Real-time Embedded Panoramic Imaging for Spherical Camera System

Main Uddin-Al-Hasan

This thesis is presented as part of Degree of Bachelor of Science in Electrical Engineering

Blekinge Institute of Technology

September 2013

Blekinge Institute of Technology
School of Engineering
Department of Electrical Engineering
Supervisor: Dr. Siamak Khatibi
Examiner: Dr. Sven Johansson
Abstract

Panoramas or stitched images are used in topographical mapping, panoramic 3D reconstruction, deep space exploration image processing, medical image processing, multimedia broadcasting, system automation, photography and other numerous fields. Generating real-time panoramic images in small embedded computer is of particular importance being lighter, smaller and mobile imaging system. Moreover, this type of lightweight panoramic imaging system is used for different types of industrial or home inspection.

A real-time handheld panorama imaging system is developed using embedded real-time Linux as software module and Gumstix Overo and PandaBoard ES as hardware module. The proposed algorithm takes 62.6602 milliseconds to generate a panorama frame from three images using a homography matrix. Hence, the proposed algorithm is capable of generating panorama video with 15.9590365 frames per second. However, the algorithm is capable to be much speedier with more optimal homography matrix. During the development, Ångström Linux and Ubuntu Linux are used as the operating system with Gumstix Overo and PandaBoard ES respectively. The real-time kernel patch is used to configure the non-real-time Linux distribution for real-time operation. The serial communication software tools C-Kermit, Minicom are used for terminal emulation between development computer and small embedded computer. The software framework of the system consist UVC driver, V4L/V4L2 API, OpenCV API, FFmpeg API, Gstreamer, x264, Cmake, Make software packages.

The software framework of the system also consist stitching algorithm that has been adopted from available stitching methods with necessary modification. Our proposed stitching process automatically finds out motion model of the Spherical camera system and saves the matrix in a look file. The extracted homography matrix is then read from look file and used to generate real-time panorama image. The developed system generates real-time 180° view panorama image from a spherical camera system. Beside, a test environment is also developed to experiment calibration and real-time stitching with different image parameters. It is able to take images with different resolutions as input and produce high quality real-time panorama image. The QT framework is used to develop a multifunctional standalone software that has functions for displaying real-time process algorithm performance in real-time through data visualization, camera system calibration and other stitching options. The software runs both in Linux and Windows. Moreover, the system has been also realized as a prototype to develop a chimney inspection system for a local company.

Keywords: Panorama Image, Image stitching, Image registration, SURF, Real-time computing, Gumstix Overo COM, PandaBoard ES, Embedded Linux, RTLinux, Real-time kernel, Embedded development, OpenCV, FFmpeg, V4L2, Computer Vision, QT framework
Acknowledgement

I would like to thank and express love to all scientist and philosophers from ancient age, middle age and modern age, whose works given me ground to work on this area of science and technology. Moreover, I would like to thank my teacher and supervisor Dr. Siamak Khatibi and Blekinge Institute of Technology for giving me opportunity to work with a thesis that encompasses the knowledge of software and hardware as well as theories. My motivation to image processing and embedded development sharply increased. Further thanks to my father Md. Shofi Ullah, Mother Ayesha Akter for their continuous encouragement to move forward in search of knowledge. I am dedicating my thesis to my father and mother.
CONTENTS

ABSTRACT ............................................................................................................................................. 3

ACKNOWLEDGEMENT .......................................................................................................................... 5

LIST OF FIGURES .................................................................................................................................... 15

LIST OF ACRONYMS ............................................................................................................................... 24

CHAPTER 1 ................................................................................................................................................ 26

INTRODUCTION ....................................................................................................................................... 26

1.1 Thesis scope....................................................................................................................................... 29

1.2 Thesis Outline.................................................................................................................................... 33

CHAPTER 2 ............................................................................................................................................... 35

RESEARCH AND DEVELOPMENT METHODS ....................................................................................... 35

CHAPTER 3 ............................................................................................................................................... 39

IMAGE STITCHING LITERATURE AND VISION SYSTEMS .................................................................... 39

3.1 Panoramic Image Stitching ................................................................................................................ 39

3.1.1 Photogrammetry .......................................................................................................................... 40

3.1.2 Different Image stitching algorithms ............................................................................................ 41

3.1.3 Image registration ........................................................................................................................ 42

3.2 Real-time Panorama Vision Systems ................................................................................................ 47

3.2.1 MITRE immersive spherical vision system .................................................................................. 47

3.2.2 Point Grey Spherical Vision ......................................................................................................... 48

3.2.3 Lucy S and Dot ............................................................................................................................ 49

CHAPTER 4 ............................................................................................................................................... 51

EMBEDDED RTLINUX AND SOFTWARE DEVELOPMENT TOOLS .................................................. 51

4.1 Linux Kernel ..................................................................................................................................... 51

4.2 Basics of Linux .................................................................................................................................. 52
4.3 Linux distributions........................................................................................................52
  4.3.1 Ångström distribution ............................................................................................53
  4.3.2 Ubuntu Linux distribution ....................................................................................53
4.4 Embedded Linux...........................................................................................................54
4.5 Real-time Linux ...........................................................................................................54
4.6 Bourne Shell ................................................................................................................56
4.7 Text Editors ................................................................................................................58
  4.7.1 Vi editor ................................................................................................................58
  4.7.2 Nano editor ............................................................................................................59
4.8 Native Compiler ..........................................................................................................60
4.9 OpenCV .......................................................................................................................61
4.10 Native code and Linking Process ..............................................................................62
4.11 Build Automation Tool ..............................................................................................62
  4.11.1 Build automation tool for OpenCV .......................................................................63
4.12 CMake ........................................................................................................................63
  4.12.1 Philosophy of using CMake ..................................................................................63
4.13 Make ..........................................................................................................................65
4.14 pkg-config....................................................................................................................66
  4.14.1 Reason of using pkg-config ................................................................................67
4.15 FFmpeg ...................................................................................................................... 67
4.16 V4L/V4L2 ....................................................................................................................68
4.17 GStreamer ...................................................................................................................68
4.18 x264 ............................................................................................................................69
4.19 USB Video class device (UVC) ................................................................................69
4.20 GDB (GNU Debugger) ..............................................................................................69
  4.20.1 Useful GDB Commands ......................................................................................70
CHAPTER 5

VISION, CAMERA AND IMAGING METHODS

5.1 Vision, Human Eye and Camera

5.2 Image Formation and Pinhole Camera Model

5.3 Focal Length and Camera Field of View

5.4 Camera Intrinsic and Extrinsic parameters

5.5 Philosophy of using Mathematics in Secondary vision

5.6 Projective Geometry

5.6.1 Euclidean space and Projective space

5.6.2 Projective point, line, plane and space

5.6.3 Projectivity and Perspectivity

5.6.4 Estimating transformation in Projective space

5.6.5 Projective transformation and Image Formation

5.6.6 Centre of Projection and Binocular disparity

5.7 Homogeneous Space and Coordinate

5.8 Lens distortions modeling

5.9 Rotation and Translation

5.10 Epipolar Geometry

5.11 Practical imaging practices

5.11.1 Exposure time

5.12 Imaging technique is 2D to 2D conversion

CHAPTER 6

IMAGE STITCHING

6.1 Image Features and Feature Matching

6.1.1 Image features and reason of using feature matching

6.1.2 Common features and Overlapping area

6.2 Speeded Up Robust Features (SURF) and SIFT
CHAPTER 8 .......................................................... 145
SYSTEM DEVELOPMENT ........................................... 145

8.1 Development Process ............................................... 145
8.2 Operating system ...................................................... 145
8.3 Gumstix COM .......................................................... 146
  8.3.1 Bootable MicroSD card ............................................. 146
  8.3.2 Copying image file to MicroSD card ............................... 151
  8.3.3 Serial Communication .............................................. 153
  8.3.4 Wi-Fi Communication .............................................. 154
  8.3.5 Operating system update .......................................... 155
  8.3.6 Native compiler ..................................................... 156
  8.3.7 CMake Build and Installation .................................... 157
  8.3.8 OpenCV Build and Installation .................................. 159
  8.3.9 FFmpeg, V4L2, GStreamer, X264 build and installation ....... 160
8.4 PandaBoard ES .......................................................... 162
  8.4.1 Bootable SD Card ................................................... 163
8.5 Configuring Linux kernel with Real-time patch ....................... 163
8.6 Debugging ............................................................... 164
8.7 Camera configuration .................................................. 165
8.8 Software development ................................................... 166
8.9 A standalone software with GUI ...................................... 166
  8.9.1 Qt Creator IDE, libraries and third party API configuration ...... 167
  8.9.2 GUI Design ........................................................ 168
  8.9.3 Configuring with installer .......................................... 172
8.10 A CLI based reconfiguration of OpenCV stitching module .......... 173
  8.10.1 Calibration part ................................................... 174
  8.10.2 Real-time stitching part .......................................... 175
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary and Secondary human vision [1]</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>Real-time System</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>A Panorama Image [1]</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Gumstix Overo Wireless Pack</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>Panda Board ES</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>MISUMI CCIQ Color Camera</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>Development sequence</td>
<td>37</td>
</tr>
<tr>
<td>8</td>
<td>Multi-level Knowledge Hierarchy</td>
<td>38</td>
</tr>
<tr>
<td>9</td>
<td>Hourglass model of this thesis progression</td>
<td>38</td>
</tr>
<tr>
<td>10</td>
<td>Georg Wiora's Photogrammetry data model [1]</td>
<td>40</td>
</tr>
<tr>
<td>11</td>
<td>Topographical mapping using aerial digital photogrammetry [1]</td>
<td>41</td>
</tr>
<tr>
<td>12</td>
<td>Image Stitching sub-processes</td>
<td>42</td>
</tr>
<tr>
<td>13</td>
<td>MITRE immersive spherical vision [32, 33, 34, 35]</td>
<td>48</td>
</tr>
<tr>
<td>14</td>
<td>Ladybug 2</td>
<td>48</td>
</tr>
<tr>
<td>15</td>
<td>Ladybug 3</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>Lucy S</td>
<td>50</td>
</tr>
<tr>
<td>17</td>
<td>Dot</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>Fundamental architecture of Linux [1]</td>
<td>51</td>
</tr>
<tr>
<td>19</td>
<td>Linux directory structure [1]</td>
<td>52</td>
</tr>
<tr>
<td>20</td>
<td>Linux kernel and Distributions</td>
<td>53</td>
</tr>
<tr>
<td>21</td>
<td>Fox embedded micro Linux system [1]</td>
<td>54</td>
</tr>
<tr>
<td>22</td>
<td>The UNIX shell family Tree [1]</td>
<td>57</td>
</tr>
</tbody>
</table>
Figure 47: Relation between focal length and field of view [1]................................. 81
Figure 48: Euclidean Space and Projective Space [52].............................................. 83
Figure 49: Point, Line and Plane in Projective Space [52]....................................... 84
Figure 50: Perspectivity and Projectivity................................................................... 85
Figure 51: Perspectivity impact in imaging [55]....................................................... 86
Figure 52: One-point and two point perspective [Wikipedia]..................................... 87
Figure 53: Pappus theorem, Desargues theorem, quadrangle and quadrilaterals [1]..... 88
Figure 54: 3D to 2D projection using homogeneous coordinates [41,53].................... 89
Figure 55: Centre of projection and Human vision [1]............................................ 92
Figure 56: Homogeneous space and coordinate system [61, 1]............................... 93
Figure 57: Radial distortion of a simple lens [41]....................................................... 94
Figure 58: Tangential distortion of a cheap camera [41]......................................... 95
Figure 59: Rotation and Translation [1]..................................................................... 96
Figure 60: Geometric properties of a rotation [41].................................................. 97
Figure 61: Effect of different centre of projection [1]............................................. 98
Figure 62: Epipolar geometry [1].............................................................................. 99
Figure 63: Epipole and epipolar lines [62, 1]............................................................ 100
Figure 64: Exposure time and Luminosity [1]........................................................... 101
Figure 65: Feature matching using SURF [1]......................................................... 104
Figure 66: Common Features between two images............................................... 105
Figure 67: Two corresponding patch....................................................................... 105
Figure 68: Canny Edge......................................................................................... 106
Figure 69: Vanishing point [1].............................................................................. 106
Figure 70: Region segmentation [1]....................................................................... 107
Figure 71: Image contour [1].................................................................................................................. 107
Figure 72: Overlapping between rectangles [1]...................................................................................... 108
Figure 73: A SIFT key point descriptor computed from Gradient Magnitude and Orientation and SURF sums [7, 63].................................................................................................................. 109
Figure 74: Projection of points on planar surface [64]........................................................................... 111
Figure 75: Basic 2D planar transformations [27].................................................................................... 113
Figure 76: Transformation matrix and invariants of planar transformations [53]................................. 113
Figure 77: Homography in Planar projection [64]................................................................................... 114
Figure 78: Pixel mapping between two image planes [1]........................................................................ 115
Figure 79: Homography matrix for Frontal plane............................................................................... 116
Figure 80: Homography Formation [64].............................................................................................. 117
Figure 81: RANSAC algorithm for model fitting.................................................................................. 119
Figure 82: Feature matching and then Homography computation using RANSAC ......................... 120
Figure 83: Homography matrix anatomy............................................................................................ 124
Figure 84: Projective transformation or Collineation or Homography [53]............................................. 126
Figure 85: Warping a sensed image into reference image [68].............................................................. 127
Figure 86: Conceptual layers in Panorama Imaging System................................................................. 134
Figure 87: White box model of the System's basic structure.............................................................. 135
Figure 88: Black box model of the system's basic structure.............................................................. 135
Figure 89: System components........................................................................................................... 136
Figure 90: Functional principle blocks ............................................................................................... 137
Figure 91: System component functionality....................................................................................... 138
Figure 92: Gumstix Computer-On-Module ....................................................................................... 139
Figure 93: Overo® Fire ......................................................................................................................... 140
Figure 94: Palo35 Expansion Board ..................................................................................................... 140
Figure 95: PandaBoard ES
Figure 96: Bootable MicroSD card
Figure 97: MISUMI Spherical Camera System
Figure 98: Usb Hub 2.0
Figure 99: Extracted root file system inside partition ext3
Figure 100: Terminal emulation between target Gumstix board and Development machine
Figure 101: DNS ip address
Figure 102: File systems inside Gumstix
Figure 103: Real-time PREEMPT kernel in PandaBoard ES
Figure 104: Building OpenCV debug version
Figure 105: A debug breakpoint
Figure 106: GUI QWidget design blocks
Figure 107: Real-Time data display via GUI in Windows
Figure 108: Real-Time data display via GUI in Linux
Figure 109: UML Model Diagram of some used classes
Figure 110: Menu options, QActions, Signals and Slots
Figure 111: Dependency Walker for finding .dll
Figure 112: Calibration and real-time stitching in Linux
Figure 113: Calibration in Linux using calib command
Figure 114: Real-time stitching in Linux using stitch command
Figure 115: Automated test environment
Figure 116: Semi-automated test environment
Figure 117: Semi-automated test environment inside Linux
Figure 118: Proposed Algorithm Structure
Figure 119: Motion modelling and Real-Time stitching ............................................. 181
Figure 120: System calibration via panoramic mosaicking ........................................ 182
Figure 121: Spherical camera system's FOV ............................................................ 183
Figure 122: Overlapping area and corresponding feature ........................................ 183
Figure 123: Homography Matrix Look File .......................................................... 184
Figure 124: Focal Length Look File .................................................................... 185
Figure 125: Motion modelling sequence diagram .................................................. 185
Figure 126: Sequential Warping or Projective mapping in a plane surface ............... 186
Figure 127: Three input images from three cameras .............................................. 186
Figure 128: Warping first image into plane compositing surface ............................ 187
Figure 129: Sequentially warping second image into plane compositing surface ...... 187
Figure 130: Sequentially warping third image into plane compositing surface ......... 187
Figure 131: Real-time stitching sequence diagram ............................................... 188
Figure 132: Our proposed algorithm's performance .............................................. 190
Figure 133: Our proposed algorithm's performance .............................................. 190
Figure 134: Generated Panorama in 62.6602 milliseconds with optimal homography matrix (With Warping and Compositing) .......................................................... 191
Figure 135: Used Gumstix COM (Overo Fire, Expansion Board, LCD and MicroSD card).... 193
Figure 136: PandaBoard ES with external hardwares ......................................... 197
Figure 137: Motion modeling inside Windows with calib command ....................... 199
Figure 138: Real-time stitching using stitch command in Windows ....................... 200
Figure 139: Two instances of real-time stitching in Windows with optimal algorithm .... 200
Figure 140: Motion modelling using calib command in Linux .............................. 201
Figure 141: Real-time stitching using stitch command in Linux ............................ 202
Figure 142: Two instances of real-time stitching in Linux with optimal algorithm ....... 202
Figure 167: Fastest response time comparison ................................................................. 219
Figure 168: Slowest response time comparison .............................................................. 220
Figure 169: Average Response time comparison ............................................................ 220
Figure 170: Calculation of Jitter ..................................................................................... 221
Figure 171: Jitter period ................................................................................................. 222
Figure 172: Obtained Jitter in different platforms .......................................................... 223
Figure 173: CPU Speed vs. Average Response Time ....................................................... 224
Figure 174: CPU Speed vs. Average Response time ....................................................... 224
Figure 175: CPU vs. Average Response time .................................................................. 225
Figure 176: Input images with variant luminosity .......................................................... 225
Figure 177: Resulted Panorama using Invariant features ................................................ 226
Figure 178: Chimney testbed ......................................................................................... 227
Figure 179: The Chimney door and top opening ............................................................ 228
Figure 180: Targeted code ............................................................................................. 228
Figure 181: Used Micro Spherical MISUMI Camera inside Chimney ........................... 229
Figure 182: Code Inspection (Without seam removal and Blending) ............................ 229
Figure 183: Code inspection (Without seam removal and Blending) ............................ 230
Figure 184: Sequential thesis steps ............................................................................... 232
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application-specific Integrated Circuit</td>
</tr>
<tr>
<td>ARM</td>
<td>Advanced RISC Machines</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BAT</td>
<td>Build Automation Tool</td>
</tr>
<tr>
<td>CP</td>
<td>Control Points</td>
</tr>
<tr>
<td>COM</td>
<td>Computer on Module</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name Server</td>
</tr>
<tr>
<td>FOV</td>
<td>Field of View</td>
</tr>
<tr>
<td>GDB</td>
<td>GNU Debugger</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>GCC</td>
<td>GNU Compiler Collection</td>
</tr>
<tr>
<td>HBC</td>
<td>Human Brain Constraint</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>I/O</td>
<td>Input Output</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>MHz</td>
<td>Megahertz</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte</td>
</tr>
<tr>
<td>OMAP</td>
<td>Open Multimedia Application Platform</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Source Computer Vision</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>PPP</td>
<td>Point-to-Point</td>
</tr>
<tr>
<td>RANSAC</td>
<td>Random Sample Consensus</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real Time Operating System</td>
</tr>
<tr>
<td>RTLinux</td>
<td>Real-time Linux</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computing</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RT</td>
<td>Rotation Translation</td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale Invariant Feature Transform</td>
</tr>
<tr>
<td>SURF</td>
<td>Speeded Up Robust Features</td>
</tr>
<tr>
<td>SMP</td>
<td>Symmetric Multiprocessing</td>
</tr>
<tr>
<td>SVD</td>
<td>Singular Value Decomposition</td>
</tr>
<tr>
<td>SSD</td>
<td>Sample Standard Deviation</td>
</tr>
<tr>
<td>SD</td>
<td>Secure Digital</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>UVC</td>
<td>USB Video Class</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>V4L</td>
<td>Video for Linux</td>
</tr>
<tr>
<td>V4L2</td>
<td>Video for Linux 2</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>2D</td>
<td>Two Dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>Three Dimensional</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

We humans of green planet earth communicate with each other by the means of different protocol. Speaking and listening are the most dominating way of human communication. However, the core ability that empowers all types of human communication is the ability to see something or Vision. It is no wonder that secondary vision (i.e. imaging via camera) ability will enable human to extremely multifold its information processing capability and to see unimaginably large distant places as a secondary viewer. And exactly that is happened in this modern age. Further development of secondary information processing of humans will invent new technologies that are unthinkable now. In this thesis, a real-time panoramic imaging system is developed using a Spherical camera system, Real-time Embedded Linux, several software libraries (e.g. OpenCV, FFmpeg, V4L/V4L2, x264, QT), real-time stitching algorithm and Gumstix COM/PandaBoard ES embedded computer.

![Figure 1: Primary and Secondary human vision](image)

The figure 1 shows the similarity between native human vision and imaging. The emergence of imaging has fathered numerous technologies. The production of innovative systems has been increased by many folds after invention of digital imaging. Real-time panoramic image processing is one of the most important breakthroughs in the field of digital
imaging technologies. It ensures the system’s scalability and reliability which is a fundamental requirement for many applications. These types of system can be used in biomedical engineering, multimedia, military hardware, security, inspection, system automation, tourism and in other commercial handheld appliances.

The term “embedded” has particular meaning. A system is embedded system when it belongs to larger system paradigm and designated to solve a particular task. In most cases, these embedded systems are designated for real-time operation. However, a handheld spherical camera system with a mobile processing unit can be termed as an embedded imaging system when it is used as an internal component of another larger system paradigm. Otherwise, a spherical camera system only can be termed as a real-time or non-real-time imaging system. The question of being embedded or not depends on the intended use of an independent imaging system.

The quality of being real-time [2, 3] predicts about the designed imaging systems performance and its throughput. In real-time systems, the time between input and output is called the response time. A system can be considered as real-time if and only if system’s response time satisfies certain time constraints. However, real-time system can be categorized into two categories- hard real-time system, soft real-time system. The failure of hard real-time systems to meet the certain time constraints has catastrophic affects (e.g. death of human because of plane crash, destructive clash between automated machine parts and so on). On the contrary, failure of soft real-time systems to meet time constraints has utmost loss of performance. The characteristics of real-time system are depicted in the figure 2.

![Real-time System Diagram](image)

Figure 2: Real-time System

The importance of real-time systems is not in its processing speed. Rather, the predictability and scalability made the real-time systems very important. Modern intelligent systems have numerous components that are integrated into a single module. These components
have inter-dependency which means one component’s operation is dependent on another component’s result. Hence, the automated synchronized processing of all parts is only possible when every component’s response time and possible jitter is known beforehand.

There is a difference between real-time and non-real-time panoramic image. The generation of panoramic image is a computation intensive task. The panorama generation time largely depends on stitching algorithm, software dependency processes and hardware solutions. In a real-time panoramic imaging system, the generation of real-time panorama image must be done within certain time constraint.

The figure 3 shows a panorama image. There are multiple technologies to capture panorama image. A panorama image can be captured by long film stripe (Meehan1990), mirrored pyramid or parabolic mirrors (Nayar1997), lenses with large FOV like fisheye lenses (Xiong and Turkowski1997). But these hardware solutions are not economic and most-often obstacle for generating panorama with ease. However, to lessen these barriers, a substantial number of software solution based on “image stitching” has been developed. One of the earliest “image stitching” software is Apple’s QuickTime VR. Later, more advanced “image stitching” software like AutoStich, Hugin has been developed to generate panorama.

The process of joining separate images in concatenation to generate a large image or panorama is known as “image stitching”. In this process, a large set of images is captured while keeping overlapping region with each neighbor image. The overlapping regions are used to get the geometric relation among images and then images are composited into a large image with higher resolution.

The image stitching algorithm response time is one of the most important factors to design a real-time panorama imaging system. The available stitching algorithms offer different response time and quality. In some deployment cases, faster response time is more important than the overall visual quality of the panorama image. In other cases, overall panorama image quality is more important than the faster response time. However, there are deployment cases where faster response time and overall visual quality of the image both are important. The
stitching algorithms are designed or configured as a sequel of intended use. The stitching
algorithms for optimal image quality and faster response time are usually deployed in the
systems that have enough processing capability. However, there are systems that have less
computation resources (e.g. handheld mobile devices like smartphone, systems that uses smaller
handheld computer like Gumstix). For these systems, stitching algorithms are designed for faster
response time by compensating overall visual quality of the panorama image slightly or
substantially.

Moreover, the panorama generation time not only depends on “stitching algorithm” but
also hardware and software dependency solutions. Earlier panorama imaging systems had long
response time because of less data computation capability and underdeveloped software
dependencies. Recent panorama imaging systems has lessened response time by many folds due
to advancement in the CPU technology, faster algorithm and smarter software dependencies.

1.1 Thesis scope

In principle, the thesis is about to realize a spherical imaging system using embedded handheld
computer for real-time panoramic image generation. Our proposed panoramic imaging system
has the following characteristics.

- Real-time panorama
- Mobile
- Small handheld embedded computer
- Hand free camera
- Optimized panoramic view
- Graphical User Interface (GUI)
- Real-Time algorithm process statistics
- Data visualization

In another project, the developed system will be used to transmit panorama video to a
destination device (e.g. IPhone 4, Galaxy SII, III) in real-time. The system consist three cameras
that are positioned adjacent to each other. The angle between the cameras approximately 60° and
jointly covers approximately 180° wide view.
The main functional parts of a real-time imaging system are *computation intensive hardware, real-time operating system, real-time software dependencies* and a *real-time process algorithm*. The main problems or tasks were designated to solve in this thesis are –

1. Study of the existing smaller handheld computer platforms, comparison and selection of the appropriate development board

2. Finding out the most suitable real-time operating system or preparing a general purpose operating system for real-time operation

3. Installing the operating system of the development board and preparing the operating system for real-time operation and corresponding troubleshooting

4. Study of the mathematical foundation of image stitching technology

5. Finding out the appropriate image stitching algorithm

6. Finding out motion estimation (motion calibration) of the spherical camera system and implementation of real-time stitching

7. Developing the test environment of the motion estimation and real-time stitching

8. Preparation of the development board with necessary software dependencies and corresponding development troubleshooting

9. Direct (without storage) pointed processing of captured image data from the Spherical camera System

10. Panorama Video sequences from the spherical camera system

11. Reading a particular code from a Chimney (Test bed) via panorama image.

12. Developing an user interface using Qt to for displaying real time data statistics, algorithm process response time and data visualization.

13. Developing portable software with simple installer.

In this thesis, we have found the best possible existing hardware solution for a real-time hand-held imaging system. Besides, a suitable operating system is also tested, selected and prepared for real-time operation. Moreover, available panorama stitching algorithms are also studied to find out the best algorithm for the system. Therefore, the algorithm flow of [4, 5, 6, 7] are further modified to meet the system’s need. Furthermore, all other tasks also been solved in this thesis.
The figure 4 and 5 shows Gumstix Overo Wireless Pack and PandaBoard ES respectively. Gumstix COM, Pandaboard ES and Linux are used as the hardware and operating system respectively. The system is very light and highly flexible for mobile operations. Besides, the camera system is hand free and can be positioned according to the user’s demand. A new model of MISUMI camera is used in the project. It is possible to put the camera inside narrow areas because of its smaller size and one meter long cable support. The figure 6 shows one of the camera that were used.
The camera capsule captures three images from the targeted area and sends it to the Gumstix COM or PandaBoard for panoramic image generation. The generated panoramic image is further processed inside mobile computer to facilitate faster image transmission. The processed panoramic image is placed in a buffer and transmitted to the destination device. However, the following chart gives an overview of the thesis scope.

<table>
<thead>
<tr>
<th>Component</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tested Handheld computer</td>
<td>PandaBoard ES and Gumstix COM</td>
</tr>
<tr>
<td>Tested Laptop computer</td>
<td>HP, Toshiba, Acer</td>
</tr>
<tr>
<td>Spherical Camera System</td>
<td>MISUMI micro spherical camera system, USB-2.0 spherical camera system, Creative camera</td>
</tr>
<tr>
<td>Operating system and patches</td>
<td>Windows, Ubuntu Desktop Linux, Ubuntu Linux (OMAP4), Ångstrom minimal Linux (OMAP3), RTLinux, Real-time OS patch</td>
</tr>
<tr>
<td>Peripherals</td>
<td>Keyboard, mouse, virtual keyboard, power connector, serial communication cable</td>
</tr>
<tr>
<td>Used bootable storage</td>
<td>MicroSD card (Gumstix), SD card (PandaBoard ES)</td>
</tr>
<tr>
<td>API and software dependencies</td>
<td>OpenCV, FFMPEG, GStreamer, UVC, V4L2,</td>
</tr>
<tr>
<td>Software tool</td>
<td>QT, pkg-config, x264</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Software tool</td>
<td>GNU GCC Compiler, CMake, C-Kermit, Minicom, Visual Studio, Qt Creator, guvcview, IBM Rational Rhapsody, Nano editor, Vi editor</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Image registration, Image stitching, RANSAC</td>
</tr>
<tr>
<td>Theories and Mathematics</td>
<td>Vision and Electromagnetism, Pinhole camera model, Euclidean geometry, Projective geometry, Homogeneous coordinates, Rotation and translation, Transformation, Direct Linear Transformation(DLT), Epipolar geometry, Feature matching, SIFT, SURF, SVD</td>
</tr>
<tr>
<td>Software development</td>
<td>CLI based reconfiguration of OpenCV stitching module, A standalone software for both Linux and Windows (Qt, C++)</td>
</tr>
</tbody>
</table>
| Test environments | **Windows:** Visual Studio 2010 with Visual studio command prompt, Developed standalone software  
**Linux:** Ubuntu Linux home directory tree with command terminal, Developed standalone software |

1.2 Thesis Outline

Chapter 2 describes about the development and research approach that have been followed during the thesis. Chapter 3 briefly describes about existing real-time panorama imaging systems and as well as existing image stitching algorithms. Chapter 4 demonstrates a fundamental understanding about Embedded Linux, software dependencies and other necessary software development tools. Chapter 5 describes human vision and electromagnetism, image formation and related mathematical concepts. Chapter 6 describes some fundamentals of image stitching method. Chapter 7 gives a system overview as well as system hardware and software. Chapter 8 describes the summary of the development activities that has been done during the thesis project. Chapter 9 discusses about proposed and implemented stitching algorithm and its architecture.
Chapter 10 describes practical Hardware implementation. Chapter 11 demonstrates some results from the developed real-time panoramic imaging system and other outcomes of the project. Chapter 12 shows an experimental testing of the system to solve a real-life problem. Chapter 13 describes the thesis summary and potential future work.

Moreover, the whole paper is written with a structure that it gives a summarized detail of embedded development project as well as fundamentals of imaging. The embedded development projects vary in terms of scale or area of application. But all types of embedded development project share some commonalities in terms of design, software and hardware.
Chapter 2

Research and Development Methods

The term “research” can be categorized into several levels according to its purpose, depth or method. Research can be done to solve a problem, explore an idea or to probe an issue. However, all type of research has two things in common, that is to solve some problem or finding some new facts with the help of knowledge exploration. The problems were designated to be solved in this thesis required knowledge about mathematics, algorithm, hardware and many software tools. The project was started with a goal of generating real-time panorama images from handheld computer and spherical camera system but how to reach that goal was not defined. Hence, it is worthy to mention that this project was more development oriented than research. It can be said because the project requires many building blocks in terms of theory, software and hardware. Subsequently, a vigorous research is difficult to be carried out individually in many areas at the same time while trying to develop a running system prototype. The following questions arose in principle and solved during the progression of the thesis project.

- What are the characteristics of real-time operation and how to create a panorama image?
- What components are necessary to realize a handheld camera system that generates panorama?
- Where to find out the external components and how to purchase them?
- Which embedded development boards are currently available in the market and which one is most suitable for the project?
- How to connect a small embedded computer with external hardware’s (e.g. keyboard, mouse, USB hub, and monitor) and find out connection cables to have a full functional development platform?
- What type of operating system can be used for real-time operation, how to install operating system in the small development board and how to solve corresponding errors?
- How to use Linux in general and how to use embedded RTLinux in small embedded computer?
- What types of software program knowledge are necessary to become a competent user in embedded Linux environment?
• Which software packages and API’s are required for this area of development and consistent with the project goal?
• Why some software packages are necessary and how these software programs facilitate to achieve the project goal?
• Where to find, how to install and compile these software packages and how to solve different development errors?
• What is image stitching algorithm and what amount of research work already been done in this field?
• Which mathematical branches is somehow related Image stitching algorithm?
• What is the architecture of image stitching algorithm and which mathematical theorems and methods lays the foundation of image stitching technology?
• What type of image stitching algorithm and domain knowledge best suites the necessity of this thesis project?
• How knowledge about mathematical theory, algorithm, hardware and software solution need to be put together to have a desired output of the thesis project?
• How to formulate the visual rendering of the system software and hardware architecture?
• How to develop a user friendly GUI for rendering performance of real-time processes of the system software?
However, in terms of development progression, development sequence of this project can be described as figure 7.

Figure 7: Development sequence
Methods are derived from a theory. The developed methods are then used in application domain. As methods follow the theory, theoretical exploration of small embedded computers, stitching algorithm, embedded RTLinux, embedded software development tools were at core of the research activities. The figure 8 shows a hierarchical relation between principle knowledge levels.

Figure 8: Multi-level Knowledge Hierarchy

Moreover, developing an embedded system requires knowledge from all levels of the above hierarchy. Having knowledge about only top level application domain technologies (e.g. different hardware, software) do not substantially help achieve the goal of an embedded development project. Subsequently, the research or exploration in this thesis covered many theories and methods that does not directly related to desired result of this thesis project. But this surfed information helped indirectly to figure out root cause of an error and then troubleshooting them. Another characteristic of this thesis project is that it was carried out independently under the supervision of Dr. Siamak Khatibi without any slightest external help. However, the progression to the thesis goal has a similarity with the Hourglass model of research. The progression flow is depicted in figure 9.

Figure 9: Hourglass model of this thesis progression
Chapter 3

Image Stitching Literature and Vision Systems

This chapter discusses about some available real-time panorama imaging systems developed by different companies and research organizations. The most of the vision systems are developed with particular configuration that is determined according to its intended industrial (e.g. industrial automation, robotics, remote sensing) use. These vision systems are proprietary, consequently inaccessible and thus discussion about these vision systems is not possible unless otherwise permitted. Moreover, this chapter also discusses about old and latest stitching algorithms.

