

9-19-2016

# Panoramic Stereovision and Scene Reconstruction

Ashish Nair  
*Santa Clara University*

Follow this and additional works at: [http://scholarcommons.scu.edu/cseng\\_mstr](http://scholarcommons.scu.edu/cseng_mstr)



Part of the [Computer Engineering Commons](#)

---

## Recommended Citation

Nair, Ashish, "Panoramic Stereovision and Scene Reconstruction" (2016). *Computer Science and Engineering Master's Theses*. 2.  
[http://scholarcommons.scu.edu/cseng\\_mstr/2](http://scholarcommons.scu.edu/cseng_mstr/2)

This Thesis is brought to you for free and open access by the Engineering Master's Theses at Scholar Commons. It has been accepted for inclusion in Computer Science and Engineering Master's Theses by an authorized administrator of Scholar Commons. For more information, please contact [rscroggin@scu.edu](mailto:rscroggin@scu.edu).

**SANTA CLARA UNIVERSITY**

Department of Computer Engineering

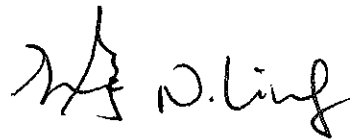
**Date: September 14, 2016**

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY  
SUPERVISION BY

Ashish Nair

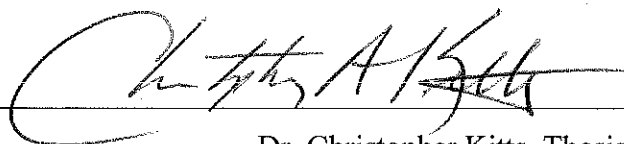
**Panoramic Stereovision and Scene  
Reconstruction**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE  
OF  
**MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING**



---

Dr. Nam Ling, Thesis Advisor and Chair of Department



---

Dr. Christopher Kitts, Thesis Reader

# **Panoramic Stereovision and Scene Reconstruction**

By

**Ashish Nair**

**COEN 497**

**Master's Thesis Research**

Submitted in Partial Fulfillment of the Requirements  
For the Degree of Master of Science  
in Computer Science and Engineering  
in the School of Engineering at  
Santa Clara University, 2016.

Santa Clara, California.

Date: 9/14/2016

# Panoramic Stereovision and Scene Reconstruction

Ashish Nair

Department of Computer Engineering  
Santa Clara University  
2016

## Abstract

With advancement of research in robotics and computer vision, an increasingly high number of applications require the understanding of a scene in three dimensions. A variety of systems are deployed to do the same. This thesis explores a novel 3D imaging technique. This involves the use of catadioptric cameras in a stereoscopic arrangement. A secondary system aims to stabilize the system in the event that the cameras are misaligned during operation. The system provides a stark advantage due to it being a cost effective alternative to present day standard state-of-the-art systems that achieve the same goal of 3D imaging. The compromise lies in the quality of depth estimation, which can be overcome with a different imager and calibration. The result was a panoramic disparity map generated by the system.

**Keywords:** Stereovision, SURF, RANSAC, 3D Imaging, Rectification, Stereo Correspondence

# Acknowledgements

I wish to thank Santa Clara University for helping me tailor my academic career to suit my research goals, as well as helping orient my skillset towards the field of Computer Vision. I also wish to thank Dr. Nam Ling for bearing with my interdisciplinary research interests, and guiding me accordingly.

Special thanks to the Robotic Systems Lab and Dr. Christopher Kitts for the amazing workspace, resources and the after-hours access during which much of my research was conducted. Thanks to NVIDIA Corp. for the equipment grant of the Jetson TK1 embedded platform. Finally, I wish to thank my parents for their ceaseless support and encouragement.

# Table of Contents

<b>Abstract</b> .....	<b>iii</b>
<b>Acknowledgements</b> .....	<b>iv</b>
<b>Table of Contents</b> .....	<b>v</b>
<b>List of Figures</b> .....	<b>vi</b>
<b>List of Tables</b> .....	<b>vii</b>
<b>1. Introduction</b> .....	<b>1</b>
1.1 Background .....	1
1.2 Literature Review .....	3
1.2.1 Facebook Surround 360 .....	3
1.2.2 Lytro Immerge .....	4
1.2.3 Velodyne HDL-64E .....	6
1.2.4 Comparison and Summary .....	7
1.3 Project Statement .....	8
<b>2. Hardware and Vision Systems Design</b> .....	<b>9</b>
2.1 System Architecture .....	9
2.2 Catadioptrics .....	9
2.2.1 Central and Non-Central Camera Models .....	10
2.2.2 Folded Cameras .....	11
2.2.3 The VSN Mobil V.360 .....	11
2.3 Stereovision .....	13
2.3.1 Generic Stereovision Pipeline .....	14
2.3.2 Catadioptric Stereoscapy .....	17
2.4 NVIDIA Jetson TK1 .....	19
<b>3. Software</b> .....	<b>21</b>
3.1 Development with VS2013 and OpenCV.....	21
3.2 Stereo Correspondence with Semi Global Block Matching .....	22
3.2.1 Theory .....	22
3.2.2 Implementation and Results .....	23
3.3 Automated Rectification .....	26

<b>4. Conclusion</b> .....	<b>30</b>
<b>References</b> .....	<b>31</b>
<b>Appendix A: Source code for computing disparity</b> .....	<b>32</b>
<b>Appendix B: Source code for automated rectification</b> .....	<b>36</b>
<b>Appendix C: VSN Mobil V.360 Users Guide</b> .....	<b>45</b>

## List of Figures

Figure 1.1: Industrial and Non-Industrial Robotics Revenue .....	1
Figure 1.2: Augmented/Virtual Reality Revenue Forecast .....	2
Figure 1.3: Facebook Surround 360 .....	3
Figure 1.4: Lytro Immerge Light Field Camera .....	4
Figure 1.5: Multiple perspectives from lenslets in a Light Field Camera .....	5
Figure 1.6: Velodyne HDL-64E LIDAR .....	6
Figure 1.7: HDL-64E Point Cloud at an Intersection .....	6
Figure 2.1: Hardware system block diagram .....	9
Figure 2.2: Catadioptric Image (radially distorted) .....	10
Figure 2.3: Central and Non-Central Camera models .....	10
Figure 2.4: Dual-mirror folded catadioptric camera .....	11
Figure 2.5: VSN V.360 .....	12
Figure 2.6: Magewell USB Capture HDMI USB-UVC Converter .....	12
Figure 2.7: Scene visualized by a stereoscopic camera .....	13
Figure 2.8: Stereovision Pipeline .....	14
Figure 2.9: Grid pattern and the effect of barrel distortion .....	15
Figure 2.10: Frames from a dual camera setup, and rectified frames brought into a parallel image plane .....	15
Figure 2.11: Disparity map of ‘Cones’ dataset .....	16

Figure 2.12: Point P at a distance Z from camera centers .....	17
Figure 2.13: Three-dimensional grayscale point cloud from ‘Cones’ Dataset .....	17
Figure 2.14: Horizontal configuration for catadioptric stereoscopy .....	18
Figure 2.15: Occlusions in a horizontal configuration of catadioptric stereovision .....	18
Figure 2.16: Vertical configuration for catadioptric stereoscopy .....	19
Figure 2.17: Jetson TK1 with a USB to mini-PCIe converter .....	20
Figure 3.1: Program Workflow .....	24
Figure 3.2: Upper camera image, Lower camera image, and Disparity Map .....	24
Figure 3.3: Calibration Scene .....	25
Figure 3.4: Curve fitting with calibration points .....	25
Figure 3.5: Test Scene .....	26
Figure 3.6: Primary sources of non-rigidity in the position adjustment system .....	27
Figure 3.7: Automated Rectification Pipeline .....	27
Figure 3.8: Tilted upper camera test for Automated Rectification system .....	28
Figure 3.9: Automated Rectification Testing; Tilted upper camera input, Lower camera input, Rectified Upper Camera feed .....	29

## List of Tables

Table 1.1: State-of-the-art 3D vision systems comparison chart.....	7
Table 3.1: Disparity-Depth Calibration .....	25
Table 3.2: Test Scene Results .....	26



# 1. Introduction

## 1.1. Background

Modern day applications in fields of robotics, virtual reality and augmented reality systems require a comprehensive perception of their immediate surroundings with the highest possible accuracy and resolution, along with robustness across a wide range of operating conditions.

Robotic systems applications extend to warehouse operations, autonomous driving, search and rescue, surveillance, building inspection, and remote surgery to name a few. In each scenario the robot has to navigate an obstacle-ridden space to perform a complex set of tasks. Augmented and VR systems deal with simulating a partial or a completely new world that superimposes, or replaces the real world.

3D vision technology is critical to the aforementioned fields given its ability to generate spatial models of the physical world. Consequently, there has been a wide adoption of the same. Success in the deployment of 3D vision methods has contributed significantly to the growth of these markets, as displayed in Figures 1.1, and 1.2.

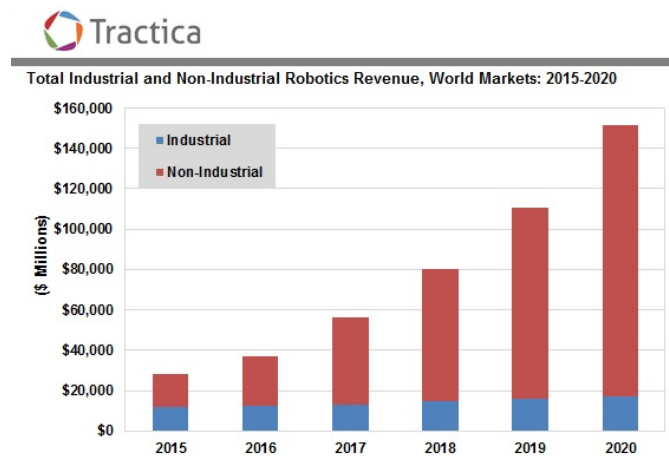


Fig 1.1: Industrial and Non-Industrial Robotics Revenue,  
Source: Tractica, "Robotics Market Forecasts", 2015.

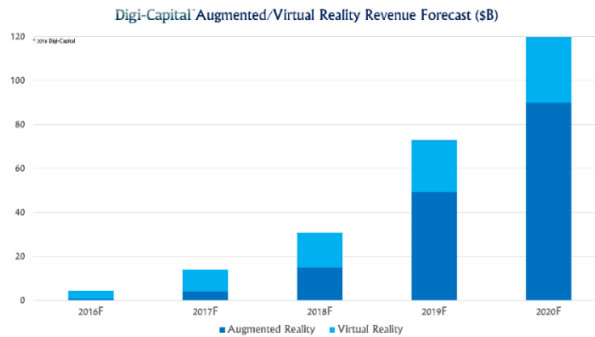


Fig 1.2: Augmented/Virtual Reality Revenue Forecast,  
 Source: Digi-Capital, “Augmented/Virtual Reality Report 2016”, 2016.

The purpose of this project is to create a novel panoramic 3D spatial modeling system that is robust and serves as a cost effective replacement to current techniques in this space, without compromising output quality. The developed system makes use of special optics to gain a panoramic field of vision and leverages two points of view to obtain stereoscopic scene information, thus gaining a sense of depth for the entire panorama.

## 1.2. Literature Review

This section provides an overview of existing state of the art industrial grade solutions for panoramic 3D imaging. Note that panoramic 3D is different from general monoscopic 360 video in the context that, in the former case the viewer is allowed six degrees of freedom for observing the scene, as compared to the latter’s three degrees of freedom – roll, pitch, and yaw where the scene is projected onto a sphere.

