COMPUTER VISION AIDED BRIDGE MONITORING SYSTEM

by

SUSHRUTH MUNE GOWDA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2014

ACKNOWLEDGEMENTS

This thesis would not have been possible without the constant support and encouragement I received from Dr. Roger Walker. From the first vague proposal of this topic to later queries on focus and connection, he was always eager to entertain my ideas and help me however he can. I sincerely thank him for his consistent efforts and true desire to keep me on track.

I would also like to thank my committee members, Dr. Jonathan Bredow and Dr. Alan Davis for serving in my defense committee despite their overwhelmingly busy schedule.

Special thanks to my friends and fellow lab-mates Ashwin Arikere, Vikram Simha and Sachin Jayaram who helped and encouraged me in every step of the way in my time at the university.

Finally, I would like to express my deepest gratitude to my family. Their support and unwavering confidence in my ability helped me achieve my academic dreams.

November 13, 2014

ABSTRACT

COMPUTER VISION AIDED BRIDGE MONITORING SYSTEM

Sushruth Mune Gowda, M.S

The University of Texas at Arlington, 2014

Supervising Professor: Roger Walker

With bridges representing a significant part of the road network in the United States, continual monitoring and early detection of deterioration in these structures is vital to prevent expensive repairs or catastrophic failures. Advances in sensing and information technologies help in monitoring and evaluating the health of a bridge. Currently, all the bridge monitoring systems are static in nature. With the help of TxDOTs Pavement Management Information System (PMIS) survey, the proposed mobile Bridge Monitoring System will periodically monitor these structural changes and help the maintanance management decision making process. The mobile Bridge Monitoring System has been the topic of ongoing research at the Transportation Instrumentation Laboratory for a few years now. The objective of this research is to solve issues related to the proposed Bridge Monitoring System which are necessary for its implementation.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

Bridges represent a significant part of the road network in the United States. Continual monitoring and early detection of deterioration in these structures is vital to prevent expensive repairs or catastrophic failures. The reason for these deteriorations is various external factors such as constant movement of cars and truck traffic, earthquakes, winds and waves. By periodically monitoring these structural changes, engineers can continually access information for the maintenance of the bridge.

Currently, there are a number of tools and technologies in use to find structural changes; however, current sensing systems often require lengthy cables, are limited in coverage, or require manual control. Thus, these systems require much effort and manpower to test the integrity of the bridge. Deploying men to nearly 607,380 bridges is a costly affair. According to the American Society of Civil Engineers' eliminating all bridge deficiencies in the United States within the next 15 years would require an investment of over $20.5 billion each year![1] Therefore, the development of efficient monitoring tools to minimize cost is a high priority.

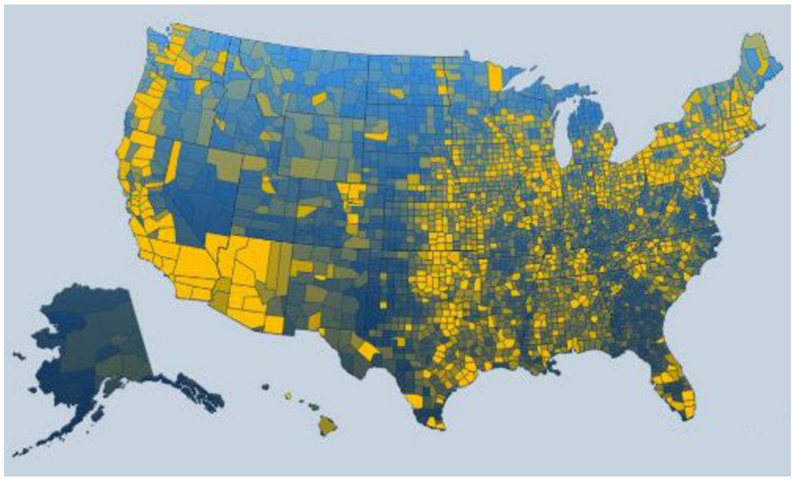


Figure 1-1: Number of Deficient Bridges per county (Blue: Low, Yellow: High)

The Texas Department of Transport (TxDOT) conducts an annual survey called Pavement Management Information System (PMIS) survey. The purpose of this survey is to collect and analyze data concerning the health of pavements. With this data the ride quality, structural adequacy, skid resistance, climate, and traffic data is added to describe the overall condition of the road network. The PMIS data is used by the decision makers at TxDOT to estimate overall pavement maintenance, rehabilitation, and reconstruction needs. The PMIS data can also be used to project future needs and the effects of funding on pavement condition. Currently, TxDOT has data collection vehicles equipped with various sensors and instruments that determine the condition of the pavement.

A Bridge Monitoring System using cameras has been a topic of ongoing research at the Transportation and Instrumentation Laboratory. It is intended to collect visual data of all the bridges and overpasses in Texas during the PMIS data collection process. The bridge monitoring system would annually update a database of all these bridges. A program would then look for changes in bridge structures by analyzing the visual information in the database. If a significant change is detected by the system, the decision makers are notified, and more thorough static measurements could be directed for those bridges.

The proposed method for a Bridge Monitoring System would be incorporated in the PMIS data collection activities. The major components used are:

1. Control System to stabilize the camera platform.
2. Camera system to capture video footage of the bridge.
3. Video processing module for post processing and measuring changes in the structure.
4. Bridge Surface Profile Information.

The objective of this research project is to solve issues related to the proposed system. Since the PMIS data collection is conducted annually, from a moving vehicle at different locations and times, it is prone to unreliability due to vibration, rotation and shift. This unreliability causes a change in the camera position, which in turn changes features in the visual data. By solving these problems the project can move to the next stage, which is, measuring the amount of change.

Several Computer Vision techniques such as lane detection, vehicle tracking feature detection and feature matching are used to make sure that the data acquisition is accurate and reliable over time.

Chapter 2 provides an overview of the research project.

Chapter 3 provides background information on all the different components involved in the project.

Chapter 4 discusses the implementation of the bridge monitoring system.

Chapter 5 contains the results of the experiments and the observations made during these experiments

Chapter 6 provides the summary of the project and suggestions for future work.

Chapter 2

PROJECT OVERVIEW AND COMPONENT DESCRIPTION

This chapter provides a description of the proposed system and system components.

2.1 Project Overview

This research project contains a thorough investigation on methods to solve challenges associated with the proposed Bridge Monitoring System. The main problems that have been identified are:

1. Camera Position/Vehicle Position: Every year the PMIS survey vehicle travels on the road collecting data. By ensuring that the camera is at same pose every year the data collection is branded reliable. Thus, running matching and analysis algorithms on the data does not result in errors.
2. Camera Jitter: the camera is prone to vibrations and jitter since it is placed on a moving vehicle. The vibrations are reduced by using dampers.
3. Rotation and Shift: even though most of this is handled by the control system there is a need for handling this during post processing to make the system more robust.
4. Illumination: two pictures taken from the same position, but at different times may differ due to the varying amount of light. This needs to be handled during post processing.

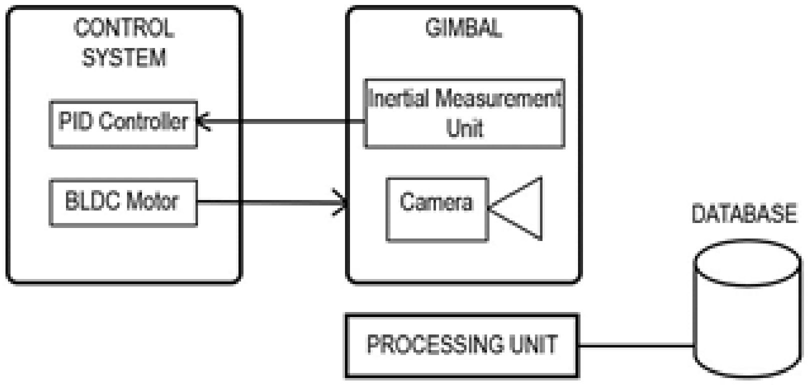


Figure 2-1: Block Diagram of the Bridge Monitoring System

It is proposed that when TxDOT conducts PMIS survey annually that visual data in conjunction with the surface-profile of bridges is also collected on these runs. Detection of structural movement in the bridge structure is done by comparing two images taken at different times. Error-free data acquisition is possible only after addressing problems like vibration of vehicle, rotation and shift of vehicle and camera. Because the final image is chosen from a video, the data must be recorded and must pass through many stages that check the reliability of the image before measuring the structural parameters. Techniques such as lane detection and Scale Invariant Feature Transforms (SIFT) are used to make the video acquisition accurate and reliable over time. Every frame in the video contains details and keypoints which are extracted by computer vision methods and registered for further processing.

To present an idea of the process of detecting structural changes, consider the following scenario. Assume that after carefully handling all the problems the database contains two different sets of data from year 1 and year 2 for a particular bridge. To measure the amount of change the system takes the difference between data from year 1 and year 2. If the system is going to correctly measure the changes, the input data has to be precise. Hence, before measuring the changes the system has to make sure it is not viewing the bridge from another perspective.

2.2 Gimbal

The camera stabilizer platform uses a setup that mimics a gyro-stabilized 2 axis gimbal by using Brushless Direct Current motors (BLDC) and a Proportional-Integral-Derivative (PID) controller. Figure 2-2 shows a drawing of the 2 axis gimbal platform. It has two motors, one for the roll axis and the other one for the pitch axis. It provides housing for the gimbal controller as well. Since the assembly is mounted over the PMIS survey vehicle, it is prone to vibrations. The vibrations are significantly reduced by dampers that are shown in the figure. The motors and Inertial Measurement Unit are attached to the camera platform. The central part is the Gimbal Controller which is manufactured by RCTimer[2].

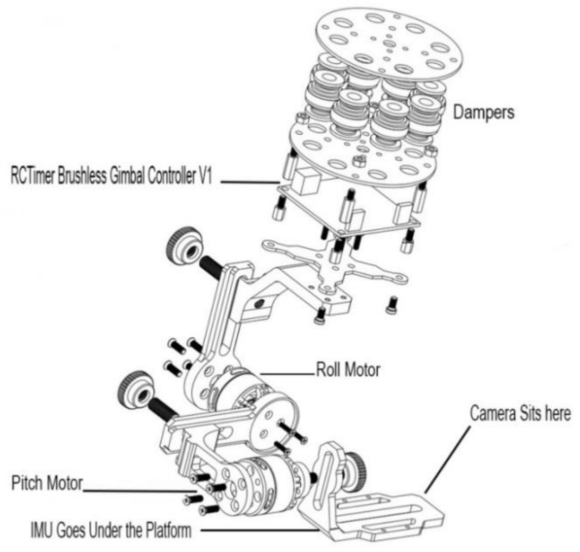

Dampers
RCTimer Brushless Gimbal Controller V1
Roll Motor
Camera Sits here
Pitch Motor
IMU Goes Under the Platform

Figure 2-2: 2-Axis Gimbal illustration

2.3 Brushless DC Motor and Motor controller

The Brushless DC (BLDC) Motor is an HP2212 brushless gimbal motor made by RCTimer. Wire windings are 0.015 mm in diameter. It has a three core setup, 70 turns per core and weighs only 0.1 Kg.

The BLDC motor controller is a RCTimer brushless gimbal controller v1.0 manufactured by RCTimer Inc. It has:

1. An ATMega328P microcontroller
2. Three channels input
3. A UART port for debugging
4. An Onboard logic level converter
5. An Inertial Measurement Unit.

2.4 Video Recording Unit

The video recording unit is a digital camera controlled by a computer program. This program uses OpenCV libraries to perform computer vision related computations. Predicated on the distance traveled and location of the vehicle, video is recorded by the application and stored for post-processing.

The application needs a camera that can be controlled from a third party application, which is using OpenCV libraries. For this project, a GoPro Hero3 White edition is chosen because of its superior capabilities and efficiency. The camera outputs data through a USB port.

Table 2-1: Camera Specifications

Video Resolution	**1080p**	**960p**	**720p**	**WVGA**
FPS	30,25	30,25	60,50,30,25	60,50
Field of View	Medium	Ultra Wide	Ultra Wide	Ultra Wide
Screen Resolution	1920 x 1080	1280 x 960	1280 x 720	848 x 480
Aspect Ratio	16:9	4:3	16:9	16:9

The camera supports H.264 codec and .mp4 file format. The camera has a Spot meter which is ideal for filming in tough lighting conditions when the camera is pointed towards a brighter outer setting from indoors. It houses an Ultrasharp 6-element a-spherical glass lens with a fixed aperture of f/2.8 [3].

2.5 Processing Unit

The embedded processing unit is the most important part of designing any such system. The key to selecting the processor is its use. There are many processors available today that perform different functions. For example, if the requirements of a system are plain signal collection and processing a Digital Signal Processor (DSP) is suitable. If the system has to perform a variety of tasks, the General Purpose Processor (GPP) is chosen instead of a DSP because the DSP does only one thing. For the application of this project, which is to collect video and run various computer vision algorithms, a general purpose processor or a graphics processing unit is used.

2.5.1 General Purpose Processor

General Purpose Processors (GPP) are designed for general purpose computers such as PCs or workstations. The computation speed of a GPP is the primary concern, and the cost of the GPP is usually much higher than that of DSPs and microcontrollers. All techniques that can increase CPU speed have been applied to GPPs. For example, GPPs often include on-chip cache and on-chip DMAs. Commonly used math operations are also supported by the on-chip hardware. GPPs are not designed for fast real-time applications. Scalar structure is typical in GPPs but rarely seen in DSPs and microcontrollers[4]. A few characteristics of GPPs are

1. Speed is the primary consideration in GPPs
2. Provide circuits for memory management like on-chip DMA, on-chip cache
3. Provide hardware circuits for commonly used math and logic operations.
4. Provide a balanced instruction set
5. Use pipelining
6. Use scalar operations
7. Use wide data buses
8. Cost is a secondary consideration[4].

2.5.2 Graphics Processing Unit (GPU)

The Graphics Processing Unit (GPU) is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images. The GPU came about as a way to offload graphic-intensive tasks from the CPU, freeing up processing power. Specialized logic chips now allow fast graphic and video implementations.

The graphics processing unit can render images more quickly than a CPU because of its parallel processing architecture, which allows it to perform multiple calculations at the same time. The resulting performance improvements have made GPUs popular chips for other resource-intensive tasks unrelated to graphics. Applications such as computer-aided design (CAD) can process over 200 billion operations per second and deliver up to 17 million polygons per second. Many scientists and engineers use GPUs for more in-depth calculated studies utilizing vector and matrix features[5].

2.6 Database and Memory Management

Mixed database management systems (DBMSs) are recommended to support this application for PMIS. Sybase SQL Server is recommended for enterprise-wide and workgroup applications. Sybase SQL Anywhere is recommended for PC workstation applications that have the potential for expansion beyond a single workstation and small workgroup applications. Microsoft Access is recommended for individual workstation database applications[6]. Since this project deals with storing videos and working on post-processing, a large database is required. For local storage, a disk with large capacity and high read and write speeds will be appropriate for this project. While selecting the memory for use in this project, the bandwidth and access time are considered. A large bandwidth and low access time are necessary for the project running on a GPP or a GPU.

For this project, the processing unit consists of Intel’s Core i7 with 8 cores at 2.67GHz. The system also has a 6GB RAM installed to make the processing even faster and an NVIDIA GeForce GTX 560 GPU which supports OpenCV to the fullest.

Chapter 3

BACKGROUND INFORMATION ON THE BRIDGE MONITORING SYSTEM

3.1 The Pavement Management Information System

The Pavement Management Information System (PMIS) is a system used by TxDOT for collecting, analyzing, storing and reporting information to help with the pavement-related decision-making process. It is an analysis tool to aid in pavement management. It supports a wide range of activities including planning, highway design, maintenance and rehabilitation, evaluation, research, and even extensive, detailed reporting of decision makers [7].

3.1.1 PMIS Data Elements

PMIS is a statewide management tool for monitoring and improving Texas roadways, most of the PMIS data is related to pavement distress conditions for discrete roadway sections. PMIS also imports and stores large amounts of data collected by other databases. Data stored on PMIS are collected and assigned to unique roadway sections. TxDOT defines and identifies the road sections by the roadway designation, beginning, and ending reference markers and the corresponding displacements from each. These reference markers and displacement data by TxDOT in the Texas Reference Marker (TRM) database and are annually imported onto PMIS in order to update the section information.

PMIS stores the data pertaining to the primary location, visual distress, nonvisual distress, condition scores, maintenance, climate, traffic, pavement type, and cross-section of a given roadway section. It includes extensive data elements to describe visual distress, deflection, skid resistance, rutting and ride quality. Apparently, though, the bulk of PMIS data are concerned with recording pavement distress types and severity for roadways sections throughout the state. However, PMIS also imports some of the data elements stored on other databases [7].

