred meters). The scanning of the road section follows the following general set-up: First, the road section is divided logically into segments. Each segment is 4m x 10m. 4 m is the lane width while the 10m is the segment length. The entire section will be scanned as a sequence of 10m segments. The 4m width is covered by two synchronized cameras as shown in Figure 3.16 below.

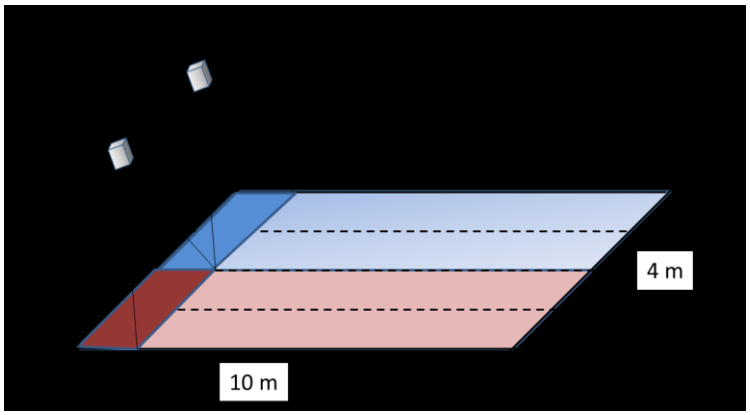


Figure 3.16: The width of two cameras covered.

The scanning is fully automated and it works as follows: the operator sends a command to the system to start scanning, storing and processing the road section. The DMI starts generating pulses at every wheel revolution. This train of pulses along with the car speed is processed to generate pulses that trigger the scanning. Each pulse causes the two cameras to scan and internally processes one segment. When the cameras fully process one segment it sends it to the PC through the data acquisition module. The GPS stamps the section file with location and the DMI stamps it with the recording time and the car speed. The (4 x 10) segment image is delivered to the PC as two synchronized (2 x 10) images, one from each camera and the segment image get stitched together by the signal processing software residing in the PC. Each of the (2 x 10) images is produced by the camera after several internal stitches. The camera can in fact be set up to take a burst of images and combine them into one integrated image taking into account the speed of the vehicle. The timing and synchronization are automatically adjusted to produce the image. The full processing steps are illustrated in the flow chart shown in Figure 3.17

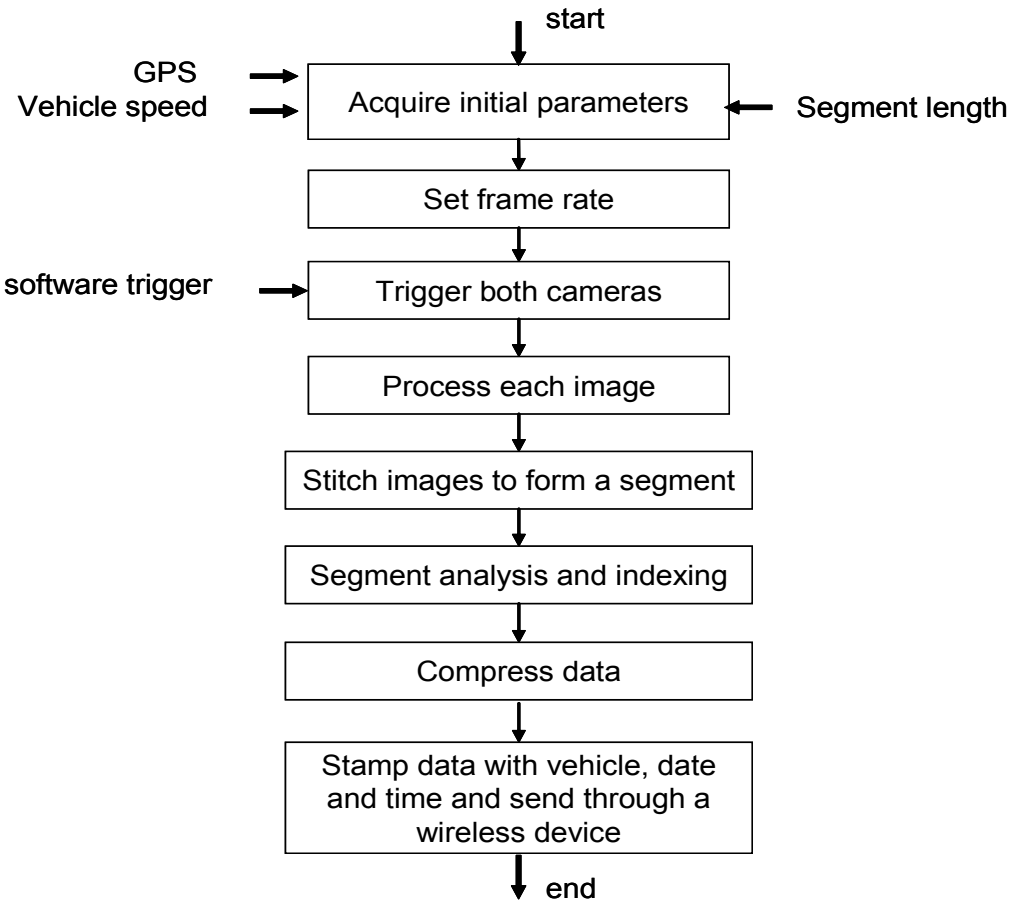


Figure 3.17: Flow diagram of the on-vehicle processing steps.

3.3.2 GPS Calibration

In order to calibrate the GPS, two different points in the same road section were selected. The coordinates of these points were checked by the coordinates that were taken by another GPS. Then the distance between these points was calculated. If the difference between the distances were more than the acceptable range of the GPS, the GPS will restore and re-setup again. Figure 3.18 illustrates the two coordinate points done by different GPS.

Figure 3.18: Two selected coordinate locations.

3.4 Image Analysis

Image analysis within the PC started by stitching the two images obtained from the two cameras to form one lane image picture of 4 x 10 m (a segment file).

1. Each section was given a unique reference number including the different traffic direction symbols such as the number of lanes (L_1 means lane number one) depending on the number of road lanes, the date of inspection (20140301), the route name (E11, emirates 11 route) and the section number (S_1, section no. 1).
2. Each section divided into10 segments. Each section is 100 meters as shown in Figure 3.19 below.

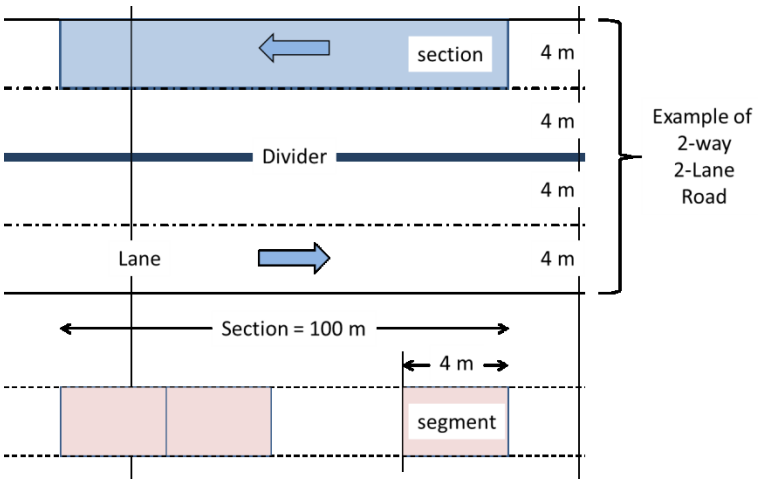


Figure 3.19: Section division to subsections.

3. Each segment was processed to produce the following information
    - Segment location (determined by the GPS), L
    - Measurement time of day (morning, afternoon or night), T
    - Vehicle speed during the measurement, V
    - Number of cracks is six categories:
        - Longitudinal crack
        - Alligator crack
        - Block crack
        - Edge crack
        - Transverse crack
        - Pothole
4. The results of image processing were listed as a table of values that can be transmitted over the wireless Internet to a central processing unit.
5. Several scenarios were used to sort the different pavement crack types and identified.

6. For each type a rank was established in order to determine the condition of the section.
7. Transfer the ranking condition to the GIS map to make the proper decision.

As the prototype vehicle which is equipped with the monitoring camera system travels the road, it sends back a sequence of signaled numbers indicating the initial quality pavement indicator of the road segments to the control station. These signals are usually noisy because they were taken on the fly and did not yet go through the typical rigorous analysis that leads to establishing a reliable quality pavement indicator. However, as numerous reports are received from the vehicle travelling the same roads over days and months the noise variance shrinks and the quality indicator converges into a more reliable value. Such statistical view of surveying the road conditions would lead to a city wide picture of how its roads are ranked. Clearly this process is not a replacement for the thorough and rigorous present analysis, but it is meant to provide an early warning system before that final step, and it builds a continuous record of real time conditions of the pavement similar to a medical check list of a human over his/her life span. The whole statistical approach may be viewed as a filtering process to indicate which segments of the road are in desperate need for repair and which segments are in good conditions. As the collected data is being analyzed, it acts as an early warning system which can provide valuable information which enables managers and engineers to act in time to make proper and timely decisions.

3.5 Software Development

There are several ways to collect and analyze the image process. The use of computer software program technique is an economic and accurate way to collect and analyze the image processing. The software for this project was developed by Carleton University using C++ language [65]. The computer software was developed and used to analyze the image of pavement and identify its level of distresses. The level of distress depends on a number of parameters including: distress types, severities, and densities. In addition, the software integrates the results of the image process into the distress map of a city. Figure 3.20 shows a flow chart describing the steps of the developed automated software. The code script of developed automated system software is illustrated in Appendix A.

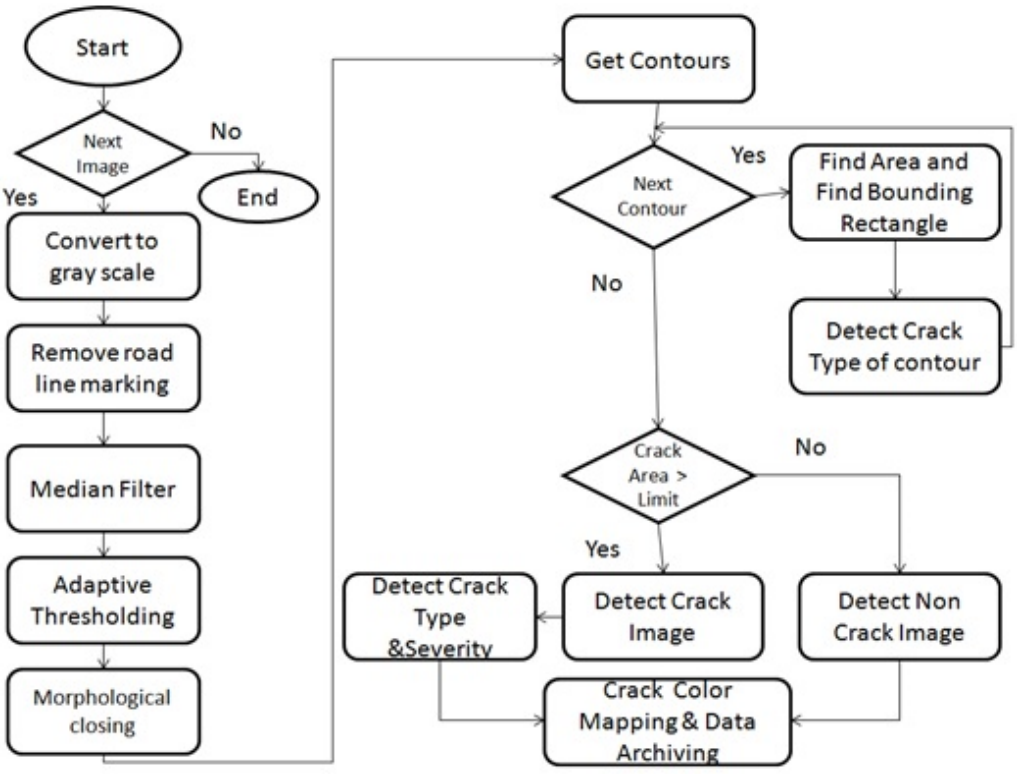


Figure 3.20: Flow Chart of the automated developed software steps.

3.5.1 Conversion Photos to Gray Scale

The recorded pictures are colour pictures with 24 bits per pixel (8 bits for each of the three basic colours: Red, Green and Blue). Since the main interest part is to detect cracks and defects on the pavement, it is a common practice to convert colour pictures into gray-scale pictures (8 bits per pixel). The 8-bit pixel coding allows for 256 of gray shades. These shade levels are called the "Luminance" value which described as brightness or intensity which can be measured on a scale from black (zero intensity) to white (full intensity). Figure 3.21 to Figure 3.24 show several examples of input images [66].

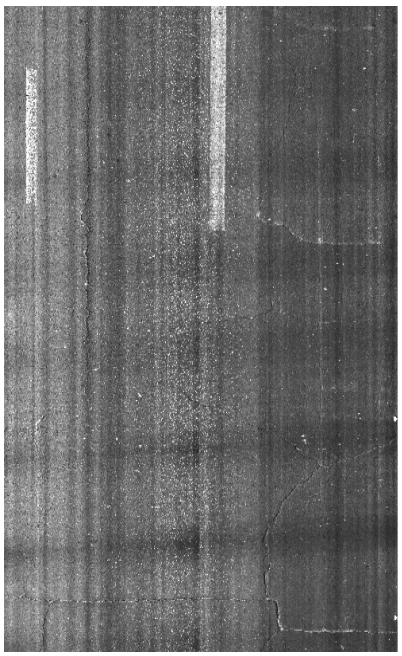


Figure 3.21: Alligator cracking input image.

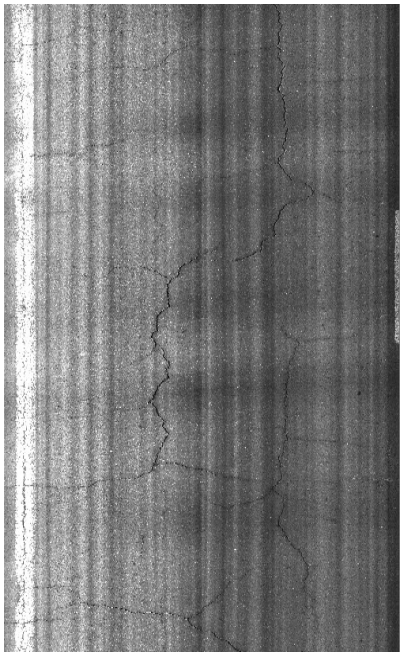


Figure 3.22: Block cracking input image.

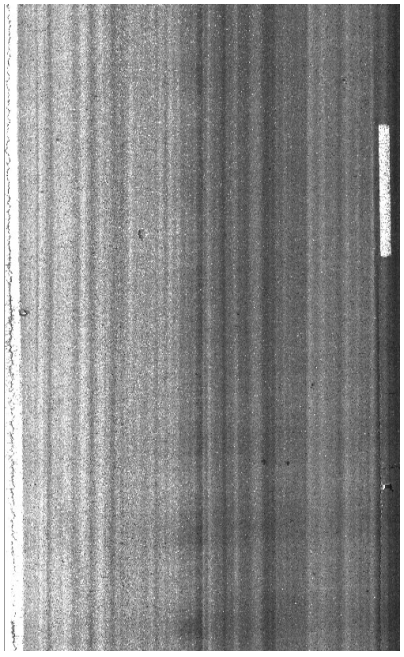


Figure 3.23: Edge cracking input image.

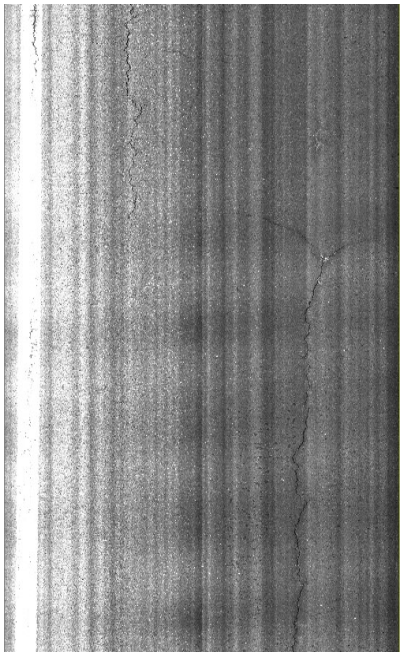


Figure 3.24: Longitudinal cracking input image.

3.5.2 Removal of Road Line Marking

The road`s white lane marking is removed by changing the high pixel values (white color) to low pixel values (dark color) using the developed software. In real-time pavement images, 170-255 ranges represent the white colour and the dark colour is represented from 0 to 100 ranges. For example, after attempting several trails to reduce the noise made by road`s white lane marking, the values were firstly were represented from 230 to 255 ranges, which gave the result shown in Figure 3.25. The second attempt used values ranging 200 to 255, which gave the result shown in Figure 3.26. Similarly, Figure 3.27 illustrates the result of applying the range of 190 to 255. Finally, the value used ranging from170 to 255 is shown in Figure 3.28 and it gave the best result in removing the white line. This step is very important for following and identifying the actual distress captured in the photos [67].

a) b)

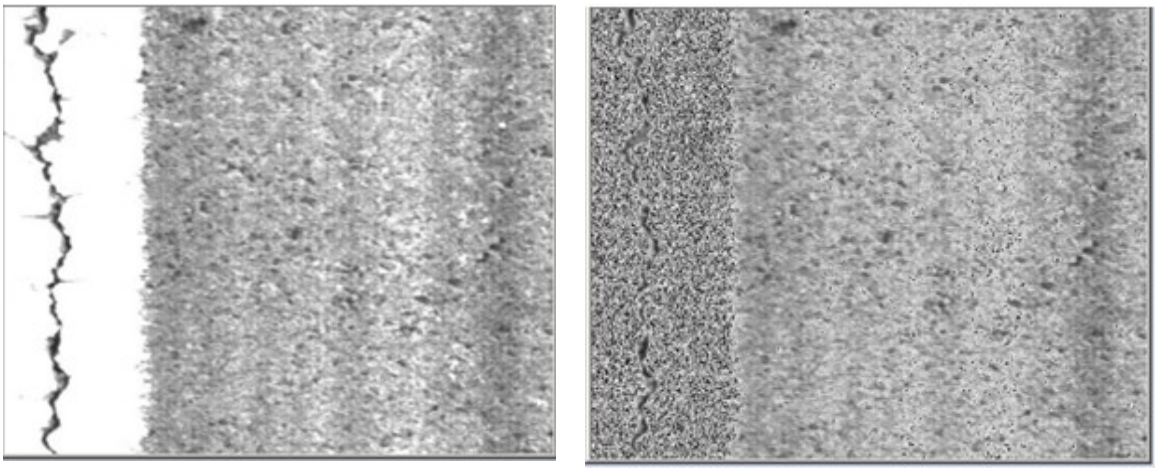


Figure 3.25: Remove road line marking; first value (a) input, (b) output.

a) b)

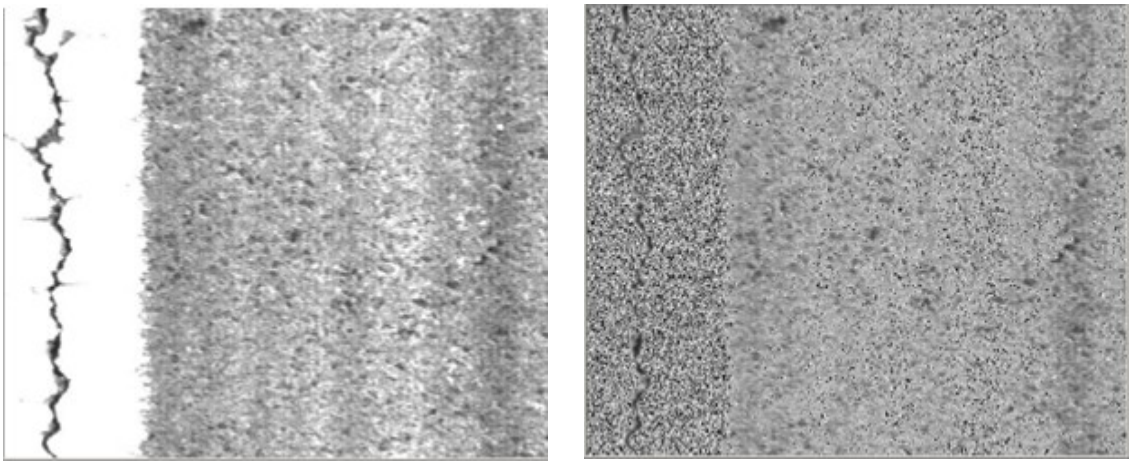


Figure 3.26: Remove road line marking; second value (a) input, (b) output.

a) b)

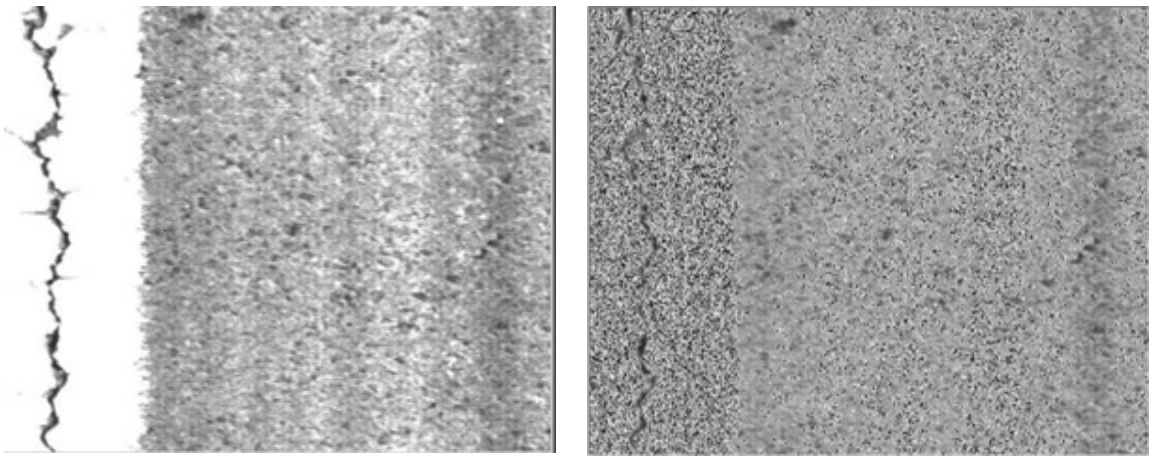


Figure 3.27: Remove road line marking; third value (a) input, (b) output.

a) b)

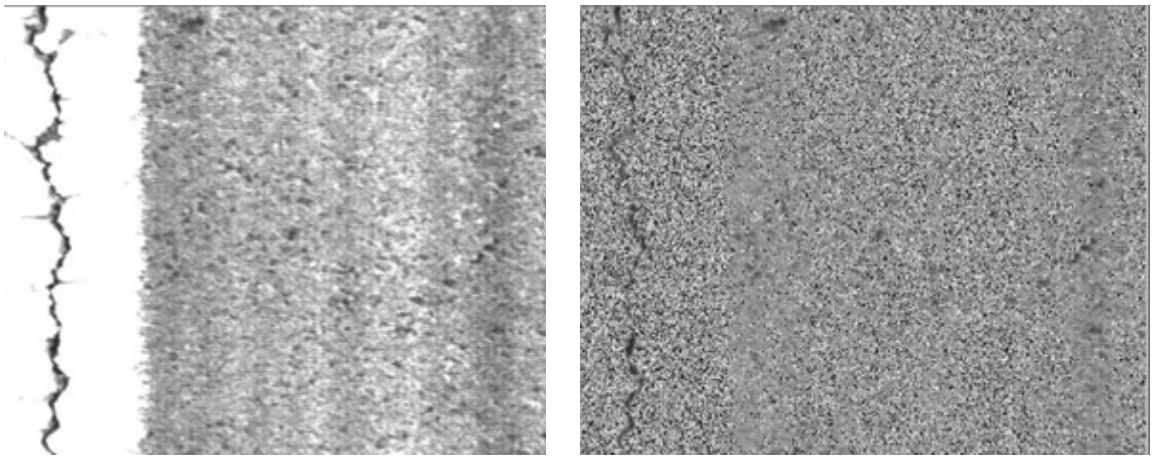


Figure 3.28: Remove road line marking; last value (a) input, (b) output.

3.5.3 Median Filter

The median filter is a nonlinear filter operation that replaces a pixel value by a weighted sum of the neighborhood pixels as opposed to smooth filters. The median filter generates and operates on a local neighborhood with replacing the centre pixel by the median value of the neighboring pixels. Noise removal is one of the median filter operations. An example of this is illustrated in Figure 3.29. The median was found by first sorting neighboring pixels in increasing order of the gray scale value and then by finding the centre of the sorted values using the formula: (n+1)/2, where n is the number of neighboring pixels. Figure 3.30 to Figure 3.33 present the median filter performed on Alligator, Block, Edge and Longitudinal distresses [68].

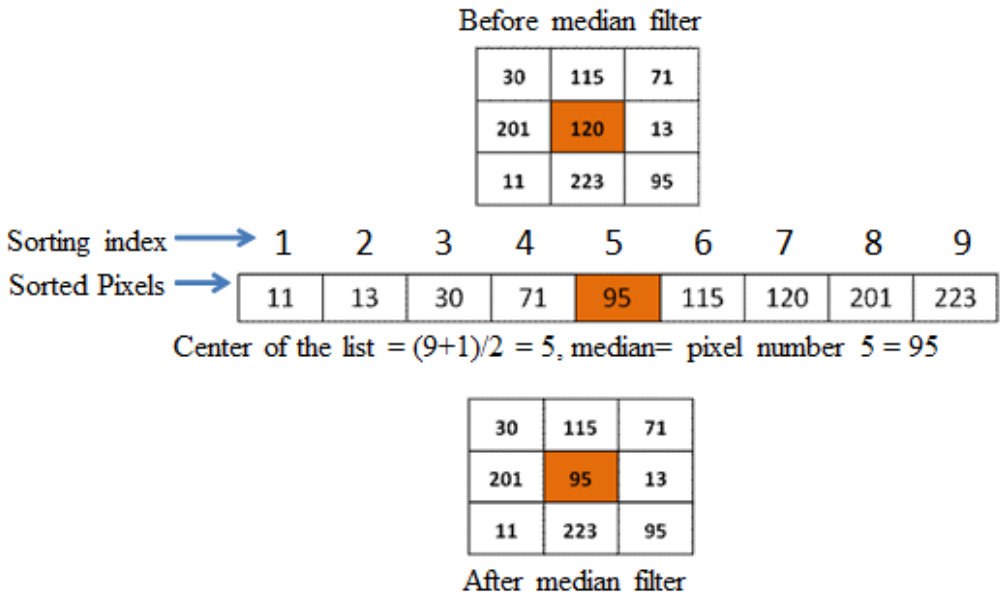


Figure 3.29: Example of the median filter (reproduced from [68]).

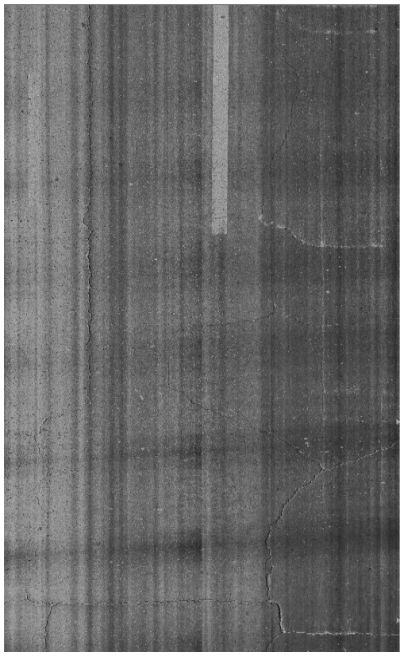


Figure 3.30: Alligator cracking median filter.

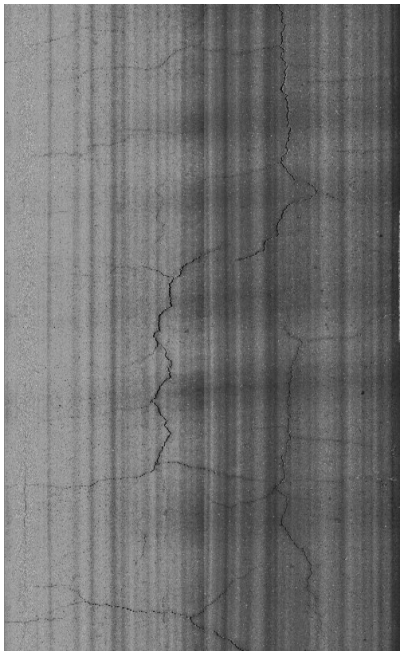


Figure 3.31: Block cracking median filter.

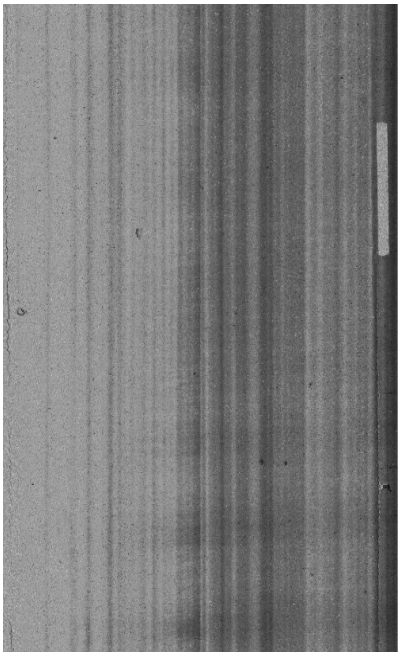


Figure 3.32: Edge cracking median filter.

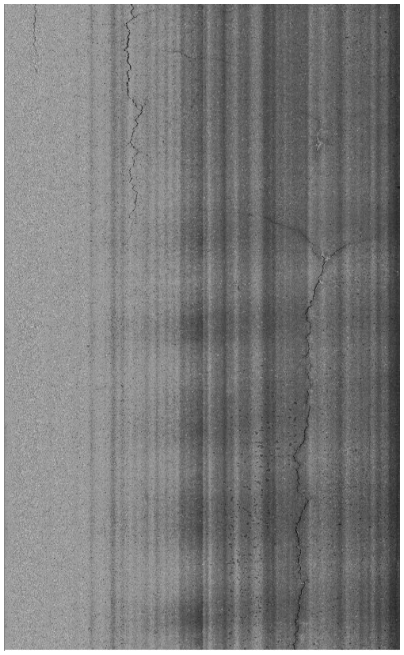


Figure 3.33: Longitudinal cracking median filter.

3.5.4 Adaptive Thresholding

In addition to the previous steps in the developed software, the goal of separate crack information from the rest of the image, and also taking into account that crack pixels represented by darker intensities than their surroundings, the use of adaptive thresholding is presented. Thresholding is a non-linear operation which transforms a gray scale image to a binary image where the two levels are assigned to pixels that are below or above the specified threshold value. An image of f(x,y) composed of a light object and background pixels that has intensity levels grouped into two dominant modes. The first mode represents the background and the other represents the object. By separating the two modes the gray level threshold (T) was chosen and every pixel was turned to black and white according to whether each pixel gray level is greater than or equal to the threshold or less than the threshold. Figure 3.34 shows the thresholding image histogram. And figures 3.35 to 3.38 present the example of adaptive thresholding done by the developed software [69].