### 1.2.1. Facebook Surround 360

The Surround 360 is Facebook’s own design for a panoramic 3D vision system. As an open-source design, both hardware and software will be made available for further

prototyping. The system is built for stationary camera VR applications, and features multiple Point Grey cameras. A global shutter ensures all camera feeds are completely in synchronization to preserve the quality of the reconstructions. Each camera in the system provides options of 4K, 6K, and 8K video resolutions.



Fig 1.3: Facebook Surround 360

Source: Facebook, "Facebook Surround 360", 2016. [Online].

Video feed at 30 FPS requires transfer rates as high as 17Gbps in video transfer rates, which is fulfilled by means of an 8-way, level-5 RAID SSD disk system. Frames from all cameras are initially in Bayer form, which are then converted to a gamma corrected RGB format. Frames are then undistorted, and bundle adjustment is performed for rectification of camera misalignments. Stereo correspondence is performed by means of optical flow estimation. The advantage of implementing optical flow is a high quality disparity map between cameras, but at the cost of an additional order of computational complexity. Novel views are then synthesized for multiple viewing positions. Panoramas are stitched via software at post production.

## 1.2.2. Lytro Immerge



Fig 1.4: Lytro Immerge Light Field Camera

Source: <http://www.lytro.com> [Online].

Lytro specializes in developing plenoptic cameras, also known as Light Field cameras. These cameras include a microscopic-lens (lenslets) array placed above a high density image sensor. This arrangement helps capture the entire light field, by tracking the direction and color intensities of bundles of rays from multiple unique points of view. With the given information it is then possible to calculate where the bundles of light rays originated and where they converge. A virtual image plane can then be generated where a particular object is in focus – thus providing refocusing and depth sensing capabilities. The Immerge system is a suite that includes a camera, processing hardware, and software tools to create a cinematic 3D VR experience that allows the user 6 degrees of freedom. The camera has provisions for multiple removable sensor arrays as displayed in Fig. 1.5. The primary disadvantage of this system is its need for a special server rack required to perform reconstructions, due to the computational complexity of the entire process.

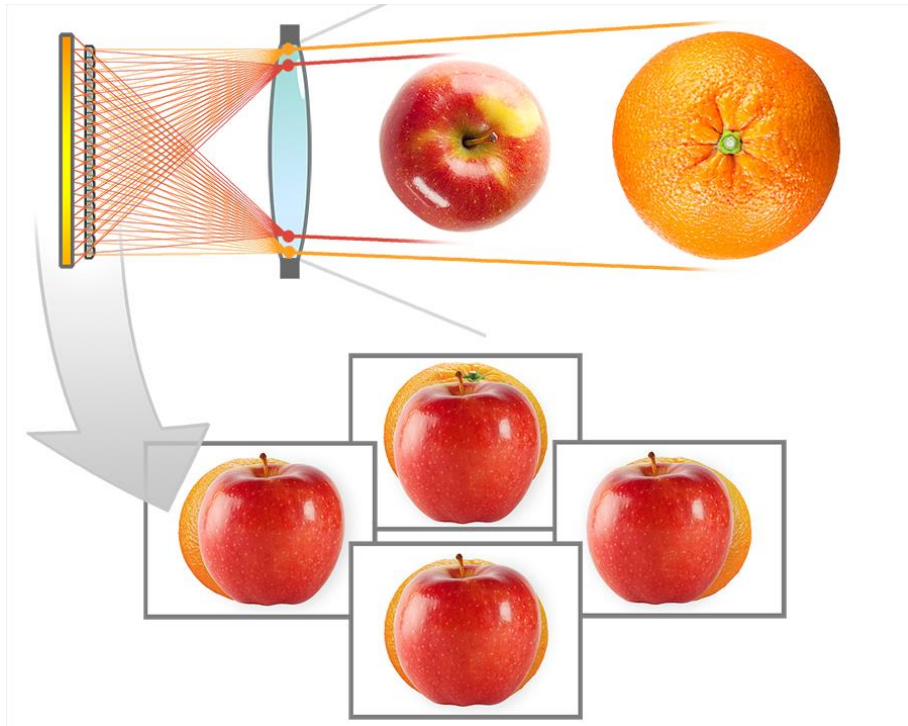


Fig 1.5: Multiple perspectives from lenslets in a Light Field Camera,  
 Source: Lytro, "The Creative benefits of Light Field", April 2016. [Online].

The secondary disadvantage of the Immerge is that all computations are performed post production. The system ideally suited for budgets aimed towards renting the equipment.

### 1.2.3. Velodyne HDL-64E

3D Lidar is known to be one of the most reliable source of mapping depth in a scene. Lidar is based on the concept of time of flight, in which a transmitter emits a laser signal, and based on the time taken to receive the signal, depth is calculated for that particular point in space. A scanning Lidar can comprehend a panoramic depth map in each rotation. Lidar detection schemes can be classified into two kinds: incoherent, and coherent. Incoherent lidar, also known as direct energy detection is primarily an amplitude measurement method. The coherent method involves heterodyne detection, where pulses are non-linearly mixed with a reference signal, and the received wave contains the signature of the original, but at carrier frequency.



Fig 1.6: Velodyne HDL-64E LIDAR

Source: Velodyne, “Velodyne Lidar HDL 64E: High Definition Real-time LIDAR”, January 2016.

The coherent method of detection is better suited for relatively low power applications, and is safer to the human eye. The HDL-64E directs 64 beams of laser into the scene, each at a unique vertical angle. Total vertical FOV is 26.8 degrees, and provides over 2.2 million points per second with a programmable framerate of up to 20Hz. The range of measurement of the HDL-64E extends to 120m. Point clouds generated by this module are accurate to within 2cm. Fig. 1.7 shows a point cloud generated by the HDL-64E.

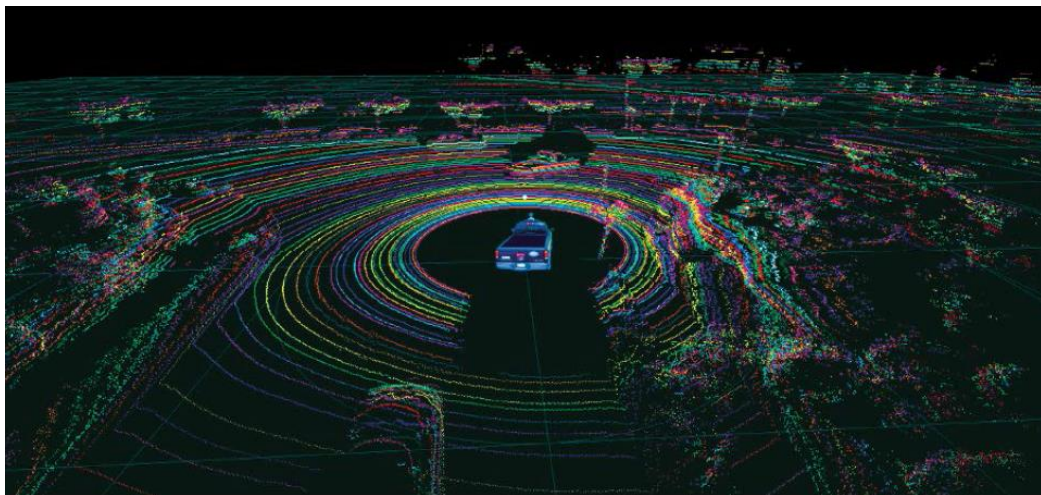


Fig 1.7: HDL-64E Point Cloud at an Intersection.

Source: Velodyne, “Velodyne Lidar HDL 64E: High Definition Real-time LIDAR”, January 2016.

## 1.2.4. Comparison and Summary

The methods discussed in sections 1.2.1, 1.2.2, and 1.2.3 are unique in terms of the technology used to generate spatial models and are effective at their respective fields of application. The former two are used in VR content creation, while the HDL-64E is used to generate terrestrial maps for robotic applications such as autonomous driving. The Surround 360 depends on stereovision from multiple cameras, the Immerge 360 works on Lightfield technology, and the HDL64E is a Lidar module.

<b>Sr. No.</b>	<b>Solution</b>	<b>Cost</b>	<b>Real-time</b>	<b>Vertical Field of View (FOV)</b>	<b>Approx. Footprint Excluding Mounts (in mm<sup>3</sup>)</b>
1.	Facebook Surround 360	\$ 30,000	No	360 <sup>0</sup>	65,098,740
2.	Lytro Immerge	Not released	No	360 <sup>0</sup>	Not released
3.	Velodyne HDL-64E	\$ 70,000	Yes	26.8 <sup>0</sup>	14,617,190

Table 1.1: State-of-the-art 3D vision systems comparison chart

In addition to the contents of table 1.1, it is noted that the Surround 360 and the Immerge both have secondary processing units that the camera rigs are connected to, which are significantly increase the form factor of the setup – making them better suited for

stationary camera applications. There currently is no qualitative data on the accuracy and resolution of the two aforementioned cameras. The HDL-64E is portable and can be connected to any device that possesses an Ethernet interface, but is unable to provide any color data from the scene. It has a depth estimation error of less than two centimeters up to a range of 120 meters. Thus each method has its own distinct set of advantages and limitations.

## 1.3. Project Statement

The goal of this research project was to explore an alternative panoramic 3D imaging solution with real-time capabilities, and a small form factor of 1,158,162 mm<sup>3</sup> – at a low cost of implementation (under \$2000). This was done by means of stereoscopic vision with two catadioptric cameras.

The advantage of deploying catadioptric cameras over regular cameras is that the former takes a panoramic image, thereby not requiring any stitching – thus reducing both processing time, as well as the overall hardware requirements of the system. This in turn helps reduce the power consumed, and minimizes the costs involved.

However, the major challenge of such a system is associated a lack of calibration data due to unknown intrinsic parameters of the cameras. Without this data, it is not possible to extrapolate the real world depth of any point in the scene. Furthermore, unexpected changes in camera orientation leads to changes in calibration that adversely affect the quality of the resultant disparity map. These two issues have been addressed in this project by means of external calibration with a secondary 3D imager, and automated stereo rectification respectively.

The result of this project was a high quality disparity map that can be used to generate a spatial model of a scene.



# 2. Hardware and Vision Systems Design

## 2.1. System Architecture

The proposed system comprises of two catadioptric cameras. Each camera outputs a live video stream over HDMI. HDMI to USB Video Class converters are used to convert each camera's HDMI feed into USB 3.0 compatible webcam feed. The converter output was then passed to a laptop, which then implements the algorithm for estimating disparity in real-time. Fig 2.1 is a block diagram of the same.

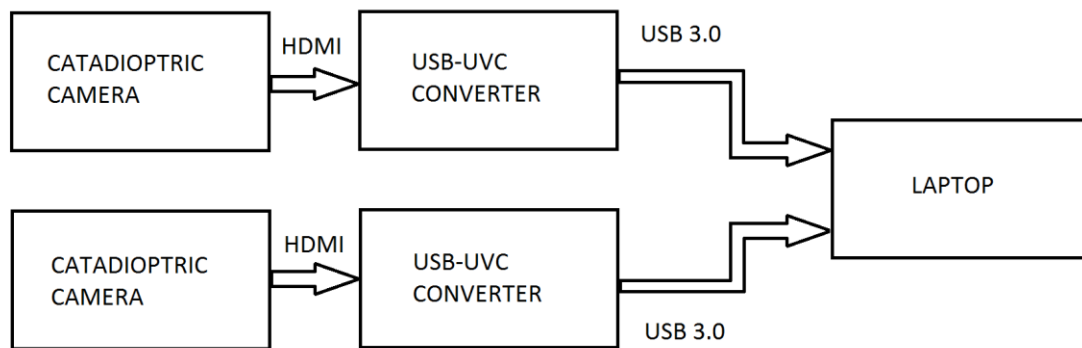


Fig. 2.1: Hardware system block diagram

## 2.2. Catadioptrics

A catadioptric optical system refers to the combination of lenses, also known as dioptrics, and curved mirrors, known as catoptrics. Catadioptric systems have been traditionally deployed in focusing systems of headlamps, telescopes, and microscopes. More recently, they have been put into effect in special purpose cameras that aim towards panoramic imaging. The feed from a catadioptric camera is displayed in Fig 2.2. There is a significant amount of observable radial distortion introduced due to a special lens and mirror arrangement.