3.1 Panoramic Image Stitching

Image stitching originated from the field of photogrammetry. Earlier image stitching manual intensive methods were based on surveyed ground control points or manually registered tie points. Later, the development of bundle adjustment was a major breakthrough for a globally consistent automatic image-mosaic positioning solution (Triggs, McLauchlan, Hartley et al. 1999). Moreover, a good number of seam removal techniques have been developed over the course of years. (Milgram 1975, 1977; Peleg 1981; Davis 1998; Agarwala, Dontcheva, Agarwala et al. 2004). During 1990s special cameras were developed to take the panorama directly by the camera solution (Meehan 1990). But during mid-1990s, seamless panoramas started to being constructed from the images of regular hand-held cameras by using automatic image alignment techniques (Mann and Picard 1994; Chen 1995; Szeliski 1996). During the stitching process, “ghost” like objects become visible because of parallax and object movement. The methods of removing “ghosts” also been realized. (Davis 1998; Shum and Szeliski 2000; Uyttendaele, Eden and Szeliski 2001; Agarwala, Dontcheva, Agarwala et al. 2004). Furthermore, the issues of globally consistent alignment (Szeliski and Shum 1997; Sawhney and Kumar 1999; Shum and Szeliski 2000) and varying exposures (Mann and Picard 1994; Uyttendaele, Eden and Szeliski 2001; Levin, Zomet, Peleg et al. 2004; Agarwala, Dontcheva, Agarwala et al. 2004; Eden, Uyttendaele and Szeliski 2006; Kopf, Uyttendaele, Deussen et al. 2007). Consequently, different stitching solutions were originated (Chen 1995; Sawhney, Kumar, Gendel et al. 1998) from the above techniques.
However, direct minimization of pixel-to-pixel dissimilarities were at the core of the old image stitching techniques, while recent algorithmic techniques relies on extraction of a sparse set of features (e.g. SIFT, SURF) from the images and then matches those features during stitching process.

3.1.1 Photogrammetry
Photogrammetry provides geometric information of objects from the photographic images. Photogrammetry now can be termed as Digital Photogrammetry [8] as its full potential of information processing started to be realized in 20th century by the process of digitalizing it. However, most its pre-developed use in 19th century was carried out by manual processes.

It provides information about a photograph both from interior orientation and exterior orientation. In inner orientation, it provides necessary intrinsic camera parameter such as focal length, optical geometric distortion. In exterior orientation, it provides necessary extrinsic camera parameter such as rotation matrix and translation vector which are necessary to determine transformation between known world reference frame and unknown camera reference frame. The figure 10 and 11 shows the Georg Wiora’s photogrammetry data model and one application of aerial digital photogrammetry respectively.

![Georg Wiora's Photogrammetry data model](image)

Figure 10: Georg Wiora's Photogrammetry data model [1]
3.1.2 Different Image stitching algorithms

A considerable number of algorithms have been developed to enhance “image stitching” by the researchers around the world. All these algorithms share some common properties while having some new innovative methods. However, available algorithms can be classified as response time optimization oriented and image quality optimization oriented according to their research direction and content.

The research papers [9, 10, 11, 12] focuses on improving overall stitching quality that ranges from algorithmic process to visual quality optimization. In addition, the research papers [13, 14, 15, 16, 17, 9] focuses on the shorter response time and fast panorama throughput. Recent advancement in the CPU technology with incredible processing power attracted substantial number of researchers and company to develop state-of-the-art real-time panorama imaging system. Moreover, the papers [18, 19, 20, 21, 22] focuses on different aspects of SIFT algorithm and its use in stitching process and so on. Furthermore, the papers [23, 24, 25, 26] discusses about image stitching process in general. These papers give a clear idea how image stitching method works. Additionally, the book [27] describes the stitching process with a great detail including other sister algorithms that has been intensely used in the field of Computer Vision.

The process of image stitching can be categorized into some principle sub-processes which is depicted in figure 12. Numerous literatures have been written on each of this sub-process. The sub-processes are-

1. Image Registration
2. Calibration
3. Blending

![Image Stitching sub-processes](image.png)

Figure 12: Image Stitching sub-processes

### 3.1.3 Image registration

The paper [28] robustly refers to 224 references regarding image registration methods, which is thoroughly studied in this thesis. In this paper, an intensive review has been done on the modern and legacy image registration methods. Though the paper does not highly elaborate the discussed methods, but the narration of the methods gradually builds up an intuition about the image registration methods in the mind of readers. The paper discusses about Feature detection, Feature matching, Transform model estimation, Image resampling and transformation and also focuses on the evaluation of the image registration accuracy, current trends and outlook for the future.

In feature detection section, it categorizes feature detection methods into area based methods and feature based methods. Area based methods embeds the feature detection with feature matching step. On the other hand, feature based feature detection methods is based on the extraction of salient structures and features from the mosaics. The features are categorized into region features, line features and point features.

In area based feature matching section, the paper discussed about correlation-like, Fourier based and Mutual information (MI) based feature matching methods. The paper discusses about core functional foundation of area based methods, advantages, disadvantages, several multiple distinct approaches and references of the methods. It compares different distinct approaches in
terms of performance and specialized needs. For example, it discusses about how correlation ratio based methods can process intensity differences between multimodal (multi-sensor) images in compare to classical cross-correlation methods under correlation-like methods. Moreover, it also discusses about how Fourier Shift Theorem based phase correlation is used to compute cross-power spectrum of sensed image and reference image and hence, identifying maxima or peak which represents matching position under Fourier based methods. Furthermore, it also discusses about how Mutual information (MI) method is used to find out correct matching point between new and old mosaic when human operator selected matching point is substantially erroneous.

In feature based feature matching section, the paper discusses about spatial relations based methods, invariant descriptor based methods, relaxation methods, pyramid and wavelets. Spatial relation based methods are used when detected features are ambiguous or if their neighborhood has local distortion. The basic working principle of graph matching algorithm, clustering technique and chamfer matching is discussed. Invariant descriptor based methods are used when detected features need to be matched using descriptors that are invariant to the probable image degradation. The descriptor should met all conditions like invariance, uniqueness, stability, independence or an appropriate trade-off among the conditions can be done. There are some intuitive methods discussed in this paper which does not fully satisfy all criteria of invariant descriptors such as assigning spatial description of neighbor features to every point feature, describing each feature point by angles between relevant intersecting lines, computing affine geometric invariants on border triangles generated by object convex hull, radial shape vector, shape matrices etc. In this paper, a substantial number of invariant descriptor of the features is discussed. Namely- chain code representation of contours, non-collinear triplets of CP’s by the sphericity, polygon based shape descriptor, moment-based invariants, affine transform invariants, complex moment, chain code representation of contours, slopes of tangents in the contour points, histogram of line-length ratios of any two line segments, histogram of angle differences of any two line segments, multi-value logical tree (MVLT) using various descriptors like ellipticity, angle, thinness, applying cross-correlation on moment based circular shaped windows etc. The relaxation method of feature matching is implemented by readjusting merits of the pair figures, extension of classical relaxation method by adding corner features such as corner sharpness, contrast and slope and so on. The pyramidal approach of feature matching is intended to reduce the computational cost. The approaches like appropriate choice of sub-window size to find corresponding window in the reference image, selecting sparse regular grid of windows for cross correlation matching, combined cross correlation with pyramidal image hierarchy, cubic spline based pyramid with mean square intensity difference minimization and mutual information (MI) maximization, combining Laplacian and Gaussian pyramid with cross correlation and sum of squared differences similarity measures, wavelet decomposition of the images are discussed in this paper. The coarse-to-fine hierarchical strategy is used in pyramidal methods. The orthogonal, biorthogonal, spline biorthogonal, haar, daubechies and gabor wavelet
based feature matching are discussed in this paper. The paper also discusses about the scope of applicability of both area based feature matching and feature based feature matching. The area based feature matching is a good choice when images have better distinctive information by graylevels or colors than local structure and shapes. The intensity functions of the reference and sensed image need to be either identical or statistically dependent in order to be able to use area based feature matching. Moreover, the area based feature matching can be used when there is only shift and small rotation between the images. The feature based feature matching is a good choice when information offered by local structure and shapes are more robust than offered by the image intensities. This method offers better registration results with complex between-image distortions and images of different natures. However, the method performs poorly when the local image structure and shapes are not salient, undetectable and mobile in time. Furthermore, the feature descriptors are needed to be robust and discriminative enough as well as invariant to possible differences between the images.

In the transform model estimation section the paper discusses about the selection of mapping functions and its parameter. The factors that need to be considered while choosing a mapping function are assumed geometric deformation of sensed image, method of image acquisition and required accuracy of the registration. The models of mapping function can be divided into two categories in terms of scope of their validity on image data. The global model is valid for the whole image and all CPs are used for estimating one set of mapping function parameter. On the other hand, image is considered as a combination of patches for local mapping model and defines parameters of the mapping function for each patch of the image. Moreover, in terms of accuracy, mapping functions can be categorized as interpolating functions and approximating functions. A balanced trade-off can be made between accuracy and other requirement of the mapping function through approximating functions, whereas interpolating functions exactly map the CPs of the sensed image on the reference image. It discusses about conditions for using “shape-preserving mapping” and perspective projection model in the global mapping model. The lower order polynomials are used in global mapping model and higher order polynomials are usually avoided in practical applications as they may inaccurately warp the sensed image when aligning with the reference image. As global mapping cannot properly handle local image distortion local mapping models are preferable. However, a group of global mapping methods named radial basis functions can handle local image distortions is being an exception. The references to the radial function such as multi-quadrics, undeland’s function and thin-plate splines are mentioned in this paper. The trade-offs of different thin-plate spline approaches are discussed in this paper. The comparison of the performance of thin-plate spline and other mapping functions is also being referred. Moreover, another type of spline family based method elastic body spline (EBS) is also discussed. The EBS showed better performance in compare to thin-plate spline (TPS) to register 3D MRI image of breast. The approach of not using any parametric mapping functions for considerably complex and locally distorted images is often termed as elastic registration. It is a clever way to estimate complex deformation and
establish feature correspondence in which images are viewed as pieces of stretched rubber sheet on which external forces are applied. The external force can be derived by the correspondence of boundary structures. Furthermore, in the conditions when image deformations are much localized fluid registration can be used. It uses viscous fluid model and model reference image as a thick fluid which flows out to match with the sensed image. The non-rigid diffusion-based registration, level sets registration, optical flow based registration is also mentioned in this paper on which diffusion-based registration and level sets registration are being referred.

In image resampling and transformation section, the forward and backward manner of the sensed image transformation is discussed. In forward manner transformation, holes or overlaps can appear in output image because of discretization and rounding. To avoid this problem, backward manner transformation is used which successfully overcomes the problem. The paper discusses about most commonly used interpolants such as nearest neighbor function, bilinear functions, bicubic functions and refers about quadratic splines, cubic B-splines, higher-order B-splines, Catmull-Rom cardinal splines, Gaussians and truncated sinc functions. It also refers to articles that compares different interpolation methods, discusses about interpolation issues and introduces new interpolation techniques.

The image registration accuracy section discusses about the challenges to exactly measure the registration error as error may occur at every stages of registration and in the image contents it is difficult to distinguish between registration inaccuracies and real physical differences. However, the basic error patterns are evolved based on empirical data which categorizes errors in classes such as localization error, matching error and alignment error. The localization error is caused by inaccurate detection of CP and causes displacement of CP coordinates. The more CP candidates are selected, the more increases the possibility of greater localization error. However, there are cases when more CP candidates are preferable even if it increases the error rate. The matching error occurs when some selected CP feature pair fails to match during establishing correspondence between the CPs. The error rate is calculated by the number of false matches. To properly detect false matching two methods are used for each feature pair of CP candidates. If the pair is matched by both methods then the pair is considered as valid, otherwise excluded. However, if there is shortage of accurate matching method then false CP pairs are rechecked by cross-validation in which false CP pairs are kept if their displacement falls under a certain threshold. The alignment error means the difference between between-image geometrical distortion and the mapping model used for registration process. The alignment error occurs if the mapping model does not correspond to actual between-image distortion and parameters of the model were calculated imprecisely. The latter is caused by fewer CPs or localization errors, whereas the former is caused by the lack of prior information about the between-image distortion. One of the ways to measure alignment error is the mean square error at the CPs. The CPs that are intentionally excluded from the mapping function design is called test point. There are cases when test points are need to be used and probable error is called
test point error. Moreover, alignment error can be estimated using consistency check in which an image is registered using two comparative methods and then results are compared. The “gold standard method” is preferable to use as comparable method in the application areas where it is available (i.e. medical imaging). However, in the application areas (i.e. computer vision, industrial inspection and remote sensing) where gold standard method is not available, any type of method that has different nature can be used as comparative method. Furthermore, the registration accuracy estimation methods can be complemented by the visual inspection of an expert image analyst. The estimation of accuracy of the registration algorithm is a principle criterion before practical implementation.

The paper discusses about the importance of image registration in image fusion, change detection, super-resolution imaging and image information system development in the current trends and outlook for the future section. Though substantial amount of work has been done on this field but it still challenging to register N-D (Where N>2) images automatically. The computational complexities of N-D images are ever increasing with continuously growing data size. Even though faster computers are emerging, it is still demandable to have registration methods with faster response time. Though the pyramidal image representation is used combinedly with fast optimization algorithm for faster computing but it performs poorly with images that have significant rotation or scaling differences. Sometime, the mixture of distinctive methods is used to achieve the domain requirement. The mutual information (MI) and feature based methods are used combinedly to solve the higher robustness and reliability problem of multimodal registration in medical imaging. The paper emphasizes the necessity of developing new invariant and modality-insensitive features to have a better image registration process. The paper concludes by expressing the urge for an ambitious super autonomous registration process that will become the foundation of next generation expert systems.

The paper [4] discusses about fully automatic image stitching using invariant features in a great detail. It is used as the source of used stitching method. Image stitching is generally done via fixed image ordering. In this paper, stitching is proposed to be done without fixed image ordering. The proposed algorithm can automatically discover the matching relationships among unordered images with varying orientation and zoom.

The paper [5] proposes a method to construct panorama with local and global alignment. In the introduction, it discusses about the disadvantages of the previous hardware and software solutions such as use of hardware intensive long film strip method and cylindrical or spherical coordinates to construct panorama. The paper then proposes a new method of constructing panorama using rotation matrix and focal length and names the method as rotational mosaics. It also offers flexibility to project the resulted panorama on a convenient viewing surface such as texture-mapped polyhedron surrounding the origin. Moreover, it also mentions about a gap closing technique to reduce misregistration error. For optimal overall registration through global optimization, it also mentions about a technique which is derived from simultaneous bundle
block adjustment of photogrammetry field. In addition, the paper proposes to compute local motion estimates between corresponding pairs of overlapping images via block based optical flow to reduce ghosting problem. The paper also mentions about the technique of roughly calculating unknown camera intrinsic parameter such as focal length from the planar projective motion model or homography of few images.

The book [3] contains a list of selected readings about the real-time signal processing. It’s rich source of articles that are very specific to real-time signal processing. The book briefly discusses about the definition of real-time system and mentions about the importance of real-time signal processing at the beginning. The book has three chapters. The chapter 1 consist a set of papers that are related to hardware implementation of real-time signal processing approaches. The provided articles are very important and intuitive that provides the reader an in depth understanding about the requirements and common practices that are necessary for real-time signal processing. The articles of chapter 2 discusses about the algorithmic approaches that are requirement for real-time signal processing. The third chapter discusses about some implementation level details from application domain.

### 3.2 Real-time Panorama Vision Systems

Latest real-time panorama vision systems have shown considerable improvement in performance than their earliest counterpart. But these systems vary in qualities and throughput because of different production cost and purpose based design. The papers [29, 30, 31] discusses about system’s that uses panorama as their core functionalities.

#### 3.2.1 MITRE immersive spherical vision system

MITRE scientists have developed a vision system that uses six cameras to capture images from 360° around. These images are stitched together into a single high resolution image and projected in a sphere. Moreover, they have also developed head mounted display equipped with sensor technology that can track head movement. The user of the system can look at any orientation of the projected sphere. The view of the projected sphere changes according to the change user’s head orientation. The system has been successfully tested for piloting Unmanned Ground Vehicle (UGV). Moreover, the system also been realized in military tanks for better aerial investigation tank’s surrounding in risky war zones. However, the MITRE scientists have used the commercial product Ladybug developed by Point Gray Research. The developed technology with sensor equipped head display can also be realized in gaming, virtual tourism, civilian security, civilian telerobots and underwater exploration. A sensor equipped head display is shown in the figure 13.
3.2.2 Point Grey Spherical Vision

Point Grey is a world leading Canadian company that have innovated many digital vision products with state-of-the-art quality. It has also developed a spherical vision system called Ladybug.

3.2.2.1 Ladybug®2

Ladybug®2 is a spherical vision system that consist six Sony 0.8 MP 1/1.8” ICx204 color image sensor. Five sensors are positioned horizontally while having one sensor on the device head. The system is able to stream 1024x768 resolution image from each sensor at the rate 30fps. The weight of the system is 1190g. The figure 14 shows Ladybug 2.

Figure 13: MITRE immersive spherical vision [32, 33, 34, 35]

Figure 14: Ladybug 2
3.2.2.2 Ladybug®3

Ladybug®3 is the enhanced spherical vision system from the Ladybug product series. It consists of six Sony 2 MP 1/1.8” ICX274 CCD color sensors. These sensors are positioned similar to Ladybug®2. It can stream composited 12 Megapixel image at the rate of 15fps. Apparently, the system can stream high resolution images by compensating frame rate. It is larger in size than Ladybug®2 as well as weight. But the system consumes 4W less power than Ladybug®2 at the same voltage level. The figure 15 shows Ladybug 3.

![Ladybug 3](image)

Figure 15: Ladybug 3

3.2.3 Lucy S and Dot

Lucy S is a new phenomenon in the field of spherical panoramic video developed by Kogeto. It depends on lens based solution to capture 360 panoramic video instead of “image stitching”. The system can work in any environment because of its lens based solution. The commercial package of the system consists of a Lucy lens and a laptop. The figure 16 shows Lucy S.
Dot is another spherical vision product from the Kogeto to capture 360 panoramic video. But is it not a system itself. Rather it is an extended lens to the Iphone 4/4S that uses the Iphone 4/4S video camera to capture the video. The figure 17 shows Dot.
Chapter 4

Embedded RTLinux and Software Development Tools

The configuration of the embedded operating system is a core part to develop the software structure of the imaging system. This configuration process requires understanding about UNIX or Linux based software tools. The build process of necessary software libraries and API require solid understanding about those libraries. The exact reasons and know-how to use those software tools is discussed in this chapter.

4.1 Linux Kernel

All types of operating system consists some types of kernel to control hardware resources like CPU, memory and I/O devices. Linux kernel is first developed and unveiled by Linus Torvalds in 1991. Since then it has been ported to different CPU architecture including DEC Alpha(AXP), Hitachi SuperH, IA-64, i386, Motorola 68K series, The MIPS processor, Motorola PowerPC, S390, Sparc, ARM, Atmel AVR, Philips LPC ARM, Microchip PIC, TI MSP430. Today, most of the world’s supercomputers are running with different variants of Linux. Furthermore, a substantial number Linux distribution has been developed since its beginning. Linux has become popular because it is open source, scalable and reliable.

Figure 18: Fundamental architecture of Linux [1]
Hardware components actually do the main computation tasks and kernel processes solely control these computation tasks. Third party user application can interact with kernel directly via kernel interface or via C library. A total embedded application is able to run itself directly as a kernel thread without being dependent on C library or other application layer code. A fundamental architecture of Linux is shown in figure 18.

4.2 Basics of Linux

In terms of usability, there are differences between windows and Linux. It is not necessary for a general user to have knowledge about computer technology to use windows. On the contrary, Linux requires some basic understanding of shell programming commands to use it. Earlier distributions of Linux were more shell oriented. But recent Linux distributions like Ubuntu have lessened the gap between Windows and Linux. The figure 19 shows Linux directory structure.

![Linux directory structure](image)

**Figure 19: Linux directory structure [1]**

4.3 Linux distributions

Linux distributions have been made based on different Linux kernel. There are different versions of the Linux kernel. All Linux distributions use some version of the kernel but they have different configurations. The difference in configuration among the distributions occurs because
of applying different patches during the build. The word “distributions” itself tells about the characteristics of Linux distributions. Because of this reason Linux distributions are prefixed or suffixed with term “Linux” while having different configuration. The relation between Linux kernel versions and various Linux distributions is shown in figure 20.

![Figure 20: Linux kernel and Distributions](image)

### 4.3.1 Ångström distribution

Ångstrom is OpenEmbedded software framework based Linux distribution. It is developed for small handheld devices and smaller embedded development boards. It offers less software packages in default build image than other Linux distributions and also known as Linux minimal file system. But its package manager “ipkg” can be used to install more software by feeding from its software repositories.

### 4.3.2 Ubuntu Linux distribution

Ubuntu [36] is the more popular Linux distribution. It has become popular because of its attractive user interface and enhanced usability. Beside, Ubuntu has a very rich software repository. Moreover, Ubuntu is well suited in both Personal Computers to smaller handheld embedded development boards.
4.4 Embedded Linux

The term “Embedded Linux” [2] is used because of use of Linux in Embedded Systems. Embedded systems are dedicated computers that only do some specific computational task. There are versatile embedded systems. It ranges from smaller computing systems to large computation systems. Different kinds of embedded Linux distributions have been developed based on different versions of Linux kernel. A Fox embedded micro Linux system is shown in figure 21.

![Figure 21: Fox embedded micro Linux system [1]](image)

4.5 Real-time Linux

Linux is a standard general purpose operating system but do not fulfill requirements of a real-time process. A real-time operating system must fulfill some requirements to be able to accomplish a real-time process. And, a real-time operating system is a fundamental requirement to meet the pre-requisites of a real-time system. Soft real-time systems must need to have at least following characteristics-

1. Predictable process response time
2. Jitter (deviation) can be pretty large
3. Jitter will utmost cause loss of performance
4. Highly schedulable

Hard real-time systems must need to have at least following characteristics-

1. Predictable process response time
2. Jitter (deviation) must need to be very slight
3. Large jitter will cause system crash
4. Highly schedulable

But general purpose Linux kernel do not poses the qualities that can make it a real-time operating system. Most of the popular general purpose Linux distributions contain a generic kernel by default. A normal Linux kernel scheduling algorithm is designed to be fair in allowing
CPU computation resources to all active processes in the computer. But in real-time operating system, we often need to set priority for one or more specific processes and have to allow full CPU resources to a computation process. However, several solutions already been developed to use Linux as real-time operating system for both embedded and non-embedded devices. The approach to use Linux as real-time operating system can be divided into three categories.

1. Substantial re-writing of the Linux kernel to develop real-time Linux OS
2. Adding an thin micro-kernel below the Linux kernel
3. Applying real-time patches to Linux kernel

A commonly used method is applying a real-time kernel patch. In this method, the requirements for real-time operations are achieved by applying kernel patch to the generic kernel. Currently, several real-time kernels [37] are available for Ubuntu Linux, which are prepared by applying different real-time patches. These customized kernels include following soft real-time and hard real-time versions. The “-preempt kernel” and “-lowlatency kernel” provides real-time characteristics with solid reliability, good power saving and throughput. However, only “-preempt kernel” and “-rt kernel” are available publicly via Ubuntu official archives.

- -preempt kernel: A soft real-time kernel
- -rt kernel: A hard real-time kernel
- -lowlatency kernel: A soft real-time kernel
- -realtime kernel: A hard real-time kernel

Moreover, There are linux based real-time operating system is also available on the market that does not require any patches. Some of these OS are proprietary and some are developed via community effort under GPL Licenses. Both proprietary and GPL OS have their own tradeoff. However, in this thesis project, we have used non-proprietary real-time Linux OS.

The TimeSys Linux/RT [38] is a real-time Linux based operating system for embedded systems developed by TimeSys Corporation. It is a very powerful real-time operating system that provides solutions to major problems in real-time application development. It uses Linux Resource Kernel (Linux/RK) as its base, developed at Carnegie Mellon University’s Real-Time and Multimedia Systems Lab. In total, Standard Linux Components, Linux Resource Kernel (Linux/RK), RED Linux (RED), Real-time Applications Interface (RTAI) are the basic component of TimeSys/RT operating system. The code for the Linux resource Kernel (Linux/RK), RED Linux, Real-Time Applications interface (RTAI) is written as Loadable Kernel Modules (LKMs). The LKMs can be loaded to the main kernel for additional functionality and can be unloaded while not necessary. Thus TimeSys Linux/RT covers wide range of functionalities while keeping smaller footprint size.
However, there are more clever ways to achieve the real-time characteristics from Linux. RTLinux is such a solution which solves the problem with minimal effort. In this thesis, we have used the RTLinux which is an actually a micro-kernel. The RT-microkernel runs real-time processes and runs the whole Linux kernel as one of its low-priority process. But when there is no real-time task to be computed then it runs the Linux kernel. This is how; real-time computation goal is achieved while keeping the Linux kernel intact.

4.5.1 Why not re-writing Linux kernel as real-time kernel?

We have mentioned earlier that Linux kernel scheduling algorithm is designed to be fair. A substantial amount of kernel re-writing is necessary to write a preemptable, low latency new kernel with low interrupt processing. This effort is almost equivalent to develop a new kernel. That is why; real-time characteristics are achieved through applying kernel patches by bypassing this tremendous effort of kernel re-writing.

4.6 Bourne Shell

Bourne Shell [39] is a text based command line interface that has been used in almost all Linux and UNIX distributions. It was first developed by Stephen Bourne. Later, C Shell and Korn Shell also developed with additional capabilities. The Shell does not require compilation as it is an interpreted Language. In this project, shell is used in different stages of system configuration.
The figure 22 shows UNIX shell family tree. The most basic shell commands of Ubuntu are given below. These commands have been adopted from the UNIX and Linux generic shell commands. The basic Linux shell commands are given in the figure 23.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Shell Command</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home directory</td>
<td>su</td>
<td>sudo su</td>
</tr>
<tr>
<td>Change directory</td>
<td>cd</td>
<td>cd &lt;directory name&gt;</td>
</tr>
<tr>
<td>Going one directory back</td>
<td>cd -</td>
<td>cd -</td>
</tr>
<tr>
<td>Going back to root directory</td>
<td>cd --</td>
<td>cd --</td>
</tr>
<tr>
<td>Showing current directory</td>
<td>pwd</td>
<td>pwd</td>
</tr>
<tr>
<td>Create directory</td>
<td>mkdir</td>
<td>mkdir &lt;directory name&gt;</td>
</tr>
<tr>
<td>Delete directory</td>
<td>rm</td>
<td>rm –rf &lt;directory name&gt;</td>
</tr>
<tr>
<td>Copy files</td>
<td>cp</td>
<td>cp &lt;filename&gt; &lt;new location&gt;</td>
</tr>
<tr>
<td>Move files</td>
<td>mv</td>
<td>mv &lt;filename&gt; &lt;new location&gt;</td>
</tr>
<tr>
<td>Rename files</td>
<td>mv</td>
<td>mv &lt;old filename&gt; &lt;new filename&gt;</td>
</tr>
<tr>
<td>Delete files</td>
<td>rm</td>
<td>rm &lt;filename&gt;</td>
</tr>
<tr>
<td>Open/ Edit text editor</td>
<td>nano</td>
<td>nano &lt;filename&gt;</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
<td>---------------</td>
</tr>
<tr>
<td>Open/ Edit text editor</td>
<td>vi</td>
<td>vi &lt;filename&gt;</td>
</tr>
<tr>
<td>Clear terminal</td>
<td>clear</td>
<td>clear</td>
</tr>
<tr>
<td>Help</td>
<td>man</td>
<td>man &lt;info or other command&gt;</td>
</tr>
<tr>
<td>Quit</td>
<td>exit</td>
<td>exit</td>
</tr>
<tr>
<td>System shutdown</td>
<td>shutdown</td>
<td>sudo shutdown –h now</td>
</tr>
<tr>
<td>System reboot</td>
<td>reboot</td>
<td>sudo reboot</td>
</tr>
<tr>
<td>Text editor line number</td>
<td>-ci</td>
<td>nano –ci &lt;filename.filetype&gt;</td>
</tr>
<tr>
<td>Open Image</td>
<td>gnome-open</td>
<td>gnome-open &lt;imagename.imagetyp&gt;</td>
</tr>
<tr>
<td>Becoming superuser</td>
<td>sudo</td>
<td>sudo su</td>
</tr>
<tr>
<td>Check Mounted file system</td>
<td>df</td>
<td>sudo df</td>
</tr>
<tr>
<td>CPU information</td>
<td>cat /proc/cpuinfo</td>
<td>cat /proc/cpuinfo</td>
</tr>
<tr>
<td>Memory information</td>
<td>cat /proc/meminfo</td>
<td>cat /proc/meminfo</td>
</tr>
</tbody>
</table>

Figure 23: Basic Linux Shell command

### 4.7 Text Editors

During the development keyboard oriented and screen oriented text editors plays an important role to access system files, creating new files and native compilation of algorithms. The two most popular editor is “Vi” editor and “nano” editor. Both of these editors are available almost on all UNIX based operating system. We have used both “Vi” and “nano” editor in the system development. However, “nano” is more convenient and powerful in compare to “Vi” editor from many aspects.

#### 4.7.1 Vi editor

Vi(Pronounced vee-eye) is an screen oriented text editor is available in almost all UNIX based operating system as well as Linux. It has been developed by Bill Joy and initially released in
1976 under BSD license. Now-a-days, there are better alternative available for editing but “vi” also works smoothly for an expert user. However, practically, many of vi commands does not work now because of considerable improvement on UNIX environment and various Linux distributions. The tasks that can be done using a “Vi” editor are -opening and closing a file, moving around in a file, elementary editing. The figure 24 shows a screenshot of the “Vi” editor.

![Figure 24: Vi editor](image)

### 4.7.2 Nano editor

“nano” is a relatively new keyboard oriented text editor in compare to “vi” editor. It has been developed by Chris Allgretta and initially released in 1999 under GNU General Public License. In terms of software usability, “nano” is better than “vi” editor. For native compilation of algorithms, “nano” offers a very quick command to create “.c” and “.cpp” file. The figure 25 shows a screenshot of the “nano” editor.
There are two types of compiler- native compiler and cross-compiler. The native compiler generates machine code for the computer platform on which it is running. On the contrary, cross-compiler generates the machine code for different computer platform rather than on which platform it is running. There are two types of native compiler available for UNIX or Linux distributions. These are- gcc and g++. “gcc” is a GNU C compiler from the GNU Compiler Collection (GCC) [40]. This compiler can compile c code from “.c” file and generates executable binary. And, “g++” is a GNU C++ compiler from the GNU Compiler Collection (GCC). This compiler can compile C++ code from “.cpp” file and generates executable binary for the native platform. The 2nd line of the following figure shows that GNU GCC is available during configuring and building OpenCV by cmake and make respectively. The figure 26 shows detected GNU version during OpenCV build.
OpenCV [41] is a powerful Open source library of different programming functions that have wide range of application in the field of computer vision and image processing. The library was first developed by Intel but later became open source. Currently, the library is maintained by Willow Garage Robotics Company. The library is divided into some structures then in different modules. Each module offers ready-made functions to implement different image processing and computer vision tasks. However, all structures commonly shared by all modules. The figure 27 shows basic structure of OpenCV.
4.10 Native code and Linking Process

A computer processor only understands instruction set which control different operations of registers including increment and decrement. These instruction set composed of 0 and 1 which is called machine code. Single computer architecture or a family of architecture with the successors and predecessors has their own machine code which is called native code. It means platform specific machine codes are called native code. However, programmers do not write instruction set using machine code because it require robust technical details of the processor as well as memorizing numerous numerical code for every instruction. Hence, it increases the system development time. Consequently, programmers use different layers of abstraction to create these instruction sets in a shorter time. The programming language that have lowest amount of abstraction to machine code is called Low-level programing Language (e.g. Assembly language). The programing language that have higher amount of abstraction to machine code is called High-level programming language (e.g. C++, Java).

We now know that every processor architecture family only understands their native machine code. We write computer programs in high-level language because directly writing machine code by assembly language is complex, error prone and time consuming task. Therefore, we need a compiler that translates this high level source code to the platform specific native code or machine code. However, we are familiar with Integrated Development Platforms (IDE) like Microsoft Visual Studio or Eclipse. We compile one or several source files just by clicking one build button by using these development platforms.

Large software libraries or API most often contain hundreds of source files that are categorized under different module. These large amount source files often have interrelation between them to generate a specific target non-source file. Consequently, compiling each source file manually at a time and manually linking between them to generate a single non-source file is a highly time consuming and disgusting task. Hence, we use the build automation tool that automatically compiles all source files of the library, make necessary linking process and generates necessary executable file and other necessary non-source files.

4.11 Build Automation Tool

There are a lot of tools are available for Build Automation. Different software packages support different tools. The tool need to be chosen based on the software packages required to install. In our system development process, we have installed OpenCV library. The latest release of OpenCV-2.3.1 requires Cmake to install it in the desired platform. Cmake is compiler and platform independent and it uses the GNU compiler and native “make” build system of the Gumstix's Ångström Linux operating system.
4.11.1 Build automation tool for OpenCV

OpenCV library is a huge collection of algorithms and thus contains hundreds of source files. The library is divided into different modules and each module contains several source file and corresponding header file. These source files needed to be compiled to generate executable binary object. The manual compilation of each source file and linking of corresponding objects into executable objects is a huge and time consuming task for a developer. That’s why, we have to use a build automation tool that will automate this process and generates all the necessary objects and links them as appropriately to generate executable objects per module.

To mention, latest OpenCV-2.3.1 release generates 14 executable binary objects after installation and each object performs their designated task. Of course, we did not use the all objects in our system development but during the building we have to build the whole library. A question may arise, if we did not used all object than why to build the whole library that consumes time. The answer is whole OpenCV library is packaged into a single source tree. And, the parameters from the source tree are written in the Cmake root file for the OpenCV “Cmakelists.txt”. If we want to build only specific module than we must need to remove those module from the source tree and have to make corresponding changes in the Cmake root file “CmakeLists.txt”.

4.12 CMake

CMake is an open source and cross-platform build automation tool that generates makefiles for Linux, other UNIX platforms, Windows, Mac OS X and so on. The build process with CMake is done in two stages. Firstly, CMake configures the OpenCV-2.3.1 and generates build file and writes it to the build directory. Secondly, Ångstrom’s native build tool “make” is used to build and install these build files. We have also used CMake variables during the configuration and build process. These variables help the developer to configure the OpenCV installation as needed.