3.1.2 Future of PMIS

One interesting aspect of the PMIS is its interaction with other major pavement/materials databases within TxDOT. By merging many maintenance aspects of the same organization into one, the cost efficiency increases and output increases. The consequence of all the aforementioned connections is that PMIS serves as a strong base to incorporate the Bridge Monitoring System.

3.2 Computer Vision

3.2.1 Introduction

Recall that the Bridge Monitoring System captures and processes visual data. To process the visual data it has to deal with images that have information about the real world in the form of a 2D image. Computer Vision is the science and engineering discipline concerned with making inferences about the real world, given one or more of its images. Science aspect comes in because there are fundamental principles associated with the physics of image formation. Engineering aspect comes in because there are many practical applications to Computer Vision, one of which is the Bridge Monitoring System[8][9].

Computer Vision is described as the inverse of Computer graphics that deals with producing an image from information such as scene geometry, reflectance and lighting. Computer vision must address the inverse problem: given an image/multiple images, reconstruct the scene geometry, reflectance and illumination. There are three interconnected processes in vision: Recognition of objects and activities, Reorganization from pixels to objects and Reconstructing 3D structure.

3.2.2 Camera

The camera is the main part of the project since it provides access to the information needed to detect structural changes in the bridge. A camera takes an image of the real world. When looking at the image in Figure 3-1, it looks like a man taking a photograph. To a computer this image is just a bunch of pixels that have some values associated with each pixel. It is the human brain that perceives values to form a structure and understand what it is. The goal of computer vision is to give computers the same capability. In a grand sense, the goal is to go from pixels to perception. So how do images get stored digitally? The answer lies in understanding how a camera works.

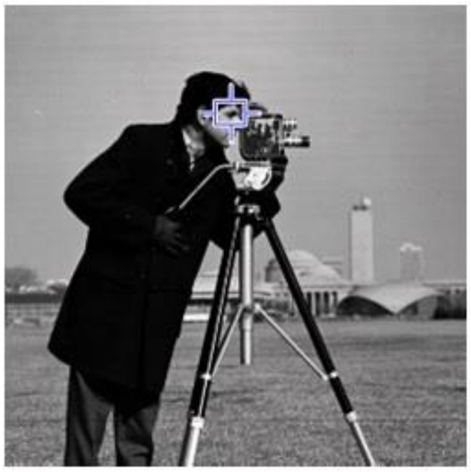
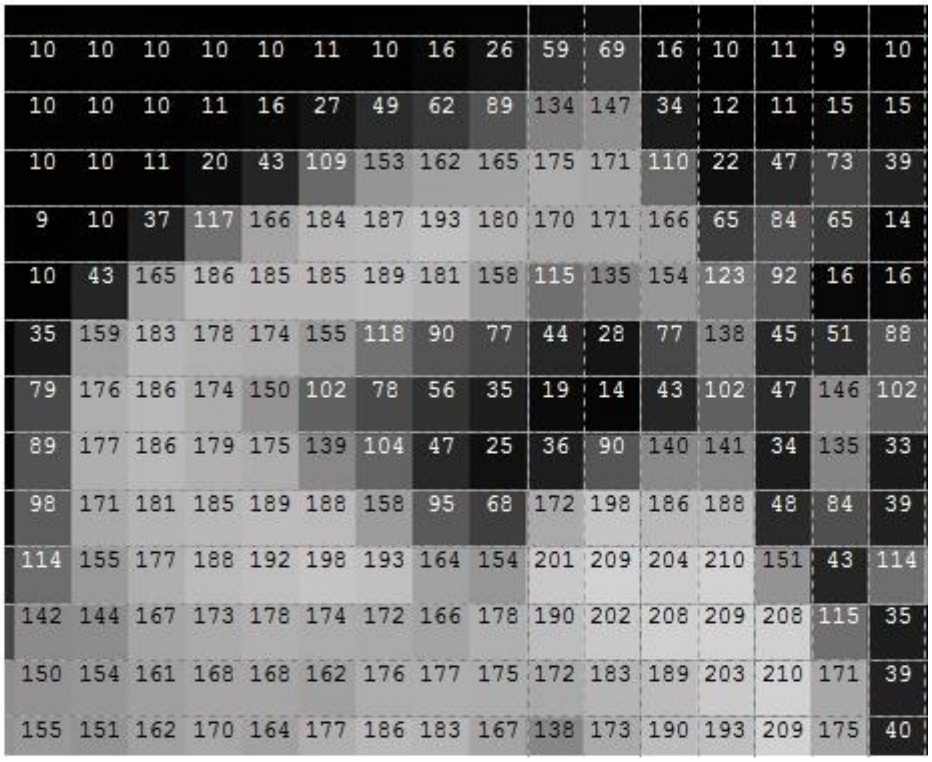


Figure 3-1: Photo of a man. Consider the pixels in the blue box.



10	10	10	10	10	11	10	16	26	59	69	16	10	11	9	10
10	10	10	11	16	27	49	62	89	134	147	34	12	11	15	15
10	10	11	20	43	109	153	162	165	175	171	110	22	47	73	39
9	10	37	117	166	184	187	193	180	170	171	166	65	84	65	14
10	43	165	186	185	185	189	181	158	115	135	154	123	92	16	16
35	159	183	178	174	155	118	90	77	44	28	77	138	45	51	88
79	176	186	174	150	102	78	56	35	19	14	43	102	47	146	102
89	177	186	179	175	139	104	47	25	36	90	140	141	34	135	33
98	171	181	185	189	188	158	95	68	172	198	186	188	48	84	39
114	155	177	188	192	198	193	164	154	201	209	204	210	151	43	114
142	144	167	173	178	174	172	166	178	190	202	208	209	208	115	35
150	154	161	168	168	162	176	177	175	172	183	189	203	210	171	39
155	151	162	170	164	177	186	183	167	138	173	190	193	209	175	40

Figure 3-2: Digital images array example.

3.2.2.1 Pin Hole Camera

Imagine a totally dark room with a tiny hole in one wall. That hole projects an accurate image of the outside world onto the opposite wall. Without film, a picture cannot be captured with it. However, the image can be traced with a pen. Aristotle was familiar with that idea, and medieval writers had a lot to say about it. Our word, camera, comes from the Latin word for a darkened chamber — a camera obscura[10].

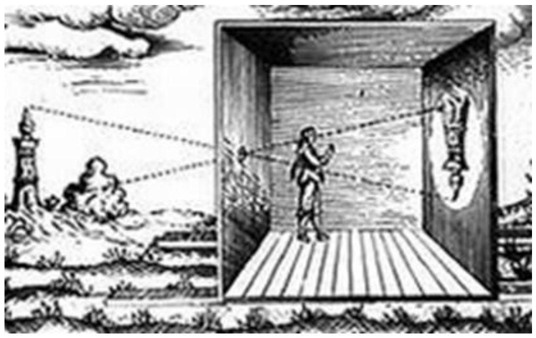


Figure 3-3: Camera Obscura

A camera maps a 3D object into a 2D image. In the figure below the point P(X,Y,Z) in the real world is projected onto the image plane at P'(x, y).

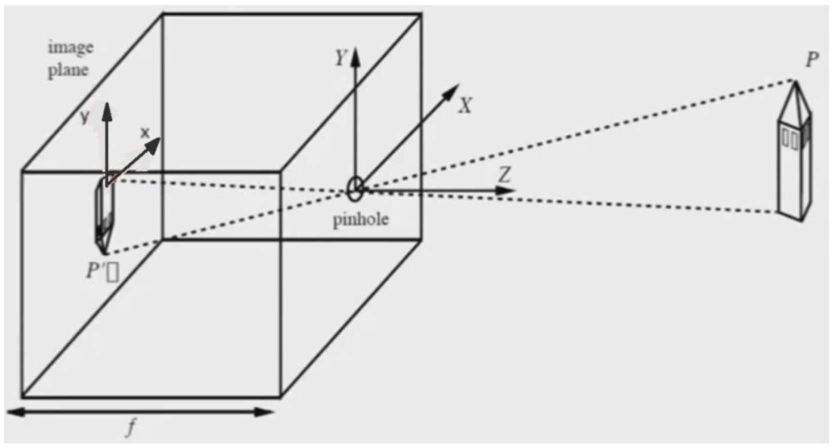


Figure 3-4: Basic Pinhole Camera

The pinhole camera has an *optical center* C (also known as camera projection center) and an *image plane* as shown in Figure 3-4. The distance of the image plane from C is the focal length, f. The principal axis or optical axis is the perpendicular line from the camera center to the image plane of the camera. The principal plane or focal plane is the plane parallel to the plane containing the optical center of the camera. The central or perspective projection define the relationship between the 3D coordinates of a scene point and the coordinates of its projection onto the image[12].

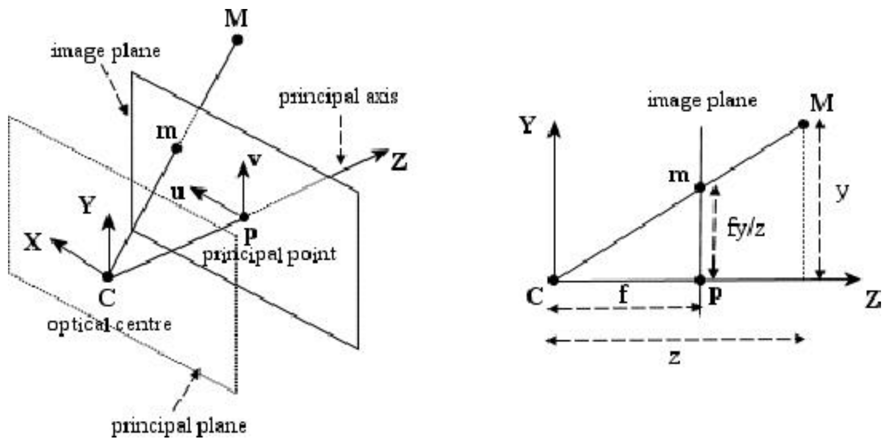


Figure 3-5: Principles of operation of a pinhole camera.

In Figure 3-5, a 3D point is projected onto the image plane with a line containing the point and the optical center. Let the center of projection, C be the origin wherein the z-axis is the principal axis. By similar triangles, it is readily seen that the 3D point $(x,y,z)^T$ can be represented as $(f_x/z\,, f_y/z)^T$ on the image plane. Therefore, X and Y are written as

$$X = fx/z \qquad (3.1)$$

$$Y = fy/z \qquad (3.2)$$

The image plane is a 2D array of sensor elements. There are mainly two types of sensors: CCD (Charge Coupled Device) and CMOS (Complementary Metal Oxide on Silicon). In CCD sensors, charge accumulates during exposure and are transferred out to shift registers, digitized and read. In CMOS sensors, the conductivity of each photo-detector is affected by light. Each value is then digitized and read out using a standard multiplexing scheme. Figure 3-6 shows the arrangement of sensors in the image plane.

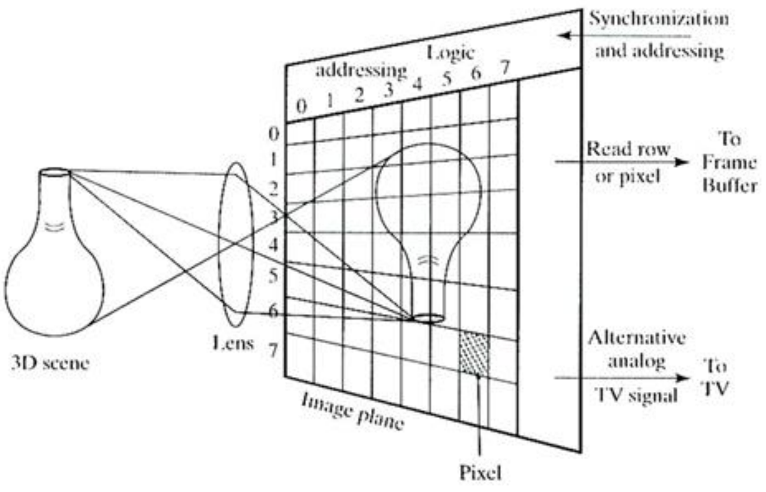


Figure 3-6: Arrangement of sensors in the camera.

Conversion between the real image and pixel image coordinates: Assume that the image center is located at the pixel ($c_{x,}$ c_y) in pixel image. The spacing of the pixels is (s_x, s_y) in some measurement units. Refer to Figure 3-7 for an illustration. Then,

$$x = (x_{im}-c_x)s_x;\ y = (y_{im}-c_y)s_y \qquad (3.3)$$

$$x_{im} = (x/s_x)+c_x;\ y_{im} = (y/s_y)+c_y \qquad (3.4)$$

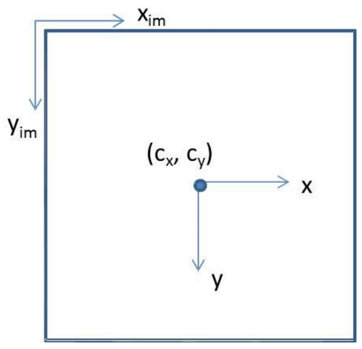


Figure 3-7: Image plane

Below is a representation of the camera projection matrix in terms of matrix multiplication.

$$\begin{pmatrix} f_x \\ f_y \\ z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad (3.5)$$

The matrix describing the mapping is called the camera projection matrix P. Rewriting the above equation,

$$zm = PM \qquad (3.6)$$

Where, M = $(x,y,z,1)^T$ are the homogeneous coordinates of the 3D point and $m=(f_x/z,f_y/z,1)^T$ are the homogeneous coordinates of the image point. The projection matrix P in equation 3.6 represents the simplest possible case, since it only contains information about the focal distance *f*.

3.2.2.2 Camera Parameters

Intrinsic Parameters:

1. Focal distance *f* (in mm).
2. Principal point (image center) coordinates c_x, c_y (in pixel).
3. Width (s_x) and height (s_y) of the pixel footprint on the camera photo-sensor (in mm).
4. Angle θ between axes (usually it is $\pi/2$).

These parameters are needed to relate an image point to a direction in the camera frame. Where $fx=f/s_x$ and $f_y=f/s_y$. Thus, knowledge of the actual values of f and s_x, s_y is not required; just the ratios are required. The lens distortion parameters also need to be considered.

The camera calibration matrix (K) encodes the transformation from image coordinates to pixel coordinates in the image plane.

$$K = \begin{bmatrix} f/s_x & f/s_x \cot\theta & c_x \\ 0 & f/s_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Extrinsic Parameters define the position and orientation of the camera in the real world.R,T are the extrinsic parameters where, T is the position of origin of real world coordinate system and R is the rotation matrix. There parameters can be used to associate points in camera's image space with points in the real world space.

3.3 Image Features

A feature is a piece of information that is relevant for solving a computational task related to an individual application. Computer vision needs image features such as points, edges, and contours for solving problems. In general image features consist of two parts, the keypoint and a descriptor.

The Bridge Monitoring System database has images of thousands of bridges. These images have features that are unique to them, and the key to detecting structural changes is to extract these features so they can be compared. A feature does not have to be something so crucial, it can be something small as a white line on the side of the road, which helps in tracking the position of the vehicle.

The coming section gives a brief description of a few methods that help in extracting and describing Keypoints. Points in an image that can be used to find features in the image are called Keypoints. Each keypoint has a descriptor, which contains information such as pixel intensity, pixel orientation, brightness, nature and distribution of light source.

3.3.1 Image Filtering

An image taken from a camera always has noise. The amount of noise depends on the quality of the sensor in the camera. The principal sources of noise in digital images arise during image acquisition and transmissions. This noise has to be eliminated as much as possible without altering the image. The next section contains a discussion of some methods used for noise reduction [13].

3.3.1.1 Gray Level Transformations (Point transformation)

Gray Level Transformation is a point operation where an input pixel value *r* is mapped to the output value *s*.

$$s = f(r) \tag{3.8}$$

The above function is applied to every pixel in the image independently. Some types of gray level transformations are

Linear Scaling: The function scales a small input range into a wide output range, essentially enhancing the contrast of the values in the input range. Values that are lower than a certain value are mapped to zero and values higher than a particular value are mapped to the maximum. So there is some loss of information in this range.

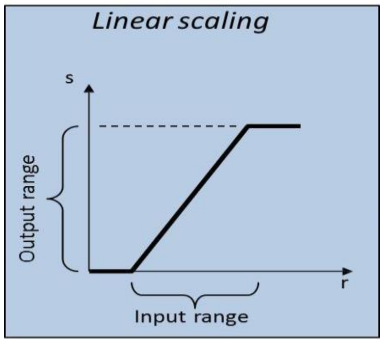


Figure 3-8: Plot of linear scaling function

Square Scaling: This function tends to enhance input intensities at the high level to occupy a larger output range at the expense of values at the lower range which occupy a smaller output range.