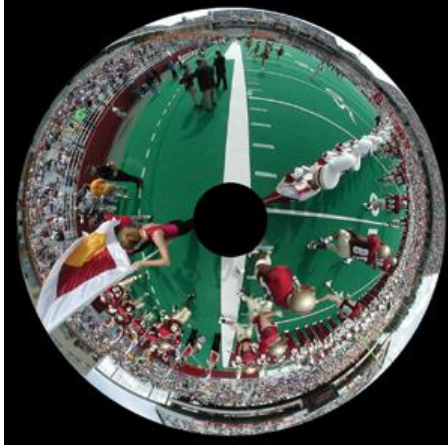


Fig 2.2: Catadioptric Image (radially distorted),  
Source: CAVE Laboratory at Columbia University.

### 2.2.1. Central and Non-Central Camera Models

Catadioptric cameras can be broadly classified into two models: central and non-central cameras. In central cameras, all incoming rays of light intersect at a unique viewpoint. This condition is also called the single viewpoint condition, and is inherently satisfied by perspective cameras. In non-central cameras, the incoming rays do not intersect at one unique viewpoint. In case of a hyperbolic mirror, there are two focal points – the rays intersect at one focal point ( $F$ ), and the camera is placed at the second focal point ( $F'$ ) as shown in Fig 2.3(a).

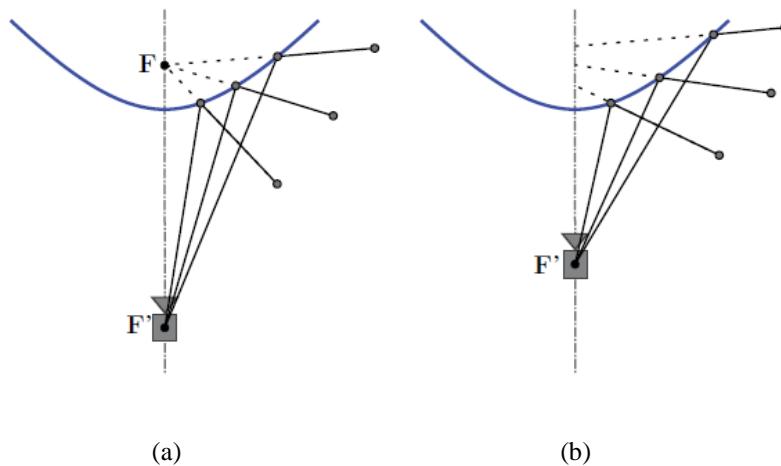


Fig 2.3: Central (a), and Non-Central (b) Camera models.

Source: M. Schonbein, Omnidirectional Stereo Vision for Autonomous Vehicles, Karlsruhe, KIT Scientific Publishing, 2014.

## 2.2.2. Folded Cameras

The drawback associated with single mirror systems is that they have a relatively large form factor for a given vertical field of view. Optical folding allows for a significantly greater vertical FOV with a smaller package size as compared to its single mirror equivalent. A folded catadioptric camera primarily involves two conic mirrors – a primary and a secondary. One of the nine forms of single viewpoint folded catadioptric system described by Benosman and Kang in [1] is displayed in Fig. 2.4(a). It makes use of dual hyperbolic mirrors. Alongside Fig. 2.4(a) is a close view of the catadioptrics of the camera hardware used in this project, Fig. 2.4(b).

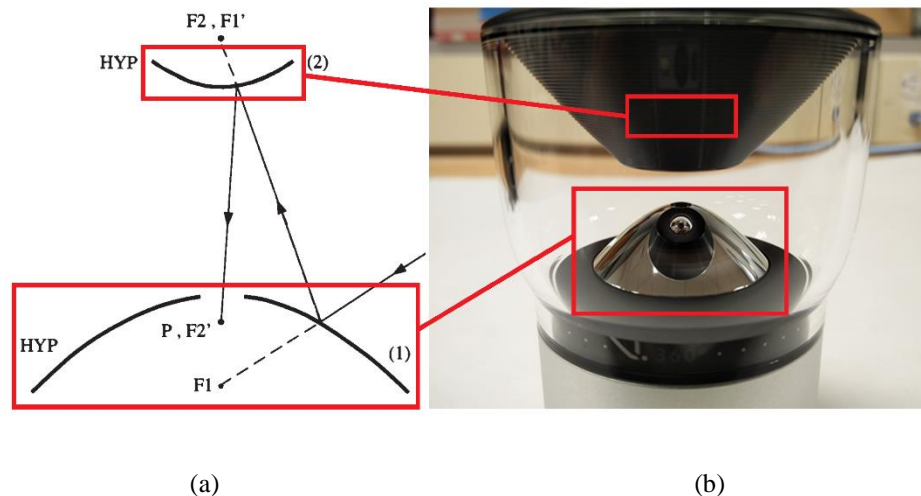


Fig 2.4: Dual-mirror folded catadioptric camera. (a) One of nine possible forms of dual-mirror single viewpoint folded catadioptric cameras. Source: R. Benosman and S. B. Kang, Panoramic Vision: Sensors, Theory, and Applications, New York: Springer Science+Business Media, 2001; (b) VSN Mobil V.360 camera catadioptrics. Corresponding positions of Primary (1) and secondary (2) mirrors are indicated by bounding boxes.

## 2.2.3. The VSN Mobil V.360

The VSN V.360 is the camera selected for the panoramic stereovision system in this project. It is a catadioptric camera with a folded configuration as indicated in section (c) of Fig 2.3. The camera acquires frames of video with a 16-megapixel imager, capable of generating resolutions ranging from 1920x320 through 6480x1080. Vertical FOV ranges

from +45 degrees to -15 degrees. Fig 2.5(a) is a picture of the camera hardware, and Fig. 2.5(b) illustrates a still image taken by the camera.



(a)



(b)

Fig 2.5: VSN V.360 (a) VSN Mobil V.360 Camera, (b) Equi-rectangular (undistorted) image

As displayed above, the camera accounts for lens distortion and undistorts images and frames of video into an equi-rectangular form by means of a Qualcomm Snapdragon 800 processor. The advantage of this feature is that epipolar lines are made linear by default, thus requiring a minimal amount of image warping for stereo rectification.



Fig 2.6: Magewell USB Capture HDMI USB-UVC Converter

Video feed can be extracted from the V.360 via two methods. The first method is over a wifi connection between the camera and a mobile device that is capable of running the V.360 app. The app offers a live view as well as recording capabilities. Live feed cannot be directly extracted via third-party scripts due to the closed source nature of the product. The second method is via a hardware link. Live feed is accessible via an on-board HDMI port. In order to connect it to a personal computer, the input needs to be of the form of a USB Video Class (UVC) Device. This is possible using a UVC capture card that converts HDMI input into USB-UVC feed. The camera's raw video feed can then be accessed like any standard webcam. The capture card deployed is the Magewell USB Capture HDMI.

## 2.3. Stereovision

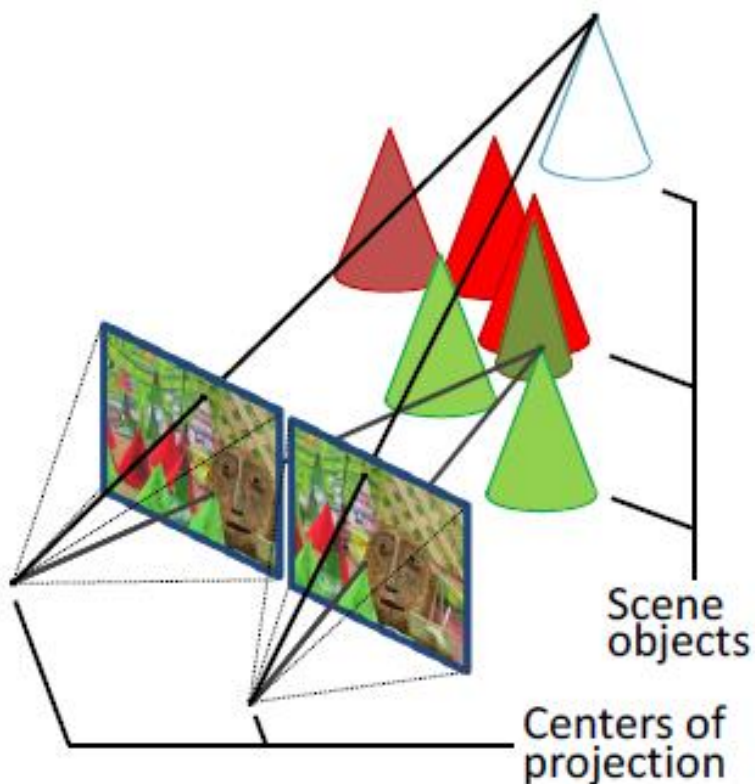


Fig 2.7: Scene visualized by a stereoscopic camera.

Source: Ensenso and IDS Imaging Development Systems GmbH, "Obtaining Depth from Stereo Images," Obersulm, 2012.

### 2.3.1. Generic Stereovision Pipeline

Stereovision is a concept that mimics the human vision system, leveraging two or more unique points of view to generate depth information and reconstruct a three-dimensional rendering of a scene. Depth is perceived by means of the relative shifts in the perspective of different components within the scene.



Fig 2.8: Stereovision Pipeline

On acquisition of the dual images, the first step of stereovision is to calibrate the cameras. This is done to understand the cameras' pose relation to the external world, as well as to the other camera in the system. The object most commonly used for calibration is a checkerboard of a known size. The 3D coordinates of the checkerboard pattern, and the camera model can be found using their pixel locations in the left and right images. The stereo camera model consists of intrinsic matrices of each camera's distortions and focal lengths, as well as the extrinsic matrix – comprising of information regarding the cameras' difference in pose with respect to each other. Stereovision can also be performed without camera calibration, but at the cost of additional computational complexity. The disadvantage associated with an uncalibrated approach is that the scene may only be reconstructed with a sense of scale, but not with the knowledge of real world ground distances.

The next step is of un-distortion and rectification of the camera images. When an image is taken with a camera, the scene captured by the imager is distorted – the kind of distortion depending on the type of lens and/or mirror arrangement. The type of distortion in the case of fisheye lenses and catadioptric cameras is barrel distortion. Barrel distortion suggests that that magnification decreases with a change in distance from the optical axis. Illustrated in Fig 2.9 is the effect of barrel distortion on a grid pattern. Images and frames of video can be undistorted using Brown's model of distortion [11].

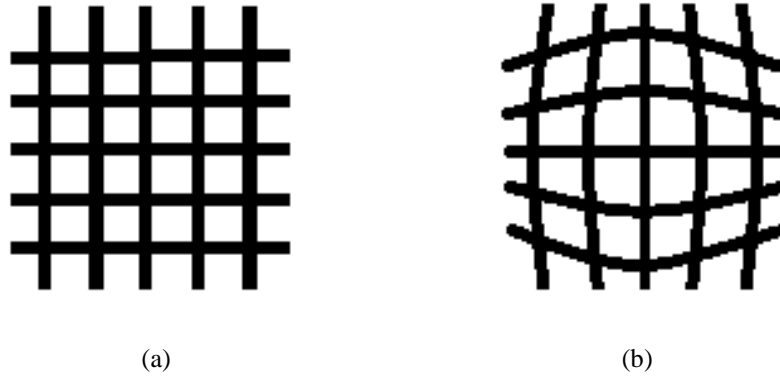


Fig. 2.9: Grid pattern (a) and the effect of barrel distortion (b).