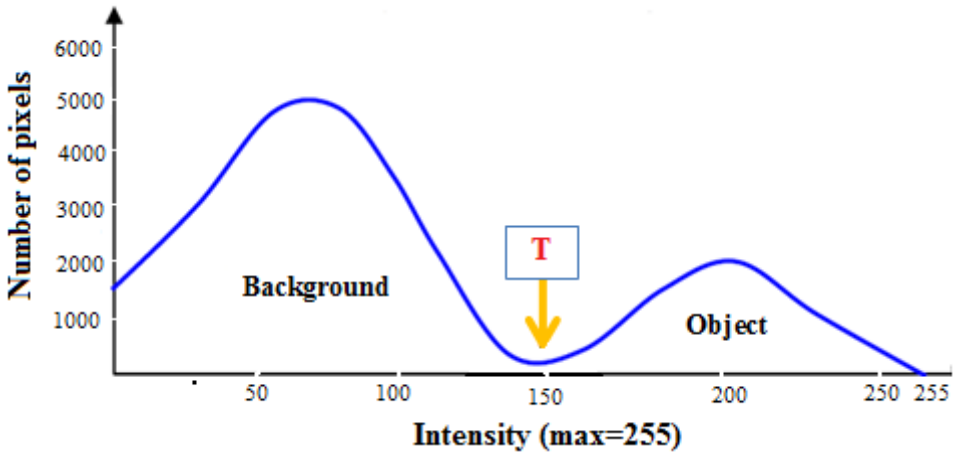


Figure 3.34: Thresholding image histogram (reproduced from [69]).

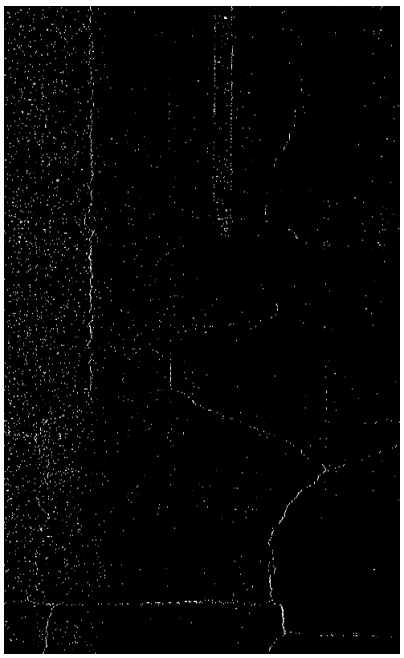


Figure 3.35: Example of alligator cracking adaptive thresholding.

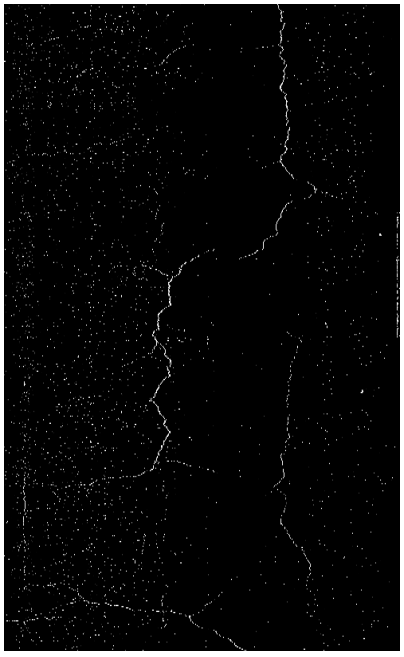


Figure 3.36: Example of block cracking adaptive thresholding.

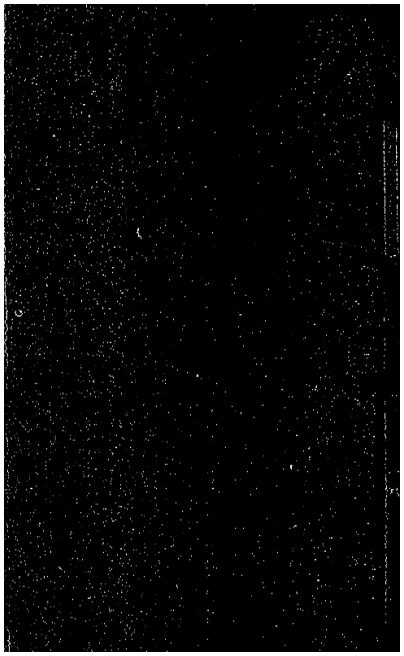


Figure 3.37: Example of edge cracking adaptive thresholding.

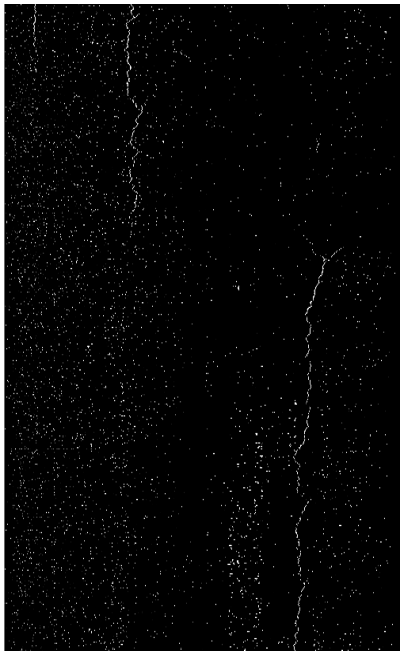


Figure 3.38: Example of longitudinal cracking adaptive thresholding.

3.5.5 Morphological Closing

Morphological image processing is a broad set of image processing operations or a range of image processing techniques that deal and process images based on shape. Morphological operations are assigned as a tool of extracting components that are useful in presenting and the description of region shapes (like boundaries and skeletons) giving the same output image size as the original input image size. The most used morphological operations are dilation and erosion. Dilation is defined as the fundamental process of adding pixels to boundaries of objects in an image while the erosion removes pixels on object boundaries. The number of added and removed pixels depends on the size and shape of the structure element used to process the image. The morphological opening is used to remove small objects while keeping the shape and size of the larger object in an image. The morphological closing is obtained by the dilation of an image first followed by erosion to remove small holes in dark regions or filling the gaps between the cracks [70]. Figures 3.39 to 3.42 illustrate an example of a morphological closing for pavement images.

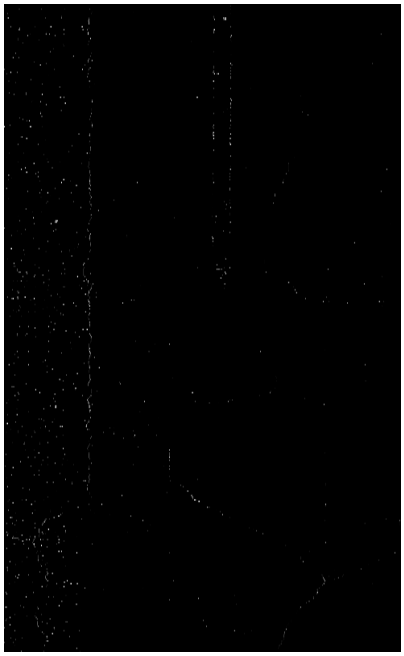


Figure 3.39: Alligator cracking morphology closing sample.

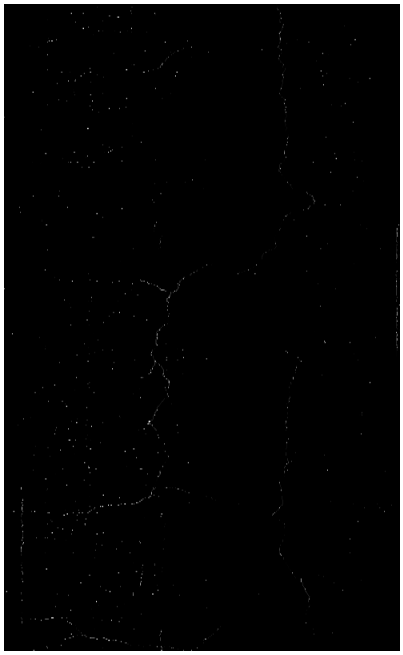


Figure 3.40: Block cracking morphology closing sample.

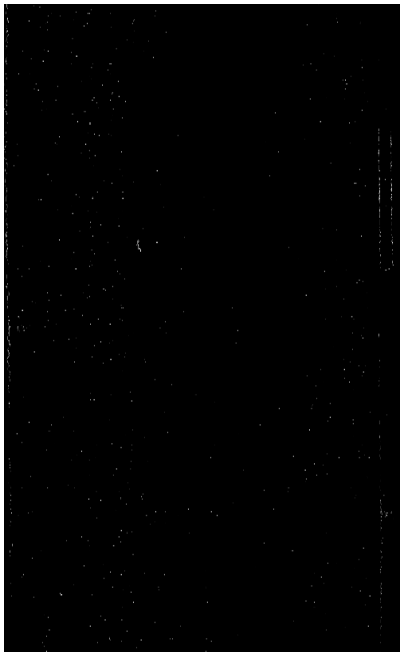


Figure 3.41: Edge cracking morphology closing sample.

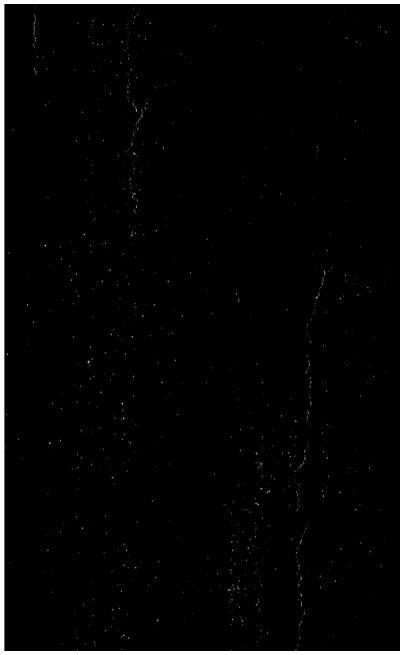


Figure 3.42: Longitudinal cracking morphology closing sample.

3.5.6 Detection of Crack by Contours

The final steps are to detect the crack types, classify their severity levels, and to find the density percentage in the pavement image by using the contours. Applying the contours to the pavement images is done by the following steps. Firstly, the direction, orientation, and location of the distresses will be detected. Secondly, each defect in the pavement images are then surrounded by bounding rectangular box to define the distress type of contour, and the processing will continue until there are no more distress contours in the pavement image. Thirdly, if the crack area in the pavement image is lower than limits according to the SHRP-LTTP manual the developed software will define the pavement image with no distresses and will continue with stamping and archiving it in the map. Finally, if the crack area is upper than the limits (according to the SHRP-LTTP manual) the developed software will analyze and detect the crack image, define the crack type, classify the crack severity levels, and find the density percentage of the area with stamping and archiving it in the map [71]. Figures 3.43 to 3.46 show the example of defining the crack type, severity levels and density. Figure 3.47 shows the final results stamped into the road map (the blue line represents there is no crack in the section and red line represents there is crack in the section).

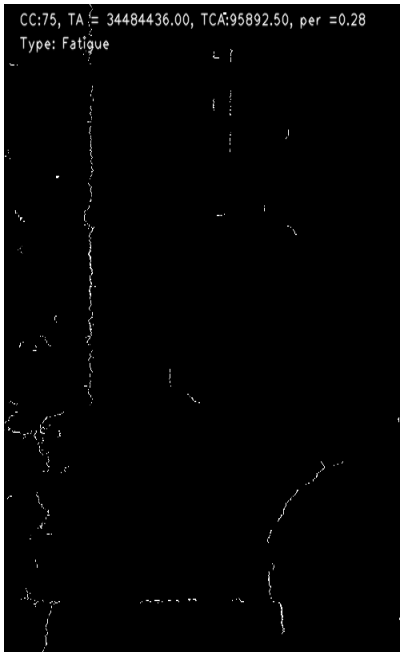


Figure 3.43: Detection of alligator crack type, severity level and density by contours.

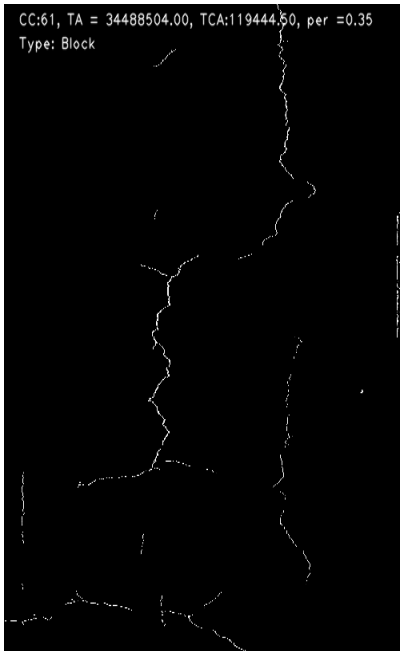


Figure 3.44: Detection of block crack type, severity level and density by contours.

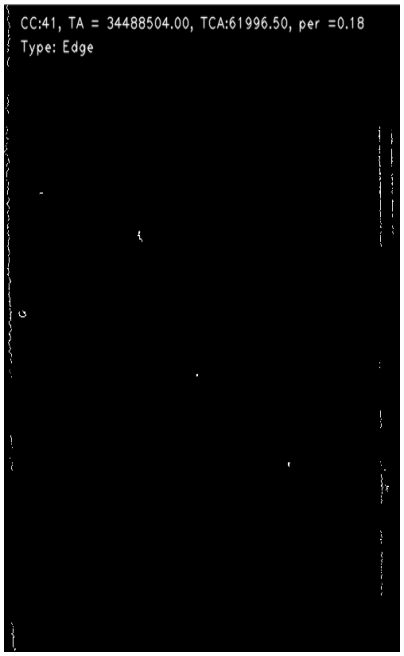


Figure 3.45: Detection of edge crack type, severity level and density by contours.

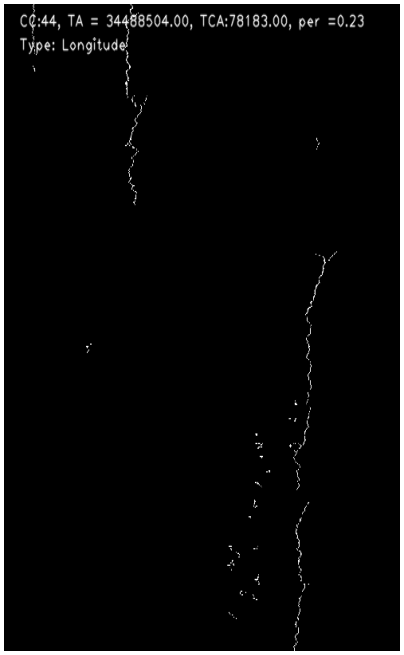


Figure 3.46: Detection of longitudinal crack type, severity level and density by contours.

Figure 3.47: The Pavement detection cracking map.

If there is a serious rapid deterioration to unforeseen damage, rebuilding, maintenance and/or rehabilitation of a road section that can be rapidly change the pavement indicator, and then the pavement indicator has to be changed by hand.

3.6 Summary

This chapter described the experimental setup used for data collection and the basic steps used to process the collected images and extracting relevant features used to characterize the quality of the pavement surface.

The approach proposed in this thesis is to equip several public vehicles with an inexpensive camera system that can capture and process images of the pavement in a fully automated manner. In addition, the cars equipped with these camera systems are not out to collect data as a primary purpose, rather, they are going about doing their functions such as transporting people or responding to emergencies and so on, and the data collection is a secondary function performed in an automated process.

In this chapter, all automated system components are listed with description of their individual functions. Additionally, data collection procedures were described, where roads are divided into sections (100 meters each) and each section is further divided into

10 segments. Each segment is 4m x 10m area. The collected images are processed onboard to produce the number and type of cracks observed in each segment. The record of each segment consists of the analysis results and is stamped with the location and time and then is transmitted via wireless links to a central control centre.

CHAPTER 4: PAVEMENT CRACK ANALYSIS

This chapter is organized into three main sections. The first section includes the data description, the second section describes two different distress measurement methods, and the third section includes a descriptive and comparative data statistical analysis.

4.1 Data Collection

4.1.1 Site Description

In order to collect different types of distress, eight sites were selected in Abu-Dhabi. A comparison between the eight sites was made and only one road of these sites was selected based on the quantity and clarity of the distress. The other sites were considered to contain less distress compared to the selected site. The selected road section was part of Sheikh Maktoum Bin Rashid Highway E11 as shown in Figure 4.1. Sheikh Maktoum Bin Rashid Highway E11 is a divided four-lanes and two-way highway. The thickness of the pavement of this road is 60 mm of asphalt wearing course; 70 mm of asphalt base course; 300 mm of granular sub-base course; and 300 mm of subgrade. Various pavement cracks can be seen as transverse cracks distress, longitudinal cracks distress, alligator cracks distress, pothole distress, edge cracks distress, and block crack distress. A description of these pavement cracks is shown in Table 4.1. Photos and schematic diagrams of these pavement cracks are shown in Figure 4.2 to Figure 4.7.

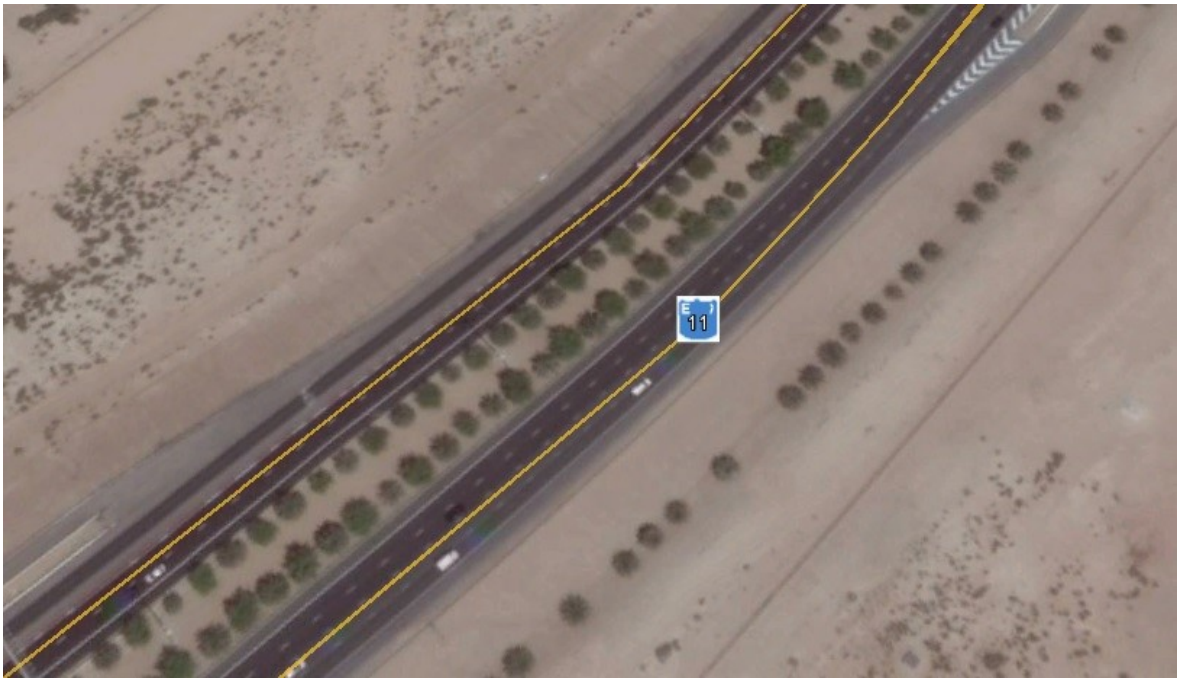


Figure 4.1: The selected site location E11 in Abu-Dhabi city.

Table 4.1: Definition of pavement cracks.

Pavement cracks	Description
Transverse cracking	Cracks are mostly vertical to the pavement centerline.
Longitudinal cracking	Cracks that are mainly located within the lane and are parallel to the pavement centerline
Alligator cracking	Cracks that are a series of interconnected cracks that appear in areas subjected to repeated traffic loads especially with chicken wire or alligator pattern
Potholes	Pothole is a type of pavement distress failure similar to bowl-shaped holes due to water present in the underlying soil structure and traffic passing though that damaged area.
Edge cracking	Edge cracking is crescent-shaped cracks or completely continuous cracks which intersect the pavement edge located within 0.6 meter of the pavement edge adjacent to the shoulder.
Block cracking	Block cracking is a distress that results from breakdown of the pavement into rectangular pieces approximately 0.1 m^2 to 10 m^2.

a) b)

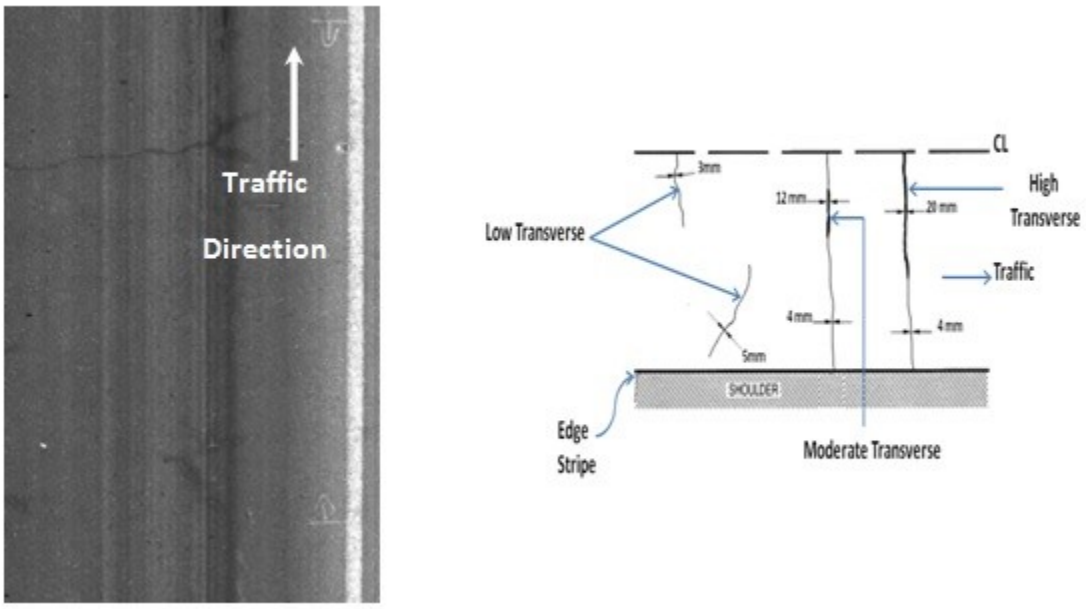


Figure 4.2: Transverse cracking; (a) Transverse cracking photo, (b) Transverse cracking schematic diagram.

a) b)

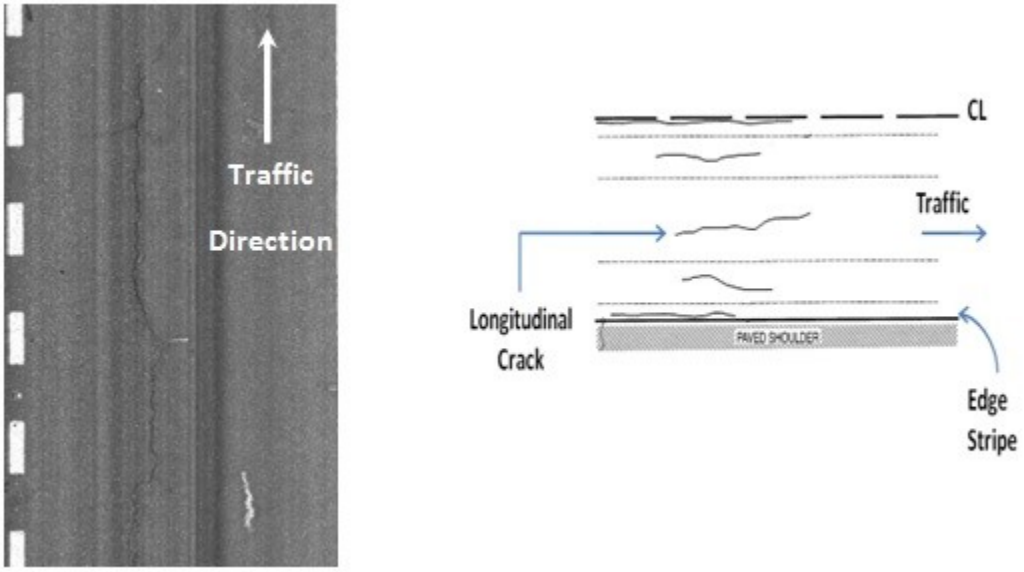


Figure 4.3: Longitudinal cracking; (a) Longitudinal cracking photo, (b) Longitudinal cracking schematic diagram.

a) b)

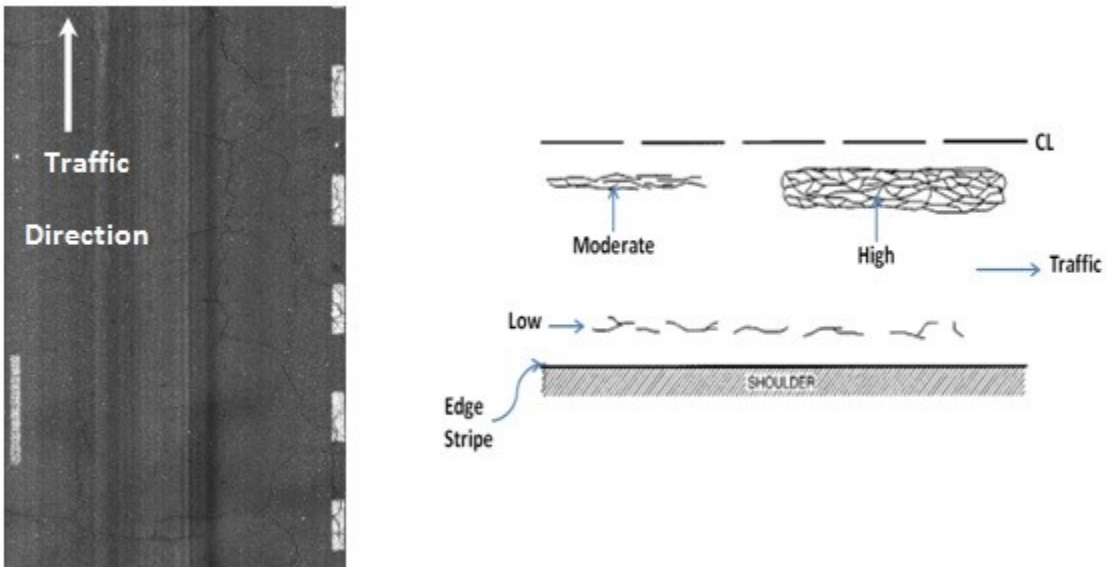


Figure 4.4: Alligator cracking; (a) Alligator cracking photo, (b) Alligator cracking schematic diagram.

a) b)

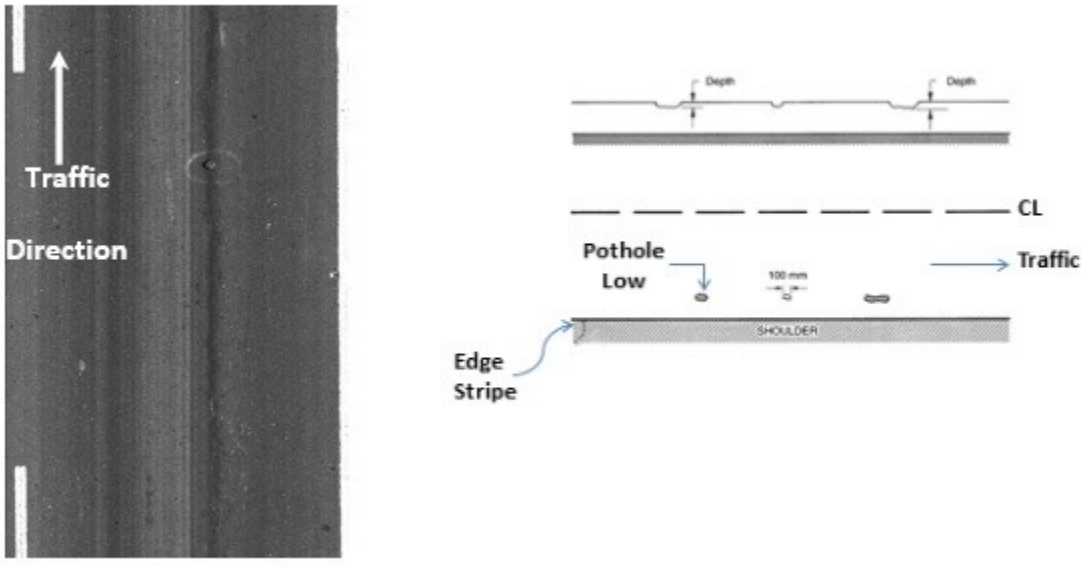
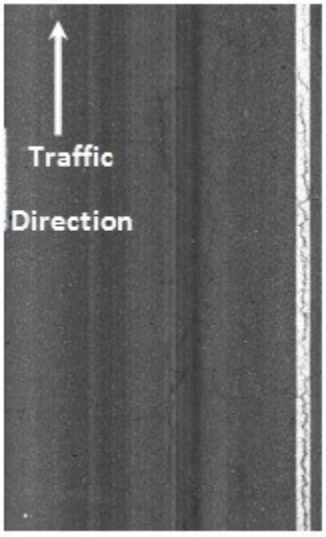


Figure 4.5: Pothole distress; (a) Pothole distress photo, (b) Pothole distress schematic diagram.

a) b)



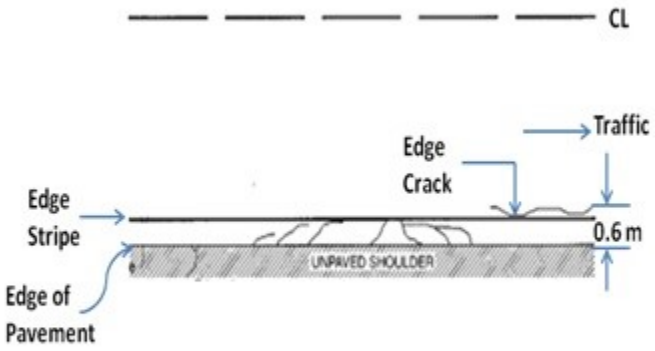
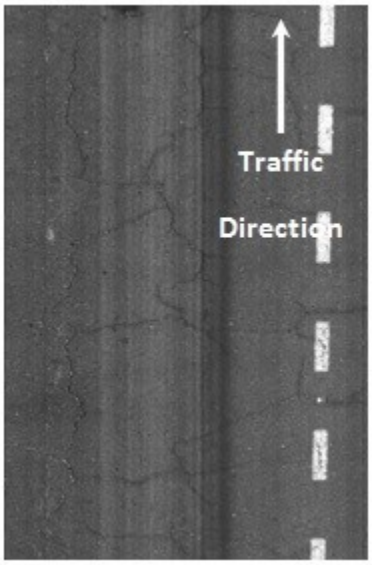


Figure 4.6: Edge cracking; (a) Edge cracking photo, (b) Edge cracking schematic diagram.

a) b)



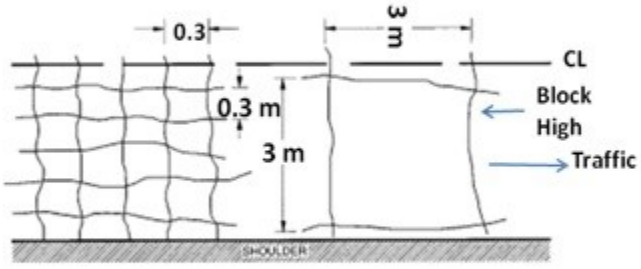


Figure 4.7: Block cracking; (a) Block cracking photo, (b) Block cracking schematic diagram.

