Image Recognition Techniques for Optical Head Mounted Displays

Master Thesis

Submitted in Fulfilment of the Requirements for the Academic Degree M.Sc. Automotive Software Engineering

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Mahendra Kondreddy
Student ID: 335410
Date: 19.01.2017

Supervising Tutors: Prof. Dr. Wolfram Hardt
Nadine Becker (Daimler AG)
Aquib Rashid
Mahendra Kondreddy
(Mahendra.kondreddy@s2013.tu-chemnitz.de)
Image Recognition Techniques for Optical Head Mounted Displays
Master Thesis, Technische Universität Chemnitz, January 2017
Acknowledgement

It is a submissive experience to acknowledge those people who have helped me right from the start of my thesis.

My sincerest thanks to my thesis mentors Prof. Dr. Wolfram Hardt, Nadine Becker, Aquib Rashid and Ariane Heller for their immense support throughout my thesis. With their valuable concepts, suggestions, and contribution, this research wouldn’t have successfully completed. I would like to express my deepest gratitude to Nadine Becker for her unwavering support, collegiality, patience and uncanny ability to teach and to motivate me throughout this project. Her patronage on me can never be forgotten making my days in Daimler AG more remarkable. Whenever I think of my master thesis, she would be the first person who comes in my mind. I would wholeheartedly thank Simona Popisti and Marius Wagener for their colossal support and invigorating my career in Daimler AG.

To all my friends and well-wishers who have accompanied me with all this time to make my life even brighter. A unusual admiration must be left here to my parents and my brother Nagendra for their unflagging intimacy and unconditional support throughout my life and studies.

To the reader, whoever you might be, I hope you enjoy reading this thesis as much as I enjoyed writing it.
Abstract

The evolution of technology has led the research into new emerging wearable devices such as the Smart Glasses. This technology provides with new visualization techniques. Augmented Reality is an advanced technology that could significantly ease the execution of much complex operations. Augmented Reality is a combination of both Virtual and Actual Reality, making accessible to the user new tools to safeguard in the transfer of knowledge in several environments and for several processes.

This thesis explores the development of an android based image recognition application. The feature point detectors and descriptors are used as they can deal great with the correspondence problems. The selection of best image recognition technique on the smart glasses is chosen based on the time taken to retrieve the results and the amount of power consumed in the process. As the smart glasses are equipped with the limited resources, the selected approach should use low computation on it by making the device operations uninterruptable. The effective and efficient method for detection and recognition of the safety signs from images is selected. The ubiquitous SIFT and SURF feature detectors consume more time and are computationally complex and require very high-level hardware components for processing. The binary descriptors are taken into account as they are light weight and can support low power devices in a much effective style. A comparative analysis is being done on the working of binary descriptors like BRIEF, ORB, AKAZE, FREAK, etc., on the smart glasses based on their performance and the requirements. ORB is the most efficient among the binary descriptors and has been more effective for the smart glasses in terms of time measurements and low power consumption.

Keywords: Smart Glasses, OpenCV4Android, Feature Point Detectors and Descriptors, Keypoints, ORB, Random Sample Consensus.
# Table of Contents

Acknowledgement .................................................................................. iii
Abstract ................................................................................................. iv
List of Abbreviations ............................................................................ ix

1 Introduction ......................................................................................... 1
  1.1 Motivation ..................................................................................... 1
  1.2 Outline of Thesis .......................................................................... 3
  1.3 Problem Statement ....................................................................... 4
  1.4 Structure of the Work ................................................................... 4

2 Optical Head Mounted Displays .......................................................... 5
  2.1 Vuzix M100 Smart Glasses ............................................................ 5
  2.2 M100 at a Glance ......................................................................... 7
  2.3 Specifications of M100 ................................................................. 10
  2.4 Vuzix M100 System File Manager ................................................ 13

3 State of the Art .................................................................................. 19
  3.1 Image Processing Basics ............................................................... 19
  3.2 Understanding Features ............................................................... 23
  3.3 Feature Detectors and Descriptors ................................................ 25
    3.3.1 Features from Accelerated Segment Test ............................... 25
    3.3.2 Binary Robust Independent Elementary Features ............... 28
    3.3.3 Binary Robust Invariable Scalable Keypoints ...................... 29
    3.3.4 Fast Retina Keypoint ............................................................... 32
    3.3.5 Accelerated KAZE ................................................................. 35
  3.4 Image Processing Tools ............................................................... 36
  3.5 Taxonomy of Descriptors .............................................................. 39

4 Concept ............................................................................................ 42
  4.1 Concept Architecture ................................................................... 42
  4.2 Oriented FAST and Rotated BRIEF ............................................. 44
  4.3 FAST Keypoint Orientation .......................................................... 46
    4.3.1 Orientation using Intensity Centroid ................................... 47
List of Figures

Figure 2.1: Vuzix M100 Smart Glasses ................................................................. 5
Figure 2.2: Display Screen of Vuzix M100 .......................................................... 6
Figure 2.3: Back View of M100 ............................................................................ 7
Figure 2.4: Front View of M100 .......................................................................... 8
Figure 2.5: System File Manager for Vuzix .......................................................... 14
Figure 3.1: Types of Image Processing Techniques ............................................. 20
Figure 3.2: Image Co-ordinates Convention for Two-Dimensional Image .......... 22
Figure 3.3: Pixel Neighborhood .......................................................................... 22
Figure 3.4: Image with Paches ............................................................................ 24
Figure 3.5: Scale Space Interest Point Detection .................................................. 30
Figure 3.6: BRISK Sampling Pattern .................................................................. 32
Figure 3.7: FREAK Descriptor – FREAK Sampling Pattern ............................... 33
Figure 3.8: FREAK Descriptor – FREAK Sampling Pairs .................................... 34
Figure 3.9: FREAK Descriptor – FREAK Orientation Pairs .............................. 35
Figure 3.10: Taxonomy of Descriptors ................................................................. 40
Figure 4.1: Concept Architecture ...................................................................... 43
Figure 4.2: Detecting Keypoints Using FAST ...................................................... 46
Figure 4.3: Flow Chart of ORB ........................................................................... 51
Figure 4.4: Overview of Concept ....................................................................... 52
Figure 5.1: Permission of AndoridManifest file ................................................... 58
Figure 5.2: Display Screen of an Android Emulator ............................................ 59
Figure 5.3: Code for Detecting the Keypoints ...................................................... 60
Figure 5.4: Keypoints Detection using ORB ........................................................ 60
Figure 5.5: Finding Good Keypoints ................................................................... 62
Figure 5.6: Code for Drawing Matches ............................................................... 63
Figure 5.7: Examples for DrawMatches ............................................................... 64
Figure 5.8: Flowchart for Implementation ............................................................ 65
Figure 5.9: Code for Storing the Resultant Image ............................................... 66
Figure 6.1: Set 1 for Image Recognition Signs ..................................................... 68
Figure 6.2: Set 2 for Image Recognition Signs ..................................................... 68
Figure 6.3: Set 3 for Image Recognition Signs ..................................................... 69
Figure 6.4: Set 4 for Image Recognition Signs ..................................................... 70
Figure 6.5: Time Readings of FAST Detector and BRIEF Descriptor ............... 71
Figure 6.6: Time Readings of FAST Detector and FREAK Descriptor ............. 71
Figure 6.7: Time Readings of ORB Detector and Descriptor .............................. 72
Figure 6.8: Time Graph of various Descriptors .................................................... 73
List of Tables

Table 2.1: Operation of Various Buttons ................................................................. 10
Table 2.2: Comparison between M100 and Google Glass ....................................... 13
Table 3.1: Binary Descriptors .................................................................................... 41
Table 5.1: OpenCV4Android and Matlab Comparison .............................................. 55
Table 6.1: Power Measurements of BRIEF Descriptor .............................................. 74
Table 6.2: Power Measurements of FREAK Descriptor ............................................ 75
Table 6.3: Power Measurements of ORB Descriptor .................................................. 76
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADB</td>
<td>Android Debug Bridge</td>
</tr>
<tr>
<td>ADT</td>
<td>Android Development Tools</td>
</tr>
<tr>
<td>AGAST</td>
<td>Adaptive and Generic Accelerated Segment Test</td>
</tr>
<tr>
<td>AKAZE</td>
<td>Accelerated KAZE</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented Reality</td>
</tr>
<tr>
<td>BRIEF</td>
<td>Binary Robust Independent Elementary Features</td>
</tr>
<tr>
<td>BRISK</td>
<td>Binary Robust Invariant Scalable Keypoints</td>
</tr>
<tr>
<td>DoG</td>
<td>Difference of Gaussian</td>
</tr>
<tr>
<td>FAST</td>
<td>Features from Accelerated Segment Test</td>
</tr>
<tr>
<td>FED</td>
<td>Fast Explicit Diffusion</td>
</tr>
<tr>
<td>FREAK</td>
<td>Fast Retina Keypoints</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HOG</td>
<td>Histogram of Oriented Gradients</td>
</tr>
<tr>
<td>LDB</td>
<td>Local Difference Binary</td>
</tr>
<tr>
<td>MI</td>
<td>Mutual Information</td>
</tr>
<tr>
<td>MR</td>
<td>Mediated Reality</td>
</tr>
<tr>
<td>NCC</td>
<td>Normalized Cross Correlation</td>
</tr>
<tr>
<td>OHMD</td>
<td>Optical Head Mounted Displays</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Source Computer Vision</td>
</tr>
<tr>
<td>ORB</td>
<td>Oriented FAST and Rotated Brief</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Controller</td>
</tr>
<tr>
<td>RANSAC</td>
<td>Random Sample Consensus</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SFM</td>
<td>System File Manager</td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale-Invariant Feature Transform</td>
</tr>
<tr>
<td>SSD</td>
<td>Sum of Squared Differences</td>
</tr>
<tr>
<td>SURF</td>
<td>Speeded Up Robust Features</td>
</tr>
</tbody>
</table>
VR  Virtual Reality
2D  Two-Dimension
3D  Three-Dimension
Introduction

1.1 Motivation

The rise of the latest communication and information technologies has tremendously influenced how individuals communicate with one another or how they see the real world. Automation in detecting objects or readable images had attracted much attention recently, mainly due to the striking announcements by sundry players from the automotive and the IT sectors. In recent years, the downsizing of portable computers has led the research into wearable computing; users wear computers and use them as a part of their daily life. A wearable computer in the optical head-mounted displays has been gathering great attraction in implementing multiple applications and mounting interfaces [1]. In a computing environment, wearable computer devices provide appropriate information to the user's situation that is collected by using sensors worn by the user.

Despite wearable computing was born in 1997, when Steve Mann envisioned a “body-worn apparatus [that] augments and mediates the human senses” [2], only with the disruptive disturbance of smartphones we obtained enough low-cost computational power and sensing capabilities to change that idea into the real-world devices. Wearable computing has for quite some time been being under development, however, “smart glasses” have quite recently evolved as a viable platform for the personal and industrial sectors. Wearable computers are miniature electronic devices that can provide a range of functions depending on the specific device. The continuous image processing, assisting the client/users in regulating the context are anticipated to be able to executed by the wearable devices, while rendering them with timely reinforced data of the surrounding environment.

Smart glasses are a type of optical head-mounted displays that integrate first person cameras and hands-free displays with prompt access to processing power able to evaluate first person images in real time with hands-free operation. Smart glasses impose virtual world in addition to what a user naturally sees, without any correlation
between the foreground and the background. Smart glasses are a wearable head device that provides visual information to the user by means of display optics. Roughly speaking, smart glasses are a recent and innovative technology the aim of which is to affiliate information into a familiar set of lenses. Continuing the smartphone revolution, which placed digital information in our hands; smart glasses potentially permits to directly ingress the digital world by means of sight without the need for hands. Smart glasses are a subset of wearable computers that are worn on the head of the user that add information to a user’s reality typically with respect to an optical head-mounted display. Ideally, smart glasses provide users with hands-free navigation and functionality but usually include touch navigation as well. Smart glasses show tremendous assurance as a channel for conveying hands-free training content. I assume that the hands-free, personalized suitability of models in the future that the erudition of complex multi-step processes will be accelerated and permits for real-time access to requisite data in sterile or roving environments. Among the recommended applications for smart glasses are to study real-time warnings and visual instructions, instant connectivity access, instant photography or video capturing and augmented reality.

Smart glasses create new augmented reality experiences to enhance the perception and interaction of the user, thus creating new research opportunities and challenges. These challenges may arise from the facts that a smart glass has: Numerous sensors that can always be on. Those are GPS, gyroscope, accelerometer, camera and compass. In addition to the ability to execute multiple applications in the background and wireless communication with the other devices, as well as a connection to the internet [3].

In the modern years, technology has seen an innovative increase, advances in mobile and wearable technologies. Today, there are multiple smart glass devices commercially available in the market enabling a plethora of interactive applications ranging from visualizing content on a smart glass heads-up display to virtually interacting with objects in physical space. Two contemporary applications, Google Glass and the Vuzix M100, both incorporate an internet empowered computer, battery, speaker, camera, and voice and touch controls in an eyeglass form factor.
The user can capture and see images, capture and play videos, capture and listen to audio, and connect to the web media with respect to the small heads-up display mounted in front of one of the user’s eyes.

1.2 Outline of Thesis

Within the background of wearable computing, typically the terms virtual, augmented and mediated reality is adopted. In Virtual Reality (VR) applications, the user interacts with an artificially constructed 3D environment. In Augmented Reality (AR) applications (a type of VR), an overlaid approach is managed to furnish the user a blend of both the real and the simulated world; the effective physical environment typically refers to the background and the structural volume in which foreground simulated elements are visualized. Mediated Reality (MR) refers to the capability of manipulating one’s perception of the real world by means of virtual entities. MdR is thus a mandatory attribute of any AR system when seen from the context of user perception. Augmented Reality is a quantum leap technology that could considerably soften execution of complex operations. Augmented Reality combines virtual and actual reality, making it available to the user new tools to corroborate efficiency in the transfer of knowledge for different processes and in different environments.

Feature detectors and descriptors make sense when dealing with the natural images and help with the correspondence problems in computer vision. The thesis presents the analysis of various feature point detectors and descriptors for image recognition of safety signs in smart glasses. An efficient algorithm is selected from the results based on the time taken to display the results and the amount of power consumed on the smart glasses. As the smart glasses are equipped with limited resources as the smart mobiles, and they last for only 2 hours of time when they are completely charged, and so the selected algorithm should use low computational power. Based on the resources an analysis has been done to display the results of the safety sign on the display screen with the least time, and for this, the tracking of results is performed on the time and power consumption of various detectors and descriptors. If speed and power consumption is an issue then considering descriptors used to be the best approach for image recognition related tasks.
The main focus will be on the image recognition of safety signs on the smart glasses and the comparison analysis of time readings and power measurements of various feature point detectors and descriptors in smart glasses.