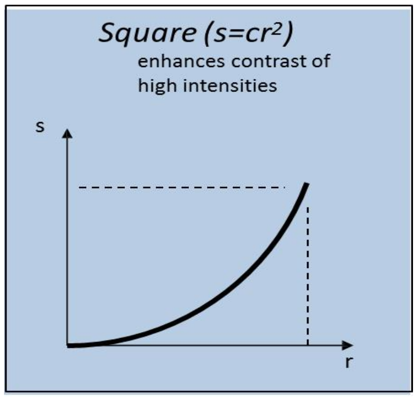


Figure 3-9: Plot of square scaling function

Logarithmic scaling: this is the opposite of square scaling. It enhances the lower values at the expense of higher values.

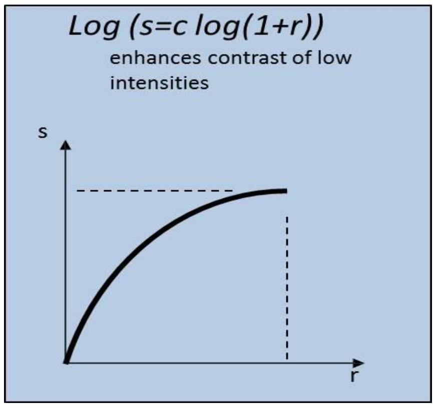


Figure 3-10: Plot of logarithmic scaling function

3.3.1.2 Spatial Filtering

Instead of mapping from one pixel to another pixel, this technique looks at the neighborhood around the pixel. In Figure 3-11, consider a pixel located at (x, y) and create a mask of size m × n around it. Then, do a sum of products of mask coefficients with corresponding pixels under the mask and store this as the output pixel at (x, y). Repeat this for every pixel by sliding the mask over the image. This process is called Cross-Correlation. For example, cross-correlating mask window (*w*) and image (*f*) to produce an output (*g*).

$$g(x,y) = \sum_{s=-\frac{m}{2}}^{\frac{m}{2}} \sum_{t=-\frac{n}{2}}^{\frac{n}{2}} w(s,t) f(x+s, y+t) = w(x,y) \otimes f(x,y) \tag{3.9}$$

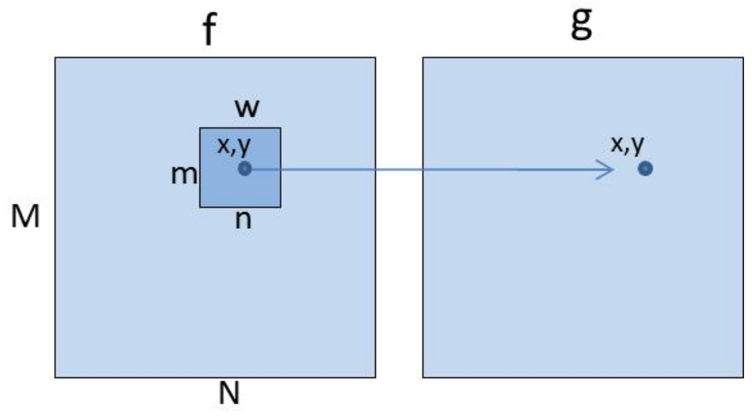


Figure 3-11: Spatial Filtering mask operation

For example, the mask matrix looks like this,

$$M = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \; or$$

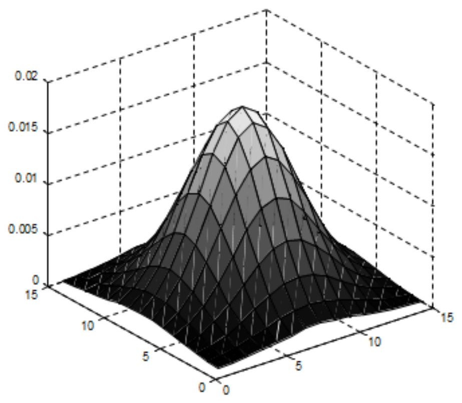
$$M = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

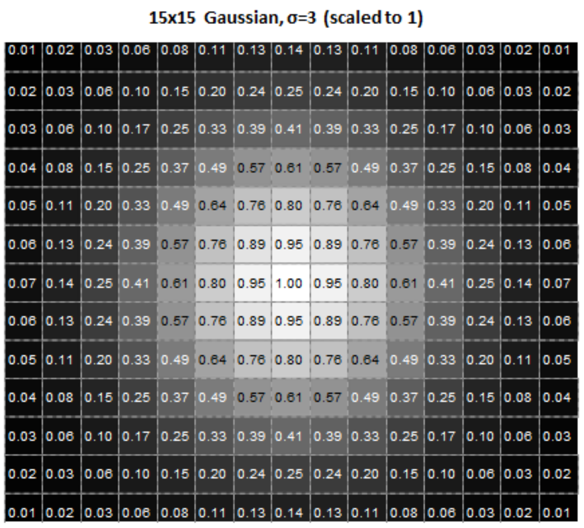
M is a box filter or averaging filter. One drawback of averaging filter is that it intuitively takes out small variations. A nonlinear filter such as a median filter has better performance as compared to an averaging filter as discussed in Appendix A.

A Gaussian filter is better than a box filter because it attenuates high frequencies. The Gaussian filter has the form

$$h(x,y) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad (3.10)$$

where, σ is the standard deviation and is a measure of spread of the Gaussian curve. Equation 3.10 is a normalized form such that the sum of all values is 1. The Gaussian curve is a bell-shaped curve centered at its mean.




15x15 Gaussian, σ=3 (scaled to 1)

0.01	0.02	0.03	0.06	0.08	0.11	0.13	0.14	0.13	0.11	0.08	0.06	0.03	0.02	0.01
0.02	0.03	0.06	0.10	0.15	0.20	0.24	0.25	0.24	0.20	0.15	0.10	0.06	0.03	0.02
0.03	0.06	0.10	0.17	0.25	0.33	0.39	0.41	0.39	0.33	0.25	0.17	0.10	0.06	0.03
0.04	0.08	0.15	0.25	0.37	0.49	0.57	0.61	0.57	0.49	0.37	0.25	0.15	0.08	0.04
0.05	0.11	0.20	0.33	0.49	0.64	0.76	0.80	0.76	0.64	0.49	0.33	0.20	0.11	0.05
0.06	0.13	0.24	0.39	0.57	0.76	0.89	0.95	0.89	0.76	0.57	0.39	0.24	0.13	0.06
0.07	0.14	0.25	0.41	0.61	0.80	0.95	1.00	0.95	0.80	0.61	0.41	0.25	0.14	0.07
0.06	0.13	0.24	0.39	0.57	0.76	0.89	0.95	0.89	0.76	0.57	0.39	0.24	0.13	0.06
0.05	0.11	0.20	0.33	0.49	0.64	0.76	0.80	0.76	0.64	0.49	0.33	0.20	0.11	0.05
0.04	0.08	0.15	0.25	0.37	0.49	0.57	0.61	0.57	0.49	0.37	0.25	0.15	0.08	0.04
0.03	0.06	0.10	0.17	0.25	0.33	0.39	0.41	0.39	0.33	0.25	0.17	0.10	0.06	0.03
0.02	0.03	0.06	0.10	0.15	0.20	0.24	0.25	0.24	0.20	0.15	0.10	0.06	0.03	0.02
0.01	0.02	0.03	0.06	0.08	0.11	0.13	0.14	0.13	0.11	0.08	0.06	0.03	0.02	0.01

Figure 3-12: Gaussian response and normalized values

The 2-D distribution of the Gaussian smoothing filter is used as a point spread function. In other words, it is symmetrical about the mean value and it has only one maximum at the mean value. The width function is directly proportional to the standard deviation, and in effect decides the amount of blurring. As the kernel width is increased, the computational complexity increases. For practical purposes, only three standard deviations are taken from the mean.

Chapter 4 contains a detailed explanation on how filtering and blurring are used to make the system more robust.

3.3.2 Edges

An edge is a point in an image where intensities are changing rapidly. It is a set of connected pixels that form a boundary between two disjoint regions. An edge is an important feature which can be used to detect objects and scenes. A bridges main feature is its edges. A lot of information can be deciphered from looking at an edge such as, length, width and angle[14]. Edge detection is also used tracking vehicle position and is explained later. Several methods exist today for edge detection and some of them are discussed in little detail in the coming sections.

3.3.2.1 Sobel Edge Detector

In Figure 3-13, an edge is represented by the dotted line, and the solid line represents the edge after applying a Gaussian Filter. The first derivative of this curve generates a peak at the edge crossing, and the curve tends to zero before and after the edge. When the second derivative is applied the curve rapidly increases to a maximum and slows down, crosses zero and rapidly decreases to a minimum and increases towards zero. In this case the edge is present at the point where the curve crosses zero in the middle.

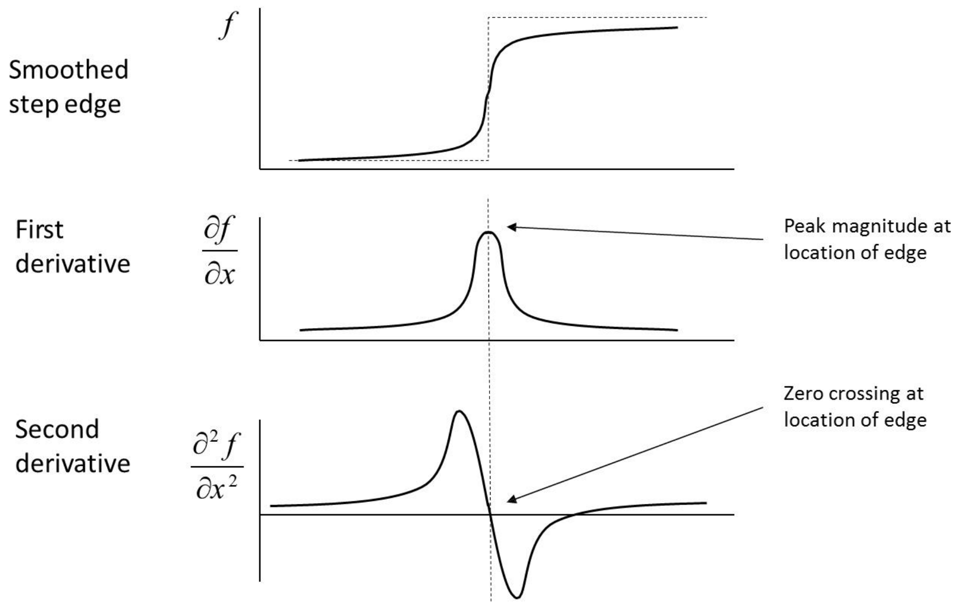


Figure 3-13: Detecting edges using Sobel operator

Going to two-dimensional images, the Sobel operator is used to estimate the derivative in the *x* and *y* directions. A typical Sobel operator would look like Figure 3-14. The masks also do a bit of averaging before doing the derivative as it is a 3 × 3 matrix derived from the product of an averaging and a differentiation kernel. Thus, convolving the mask with underlying pixels of the image, A, produces the gradient with smoothing.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Where $K_x$and $K_y$are the mask operators to compute gradient in x-direction and y-direction respectively. The gradient vector is the vector composed of both the x-gradient and the y-gradient.

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} \qquad (3.11)$$

Where, $G_x = K_x * A$ and $G_y = K_y * A$. Compute the gradient at every pixel and then the magnitude of the gradient. This results in a high magnitude at the x-direction or the y-direction or both and signals the presence of an edge.

$$|\nabla f| = \left[(G_x)^2 + \left(G_y\right)^2\right]^{1/2} \qquad (3.12)$$

The angle of the gradient is computed by taking the arctangent of the x and y derivatives.

$$\theta = \tan^{-1}\left(G_x/G_y\right) \qquad (3.13)$$

The advantage of using the Sobel mask for edge detection is that it provides a smoothing effect along with providing differentiation. It performs well even when the image is noisy. Since the sum of the coefficient of all these masks is zero, it eliminates all the low-frequency components of the image, i.e., when these masks operate in the low-frequency region; the output is close to zero.

3.3.2.2 Canny Edge Operator

Canny Edge Detector is probably the most common and widely used operator for edge detection. The canny edge detector is used in the bridge monitoring system to get rid of extra data from the image. Canny derived the optimal edge operator to find step edges in the presence of white noise, where “optimal” means

1. Sound detection: minimize the probability of detecting false edges and missing real edges.
2. Good localization: detected edges in the image must be as close as possible to the real edges.
3. Single response: marks an edge only once and not be affected by noise. [15]

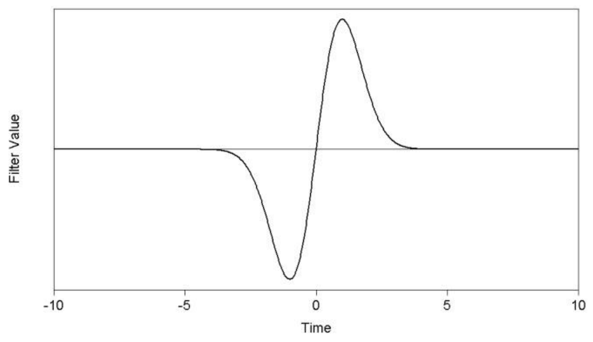


Figure 3-14: First derivative of Gaussian Function

Canny found that a very good approximation to the optimal operator is the first derivative of a Gaussian, in the direction of the gradient. He then suppresses the non-maxima along this direction. The algorithm steps are as follows:

1. Since edge detectors are prone to noise, the image is convolved with the derivative of Gaussian operators $(\partial G/\partial x, \partial G/\partial y)$ for smoothing it.
2. Find the gradient magnitude and direction at each pixel; quantize into one of the four directions (north-south, east-west, northeast-southwest, and northwest-southeast).
3. If the magnitude of the gradient is larger than the two neighbors in the same direction, it is a candidate edge point.

3.3.3 Edge Linking

After detecting edge points, linking each point to its neighbor forms continuous curves or a line. These lines are helpful for object recognition where a long straight line may correspond to the boundary of an object. One of the problems with edge linking is that some edge points along the curve may be weak causing the algorithm to miss these points. If the contrast of the object against the background is less in some places than other places, it will miss these edge points. That results in a broken curve.

The solution for this problem is to use a two-step approach:

1. Using a high threshold ensures that the algorithm captures real edge points.
2. Given those detected points, link additional edge points into contours using a lower threshold, which essentially is a *hysteresis* effect.

The following steps are used to link edges:

1. Find all edge points greater than the high threshold, T_{high}
2. From each strong edge point, follow the chains of connected edge points in both directions perpendicular to the edge.
3. Mark all points lower than the low threshold, $T_{low.}$

3.3.4 Corners

The point in an image where two edges intersect is called a corner. A corner represents a point that has two dominant and different edge directions in the local neighborhood of the point. Hence, the gradient of the image along the two edges have a high variation. This property makes a corner an interest point since it has a well-defined position and can be robustly detected.

3.3.4.1 Harris Corner Detector

The quality of a corner detector is defined by its ability to detect a corner in many similar images, under conditions of varying illumination, translation and rotation. A simple approach to detecting corners in images is using correlation. However, correlation gets computationally expensive. One of the frequently used methods for corner detection is the Harris Corner Detector because of its simplicity and robustness.

The idea behind the Harris corner detector is to localize the point quickly by looking through a small window. As shown in Figure 3-16, shifting a small window in any direction should give a large change in intensity in at least two gradient directions for the point to be classified as a corner. [16]

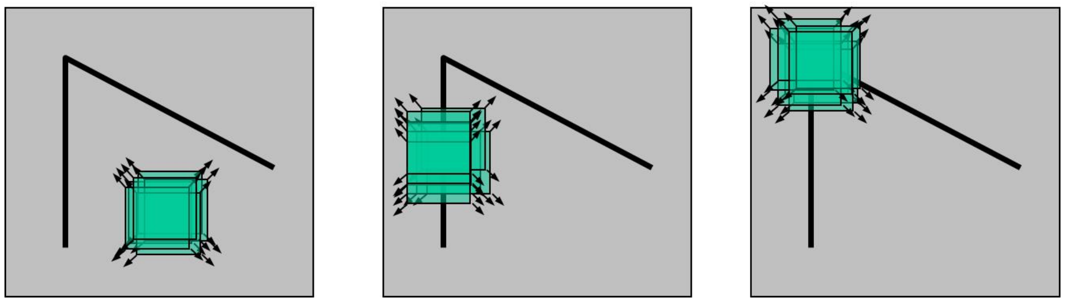


Figure 3-15: Moving window on a flat region, one edge and a corner

Figure 3-17 shows three generic cases; an edge in one direction, a flat region and a corner. Taking the x-derivative of the linear edge produces a strong gradient where the edge is present. The x-derivative of the flat region has no sharp gradient. The x-derivative of the corner also produces a sharp gradient along one edge. Now, taking the y-derivative, linear edge and flat image do not produce a sharp gradient. However, the corner creates a gradient along an edge.