4.12.1 Philosophy of using CMake

In real world, there is no unified vendor for making computing systems. Since the beginning of digital boom of bits and bytes, several technology corporations have been emerged who have designed their unique hardware and software architecture. Subsequently, the market of digital computing systems is shared by several architectures with their corresponding market share. As a result, an application programming interface (API) should be executable on all available architectures on the market to be able to catch greater market share. However, to create a native software library for highly popular to less popular architectures is not an economically feasible
solution. Hence, a need for universal software paradigm is emerged and subsequently, application programming interfaces has been written once for all architectures. And, cross-platform build systems like CMake generate native source code of API for each of the architectures. Later, the native source code is build using native build tool like make and so on. The figure 28 shows CMake functionality.

![Figure 28: CMake functionality](image)

The operation of CMake can be depicted using an analogy example. Let us think a collection of pieces of bricks as a universal application programming interface library and CMake as the builders. Similarly, different hardware and software computing architectures are considered as different houses with different design and layouts. The builders have to build houses of different design and layouts with bricks. Here, the builders are building different houses using same construction material bricks. But because of different design and layouts the houses are different. Here, builders are working as transformation catalyst that is shaping the raw bricks into a useful shape according to the required house design and layouts. From the viewpoint of software architecture paradigm, the CMake works like a transformation catalyst that transform universal application programming interface code into native source files for different computing architectures using corresponding system parameters. The figure 29 shows a CMake analogy representation.
4.13 Make

GNU “Make” is a software that do automated compilation task. But *Make* get the knowledge of compiling source codes from *Makefile*. The *Makefile* is written by the developers of a software library or API. The *Makefile* know the necessary program details with a list of non-source files and know how to generate non-source file from source files. Hence, any developer can use *make* to build and install any software libraries that includes *Makefile* with it. *Makefile* uses a rule that tales *Make* which shell commands are necessary to build a target from source files. If dependencies are necessary to generate a target file, then it also specifies lists of all those dependencies.

For example, in this thesis project, we have first build and installed software dependencies (e.g. FFmpeg, V4L/V4L2, GStreamer, x264) in the development computer. Then we have used CMake to configure the OpenCV library with necessary dependencies and generated the native *Makefile* that has a list of necessary dependencies which OpenCV uses and knows where to find them.

Moreover, a developer may need to change some source code during the system development. In this case, re-compiling all source files of the entire software library surely will be waste of computation resources and time. To solve this problem, *Make* has some intelligent features that help developer to reduce the development time. For example, Based on which source files have been changed *Make* automatically knows which non-source files need to be updated. Hence, *Make* will update only those non-source files that are directly or indirectly linked with changed source files after re-running *Make*.
For example, in this thesis project, we have often used OpenCV stitching module and changed its different source codes. After this changing process we have rerun the Make. But this rerun has taken much lesser time than the first build. It happened because; Make knows which files it needs to re-compile based on its knowledge which files we have changed. Hence, Make only compiled stitching module and therefore less time is taken during this rebuilding. The CMake and make workflow is shown in figure 30.

![Figure 30: CMake and make workflow](image)

### 4.14 pkg-config

“pkg-config” is a relatively small but powerful software tool. Its unified interface do queries for installed libraries that is necessary for compiler to compile library dependent code. It can query and provides parameters for gcc and g++ compiler, linker parameter, version of the package from the installed libraries.
4.14.1 Reason of using pkg-config

We usually use software programs that use external software libraries or API (e.g. OpenCV, FFmpeg) during any kind software development project. However, a compiler must know the location of external libraries when we try to compile source code that is fully or partially dependent on external libraries. Otherwise, the compiler will show error because it cannot get linked with external libraries source code. However, to solve this problem, we need a software tool that tells the location of the installed external libraries of the computer to the compiler. And exactly that location telling job is done by “pkg-config” software tool.

4.15 FFmpeg

FFmpeg is a powerful API that includes leading audio-video codec library libavcodec. It helps to record, covert and stream audio and video. It is free software under GPL or LPGL license. OpenCV uses this API to facilitate simultaneous video capture and processing. For that reason, we must need to make sure that FFmpeg is installed in the development machine before building OpenCV API for video capture and processing. In the following figure, the “YES” status of FFmpeg API under “Video I/O” indicates that FFmpeg is available in the computer and activated for the OpenCV build. The figure 31 shows “FFmpeg” status during OpenCV build.

![Figure 31: Enabled FFmpeg during OpenCV build](image.png)
4.16 V4L/V4L2

Video for Linux 2 is API that facilitates video capture in Linux operating system. Video for Linux 1 was the predecessor of V4L2. It’s integrated with the Linux kernel. OpenCV’s highgui module uses this API to facilitate video capture. There are different V4L/V4L2 capture parameters that can be changed from the OpenV’s highui module source files. In the following figure, the status “Using libv4l” indicates that V4L/V4L2 is available in the computer and activated for OpenCV build. To mention, status “1/1” instead of “Using libv4l” also indicates that V4L/V4L2 is available and activated. The figure 32 shows V4L/V4L2 status during OpenCV build.

![Figure 32: Enabled V4L/V4L2 during OpenCV build](image)

4.17 GStreamer

GStreamer is a powerful cross-platform pipeline-based multimedia framework. It can be used with OpenCV and edited from OpenCV’s highgui module. Based on this platform various applications can be developed for recording, audio-video playback, streaming, editing and other imaging applications. In the following figure, the “YES” status of GStreamer under “Video I/O” indicates that GStreamer is available in the computer and activated for OpenCV build. The figure 33 shows GStreamer status during OpenCV build.
4.18 x264

X264 is a video codec library that encodes video streams into widely used format H.264/MPEG-4 AVC format. It is free software under GNU license. We have used this library as a dependency of OpenCV library.

4.19 USB Video class device (UVC)

UVC is a standard which support a group of cameras from different manufactures. It supports transmission of compressed video stream in various formats. In Linux operating system, the support for UVC is provided by LINUX UVC driver.

4.20 GDB (GNU Debugger)

The error in the source code often termed as BUG. Thus, debugger means removing the bug or error from the source code which is already been reflected by the name Debugger. GDB is a powerful programming tool or debugger that helps to debug source code. It has been updated in an often manner since its first release in 1986. A software program must need to be build as a debug version to debug that program by GDB. Otherwise, GDB will not be able to read symbols from the targeted executable binary. GDB offers an enriched set of feature commands that offers
flexible debugging after initiating debugger with the desired executable binary. The figure 34 shows an instance of debugging during the development.

![Debugging Image]

Figure 34: An instance of debugging

In the following chart, we are providing some useful GDB commands. The characters of the command syntax that are written under the [] bracket can be exempted.

### 4.20.1 Useful GDB Commands

<table>
<thead>
<tr>
<th>Command syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>b[reak] &lt; name &gt;</td>
<td>set a breakpoint at function</td>
</tr>
<tr>
<td>b[reak] &lt;class&gt;::&lt;name&gt;</td>
<td>set a breakpoint at &lt;name&gt; in &lt;class&gt;</td>
</tr>
<tr>
<td>r[un]</td>
<td>executes to the next breakpoint or to the program end</td>
</tr>
<tr>
<td>s[tep]</td>
<td>a single step, descending into the functions</td>
</tr>
<tr>
<td>n[ext]</td>
<td>a single step, without descending into the</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>c[ontinue]</td>
<td>Ccontinue to the next breakpoint or to the program end</td>
</tr>
<tr>
<td>p[rint] &lt;name&gt;</td>
<td>Print value of the variable called &lt;name&gt;</td>
</tr>
<tr>
<td>p/x &lt;name&gt;</td>
<td>Print value of &lt;name&gt; in hex format</td>
</tr>
<tr>
<td>up</td>
<td>Go up one level</td>
</tr>
<tr>
<td>do[wn]</td>
<td>Go down one level (applicable after up)</td>
</tr>
<tr>
<td>fin[ish]</td>
<td>Finish current function or loop etc.</td>
</tr>
<tr>
<td>i</td>
<td>List information about commands</td>
</tr>
<tr>
<td>i[info] b</td>
<td>List breakpoints</td>
</tr>
<tr>
<td>cond[ition] 4 &lt;expr&gt;</td>
<td>Stop at breakpoint 4 if &lt;expr&gt; is true</td>
</tr>
<tr>
<td>dis[able] 4</td>
<td>Disable breakpoint 4</td>
</tr>
<tr>
<td>d[lete] 4</td>
<td>Delete breakpoint 4</td>
</tr>
<tr>
<td>d 4 5</td>
<td>Delete breakpoints 4 and 5</td>
</tr>
<tr>
<td>d</td>
<td>Delete all breakpoints</td>
</tr>
<tr>
<td>ena[ble] 4</td>
<td>Enable breakpoint 4</td>
</tr>
</tbody>
</table>
Chapter 5

Vision, Camera and Imaging Methods

The concrete foundation of vision and imaging techniques goes back to 987 years to Ibn al-Hytham. Ibn al-Haytham [42, 43, 44, 45, and 46] first developed the founding basis of modern physical optics Theory of Vision [44] and principal of photography Pinhole Camera [42]. The works of Ibn al-Haytham was published in his book Kitab al-Manazir or Book of Optics I-VII [47, 46]. Moreover, the work of Descartes, Kepler, Galileo, Hooke, Newton, Snell, Fermat and Euler formed the mathematical and physical advances during 1600 and 1700s. However, the modern imaging technologies and cameras are far more superior to their earlier counterparts. The contribution of numerous researchers, research organizations and business corporations made ultra-modern imaging technologies and applications possible.

![Digital Imaging from high level point of view](image)

Figure 35: Digital Imaging from high level point of view
The figure 35 shows digital imaging from a high level point of view.

5.1 Vision, Human Eye and Camera

The world surrounding us is the result of contrast in the reflected light. This contrast is resulted because of different spectrum of visible light. Our brain interprets this contrast and perceives the dynamics of contrast as vision. If there was no contrast surrounding us, the known world will turn into a world of uniform luminosity and subsequently, human brain will fail to understand contrast of colors. We have developed the method to record this contrast dynamics and this ability resulted non-native image. The established theory of electromagnetic radiation explains interaction between real-world object and light.

In principal, vision is the way of perceiving the external world surrounding us. The light source (e.g. sun during day time, light bulb during night) are the two common sources of light. These sources of light emits light ray which struck at the real-world objects. Real world objects absorb much of the light and rest of the light gets reflected. When this reflected light rays from a particular object enter into retina [48] of our eye, we see that object and color. The figure 36 shows how electromagnetism helps in human vision.

Figure 36: Electromagnetism and human vision [1]

Moreover, the visible light is actually a small fraction from the vast electromagnetic spectrum. The electromagnetic spectrum ranges from smaller high-frequency Gamma ray to low-frequency Radio wave. The light we experience in our surrounding world is actually lays inside the visible fraction of electromagnetic spectrum. Furthermore, electricity and magnetism were thought to be separate force until James Clerk Maxwell [49] defined the relationship between electricity and magnetism in 1873 in his book *Treatise on Electricity and Magnetism*. The mechanism that explains the relationship between electricity and magnetism is known as electromagnetism [49].
The figure 37 shows electromagnetic spectrum and visible spectrum and the figure 38 shows Al-Haythm’s study of human eye. The concept of camera is inspired by the understanding of human-eye structure. Ibn Al-Haythm’s study of lens, retina and optic nerve was a pioneering work in the field of vision and perception mechanics. It appears that he became inspired to formulate the principle of pinhole camera from his study of human eye. Haytham’s pinhole camera model is the classical way of representing geometrical structure of vision.
Figure 39: Modern study of human eye [1]

The figure 39 shows modern study of human eye. The modern study human eye has revealed that human vision is much more complex than it looks like in superficial view. Many aspects of primary human vision (e.g. interpretation of optical nerve signal via brain) are not properly understood yet. However, the cutting edge research on human eye is actively running at various research organizations and may contribute to further understanding of human vision.

Figure 40: The human eye and vision [1]
In the figure 40, the basic principle of primary human vision is depicted. There are several analogies between human eye and camera in terms of structural component and vision process. It is understandable because the basic process of vision and classical pinhole camera was inspired by human eye.

5.2 Image Formation and Pinhole Camera Model
The scenarios that surround us are in three-dimensional world. The camera maps this physical world scenario into 2D image plane. The unit of the physical world is (e.g. meter) while unit of the camera is pixel. The geometry of the camera model is a fundamental requirement to understand this mapping of 3D world co-ordinates to 2D Cartesian coordinates (Image coordinate). Moreover, camera uses lenses to gather more light which provides better picture. This impact of converged light and diverged light can be practically understood from real camera sensing. We have opened up a USB digital camera to demonstrate this phenomenon.

Figure 41: Image capture using diverged light

In the figure 41, the right image was taken with diverged light that falls upon on the naked image sensor. The sensor can be seen from the left image which is located on the centre point of the camera circuit board.
In the figure 42, the right image is taken with lens-converged light that falls upon on the same sensor. The difference between right image of the first figure and right image of the latter figure demonstrates how convergence of light impact imaging. But the lens introduces some distortion which has a strong influence on image quality. The basic geometry of pinhole camera model is simple but necessary to be extended to compensate for the distortion introduced by the lens. The process of mathematically correcting the deviations from the simple pinhole camera model that is caused by the use of lens is known as camera calibration. The process of camera calibration uses two informational models (i.e. camera’s geometry and distortion model of the lens). We get the camera intrinsic parameters from these two informational models.

The simplest pinhole camera model of figure 43 is not enough to produce a high quality image because of less concentration of light through the pinhole. But understanding the basic geometry of simplest camera model is of particular importance to understand further mathematical manipulations.
In the figure 44, X is the length of a real world object from which light is emitted. The emitted light ray entered inside camera through the pinhole aperture and projected onto the image plane or projective plane upside down. The distance between pinhole aperture and image is the focal length f. The size of the object in the projected image relative to the real-world object can be perfectly measured by focal length. The distance from the camera to the object is given by Z and object’s image on the image plane is given by x. A perpendicular line from image plane to real-world object through the pinhole aperture point is perceived as optical axis. According to similar triangle relationship, the following mathematical relationship holds for this mathematical model of pinhole camera.

\[-x = \frac{X}{f} \text{ or } -x = f \frac{x}{z}\]

However, this mathematical presentation results an inverted and inconvenient projected image with negative sign. To resolve this issue another mathematically equivalent model of pinhole camera is developed in which pinhole and image plane are swapped. Subsequently, the image plane and the projected object are now appearing frontal and right-side up respectively.
Figure 45: Pinhole camera model geometry 2 [41]

In the figure 45, the point of pinhole aperture is renamed as *centre of projection* and the point at the intersection of image plane and perpendicular optical axis is renamed as *principal point*. The following mathematical relationship holds for this representation of pinhole camera model.

\[
\frac{x}{f} = \frac{X}{Z} \quad \text{or} \quad x = f \frac{X}{Z}
\]

We can see that, the projected image x is now positive because it is not upside down. In this way, the pinhole camera’s geometric model is perceivably convenient and mathematically equivalent with the previous model. The image plane represented here is actually the imaging chip in digital cameras. And, the centre of the chip is usually not on the optical axis because of subtle inaccuracy. Subsequently, there is a possible displacement of the centre of the co-ordinates on the imaging chip. To model this displacement, two new translation parameter \( t_x \) and \( t_y \) are introduced for X axis and Y axis respectively. Consequently, the projection of 3D world object Q with co-ordinates X, Y, Z on the imager or image plane at pixel location \( (X_{\text{screen}}, Y_{\text{screen}}) \) is given by-

\[
X_{\text{screen}} = f_x \left( \frac{X}{Z} \right) + t_x, \quad Y_{\text{screen}} = f_y \left( \frac{Y}{Z} \right) + t_y
\]
To measure \((X_{\text{screen}}, Y_{\text{screen}})\) with resolution \(r_x\) and \(r_y\) pixels/inch in both \(x\) and \(y\) direction, we can express the above relation as follows-

\[
X_{\text{screen}} = f_x r_x \left( \frac{X}{Z} \right) + r_x t_x, \quad Y_{\text{screen}} = f_y r_y \left( \frac{Y}{Z} \right) + r_y t_y
\]

5.3 Focal Length and Camera Field of View

Focal length is the distance from the centre of the lens to the image plane (e.g. imager or sensor) in which light converges to a focal point. The measurement of focal length is very important in calculating the camera intrinsic parameters.

![Focal Length Diagram](image)

Figure 46: Focal length [1]

The figure 46 shows the definition of focal length. Focal length and camera field of view are interrelated and is a fundamental relation of imaging. If we increase the focal length then field of view decreases and if we decrease the focal length then field of view increases. It means that with a shorter focal length we can converge or diverge greater amount light rays. On the other hand, with longer focal length we can converge or diverge relatively less amount of right rays. The figure 47 shows the relation between focal length and field of view.
5.4 Camera Intrinsic and Extrinsic parameters

We transform real world scenario into synthetic scenario via mapping real world 3D point into camera pixel. This camera transformation process is characterised by two relations. One is inner relation which fully belongs to the camera physical structure. Another is outer relation which belongs to the real world. The parameter that defines the inner relation is called intrinsic parameter. And, the parameter that defines the outer relation is called extrinsic parameter. We can categorize intrinsic and extrinsic parameters as follows-

**5.4.1 Camera intrinsic parameters**
- Focal length
- Two Centre of Image plane displacement modeling parameter
- 2-3 Radial distortion modelling parameter
- 2 Tangential distortion modelling parameter
- Image Plane coordinates (2D)
- Pixel coordinates (2D,int)

**5.4.2 Camera Extrinsic parameters**
- Rotation and Translation
- World coordinates (3D)
- Camera coordinates (3D)
5.5 Philosophy of using Mathematics in Secondary vision

Our principal source of vision is the eye-brain interactive system. In real world, we see different objects in different shape because of both different human body and object mode such as movement, distance, and angle. For example, a bookshelf looks different from side than it looks from the front. Moreover, seeing a moving object and still object gives different perspective. Furthermore, looking at objects with substantial movement and without movement naturally provides different observation and multidimensional clarity. However, this pleasure of seeing things at their multidimensional clarity has become possible because of the automatic processing of our brain. However, we have created our own source of secondary vision through imaging. Now, the question is how to give multidimensional clarity to objects in our manipulative imaging system or artificial graphical world? The mathematical model of rotation, translation and scaling etc. and 3D reconstruction at large has come into existence as an answer to this question. Subsequently, we have become able to represent multidimensional clarity to the world of graphics using these manipulative mathematical models.

5.6 Projective Geometry

The projective geometry [51,52,53] is an important area of mathematics that has applications in modern linear algebra, hyperbolic geometry, spherical geometry and in modern graphics programs. The knowledge of projective geometry goes back to the ancient greek age. The basic structural concepts in projective geometry are projective point, projective line, projective plane, and projective space.

5.6.1 Euclidean space and Projective space

When a Euclidean space is extended by a theoretical correspondence to infinity in terms of relative point, line and plane at infinity then a projective space is formed. All the properties of Euclidean space are compatible in projective space. However, when we use Euclidean space, we are able to describe objects “as exactly as they are”. But when we use projective space, we are able to describe objects “as exactly as they are” and as well as “as exactly as they appear”. The figure 48 shows Euclidean space and projective space representation.
5.6.2 Projective point, line, plane and space

In the projective space [51,54,52], each point belongs to a line, and each line belongs to a plane and each plane belongs to a space. Moreover, in projective space, each family of parallel lines form a common point at infinity, nonparallel lines form distinct point at infinity, each family of parallel plane form a common line at infinity, nonparallel lines form a distinct line at infinity. The figure 49 shows point, line and space in projective space.
5.6.3 Projectivity and Perspectivity

Two important aspects of projective geometry are *perspectivity* and *projectivity*. Projectivity exists because perspectivity exists. In other word, a sequence of perspectivity gives the projectivity. The question may arise why to care about perspectivity in imaging? The answer is we cannot ignore the phenomenon that exists in the human realized physical world as a result of human-physical world interaction. To mention, perspectivity do not exist in real world, instead it exist in our vision paradigm.
In the figure 50, we can see that, point A, B, C of Line1 are associated with corresponding point A', B', C' of Line2 via centre of perspectivity O. This association of points on Line1 with the points on Line2 is perspectivity. Similarly, point A', B', C' of Line2 are associated with corresponding point A'', B'', C'' of Line3 via perspective point P. This association of points on Line2 with the points on Line3 is another perspectivity. Hence, point A, B, C of Line1 has a correspondence with point A'', B'', C'' of Line3 respectively and forms a projectivity from Line1 to Line3 and vice versa. And, any numbers of sequences of perspectivities that follow one after the other form the projectivity as a whole. How perspectivity affects the projection of real world object into camera image plane. The following clearly depicts how perspectivity affects imaging.
In the figure 51, we can see that, shorter focal length increases the FOV and keeps strong perspective effect. In the left picture, the front side of the building provides appearance of three lines. These three lines provide two parallelisms by sharing middle line twice. And, these lines appear to be tending towards a single point at infinity and this phenomenon preserves strong perspective effect. On the other hand longer focal length decreases the FOV and reduces the perspective effect. The right picture has preserved less perspective effect than the left picture. However, perspectivity in imaging is the representation of reality. Our built-in eye-brain interactive vision system sees the world with various perspective effects. Hence, it is particularly important to represent the similar effect in imaging to represent the best possible secondary representation of the real world. However, the real world perspectivity can be described into several categories such as One-point perspective, Two-point perspective, Three-piont perspective, Four-point perspective and so on. The following images demonstrate an example of One-point perspective and Two-point perspective.
In the left image of the figure 52, we can see how two parallel railway line iron bar visually moves toward a single point at infinity in a gradual manner and results one-point perspective. In the right image of the figure 52, we can how two parallel lines moves in rightward and leftward direction to form two distinct single points at infinity and results two-point perspective. Similarly, more point perspective also exists in real world. The point at infinity to which parallel lines converges often termed as “vanishing point”.

5.6.4 Estimating transformation in Projective space
There are various categories of transformations that occur in projective space. How do we distinguish between these transformations that occur in projective space? In terms of mathematical method, Pappus theorem [56,57], Desargues theorem [56,58] and the relationship between the quadrangles and quadrilaterals [59] forms some very important properties of projective geometry. These theorems formulate various methods to calculate various mappings or to estimate transformation type in projective space. The figure 53 shows pappus theorem, Desargues theorem and relation between quadrangle and quadrilaterals.
In the previous section, we have discussed about the geometrical relation that exist between 3D real world object and its projection onto 2D camera image plane. This geometrical relation is modelled using projective transformation. Homogeneous coordinates [60] are the most convenient coordinates for projective transformation that transforms a real world 3D scene into 2D scene. The figure 54 shows 3D to 2D projection using homogeneous coordinates.

**5.6.5 Projective transformation and Image Formation**

In the previous section, we have discussed about the geometrical relation that exist between 3D real world object and its projection onto 2D camera image plane. This geometrical relation is modelled using projective transformation. Homogeneous coordinates [60] are the most convenient coordinates for projective transformation that transforms a real world 3D scene into 2D scene. The figure 54 shows 3D to 2D projection using homogeneous coordinates.

Figure 53: Pappus theorem, Desargues theorem, quadrangle and quadrilaterals [1]
In homogeneous coordinates, if a projective space is n dimensional then a point in that projective space is considered as (n+1) dimensional vector and it means $(x, y, z)$ becomes $(x, y, z, w)$. Hence, a 2D point $(x, y)$ on the projective space of image plane can be represented as 3D vector $(x_i, y_i, z)$. In the above figure, we have used $f$ instead of $z$, but why? The reason is focal length $f$ is the distance between centre of projection and image plane. We can get 2D value of $(x, y)$ by dividing $x_i$ and $y_i$ by $f$.

$$x = \frac{x_i}{f}, \quad y = \frac{y_i}{f}$$
Subsequently, we can express the camera intrinsic parameters $f_x, f_y, t_x$ and $t_y$ as a 3-by-3 matrix which is often termed as camera intrinsic matrix. The intrinsic matrix projects a 3D real world point $Q$ into a 2D camera point $q$ according to the following relationship.

\[
q = lQ
\]

\[
T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad l = TF = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & t_x \\ 0 & f_y & t_y \\ 0 & 0 & 1 \end{bmatrix}
\]

With skew parameter, \( I = \begin{bmatrix} f_x & s & t_x \\ 0 & f_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \)

\[
q = \begin{bmatrix} x \\ y \\ f \end{bmatrix}, \quad I = \begin{bmatrix} f_x & s & t_x \\ 0 & f_y & t_y \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}
\]

\[
\begin{bmatrix} x \\ y \\ f \end{bmatrix} = \begin{bmatrix} f_x & s & t_x \\ 0 & f_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}
\]

With resolution \( r_x \) and \( r_y \) pixels/inch in both \( x \) and \( y \) direction, we can write,

\[
\begin{bmatrix} x \\ y \\ f \end{bmatrix} = \begin{bmatrix} f_x r_x & s & t_x r_x \\ 0 & f_y r_y & t_y r_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}
\]

Here, \( Q \) is the real world point, \( T \) is the image plane translation, \( F \) is the focal length scaling, \( M \) is the intrinsic camera matrix with Focal length scaling \( F \), image plane translation (centre of imager displacement correction) \( T \) and skew parameter \( s \). \( q \) is the projected point \( Q \) into the image plane.

However, the real world object plane differs by some rotation and translation with the image plane. Hence, we need a rotation and translation to relate object plane with camera coordinate system. The translation for \( X,Y,Z \) coordinates system is given by \( T_x, T_y, T_z \) respectively and hence \( t(T_x, T_y, T_z) \). The 3D rotation is given by \( R \) which is a \( 3 \times 3 \) matrix. Hence, we have a \( 3 \times 4 \) extrinsic parameter matrix \( C \) by applying rotation and translation.
\[
C = [R \ t]
\]

So, the complete camera transformation can be described as-

\[
q = ICQ
\]

Homography, \( H = IC \)

Now, the real world point \( Q \) need to be considered in 4D homogeneous coordinates and camera image plane point \( q \) will be in 3D homogenous coordinates which is transformed by \( H \). Subsequently, by dividing the first two coordinates by the third coordinate we will get back the 2D location of the camera point or image plane point \( q \).

### 5.6.6 Centre of Projection and Binocular disparity

Center of projection is an important geometrical mathematics concept. The concept of centre of projection and its effect can be understood from the following thumps up example. A person ups the thump to an object in a straight position and then looks at the thumb while having the right-eye closed. The person will see a relative change in the position of thump and thump will move rightward from the targeted object. Now, if the person opens his right-eye while having the left-eye closed, he will observe that the position of the upped thumb has moved sharply to the leftward to the object. So the upped thump appears to be in different position in space in a separate one-eyed vision with left-eye and right-eye. The position of upped thumb has appeared to be different in space because of change in centre of projection of the eye. Our right-eye and left-eye has different centre of projection which is caused by the distance between right-eye and left-eye. Actually, we have built-in stereo vision that is provided by combination of right-eye and left-eye. This effect of centre of projection is also occurs in multi-view camera setup either from a single camera or multiple cameras, hence centre of projection constitutes an important factor in imaging. Subsequently, we can understand that, different centre of projection gives the different perspective view and different coordinate system of the object in the space. This difference in object location in space because of horizontal distance between eyes is called binocular disparity. The figure 55 shows impact of centre of projection in human vision.
Figure 55: Centre of projection and Human vision [1]

5.7 Homogeneous Space and Coordinate

The homogenous coordinate [60,61] system is fundamentally important to conveniently understand the transformation that transform a 3D real world object into 2D object. When a projective space uses homogeneous coordinate system then that projective space can be defined as homogeneous space. It provides a unified framework for perspective projection, rotation, translation and scaling. However, the concept of homogeneous coordinate is also a very powerful mathematical tool to explain different phenomenon in other areas of science and technology. The idea of understanding projective geometry using lines through a unified origin in 3D space [Möbius, Plücker] revolutionized the way of dealing with transformation.

A 3D point (X,Y,Z) in real space is represented as four-element vector (x,y,z,w) in homogeneous space. There is a relation between w and x, y, z. If we divide x, y, z by w then we will get back values of X, Y, Z in real space. So, the values of X, Y and Z in real space is the proportion between x, y, z and w. Hence, the following relation holds back and forth between real space and homogeneous space.

\[
X = \frac{x}{w}, \quad Y = \frac{y}{w}, \quad Z = \frac{z}{w}
\]

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

We can see that, as we get back value of a real space point as a result of proportion in homogeneous point, we can represent a real space point using various homogeneous space vectors. And, the proportionality gives the power to scale the object in projective space without affecting its content. The power of homogeneous coordinate is that, we can represent nearest point in real space as well as points that are infinitely far way under a mathematically convenient unified coordinate system. For example, suppose a homogeneous space vector is (2,0,0,w). The corresponding real space vector of this homogeneous space vector is (2/w, 0, 0). We can clearly
see that, the more we make the w smaller the bigger the value of the point X in real space. If we set w=0, then the point X in real space become infinitely larger. Hence, the points in homogeneous space forms a plane at infinity when w=0. The figure 56 shows homogeneous space and coordinate system.

![Figure 56: Homogeneous space and coordinate system [61, 1]](image)

In homogeneous coordinates, a point in projective plane corresponds to a line through the unified origin. And a family of parallel lines has a correspondence among them as they form a point at infinity or horizon.

### 5.8 Lens distortions modeling

The ideal structure of the camera geometry is discussed in the previous sections. However, because of manufacturing constraints, the camera lenses are not mathematically ideal. It means lens do not project light ideally as it should be. The lens of the camera introduces distortions that correspondingly affect the mapping of real world coordinates into camera coordinates. The most two prominent distortions are radial distortion and tangential distortion out of all distortions. The imperfect shape of the lens introduces radial distortion and nonparallel imaging chip relative to camera lens introduce tangential distortion. Fortunately, this distortion can be modelled and hence reducing its effect on the projected scene. The radial distortion can be modelled using first two term $d_1$ and $d_2$ of Taylor series expansion. However, for highly distorted lenses (e.g. fisheye) third term $d_3$ also need to be used. The distortion model using Taylor series expansion is as follows-

$$x_{corrected} = x(1 + d_1 r^2 + d_2 r^4 + d_3 r^6)$$

$$y_{corrected} = y(1 + d_1 r^2 + d_2 r^4 + d_3 r^6)$$
Here, \((x, y)\) is the naturally projected location of a point, but distorted from ideal projection. The new location \((x_{\text{corrected}}, y_{\text{corrected}})\) resulted after correcting the point using distortion model. The figure 57 shows radial distortion of simple lens.

![Radial distortion of a simple lens](image)

**Figure 57: Radial distortion of a simple lens [41]**

Moreover, the mechanical alignment of camera lens and imaging plane is needed to be perfectly parallel in order to be ideal. But this alignment is not often achieved because of manufacturing defects or cheap manufacturing process which causes tangential distortion. However, this distortion can be also be modelled using two parameter as follows:

\[
\begin{align*}
x_{\text{corrected}} &= x + [2t_1y + t_2(r^2 + 2x^2)] \\
y_{\text{corrected}} &= y + [t_1(r^2 + 2y^2) + 2t_2x]
\end{align*}
\]

Here, \((x, y)\) is the naturally projected location of a point, but deviated from ideal location. The new location \((x_{\text{corrected}}, y_{\text{corrected}})\) resulted after correcting the point using distortion model. The figure 58 shows a probable tangential distortion of a cheap camera.
Consequently, we have three distortion coefficients for radial distortion and two distortion coefficients for tangential distortion that perfectly model the radial distortion and tangential distortion. These distortion coefficients are generally bundled into a 5-by-1 matrix which is called distortion vector.

5.9 Rotation and Translation
Rotation and Translation is a fundamental mathematical model to understand the basic principles of imaging. Hence, it is very important to understand the mathematical structure of panorama image stitching. The figure 59 shows an intuitive depiction about general rotation and translation.
We can perfectly model the pose of the real world 3D object relative to the 2D image plane of camera coordinate system using rotation and translation. The term rotation interprets in this case as the relocation of a point in different coordinate system. The two dimensional rotation around each x, y and z axis with constant pivot axis measurement can composed into a three dimensional rotation. Hence, rotation around x, y and z axis with angle $\alpha$, $\beta$, $\gamma$ respectively generates a final rotation matrix $R$ that is the product of three matrices $R_x(\alpha)$, $R_y(\beta)$, $R_z(\gamma)$ which is $R = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma)$. Here, $R_x(\alpha)$, $R_y(\beta)$, $R_z(\gamma)$ holds the following geometrical properties. Each column vector of the following matrices represents a plane; hence it becomes very clear how we have transformed a 2D property into 3D by padding one more plane that provides more intuitive and convenient computation.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix}, \quad R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix}, \quad R_z(\gamma) = \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 59: Rotation and Translation [1]
In the figure 60, we can see that rotating a point by an angle \( \theta \) is equivalent as counter-rotating the axis of the co-ordinate system by an angle \( \theta \). The above rotation can be described in a functional form as follows:

\[
\begin{align*}
    x' &= x\cos \theta + y\sin \theta \\
    y' &= -x\sin \theta + y\cos \theta
\end{align*}
\]

We can write down the equation in matrix form as follows:

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & \sin \theta \\
    -\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
    x \\
    y
\end{bmatrix}
\]

Moreover, we can represent 2D vector as 3D vector using homogeneous coordinates to facilitate translation as matrix multiplication as follows:

\[
\begin{bmatrix}
    \cos \theta & \sin \theta & 0 \\
    -\sin \theta & \cos \theta & 0 \\
    0 & 0 & 1
\end{bmatrix}
\]

A rotation can be associated with a translation and similarly a translation can be associated with a rotation. Translation is the representation of the shift in origin from one coordinate system to another coordinate system that has different origin. In two different coordinate systems with different origin, translation defines the offset from the origin of one coordinate system to the origin of another coordinate system. Hence, the translation of a real
world object’s origin from the real world coordinate system to the camera coordinate system’s origin holds the following relation:

\[ T = \text{origin}_{\text{object}} - \text{origin}_{\text{camera}} \]

Subsequently, the coordinate of a point \( W \) from the real world coordinate system has coordinate \( C \) in the camera coordinate system as follows:

\[ C = R (W - T) \]

The real world point \( W \) is transformed into camera point \( C \) in which \( W \) is translated by \( T \) and Rotated by \( R \).