4.1.2 Photo Data Collection Equipment

In order to capture pavement crack along a section of the highway, photo data were collected using two cameras as described in Chapter Three.

4.1.3 Photo Data Collection

Photo data was collected from sections of Sheikh Maktoum Bin Rashid Highway E11 in order to capture pavement cracks during three different time periods in one day. Each time period was two hours long. The first time period was between 8:00 AM and 10:00 AM. The second time period was between 1:00 PM and 3:00 PM. While the third time period was between 8:00 PM and 10:00 PM. Data collection was performed in April for 7 days in 2014. Table 4.2 shows a summary of the data collection duration, number of sections covered and the number of photos captured.

Table 4.2: Total number of road sections and photos captured.

Time Period	Sections	Photos
8:00 AM – 10:00 AM	174	1740
1:00 PM – 3:00 PM	166	1660
8:00 PM – 10:00 PM	167	1670
Total	507	5070

4.1.4 Photo Data Organization

The observation methodology consists of a review of the photos and an identification of pavement cracks. The records of pavement cracks are then assigned various attributes, depending on the type and severity of each distress. The observation matrix is composed of a set of rows, each one representing a section that includes ten photos, and a set of columns, each one representing a particular observation (numeric value) or an attribute (0 or 1) based on whether the occurrence represented by this attribute was observed.

4.2 Distress Evaluation Methods

In addition to the automated measurement method investigated in this study, a semi-automated measurement method was also used in order to examine the accuracy and validity of the automated measurement method. Therefore, two evaluation methods were used in this study, and can be classified as: [i] Semi-automated Measurement method (SAM), and [ii] Automated Measurement (AM) method. The evaluation of the two measurement methods is described in the following subsections.

4.2.1 Evaluation of the Semi-Automated Measurement Method (SAM)

The semi-automated measurement method was used to identify the types of distress of the road pavement. Photos were observed and analyzed to identify possible types of distress. This method is used as a benchmark to evaluate the accuracy of the automated method. Each photo was independently reviewed by two different reviewers in different sessions. The first observer was the thesis author and the second observer was an experienced engineer. All extracted data were categorized and saved in an Excel file as

described in the previous section. The included data are; (1) distress types, (2) distress severity and (3) the percentage of the distress area. Six distress types were observed in all sections from the area under study which consisted of: transverse cracking, longitudinal cracking, alligator cracking, pothole, edge cracking and block cracking types. Distress severity was classified into three levels: low, medium and high severity. The percentage of the distress area was defined as the percentage area of each distress type out of the road section area. The dimension of the road section area was 100 meters by 4 meters, and the total number of the sections surveyed was 507 sections.

A total of 5070 photos were captured and 5137 pavement crack cases were identified manually by two reviewers and included the six pavement crack types. Table 4.3 shows the results of the reviewed pavement cracks by the two reviewers.

Table 4.3: Total number of distresses detection by two reviewers.

Pavement crack	1st reviewer*	2nd reviewer
Transverse	186	167
Longitudinal	993	942
Alligator	683	635
Pothole	51	45
Edge	155	137
Block	593	550
Total	2661	2476

* Thesis author

The statistical analysis of data is based on what is known as the Kappa Coefficient. The Kappa Coefficient (k) is used to evaluate the level of agreement between the results of the two reviewers. Kappa coefficient is defined as "a statistical measure of the inter-agreement or inter-annotator agreement", and is calculated as follows:

$$k = \frac{p_o - p_c}{1 - p_c} \quad \text{..... (4.1)}$$

Where;

k: kappa coefficient.

p_o: the observed proportion of agreement.

p_c: the proportion of agreement expected by chance.

Depending on the k value between two sets of observations, different levels of agreement between any two sets of observation can be obtained [72]. Table 4.4 shows different levels of agreement according to the range of k value.

Table 4.4: Kappa agreement strength [72].

Kappa Statistic K	**Agreement strength**
-1.00	Complete disagreement
-0.99 < 0.00	Disagreement
0.00	Chance agreement
0.01 to 0.20	Slight agreement
0.21 to 0.40	Fair agreement
0.41 to 0.60	Moderate agreement
0.61 to 0.80	Substantial agreement
0.81 to 0.99	Almost perfect agreement
1.00	Perfect agreement

The agreement between the two reviewers is described in the following subsections.

4.2.1.A Agreement by Pavement crack Type

The strategy that was used to identify any pavement crack from a photo is noted as either 1 "crack" or 0 "no crack" [73]. Table 4.5 shows a sample of the pavement crack types that were identified by the first reviewer. Table 4.6 shows the measure of agreement between the two reviewers.

Table 4.5: Sample of extract pavement crack types data.

Section	Distress						
	Speed (km/h)	TRANS	LONG	ALLIGATOR	POTHOLE	EDGE	BLOCK
Section#1	20	0	0	0	1	0	0
Section#2	20	1	1	0	0	0	0
Section#3	20	0	0	0	2	0	0
Section#4	20	0	0	0	0	0	0
Section#5	20	0	1	0	1	1	0
Section#6	20	0	0	0	1	0	0
Section#7	20	0	1	0	0	0	0
Section#8	20	0	0	1	0	0	0
Section#9	20	0	0	0	0	0	0
Section#10	20	0	0	0	0	0	0
Section#11	20	1	1	1	0	0	1
Section#12	20	0	1	2	1	2	0
Section#13	20	0	1	0	1	0	0
Section#14	20	0	3	0	1	0	2
Section#15	20	0	7	1	1	0	1
Section#16	20	0	3	0	1	0	0
Section#17	20	0	4	4	0	1	1
Section#18	20	0	0	8	0	0	2
Section#19	20	1	2	3	0	0	4
Section#20	20	0	0	2	0	0	8

Table 4.6: Kappa agreement strength of pavement crack types between the two reviewers.

Pavement crack Type	Kappa "K" Value	Agreement
Transverse	0.93	Almost perfect agreement
Longitudinal	0.94	Almost perfect agreement
Alligator	0.94	Almost perfect agreement
Pothole	0.92	Almost perfect agreement
Edge	0.94	Almost perfect agreement
Block	0.94	Almost perfect agreement
Overall pavement crack	0.98	Almost perfect agreement

As shown in Table 4.6, the Kappa results showed that the agreement between the two reviewers was excellent and close to being a perfect agreement for all pavement crack types.

4.2.1.B Agreement by Pavement crack Severity

The severity of each pavement crack was estimated from the measurement results. There were three different levels of severity that were identified: [i] low severity, [ii] moderate severity, and [iii] high severity. For example, the following steps described the estimated severity level of the longitudinal cracking distress.

Step_1: Longitudinal cracking low severity.

When the mean width of the crack is less or equal to 6 mm, the severity is defined as low severity as shown in Figure 4.8.

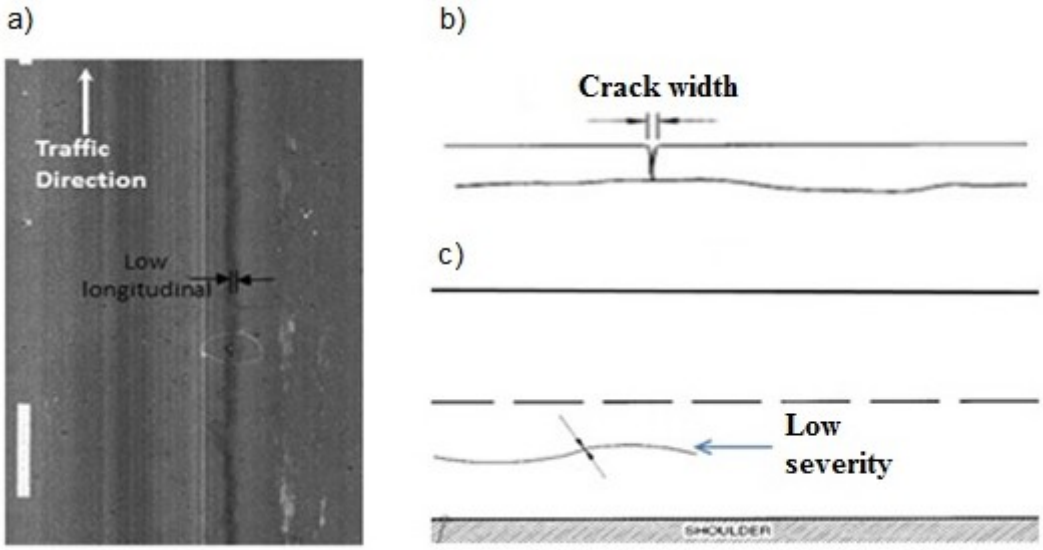


Figure 4.8: Low severity of the longitudinal pavement cracking; (a) Longitudinal cracking road image, (b) Schematic diagram of the side view of the cracking wide, (c) Longitudinal cracking the top view.

Step_2: Longitudinal cracking moderate severity

When the mean width of the crack is greater than 6 and less or equal than 19 mm, the severity is defined as moderate severity as shown in Figure 4.9.

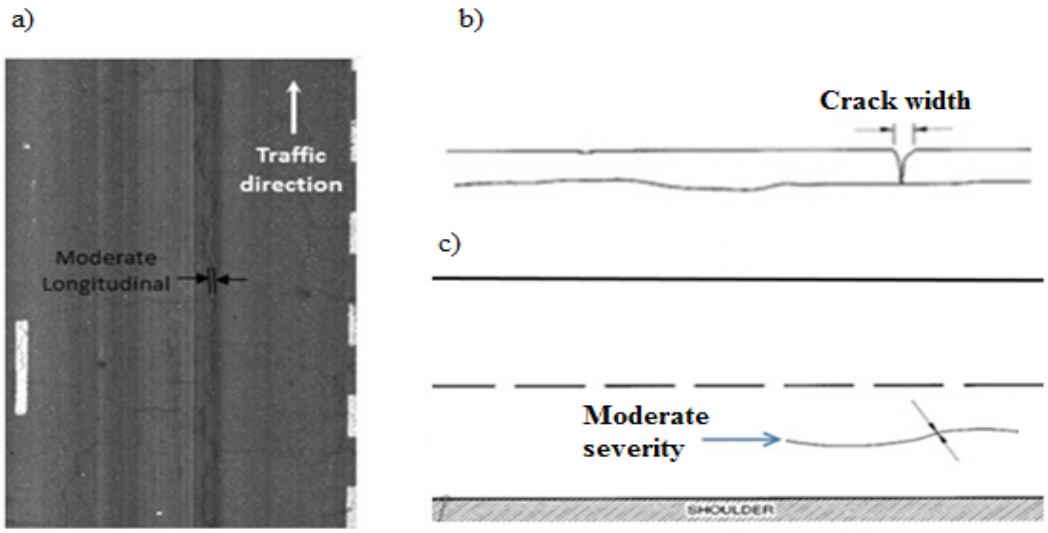


Figure 4.9: Moderate severity of the longitudinal pavement cracking; (a) Longitudinal cracking road image, (b) Schematic diagram of the side view of the cracking wide, (c) Longitudinal cracking the top view.

Step_3: Longitudinal cracking high severity

When the mean width of the crack is greater than 19 mm, the severity is considered high severity as shown in Figure 4.10.

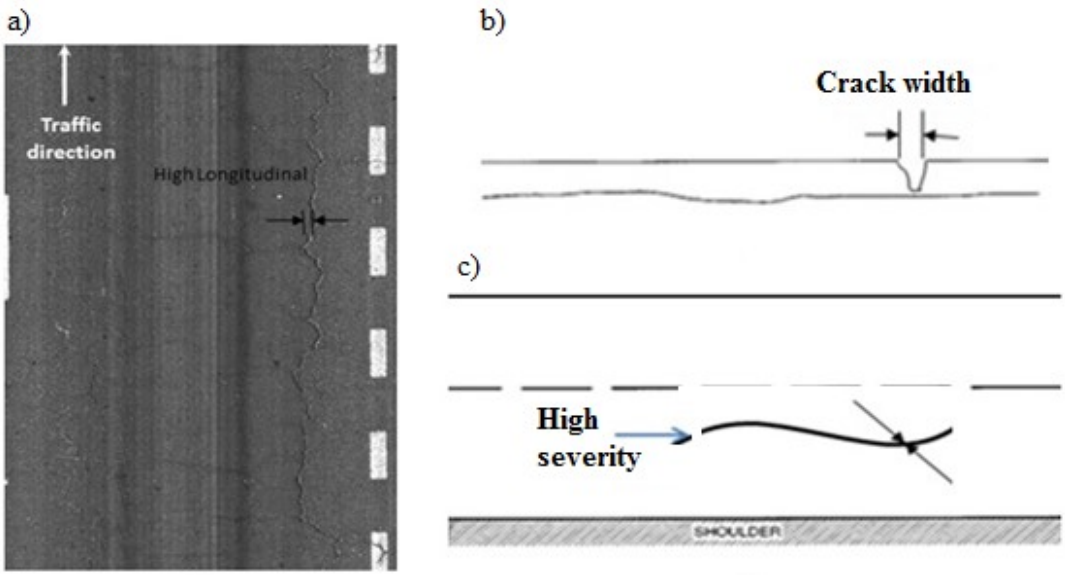


Figure 4.10: High severity of the longitudinal pavement cracking; (a) Longitudinal cracking road image, (b) Schematic diagram of the side view of the cracking wide, (c) Longitudinal cracking the top view.

The same strategy that was used to identify the pavement crack types from the captured photos is used to classify the severity of each pavement crack type. Table 4.7 shows a sample of the estimated severity levels of each type that was estimated by the first reviewer.

Table 4.7: Sample of estimated severity levels of pavement crack types.

Section #	Segment #	Transverse			Longitudinal			Alligator			Pothole			Edge			Block		
		L*	M**	H***	L*	M**	H***	L*	M**	H***	L*	M**	H***	L*	M**	H***	L*	M**	H***
66	S66-131	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	S66-132	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	S66-133	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	S66-134	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	S66-135	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	S66-136	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	S66-137	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	S66-138	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	S66-139	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	S66-140	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

*L: Low
**M: Moderate
***H: High

Table 4.8 shows the agreement results of pavement severity distress types as obtained by comparing the results of the two reviewers.

Table 4.8: Kappa agreement strength crack severity levels between two reviewers.

Pavement crack Type	**Kappa "K" Value**		
	Low	**Moderate**	**High**
Transverse	0.95 Almost perfect agreement	0.85 Almost perfect agreement	0.79 Substantial agreement
Longitudinal	0.94 Almost perfect agreement	0.96 Almost perfect agreement	0.93 Almost perfect agreement
Alligator	0.96 Almost perfect agreement	0.95 Almost perfect agreement	0.96 Almost perfect agreement
Pothole	0.92 Almost perfect agreement	0.89 Almost perfect agreement	Not applicable
Edge	0.95 Almost perfect agreement	0.91 Almost perfect agreement	0.94 Almost perfect agreement
Block	0.96 Almost perfect agreement	0.95 Almost perfect agreement	0.90 Almost perfect agreement

As shown in Table 4.8, the Kappa result for the pavement severity levels shows that the agreement between the two reviewers was excellent for all reported pavement crack types.

4.2.2 Evaluation of the Automated Measurement Method

4.2.2.A Accuracy of AM Method

The results of the analysis of pavement crack using the AM method were compared to the SAM method results for the two scenarios, pavement crack types and severity levels. Kappa coefficient (k) was used to evaluate the level of agreement between the two pavement crack measurement methods. The agreement between the two methods is described in the following subsections.

4.2.2.B Agreement by Pavement crack Type

Table 4.9 shows the agreement results of pavement crack types as obtained by comparing the results of the SAM versus AM methods.

Table 4.9: Kappa agreement strength of pavement crack types between the SAM and AM methods.

Pavement crack Type	Kappa "K" Value	Agreement
Transverse	0.65	Substantial agreement
Longitudinal	0.81	Almost perfect agreement
Alligator	0.87	Almost perfect agreement
Pothole	0.79	Substantial agreement
Edge	0.79	Substantial agreement
Block	0.85	Almost perfect agreement
Overall pavement crack	0.89	Almost perfect agreement

4.2.2.C Agreement by Pavement crack Severity

Table 4.10 displays the agreement results of pavement crack severity levels as obtained by comparing the results of the SAM versus AM methods.

As shown in Table 4.10, the Kappa result indicates that the majority agreement between the SAM and AM methods was found to be substantial agreement for each pavement crack at the three levels of pavement severity. Additionally, there was no severity noticed at the high severity level when the pavement crack type was identified as pothole.

Table 4.10: Kappa agreement strength of pavement crack severity levels between the SAM and AM methods.

Pavement crack Type	**Kappa "K" Value**		
	Low	**Moderate**	**High**
Transverse	0.75 Substantial agreement	0.75 Substantial agreement	0.67 Substantial agreement
Longitudinal	0.71 Substantial agreement	0.77 Substantial agreement	0.54 Moderate agreement
Alligator	0.74 Substantial agreement	0.85 Almost perfect agreement	0.77 Substantial agreement
Pothole	0.88 Almost perfect agreement	0.76 Substantial agreement	Not Applicable*
Edge	0.87 Almost perfect agreement	0.87 Almost perfect agreement	0.73 Substantial agreement
Block	0.68 Substantial agreement	0.81 Almost perfect agreement	0.78 Substantial agreement

*No severity noticed

4.2.2.D Validation of AM Method

In order to validate the AM method, the coefficient of determination, R^2, was calculated. The R^2 was found to be 0.925 which indicate that there was a very good fit relationship between the two methods. Coefficient of determination, R^2, of 0.925 was found between the individual pavement crack measurement methods, which are the estimation of the pavement cracks for the 5070 reviewed captured photos, using SAM and AM methods.

The difference in counting cracks on the overall sections between the semi-automated and automated measurement methods was calculated. If the difference is zero then the records agree; otherwise there could be a difference ranging from -1 to +10. Figure 4.11 shows the count of the difference divided by 507 versus the numerical value of the difference. The correlation coefficient, *r*, was found to be a positive correlation and was equal to 0.961.

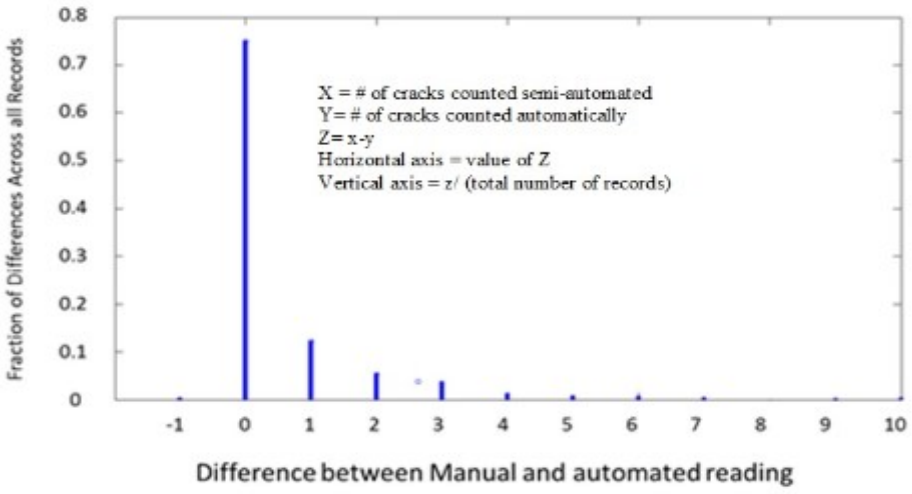


Figure 4.11: Correlation between semi-automated and automated measurement methods.

4.3 Data Analysis and Results

4.3.1 Statistics Analysis

Figure 4.12 provides frequency distributions of pavement crack types of the AM method. As shown in these figures, the majority of the pavement cracks were longitudinal and alligator cracking.

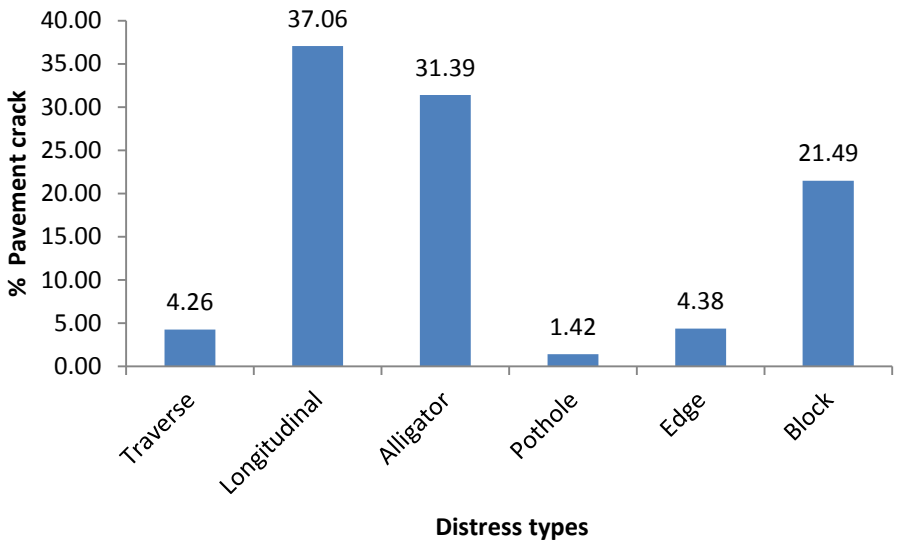


Figure 4.12: Pavement crack types distribution.

Figure 4.13 to Figure 4.15 illustrate the effect of the time period during the process of capturing photos during the day. As shown from the figures, pavement cracks were very clear in the morning period. Therefore, the diagnosis of pavement crack from the captured photos of the morning was very clear, although the survey of the road section was the same during a different time period. Specifically, morning measurements provided a higher percentage of cracks which meant that the lighting condition provided a better visibility to identify pavement cracks.

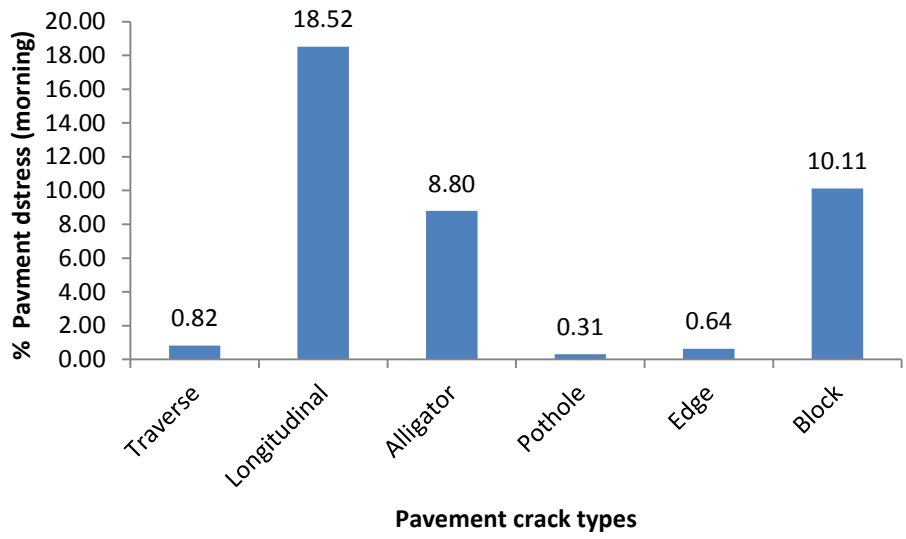


Figure 4.13: Pavement cracks distribution in the morning.

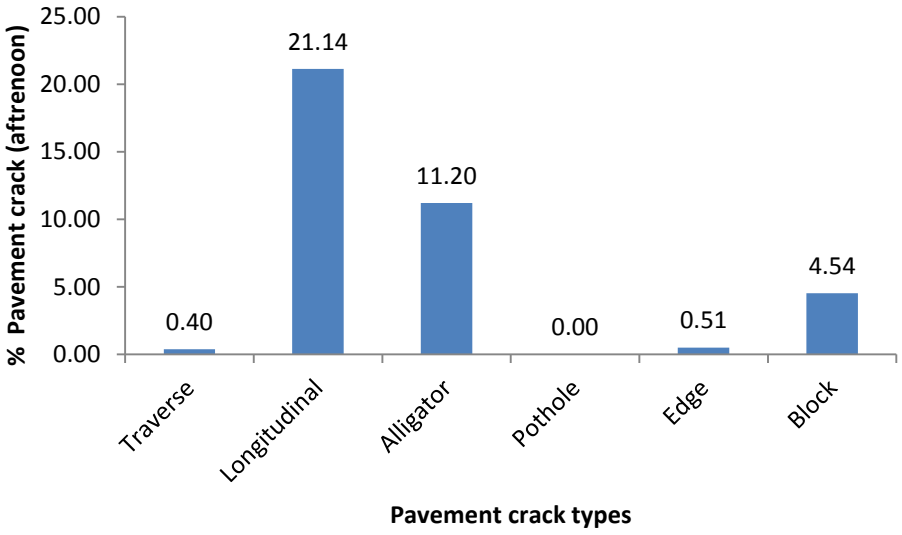


Figure 4.14: Pavement cracks distribution in the afternoon.

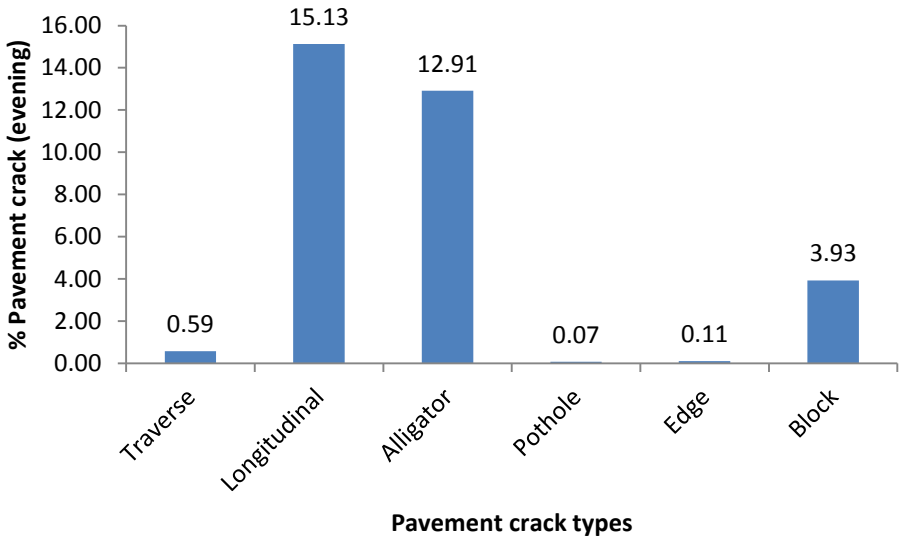


Figure 4.15: Pavement cracks distribution in the evening.

In order to make the automated measurement method more efficient, five different speed photos captures were used to survey the whole road section while capturing pavement crack photos as shown in Figure 4.16. Table 4.11 shows the results of the total pavement crack related to each speed. Figure 4.17 shows the weighted average speed photo captures, which was calculated by weighing every reported speed by the sample size. Weighted mean speed and standard deviation were 58.56 km/h and 28.24 km/h respectively.

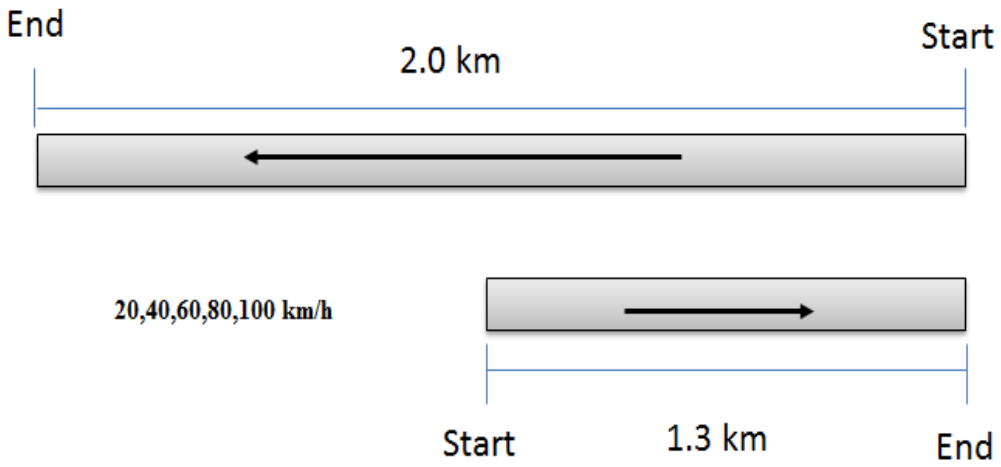


Figure 4.16: Schematic diagram of the surveyed road section.

Table 4.11: Total number of pavement crack under different speed photo captures detection sensitivity to speed.

Pavement crack	**Speed photo captures km/h**					**Total**
	20	**40**	**60**	**80**	**100**	
Transverse	18	24	20	23	17	102
Longitudinal	195	205	170	173	145	888
Alligator	140	104	161	171	176	752
Pothole	8	4	7	3	12	34
Edge	32	26	22	16	9	105
Block	135	102	106	99	73	515
Total	528	465	486	485	432	2396

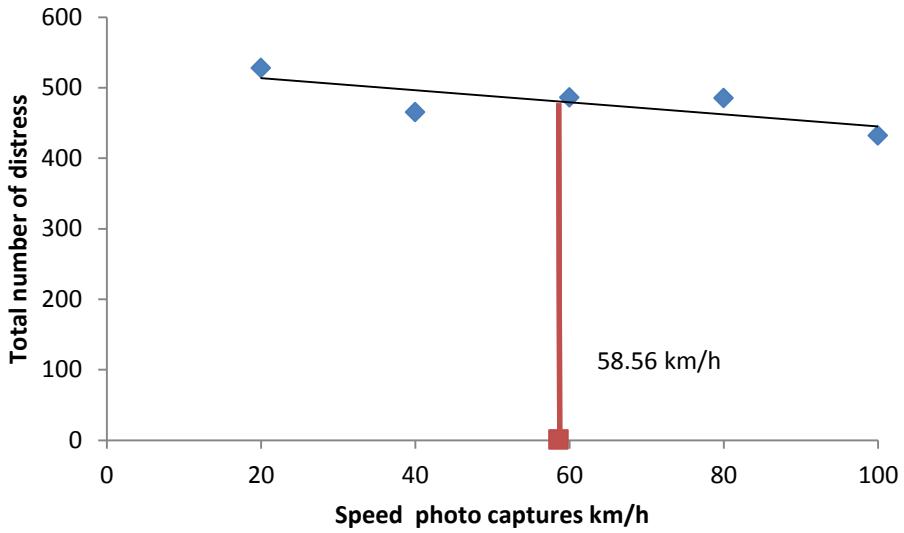


Figure 4.17: The weighted average speed photo captures.

Figure 4.18 to Figure 4.23 provides frequency distributions of speed photo captures for each distress type for the AM method. As shown in these figures, no relationship can be seen between the number of the transverse crack and pothole distress with the speed photo captures. For the alligator crack, the number of crack increased when the speed photo captures increased. While in the case of the longitudinal, edge and block cracks, and the number of crack decreased when the speed photo captures increased.