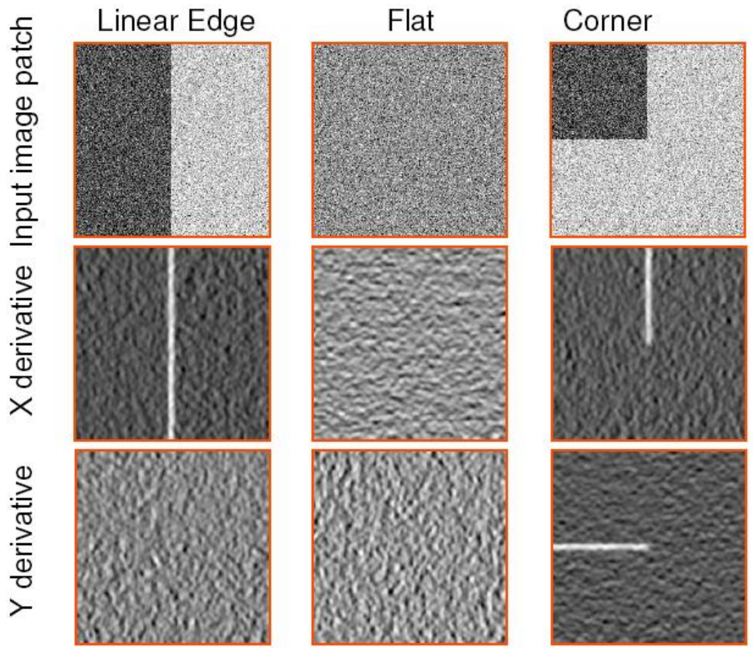


Figure 3-16: Gaussian response comparison

The distribution of Eigenvalues for areas with no edges, one edge and a corner is shown in Figure 3-18.

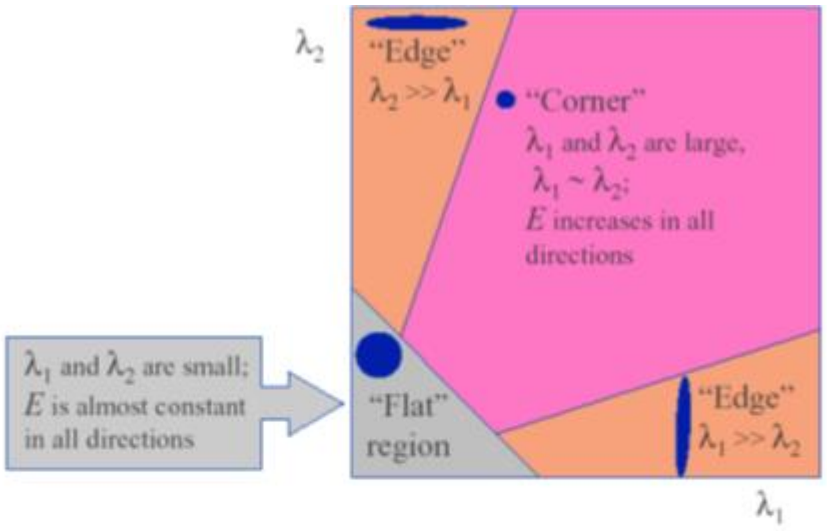

λ_2
"Edge"
$\lambda_2 >> \lambda_1$
"Corner"
λ_1 and λ_2 are large,
$\lambda_1 \sim \lambda_2$;
E increases in all directions
λ_1 and λ_2 are small;
E is almost constant in all directions
"Flat" region
"Edge"
$\lambda_1 >> \lambda_2$
λ_1

Figure 3-17: Eigenvalue distribution

The window averaged change of intensity by shifting the image data by [*u,v*] is represented as

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \qquad (3.14)$$

Where, w(x,y) is a Gaussian function and *I* is the intensity. Considering small shifts by Taylor’s expansion,

$$E(u,v) = \sum_{x,y} w(x,y)\left[I_x u + I_y v + O(u^2 v^2)\right]^2 \qquad (3.15)$$

$$E(u,v) = Au^2 + 2Cuv + Bv^2$$

Where,

$$A = \sum_{x,y} w(x,y) I_x^2(x,y)$$

$$B = \sum_{x,y} w(x,y) I_y^2(x,y)$$

$$A = \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y)$$

A bilinear equation gives the Taylor series approximation,

$$E(u,v) \cong [u,v]\, M \begin{bmatrix} u \\ v \end{bmatrix} \qquad (3.16)$$

M is a 2x2 matrix computed from image intensity derivatives,

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \qquad (3.17)$$

The Eigen Values are computed from this matrix by solving for its trace and determinant.

$$\det(M) = \lambda_1 \lambda_2$$

$$\mathrm{trace}(M) = \lambda_1 - \lambda_2$$

To determine the direction of the edges, look at the eigenvectors. They point along prominent directions that many data points lie on. The corresponding eigenvalue shows the magnitude of the gradient in that direction.

3.4 Control System

3.4.1 Introduction

Control systems are intimately related to the concept of automation. The two fundamental types of control systems are feedforward and feedback. The bridge monitoring system uses a feedback control system to maintain the stability of the camera platform. The system uses a Proportional-Integral-Derivative (PID) controller to manage motors that rotate the platform in the roll and pitch axis. The feedback information about the movement of the platform is provided by an Inertial Measurement Unit (IMU) which measures orientation and acceleration.

The stabilizer has two fundamental characteristics:

1. The amount of rotation is varied by using the power from a local source rather than a signal. Thus, the operation of moving the motors does not load or distort the signals on which the accuracy of control depends on.
2. The rate at which energy is fed to the motor to effect variations in the angle depends on the present and desired angle.

A control system possessing these fundamental properties is called a closed-loop control system.

3.4.2 PID Controller

The control system in the bridge monitoring system uses motors to rotate the platform. To be able to control the motors precisely a Proportional-Integral-Derivative controller (PID controller) is required. A PID controller is a closed loop feedback controller used in many control system environments. The controller calculates the error value as the difference between the actual value of the angle and the desired value. The controller minimizes the error by adjusting the process that changes the variable. [17]

The three values used to minimize error can be interpreted in terms of time as: Proportional Term (*P*) depends on the present error, Integral Term (*I*) depends on the accumulation of past errors, and Differential Term (*D*) is a prediction of future errors based on current rate of change. The sum of these three is used to adjust the process.

The controller output is given by

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t) \qquad (3.18)$$

3.4.2.1 Proportional Response

The proportional component depends only on the difference between the desired value and the control variable. This difference is referred to as the error term. The proportional gain (K_p) determines the ratio of the output response to the error signal. In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate. If K_p increased further, the oscillations would become larger, and the system will become unstable and may even oscillate out of control.[18] The proportional term is given by

$$P_{out} = K_p e(t) \qquad (3.19)$$

3.4.2.2 Integral Response

The integral component sums the error term over time. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless error is zero, so the effect is to drive the Steady-State error to zero. Steady-State error is the final difference between the process variable current and desired value. A phenomenon called integral windup results when integral action saturates a controller without the controller driving the error signal toward zero.[18] The Integral term is given by

$$I_{out} = K_i \int_0^t e(t) d\tau \qquad (3.20)$$

3.4.2.3 Derivative Response

The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable. Increasing the derivative time (T_d) parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response. Most practical control systems use tiny derivative time (T_d), because the derivative response is highly sensitive to noise in the process variable signal. If the sensor feedback signal is noisy or if the control loop rate is too slow, the derivative response can make the control system unstable.[18] The derivative term is given by

$$D_{out} = K_d \frac{d}{dt} e(t) \qquad (3.21)$$

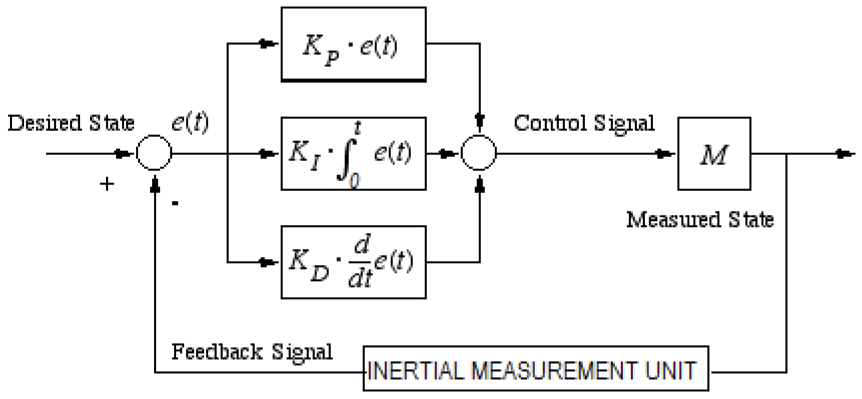


Figure 3-18: PID Operation

The idea is to use data from an IMU (Inertial Measurement Unit) to control two brushless DC motors connected to the camera platform. These motors will rotate the platform in the pitch and roll axis to keep it stable.

3.4.3 Gimbal

A gimbal is a pivoted support that allows the rotation of an object about a single axis. When three gimbals are fixed orthogonal to each other, the setup can be used to allow an object mounted on the innermost gimbal to remain independent of its support rotation. For example, these are used on ships on drink holders to keep the drinks upright even when the ship is pitching and rolling.

A similar concept is used to stabilize the camera platform using a PID controller and two motors. This is just set up to stabilize the camera along roll and pitch axes.

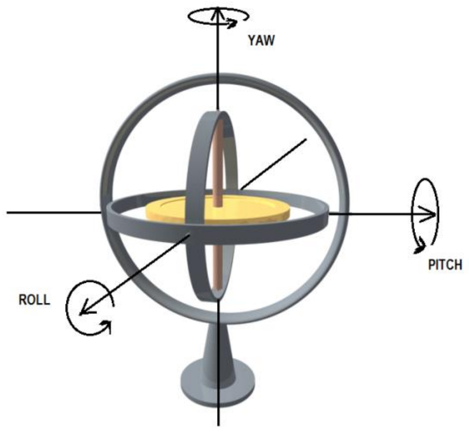


Figure 3-19: Gimbal

3.4.4 Brushless DC Motor (BLDC)

The BLDC Motor, unlike regular motors, has no brushes. Instead, BLDC motor has the permanent magnets glued on the rotor. It usually has three magnets around the perimeter. The stator of the motor comprises of electromagnets, placed in a cross pattern with $90°$ angle between each electromagnet. An advantage of a BLDC motor is that it can be made smaller and lighter than a brush type with the same power output. The downside is they need electronic management units, like a microcontroller using input from sensors indicating position of the rotor to control the stator coils. Whereas, the system can achieve accurate speed and torque control as well as ensure that motors are running at peak efficiency[19]. For a more detailed explanation of how the BLDC works, refer to Appendix B.

Chapter 4

IMPLEMENTATION OF THE BRIDGE MONITORING SYSTEM

The previous chapters provided insight into a few computer vision fundamentals and algorithms which are essential for this project. This chapter provides information on how the components put together form the bridge monitoring system.

The process contains five parts; setting up the control system, tracking vehicle position, compensating for scale variance, feature extraction and matching, and detecting structural changes.

4.1 Camera Stabilization

The need for getting the visual data of the bridge structure with minimum jitter and change is of utmost importance. A control system mounted on the vehicle controls the movement of the camera platform. The movement of the vehicle is negated by rotating the camera platform. The control system calculates the direction, amount and speed of rotation required to negate the shift. In this project, a programmable PID controller is used to control the motor.

The platform needs to be stabilized in real time so as to collect accurate video information. A system that can cope with the frame rate of the camera is suitable for this purpose. Assuming that the camera is running at 30 fps, the platform must become stable in less than 33.33 milliseconds. There are several controllers in the market that suit the needs of this project. The RCTimer Brushless-Gimbal (BruGi) V1 board is used in this project. It has an Future Technology Devices International (FTDI) USB controller chip. This board requires the FTDI 2.8.24 Windows driver to communicate with the processing unit running a Windows operating system. It comes completely pre-soldered with all the pin-outs needed. This system has the capability to react quickly to changes in real time.

Figure 4-1 shows the connection diagram for the motors, IMU and the BruGi board.

Figure 4-2 shows the complete setup with the camera in place.

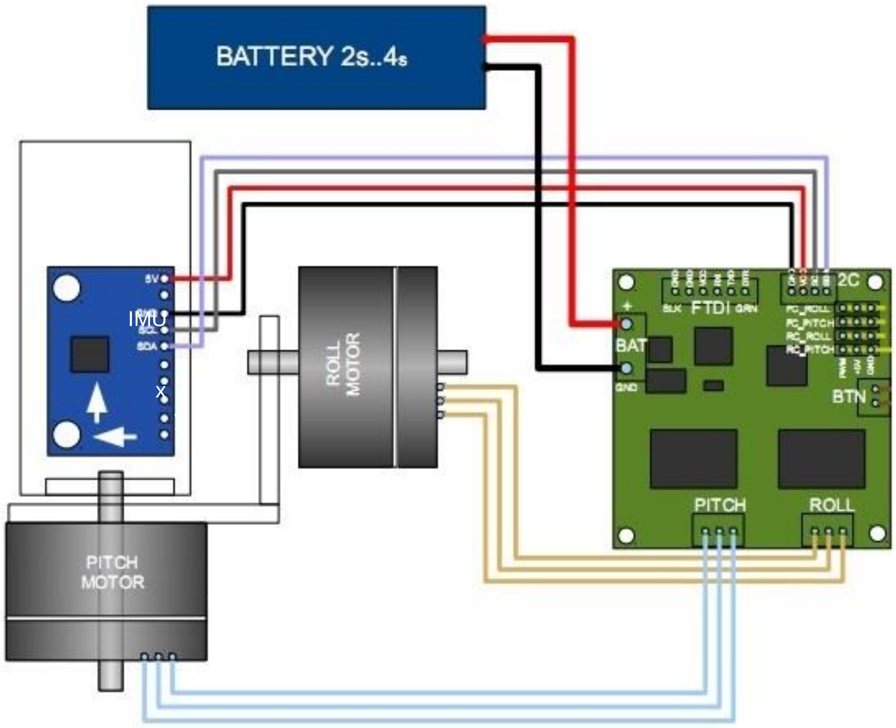


Figure 4-1: Connection diagram for 2-Axis Brushless Gimbal Controller

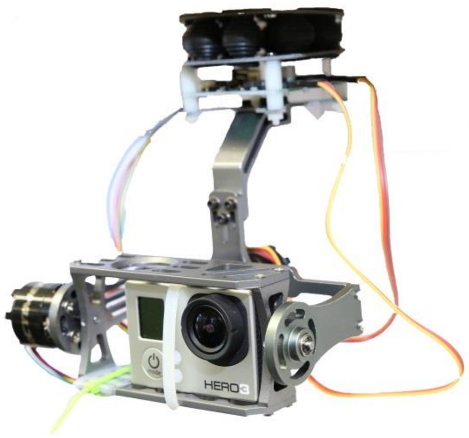


Figure 4-2: 2-Axis Gimbal with the recording unit.

4.2 Vehicle Position Tracking

As mentioned in the previous chapters, keeping track of the position of the vehicle with respect to the road is important because the camera pose depends on the vehicle position. Hence, maintaining the same vehicle track on every run is crucial. There are a few ways to determine the vehicles relative position on the road. The simplest and fastest way is to detect solid white line on the road.

4.2.1 Hough Line Transform and Line Detection

In this project, the Hough line transform is used to detect lines. Before running the Hough line transform, running an edge detection algorithm such as the Canny Edge detector is desirable. Canny algorithm finds gradients on the image that has sharp changes in the pixel intensities. These are likely contours, and the output is just a binary map that shows the location of contours in the image. In an image, a line can be expressed in two forms, Cartesian coordinate system (*x,y*) or Polar coordinate system (*r,θ*). [13]

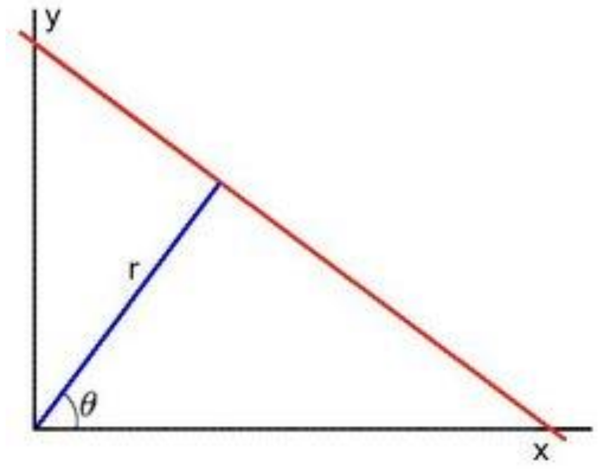


Figure 4-3: Representation of a line

In Polar Coordinates, the line is represented as

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right) \tag{4.1}$$

Or,

$$r = x\cos\theta + y\sin\theta \tag{4.2}$$

If at each point (x_0,y_0), a family of lines that go through that point can be defined as

$$r_\theta = x_0\cos\theta + y_0\sin\theta \tag{4.3}$$

Here, each pair (r_θ,θ) represents a line that passes through (x_0,y_0). For a given point, plotting the family of lines that goes through it, produces a sinusoid. Consider only points such that $r>0$ and $0<\theta<2\pi$.