### 5.10 Epipolar Geometry

We have already discussed the effect of different centre of projection on the perspective view of a real world object. The knowledge of Epipolar geometry is important to find out correspondence among different perspective view of a single real world object in a multi-camera or multi-view setup. The basic building blocks of this geometrical setup are Epipolar point, Epipolar line and Epipolar plane. The definition of these blocks is important to deeply understand this geometrical arrangement. In the following figure, it can be clearly seen that how different centre of projection changes the perspective projection of the same real world scene into a camera image. The figure 61 shows the effect of different centre of projection in imaging.

Figure 61: Effect of different centre of projection [1]
In the figure 62, we have two simple pinhole camera models positioned left side and right side with centre of projection $O_L$ and $O_R$ respectively. The left camera has mapped real world point $X$ into image plane point $X_L$ and right camera has mapped the same point $X$ into image plane point $X_R$. It means both left and right cameras have mapped the same real world point $X$ into a point in their corresponding image plane. Hence, the point $X$ will have different perspective view $X_L$ and $X_R$ in two different cameras.

The projection left camera’s centre of projection $O_L$ into the image plane of right camera’s gives a point $e_R$. Symmetrically, the projection of right camera’s centre of projection $O_R$ into the image plane of left camera’s gives a point $e_L$. The distinct point $e_L$ and $e_R$ are called epipole or epipolar points. Moreover, the line $O_L X$ is considered as a point for the left camera while the same line is considered as a line $X_R e_R$ for the right camera. This $X_R e_R$ is called the epipolar line. Symmetrically, the line $O_R X$ is considered as a point for the right camera while it is considered as a line $X_L e_L$ for the left camera. In the latter case, the epipolar line is $X_L e_L$. The two epipolar lines we have discussed so far is the result a single real world point $X$. However, generally any real world object contains more 3D points. And, we have a corresponding epipolar line for each point of the real world object. In other word, we can say that we have a set of epipolar lines in each image. Now, the epipolar line $X_R e_R$ must pass through the epipolar point $e_R$ as the corresponding line $O_L X$ passes through the projection centre $O_L$. Moreover, the whole set of epipolar lines of the right image must need to pass through the epipolar point $e_R$. Symmetrically, all the epipolar lines of left image must need to pass the epipolar point of the left image. Conversely, it can be said that, all the lines that passes through the epipolar point is the
epipolar line since the lines can be derived from the corresponding real world object point. The area that is spanned by the points $O_L$, $O_R$ and $X$ forms a plane which is called epipolar plane. Each epipolar plane intersects the image plane at the same position where the plane forms corresponding epipolar line and epipolar line and plane both passes through the epipolar point.

![Image](image.png)

Figure 63: Epipole and epipolar lines [62, 1]

In the figure 63, we can clearly see how the total set of epipolar lines is passing through the corresponding epipolar point.
5.11 Practical imaging practices

5.11.1 Exposure time
The exposure time is important because it strongly influence image luminosity. A short and single shutter cycle constitutes to an exposure. A lengthy shutter cycle constitutes to long exposure. A series of short shutter cycle constitutes to multiple exposure.

Figure 64: Exposure time and Luminosity [1]
In the figure 64, we can see the relations between key exposure factors and its effect on image. The relation between exposure and its affect in imaging can be expressed according to following equation.

\[ H_v = E_v \cdot t \]
Where, $H_v$ is the luminous exposure (lux seconds), $E_v$ is the image plane illuminance (lux) and $t$ is the exposure time (second). Hence, $H_v$ is the product of image plane illuminance $E_v$ and exposure time. In imaging, $H_v$ is the accumulated quantity of visible light that falls upon the image sensor or image plane during a given exposure time. So, we can clearly see the relationship between the exposure time and image luminance. Exposure time is an important factor in imaging because it determines what amount of converged light will be let to fall upon image plane. But how exposure time going to be related with our thesis? We are going to use three cameras and if they have different exposure time then the resultant images will have different image luminosity. Subsequently, we have to compensate for exposure time differences in our image stitching algorithm. But as there is no exposure time difference among the cameras, we have not compensated for exposure and hence stitching algorithm runs faster.

5.12 Imaging technique is 2D to 2D conversion

In the previous sections, we have mentioned that, we convert a 3D real world point into 2D image plane point. But is it the way we actually do the imaging? The answer is not. Real world 3D points do not jump into the image plane via camera lens to become a 2D image point. Is a real world point a 3D point? A real world point is a 3D point in a sense that we mathematically understand or locate a real world point by the means of X, Y, Z axis of Euclidean and homogeneous coordinate. It is not necessarily a constraint that we always need to locate a point by the means three dimension. But we still use 3D X, Y, Z axis because it is the established and logically perceived mathematical model until now. Having said that, I meant, we can say that a real world point is a 1D or 7D point if and only if we can mathematically perceive that and logically prove that.

Let us consider that a real 3D world point is a photon or light particle. So, actually a beam of photon falls into the image sensor or image plane. The sensor registers the average color of the light by sampling process. So, in terms of human developed imaging techniques, it’s a 2D to 2D conversion. Then why we need to model it in the literature as 3D to 2D conversion process? Then again, we actually get the photon from the real world and we cannot locate a point in real world without 3D mathematical model. Hence, we cannot explain the exact physics of the imaging process without mentioning 3D to 2D conversion.
Chapter 6

Image Stitching

Image stitching is a method of generating large resolution image by stitching several smaller images. The relevant literatures of this area mention stitched image as mosaic. In real world, we use smaller pieces of building blocks to build a larger art. Similarly, several smaller images are used as a building block to build a composite image in image stitching. It appears that, this is the reason why stitched larger image is termed as mosaic. From a mathematical point of view, image stitching algorithm is a combination of mathematical concepts that process several image matrices to generate a large image matrix. Digital images are nothing but a pure matrix with different intensity values. However, images represent different kind real-life object or information by displaying different intensity values in each pixel point. These objects in the image can be detected, extracted to use for object identification, object verification, image matching, object matching, target detection or tracking an object in a sequence of images. The algorithm that generates a panoramic image is a combination of Feature matching (e.g. Invariant SIFT, SURF Features), Homography Estimation, Warping, Compositing and Blending methods.

6.1 Image Features and Feature Matching

Every digital image is full of real-life object. These objects can vary from human, animal, natural objects like water, trees to human made artificial objects like house, box etc. In another term, we can name this objects as feature. More than one digital image can share same feature or object properties. Sometimes, matching this features become necessary to solve some problems. There are various categories of features in a digital image. It can be edge, points, lines, active contours or region segmentation. These features extract information about the represented objects in the image. The common objects or overlapping region can be used to do some useful processing on these images. The matching of features using SURF from two images is clearly depicted in the figure 65.
6.1.1 Image features and reason of using feature matching

The philosophy of using feature matching to establish correspondence between images is that a set of images has no self-consciousness to automatically project a larger scene from multiple images. The human brain can automatically scan a larger scene using several gaze one after another and then can have an automatically projected brain consciousness about the whole scenario. However, we often need to project images with larger field of view of scenarios using synthetic data processors like digital computers. So, to project a larger image of the scenario, we select some salient feature from the overlapping area of each image. And, based on the similarities between the corresponding feature pair sets, we establish correct local and global relationship among the image mosaics of a particular scene. There are some common objects in the following images of figure 66.
There are two important approaches to find feature point and their correspondences. Firstly, features can be tracked using correlation or least squares after finding feature from a single image. Secondly, feature points can be detected separately from the set of available images and then image features can be matched by using their local appearance. The above approaches falls within the category of area based feature matching and feature based feature matching as discussed in the literature review section of this paper. The following patches in the figure 67 are from two different images that can be easily matched.

Edges are an important feature of the image. When a sharp change of color intensity occurs in the image then the edge appears. The information obtained from an edge can be used to determine object boundary or to identify changes in image orientation. The following image in figure 68 depicts the edges of one image used in Figure 66 using canny Edge detector. The boundary of the box is clearly identifiable from its sharp vertical edge.
Line is an important and common aspect in the knowledge of human about geometry. Apparently, geometrical shape rectangle, triangle, trapezium, square all are combination of straight lines. Accordingly, lines appear in all types of innovative product design as well as spatial designs. Thus a line represents an important property of the digital image. However, sometimes lines do not appear linearly in an image. In an image, lines can appear as a straight line, as a curve or can be piecewise linear. There have been some algorithms [27] developed to extracts this type of curve or piecewise linear poly-lines. Moreover, real-world disconnected components or collinear line segments can be grouped as extended lines using Hough transform (Hough 1962). This information can be used to find out vanishing points in the image. The “vanishing point” is a useful feature to calibrate camera or getting its orientation regarding to the scene. The geometrical depiction of vanishing point is clearly visible in the figure 69.

Moreover, region segmentation is also an important feature. In the image, it is a common phenomenon that a group of pixels appears with same intensity level. All these pixels with same
intensity can be grouped as a separate region in the image. The extraction of region segment can be used to find the boundary of an object or different parts of the object which is depicted in the figure 70.

Figure 70: Region segmentation [1]

Furthermore, contour is also an important feature representation that can be used in feature matching. Contour line connects points of an image with equal elevation and thus represents objects with varying elevation. The elevation modeling of objects using contour line depicted in the figure 71.

Figure 71: Image contour [1]
6.1.2 Common features and Overlapping area

We know that every image has numerous features but where to look for common features between two images or among a large set of images? Indeed, the common features will lie in the overlapping region of the images. Digital images with any geometrical shape can have overlapping region between them. However, we commonly represent digital image using geometrical rectangular shape.

![Image of overlapping areas between rectangles]

Figure 72: Overlapping between rectangles [1]

In the figure 72, we can see the overlapping areas between the rectangles. In a similar way, we can have overlapping areas among the images and we extract features from these areas. As we extract same feature from different images, we can establish geometrical and spatial relation among the images using mapping. The discussion of overlapping areas in terms of camera FOV is further discussed in the “Proposed Stitching Algorithm” chapter.

6.2 Speeded Up Robust Features (SURF) and SIFT

Scale Invariant Feature Transform (SIFT) [7] provides invariant feature from the images. However, SURF [63] is another feature descriptor which provides all the benefits of SIFT with a much faster response time. Real world images often have different rotation, scaling and luminance. That is why it is necessary to have some properties of the features that do not change with these conditions. The properties of all features do not show the same behavior in different conditions. Some of them changes with different rotation, scaling and luminance. The common feature matching techniques like correlation of image patches around Harris corners or ordinary translational correlation changes with scaling and rotation respectively. However, there are some properties that are invariant with respect to rotation, scaling and luminance. The method we have used for extracting these invariant features is called Speeded Up Robust Features (SURF). The
SURF descriptor performs more robustly than SIFT descriptor against different image disturbances because of global integration. The figure 73 shows locally operated SIFT descriptor and globally integrated SURF descriptor.

![Figure 73: A SIFT key point descriptor computed from Gradient Magnitude and Orientation and SURF sums][1]

The SURF method consists of a detector and descriptor. The detector and descriptor is kept fast and simplified without loss of performance by maintaining a balanced point between speed and performance in comparison to the state-of-the-art. The detection scheme is simplified but kept accurate and descriptor’s size is reduced but kept distinctive enough. Firstly, the interest points are detected using a detector at distinctive image locations such as blobs, corners, T-junctions in the image. The reliability of the detector for detecting the same physical interest points under different viewing conditions is ensured because the detector has an important property of repeatability. Secondly, the neighborhood of each interest point is described by a feature vector which is called a descriptor. The descriptor needs to be robust to noise, geometric and photometric deformations and detection displacements while at the same time need to be distinctive. Thirdly, the descriptors are matched between two images based on distance (e.g. Euclidean distance) between two vectors. The descriptor can be higher dimensional and lower dimensional. The higher dimensional descriptors are highly distinctive but at the same time it takes longer time for interest point matching. On the other hand, the lower dimensional

---

[1]: http://example.com/figure73.png
descriptors are less distinctive but take lesser time for interest point matching than higher dimensional descriptors.

### 6.3 Planar Homography or Collineation or Projective Transformation Estimation

Imaging is the transformation of a real world scene into a camera image plane scene through the projective transformation. In other words, we are transforming a real world scene from one plane to another through imaging. The projective transformation from one plane to another is called planar homography. According to the definition, the mapping of a 3D real world point into 2D camera image plane point is also a planar homography. The mapping of points from one plane to another can be easily done by matrix multiplication as we have used homogeneous projective coordinate for both plane. For example, if we transform a 3D point \( A \) from one plane to another 2D plane point \( b \) then the following relationship holds.

\[
b = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, \quad A = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}
\]

\[
b = HA
\]

Here, \( H \) represents the homography matrix, \( A \) represents the point that is going to be transformed, \( b \) represents the point that is obtained after transformation. The homography matrix \( H \) has two important intuitive parts namely physical transformation part and projection part. These two parts by itself gives an intuitive idea how the transformation is done. The physical transformation part is the sum of rotation matrix \( R \) and translation vector \( t \) which describes the relation between real world plane and image plane or between two distinct planes. In other words, physical part describes how much rotation \( R \) and translation \( t \) is needed to transform a 3D point \( A \) to the image plane point \( b \). The projection part describes the camera intrinsic matrix. The rotation matrix \( R \) and translation vector \( t \) can be placed into a single matrix as an advantage of using homogeneous coordinate. Hence,

\[
C = [R \ t]
\]

The matrix \( C \) is representing physical transformation part of the homography matrix \( H \). We have discussed earlier that \( R \) is a 3-by-3 rotation matrix and \( t \) is a 3-by-1 translation vector which in combined gives a 3-by-4 matrix \( C \). The camera intrinsic matrix represents the projection part of the homography matrix \( H \). If we consider camera intrinsic matrix as \( I \) then following relationship holds.
We have mentioned earlier that A is a 3D point. But in practice we are interested about a point A which is considered as a point on a particular 2D plane, not scattered on all real world space. Subsequently, point A turns out to be 2D point as Z is considered equal to 0. We have mentioned earlier that each column of the rotation matrix represents a plane. As Z is equal to 0, the corresponding column vector of the rotation matrix is not needed. Hence, 3-by-3 rotation matrix \( R = [r_1 \ r_2 \ r_3] \) turns out into a 2-by-2 matrix \( R = [r_1 \ r_2] \). Subsequently, the matrix C turns out into a 3-by-3 matrix from 3-by-4 matrix. The above simplification process is further elaborated in the following discussion.

![Diagram](image)

Figure 74: Projection of points on planar surface [64]

In the figure 74, the relation between real-world object plane coordinate and camera coordinate, 2D planar surface and 2D image plane intuitively depicted. The reason for considering a 3D real-world point as 2D point on a plane is also obvious now. Moreover, we can also pictorially see that how the orientation of a real world object plane can differ from the camera coordinates and can be coincided by rotation and translation matrix.
The derivation process of 3 $\times$ 3 homography matrix through simplification is self-explanatory and we can see the effect of considering a real-world point on a 2D planar surface. The projective transformation can occur in 2D or 3D Euclidian space. In the figure 75, 2D transformations are classified in different categories according to their characteristics.
The characteristic of each category of transformation given in the figure 76 is also reflected in their associated transformation matrix. The homography matrix of projective transformation has 8 degrees of freedom from which 4 degrees belong to rotation matrix, 2 degrees belong to translation vector, and 2 degrees belong to perspective vector. The affine transform, similarity and Euclidean transform has no perspective change and hence we can see two zeros in the third row of their corresponding transformation matrices. Moreover, we can also see that what properties of a geometrical entity stay unchanged after the mentioned transformations. However, when we consider two images from two cameras then the homography describes the relationship between the camera image planes or images which can be seen from the figure 77.
In image stitching, after getting the feature correspondence between the images, the most important task is to find geometrical perspective changes in the feature pixels with different camera projection centre. This is often termed as “Motion Modeling”. Motion modeling establishes mathematical relationship that maps pixel from one image to another. However, this mathematical relation has been established in a form of Homography matrix. Now the important observation is that, as we have corresponding feature from two images, we can calculate homography matrix without knowing about the camera intrinsic matrix. To mention, we actually calculate the unknown camera intrinsic parameters (e.g. focal length) via calculating multiple homographies from multiple views of the camera.

For the sake of understanding, we can compare planar transformation with an electronics system. We know that, an electronic system’s internal property response (transfer function) to the system input produces the output. In a similar way, if we consider $x$ as a input vector from
one plane, and $H$ is the homography matrix that works like a transfer function then $x'$ is the output vector into another plane.

$$x' = Hx$$

Figure 78: Pixel mapping between two image planes [1]

In the figure 78, $H$ represents homography matrix. Here, a point or pixel $x$ from the projected line or ray $c$ has transformed to a new point or pixel $x'$ at the projected line or ray $c'$. Now, if the planar surface is perpendicular to the optical or principal axis ($z$ axis) which means if the plane is frontal then we will have simpler rotation matrix. Subsequently it will also affect homography matrix. The rotation and translation belongs to the physical transformation part of the homography matrix we have mentioned earlier. For every rotation in a 2D plane, rotation matrix has following properties. The rotation angle $\theta$ (theta) is positive if direction of vector rotation is counterclockwise. Hence,

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

The rotation angle $\theta$ (theta) is negative if direction of vector rotation is clockwise. Hence,

$$R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

The counterclockwise rotation can be described in a functional form as follows-

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

The above equations can be written in a matrix form as follows-

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
The 2D column vectors of the above matrix can be written in as 3D vector using homogeneous coordinates to simplify the calculation as follows-

\[
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

So, the homography matrix for a frontal planar surface can be derived as in figure 79.

Figure 79: Homography matrix for Frontal plane
6.3.1 Homography Decomposition

We have understood until now that homography is planar projective transformation. But in terms of physical imaging process, homography is the composition of other groups of transformation and hence it can be decomposed into those transformations.

Figure 80: Homography Formation [64]

In the figure 80, $M_{\text{ext}}$ is the extrinsic matrix, $M_{\text{proj}}$ is the projective matrix, $M_{\text{aff}}$ is the affine matrix, $M_{\text{int}}$ is the intrinsic matrix, and $M$ is the homography matrix. Moreover, the kind of transformation that happens with each subsystem of the total physical process of imaging and generates these respective matrices is also clearly visible. Hence, the following relationship holds:

$$M_{\text{intrinsic}} = M_{\text{projective}} M_{\text{affine}}$$

$$M_{\text{homography}} = M_{\text{intrinsic}} M_{\text{extrinsic}}$$
6.4 Random Sample Consensus (RANSAC) and Homography

We have discussed about the feature matching in the previous section which provides correspondence between two images or among the set of images. This correspondence is established by a form of homography matrix. The RANSAC [65, 66, 67] can be used for determining homography matrix from matched features (consisting inliers and outliers) with minimum number of trials. RANSAC is an algorithm or mathematical paradigm to calculate inliers that establishes correct correspondence between any types of two models. The linguistic meaning of Random Sample Consensus (RANSAC) itself gives an insight on its operation which means the consensus or agreement among randomly chosen samples. In terms of data identity, two types of data definition are important in RANSAC. These two data definition are inliers and outliers. The basic RANSAC algorithm [65, 66, 67] follows the following execution sequence and depicted in figure 81.

1. We have two models with two data sets or object sets
2. Randomly select minimum number of data points or objects from both model required to calculate model parameters
3. Calculate the parameters of the model
4. Calculate How many data points or objects from the all data points or objects (fetched in step one with one round) fit with a predefined tolerance $t$. The data points or objects that fits within given tolerance, consider them as inlier otherwise outlier.
5. If the fraction of the number of inliers or objects over the total number of data points or objects greater than a predefined threshold $I_t$ (inlier threshold), re-compute parameters of the model and terminate. Otherwise repeat steps 2 to 5.
Figure 81: RANSAC algorithm for model fitting
In our case, these two sets of data’s are extracted robust and invariant features from two corresponding images. The figure 82 shows feature matching and then homography computation using RANSAC. We have extracted robust features from the overlapping area between two
images and then choose the best feature match from two nearest neighbors for either image. There are false positives in the automatic feature detection process even though they fit within threshold. Hence, we construct point-point correspondence for all features and then randomly choose minimally required set of key points and compute homography. After applying the homography, we choose the inliers that fit within a customizable SSD threshold and reject others as outlier. Now, if the number of inliers fit within a threshold then we again construct point-point correspondence for inliers only and re-compute homography on inliers otherwise we re-run the RANSAC process.

Now the question is why they are outliers even if they are matched? The reason is that in an image only geometrically consistent match is proper match. We might have matched features but if they are not geometrically consistent relative to each other and hence considering them as inlier will produce geometrically wrong result (i.e. skewed transformation matrix) in further calculation.

### 6.5 Calculation of Homography or Collineation matrix and DLT

The elaboration of DLT is Direct Linear Transform. We linearly transform pixels from one plane to another plane using direct linear mapping. The transformation of a pixel between two camera image planes in 3D can be written as follows-

\[ D_1 = HS_1, \text{ where } S_1 \in R^3 \]

As we are using homogeneous coordinates in image planes the following relation holds.

\[ \lambda_f d_1 = D_1 \]

\[ \lambda_f s_1 = S_1 \]

The product of 2D value \( r_1 \) and \( s_1 \) with factor \( \lambda_f \) gives corresponding 3D value.

\[ \lambda_f d_1 = H \lambda_f s_1 \]

We can see that, \( r_1 \) is equal to \( Hs_1 \) up to a scale factor.

\[ D_1 \sim Hs_1 \]

From the above relations, we can also see that, there exists a \textit{direct linear mapping or direct linear transformation relation} between point’s lies on image planes. We are dividing the discussion about the homography computation in two sections. First, we will discuss about the
mathematical process of finding out Homography matrix. Second, we will discuss about the computation of Homography matrix using OpenCV.

\[
\begin{bmatrix}
D_{x_1} \\
D_{y_1} \\
1
\end{bmatrix} =
\begin{bmatrix}
H_{11} & H_{12} & H_{13} \\
H_{21} & H_{22} & H_{23} \\
H_{31} & H_{32} & H_{33}
\end{bmatrix}
\begin{bmatrix}
S_{x_1} \\
S_{y_1} \\
1
\end{bmatrix}
\]

From the above equation we can see that each point pair provide two independent equations:

\[
D'_{x_1} = \frac{D_x}{1} = \frac{H_{11}S_{x_1} + H_{12}S_{y_1} + H_{13}}{H_{31}S_{x_1} + H_{32}S_{y_1} + H_{33}}
\]

\[
D'_{y_1} = \frac{D_y}{1} = \frac{H_{21}S_{x_1} + H_{22}S_{y_1} + H_{23}}{H_{31}S_{x_1} + H_{32}S_{y_1} + H_{33}}
\]

After rearranging and enforcing \( H_{33} = 1 \), we get,

\[
D'_{x_1} (H_{31}S_{x_1} + H_{32}S_{y_1} + 1) = H_{11}S_{x_1} + H_{12}S_{y_1} + H_{13}
\]

\[
H_{31}S_{x_1}D'_{x_1} + H_{32}S_{y_1}D'_{x_1} + D_{x_1} = H_{11}S_{x_1} + H_{12}S_{y_1} + H_{13}
\]

\[
H_{11}S_{x_1} + H_{12}S_{y_1} + H_{13} - H_{31}S_{x_1}D'_{x_1} - H_{32}S_{y_1}D'_{x_1} = D'_{x_1}
\]

\[
D'_{y_1} (H_{31}S_{x_1} + H_{32}S_{y_1} + 1) = H_{21}S_{x_1} + H_{22}S_{y_1} + H_{13}
\]

\[
H_{31}S_{x_1}D'_{y_1} + H_{32}S_{y_1}D'_{y_1} + D_{y_1} = H_{21}S_{x_1} + H_{22}S_{y_1} + H_{13}
\]

\[
H_{21}S_{x_1} + H_{22}S_{y_1} + H_{13} - H_{31}S_{x_1}D'_{y_1} - H_{32}S_{y_1}D'_{y_1} = D'_{y_1}
\]

We need at least 4 pairs of matched inlier points to compute the 8 unknown free parameters of homography matrix [68, 69, 70] which is achieved via solving the following system.
In the above equation \((S_{x_1}, S_{y_1}) (D'_{x_1}, D'_{y_1})\) is the coordinate of the first point from two images or from the source image and destination image or from the sensed image and reference image respectively. We have three more matched inlier point pairs in the above equation. The more pairs of matched inlier points will be used the more optimal homography matrix will be derived in terms of least-square error. The equation can be solved as follows:

\[
Ah = b
\]

The least-square solution is \(h\) which satisfies,

\[
A^T A h = A^T b
\]

\[
h = (A^T A)^{-1} A^T b
\]

We will get the following homography matrix by solving the above least-square equation using Singular Value Decomposition (SVD).
We have discussed earlier about the physical transformation part and projection part of the homography matrix. The contribution of each part of the homography matrix is depicted in the figure 83 with more details.

![Homography matrix anatomy](image)

**Figure 83: Homography matrix anatomy**

The perspective vector gives the non-linear perspective effect in an image. This computed homography matrix is then used to warp or resample or projectively transform the pixels of the sensed image to the reference image.

The following OpenCV function is used to calculate Homography.

```cpp
findHomography(InputArray srcPoints, InputArray dstPoints, int method=0, double ransacReprojThreshold=3, OutputArray mask=noArray())
{
    ...
}
```

The parameter “srcPoints” and “dstPoints” are the coordinates of the points in original and target plane respectively. Moreover, “method” parameter indicates the type of method that need to be used to calculate homography. The value “0” for “method” parameter computes homography using a regular method that uses all the points. The value “CV_RANSAC” and “CV_LMEDS” for “method” parameter computes homography using RANSAC and Least-Median based robust methods. The “ransacReprojThreshold” parameter indicates the maximum allowed reprojection error threshold to consider a key-point pair as inlier and only used with RANSAC method. A key-point pair $i$ that produces reprojection error above this threshold considered as outlier as follows-
The value of this parameter can usually be set to 1 to 10 if srcPoints and dstPoints are measured in pixels.

### 6.5.1 Singular Value Decomposition (SVD) and Homography computation

The homography matrix provides perspective transformation between the points of source and destination image plane in such way that the back-projection error is minimized as follows-

$$
\sum_i \left( D'_{xi} - \frac{H_{11}S_{x_i} + H_{12}S_{y_i} + H_{13}}{H_{31}S_{x_i} + H_{32}S_{y_i} + H_{33}} \right)^2 + \left( D'_{yi} - \frac{H_{21}S_{x_i} + H_{22}S_{y_i} + H_{23}}{H_{31}S_{x_i} + H_{32}S_{y_i} + H_{33}} \right)^2
$$

The reason for using SVD to calculate homography matrix can only be understood if we can understand geometrical interpretation of SVD. SVD decomposes a matrix as follows-

$$
A = USV^T
$$

Here, $V$ moves the original basis from old one to new one, $S$ does the scaling by stretching along new basis, and $U$ describes the final result with respect to original data. So, we can see that $U$ and $V$ only performs rotation and $S$ performs scaling. Hence, SVD actually does the computation on the data with geometrical abstraction.

### 6.6 Warping or Projective mapping or Resampling

Warping is the process of mapping one pixel point from one plane to another plane. If we have three images then we can say we have three images with three planes. The pixel values of each image belong to a plane and when we map pixel points of sensed images into another plane of reference image then we call this process as warping or projective mapping or resampling. In short, we choose an image or image plane as the reference plane and then we project or map the pixels of all sensed images into reference plane or reference image. But what is the use warping in image stitching? We know that we have overlapping areas between images and it means we have same set of pixels in the overlapping area from two different images with their respective perspectivity. But are we going to keep the both sets of common pixels in the final panorama? The answer is no as we are not going to keep repetitive features or pixels in the final panorama. So, what we do with one additional set of overlapping common pixels? We take the average of overlapping pixels to compute the final pixel value for each pixel point.

But do we calculate transformation matrix using all pixels of the overlapping area? The answer no because we only choose geometrically consistent and robust enough matched features
that falls under the threshold out of all available features. We calculate homography matrix from these corresponding inlier feature pairs and then map all pixels of sensed image into reference plane or reference image using this homography matrix. In other words, we join a set of images with geometric consistency to achieve a larger scene mosaic which is achieved by the means of warping all sensed images into reference image or reference plane.

![Figure 84: Projective transformation or Collineation or Homography [53]](image)

In the figure 84, we have warped three collinear pixel points from a rectangle to collinear points in quadrilateral. According to mathematical norm, we have linearly mapped a collinear set of points into other collinear set of points in homogeneous coordinates. But what type of benefit we are getting by transforming a collinear set of points from a rectangle to another collinear set of points in quadrilaterals and what is its use in warping or projective mapping? The answer is by mapping from one geometrical shape to another geometrical shape we can warp the matched pixels of the sensed image into corresponding pixels of reference image with right orientation. We understand the geometrical position of an image pixel by its native image plane. It means each image from a set of images have their own native plane. Hence, matched pair of image feature has different orientations in terms of their native image plane. If so, how computer can warp a pixel from sensed image which has its own native orientation to a corresponding pixel in the reference image and its orientation? We humans can understand this by built-in brain capacity but how computer can computationally achieve that? The answer is computer computationally achieve that by using several mathematical methods based on projective geometry theorems namely Pappus theorem, Desargues theorem, quadrangle and quadrilaterals.
In the figure 85, we have warped the right image on the left image. We can clearly see that by doing a geometrically consistent warping we have actually transformed rectangular image into a quadrilateral. We can clearly see that use of immediate previous figure in practical image warping.

6.7 Compositing
After determining the motion model of the camera, the image mosaics needed be wrapped and composited into a flat or cylindrical or spherical surface. The flat composition generally referred as reference composition because of using one image mosaic as a reference plane. Sometimes, Alpha blending is used for better graphical representation where the value (0.0 to 1) of the alpha (α) is used to control the translucency of the background and foreground color to produce a new blended color.
6.7.1 Blending

Proper modeling of the camera most often reduces the necessity of Blending. But there are some unmodelled effects like vignetting, parallax effect etc. which causes distortion and keeps visible seam in the finally composited image mosaic. Blending in the final panorama requires a weighted average of all pixel values from the all image mosaics. For each image the weight function is,

\[ W(x, y) = w(x)w(y) \]

Here, \( w(x) \) ranges from 0 (at the edge) to 1 (at the image centre). In a spherical coordinate, linear blending can be done using following weight function which performs a weighted sum of the image pixel values along each ray.

\[ I_{linear}(\theta, \phi) = \frac{\sum_{i=1}^{n} I^i(\theta, \phi)W^i(\theta, \phi)}{\sum_{i=1}^{n} W^i(\theta, \phi)} \]

Here, \( I_{linear}(\theta, \phi) \) is the final composited image. To avoid the blurring of high frequency detail due to mis-registration multi-band blending is required. Multi-band blending blends high frequencies in short range whereas low frequencies in large spatial range. Multi-band blending weight is measured by getting the set of points for which an image mostly responsible.

\[ W_{max}^{i}(\theta, \phi) = \begin{cases} 1 & \text{if } W^{i}(\theta, \phi) = \arg \max_j W^{j}(\theta, \phi) \\ 0 & \text{Otherwise} \end{cases} \]

Here, \( W_{max}^{i}(\theta, \phi) \) has a maximum value of 1 where a single image \( i \) has the maximum weight to the values \( \theta, \phi \) and 0 where maximum weight is contributed by other image mosaics.
Chapter 7

Methods

7.1 Requirement Engineering
The goal of this thesis project was to develop a real-time panoramic imaging system that will generate real-time panorama. To achieve this goal, we have defined the requirements of the system first. The requirements can be categorized as functional requirements and non-functional requirements. The functional requirements are directly related to the desired functions of the system. The types of functional components are needed to have a working prototype is defined by the functional requirements. On the other hand, system’s overall quality related issues are defined by non-functional requirements.

7.1.1 Functional requirements
The functional requirement of the system in terms of software and hardware can be categorized as follows-

- **Functional requirements:**
  - **Hardware:**
    1. Gumstix COM
    2. PandaBoard ES
    3. Laptop Computer
    4. MicroSD and SD Card
    5. LCD Display
    6. Power Adaptor
    7. Peripheral Connector
    8. Serial Communication Cable
    9. Spherical Camera System
    10. A Chimney Test Bed
Software:
1. Operating System
2. Real-Time Kernel Patch
3. Image Stitching Algorithm
4. Build Automation Tools
5. Relevant API’s
6. Software Development Tools
7. Graphical User Interface (GUI)
8. A Standalone Software with Simple Installer
9. Programming Languages for development

The each category of software functional requirement needs to be elaborated to show the necessary components.

- **Software -> Operating System**
  - Ångstrom Minimal Linux
  - Ubuntu Desktop
  - Windows 7

- **Software -> Real-Time Kernel Patch**
  - RTLinux

- **Software -> Image Stitching Algorithm**
  - Rotational Mosaics
  - SURF

- **Software -> Build Automation Tool**
  - Cmake
  - Make

- **Software -> Relevant API’s**
  - OpenCV
  - FFMPEG
- Gstreamer
- V4L2
- x264
- Qt Library

- **Software -> Software Development Tools**
  - GNU GCC Compiler (gcc, g++)
  - Nano Editor
  - Vi Editor
  - Pkg-config
  - GDB
  - Visual Studio
  - Qt Creator
  - C-Kermit
  - Minicom

- **Software -> Graphical User Interface(GUI)**
  - Real-Time process statistics
  - Response Time Display
  - Data Visualization

- **Software -> A Standalone software with simple installer**
  - Dependency Walker
  - NSIS

- **Software -> Programming Languages for Development**
  - C
  - C++
  - Linux Shell

- **Software -> Image Stitching Algorithm**
  - Theories
  - Mathematical Methods
Camera geometry
Panoramic Image Stitching Process
Image registration method

Software -> Image Stitching Algorithm -> Theories
Vision and Electromagnetism

Software -> Image Stitching Algorithm -> Mathematical Methods
Euclidean space
Projective Geometry
Homogeneous space and coordinate
Rotation and Translation
Transformation groups
Singular Value Decomposition (SVD)

Software -> Image Stitching Algorithm -> Camera geometry
Pinhole camera model
Focal length
2D image plane
Camera intrinsic and extrinsic parameter
Lens distortion modelling

Software -> Image Stitching Algorithm -> Panoramic image stitching process
Motion modeling or Homography calculation: Using image registration
Real-time stitching: Warping, Seam Finding, Compositing, Blending

Software -> Image Stitching Algorithm -> Image registration method
Photogrammetry
Overlapping and feature correspondence
Feature extraction and Matching
7.1.2 Non-functional requirements
The non-functional requirements of the system are as follows-

1. Real-time panorama
2. Handheld spherical camera system
3. Mobile

7.2 System Overview
A group of components can be defined as a system when they interact with each in an independent or semi-independent fashion for a common set of goal. The panorama imaging system has several hardware and software components. All components interact with each other to facilitate the generation of a panorama imaging system.