1.3 Problem Statement

As the virtual reality and augmented reality software and hardware have much developed over the past decade, the concentration of the field is moving away from the basic engineering technology and towards the science and uses of VR and AR methods. In the trainee hall at the Daimler AG plant, there will be a lot of trainees who work over there, and there will be a considerable measure of safety signs on the doors, walls and on the machinery unit. The trainees don't know what it is for which resulting in a lot of mistakes. In order to overcome the issue for all the new trainees, Daimler AG would like to introduce the usage of smart glasses in the trainee hall as the initial step to reduce the mistakes. Augmented reality supplements the perception and interaction with this real world and permits the user to be in an augmented real environment with further information generated by the computer. The impetus of this thesis is to develop an image recognition application using feature point detectors and descriptors for the smart glasses which ensue in low computational power consumption and time to display the results.

1.4 Structure of the Work

This thesis document is categorized into different sections where each section gives an idea on different topics as follows: In chapter 2 an introduction to Vuzix M100 smart glasses and their features and specifications along with the M100 System File Manager is given. Chapter 3 describes state of the art, in which different feature point detectors and descriptors are discussed along with the binary based descriptors. In chapter 4, the concept of ORB is explained in detail with the efficient rotation of the BRIEF operator and learning good binary features. Chapter 5 presents the implementation and realization of ORB detector and descriptor in smart glasses for recognizing various safety sign images. Chapter 6 presents the results of recognizing various safety signs on smart glasses and analysis of time and power measurements of various binary based descriptors on smart glasses. At last, this report is summarized, and the conclusion is given in detailed at the end.
2 Optical Head Mounted Displays

In this chapter, a detailed overview on Vuzix M100 smart glasses and its features, specifications with the comparison of google glass and operations are discussed.

2.1 Vuzix M100 Smart Glasses

The invention called smart glasses had been around for quite a while. It remained unpopular because of its initial purpose and the usage of the smart glasses came around when companies like Google and Vuzix recognized this awesome innovation that could be used for far more applications to make human life much better with the Augmented Reality.

Vuzix is the name of a technology firm that operates from Rochester, NY. It is one of the first companies that jumped at the idea of developing smart wearable glasses for the public. A unit of this gadget was developed and called the M100 series. It was released to the public in December 2013 to be sold at the price of $1,000. The M100 is easy to use smart glasses; it allows the user to develop a custom application with an integrated software development kit that comes with it in the packaging. Every user will also be able to use this device with their smartphones and all other compatible devices. For data storage concerns, users will have access to the cloud storage where all their data and relevant information can be securely stored [4].

The M100 is simply awesome. It runs on the android OS. The spectacular features that have set the M100 apart from other similar smart glasses in the market are unique, and these features have encouraged users to choose it over others with a lot of encouraging feedback. Some of these unique features are:
Voice recognition - This can be set and modified by the user as they wish.

Amplified heads up display.

An opaque outlook which gives a monocular view to the user.

Adjustable parts - These parts can be adjusted for more comfort to suit the different kinds of facial structure.

The unique control buttons which can be used to turn off and on the device and use other integrated features of the gadget.

These features have been expertly integrated into the system to run flawlessly on the android OS. Every user will find the functionality of the M100 spectacularly friendly. Because of its portability, it can be managed indoors and outdoors. The user would be capable to configure the settings as he/she wishes to use on the device by pairing it with any of the compatible devices from accessing the settings and make any changes. Once the settings are made, the user can use the smart glasses unhindered and independent of any other source for the power of operation. As expected, the user can have access to the basic android applications that come preloaded in the device. File sharing has become very routine application, the market is still looking for an easier and quicker way in this domain, sharing files is easy with the M100. When a user has uploaded data to the cloud or other compatible devices, it will be able to share any of the files from there easily.

The Vuzix M100 smart glasses have been designed to meet the needs of every user. The features that every user will expect to enjoy have been made accessible to this device. Many users will need features like a good camera, video recording, voice
control, and portability. These features are available in the M100 and can be upgraded as better updates for the android OS are released yearly. This means a user can have their M100 without being afraid that it will become obsolete and left behind with the introduction of newer features or android operating systems.

### 2.2 M100 at a Glance

The M100 Smart Glasses are a universal product undergoing continual development at every stage. Here is the detailed explanation on it when it is viewed from the back and front.

**M100 Smart Glasses – Back**

The back view of M100 device is as shown in the below Figure 2.3

![Figure 2.3: Back View of M100](image)

1. **Power/Data Port:**
   Micro USB port is used for charging battery and loading firmware updates and other software.

2. **Speaker:**
   Speaker and setting assembly base.

3. **Memory Card Slot:**
   Micro SD Card slot is used for removable memory.

4. **Display Screen:**
   Display screen with full color.

5. **Display/Camera Head:**
   It adjusts the tilt of the assembly.

6. **Display Arm:**
   It is used for bending and extending to enable display positioning.
7. **Power Button:**
   Power ON/OFF and Sleep.

**M100 Smart Glasses – Front**
The front view of M100 is as shown in the below Figure 2.4.

![Figure 2.4: Front View of M100 [4]](image)

1. **Front Button:**
   It moves the on-screen choice forward (Short Press) / Opens menu (Long Press).

2. **Center Button:**
   It moves the on-screen choice backward (Short Press) / Go home (Long Press).

3. **Rear Button:**
   The objects that are visible on-screen are chosen in the Smart Glasses (Short Press) / Go back (Long Press).

4. **White Power Indicator:**
   Indicates the power and charge status (Solid white – fully charged / 1 intermittent blink – Is charging / 3 fast blinks – nearly 0 charge and will not start due to lack of charge).

5. **Blue Notification Indicator:**
   It provides the status notification.

6. **Display Arm:**
   Bends and extends in and out to enable the display positioning.

7. **Gesture Sensor:**
   Input sensor is used for gesture control (Located between (4) and (5) on M100 device).

8. **Camera:**
   High definition camera is used for recording video and capturing still images.
9. Red LED Camera Indicator:
   It indicates that the camera is in operation.

**Smart Glasses Tour**

**User Interface Controls**

The M100 Smart Glasses support different user interface methods containing hardware buttons, gesturing, voice navigation and remote user interface software running on a Partner Device which is paired.

- Rear Button
- Center Button
- Front Button
- Gesture Sensor for operating the device.
- Microphone
- On/Off Power Button

**Buttons:**

The hardware buttons in Vuzix M100 function similarly to those on other Android mobile devices. In many instances, a fast “Short Press” and a “Long Press” holding the button down for a larger period may be interpreted differently by different apps. The list of operations by various buttons in M100 is displayed in Table 2.1 [4].

<table>
<thead>
<tr>
<th>Button</th>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rear Button (1)</td>
<td>Short Press</td>
<td>Android’s regular SELECT function</td>
</tr>
<tr>
<td></td>
<td>Long Press</td>
<td>Android’s regular BACK function</td>
</tr>
<tr>
<td>Center Button (2)</td>
<td>Short Press</td>
<td>Move preference BACK</td>
</tr>
<tr>
<td></td>
<td>Long Press</td>
<td>Android’s regular HOME function</td>
</tr>
<tr>
<td>Front Button (3)</td>
<td>Short Press</td>
<td>Move preference FORWARD</td>
</tr>
<tr>
<td></td>
<td>Long Press</td>
<td>Android’s regular MENU function</td>
</tr>
</tbody>
</table>
### Table 2.1: Operation of Various Buttons

<table>
<thead>
<tr>
<th>On/Off Power Button (6)</th>
<th>Short Press</th>
<th>Sleep mode by M100 device</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long Press</td>
<td>To turning off the M100 device by opening power menu or toggle it to Airplane mode</td>
</tr>
<tr>
<td></td>
<td>Long Press (10 Seconds)</td>
<td>Force complete shutdown of the M100 device</td>
</tr>
</tbody>
</table>

### 2.3 Specifications of M100

The technical specifications of an M100 device are as follows.

#### General
- Display resolution of: WQVGA Color display
- Aspect ratio of: 16:9
- Field of view (Diagonal): 15 degrees in angle
- Equivalent to the 4 inches mobile device screen viewed at 14 inches
- Brightness of: >2000 nits
- 24-bit color
- Right or left eye usable
- OMAP4460 at 1.2GHz
- 1 GB of RAM
- Android ICS 4.04, With API 15
- 4GB of flash

#### External Flash Slot
- Micro SD which support up to 32GB of memory

#### Controls
- Control buttons: 4
- Remote control app, which runs on paired Android device
- Customizable voice navigation support
- Gesturing support
Sensor Systems
- 3 DOF gesture engine (N/F, L/R, U/D)
- Ambient light
- GPS
- Proximity

Integrity Head Tracker
- 3 degree of head tracking freedom
- 3-axis gyro
- 3-axis accelerometer
- 3-axis mag or integrated compass

Battery
- 550mAh rechargeable with internal battery
- Up to 6 hours of hands-free (display off)
- 2 hours of hands-free + display
- 1 hour of hands free+ display + camera + high CPU loading

External Battery Pack
- Rechargeable battery of 3800mAh
- Ultra-thin USB mini-B cable
- Powers & recharges for M100
- 115 x 58.6 x 10MM
- Increases runtime of up to 6.5 times over base M100

Hands Free
- Ear speakers
- Microphone with noise canceling

Camera
- 5-megapixel pictures
- 1080p video recording
- Aspect ratio of 16:9

Mounting Options
- Mounting Overhead
- Mounting with Safety glasses
- Use with right or left eye

**Connectivity**
- Micro USB to: control/power/upgrade
- Wi-Fi of 802.11b/g/n
- Bluetooth

The below Table 2.2 shows a detailed comparison of both Vuzix M100 and Google glasses based on their specifications [5]. The classification is done based on the different attributes.

<table>
<thead>
<tr>
<th>CLASSIFICATION</th>
<th>VUZIX M100</th>
<th>GOOGLE GLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Navigation</td>
<td>• Two buttons to scroll forward and back&lt;br&gt;• One select button&lt;br&gt;• Holding buttons results in secondary action&lt;br&gt;• Buttons are stiff and have same feel and location on device</td>
<td>• Touchpad located on the frames of the glasses&lt;br&gt;• Multi-directional swipe fusing touchpad&lt;br&gt;• Tap touchpad to select</td>
</tr>
<tr>
<td>Voice Navigation</td>
<td>• Voice recognition that is running continuously&lt;br&gt;• Includes basic library of recognized terms&lt;br&gt;• Expandable with subscription fees</td>
<td>• Voice recognition can be initiated by saying “O.K. Glass”&lt;br&gt;• Includes extensive library of “approved commands”</td>
</tr>
<tr>
<td>Functionality</td>
<td>• Developer apps can be uploaded to the device, but with some interface challenges&lt;br&gt;• Standard 4 GB storage expandable with micro SD&lt;br&gt;• WIFI / Bluetooth</td>
<td>• Developer-created apps easy to load&lt;br&gt;• 12 GB of local storage (standard) and 12 GB of cloud storage&lt;br&gt;• WIFI / Bluetooth</td>
</tr>
<tr>
<td>Programmability</td>
<td>• Uses Google’s proprietary application programming interface (API)&lt;br&gt;• Restricted to limited speech recognition library</td>
<td>• Uses Google’s own API&lt;br&gt;• Provides developer helper apps&lt;br&gt;• Open library for voice control</td>
</tr>
</tbody>
</table>
| Hardware | • Combination of multiple pieces  
• Highly adjustable but not intuitive or natural  
• Weighted on one side | • Simplistic, uniform design similar to standard glasses with few removable parts  
• Limited adjustability  
• Lightweight, but unbalanced design |
| Capture and Playback of Audio and Video | • 5MP photos  
• 1080p video  
• Opaque visual display with attached camera tilting of 15°  
• Audio playback through a speaker that can be augmented through use of rubber earpiece | • 5 MP photos  
• 720p video  
• Bone conduction speakers output audio to the user  
• Transparent display that doesn’t interfere with vision  
• Fixed camera |
| Battery Life & Management | • Rechargeable battery; lasts 1-2 hours when in use | • Rechargeable battery; overheats when using camera or playback features |

Table 2.2: Comparison between M100 and Google Glass

2.4 Vuzix M100 System File Manager

Let us discuss the file manager system that runs in the M100 smart glasses. It is used like other file managers that might already know for different purposes. Some common actions that can be done with the file manager are: managing applications and editing the files in the gallery, applying patches and upgrades actions [6]. This will be discussed further using the illustration in next page. The descriptions will be made according to the highlighted numbers in Figure 2.5 for clarity.

The user will have a direct view of the processes running while using the file manager. The user will be capable to see the progress of any action being taken. It has the choice to refresh the activity if incase there are any alter that has to be made. Actions can only be made on the file manager when the idle state has been activated, if there is an action running and attempting to disrupt it, it will get a reply that the system is currently inaccessible because of the ongoing action. The user has to wait until the “Manager Idle” state.
There are three functional operation modes for the file manager system; the below are the details.

1) M100 OS Updates

There is an easy way to confirm the current OS the device is running on. Using the circular button that indicates the M100 OS updates will reveal the information. The information below reveals how to easily run a flash program on the device. From the file manager, the user will clearly see the current version at the point (11), click on this version to check for any updates. The reply will include the changes that have been made, and all the new features the user will enjoy with the new version. If the user wishes to continue, then click the icon for the update, and the device will be flashed. It is crucial to consider here that this process will erase all the files and so be sure to perform a backup of all the files before doing this operation.

There are some helpful tips that are available to the user while running this process. The user will get some hints at the point indicated (20). The user will also monitor everything through the live log visible at the point (14). At any point, wishing to cancel...
the progress, the user can use the abort button at point (19). The prevalent action of the device should not be heckled and it needs to be done with care.

There are other ways to perform an update on the device. For this optional method, the user will need to get the .pkg files directly from the Vuzix platform this should be saved directly on the hard drive. The next step is to enable the advanced settings from the location indicated as (21). The user will be able to observe the list of available updates here, what the user should do next is to click the upload file that wanted, and the download should start shortly.

2) App Control

Every user will be particularly happy to manage their installed applications easily. Please have a look at the point labeled (12) on the illustration, the user will find the control button for the installed applications. They are grouped into the system app, those apps that come with the device and the user installed apps. The user will be able to edit only the user installed apps as wanted. Any attempts to edit the system apps might lead to the device malfunctioning.

To install an app, the user will first have to locate the file containing the app data by selecting the browse button located at the position (5) on the illustration above. The user will see all the apps on the device and can install one or more applications at the same time. Install the selected application that wanted and to select more than one application for addition; the user should hold down the control key and select the apps that need to be installed. Note that only the apps that are compatible with android will be successfully installed.

3) Show Misc Files

Viewing the Misc files will require the same process as above. From the radial button, the user will gain access to the list as indicate at the position (12). Clicking on any of the lists will reveal the Misc files. The operations are similar to the common compute processes. If the user needs to delete a file, this can be done by double-clicking on it; the user will be induce with a request to confirm the deletion. As with a computer system, the user can also right click on file to see many alternative options for several
actions that want to take. The user can also manage the storage from here by monitoring those apps that are taking up too much space.