The next stage is stereo-rectification. To perform this, it is necessary to understand the epipolar geometry of the camera setup. Assume  $I$  and  $I'$  are camera centers. The baseline is the line segment joining the two centers.

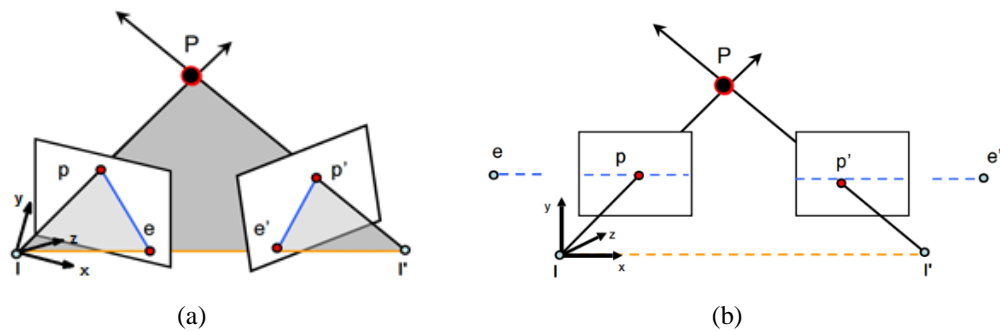


Fig 2.10: Frames from a dual camera setup (a), and Rectified frames (b) brought into a parallel image plane. Source: Silvio Savarese, “Chapter 6: Stereo Systems Multi-view Geometry”, Stanford University, 2016.

$P$  is a point that is viewed by the two cameras, at pixel positions  $p$  and  $p'$ .  $I - I' - P$  forms the epipolar plane. The baseline may or may not intersect the image planes. Fig 2.9(a) contains intersections of the baseline with the image planes at points  $e$  and  $e'$ , also known as the epipoles. The line segments passing through pairs  $(p, e)$  and  $(p', e')$  are the epipolar lines. The camera matrices found from the calibration stage, along with a rotation matrix and a translation vector are then used to warp the images and project them onto a parallel image plane, such that the epipoles lie at infinity as displayed in Fig 2.10(b).

Stereo-correspondence is the stage that comes after rectification. It is comprised of a pixel-wise search of every pixel in the left image, for its corresponding match in the right

image. This takes  $O(n^2)$  time to implement. But due to rectification previously performed on the images it is possible to implement this search in linear time, thus reducing processing time by an order of magnitude. The correspondence problem is fairly open ended due to issues of varying degrees of exposure between cameras, multiple homogeneous regions that have similar color intensities, occlusions and foreshortening to name a few. A commonly used practice is block matching, where blocks of  $N \times N$  pixels are searched, along with normalized cross correlation in order to mitigate issues in illumination. The result of the correspondence stage is a disparity map that provides pixel displacements – a relative sense of depth of objects in the scene.

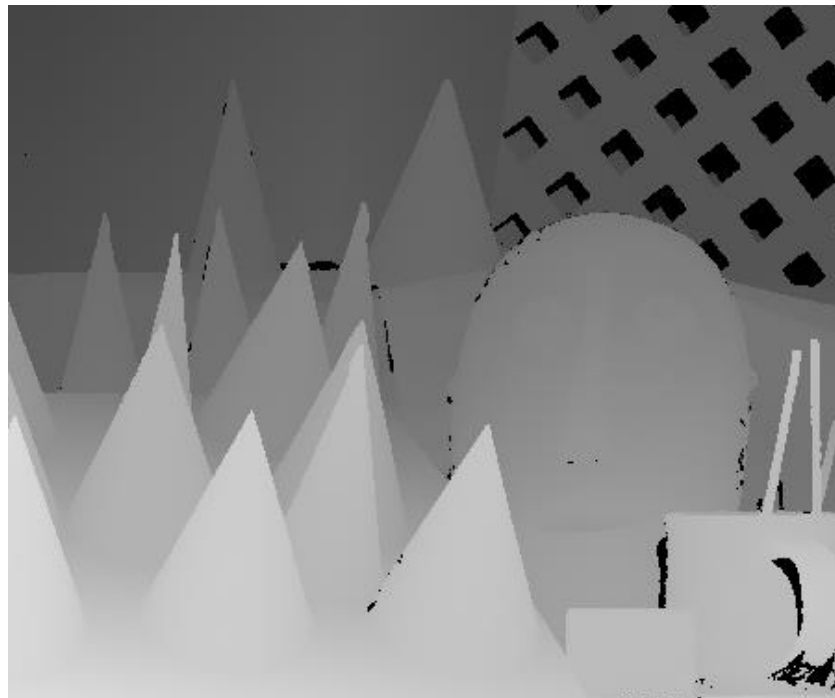


Fig 2.11: Disparity map of 'Cones' dataset

Source: Middlebury College Stereo Dataset 'Cones'

The final step in the stereovision process is triangulation and reconstruction – a representation of the stereo data into a three-dimensional space. The secondary advantage of rectification is that the triangulation process is reduced down to a similar triangles problem. The equation that is used to determine the depth of a point in the disparity map is  $z = B.F/d$ , where  $z$  is the depth in meters,  $B$  is the baseline in meters,  $F$  is the focal length in pixels and  $d$  is the pixel disparity. Fig 2.12 illustrates the same.



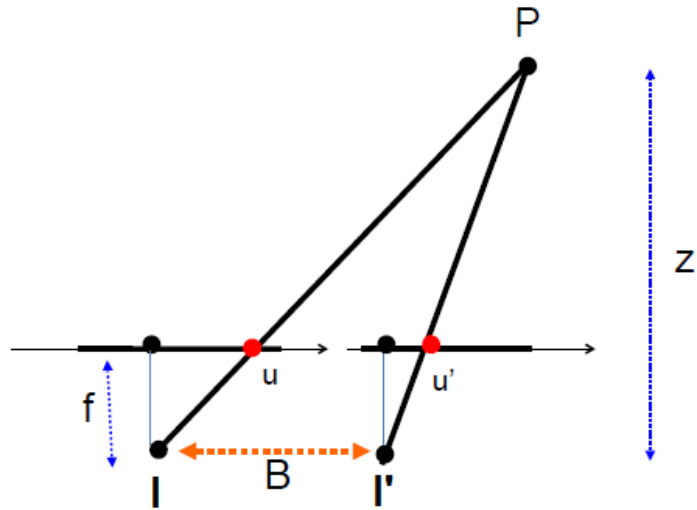


Fig 2.12: Point P at a distance Z from camera centers

Source: Silvio Savarese, "Chapter 6: Stereo Systems Multi-view Geometry", Stanford University, 2016.

Also note that the camera's FOV needs to be taken into account to be able to generate an accurate point cloud, again requiring the camera system's intrinsic matrices. The result of this stage is a three-dimensional point cloud as shown in Fig. 2.13.

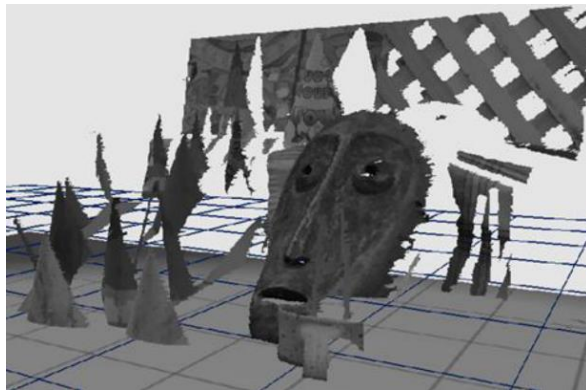


Fig 2.13: Three dimensional grayscale point cloud from 'Cones' Dataset

Source: Ensenso and IDS Imaging Development Systems GmbH, "Obtaining Depth from Stereo Images," Obersulm, 2012.

### 2.3.2. Catadioptric Stereoscapy

The use of catadioptric cameras is ideal for stereo vision when an entire panoramic scene is to be captured in 3D. The vertical field of view ( $+45^{\circ}$ ,  $-15^{\circ}$ ) makes the V.360 better suited for long range imaging applications such as autonomous driving, and aerial terrain

mapping. This project studies two arrangements for imaging the scene in 3D. The first arrangement places both cameras next to each other in the same horizontal plane as shown in Fig 2.14.



Fig. 2.14: Horizontal configuration for catadioptric stereoscopy

The issue associated with placing cameras side by side is the potential for occlusions on the left and right sides of the images as illustrated by Fig 2.15. The angle of occlusion  $\theta$  is a function of the length of the baseline  $b$  and the size of the camera.

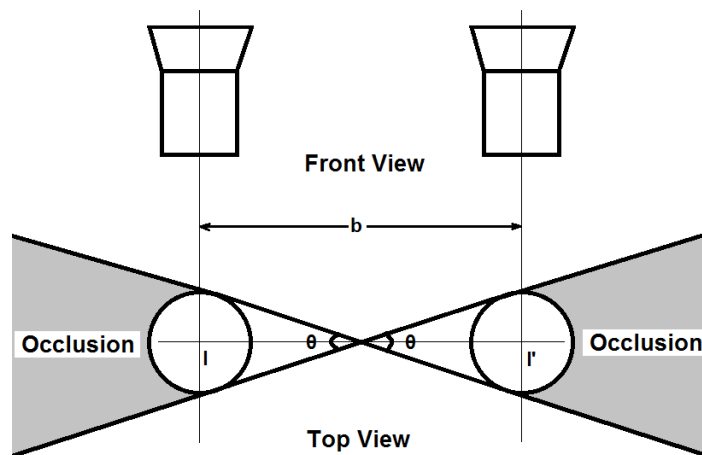


Fig 2.15: Occlusions in a horizontal configuration of catadioptric stereovision

Generic methods for stereovision are aimed at cameras with a horizontal FOV of less than or equal to  $180^{\circ}$ . The first set of tests used these standard techniques with each equirectangular panoramic frame split in half, such that a disparity map would be

computed for the parts of the frame pair facing forwards, and a separate disparity computation for frame pair facing the rear half of the scene. This setup was thus rejected.



Fig. 2.16: Vertical configuration for catadioptric stereoscopy

The second setup implemented was placement of the cameras vertically, such that their optical axes coincide, as shown in Fig. 2.16. The vertical configuration allows for a better horizontal FOV with negligible occlusion caused by the mounting structure. In this case vertical disparity is calculated, for the entire frame in a single iteration, contrary to the previous configuration. Due to lack of available information about the cameras' intrinsic parameters it is not possible to measure real world distances through the camera system alone. The proposed method is extended to make use of an Xbox Kinect to generate ground truths of a fraction of the scene, thereby enabling the system to get an anchor point in the real world – and disparity values can be accordingly mapped to real world depths.

## 2.4. NVIDIA Jetson TK1

This project proposal received a hardware grant of a Jetson TK1 from NVIDIA Corp. The Jetson TK1 is a single board computer that runs Ubuntu 14.04 with preconfigured drivers. It features a Tegra K1 SOC (system on chip) which comprises of a quad core,

2.3GHz ARM Cortex-A15 CPU, a GK20A (192 core) GPU based on the Kepler microarchitecture, and an ISP on the same chip.