7.2.1 System structure
The panorama imaging system can be described as follows according to the functional characteristics of the system component. The component Spherical camera system, USB 2.0 Hub, Gumstix COM/PandaBoard ES are physically independent component. But they interact with each other via wire connection. Embedded Linux operating system is contained inside the MicroSD, SD card of the Gumstix and PandaBoard ES respectively. And, stitching algorithm, API and software dependencies are contained as a software component of the Embedded Linux operating system.

1. Spherical camera system
2. USB 2.0 Hub
3. Gumstix COM/PandaBoard ES
4. MicroSD/SD card
5. Embedded Linux operating system
6. Stitching Algorithm
7. API and Software dependencies
The figure 86 shows conceptual layers in panorama imaging system. The camera module interacts with Gumstix COM/PandaBoard ES through a USB 2.0 Hub. Gumstix/PandaBoard stitches three captured images into one panorama image. The generated panorama will be transmitted to a target destination device via wireless network for inspection. The transmission part of the system is outside of the scope of this project and has been conducted as part of another project. The layer representation of the image stitching system demonstrates the data flow that follows from output peripheral to inward. The above figure can be elaborated in terms of representation of real world light into sensed signal. The spherical camera system’s sensor senses the real world light via sampling and transforms it into a digital image. This digital image is then moves to the Gumstix/PandaBoard’s peripheral via USB Hub. The Gumstix/PandaBoard then processes this signal to generate output. In terms of physical layer representation of the signal we can say that, we represent real world color or light in terms of electrical pulses (Digital Signal), then we manipulate these pulses and then we encapsulate processed pulses using a mathematical paradigm. The mathematical paradigm is image processing mathematical methods and techniques. And, the encapsulation is done by representing processed electrical pulses as a digital image in the computer’s screen. Moreover, the representation in computer’s home screen is again is a
variation of luminance intensity. Hence, digital imaging is the representation of real world light by using manipulated electrical pulses which is actually a form of light.

Figure 87: White box model of the System's basic structure

The white box model of a system elaborately describes internal components of a system. The system can be realized in different subsystems and components of each sub system can be viewed externally. For example, the two most important subsystems of our imaging system are PandaBoard ES/Gumstix COM and Spherical camera system. In white box model of the system, we can clearly see the components of these two sub systems which are demonstrated in the figure 87.

Figure 88: Black box model of the system's basic structure

The figure 88 shows the black box model of the system’s basic structure. The black box model of the system hides many internal components of a system. The system can be realized in different subsystems and all components of each sub system cannot be viewed externally. For example, PandaBoard ES/Gumstix COM and Spherical camera system’s internal component cannot be viewed externally in the black box model of the system.
7.2.2 System Component

The figure 89 shows system components.

Figure 89: System components
7.2.3 Function description

The main goal of the system is to generate real-time panorama frame sequences from Gumstix COM/PandaBoard ES. However, the result obtained in this project will be used in another project for wireless transmission to target device. As described above, each system block and all components in each block functions interactively to achieve this goal. We can describe the system in terms of function block to provide an overall idea of the system functionality which is depicted in the figure 90.

![Diagram of functional principle blocks](image)

Figure 90: Functional principle blocks

The image capture functional block consist three side by side positioned MISUMI camera sensors and its corresponding circuit. These cameras start capturing images when video capture function is called from the Gumstix/PandaBoard and sends the captured images to Gumstix/PandaBoard for further processing. The image processing functional block consist Gumstix/PandaBoard, operating system and stitching algorithm. Here, three instantaneously captured images stitched together and forms a panorama image with approximately 180° wide view. Subsequently, this panorama image becomes ready for transmission to destination device for inspection. The image processing block and image capture block works in unison to capture and generate real-time panorama image. The figure 91 shows system component functionality with more details.
Figure 91: System component functionality
7.3 System Hardware
The imaging system is realized in both Gumstix and PandaBoard ES. Both Gumstix and PandaBoard use ARM Cortex as CPU architecture.

7.3.1 Gumstix Overo Computer on Module
Gumstix COM is a single-board minicomputer developed by Gumstix. The size of the computer is 17 mm x 58 mm x 4.2 mm (0.67 in. x 2.28 in. x 0.16 in.). The computer is named Gumstix because of its size similarity with stick of Gum. Recently, it has become extremely popular to develop different types of dedicated and distributed system based on OMAP platform. Gumstix Overo series of computers have different variants. We have used Overo® Fire COM. It has OMAP and Cortex-A8 based processor. We have used Overo Fire in our project. The figure 92 shows Gumstix COM and compares its size with a battery.

![Figure 92: Gumstix Computer-On-Module](image)

7.3.1.1 Overo Fire
The Overo fire is tiny but considerably powerful computer to perform different tasks. It has 720 MHz processor along with 512MB RAM which makes it good choice for handheld development. The figure 93 shows Overo Fire.
7.3.1.2 Palo35 Expansion board

Two types of Expansion board for Gumstix Overo COM’s are available from Gumstix. We have used Palo35 Expansion in our project because it is relatively smaller in size than Palo43. The figure 94 shows Palo35 Expansion Board.

7.3.2 PandaBoard ES

The PandaBoard is a popular development board that has been used throughout the globe for different industrial, academic and multimedia project. The PandaBoard ES is the latest version of the PandaBoard which have a dual core processor and extended features than its earlier counterpart. The figure 95 shows PandaBoard ES board.
7.3.3 Bootable MicroSD/SD Card

Gumstix Overo COM uses OpenEmbedded software architecture based Linux operating system. The boot files and root file system of the operating system is not embedded with the hardware. Instead, it uses a bootable MicroSD Card. The configured bootable MicroSD card contains two partitions. One partition contains boot files and another partition contains root file system of the operating system. The figure 96 shows a bootable MicroSD card.

![Bootable MicroSD card](image)

Figure 96: Bootable MicroSD card

7.3.4 MISUMI Camera

The following cameras are produced by Taiwanese company MISUMI Electronics Corporation that produces high quality micro camera. We have used “MO-S1555W-3S” model camera and “ucd-210M” model frame grabber. The frame grabber is powered by “UVC” driver. Moreover, the camera can capture 30 fps in “NTSC” video format and 25 fps in “PAL” video format. Furthermore, the cameras are compatible with Windows, Linux and MAC OS. The figure 97 shows the used MISUMI Spherical camera system.
7.3.5 USB 2.0 Hub
The figure 98 shows the used USB Hub 2.0.

7.4 System Software Architecture
The whole software structure of the system is contained inside the MicroSD/SD card. All of the software components work coherently and interactively to facilitate the system’s desired output. In reality, software is nothing but thousand lines of code. But to understand their functionality these components can be structured according to their different layers and function characteristics.
7.4.1 Software Structure

The entire software architecture of imaging system is an Embedded Linux system. This Embedded Linux system is used as the embedded operating system of embedded panorama imaging system. In other word, this embedded panorama imaging system is an embedded computer system that only generates real-time panorama image. Embedded computer system is a special purpose computer system that is designed to perform one or few task under real-time computing constraints.

The software architecture of an embedded system consists four parts. These are-

1. Boot loader
2. Linux Kernel
3. Standard C Library
4. Third party application and library

The boot loader is responsible for the basic system initialization and initialized by the hardware. Its loads and executes kernel processes which controls the memory management, networking and drivers for all peripherals. Third party applications and corresponding libraries use Linux kernel via Standard C library to perform designated task. And, Standard C Library works an interface between Linux kernel and third party applications. In another word, boot loader, kernel and Standard C library forms the foundation for the embedded operating system. Third party applications are being developed based on this foundation and make up a root file system together with standard C library. In our project, we have used a bootable MicroSD/SD card to boot the operating system. The bootable MicroSD/SD card consist boot loader, Linux kernel and root file system.

7.4.2 Wi-Fi Communication

Gumstix Overo Fire has built-in Wi-Fi with 802.11b/g standard wireless communication. But the Wi-Fi configuration of this smaller handheld computer is slightly different and comparatively harder than the normal laptop or desktop computers. The WLAN frequently stays down during the operating system booting process. This problem is solved by manual configuration of the WLAN. However, this configuration can be done either using terminal or changing the network interface setup of the operating system through desktop. Sometime it becomes necessary to use both, terminal and network interface setting through desktop to enable the WLAN properly. In our project, WLAN did not enable itself automatically during the system booting process. We have manually enabled the WLAN through the terminal and network setup settings of the operating system.
The PandaBoard ES is comparatively much more powerful handheld computer than Gumstix. The WLAN of the PandaBoard enables itself automatically when operating system boots up. Consequently, it was not required to configure PandaBoard WLAN manually.
Chapter 8

System Development

8.1 Development Process
The main obstacle in the development process of this type is resource selection. There is abundance of materials but none of these resources have common platform to get familiar with it. The characteristics of embedded system development do not provide a predetermined linear navigation to available resources to develop a new system. A novice developer has to choose necessary resources through understanding and experiment.

There are several ways to build an embedded Linux application. Each way has own tradeoff. Developing a Linux system from scratch is the best possible way that provides robust and secure result. But while it is the most effective but it is a highly time consuming and laborious task. On the contrary, developing an embedded Linux application using available distributions is comparatively less robust and secure. But the tradeoff is that it takes lesser development time.

In this thesis project, we have used two types of Linux distribution. In Gumstix, we have used minimal embedded Linux operating system. Though this minimal file system does not provide robust result like building Linux system from scratch but it has a minimal footprint in compare to other existing embedded Linux distributions. On the contrary, we have used both minimal and larger file system in PandaBoard.

8.2 Operating system
Ångstrom Linux distribution and Ubuntu Linux distribution is used with Gumstix and PandaBoard ES as the operating system respectively. There are different types of Linux distribution that can be used as embedded operating system. In our development, we have used Ångstrom Linux distribution which is specially developed for the small hand-held embedded devices. We have used one pre-built stable image file from the Gumstix inside the bootable MicroSD card. This stable image file consist MLO, u-boot.bin, uImage and a root file. We have partitioned a normal MicroSD card into two partitions- “dev/sdb1” and “dev/sdb2”. The “dev/sdb1” is a W95 FAT32 file system and we have copied and placed MLO, u-boot.bin and uImage inside this partition and have made this partition bootable. Moreover, we have copied and placed the root file inside the “dev/sdb2” partition.
The operating system of the PandaBoard ES can be configured as like as Gumstix. But there are simpler methods for configuring PandaBoard in compare to Gumstix. The main reason is that PandaBoard ES has substantially more powerful CPU and it is dual core. Consequently, it can support Ubuntu Linux operating system which is much heavier than Ångstrom Linux distribution in terms of size, graphics and features. Moreover, Ubuntu Linux has much richer software repository than Ångstrom Linux. Subsequently, configuring PandaBoard with different API and software dependencies was much easier in compare to Gumstix. However, the configuration process of the PandaBoard is also mentioned in the thesis.

8.3 Gumstix COM

8.3.1 Bootable MicroSD card

8.3.1.1 Partitioning Bootable MicroSD card [71]

We have inserted the MicroSD card into the development machine's card reader slot to make two partitions inside it. By default, the MicroSD card has only one partition. But we must need to have two partitions to install the Ångstrom Linux distribution.

However, we already have different file systems inside the development machine that is holding the development machines operating system. Additionally, we have inserted two file systems of MicroSD card through the card reader slot. Now we have used “df” command in the terminal. The “df” command displays “Filesystem” name, total blocks, used blocks, available blocks and used disk space percentage for all available partitions. We can see that, a single partition “/dev/sdb1” is available inside the MicroSD card. But we must need to have two partitions inside MicroSD card to make it bootable.

```
mainhasan@ubuntu:~$ sudo df

Filesystem  1K-blocks  Used  Available  Use%  Mounted on
/dev/loop0  16876388 15131088 888008  95%  /
none        1921916  360  1921556   1%  /dev
none        1926600  300  1926300   1%  /dev/shm
none        1926600  92  1925608   1%  /var/run
none        1926600   4  1926596   1%  /var/lock
none        1926600   0  1926600   0%  /lib/init/rw
```
To partition the default partition into two partitions, we have un-mounted the partition “/dev/sdb1” and then used fdisk utility that performs disk partitioning functions. Here, we have used the whole device “/dev/sdb” as the fdisk argument instead of a single partition.

Now, we have printed the default physical geometry of the MicroSD card. It shows the total size, numbers of heads, sectors per track and number of cylinders.
We have entered into the “Expert” mode. Now, we have changed the default geometry of the MicroSD card. We have set the head to 255, sectors to 63 and calculated the number of cylinder from the number of available bytes instead of default heads, sectors and cylinders. Moreover, we have divided the number of bytes by 255 heads, then by 63 sectors and then by 512 bytes per sector. Subsequently, we have got 967.59 cylinders (7958691840/255/63/512). But we have rounded to 967 cylinders.

Command (m for help): x

Expert command (m for help): h
Number of heads (1-256, default 151): 255

Expert command (m for help): s
Number of sectors (1-63, default 15): 63
Warning: setting sector offset for DOS compatibility

Expert command (m for help): c
Number of cylinders (1-1048576, default 6862): 967

Now, we have returned to the fdisk’s main mode and created a 32 MB large FAT partition.

Expert command (m for help): r

Command (m for help): n
Command action
  e  extended
  p  primary partition (1-4)

p
Partition number (1-4): 1
First cylinder (1-967, default 1): 1
Last cylinder, +cylinders or +size{K,M,G} (1-967, default 967): +32M

Here, we have changed new partition type to FAT32.

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))
We have used 32MB FAT partition or partition 1 to keep boot files. So, we have made this partition bootable.

Command (m for help): `a`
Partition number (1-4): `1`

Now, we have created another ext3 partition to keep root file system.

Command (m for help): `n`
Command action
`e` extended
`p` primary partition (1-4)

Partition number (1-4): `2`
First cylinder (6-967, default 6): `6`
Last cylinder, +cylinders or +size{K,M,G} (6-967, default 967): `967`

Now, we have looked at the partitions to check that everything is done properly or not.

We have already two partitions. Partition 1 for boot files and partition 2 for root file system. Now, we have printed the MicroSD card’s physical geometry again for verification.

Command (m for help): `p`

Disk /dev/sdb: 7958 MB, 7958691840 bytes
255 heads, 63 sectors/track, 967 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xb2a0e61c

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/sdb1</td>
<td>*</td>
<td>1</td>
<td>5</td>
<td>40131</td>
<td>c</td>
<td>W95 FAT32 (LBA)</td>
</tr>
<tr>
<td>/dev/sdb2</td>
<td>6</td>
<td>967</td>
<td>7727265</td>
<td>83</td>
<td>Linux</td>
<td></td>
</tr>
</tbody>
</table>

We have successfully created two partitions inside the MicroSD Card. Now, we have written these two new partitions in the partition table of MicroSD card to overwrite the previous partition table. The partition table inside the MicroSD card has updated with the two new partitions.
Command (m for help): \texttt{w}  
The partition table has been altered!  
Calling ioctl() to re-read partition table.  
WARNING: If you have created or modified any DOS 6.x partitions, please see the fdisk manual page for additional information.  
Syncing disks.  

Now, we have formatted both partitions. To do that, we have un-mounted both partition from the system. Otherwise the partitions will not be formatted.  

\begin{verbatim}
root@ubuntu:/home/mainhasan# sudo umount /dev/sdb1  
root@ubuntu:/home/mainhasan# sudo umount /dev/sdb2  
\end{verbatim}

Here, we have formatted the 32MB FAT partition or “/dev/sdb1” partition.  

\begin{verbatim}
root@ubuntu:/home/mainhasan# sudo mkfs.vfat -F 32 /dev/sdb1 -n FAT  
mkfs.vfat 3.0.9 (31 Jan 2010)  
\end{verbatim}

Now, we have formatted the comparatively large ext3 partition or “/dev/sdb2”.  

\begin{verbatim}
root@ubuntu:/home/mainhasan# sudo mkfs.ext3 /dev/sdb2  
mke2fs 1.41.12 (17-May-2010)  
Filesystem label=  
OS type: Linux  
Block size=4096 (log=2)  
Fragment size=4096 (log=2)  
Stride=0 blocks, Stripe width=0 blocks  
483328 inodes, 1931816 blocks  
96590 blocks (5.00%) reserved for the super user  
First data block=0  
Maximum filesystem blocks=1979711488  
59 block groups  
32768 blocks per group, 32768 fragments per group  
8192 inodes per group  
Superblock backups stored on blocks:  
32768, 98304, 163840, 229376, 294912, 819200, 884736,  
1605632  
Writing inode tables: done  
Creating journal (32768 blocks): done  
Writing superblocks and filesystem accounting information: done  
\end{verbatim}
This filesystem will be automatically checked every 25 mounts or 180 days, whichever comes first. Use `tune2fs -c` or `-i` to override.

### 8.3.2 Copying image file to MicroSD card

We have already created and formatted two partitions inside MicroSD card. Now, we have again mounted the MicroSD card inside the development machine. After using “df” command both partition of MicroSD card becomes visible.

```
mainhasan@ubuntu:~$ sudo df

Filesystem   1K-blocks Used Available Use% Mounted on
/dev/loop0    16876388  15131088    888008  95% /
none          1921916     360  1921556   1% /dev
none          1926600     300  1926300   1% /dev/shm
none          1926600      92  1926508   1% /var/run
none          1926600       4  1926596   1% /var/lock
none          1926600       0  1926600   0% /lib/init/rw
/dev/sda2    158149628  96025460  62124168  61% /host
/dev/sda4    63026172  40432624  22593548  65% /media/G
/dev/sdb2    7597856    148184  7063712   3% /media/b70e8b5c-de5b-40e9-8ab0-ad5c4701ceeb
/dev/sdb1    39497        1  39497  1% /media/FAT
```

From these columns, names of the File system and mounted location are important for us. The Filesystem name is used when un-mounting the MicroSD card's partition. Moreover, Filesystem's mount location is used to copy the image file to the particular partition. We need to keep in mind that, each time we mount the MicroSD card, it’s “/dev/sdb2” partition is mounted on new location. This new location is necessary to copy the Image files from development machine to partitions inside the MicroSD card. We have placed the boot files and root file system inside the development machine’s home directory. Consequently, we have to change our directory to the home directory.

```
mainhasan@ubuntu:~$ sudo su
```
Firstly, we have copied boot file “MLO”, “u-boot.bin”, “uImage” to the “/dev/sdb1” partition using “cp” command. The partition “/dev/sdb1” is mounted on “media/FAT” location. Subsequently, boot files are copied to the “/media/FAT” location. Secondly, we have copied the root file system “mainhasan_rfs.tar.bz2” to the “/dev/sdb2” partition. The partition “/dev/sdb2” is mounted on “/media/b70eb85c-de5b-40e9-8ab0-ad5c4701ceeb” location. Subsequently, we have copied the root file system to the “/media/b70eb85c-de5b-40e9-8ab0-ad5c4701ceeb” location. After that we have changed our directory to the “/media/b70eb85c-de5b-40e9-8ab0-ad5c4701ceeb” location or partition “/dev/sdb2”. Now, we have extracted the “mainhasan_rfs.tar.bz2” root file system inside this directory. When extraction is finished then the MicroSD card becomes ready to boot the Ångstrom operating system. The figure 99 shows extracted root file system inside partition ext3.

Figure 99: Extracted root file system inside partition ext3
8.3.3 Serial Communication

After we have successfully partitioned and copied image files, we have initialized the operating system. The operating system boots automatically after giving the power supply. The system's operating system gets ready after calibration of the LCD touch-screen during the first boot. We have access the computer as root user from the terminal. But small LCD screen makes it harder or at least inconvenient to interact with the system's operating system. To solve this problem, we have used the C-Kermit.

C-Kermit is a network and serial communication software package that offers cross-platform connection establishment, terminal session between development machine and target machine. The software package mainly a scripting language that facilitates the development in various ways. However, we have used C-Kermit only for terminal session. It means, we were able to access to the Gumstix's operating system terminal from the development machine and control the computer as a root user. A serial cable is used to connect between development machine's USB port and Gumstix's mini-USB console port. The connection establishment is done using the following algorithm from the development machine's terminal.

$ kermit -l /dev/ttyUSB0

C-Kermit> set flow-control none
C-Kermit> set carrier-watch off
C-Kermit> set speed 115200
C-Kermit> connect

Figure 100: Terminal emulation between target Gumstix board and Development machine
At this point, we have given the power supply to the Gumstix. Then, the whole boot up process will appear after the dotted line given above. The C-Kermit can be installed in the Ubuntu-10.04 Lucid development machine using the following command that is connected with internet. The figure 100 shows terminal emulation between target Gumstix board and development machine.

```
mainhasan@ubuntu:~$ sudo apt-get install ckermit
```

### 8.3.4 Wi-Fi Communication

To enable the wireless communication through the operating system’s network setup settings, we can use either Static ip address or Dynamic Host Configuration Protocol (DHCP) protocol or Point-to-Point (PPP) protocol. We have used DHCP. Besides, we have put the right Domain Name Server (DNS) address of the available wireless network in the network setting interface. We have found the DNS ip address using the following command in the terminal.

```
cat /etc/resolv.conf
```

![Image of terminal output showing DNS ip address](image)

**Figure 101: DNS ip address**

The figure 101 shows the process of finding out DNS ip address. Sometimes, putting the DNS in the network setting interface does not change the DNS address. In that case, we have changed the DNS address inside the “resolv.conf” file of the operating system. The file is accessed using the following command.

With **nano** editor-

```
nano /etc/resolv.conf
```

With **vi** editor-

```
vi /etc/resolv.conf
```
Furthermore, the Wi-Fi of the Gumstix can be configured through the terminal. This process is faster and saves valuable development time. We have enabled the WLAN in the Gumstix using the following algorithm.

```
>> iwconfig wlan1 essid any
>> ifconfig wlan1 up
>> iwlist wlan1 scan
>> ifdown wlan1
>> ifup wlan1
```

Here, “iwconfig wlan1 essid any” configures or enable the wlan1 for any type of wireless network. After that, “ifconfig wlan1 up” initialize the wlan1. Then, “iwlist wlan1 scan” shows the list of scanned available wireless network. After this command, all captured wireless networks by WLAN antenna will be shown in the list with all of their properties like network name, security type, signal strength and used channel. Then, “ifdown wlan1” and “ifup wlan1” is used to tweak the WLAN. If WLAN is up then it is turned down and if WLAN is down then it is turned up respectively.

### 8.3.5 Operating system update

The operating system must need to be updated to with latest software packages to ensure proper and smooth development environment. The Ångstrom Linux distribution uses slightly different package management tool to install new software packages than other Linux distributions.

By default, Ångstrom is not updated with all C and C++ development components. The operating system can be updated either manually or via Ångstrom's built-in package management tool “opkg”. Manual updated is harder and error-prone because of large amount of dependencies. We have installed all the necessary software components for C and C++ SDK using the following command. To mention, before using “opkg”, we have to do the “opkg update” first to install C and C++ SDK components. Beside, active internet connection to the Gumstix is also needed for the installation of the SDK. All necessary development component including “gcc” and “g++” is installed after system update.

```
root@overo: opkg update
root@overo: opkg install task-native-sdk
```
8.3.6 Native compiler

Native compilers are essential tools for compilation of different algorithms and in the creation of executable binary. We have used the native compiler in the sporadically throughout the development process. We have mostly used g++ compiler. A “.cpp” file can be compiled to create an executable object as follows. We are demonstrating the use of the compiler with a simple C++ program to make it more understandable.

First we have created a “.cpp” file using nano text editor.

```
root@ubuntu:/home/mainhasan# nano mainhasan.cpp
```

Now, we have created an object “testobject” by compiling mainhasan.cpp file using g++. We can create an object by any name.

```
root@ubuntu:/home/mainhasan# g++ -o testobject mainhasan.cpp
```

Here, we have given the command to the computer to execute that object.

```
root@ubuntu:/home/mainhasan# ./testobject
```

Enter the first value:
10
Enter the 2nd value
10
Enter the 3rd value
1

The result is 7

```
root@ubuntu:/home/mainhasan#
```

We can clearly see, after running the created object it runs the program what was written inside the mainhasan.cpp file. This example program calculates average of three integer value entered by the user. We have used this procedure to test different source files.

Furthermore, when we have compiled the programs that used external library like OpenCV then we have told the compiler where our library is situated. Otherwise, the compiler will show the error. We have used “pkg-config” software to query the installed library for opencv. “pkg-config” software provides a unified interface to query any installed library for compiling software source code from that library. To use “pkg-config”, we first need to install “pkg-config” in the computer.
We have told “pkg-config” to query for installed OpenCV library's library files and linker flags. Now, the compiler knows where to find OpenCV library during the compilation of the source code.

8.3.7 CMake Build and Installation

We have installed OpenCV using CMake. It was mentioned earlier about CMake including the reason to use it. First we have to install CMake inside the Gumstix as a prerequisite to install OpenCV library.

CMake is available in both binary and source distribution. To mention, binary distribution is pre-compiled and available with necessary executable objects. It is only necessary to put it in the Gumstix and link it to the Gumstix’s native compiler. However, we have used source distribution and compiled it inside the Gumstix. The CMake source distribution file “cmake-2.8.6.tar.gz” is copied to the larger “dev/sdb2” partition of the MicroSD card. After transferring the CMake source distribution file to the MicroSD card, we have boot the operating system from that same MicroSD card. The remaining procedure is done inside the Gumstix.

Now we have to use “df” command in the terminal. The “df” command displays “Filesystem” name, total blocks, used blocks, available blocks and used disk space percentage for all available partitions. From these columns, names of the File system and mounted location are important for us. Now, a question may arise how we have chosen the right partition that containing the CMake source file? To mention, during the practical development more partition will appear which can be seen from Appendix B. Here, we have used our intuition to select the right partition. When we have partitioned the MicroSD card, we have made one partition with higher memory capacity and another one with lower memory capacity. And, we have used smaller partition for boot files and larger partition for root file system and other software packages. From the “1K-blocks” column we can decide the larger and smaller partition. Beside, “Use%” column also helps to decide larger and smaller partition in some sense.

We have transferred the CMake source file inside the “/dev/mmcblk0p1” partition and it is mounted on “/media/mmcblk0p1”. Consequently, we have changed the directory and entered inside the partition. After that, we have extracted the “cmake-2.8.6.tar.gz” source distribution file and it will extract a folder named “cmake-2.8.6”. Subsequently, we have changed our directory again and entered inside that folder. Now we have used “./configure” script to configure and generate build files from the source files. After generation is done, we have build those files using Angstrom's native build tool “make” and installed them in the system.
The figure 102 shows the file systems inside Gumstix.
8.3.8 OpenCV Build and Installation

The process of building OpenCV-2.3.1 inside Gumstix is almost similar to the Cmake. But still there are some differences. First we have copied the OpenCV-2.3.1 source file to the appropriate MicroSD card partition. Later we have entered inside that partition and extracted “OpenCV-2.3.1a.tar.bz2” source file. Consequently, we have changed our directory and entered inside “OpenCV-2.3.1” folder.

Now, we have created a new folder inside this folder named “mainhasan” using “mkdir” command. The “mkdir” command makes a new directory inside any folder. Then we have changed our directory again and entered inside the “mainhasan” folder. Now, we have used “cmake” and Cmake variables which configure the “OpenCV-2.3.1” source files and generate build files. The details of configuration and generation are given in the Appendix C. Here we have used three Cmake variable “CMAKE_BUILD_TYPE”, “CMAKE_INSTALL_PREFIX” and “BUILD_PYTHON_SUPPORT” to send -D command line argument “RELEASE”, “/usr/local”, “ON” to cmake. During the development we have also build the debug version of the OpenCV. During debug build, we have the changed the “CMAKE_BUILD_TYPE” variable argument “RELEASE” to “DEBUG”. Thus we have build the both “RELEASE” and “DEBUG” version of OpenCV-2.3.1.

We have build the OpenCV inside the folder or directory named “mainhasan” instead of “OpenCV-2.3.1” folder. It is because we have made a out-of-source build instead of in-source build. If we build the OpenCV inside “OpenCV-2.3.1” directory than all the build files will be written to the source tree. That is why it is named as in-source build. Once build files are written to the source tree, we cannot make a new build from that source tree. But it is often required to make more than one build during the practical development. If we build the OpenCV using a separate directory than all the build files will be written to that new directory and OpenCV source tree will remain intact. Thus, we can create several directories and create multiple build variants of OpenCV.

```bash
root@overo:~# cd /media/mmcblk0p2
root@overo:/media/mmcblk0p2# sudo tar xvf OpenCV-2.3.1a.tar.bz2
root@overo:/media/mmcblk0p2# cd OpenCV-2.3.1
root@overo:/media/mmcblk0p2/OpenCV-2.3.1# mkdir mainhasan
root@overo:/media/mmcblk0p2/OpenCV-2.3.1# cd mainhasan
root@overo:/media/mmcblk0p2/OpenCV-2.3.1/mainhasan# cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON ..
```
Moreover, the following method will also work like the above method. Here, instead of using Cmake variable and argument if can simply use “\texttt{cmake ..}” and then we have to give the command “\texttt{make install}”.

8.3.9 FFmpeg, V4L2, GStreamer, X264 build and installation
The similar method of manually installing Cmake and OpenCV is also used to build and install FFmpeg, V4L2, GStreamer, x264. However, the above software dependencies can be directly installed on the computers using command terminal from the Ubuntu repository. The host computer must need to be connected to the internet in order to be able to use the Ubuntu repository. However, manual installation method is used mostly in this thesis project to avoid some random error. The following methods and shell commands are used randomly for manual and direct installation.