The entire list or a single list in the directory can be copied. Every user entailed to do is to choose what is needed to copy or the entire directory file. Paste in the location where it wanted to move them to. The user can copy and paste to the clipboard for easy access as well. There are two options to add files to the smart glasses. The user can either copy and paste from the location on the computer to the directory on the M100 or download it from the Vuzix menu seen at position (21), enable the advanced options menu and locate the directory and paste the file- easy. All these actions will need to be confirmed by the user. This will reduce the frequency of any mistakes, even though they are all reversible actions.

The user might be limited in this view because of those hidden directories in the system; the user should be able to easily solve this challenge by directly accessing the Vuzix menu (21) while the advanced options menu is activated. This will reveal a second menu highlighted as system directories. If this is clicked, the user will have successfully included it to the window. It cannot be hidden much longer. These files have been hidden to avoid any distortion in their set up that will surely lead to the device not functioning properly. It is advised that only a well-trained expert should access these files only when necessary. Finding a directory is easy. After the first experience, the user will easily perform this action again. The user needs to the right click on the window (12), and it will display the tab- search directories, clicking on this will bring up a list of directories from which the user can search for the file. There is a shortcut to skip looking through all the directories that pop up, if the user knows what to look for, simply click on the search directories, type in the directory beginning with ./ and it will get the results easily.

4) Other Features

Other features that are shown in the file manager for the M100 are discussed below-
(7) Check for Updates: With this feature, the user will be able to either refresh or update the version to the latest version on the FTP. To get the best experience, it is always advised to use the device only with the latest updates released.
(8) View Log file: The log file gives a summary of the actions that have been run on the device. It is easy to solve any system challenges by carefully looking through the log file to know where something went wrong in the system and the time of the error. Simply click this button to view the log file illustrated in the picture as log window (14).

(9) Help: The help button is a familiar button in any system. It is used to generate queries and get directions.

(10) This is the gateway to important information about the system like confirming the version of ADB currently running on the system, the flash image and a serial number of the device.

(13) The refresh button assists to restart all the current running processes and also provides the real time values and status. This operation happens from the system file manager.

(15) Snapshot: It is used to get a snapshot of the current display and save it to the viewpoint (18).

(16) Capture Video: this is a cool feature that transmits the display to the viewpoint. This is done at a recorded 5 frames per second. It continues as long as the user let it but the last frame will be saved on the viewpoint when the user shut down the captured video. However, this feature cannot be used alongside the refresh feature and other actions on the system file manager.

(17) Logcat Capture: This feature captures a comprehensive detail of every operation carried out by the M100 device. It runs when the user triggers it with the radial button, and it will get the log report from the system folder as soon as this feature stops working. It is advised to save this report on the hard drive as it will be useful for any troubleshooting that might need to carry out in future.

(18) Viewport: This is a feature that allows controlling and making changes to the M100 device from the system. From the Vuzix menu (21), the user will be able to do this when the mouse is moved in the viewport region. This only works when the mouse is in this region. The user can communicate with the M100 with the mouse and the keyboard when the viewport is active.
(21) System Menu: The system menu shows a lot of features that the user can make changes to. The user can open up this menu from the Vuzix logo; it is located in the upper left-hand area. The subsequent activity can be taken from the system menu-

1) Flush Log File – This is a feature that can use to clear the log files. It clears those files on the log file, but the stored data on the log cat are not affected by this action.

2) Advanced Features – This features when activated gives the access to areas like the browse buttons (4) and button (6) by the M100 updates (1), Misc files (3) and also the system directories location.

3) Continuous Capture – This feature activates a continuous capture at a higher frame rate. It gives better results but will require more power to sustain the process.

4) Show System Directories – If the user needs to view the system directories in the window display (12), and then simply click on this feature. However, the system should be in the show Misc file mode for this to happen.

5) Show System APKs – The user can view the system APKs by activating this feature. They will show up in the app control (2), many people hide their APKs because they tend to make the display look crowded and untidy.

(22) Close App: This is a backdoor route to force stop any running apps. This will become necessary as some apps can suddenly freeze in the process of operation. The user will need to force these apps to close.
3 State of the Art

The image is very simple for the human eye, but it is crucial to know how it is for the computer. There are many image processing algorithms available now. Basic ideas of an image and feature point detectors and descriptors are explained in detail in this chapter.

3.1 Image Processing Basics

Image processing is a tactic to transform an image into a digital form and perform some basic operations on it and in order to acquire a much enhanced image or to excerpt some effective data from it. It is a sort of signal exemption in which input is an image, comparable photo or video frame, and image might be an output or attributes connected with that image. Usually, image processing system includes treating the image as two-dimensional signals while applying already set signal processing methods to them. Images are also processed as three-dimensional signals whereas in which third-dimension belongs to Z-axis.

Image processing basically includes the below subsequent steps:

- Importing the image with an digital photography or by optical scanner.
- Inspecting and operating the image which comprises of data compression and image enhancement and looking patterns there are not to human eyes like satellite images.
- Output is the hindmost stage in which the output can be reformed image or report that is formed on image analysis.

Image Processing Purpose

The goal of image processing is divided into 5 groups. They are:

- Visualization – The objects that are not visible are observed.
- Image sharpening and restoration – A better image is created.
- Image retrieval - Seek for the image of interest.
- Measurement of pattern – Different objects in an image are measured.
- Image Recognition - Distinguish the objects in an image.
**Image Processing Types**

Analog and digital image processing are the two different types of methods that are used for Image processing. Digital image processing accords with the handling of digital images through a digital computer. It focus particularly on images even though it is a branch of systems and signals. Images are constituted in a two-dimensional image as a limited set of digital values termed as pixels or picture elements in a digital image. Association is another important tool in image processing though visual techniques. So analysts apply a combination of personal acknowledge and collateral data to image processing.

![Figure 3.1: Types of Image Processing Techniques](image)

**Image:**

According to Oxford English Dictionary, an image is a representation of the external form of a person or thing in art. The image is a visible impression obtained by a camera, telescope, scanner or any other imaging appliance or projected on a computer or video screen. The image is an optical display formed by light from an object reflected through the lens.

A digital image is a set of bits (zeros and ones) that represents an image. The computer can process the digital image as computer also uses zeros and ones as its language. Images are matrices of numbers to a computer. Two main categories of digital images are vector image and raster images. Vector images are formed with mathematical equations like points, lines, and polygons where Raster Images are represented by pixels. A pixel is called a picture view, is the smallest quantitative unit
in raster image [7]. Resolution is the number of pixels that is used to represent the image.

There are different types of images according to color

1. Binary Image
   Only two colors are used for the image; one color represents “1” and other one is for “0”.

2. Grayscale Image
   Grayscale image store the information of the intensity of light in it not the color. There are many ways to generate gray scale image. Such as 256 shades which lies between 0 as Black and 255 as White.

3. True color
   True color means the natural color images. There are different color spaces like RGB (Red, Green, Blue), CMYK (Cyan, Magenta, Yellow, Black).

Digital images can be stored in different file formats. Some popular formats are given below:

1. JPG or JPEG (Joint photographic Experts Group)
   It supports the color depth of 24 bits (3 color channels of 8 bits each).

2. GIF (Graphic Interchange Format)
   It supports the color depth of 8 bits only.

3. PNG (Portable Network Graphics)
   It supports the color depth of 48 bits (3 color channels of 16 bits each).

4. TIFF (Tagged Image File Format)
   It supports the color depth of 48 bits (3 color channels of 16 bits each).

5. Open EXR
   This is an open standard.

An image is captured by the imaging devices like camera, scanner. The image is the send to the computer or any other processor for processing. This image is represented as a two-dimensional array or matrix using a pair of co-ordinates (X, Y). The pixel represented by these co-ordinates depends upon the convention followed. Here, X represents horizontal axis and Y is vertical axis show in Figure 3.2. Each
pixel location \((i, j)\) contains the intensity value. Figure 3.2 shows the co-ordinate convention followed to display digital image. X-axis is representing the height of an image and Y-axis is representing the width of an image. Image quality improves when the image processing techniques can analyze a higher number of pixel of an image. Two ways can be used to measure the resolution of an image. Spatial resolution is a measure of a number of lines per millimeter or the number of dots per inch which are visible to human eye. Intensity resolution is the measure of smallest possible distinguishable intensity change of an image. Generally, it is 256. Intensity resolution increases with an increase intensity range. As in Figure 3.2, each pixel can be represented by \((X, Y)\) co-ordinate system.

![Figure 3.2: Image Co-ordinates Convention for Two-Dimensional Image [7]](image)

Figure 3.2 shows how to manipulate the neighbors of the desired pixel in an image. Pixel \(P (X, Y)\) can have three types of neighbors. They are four, eight or diagonal neighborhood. Using these neighborhoods image processing can manipulate any range within an image.

![Figure 3.3: Pixel Neighborhood [7]](image)
3.2 Understanding Features

In this part, I will be elaborating about a feature and a feature descriptor. A jigsaw puzzle game is played by most of the people whereby the very tiny pieces of larger images and each tiny image are needed to arrange them to form the actual image. The question can arise on how this can be achieved with ease. Or, better yet, what if a computer could be taught to play the same jigsaw puzzle games? Given that computers are far better than humans in many areas, what if computers could be taught to patch together natural images of scenes in the same way a jigsaw is made? Even further on, what if the computer being fed with the right pictures of the building in need then allowed to make a 3-dimensional model of the building?

While this can be explained about what is possible and what is not, here it can be answered the questions that led to all the possibilities; that of the basic concept of playing a jigsaw puzzle. How does not bring together all the images and create a signal big one? The fact that can be answered is that seeking specific patterns to the features that can be termed both as unique and comparable. Essentially, it can point out what feature can be found across the various individual images. This is the same reason of giving the game to a toddler for easily putting together the game as required by the rules. For this reason, playing a jigsaw puzzle requires to draw a certain similarity between the pieces. Those that share the same traits; they go together in the same jigsaw. While it point that the arrangement of the pieces is a result of the shape of color of the pieces, it is the natural tendency to seek continuity that is found in humans that leads us.

Thus, asking what the features that lead to continuity are in reality. Computers should be able to comprehend the specific features we are looking for in our jigsaw puzzle. Humans have the natural tendency to find the specific features that we look for in a jigsaw puzzle. If taking a closer look at some images like the one in next page in Figure 3.4, you will notice something interesting emerging. The image here is a very simple as it has three tiny images patches at its top. The task at hand is finding the location of the patches in the initial image. If it is closely observed, how many results can it have?
Figure 3.4: Image with Patches

A; is a flat surface and it is thus spread over a large area. Finding the exact location of the patches is thus a difficult task.

B; although finding the exact location remains difficult; it is a bit simpler find an appropriate location as B is the edges of the building. Along the edge of the building, the surface is uniform. However, the surface is different from the edge. For this reason, edge is better than flat surface area although not good enough.

C; is the corners of the building which can be found with ease. The reason for this is that any movement at the corners will change the look of the piece. For this reason, the corner is a good feature.

We have to look for the regions exhibiting the highest variation with small movements in the regions around them. Finding these features in an image is referred to as “Feature Detection”. Taking a region around a keypoint and describing it is called as “Feature Description”. Generally, the descriptor is a vector of values, which somehow describes the image patch around a keypoint [8]. It can be as easy as the values of a raw pixel, or it could be more complex such as the Histogram of Gradient (HoG) orientations.
3.3 Feature Detectors and Descriptors

In this part of the section, I will be discussing some of the detectors and descriptors that are currently available and can be used in this thesis.

3.3.1 Features from Accelerated Segment Test

Drummond and Rosten developed FAST as a corner detector. It works by considering a circle made of sixteen pixels around a corner candidate named $p$. Originally, the detector classified $p$ a corner only if there was a set of contiguous pixels $n$ in a circle that are brighter than candidate pixel $I_p$ and the threshold $t$. It can also be darker than $I_p - t$. $n$ is chosen as twelve given that it will admit a high-speed test that can be put to use in excluding a large number of points that are non-corners. The examination of the pixels 1 and 9 is done by a high speed test. If both of them are within $t$ if $I_p$, then $p$ is not a corner. If $p$ has the possibility of being a corner, we examine pixels 5 and 13. $p$ is being a corner means that a minimum of three of them should be brighter than $I_p + t$ or darker than $I_p - t$. If none of them is true, $p$ is not a corner. The image is referred in Figure 4.2 for giving a better idea. All the remaining pixels in our circle can be examined using this method. Even with the high performance there are several weaknesses [9].

1) Using the high-speed test will not reject many $n < 12$ candidates given that a point qualifies to be a corner only if two of four pixels are both brighter or both darker than $p$. Other tests are needed in finding out whether the complete test should be done a bright or a dark ring.

2) The detector’s efficiency depends on the order of the questions and distribution of the appearances of the corners. This choice of pixels is likely not optimal.

3) There are multiple features that are detected adjacent to each another.

B) Improving the Generality and the Speed with Machine Learning:

This section addresses numbers 1 and 2 of the points with machine learning while the next section will address the third point. There are two stages in this the first of which is to build a corner detector to the given $n$ with 16 of the pixel rings being extracted for a set of images from the application domain in the target. They are then
labeled with a straightforward implementation of each segment test criterion to \( n \) with a threshold that is convenient [10].

Using \( P \) as the set of all pixels in the training images then \( x \) partitions \( P \) into 3 subsets \( P_d, P_s, P_b \), with:

\[
P_b = \{ p \in P : S_{p\to x} = b \},
\]

and \( P_d \) and \( P_s \) have similar definitions. What we mean is that the given choice \( x \) is being used in partitioning the data into 3 sets with set \( P_d \) containing all points in which pixel \( x \) is much darker than the pixel at the by a threshold of \( t \), \( P_b \) has points that brighter than the pixel at the center by \( t \), \( P_s \) has all the remaining points in which pixel \( x \) has similarity to the pixel at the center.

If \( K_p \) is a Boolean variable that is true only if \( p \) is a corner-point, the second stage will employ the algorithm that found use in ID3. It starts by the selection of \( x \) that yields much information on whether the pixel that is a candidate is a corner or not based on a measurement of \( K_p \)’s entropy.

To find \( K \)’s total entropy for an arbitrarily set of corners, \( Q \) we use:

\[
H(Q) = (c + \bar{c}) \log_2 (c + \bar{c}) - c\log_2 c - \bar{c}\log_2 \bar{c}
\]

Where \( c = |\{i \in Q : K_i \text{ is true}\}| \) (number of corners)

And \( \bar{c} = |\{i \in Q : K_i \text{ is false}\}| \) (number of non-corners)

Our choosing of \( x \) will then yield an information gain of \( (H_g) \):

\[
H_g = H(P) - H(P_d) - H(P_s) - H(P_b)
\]

After selecting the \( x \) that will yield the most information, we apply this same process recursively on the three subsets. \( x_b \) is chosen in partitioning \( P_b \) into \( P_{b,d}, P_{b,s}, P_{b,b} \), \( x_s \) is chosen in partitioning \( P_s \) into \( P_{s,d}, P_{s,s}, P_{s,b} \) on and on with each \( x \) being with the aim of yielding the maximum information on the chosen set. The process of recursion is terminated if a subset’s entropy is zero. What this means is all \( p \) in the subset would have an equal value of \( K_p \), meaning they are all non-corners or corners. This aspect is a guarantee given that \( K \) would be the exact data function. To sum everything up, the process above results into a decision tree that can be used to classify all the corners.
in a training set correctly. This is a close embodiment of the given rules for the FAST corner detector.