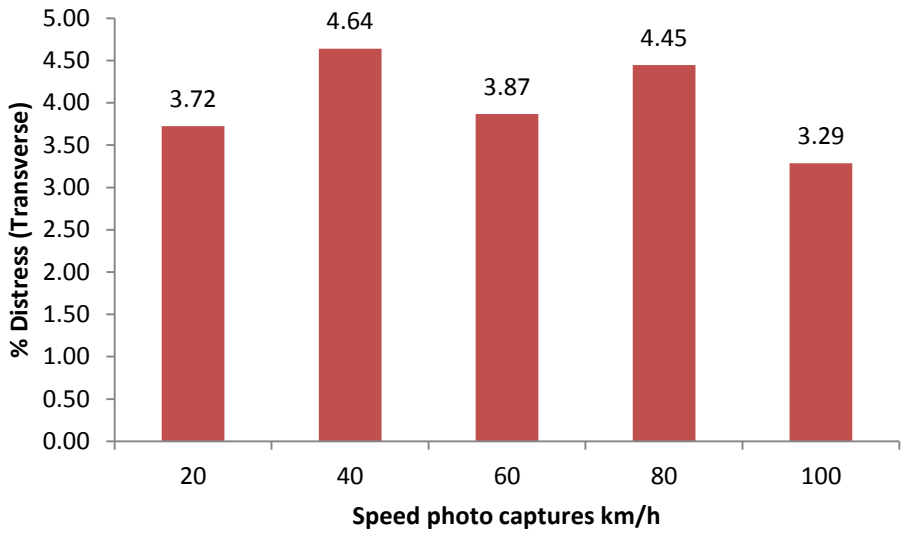


Figure 4.18: Transverse cracking distribution under different speed photo captures.

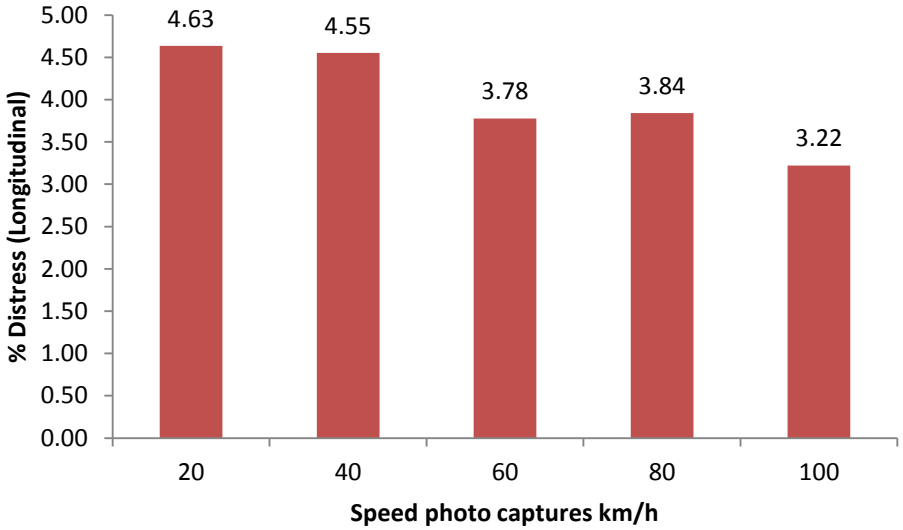


Figure 4.19: Longitudinal cracking distribution under different speed photo captures.

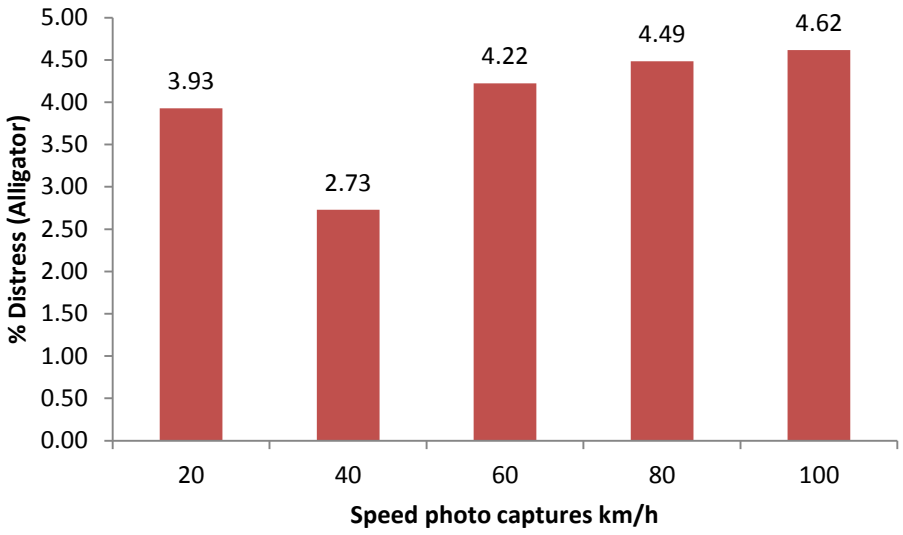


Figure 4.20: Alligator cracking distribution under different speed photo captures.

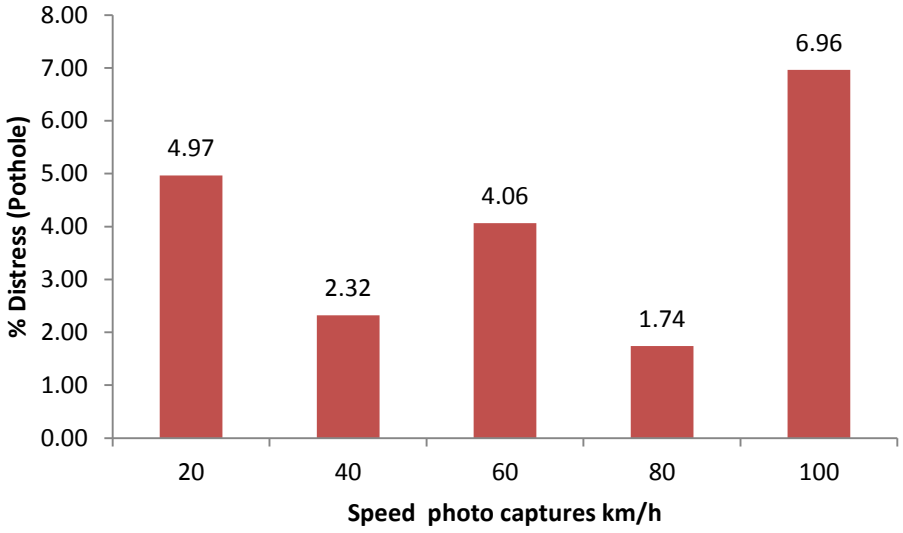


Figure 4.21: Pothole distress distribution under different speed photo captures.

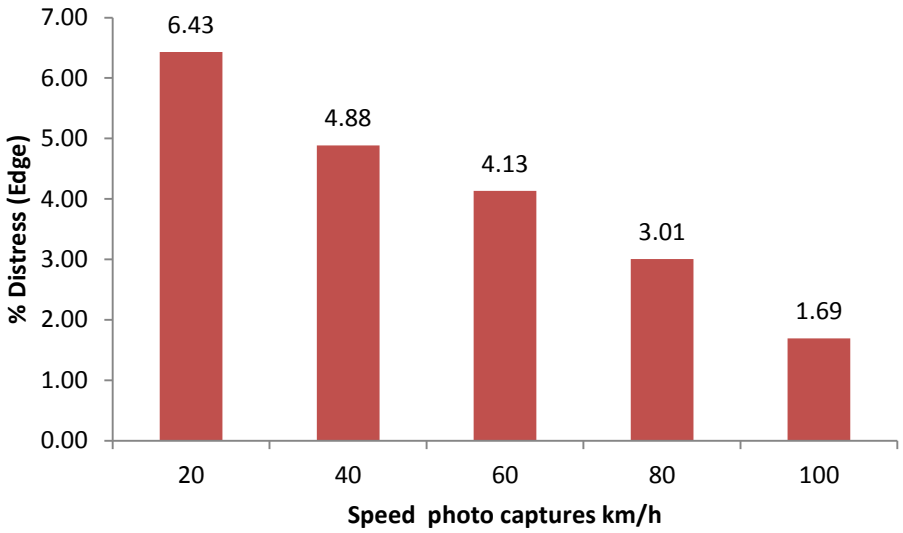


Figure 4.22: Edge cracking distribution under different speed photo captures.

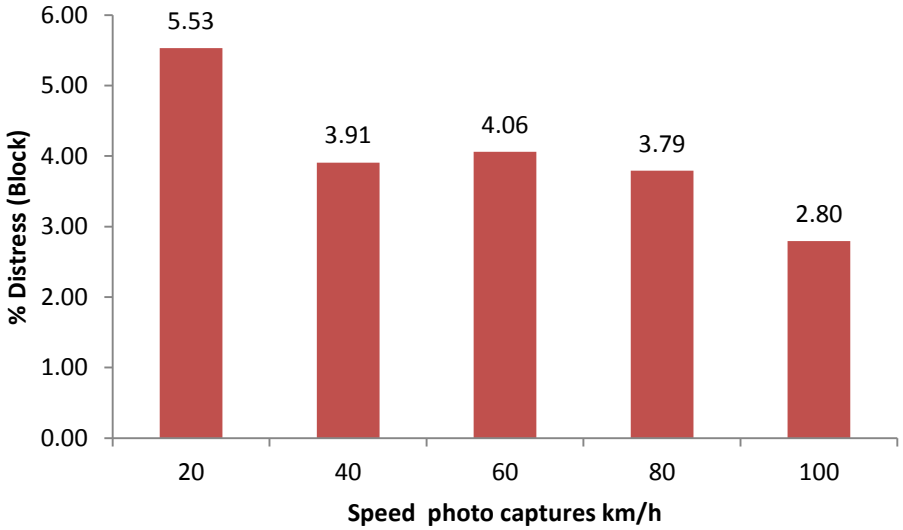


Figure 4.23: Block cracking distribution under different speed photo captures.

4.3.2 Regression Modelling of Collected Data

The overall objective of measuring the road distresses is to estimate the status of the road at a different location and to use this information to develop an efficient maintenance schedule. To that end, one can follow the following generic approach:

(a) Collect some initial data and process them using the two methods mentioned above AM and SAM.

(b) Consider the SAM to be the actual status of the road at the time of measurement and develop a regression model to fit the AM data.

(3) Update the regression model continuously using AM data only.

The rationale behind this three step approach is twofold: first, the measurements has to be done fully automated and to run continuously to infinity such that road managers can at any time look at a city map and know which section needs repairs and which sections are in good shape. Second, the effectiveness of the AM under different lighting and speed conditions has to be examined. In steps (a) and (b) the SAM were to be used on a temporary basis to simply establish the effectiveness of such parametric effectiveness.

The approach can be further explained using the following example: Consider a single road of length L meters. Divide the road into measurement segments numbered 1 to M, where the length of each segment is L/M meters (10 meters in our case). Public vehicles can be used to take measurements at different times while they are travelling at different speeds. Note that monitoring the road status is not the primary function of these public vehicles. These vehicles could be public transportation buses, or any cars used by municipalities for various functions. Sampling the road status is performed as a

secondary function for such vehicles and it is done automatically without any participation from the driver or any other operator. Therefore, the sampling of the road status (number of cracks) is done in a non-uniform way. Over a certain period of time, some road segments could be measured several times while other segments may not be measured at all. Moreover, the equipment used to perform the measurements is assumed to be inexpensive (installed in many cars) and hence the measurements are not very accurate. With all these facts in mind, it needs to try to estimate the number of cracks in each segment. Figure 4.24 schematically illustrates the relation between the actual number of distresses and the estimated numbers obtained from the in-accurate measurements described above.

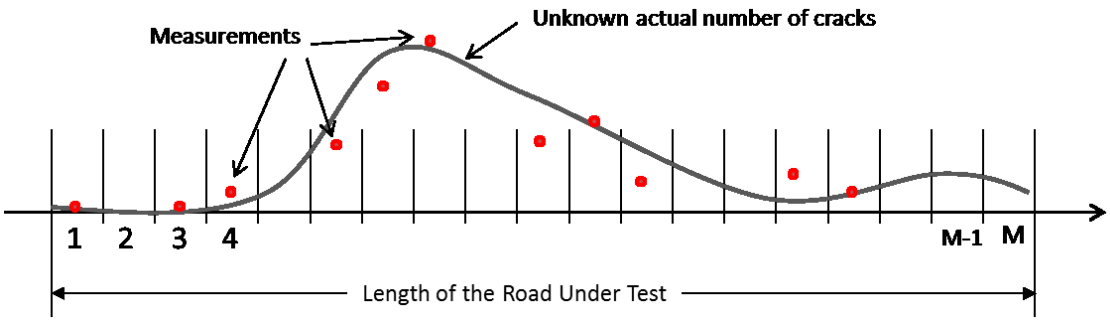


Figure 4.24: Schematic diagram; relationship between the actual and estimated numbers of distresses in-accurate measurements.

Note that the measurement points shown schematically in the graph do not have the same level of reliability. The Confidence in their values depends on the travelling speed and the time of day. Then, all data were (with different levels of reliability) to form the most likely value of the unknown function (represented by the solid line), and the measurement parameters must be accounted for.

In the following sections, a Poisson Regression Model and the Akaike Information Criterion (*AIC*) were used to effectively estimate the road status.

4.3.3 Regression Analysis of Count Data

Throughout the thesis, the crack counts were used as the main measure of road distress. Since cracks are counted for as integer numbers, the Poisson model was used.

The goal of this section is to develop statistical models which can be used to estimate the effect of different factors that could be affected on the estimation of the count number of the pavement crack by using the automated measurement method [74].

4.3.3.A Count Models

Several methods can be used to model data counts (pavement crack in the thesis). The most popular regression that can be used to model data counts is a Poisson model [75]. Regression analysis for Poisson regression takes place within the framework of generalized linear regression as follows:

$$m_i = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_n x_n} \quad \text{.....(4.2)}$$

Where *mi* is the mean value of the pavement crack count given the values of variables x_1, x_2, …, x_n.

The typical objective of regression analysis is to estimate the parameters β_0, β_1, β_2,… ,β_n. The covariates can potentially affect the frequency of pavement crack counts and this is tested during the parameter estimation process.

In any scan there are two conditions represented by two parameters: The first condition is the time of day (morning (βt_1), afternoon (βt_2) and evening (βt_3)), and the second condition is the car speed (20 km/hr (βs_1), 40 km/hr (βs_2), 60 km/hr (βs_3), 80 km/hr (βs_4) and 100 km/hr (βs_5)). Another parameter is the intercept point β_0.

Each scan produces a number of cracks (distresses). The purpose of the regression model is to estimate the number of cracks given the time of day and the car speed. Therefore, the regression model has three coefficients: β_0, βt_k and βs_n, where k =1,2 or 3 and n=1,2,3,4, or 5. Time and speed coefficients are associated with a covariant value: xt_1, xt_2, xt_3, xs_1, xs_2, xs_3, xs_4 and xs_5. Therefore, the basic model is:

$$\hat{m} = exp^{(\beta_0 + xt_k\, \beta t_k + xs_n\, \beta s_k)} \qquad \text{..... (4.3)}$$

Where $\hat{m} = the\ estimated\ number\ of\ cracks$

The covariates can potentially affect the frequency of pavement crack counts and this is tested during the parameter estimation process.

4.3.3.B Effect of Pavement crack Types, Day Time Captures, and Speed Photo Captures on Pavement crack Counts

In order to study the relationship between the dependent variables (pavement crack counts) and the independent variables (distress types, day time captures, and speed photo captures), Poisson regression, and goodness of fit measures were conducted using relevant packages in the *R* statistical computing language [76] . The strategy that was used to identify any pavement crack from a photo is noted as either 1 "crack" or 0 "no crack".

Several fitted models were conducted and the comparison of model fit is conducted using the Akaike Information Criterion (*AIC*) [77]. The improved model fit is dependent upon the lower values of *AIC* without using too many parameters. The *AIC* is defined as the measurement of model quality for statistical models to select the best model among models;

$$AIC = -2log - likelihood + k * n_{par} \quad \text{.....(4.4)}$$

Where *AIC* is the Akaike Information Criterion, $k = 2$ for the usual *AIC* or $k = log\ (n)$ (n is the number of the observation), and n_{par} represents the number of parameters in the fitted model. Appendix B illustrated the code written by the R language for count regression model.

Table 4.12 shows the results of the Poisson regression. Depending on the *AIC* value (3957.6) [78], the best model was the model which used the morning time as the day time photo captures as well as 100 km/h speed photo captures.

Table 4.12: Summary of the results of the Poisson regression of the pavement cracks.

Model No.	Distribution	Intercept (p_value)	Day Time Photo Captures			Speed (km/h)					
			Morning	Afternoon	Evening	20	40	60	80	100	AIC
		β_0	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	
1	Poisson	1.47 (<0.001)	0.14 (0.007)	0.095 (0.063)	-		-	-	-		3960
2	Poisson	1.52 (<0.001)	0.09 (0.04)	-	-	-	-	-	-		3961.5
3	Poisson	1.57 (<0.001)	-0.04 (0.41)	-	-0.95 (0.063)	-	-	-	-		3960
4	Poisson	1.47 (<0.001)	0.14 (0.007)	0.095 (0.063)	NA	-	-	-	-	-	3960
5	Poisson	1.61 (<0.001)	-	-0.04 (0.41)	-0.14 (0.007)	-	-	-	-		3960
6	Poisson	1.38 (<0.001)	0.14 (0.007)	0.095 (0.063)	-	0.13 (0.043)	0.073 (0.27)	0.12 (0.075)	0.12 (0.078)	-	3962.7
7	Poisson	1.46 (<0.001)	-	-	-	0.13 (0.04)	0.074 (0.27)	0.12 (0.075)	0.12 (0.08)	-	3966.2
8	Poisson	1.48 (<0.001)	0.14 (0.007)	0.095 (0.064)	-	0.053 (0.28)	-	-	-	-	3960.8
9	Poisson	1.47 (<0.001)	0.14 (0.007)	0.095 (0.063)	-	-	-0.0199 (0.7)	-	-	-	3961.9
10	Poisson	1.47 (<0.001)	0.14 (0.007)	0.095 (0.063)	-	-	-	0.035 (0.49)	-	-	3961.5
11	Poisson	1.47 (<0.001)	0.14 (0.0068)	0.096 (0.062)	-	-	-	-	0.034 (0.51)	-	3961.6
12	Poisson	1.49 (<0.001)	0.14 (0.007)	-	-	-	-	-	-	-0.11 (0.038)	3957.6

4.4 Summary

This chapter presented the statistical analysis and models in order to evaluate the data collection of the prototype vehicle. A total of 5070 images and 507 sections were tested, and six types of pavement cracks were tested. There were two distress measurement methods used to identify pavement cracks; semi-automated measurement method and automated measurement method. In order to evaluate the accuracy of the automated measurement method, two expert observers were used individually to extract the pavement crack by using the semi-automated measurement method. The Cohen's weighted Kappa was used to determine the agreement between the two observers. The overall agreement result of the pavement cracks between the two observers was 98%, which is an almost perfect agreement. Then, the comparison between the semi-automated and automated measurement method was conducted by using the Cohen's weighted Kappa. The overall agreement result of the pavement cracks between the two measurement methods was 89%, which is almost perfect agreement. In addition, the automated measurement method was validated by using R^2 method and found to be 0.93. The correlation coefficient, r, was found to be a positive correlation and equal to 0.96.

The relationship between the photo capture speed and the number of the all distresses was demonstrated and the weighted average photo capture speed was found to be 58 km/h. No relationship can be seen between the number of the transverse crack and pothole distress with the speed photo captures when tested individually. For the alligator crack, the number of cracks increased when the speed photo captures increased. While in the case of the longitudinal, edge and block cracks, the number of the crack decreased when the speed photo captures increased.

Several statistical methods were developed to measure the effect of the different factors on pavement crack counts. These factors are photo captures speeds at 20, 40, 60, 80, and 100 km/h and day times. The best fitted model was done by *AIC* method and it is found to be 39, 57.6. The best fit model included the photo capture speed at 100 km/h, with morning as a daytime.

CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

5.1 Summary and Conclusions

In order to ensure that constructed asphalt pavements will meet their long term objectives, highway authorities must be able to schedule and implement timely maintenance. This requires the monitoring and detection of pavement cracks utilizing reliable equipment to collect accurate data. In the past the detection and identification of pavement crack was carried out manually by trained technicians or expert inspectors who would collect the required information by walking along the road and recording the pavement cracks using data forms or by driving along the road at speeds of 5-10 km/h using clipboard or special keyboard of PC acquisition device. The limitations of using this manual method are:

1- The quality of the pavement data is affected by the experience of the rater;

2- Collecting and analyzing data using this method is time consuming;

3- To check and correct any mistakes, it requires the return to the site again resulting in more time and costs;

4. It subjects the raters to safety hazards; and

5. Manual methods cannot cover every sector of every road on a continuous basis.

In this thesis, an automated method to scan an entire city at a very high frequency of testing is presented. The new developed measurement method consisted of two cameras, GPS and Distance Measuring Instrument (DMI) as well as a board processor. The system is mounted on a public vehicle such as public transportation buses or municipality service vehicles. As those vehicles roam around doing their respective functions, is the vehicles take snapshots of the pavement, process the snapshots on board,

and transmit the reports to a control centre. Each snapshot is stamped by location and time.

A 2 km road sections in Abu Dhabi was used as a test bed. More than 5000 snapshots were recorded and processed at different times (morning, afternoon and evening) and at different speed (20, 40, 60, 80 and 100 km/h). Each file was processed automatically by an Automatic Measurement (AM) to determine the number and severity of pavement cracks. The distresses are cracks of 6 different types: (1) longitudinal cracking, (2) alligator cracking, (3) block cracking, (4) pothole distress, (5) transverse cracking and (6) edge cracking. Three severity levels were considered: (1) low, (2) medium and (3) high.

The files were also processed manually by two independent observers. This manual inspection of the photo is called as Semi-Automated Measurement Method (SAM).

The statistical analysis was performed on the collected data as follows: First, the results of the SAM were studied by correlating the results reported by the two independent observers. The "Kappa" factor was then used to test this correlation. The Kappa result was found to be 98% between the two different reviewers, which is almost perfect agreement. Also, the automated measurement method was validated using R^2 method, which was found to be 92.5%. In addition, the correlation coefficient, r, between the Semi-Automated measurement and the Automated measurement methods was found to be a positive correlation and equal to 96.1%. Akaike Information Criterion was used to test the quality of distress estimation under different lighting and speed conditions. R-software was used to obtain the required statistical parameters as was explained in Chapter 4.

5.2 Conclusions

The work and results presented in the thesis support the following conclusions:

1- This research presents a reliable and cost effective technique to collect comprehensive field data related to pavement crack and pavement condition.

2- The newly developed system is unique and innovative since it mobilizes public transit vehicles with the equipment of the new system. Thus, unlimited amounts of data collection can be performed over the entire road network for most of the day and for seven days a week.

3- The new developed system is safer and less expensive than the other current systems and would allow images to be taken in various times of the day as well as with different speeds which provide several advantages over the ones that have to be operated either manually or with high speed vehicles.

4- The new developed system can be run by using the inventor as a power source. This feature added another advantage which is less carbon emission and more "environmentally friendly".

5- Practically, the new developed measurement method can support large-scale data collection for investigating different pavement cracks.

6- It was found that the speed to capture more reliable and clear pavement data was 58 km/h. Since the maximum allowable speed of the buses at the surveyed city is 60 km/h, then the range flows within the maximum speed of public

transit buses. Also, the fact that the buses stop can provide a more reliable pictures at their stops and intersections.

7- No relationship was found between the number of the transverse cracking and pothole distress with the speed photo captures. In the alligator cracking, the amount of the cracking increased when the speed photo captures increased. While in the case of the longitudinal cracking, edge cracking and block cracking, the number of cracks decreases as the photo speed capturing increases.

8- There is no significant difference in the percentage of Alligator cracking detection using different speed of photo capturing.

9- Depending on the lowest value of the *AIC* that expressed by "$AIC = -2\log -likelihood + k * n_{par}$ ", the best fit model done by (*AIC*) method was 3957.6. This model included the factors of, morning as the time of day, and the vehicle speed at 100 km/h.

10- Longitudinal and alligator cracking were found to be the majority of the pavement crack types among all different pavement crack types reported on the test sections of this research.

5.3 Recommendations and Future Work

Based on the results and conclusions of this thesis, the following points require further research:

1- The presented technique can benefit from automated road user classification using 3D cameras. This can be utilized to measure other distress types that cannot be measured using the 2D cameras. Clearly, surface rutting and deformation will be ideal for the 3-D cameras.

2- Road map can be added to the presented technique in order to prioritize the pavement maintenance and rehabilitation projects.

3- In this thesis, if the data were to be collected for a long time, then to estimate the status of the road and improve these estimations on a continuous basis using the collected data, then the true value of the road status without assuming a specific model can be approached. A very large amount of data collected can be relied upon and processed automatically and continuously using a fairly large number of public vehicles.

4- There are huge advanced technologies such as computer vision that continuously reduce the cost and improve the quality of the system proposed in this thesis. Therefore, this technique is accepted and can be replaced by the all other techniques.

Bibliography

[1] S. P. Washington, M. G. Karlaftis and F. L. Mannering, Statistical and Econometric Methods for Transportation Data Analysis, Chapman and Hall/CRC, 2010.

[2] Federal Highway Administration, "Table HM-20: Public Road Length,2013,Miles By Funnctional System," Office of Highway Policy Information, Washington, DC, 2014.

[3] W. Cox and J. Love, The Best Investment A Nation Ever Made: A Tribute to The Dwight D.Eisenhower System of If Interstate and Defense Highways, American Highway Users Alliance, June 1996.

[4] US Department of Transportation, "U.S. Department of Transportation, Federal Highway Administration," The National Highway System, 4 Apirl 2011. [Online]. Available: http://www.fhwa.dot.gov/planning/nhs/. [Accessed 14 2 2013].

[5] National transportation Library, Intermodal Surface Transportation Efficiency Act of 1991, Washington, DC: U.S. Department of Transportation/Research and Innovative Technology Administration, 2013.

[6] U.S. Department of Transportation, " U.S. Department of Transportation, Federal Highway Administration," 2009. [Online]. Available: http://www.fhwa.dot.gov/policyinformation/statistics/2009/vmt421.cfm. [Accessed 7 2 2013].

[7] Canadian National Committee of the World Road AssociationIn Partnership with Ontario, "Optimizing Road Infrastructure Investments and Accountability," in *XXth World Road Congress*, Seoul, 2015.

[8] R. C. Hass, W. R. Hudson and J. P. Zaniewski, Modern Pavement Management, Malabar,FL: Krieger Pub. Co, 1994.

[9] A. Ferreira, R. Micaelo and R. Souza, "Cracking Models for Use in Pavement Maintenance Management," *7th RILEM International Conference on Cracking In Pavements,* pp. 429-439, 2012.

[10] U.S.Department of Transportation, "Transportation in the United States-A Review," Bureau of Transportation Statistics, Washington, DC, 2010.

[11] K. A. Zimmerman, L. R. Neeley and B. C. Schvaneveldt, "Building on A State`s Experience in Pavement Management Over Time," in *5th International Conference on Managing Pavement*, Seattle, 2001.

[12] S. S. Adlinge and A. K. Gupta, "Pavement Deterioration and Its Causes," *IOSR Journal of Mechanical & Civil Engineering,* pp. 09-15, 2013.

[13] Oregon Department of Transportation, "Pavement Management Report," Public Works Maintenance, Surface Technical Team, Oregon, 2015.

[14] S. Friedman, "The Effects of Dynamic Decision Making on Resource Allocation: The Case of Pavement Management," Worcester Polytechnic Institute, Worcester, 2003.

[15] R. G. Hicks and J. P. Mahoney , "Collection and Use of Pavement Condition Data," Transportation Research Board, Washintgton, DC, 1981.

[16] L. M. Pierce, G. McGovern and K. A. Zimmerman , "Practical Guide For Quality Management of Pavement Condition Data Collection," Federal Highway Administration, U.S. Department of Transportation, Washington, DC, 2013.

[17] N. Attoh-Okine and O. Adarkwa , "Pavement Condition Surveys- Overview of Current Practices," Delaware Center for Transportation, Newark, Delaware, 2013.

[18] K Feighan PMS Ltd, "Pavement Condition Study Report," Department of Environment, Hertiage and local Government, Ireland, 2015.

[19] M. Y. Shahin, Pavement Management for Airports, Roads and Parking lots, Newyork: Chapman and Hall, 2011.

[20] S. Zinke, J. Mahoney and T. H. Meyer, "Evaluating the Long-Trem Performance of Pavements Thermally Imaged During Construction Phase1: Developing Spatial Tools for Location Indentification," Connecticut Transportation Institute report number Ct-2240-f-08-10, Connecticut, 2009.

[21] Department of Transportation-AbuDhabi, "Pavement Data Collection Reports," Abu Dhabi DOT, Abu Dhabi, 2015.

[22] Arkansas State Highway and Transportation Departmen, "Arkansas State Highway and Transportation Department," 2014. [Online]. Available: Http://www.arkansashighways.com/planning_research/pavement_management/pav

ement_managment.aspx. [Accessed 15 1 2015].

[23] The LPA Group Transportation Consultants, "Final Report Landside Pavement Evaluation," Jacksonville Aviation Authority, Jacksonville, 2012.

[24] American Society of Testing Materials, Standard Practice for Roads and Parking lots for Pavement Condition Index Surveys, D6433, West Conshohocken, PA: American Society of Testing Materials, 2011.

[25] Strategic Highway Research Program, National Research Council, Identification Manual for the long-Term Pavement Performance Project, Washington, DC: National Research Council, 1993.

[26] American Association of State Highway and Transportation Officials, Standard Practice for Quantifying Cracks in Asphalt Pavement Surface, AASHTO R 55-10, Washington, DC: American Association of State Highway and Transportation Officials, 2013.

[27] New York State Department of Transportation, "Pavement Condition Assessment, Volume 2," New York State Department of Transportation, New York, 2010.

[28] N. Vtillo, N. Gucunski, C. Rascoe and S. Zaghloul, "Evaluation of the Automated Distress Survey Equipment," New Jersey Department of Transportation, New Jersey, 2009.

[29] Oregon Department of Transportation, "Automated Data Collection Equipment for Monitoring Highway Condition," Oregon Department of Transportation, Oregon,

2005.

[30] Oregon Department of Transportation, "Automated Data Collection Equipment for Mointoring Highway Condition," Oregon Department of Transportation, Oregon, 2005.

[31] G. Flintsch and K. K. McGee, "Quality Management of Pavement Condition Data Collection," Transportation Research Board, Charlottesville, Virginia, 2009.

[32] D. T. Hartgen and J. J. Shufon, "Windshield Surveys of Highway Condition: A Feasible Input to Pavement Management," *Transportation Research Board,* no. 938, pp. 73-81, 1984.

[33] H. Rababaah, Asphalt Pavement Crack Classification: A Comparative Study of Three AI Approaches: Multilayer Perception, Genetic Algorithms and Self-Organizing Maps, Indiana: Indiana University, May 2005.

[34] D. H. Timm and J. M. McQueen, "A Study of Manual VS. Automated Pavement Condition Surveys," Alabama Department of Transportation, Montgomery, AL, 2004.