- **FFMPEG [72]**
  - To get ffmpeg enabled in OpenCV, we have put the FFmpeg source code inside the OpenCV directory. Then we have extracted the FFmpeg source code and build it. Now, we have “make” the OpenCV again and its configuration is shown as FFmpeg enabled.
  - "make distclean (if necessary)"
  - 
    ```bash
    ./configure --enable-gpl --enable-libfaac --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libtheora --enable-libvorbis --enable-
    ```
libx264 --enable-libxvid --enable-nonfree --enable-postproc --enable-version3 --enable-x11grab --enable-shared --enable-pic

- make
- sudo make install

---

### Gstreamer: installing GStreamer plugins on Ubuntu Linux

- sudo apt-get install gstreamer0.10-schroedinger

---

### Gstreamer: more plugins Installation [72]

- sudo apt-get install libgstreamer0.10-0 libgstreamer0.10-dev gstreamer0.10-tools gstreamer0.10-plugins-base libgstreamer-plugins-base0.10-dev gstreamer0.10-plugins-good gstreamer0.10-plugins-ugly gstreamer0.10-plugins-bad gstreamer0.10-ffmpeg

---

### GEntrans [73]

- The GEntrans API was tested during the project for time stamping the incoming video frames. The API was downloaded from the internet and following commands are used to install it.

- Downloaded from the internet
- ./configure --prefix=/usr
- sudo make install

---

### UVC [74]

- sudo aptitude install mercurial build-essential linux-headers libncurses5-dev
- hg clone http://linuxtv.org/hg/v4l-dvb/
- cd v4l-dvb
- sudo cp /boot/config-`uname -r` v4l/.config
- sudo make menuconfig
- make
- sudo make install
- sudo add-apt-repository ppa:libv4l
- sudo aptitude update
- sudo aptitude full-upgrade
- **V4L2 [75]**
  - Downloaded from the internet
  - make
  - `sudo make install`

- **x264 [76]**
  - Downloaded from the internet
  - make
  - `sudo make install`
  - In 64-bit version of Ubuntu and for ARM architecture, x264 is configured as shown in the following command:
    ```
    ./configure --enable-shared --enable-pic
    ```

- **Other important dependencies installed**

1. All the dependencies for x264 and ffmpeg.
   - `sudo apt-get update`
   - `sudo apt-get install build-essential checkinstall git cmake libfaac-dev libjack-jackd2-dev libmp3lame-dev libopenpcore-amrnb-dev libopenpcore-amrwb-dev libsdl1.2-dev libtheora-dev libva-dev libvdpau-dev libvorbis-dev libx11-dev libxfixes-dev libxvidcore-dev texi2html yasm zlib1g-dev`

2. gtk, pkg-config and libjpeg installation
   - `sudo apt-get install libgtk2.0-0 libgtk2.0-dev`
   - `sudo apt-get install pkg-config`
   - `sudo apt-get install libjpeg8 libjpeg8-dev`

### 8.4 PandaBoard ES
The configuration process of PandaBoard ES is comparatively easier than Gumstix COM because it allows installing heavier operating system like Ubuntu Desktop. However, the knowledge of installation process used for the Gumstix COM also applies to the PandaBoard ES. In this thesis project, we have used different power adaptor, bootable SD card, connector for PandaBoard ES than those used for Gumstix COM. The biggest advantage of PandaBoard
ES is that it allows working with Ubuntu Desktop operating system in a larger monitor. This makes the development setup much easier without needing to use serial communication.

8.4.1 Bootable SD Card
We have already mentioned that the knowledge of installation process used for Gumstix COM also applies to PandaBoard ES. However, there is alternative method [77] which allows preparing bootable SD card for PandaBoard ES more easily. The method we have followed is as below-

8.4.1.1 Configurations steps

1. Downloading operating system
2. Placing the SD card inside the host computer
3. Un-mounting SD card if it is mounted
4. Finding out raw device (SD card) name like /dev/sdb not device partition like /dev/sdb1
5. Running installation commands on the terminal

8.4.1.2 Installation commands

Method 1:

```
>> zcat ./ubuntuOS.img.gz | sudo dd bs=4M of=/dev/sdb ; sudo sync
```

Method 2:

```
>>gunzip ubuntuOS.img.gz

>>sudo dd bs=4M if=ubuntuOS.img.gz of=/dev/sdb

>>sudo sync
```

8.5 Configuring Linux kernel with Real-time patch
We have build the Linux kernel with real-time configuration from the source. It means that, we have not just applied real-time patches using terminal window while the operating system is running. The building from source means we have configured operating system boot files such as “uImage” manually by applying real-time kernel patches before preparing the bootable SD card. However, the configuration sequence can be described as follows-
- Downloading kernel source code
- Downloading real-time patches
- Compiling new “uImage” and “module” files
- Installing module files inside a directory
- Inserting SD card into development machine
- Copying “uImage” and module directory to the SD card
- Unmounting SD card and removing from development machine
- Inserting SD card into PandaBoard

In the figure 103, we can see the kernel version and patch using “uname –a” command on the terminal.

![Figure 103: Real-time PREEMPT kernel in PandaBoard ES](image)

### 8.6 Debugging

Debugging has been an extremely useful method to solve different programming problems throughout the development activities. However, it was necessary to build a separate DEBUG version of the OpenCV library to perform debugging. As the development environment was Linux, GDB has been used most of the time. An instance of debugging is given in the figure 104 and 105.
A camera needs a driver to interact with any type of operating system. We have used cameras that were made by MISUMI Electronics Corporation and uses UVC video driver. Moreover, we have also used cheap USB-2.0 cameras. The needed drivers were updated to its latest version. Furthermore, during different point of practical development troubleshooting, we had to use different video streaming software like “guvcview”, “xawtv”, “cheese”.

8.7 Camera configuration
<table>
<thead>
<tr>
<th>Command</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls /dev/video*</code></td>
<td>Displays connected video devices</td>
</tr>
<tr>
<td><code>ls –ltlh /dev/video*</code></td>
<td>Displays connected video devices with more details</td>
</tr>
<tr>
<td><code>lsusb</code></td>
<td>List all devices attached to usb</td>
</tr>
<tr>
<td><code>sudo apt-get install guvcview</code></td>
<td>Installs guvcview from appropriately supported repository of Linux distribution</td>
</tr>
<tr>
<td><code>sudo apt-get install xawtv</code></td>
<td>Installs xawtv from appropriately supported repository of Linux distribution</td>
</tr>
<tr>
<td><code>sudo apt-get install cheese</code></td>
<td>Installs cheese from appropriately supported repository of Linux distribution</td>
</tr>
</tbody>
</table>

### 8.8 Software development

We have used OpenCV as the base algorithm for this project. However, we have modified and restructured it substantially and written new code that created new software program with new functionalities. The result can be divided into two steps-

1. A software with Graphical user interface (GUI) that run both on Windows and Linux

The first software with its own GUI runs independently which means additional third party dependency support is not required. The software uses third party dependencies but all dependencies are included in the software installer package. Hence, the software can run in any Windows and Linux platform without requiring installation of dependencies. The second software need to be build inside the OpenCV directory with necessary dependencies. However, once the code is compiled and the executable binary is generated, the .exe can be used from any directory inside native Linux operating system.

### 8.9 A standalone software with GUI

The Qt-4.8 is used to develop the GUI and to integrate other system processes. The new Qt-5 is tested but later abandoned because of its lacking in support of all required necessary
dependencies for this project. However, the development work can be divided into the following parts-

1. Qt Creator IDE, libraries and third party API configuration
2. GUI design
3. Functional implementation
4. Installer support

8.9.1 Qt Creator IDE, libraries and third party API configuration

8.9.1.1 Windows OS
The Qt creator IDE and libraries were configured step by step. The proper configuration steps reduce the error as the Qt creator is dependent on the Qt library and Qt library needed to be configured for third party API (e.g. OpenCV) support. There were errors during the configuration process and hence trouble-shooting was done accordingly. However, the configuration sequence was followed as below-

- Qt Creator, Qt library, CMake and OpenCV downloaded
- OpenCV is configured using Cmake and compiled using Visual Studio compiler
- Qt library installed
- Qt library is configured for OpenCV support using CMake
- Qt Creator installed and linking with Qt library
- Qt Creator project file (.pro) linking with OpenCV library via appropriate path

8.9.1.2 Linux OS
The configuration process in Linux is almost similar to the steps followed in Windows with slight differences. However, the configuration sequence was followed as below-

- Qt library, OpenCV, Cmake, V4L2, FFMPEG downloaded and installed
- Qt Creator is installed via Ubuntu software centre, alternatively also via terminal
- Qt Creator project file was linked to OpenCV build directory and library files via appropriate path
8.9.2 GUI Design

The GUI is very important for an interactive system. In other words, the GUI is the door for the human user to enter the computer’s abstract data processing layer. Moreover, a GUI needs to be user friendly besides having all functionalities to ensure a healthy human-computer interaction. The aspect of human-computer interaction is also an embedded part of optimal system gain. We have ensured the simplicity in GUI design in order to make the software user friendly. The conceptual diagram of the top-level widget and child widgets is given in the figure 106.

![GUI QWidget design blocks](image)

Figure 106: GUI QWidget design blocks

The GUI is developed using QWidget class. The QWidget class is the base class for all user interface objects and hence all GUI widgets inherit this class. A composite widget and several child widgets are used to create GUI blocks. The nesting of composite widget and other child widget ensures the proper shape and size of GUI component widgets. The top-level widget and other child widgets are bonded in a layout provided by QGridLayout. The figure 107 and 108 shows the GUI in Windows and Linux respectively.
Figure 107: Real-Time data display via GUI in Windows

Figure 108: Real-Time data display via GUI in Linux
A menubar is created and all six menus were inserted inside the menubar. The icons were used to make the menus more user-friendly. “QMenuBar” and “QMenu” is used to create menubar and menus respectively. “QAction” and it’s appropriate signal is used to link each menu with appropriate slot. The SIGNAL ans SLOT mechanism were used throughout the linking process between menu and its corresponding operation. “QLabel” and “QLineEdit” is used to create the left-side data display of the GUI. The “QString” class is used to provide data to the “QLineEdit”. “QAbstractItemView” is used to create the pie chart in the middle of the GUI and “QStandardItemModel”, “QTableView”, “QHeaderView” is used to create the data table on the right-side of the GUI. The “QAbstractItemModel” and “QItemSelectionModel” is used for both pie chart and data table.

The algorithm response time data is collected and inserted into the pie chart and data table. The data is stored in the “QVector” class, process names are stored in “QStringList” class and random color for pie chart is provided by the “QColor” class. The “QModelIndex” is used to insert the data into the data table and pie chart. The “QImage” and “QPixmap” class is used to display image in the child widget. The SLOTS uses OpenCV classes (e.g. Mat, IplImage, VideoCapture, CvCapture etc.) and “PieChart” class also uses more classes.
In the figure 109, we have given a summary of the some used high-level classes and relational links among them that were used to implement the GUI and menus and its corresponding stitching and other functionalities.
In the figure 110, we can see that each menu have one or several options. And each option is associated with a distinct QAction object. Each QAction object is a sender as each of them sends a signal. In our case, these QAction object sends a signal “triggered()” which means an option of the menu is clicked. Then the “this” pointer receives those emitted signals and puts the signals into corresponding slots which executes the corresponding code that resides inside it. This is how signal and slot mechanism works in Qt.

8.9.3 Configuring with installer
An installer package was used to put everything together. This was done in two steps. First, we have found out the dependencies for the .exe application using dependency walker. The dependency walker shows all the necessary operating system .dll files and third party libraries .dll files on which the .exe application depends on. We have copied all necessary .dll file into a single folder along with .exe application. Then, we have used NSIS installer package to convert the folder into a simple .zip based portably installable application file. We have tested the installable software into a computer that does not have any (except operating system .dlls) necessary dependencies (e.g. OpenCV) installed. The software successfully executed all of its functionalities and hence proved that all extracted dependency .dlls are running smoothly. The figure 111 shows the use of Dependency Walker for finding out .dll (Dynamic-link library).
8.10 A CLI based reconfiguration of OpenCV stitching module

We have restructured the OpenCV stitching module. The default module did not have support for homography matrix and focal length extraction and real-time stitching from a set of images. The reconfiguration process can be divided into two parts. In other word, we have modified the code and restructured it into two parts and then added two new commands to the CLI. One part does the calibration and saves the extracted homography matrix into a look file. Another part read the homography matrix from the look file and uses the homography matrix to generate real-time panorama. We have debugged through the default code to understand its execution time data flow using GDB and then reconfigured it. We have used OpenCV “FileStorage” class to store the extracted homography matrix and focal length and reused the class to read back the stored data. The “FileStorage” class stores the data into “YML” file type. The figure 112 shows the calibration and real-time stitching in Linux.
8.10.1 Calibration part

This part starts execution by writing the ‘‘--calib yes’’ command beside other necessary input commands on the terminal. It extracts focal length and homography matrix and saves them into two separate .yml file. Moreover, it also displays the extracted focal length and homography matrix on the CLI. An instance of the execution is given in the figure 113.
8.10.2 Real-time stitching part

This part starts execution by writing the “--stitch yes” command beside other necessary input commands on the terminal. It reads back the previously saved homography matrix and focal length and saves them into appropriate data container class instances. After that, the real-time stitching part uses focal length and homography matrix to generate real-time panorama. The figure 114 shows the real-time stitching in Linux using stitch command.
8.10.3 Automated calibration and real-time stitching in Linux

We have created an automated version of the algorithm which automatically calibrates the camera system and then starts real-time stitching automatically just by running the executable binary once. To facilitate this automated process, we have assumed that the connected camera system is not calibrated as a default condition. We have marked this default condition as 0 and saved this information into a status indicator .yml file. The executed .exe stitching application first checks the file and finds that the camera system is not calibrated and hence it extracts the focal length and homography matrix first and then starts real-time stitching iteratively. Moreover, we have also created another automated test environment. This test environment works in a similar process but slightly in a different way. In this test environment, the executed .exe stitching application first read the status indicator .yml file and finds that camera system is not calibrated. Hence, it extracts the focal length and homography matrix and saves them into two separate .yml file and then changes the default condition of the status indicator .yml file into 1. Subsequently, during next execution, the stitching application again read the default condition of the status indicator .yml file and finds that the camera system calibrated. Hence, it skips the calibration and only read back previously saved focal length and homography matrix and runs the real-time stitching part iteratively. The figure 115 and 116 shows the automated and semi-automated test environment. Moreover, the figure 117 shows an instance of semi-automated test environment inside Linux.
Figure 115: Automated test environment

Figure 116: Semi-automated test environment
Figure 117: Semi-automated test environment inside Linux
Chapter 9

Proposed Image Stitching Algorithm

The implemented algorithm is adopted from the available solutions for image stitching to generate panorama. The available stitching solutions offer a lot of features to solve a wide range of stitching problems. However, developed panoramic imaging system under this thesis project has certain characteristics that require modification and task specific algorithm. Firstly, there are three images needed to be stitched from three fixed cameras. Secondly, the rotation among the camera’s centre of project is not known (not calibrated or fine-tuned) but can be intuitively approximated as the whole camera system has total FOV of approximately 180°. This intuitive approximation of rotation can be used to verify the result. Thirdly, the order of raw images that consists overlapping regions is known which excludes the necessity of using bundle adjuster to compute a globally consistent alignments.

9.1 Problem formulation

The stitching problem needed to be solved for this thesis has the following specific characteristics-

1. We have three cameras each with different centre of projection
2. We have three images with known order
3. We have first overlapping areas between 1st and 2nd image and second overlapping areas between 2nd and 3rd image.
4. There exist projective transformational relations among the three images because of the spherical camera system’s physical setup.
5. Each image is associated with distinct rotation, translation and focal length respective to each other.
6. Each camera has same exposure time during image capture.

9.2 Proposed Algorithm

We have used the OpenCV’s stitching module as the source of base algorithm to solve the requirement of this thesis project. The original algorithm is written to solve various functions
that may arise during image stitching. It gives independence to the user to provide different input parameters and hence getting desired output. However, we have a predefined requirement in this thesis project and subsequently all the functions of the original algorithm is not necessary for this project. Therefore, we have segmented the algorithm according to our need, further modified and written our new code to produce the desired output. Our proposed algorithm has 2 parts namely *camera system’s motion modeling* and *real-time stitching*.

![Proposed Algorithm Structure](image)

**Figure 118: Proposed Algorithm Structure**

The left side of the flow chart in figure 118 is the *camera system’s motion modeling or system calibration* part and right side is the real-time stitching part. The details execution flow of the each part is provided in the figure 119.
In the figure 119, we can see the execution flow of the proposed stitching algorithm. The left side of the above flow chart needed to be run only once for each new camera system. The right side of the above flow chart does the stitching irrespective of the image content in terms of features and other image quality parameters.
9.3 Motion Modeling or System calibration

This mathematical relation has a form of matrix which holds the geometrical properties of rotation matrix, translation vector and perspective vector to facilitate transformation of image pixels from one planar perspective projection to another. There are variety of such parametric motion models are possible. For example, simple 2D transforms planar perspective models, 3D camera rotation, lens distortions, non-planar perspective models. We have chosen the motion model that best suits the specific data situation of this project.

![Panorama Imaging System](image)

Figure 120: System calibration via panoramic mosaicking

In the figure 120, we have realized spherical imaging system in terms of input, output and transfer function. In this case, the transfer function of the spherical camera system is the unknown camera mapping which holds information about rotation, translation and camera intrinsic parameters. The rotation, translation and intrinsic parameters of the spherical camera system’s each camera can be known from the process of generating panoramic image mosaic.

First we have captured three images then calculated the projective transformation or homography that exists among the images to stitch the images in a unified plane or surface. Moreover, we also calculate median focal length in this part. This calculated relation or homography matrix and median focal length are then stored in a look file. This camera system’s motion modeling part needed to be executed only once. Because, once we have the motion model of the camera system then we can use this model repetitively unless we use a new spherical camera system.
The first step for motion modeling or system calibration is extracting salient features and then matching them pairwise. In the figure 121, we can see the camera system’s that we have used. The two red lines in the left side indicates the FOV of left camera, the two green lines in the middle indicates the FOV of middle camera and the two blue lines in the right side indicates the FOV of right camera. The area spanned by red and green line and marked with black circle is the overlapping area between left camera and middle camera. And, the area spanned by blue and green line and marked with orange circle is the overlapping area between middle camera and right camera. The corresponding features among the images exist because of these overlapping areas in the camera system’s FOV. In the figure 122, we have emulated three images with similar feature.

![Figure 121: Spherical camera system's FOV](image)

In the figure 122, the rectangular boxes with different border color are represented as image 1, image 2, and image 3 respectively from left to right. The overlapped region contains two stars and one hexagon that represent common feature. In a similar method, we select the
salient features from the overlapping area. Then we match these features pairwise to find out the corresponding features between two images. For example, from the above emulated images, the hexagon of the image 1 will match with hexagon of the image 2, so they are the corresponding features. However, the corresponding features from the real images will have difference between them in terms of perspective projection. This difference between corresponding features exists because they are captured with different camera that have different centre of projection and intrinsic parameters. Hence, a pair of corresponding features will provide pairwise homographies.

In the next step, we calculate the homography matrix that are associated with each image and then calculate the median focal length for each camera. We use this extracted data in real-time stitching part. In the figure 123, we can see an instance of homography matrices that have been generated from three cameras or three images. An instance of calculated median focal lengths is given in the figure 124.

Figure 123: Homography Matrix Look File
9.3.1 Motion model sequence diagram

The sequential model diagram provides an insight about the programs execution and as well as data flow. The figure 125 shows the sequence diagram of the motion modelling or calibration part.
9.4 Real-time Stitching

In real-time stitching part, we have used the obtained homography matrices from the camera system’s motion modeling part. First, we have captured three images and then we have stitched them using the homography matrix. In this part, we do not need to any kind of feature matching and homography matrix calculation. We just read back the calculated homography matrices and median focal lengths from the look file and use them to create panorama. We have created a plane compositing surface using warping scale calculated from focal lengths and then used the homography matrices we warp captured images into plane compositing surface. The figure 126, 127, 128, 129, 130 shows the processing warping three images into a plane surface.

![Figure 126: Sequential Warping or Projective mapping in a plane surface](image1)

![Figure 127: Three input images from three cameras](image2)
Figure 128: Warping first image into plane compositing surface

Figure 129: Sequentially warping second image into plane compositing surface

Figure 130: Sequentially warping third image into plane compositing surface
We have assigned the focal length to the projector. Then we have sequentially warped or perspectively mapped image 1, image 2 and image 3 into a plane compositing surface as follows

\[
\text{Output image}_{\text{warped}} = H_i \text{ Input image}_i
\]

Here, \( i = 1,2,3 \)

For all values of \( i = 1,2,3 \)

\[
\text{Output image}_{\text{panorama}} = H_i \text{ Input image}_i
\]

We have used “Plane” compositing surface as it consumes less amount of time in terms of coordinate transformation as well as preserves good visual quality.

### 9.4.1 Real-time stitching sequence diagram

The figure 131 shows the sequence diagram of real-time stitching part.

![Real-time stitching sequence diagram](image)

Figure 131: Real-time stitching sequence diagram

### 9.5 Algorithm Optimization

The original OpenCV algorithm’s response time was pretty high because of its multiple functionalities. It appeared to be implemented with all possible stitching scenarios that might occur. But in this thesis, we have specific problem of stitching three images with certain imaging condition. Hence, we have implemented our own algorithm and it considerably takes
less amount of time to stitch three images. To mention, we have tested semi-optimal algorithm with PandaBoard ES and Gumstix COM because these hardware were not available when we have obtained the optimal algorithm. The optimal algorithm was only tested with laptop computer.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Applied Stitching Condition</th>
<th>Other Stitching Conditions</th>
<th>Reference OpenCV Algorithm (Millisecond)</th>
<th>Proposed Algorithm (Millisecond)</th>
<th>Reduction (Percentage %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 x 240</td>
<td>Warping, Compositing</td>
<td>&gt;&gt; Matching Confidence 0.6</td>
<td>OS: Windows</td>
<td>OS: Windows</td>
<td>OS: Windows</td>
</tr>
<tr>
<td>(Three images)</td>
<td></td>
<td>&gt;&gt; No exposure compensation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt; No Blending</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt; No seam finding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;&gt; Plane Compositing surface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>415.175</td>
<td>62.6602</td>
<td>84.90752092%</td>
</tr>
</tbody>
</table>

The figure 132 and 133 shows the amount of optimization carried out.
Figure 132: Our proposed algorithm’s performance

Figure 133: Our proposed algorithm’s performance
9.5.1 Optimal Homography Matrix

The important part of our stitching algorithm is a homography matrix. The optimal homography matrix we will have the lesser noise and other inconsistencies will be contained in the final panorama. However, there is always a trade-off between stitching response time and visual quality and the choice depends on the area of application. We compute homography matrix using establishing feature correspondences between images through matched inlier feature pairs. To mention, the matching confidence threshold “match_conf” highly affect the quality control of matched feature because this threshold deeply affect the process of choosing best neighbor from two nearest neighbors for either image. The following very tiny code excerpt clearly depicts how it affects the quality of matched feature for either image.

```cpp
for (size_t i = 0; i < pair_matches.size(); ++i)
{
    if (pair_matches[i].size() < 2)
        continue;
    const DMatch& m0 = pair_matches[i][0];
    const DMatch& m1 = pair_matches[i][1];
    if (m0.distance < (1.f - match_conf_) * m1.distance)
    {
        matches_info.matches.push_back(m0);
        matches.insert(make_pair(m0.queryIdx, m0.trainIdx));
    }
}
```

Hence, this threshold also affects homography computation. To demonstrate this fact, we have calculated homography matrices with various matching confidence threshold and tested these homography matrix in real-time stitching part. The result is given below as a table. However, the results will vary from scene to scene. So, with an optimal homography matrix for respective camera system, the proposed algorithm will provide faster response time. The figure 134 shows the generated panorama with fastest response time.

![Generated Panorama in 62.6602 milliseconds with optimal homography matrix (With Warping and Compositing)](image_url)
<table>
<thead>
<tr>
<th>Matching Confidence</th>
<th>Homography Matrix computation possible?</th>
<th>Panorama Generated?</th>
<th>Panorama Generation Response Time (Milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>Yes</td>
<td>Split warping</td>
<td>290.501</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Not panorama)</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>Yes</td>
<td>Split Warping</td>
<td>157.705</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Not panorama)</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>Yes</td>
<td>Yes</td>
<td>137.112</td>
</tr>
<tr>
<td>0.3</td>
<td>Yes</td>
<td>Yes</td>
<td>84.8533</td>
</tr>
<tr>
<td>0.4</td>
<td>Yes</td>
<td>Yes</td>
<td>109.33</td>
</tr>
<tr>
<td>0.5</td>
<td>Yes</td>
<td>Yes</td>
<td>123.148</td>
</tr>
<tr>
<td>0.55</td>
<td>Yes</td>
<td>Yes</td>
<td>116.381</td>
</tr>
<tr>
<td>0.6</td>
<td>Yes</td>
<td>Yes</td>
<td>62.6602</td>
</tr>
<tr>
<td>0.65</td>
<td>Yes</td>
<td>Yes</td>
<td>84.9118</td>
</tr>
<tr>
<td>0.7</td>
<td>Yes</td>
<td>Yes</td>
<td>193.904</td>
</tr>
<tr>
<td>0.8</td>
<td>Yes</td>
<td>Split Warping</td>
<td>560.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Not panorama)</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>No</td>
<td>No</td>
<td>X</td>
</tr>
<tr>
<td>1.0</td>
<td>No</td>
<td>No</td>
<td>X</td>
</tr>
</tbody>
</table>
Chapter 10

Hardware Implementation

The stitching process is implemented inside prepared embedded Linux operating system of Gumstix board and PandaBoard ES. Besides, we have also tested the software in several laptop computer platforms both in Windows and Linux. Some auxiliary hardware e.g. monitor, mouse, keyboard has been used throughout the hardware implementation part of the thesis. In this chapter, we have included some samples from the hardware implementation. The figure 135 shows the assembled Gumstix COM.

10.1 Implementation inside Gumstix

![Gumstix COM](image)

Figure 135: Used Gumstix COM (Overo Fire, Expansion Board, LCD and MicroSD card)

10.1.1 Homography or Motion estimation