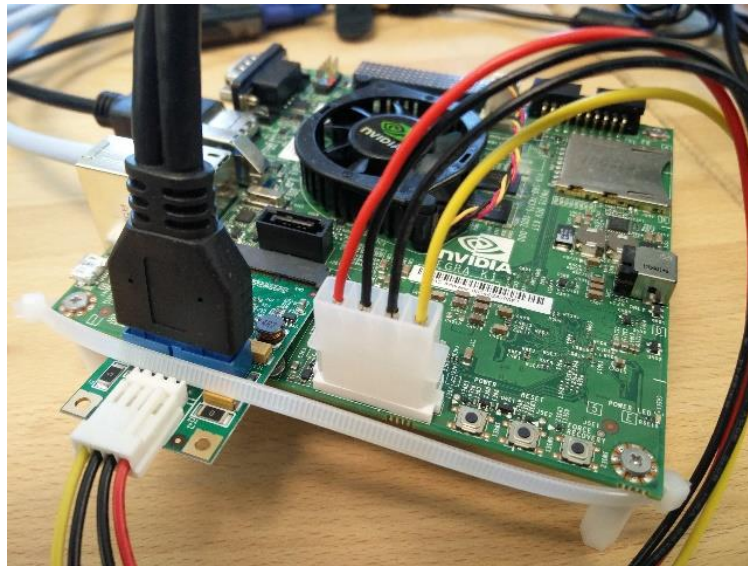


Fig 2.17: Jetson TK1 with a USB to mini-PCIe converter

The TK1 provides 2 gigabytes of DDR3 Dynamic-RAM, and 16GB of eMMC storage. Additional storage can be provided via an external SD/MMC card, a slot for which is present on the board. Multiple means of communication are possible using USB 2/3.0, or via the RS232 or Ethernet ports available on the TK1. In this project, most of the experiments with the TK1 have been with the Kinect, whose open source USB drivers have an unresolved issue where the camera needs to be disconnected and reconnected every time the TK1 is booted up. A USB to mini-PCIe converter is connected to the TK1's mini-PCIe port to bypass this issue. Figure 2.16 shows the Jetson TK1 with a Syba USB to mini-PCIe converter plugged in. The converter draws power from the TK1's onboard power supply.

The Jetson TK1 was designed to be the target platform for performing stereovision along with initial prototyping on a laptop, but compatibility issues were encountered during migration from OpenCV 3.1 on the laptop to an OpenCV4Tegra based build environment on the Jetson TK1. Hence the final build was implemented on a Dell Inspiron 15 7559 equipped with an Intel Core i7-6700HQ processor and 8GB RAM.

## 3. Software

### 3.1. Development with VS2013 and OpenCV

The software components in this system were built on Visual Studio 2013. OpenCV libraries were included. Together they provide a comprehensive platform for rapid prototyping, development and testing of computer vision applications.

Visual Studio was used for its exceptional code editor and debugging capabilities. The code editor supports syntax highlighting and automatic code completion suggestions for various components that may be included or previously linked. It also supports bookmarks, collapsing code blocks and incremental search options, in addition to normal text search methods. Other noteworthy functionalities include code refactoring, interface extraction and encapsulation. The feature of VS2013's code editor that proved to be the most useful to this project is the background compilation tool, which performs code compilation in the background as it is being written, and returns possible syntax/compilation errors that would be potentially encountered upon actual compilation. The VS2013 debugger is efficient with both source, and machine level debugging operations. It features breakpoints, step by step debugging and allows for code to be edited as it is being debugged. It can also provide the disassembly if a particular source is unavailable, and viewing options for the memory dump.

OpenCV (Open-source Computer Vision) was the C++ library used to speed up development of the stereovision pipeline. It provides functions that aim towards real-time computer vision applications that are independent of the hardware, operating system, and window-managers, although further GPU acceleration is also possible by means of CUDA or OpenCL support. OpenCV offers features spanning image and video frame manipulation, I/O, specialized data structures, matrix and vector algebra, structure and motion analysis, camera calibration, object recognition, labelling, and UI tools.

## 3.2. Stereo Correspondence with Semi Global Block Matching

### 3.2.1. Theory

Semi global block matching (SGBM) is a stereo correspondence method that depends on the concept of Mutual Information, and a global two-dimensional smoothness constraint approximation by means of multiple one-dimensional constraints. The algorithm presented by H. Hirschmuller [5] computes pixel matching costs based on the Mutual Information method. The cost function  $C(p,d)$  at pixel  $p$  and disparity  $d$  is calculated for the rectified images  $I_L$  and  $I_R$ , as follows:

$$C(p, d) = \min(d(p, p - d, I_L, I_R), d(p - d, p, I_R, I_L)) \quad (1)$$

Where

$$d(p, p - d, I_L, I_R) = \min_{p-d-0.5 \leq q \leq p-d+0.5} |I_L(p) - I_L(q)| \quad (2)$$

The SGBM algorithm then minimizes an energy function to get a better quality disparity map for a pair of images as illustrated in (3).

$$E(D) = \sum_p (C(p, D_p) + \sum_{q \in N_p} P_1 I[|D_p - D_q| = 1] + \sum_{q \in N_p} P_2 I[|D_p - D_q| > 1]) \quad (3)$$

$E(D)$  is the disparity image energy,  $p$  and  $q$  are the pixel locations, and  $N_p$  is the eight-connected neighborhood of pixel  $p$ .  $P_1$  and  $P_2$  are penalties for a change of disparity equal to 1 and greater than 1 respectively, amongst two neighboring pixels.  $P_2$  is always externally set by to be greater than or equal to  $P_1$ .  $I[x]$  is a function that returns a 1 if the argument  $x$  is true, and otherwise returns a 0.

Performing the aforementioned 2D minimization for an entire image space is an NP-Complete problem. To reduce complexity SGBM performs multiple 1D minimizations in different directions to approximate the 2D minimization. Costs are aggregated by the matching function on multiple paths that converge on the corresponding pixel being

considered, as illustrated in (4). This multiple path approach is highly advantageous to unrectified stereovision methodologies because the pixel search is not purely vertical.

$$S(p, d) = \sum_r L_r(p, d) \quad (4)$$

Where

$$L_r(p, d) = C(p, d) + \min[L_r(p - r, d), L_r(p - r, d - 1) + P_1, L_r(p - r, d + 1) + P_1, \min_i L_r(p - r, i) + P_2] - \min_k L_r(p - r, k) \quad (5)$$

In equation (4),  $S(p,d)$  is the aggregate cost for pixel  $p$  with disparity  $d$ .  $r$  is the direction that pixel  $p$  is converged to, and  $L_r(p,d)$  is the minimum cost in the  $r$  direction.  $L_r(p,d)$  is computed in (5) where  $C(p,d)$  is added to the minimum of the previous pixel's cost with disparity  $d$ , previous pixel's cost with  $d \pm 1$  with an additional penalty  $P_1$ , and previous pixel's cost with  $d$  beyond the range of  $\pm 1$  with an additional penalty  $P_2$ . The minimum value of the previous pixel's minimum cost is subtracted to limit the monotonically increasing  $L_r(p,d)$  for a particular path. The maximum value  $L_r(p,d)$  can take is the maximum value of  $C(p,d) + P_2$ . The computational complexity for the algorithm is  $O(\text{width} * \text{height} * \text{number of disparities})$ .

### 3.2.2. Implementation and Results

The original implementation by H. Hirschmuller made 1D minimizations in eight directions, and is a two pass algorithm. The OpenCV implementation of the same, in this project makes minimizations in 5 directions, and uses the method deployed by Birchfield and Tomasi in [7] to compute cost. The program workflow is shown in Fig. 3.1 that process subsequent frames of live video feed from both cameras to compute and display the disparity map. Note that frames are rotated by  $90^0$  in the clockwise direction. This is done so that the vertical disparity computation problem is then converted to a horizontal disparity computation problem, which the OpenCV implementation is designed to handle by default. The disparity map obtained by this operation is then required to be rotated by  $90^0$  in the counter-clockwise direction to align with the original video feed from the cameras.

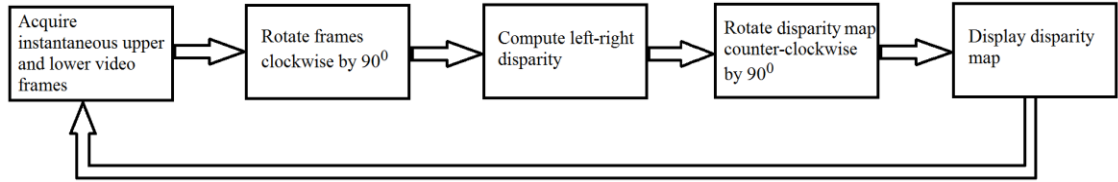


Fig. 3.1: Program Workflow

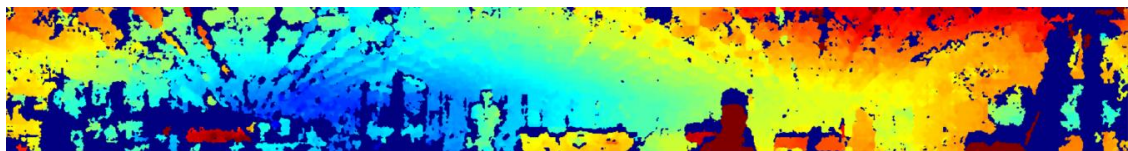
Fig 3.2(c) Illustrates a snapshot of the disparity map generated from parallel streams of size 1920 x 320, from the two cameras in a vertical stereo configuration. The disparity map is represented as an RGB image where the objects closer to the camera system are represented by the red end of the visible color spectrum, and objects farther away in the scene are visible in blue. Note that the disparity map is not of the same height as the original frames due to non-overlapping regions in the frames taken by the cameras.



(a)



(b)



(c)

Fig. 3.2: Upper camera image (a), Lower camera image (b), and Disparity Map (c).

Due to an uncalibrated, and minimally rectified approach to stereovision, it is not possible to calculate real world depths from the disparity map. Hence, a Kinect V2 was used to set up a set of six calibration points (objects in the scene) and their ground truth depths were obtained with an accuracy of 1cm. The calibration points are shown in Fig. 3.3.





Fig. 3.3: Calibration Scene

Sr.	Location in Scene	Depth in m (Kinect)	Disparity in pixels (system)
1.	Whiteboard	5.39	135
2.	Chair Backrest	4.05	153
3.	Quadrotor Enclosure (left)	2.87	184
4.	Quadrotor Enclosure (right)	3.28	171
5.	Monitor	2.75	197
6.	Function Generator	2.06	216

Table 3.1: Disparity-Depth Calibration

The depth values were then assigned to the corresponding points in the disparity map, hence creating a reference for predicting depth values for any disparity level. This predictive reference curve in Fig. 3.4 was then fit, based on the inversely proportional relation between the disparity and the z axis distance.