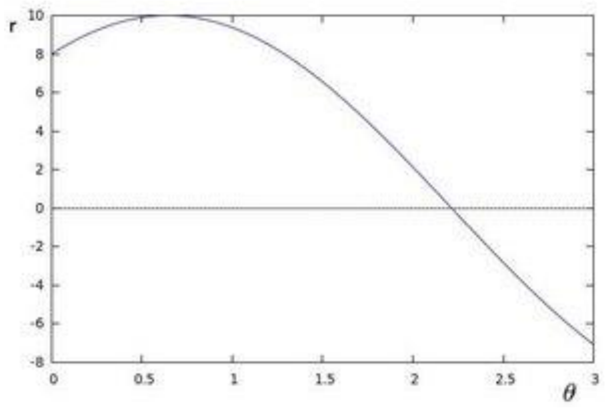


Figure 4-4: Plot of Hough Transform of a point

Repeating this operation for all the points in an image produces multiple sinusoids. If the curves of two different points intersect at (r,θ) that means both those points are on the same line. Thus, lines can be detected by finding the number of intersections between curves. If more curves intersect at a point in the graph, it indicates that the line has more points in it. In this algorithm, the threshold is defined as the minimum number of intersections needed to detect a line. If the number of intersections is above some threshold, then it declares it as a line.

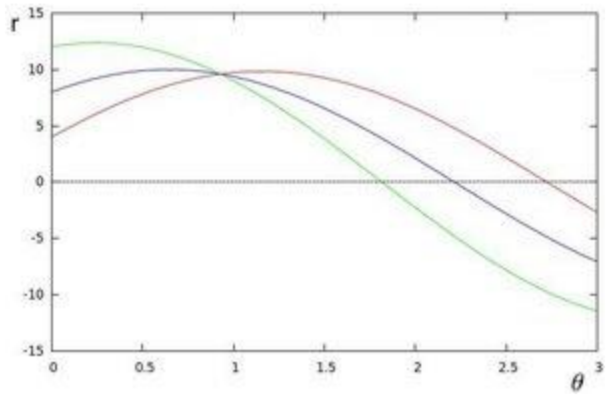


Figure 4-5: Plot of Hough Transform for multiple points

At this point, the algorithm detects many possible lines in the image, out of which one of them is a solid white line marking the edge of the road. Since, the PMIS survey vehicle drives on the extreme right lane the solid white line is present only in the bottom right of the corner. Taking the bottom right as the Region of Interest (ROI) in the image helps the algorithm shortlist the candidate lines. When driving on the right lane, it appears that the solid white line is closest to the vehicle on the right-hand side.

4.2.1.1 Finalizing Vehicle Position

The above procedure describes how to detect the solid white line on the road. Since the camera is mounted exactly in the middle of the car, the vertical center of every frame captured by the camera should represent the middle of the car. The distance of the candidate lines from the center of the image (center of the vehicle) can be calculated and the line closest to the center is chosen. After careful observation, it seems that when driving on the right lane the solid white line is closest to the vehicle. The white line acts as a reference marker in every run the vehicle makes.

4.2.2 Shortcomings of this method

Even though this algorithm serves the purpose it will run into problems in some uncertain situations. To state a few:

1. The automobile driver needs to be very precise, which is not possible in every run and will prove to be a problem. To solve this problem, the camera platform needs minor adjustments to keep it in the middle. A small system with a motor and gear system will help make these adjustments.
2. In case, there is some roadwork on the side of the road and the solid white line is not visible, then this algorithm fails.
3. If snow covers the line, the algorithm fails. However, according to TxDOT the data collection happens in the spring season.

4.3 Image Feature Extraction and Matching

At this point, the position of the Vehicle/Camera is assumed to be the same on every run of the test vehicle. Now the scene frame that matches the reference object frame is extracted from the video. In the very first run, the TxDOT vehicle collects bridge data for future reference. These videos contain many frames that are suitable to serve as a reference. Any of the frames can be manually handpicked and stored as a reference image. This image is referenced for matching and analysis of changes in the bridge structure in the future. The data from the second run is analyzed against data from the first run. The data from the third run is analyzed against data from the second run, and so on.

The best image match is found by matching unique local features in the scene and object images. The features are identified by using a very strong feature extraction algorithm called SIFT (Scale Invariant Feature Transform).[20] These features help in finding the matching scene frame from the video. The procedure consists of three steps:

1. Video synchronization: the recording of the video is synchronized to the previous reference data with the use of some reference markers or TRM data.
2. Feature Extraction: local features are extracted from each scene frame.
3. Feature Matching: every frame is checked for similarities with the reference image and the one with the maximum number of matches is selected and analyzed.

4.3.1 Scale Invariant Feature Transform (SIFT) Algorithm

The goal of SIFT is to extract distinctive, image scale and rotation invariant features that can be correctly matched against a large database of features from many images. It is an algorithm that shows robustness towards affine distortion, change in viewpoint, addition of noise, and change in illumination. This detection should ideally be possible when the image shows the object with different transformations, mainly scale and rotation, or when parts of the object are occluded.

The process consists of three overall steps:

1. Detection: Automatically identify interesting features, interest points. The detection must be done robustly; i.e., the same feature should always be detected regardless of viewpoint. Creating scale space and finding local extrema of the images helps in finding such features. From these extrema, important keypoints are selected.

2. Description: Each interest point should have a unique description that does not depend on the features scale and rotation. For each key point, descriptors are calculated by finding histograms of gradient directions and creating a feature vector out of the histograms.
3. Matching: Given an input image, determine which objects it contains, and possibly a transformation of the object, based on predetermined interest points.

The advantages of SIFT are

1. Locality: features are local, meaning they are applicable to a neighborhood of pixels. This property makes it robust to occlusion and clutter.
2. Distinctiveness: individual features can be matched to a large database of objects.
3. Quantity: even small objects sometimes generate many features.
4. Efficiency: the algorithm exhibits close to real-time performance.

4.3.1.1 Creating Scale Space Images

Scale space is used for handling image structures at different scales. The scale-space of an image is represented as a one-parameter family of smoothed images. The scale of smoothing depends on one parameter, the size of the smoothing kernel (σ) used for suppressing fine-scale structures. The primary type of scale space is the Gaussian scale space. The Gaussian scale-space constitutes the canonical way to generate a linear scale-space, based on the essential requirement that new structures must not be created when going from a fine scale to any coarser scale.

To create a scale-space, a series of progressively blurred out images are created from the original image. These blurred images form an Octave. The next step is to resize the original image to half of the current size and repeat the process to derive another Octave.

4.3.1.2 Laplacian of the Gaussian (LoG)

The Laplacian of the Gaussian is found for the image by varying the scale size (σ) values. LoG acts as a blob detector which detects blobs of sizes that are the same as σ. Due to variation in σ the LoG finds blobs of different sizes. The scale size σ of the LoG filter determines the scale of the blob. So, this process finds the local maxima and minima across the scale space and gives a list of (x,y,σ). This list contains the location of potential keypoints at (x,y) of σ scale.

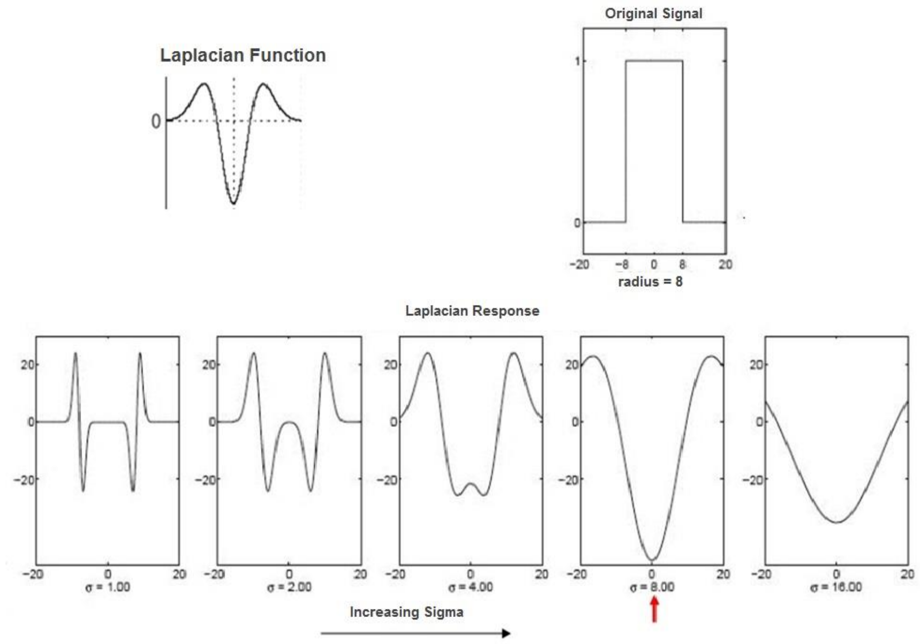


Figure 4-6: Laplacian operator curve and its responses

Figure 4-6 shows the response of a signal when applied with the given Laplacian function. The given signal has a radius of 8 units. The response curves are shown for varying σ values. Notice that when the σ value is same as the signal radius the peak is at a maximum. The response shows the scale at which this feature point can be detected and the location in the image.

Figure 4-7 shows how these blobs of maxima and minima look in an image. This example uses a value of σ = 2.

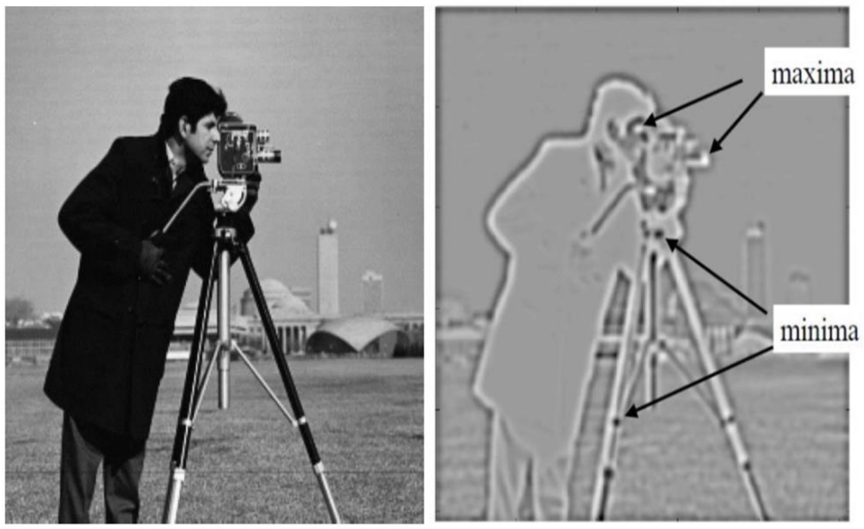


Figure 4-7: Illustration of LoG operation

4.3.1.3 Difference of the Gaussian (DoG)

For real-time applications, the LoG operation is not suitable. However, a close approximation of the LoG is the DoG. It is obtained by the difference of Gaussian blurring of an image with two different σ. Take, for example, the following differences (σ-kσ), (σ-k^2σ), (σ-k^3σ), (σ-k^4σ), (σ-k^5σ). An illustration of this example is shown in Figure 4-8. The difference of two consecutive blurred images is taken to produce a DoG image. This DoG image is later scouted for extrema.

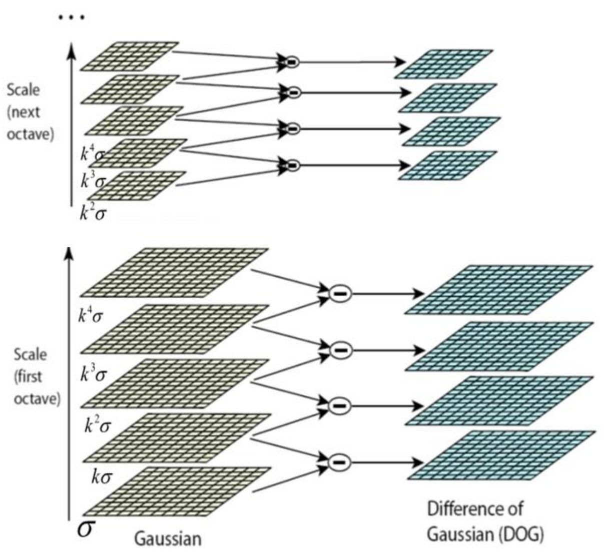


Figure 4-8: Creating DoG Scale Space

4.3.1.4 Scale Space Peak Detection

After calculating the DoG of all Octaves, images are scouted for local extrema over a scale and space. For example refer figure 4-9, compare the pixel marked X with the current and adjacent scales (green circles). The pixel X is selected if it is the largest or the smallest out of all the 26 pixels surrounding it. If it is a local extrema, it is a potential keypoint. It means that keypoint is best represented in that scale.

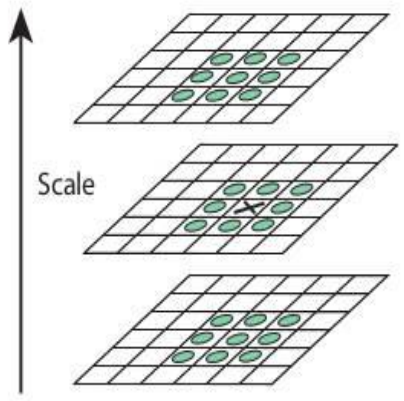


Figure 4-9: Illustration of the 26 pixels considered

4.3.1.5 Keypoint Localization

Once potential keypoint locations are found, calculating the extrema proves to be computationally expensive because of an excessive number of keypoints produced by DoG. Hence, the keypoints have to be sampled to detect the most stable subset with a coarse sampling of scales.

Initial rejection of keypoints is done using a Taylor series expansion of the scale space to get an accurate location of the extrema. If the intensity at this extrema is less than a threshold, it is rejected. The Taylor series expansion of the DoG, D for a point x = $(x,y,\sigma)^T$ is given by

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \qquad (4.4)$$

The minima or maxima is located at

$$\hat{x} = -\frac{\partial D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \qquad (4.5)$$

In this project, the DoG is used instead of the LoG, because edges have higher response than that that of the LoG. The edges are removed for better results. By using a concept similar to the Harris corner, detector the edges are removed to reduce the number of keypoints even further. The Principal Curvature (PC) along the edge is very low, and across the edge is very high. They used a 2x2 Hessian matrix (H) to compute the PC. The PC is given by the eigenvalues of the gradient at the edge point.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{4.6}$$

$$\mathrm{Trace}(H) = D_{xx} + D_{yy} = \lambda_1 + \lambda_2$$

$$\mathrm{Det}(H) = D_{xx}D_{yy} - \left(D_{xy}\right)^2 = \lambda_1\lambda_2$$

The points are rejected using the ratio,

$$\frac{\mathrm{Trace}(H)^2}{\mathrm{Det}(H)} = \frac{(r+1)^2}{r}$$

$$r = \frac{\lambda_1}{\lambda_2} \tag{4.7}$$

According to the Harris corner detector, in edges, one eigenvalue is larger than the other. If the ratio is greater than a threshold, that keypoint is discarded. So it eliminates any low-contrast keypoints and edge keypoints and what remains is high-interest points.

4.3.1.6 Orientation Assignment

After selecting the keypoints, an orientation is assigned to it so as to achieve rotation invariance. Depending on the scale, a neighborhood around the keypoint location is selected. Central derivatives are computer giving the gradient magnitude (equation 4.8) and direction of the smooth image (L) (equation 4.9) in the neighborhood of a keypoint (x,y,σ).

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \tag{4.8}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \tag{4.9}$$

A histogram is created using the gradient directions of the keypoints. The orientation of this neighborhood is decided by selecting the direction of the highest peak in the histogram. Any peak above 80% of the highest is also considered to calculate the orientation. This orientation assignment contributes to the stability of matching. [20]

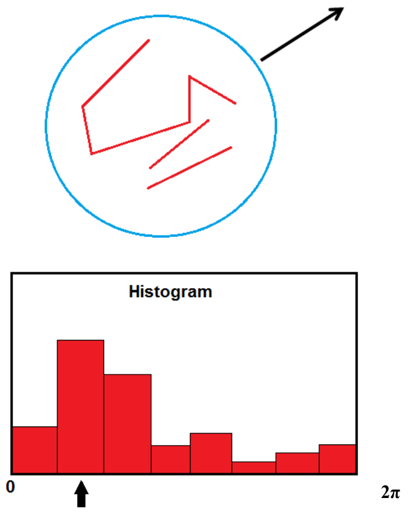


Figure 4-10: Computing Orientation of the window

4.3.1.7 Creating the Keypoint Descriptors

Now that the keypoint descriptor is created, a 16x16 neighborhood around the keypoint is divided into 16 sub-blocks of 4x4 sizes. For each sub-block, an 8-bin orientation histogram is created. So a total of 128 bin values are available. The 16 histograms are concatenated to the keypoint descriptor vector or feature vector. This vector contains 128 descriptor values.