There are cases where there is a similarity between the first two subtrees. In such a case, we remove the Boolean test that separates them. We then convert the decision tree to C code to create a long string of if-else statements that are nested, compiled and then used as corner detectors. To do the highest speed operation, we use profile guided optimizations that allow the prediction of branches and the reordering of block optimizations to compile the code. To further optimize it, we will force $x_b$, $x_d$ and $x_s$ to be all equal. This case will always have the second texted pixel as the same. This being the case, we can then perform tests 1 and 2 in batch. This aspect allows for the performance of two of the first tests in parallel to a pixel strip with the vectorizing instructions that can be found on most microprocessors of high performance. This aspect will significantly increase the speed test given that most of the points will be rejected after two tests. It ought to be noted that there is a difference between the learned detector and the segment test detector as all the possible corners are not completely covered by the data. When dealing with FAST-n detectors, one ought to include an instance of all combinations of pixels as there are $3^{16} = 43,046,721$ possible combinations [11].

**C) Non-maximal Suppression:**

Given that the segment test will not compute a function for a corner response, non-maximal suppression will not be directly applied to the features resulting from this. Given $n$, as $t$ increases, the corners detected corners will thus decrease in number. As $n = 9$ will produce the results that can be easily repeated, any variations in $n$ will be ignored. The corner strength will thus be defined as highest value of $t$ for which any point shall be detected to be a corner.

We can employ the given decision tree in determining the pixel’s class for any value of $t$. The pixel’s class such as 1 for a corner and 0 for a non-corner is a function of $t$ that is monotonically decreasing. We can thus employ bisection in efficiently finding the point that sees to the change of the function from 1 to 0. The point will give us the highest value of $t$ where it is detected as a corner. Given that $t$ is discrete the result
is the binary search algorithm. We can use an iteration scheme as the alternative. In this case, a pixel on a ring will be within $t$ of the center to pass the segment test. If a given number of pixels fail, the point is not a corner. Increasing the threshold while rerunning the test to a point of failure is the aim in this test. Given that the speed is a heavy reliant on the learned tree besides the architecture of the processor, no technique has a speed advantage over another. Then Perform the non-maximal suppression with a 3 X 3 mask.

3.3.2 Binary Robust Independent Elementary Features

SIFT uses 128-dim vector for descriptors. Since it uses floating point numbers, it takes fundamentally 512 bytes. Likewise, SURF also takes a minimum of 256 bytes (for 64-dim). For thousands of features creating a vector consumes a lot of memory which is not at all feasible for the resource-constraint applications especially for the embedded systems. In short, larger the memory then longer the time it takes for matching. Still, all these dimensions may not be considered for actual matching. It can be squeezed by using various methods like PCA, LDA, etc. Indeed different techniques like hashing using LSH (Locality Sensitive Hashing) is been used to convert these SIFT descriptors to binary strings from floating point numbers. These binary strings are used to match these features using Hamming distance [12]. Calculating a hamming distance is simply performing XOR operation and bit count, as this gives much better speed-up which is much snappier in present day CPUs with SSE guidelines. But here, at first it needs to find the descriptors, and only then hashing can be involved to it, which doesn’t resolve the underlying issue on the memory.

BRIEF comes into account at this moment. Binary Robust Independent Elementary Features (BRIEF), is a low bit rate descriptor, which is introduced for image matching with random forest and random ferns classifiers. BRIEF belongs to the family of binary descriptors such as BRISK and LBP, which only executes simple binary comparison test and uses Hamming distance as a replacement of Euclidean or Mahalanobis distance. Briefly, for building a binary descriptor, it is only essential to compare the intensity between two-pixel positions placed around the detected interest points in an image. This allows getting a representative description at a very
low computational cost. Besides, matching the binary descriptors entails just the computation of Hamming distances which can be executed very fast by using XOR primitives on modern architectures.

To constitute an image patch as a binary based string, the BRIEF algorithm relies on a relatively trivial number of intensity difference tests. More precisely, for a patch of pixels in a binary descriptor of size \(S \times S\) is built by concatenating the results of the below equation

\[
T = \begin{cases} 
1, & \text{if } I(P_j) > I(P_i) \\
0, & \text{Otherwise}
\end{cases}
\]

In the above equation, \(I(P_i)\) denotes the value of pixel intensity at \(P_i\). The set of binary tests are defined uniquely by the selection of location of all the \(P_i\). To increase the robustness of the descriptor, the patch of pixels is pre-smoothed with a Gaussian kernel with a variance which is equal to 2 and size equal to 9 x 9 pixels. This \(T\) can be 128, 256 or 512 bits. Even though the OpenCV buttress all of these, but by basic, it would be 256 (Bytes are used in OpenCV. So the values will be of 16, 32 and 64). After obtaining this, one can use the Hamming distance in order to match these descriptors. One important aspect of BRIEF is that it is a feature descriptor. It has to use any other various feature detectors like SIFT, SURF, FAST etc., as it doesn’t render any method in an image to find the features in it. The BRIEF is considered to be very efficient both to compute and to store in memory. Unfortunately, BRIEF descriptor is not robust against a rotation which is larger than 35 degrees. In short, BRIEF is a faster method feature descriptor calculation and for matching.

### 3.3.3 Binary Robust Invariable Scalable Keypoints

BRISK (Binary robust invariable scalable keypoints) was proposed in ICCV, 2011 [13]. BRISK has a binary descriptor and a feature detector. As far as the binary descriptor is taken into account, BRISK is also build on analogize intensities between two sampling patterns, but with a dissimilar sampling and feature selection approach. Once generated, the BRISK keypoints can be emulated very efficiently acknowledgment to the binary nature of the descriptor [13]. From an image to create a keypoints, BRISK proposes a novel method which is structured as follows:
Scale-Space Keypoint Detection

The detection methodology is inspired by the work of Mair et al [14] for detecting regions of interest in the image by the focus on efficiency of computation. The aim of achieving invariance to scale is crucial for more-quality keypoints, advance a step nurture by searching for maxima not solely in image plane, but also in scale-space using the FAST score \( s \) as a calculate for saliency [15]. Than alternative high-performance detectors despite discretizing the scale axis at coarser intervals, the BRISK detector evaluates the continuous scale-space for true scale of each keypoint.

Figure 3.5: Scale Space Interest Point Detection [13]

The scale-space pyramid layers in BRISK framework consist of \( n \) octaves \( c_i \) and \( n \) intra-octaves \( d_i \), for \( i = \{0, 1... \ n-1\} \) and typically \( n = 4 \). By sampling half of the original image (corresponding to \( c_0 \)) progressively octaves are formed. Each intra-octave \( d_i \) is located in between layers \( c_i \) and \( c_{i+1} \) (as illustrated in Figure 3.5). The first intra-octave \( d_0 \) is obtained by down sampling the original image \( c_0 \) by a factor of 1.5, by
successive half sampling rest of intra-octave layers are derived. Therefore, if \( t \) denotes scale then \( t(c_i) = 2^i \) and \( t(d_i) = 2^i \times 1.5 \).

It is important to note that different alternatives of mask shapes for keypoint detection are provided by both AGAST and FAST. In BRISK, For the FAST criterion to be fulfilled we mostly wield the 9-16 mask, which essentially entails at least 9 consecutive pixels in the 16 pixel circle to either sufficiently brighter or darker than the central pixel. Initially, to identify potential regions of interest FAST 9-16 detector is applied on each octave and intra-octave separately using the same threshold \( T \). Further, the points associated to these regions are concerned to a non-maxima suppression in scale-space: At first, the point in the query entailed to fulfill the atmost condition with respect to its 8 neighboring FAST scores ‘\( s \)’ in the same layer. Considering an image point corner the score \( s \) is defined as the maximum threshold. Secondly, the scores will need to be lower in the above and below layers. When the equally sized square patches checked inside: With the suspected maximum the side-length is preferred to be 2 pixels in the layer. Some interpolation is applied at the boundaries of the patch, since the neighboring layers (and therefore its FAST scores) are represented with a different discretization. The above Figure 3.5 portrays an example of this sampling and the maxima search. The detection of maxima across the scale axis at octave \( c_0 \) is a special case: we apply the FAST 5-8 mask on \( c_0 \) in order to obtain the FAST scores for a virtual-octave \( d_{-1} \) below \( c_0 \). However, the scores in patch of \( d_{-1} \) are in this situation not entailed to be lower than the score of the analyzed point in octave \( c_0 \).

**Keypoint Description**

Set of keypoints (consisting of associated floating-point scale values and sub-pixel refined image locations), by concatenating the results of simple brightness comparison tests the BRISK descriptor is composed as a binary string. This idea is very efficient; however, employ it in a far more qualitative manner. In order to achieve rotation invariance which is key to general robustness in BRISK, to allow orientation-normalized descriptors identify the characteristic direction of each keypoint. With the focus on maximizing descriptiveness choose the brightness juxtapose.
Pattern Sampling and the Rotation Estimation

The BRISK descriptor forge the use of a pattern that is engaged for sampling the keypoint neighborhood. The concentric with the keypoint clarifies N places equivalently spaced on circles as the pattern portrayed in below Figure 3.6. DAISY descriptor resembles this pattern, its use in BRISK is entirely different, capturing further information and thus resulting in consequence to demanding speed and storage requirements. DAISY was specially built for dense matching.

![BRISK Sampling Pattern](image)

Figure 3.6: BRISK Sampling Pattern [13]

3.3.4 Fast Retina Keypoint

More or less FREAK is much similar to BRISK by having a handcrafter sampling pattern and it is also similar to ORB by using the machine learning techniques in order to learn the optimal set of sampling pairs. FREAK also has an orientation mechanism which is similar to that of BRISK [16].

Retinal Sampling Pattern Outline:

Numerous sampling patterns are possible to compare the pairs of pixel intensities. As we already know that BRIEF uses random pairs, ORB uses learned pairs and then BRISK uses a circular pattern where points are equivalently spaced on circles concentric, similar to that of DAISY [17].
FREAK recommends using the retinal sampling grid which is also circular with the difference of having higher density of points near to the center. The difference with BRISK is that the exponential change in size and the overlapping receptive fields. The density of points drops exponentially, and it can be shown in the below Figure 3.7. Each circle in the below Figure 3.7 represents a receptive field where the image is smoothed with its corresponding Gaussian kernel, and the radius of the circle represents the size of the standard deviation of the kernel.

![Figure 3.7: FREAK Descriptor – FREAK Sampling Pattern [16]](image)

**Sampling Pairs Learning:**

With the number of few dozen sampling points, thousands of sampling pairs can be considered into account. However, many of these pairs might not be useful efficiently to describe a patch. A feasible strategy can be to follow BRISK’s approach and select the pairs according to their spatial distance. However, the selected pairs can be of highly correlated and are not discriminant. Apparently, FREAK follows ORB’S approach and then tries to learn the pairs by maximizing variance of the pairs and taking pairs which are not correlated.
Fascinatingly, there is a structure in the resulting pairs – a coarse-to-fine perspective which is equivalent to our apprehension of the model of the human retina. The sampling points in the external rings of the pattern are predominantly juxtapose when the first pairs that are selected. This is much close to the way the human eye operates, as it first employs the parafoveally receptive fields to estimate the location of an object of interest. Then, the validation is presented with the additional densely distributed receptive fields in the fovea area. The sampling pairs of FREAK are represented in the following Figure 3.8, where each figure contains 128 pairs.

![Figure 3.8: FREAK Descriptor – FREAK Sampling Pairs [16]](image)

FREAK’s takes the edge of this coarse-to-fine structure to minimize the time and to use a cascade approach which additionally speed up the matching: When matching the two descriptors, we first compare barely the first 128 bits. If the resultant distance is smaller than a threshold, we additionally continue the comparison of the further 128 bits. As a consequence, a cascade of juxtapositions is conducted accelerating
even further the matching as more than 90% of the candidates are rejected with the first 128 bits of the descriptor.

**Orientation Assignment:**

To slightly compensate for rotation changes, FREAK computes the orientation of the keypoint and rotates the sampling pairs by measuring angle. FREAK’s mechanism for computing the orientation is compared to that of BRISK only that rather than using long distance pairs, FREAK employs a predefined set of 45 symmetric sampling pairs.

![Figure 3.9: FREAK Descriptor – FREAK Orientation Pairs [16]](image)

### 3.3.5 Accelerated KAZE

**Building the Nonlinear Scale Space:**

For detecting and describing the features, FED schemes are used for building a nonlinear scale space. With any additional kind of discretization scheme, a nonlinear scale space can be built much quicker by means of FED schemes. Additionally, when compared with the other approaches such as Additive Operator Splitting [18] the FED schemes are especially easy to implement and are much accurate. The principal target is to carry out $M$ cycles of $n$ explicit diffusion steps with diverging step sizes $T_i$ that arise from the factorization of a box filter:
$$T_j = \frac{T_{\text{max}}}{2 \cos^2 \left( \pi \frac{2j + 1}{4n + 2} \right)}$$

**Feature Detection:**

For every filtered image $L^i$ in the nonlinear scale space calculate the determinant of the Hessian. By using a normalized scale factor that takes into consideration the octave of every specific image in the nonlinear scale space, the set of distinctive multiscale operators are normalized with respect to the scale. Then the maxima of the detector response in scale and spatial location is searched.

$$L_{\text{Hessian}}^i = \sigma_{i,\text{norm}}^2 \left( L_{xx}^i L_{yy}^i - L_{xy}^i L_{yx}^i \right)$$

**Feature Description:**

A Modified-Local Difference Binary is used that utilizes gradient and intensity information from the nonlinear scale space. The same principal of BRIEF [12] is followed by the LDB descriptor [19]. Instead of single pixels for further robustness but using the binary tests between the average of areas. The mean of the horizontal and vertical derivatives in the areas being compared is used in addition to the intensity values, resulting in 3 bits per comparison. The rotation invariance is achieved by calculating the main orientation of the keypoint in KAZE, and the grid of the LDB rotated correctly. The descriptor is robust to change in scale by the scale dependent sampling.