```
root@overo:/media/mmcblk0p2/ow/mainhasan/bin# ./opencv_stitching
image14.jpg image15.jpg image16.jpg --preview --calib yes
Finding features...
Features in image #1: 521
Features in image #2: 353
Features in image #3: 485
Finding features, time: 74.7832 sec
Pairwise matching...
Pairwise matching, time: 5.3017 sec
Estimating rotations...
```
10.1.1.1 Result after changing advanced default CPU parameters

<table>
<thead>
<tr>
<th>General</th>
<th>Caches</th>
<th>Edje Cach</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Framerate: 30 fps</td>
<td>• Cache flush interval:</td>
<td>• Number of Edje files to</td>
</tr>
<tr>
<td>• Applications priority: 0</td>
<td>512 ticks</td>
<td>cache: 32 files</td>
</tr>
<tr>
<td></td>
<td>• Font cache size: 0.5 MB</td>
<td>• Number of Edje collections</td>
</tr>
<tr>
<td></td>
<td>• Image cache size: 4 MB</td>
<td>to cache: 64 collections</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Result</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nearly 50% faster processing</td>
</tr>
</tbody>
</table>

root@overo:/media/mmcblk0p2/ow/mainhasan/bin#./opencv_stitching
image14.jpg image15.jpg image16.jpg --preview --calib yes
Finding features...
Features in image #1: 521
Features in image #2: 353
Features in image #3: 485
Finding features, time: 36.8938 sec
Pairwise matching...
Pairwise matching, time: 2.67859 sec
Estimating rotations...
Estimating rotations, time: 0.00106811 sec
Initial focal length #1: 2609.86
Initial focal length #2: 2609.86
Initial focal length #3: 2609.86
Bundle adjustment..........................
...
Bundle adjustment, final error: 8.47803
Bundle adjustment, iterations done: 64
Bundle adjustment, time: 0.315033 sec
Camera #1 focal length: 1789.95
Camera #2 focal length: 1786.33
Camera #3 focal length: 1787.21
Wave correcting...
Wave correcting, time: 0.00390625 sec
First Matrix: [0.98564869, -0.014340634, 0.16819923;
  0.00427067, 0.9981845, 0.06007893;
  -0.16875544, -0.058498401, 0.98392051]
Second Matrix: [0.99996316, 0.0080944132, 0.0028666465;
  -0.0082451161, 0.9983297, 0.057181671;
  -0.0023990064, -0.0572032, 0.99835974]
Third Matrix: [0.9852404, 0.0061483886, -0.17106587;
  0.0040958873, 0.99822181, 0.059467603;
  0.17112733, -0.059290554, 0.98346335]
Finished, total time: 39.923 sec

10.1.2 Real-time Panorama stitching

Stitching without Seam finder.

root@overo:/media/mmcblk0p2/ow/mainhasan/bin# ./opencv_stitching
image14.jpg image15.jpg image16.jpg --stitch yes --seam no --output
hasannow.jpg
Warping images (auxiliary)...
Warping images, time: 2.86645 sec
Compositing...
Compositing image #1
Multi-band blender, number of bands: 5
Compositing image #2
Compositing image #3
Compositing, time: 9.96079 sec
Finished, total time: 12.82724 sec

Stitching with seam finder.

root@overo:/media/mmcblk0p2/ow/mainhasan/bin# ./opencv_stitching
image14.jpg image15.jpg image16.jpg --stitch yes --output
hasannow_withoutseam.jpg
Warping Images (auxiliary)...
Warping images, time: 3.023567 sec
Finding seams...
Finding seams, time: 5.60000 sec
Compositing...
Compositing image #1
Multi-band blender, number of bands: 5
Compositing image #2
Compositing image #3
Compositing, time: 10.82297 sec
Finished, total time: 19.446537 sec
root@overo:/media/mmcblk0p2/ow/mainhasan/bin# exit

10.1.2.1 Result after changing advanced default CPU parameters

<table>
<thead>
<tr>
<th>General</th>
<th>Caches</th>
<th>Edje Cach</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Framerate: 30 fps</td>
<td>• Cache flush interval: 512 ticks</td>
<td>• Number of Edje files to cache:32 files</td>
</tr>
<tr>
<td>• Applications priority: 0</td>
<td>• Font cache size: 0.5 MB</td>
<td>• Number of Edje collections to cache:64 collections</td>
</tr>
<tr>
<td></td>
<td>• Image cache size: 4 MB</td>
<td></td>
</tr>
</tbody>
</table>

Result
Nearly 50% faster processing

Stitching without seam finder

root@overo:/media/mmcblk0p2/ow/mainhasan/bin# ./opencv_stitching image14.jpg image15.jpg image16.jpg --stitch yes --seam no --output hasantesting.jpg
Warping images (auxiliary)...
Warping images, time: 1.54884 sec
Compositing...
Compositing image #1
Multi-band blender, number of bands: 5
Compositing image #2
Compositing image #3
Compositing, time: 5.22503 sec
Finished, total time: 6.77387 sec

Stitching with seam finder

root@overo:/media/mmcblk0p2/ow/mainhasan/bin# ./opencv_stitching image14.jpg image15.jpg image16.jpg --stitch yes --output hasantest.jpg
Warping images (auxiliary)...
Warping images, time: 1.88242 sec
10.2 Implementation inside PandaBoard

The figure 136 shows the PandaBoard ES with external hardwares.

Figure 136: PandaBoard ES with external hardwares
10.3 Implementation inside Laptop Computer
The system is also tested on the laptop with both Windows 7 and Ubuntu Linux 64 bit operating system. The image capturing, motion modeling and stitching process works smoothly in laptop computer without any difficulties.

10.3.1 Inside Windows operating system
The reconfigured OpenCV stitching module also tested in Windows environment. The OpenCV was configured using Visual Studio IDE and Visual Studio command prompt is used to test the stitching module. We have followed the following steps in to implement stitching in windows.

- OpenCV downloaded, CMake downloaded and installed
- OpenCV native source files are generated using CMake
- Reconfiguration of stitching module
- Compiling OpenCV native source file using Visual Studio compiler
Used Visual Studio command prompt to do the motion modeling and real-time stitching.

We have achieved best stitching result under Windows operating system as well as least jitter. The Windows performed with real-time characteristics even though we have not re-configured or patched it for real-time operation. This is understandable as the Microsoft always tries to deliver top quality products as a return of customers’ valuable money. The figure 137, 138, 139 shows the motion modeling part using calib command, real-time stitching part using stitch command and two instances of real-time stitching respectively.

![Visual Studio x64 Cross Tools Command Prompt (2010)](image)

Figure 137: Motion modeling inside Windows with calib command
Figure 138: Real-time stitching using stitch command in Windows

Figure 139: Two instances of real-time stitching in Windows with optimal algorithm
10.3.2 Inside Linux operating system

The figure 140, 141, 142 shows the motion modeling part using calib command, real-time stitching part using stitch command and two instances of real-time stitching respectively.

Figure 140: Motion modelling using calib command in Linux
10.4 Simultaneous frame capture

Simultaneous image capture was possible only with Intel HM77 Chipset without any software modification or external use of hardware resources. Only single camera capture at a time was possible with PandaBoard ES and Gumstix COM. The figure 143 shows the Intel HM77 express chipset platform block diagram.
Figure 143: Intel HM77 Express Chipset Platform Block diagram
Chapter 11

Results and Data Analysis

The range of this thesis project spans from initial system design thinking to the analysis processed result from the realized system. However, the concrete result of this thesis project can be categorized in short such as *realized image stitching systems, optimized image stitching process, multifunction image stitching software, CLI based reconfiguration of OpenCV stitching module* and *an experimental test-bed implementation*. The first four parts are discussed in this chapter. The fifth part is discussed in the subsequent chapter. Moreover, in this chapter, we will analyze the system’s input and output in terms of histogram analysis and frequency domain power spectrum to validate the used algorithm’s robustness. Furthermore, the utmost refined homography matrix is the key to have the fastest response time as well as ultimate visual quality. We have recorded the response times with the best possible homography matrix we have generated. But as the algorithm is feature based, any camera system can be calibrated with more refined homography matrix which in turn will create speedier panorama generation system with improved visual quality.

11.1 Panning Stitching

Panning a camera means moving the camera horizontally left and right about a central axis. In this case, camera is mounted in a fixed location on a tripod or on a revolvable base. The panning stitching is a common phenomenon in panorama imaging or image stitching. Hence, we have also tested our algorithm in this setup to test the adaptability of the algorithm with different setup. The figure 144 shows the creative webcam used for panning stitching.

![Creative webcam](image)

Figure 144: Creative webcam
11.1.1 Input

Sample 1

Figure 145: Three input images (640x480)

11.1.2 Output

Sample 1

Figure 146: Panorama (without seam finder) from Gumstix COM (1248x483)

Figure 147: Seamless panorama (with seam finder) Gumstix COM (1448x483)
The figure 145 shows the input images and the figure 146 and 147 shows the output panorama image with two different stitching conditions.

11.2 Spherical MISUMI Analogue Video Camera System
A MISUMI spherical camera system used to capture the input images. The cameras are analogue video camera. The analog signal converted to digital signal by using frame grabber. Because of this process, capture time little bit higher with these three tiny-cameras in compare to built-in digital cameras. The figure 148 shows the MISUMI Camera system.

![MISUMI Camera System](image)

Figure 148: MISUMI Camera System

11.2.1 Stitching result with MISUMI camera

11.2.1.1 Input

Sample 1

![Three input images](image)

Figure 149: Three input images
The figure 149 and 150 shows the two samples of three input images. Moreover, the figure 151 and 152 shows the output panorama image from both samples computed by Gumstix COM.

Sample 2

Figure 150: Three input images

11.2.1.2 Output

Sample 1

Figure 151: Resulted panorama image from Gumstix COM
11.2.1.3 Video sequences

The captured frame data from this camera system directly stitched into panorama and saved the panorama in the computer. The system continuously captures images and generates panorama once initiated. The figure 153 shows the assembled spherical USB camera system.

11.3 Spherical USB camera system

Three USB cameras were also used to make a spherical camera system. The physical arrangements of the three cameras capture multiple-center-of-projection images. The motion model of the system was extracted and used for real-time stitching from this system.
11.3.1 Stitching result with USB camera system
The figure 154 shows three input images and the figure 155 shows the output panorama image.

11.3.1.1 Input
Sample 1

![Figure 154: Three input images*](image1)

*The above images are shrank than their original size

11.3.1.2 Output
Sample 1

![Figure 155: Resulted panoramic image** from PandaBoard](image2)

**The above result is shrank than its original size
11.3.1.3 Video sequences

The captured frame data from this camera system directly stitched into panorama and saved the panorama in the computer. The system continuously captures images and generates panorama once initiated.

11.4 Histogram analysis of input and output images

The histogram provides the distribution of distinct pixel values in an image. Thus, we can get an insight about the internal data property of the image. Hence, the histogram of input images and output images will provide an insight on the intensity range error rate of the stitching algorithm. The figure 156,157,158 shows the histogram of three input image 1, 2, 3 respectively. Moreover, the figure 159 shows the histogram of output panorama image.

![Histogram of the input image 1](image)

Figure 156: Histogram of input image 1
Figure 157: Histogram of input image 2

Figure 158: Histogram of input image 3
We can clearly understand the distinct pixel value’s distribution and intensity range from the histogram of three input images and output image. The input image 1 has pixel intensity range approximately within 25-215, the input image 2 has pixel intensity range approximately within 25-180, the input image 3 has pixel intensity range approximately within 25-240. The output panorama image has pixel intensity range approximately within 25-240, which indicates the robustness of our algorithm. The algorithm has purely preserved the pixel intensity value’s distribution in the panorama image from the input images.

11.5 Frequency domain analysis of input and output images
Every digital image is composed of numerous features. Hence, it is time consuming and practically error prone to investigate consistency between images by visual inspection. A visual inspection sometimes offers better results to find out correlation of visual representation but it takes bulky time to find out correlation among feature to feature of an image. To resolve this problem, we convert an image from spatial domain to frequency domain and from the magnitude of the numerous frequency components (i.e. features) we take clever decisions on the inherent properties of the image. Similarly, we have taken the power spectrum of the three input images and resulted output image. Subsequently, we have become able to find out total integrity inherent properties between input images and output image.
The different frequencies of features of an image are located in different directions and distances from the origin. We put the origin in the centre of power spectrum as a common practice and also for investigative convenience. The features with lower frequency will be located near the origin while the features with higher frequency will be located far from the origin. The different orientation of image features is represented by different directions from the origin. Hence, we can extract inherent properties if the image features very quickly and it helps to check inherent property consistency among images in a less error prone and flexible way. The figure 160, 161,162 shows the power spectrum of input image 1, 2, 3 respectively. Moreover, the figure 163 shows the power spectrum of output panorama image.

![Power spectrum of the input image 1](image)

Figure 160: Power spectrum of the input image 1
Figure 161: Power spectrum of the input image 2

Figure 162: Power spectrum of the input image 3
The power spectrum of the input images and output image shows that, the frequency and orientation of the features of the input images are satisfactorily preserved in the output panorama image.

Figure 163: Power spectrum of the output panorama image

11.6 Optimization in Gumstix by changing Advanced default CPU parameters

We have changed the advanced default CPU parameters of Gumstix COM. It has increased computing speed of the stitching algorithm by nearly 50%. The details of these parameters and corresponding computing speed enhancement is provided in the “Hardware Implementation” chapter. In this section, the optimization in Gumstix is demonstrated and compared by running stitching algorithm using seam finder and without seam finder along with warping and compositing. Moreover, total response time is also compared under mentioned conditions. These data plots provide a quick insight about stitching algorithm’s performance and rate of optimization. The figure 164, 165, 166 shows the bar graph of Gumtix optimization.
Figure 164: Gumstix optimization_Without seam finder

Figure 165: Gumstix optimization_With seam finder
11.7 A stand-alone GUI based software

The used software tools in this thesis project were distributed and had to be gathered and build separately. However, all the necessary software dependencies are integrated into one platform as a Qt framework based standalone software. The software includes the findings of this thesis as its core functionality. We have mentioned details about the software in system development chapter. The software has the following categorical functionalities:

1. GUI (i.e. Menus, Window, Label, Icons)
2. Real-Time stitching (i.e. real-time algorithm, test environment)
3. System performance display (i.e. algorithm process statistics, pie chart, system response time table data table)
4. Camera order determination (i.e. input image sequence check)
5. Save three images from the camera system.
6. An help file
11.8 A CLI based Reconfiguration of OpenCV stitching module

The CLI based modular software operates in Linux environment. We have mentioned details about the reconfiguration process in the system development chapter. The software has following functionalities.

1. Extraction of homography matrix from any number of images
2. Generation of real-time panorama with any number of images
3. It can work as a development tool for developing panorama imaging system with any number of cameras

11.9 Response time and real-time data analysis

We have generated panorama image using proposed algorithm both in Windows and Linux operating system. We have used laptop computer, small embedded computer Gumstix COM, PandaBoard ES. However, we have tried to achieve real-time performance under Linux operating system because we have been able to configure the operating system for real-time operation. The computer’s mentioned above were configured as follows-

1. Ångstrom Minimal Linux installed in Gumstix COM
2. Lighter Ubuntu Desktop Linux installed in PandaBoard ES
3. Ubuntu Desktop Linux installed in Laptop computer
4. Windows 7 installed in Laptop computer

The algorithm can work with different stitching parameters but each parameter has their own trade-off. We have tested the algorithm with different parameters. The recorded fastest response time of the proposed algorithm in the above platforms is provided in the following chart. To mention, all of the following results are not from the results that are provided in the “Hardware Implementation” chapter.

<table>
<thead>
<tr>
<th>Image Resolution</th>
<th>Number of Images</th>
<th>Stitching Condition</th>
<th>Hardware</th>
<th>Operating system</th>
<th>Fastest response time (Second)</th>
<th>Slowest response time (Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>3</td>
<td>Warping, Compositing</td>
<td>Laptop Computer</td>
<td>Windows</td>
<td>0.0626602</td>
<td>0.0656602</td>
</tr>
<tr>
<td>320x240</td>
<td>3</td>
<td>Warping, Compositing</td>
<td>Laptop computer</td>
<td>Ubuntu Linux</td>
<td>0.0954364</td>
<td>0.1014064</td>
</tr>
</tbody>
</table>
We can see that the fastest algorithm response was 62.6602 milliseconds in windows and 95.4364 milliseconds in Linux. In the following figures, we have plotted obtained data from all four platforms. The data plot of fastest system response, slowest system response and average system response provides an insight about the response time of the stitching algorithm under different hardware and software platforms. The figure 167,168,169 shows the comparison of fastest, slowest and average response time respectively in bar graphs.

<table>
<thead>
<tr>
<th>Platform Size</th>
<th>Warping, Compositing</th>
<th>Operating System</th>
<th>Ubuntu Linux (OMAP4)</th>
<th>2.9801</th>
<th>3.0320</th>
</tr>
</thead>
<tbody>
<tr>
<td>640x480</td>
<td>3</td>
<td>PandaBoard ES</td>
<td>Ubuntu Linux</td>
<td>2.9801</td>
<td>3.0320</td>
</tr>
<tr>
<td>640x480</td>
<td>3</td>
<td>Gumstix minimal Linux</td>
<td>Ångström minimal Linux</td>
<td>4.1937</td>
<td>4.40367</td>
</tr>
</tbody>
</table>

Figure 167: Fastest response time comparison
Figure 168: Slowest response time comparison

Figure 169: Average Response time comparison
The real-time implementation is only tested under Linux operating system but we are going to analyze the data in terms of real-time point of view for all operating systems that we have used. We have recorded samples from several execution instances of the stitching algorithm. The jitter is then calculated from these samples. We have first taken differences between two subsequent samples from a set of samples and then the differences are summed together. The sum of differences is then divided by number of differences to calculate jitter. This method is depicted graphically in the figure 170.

Figure 170: Calculation of Jitter

The occurred jitter lies within a range and this range is the jitter range. The slowest response time is the lower part of the jitter range and fastest response time is the upper part of the jitter range. We have plotted the fastest response time and slowest response time using obtained data in which jitter period is clearly visible.
In the figure 171, the length of green bar which is not encapsulated by green bar is the jitter period.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating system</th>
<th>Stitching condition</th>
<th>Jitter (Millisecond)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop (Toshiba)</td>
<td>Windows</td>
<td>Warping, Compositing</td>
<td>1.4637</td>
</tr>
<tr>
<td>Laptop (HP)</td>
<td>Ubuntu Linux</td>
<td>Warping, Compositing</td>
<td>1.942</td>
</tr>
<tr>
<td>PandaBoard ES</td>
<td>Ubuntu Linux</td>
<td>Warping, Compositing</td>
<td>11.78</td>
</tr>
<tr>
<td>Gumstix COM</td>
<td>Ängström Minimal Linux</td>
<td>Warping, Compositing</td>
<td>54.794</td>
</tr>
</tbody>
</table>

The calculated jitter is given in the above table and plotted with bar graph in the figure 172.
We have tested algorithm simultaneously in different platforms. Hence, we have observed different relational data's. These plotted data provides an understanding about the tested platforms performance and other relational attributes. The processing speed of CPU cores are directly related to system response time or computing performance. This phenomenon is also clearly visible in the obtained data from different platforms.
In the figure 173, we can clearly observe that, while CPU speed is decreasing response time increasing.

Figure 174: CPU Speed vs. Average Response time
In the figure 174, we have plotted a bar graph which clearly depicts that fastest response time is attained when we have higher CPU processing capacity. The figure 175 also provides an intuitive relation between CPU speed and response time.

![Graph Image]

Figure 175: CPU vs. Average Response time

11.11 Invariant Feature Verification

We have used invariant features which should be able to cope with images with variant luminosity. We have tested the algorithm against images that have variant luminosity in the overlapping areas as well as other areas to verify the robustness of the features.

![Input Images]

Figure 176: Input images with variant luminosity

In the figure 176, the tree has been captured with different luminosity. The brightness intensity of the tree is clearly different between first, third and middle image. However, as we have used invariant features, these differences have not created any difficulties for finding out correspondences among the images. The following resulted panorama from the images given
in the immediate previous figure demonstrates the power and robustness and benefit of using invariant features. The figure 177 shows the resulted panorama using invariant features.

Figure 177: Resulted Panorama using Invariant features
Chapter 12

Experimental Test-bed

Real-time panoramic imaging system has many industrial applications. It can used to investigate narrow passage, sewage, pipes or chimneys. Inspection of fire protection in the fireplaces and solid burning appliances and associated flues, and the kitchen flue using real-time video offers great flexibility and accuracy. There are numerous types of system that can capture real-time video but only a dedicated system can ensure high quality and accurate inspection.

The inspection company “Sotningsdistriktet i Karlskrona” needs an light inspection device that will offer a wide $180^\circ$ view of the inspection area using real-time panorama video. This inspection system will replace the existing mechanical solution to get wide view of the inspection area. Moreover, this inspection system will be multiple times lighter than the existing system and offers extra flexibility to the inspectors. The figure 178 shows the Chimney testbed.

![Figure 178: Chimney testbed](image)
The figure 179 shows the chimney door and top opening. Moreover, the figure 180 shows the targeted code written on the chimney door.

Figure 179: The Chimney door and top opening

Figure 180: Targeted code
The figure 181 shows the used micro spherical MISUMI Camera inside the chimney. As a test method it was designated to be able read a code from the dark wall of the above chimney. The code can be clearly read from the generated panorama by the spherical camera system. We had to find out a balanced point to ensure fast panoramic image creation and at the same time good quality panoramic image that makes the code clearly visible. The Spherical camera system with MISUMI cameras were used inside the chimney. The seams between the images are clearly visible in the above panoramic image generated because it was created without any seam removal and blending to facilitate faster processing. The panorama is with good quality which ensures clearly visible and readable code. The figure 182 and 183 shows two resulted panorama image from two inspection instances.

**Inspection Sample 1**

Figure 182: Code Inspection (Without seam removal and Blending)
Inspection Sample 2

Figure 183: Code inspection (Without seam removal and Blending)
Chapter 13

Summary and Conclusions

Extremely fast and super autonomous image stitching has good potential to become a powerful tool for next generation scientific and technological boom but how and why? To find out the answer we have to investigate ourselves. The attributes that enabled human to be a human are “soul and brain”, “vision and perception”, “ability to speak and listen”. The other human attributes such as movement, feeling pleasure, taste of food and so on all are getting values because of these three abilities given above. Vision and perception out of above three is the 2nd core ability that actually defines human’s ability to flexibly perceive knowledge and contributing to human civilization.

In this modern age, we have created our new artificial world of working systems. These systems do not care about taste or pleasure of vision as they do not have life but still we need them with more intelligence to make our life easier. We have provided them “controller” which is safe and good enough to obtain gain from them. However, we have also given them vision via algorithmic computer vision and speaking and listening via algorithmic speaking and voice recognition but merely with poor quality. Hence, for next generation artificial intelligent systems, we should be able to provide them better perceptual flexibility as much as possible like us. Image stitching is a technology that offers larger scale surrounding vision using smaller scale hardware. Thus, a super autonomous and extremely faster image registration method will upgrade the perception capability of the self-directed mobile systems. Hence, Image stitching has huge potential to become a powerful tool that can help us for constructing next generation expert systems. These systems may range from super artificially intelligent humanoid robots, industrial robots and embedded systems and so on. Specially, the humanoid robot with its two eye resembled camera can use image stitching for surrounding vision instead of unnaturally placing hardware solutions like a distortion producing fisheye lens above the head or placing multiple camera around the head.

In this thesis, we have tried to depict an overall description of non-proprietary existing open source technologies that can be used for the prototype development of above ambition. In this paper, we have covered mathematical theories and methods, operating system and software libraries, software development tools, algorithms, computing platform from a semi-high level and semi-low level point of view. Because, it is not timely feasible to write all low level attempts that were made that go beyond the thesis scope. The interesting parties can dive into low level to explore the possibilities for developing further optimized system by following the mentioned items. One of the important obstacles in this thesis project was hardware constraint. The hand-held computer Gumstix and PandaBoard ES cannot capture
from three cameras simultaneously. The solution of this problem can remove a development barrier from the way which can be achieved via hardware solution or software solution. A short summary of the thesis progression is depicted in the figure 184.

![Sequential thesis steps](image)

**Figure 184: Sequential thesis steps**

### 13.1 Future work

One of the ambitious and continues goal of this project can be very shorter system response time. A standard to reach this goal can be 10 milliseconds system response time for both image capturing and panorama generation. However, both hardware and software optimization are required to reach that goal. Here, I am proposing some future steps that can be taken to reach the goal mentioned above. I will disregard the fact of economic feasibility because it depends on the volume of production and application areas market demand. I have provided a list of bibliography in this paper which may help another researcher for further improvement.
13.1.1 Development of image stitching chip

The word real-time has its own intuitive depth. In a true sense, a real-time system must need to have real-time characteristics in all of its operation. Hence, hundred percent real-time panoramic imaging is possible only and only if all system hardware and system software has the ability to operate in real-time. Thus, we need to go low level hardware design and software design to build a fully dedicated image stitching chip. Fortunately, we have a lot of good algorithm to construct panorama. Having said that, I meant, we have a known set of computational execution flow for generating larger image mosaics. This known set of data processing sequence can be used to design a fully dedicated image stitching chip. But unfortunately, the availability of dedicated hardware that was purposefully designed for real-time image stitching is not seems visible on the market. There are may be proprietary systems available but that must need to be disregarded as we cannot study them. In my view, image stitching is highly used in various fields and hence it has good market share in the imaging based industry. So, the development of such dedicated image stitching chip will not be a bad investment. The book [3] can be a good beginning point to get some nitty-gritty ideas of the relevant approach in this regard.

13.1.2 Stitching Algorithm optimization

The image stitching process can be optimized by finding out more robust invariant features and optimized algorithm in general. However, we can also think outside of the box which often changes the computation paradigm. We can say that because the whole stitching algorithm is a mathematical model to process the data. And we know that, mathematics is all about ideas to perceive data and process data. There is always a room left for optimization and that is also true for mathematics. Hence, new mathematical model to realize the stitching algorithm could possibly revolutionize the image stitching process by designing faster algorithm. Moreover, optimal homography matrix can be a general approach to find out the best trade-off between panorama generation time and visual quality in an optimized way.

13.1.3 More powerful handheld computer for smaller systems

The perfect design of hardware and software is principally necessary for real-time systems rather than more and more powerful computers. However, powerful computers always improve the performance in some scale. In this thesis project, we have used OMAP3 based Gumstix Overo with 720 MHz processor speed and OMAP4 based PandaBoard ES with 2x1.2 GHz processor speed. However, a substantially more powerful OMAP5 now came to the market. The OMAP5 has following features-

- Multi-core ARM® Cortex™ processors
- Two ARM Cortex-A15 MPCore processors capable of speeds up to 2 GHz each
• Two ARM Cortex-M4 processors for low-power offload and real-time responsiveness
• Multi-core POWERVR™ SGX544-MPx graphics accelerators drive 3D gaming and 3D user interfaces
• Dedicated TI 2D BitBlt graphics accelerator
• IVA-HD hardware accelerators enable full HD, multi-standard video encode/decode as well as stereoscopic 3D (S3D)
• Faster, higher-quality image and video capture

13.1.4 Fully embedded operating system
A lighter approach than development of an image stitching chip can be fully embedded operating system. Having said that, I meant, the operating system will have only one application layer application process and that is image stitching.

13.1.5 Development of Application specific-integrated circuit (ASIC)

13.1.6 Parallelization of Stitching Algorithm using SMP
An application has a single process. A process can have one thread or several threads. Currently, most of the computing systems have multicore CPUs. More than one thread in an application process means it can exploit processing capacity of all CPU cores. To be able to use more threads, we have to parallelize computing sequence of the algorithm. But parallelizing computing sequence can be inconsistent with data integrity or not. A high level multithreading was tested in this thesis using QT’s thread class “QThread” but it did not produced satisfactory result. However, parallelizing the computation at the very low level of the algorithm possibly can optimize the response time by exploiting processing capacity of all CPU cores while keeping data integrity.

The computing systems that contain more than one processor are termed as Multiprocessor systems. The PandaBoard ES is a multiprocessor system. It uses symmetric multiprocessing (SMP) architecture which can be exploited to optimize the image stitching algorithm’s response time.

13.1.7 Development project using a single brain
It will not be wrong if we say we are simply brains not human. It is actually brain that defines who we are. The fat elephant, sperm whales are superior than us in size but not in intellect. This few kilograms of brain enabled us to interact and develop, to launch space missions to space while big elephant and sperm whales are still surfing in land and water for finding out daily meal respectively. I have just emphasized on the importance of brain. The large development projects are coordinated by several brains which provide benefits of specialized knowledge. However, there are disadvantages of work division among brains. The work division provide every brain a partial view of the problem not an in depth insight.
Subsequently, we cannot achieve ultimate optimization. The truth is only a single brain with encompassing knowledge in an issue can provide the ultimate optimized solution. This is because we humans cannot interact like computers. Every human brain has a unique experience but every human brain has limitation of medium of expression. We humans interact with each other by the means of sound expression, body language, facial expression and writing. It means that we cannot share our brain with another brain fully which is a constraint. I want to term this constraint as “Human brain constraint (HBC)”. This “Human brain constraint (HBC)” disables us to obtain best possible solutions. Having said all above I want to argue that to develop an extremely faster and autonomous stitching algorithm a single brain must need to have vast command of knowledge at philosophical level as well nitty-gritty low level implementation knowledge about numerous theories, applied mathematics, software and hardware. The bibliography section can be helpful for any interested researchers.
References


[22] Zhu Daixian, “SIFT Algorithm Analysis and Optimization”, Xi’an University of Science and Technology, Xi’an, China.


[34] “MITRE Immersive Vision System (MIVS)”, February 27 2009, <http://www.youtube.com/watch?index=0&feature=PlayList&v=811M9TDSHOQ&list=PL84E18196AACAB7C>


[56] “Theorems of Desargues and Pappus”, < http://math.sjtu.edu.cn/course/gdds%E6%95%99%E5%AD%A6%E8%AF%BE%E4%BB%B6/Hexagon%20Theorem%20of%20PappusPascal.pdf>


Bibliography

This Bibliography is provided as a part of future work section which may help another student as a quick source of information for any further works that relates Image Stitching, Embedded Linux, Real-time OS, OpenMP, SMP and Embedded Vision Systems. Hence, the books and research papers that were not studied in the thesis also included in the bibliography beside the references.


A. Downton and D. Crookes, “Parallel architectures for image processing”.


Akiyoshi Wakatani, “Parallel implementation of DPCM decoding for SMP systems”, Faculty of Science and Engineering, Konan University, 8-9-1, Okamoto, Higashinada, Kobe, 658-8501, JAPAN, Sixth International Conference on Software Engineering Research, Management and Applications.

Ayon Basumallik, Seung-Jai Min, Rudolf Eigenmann, “Programming Distributed Memory Sytems Using OpenMP”, Purdue University,School of Electrical and Computer Engineering,West Lafayette, IN 47907-1285 USA.


Berthold K.P. Horn, “Tsai’s camera calibration method revisited”.
Barbara Chapman*, Lei Huang*, Eric Biscondi**, Eric Stotzer**, Ashish Shrivastava** and Alan Gatherer**, “Implementing OpenMP on a high performance embedded multicore MPSoC”, * University of Houston, Houston, TX, USA. ** Texas Instruments, Houston, TX, USA.


Bo Youn Kim, Kyung Ho Jang, Soon Ki Jung, “Adaptive Strip Compression for Panorama Video Streaming”, VR Lab. Department of Computer Engineering, Kyungpook National University, South Korea.


Beom Su Kim, Hyung Il Koo, Nam Ik Cho, “A NEW IMAGE PROJECTION METHOD FOR PANORAMIC IMAGE STITCHING”, School of Electrical Engineering and Computer Sciences, Seoul National University, Institute of New Media & Communication, San 56-1, Shilim-Dong, Gwanak-Gu, Seoul, Korea.

Bo Han, and Xinggang Lin, “A Novel Hybrid Color Registration Algorithm for Image Stitching Applications”.


Bo Han, and Xinggang Lin, Senior Member, IEEE, “A Novel Hybrid Color Registration Algorithm for Image Stitching Applications”.


Chao Liu, Susumu Shibusawa, Tatsuhiro Yonekura, “A Walkthrough System with Improved Map Projection Panoramas from Omni Directional Images”, Ibaraki University, Japan.

Cheolwhan Lee, Tao Yang Yuan-Fang Wang, “Partitioning and Scheduling for Parallel Image Processing Operations”, Department of Computer Science, University of California, Santa Barbara, CA 93106.

Cristina Nicolescu and Pieter Jonker, “A Data and Task Parallel Image Processing Environment for Distributed Memory Systems”, Delft University of Technology, Faculty of Applied Physics, Pattern Recognition Group, Lorentzweg 1, 2628CJ Delft, The Netherlands.


Carlos Böke, Marcelo Götz, Tales Heimfarth, Dania El Kebbe, Franz J. Rammig, Sabina Rips, “(Re-) Configurable Real-Time Operating Systems and Their Applications”, Heinz Nixdorf Institut (HNI), Universität Paderborn.


Derek Bradley, Alan Brunton, Mark Fiala, Gerhard Roth, “Image-based Navigation in Real Environments Using Panoramas”, SITE, University of Ottawa, Canada Institute for Information Technology, National Research Council of Canada.

David Clark, “OpenMP: a parallel standard for the masses”.

David G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints”, Computer Science Department, University of British Columbia, Vancouver, B.C., Canada.

Dr. Phillip A. Laplante, “Real-Time Image and Signal Processing”, Selected Reading, IEEE.

D.Gansawat, W.Sinthupinyo, “Panorama Images Stitching on Android Phones”, Image Technology Laboratory, National Electronic and Computer Technology Center 112 Phahonyothin Road, Klong Luang, Pathumthani 12120, Thailand.

Denan Li, Zhiyun Xiao, “Design of Embedded Video Capture System Based on ARM9”, Electric power college, Inner Mongolia University of Technology, Hohhot, China.

David Johnston, Martin Fleury, Andy Downton, “Multi-Paradigm Framework for Parallel Image Processing”, University of Essex, Multimedia Architectures Laboratory, Colchester, Essex, CO4 3SQ, UK.


E.Ramraj*, A.Senthil Rajan**, “Using Multi-core Processor to support Network Parallel Image Processing Applications”, *Director, Computer Department, Alagappa University, Karaikudi, India. **Head of the Department, Jyoti Nivas College, Bangalore, India.


Frank Nielsen1, Alexis Andr´e1,2, and Shigeru Tajima1, “Real-Time Spherical Videos from a Fast Rotating Camera”, 1 Sony Computer Science Laboratories, Inc., 3-14-13 Higashi Gotanda, 141-0022 Shinagawa, Tokyo Japan, 2 Tokyo Institute of Technology, 2-12-1 Oookayama, 152-8552 Meguro, Tokyo Japan.


F.J. Seinstra and D. Koelma, “Transparent Parallel Image Processing by way of a Familiar Sequential API”, Intelligent Sensory Information Systems, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands.


Feng Xu, College of Computer & Information, Hohai University, Department of Computer Science & Technology, Nanjing University, Nanjing, China. Wentao Wang, College of Computer & Information, Hohai University, Nanjing, China. “Design of Flood Control and Video Surveillance System of Water Resources”.

Fu-Shin Lee Rou-Jiun Shiu Ming-Da Chiang, “Development of an Embedded Controller for BioCD Inspections”, Mechatronical Engineering Department, Huafan University, Huafan University, 223, Shihhtin, Taipei County, TAIWAN.

GORDON BELL AND CATHARINE VAN INGEN, “DSM Perspective: Another Point of View”.

Guangqi Hou, Ping Wei, “A Panorama-Based Architecture for Multimedia from Hand-Held Devices”, School of Information Science and Technology, Beijing Institute of Technology, Beijing, 100081, China.


Guang-Quan KANG, “Image stitching based on motion estimation”, Electronic Information Engineering School, Sanjiang University, Nanjing, P.R.China. 2010 Third International Conference on Information and Computing.

Gökhan Bas., Güleser Kalaycı Demir, “Sinc Interpolating Approach to Sub-Pixel Image Stitching for Virtual Microscopy”, Department of Electrical and Electronics Engineering, Dokuz Eylül University, Buca, Izmir, TR35160 Turkey.


Gary Bradski, Adrian Kaehler, “Learning OpenCV- Computer Vision with the OpenCV Library”, O’reilly.


Hideaki Kimata, Shinya Shimizu, Yutaka Kunita, Megumi Isogai, and Yoshimitsu Ohtani, “Panorama video coding for user-driven interactive video application”, NTT Cyber Space Laboratories, NTT Corporation Yokosuka, Japan.

Hong Zhang, Yuecheng Li, Steven A. Hackworth, Yaofeng Yue, Chengliu Li, Guifang Yan, Mingui Sun, “The design and realization of a wearable embedded device for dietary and physical activity monitoring”.

Harshad B. Prajapati*, Dr. Sanjay K. Vij**, “Analytical Study of Parallel and Distributed Image Processing”, * Information Technology Department, Dharmsinh Desai University, Nadiad-387001, Gujarat, INDIA. ** Director, Sardar Vallabhbhai Patel Institute of Technology, Sardar Vallabhbhai Patel Institute of Technology, Vasad-388306, Gujarat, INDIA.

Hyung Il Koo, Beom Su Kim, and Nam Ik Cho, “A NEW METHOD TO FIND AN OPTIMALWARPING FUNCTION IN IMAGE STITCHING”, School of Electrical Engineering, Seoul National University, Seoul 151-744 Korea.
Hyung Il Koo, Nam Ik Cho, Member, IEEE, “Feature-based Image Registration Algorithm for Image Stitching Applications on Mobile Devices”.


Hyung Il Koo, Beom Su Kim, and Nam Ik Cho, “A NEW METHOD TO FIND AN OPTIMAL WARPING FUNCTION IN IMAGE STITCHING”, School of Electrical Engineering, Seoul National University, Seoul 151-744 Korea.


Hui Xiaowei*, Shen Qinglei**, Miao Changyun***, “Acquisition Board Design of High-speed Image Data Based on ARM and FPGA”, *Academy of Electronics and Information Engineering, Liaoning Technology University, **Liaoning Technology University, ***Tianjin Polytechnic University. Huludao Liaoning China, Tianjin China.


Helga Kolb, “How the retina works”, A reprint from American Scientist, the magazine of Sigma Xi, The Scientific Research Society.

Huilin Zhang*, Huifeng Xue*, Wei Wang**, “System design of image gathering and saving based on embedded technology”, *College of Automation, Northwestern Polytechnical University, Xi’an 710072, China. **College of Power and Energy, Northwestern Polytechnical University, Xi’an 710072, China.

Ishfaq Ahmad, “Gigantic clusters: Where are they and what are they doing?”, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.


Jing Xing, Zhenjiang Miao, “An Improved Algorithm on Image Stitching based on SIFT features”, Institute of Information Science, Beijing Jiaotong University, Beijing 100044, P.R. China.


Johannes Weber, Andreas Rehkopf, “Description of a formalized approach to build reproducible Linux images for a project-specific electronics platform”, TU Bergakademie Freiberg, Institut für Automatisierungstechnik, 09596 Freiberg, Germany.

JIAO Guo-sheng, QIN Hui-bin, “Design of Remote Image Monitoring System Based on Embedded Linux and OMAP3530”, College of Electronic information, Hangzhou Dianzi University, Hangzhou, China.


Jeffrey C. Gealow, Student Member, IEEE, Frederick P. Herrmann, Member, IEEE, Lawrence T. Hsu, Membrtr, IEEE, and Charles G. Sodini, Fellow, IEEE, “System Design for Pixel-Parallel Image processing”, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL 4, NO. 1, MARCH 1996.
Jaafar M.H. Elmirghani, “Optical computing techniques for parallel image processing”, Department of Electrical and Electronic Engineering, The Manchester Metropolitan University, Chester street, Manchester, M1 SGD, UK.


Jing Li Weidong Hao, “Research and Design of Embedded Network Video Monitoring System Based on Linux”, School of Mechanical and Electrical Engineering, Guilin, University of Electronic Technology, GUET, Guilin, P.R.China. 2008 International Conference on Computer Science and Software Engineering.

Jonathan M. R. Byrd, Stephen A. Jarvis and Abhir H. Bhalerao, “Reducing the Run-time of MCMC Programs by Multithreading on SMP Architectures”, University of Warwick, Department of Computer Science, Coventry, CV4 7AL, UK.


Keil Thomas, “Introducing the bash shell”, Beginning Ubuntu Linux-From Novice to Professional, Apress.
K.N. Ngan, PhD, A.A. Kassim, Beng, H.S. Singh, PhD. “Parallel image-processing system based on the TMS32010 digital signal processor”.

Kichul Kim and V. K. Prasanna Kumar, “Parallel Memory Systems for Image Processing”, Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089-0781.


Karthik Sankaranarayanan, James W. Davis, “A Fast Linear Registration Framework for Multi-Camera GIS Coordination”, Dept. of Computer Science and Engineering, Ohio State University, Columbus, U.S.A


Lionel Robinault, Stéphane Bres, Serge Miguet, “Panoramic mosaicing optimization”, LIRIS - Lyon2 University, France.

Liu Yang, Heming Pang, Linying Jiang, Kun Yue, “The Research of Embedded Linux and SQLite Database Application in the Intelligent Monitoring System”, Software College, Northeastern University, Shenyang 110004, China.


L Sousa 2,3, J Barrios 2,3, A Costa 1,3, M Piedade 1,3, “PARALLEL IMAGE PROCESSING FOR TRANSPUTERS BASED SYSTEMS”, 1 – IST, UTL, Portugal. 2 - FCT, UNL, Portugal. 3 - INESC and CEAUTL (INIC), Portugal.


Li Bing*, Sun JianPing**, “Remote Video Monitoring System Based on Embedded Linux and GPRS”, *Science & Technology college, North China Electric Power University, BaoDing, China. **School of Control Science and Engineering, North China Electric Power University, BaoDing, China.

LI Da-zhai, LI Yu-xia, “Design of Embedded Image Data Processing System”, School of Mechanical Engineering and Automation, BeiHang University, BeiJing, China, 100191.

Mai Zheng, Xiaolin Chen, and Li Guo, “Stitching Video from Webcams”, University of Science and Technology of China.


Mitsuhisa Sato, “OpenMP: Parallel programming API for shared memory multi processors and on-chip multiprocessors”, University of Tsukuba, 1-1-1 Tenou-dai,Tsukuba, Ibaraki 305-8577 Japan.

Ms. Durga Patidar*, Mr. Akshay Jain**, “Automatic Image Mosaicing: An Approach Based on FFT”, *Department of Electronics & communication Engineering, SSSIST Sehore, Madhay Pradesh (INDIA), ** Department of Electronics & communication Engineering, SPITM Mandleshwer, Madhya Pradesh (INDIA).


Michael S. Moore, Gabor Karsai, and Janos Sztipanovits, “MODEL-BASED PROGRAMMING FOR PARALLEL IMAGE PROCESSING”, Vanderbilt University, Measurement and Computing Systems Laboratory, Department of Electrical Engineering, Nashville, Tn. 37215.


M. Trakaa,a,b, G. Tziritasa,a,b,*, “Panoramic view construction”, a Institute of Computer Science—FORTH, Heraklion, Greece, b Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, Greece.

Michael Farrell, Benjamin Walthall, Avinay Vaswani, Francis. G. Thompson, Michael D. DeVore, “Rapidly Deployable Distributed Video Surveillance System for Resource Constrained Applications”.

Mohammad Javad Fadaeieslam, Mahmood Fathy, Mohsen Soryani, “Key Frames Selection into Panoramic Mosaics”, Department of computer engineering, Iran University of Science & Technology, Tehran, Iran.


Nikolaos Drosinos and Nectarios Koziris, “Performance Comparison of Pure MPI vs Hybrid MPI-OpenMP Parallelization Models on SMP Clusters”, National Technical University of
Athens, School of Electrical and Computer Engineering, Computing Systems Laboratory, Zografou Campus, Zografou 15773, Athens, Greece.


Oscar Hernandez*, Ramachandra C Nanjegowda* and Barbara Chapman*, Van Bui**, Richard Kufrin***, “Open Source Software Support for the OpenMP Runtime API for Profiling”, *Dept. Computer Science, University of Houston, Houston, USA, **Mathematics and Computer Science Division, Argonne National Laboratories City, USA, ***NCSA, University of Illinois at Urbana-Champaign City, USA. 2009 International Conference on Parallel Processing Workshops.


Philip J Morrow*, Danny Crookes**, “USING PROLOG TO IMPLEMENT A COMPILER FOR A PARALLEL IMAGE PROCESSING LANGUAGE”, *School of Information and Software Engineering, University of Ulster, Coleraine, BT52 1 SA, N. Ireland. **Department of Computer Science, The Queen's University of Belfast, Belfast, BT7 INN, N. Ireland.

Qichang Duan, Yi Tang, Congli Zhang, “Design and Realization of Network Video Server Based on Single DSP”, Department of automation, Chongqing University, Chongqing, 400044, China.


Qian Xia*, Yong Zhang**, “A Robust Algorithm of Constructing Panorama”, *College of media and communications Guizhou University for Nationalities, Guiyang, China. **College of science and technology, Guizhou University, Guiyang, China. 2010 International Conference on Networking and Digital Society.


Seong Jong Ha, Hyung Il Koo, Sang Hwa Lee, Nam Ik Cho, Soo Kyun Kim, “Panorama Mosaic Optimization for Mobile Camera Systems”.


M Maddalena 12, 38100 Trento, Italy, ** University of Kent at Canterbury, Canterbury, Kent, CT2 7NT, UK.


Shimin Wang, Jihua Ye, “Research and Implementation of Embedded Face Recognition System Based on ARM9”, College of Computer Information Engineering, Jiangxi Normal University, Nanchang, China.


Simone Leorin, Luca Lucchese, Oregon State University, Ross G. Cutler, Microsoft Research, “Quality Assessment of Panorama Video for Videoconferencing Applications”.


Sergey V. Nikitin, Igor S. Gruzman, “Image Stitching Under Projective and Nonlinear Distortion”, Novosibirsk State Technical University, Novosibirsk, Russia. 8th INTERNATIONAL SIBERIAN WORKSHOP AND TUTORIALS EDM’2007, SESSION IV, JULY 1-5, ERLAGOL.


Sumantra Dasgupta, Amarnath Banerjee, “A REAL-TIME PANORAMIC VISION SYSTEM FOR AUTONOMOUS NAVIGATION”, Department of Industrial Engineering, 3131 TAMU, Texas A&M University, College Station, TX 77843 U.S.A. Proceedings of the 2004 Winter Simulation Conference.

Seong Jong Ha, Sang Hwa Lee, Yu Ri Ahn, and Nam Ik Cho, “PANORAMIC MOSAIC SYSTEM FOR MOBILE DEVICES”, School of Electrical Engineering and Computer Science, INMC, Seoul National University, Korea.


S. J. Ha et al., “Panorama Mosaic Optimization for Mobile Camera Systems”, IEEE

Seong Jong Ha, Hyung Il Koo, Sang Hwa Lee, Nam Ik Cho, Soo Kyun Kim, Member, IEEE, “Panorama Mosaic Optimization for Mobile Camera Systems”.

Shigang Li, “Full-View Spherical Image Camera”, Faculty of Engineering, Iwate University, Japan.


Takeaki Sugimura, JeoungChill Shim, Hiroyuki Kurino, member, IEEE, and Mitsumasa Koyanagi, Fellow, IEEE, “Parallel Image Processing Field Programmable Gate Array for Real Time Image Processing System”.

Trevor Hobbs, Brendan Boesen, Ickseo Kim, Colin Vance, Don Fraser, “A Parallel Image Processing And Display System (PIPADS) Hardware Architecture and Control Software”, Department of Electrical Engineering, University of New South Wales University College, Australian Defence Academy, Canberra, ACT.

T. Yamamoto and M. Doi, “PanoVi: PANORAMIC MOVIE SYSTEM FOR REAL-TIME NETWORK TRANSMISSION”, Department of Electronics and Information Engineering, Graduate School of Engineering, Hokkaido University, Sapporo, Japan.

Thomas Pintaric, Ulrich Neumann & Albert Rizzo, “Immersive Panoramic Video”, Integrated Media Systems Center, University of Southern California, Los Angeles CA 90089


Tu Xionggang, Chen Jun, “Parallel image processing with OpenMP”, School of Computer Science Zhejiang Industry Polytechnic, College Shaoxing, Zhejiang, China.


Vladan Rankov, Rosalind J. Locke, Richard J. Edens, Paul R. Barber and Borivoj Vojnovic, “An algorithm for image stitching and blending”, Advanced Technology Development Group, Gray Cancer Institute, Mount Vernon Hospital, Northwood, Middlesex, HA6 2JR, United Kingdom.


Wai Chong Chia, Li-Minn Ang, and Kah Phooi Seng, “Performance Evaluation of Feature Detection in using Subsampled Images for Image Stitching”, Department of Electrical and Electronic Engineering, The University of Nottingham Malaysia Campus, Jalan Broga, 43500, Semenyih, Selangor, Malaysia.

Wang Jisen, Zhao Rubin, “The Design of Image Acquisition System Based On ARM”, School of Automation, Northwestern Polytechnical University, Xi’an, China.

WINSER E. ALEXANDER, MEMBER, IEEE, DOUGLAS S. REEVES, AND CLAY S. GLOSTER JR., “Parallel Image Processing with the Block Data Parallel Architecture”.


Xiaoming Deng, Fuchao Wu, Yihong Wu, and Chongwei Wan, “Automatic Spherical Panorama Generation with Two Fisheye Images”, National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, P. O. Box 2728, Beijing 100190, P. R. China.

Xianyong Fang, “Feature Based Stitching of a Clear/Blurred Image Pair”, MOE Key Lab. of Intelligent Computing and Signal Processing, Anhui University, Hefei, China.

Xueyan Hu, Zhonglin Xia, Zhiyong Yuan, “Study on Image Deformation Simulation based on ARM Linux and OpenGL ES”, School of Computer Science, Wuhan University, Wuhan Hubei 430072, China.


Yining Deng* and Tong Zhang, “Generating Panorama Photos”, Hewlett-Packard Labs, 1501 Page Mill Road, Palo Alto, CA 94304

Yihong Gong, Guido Proietti, David LaRose, “A Robust Image Mosaicing Technique Capable of Creating Integrated Panoramas”, Robotics Institute, Carnegie Mellon University, Pittsburgh, U.S.A

Yang Linhong and Masahito Hirakawa, “A Stitching Algorithm of Still Pictures with Camera Translation”, Graduate School of Engineering, Hiroshima University, Japan.

Yingen Xiong and Kari Pulli, “Sequential Image Stitching for Mobile Panoramas”, Nokia Research Center, Palo Alto, CA 94304, USA.

Yu Tang, Huiyan Jiang, “Highly Efficient Image Stitching Based on Energy Map”, Multimedia Medical Information Technology Laboratory, Software College, Northeastern University, Shenyang 110004, China.

Yoshikizu YABE, Toliu KIMURA, Yoshilicitu AIMOTO, Hitlkri FIEIUCHI, Ynietsu NAKAZ4 WA, Trrki.iyi KOCA, Yo.,diiiiiro FUJITA, Mmqvki HAMADA, Tcrkuho
TANICA WA, Huitne NOBUSWA WA, and Kiitiiici KOYAMA, “Control and Synchronization Scheme for Parallel Image Processing RAM with 128 Processor Elements and 16-Mb DRAM”, NEC Corporation, 1 120 Simokumwa, Sagaini ham, Kanagawn, 229, Japan.

Yen-Lin Chen*, Chuan-Y en Chiang**, “Embedded On-road Nighttime Vehicle Detection and Tracking System for Driver Assistance”, *Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei, Taiwan. **Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan.


Yi-Yuan Chen 1, Yuan-Yao Tut 2, Cheng-Hsiang Chiul 3, and Yong-Sheng Chent 4, “An Embedded System for Vehicle Surrounding Monitoring”, 1 Industrial Technology Research Institute, Chutung City, Hsinchu, Taiwan, 2,3,4 National Chiao Tung University, Hsinchu, Taiwan.

Yu Fanhua*, Wu Dengfeng**, Zhao Hongwei*, Zhen Zhiqiang*, “Video image Acquisition System based on embedded Linux”, *College of Computer Science and Technology, Jilin University, Changchun, China. **College of Computer Science and Technology, Changchun Normal University, Changchun, 130012, China.

Yingen Xiong and Kari Pulli, “Fast and High-Quality Image Blending on Mobile Phones”, Nokia Research Center, 955 Page Mill Road, Palo Alto, CA 94304 USA.

Yingen Xiong and Kari Pulli, “Fast Image Stitching and Editing for Panorama Painting on Mobile Phones”, Nokia Research Center, 955 Page Mill Road, Palo Alto, CA 94304, USA.

Yihua Lan, Haozheng Ren, Cunhua Li, Xuefeng Zhao, “School of Computer Engineering Huaihai Institute of Technology, Lianyungang, China”. Zhifang Min, Huazhong Institute of Optoelectronics, Wuhan, China. “Feature based sequence image stitching method”.

Yao Li, Lizhuang Ma, “A Fast and Robust Image stitching Algorithm”, Department of Computer Science, Shanghai Jiao Tong University, Shanghai, China. Proceedings of the 6th World Congress on Intelligent Control and Automation, June 21 -23, 2006, Dalian, China.

Y. Charlie Hu*, Honghui Lu**, Alan L. Cox* and Willy Zwaenepoel*, “OpenMP for Networks of SMPs”, *Department of Computer Science, **Department of Electrical and Computer Engineering, Rice University, Houston, Texas 77005.


Yun He and Chris H.Q. Ding, “MPI and OpenMP Paradigms on Cluster of SMP Architectures: the Vacancy Tracking Algorithm for Multi-Dimensional Array”
Transposition”, NERSC Division, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA 94720.

Yingen Xiong and Kari Pulli, Member, IEEE, “Fast Panorama Stitching on Mobile Devices”.

Yong-lin LIU(1,2), Ying LIANG(1,3), Yao-yu CHENG(1,2), “Design and Implementation of Image Acquisition System Based on ARM and Linux”. 1.College of Information and Communication Engineering, North University of China, Taiyuan Shanxi,China. 2.Modern Nondestructive Testing Engineering Technology Research Center(North University of China), Shanxi,Taiyuan Shanxi,China. 3.Engineering Technology Research Center of Opto-Electronic Information and Instrument(North University of China), Shanxi,Taiyuan Shanxi, China.


Yang-Suk Kee, “OpenMP extension to SMP clusters”, IEEE.

Yanfang Li, Yaming Wang, Wenqing Huang, Zuoli Zhang, “Automatic Image Stitching Using SIFT”, Zhejiang Science and Technology University, Model Recognition Library.

ZOU Xiao-chun, HE Ming-yi, ZHAO Xin-bo, FENG Yan, “A Robust Mosaic Panorama Technique for Video”, Northwestern Polytechnical University, China.

ZHAO Xin-bo, ZOU Xiao-chun, MA Zhong, FENG Yan, “An Efficient Video Mosaic Panorama Algorithm”, Northwestern Polytechnical University, China.


Zhifeng Xiao, Binglong Zhang, “Parallel Image Processing Based on Pipeline”, State Key Laboratory of Information Engineering in Surveying, Mapping, and Remote Sensing, 129 Luoyu Road, Wuhan, 430079, China.

Zhengxi Wei, Lecai Cai, “Design for Motion Detection System Based on Embedded Linux”, School of Computer Science, Sichuan, University of Science & Engineering, Zigong, Sichuan 643000, PR China.
Zhao Ji-chun, Sun Su-fen Yu Feng Luan Ru-peng Zhang Wei, “The Design and Realization of Embedded Wireless Video monitoring System Based on GPRS”, Beijing Academy of Agriculture and Forestry Sciences, Institute of Information on Agricultural Science and Technology, Beijing, China, 100097.

Zhongdong Hu, Xiaoli Chen, “Design of Embedded Wireless Video Surveillance System Based on Internet”, School of Information Engineering, Jiangxi University of Science and Technology, Ganzhou, P.R.China.

Zhu Daixian, “SIFT Algorithm Analysis and Optimization”, Xi’an University of Science and Technology, Xi’an, China


Zhang Yong-qiang Zhao Ji-chun Wu ning, “Design and Realization of Embedded Image Transmission System Based on GPRS”, Hebei University of Engineering, Hebei Handan, China, 056038.

Appendix A

Errors

A substantial number of errors occurred during the development of the system. All the appeared problems were solved to get desired output of the project. Most of the errors were not saved after solving it. A few of the software errors are given below-

1. Figure: Stitching error
2.

Figure: Stitching error

3.

text after opencv make error

[ 1%] Built target pch_Generate_opencv_imgproc
[ 1%] Built target opencv_core_pch_dephelp
[ 2%] Built target pch_Generate_opencv_core
[ 6%] Built target opencv_core
[ 14%] Built target opencv_imgproc
[ 14%] Built target opencv_calib3d_pch_dephelp
[ 15%] Built target pch_Generate_opencv_calib3d
[ 15%] Built target opencv_features2d_pch_dephelp
[ 15%] Built target pch_Generate_opencv_features2d
[ 15%] Building CXX object
modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/opencv_flann_pch_dephelp.o
In file included from /media/mmcblk0p2/OpenCV-2.3.1/modules/flann/src/precomp.hpp:9,
   from /media/mmcblk0p2/OpenCV-2.3.1/mainhasan/modules/flann/opencv_flann_pch_dephelp.cxx:1:
   /media/mmcblk0p2/OpenCV-2.3.1/modules/flann/include/opencv2/flann/dist.h: In function 'T cvflann::abs(T) [with T = long double]':
   /media/mmcblk0p2/OpenCV-2.3.1/modules/flann/include/opencv2/flann/dist.h:63: error: 'fabsl' was not declared in this scope
make[2]: *** 
[modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/opencv_flann_pch_dephelp.o] Error 1
make[1]: *** 
[modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/all] Error 2
make: *** [all] Error 2
root@overo:/media/mmcblk0p2/OpenCV-2.3.1/mainhasan#

4.

[ 25%] Building CXX object modules/highgui/CMakeFiles/opencv_highgui.dir/src/cap_ffmpeg.o
In file included from /media/mmcblk0p2/OpenCV-2.3.1/modules/highgui/src/cap_ffmpeg.cpp:45:
 /media/mmcblk0p2/OpenCV-2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:
In member function 'bool CvCapture_FFMPEG::reopen()':
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:486: warning: 'int av_open_input_file(AVFormatContext**, const char*, AVInputFormat*, int, AVFormatParameters*)' is deprecated (declared at
/usr/local/include/libavformat/avformat.h:1093)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:486: warning: 'int av_open_input_file(AVFormatContext**, const char*, AVInputFormat*, int, AVFormatParameters*)' is deprecated (declared at
/usr/local/include/libavformat/avformat.h:1093)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:494: warning: 'int avcodec_thread_init(AVCodecContext*, int)' is deprecated (declared at
/usr/local/include/libavcodec/avcodec.h:3680)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:494: warning: 'int avcodec_thread_init(AVCodecContext*, int)' is deprecated (declared at
/usr/local/include/libavcodec/avcodec.h:3680)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:524: warning: 'int av_open_input_file(AVFormatContext**, const char*, AVInputFormat*, int, AVFormatParameters*)' is deprecated (declared at
/usr/local/include/libavformat/avformat.h:1093)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:524: warning: 'int av_open_input_file(AVFormatContext**, const char*, AVInputFormat*, int, AVFormatParameters*)' is deprecated (declared at
/usr/local/include/libavformat/avformat.h:1093)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:541: warning: 'int avcodec_thread_init(AVCodecContext*, _int)' is deprecated (declared at
/usr/local/include/libavcodec/avcodec.h:3680)
 /media/mmcblk0p2/OpenCV-
2.3.1/modules/highgui/src/cap_ffmpeg_impl.hpp:541: warning: 'int avcodec_thread_init(AVCodecContext*, _int)' is deprecated (declared at
/usr/local/include/libavcodec/avcodec.h:3680)
In member function 'void CvVideoWriter_FFMPEG::close()':

In member function 'bool CvVideoWriter_FFMPEG::open(const char*, int, double, int, int, bool)':

In member function 'void CvVideoWriter_FFMPEG::open(const char*, int, int, int, int, bool)':

5.

[13%] Building CXX object modules/highgui/CMakeFiles/opencv_highgui.dir/src/loadsave.o

[12%] Building CXX object modules/imgproc/CMakeFiles/opencv_imgproc.dir/src/histogram.o

Linking CXX shared library ../../../lib/libopencv_imgproc.so

[13%] Built target opencv_imgproc

[13%] Generating opencv_calib3d_pch_dephelp.cxx
Scanning dependencies of target opencv_calib3d_pch_dephelp
[ 13%] Building CXX object
modules/calib3d/CMakeFiles/opencv_calib3d_pch_dephelp.dir/opencv_calib3d_pch_dephelp.o
In file included from
/media/mmcblk0p2/opencv/modules/calib3d/include/opencv2/calib3d/calib3d.hpp:47,
   from
/media/mmcblk0p2/opencv/modules/calib3d/src/precomp.hpp:53,
   from
/media/mmcblk0p2/opencv/hasan/modules/calib3d/opencv_calib3d_pch_dephelp.p.cxx:1:
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:47:39: error: opencv2/flann/miniflann.hpp: No such file or directory
In file included from
/media/mmcblk0p2/opencv/modules/calib3d/include/opencv2/calib3d/calib3d.hpp:47,
   from
/media/mmcblk0p2/opencv/modules/calib3d/src/precomp.hpp:53,
   from
/media/mmcblk0p2/opencv/hasan/modules/calib3d/opencv_calib3d_pch_dephelp.p.cxx:1:
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:1293: error: 'cv::flann' has not been declared
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:1293: error: ISO C++ forbids declaration of 'Index' with no type
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:1293: error: expected ';' before '*' token
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2568: error: 'flann' was not declared in this scope
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2568: error: template argument 1 is invalid
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2569: error: 'flann' was not declared in this scope
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2569: error: template argument 1 is invalid
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2594: error: 'flann' was not declared in this scope
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2594: error: template argument 1 is invalid
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2595: error: 'flann' was not declared in this scope
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2595: error: template argument 1 is invalid
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2596: error: 'flann' was not declared in this scope
/media/mmcblk0p2/opencv/modules/calib3d/../../../features2d/include/opencv2/features2d/features2d.hpp:2596: error: template argument 1 is invalid
6. Figure: Clock-Skew error

7.
[ 1%] Built target pch_Generate_opencl_imgproc
[ 1%] Built target opencv_core_pch_dephelp
[ 2%] Built target pch_Generate_opencv_core
[ 6%] Built target opencv_core
[14%] Built target opencv_imgproc
[14%] Built target opencv_calib3d_pch_dephelp
[15%] Built target pch_Generate_opencv_calib3d
[15%] Built target opencv_features2d_pch_dephelp
[15%] Built target pch_Generate_opencv_features2d
[15%] Building CXX object modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/opencv_flann_pch_dephelp.o

In file included from /media/mmcblk0p2/OpenCV-2.3.1/modules/flann/src/precomp.h:9,
  from /media/mmcblk0p2/OpenCV-2.3.1/mainhasan/modules/flann/opencv_flann_pch_dephelp.cxx:1:
  /media/mmcblk0p2/OpenCV-2.3.1/modules/flann/include/opencv2/flann/dist.h: In function 'T cvflann::abs(T) [with T = long double]':
 /media/mmcblk0p2/OpenCV-2.3.1/modules/flann/include/opencv2/flann/dist.h:63: error: 'fabsl' was not declared in this scope
make[2]: *** [modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/opencv_flann_pch_dephelp.o] Error 1
make[1]: *** [modules/flann/CMakeFiles/opencv_flann_pch_dephelp.dir/all] Error 2
make: *** [all] Error 2
root@overo:/media/mmcblk0p2/OpenCV-2.3.1/mainhasan#
Appendix B

Gumstix Overo Fire

Processor:
- Texas Instruments OMAP 3530 Applications Processor
- ARM Cortex-A8 CPU
- C64x+ digital signal processor (DSP) core
- POWERVR SGX for 2D and 3D graphics acceleration

Clock (MHz): 720 MHz

Performance: Up to 1200 Dhrystone MIPS

Memory: 512MB RAM, 512MB Flash

Features:
- OMAP3530 Application Processor 802.11b/g wireless communications
- Bluetooth communications
- MicroSD card slot
- TPS65950 Power Management

Expandability:
- Via one 140-pin expansion board of Gumstix Overo series or custom, 140-pin expansion board
- Via 27-pin camera board

Connections:
- 70-pin connectors with 140 signals for:
  - I2C, PWM lines (6), A/D (6), 1-wire
  - UART, SPI, Extra MMC lines
- Headset, Microphone  
- Backup battery  
- High Speed USB Host and USB OTG  

- **27-pin connector** with signals for camera board  
  (2) U.FL antenna connectors

**Power:** Powered via expansion board connected to dual 70-pin connectors

**Layout:** layout version R2606

**Size:** 17mm x 58mm x 4.2mm  
(0.67 in. x 2.28 in. 0.16 in.)

**Temperatures:** Built with components rated 0C < T <75C

**Mounting:** Four mounting holes for securing to Overo-series, or custom, expansion board

---

**Palo35 Expansion board**

**SKU:** GX.Palo35

**Model Number:** PKG30035

**Ports:**

70-pin connectors: AVX 5602 series 40-pin header (not populated)  
Two two-wire serial ports  
One 1-wire port  
6 PWM lines  
I2C port  
SPI bus  
6 A/D input lines  
Processor control signals

**Other:**

LCD-ready for 3.5" LG touch-screen  
Touchscreen controller  
Provides 24 bits per pixel  
USB mini-AB with OTG signals  
USB mini-B connector with USB Host signals  
USB console
Coin cell battery backup and accelerometer
Stereo audio in
Stereo audio out
Signals available on 0.100" through-holes at 1.8V logic

Size: 98.5 x 64mm

Power: 4V to 5.5V Input
Appendix C

Cmake Installation in Gumstix

overo login: root

root@overo:~# sudo df

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>1K-blocks</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootfs</td>
<td>7605888</td>
<td>2956152</td>
<td>4263376</td>
<td>41%</td>
<td>/</td>
</tr>
<tr>
<td>/dev/root</td>
<td>7605888</td>
<td>2956152</td>
<td>4263376</td>
<td>41%</td>
<td>/</td>
</tr>
<tr>
<td>devtmpfs</td>
<td>120888</td>
<td>220</td>
<td>120668</td>
<td>1%</td>
<td>/dev</td>
</tr>
<tr>
<td>tmpfs</td>
<td>40</td>
<td>0</td>
<td>40</td>
<td>0%</td>
<td>/mnt/.splash</td>
</tr>
<tr>
<td>none</td>
<td>120888</td>
<td>220</td>
<td>120668</td>
<td>1%</td>
<td>/dev</td>
</tr>
<tr>
<td>/dev/mmcblk0p1</td>
<td>39497</td>
<td>3055</td>
<td>36442</td>
<td>8%</td>
<td>/media/mmcblk0p1</td>
</tr>
<tr>
<td>/dev/mmcblk0p2</td>
<td>7605888</td>
<td>2956152</td>
<td>4263376</td>
<td>41%</td>
<td>/media/mmcblk0p2</td>
</tr>
<tr>
<td>tmpfs</td>
<td>120888</td>
<td>436</td>
<td>120452</td>
<td>1%</td>
<td>/var/volatile</td>
</tr>
<tr>
<td>tmpfs</td>
<td>120888</td>
<td>0</td>
<td>120888</td>
<td>0%</td>
<td>/dev/shm</td>
</tr>
<tr>
<td>tmpfs</td>
<td>120888</td>
<td>0</td>
<td>120888</td>
<td>0%</td>
<td>/media/ram</td>
</tr>
</tbody>
</table>

root@overo:~# cd /media/mmcblk0p2

root@overo:/media/mmcblk0p2# cd cmake-2.8.6

root@overo:/media/mmcblk0p2/cmake-2.8.6# ./configure

--------------------------------------------
CMake 2.8.6, Copyright 2000-2009 Kitware, Inc.

Found GNU toolchain

C compiler on this system is: gcc

C++ compiler on this system is: g++

Makefile processor on this system is: make

g++ is GNU compiler
g++ has STL in std:: namespace

g++ has ANSI streams

g++ has streams in std:: namespace

g++ has sstream

g++ has operator!=(string, char*)

g++ has stl iterator_traits

g++ has standard template allocator

g++ has allocator<>::rebind<>

g++ does not have non-standard allocator<>::max_size argument

g++ has stl containers supporting allocator objects

g++ has header cstddef

g++ requires template friends to use <>

g++ supports member templates

g++ has standard template specialization syntax

g++ has argument dependent lookup

g++ has struct stat with st_mtim member

g++ has ios::binary openmode

g++ has ANSI for scoping

make: `cmake' is up to date.

loading initial cache file /media/mmcblk0p2/cmake-2.8.6/Bootstrap.cmk/InitialCacheFlags.cmake

-- Could NOT find Qt4 (missing: QT_QMAKE_EXECUTABLE QT_MOC_EXECUTABLE QT_RCC_EXECUTABLE QT_UIC_EXECUTABLE QT_INCLUDE_DIR QT_LIBRARY_DIR QT_QTCORE_LIBRARY)

-- Configuring done

-- Generating done

-- Build files have been written to: /media/mmcblk0p2/cmake-2.8.6
CMake has bootstrapped. Now run make.

root@overo:/media/mmcblk0p2/cmake-2.8.6# **make install**

[  4%] Built target cmsys
[  4%] Built target cmsysTestDynload
[  6%] Built target cmsys_c
[  6%] Built target cmsysTestProcess
[  7%] Built target cmsysTestSharedForward
[  7%] Built target cmsysTestsC
[ 10%] Built target cmsysTestsCxx
[ 12%] Built target cmzlib

(continued)

-- Installing: /usr/local/doc/cmake-2.8/cmake-properties.html
-- Installing: /usr/local/doc/cmake-2.8/cmake-variables.html
-- Installing: /usr/local/doc/cmake-2.8/cmake-modules.html
-- Installing: /usr/local/doc/cmake-2.8/cmake-commands.html
-- Installing: /usr/local/doc/cmake-2.8/cmake-compatcommands.html
-- Installing: /usr/local/doc/cmake-2.8/ctest.html
-- Installing: /usr/local/doc/cmake-2.8/cpack.html
-- Installing: /usr/local/doc/cmake-2.8/ccmake.html
-- Installing: /usr/local/doc/cmake-2.8/cmake.txt
-- Installing: /usr/local/doc/cmake-2.8/cmake.docbook
-- Installing: /usr/local/doc/cmake-2.8/cmake-policies.txt
-- Installing: /usr/local/doc/cmake-2.8/cmake-properties.txt
-- Installing: /usr/local/doc/cmake-2.8/cmake-variables.txt
-- Installing: /usr/local/doc/cmake-2.8/cmake-modules.txt
-- Installing: /usr/local/doc/cmake-2.8/cmake-commands.txt
-- Installing: /usr/local/doc/cmake-2.8/cmake-compatcommands.txt
-- Installing: /usr/local/doc/cmake-2.8/ctest.txt
-- Installing: /usr/local/doc/cmake-2.8/ctest.docbook
-- Installing: /usr/local/doc/cmake-2.8/cpack.txt
-- Installing: /usr/local/doc/cmake-2.8/cpack.docbook
-- Installing: /usr/local/doc/cmake-2.8/ccmake.txt
-- Installing: /usr/local/doc/cmake-2.8/ccmake.docbook
-- Up-to-date: /usr/local/share/aclocal/cmake.m4

root@overo:/media/mmcblk0p2/cmake-2.8.6#
Appendix D

OpenCV installation in Gumstix