[35] Oregon Department of Transportation, "Distress Survey Manual," Oregon Department of Transportation, Oregon, 2010.

[36] U.S. Department of Transportation, "Roadway Safety Hardware Asset Management Systems Case Studies," Federal Highway Administration, New Jersey, 2005.

[37] J. A. Walther and M. Y. Shahin, "Pavement Maintenance Management for Roads and Streets Using Paver System," US Army Corps of Engineers, Champaign, 1990.

[38] K. H. McGhee, "NCHRP Synthesis of Highway Practice 334: Automated Pavement Distress Collection Techniques," National Cooperative Highway Research Program, Transportation Research Board, Washington, DC, 2004.

[39] Virginia Department of Transportation, "A Guide to Evaluating Pavement Distress Through the Use of Video Images," Virginia Department of Transportation, Richmond, 1998.

[40] K. C. Wang, "Designs and implementations of Automated Systems for Pavement Surface Distress Survey," *Journals of Infrastrucutre Systems, ASCE,* vol. 6, no. 1, pp. 24-32, 2000.

[41] T. Fukuhara, . K. Terada, . M. Nagao, A. Kasahara and S. Ichihashi, "Automatic Pavement-Distress-Survey System," *Journal of Transportation Engineering,* vol. 116, no. 3, pp. 280-286, 1990.

[42] IMS Infrastructure Management Services, "An Overview of the PAVUE Fully Automated Surface Distress Image Processing System," IMS, 1996.

[43] CGH Pavement Engineering Inc., "Distress Surveys," CGH Pavement Engineering Inc., 2002. [Online]. Available: www.cgh-pavement.com. [Accessed 13 12 2012].

[44] Pathway Services Inc., "Pathway Systems," Pathway Services Inc., 2002. [Online]. Available: www.pathwayservices.com. [Accessed 16 12 2012].

[45] Mandli Communications Inc., "Mandli`s Pavement System," Mandli Communications Inc., 2002. [Online]. Available: www.mandli.com. [Accessed 17 12 2012].

[46] R. W. Miller, M. Raman , M. Hossian, G. Cumberledge and J. Reigle, "Comparison of Automated Distress Survey Results with the Manual Distress Surveys," *Transportation Research Board,* 2003.

[47] International Cybernetics Corporation, "ICC Collection System," International Cybernetics Corporation, 2011. [Online]. Available: www.intlcybernetics.com. [Accessed 12 1 2013].

[48] S. Cafiso, A. Di Graziano and B. Sebastiano, "Evaluation of Pavement Surface Distress Using Digital Image Collection and Analysis," in *Seventh International Congress on Advances in Civil Engineering*, Yildiz, Turkey, October 2006.

[49] J. A. Epps and C. L. Monismith, "Equipment for Obtaining Pavement Condition and Traffic Loading Data," Transportation research Board, NCHRP Synthesis 126, Washington, DC, 1986.

[50] " Transport Research Laboratory (TRL)," Highways Agency Road Research Information System, [Online]. Available: http://www.trl.co.uk/facilities/mobile_test_equipment/ . [Accessed 15 1 2013].

[51] J. K. Cable and V. J. Marks, "Automated Pavement Distress Data Collection Equipment," Federal Highway Administration, Washington, DC, 1990.

[52] Road Group Inc., "Auotmatic Road Analyzer System," Road Group Inc., 2002. [Online]. Available: www.roadware.com. [Accessed 3 2 2013].

[53] D. P. William, "Proposal of Universal Cracking Indicator for Pavements," *Transportation Research Board,* no. 1445, pp. 69-74, 1994.

[54] Texas Department of Transportation, "Pavement Management Information Systems Rater`s Manual," Texas Department of Transportation, Austin, 1999.

[55] K. Wang, W. Gong, X. Li, R. Elliott and J. Daleiden, "Data Analysis of Real-Time System for Automated Distress Survey," *Transportation Research Board,* no. 1806, pp. 101-109, 2002.

[56] G. D. Cline, M. Y. Shahin and J. A. Burkhalter, "Automated Data Collection For Pavement Condition Index Survey," *Transportation Research Board,* 2003.

[57] K. Wang, "Automated Imaging Technologies for Pavement Distress Survey," Transportation Research Board, Washington, DC, 2003.

[58] J. L. Groeger, . P. Stephanos, P. Dorsey and M. Chapamn, "Implementation of Automated Network-Level Crack Detection Processes in Maryland," *Transportation Research Board,* vol. 1860, no. 03-3199, pp. 109-115, 2003.

[59] H. N. Koutsopoulos and A. B. Downey, "Primitive-based Classification of Pavement Cracking Images," *Journal of Transportation Engineering,* vol. 119, no. 3, pp. 402-418, 1993.

[60] M. S. Kaseko, Z.-P. Lo and S. G. Ritichie, "Comparison of Traditional and Neural

Classifiers for Pavement-Crack Detection," *Journal of Transportation Engineering,* vol. 120, no. 4, pp. 552-569, July 1994.

[61] Adhara Systems Inc., "Adhara System," Adhara Systems Inc., 2011. [Online]. Available: www.adharasys.com. [Accessed 12 2 2013].

[62] H. Lee and . J. Kim, "Development of A Manual Crack Quantification and Automated Crack Measurement System," Iowa Deepartment of Transportation, Iowa, 2005.

[63] Dynatest Consulting Inc., "Uni analyze," Dynatest Consulting Inc., 2011. [Online]. Available: www.dynatest.com. [Accessed 1 3 2013].

[64] G. Weiguo and W. Kelvin, "Real-Time Automated Survey of Pavement Surface Distress," *Journal of Infrastructure Systems,* vol. 11, no. 3, pp. 154-164, 2005.

[65] N. A. Solter and S. J. Kleper, Professional C++, New Jersey : Wiley Publishing, Inc, 2005.

[66] M. A. Joshi, Digital Image Processing An Algorithmic Approach, New Delhi: Prentice Hall of India Private Limited, 2007.

[67] L. J. Galbiati, Machine Vision and Digital Image Processing Fundamentals, New Jersey: Prentice Hall, 1990.

[68] T. Acharya and A. K. Ray, Image Processing Principles and Applications, New Jersey: Wiley-Interscience, 2005.

[69] R. Graham, Digital Imaging, Scotland: Whitles Publishing Services, 1998.

[70] R. C. Gonzalez, R. E. Woods and S. L. Eddins, Digital Image using Matlab Procesing, New Jersey: Pearson Prentice Hall, 2004.

[71] L. Sun, M. Kamaliardakani and Y. Zhang, "Weighted neighborhood Pixels Segmentation Method for Automated Detection of Cracks on Pavement Surface Images," *Jorunals of Computing in Civil Engineering,* 2015.

[72] A. J. Viera and J. M. Garrett, "Understanding Interobserver Agreement: The Kappa Statistic," *Family Medicine,* vol. 37, no. 5, pp. 360-363, 2005.

[73] S. Goodman, "Assessing Variability of Surface Distress Surveys in Canadian Long-Term Pavement Performance Program," *Trabsportaion Research Board,* vol. 1764, pp. 112-118, 2001.

[74] F. Hong and J. A. Prozzi, "Using Count to Model Infrastructure Distress Initiation and Progression," *Journal of Infrastructure Systems,* vol. 21, no. 1, pp. 04014028-1-7, 2015.

[75] H.-W. Ker, Y.-H. Lee and P.-H. Wu, "DEVELOPMENT OF FATIGUE CRACKING PERFORMANCE PREDICTION MODELS FOR FLEXIBLE PAVEMENTS USING LTPP DATABASE," *Transportation Research Board,* vol. 1999, p. 14, 2007.

[76] R_Core_Team, R: A Language and Environment for Statistical Computing, Vienna: R Foundation for Statistical Computing, 2012.

[77] H. Akaike, "A New Look at the Statistical Model Identification," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. AC-19, KO. 6, DECEMBER*

1974 , Vols. AC-19, no. 6, pp. 716-722, 1974.

[78] C. Chen , Evaluation of Iowa asphalt pavement joint cracking, Ames: Iowa State University , 2014.

Appendices

Appendix A : Automated Data Software development analysis code

```
#ifndef CONFIG_H
#define CONFIG_H
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <cv.h>
#include <highgui.h>
#include "utils.h"
//using namespace cv;
#define MAX_STRLEN                  24
#define MID_STRLEN                  64
#define LONG_STRLEN                 64
#define MAX_STRLEN                  256
#define USING_BACKGROUND 0
#if USING_BACKGROUND
//nothing
#else
//#define RESIZE_FRAME_SIZE
#endif
#ifdef RESIZE_FRAME_SIZE
#define RESIZE_FRAME_SIZE_WIDTH 503
#define RESIZE_FRAME_SIZE_HEIGHT 528
#endif
#define RESIZE_WINDOW //resize for the scrien just viewing
#ifdef RESIZE_WINDOW
#define RESIZE_WIDTH 320
#define RESIZE_HEIGHT 240
#endif
#define USING_COMMAND_ARGMENTS 1
#define NUMBER_PARTS_WIDTH 8
#define NUMBER_PARTS_HEIGHT 16
#define INPUT_CONTOURS_LOCAL_IMAGE_WIDTH 500
#define INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT 550
#endif //CONFIG_H
Globals.cpp
#include "globals.h"
float       gWorkingWidth     = 320.f;
float       gWorkingHeight    = 240.f;
int                 gFrameID                                        = 0;
const CvFont        gCvFont             = cvFont (1.0f, 1);
FrameSrc* gFrameSrc                     = 0L;
IplImage* gSrcGray                      = 0L;
TechInput* gTechInput       = 0L;
TechAdaptiveThreashold*             gTechAdaptiveThreashold = 0L;
TechMedianFilter*           gTechMedianFilter = 0L;
TechMorphology*             gTechMorphology = 0L;
TechContours*               gTechContours = 0L;
TechBackground*             gTechBackground= 0L;
TechSubtract*               gTechSubtract = 0L;
int                 gWidth                          = -1;
int                 gHeight                         = -1;
char* gDataDir = "";
char* gImagesRelDir = "c:/ImageProcessing/Images/";
char  gResultFileName[MAX_STRLEN]="\0";
```

```
char gImagesDir[MAX_STRLEN] = "\0";
char gImagesDirBackground[MAX_STRLEN] = "c:/ImageProcessing/ImagesBackground/";
char gSettingDir[MAX_STRLEN] = "c:/ImageProcessing/Setting/";
const CvScalar gCvWhite                 = cvScalarAll (255)         ;
const CvScalar gCvLite                  = cvScalarAll (180)         ;
const CvScalar gCvGray                  = cvScalarAll (128)         ;
const CvScalar gCvDark                  = cvScalarAll (50)          ;
const CvScalar gCvBlack                 = cvScalarAll (0)           ;
const CvScalar gCvRed       = cvScalar   (   0,   0, 255);
const CvScalar gCvGreen     = cvScalar   (   0, 255,   0);
const CvScalar gCvBlue      = cvScalar   ( 255,   0,   0);
const CvScalar gCvAqua      = cvScalar   ( 255, 255,   0);
const CvScalar gCvYellow    = cvScalar   (   0, 255, 255);
IplImage* gDebugImg                     = 0L;
int             gViisabiltyOfWindows = 1;
IplImage* gPartProcessing    = 0L;
int             gWidthPartProcessing        = -1;
int             gHeightPartProcessing       = -1;
CvRect  gRectCurrentPartProcessing = cvRect(0,0,0,0);
bool        gPartsProcessingCompleted = true;
GLOBALS_H
ifndef __L2FGLOBALS_H__
#define __L2FGLOBALS_H__
#include "config.h"
#include "frameSrc.h"
#include "techInput.h"
#include "techAdaptiveThreashold.h"
#include "techMedianFilter.h"
#include "techMorphology.h"
#include "techContours.h"
#include "techBackground.h"
#include "techSubtract.h"
// Text
#define MAX_TEXT_LINES          7
#define TEXT_LINE_HEIGHT        15
#define BORDER_SIZE                     5
// ----
// FPS
extern int          gViisabiltyOfWindows;
extern int          gFrameCnt;
extern int          gGoodFrameCnt;
extern int          gFrameID;
extern int                      gWidth;
extern int                      gHeight;
extern float                    gWorkingWidth;
extern float                    gWorkingHeight;
extern char*                    gLogFileName;
extern char*                    gDataDir;
extern const CvFont             gCvFont;
extern char         gResultFileName[MAX_STRLEN];
extern char*                    gImagesRelDir;
extern char                             gImagesDir[MAX_STRLEN];
extern char                             gImagesDirBackground[MAX_STRLEN];
extern char                             gSettingDir[MAX_STRLEN];
extern FrameSrc*            gFrameSrc;
extern IplImage*            gSrcGray;
extern TechInput*           gTechInput;
extern TechAdaptiveThreashold*          gTechAdaptiveThreashold;
extern TechMedianFilter*                gTechMedianFilter;
extern TechMorphology*                  gTechMorphology;
extern TechContours*                    gTechContours;
extern TechBackground*                  gTechBackground;
```

```
extern TechSubtract*                    gTechSubtract;
// ----
extern const CvScalar           gCvWhite ;
extern const CvScalar           gCvLite  ;
extern const CvScalar           gCvGray  ;
extern const CvScalar           gCvDark  ;
extern const CvScalar           gCvBlack ;
extern const CvScalar           gCvRed   ;
extern const CvScalar           gCvGreen ;
extern const CvScalar           gCvBlue  ;
extern const CvScalar           gCvAqua  ;
extern const CvScalar           gCvYellow;
extern IplImage*                gDebugImg;
extern IplImage* gPartProcessing;
extern int              gWidthPartProcessing;
extern int              gHeightPartProcessing;
extern CvRect   gRectCurrentPartProcessing;
extern bool                     gPartsProcessingCompleted;
#endif     //__L2FGLOBALS_H__ utils.h
#include "utils.h"
#include "config.h"
#include "globals.h"
#define min(a,b) ((a > b)? b : a)
void PathCat(char* strOut, const char* pathIn, const char* strIn)
{
        strcat_s(strOut, MAX_STRLEN, pathIn);
        strcat_s(strOut, MAX_STRLEN, strIn);
}
void ResizeImage(IplImage* img, IplImage* dst,int width , int height)
{
#ifdef RESIZE_WINDOW
        if(width !=0 && height != 0)
        {
                dst->width = width;
                dst->height = height;
                cvResize(img, dst);
        }
        Else
        {
        //Mat srcMat(img);
        //Mat dstMat(dst);
  // resize(srcMat, dstMat,cvSize((int)(RESIZE_WIDTH) , (int)(RESIZE_HEIGHT) ));
        dst->width = RESIZE_WIDTH;
        dst->height = RESIZE_HEIGHT;
        cvResize(img, dst);
        //dst = cvCloneImage(&(IplImage)dstMat);
        }
#endif
}
void WriteParmaeters(char* fileName, int count,...)
{
 int i;
 int val;
 va_list vl;
 va_start(vl, count);
 FILE * pf;
 char line [100];
 pf = fopen(fileName, "w+");
 if(pf != NULL)
 {
        for (i = 0 ; i < count ; i++ )
        {
```

```
                        val = va_arg(vl,int);
                        sprintf(line, "%04X\n", val);
                        fputs(line,pf);
}
                fclose(pf);
 } va_end(vl);
}
void ReadParmaeters(char* fileName, int count, ...)
{
                FILE * pf;
                char line [100];
                pf = fopen(fileName, "r");        int i;
  int* val = 0L;;
  va_list vl;
  va_start(vl, count);
                if(pf != NULL)
                {
                for (i = 0 ; i < count ; i++ )
                {
                        if(fgets( line, sizeof(line), pf )!= NULL)
                        {
                                val = va_arg(vl,int*);
                                sscanf(line,"%04X\n",val);
                        }
                }
                 fclose(pf);
                }
   va_end(vl);
}
void OpenNewFile(char* fileName)
{
                FILE * pf;
                char line [100];
                pf = fopen(fileName, "w+");
                if(pf != NULL)
                {
                        fclose(pf);
                }
}
void AppendHeaderToFile(char* fileName)
{
  int i;
  FILE * pf;
  char line [200];
  pf = fopen(fileName, "a+");
  if(pf != NULL)
  {
                  sprintf(line, "Image#          CRKS/\NON         CRKScount          TA                    TCA
                %%                    TCA                  Type                 Severity\n");
                fputs(line,pf);
                fclose(pf);
  }
}
void AppendToFile(char* fileName, int frameId,int crackedOrNot,int countOfcracked, float totalImageArea,float
totalCrackedArea,float PercentageOfCreacked, float area, char* type, char *severity)
{

int i;
  FILE * pf;
 if(pf != NULL)
```

```
 {
        sprintf(line, " %d     %d                    %d                  %.2f      %.2f      %f        %.2f      %-12s
        %s\n",frameId,crackedOrNot,countOfcracked, totalImageArea,totalCrackedArea,PercentageOfCreacked,
area, type, severity);
        fputs(line,pf);
        fclose(pf);
 }
}
void PutText(
                        IplImage* img
                        , CvPoint point
                        , const char* data
                        , CvScalar colour /* = cvScalarAll(255)*/)
{
        cvPutText(img, data, point, &gCvFont, colour);
}
void PutTextInternal(
                IplImage* src
                , IplImage* dst
                , CvPoint point
                , CvScalar colour
                , const char* data
                , ...)
{
        //Build string
        char tmpString[MAX_STRLEN];
        va_list marker;
        va_start(marker, data);
        vsprintf_s(tmpString, data, marker);
        //Copy src to dst
        cvSetImageROI(dst, cvGetImageROI(src));
        cvCopy(src, dst);
        //Add text
        cvPutText(dst, tmpString, point, &gCvFont, colour);
}
void PutTextExternal(
                IplImage* src
                , IplImage* dst
                , const char* data
                , ...)
{
        //Build string
        char tmpString[MAX_STRLEN];
        va_list marker;
        va_start(marker, data);
        vsprintf_s(tmpString, data, marker);
        //Split string to lines
        char* token = 0L;
        char* context = 0L;
        char  delims[]        = "\n";
        char* lines[MAX_TEXT_LINES];
        int lineNumber = 0;
        token = strtok_s(tmpString, delims, &context);
        while (token != 0L && lineNumber < MAX_TEXT_LINES)
        {
                lines[lineNumber] = token;
                lineNumber++;
                token = strtok_s(0L, delims, &context);
        }
        CvRect rect = cvGetImageROI(src);
        CvRect region;
        //if requested lines excceded the maximun lines set height to max else set height to requested lines.
```

```
		const int newHeight = min(gHeight + (MAX_TEXT_LINES * TEXT_LINE_HEIGHT) + BORDER_SIZE,
rect.height + ((lineNumber) * TEXT_LINE_HEIGHT) + BORDER_SIZE);
		region.x = 0;
		region.width = rect.width;
		if(newHeight > rect.height)
		{
			//Copy src to dst
			cvSetImageROI(dst, rect);
			cvCopy(src, dst);
			cvResetImageROI(dst);
			//Reset lines
			region.y = rect.height;
			region.height = newHeight;
			cvSetImageROI(dst, region);
			cvZero(dst);
			cvResetImageROI(dst);
		}
		else //Length of src image more than or equal Max Lenght
		{
			return;
		}
		//Set ROI
		region.y = 0;
		region.height = newHeight;
		cvSetImageROI(dst, region);
		//Add Text to image
		int currentHeight = 0;
		const int MaxHeight = gHeight + MAX_TEXT_LINES * TEXT_LINE_HEIGHT;
		for(int i = 0; i < lineNumber; i++ )
		{
			currentHeight = rect.height + ((i + 1) * TEXT_LINE_HEIGHT);
			if( currentHeight >= MaxHeight)
			{
				//warning
				region.height = currentHeight + BORDER_SIZE;
				cvSetImageROI(dst, region);
				char temp[MAX_STRLEN] = "\0";
				strcpy_s(temp, MAX_STRLEN, lines[i]);
				strcat_s(temp, MAX_STRLEN,"...");
				cvPutText(dst, temp, cvPoint(0,currentHeight), &gCvFont, gCvWhite);
				break;
			}
			cvPutText(dst, lines[i], cvPoint(0,currentHeight), &gCvFont, gCvWhite);
UTILS_H
}
}
#ifndef UTILS_H
#define UTILS_H
#include "config.h"
#include <stdarg.h>
void PathCat(char* strOut, const char* pathIn, const char* strIn);
void ResizeImage(IplImage* img, IplImage* dst,int width  = 0, int height = 0);
void WriteParmaeters(char* fileName, int count, ...);
void ReadParmaeters(char* fileName, int count, ...);
void OpenNewFile(char* fileName);
void AppendToFile(char* fileName, int frameId,int crackedOrNot,int countOfcracked, float totalImageArea,float
totalCrackedArea,float PercentageOfCreacked, float area, char* type, char *severity);
void AppendHeaderToFile(char* fileName);
 //brief Draws a given text on an image at a location (alters the image).
void PutText(
			IplImage* img
			, CvPoint point
```

```
                        , const char* data
                        , CvScalar colour = cvScalarAll(255));
 //brief Outputs a given text on an image.
void PutTextInternal(
                        IplImage* src
                        , IplImage* dst , CvPoint point
                        , CvScalar colour
                        , const char* data
                        , ...);
 //brief Outputs a given text on an image at the bottom.
 void PutTextExternal(
                        IplImage* img
                        , IplImage* dst
                        , const char* data
                        , ...);
#endif //UTILS_H
FrameSrc.h
#ifndef FRAMESRC_H
#define FRAMESRC_H
#include "config.h"
#include "assert.h"
struct SourceInfo
{
        enum SourceType
        {
                FS_NULL = 0
                , FS_SINGLE
                , FS_RANGE
                , FS_STREAM
                , FS_VIDEOSTREAM
                , FS_PARTS_SINGLE
                , FS_PARTS_RANGE
        };
        SourceType m_sourceType;
        int          m_seqStartIndex;
        int   m_seqEndIndex;
        char  m_seqImageName[MID_STRLEN];
        SourceInfo(void):
                m_sourceType(FS_SINGLE)
                , m_seqStartIndex(-1)
                , m_seqEndIndex(-1)
        {
                strcpy_s(m_seqImageName, MID_STRLEN, "\0");
        };
        void Init(
                        SourceType sourceType
                        , char* seqImageName = 0L
                        , int seqStartIndex = -1
                        , int seqEndIndex = -1)
        {
                m_sourceType = sourceType;
                if( seqImageName != 0L )
                {
                        strcpy_s(m_seqImageName, MID_STRLEN, seqImageName);
                }
                Else
                {
                        strcpy_s(m_seqImageName, MID_STRLEN, "\0");
                }
                m_seqStartIndex = seqStartIndex;
                m_seqEndIndex = seqEndIndex;
        };
```

```
        void Init(const SourceInfo &sourceInfo)


        {
                m_sourceType = sourceInfo.m_sourceType;
                assert((sourceInfo.m_seqImageName != 0L));
                strcpy_s(m_seqImageName, MID_STRLEN, sourceInfo.m_seqImageName);

                m_seqStartIndex = sourceInfo.m_seqStartIndex;
                m_seqEndIndex = sourceInfo.m_seqEndIndex;
        };
};

class FrameSrc
{
        public:
        virtual ~FrameSrc                               (void)
                {};
        virtual void InitFrameSrc               (const char* inImagesDir, SourceInfo* sourceInfo, int* width, int*
height)    = 0;

        virtual void UpdateFrameSrc             (void* outImg, void* wholeImage = NULL)       = 0;
        virtual const void* GetFrame (void)                                                   const       {return
0L;};
        virtual void EndFrameSrc                (void)
        = 0;
        virtual void GetDims                            (int* width, int* height)       const       = 0;
        virtual void GetFps                             (double* outFps)
        {};
        virtual const SourceInfo::SourceType GetSrcType             (void)             const       {return
SourceInfo::FS_NULL;};
};
#endif     //FRAMESRC_H__
frameSrcImp.ccp
#include "frameSrcImp.h"
FrameSrcImp::FrameSrcImp  (void): m_Src(0L), m_SrcRgbNoRelease(0L), m_Cpt(0L), m_width(-1), m_height(-1)
{
        strcpy_s(m_extention, MAX_STRLEN, "\0");
        strcpy_s(m_nameWithoutExt, MAX_STRLEN, "\0");
        strcpy_s(m_imagesDir, MAX_STRLEN, "\0");
}
FrameSrcImp::~FrameSrcImp             (void)
{
        //
}
void FrameSrcImp::InitFrameSrc(
                const char* inImagesDir
                , SourceInfo* sourceInfo
                , int* outWidth, int* outHeight)
{
        m_sourceInfo.Init(*sourceInfo) ;
        //if source type == IMAGE_RANGE then start index must be < end index
        assert(m_sourceInfo.m_sourceType != SourceInfo::FS_RANGE                                    ||
m_sourceInfo.m_seqStartIndex <= m_sourceInfo.m_seqEndIndex);
        strcpy_s(m_imagesDir, MAX_STRLEN, inImagesDir);
        if (m_sourceInfo.m_sourceType == SourceInfo::FS_SINGLE ||m_sourceInfo.m_sourceType ==
SourceInfo::FS_RANGE)
        {
                char fullFileName[MAX_STRLEN] = "\0";
                IplImage* tempImage;
                if (m_sourceInfo.m_sourceType == SourceInfo::FS_RANGE)
```

```
		{
			m_seqCounter = m_sourceInfo.m_seqStartIndex - 1;
			char *token1, *next_token1;
			token1 = strtok_s(m_sourceInfo.m_seqImageName, ".", &next_token1);
			if (strcmp (next_token1, "") == 0 ) //Name  .jpg
			{
				strcpy_s(m_nameWithoutExt, MAX_STRLEN, "\0");
				strcpy_s(m_extention, MAX_STRLEN, token1);
			}
			Else
			{
				strcpy_s(m_nameWithoutExt, MAX_STRLEN, token1);
				strcpy_s(m_extention, MAX_STRLEN, next_token1);
	}
			// need to load one image to obtain values for dimentions
			ObtainSeqRangeImageName(m_sourceInfo.m_seqStartIndex, fullFileName)
			while ((tempImage = cvLoadImage(fullFileName)) == 0L)
			{
				m_sourceInfo.m_seqStartIndex++;
				if (m_sourceInfo.m_seqStartIndex > m_sourceInfo.m_seqEndIndex)
				{
					assert(0);
				}
				ObtainSeqRangeImageName(m_sourceInfo.m_seqStartIndex, fullFileName);
			}
			m_seqCounter = m_sourceInfo.m_seqStartIndex - 1;
			if (m_sourceInfo.m_seqStartIndex == m_sourceInfo.m_seqEndIndex)
			{
				// treat it as FS_SINGLE; i.e., need to cache image in m_Src
				m_Src = cvCreateImage(cvGetSize(tempImage), 8, 1);
				cvCvtColor(tempImage, m_Src, CV_BGR2GRAY);
				// and make sure it is treated as single from now on
				m_sourceInfo.m_sourceType = SourceInfo::FS_SINGLE;
			}
#ifdef RESIZE_FRAME_SIZE
			m_width		= RESIZE_FRAME_SIZE_WIDTH;
			m_height = RESIZE_FRAME_SIZE_HEIGHT;
#else
			m_width		= tempImage->width;
			m_height = tempImage->height;
#endif
		}
		else if(m_sourceInfo.m_sourceType == SourceInfo::FS_SINGLE)
		{
			// if single, then need to cashe the single image in m_Src
			char imageName[MID_STRLEN] = "\0";
			strcpy_s(imageName, MID_STRLEN, m_sourceInfo.m_seqImageName);
#if USING_COMMAND_ARGMENTS
			strcpy_s(fullFileName,MID_STRLEN,imageName);
#else
			PathCat(fullFileName, m_imagesDir, imageName);
#endif
			printf("Loading image:%s\n", fullFileName);
			tempImage = cvLoadImage(fullFileName);
#ifdef RESIZE_FRAME_SIZE
			m_Src =
cvCreateImage(cvSize(RESIZE_FRAME_SIZE_WIDTH,RESIZE_FRAME_SIZE_HEIGHT), 8, 1);
			IplImage* tempImgForResize = cvCreateImage(cvGetSize(tempImage), 8, 1);
			cvCvtColor(tempImage, tempImgForResize, CV_BGR2GRAY);
			cvResize(tempImgForResize, m_Src);
			cvReleaseImage(&tempImgForResize);
#else
```

```
                        m_Src = cvCreateImage(cvGetSize(tempImage), 8, 1);
                        cvCvtColor(tempImage, m_Src, CV_BGR2GRAY);
#endif
                m_width             = m_Src->width;
                m_height = m_Src->height;
                }
                else assert(0);
                cvReleaseImage(&tempImage);
        }
        else if(m_sourceInfo.m_sourceType == SourceInfo::FS_PARTS_SINGLE)
                {
                        char fullFileName[MAX_STRLEN] = "\0";
                        IplImage* tempImage;
                        char imageName[MID_STRLEN] = "\0";
                        strcpy_s(imageName, MID_STRLEN, m_sourceInfo.m_seqImageName);
#if USING_COMMAND_ARGMENTS
                        strcpy_s(fullFileName,MID_STRLEN,imageName);
#else
                        PathCat(fullFileName, m_imagesDir, imageName);
#endif
                        printf("\nLoading image:%s\nProcessing..", fullFileName);
                        tempImage = cvLoadImage(fullFileName);
                        m_Src = cvCreateImage(cvGetSize(tempImage), 8, 1);
                        m_SrcForCompleteImage = cvCreateImage(cvGetSize(tempImage), 8, 1);
        cvCvtColor(tempImage, m_Src, CV_BGR2GRAY);
                        cvCopy(m_Src, m_SrcForCompleteImage);
                        m_width             = m_Src->width;
                        m_height = m_Src->height;
                        gWidthPartProcessing = m_width / NUMBER_PARTS_WIDTH;
                        gHeightPartProcessing = m_height / NUMBER_PARTS_HEIGHT;
                        m_partImageRanningIndexWidth  = 0;
                        m_partImageRanningIndexHeight = 0;
                }
                else if(m_sourceInfo.m_sourceType == SourceInfo::FS_PARTS_RANGE)
                {
                        char fullFileName[MAX_STRLEN] = "\0";
                        IplImage* tempImage;
                        char imageName[MID_STRLEN] = "\0";
                        m_seqCounter = m_sourceInfo.m_seqStartIndex - 1;
                        char *token1, *next_token1;
                        token1 = strtok_s(m_sourceInfo.m_seqImageName, ".", &next_token1);
                        if (strcmp (next_token1, "") == 0 ) //Name  .jpg
                        {
                                strcpy_s(m_nameWithoutExt, MAX_STRLEN, "\0");
                                strcpy_s(m_extention, MAX_STRLEN, token1);
                        }
                        Else
                        {
                                strcpy_s(m_nameWithoutExt, MAX_STRLEN, token1);
                                strcpy_s(m_extention, MAX_STRLEN, next_token1);
                        }
                        // need to load one image to obtain values for dimentions
                        ObtainSeqRangeImageName(m_sourceInfo.m_seqStartIndex, fullFileName);
                                printf("Loading image:%s\nProcessing..", fullFileName);
        while ((tempImage = cvLoadImage(fullFileName)) == 0L)
                        {
                                m_sourceInfo.m_seqStartIndex++;
                                if (m_sourceInfo.m_seqStartIndex > m_sourceInfo.m_seqEndIndex)
                                {
                                        assert(0);
                                }
                                ObtainSeqRangeImageName(m_sourceInfo.m_seqStartIndex, fullFileName);
```

```
                        }
                        m_seqCounter = m_sourceInfo.m_seqStartIndex - 1;

                        if (m_sourceInfo.m_seqStartIndex == m_sourceInfo.m_seqEndIndex)
                        {
                                // treat it as FS_SINGLE; i.e., need to cache image in m_Src
                                m_Src = cvCreateImage(cvGetSize(tempImage), 8, 1);
                                cvCvtColor(tempImage, m_Src, CV_BGR2GRAY);
                                // and make sure it is treated as single from now on
                                m_sourceInfo.m_sourceType = SourceInfo::FS_SINGLE;
                        }
#ifdef RESIZE_FRAME_SIZE
                        m_width         = RESIZE_FRAME_SIZE_WIDTH;
                        m_height = RESIZE_FRAME_SIZE_HEIGHT;
#else
                        m_width         = tempImage->width;
                        m_height = tempImage->height;
#endif

                        m_seqCounter++;
                        m_Src = cvCreateImage(cvGetSize(tempImage), 8, 1);
                        m_SrcForCompleteImage = cvCreateImage(cvGetSize(tempImage), 8, 1);
                        cvCvtColor(tempImage, m_Src, CV_BGR2GRAY);
                        cvCopy(m_Src, m_SrcForCompleteImage);
                        m_width         = m_Src->width;
                        m_height = m_Src->height;
                        gWidthPartProcessing = m_width / NUMBER_PARTS_WIDTH;
                        gHeightPartProcessing = m_height / NUMBER_PARTS_HEIGHT;
                        m_partImageRanningIndexWidth  = 0;
                        m_partImageRanningIndexHeight = 0;
                }
        GetDims(outWidth, outHeight);
}
void FrameSrcImp::UpdateFrameSrc(void* outImg, void* WholeImage)
{
        if(SourceInfo::FS_PARTS_RANGE == m_sourceInfo.m_sourceType)
        {
                if(gPartsProcessingCompleted)
        {
                cvCopy(m_SrcForCompleteImage, (IplImage*)WholeImage);
        }
                CvRect rect = cvRect(m_partImageRanningIndexWidth, m_partImageRanningIndexHeight,
gWidthPartProcessing,gHeightPartProcessing);
                cvSetImageROI(m_Src, rect);
                gRectCurrentPartProcessing.height =  rect.height;
                gRectCurrentPartProcessing.width = rect.width;
                gRectCurrentPartProcessing.x =          rect.x;
                gRectCurrentPartProcessing.y =          rect.y;
                cvCopy(m_Src, (IplImage*)outImg);
                gPartsProcessingCompleted = false;
                if(m_partImageRanningIndexWidth < (m_width-(gWidthPartProcessing + (m_width %
NUMBER_PARTS_WIDTH))))
                {
                        m_partImageRanningIndexWidth = m_partImageRanningIndexWidth +
gWidthPartProcessing;
                        printf(".");
                }
                else
                {
                        m_partImageRanningIndexWidth = 0;
                        if(m_partImageRanningIndexHeight < (m_height - (gHeightPartProcessing + (m_height
% NUMBER_PARTS_HEIGHT))))
```

```
                                        {
                                                m_partImageRanningIndexHeight =
m_partImageRanningIndexHeight + gHeightPartProcessing;
                                                printf(".");
                                        }
                                else
                                        {
                                                //complete image
                                                gPartsProcessingCompleted = true;
                                                m_partImageRanningIndexHeight = 0;
                                                IplImage* tempImage;
                                                char fullFileName[MAX_STRLEN] = "\0";
                                                gFrameID = m_seqCounter;
                                                m_seqCounter = (m_seqCounter == m_sourceInfo.m_seqEndIndex)
?
                                                        (m_sourceInfo.m_seqStartIndex)
                                                        : (++m_seqCounter);
                                                ObtainSeqRangeImageName(m_seqCounter, fullFileName);
                                                if(m_sourceInfo.m_seqEndIndex > gFrameID)
                                                {
                                                printf("\nLoading image:%s\nProcessing..", fullFileName);
                                                }
                                                while ((tempImage = cvLoadImage(fullFileName)) == 0L)
                                                {
                                                        m_seqCounter = (m_seqCounter ==
m_sourceInfo.m_seqEndIndex) ?
                                                                (m_sourceInfo.m_seqStartIndex)
                                                                : (++m_seqCounter);
                                                        ObtainSeqRangeImageName(m_seqCounter,
fullFileName);
                                                }
                                                cvResetImageROI(m_Src);
                                                if((m_Src->height != tempImage->height) || (m_Src->width !=
tempImage->width))
                                                {
                                                        printf("sizes of images are different. should be same.\n");
                                                        assert(0);
                                                }
                                                //TODO implement if size of images is diffrent
                                                //must also change part processing
                                                /*cvReleaseImage(&m_Src);
                                                cvReleaseImage(&m_SrcForCompleteImage);
                                                m_Src = cvCreateImage(cvGetSize(tempImage), 8, 1);
                                                m_SrcForCompleteImage = cvCreateImage(cvGetSize(tempImage),
8, 1);
                                                */
                                                cvCvtColor(tempImage, m_Src, CV_BGR2GRAY);
                                                cvCopy(m_Src, m_SrcForCompleteImage);

                                                cvReleaseImage(&tempImage);
                                        }
                        }

        }
        else if (SourceInfo::FS_PARTS_SINGLE == m_sourceInfo.m_sourceType)
        {
                gFrameID =1 ;
                cvCopy(m_SrcForCompleteImage, (IplImage*)WholeImage);
                CvRect rect = cvRect(m_partImageRanningIndexWidth, m_partImageRanningIndexHeight,
gWidthPartProcessing,gHeightPartProcessing);
                cvSetImageROI(m_Src, rect);
                gRectCurrentPartProcessing.height =  rect.height;
```

```
                gRectCurrentPartProcessing.width = rect.width;
                gRectCurrentPartProcessing.x =          rect.x;
                gRectCurrentPartProcessing.y =          rect.y;
                cvCopy(m_Src, (IplImage*)outImg);
                gPartsProcessingCompleted = false;
                if(m_partImageRanningIndexWidth < (m_width-(gWidthPartProcessing + (m_width %
NUMBER_PARTS_WIDTH))))
                {
                        m_partImageRanningIndexWidth = m_partImageRanningIndexWidth +
gWidthPartProcessing;
                        printf(".");
                }
                else
                {
                        m_partImageRanningIndexWidth = 0;
                        if(m_partImageRanningIndexHeight < (m_height - (gHeightPartProcessing + (m_height
% NUMBER_PARTS_HEIGHT))))
                                {
                                        m_partImageRanningIndexHeight =
m_partImageRanningIndexHeight + gHeightPartProcessing;
                                        printf(".");
                                }
                        else
                                {
                                        //complete image
                                        gPartsProcessingCompleted = true;
                                        m_partImageRanningIndexHeight = 0;
                                }
                }
        }
        else if (SourceInfo::FS_SINGLE == m_sourceInfo.m_sourceType)
        {
                gFrameID =1 ;
                cvCopy(m_Src, (IplImage*)outImg);
        }
        else
        {
                if (SourceInfo::FS_STREAM == m_sourceInfo.m_sourceType
                                || SourceInfo::FS_VIDEOSTREAM == m_sourceInfo.m_sourceType)
                {
                        gFrameID++;
                        m_SrcRgbNoRelease = cvQueryFrame(m_Cpt);
                        cvCvtColor(m_SrcRgbNoRelease, (IplImage*)outImg, CV_BGR2GRAY);
                }
                else if (SourceInfo::FS_RANGE == m_sourceInfo.m_sourceType)
                {
                        IplImage* tempImage;
                        char fullFileName[MAX_STRLEN] = "\0";
                        m_seqCounter = (m_seqCounter == m_sourceInfo.m_seqEndIndex) ?
                                (m_sourceInfo.m_seqStartIndex)
                                : (++m_seqCounter);
                        gFrameID = m_seqCounter;

                        ObtainSeqRangeImageName(m_seqCounter, fullFileName);
                        printf("Loading image:%s\n", fullFileName);
                        while ((tempImage = cvLoadImage(fullFileName)) == 0L)
                        {
                                m_seqCounter = (m_seqCounter == m_sourceInfo.m_seqEndIndex) ?
                                        (m_sourceInfo.m_seqStartIndex)
                                        : (++m_seqCounter);
                                ObtainSeqRangeImageName(m_seqCounter, fullFileName);
                        }
```

```
#ifdef RESIZE_FRAME_SIZE

                                IplImage* tempImgForResize = cvCreateImage(cvGetSize(tempImage), 8, 1);
                                cvCvtColor(tempImage, tempImgForResize, CV_BGR2GRAY);
                                cvResize(tempImgForResize, outImg);
                                cvReleaseImage(&tempImgForResize);
#else
                                cvCvtColor(tempImage, (IplImage*)outImg, CV_BGR2GRAY);
#endif
                                cvReleaseImage(&tempImage);
                        }
                        else assert(0);
                }
}
void FrameSrcImp::EndFrameSrc(void)
{
   if(m_sourceInfo.m_sourceType == SourceInfo::FS_STREAM)
                {
                        cvReleaseCapture(&m_Cpt);
                }
                else
                {
                        cvReleaseImage(&m_Src);
                }
}
void FrameSrcImp::GetDims(int* width, int* height) const
{
                *width = m_width;
                *height = m_height;
}
void FrameSrcImp::ObtainSeqRangeImageName(int inSeqCntr, char* outFullName)
{
                assert(SourceInfo::FS_RANGE == m_sourceInfo.m_sourceType || SourceInfo::FS_PARTS_RANGE ==
m_sourceInfo.m_sourceType);
                assert(inSeqCntr <= m_sourceInfo.m_seqEndIndex);
                char imageName[MID_STRLEN] = "\0";
                sprintf_s(imageName, MID_STRLEN, "%s%d.%s", m_nameWithoutExt, inSeqCntr, m_extention);
                strcpy_s(outFullName, MAX_STRLEN, "\0");
#if USING_COMMAND_ARGMENTS
                                strcpy_s(outFullName,MID_STRLEN,imageName);
#else
                                PathCat(outFullName, m_imagesDir, imageName);
#endif
}
FREAMSRCIMP.h
#ifndef FREAMSRCIMP_H
#define FREAMSRCIMP_H
#include "FrameSrc.h"
#include "globals.h"
class FrameSrcImp : public FrameSrc
{
                private:
                                typedef                 FrameSrc                inherited;
                                IplImage*               m_Src;
                                IplImage*               m_SrcRgbNoRelease;
                                CvCapture*              m_Cpt;
//for partting image
                                IplImage*               m_SrcForCompleteImage;
                                int                             m_partImageRanningIndexWidth;
                                int                             m_partImageRanningIndexHeight;
```

```
		int			m_width, m_height;
		int			m_seqCounter;
		char		m_extention			[MAX_STRLEN];
		char		m_nameWithoutExt	[MAX_STRLEN];
	SourceInfo	m_sourceInfo;
		char		m_imagesDir			[MAX_STRLEN];
	private:
		void			ObtainSeqRangeImageName(int inSeqCntr, char* outFullName);
	public:
				FrameSrcImp		(void);
		virtual			~FrameSrcImp	(void);
		virtual	void	InitFrameSrc	(
												const char*
inImagesDir
												, SourceInfo*
sourceInfo
												, int* width
												, int* height);
		virtual	void	UpdateFrameSrc	(void* outImg, void* wholeImage = NULL);
		virtual	const void*	GetFrame(void) const			{return
m_SrcRgbNoRelease;};
		virtual	void	EndFrameSrc		(void);
		virtual void	GetDims			(int* width, int* height) const;
		virtual const	SourceInfo::SourceType	GetSrcType (void) const {return
m_sourceInfo.m_sourceType;};
};

#endif	//FREAMSRCIMP_H
```