**3.4 Image Processing Tools**

In this section, the image processing tools that are used in this project are discussed. It is an advance technique of computer vision and Artificial Intelligence. The selection of proper tools which can deliver high performance with low cost is one of the most important agenda of this thesis. Some of the tools for developing image processing software are Open Source Computer Vision (OpenCV), Mat lab, ToolIP-Image Processing from Fraunhofer ITWM, LEAD Tools Image Processing SDK. There are also some other tools available, but most of them are application specific. The above four tools are application specific tools. Any image processing software can be developed using these tools. The comparison has been done among these four tools for our software development purpose. The selection is done on the basis of low
cost, good development, and cross platform availability. The following paragraph discusses the merits and demerits of all these four tools. ToolIP is a programming tool that supports in image processing and its analysis. This tool is used to solve complex image processing algorithm and has also many additional features such as graph descriptive, highly modular and reusable, and rapid prototyping is allowed. The graph descriptive feature allows the user to create complex image processing algorithms with the help of nodes and edges. The nodes represent the components, and the edges describe the signal flow among nodes. The development with the help of nodes and edges is also called prototyping phase, and it can be exported as XML format.

The other tool is LEAD Tools. This tool has also most of the feature for image processing. The SDK is available for many programming languages. ToolIP and LEAD Tools, both are well capable for thesis task. But, one of the purposes of thesis is to develop very cheap system. These two tools are not open Source. They have less development support, and enough amount of supporting documentation is limited compared with open source tools like OpenCV. Now the choice remains in between OpenCV and Mat lab. Though Matlab is also not open source software but it has all the necessary facilities, very good development support and enough documentation for the concept developed. In latter section, the comparison between OpenCV and Mat lab is done.

OpenCV:

OpenCV stands for an acronym of Open Source Computer Vision. OpenCV is an image processing library developed by Intel and later supported by Willow Garage and now maintained by Itseez. Under the open source license, it has been feasible since 2000. The OpenCV is natively written in C++ and even runs under Linux, Windows and Mac OS X. Active development on interfaces is available for Python, Ruby, Matlab and other languages. The OpenCV was designed for computational efficiency and with a strong focus on real time applications. It is a group of frequently used functions that perform operations that are related to computer vision. The library render by OpenCV is a collection of low-overhead, high performance operations performed on the images. As most of the Android application development is being
done in Java, OpenCV has also been ported as an SDK so that the developers can use to implement it in their apps and make them vision enabled [20].

OpenCV is a free Cross-Terrance library for real-time image processing that has develops into a standard tool for all things relevant to Computer Vision. OpenCV is aimed at implementing the tools necessary to solve computer-vision problems. In 2010 a new module that maintains GPU acceleration was added to OpenCV. The GPU module covers an important part of the library’s functionality and is still in active progress. It is executed using CUDA and in therefore gain from the CUDA system, including libraries, for example, NVIDIA Performance Primitives (NPP). The GPU module permits the users to avail from GPU acceleration without exacting training in GPU programming [21]. The module is constant with the CPU version of OpenCV, which makes maintenance easy. There are distant, however, the most important of which is the memory model. In 2012, the nonprofit organization OpenCV.org took on the charge of overseeing a support site for developers and users. One of OpenCV goal is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. More than 2500 optimized algorithms are contained in OpenCV library, which contains a comprehensive pair of both classic and state-of-the-art computer vision and algorithms of machine learning.

The OpenCV library is divided into several built in application specific modules as mentioned below.

- **Opencv_core**: The core module contains the core functionalists the basic and dynamic data structures and arithmetic functions. For example, Mat data structure and operations, etc.
- **Opencv_imgproc**: Image processing functions and methods are available in this module including, image filtering, geometric transformation, image histogram calculations, object and feature detection.
- **Opencv_highgui**: This module provides high level graphical interface and media input/output interface, for example, reading and writing image or video files.
- **Opencv_calib3d**: This module provides camera calibration, projecting 3D reconstructing view, two-view geometric estimation and stereo functions.
• **Opencv_features2d**: This module provides common interfaces for feature matches description and detections, object categorization and drawing functions, etc.

• **Opencv_video**: The video module contains methods for motion analysis and estimation, feature tracking and foreground extraction, etc.

• **Opencv_Mi**: OpenCV machine learning module contains a set of classes and functions for statistical classifications, regression, and clustering of data.

### 3.5 Taxonomy of Descriptors

In this section, the various descriptors are discussed based on their computation cost incurred and the amount of memory used for storing by those descriptors. The computer vision in large-scale image registration and recognition has accompanied to an explosion in the portion of data being processed in simultaneous localization and mapping, reconstruction from photo-collections, object recognition, and panorama stitching applications. With the increasing load of data in these applications, the complication of robust features embellishes a hindrance. For instance, keeping high dimensional descriptors in floating-point representation consumes momentous amounts of memory and the time entailed to juxtapose descriptors within large datasets becomes longer. Another factor is the proliferation of camera-enabled mobile devices (e.g. phones and tablets) that have finite computational power and storage space. This further necessitates features that compute in rapid pace are compact in their representation. This new rate of processing has directed various modern works that advocate binary feature detectors and descriptors, auspicious both increased performance as well as compact representation [22].

Feature performance (detection, description, and matching) is critical to countless computer vision applications. The vast scale feature based applications attracts attention regarding their run time necessities while numerous interpretations address the execution of a feature. The Figure 3.10 in next page shows the feature’s computation outlay (detection, description, or matching) and the portion of memory entailed (to store and use).
Corresponding to a real value parameterization, the small-scale applications can repeatedly afford large computational and memory necessities. Techniques in this category bet on a parameterization of an image region, in which every dimension is of a type floating-point (or a discretization of a float, precluding binary). These approaches analyzed in use spatial frequencies, image gradients, etc., to express the local image patch and to test for likeness by using the L2 norm, Mahalanobis distance, etc. The concern such as scale, rotation, viewpoint, or illumination variation can be tackled much effectual by these descriptors. The most popular SIFT is presented in this class. With arise in computation and storage necessities there comes increased complexity and robustness. High performance and parallel hardware (e.g. graphics processors) can be used to mitigate higher computational expenses, but even then, descriptor computation can still be the most time consuming point of a system.

The patch-based descriptors are represented to diminish the computational bottleneck of a system. To straightly indicate it these methods use an image patch surrounding the feature. To calculate the pair equivalence distance measures such as the Sum of Squared Differences (SSD), Normalized Cross Correlation (NCC), or Mutual Information (MI) are used. The pixels in the patch must be reserved, with
quadratically increasing requirements for larger patches. However, in large scale
databases, the major constraints are the matching speed, bandwidth, and the
amount of storage entailed to the descriptor.

The portion that comes next in the taxonomy are the binarized descriptors which
consists of techniques that have high estimation but low storage requirements.
Binarized descriptors rely on hashing approaches to reduce high-dimensional, real-
value parameterizations into compact binary codes. Using the Hamming distance
measure it also fastens the comparison times when considering the compressing
storage constraints. Thus, computational requirements are still high as the full real-
value parameterization must be computed before the hashing can occur.

The final territory in our taxonomy is the binary descriptors. By calculating the pixel-
level comparisons of the descriptor directly, these descriptors have a solid binary
representation and meagre requirements of the computational. This makes them an
appealing solution to many modern applications, particularly for the mobile platforms
and smart glasses where both compute and memory resources are limited. This is
where binary descriptors come in access. Predominantly, Binary descriptors are
composed of three parts they are sampling pattern, orientation compensation and
sampling pairs [23]. Each binary descriptor has their own sampling template, own
method of orientation calculation and its own set of sampling pairs as shown in the
below Table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>Sampling Pattern</th>
<th>Orientation Calculation</th>
<th>Sampling Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIEF</td>
<td>No</td>
<td>No</td>
<td>Random</td>
</tr>
<tr>
<td>ORB</td>
<td>No</td>
<td>Moments</td>
<td>Pairs learned</td>
</tr>
<tr>
<td>BRISK</td>
<td>Concentric circles with additional points on the outer rings</td>
<td>Gradient of long pairs comparison</td>
<td>Apply only short pairs</td>
</tr>
<tr>
<td>FREAK</td>
<td>Overlapping concentric circles with additional points on inner rings</td>
<td>Preselected 45 pairs of gradient comparison</td>
<td>Pairs learned</td>
</tr>
</tbody>
</table>

Table 3.1: Binary Descriptors
4 Concept

Most computer vision issues are rooted in feature matching, and they include structure from motion and object recognition. The presently available methods heavily focus on very expensive descriptors for the matching or detection. However, there is a better yet cheaper method referred to as ORB. This chapter deals with the main working concepts of ORB. This method is quite fast and is a binary descriptor that is based on BRIEF. The best part is that it resists noise and it is rotation invariant [24]. The ORB is used to detect the keypoints in the image captured by the user and to match with the keypoints of the image in the database.

4.1 Concept Architecture

The concept architecture is as shown in the next page in Figure 4.1. When an image is captures by the user from the smart glasses, then the feature point detector is used to detect the keypoints in an image. A feature detector is a sub system of a vision system which detects the certain features presence or absence in a visual scene. Feature detectors are low level front end operations that recognize features by examining a local neighborhood around a pixel. The descriptor is used to describe the region around the keypoints. A descriptor is usually a real valued vector which is of fixed dimension that represents the neighborhood around a feature point. Each descriptor is associated with a keypoint. A descriptor can moreover be binary and de able to extract in a dense manner using a regular grid, but the points of extraction are mainly determined by feature detectors. A brief introduction to keypoints and descriptors are given in Chapter 3. From an image after extracting features and their descriptors then the next step is to establish a preliminary feature matches between the images.

Once the keypoints are being generated, then the descriptor matcher is used to calculate the distances between the keypoints from the user captured image to the reference images stored. There are different types of descriptor matchers. The descriptor matcher selection is based upon the type of descriptors that is used. There should be a threshold value which satisfies the criteria. For each and every query descriptor, the methods discover such training descriptors that the distance is equal
or smaller than the maximum distance in between the query descriptor and the training descriptor. The maximum distance is the threshold between matched descriptors. The distance is measured in metric distance but not the distance between the coordinates which is usually measured in pixels. The data related to the reference image which has the least distance is displayed on the dialog box of the display screen.

![Concept Architecture Diagram](image)

Figure 4.1: Concept Architecture
4.2 Oriented FAST and Rotated BRIEF

Using the SIFT keypoint descriptor and detector has been effective for more than a decade now and has been particularly useful when it comes to some applications that use visual effects such as visual mapping, image stitching, and object recognition. The downside it has is that is occasions a very big computational burden for low-power gadgets such as cellphones and systems based on real-time operations like visual odometry. This downside has led researchers into the search for methods that speed up will speed up SIFT with GPU devices mainly being reducing the computational costs [25].

The alternative to SIFT is the ORB which can be used in real-time performance, does not reduce its effectiveness owing to the image noise, but has a matching performance similar to that of SIFT. The motivation behind ORB is the enhancement of most image-matching applications. In this way, devices with low power needs can be used without the acceleration of GPU in performing patch tracking and panorama stitching. It will also help in reducing the time feature-based detection that is done on standard PCs. This descriptor has a performance similar to SIFT but is faster than SIFT by almost two orders magnitude [26].

Efficiency is the biggest advantage of BRIEF. With only a small amount of computations, it could achieve modest matching performance. BRIEF has its own disadvantages, i.e., it cannot handle with the rotation and scale changes. To overcome these disadvantages, ORB is being proposed for keypoint detection using scale invariant FAST detector and keypoint orientation computation using intensity centroid based method [24]. The reason the feature in proposal is called the ORB is that it is based on the FAST keypoint detector with the BRIEF descriptor thus ORB which stands for Oriented FAST and Rotated BRIEF. Both methods require low costs but have high performance. Here are some limitations of this approach when compared to SIFT.

ORB’s contributions include;
- Adding an accurate and fast component to FAST.
- Better computation of the oriented BRIEF features.
- Analyzing the variance and correlation of the oriented BRIEF features.
- A method to learn the de-correlating BRIEF features to attain better performance as far as nearest-neighbor applications is concerned.
- It is not restricted by the licensing laws put on SURF and SIFT.

**Keypoints**

Among the methods used to identify keypoints in real-time systems which match their visual features (like Mapping and Parallel Tracking) is the use of FAST and some of its variants. FAST is chosen owing to its efficiency and the fact that it finds various corner points. Its weakness is that it ought to be amplified with pyramid schemes for purposes of scale. While most keypoint detectors (such as SURF and SIFT) contain an orientation operator, FAST is an exception to this requirement [27]. The various ways existing of describing the orientation of a keypoint involve the use of histograms of gradient computations such as SIFT while SURF employs the approximation by the block patterns. The methods either yield poor approximations or are computationally demanding. We shall thus borrow from the centroid technique by Rosin as it is among the best methods to measure corners’ orientation. This centroid operator has advantage of giving one dominant result, unlike others which give multiple points [28].

**Descriptors**

One of the latest feature descriptors on the scene is BRIEF which works like SIFT save for the high sensitivity to the in-plane rotation. BRIEF is a result of the use of binary tests in training a number of classification trees. The method employs the training of the trees on a set of typical keypoints then using them in the return of a signature for any arbitrary keypoints. Focus to find the tests that are least sensitive to changes in orientation. The conventional way would put to use the Principal Component Analysis in order to remove the information that is mostly redundant [29]. For this case, however, the PCA method is not enough and so restores to research that is exhaustive. To find uncorrelated binary tests, visual vocabulary methods are put to use. These methods employ offline clustering in finding the examples which are uncorrelated and can thus be employed in matching. Among the closest systems to ORB, that proposes a multi-scale Harris keypoint and is an oriented patch
descriptor is put to use for image stitching, and it shows scale and rotational variance. The problem is that it is not efficient in computation as the method [28].

4.3 FAST Keypoint Orientation

The features of FAST are in wide usage since they have good computational properties. The problem is that they lack an orientation component. ORB adds an orientation that is efficiently computed here.

FAST Detector

The starting point is the detection of FAST points on the image. In this way, FAST will take a single parameter which is the intensity threshold between the pixel at the center and the pixels in a circle about the center. Due to its good performance, we will be considering FAST-9 (circular radius 9). One thing about FAST is its lack of production of cornerness and having large responses at edges. To arrange the FAST keypoints, we will need a Harris corner measure.

![Figure 4.2: Detecting Keypoints Using FAST](image)

We have to lower the threshold if we are looking for N number of keypoints so that we obtain more than N keypoints then use the Harris measure and find the best N points. Given that FAST is not the best in producing multi-scale features, and so add a scale pyramid of the image, and then produce the needed FAST features at every level in the pyramid. The above Figure 4.2 describes the detection of keypoints using
FAST in an image. Out of the 16 pixels in the image, there has to be 9 consecutive pixels which should be either all darker or all brighter in order to consider it as a keypoint. In the first case, there has to be 3 pixels at least from the pixels of 1, 9, 5 and 13. If this satisfies then, it calculates all the remaining pixels to check the necessary condition.