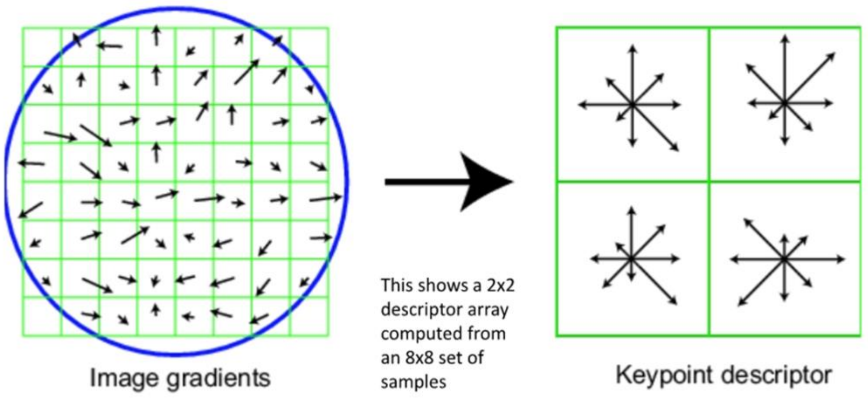


Image gradients
This shows a 2x2 descriptor array computed from an 8x8 set of samples
Keypoint descriptor

Figure 4-11: Illustration of a 2x2 Keypoint Descriptor Vector

For example, any gradient in 0-44 degrees is added to the first bin, 45-89 degrees is added to the next bin, and so on. Also, the number of points added depends on the distance from the keypoint. So, gradients that are farther away from the keypoint will add smaller values to the histogram. The feature vector is normalized to the unit vector to make it invariant to illumination (affine changes). For non-linear intensity transforms the unit vector is bound to a maximum of 0.2 (removing larger gradients), and renormalize the unit vector.

4.3.2 Scene Selection from the Video

Once the keypoints and descriptor vectors are obtained for each frame, finding the scene frame that matches the object reference image is the next step. A matching algorithm is run on each scene frame against the object. Two algorithms commonly used for feature matching are discussed below.

4.3.2.1 Brute Force Matcher

Brute Force Matcher is the simplest matching algorithm. It takes the descriptor of one feature in the first set and matches with all other features in the second set. There may be more than one close match at this point. To decide which keypoint is the best match, Euclidian distance between the two matching points is calculated, and the pair of points with the least distance is selected as the closest match.

4.3.2.2 FLANN Based Matcher

Fast Library for Approximate Nearest Neighbor (FLANN) contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and high-dimensional features. It works faster than Brute Force Matcher for large datasets. [21]

The Algorithms used in FLANN are:

1. K-D Tree: K-D stands for K-Dimension. It is a type of the binary tree for multi-dimension vectors. The tree is balanced when built by splitting the nodes at the median values. The dimension that could divide the samples into half (largest variance) at each level is chosen. The tree is built with a training set of feature vectors. It is used to find query specific value or value ranges and nearest neighbors.
2. Randomized K-D Tree: Improve the approximation of nearest neighbors from the query point by searching simultaneously across a number of randomized trees. The tree is built from the same set of samples.
3. Hierarchical K-Means Tree: It is a tree in which every inner node is split in K-ways. K-means clustering is used to classify the data subset at each node. An L level tree would have approximately K^L leaf nodes.

In this project, brute force matching is suitable since the system is not dealing with a large dataset. Running the matching algorithm, gives an idea about the number of matches in each frame. The image with the largest number of matches is selected as the Final Scene Image. This scene image and corresponding feature vector are stored in the database and forwarded to the next stage of this project; i.e., detecting and measuring changes in the bridge structure.

Chapter 5

EXPERIMENTS AND RESULTS

5.1 Testing the Gimbal

The gimbal with the RCTimer Brushless Gimbal Controller is well known for its real-time performance. It reacts well to sudden movements of small magnitude. If the movement is large, the gimbal loses stability and regains it after one or two seconds.

5.2 Testing the Line Detection Application

5.2.1 Robustness and Accuracy

Figure 5-1 shows the region of interest (ROI) in the image. The ROI is the area in the frame where the application scouts for lines. From observation, it is determined that the white line appears in this area.

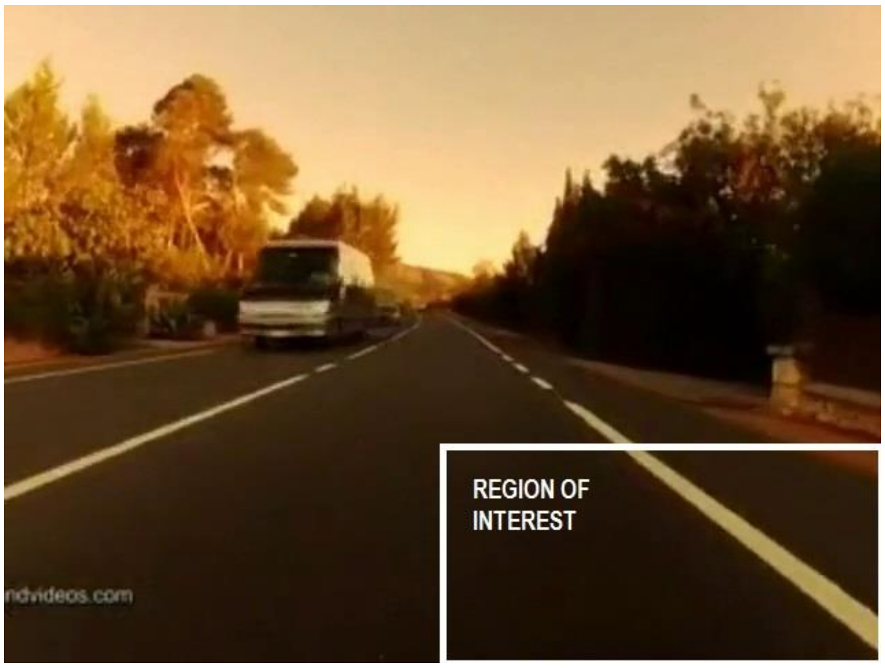


Figure 5-1: Image Region of Interest

Figure 5-2 shows the detection of the solid white line on the road under normal conditions where there is nothing blocking it. Embedded in this figure is the Canny edge detection output for the ROI. Blue line indicates the detected white line. Red line denotes the distance.

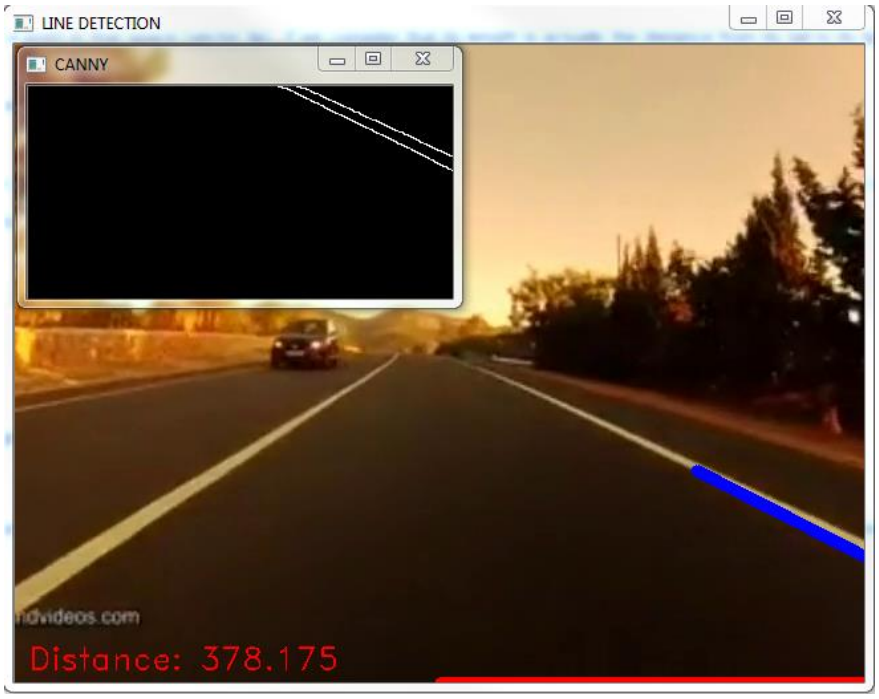


Figure 5-2: General case line detection

Figure 5-3 shows how the Euclidean distance is calculated. This information can be used for tracking the vehicle's relative position on the road on each Pavement Management Information System (PMIS) survey. Figure 5-4 shows that the algorithm works even when the solid white line is broken.

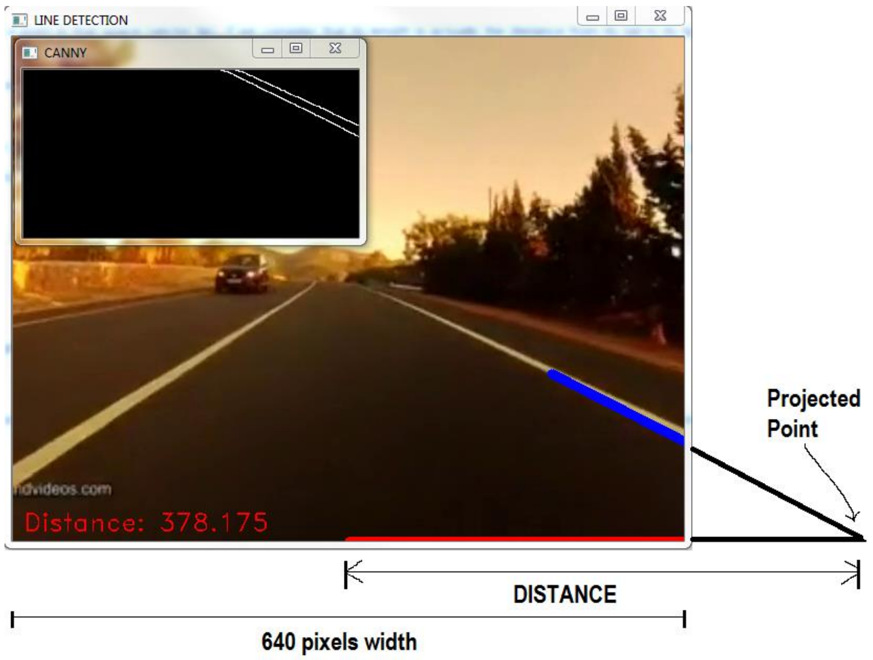


Figure 5-3: Illustration of distance calculation

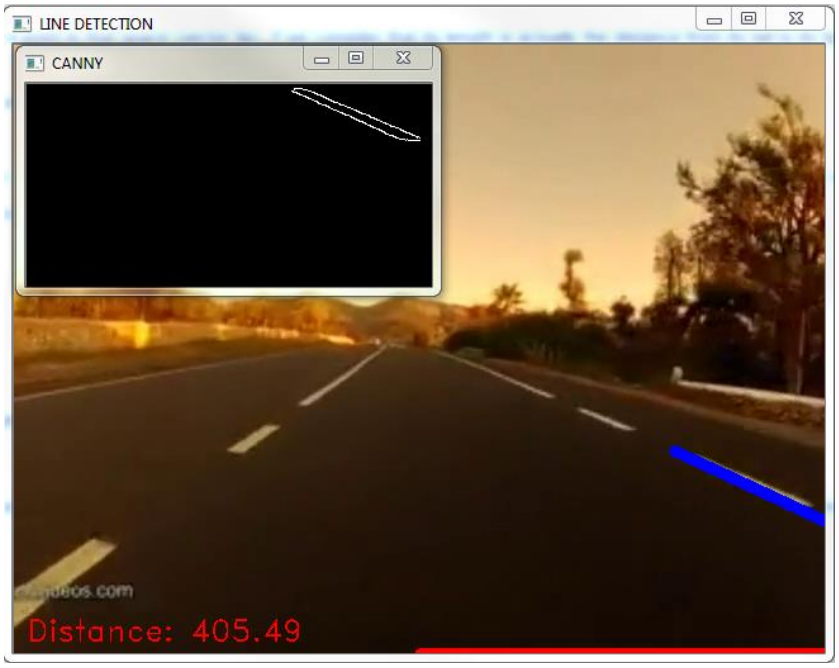


Figure 5-4: Broken line detection

Figure 5-5 shows an example of line detection in another video captured from the helmet cam of a cyclist. Throughout the video, the cyclist moves his head randomly. The algorithm can detect the white line precisely even in this case. The video footage from the PMIS vehicle is much more stable, and random movement is taken care of by the Camera Platform Stabilizer.

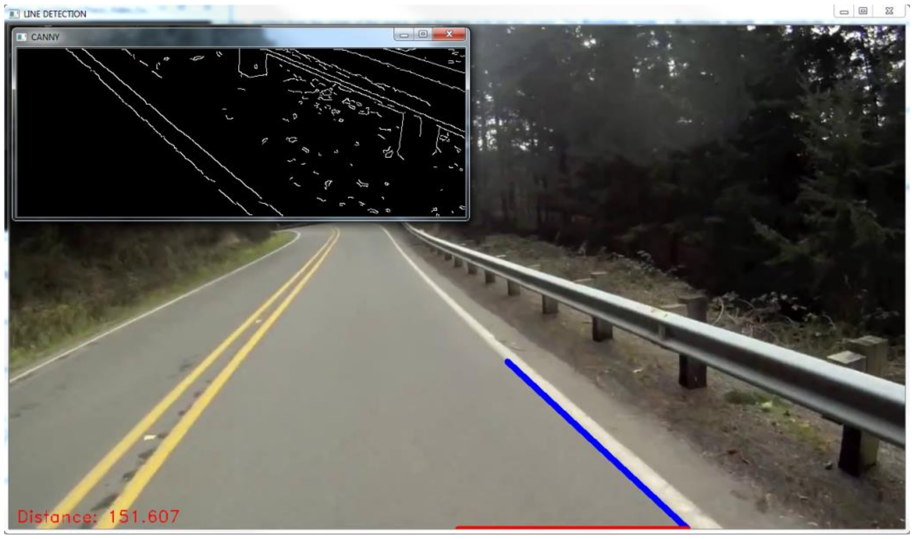


Figure 5-5: General case (video 2)

Figure 5-6 shows a case where the cyclist turns his head to the left such that the Solid White line on the other side of the road appears to be closest to the center.

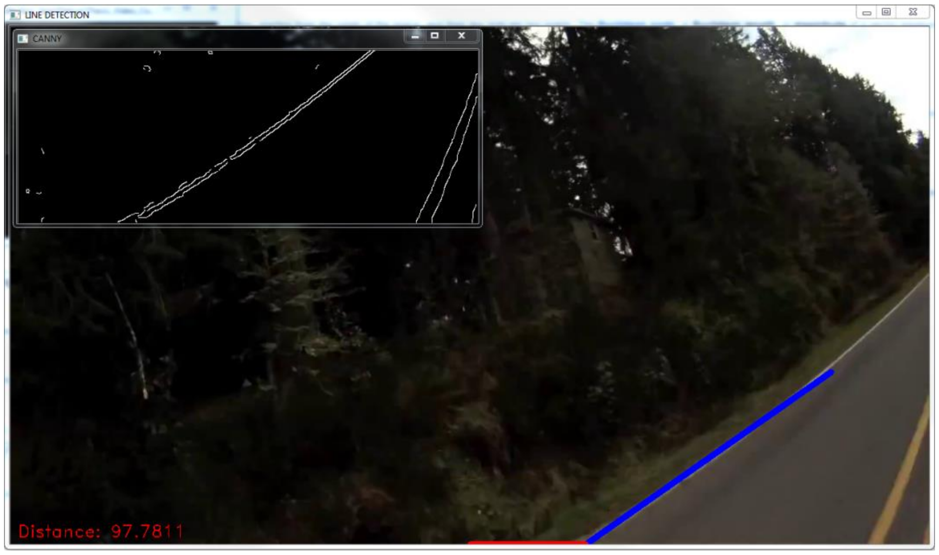


Figure 5-6: Detects line closest to the center of the image

Figure 5-7 shows the detection of the white line with an obstacle blocking the view partially. The algorithm picks up the line even when there is some hindrance.