Main.cpp

```
// Video Image PSNR and SSIM
#include <iostream> // for standard I/O
#include <string>   // for strings
#include <iomanip>  // for controlling float print precision
#include <sstream>  // string to number conversion
#include <opencv2/imgproc/imgproc.hpp>  // Gaussian Blur
#include <opencv2/core/core.hpp>        // Basic OpenCV structures (cv::Mat, Scalar)
#include <opencv2/highgui/highgui.hpp>  // OpenCV window I/O
#include "frameSrcImp.h"
typedef enum
{
	ARGMENT_TYPE_SINGLE_IMAGE = 1,
	ARGMENT_TYPE_RANGE_IMAGE = 2,
}enumArgmentType;
static int rangeStartIndex = 0;
static int rangeEndIndex = 0;
void Init(enumArgmentType argType = ARGMENT_TYPE_SINGLE_IMAGE, char *imagePath = "", char
*configPath = "", int rangeFrom = 0, int rangeTo = 0)
{
	//result file each run will empty file
	sprintf_s(gResultFileName, MAX_STRLEN, "%s%s", gSettingDir,"ResultFile");
	OpenNewFile(gResultFileName);
	SourceInfo sourceInfo;
	gFrameSrc		= new FrameSrcImp;
#if USING_COMMAND_ARGMENTS
	if (argType == ARGMENT_TYPE_SINGLE_IMAGE)
	{
		sourceInfo.Init(SourceInfo::FS_PARTS_SINGLE, imagePath);
	}else
	{
		char fullFileName[MAX_STRLEN] = "\0";
```

```
                char imageName[] = "\\.jpg";
                PathCat(fullFileName, imagePath, imageName);
                rangeStartIndex = rangeFrom;
                rangeEndIndex = rangeTo;
                sourceInfo.Init(SourceInfo::FS_PARTS_RANGE, fullFileName, rangeFrom, rangeTo);
        }
#else
#if USING_BACKGROUND
        //shold be same size of background
        sourceInfo.Init(SourceInfo::FS_SINGLE, "a.jpg");
#else
        //sourceInfo.Init(SourceInfo::FS_SINGLE, "DSC_9463.jpg");
        rangeStartIndex = 1;
        rangeEndIndex = 26;
        //sourceInfo.Init(SourceInfo::FS_RANGE, "bb.png", rangeStartIndex, rangeEndIndex);
        //sourceInfo.Init(SourceInfo::FS_SINGLE, "bb9.png");
        //sourceInfo.Init(SourceInfo::FS_PARTS_SINGLE, "a53.jpg");
        sourceInfo.Init(SourceInfo::FS_PARTS_RANGE, ".jpg", rangeStartIndex, rangeEndIndex);
        //sourceInfo.Init(SourceInfo::FS_RANGE, "DSC_00.jpg", 11, 50);
#endif
#endif
        PathCat(gImagesDir, gDataDir, gImagesRelDir);
        gFrameSrc->InitFrameSrc(gImagesDir, &sourceInfo, &gWidth, &gHeight);
        gSrcGray = cvCreateImage(cvSize(gWidth, gHeight), 8, 1);
        gDebugImg = cvCreateImage(cvSize(gWidth, gHeight + MAX_TEXT_LINES * TEXT_LINE_HEIGHT +
BORDER_SIZE + 20), 8, 1);
        cvSetImageROI(gDebugImg,cvGetImageROI(gSrcGray));
        if (gFrameSrc->GetSrcType() == SourceInfo::FS_PARTS_SINGLE || gFrameSrc->GetSrcType() ==
SourceInfo::FS_PARTS_RANGE)
        {
                gPartProcessing = cvCreateImage(cvSize(gWidthPartProcessing, gHeightPartProcessing), 8, 1);
        }
#if USING_BACKGROUND
        gTechInput = new TechInput();
        gTechInput->InitTech();
        gTechBackground = new TechBackground();
        gTechBackground->InitTech();
        gTechSubtract = new TechSubtract();
        gTechSubtract->InitTech();
        gTechMedianFilter = new TechMedianFilter();
        gTechMedianFilter->InitTech();
        gTechAdaptiveThreashold = new TechAdaptiveThreashold();
        gTechAdaptiveThreashold->InitTech();
#else
        gTechInput = new TechInput();
        gTechInput->InitTech();
        gTechMedianFilter = new TechMedianFilter();
        gTechMedianFilter->InitTech();
        gTechAdaptiveThreashold = new TechAdaptiveThreashold();
        gTechAdaptiveThreashold->InitTech();
        gTechMorphology = new TechMorphology();
        gTechMorphology->InitTech();
        gTechContours = new TechContours();
        gTechContours->InitTech();

#endif
/*
*/
}
void Run(void)
{
        IplImage* threshed;
```

```
		IplImage* dst;
		if (gFrameSrc->GetSrcType() == SourceInfo::FS_PARTS_SINGLE || gFrameSrc->GetSrcType() ==
SourceInfo::FS_PARTS_RANGE)
		{
		 threshed = cvCreateImage(cvGetSize(gPartProcessing), 8, 1);
		 dst = cvCreateImage(cvGetSize(gPartProcessing), 8, 1);
		}
		else
			{
				threshed = cvCreateImage(cvGetSize(gSrcGray), 8, 1);
				dst = cvCreateImage(cvGetSize(gSrcGray), 8, 1);
			}
#if USING_BACKGROUND
		IplImage* background = cvCreateImage(cvGetSize(gSrcGray), 8, 1);
#endif
while(1)
{
#if USING_BACKGROUND
		gFrameSrc->UpdateFrameSrc(gSrcGray);
		gTechInput->ImplementTech(gSrcGray);
		gTechBackground->ImplementTech(gSrcGray/*not used*/, background);
		gTechSubtract->ImplementTech(background, gSrcGray);
//		cvSaveImage("C:\\ImageProcessing\\Images\\a22.jpg",gSrcGray);
		gTechMedianFilter->ImplementTech(gSrcGray, threshed);
		gTechAdaptiveThreashold->ImplementTech(threshed, threshed);
#else
		if (gFrameSrc->GetSrcType() == SourceInfo::FS_PARTS_SINGLE || gFrameSrc->GetSrcType() ==
SourceInfo::FS_PARTS_RANGE)
		{
			gFrameSrc->UpdateFrameSrc(gPartProcessing, gSrcGray);
			gTechInput->ImplementTech(gSrcGray);
			gTechMedianFilter->ImplementTech(gPartProcessing, threshed);
			gTechAdaptiveThreashold->ImplementTech(threshed, threshed);
			gTechMorphology->ImplementTech(threshed, threshed);
			gTechContours->ImplementTech(threshed, dst);
			//while stop condition
		//if(gViisabiltyOfWindows == 0)
		{
		if(gFrameSrc->GetSrcType() == SourceInfo::FS_PARTS_RANGE && gPartsProcessingCompleted)
		{
			if(gFrameID == rangeEndIndex)
				{
					cvWaitKey(10);
					printf("\nComplete Range\n");
					break;
				}
		}
		else if(gFrameSrc->GetSrcType() == SourceInfo::FS_PARTS_SINGLE && gPartsProcessingCompleted)
		{
			cvWaitKey(10);
			printf("\nComplete Single Image\n");
			break;
		}
		}
		}
		else
		{
		//Get Frame
		gFrameSrc->UpdateFrameSrc(gSrcGray);
		gTechInput->ImplementTech(gSrcGray);
		gTechMedianFilter->ImplementTech(gSrcGray, threshed);
		gTechAdaptiveThreashold->ImplementTech(threshed, threshed);
```

```
		gTechMorphology->ImplementTech(threshed, threshed);
		gTechContours->ImplementTech(threshed, threshed);
		//while stop condition
		if(gViisabiltyOfWindows == 0)
		{
		if(gFrameSrc->GetSrcType() == SourceInfo::FS_RANGE)
		{
			if(gFrameID == rangeEndIndex)
		{
			printf("Complete Range\n");
			break;
		}
		}
		else if(gFrameSrc->GetSrcType() == SourceInfo::FS_SINGLE)
		{
			printf("Complete Single Image\n");
			break;
		}
		}
		}//end else SourceInfo::FS_PARTS_SINGLE
#endif
cvWaitKey(10);
}
cvWaitKey(0);
//getchar();
}
void CleanUp(void)
{
		if (gFrameSrc->GetSrcType() == SourceInfo::FS_PARTS_SINGLE || gFrameSrc->GetSrcType() ==
SourceInfo::FS_PARTS_RANGE)
		{
			cvReleaseImage(&gPartProcessing);
		}
#if USING_BACKGROUND
		gTechInput->EndTech();
		delete gTechInput;
		gTechBackground->EndTech();
		delete gTechBackground;
		gTechSubtract->EndTech();
		delete gTechSubtract;
		gTechMedianFilter->EndTech();
		delete gTechMedianFilter;
		gTechAdaptiveThreashold->EndTech();
		delete gTechAdaptiveThreashold;
		gFrameSrc->EndFrameSrc();
		delete gFrameSrc;
#else
		gTechInput->EndTech();
		delete gTechInput;
		gTechMedianFilter->EndTech();
		delete gTechMedianFilter;
		gTechAdaptiveThreashold->EndTech();
		delete gTechAdaptiveThreashold;
		gTechContours->EndTech();
		delete gTechContours;
		gFrameSrc->EndFrameSrc();
		delete gFrameSrc;
#endif
		cvReleaseImage(&gSrcGray);
		if (gDebugImg != 0L)
		{
			cvReleaseImage(&gDebugImg);
```

```
		}
}
int main(int argc, char** argv)
{
#if USING_COMMAND_ARGMENTS
		//printf( "argc = %d\n", argc );
   for( int i = 0; i < argc; ++i )
		{
    //  printf( "argv[ %d ] = %s\n", i, argv[ i ] );
   }
		int type = atoi(argv[ 1 ]);
if(type == ARGMENT_TYPE_SINGLE_IMAGE)
{
		gViisabiltyOfWindows = atoi(argv[ 3 ]);
		Init((enumArgmentType)type, argv[ 2 ]);
}else
{
			gViisabiltyOfWindows = atoi(argv[ 6 ]);
			int rangeFrom = atoi(argv[ 4 ]);
			int rangeTo = atoi(argv[ 5 ]);
			Init((enumArgmentType)type, argv[ 2 ], argv[ 3 ],rangeFrom,rangeTo);
}
#else
		Init();
#endif

		Run();
		CleanUp();
		return 0;
}
techAdaptiveThreashold.cpp
#include "techAdaptiveThreashold.h"
#include "globals.h"
int			TechAdaptiveThreashold::m_KernelSize			= 5;
int			TechAdaptiveThreashold::m_MeanOffset			= 5;
char	TechAdaptiveThreashold::m_FileName[MAX_STRLEN] = "\0";
TechAdaptiveThreashold::TechAdaptiveThreashold(void)
{
		strcpy_s(m_winName, MAX_STRLEN, "AdaptiveThreashold");
		sprintf_s(m_FileName, MAX_STRLEN, "%s%s", gSettingDir,"AdaptiveThreashold");
		ReadParmaeters(m_FileName, 2, &m_KernelSize, &m_MeanOffset);
}
TechAdaptiveThreashold::~TechAdaptiveThreashold(void)
{
		EndTech();
}
void TechAdaptiveThreashold::InitTech (const char* winName /* = 0L */)
{
if(gViisabiltyOfWindows)
{
		cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
		cvMoveWindow(m_winName, 340,20);
		cvCreateTrackbar( "MeanOffset", m_winName, &m_MeanOffset, 40, OntTackbarMeanOffset);
		cvCreateTrackbar( "KernelSize", m_winName, &m_KernelSize, 20, OntTackbarKernelSize);
		cvShowImage(m_winName,gSrcGray);
		//cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
		m_wholeImageProcessing = cvCreateImage(cvSize(gWidth, gHeight), 8, 1);
		cvSet(m_wholeImageProcessing,cvScalar(255));
		m_resizeImageLocally = cvCreateImage( cvSize((int)(INPUT_CONTOURS_LOCAL_IMAGE_WIDTH) ,
(int)(INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT) ), 8, 1 );
}
}
```

```
void TechAdaptiveThreashold::ImplementTech (
                        IplImage* src /* = NULL */
                        , IplImage* dst /* = NULL */
                        , const CvSeq* inSpecData /* = NULL */
                        , void** outSpecData /* = NULL */)
{
        cvAdaptiveThreshold(
                                src
                                , dst
                                , 255       //max
                                , CV_ADAPTIVE_THRESH_MEAN_C             //adaptive method
        //CV_ADAPTIVE_THRESH_GAUSSIAN_C
                                //, CV_ADAPTIVE_THRESH_GAUSSIAN_C
                                , CV_THRESH_BINARY_INV                                  //threshold type
                                , m_KernelSize      //3, 5, 7 ...
                                , m_MeanOffset);
if(gViisabiltyOfWindows)
{
#ifdef RESIZE_WINDOW
        cvSetImageROI(m_wholeImageProcessing, gRectCurrentPartProcessing);
        cvCopy(dst, (IplImage*)m_wholeImageProcessing);
        cvResetImageROI(m_wholeImageProcessing);
        ResizeImage(m_wholeImageProcessing, m_resizeImageLocally,
INPUT_CONTOURS_LOCAL_IMAGE_WIDTH,INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT);
        cvShowImage(m_winName,m_resizeImageLocally);
        //ResizeImage(dst, m_resizeImage);
        //cvShowImage(m_winName,m_resizeImage);
#else
        cvShowImage(m_winName,dst);
#endif
}
}
void TechAdaptiveThreashold::EndTech (void)
{
        cvReleaseImage(&m_wholeImageProcessing);
        cvReleaseImage(&m_resizeImageLocally);
        cvDestroyWindow(m_winName);
        Inherited::EndTech();
}
techAdaptiveThreashold.h
#ifndef TECH_ADAPTIVE_THREASHOLD
#define TECH_ADAPTIVE_THREASHOLD
#include "techBase.h"
class TechAdaptiveThreashold : public TechBase
{
        private:
                typedef         TechBaseInherited;
                IplImage* m_wholeImageProcessing;
                IplImage* m_resizeImageLocally;
        public:
        static int m_KernelSize;
        static int m_MeanOffset;
        static char m_FileName[MAX_STRLEN];
        public:                                                                 TechAdaptiveThreashold
        (void);
                virtual                 ~TechAdaptiveThreashold     (void);
                virtual void    InitTech                        (const char* winName = 0L);
                virtual void    ImplementTech
```

```
                                IplImage* src = NULL
                                , IplImage* dst = NULL
                                , const CvSeq* inSpecData = NULL
                                , void** outSpecData = NULL);
        virtual void                    EndTech                 (void);
        static  void                    OntTackbarKernelSize(int pos)
                                                                {
                                                                        if(pos <=3)
                                                                                m_KernelSize = 3;
                                                                        else if(pos%2 == 0)
                                                                                m_KernelSize  =
pos+1;
                                                                        else
                                                                                m_KernelSize  =
pos;
                                                                        m_KernelSize
=m_KernelSize+40;
        WriteParmaeters(m_FileName, 2, m_KernelSize, m_MeanOffset);
                                                                }
        static  void                    OntTackbarMeanOffset(int pos)
                                                                {
                                                                        m_MeanOffset  = pos;

        WriteParmaeters(m_FileName, 2, m_KernelSize, m_MeanOffset);
                                                                }
};

#endif //TECH_ADAPTIVE_THREASHOLD
techBackground.cpp
#include "techBackground.h"
#include "globals.h"
int             TechBackground::m_CountOfFrames         = 5;
int             TechBackground::m_Alfa          = 4;
int             TechBackground::m_doProcess             = 0;
char    TechBackground::m_FileName[MAX_STRLEN] = "\0";
TechBackground::TechBackground(void)
{
        strcpy_s(m_winName, MAX_STRLEN, "Background");
        sprintf_s(m_FileName, MAX_STRLEN, "%s%s", gSettingDir,"Background");
        ReadParmaeters(m_FileName, 3, &m_CountOfFrames, &m_Alfa, &m_doProcess);
}
TechBackground::~TechBackground(void)
{
        EndTech();
}
void TechBackground::InitTech (const char* winName /* = 0L */)
{
        if(gViisabiltyOfWindows)
{
        cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
        cvCreateTrackbar( "Count", m_winName, &m_CountOfFrames, 30, OntTackbarCountOfFrames);
        cvCreateTrackbar( "Alfa", m_winName, &m_Alfa, 20, OntTackbarAlfa);
        cvCreateTrackbar( "Run", m_winName, &m_doProcess, 1, OntTackbarDoProcess);
        cvMoveWindow(m_winName, 680,20);
        cvShowImage(m_winName,gSrcGray);
        //cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
        }
}
void TechBackground::ImplementTech (
                IplImage* src /* = NULL */
                , IplImage* dst /* = NULL */
                , const CvSeq* inSpecData /* = NULL */
```

```
                    , void** outSpecData /* = NULL */)
{
        char imageName[MAX_STRLEN] = "\0";
        char backgroundName[MAX_STRLEN] = "background";
        IplImage *imgBackground,*imgBackgroundTemp, *inputFrame,*inputFrameTemp;
        double alfa = (double)1/m_Alfa;
if(m_doProcess == 1)
{
        for (int i  = 1; i <= m_CountOfFrames; i++)
        {
                sprintf_s(imageName, MAX_STRLEN, "%s%d.%s", gImagesDirBackground, i, "jpg");
                if (i == 1)
                {
                        imgBackgroundTemp = cvLoadImage(imageName);
                        imgBackground = cvCreateImage( cvSize((int)(imgBackgroundTemp->width) ,
(int)(imgBackgroundTemp->height) ), 8, 1 );
                        cvCvtColor(imgBackgroundTemp, imgBackground, CV_BGR2GRAY);
                        cvReleaseImage(&imgBackgroundTemp);
                }else
                {
                        inputFrameTemp =  cvLoadImage(imageName);
                        inputFrame = cvCreateImage( cvSize((int)(inputFrameTemp->width) ,
(int)(inputFrameTemp->height) ), 8, 1 );
                        cvCvtColor(inputFrameTemp, inputFrame, CV_BGR2GRAY);
                        cvConvertScale(inputFrame,inputFrame,alfa);
                        cvConvertScale(imgBackground,imgBackground,(double)(1-alfa));
                        cvAdd(inputFrame,imgBackground,imgBackground);
                        cvReleaseImage(&inputFrame);
                        cvReleaseImage(&inputFrameTemp);
                }
        }
        sprintf_s(imageName, MAX_STRLEN, "%s%s.%s", gImagesDirBackground,backgroundName, "jpg");
        cvSaveImage(imageName,imgBackground);
}
else if(m_doProcess == 0)
{
        sprintf_s(imageName, MAX_STRLEN, "%s%s.%s", gImagesDirBackground,backgroundName, "jpg");
        imgBackgroundTemp = cvLoadImage(imageName);
        imgBackground = cvCreateImage( cvSize((int)(imgBackgroundTemp->width) , (int)(imgBackgroundTemp-
>height) ), 8, 1 );
        cvCvtColor(imgBackgroundTemp, imgBackground, CV_BGR2GRAY);
        cvReleaseImage(&imgBackgroundTemp);
}
        cvCopy(imgBackground, dst);
        cvReleaseImage(&imgBackground);
if(gViisabiltyOfWindows)
{
#ifdef RESIZE_WINDOW
        ResizeImage(dst, m_resizeImage);
        cvShowImage(m_winName,m_resizeImage);
#else
        cvShowImage(m_winName,dst);
#endif
}
}
void TechBackground::EndTech (void)
{
        Inherited::EndTech();
}
techBackground.h
#ifndef TECH_BACKGROUND
#define TECH_BACKGROUND
```

```
#include "techBase.h"
class TechBackground : public TechBase
{
	private:
		typedef		TechBaseInherited;
	public:
		static int m_CountOfFrames;
		static int m_Alfa;
		static int m_doProcess;
		static char m_FileName[MAX_STRLEN];
			TechBackground	(void);
		virtual		~TechBackground	(void);
		virtual void	InitTech		(const char* winName = 0L);
		virtual void	ImplementTech		(
			IplImage* src = NULL
			, IplImage* dst = NULL
			, const CvSeq* inSpecData = NULL
			, void** outSpecData = NULL);
		virtual void		EndTech			(void);
		static  void		OntTackbarCountOfFrames(int pos)
				{
					m_CountOfFrames  = pos;
WriteParmaeters(m_FileName, 3, m_CountOfFrames, m_Alfa,m_doProcess);
				}
		static  void		OntTackbarAlfa(int pos)
				{
					m_Alfa  = pos;
	WriteParmaeters(m_FileName, 3, m_CountOfFrames, m_Alfa,m_doProcess);
				}


		static  void		OntTackbarDoProcess(int pos)
				{
					m_doProcess  = pos;
WriteParmaeters(m_FileName, 3, m_CountOfFrames, m_Alfa,m_doProcess);
				}
};
#endif //TECH_BACKGROUND
techBase.cpp
#include "techBase.h"
void TechBase::InitTech(const char* winName /* = 0L */)
{
	if(winName == 0L)
	{
		cvNamedWindow(winName, CV_WINDOW_AUTOSIZE );
		strcpy_s(m_winName, MAX_STRLEN, winName);
	}
}
void TechBase::ResetTech	(void)
{
	assert(0);
}
void TechBase::EndTech		(void)
{
	cvDestroyWindow(m_winName);
}
techBase.h
#ifndef TECH_BASE_H
#define TECH_BASE_H
#include "config.h"
class TechBase
{	protected:
```

```
		char			m_winName[MAX_STRLEN];
	public:
		inline			TechBase		(void)
		{
			#ifdef RESIZE_WINDOW
				m_resizeImage = cvCreateImage( cvSize((int)(RESIZE_WIDTH) ,
(int)(RESIZE_HEIGHT) ), 8, 1 );
			#endif
		}
		virtual			~TechBase		(void) {}
		virtual void		InitTech		(const char* winName = 0L);

		virtual void		ImplementTech	(
			IplImage* src = NULL
                                                                                ,
IplImage* dst = NULL
                                                                                , const
CvSeq* inSpecData = NULL
                                                                                , void**
outSpecData = NULL) = 0;
		virtual void		ResetTech		(void);
		virtual void		EndTech			(void);
	public:
		inline	char*	WinName				(void) {return m_winName;};
		IplImage* m_resizeImage;
};
#endif //TECH_BASE_H
techContours.cpp
#include "techContours.h"
#include "globals.h"
int		TechContours::m_MinSize		= 5;
int		TechContours::m_MaxSize		= 5;
char	TechContours::m_FileName[MAX_STRLEN] = "\0";
#define LIMIT_AREA_CONTOUR_SIZE 500
#define DISTANCE_BETWEEN_CONTOURS_TO_CONNECT 30
#define CRACK_PERCENTAGE_WHOLE_IMAGE 0.05f
#define UNKNOWN_CRACKED_PERCENTAGE 4.00f
#define DRAW_RECTANGLE 0
double	findShortestDistance(CvRect rect1,CvRect rect2,int* outCorner1,int* outCorner2);
double	Distance(double dX0, double dY0, double dX1, double dY1);
void	DrawLineBetweenContoues(IplImage* img, CvRect rect1,CvRect rect2,int outCorner1,int outCorner2);
typedef enum CrackTypes
{
	CRACK_TYPE_EDGE = 0
	, CRACK_TYPE_BLOCK
	, CRACK_TYPE_LONGITUDE
	, CRACK_TYPE_TRANSVERSE
	, CRACK_TYPE_FATIQUE
	, CRACK_TYPE_POTHOLES
	, CRACK_TYPE_COUNT
	, CRACK_TYPE_NONE
};
char *TypeName[] = {"Edge", "Block", "Longitude", "Transverse", "Fatigue", "Potholes"};
typedef enum SeverityLevel
{
	SEVERITY_LOW = 0
	, SEVERITY_MODERATE
	,SEVERITY_HIGH
};
char *SeverityName[] = {"Low", "Moderate", "High"};
#define WIDTH_METER 4
#define MAX_NUMBER_PARTION_OF_LOGITUDE 4
```

```
typedef struct LongitudinalCrack
{
	CvPoint pointStart;
	CvPoint pointEnd;
	int count;
	float areaSamePartition;
};
#define MAX_NUMBER_PARTION_OF_TRANSVERSE 4
typedef struct TransverseCrack
{
	CvPoint pointStart;
	CvPoint pointEnd;
	int count;
	float areaSamePartition;
};
typedef struct Crack
{
	CrackTypes type;
	SeverityLevel severity;
	float area;
	LongitudinalCrack LongCracks[MAX_NUMBER_PARTION_OF_LOGITUDE];
	TransverseCrack TransCracks[MAX_NUMBER_PARTION_OF_TRANSVERSE];
	int countOfCracks;
	float percentage;
};
Crack Cracks[CRACK_TYPE_COUNT];
#define MAX_NUMBER_CRACK_FOR_POTHOLES 1000
int counterofCrackedForPhotholes = 0;
typedef struct CrackPotholes
{
	CvRect rect;
	float area;
};
CrackPotholes Potholes[MAX_NUMBER_CRACK_FOR_POTHOLES];
bool CalculateCrackPotholes(float totalCrackedArea);
void FindCrackType(IplImage* img, SeverityLevel* severity, CrackTypes* type,float totalCracedArea);
void	DetectType(IplImage* img,CvRect rect, float area);
CrackTypes GetCrackType(float totalCracedArea, SeverityLevel* severity);
void initCrackTypes(IplImage* img);
int GetLongitudeIndex(CvRect rect);
int GetTransverseIndex(CvRect rect);
void SetPercentagesCrackTypes(float totalCracedArea);
TechContours::TechContours(void)
{
	strcpy_s(m_winName, MAX_STRLEN, "Contour");
	sprintf_s(m_FileName, MAX_STRLEN, "%s%s", gSettingDir,"Contours");
	ReadParmaeters(m_FileName, 2, &m_MinSize, &m_MaxSize);
	m_CountOfContours = 0;
	m_totalCracedArea = 0.f;
	m_CrackedPercentage = CRACK_PERCENTAGE_WHOLE_IMAGE;
	m_unknownCrackedPercentage = UNKNOWN_CRACKED_PERCENTAGE;
	AppendHeaderToFile(gResultFileName);
}
TechContours::~TechContours(void)
{
	EndTech();
}
void TechContours::InitTech (const char* winName /* = 0L */)
{
	if(gViisabiltyOfWindows)
{
	cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
```

```
		cvMoveWindow(m_winName, 1020,300);
		cvCreateTrackbar( "Max size", m_winName, &m_MaxSize, 20, OntTackbarMeanOffset);
		cvCreateTrackbar( "Min size", m_winName, &m_MinSize, 10, OntTackbarKernelSize);
		cvShowImage(m_winName,gSrcGray);
		//cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
		m_resizeImageLocally = cvCreateImage( cvSize((int)(INPUT_CONTOURS_LOCAL_IMAGE_WIDTH) ,
(int)(INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT) ), 8, 1 );
}
		m_wholeImageProcessing = cvCreateImage(cvSize(gWidth, gHeight), 8, 1);
		cvSet(m_wholeImageProcessing,cvScalar(255));
		m_MinSize = m_MinSize * MIN_TRACKER_COFIEIENT;
		m_MaxSize = m_MaxSize * MAX_TRACKER_COFIEIENT;
		initCrackTypes(m_wholeImageProcessing);
}
void TechContours::ImplementTech (
			IplImage* src /* = NULL */
			, IplImage* dst /* = NULL */
			, const CvSeq* inSpecData /* = NULL */
			, void** outSpecData /* = NULL */)
{
		CvMemStorage *mem;
		mem = cvCreateMemStorage(0);
		CvSeq *contours = 0;
		CvScalar(ext_color);
		int n = cvFindContours(src, mem, &contours, sizeof(CvContour), CV_RETR_CCOMP,
CV_CHAIN_APPROX_NONE, cvPoint(0,0));
		cvZero(dst);
		ext_color = CV_RGB( 255, 255, 255 );
		float area = 0.f;
		int countOfContours = 0;
		CvRect rect ;
		const int totalContors =n;
		CvRect *rectList= new CvRect[totalContors];
		for (; contours != 0; contours = contours->h_next)
		{
			area = fabsf((float)cvContourArea(contours /*const CvArr* points*/, CV_WHOLE_SEQ));
			if(m_MaxSize == 20 * MAX_TRACKER_COFIEIENT)
			{
				m_MaxSize = 0x7fffffff;
			}
			if(area > m_MinSize && area < m_MaxSize)
			{
				rect = cvBoundingRect(contours);
				cvDrawContours(dst, contours, ext_color, CV_RGB(0,0,0), -1, CV_FILLED, 8,
cvPoint(0,0));
#if DRAW_RECTANGLE
				cvRectangle(dst,
			cvPoint(rect.x, rect.y),
			cvPoint(rect.x+rect.width, rect.y+rect.height),
			cvScalar(255,255, 255, 0),
			1, 8, 0);
#endif
				rectList[countOfContours].height = rect.height;
				rectList[countOfContours].width = rect.width;
				rectList[countOfContours].x = rect.x;
				rectList[countOfContours].y = rect.y;
				countOfContours++;
			}
		}
// mantainance image by completeing lines
		double distancelongLimit = DISTANCE_BETWEEN_CONTOURS_TO_CONNECT;
		int outCorner1 = 0;
```

```
		int outCorner2 = 0;
		double shortestDis = 9999999;
		double shortestDisTemp = 9999999;
		int iTemp =0;
		int jTemp = 0;
		int outTmep1 = 0;
		int outTemp2 = 0;
		for(int i =0 ;i< countOfContours;i++)
		{
			shortestDisTemp = 9999999;
			shortestDis = 9999999;
			for (int j = i+1;j < countOfContours;j++)
			{
				shortestDisTemp = findShortestDistance(rectList[i],
rectList[j],&outCorner1,&outCorner2);
				if(shortestDisTemp < shortestDis)
				{
					shortestDis = shortestDisTemp;
					iTemp = i;
					jTemp = j;
					outTmep1 = outCorner1;
					outTemp2 = outCorner2;
				}
			}
			if (shortestDis <= distancelongLimit)
			{
		DrawLineBetweenContoues(dst,rectList[iTemp],rectList[jTemp],outTmep1,outTemp2);
				//complete two linestogther
			}
		}
		delete[] rectList;
//proceessing after connect crackes
		IplImage* tempImage = cvCreateImage(cvGetSize(dst), 8, 1);
		cvZero(tempImage);
		n = cvFindContours(dst, mem, &contours, sizeof(CvContour), CV_RETR_CCOMP,
CV_CHAIN_APPROX_NONE, cvPoint(0,0));
		for (; contours != 0; contours = contours->h_next)
		{
			rect = cvBoundingRect(contours);
			area = fabsf((float)cvContourArea(contours , CV_WHOLE_SEQ));
			if(area > LIMIT_AREA_CONTOUR_SIZE)
			{
				m_CountOfContours++;
				m_totalCracedArea = m_totalCracedArea + area ;
				cvDrawContours(tempImage, contours, ext_color, CV_RGB(0,0,0), -1, CV_FILLED, 8,
cvPoint(0,0));
				DetectType(m_wholeImageProcessing, rect,area);
			}

		}

//cvShowImage(m_winName,tempImage);
//#ifdef RESIZE_WINDOW
		//ResizeImage(gDebugImg, m_resizeImage);
		//cvShowImage(m_winName,m_resizeImage);
		//cvShowImage(m_winName,gDebugImg);
#if 1
		//cvNamedWindow("processing", CV_WINDOW_AUTOSIZE );
		//cvShowImage("processing", dst);
			cvSetImageROI(m_wholeImageProcessing, gRectCurrentPartProcessing);
			cvCopy(tempImage, (IplImage*)m_wholeImageProcessing);
			cvResetImageROI(m_wholeImageProcessing);
```

```
                        if(gViisabiltyOfWindows)
                        {
                                ResizeImage(m_wholeImageProcessing, m_resizeImageLocally,
INPUT_CONTOURS_LOCAL_IMAGE_WIDTH,INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT);
                                cvShowImage(m_winName,m_resizeImageLocally);
                        }
if(gPartsProcessingCompleted)
        {
        /*      cvDilate(m_wholeImageProcessing, m_wholeImageProcessing,NULL,10);
                cvNamedWindow("processing", CV_WINDOW_AUTOSIZE );
                ResizeImage(m_wholeImageProcessing, m_resizeImage);
                cvShowImage("processing", m_resizeImage);*/
                float totalImageSize = m_wholeImageProcessing->height*m_wholeImageProcessing->width;
                float percentageOfCreackedArea = ((float)m_totalCracedArea/(totalImageSize))*100;
                CrackTypes type;
                SeverityLevel severity;
                type = GetCrackType(m_totalCracedArea, &severity);
                //FindCrackType(m_wholeImageProcessing, &severity, &type, m_totalCracedArea );
        //printf( "CC: %i, TA = %.02f, TCA: %.02f, per = %.02f limitCA =
%.02f\n",m_CountOfContours,totalImageSize, m_totalCracedArea, percentageOfCreackedArea,
m_CrackedPercentage);
        if(percentageOfCreackedArea >m_unknownCrackedPercentage)
        {
                AppendToFile(gResultFileName,gFrameID, 3,m_CountOfContours,totalImageSize,
m_totalCracedArea, percentageOfCreackedArea, m_totalCracedArea, "Unknown", "");
        }
        else if(percentageOfCreackedArea > m_CrackedPercentage)
        {
                AppendToFile(gResultFileName,gFrameID, 1,m_CountOfContours,totalImageSize,
m_totalCracedArea, percentageOfCreackedArea, m_totalCracedArea, TypeName[type], SeverityName[severity]);
        }
        else
        {
                AppendToFile(gResultFileName,gFrameID, 2,m_CountOfContours,totalImageSize,
m_totalCracedArea, percentageOfCreackedArea, m_totalCracedArea, "Non Crack", "Non Crack");
        }
        initCrackTypes(m_wholeImageProcessing);
                if(gViisabiltyOfWindows)
                {
                        ResizeImage(m_wholeImageProcessing, m_resizeImageLocally,
INPUT_CONTOURS_LOCAL_IMAGE_WIDTH,INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT);
                        PutTextInternal(m_resizeImageLocally,
m_resizeImageLocally,cvPoint(20,20),cvScalarAll(255), "CC:%i, TA = %.02f, TCA:%.02f, per
=%.02f",m_CountOfContours,totalImageSize, m_totalCracedArea, percentageOfCreackedArea);
                        if(percentageOfCreackedArea >m_unknownCrackedPercentage)
                        {
                                PutTextInternal(m_resizeImageLocally,
m_resizeImageLocally,cvPoint(20,40),cvScalarAll(255), "Unknown");
                        }
                        else if(percentageOfCreackedArea > m_CrackedPercentage)
                        {
                                PutTextInternal(m_resizeImageLocally,
m_resizeImageLocally,cvPoint(20,40),cvScalarAll(255), "Type: %s",TypeName[type]);
                        }
                        else
                        {
                                PutTextInternal(m_resizeImageLocally,
m_resizeImageLocally,cvPoint(20,40),cvScalarAll(255), "Non Crack");
                        }
                        cvShowImage(m_winName,m_resizeImageLocally);
                //cvWaitKey(0);
                        cvSet(m_wholeImageProcessing,cvScalar(255));
```

```
                }
                m_CountOfContours = 0;
                m_totalCracedArea = 0.f;
        }

#endif
//#else
//        cvShowImage(m_winName,gDebugImg);

//#endif
        cvReleaseImage(&tempImage);
}
void TechContours::EndTech (void)
{
        cvReleaseImage(&m_wholeImageProcessing);
        cvReleaseImage(&m_resizeImageLocally);
        Inherited::EndTech();
}
double Distance(double dX0, double dY0, double dX1, double dY1)
{
   return sqrt((dX1 - dX0)*(dX1 - dX0) + (dY1 - dY0)*(dY1 - dY0));
}
double findShortestDistance(CvRect rect1,CvRect rect2,int* outCorner1,int* outCorner2)
{
        /* corners
        1   2
        3   4
        */
        int x1,x2,x3,x4;
        int y1,y2,y3,y4;
        x1 = rect1.x;
        x2 = rect1.x + rect1.width;
        y1 = rect1.y;
        y2 = rect1.y + rect1.height;
        x3 = rect2.x;
        x4 = rect2.x + rect2.width;
        y3 = rect2.y;
        y4 = rect2.y + rect2.height;
        double shortestDistance = 9999999;
        double currentDistance = 0;
        if( shortestDistance > (currentDistance =  Distance(x1,y1,x3,y3)))
        {
                shortestDistance = currentDistance;
                *outCorner1 = 1;
                *outCorner2 = 1;
        }
        if( shortestDistance > (currentDistance =  Distance(x1,y1,x4,y3)))
        {
                shortestDistance = currentDistance;
                *outCorner1 = 1;
                *outCorner2 = 2;
        }
        if( shortestDistance > (currentDistance =  Distance(x1,y1,x3,y4)))
        {
                shortestDistance = currentDistance;
                *outCorner1 = 1;
                *outCorner2 = 3;
        }
        if( shortestDistance > (currentDistance =  Distance(x1,y1,x4,y4)))
        {
                shortestDistance = currentDistance;
                *outCorner1 = 1;
```

```
            *outCorner2 = 4;
}

if( shortestDistance > (currentDistance =  Distance(x2,y1,x3,y3)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 2;
            *outCorner2 = 1;
}
if( shortestDistance > (currentDistance =  Distance(x2,y1,x4,y3)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 2;
            *outCorner2 = 2;
}
if( shortestDistance > (currentDistance =  Distance(x2,y1,x3,y4)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 2;
            *outCorner2 = 3;
}
if( shortestDistance >(currentDistance =  Distance(x2,y1,x4,y4)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 2;
            *outCorner2 = 4;
}


if( shortestDistance > (currentDistance =  Distance(x1,y2,x3,y3)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 3;
            *outCorner2 = 1;
}
if( shortestDistance >( currentDistance =  Distance(x1,y2,x4,y3)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 3;
            *outCorner2 = 2;
}
if( shortestDistance >( currentDistance =  Distance(x1,y2,x3,y4)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 3;
            *outCorner2 = 3;
}
if( shortestDistance > (currentDistance =  Distance(x1,y2,x4,y4)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 3;
            *outCorner2 = 4;
}


if( shortestDistance > (currentDistance =  Distance(x2,y2,x3,y3)))
{
            shortestDistance = currentDistance;
            *outCorner1 = 4;
            *outCorner2 = 1;
}
if( shortestDistance > (currentDistance =  Distance(x2,y2,x4,y3)))
```

```
		{
			shortestDistance = currentDistance;
			*outCorner1 = 4;
			*outCorner2 = 2;
		}
		if( shortestDistance > (currentDistance =  Distance(x2,y2,x3,y4)))
		{
			shortestDistance = currentDistance;
			*outCorner1 = 4;
			*outCorner2 = 3;
		}
		if( shortestDistance > (currentDistance =  Distance(x2,y2,x4,y4)))
		{
			shortestDistance = currentDistance;
			*outCorner1 = 4;
			*outCorner2 = 4;
		}
return shortestDistance;
}
void   DrawLineBetweenContoues(IplImage* img, CvRect rect1,CvRect rect2,int outCorner1,int outCorner2)
{
		int x1,x2,x3,x4;
		int y1,y2,y3,y4;
		x1 = rect1.x;
		x2 = rect1.x + rect1.width;
		y1 = rect1.y;
		y2 = rect1.y + rect1.height;
		x3 = rect2.x;
		x4 = rect2.x + rect2.width;
		y3 = rect2.y;
		y4 = rect2.y + rect2.height;
		//detect if crack extent on  y or x
		int extenOnFirstContur = abs(x2-x1) > abs(y2-y1) ? 1 : 2;
		int extenOnSecondContur = abs(x4-x3) > abs(y4-y3) ? 1 : 2;
		CvScalar s;
		CvPoint firstPoint, secondPoint;
				/*cvRectangle(img,
			cvPoint(rect1.x, rect1.y),
			cvPoint(rect1.x+rect1.width, rect1.y+rect1.height),
			cvScalar(255,255, 255, 0),
			1, 8, 0);   */
// First contour
		if(extenOnFirstContur == 2)//y extend on y check x to find first bixel
		{
				if(outCorner1 == 1 || outCorner1 == 2)
				{
					for (int i = x1; i <=x2 ;i++)
					{
						s=cvGet2D(img,y1+1,i); // get the (i,j) pixel value
						if(s.val[0] == 255)
						{
							firstPoint.x = i;
							firstPoint.y = y1+1;
							break;
						}
					}
				}
				else if(outCorner1 == 3 || outCorner1 == 4)//x
				{
					for (int i = x1;i <=x2 ;i++)
					{
						s=cvGet2D(img,y2-1,i); // get the (i,j) pixel value
```

```
                                        if(s.val[0] == 255)
                                                {
                                                        firstPoint.x = i;
                                                        firstPoint.y = y2-1;
                                                        break;
                                                }
                                        }
                                }
                        }else if(extenOnFirstContur ==1)
                        {
                                if(outCorner1 == 1 || outCorner1 == 3)
                                        {
                                                for (int i = y1; i <=y2 ;i++)
                                                {
                                                        s=cvGet2D(img,i,x1+1); // get the (i,j) pixel value
                                                        if(s.val[0] == 255)
                                                        {