4.3.1 Orientation using Intensity Centroid

The measure for corner orientation called the intensity centroid is both simple and effective. This intensity centroid makes the assumption that the intensity of a corner is offset from its own center. For this reason, the vector can be used in imputing an orientation. According to Rosin, the moments of a patch are:

\[ m_{pq} = \sum_{x,y} x^p y^q I(x,y) \]

The centroid can be found with these moments

\[ C = \left( \frac{m_{10}}{m_{100}}, \frac{m_{01}}{m_{100}}, \frac{m_{00}}{m_{100}} \right) \]

From the corners center, \( O \), to the centroid \( \rightarrow \) we can construct a vector. Then simply the patch orientation is

\[ \Theta = \text{atan2}(m_{01}, m_{10}) \]

Quadrant-aware version of arctan is denoted by \( \text{atan2} \). One should take into cogitation that whether corners are light or dark: however, given that the measures of the angle remain consistent no matter the type of corner, and so this can be of ignored [28].

In improving the rotation invariance for this measure, one has to ensure that \( x \) and \( y \) are within the circular region of the radius \( r \) before computing it. The radius will be the size of the patch and \( x \) and \( y \) will run from \([-r, r]\). In this case, the approach of \( |C| \) towards 0 makes the measure unstable, but this is not the case with FAST corners.
4.4 Rotation-Aware Brief

This section involves discussing about a BRIEF descriptor that is steered and then illustrating how to efficiently compute it. It is a poor performer with rotation. The next step will involve introducing learning steps to finding binary tests that are less correlated and which will lead to the descriptor rBRIEF.

Efficient Rotation of the BRIEF Operator:

Overview of BRIEF

BRIEF descriptors are bit string descriptions of image patches constructed from a given set of binary intensity tests. We shall consider smoothed image patch, \( P \). Binary test \( T \) will be defined by:

\[
T(P; x,y) := \begin{cases} 
1: P(x) < P(y) \\
0: P(x) \geq P(y)
\end{cases}
\]

\( P(x) \) being the intensity of \( P \) at point \( x \). A vector of \( n \) binary test is defined by a feature as

\[
f_n(P) := \sum_{1 \leq i \leq n} 2^{i-1}T(P; x_i, y_i)
\]

A Gaussian distribution about the patch’s center is chosen as the distribution while the vector length \( n = 256 \). Smoothing the image prior to performing tests is important. For smoothing, an integral image is used with test points having a 5 X 5 sub window of a 31 X 31 patch of the pixel [30].

Steered BRIEF

The BRIEF is allowed to be invariant to the in-plane rotation. In matching BRIEF’s performance, it is found to fall off sharply when on an in-plane rotation for more than few degrees. Calonder [12] makes a suggestion to compute a BRIEF descriptor of a given rotation set with perspective wraps for each patch which is an expensive solution. Steering BRIEF in accordance with the orientation of the keypoints is a better method. For any given feature set with \( n \) binary tests at point \( (x_i, y_i) \), we define the \( 2 \times n \) matrix

\[
S = \begin{pmatrix} 
x_1 & \cdots & x_n \\
y_1 & \cdots & y_n
\end{pmatrix}
\]
We can use the patch orientation of $\Theta$ and its corresponding rotation matrix of $R_\Theta$, to construct the “steered” version of $S$ called $S_\Theta$ as follows:

$$S_\Theta = R_\Theta S,$$

The steered BRIEF operator is now:

$$g_n(P, \theta) := f_n(P)(x_i, y_i) \in S_\Theta$$

The discretization of the angle is made at $2\pi/30$ (12 degrees) increments so that we can create a lookup table for the BRIEF patterns that are precomputed. If we keep the keypoint orientation $\Theta$ consistent across all views, we can compute the correct point set $S_\Theta$.

4.4.1 Variance and Correlation

The varied response of the high variance to inputs will make the feature more discriminative. To have each test contribute to the desired result, we can have the tests made not to correlate. For analysis of the variance and correlation of the tests BRIEF vectors, he authors had a look at the responses to the 100k keypoints both for the BRIEF and the steered BRIEF. Both Steered BRIEF and BRIEF show high original eigenvalues, signifying correlation in binary tests. This means that all information is in 10 to 15 of the first components. However, the steered BRIEF has lower variance but is not discriminative since its eigenvalues are lower. For it to perform well, BRIEF relies on the random orientation of the keypoints.

4.4.2 Learning the Good Binary Features

Recovering from loss of variance in the steered BRIEF and reducing the correlation in the binary tests will require developing a learning method to enable us choose good subsets of the binary tests. A good strategy is using PCA or other dimensionality-reduction methods to start from large sets of binary tests then identifying 256 of the new features with high variance correlation over large training sets. Given that these new features are made from larger numbers of binary tests, their computational efficiency is less than that of steered BRIEF. For this reason, all the binary tests will be searched that are possible so that the ones with both high
variance and no correlation can be found. They should have means as close to 0.5 as possible.

The method which is to be used is as follows. Start with setting up training sets of around 300k keypoints which are drawn from the images from the PASCAL 2006 set. All the possible binary tests that are drawn from the 31 X 31 pixel patch will be enumerated. Each of the tests is a pair of the 5 X 5 sub-windows of a patch. Let our patch’s width be $w_p = 31$ while the width of each test sub-window will be $w_t = 5$. We thus get $N = (w_p - w_t)^2$ as the possible number of sub-windows. Given that we have to select pairs, we will use $\binom{N}{2}$ binary tests. It is required that overlapping tests are eliminated. That will leave with $M = 205590$ conceivable number of tests.

The algorithm becomes:

1. Running every test against the training patches.
2. Ordering the tests basing on their distance from the mean of 0.5 thus creating our vector $T$.
3. Carrying out a Greedy search:
   a. Putting test $q$ into results of vector $R$ then removing it from $T$.
   b. Taking next test from $T$ then comparing it against all the tests in $R$. If it has absolute correlation that is larger than the threshold then discard it. It has to add it to $R$ if not.
   c. Repeating the previous steps until we get 256 tests in our $R$. we try again with a higher threshold if the tests are less than 256.

**Flow Chart of ORB:**

According to the rules of the ORB algorithm, we can get the flow chart on the next page as shown in Figure 4.3. The pyramid makes a set of reduced images and performs ORB on each of them. When the object gets nearer or more remote from the camera it abet to track the points over the scale changes. For ex, a point can be observed on the first, unresized image then the object gets closer, and one can find the same keypoint on the second layer of the pyramid because it’s larger. The ORB generates a 256 bits descriptor values for each keypoint in an image.
Figure 4.3: Flow Chart of ORB

START

Find the position of the keypoints by FAST

Selecting N best points by Harris

Scale-pyramid transform

Add a direction of the points in Intensity Centroid

Extracting Binary descriptor by BRIEF

Get Steered BRIEF

Find low correlative pixel blocks in greedy algorithm

Receive a 256-bit descriptor
Overview of concept:

In this section, the overview of the concept is been shown in the below Figure 4.4. When the user wears the smart glasses and captures, any safety sign on his way then in the first stage the image needs to be preprocess and then the keypoints needs to detect using any feature point detector like FAST/ORB/BRISK. The descriptors like BRIEF/ORB/BRISK are used to describe the regions around the keypoint. We have to check with some of the detectors and binary descriptors which provide us with the most efficient solution without using much computational task on these smart glasses.

Figure 4.4: Overview of Concept
In a few cases, we have to use some of the detectors with the binary descriptors as they do not detect any keypoints on the image. We have to use FAST detector for BRIEF descriptor as BRIEF is only a descriptor and for finding a keypoints, we have to use the available other detectors. BRIEF in combination in with FAST produces good results. ORB and BRISK has its own detector and descriptors. The thesis is carried out with the various binary descriptors and a comparative analysis is given on the time and the power measurements of various descriptors when used in the smart glasses. The binary descriptors such as BRIEF / ORB / AKAZE / BRISK / FREAK are taken into account to measure their efficiency on the smart glasses. The descriptor whose results are much efficient when compared with the other descriptors is been selected. The results of the various detectors and descriptors when used in combination are shown in the later chapters with respect to their power measurements and the time readings in detecting an image.
5 Implementation and Realization

In the previous chapter, the concept to implement the system and the working principle of the ORB algorithm is discussed. This chapter shows the implementation part of the thesis. Selecting proper hardware and software is very important to implement the system in the beginning. The hardware and software are selected based on its cost, performance, usability and availability.

5.1 Realization of Hardware and Software

In terms of hardware selection, a smart glass should be cost effective, easily available and user convenient. The reason behind selecting the Vuzix M100 device is because of its usability. It requires a lot of time to train on the Google Glass device. Even though Glass is an amazing device to use but its interface is entirely different from M100. The battery can be quickly drained in Glass, and it cannot sustain for much more time when compared to M100. In Glass, there will be no indication when clicking the pictures which makes quite uncomfortable when using it. The installation of applications and working on the M100 is easy when compared to Google Glass as it requires few other additional steps for installing the applications into it. The main drawback of Google Glass is that the overheating of the battery issues and the camera cannot be tilted to view and it is in a fixed position. To sign into the websites Glass doesn’t buttress the use of forms and it even can’t support for adobe flash.

The advanced technique of computer vision is the image processing. Many tools are available for digital image processing. Among them, the selection of proper tool is which can deliver high performance with the low cost. The tool should be user friendly also. OpenCV4Android and Matlab are the most popular image processing tools among them. Matlab is very popular and widely used now a day. The ordinance of this tool can deliver many advantages. So many algorithms for image processing are already implemented in Matlab and memory management is done by self. The ordinance of this tool is immensely simple, and it can be learned very easy. Even though it has some limitations which are that the tool is very costly. The license cost is around 2000 USD. Again it makes the system slow taking top priority from the operating systems. In terms of image processing, it can process only four or five
frames per second where OpenCV4Android can process more than thirty frames. OpenCV4Android is free and has development pack. The target of this thesis is to develop low cost system which provides high performance in object recognition. So OpenCV4Android is selected for the implementation of the system. A comparison between OpenCV4Android and Matlab are given below Table 5.1.

<table>
<thead>
<tr>
<th>OpenCV</th>
<th>MATLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is Faster in image processing</td>
<td>MATLAB is slow</td>
</tr>
<tr>
<td>It is specific to image processing</td>
<td>It is general purpose tool</td>
</tr>
<tr>
<td>It delivers high performance in terms of</td>
<td>It only process 4 to 5 frames per second</td>
</tr>
<tr>
<td>speed with the processing 30+ frames per</td>
<td></td>
</tr>
<tr>
<td>second</td>
<td></td>
</tr>
<tr>
<td>Different IDE can be used for</td>
<td>There is no other choice other than Matlab editor</td>
</tr>
<tr>
<td>OpenCV4Android such as eclipse, visual</td>
<td></td>
</tr>
<tr>
<td>studio, and Qt. This gives more choice to</td>
<td></td>
</tr>
<tr>
<td>developer.</td>
<td></td>
</tr>
<tr>
<td>In terms of cost, OpenCV4Android is BSD</td>
<td>Matlab license can cost around</td>
</tr>
<tr>
<td>licensed, which is free</td>
<td>$2000</td>
</tr>
</tbody>
</table>

Table 5.1: OpenCV4Android and Matlab Comparison

**Image Recognition Techniques:**

The selection of an image recognition technique is based on the several criteria like using low computational cost and power on the smart glasses and the time taken to display the results needs to be faster. Here I will be discussing in short about some of the other techniques that can be used but are not much efficient on smart glasses.

**Deep Learning:**

The neural networks with many hidden layers can be referred to as deep learning or deep neural network. The deep learning can be learned by using a back propagation method. Deep learning is a wide range family of machine learning methods that seeks to learn high level features from the given data. The chance of not working with the deep learning is because of it is extremely computationally expensive to train. It requires expensive GPUs which smart glasses are not much capable of
handling it. It takes weeks to train the most complex models using hundreds of machines equipped with high cost GPUs. Deep learning requires large amount of data to train. It is very difficult to understand what it is learned.

**Template Matching:**

Many computer vision errands such as object detection / recognition, object comparison, and computation depth can be uniquely resolved with template matching techniques. The template matching depends on the probability and statistics, signal processing and physics (imaging). The identification of the parts on an image that matches with a predetermined template is the identical template which is a high level machine vision method. It is a technique for finding areas of an image that match to a template image (patch) or it can help to locate certain features in a given image. Template matching works by drifting the template across the original image [31]. As it drifts, it compares or matches the template to the part of the image directly under it. It does by matching the calculating a number. This number denotes to what range that the template provided and the part of the original is equivalent. The defined number depends on the calculation used. Some denote a complete match by a 0 or a 1 (Indicating a complete match). When a template matching is performed in OpenCV, an image will be obtained which displays the degree of similarity or correlation between the portion and the portion under the template. The greater the intensity, the greater is the correlation between the template and the portion. As we anticipate discovering it in the image the template indicates the object.

There are some obvious flaws in template matching as their applicability is limited mostly by the available computational power, as the identification of big and complex templates can be much more time consuming. As a model for perception the contexts are hardly compelled is the complication with template matching. The object can be scaled certainly or rotated. This technique requires a separate template for each scale and orientation which make it completely outrageous [32]. The template matching is also sensitive to noise and occlusions. Illumination alteration affects images in a complex way, cutting down the effectiveness of template matching techniques. Even with the replication and normalization, it would be difficult to represent the third dimension with template matching. The calculation of NCC is
highly ineffectively computationally. Template matching is a rarely used technique. Instead of using template matching the can feature detectors and descriptors can be used for generating much efficient results.

**Feature Based Methods:**

The feature based methods consumes low computational cost, power and even display the results in a short span of time which makes it more efficient in terms of hand held devices and the smart glasses. The computational cost incurred by it on the device architecture is well suited even with low GPUs. The below are the more points which distinguish the feature based techniques from other image recognition techniques.

Locality: Features are local and are so robust to occlusion and clutter.

Distinctiveness: A large database of objects can be differentiated.

Efficiency: Real time performance can be achievable.

Generality: Exploit different types of features in various situations.

Feature based methods are important in the sense that they can

(i) Summarize the image information because they can discard image regions that are less important such as the flat non-varying surfaces.

(ii) Correspondence problems can be solved efficiently.

(iii) Can work without learning, all that is required is a database of model descriptors and a fast matching technique.

(iv) They tend to be righteousness for instance level object recognition.

Thus these systems are appealing for real-time image recognition such as in cases of camera pose estimation useful in augmented reality and virtual reality.

**5.2 Implementation Work**

**Android SDK:**

It is a software development kit that allows the developers to create applications for the Android platform. The Android SDK comes with a package of sample projects
with source code, tools for the development, an emulator, and required libraries to develop an Android application. Java programming language is used to write the applications and run on Dalvik, a custom virtual machine which is designed for embedded use that runs on top of a Linux kernel. The compilation of the code along with any data and resource files is done by the Android SDK tools into an APK, an Android package, which is an archive file with the suffix of .apk. All the contents of an android app is in a single APK file that Android-powered devices use to install the app.

Before starting an app component by the Android system, the system should aware that the components manage by reading the app’s manifest file, AndroidManifest.xml. All the components of the app must be declared in this file, which present at the root of the app project directory.