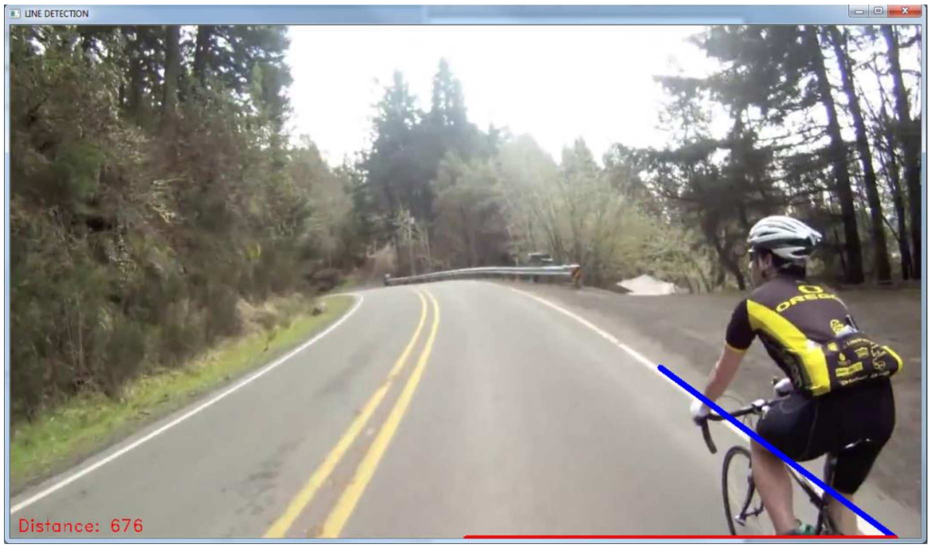


Figure 5-7: Line Detection with an obstacle in the way

5.2.2 Processing Time

The average amount of time taken to capture and store one frame is roughly 0.0333 seconds, which is close to capturing every frame (720x420 pixels) that the camera passes at 30 frames per second.

The average amount of time taken to capture, detect line and store one frame (720x480 pixels) is roughly 0.0624 seconds. This causes the program to miss half of the incoming frames. Currently, the application is not threaded and is running as a serial program. Real-time performance can be achieved by parallelizing the reading, line detection and storing tasks.

5.3 Testing the Feature Detection and Matching Application

5.3.1 Feature Detection

The Feature Detector extracts keypoints and computes their descriptors using the Scaled Invariant Feature Transform Algorithm. For experimental purposes, a frame was chosen at random from the test video to be the reference image for the Feature Matching part. Figure 5-8 shows the output of the Feature Detection algorithm when the input is the reference frame. The center of each circle is a keypoint, the radius of the circle is the scale of the keypoint, and the direction of the line denoting the radius is the orientation of that keypoint. For this frame, the algorithm detects 196 unique keypoints and computes descriptor array of size 128 for each keypoint.

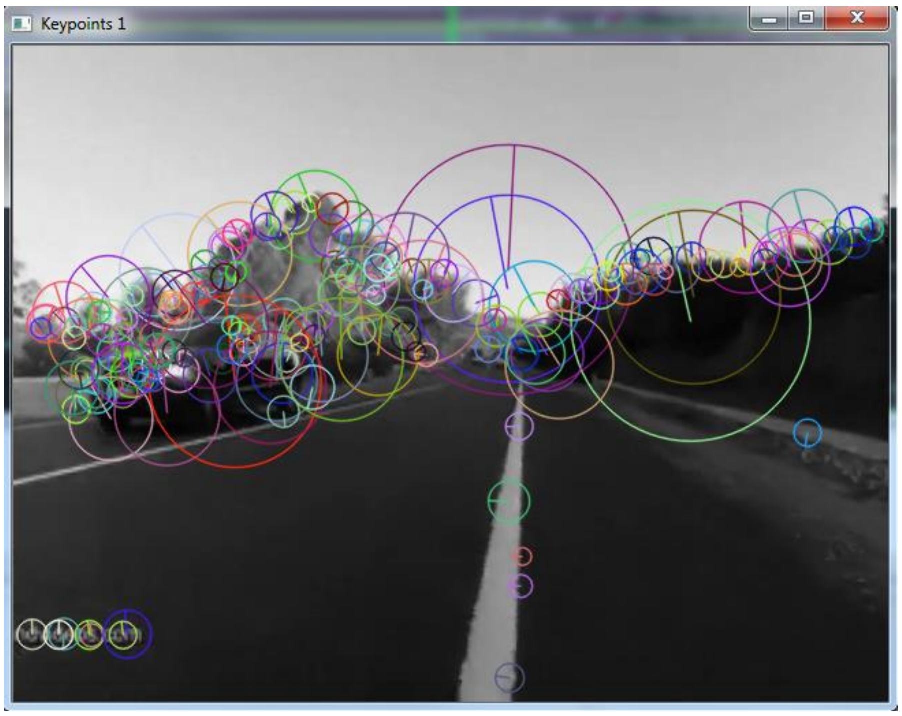


Figure 5-8: Keypoints of the reference image. Keypoints with similar scale have the same color.

5.3.2 Feature Matching

The feature matching algorithm uses a Brute Force Matcher to compare frames from the incoming video to the above Reference Image. The algorithm detects keypoints and computes descriptors for each frame and compares it with the descriptors of a reference image. The similarities between these images are called a matches. If the current frame has the maximum number of matches, the index of this frame is noted. If another frame appears to have the maximum number of matches then, its index replaces the previous match. After going through all the frames the image that had the maximum number of matches, its keypoints and descriptors are returned.

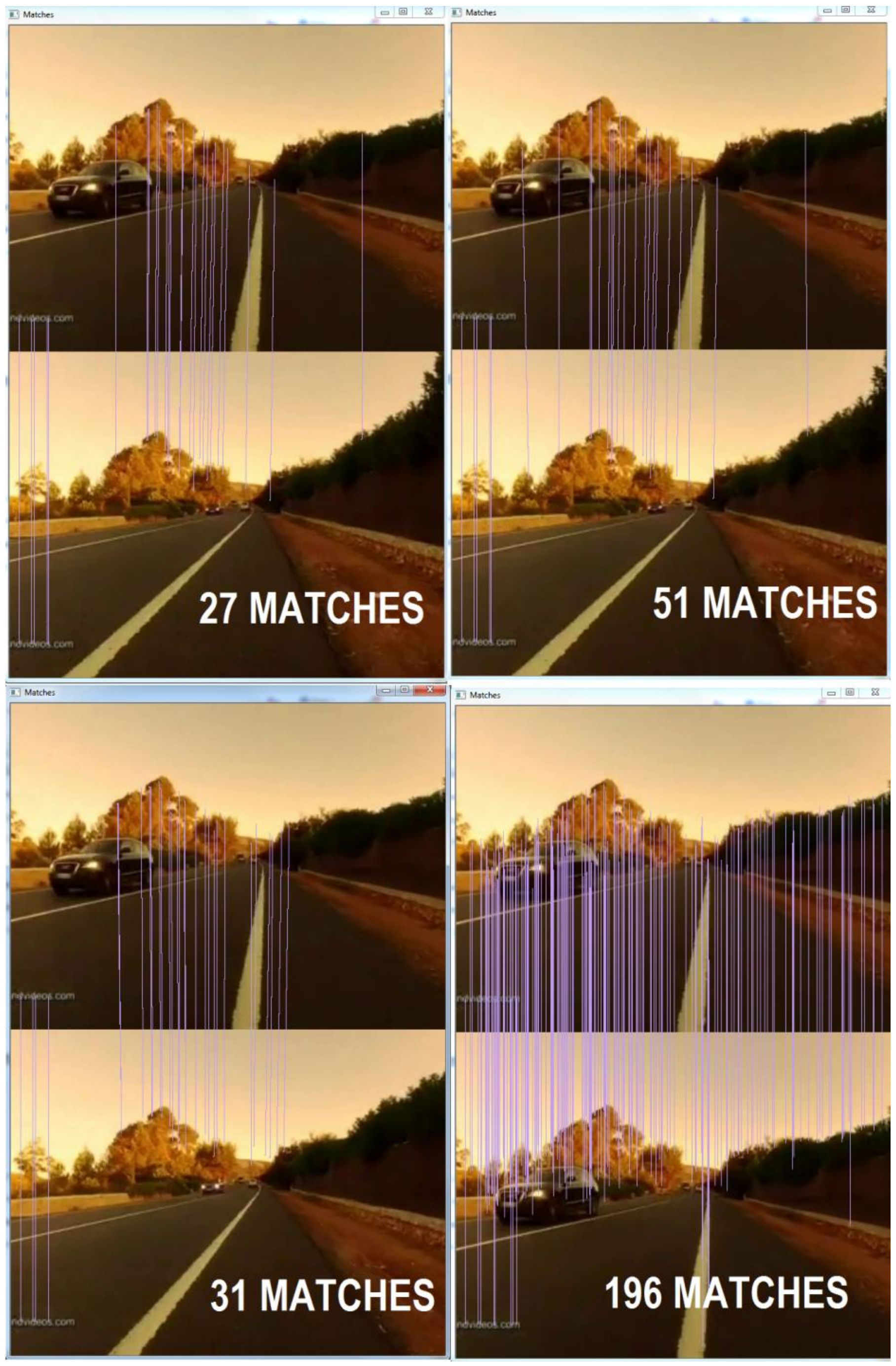


Figure 5-9: Illustration of matching algorithm.

Figure 5-9 shows an illustration of the comparison process. As seen, it perfectly detects the matches between each frame. The matching algorithm finds the frame that has the most number of similarities to the reference image. This frame is forwarded to the final process that detects changes in the bridge structure.

Chapter 6

SUMMARY AND FUTURE WORK

The proposed method for integrating the bridge monitoring system into the Pavement Management Information System (PMIS) has few known problems. This research project covered the complexity involved in detecting the bridge movement using computer vision. It provides solutions to challenges associated with the bridge monitoring system. One of the challenges is to reduce camera shift, rotation and vibration. The rotation and vibration problems are taken care of by the 2-Axis brushless gimbal and controller. Since the camera shift depends on the vehicle movement, a line detection algorithm is used to determine the lateral position of the vehicle on the road. By handling this problem, the system becomes more reliable because it is less prone to collecting visual data from a wrong viewpoint.

The second challenge is the selection of a video frame that makes the best match when compared with a reference image. The comparison is done using the Scaled Invariant Feature Transform (SIFT) algorithm, which is robust to a small shift, rotation and illumination changes. The algorithm outputs unique features that are compared with a reference image features to determine if it is a match.

The experimental results show the accuracy, robustness and speed of the overall process. These results are conclusive enough to state that the bridge monitoring system can be integrated with the PMIS.

That said, the bridge monitoring system still has room for improvement, and further work has to be done to complete the entire system.

1. Even though the control system and line detection reduce the amount of the camera shift, the accuracy depends on the driver of the vehicle. To

make this system perfect, another controller can be used to move the entire 2-Axis gimbal horizontally to compensate for the driver's errors.

2. The PMIS has a pavement profiler that consists of a distance encoder. The distance information can be used as a reference to start and stop the video recording.
3. The final stage of the bridge monitoring system is to detect the changes in the bridge structure. One technique that can be used is the stereo camera system which is explained in Appendix A. The stereo camera system provides information about distance in the image that makes it easier to measure the amount of change in the structure.

Appendix A

BRUSHLESS DC MOTOR BASICS

A.1 Fundamentals of Brushless DC Motors

The BLDC motor’s electronic commutator sequentially energizes the stator coils generating a rotating electric field that ‘drags’ the rotor around with it. N electrical revolutions equates to one mechanical revolution, where N is the number of magnet pairs. The BLDC motors experience something called asHall Effect. If a current carrying conductor is kept in a magnetic field, the magnetic field exerts a force on the moving charge carriers, tending to push them to one side of the conductor, producing a measurable voltage difference between the two sides of the conductor. This phenomenon is known as Hall Effect.

Hall sensors are used to detect the position of the rotor. These sensors can detect the North or South Pole. The hall sensor will transmit this signal to the controller of the motor. The controller will then switch on or off the appropriate coils in order to provide torque.

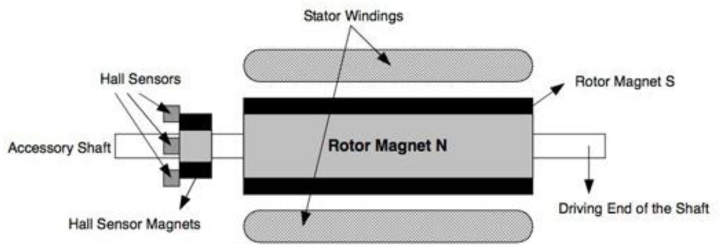


Figure B- 1: BLDC Arrangement

Figure shows a typical arrangement of driving a BLDC motor with Hall Sensors. Figure shows a typical BLDC system which the three coils of the motor are arranged in a “Y” formation, an 8 bit microcontroller, an Insulated Gate Bipolar Effect Transistor (IGBT) Driver, and a three-phase inverter comprising of six IGBTs. The output from the microcontroller comprises pulse width modulated (PWM) signals that determine the average voltage and average current to the coils (and hence motor speed and torque).

The motor uses three hall sensors to indicate rotor position. The rotor uses two pairs of magnets to generate magnetic flux.

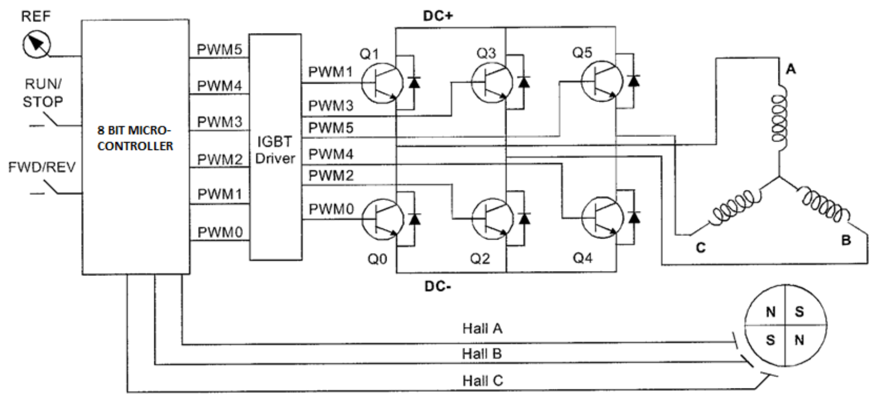


Figure B- 2: BLDC Control System Arrangement

Table A-1: 8-bit values for Clockwise operation of BLDC with 3 magnets

Sequence #	Hall Sensor Input			Active PWMs		Phase Current		
	A	B	C			A	B	C
1	0	0	1	PWM1(Q1)	PWM4(Q4)	DC+	Off	DC-
2	0	0	0	PWM1(Q1)	PWM2(Q2)	DC+	DC-	Off
3	1	0	0	PWM5(Q5)	PWM2(Q2)	Off	Dc-	DC+
4	1	1	0	PWM5(Q5)	PWM0(Q0)	DC-	Off	DC+
5	1	1	1	PWM3(Q3)	PWM0(Q0)	DC-	DC+	Off
6	0	1	1	PWM3(Q3)	PWM(Q4)	Off	DC+	DC-

Table A-2: 8-bit values for Counter-Clockwise operation of BLDC with 3 magnets

Sequence #	**Hall Sensor Input**			**Active PWMs**		**Phase Current**		
	A	**B**	**C**			**A**	**B**	**C**
1	0	1	1	PWM5(Q5)	PWM2(Q2)	Off	DC-	DC+
2	1	1	1	PWM1(Q1)	PWM2(Q2)	DC+	DC-	Off
3	1	1	0	PWM1(Q1)	PWM(Q4)	DC+	Off	DC-
4	1	0	0	PWM3(Q3)	PWM(Q4)	Off	DC+	DC-
5	0	0	0	PWM3(Q3)	PWM0(Q0)	DC-	DC+	Off
6	0	0	1	PWM5(Q5)	PWM0(Q0)	DC-	Off	DC+

Figure shows the current flow in an identical arrangement of coils to the motor in figure above for each of the six steps, and Figure shows the subsequent Hall Effect sensor outputs and coil voltages.