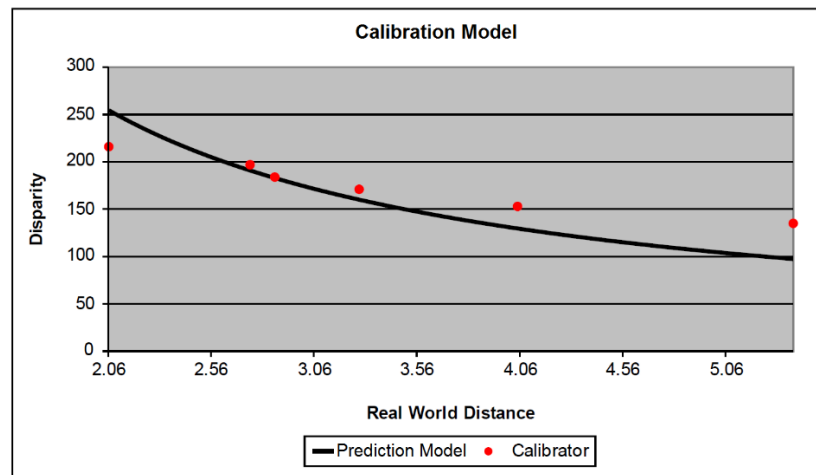


Fig 3.4: Curve fitting with calibration points

The entire setup was then moved to a second location, where a set of new points were selected for testing the prediction model, as displayed in Fig. 3.5. Ground truth depths from the Kinect V2 were then compared with predicted depths for corresponding

disparities at the test points and the system's error was computed, as illustrated by Table 3.2. The system processes video at 1.56 frames per second.



Fig 3.5: Test Scene

Sr.	Location in Scene	Disparity in pixels (system)	Predicted Depth in m (ref. model)	Ground Truth Depth in m (Kinect)	Error in m,   Predicted Depth – Ground Truth
1.	Monitor	127	4.13	4.18	0.05
2.	Quadrotor Enclosure (left)	140	3.75	3.42	0.33
3.	Quadrotor Enclosure (right)	134	3.92	3.52	0.40
4.	Chair Backrest	109	4.82	4.29	0.53
5.	Whiteboard	130	4.04	3.63	0.41

Table 3.2: Test Scene Results

It was noted that as per the inverse relation between disparity and depth, the disparity monotonically decreased for an increase in real world depths. Furthermore, with the exception of observation number 1. the prediction error increases with an increase in the distance of an object from the camera system. The system could potentially lower the error in depth prediction if the cameras' intrinsic parameters were known, and if the stereo camera setup were calibrated.

### 3.3. Automated Rectification

Although algorithms such as SGBM perform reasonably well in case of uncalibrated and unrectified, and hand rectified stereovision systems, rectification helps reduce the search complexity to a nearly unidimensional space.

It was noted that the primary sources of non-rigidity in the setup's position adjustment system was a roll in the end grippers of the mounts as displayed in Fig. 3.6. Hence there was a need for a method to automatically warp one of the two video feeds, such that matching features in the stereo pair would align vertically, thus mitigating the effects of non-rigidity from the given sources, and all other possible forms due to factors such as vibration, if mounted on a moving platform. Automated rectification was thus the preferred next step.



Fig 3.6: Primary sources of non-rigidity in the position adjustment system

In an uncalibrated system the concept of automated rectification revolves around understanding the transformation between the images and then aligning one with respect to the other to rectify the pair of frames. The designed pipeline performs the following operations on a stereo pair of images or frames of video, as illustrated in Fig 3.7.



Fig 3.7: Automated Rectification Pipeline

The first step was to detect key features in the pair. A feature detector was used to accomplish the same. The next step consisted of finding descriptors of the detected

features. The SURF algorithm by Bay et. al. [4] was chosen because of its ability to detect features, as well as provide descriptors for those features. After descriptor extraction, the descriptors were then matched using the K Nearest Neighbors technique using FLANN (Fast Library for Approximate Nearest Neighbors) [6]. The matches are then subject to RANSAC (Random Sample Consensus) [8] to retain only the good matches.



Fig. 3.8: Tilted upper camera test for Automated Rectification system

The homography matrix between the two frames was then calculated to compute the perspective transformation between the pair of frames. The top image was then warped to be aligned with the lower image, such that the features aligned vertically. The system was tested by generating a random tilt in the orientation of the upper camera as shown in Fig. 3.8. The results obtained are displayed in Fig. 3.9.



(a)



(b)



(c)

Fig. 3.9: Automated Rectification Testing; (a) Tilted upper camera input, (b) Lower camera input, (c) Rectified Upper Camera feed.

The input in this test case was split into two halves; one facing away from the mounting apparatus, and the other facing towards it. Since the rotation on one of these halves is the exact opposite of that of the other (refer fig. 3.6), the algorithm needs to be applied to only one half, until the warp stage. The same homography matrix was then modified to reverse the rotation and was applied to the other half, thus decreasing the execution time of the pipeline by a factor of two.

It was noted that the rectification of the central region was rectified to a greater extent as compared to the sides. This is because the point of tilt was assumed to be the center of the frame, but it could be at any position that may be. To be able to counter this limitation, one could implement a motion estimation algorithm between upper and lower frames and compute the coordinates of the point around which multiple motion vectors revolve. This information could then be used to modify the homography matrix and work adaptively because some regions of the image require more warping with regards to the others.

## 4. Conclusion and Future Work

The objective of this project was to explore an alternative means to panoramic 3D imaging. The proposed system generates a panoramic depth image of a scene. The automated rectification technique presented in this project can potentially solve issues in camera position inconsistencies in real time that usually arise in moving camera systems. It was noted that the automated rectification method in section 3.3 aligned the upper frame to the maximum possible extent with the reference frame (lower frame), thus yielding incorrect vertical disparity values. A possible solution to the same would be to compare subsequent frames from the same camera (for both feeds of video) using a motion estimation algorithm, and establishing smoothness constraints between the two feeds of video; the violation of which could indicate and measure a change in the orientation of one camera with respect to the other, thereby preventing the need for re-calibration.

Additional performance improvements are possible via parallelization of the block matching and feature detection sections using a GPU acceleration framework such as CUDA. The system can also be scaled to cover larger depth ranges by increasing the baseline distance. Furthermore, a high-accuracy point cloud can be generated if the intrinsic parameters of the cameras are known, which in turn can enable stereo calibration and hence a better means to estimating depth.

A future implementation of such a system could be equipped with high-speed wireless streaming capabilities, and cloud based computing by means of an Amazon EC2-G2 GPU cluster that could generate and broadcast point clouds in real-time, thus creating a truly immersive VR livestreaming experience.

# References

- [1] R. Benosman and S. B. Kang, *Panoramic Vision: Sensors, Theory, and Applications*, New York: Springer Science+Business Media , 2001.
- [2] M. Schonbein, *Omnidirectional Stereo Vision for Autonomous Vehicles*, Karlsruhe: KIT Scientific Publishing, 2014.
- [3] Ensenso and IDS Imaging Development Systems GmbH, "Obtaining Depth from Stereo Images," Obersulm, 2012.
- [4] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," *9th European Conference on Computer Vision*, vol. 3951, pp. 404-417, 2006.
- [5] H. Hirschmuller, "Stereo Processing by Semi-Global Matching and Mutual Information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30(2), pp. 328-341, 2008.
- [6] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," in *International Conference on Computer Vision Theory and Applications* , Lisboa, Portugal, 2009.
- [7] S. Birchfield and C. Tomasi, "Depth Discontinuities by Pixel-to-Pixel Stereo," in *IEEE International Conference on Computer Vision*, Bombay, India, 1998.
- [8] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," in *Communications of the ACM*, New York, USA , 1981.
- [9] B. K. Cabral, "Introducing Facebook Surround 360: An open, high-quality 3D-360 video capture system," Facebook, 12 April 2016. [Online]. Available: <https://code.facebook.com/posts/1755691291326688/introducing-facebook-surround-360-an-open-high-quality-3d-360-video-capture-system/>.
- [10] Lytro, "The Creative benefits of Light Field," Lytro, 5 April 2016. [Online]. Available: <http://blog.lytro.com/post/142305362395/the-creative-benefits-of-light-field>.
- [11] D. C. Brown, "Decentering Distortion of Lenses," *Photogrammetric Engineering*, vol. 32, p. 444–462, 1966.

# Appendix A: Source code for computing disparity

```
#include <iostream>
#include "opencv2/imgcodecs.hpp"
#include <opencv2/core.hpp>
#include "opencv2/calib3d.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/core/utility.hpp"
#include <string>

using namespace cv;
using namespace std;
bool flag = 0;

Mat disp1, disp1_8, disp2, disp2_8;
void mystereo(Mat m1, Mat m2);

int main(int argc, char* argv[])
{
    Mat Front_left, Front_right, Back_left, Back_right;

    VideoCapture capL(0);
    // open the video camera no. 0
    //make sure (0) is mentioned in the same line, or cap.open(0) is specified before
    //cap.set, BugFix #948 for OpenCV
    VideoCapture capR(1);

    if (!capL.isOpened()) // if not success, exit program
    {
        cout << "Cannot open the video cam Left" << endl;
        return -1;
    }
    if (!capR.isOpened()) // if not success, exit program
    {
        cout << "Cannot open the video cam Right" << endl;
        return -1;
    }

    capL.set(CV_CAP_PROP_FRAME_WIDTH, 1920);
```



```

//set the width of frames of the video // 1920, 2880, 3840, 6480
capL.set(CV_CAP_PROP_FRAME_HEIGHT, 320);
//set the height of frames of the video // 320, 480, 640, 1080

capR.set(CV_CAP_PROP_FRAME_WIDTH, 1920);
//set the width of frames of the video // 1920, 2880, 3840, 6480
capR.set(CV_CAP_PROP_FRAME_HEIGHT, 320);
//set the height of frames of the video // 320, 480, 640, 1080

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

//cout << "Frame size : " << dWidth << " x " << dHeight << endl;

//namedWindow("MyVideo", CV_WINDOW_AUTOSIZE);
//create a window called "MyVideo"
//namedWindow("MyVideo", CV_WINDOW_KEEPRATIO); //works
//namedWindow("MyVideo", CV_WINDOW_FREERATIO);

//namedWindow("MyVideo Left", CV_WINDOW_NORMAL);
//namedWindow("MyVideo Right", CV_WINDOW_NORMAL);

namedWindow("MyVideo Top", CV_WINDOW_FREERATIO);

//namedWindow("MyVideo Left", CV_WINDOW_FREERATIO);
namedWindow("MyVideo Bottom", CV_WINDOW_FREERATIO);
namedWindow("D1", CV_WINDOW_FREERATIO);
namedWindow("D2", CV_WINDOW_FREERATIO);
//namedWindow("Front", CV_WINDOW_FREERATIO);
int flag = 1;

while (1)
{
    Mat frameL, frameR;

    capL >> frameL;
    capR >> frameR;

    frameL = frameL(Rect(0, 380, 1920, 320));
    frameR = frameR(Rect(0, 380, 1920, 320));

    imshow("MyVideo Top", frameL);
    //show the frame in the "MyVideo Left" window
    imshow("MyVideo Bottom", frameR);
    //show the frame in the "MyVideo Right" window

```

```

transpose(frameL, frameL);
flip(frameL, frameL, 0);
transpose(frameR, frameR);
flip(frameR, frameR, 0);

Front_left = frameL;
Front_right = frameR;

//Back_left = frameR(Rect(frameR.cols / 2, 0, frameR.cols / 2, frameR.rows));
//Back_right = frameL(Rect(frameL.cols / 2, 0, frameL.cols / 2, frameL.rows));

mystereo(Front_left, Front_right); //outwards
//works well
normalize(dis1, disp1_8, -150, 455, CV_MINMAX, CV_8U);
normalize(dis2, disp2_8, -150, 455, CV_MINMAX, CV_8U);
//normalize(dis1, disp1_8, 0, 255, CV_MINMAX, CV_8U);
//normalize(dis2, disp2_8, 0, 255, CV_MINMAX, CV_8U);

applyColorMap(disp1_8, disp1_8, COLORMAP_JET);
transpose(disp1_8, disp1_8);
flip(disp1_8, disp1_8, 1);
//disp8 = disp8(Rect(0, 380, disp8.cols, disp8.rows - 375 - 385));
applyColorMap(disp2_8, disp2_8, COLORMAP_JET);
transpose(disp2_8, disp2_8);
flip(disp2_8, disp2_8, 1);

imshow("D1", disp1_8);
imshow("D2", disp1_8);

//mystereo(Back_left, Back_right); //outwards
////normalize(disp, disp8, -135, 455, CV_MINMAX, CV_8U);
//normalize(disp, disp8, -90, 455, CV_MINMAX, CV_8U);
//applyColorMap(disp8, disp8, COLORMAP_JET);
//disp8 = disp8(Rect(0, 400, disp8.cols - 40, disp8.rows - 400 - 300));
//imshow("Back", disp8);

if (flag == 1)
    cout << "rows: " << disp1_8.rows << "\t" << "cols: " << disp1_8.cols
    << endl;
flag = 0;

if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed,
    // break loop
{
    cout << "esc key is pressed by user" << endl;
}