In addition to declaring the app’s components the manifest does a lot of things, such as the following stated below

- The user permission required by the app is identified; such as only read access the contacts of the user’s or the internet access.
- The least API level entailed by the app is declared, based on which APIs the app uses.
- The hardware and software features such as camera, Bluetooth services which are required for the app are declared.
- The API libraries are declared which the app requires to be linked against, such as the Google Maps Library.

```xml
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="21" />

<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus"
    android:required="false" />
<uses-feature android:name="android.hardware.camera.flash"
    android:required="false" />
```

Figure 5.1: Permission of AndoridManifest file
The Figure 5.1 in previous page shows the permission declared in the AndroidManifest.xml file. Inspite preventing the app from being installed on the devices that lack features required by the designed app, it’s important that the designer certainly point a profile for the kind of devices the app supports by declaring device and requirements for the software in the manifest file. The device which does not have a camera and whose Android version is lower than 2.2 cannot install the application.

The Android SDK provides an emulator for the developers to easily test the applications without being installed it on the real devices or even having one. With the proper configuration of the emulator, it is feasible to test the situations which are hardly seen on a physical one. The working of an app on the emulator is bit slower when compared with the real hand device. The display screen of an Android emulator is as shown in the below Figure 5.2. One has to make sure that the app have assigned permission for the webcam while creating an emulator.

Figure 5.2: Display Screen of an Android Emulator
Detection of Keypoints:

The below code in Figure 5.3 is used to detect the keypoints in an image. In this code, the ORB is used for both the detector and the descriptor.

```java
// A feature detector, which finds keypoints in an image.
private final FeatureDetector mFeatureDetector = 
    FeatureDetector.create(FeatureDetector.ORB);

// A descriptor extractor, which creates descriptors of
// keypoints.
private final DescriptorExtractor mDescriptorExtractor = 
    DescriptorExtractor.create(DescriptorExtractor.ORB);

// A descriptor matcher, that matches keypoints based on their
// descriptors.
private final DescriptorMatcher mDescriptorMatcher = 
    DescriptorMatcher.create( 
        DescriptorMatcher.BRUTEFORCE_HAMMINGLUT);
```

Figure 5.3: Code for Detecting the Keypoints

The resultant images from the above code using ORB is to detect the keypoints in an image and creating the descriptors. Here is some of the example shown in the Figure 5.4 for resultant images generated by using ORB. Different detectors use different principles to detect the keypoints in various regions. The blue circle on the image represents the keypoints that are being generated using ORB detector.

Figure 5.4: Keypoints Detection using ORB
We will be using combinations of techniques that OpenCV calls FeatureDetector.ORB for detector, DescriptorExtractor.ORB for descriptor, Descriptor Matcher.BRUTEFORCE_HAMMINGLUT for matcher. This combinations is relatively fast and robust. Unlike some alternatives, it can track the objects at various magnifications, resolutions or distances, and at various angles of view.

**Brute Force Matching:**

The brute force matcher is unpretentious. It considers the descriptor of one feature in first set and is matched with the all other features in the second set using a distance calculation, and the closest one is returned. As the ORB is a binary descriptor which comprises of 1’s and 0’s we use BRUTEFORCE_HAMMINGLUT for comparing the two descriptors. It takes very less amount of time for calculation as it is simply based on the XOR operation. The number of situations at which there is a difference in corresponding numbers between the two descriptors is the hamming distance between them.

The default length of each ORB descriptor is 32 bytes where each byte contains 8 pixel intensity comparisons. An ORB is a 256 bit descriptor which is of 32 bytes for each and every descriptor. If it detects a keypoints of 450 on image, then it gives us 450 * 32 byte descriptors. The BRUTEFORCE_HAMMINGLUT then compares the descriptor of one image to all the other descriptors in the second image. Then a matching is being performed on both the descriptors. It just count the total number of bits where the strings differ, or identically applying sum(xor(string1, string2)).

The helper method of findSceneCorners() is a bigger complex of code, but a lot of it merely iterates through the matches to build a list of the best ones. As designated by a larger distance value if all the matches are really bad, suspect that the target image is not in the scene and its corner locations of any previous estimates have to clear. On the off chance that the matches are neither great nor awful, we speculate that the objective is may be some place in the scene, yet we keep our previous estimates of its corner areas. This policy assists us to stabilize the estimate of the corner locations. Find the homography and the estimated corner points are updated using it and only if at least there are four good matches.
final List<KeyPoint> referenceKeypointsList = mReferenceKeypoints.toList();
final List<KeyPoint> sceneKeypointsList = mSceneKeypoints.toList();

// Calculate the max and min distances between keypoints.
double maxDist = 0.0;
double minDist = Double.MAX_VALUE;
for (final DMatch match : matchesList) {
    final double dist = match.distance;
    if (dist < minDist) {
        minDist = dist;
    }
    if (dist > maxDist) {
        maxDist = dist;
    }
}

// The thresholds for minDist are chosen subjectively
// based on testing. The unit is not related to pixel
// distances; it is related to the number of failed tests
// for similarity between the matched descriptors.
if (minDist > 50.0) {
    mSceneCorners.create(0, 0, mSceneCorners.type());
    return false;
} else if (minDist > 25.0) {
    return false;
}

// Identify "good" keypoints based on match distance.
final ArrayList<Point> goodReferencePointsList =
    new ArrayList<Point>();
final ArrayList<Point> goodScenePointsList =
    new ArrayList<Point>();
final double maxGoodMatchDist = 1.75 * minDist;
for (final DMatch match : matchesList) {
    if (match.distance < maxGoodMatchDist) {
        goodReferencePointsList.add(
            referenceKeypointsList.get(match.trainIdx).pt);
        goodScenePointsList.add(
            sceneKeypointsList.get(match.queryIdx).pt);
    }
}

if (goodReferencePointsList.size() < 4 ||
    goodScenePointsList.size() < 4) {
    return false;
}

Figure 5.5: Finding Good Keypoints
62
The source image matrix, the destination image matrix and the type of color conversion are the three parameters that is taken into account by the method cvtColor(). It is used to convert an image from one color space to another. The below code is used for drawing the matches from the user captured image to the reference image which has the least distance between the matched descriptors. The match of keypoints from two images in the output image is drawn using this function. A match is a line connecting two keypoints from one image to the another image. It takes the following parameters mSrcBGR is the first source image, mSceneKeypoints is the keypoints in the first source image, mReferenceImageBGR is the second source image, mReferenceKeypoints is the keypoints in the second source image, mMatches indicates the matches from the first image to the second image, which means that the keypoints1[i] has a corresponding point in keypoints2[matches(i)]. mDstBGR is the output image that has been generated. It consists of lines between one image to the another image.

```
Imgproc.cvtColor(dst, mSrcBGR, Imgproc.COLOR_RGBA2BGR);

Features2d.drawMatches(mSrcBGR, mSceneKeypoints,
                      mReferenceImageBGR, mReferenceKeypoints, mMatches,
                      mDstBGR);
```

**Figure 5.6: Code for Drawing Matches**

Here are some examples of the output images that have been shown in the Figure 5.7 which is generated after matching and is stored in the gallery. The image on the left side is the user captured image and the image on the right side is the image which is recognized based on the descriptor matched values.
Between a reference image and the matching image in the scene detect the homography. A homography is a 3D transformation that would be essential to line up the two projected 2D images (or come as close as conceivable to lining them up). It is measured depending on the two images matching feature points. The points which exactly match with the other image are called as the inliers and the points match but with the other points in the image can be referred to as the outliers. The outliers may cause some of the destruction while recognizing the image. The RANSAC is employed to eliminate the outliers. The each element of src is transformed by the function perspectiveTransform by contemplating it as a 2D or 3D vector in the following way.

\[
(x, y, z) = \left(\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}\right)
\]

where \((x', y', z', w') = \text{mat.} [x \; y \; z \; 1]\)

and \(w = \begin{cases} w' & \text{if } w' \neq 0 \\ \infty & \text{otherwise} \end{cases}\)
5.3 Flowchart for Implementation:

The implementation part that is discussed in the previous sections is drawn in the Figure 5.8 for easier approach.

Figure 5.8. Flowchart for Implementation
The below code in the Figure 5.9 is used to convert the mat to a bitmap and to store the matching image into the SD card of the M100 device. The image is stored in the folder “Gallery” with the current date and timestamp. We can just have a cross look into those images for matching.

```java
final String TAG = "TagMessage";

Bitmap bmp = null;
try {
    bmp = Bitmap.createBitmap(mDstBGR.cols(), mDstBGR.rows(), Bitmap.Config.ARGB_8888);
    Util.matToBitmap(mDstBGR, bmp);
    Log.d(TAG, "Executed");
} catch (Exception e) {
    Log.d(TAG, e.getMessage());
}

mDstBGR.release();

FileOutputStream out = null;

String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());

String filename = "frame_" + timeStamp + ".png";

File sd = new File(Environment.getExternalStorageDirectory() + "/Gallery");
boolean success = true;
if (!sd.exists()) {
    success = sd.mkdir();
    Log.d(TAG, "Directory Found");
}
if (success) {
    File dest = new File(sd, filename);
    try {
        out = new FileOutputStream(dest);
        bmp.compress(Bitmap.CompressFormat.PNG, 100, out);
        Log.d(TAG, "Image Created");
    } catch (Exception e) {
        e.printStackTrace();
        Log.d(TAG, e.getMessage());
    } finally {
        try {
            if (out != null) {
                out.close();
                Log.d(TAG, "OK!!");
            }
        } catch (IOException e) {
            Log.d(TAG, e.getMessage() + "Error");
            e.printStackTrace();
        }
    }
}
```

Figure 5.9: Code for Storing the Resultant Image
6 Results and Analysis

In this chapter, we see the results obtained from complete thesis work and the analysis of various feature descriptors on the smart glasses. The Vuzix M100 smart glasses is been used in this project. In the first section, I will be presenting the screenshots of the dialog box obtained after capturing the images on the smart glasses and in the next section I will be presenting the time measurements for recognizing an image followed by the power consumptions of each descriptor on the smart glasses.

6.1 Recognition of Various Safety Signs on Smart Glasses

When the user wears the smart glasses and captures any of the safety signs on his way, then the glasses needs to display a dialog box representing the meaning of the safety sign. The descriptor matcher has to return the data related to the image which has the least distance when compared with all the other images. Whenever the user sees a safety sign, then he can click on the “TAKE PHOTO” option so that it captures the image. I tried working with various descriptors on the smart glasses for recognizing the various safety signs and here are the resultant screenshots after recognizing a safety sign.

The below image set shows the safety sign that has been captured from the camera of the safety glasses and the resultant dialog box that has been displayed. The below Figure 6.1 set shows that if a below image is captured then a dialog box which contains “No Smoking” has to be displayed.
The below Error! Reference source not found. set shows that if a below image is captured then a dialog box which contains “Wear Safety Shoes” has to be displayed. When the user clicks on the “OK” button, the app resumes its operation further.
The below Figure 6.3 set shows that if a below image is captured then a dialog box which contains “No Entry” has to be displayed. When the user clicks on the “OK” button, the app resumes its operation further. As some of the descriptors like ORB, BRISK are rotation invariant no matter how the image is rotated. It detects the matches even if the image is rotated to an angle of 270 degrees. The time taken to display the dialog box needs to be fast and the time taken to resume the next cycle needs to be short without hanging the app.

![Image of No Entry dialog box](image1.png)

Figure 6.3: Set 3 for Image Recognition Signs

The below Figure 6.4 set shows that if a below image is captured then a dialog box which contains “Wear Safety Jacket” has to be displayed. When the user clicks on the “OK” button, the app resumes its operation further. The primary intention of this project is to minimize the mistakes the trainees are able to do without having any knowledge on what does the safety sign represents for. I have shown only some safety signs for an idea of what it is as a part of this documentation.
6.2 Time Readings of various Descriptors

In this section, I will be presenting the time measurements of various descriptors. The time in between the user clicking on the “Take Photo” option and the display of the dialog box on the screen is taken into account. Once the user captures an image, then the algorithm needs to take loss amount of time to display the results as the user can’t wait for much more time. For this case, a time measurement readings are considered for selecting a best algorithm for the smart glasses.

**BRIEF Descriptor:**

The Figure 6.5 in next page shows the time measurements of FAST detector and a BRIEF descriptor. As BRIEF is only a descriptor, I tried using with the FAST detector along with the BRIEF descriptor to produce good results. The app works fine in every
cycle, but the drawback with this is that it is not rotation invariant making it less attractive. The time readings are at a moderate level on the smart glasses. The average time taken to detect an image is around 1 second.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 029 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1000 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1245 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 997 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1046 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1155 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1110 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 076 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 964 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 926 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1170 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and BRIEF Descriptor: 1224 ms</td>
</tr>
</tbody>
</table>

**Figure 6.5: Time Readings of FAST Detector and BRIEF Descriptor**

**FREAK Descriptor:**

The below Figure 6.6 shows the time readings of FAST detector and a FREAK descriptor. As FREAK is also only a descriptor, I tried using with the FAST detector along with the FREAK descriptor. Even though the app works on the smart glasses it is still not enough to enable on the smart glasses as the launching of an app is consuming much more time and for resuming the next cycles. The time readings are at a higher level on the smart glasses. The average time taken to detect an image is around 1.5 to 2 seconds. FREAK doesn’t stays in the category of selecting it as it is taking too much of time on the smart glasses.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 2557 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1331 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1170 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1647 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1422 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1666 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1480 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 3328 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 3102 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1525 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1516 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for FAST Detector and FREAK Descriptor: 1781 ms</td>
</tr>
</tbody>
</table>

**Figure 6.6: Time Readings of FAST Detector and FREAK Descriptor**
**ORB Descriptor:**

The below Figure 6.7 shows the time readings of ORB detector and descriptor. The app works great on the smart glasses. The time taken to launch an app and the time taken to resume the further cycles while recognizing the images is too short. The time readings are at a much lower level on the smart glasses. The average time taken to detect an image is around less than 1 second. The working of ORB descriptor is well suited for the small resource devices like mobile devices and the smart glasses. The ORB doesn’t need much computational cost on the smart glasses.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 824 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 833 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 910 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 912 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 1026 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 877 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 930 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 830 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 864 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 1079 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 934 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 1074 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 957 ms</td>
</tr>
<tr>
<td>Status</td>
<td>Estimated Time for ORB Detector and ORB Descriptor: 991 ms</td>
</tr>
</tbody>
</table>

*Figure 6.7: Time Readings of ORB Detector and Descriptor*

**BRISK and AKAZE Descriptors:**

When I tried using with the BRISK descriptor and the AKAZE descriptors, the app is not working fine in a normal way on the smart glasses. The app takes too much of time to respond and even terminates in the middle of the process. The BRISK and AKAZE needs high GPUs in order to run much effectively on a device. The launching of an app takes nearly 20 seconds of time on the smart glasses. The BRISK and AKAZE descriptors are not optimized to mobile GPUs. As the smart glasses are equipped with a limited resource functionality, they don’t have enough architecture to make the BRISK and AKAZE descriptors work fine and to generate much effective results when installed on it. As an outcome of this least performance on the smart glasses, I thought of not considering it to any further in selection of an algorithm.
The below Figure 6.8 shows the comparison of time measurements of various descriptors on the smart glasses. The peak time in the graph is due to the lightening illusion occurred due to the viewport angle. When compared with the other descriptors as discussed ORB is much faster than the other descriptors. ORB outperforms the other binary descriptors as they work much efficient on the smart glasses.