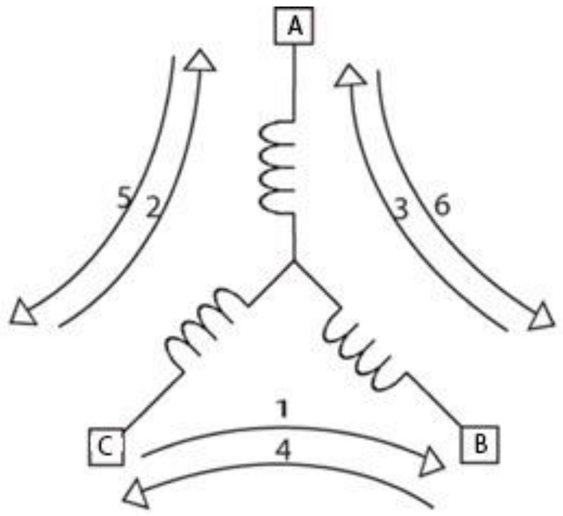


Figure A- 3: Current flow in a 3 Motor arrangement

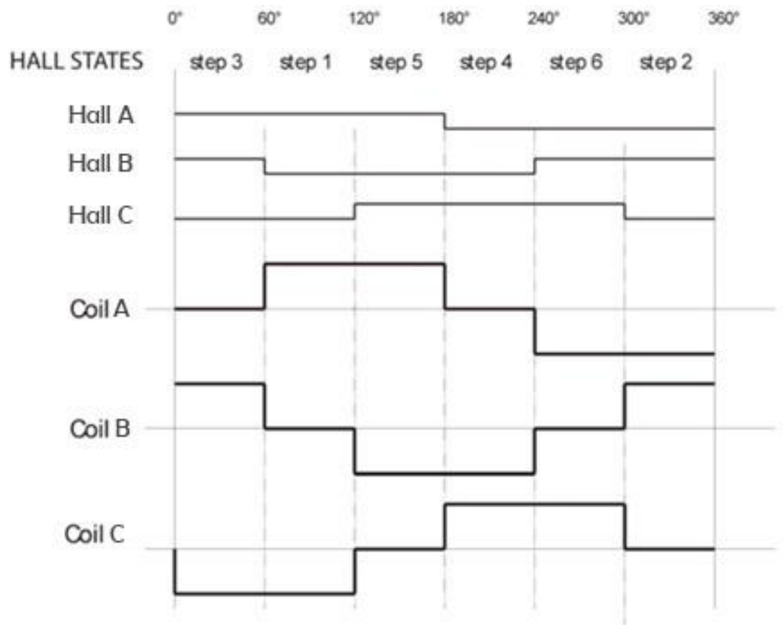


Figure B- 4: Hall Sensor Response

A pair of Hall sensors determines when the microcontroller energizes a coil. In this example, sensors Hall A and Hall B determine theswitching of Coil A. When Hall B detects a North Pole, coil A is positively energized. When Hall A detects a North Pole, coil A is switched to open.

Appendix B

CODE

B.1 Real-Time Video Capture And Line Detection

B.1.1 Capture.cpp

```
#include "opencv2/opencv.hpp"
#include <ctime>
#include <conio.h>

#define PI 3.1415926

using namespace cv;

void main(){
	int houghVote = 200;
	int saveFlag = 0;
	//VideoCapture cap(0); // open the default camera
	VideoCapture cap("drive.avi"); // open the video
	if(!cap.isOpened())  // check if we succeeded
	{
		getch();
		return;
	}

	char imgFileName[50]="";
	int n = 0;
	double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH); //get the width of 
frames of the video
	double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT); //get the height 
of frames of the video

	std::cout << "Frame Size = " << dWidth << "x" << dHeight << 
std::endl;


	Size frameSize(static_cast<int>(dWidth), static_cast<int>(dHeight));
	Size 
cannyFrame(static_cast<int>(dWidth/2),static_cast<int>(dHeight/3));

	VideoWriter videoWriter ("LaneDetection.avi", 
CV_FOURCC('P','I','M','1'), 30, frameSize, true); //initialize the 
VideoWriter object
	VideoWriter newvideo 
("canny.avi",CV_FOURCC('P','I','M','1'),30,frameSize,true);

	namedWindow("LINE DETECTION",1);

	while (1)
	{
		double begin = getTickCount();
		Mat image;
		Mat gray;
		std::vector<Vec2f> lines;
		cap >> image; // get a new frame from camera


		if (image.empty())
			break;
		if(waitKey(10) >= 0) break;

		n++;


		//videoWriter.write(image);


		cvtColor(image,gray,CV_RGB2GRAY);
		vector<string> codes;

		Rect
roi(image.cols/2,2*image.rows/3,image.cols/2,image.rows/3);// set the ROI
for the image
		Mat imgROI = image(roi);

		// Canny algorithm
		Mat contours,imgBLUR;
		blur(imgROI,imgBLUR,Size(3,3));
		Canny(imgBLUR,contours,70,90,3);
		Mat contoursInv;
		threshold(contours,contoursInv,128,255,THRESH_BINARY_INV);
		imshow("CANNY",contours);
		//newvideo.write(contours);
		/*
		Hough tranform for line detection with feedback
		Increase by 25 for the next frame if we found some lines.
		This is so we don't miss other lines that may crop up in the
next frame
		but at the same time we don't want to start the feed back loop
from scratch.
		*/

		if (houghVote < 1 || lines.size() > 2){ // we lost all lines.
reset
			houghVote = 200;
		}
		else{ houghVote += 25;}
		while(lines.size() < 5 && houghVote > 0){
			HoughLines(contours,lines,1,PI/180, houghVote);
			houghVote -= 5;
		}
		std::cout << houghVote << "\n";
		Mat result(imgROI.size(),CV_8U,Scalar(255));
		imgROI.copyTo(result);

		Point PT1,PT2,prev_pt2;
		Point center;
		float dist,prev_dist = 0,minDist = 0;
		float x,y;
		center.x = image.cols/2;
		center.y = image.rows;
		// Draw the lines
		std::vector<Vec2f>::const_iterator it= lines.begin();
		Mat hough(imgROI.size(),CV_8U,Scalar(0));
		while (it!=lines.end()) {

			float rho= (*it)[0];   // first element is distance rho
			float theta= (*it)[1]; // second element is angle theta

			if ( theta > 0.09 && theta < 1.48 || theta < 3.14 &&
theta > 1.66 ) { // filter to remove vertical and horizontal lines

				// point of intersection of the line with first
row
				Point pt1(rho/cos(theta),image.rows-
result.rows);
				// point of intersection of the line with last
row
				Point pt2((rho-
result.rows*sin(theta))/cos(theta),image.rows);

				dist = sqrt(pt2.x*pt2.x + pt2.y*pt2.y);
				if(minDist == 0)
				{
					minDist = dist;
					PT1 = pt1;
					PT2 = pt2;
				}
				else if(dist < minDist)
				{
					minDist = dist;
					PT1 = pt1;
					PT2 = pt2;
				}

			}
			++it;
		}
		// Display the detected line image
		PT1.x = PT1.x+image.cols/2;
		PT2.x = PT2.x+image.cols/2;
		line( image, PT1, PT2, Scalar(255,0,0), 8);
		line( image, center, PT2, Scalar(0,0,255), 8);
		if(minDist == 0) minDist = image.cols/2;
		std::stringstream stream;
		stream << "Distance: " << minDist-image.cols/2;

		putText(image, stream.str(), Point(10,image.rows-10), 2, 0.8, 
Scalar(0,0,255),0);
		imshow("LINE DETECTION",image);
		//videoWriter.write(image);

		char key = (char) waitKey(10);
		lines.clear();
		double end = getTickCount();
		double elapsed_secs = double(end - begin) / 
getTickFrequency();
		printf("%f\n",elapsed_secs);
		lines.~vector();
		codes.~vector();
	}
	cap.release();
	cap.~VideoCapture();
	videoWriter.release();
	videoWriter.~VideoWriter();
}
```

B.2 Feature Detection and Matching

B.2.1 Match.cpp

```
#include "Match.h"
#include <iostream>
//#include "LaneDetect.cpp"

using namespace cv;



Mat CombineImagesVertically(Mat img_temp1,Mat img_temp2)
{
	Mat img_temp_combined;
	img_temp_combined.rows = img_temp1.rows + img_temp2.rows;
	img_temp_combined.cols = MAX(img_temp1.cols, img_temp2.cols);
	img_temp_combined = 
cvCreateImage(cvSize(img_temp_combined.cols,img_temp_combined.rows),8,3);
	Mat top(img_temp_combined,Rect(0,0,img_temp1.cols,img_temp1.rows));
	Mat 
bottom(img_temp_combined,Rect(0,img_temp1.rows,img_temp2.cols,img_temp2.row
s));
	img_temp1.copyTo(top);
	img_temp2.copyTo(bottom);
	return img_temp_combined;
}
```

```
Mat DrawLine(vector<KeyPoint>keypoints_1,vector<KeyPoint>keypoints_2,int
num_matches,int offset,Mat combined,vector<DMatch>matches)
{
	float slope,rad,deg;
	float dist;
	float x,y;
	Point2f point_old,point_new;
	RNG rng;
	int icolor = (unsigned) rng;
	int n = 0;

	for(int i = 0;i < num_matches; i++)
	{
		point_old = keypoints_1[matches[i].queryIdx].pt;

		point_new = keypoints_2[matches[i].trainIdx].pt;

		point_new.y += offset;

		slope = ((point_old.y-point_new.y)/(point_old.x-point_new.x));

		rad = atan(slope);
		deg = rad * 180 / CV_PI;

		x = point_old.x-point_new.x;
		y = point_old.y-point_new.y;
		dist = sqrt(x*x + y*y);

		if(((abs(deg)<92.1) && (abs(deg)>87.9)))
		{
			line(combined, point_old, point_new, Scalar(
icolor&255, (icolor>>8)&255, (icolor>>16)&255 ), 1, 1, 0);
			n++;
		}
	}
	printf("%d ",n);
	return combined;
}

int main(int argc, char* argv[])
{
	cv::initModule_nonfree();
	string vidName, imgName;
	if(argc ==  3)
	{
		vidName = argv[1];
		imgName = argv[2];
	}
	else
	{
		std::cout << "Invalid syntax" << std::endl;
	}


	VideoCapture cap(vidName); // open the default camera
	if(!cap.isOpened())  // check if we succeeded
		return -1;

	cap.set(CV_CAP_PROP_FPS,30);
	double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH); //get the width of
frames of the video
	double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT); //get the height
of frames of the video

	Size frameSize(static_cast<int>(dWidth), static_cast<int>(dHeight));

	Mat img_object, img_scene, img_best_match,combined,output;
	std::vector<KeyPoint> keypoints_scene, keypoints_object,
keypoints_best_match;
	Mat descriptors_object, descriptors_scene, descriptors_best_match;
	img_object = imread(imgName,CV_LOAD_IMAGE_COLOR);
	int minHessian = 1000;
	int matchsize=0, max_matchsize=0;
	int flag = 0;
	SiftFeatureDetector detector( minHessian );
	detector.detect( img_object, keypoints_object );

	SiftDescriptorExtractor extractor;
	extractor.compute( img_object, keypoints_object, descriptors_object
);

	printf("Keypoints: %d, Descriptors:
%d\n",keypoints_object.size(),descriptors_object.size());

	Ptr<DescriptorMatcher> matcher = new BFMatcher(NORM_L2,false);

	std::vector< DMatch > best_match;
	int nCount = 0;

	while(1)
	{
		double begin = getTickCount();
		nCount++;
		cap >> img_scene;
		//cap.read(img_scene);
		if (img_scene.empty())
			break;
		if(waitKey(10) >= 0) break;

		imwrite("video_frame.jpg", img_scene);
		//-- Step 1: Detect the keypoints using SIFT Detector
		img_scene = imread("video_frame.jpg",CV_LOAD_IMAGE_COLOR);
		detector.detect( img_scene, keypoints_scene );
		//-- Step 2: Calculate descriptors (feature vectors)

		extractor.compute( img_scene, keypoints_scene,
descriptors_scene );
		//-- Step 3: Matching descriptor vectors using Brute-Force
matcher

		std::vector< DMatch > matches;

		matcher->match( descriptors_object, descriptors_scene, matches
);
		//
		double max_dist = 0; double min_dist = 100;

		//-- Quick calculation of max and min distances between
keypoints
		for( int i = 0; i < matches.size(); i++ )
		{ double dist = matches[i].distance;
		if( dist < min_dist ) min_dist = dist;
		if( dist > max_dist ) max_dist = dist;
		}

		//-- Draw only "good" matches (i.e. whose distance is less
than 3*min_dist )
		std::vector< DMatch > good_matches;

		for( int i = 0; i < matches.size(); i++ )
		{
			if( matches[i].distance <= 1.82*min_dist )
			{
				good_matches.push_back( matches[i]);
			}
		}
		double end = getTickCount();
		double elapsed_secs = double(end - begin) /
getTickFrequency();
		matchsize = good_matches.size();
		int offset = img_object.rows;
		printf("# of Matches in %d: ",nCount);
		output = CombineImagesVertically(img_object,img_scene);
		output =
DrawLine(keypoints_object,keypoints_scene,matchsize,offset,output,good_matc
hes);
		printf("OF %d\tElapsed Time:%f\t\n",matchsize,elapsed_secs );

		matchsize = 0;
		flag = 1;
		imshow("Matches",output);
		waitKey(1);
		matches.~vector();
	}


	waitKey(0);
	return 0;
}
```

B.2.2 Match.hpp

```
#include <stdio.h>
#include <iostream>
#include <vector>

#include "opencv2/opencv.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/nonfree/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include "opencv2/objdetect/objdetect.hpp"


#define PI 3.1415926

using namespace cv;
/*
Declarations
*/

Mat CombineImagesVertically(Mat,Mat);

Mat DrawLine(vector<KeyPoint>,vector<KeyPoint>,int,int,Mat,vector<DMatch>);

extern void LineDetect(Mat);

extern int InversePerspectiveTransform( Mat );
```

References

[1] American Society of Civil Engineers (2014, October 1). *2013 Report Card for America's Infrastructure Findings* [Online]. Available: http://www.infrastructurereportcard.org/a/documents/Bridges.pdf

[2] RCTimer Power Model Co., (2014, July 1). *2-Axis Brushless Gimbal Controller V1.0 Product Details* [Online]. Available: http://rctimer.com/product-872.html

[3] GoPro, Inc. (2014, July 1). *GoPro Hero-3 White Edition Product Details* [Online]. Available: http://shop.gopro.com/cameras/hero3-white/CHDHE-302.html

[4] Koskie, S. (2014, October 10) *General Purpose Processors Lecture Notes* [Online] Available: http://www.engr.iupui.edu/~skoskie/ECE362/lecture_notes/LNC04_html/img19.html

[5] Owens, John D., et al. "GPU computing." Proceedings of the IEEE 96.5 (2008): 879-899.

[6] Zhang, Z., & Hudson, W. R. (2001). *GIS Implementation Plan for the TxDOT PMIS* [Online]. Available: FTP: ftp://ftp.dot.state.tx.us/pub/txdot-info/rti/psr/ File: 1747_s.pdf

[7] Zhang, Z., & Machemehl, R. B. (2004). Pavement-related Databases in TxDOT. Center for Transportation Research, University of Texas at Austin.

[8] Wikipedia (2014, October 1). *Computer Vision* [Online]. Available: http://en.wikipedia.org/wiki/Computer_vision

[9] Forsyth, D. A., & Ponce, J. (2002). Computer vision: a modern approach. Prentice Hall Professional Technical Reference.

[10] Newhall, B. (1972). The history of photography: from 1839 to the present day.

[11] Wikipedia (2014, October 10). *Pinhole Camera* [Online]. Available: http://en.wikipedia.org/wiki/Pinhole_camera

[12] Fusiella, A. (2014, October 5). *Elements of Geometric Computer Vision* [Online]. Available:
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO4/tutorial.html

[13] Jensen, J. R. (1996). Introductory digital image processing: a remote sensing perspective (No. Ed. 2). Prentice-Hall Inc.. pp: 121-146

[14] L. S. Davis. A Survey of Edge Detection Techniques (1975). Computer and Image Processing 4(3) pp. 248-270

[15] Canny, J. (1986). A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, (6), 679-698.

[16] Harris, C., & Stephens, M. (1988, August). A combined corner and edge detector. In Alvey vision conference (Vol. 15, p. 50).

[17] Goodwin, G. C., Graebe, S. F., & Salgado, M. E. (2001). Control system design (Vol. 240). New Jersey: Prentice Hall.

[18] National Instruments, "PID Theory Explained" (2011)

[19] Digi-Key, Co. (2014, October 15). *An Introduction to Brushless DC Motor Control* [Online]. Available: http://www.digikey.com/en/articles/techzone/2013/mar/an-introduction-to-brushless-dc-motor-control

[20] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

[21] OpenCV Open Source (2014, October 1). *OpenCV 4.3.9 Developers Documentation [*Online]. Available: http://docs.opencv.org/

Biographical Information

Sushruth Mune Gowda was born in Bangalore, India in 1988. He received his Bachelor of Engineering in Telecommunication Engineering from Visvesvaraya Technological University (VTU) in 2010. While studying at VTU, he completed various projects in embedded microcontrollers, image processing and digital communication. After achieving his Bachelors', he worked as a Systems Engineer at Infosys Ltd., for one year. He quit his job to teach children mathematics and science at Pankhudi, an NGO in Bangalore, India that helps educate underprivileged children. In fall 2012, he joined the University of Texas at Arlington to pursue his Masters' in Electrical Engineering. While studying at UTA, he worked as the Graduate Teaching Assistant for CSE 5343 and did research work in the Transportation Instrumentation Laboratory. His research interests include embedded microcontrollers, image processing, computer vision and optimization.