                                                                firstPoint.x = x1+1;
                                                                firstPoint.y = i;
                                                                break;
                                                        }
                                                }
                                        }
                                else if(outCorner1 == 2 || outCorner1 == 4)
                                        {
                                                for (int i = y1; i <=y2 ;i++)
                                                {
                                                        s=cvGet2D(img,i,x2-1); // get the (i,j) pixel value
                                                        if(s.val[0] == 255)
                                                        {
                                                                firstPoint.x = x2-1;
                                                                firstPoint.y = i;
                                                                break;
                                                        }
                                                }
                                        }
                        }
//second contor
                        if(extenOnSecondContur == 2)//y extend on y check x to find first bixel
                        {
                                if(outCorner2 == 1 || outCorner2 == 2)
                                {
                                        for (int i = x3; i <=x4 ;i++)
                                        {
                                                s=cvGet2D(img,y3+1,i); // get the (i,j) pixel value
                                                if(s.val[0] == 255)
                                                {
                                                        secondPoint.x = i;
                                                        secondPoint.y = y3+1;
                                                        break;
                                                }
                                        }
                                }
                                else if(outCorner2 == 3 || outCorner2 == 4)//x
                                {
                                        for (int i = x3;i <=x4 ;i++)
                                        {
                                                s=cvGet2D(img,y4-1,i); // get the (i,j) pixel value
                                                if(s.val[0] == 255)
                                                        {
                                                                secondPoint.x = i;
                                                                secondPoint.y = y4-1;
```

```
                                                                        break;
                                                                }
                                                        }
                                                }
                                        }else if(extenOnSecondContur ==1)
                                        {
                                                if(outCorner2 == 1 || outCorner2 == 3)
                                                        {
                                                                for (int i = y3; i <=y4 ;i++)
                                                                {
                                                                        s=cvGet2D(img,i,x3+1); // get the (i,j) pixel value
                                                                        if(s.val[0] == 255)
                                                                        {
                                                                                secondPoint.x = x3+1;
                                                                                secondPoint.y = i;
                                                                                break;
                                                                        }
                                                                }
                                                        }
                                                else if(outCorner2 == 2 || outCorner2 == 4)
                                                        {
                                                                for (int i = y3; i <=y4 ;i++)
                                                                {
                                                                        s=cvGet2D(img,i,x4-1); // get the (i,j) pixel value
                                                                        if(s.val[0] == 255)
                                                                        {
                                                                                secondPoint.x = x4-1;
                                                                                secondPoint.y = i;
                                                                                break;
                                                                        }
                                                                }
                                                        }
                                        }

                                        CvScalar red = CV_RGB(255,255,255);
                                        cvLine(img,firstPoint,secondPoint,red,2,8);
                                        /*cvNamedWindow("processing", CV_WINDOW_AUTOSIZE );
                                        cvShowImage("processing", img);*/
//
}
void FindCrackType(IplImage* img, SeverityLevel* severity, CrackTypes* type, float totalCracedArea)
{
                                        CvMemStorage *mem;
                                        mem = cvCreateMemStorage(0);
                                        CvSeq *contours = 0;
                                        CvRect rect ;
                                        CvScalar(ext_color);
                                        float area = 0.f;
                                        initCrackTypes(img);
                                        int n = cvFindContours(img, mem, &contours, sizeof(CvContour), CV_RETR_CCOMP,
CV_CHAIN_APPROX_NONE, cvPoint(0,0));
                                        for (; contours != 0; contours = contours->h_next)
                                        {
                                                rect = cvBoundingRect(contours);
                                                area = fabsf((float)cvContourArea(contours , CV_WHOLE_SEQ));
                                                DetectType(img, rect,area);
                                        }
                                        *type = GetCrackType(totalCracedArea, severity);
}
void    DetectType(IplImage* img,CvRect rect, float area)
{
                                        rect.x += gRectCurrentPartProcessing.x;
```

```
rect.y += gRectCurrentPartProcessing.y;
Potholes[counterofCrackedForPhotholes].area = area;
Potholes[counterofCrackedForPhotholes].rect.x = rect.x;
Potholes[counterofCrackedForPhotholes].rect.y = rect.y;
counterofCrackedForPhotholes++;
float halfMeter = (float)(0.5 * img->width) / WIDTH_METER;
float areaworking =(float)img->width -  2 * halfMeter;
float thiredWorking = areaworking/3;
float potholesPixelsRange = 50;
//edge
if( (rect.x + rect.width < halfMeter) || ((img->width - halfMeter) < rect.x ))
{
	Cracks[CRACK_TYPE_EDGE].area += area;
	Cracks[CRACK_TYPE_EDGE].countOfCracks++;
}
else
{
	if(abs(rect.height - rect.width) < potholesPixelsRange )
	{
		Cracks[CRACK_TYPE_POTHOLES].area += area;
		Cracks[CRACK_TYPE_POTHOLES].countOfCracks++;
		//fatique
	if ((rect.x < thiredWorking+ halfMeter && rect.x > halfMeter)
		|| (rect.x > (2 *thiredWorking + halfMeter) && rect.x < 3 * thiredWorking))
	{
		Cracks[CRACK_TYPE_FATIQUE].area += area;
		Cracks[CRACK_TYPE_FATIQUE].countOfCracks++;
	}
	if(rect.height > 2 * rect.width)
	{
Cracks[CRACK_TYPE_LONGITUDE].LongCracks[GetLongitudeIndex(rect)].count++;

Cracks[CRACK_TYPE_LONGITUDE].LongCracks[GetLongitudeIndex(rect)].areaSamePartition +=area;
		Cracks[CRACK_TYPE_LONGITUDE].countOfCracks++;
		Cracks[CRACK_TYPE_LONGITUDE].area+=area;
	}
	else if(rect.height *2 <  rect.width)//transverse
	{
Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[GetTransverseIndex(rect)].count++;
Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[GetTransverseIndex(rect)].areaSamePartition +=area;
		Cracks[CRACK_TYPE_TRANSVERSE].countOfCracks++;
		Cracks[CRACK_TYPE_TRANSVERSE].area+=area;
	}

	}
	else
	{
	//block
	Cracks[CRACK_TYPE_BLOCK].area += area;
	Cracks[CRACK_TYPE_BLOCK].countOfCracks++;
	//fatique
	if ((rect.x < thiredWorking+ halfMeter && rect.x > halfMeter)
		|| (rect.x > (2 *thiredWorking + halfMeter) && rect.x < 3 * thiredWorking))
	{
		Cracks[CRACK_TYPE_FATIQUE].area += area;
		Cracks[CRACK_TYPE_FATIQUE].countOfCracks++;
	}
	//longutitde
	if(rect.height > 2 * rect.width)
	{
		Cracks[CRACK_TYPE_LONGITUDE].LongCracks[GetLongitudeIndex(rect)].count++;
Cracks[CRACK_TYPE_LONGITUDE].LongCracks[GetLongitudeIndex(rect)].areaSamePartition +=area;
```

```
                        Cracks[CRACK_TYPE_LONGITUDE].countOfCracks++;
                        Cracks[CRACK_TYPE_LONGITUDE].area+=area;
                }
                else if(rect.height *2 <  rect.width)//transverse
                {
        Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[GetTransverseIndex(rect)].count++;
        Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[GetTransverseIndex(rect)].areaSamePartition +=area;
                        Cracks[CRACK_TYPE_TRANSVERSE].countOfCracks++;
                        Cracks[CRACK_TYPE_TRANSVERSE].area+=area;
                }
                }
        }
}
CrackTypes GetCrackType(float totalCracedArea, SeverityLevel* severity)
{
        CrackTypes result = CRACK_TYPE_NONE;
        SetPercentagesCrackTypes(totalCracedArea);
        int initialPercentage = 50;
        int initialPercentageTreferse = 30;
        if (CalculateCrackPotholes(totalCracedArea))
        {
                result = CRACK_TYPE_POTHOLES;
                *severity = SEVERITY_HIGH;
        }
        else if(Cracks[CRACK_TYPE_EDGE].percentage > initialPercentage )
        {
                result = CRACK_TYPE_EDGE;
                *severity = SEVERITY_HIGH;
        }
        /*else if (Cracks[CRACK_TYPE_POTHOLES].percentage > initialPercentage)
        {
                result = CRACK_TYPE_POTHOLES;
                *severity = SEVERITY_HIGH;
        }*/
        else if (Cracks[CRACK_TYPE_FATIQUE].percentage > 50 &&
Cracks[CRACK_TYPE_LONGITUDE].percentage < 55 && Cracks[CRACK_TYPE_TRANSVERSE].percentage
<55 && Cracks[CRACK_TYPE_BLOCK].percentage < 74)
        {
                result = CRACK_TYPE_FATIQUE;
                *severity = SEVERITY_HIGH;
        }
        else  if (Cracks[CRACK_TYPE_LONGITUDE].percentage > 70)
        {
                result = CRACK_TYPE_LONGITUDE;
                *severity = SEVERITY_HIGH;
        }
        else  if (Cracks[CRACK_TYPE_TRANSVERSE].percentage > initialPercentageTreferse )
        {
                result = CRACK_TYPE_TRANSVERSE;
                if (Cracks[CRACK_TYPE_TRANSVERSE].percentage > 70)
                {
                        *severity = SEVERITY_HIGH;
                }else if (Cracks[CRACK_TYPE_TRANSVERSE].percentage > 65)
                {
                        *severity = SEVERITY_MODERATE;
                }
                else
                {
                        *severity = SEVERITY_LOW;
                }
        }
        else   if (Cracks[CRACK_TYPE_BLOCK].percentage > 10)
```

```
		{
			result = CRACK_TYPE_BLOCK;
			*severity = SEVERITY_HIGH;
		}

		return result;
}
void SetPercentagesCrackTypes(float totalCracedArea)
{
		for(int i =0 ; i < CRACK_TYPE_COUNT ;i++)
		{
			Cracks[i].percentage = (float)(Cracks[i].area * 100)/totalCracedArea;
		}
}
int GetTransverseIndex(CvRect rect)
{
		for(int i =0; i < MAX_NUMBER_PARTION_OF_TRANSVERSE;i++)
		{
			if(rect.y > Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].pointStart.y
				&& rect.y < Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].pointEnd.y)
			{
				return i;
			}
		}
return 0;
}
int GetLongitudeIndex(CvRect rect)
{
		for(int i =0; i < MAX_NUMBER_PARTION_OF_LOGITUDE;i++)
		{
			if(rect.x > Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].pointStart.x
				&& rect.x < Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].pointEnd.x)
			{
				return i;
			}
		}
return 0;
}
void initCrackTypes(IplImage* img)
{
		for(int i =0 ; i < CRACK_TYPE_COUNT ;i++)
		{
			Cracks[i].type = (CrackTypes)i;
			Cracks[i].area = 0;
			Cracks[i].percentage =0;
		}
		float halfMeter = (float)(0.5 * img->width) / WIDTH_METER;
		float areaworking =(float)img->width -  2 * halfMeter;
		float partionlong =  (float)areaworking / MAX_NUMBER_PARTION_OF_LOGITUDE;
		for(int i =0; i < MAX_NUMBER_PARTION_OF_LOGITUDE;i++)
			{
				Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].count = 0;
				Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].areaSamePartition = 0;
				Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].pointStart.x = halfMeter + i *
partionlong;
				Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].pointStart.y = 0;
				Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].pointEnd.x = halfMeter + (i+1) *
(partionlong);
				Cracks[CRACK_TYPE_LONGITUDE].LongCracks[i].pointEnd.y = 0;
			}
		float partionTrans=  (float)img->height / MAX_NUMBER_PARTION_OF_TRANSVERSE;
		for(int i =0; i < MAX_NUMBER_PARTION_OF_TRANSVERSE;i++)
```

```
		{
			Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].count = 0;
	Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].areaSamePartition = 0;
			Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].pointStart.x = 0;
			Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].pointStart.y = i*partionTrans;
			Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].pointEnd.x =0;
			Cracks[CRACK_TYPE_TRANSVERSE].TransCracks[i].pointEnd.y =
(i+1)*partionTrans;
		}
}
bool CalculateCrackPotholes(float totalCrackedArea)
{
	float maxlimitdistancebetweenCrackedSamePhothole = 100;
	float areaAlowedToBePhotholeType = (totalCrackedArea * 17) / 100;//
	float diffX = 0;
	float diffy = 0;
	float dis  = 0;
	float areaForSamePhothel  = 0;
	for (int i =0 ; i< counterofCrackedForPhotholes ; i++)
	{
		areaForSamePhothel = Potholes[i].area;
		for (int j =0 ; j< counterofCrackedForPhotholes ; j++)
		{
			if(i != j)
			{
			diffX = (Potholes[i].rect.x -  Potholes[j].rect.x) * (Potholes[i].rect.x -  Potholes[j].rect.x);
			diffy = (Potholes[i].rect.y -  Potholes[j].rect.y) * (Potholes[i].rect.y -  Potholes[j].rect.y);
			dis = sqrtf(diffX + diffy);
			if(dis < maxlimitdistancebetweenCrackedSamePhothole)
			{
				areaForSamePhothel +=Potholes[j].area;
			}
			}
		}
		if(areaForSamePhothel > areaAlowedToBePhotholeType)
		{
			counterofCrackedForPhotholes = 0;
			return true;
		}
	}
	counterofCrackedForPhotholes = 0;
	return false;
}


techContours.h
#ifndef TECH_CONTOURS
#define TECH_CONTOURS
#include "techBase.h"
#define MIN_TRACKER_COFIEIENT 15
#define MAX_TRACKER_COFIEIENT 200
class TechContours : public TechBase
{
	private:
		typedef		TechBaseInherited;
		IplImage* m_resizeImageLocally;
		IplImage* m_wholeImageProcessing;
		int m_CountOfContours;
		float m_totalCracedArea;
		float m_CrackedPercentage;
		float m_unknownCrackedPercentage;
	public:
```

```
        static int m_MinSize;
        static int m_MaxSize;
        static char m_FileName[MAX_STRLEN];

        public:
                                                TechContours        (void);
                virtual                 ~TechContours       (void);

                virtual void        InitTech                    (const char* winName = 0L);
                virtual void        ImplementTech               (
                                    IplImage* src = NULL
                                    , IplImage* dst = NULL
                                    , const CvSeq* inSpecData = NULL
                                    , void** outSpecData = NULL);
                virtual void                    EndTech                     (void);
                static  void                    OntTackbarKernelSize(int pos)
                                                                    {

WriteParmaeters(m_FileName, 2, m_MinSize, m_MaxSize /MAX_TRACKER_COFIEIENT );
                                                                                                    m_MinSize  = pos
* MIN_TRACKER_COFIEIENT;
                                                                    }
                static  void                    OntTackbarMeanOffset(int pos)
                                                                    {
WriteParmaeters(m_FileName, 2, m_MinSize / MIN_TRACKER_COFIEIENT, m_MaxSize);
                                                                                                    m_MaxSize  = pos *
MAX_TRACKER_COFIEIENT;

                                                                    }
};
#endif //TECH_CONTOURS
techInput.cpp
#include "techInput.h"
#include "globals.h"
TechInput::TechInput(void)
{
        strcpy_s(m_winName, MAX_STRLEN, "Input");
}
TechInput::~TechInput(void)
{
        EndTech();
}
void TechInput::InitTech (const char* winName /* = 0L */)
{
        if(gViisabiltyOfWindows)
{
        cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
        cvMoveWindow(m_winName, 1020,20);
        m_resizeImageLocally = cvCreateImage( cvSize((int)(INPUT_CONTOURS_LOCAL_IMAGE_WIDTH) ,
(int)(INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT) ), 8, 1 );
}
}
void TechInput::ImplementTech (
                IplImage* src /* = NULL */
                , IplImage* dst /* = NULL */
                , const CvSeq* inSpecData /* = NULL */
                , void** outSpecData /* = NULL */)
{
if(gViisabiltyOfWindows)
{
#ifdef RESIZE_WINDOW
```

```
		ResizeImage(src, m_resizeImageLocally,
INPUT_CONTOURS_LOCAL_IMAGE_WIDTH,INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT);
		cvShowImage(m_winName,m_resizeImageLocally);
#else
		cvShowImage(m_winName,src);
#endif
}
}
void TechInput::EndTech (void)
{
		cvReleaseImage(&m_resizeImageLocally);
		Inherited::EndTech();
}
techInput.h
#ifndef TECH_INPUT
#define TECH_INPUT
#include "techBase.h"
class TechInput : public TechBase
{

		private:
			typedef		TechBaseInherited;
			IplImage* m_resizeImageLocally;
		public:
							TechInput		(void);
			virtual			~TechInput		(void);
			virtual void	InitTech		(const char* winName = 0L);
			virtual void	ImplementTech	(
			IplImage* src = NULL
			, IplImage* dst = NULL
			, const CvSeq* inSpecData = NULL
			, void** outSpecData = NULL);
			virtual void		EndTech			(void);
};
#endif
techMedianFilter.cpp
#include "techMedianFilter.h"
#include "globals.h"
int			TechMedianFilter::m_KernelSize		= 1;
char		TechMedianFilter::m_FileName[MAX_STRLEN] = "\0";
void RemoveSpecificColor(IplImage* img, IplImage* dst)
{
		int x,y;
		for (y = 0; y < img->height; y++){
  uchar *ptr = (uchar*)(img->imageData + y * img->widthStep);
  for (x = 0; x < img->width; x++){
    if( ptr[x] > 170 && ptr[x] <=255 )
		{
			ptr[x] = rand() % 200 + 100;
		}
  }
}
}
static IplImage* equalize_histogram(
			IplImage* input
			,IplImage* imgEqualized);
TechMedianFilter::TechMedianFilter(void)
{
		strcpy_s(m_winName, MAX_STRLEN, "Median");		sprintf_s(m_FileName,
MAX_STRLEN, "%s%s", gSettingDir,"MedianFilter");		ReadParmaeters(m_FileName, 1,
&m_KernelSize);
}
```

```
TechMedianFilter::~TechMedianFilter(void)
{
	EndTech();
}
IplImage* equalize_histogram(IplImage* input,IplImage* imgEqualized)
{
	IplImage *output = cvCreateImage(cvGetSize(input),IPL_DEPTH_8U,1);
	int height = input->height;
	int width = input->width;
	int temp = 0;
	uchar* data_input, *data_output;
	data_input = (uchar*)input->imageData;
	data_output = (uchar*)imgEqualized->imageData;
	int array_input[256],array_output[256];
	//compute histogram of input image
	for (int i = 0; i < height; i++)
	{
		for (int j = 0; j < width; j++)
		{
			temp = data_input[i*(input->widthStep) + j];
			array_input[temp]++;
		}
	}
	array_input[0] = 0;
	//form cumulative histogram  for (int p=1;p<256;p++)
	{
		array_input[p] = array_input[p-1] + array_input[p];
	}
	//populate output histogram (normalized and scaled)
	for (int p=0;p<256;p++)
	{
		array_output[p] = (int)(((256-1)*(array_input[p]))/(height*width) + 0.5);
	}
	temp = 0;
	for (int i = 0; i < height; i++)
	{
		for (int j = 0; j < width; j++)
		{
			temp = data_input[i*(input->widthStep) + j];
	data_output[i*(input->widthStep) + j] = array_output[temp];
		}
	}
	return(imgEqualized);
}
void TechMedianFilter::InitTech (const char* winName /* = 0L */)
{
	if(gViisabiltyOfWindows)
{
	cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
	cvMoveWindow(m_winName, 680,20);
	cvCreateTrackbar( "KernelSize", m_winName, &m_KernelSize, 20, OntTackbarKernelSize);
	cvShowImage(m_winName,gSrcGray);
	//cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
	m_wholeImageProcessing = cvCreateImage(cvSize(gWidth, gHeight), 8, 1);
	cvSet(m_wholeImageProcessing,cvScalar(255));
	m_resizeImageLocally = cvCreateImage( cvSize((int)(INPUT_CONTOURS_LOCAL_IMAGE_WIDTH) ,
(int)(INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT) ), 8, 1 );
}
}
void TechMedianFilter::ImplementTech (
		IplImage* src /* = NULL */
		, IplImage* dst /* = NULL */
```

```
                , const CvSeq* inSpecData /* = NULL */
                , void** outSpecData /* = NULL */)
{
        //cvNot(src, dst);
        //equalize_histogram(src, dst);
        //cvEqualizeHist( src, src);
RemoveSpecificColor(src,src);
        //cvInRangeS(src, cvScalar(0,0,0), cvScalar(180,180,180), dst);
        cvSmooth(src, dst, CV_MEDIAN, m_KernelSize);
        //Mat srcMat(src);
        //Mat dstMat(dst);
        //medianBlur(srcMat,dstMat,m_KernelSize);
        //dst = cvCloneImage(&(IplImage)dstMat);
if(gViisabiltyOfWindows)
{
#ifdef RESIZE_WINDOW
        cvSetImageROI(m_wholeImageProcessing, gRectCurrentPartProcessing);
        cvCopy(dst, (IplImage*)m_wholeImageProcessing);
        cvResetImageROI(m_wholeImageProcessing);
        ResizeImage(m_wholeImageProcessing, m_resizeImageLocally,
INPUT_CONTOURS_LOCAL_IMAGE_WIDTH,INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT);
        cvShowImage(m_winName,m_resizeImageLocally);
        //      ResizeImage(dst, m_resizeImage);
//      cvShowImage(m_winName,m_resizeImage);
#else
        cvShowImage(m_winName,dst);
#endif
}
}
void TechMedianFilter::EndTech (void)
{
        cvReleaseImage(&m_wholeImageProcessing);
        cvReleaseImage(&m_resizeImageLocally);
        Inherited::EndTech();
}
techMedianFilter.h
#ifndef TECH_MEDIAN_FILTER
#define TECH_MEDIAN_FILTER
#include "techBase.h"
class TechMedianFilter : public TechBase
{
        private:
                typedef         TechBaseInherited;
                IplImage* m_wholeImageProcessing;
                IplImage* m_resizeImageLocally;
        public:
        static int m_KernelSize;
        static char m_FileName[MAX_STRLEN];
        public:
                                                TechMedianFilter   (void);
                virtual                 ~TechMedianFilter (void);
                virtual void    InitTech                (const char* winName = 0L);
                virtual void    ImplementTech   (
                                IplImage* src = NULL
                                , IplImage* dst = NULL
                                , const CvSeq* inSpecData = NULL
                                , void** outSpecData = NULL);
                virtual void                    EndTech                 (void);
                static  void                    OntTackbarKernelSize(int pos)
                                                                {
                                                                        if(pos <=1)
                                                                                m_KernelSize = 1;
```

```
                                                                        else if(pos%2 == 0)
                                                                                m_KernelSize  =
pos+1;
                                                                        else
                                                                                m_KernelSize  =
pos;
WriteParmaeters(m_FileName, 1, m_KernelSize);
                                                                }
};
#endif //TECH_MEDIAN_FILTER
techMorphology.cpp
#include "techMorphology.h"
include "globals.h"
int             TechMorphology::m_KernelSize            = 5;
int             TechMorphology::m_Type          = 1;
int             TechMorphology::m_Iteration             = 2;
char    TechMorphology::m_FileName[MAX_STRLEN] = "\0";
IplConvKernel* TechMorphology::m_strlElem = 0L;
TechMorphology::TechMorphology(void):m_imgTmp(0L)
{
        strcpy_s(m_winName, MAX_STRLEN, "Morphology");
        sprintf_s(m_FileName, MAX_STRLEN, "%s%s", gSettingDir,"Morphology");
        ReadParmaeters(m_FileName, 3, &m_KernelSize, &m_Type, &m_Iteration);
        m_imgTmp = cvCreateImage(cvSize(gWorkingWidth, gWorkingHeight), 8, 1);
}
TechMorphology::~TechMorphology(void)
{
        EndTech();
}
void TechMorphology::InitTech (const char* winName /* = 0L */)
{
        if(gViisabiltyOfWindows)
{
        cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
        cvMoveWindow(m_winName, 10,20);
        cvCreateTrackbar( "KernelSize", m_winName, &m_KernelSize, 20, OntTackbarKernelSize);
        cvCreateTrackbar( "Type", m_winName, &m_Type, 2, OntTackbarType);
        cvCreateTrackbar( "Iterations", m_winName, &m_Iteration, 10, OntTackbarIterations);
        cvShowImage(m_winName,gSrcGray);
        //cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
        m_wholeImageProcessing = cvCreateImage(cvSize(gWidth, gHeight), 8, 1);
        cvSet(m_wholeImageProcessing,cvScalar(255));
#
        m_resizeImageLocally = cvCreateImage( cvSize((int)(INPUT_CONTOURS_LOCAL_IMAGE_WIDTH) ,
(int)(INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT) ), 8, 1 );
        }
}
void TechMorphology::OntTackbarKernelSize(int pos)
                                                                {
                                                                        if(pos <=3)
                                                                                m_KernelSize = 3;
                                                                        else if(pos%2 == 0)
pos+1;
                                                                        else
                                                                                m_KernelSize  =
pos;
                                                                        //release current structElem
        cvReleaseStructuringElement(&m_strlElem);
                                                                //create a new one given new data
                                                                                m_KernelSize  =
```

```
                                                        m_strlElem = cvCreateStructuringElementEx (
                                                                m_KernelSize
                //int cols
                                                                , m_KernelSize
                //int rows
                                                                , (m_KernelSize - 1)>>1
        //int anchor_x
                                                                , (m_KernelSize - 1)>>2
        //int anchor_y
                                                                /****int shape****/
                                                                //, CV_SHAPE_RECT
        //The kernel is rectangular
                                                                //, CV_SHAPE_CROSS
        //The kernel is cross shaped
                                                                ,
m_Type//CV_SHAPE_ELLIPSE                //The kernel is elliptical
                                                                //, CV_SHAPE_CUSTOM
        //The kernel is user-defi ned via values
                                                                //int* values = NULL
                                                                );

        WriteParmaeters(m_FileName, 3, m_KernelSize, m_Type, m_Iteration);
                                                                }
void                    TechMorphology::OntTackbarType(int pos)
                                                                {
                                                        m_Type  = pos;
        cvReleaseStructuringElement(&m_strlElem);
                                                        //create a new one given new data
                                                        m_strlElem = cvCreateStructuringElementEx (
                                                                m_KernelSize
                //int cols
                                                                , m_KernelSize
                //int rows
                                                                , (m_KernelSize - 1)>>1
        //int anchor_x
                                                                , (m_KernelSize - 1)>>2
        //int anchor_y
                                                                /****int shape****/
                                                                //, CV_SHAPE_RECT
        //The kernel is rectangular
                                                                //, CV_SHAPE_CROSS
        //The kernel is cross shaped
                                                                ,
m_Type//CV_SHAPE_ELLIPSE                //The kernel is elliptical
                                                                //, CV_SHAPE_CUSTOM
        //The kernel is user-defi ned via values
                                                                //int* values = NULL
                                                                );

        WriteParmaeters(m_FileName, 3, m_KernelSize, m_Type, m_Iteration);
                                                                }
void TechMorphology::ImplementTech (
                IplImage* src /* = NULL */
                , IplImage* dst /* = NULL */
                , const CvSeq* inSpecData /* = NULL */
                , void** outSpecData /* = NULL */)
{
//before apply morphology remve small area
        CvMemStorage *mem;
        mem = cvCreateMemStorage(0);
        CvSeq *contours = 0;
        CvScalar(ext_color);
```

```
   float area = 0.f;
        ext_color = CV_RGB( 0, 0, 0 );
        int minmumSize = 50;
        int n = cvFindContours(src, mem, &contours, sizeof(CvContour), CV_RETR_CCOMP,
CV_CHAIN_APPROX_NONE, cvPoint(0,0));
        for (; contours != 0; contours = contours->h_next)
        {
                area = fabsf((float)cvContourArea(contours /*const CvArr* points*/, CV_WHOLE_SEQ));
                if(area < minmumSize)
                {
                        cvDrawContours(src, contours, ext_color, CV_RGB(0,0,0), -1, CV_FILLED, 8,
cvPoint(0,0));
                }
        }
        /*cvDilate(src, dst,NULL,2);*/
        cvMorphologyEx (
                                src                                             //const CvArr* src
                                , dst                                   //CvArr* dst
                                , m_imgTmp                              //CvArr* temp
                                , m_strlElem                    //IplConvKernel* element
                                , CV_MOP_CLOSE                  //(see note below)
                                , m_Iteration                   //int iterations = 1
                                );
                // CV_MOP_OPEN                  //No temp required
                // CV_MOP_CLOSE                         //No temp required
                // CV_MOP_GRADIENT              //temp image always required
                // CV_MOP_TOPHAT                //temp image required for in-place only (src = dst)
                // CV_MOP_BLACKHAT              //temp image required for in-place only (src = dst)
if(gViisabiltyOfWindows)
{
#ifdef RESIZE_WINDOW
        cvSetImageROI(m_wholeImageProcessing, gRectCurrentPartProcessing);
        cvCopy(dst, (IplImage*)m_wholeImageProcessing);
        cvResetImageROI(m_wholeImageProcessing);
        ResizeImage(m_wholeImageProcessing, m_resizeImageLocally,
INPUT_CONTOURS_LOCAL_IMAGE_WIDTH,INPUT_CONTOURS_LOCAL_IMAGE_HEIGHT);
        cvShowImage(m_winName,m_resizeImageLocally);
//      ResizeImage(dst, m_resizeImage);
//      cvShowImage(m_winName,m_resizeImage);
#else
        cvShowImage(m_winName,dst);
#endif
}
}
void TechMorphology::EndTech (void)
{
        cvReleaseImage(&m_wholeImageProcessing);
        cvReleaseImage(&m_resizeImageLocally);
        cvReleaseImage(&m_imgTmp);
        Inherited::EndTech();
}
techMorphology.h
#ifndef TECH_MORPHOLOGY
#define TECH_MORPHOLOGY
#include "techBase.h"
class TechMorphology : public TechBase
{
        private:
                typedef         TechBaseInherited;
                IplImage* m_wholeImageProcessing;
                IplImage* m_resizeImageLocally;
        public:
```

```
        static int m_KernelSize;
        static int m_Type;
        static int m_Iteration;
        static char m_FileName[MAX_STRLEN];
        static IplConvKernel* m_strlElem;
        IplImage* m_imgTmp;
        public:


                virtual void        ImplementTech               (
                                    IplImage* src = NULL
                                    , IplImage* dst = NULL
                                    , const CvSeq* inSpecData = NULL
                                    , void** outSpecData = NULL);

                virtual void                        EndTech                     (void);
                static  void                        OntTackbarKernelSize(int pos);
                static  void                        OntTackbarType(int pos);
                static  void                        OntTackbarIterations(int pos)
                                                                                {
                                                                                    m_Iteration  = pos;
                WriteParmaeters(m_FileName, 3, m_KernelSize, m_Type, m_Iteration);
                                                                                }
};
#endif //TECH_MORPHOLOGY
techSubtract.cpp
}
void TechSubtract::ImplementTech (
include "techSubtract.h"
#include "globals.h"
char        TechSubtract::m_FileName[MAX_STRLEN] = "\0";
TechSubtract::TechSubtract(void)
{
        strcpy_s(m_winName, MAX_STRLEN, "Subtract");
        sprintf_s(m_FileName, MAX_STRLEN, "%s%s", gSettingDir,"Subtract");
}
TechSubtract::~TechSubtract(void)
{
        EndTech();
}
void TechSubtract::InitTech (const char* winName /* = 0L */)
{
        if(gViisabiltyOfWindows)
{
        cvNamedWindow(m_winName, CV_WINDOW_AUTOSIZE );
        cvMoveWindow(m_winName, 340,20);
}

                IplImage* src /* = NULL */
                , IplImage* dst /* = NULL */
                , const CvSeq* inSpecData /* = NULL */
                , void** outSpecData /* = NULL */)
{
        cvAbsDiff(src, dst,dst);
        if(gViisabiltyOfWindows)
{
#ifdef RESIZE_WINDOW
        ResizeImage(dst, m_resizeImage);
        cvShowImage(m_winName,m_resizeImage);
#else
        cvShowImage(m_winName,dst);
#endif
```

```
		}
}
void TechSubtract::EndTech (void)
{
		Inherited::EndTech();

}
```