![Figure 6.8: Time Graph of various Descriptors](image)

### 6.3 Power Measurements of various Descriptors

In this section, I will be showing the power measurements of various descriptors when used on the Vuzix M100 smart glasses for a time frame of 5 mins. I have worked with every descriptor for a period of 5 mins and captured 15 images on the smart glasses and placed the app in an idle state for the remaining time frame. The algorithm which consumes low power when operated on the smart glasses is also taken as a criterion along with the low power measurements. As the glasses don't lasts for more time, the selected algorithm should consume low power when operated on it.

The App-Tune up Kit application is used to measure the power consumed by an app. The power measurements are recorded by discharging the power cable to the smart glasses.
BRIEF Descriptor:

The power measurements of BRIEF descriptor is as shown in the below Table 6.1. The App Tune-up Kit gives us a detailed report of the power measurements including the peak power readings. The average power and peak power consumption are taken into the account. The CPU load is average for BRIEF descriptor, and the average power measurement is good when working with the smart glasses. The battery of the smart glasses is charged to 100% before starting of this application.

<table>
<thead>
<tr>
<th>Application</th>
<th>BRIEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP CATEGORY</td>
<td>Not set</td>
</tr>
<tr>
<td>DURATION</td>
<td>5 min 0 sec</td>
</tr>
<tr>
<td>DEVICE</td>
<td>VUZIX M100</td>
</tr>
<tr>
<td>PROCESSOR MODEL</td>
<td>OMAP4</td>
</tr>
<tr>
<td>AVERAGE CPU LOAD (PROFILED APP)</td>
<td>19.5%</td>
</tr>
<tr>
<td>MAX CPU LOAD (PROFILED APP)</td>
<td>54.0%</td>
</tr>
<tr>
<td>AVERAGE CPU LOAD (NORMALIZED)</td>
<td>19.4%</td>
</tr>
<tr>
<td>MAX CPU LOAD (NORMALIZED)</td>
<td>50.0%</td>
</tr>
<tr>
<td>AVERAGE CPU LOAD (SYSTEM)</td>
<td>40.2%</td>
</tr>
<tr>
<td>MAX CPU LOAD (SYSTEM)</td>
<td>86.0%</td>
</tr>
<tr>
<td>AVERAGE CPU1 LOAD</td>
<td>40.3%</td>
</tr>
<tr>
<td>MAX CPU1 LOAD</td>
<td>91.0%</td>
</tr>
<tr>
<td>AVERAGE CPU2 LOAD</td>
<td>40.2%</td>
</tr>
<tr>
<td>MAX CPU2 LOAD</td>
<td>93.0%</td>
</tr>
<tr>
<td>AVERAGE POWER*</td>
<td>682 mW</td>
</tr>
<tr>
<td>PEAK POWER*</td>
<td>984 mW</td>
</tr>
<tr>
<td>INITIAL BATTERY STATUS</td>
<td>Discharging</td>
</tr>
<tr>
<td>SCREEN BRIGHTNESS</td>
<td>100%</td>
</tr>
<tr>
<td>SCREEN RESOLUTION</td>
<td>432 x 244</td>
</tr>
</tbody>
</table>

Table 6.1: Power Measurements of BRIEF Descriptor
FREAK Descriptor:

The power measurements of FREAK descriptor is as shown in the below Table 6.2. The App Tune-up Kit gives us a detailed report of the power measurements including the peak power readings. The average power and peak power consumption are taken into the account. The peak power consumption is high for the FREAK descriptor, and the average power measurement is of maximum amount when compared with the other descriptors. The FREAK is no longer in selection criteria in choosing an algorithm for the smart glasses.

| APPLICATION | FREAK |
| APP CATEGORY | Not set |
| DURATION | 5 min 0 sec |
| DEVICE | VUZIX M100 |
| PROCESSOR MODEL | OMAP4 |
| AVERAGE CPU LOAD (PROFILED APP) | 32.5% |
| MAX CPU LOAD (PROFILED APP) | 51.8% |
| AVERAGE CPU LOAD (NORMALIZED) | 23.2% |
| MAX CPU LOAD (NORMALIZED) | 46.0% |
| AVERAGE CPU LOAD (SYSTEM) | 50.5% |
| MAX CPU LOAD (SYSTEM) | 98.0% |
| AVERAGE CPU1 LOAD | 42.9% |
| MAX CPU1 LOAD | 100.0% |
| AVERAGE CPU2 LOAD | 58.2% |
| MAX CPU2 LOAD | 100.0% |
| AVERAGE POWER* | 710 mW |
| PEAK POWER* | 1,153 mW |
| INITIAL BATTERY STATUS | Discharging |
| SCREEN BRIGHTNESS | 100% |
| SCREEN RESOLUTION | 432 x 244 |

Table 6.2: Power Measurements of FREAK Descriptor
ORB Descriptor:

The power measurements of ORB descriptor is as shown in the below Table 6.3. The average power and peak power consumption are taken into the account. The peak power consumption is good for the ORB descriptor, and the average power measurement is at a low amount when compared with the other descriptors. ORB consumes low power when it is operated on the smart glasses and produces much effective results.

<table>
<thead>
<tr>
<th>App Tune-up Kit - Detailed Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION</td>
</tr>
<tr>
<td>APP CATEGORY</td>
</tr>
<tr>
<td>DURATION</td>
</tr>
<tr>
<td>DEVICE</td>
</tr>
<tr>
<td>PROCESSOR MODEL</td>
</tr>
<tr>
<td>AVERAGE CPU LOAD (PROFILING APP)</td>
</tr>
<tr>
<td>MAX CPU LOAD (PROFILING APP)</td>
</tr>
<tr>
<td>AVERAGE CPU LOAD (NORMALIZED)</td>
</tr>
<tr>
<td>MAX CPU LOAD (NORMALIZED)</td>
</tr>
<tr>
<td>AVERAGE CPU LOAD (SYSTEM)</td>
</tr>
<tr>
<td>MAX CPU LOAD (SYSTEM)</td>
</tr>
<tr>
<td>AVERAGE CPU1 LOAD</td>
</tr>
<tr>
<td>MAX CPU1 LOAD</td>
</tr>
<tr>
<td>AVERAGE CPU2 LOAD</td>
</tr>
<tr>
<td>MAX CPU2 LOAD</td>
</tr>
<tr>
<td>AVERAGE POWER*</td>
</tr>
<tr>
<td>PEAK POWER*</td>
</tr>
<tr>
<td>INITIAL BATTERY STATUS</td>
</tr>
<tr>
<td>SCREEN BRIGHTNESS</td>
</tr>
<tr>
<td>SCREEN RESOLUTION</td>
</tr>
</tbody>
</table>

Table 6.3: Power Measurements of ORB Descriptor
The power measurements of BRISK and AKAZE are not presented here as they are not an efficient solution for the smart glasses. The power measurements of AKAZE and BRISK algorithms are too high when compared with the ORB, BRIEF, and FREAK. The BRISK and FREAK are outrageous in terms of performance and power consumptions.

6.4 Discussion

In this section, I will be discussing about the performance and efficiency of the descriptors. More or less the descriptors like FREAK, BRISK, and AKAZE, performs in a similar way on the smart glasses. The time take to display the results, and the amount of power consumption is too high. The computational cost incurred by these descriptors on the smart glasses is not well suited. These descriptors require high GUIs in order to process them more efficiently. The descriptors like BRISK and AKAZE requires much more time to load the application and sometimes they even freeze the application as they are computationally heavy. They might be effective if they are used on personal computers or in any other application but not on the limited GPU devices. Using these descriptors on the smart glasses and even on a mobile devices is not a much efficient solution for our problem.

ORB outperforms all the other descriptors like BRIEF, FREAK, BRISK and AKAZE in terms of low time readings and low power measurements. BRIEF, when used with FAST, is not a rotation invariant, but ORB is a rotation invariant. As ORB is built on the modifications of FAST and BRIEF, it exhibits much performance. ORB is mainly targeted for the low power devices like mobile phones, and at the same time, it can be used on the smart glasses too as it is similar to the low end mobiles. It can recognize the images even they are rotated to a much higher degree. The time taken to detect the images is around 1 sec as stated in earlier and the amount of power consumed by the ORB on the smart glasses is also of much lower. ORB uses low computational power on the devices. It works more efficient on the low GUI devices. It’s usage on the smart glasses is much efficient than the other algorithms. The app can even resume the further cycles for recognizing the images in a less time frame. Thus, based on the work carried out I would like to conclude that the ORB is a much efficient algorithm for the smart glasses.
6.5 Areas of Application

In this section, I will be discussing about the areas of application in which it can be used. The image recognition technique suggested in this thesis is used in the smart glasses for image recognition. There will be a lot of safety signs on the walls and on the doors of the production hall. The trainees who work in the hall are unaware of those safety signs which lead to a lot of mistakes committed by them. The manager or the supervisor of the trainees can’t be with them all the time. It costs them extra time and work for explaining each and every safety sign for every student. When the trainee wears the smart glasses and works in the production hall or in the lab work and if he captures the safety sign then he knows what the sign is for and what he has to do in order to obey the rules. The smart glasses with an efficient image recognition algorithm serves the best for the trainees in every aspect of learning as it doesn’t consume extra time of the supervisors.

In the workshops, it can be much more helpful if the students are learning about the PLCs, various components and parts of the automobile. The smart glasses can be used in a much efficient way in order to reduce the time of the supervisor for explaining what the part is in a PLC unit. The selected algorithm can even detect a 3D image which is located in front of the user. This advantage has made it to integrate the usage of smart glasses in this field. The trainees can automatically learn themselves by wearing the smart glasses which improves there memory by seeing it in an Augmented Reality projection. The AR projection helps the trainees to remember the parts much easier. The parts of a PLC or the components of the automobile are recognized in a much understandable way for the trainees.
In this chapter, I would like to give a conclusion on my thesis. The thesis was started with the goal of developing an image recognition application for recognizing various safety signs on the smart glasses and to provide an efficient solution in terms of speed and power consumption. Vuzix M100 smart glasses are used in the thesis. Java with OpenCV4Android and Android SDK and ADT tools are used to develop the application for image recognition. M100 drivers are used along with System File Manager to install the application on the M100 device. Apart from the smart glasses and its component software’s, the remaining software’s are selected as they are open source and free of cost. Especially OpenCV4Android is selected for development as it is open source platform dedicated for image processing and Android SDK is used to develop an android application.

There are many approaches proposed by the researches in the field of image recognition. The thesis aim was also to select the best image recognition technique by keeping the architecture and limited resource capacity of smart glasses in mind. For this, the feature point detectors and descriptors are used as they can deal great with the correspondence problems. The binary descriptors are used as they use less computation cost and power when compared with the other descriptors and techniques in the field of image recognition.

In the present thesis work, a comparison work is made in between various binary descriptors like ORB, BRISK, AKAZE, FREAK and BRIEF on the basis of time taken to display the results and the amount of power consumed by these descriptors for a time frame of 5 Mins. The application needs to recognize various safety signs on the smart glasses. The selected algorithms are discussed earlier in this report to give a better idea on different possibilities. As the smart glasses come with a limited resources capacity as our smart phones, the computational cost should be low. The binary descriptor which operates in a much ease of way without interrupting any processes on the glasses is selected. ORB is much effective as it takes less time to retrieve the results from the local database and it consumes very low power when compared with the other binary descriptors. Inappropriate selection of algorithm
which requires much computational cost on the limited resources capacity results in hanging the devices, and it takes much time to resume the operation in the second cycle after recognizing the first image. The time measurements are taken into account from a period of the user capturing the safety sign and the results been displayed on the display screen.

There was also tough time in taking the power readings of different applications on the smart glasses. I had worked on various power consumption applications in order to measure the power readings in a much efficient style. The App-Tune up Kit application is been selected from a bunch of various applications to read the power measurements. As per the readings in this thesis, I can state that the ORB works more efficient than any other feature point detectors and descriptors on the smart glasses.
Appendix

Installing the Vuzix M100 Hardware:

The M100 can be used on many versions of the Windows PC. To run the installation hardware, just plug the gadget to a PC, the installation should start automatically. There might be a glitch if the ADB USB user interface is not properly installed. This means the device will only be accessible for media purposes. To manage files and check the system folders, the ADB USB interface must be properly installed. It is easy-plug and play instructions.

Installing the Android Debug Bridge (ADB) USB Interface (Windows):

The user platform of ADB is the platform that facilitates the management and development of applications for the M100 device. It is an abbreviation for Android Debug Bridge user interface. It facilitates the operation of the M100 with other Android devices. The following segment gives a breakdown of how to install the ADB and get it running smoothly.

1) Connect the M100 to the PC.
2) Check to ensure that the device is detected and ready for use on the PC.
3) If an error message is displayed indicating that the device was not recognized, open the control panel on the PC and access device manager. Locate the Vuzix M100 icon among the device manager list.
4) To continue, update the driver, by the right click on the icon.
5) Check for the driver software from the computer program files.
6) Look for an option to select the device driver on the computer from the list and choose this option.
7) Click on the “show all devices” followed by the “next” option to continue.
8) select -have disk
9) Select – browse.
10) Choose the program files for the Vuzix M100; it should appear like this on the screen-c:\Programfiles\VuzixCorporation\VuzixM100 systemfilemanager\USBdriver.
11) Select the list- android_winusb.inf followed by clicking OK.
12) Click Next to continue.
13) It will be prompted to click yes, this will install the driver.
14) If a warning is displayed indicating that the driver is not digitally signed, continue from here by clicking “install this driver software anyway”.
15) At this point, wait until the appearance of verification notice regarding the successfully installation of the ADB user interface.
A. Annex for Compact Disc (CD)

<table>
<thead>
<tr>
<th>Path in CD</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASEMMasterThesis\Mahendra\ThesisReport</td>
<td>Documentation of Thesis Work in PDF</td>
</tr>
<tr>
<td>ASEMMasterThesis\Mahendra\ThesisCode</td>
<td>Source Code for Thesis Project</td>
</tr>
<tr>
<td>ASEMMasterThesis\Mahendra\SampleVideo</td>
<td>Sample Video for Image Recognition</td>
</tr>
<tr>
<td>ASEMMasterThesis\Mahendra\Software</td>
<td>Softwares used in this Thesis</td>
</tr>
</tbody>
</table>
Bibliography


[26] M. Ramakrishna and S. Shylaja, “Is ORB Efficient Over SURF for Object