```

```

        break;
    }
}
return 0;
}

```

```

void mystereo(Mat m1, Mat m2)
{

```

```

    //StereoSGBM sbm;
    //sbm.SADWindowSize = 3;
    //sbm.numberOfDisparities = 192; //96;
    //sbm.preFilterCap = 63;
    //sbm.minDisparity = -39;
    //sbm.uniquenessRatio = 10;
    //sbm.speckleWindowSize = 200;
    //sbm.speckleRange = 32;
    //sbm.disp12MaxDiff = 1;
    //sbm.fullDP = false;
    //sbm.P1 = 216;
    //sbm.P2 = 864 * 2;
    //sbm(m1, m2, disp);

    //works decently well
    //Ptr<StereoSGBM> sbm
        = StereoSGBM::create(-39, 96, 5, 216 * 2, 864 * 2, -1, 63, 10, 100, 2);
    Ptr<StereoSGBM> sbm
        = StereoSGBM::create(-39, 96, 5, 216 * 2, 864 * 2, -1, 63, 10, 100, 2,
StereoSGBM::MODE_HH); //full-scale two-pass dynamic programming algorithm.It
will consume O(W*H*numDisparities) bytes
    sbm->compute(m1, m2, disp1);
    sbm->compute(m2, m1, disp2);

    //cv::StereoBM sbm;
    //sbm.state->SADWindowSize = 9;
    //sbm.state->numberOfDisparities = 112;
    //sbm.state->preFilterSize = 5;
    //sbm.state->preFilterCap = 1;
    //sbm.state->minDisparity = 0;
    //sbm.state->textureThreshold = 5;
    //sbm.state->uniquenessRatio = 5;
    //sbm.state->speckleWindowSize = 0;
    //sbm.state->speckleRange = 20;
    //sbm.state->disp12MaxDiff = 64;
    //sbm(m1, m2, disp);
    //normalize(disp, disp8, 0.1, 255, CV_MINMAX, CV_8U);

```

```
//StereoBM sbm;  
//sbm.state->SADWindowSize = 5;  
//sbm.state->numberOfDisparities = 112;  
//sbm.state->preFilterSize = 5;  
//sbm.state->preFilterCap = 61;  
//sbm.state->minDisparity = -39;  
//sbm.state->textureThreshold = 507;  
//sbm.state->uniquenessRatio = 0;  
//sbm.state->speckleWindowSize = 0;  
//sbm.state->speckleRange = 8;  
//sbm.state->disp12MaxDiff = 1;  
//sbm(m1, m2, disp);  
  
}
```

## Appendix – B:

# Source code for automated rectification

```
#include <iostream>  
#include "opencv2/core/core.hpp"  
#include "opencv2/calib3d/calib3d.hpp"  
#include <opencv2/highgui/highgui.hpp>  
#include <opencv2/imgproc/imgproc.hpp>  
#include "opencv2/contrib/contrib.hpp"  
#include "opencv2/nonfree/nonfree.hpp"  
#include "opencv2/features2d/features2d.hpp"  
#include "opencv2/nonfree/features2d.hpp"  
#include <opencv/cv.h>  
  
#include <stdio.h>  
#include <string.h>  
  
#define PI 3.14159265  
  
using namespace cv;  
using namespace std;
```

```

bool flag = 0;

Mat disp;
void mystereo(Mat m1, Mat m2);

char *windowDisparity = "Disparity";
char *windowDisparitySGM = "Disparity of SGM";
char *windowMatch = "TP matched";

void rotate(cv::Mat& originalImage, cv::Mat& rotatedImage, cv::InputArray rotated,
            cv::Mat& dst) {
    std::vector<cv::Point2f> original(4);
    original[0] = cv::Point(0, 0);
    original[1] = cv::Point(originalImage.cols, 0);
    original[2] = cv::Point(originalImage.cols, originalImage.rows);
    original[3] = cv::Point(0, originalImage.rows);

    dst = cv::Mat::zeros(originalImage.rows, originalImage.cols, CV_8UC3);
    cv::Mat transform = cv::getPerspectiveTransform(rotated, original);
    cv::warpPerspective(rotatedImage, dst, transform, dst.size());
}

float angleBetween(const Point &v1, const Point &v2)
{
    float len1 = sqrt(v1.x * v1.x + v1.y * v1.y);
    float len2 = sqrt(v2.x * v2.x + v2.y * v2.y);

    float dot = v1.x * v2.x + v1.y * v2.y;

    float a = dot / (len1 * len2);

    if (a >= 1.0)
        return 0.0;
    else if (a <= -1.0)
        return Pi;
    else{
        int degree;
        degree = acos(a) * 180 / Pi;
        return degree;
    }
}

int main(int argc, char* argv[])
{
    Mat Front_left, Front_right, Back_left, Back_right;

```

```

Mat imgLeft, imgRight;
Mat outputLeft, outputRight;
Mat descriptors1, descriptors2;
Mat img_matches;
Mat disp, disp8U;

VideoCapture capL(0);
VideoCapture capR(1);

if (!capL.isOpened()) // if not success, exit program
{
    cout << "Unable to open video cam Left" << endl;
    return -1;
}
if (!capR.isOpened()) // if not success, exit program
{
    cout << "Unable to open video cam Right" << endl;
    return -1;
}

capL.set(CV_CAP_PROP_FRAME_WIDTH, 1920); //set the width of frames of the video
// 1920, 2880, 3840, 6480
capL.set(CV_CAP_PROP_FRAME_HEIGHT, 1080); //set the height of frames of the
//video 320, 480, 640, 1080

capR.set(CV_CAP_PROP_FRAME_WIDTH, 1920); //set the width of frames of the video
// 1920, 2880, 3840, 6480
capR.set(CV_CAP_PROP_FRAME_HEIGHT, 1080); //set the height of frames of the
//video 320, 480, 640, 1080

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

//cout << "Frame size : " << dWidth << " x " << dHeight << endl;

//namedWindow("MyVideo", CV_WINDOW_AUTOSIZE);
//create a window called "MyVideo"
//namedWindow("MyVideo", CV_WINDOW_KEEPRATIO); //works
//namedWindow("MyVideo", CV_WINDOW_FREERATIO);

//namedWindow("MyVideo Left", CV_WINDOW_NORMAL);
//namedWindow("MyVideo Right", CV_WINDOW_NORMAL);

//namedWindow("MyVideo Left", CV_WINDOW_FREERATIO);

```

```

namedWindow("MyVideo Top", CV_WINDOW_FREERATIO);
namedWindow("MyVideo Bottom", CV_WINDOW_FREERATIO);

namedWindow("Rectified Top", CV_WINDOW_FREERATIO);
//namedWindow("Front", CV_WINDOW_FREERATIO);
//namedWindow("Front", CV_WINDOW_FREERATIO);
int flag = 1;

//namedWindow(windowMatch, CV_WINDOW_NORMAL);
namedWindow(windowDisparitySGM, CV_WINDOW_FREERATIO);

waitKey(1000);

while (1)
{
    Mat frameL, frameR;

    capL >> frameL;
    capR >> frameR;

    imshow("MyVideo Top", frameL);
    //show the frame in the "MyVideo Left" window
    imshow("MyVideo Bottom", frameR);
    //show the frame in the "MyVideo Right" window

    cvtColor(frameL, frameL, CV_BGR2GRAY);
    cvtColor(frameR, frameR, CV_BGR2GRAY);

    //transpose(frameL, frameL);
    //flip(frameL, frameL, 0);
    //transpose(frameR, frameR);
    //flip(frameR, frameR, 0);

    //frameL = frameL(Rect(380, 0, frameL.cols - 760, frameL.rows));
    //frameR = frameR(Rect(380, 0, frameR.cols - 760, frameR.rows));
    //cout << "done" << endl;

    imgLeft = frameL;
    imgRight = frameR;

    if (!imgLeft.data || !imgRight.data)
    {
        std::cout << " --(!) Error reading images " << std::endl; return -1;
    }
}

```

```

Mat char1 = Mat(frameR, Rect(480, 380, frameR.cols - 960, frameR.rows -
760));
Mat image = Mat(frameL, Rect(480, 380, frameL.cols - 960, frameL.rows -
760));

transpose(char1, char1);
flip(char1, char1, 0);
transpose(image, image);
flip(image, image, 0);

//imshow("bottom", char1);
//imshow("topimg", image);

//waitKey(0);

//Detect the keypoints using SURF Detector
int minHessian = 200;

SurfFeatureDetector detector(minHessian);
std::vector<KeyPoint> kp_object;

detector.detect(char1, kp_object);

//Calculate descriptors (feature vectors)
SurfDescriptorExtractor extractor;
Mat des_object;

extractor.compute(char1, kp_object, des_object);

FlannBasedMatcher matcher;
std::vector<Point2f> obj_corners(4);

//Get the corners from the object
obj_corners[0] = cvPoint(0, 0);
obj_corners[1] = cvPoint(char1.cols, 0);
obj_corners[2] = cvPoint(char1.cols, char1.rows);
obj_corners[3] = cvPoint(0, char1.rows);

//Mat frame;
Mat des_image, img_matches;
std::vector<KeyPoint> kp_image;
std::vector<vector<DMatch >> matches;
std::vector<DMatch > good_matches;
std::vector<Point2f> obj;
std::vector<Point2f> scene;
std::vector<Point2f> scene_corners(4);

```



```

Mat H;
Mat result = Mat(320, 960, CV_8U);
Mat char2 = Mat(320, 960, CV_8U);
Mat char2_left = char2(Rect(0, 0, 480, 320));
Mat char2_right = char2(Rect(480, 0, 480, 320));

Mat(frameR, Rect(1440, 380, 480, 320)).copyTo(char2_left);
Mat(frameR, Rect(0, 380, 480, 320)).copyTo(char2_right);

Mat image2 = Mat(320, 960, CV_8U);
Mat image2_left = image2(Rect(0, 0, 480, 320));
Mat image2_right = image2(Rect(480, 0, 480, 320));

Mat(frameL, Rect(1440, 380, 480, 320)).copyTo(image2_left);
Mat(frameL, Rect(0, 380, 480, 320)).copyTo(image2_right);

Mat result2;// = Mat(320, 960, CV_8U);

Mat top = Mat(320, 1920, CV_8U);
Mat top_left = top(Rect(0, 0, 480, 320));
Mat top_middle = top(Rect(480, 0, 960, 320));
Mat top_right = top(Rect(1440, 0, 480, 320));

detector.detect(image, kp_image);
extractor.compute(image, kp_image, des_image);

matcher.knnMatch(des_object, des_image, matches, 2);

for (int i = 0; i < min(des_image.rows - 1, (int)matches.size()); i++)
//THIS LOOP IS SENSITIVE TO SEGFAULTS
{
    if ((matches[i][0].distance < 0.6*(matches[i][1].distance)) &&
        ((int)matches[i].size() <= 2 && (int)matches[i].size()>0))
        //if ((matches[i][0].distance < 0.6*(matches[i][1].distance)) &&
            ((int)matches[i].size() <= 5 && (int)matches[i].size()>0))
        {
            good_matches.push_back(matches[i][0]);
        }
}

//Draw only "good" matches

if (good_matches.size()> 5) {