```
root@overo:/media/mmcblk0p2/cmake-2.8.6# cd /media/mmcblk0p2/
root@overo:/media/mmcblk0p2# cd OpenCV-2.3.1
root@overo:/media/mmcblk0p2/OpenCV-2.3.1# mkdir mainhasan
root@overo:/media/mmcblk0p2/OpenCV-2.3.1# cd mainhasan
root@overo:/media/mmcblk0p2/OpenCV-2.3.1/mainhasan# cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON ..

-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detected version of GNU GCC: 43 (403)

.......................

.......................

(continued)

-- Parsing 'cvconfig.h.cmake'

--
```
General configuration for opencv 2.3.1

Built as dynamic libs?: YES
Compiler: /usr/bin/c++

C++ flags (Release): -Wall -pthread -ffunction-sections -O3 -DNDEBUG -fomit-frame-pointer -DNDEBUG

C++ flags (Debug): -Wall -pthread -ffunction-sections -g -O0 -DDEBUG -D_DEBUG -ggdb3

Linker flags (Release):
Linker flags (Debug):

GUI:
GTK+ 2.x: NO
GThread: YES

Media I/O:
ZLib: YES
JPEG: TRUE
PNG: build
TIFF: build
JPEG 2000: TRUE

OpenEXR: NO
OpenNI: NO
OpenNI PrimeSensor Modules: NO
XIMEA: NO

Video I/O:
<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC1394 1.x</td>
<td>NO</td>
</tr>
<tr>
<td>DC1394 2.x</td>
<td>NO</td>
</tr>
<tr>
<td>FFMPEG</td>
<td>YES</td>
</tr>
<tr>
<td>codec</td>
<td>YES</td>
</tr>
<tr>
<td>format</td>
<td>YES</td>
</tr>
<tr>
<td>util</td>
<td>YES</td>
</tr>
<tr>
<td>swscale</td>
<td>YES</td>
</tr>
<tr>
<td>gentoo-style</td>
<td>YES</td>
</tr>
<tr>
<td>GStreamer</td>
<td>YES</td>
</tr>
<tr>
<td>UniCap</td>
<td>NO</td>
</tr>
<tr>
<td>PvAPI</td>
<td>NO</td>
</tr>
<tr>
<td>V4L/V4L2</td>
<td>1/1</td>
</tr>
<tr>
<td>Xine</td>
<td>NO</td>
</tr>
<tr>
<td>Other third-party libraries:</td>
<td></td>
</tr>
<tr>
<td>Use IPP</td>
<td>NO</td>
</tr>
<tr>
<td>Use TBB</td>
<td>NO</td>
</tr>
<tr>
<td>Use ThreadingFramework</td>
<td>NO</td>
</tr>
<tr>
<td>Use Cuda</td>
<td>NO</td>
</tr>
<tr>
<td>Use Eigen</td>
<td>NO</td>
</tr>
<tr>
<td>Interfaces:</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>YES</td>
</tr>
<tr>
<td>Python interpreter</td>
<td>/usr/bin/python -B (ver 2.6)</td>
</tr>
<tr>
<td>Python numpy</td>
<td>NO (Python wrappers will not be generated)</td>
</tr>
<tr>
<td>Java</td>
<td>NO</td>
</tr>
</tbody>
</table>
--
-- Documentation:
-- Sphinx: NO
-- PdfLaTeX compiler: NO
-- Build Documentation: NO
--
-- Tests and samples:
-- Tests: YES
-- Examples: NO
--
-- Install path: /usr/local
--
-- cvconfig.h is in: /media/mmcblk0p2/OpenCV-2.3.1/mainhasan
-- --------------------------------------------------------------------------
--
-- Configuring done
-- Generating done

CMake Warning:

Manually-specified variables were not used by the project:

BUILD_PYTHON_SUPPORT

-- Build files have been written to: /media/mmcblk0p2/OpenCV-2.3.1/mainhasan

root@overo:/media/mmcblk0p2/OpenCV-2.3.1/mainhasan# make install

[ 1%] Generating opencv_imgproc_pch_dephelp.cxx

Scanning dependencies of target opencv_imgproc_pch_dephelp

[ 1%] Building CXX object
modules/imgproc/CMakeFiles/opencv_imgproc_pch_dephelp.dir/opencv_imgproc_pch_dephelp.o

Linking CXX static library ../../lib/libopencv_imgproc_pch_dephelp.a
[ 1%] Built target opencv_imgproc_pch_dephelp
Scanning dependencies of target pch_Generate_opencv_imgproc
[ 1%] Generating precomp.hpp
[ 1%] Generating precomp.hpp.gch/opencv_imgproc_RELEASE.gch
[ 1%] Built target pch_Generate_opencv_imgproc
[ 1%] Generating opencv_core_pch_dephelp.cxx
Scanning dependencies of target opencv_core_pch_dephelp
[ 1%] Building CXX object modules/core/CMakeFiles/opencv_core_pch_dephelp.dir/opencv_core_pch_dephelp.o
Linking CXX static library ../../lib/libopencv_core_pch_dephelp.a
[ 1%] Built target opencv_core_pch_dephelp
Scanning dependencies of target pch_Generate_opencv_core
[ 1%] Generating precomp.hpp
[ 1%] Generating precomp.hpp.gch/opencv_core_RELEASE.gch
[ 2%] Built target pch_Generate_opencv_core
Scanning dependencies of target opencv_core
[ 2%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/rand.o
[ 2%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/persistence.o
[ 2%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/convert.o
[ 3%] Building CXX object modules/core/CMakeFiles/opencv_core.dir/src/matrix.o
........................................
....................................
Continued
[ 98%] Building CXX object modules/gpu/CMakeFiles/opencv_test_gpu.dir/test/test_features2d.o

[ 99%] Building CXX object modules/gpu/CMakeFiles/opencv_test_gpu.dir/test/test_hog.o
Linking CXX executable ../../bin/opencv_test_gpu

[ 99%] Built target opencv_test_gpu
Scanning dependencies of target opencv_stitching

[ 99%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/seam_finders.o
[ 99%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/util.o
[ 99%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/matchers.o
[ 99%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/warpers.o
[ 99%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/main.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/exposure_compensate.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/motion_estimators.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/precomp.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/blenders.o
[100%] Building CXX object modules/stitching/CMakeFiles/opencv_stitching.dir/autocalib.o

Linking CXX executable ../../bin/opencv_stitching

[100%] Built target opencv_stitching

Install the project...

-- Install configuration: "Release"

-- Installing: /usr/local/share/OpenCV/OpenCVConfig.cmake

-- Installing: /usr/local/share/OpenCV/OpenCVConfig-version.cmake

-- Installing: /usr/local/lib/pkgconfig/opencv.pc

-- Up-to-date: /usr/local/include/opencv/highgui.h

-- Up-to-date: /usr/local/include/opencv/cxcore.hpp

-- Up-to-date: /usr/local/include/opencv/cvaux.hpp

-- Up-to-date: /usr/local/include/opencv/cxcore.h

-- Up-to-date: /usr/local/include/opencv/cxeigen.hpp

-- Up-to-date: /usr/local/include/opencv/cv.hpp

-- Up-to-date: /usr/local/include/opencv/cv.h
-- Up-to-date: /usr/local/include/opencv/ml.h
-- Up-to-date: /usr/local/include/opencv/cxmisc.h
-- Up-to-date: /usr/local/include/opencv/cvaux.h
-- Up-to-date: /usr/local/include/opencv/cvwimage.h
-- Up-to-date: /usr/local/include/opencv2/opencv.hpp
-- Installing: /usr/local/lib/libopencv_calib3d.so.2.3.1
-- Up-to-date: /usr/local/lib/libopencv_calib3d.so.2.3

..........................
..........................
continued
-- Up-to-date: /usr/local/include/opencv2/gpu/devmem2d.hpp
-- Up-to-date: /usr/local/include/opencv2/gpu/stream_accessor.hpp
-- Up-to-date: /usr/local/include/opencv2/gpu/NPP_staging.hpp
-- Up-to-date: /usr/local/include/opencv2/gpu/NCV.hpp
-- Installing: /usr/local/bin/opencv_stitching
-- Set runtime path of "/usr/local/bin/opencv_stitching" to "/usr/local/lib"
-- Up-to-date: /usr/local/share/OpenCV/doc/haartraining.htm

..........................
......................
continued
-- Up-to-date: /usr/local/share/OpenCV/haarcascades/haarcascade_righteye_2splits.xml
-- Up-to-date: /usr/local/share/OpenCV/haarcascades/haarcascade_eye_tree_eyeglasses.xml
-- Up-to-date: /usr/local/share/OpenCV/lbpcascades/lbpcascade_frontalface.xml

root@overo:/media/mmcblk0p2/OpenCV-2.3.1/mainhasan#
Appendix E

Extracting the root file system

var/lib/opkg/
var/lib/opkg/base
var/lib/opkg/gstreamer
var/lib/opkg/tmp/
var/lib/opkg/python
var/lib/opkg/overo
var/lib/opkg/debug
var/lib/opkg/perl
var/lib/opkg/no-arch
var/lib/fontconfig/
var/lib/fontconfig/3830d5c3ddf5cd38a049b759396e72e-arm.cache-3
var/lib/fontconfig/7ef2298fde41cc6eeb7af42e48b7d293-arm.cache-3
var/lib/NetworkManager/
var/lib/NetworkManager/NetworkManager.state
var/lib/NetworkManager/timestamps
var/lib/dhcp/
var/lib/dhcp/dhclient.leases
var/lib/sudo/
var/lib/misc/
var/lib/bluetooth/
var/lib/dbus/
var/lib/dbus/machine-id
var/cron/

var/cron/tabs/

var/cron/tabs/root

var/syslog-ng.persist

var/lock

root@ubuntu:/media/b70eb85c-de5b-40e9-8ab0-ad5c4701ceeb#