techSubtract.h

```
#ifndef TECH_SUBTRACT
#define TECH_SUBTRACT
#include "techBase.h"
class TechSubtract : public TechBase
{
		private:
				typedef		TechBaseInherited;
		public:
		static char m_FileName[MAX_STRLEN];
		public:
				TechSubtract		(void);
				virtual		~TechSubtract	(void);
				virtual void	InitTech		(const char* winName = 0L);
				virtual void	ImplementTech		(
					IplImage* src = NULL
					, IplImage* dst = NULL
					, const CvSeq* inSpecData = NULL
					, void** outSpecData = NULL);
				virtual void		EndTech			(void);
};
#endif //TECH_SUBTRACT
```

Appendix B : The Code of the Count Data model Regression Analysis.

```
summary(poisson_models.freq_1)
Call:
glm(formula = y_cr ~ y_mo + y_af + y_ev, family = "poisson")
Deviance Residuals:
   Min      1Q  Median      3Q     Max
-3.1623  -2.9547  -0.1774   1.9654   2.3040
Coefficients: (1 not defined because of singularities)
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.47368    0.03704  39.790  < 2e-16 ***
y_mo         0.13576    0.05021   2.704  0.00686 **
y_af         0.09519    0.05125   1.857  0.06326 .
y_ev              NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
   Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2767.2  on 504  degrees of freedom
AIC: 3960
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_2)
Call:
glm(formula = y_cr ~ y_mo, family = "poisson")
Deviance Residuals:
```

```
   Min       1Q   Median       3Q      Max
-3.1623  -3.0274  -0.2782   1.9654   2.1844
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.52226    0.02560  59.467   <2e-16 ***
y_mo         0.08718    0.04248   2.052   0.0402 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
   Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2770.6  on 505  degrees of freedom
AIC: 3961.5
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_3)
Call:
glm(formula = y_cr ~ y_mo + y_af, family = "poisson")
Deviance Residuals:
   Min       1Q   Median       3Q      Max
-3.1623  -2.9547  -0.1774   1.9654   2.3040
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.47368    0.03704  39.790  < 2e-16 ***
y_mo         0.13576    0.05021   2.704  0.00686 **
```

```
y_af         0.09519    0.05125   1.857  0.06326 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2767.2  on 504  degrees of freedom
AIC: 3960
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_4)
Call:
glm(formula = y_cr ~ y_mo + y_ev, family = "poisson")
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.1623  -2.9547  -0.1774   1.9654   2.3040
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.56887    0.03542  44.292   <2e-16 ***
y_mo         0.04057    0.04903   0.827   0.4080
y_ev        -0.09519    0.05125  -1.857   0.0633 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 2774.8  on 506  degrees of freedom
```

```
Residual deviance: 2767.2  on 504  degrees of freedom
AIC: 3960
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_5)
Call:
glm(formula = y_cr ~ y_af + y_ev, family = "poisson")
Deviance Residuals:
   Min       1Q   Median       3Q      Max
-3.1623  -2.9547  -0.1774   1.9654   2.3040
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.60944    0.03390  47.472  < 2e-16 ***
y_af        -0.04057    0.04903  -0.827  0.40798
y_ev        -0.13576    0.05021  -2.704  0.00686 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
   Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2767.2  on 504  degrees of freedom
AIC: 3960
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_6)
Call:
```

```
glm(formula = y_cr ~ y_mo + y_af + y_ev + y_sp20 + y_sp40 + y_sp60 +
    y_sp80 + y_sp100, family = "poisson")

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.2282  -2.9969  -0.1894   1.8973   2.5200

Coefficients: (2 not defined because of singularities)
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.38426    0.05716  24.217   <2e-16 ***
y_mo         0.13519    0.05022   2.692   0.0071 **
y_af         0.09526    0.05125   1.859   0.0631 .
y_ev             NA         NA      NA       NA
y_sp20       0.13126    0.06488   2.023   0.0431 *
y_sp40       0.07361    0.06682   1.102   0.2706
y_sp60       0.11778    0.06612   1.781   0.0749 .
y_sp80       0.11665    0.06616   1.763   0.0779 .
y_sp100          NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2761.8  on 500  degrees of freedom
AIC: 3962.7

Number of Fisher Scoring iterations: 5
```

```
> summary(poisson_models.freq_7)

Call:

glm(formula = y_cr ~ y_sp20 + y_sp40 + y_sp60 + y_sp80 + y_sp100,

    family = "poisson")

Deviance Residuals:

    Min       1Q   Median       3Q      Max

-3.1415  -3.0496  -0.3089   1.9989   2.3295

Coefficients: (1 not defined because of singularities)

            Estimate Std. Error z value Pr(>|z|)

(Intercept)  1.46326    0.04811  30.414   <2e-16 ***

y_sp20       0.13301    0.06487   2.050   0.0403 *

y_sp40       0.07361    0.06682   1.102   0.2706

y_sp60       0.11778    0.06612   1.781   0.0749 .

y_sp80       0.11572    0.06616   1.749   0.0802 .

y_sp100          NA         NA      NA       NA

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2774.8  on 506  degrees of freedom

Residual deviance: 2769.4  on 502  degrees of freedom

AIC: 3966.2

Number of Fisher Scoring iterations: 5

> summary(poisson_models.freq_8)
```

Call:

glm(formula = y_cr ~ y_mo + y_af + y_ev + y_sp20, family = "poisson")

Deviance Residuals:

Min	1Q	Median	3Q	Max
-3.2282	-3.0176	-0.1546	1.9955	2.3310

Coefficients: (1 not defined because of singularities)

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	1.46263	0.03849	38.004	<2e-16	***
y_mo	0.13496	0.05022	2.688	0.0072	**
y_af	0.09480	0.05125	1.850	0.0644	.
y_ev	NA	NA	NA	NA	
y_sp20	0.05312	0.04929	1.078	0.2812	

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 2774.8 on 506 degrees of freedom

Residual deviance: 2766.0 on 503 degrees of freedom

AIC: 3960.8

Number of Fisher Scoring iterations: 5

> summary(poisson_models.freq_9)

Call:

glm(formula = y_cr ~ y_mo + y_af + y_ev + y_sp40, family = "poisson")

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.1684  -2.9605  -0.1854   1.9556   2.3430


Coefficients: (1 not defined because of singularities)
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.47758    0.03837  38.504  < 2e-16 ***
y_mo         0.13571    0.05021   2.703  0.00687 **
y_af         0.09521    0.05125   1.858  0.06319 .
y_ev             NA         NA      NA       NA
y_sp40      -0.01988    0.05166  -0.385  0.70041
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2767.0  on 503  degrees of freedom
AIC: 3961.9
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_10)
Call:
glm(formula = y_cr ~ y_mo + y_af + y_ev + y_sp60, family = "poisson")
Deviance Residuals:
    Min       1Q   Median       3Q      Max
```

```
-3.2073  -2.9967  -0.1628   1.9832   2.3213

Coefficients: (1 not defined because of singularities)

            Estimate Std. Error z value Pr(>|z|)

(Intercept)  1.46662    0.03845  38.145  < 2e-16 ***

y_mo         0.13584    0.05021   2.705  0.00682 **

y_af         0.09514    0.05125   1.857  0.06337 .

y_ev              NA         NA      NA       NA

y_sp60       0.03523    0.05080   0.693  0.48802

---

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2774.8  on 506  degrees of freedom

Residual deviance: 2766.7  on 503  degrees of freedom

AIC: 3961.5

Number of Fisher Scoring iterations: 5

> summary(poisson_models.freq_11)

Call:

glm(formula = y_cr ~ y_mo + y_af + y_ev + y_sp80, family = "poisson")

Deviance Residuals:

    Min       1Q   Median       3Q      Max

-3.2054  -2.9946  -0.2329   1.9825   2.3211

Coefficients: (1 not defined because of singularities)

            Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept)  1.46670    0.03853  38.064  < 2e-16 ***
y_mo         0.13604    0.05021   2.709  0.00674 **
y_af         0.09556    0.05125   1.864  0.06225 .
y_ev             NA         NA      NA       NA
y_sp80       0.03380    0.05085   0.665  0.50620
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2766.7  on 503  degrees of freedom
AIC: 3961.6
Number of Fisher Scoring iterations: 5
> summary(poisson_models.freq_12)
Call:
glm(formula = y_cr ~ y_mo + y_af + y_ev + y_sp100, family = "poisson")
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.1951  -2.9858  -0.2205   1.9128   2.5203
Coefficients: (1 not defined because of singularities)
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.49456    0.03826  39.063  < 2e-16 ***
y_mo         0.13552    0.05021   2.699  0.00695 **
y_af         0.09531    0.05125   1.860  0.06290 .
```

```
y_ev            NA        NA     NA       NA
y_sp100     -0.11043    0.05314  -2.078  0.03769 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 2774.8  on 506  degrees of freedom
Residual deviance: 2762.8  on 503  degrees of freedom
AIC: 3957.6
Number of Fisher Scoring iterations: 5
```