```

```

drawMatches(char1, kp_object, image, kp_image, good_matches,
img_matches, Scalar::all(-1), Scalar::all(-1), vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

for (int i = 0; i < good_matches.size(); i++)
{
    //Get the keypoints from the good matches
    obj.push_back(kp_object[good_matches[i].queryIdx].pt);
    scene.push_back(kp_image[good_matches[i].trainIdx].pt);
    //cout << angleBetween(obj[i], scene[i]) << endl;
    //angles between images
}

H = findHomography(obj, scene, CV_RANSAC);

perspectiveTransform(obj_corners, scene_corners, H);

// cout<<angleBetween(obj[0], scene[0])<<endl;
//cout << scene_corners << endl;

rotate(char1, image, scene_corners, result);
transpose(result, result);
flip(result, result, 1);

//second set

//std::vector<Point2f> scene_corners2(4);
//scene_corners2[0] = scene_corners[1];
//scene_corners2[1] = scene_corners[0];
//scene_corners2[2] = scene_corners[2];
//scene_corners2[3] = scene_corners[3];

transpose(image2, image2);
flip(image2, image2, 0);
transpose(char2, char2);
flip(char2, char2, 0);

rotate(char2, image2, scene_corners, result2);

transpose(result2, result2);
flip(result2, result2, 1);

//transpose(img_matches, img_matches);
//flip(img_matches, img_matches, 1);
//imshow("Good Matches", img_matches);

Mat(result2, Rect(480, 0, 480, 320)).copyTo(top_left);
result.copyTo(top_middle);

```

```

    Mat(result2, Rect(0, 0, 480, 320)).copyTo(top_right);

    imshow("Rectified Top", top);
    //imshow("result2", result2);

    //waitKey(0);

}

int ndisparities = 80; //16 * 2; /**< Range of disparity */
int SADWindowSize = 3; /**< Size of the block window. Must be odd */

StereoSGBM sgbm;

sgbm.preFilterCap = 63;
sgbm.SADWindowSize = SADWindowSize > 0 ? SADWindowSize : 3;

int cn = outputLeft.channels();

sgbm.P1 = 216 * 2; //8 * cn*sgbm.SADWindowSize*sgbm.SADWindowSize;
sgbm.P2 = 864 * 2; //32 * cn*sgbm.SADWindowSize*sgbm.SADWindowSize;
sgbm.minDisparity = -39;
sgbm.numberOfDisparities = ndisparities;
sgbm.uniquenessRatio = 10;
sgbm.speckleWindowSize = 100;
sgbm.speckleRange = 2;
sgbm.disp12MaxDiff = 1;

sgbm(top, Mat(frameL, Rect(0, 380, frameL.cols, frameL.rows - 760)), disp);

double minVal; double maxVal;

minMaxLoc(disp, &minVal, &maxVal);

//disp.convertTo(disp8U, CV_8UC1, 255 / (maxVal - minVal));
normalize(disp, disp8U, -50, 255, CV_MINMAX, CV_8U);
applyColorMap(disp8U, disp8U, COLORMAP_JET);

//transpose(disp8U, disp8U);
//flip(disp8U, disp8U, 1);
//disp8U = disp8U(Rect(0, 380, disp8U.cols, disp8U.rows - 760));

imshow(windowDisparitySGM, disp8U);

```

```

    if (flag == 1) {
        cout << "rows: " << frameL.rows << "\t" << "cols: " << frameL.cols <<
            endl;
        cout << "rows: " << frameR.rows << "\t" << "cols: " << frameR.cols <<
            endl;
        cout << "rows: " << disp8U.rows << "\t" << "cols: " << disp8U.cols <<
            endl;
    }
    flag = 0;

    if (waitKey(30) == 27)
        //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
        {
            cout << "esc key is pressed by user" << endl;
            break;
        }
    }
    return 0;
}

```

# Appendix – C:

## VSN Mobil V. 360 User's Guide

### TJ MARTELL FOUNDATION

VSN MOBIL IS A PROUD SUPPORTER AND SPONSOR OF THE T.J. MARTELL FOUNDATION. THE T.J. MARTELL FOUNDATION IS THE MUSIC INDUSTRY'S LARGEST FOUNDATION THAT FUNDS INNOVATIVE MEDICAL RESEARCH FOCUSED ON FINDING CURES FOR LEUKEMIA, CANCER AND AIDS. THE FOUNDATION SOURCES AND SUPPORTS EARLY-STAGE RESEARCH PROJECTS AIMED AT DEVELOPING MORE EFFECTIVE CLINICAL TREATMENTS FOR PATIENTS WHICH OTHERWISE MIGHT NOT BE FUNDED. FOR MORE INFORMATION, PLEASE VISIT [WWW.TJMARTELL.ORG](http://WWW.TJMARTELL.ORG).

### LEGAL

V.360° AND VSN MOBIL ARE TRADEMARKS OF VSN TECHNOLOGIES, INC. IPHONE, IPAD AND APP STORE ARE REGISTERED TRADEMARKS OF APPLE INC. ANDROID AND GOOGLE PLAY ARE TRADEMARKS OF GOOGLE INC. © 2014 VSN MOBIL. ALL RIGHTS RESERVED.

P/N: QV360001A

 V.360°™



 V.360°™

### BASIC CARE

THE V.360° IS DESIGNED FOR ACTIVE LIFESTYLES. PROPER USE AND CARE OF THE V.360° IS ESSENTIAL FOR OPTIMAL PERFORMANCE TO KEEP THE CAMERA IN PROPER WORKING ORDER, MAKE SURE THE BATTERY DOOR IS PROPERLY INSTALLED TO PROVIDE A WATER-TIGHT SEAL. FAILURE TO DO SO WILL RESULT IN DAMAGE TO THE CAMERA.

### CLEANING INSTRUCTIONS

CLEAN CAMERA WITH THE INCLUDED MICROFIBER POUCH OR A SOFT CLOTH OR TOWEL. THE USE OF CLEANING AGENTS INCLUDING ISOPROPHYL ALCOHOL ARE NOT RECOMMENDED.

### STORAGE

FOR BEST RESULTS, STORE YOUR CAMERA INSIDE THE INCLUDED MICROFIBER POUCH AT ROOM TEMPERATURE IN A CLEAN AND DRY AREA.

## LEARN MORE

TO LEARN MORE ABOUT YOUR V.360° FEATURES AND SAFETY INFORMATION, PLEASE VISIT [WWW.V360LIFE.COM](http://WWW.V360LIFE.COM)

## GETTING SUPPORT

PLEASE VISIT [WWW.V360LIFE.COM/SUPPORT](http://WWW.V360LIFE.COM/SUPPORT) FOR DEVICE SUPPORT.

## SOFTWARE UPDATES

TO ENSURE YOUR CAMERA HAS THE LATEST FEATURES, VISIT [WWW.V360LIFE.COM](http://WWW.V360LIFE.COM) TO DOWNLOAD AND INSTALL SOFTWARE UPDATES.

## SD CARDS

ALL SD CARDS ARE NOT COMPATIBLE WITH THE V.360° CAMERA. FOR BEST RESULTS, PLEASE USE THE FOLLOWING SD CARD TYPES:

TYPE	SPEED	CAPACITY
MICROSDHC	CLASS 10, UHS-1 OR HIGHER	UP TO 128 GB
MICROSDXC	CLASS 10, UHS-1 OR HIGHER	UP TO 128 GB

## WARRANTY

FOR DETAILS REFER TO FULL WARRANTY TERMS AND CONDITIONS FOUND AT [WWW.V360LIFE.COM/SUPPORT](http://WWW.V360LIFE.COM/SUPPORT)

FOR A COPY OF THE WARRANTY TERMS & CONDITIONS, PLEASE SEND YOUR REQUEST TO: [SUPPORT@VSNMOBIL.COM](mailto:SUPPORT@VSNMOBIL.COM)

## TERMS AND CONDITIONS

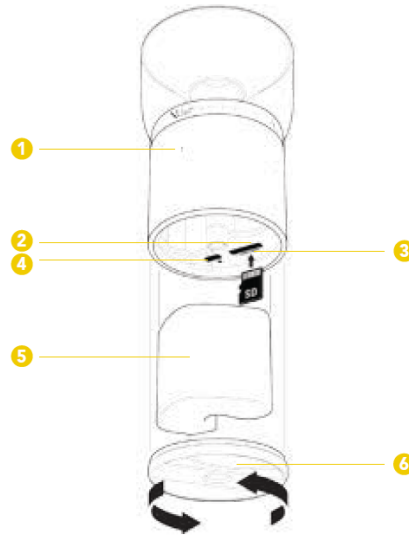
THE USE OF V360° IS SUBJECT TO THE TERMS AND CONDITIONS FOUND AT [WWW.V360LIFE.COM/TERMSANDCONDITIONS](http://WWW.V360LIFE.COM/TERMSANDCONDITIONS)



**PLEASE READ THIS GUIDE BEFORE YOU USE THIS PRODUCT.**

- 1 LED INDICATOR
- 2 USB 3.0 CHARGING PORT
- 3 SD CARD SLOT – A MICRO-SD CARD IS REQUIRED FOR OPERATION.  
INSERT SD CARD INTO SLOT. PRESS DOWN GENTLY TO LOCK INTO PLACE
- 4 HDMI PORT – CABLE NOT INCLUDED. USE OF FERRITE CORE HDMI CABLES IS HIGHLY RECOMMENDED
- 5 BATTERY:  
- INSERT BATTERY WITH THE PULL-TAB FACING DOWN  
- CHARGE FOR AT LEAST 2 HOURS PRIOR TO USE BY CONNECTING THE SUPPLIED USB 3.0 CABLE  
- YOUR CAMERA'S LED INDICATOR WILL TURN SOLID GREEN WHEN CHARGING IS COMPLETE
- 6 BATTERY DOOR:  
- TURN COUNTER CLOCKWISE TO UNLOCK  
- TO ATTACH, TURN CLOCKWISE TO LOCK IT INTO PLACE  
IMPORTANT: VERIFY THE BATTERY DOOR IS SEATED PROPERLY TO ENSURE THE CAMERA'S WATERTIGHT SEAL IS INTACT

## ASSEMBLY



## INITIAL SET UP V.360° COMPANION APPLICATION

THE V.360° APPLICATION CONTROLS YOUR CAMERA VIA AN ANDROID™ OR IOS® SMARTPHONE OR TABLET.

TO GET STARTED, DOWNLOAD THE FREE V.360° COMPANION APPLICATION FROM GOOGLE PLAY™ OR THE APPLE APP STORE® AND INSTALL IT ONTO YOUR DEVICE.

IT IS RECOMMENDED THAT YOU PERSONALIZE YOUR CAMERA BY CHANGING THE DEFAULT CAMERA NAME AND PIN AFTER INITIAL SETUP.

## CAMERA ON/OFF

THE CAMERA WILL TURN ON AUTOMATICALLY WHEN YOU CONNECT VIA THE V.360° APP. TO TURN YOUR CAMERA OFF, TOUCH THE GENERAL SETTINGS ICON ON YOUR APP AND SLIDE THE SWITCH NEXT TO POWER OFF CAMERA TO OFF.

## CONNECT TO CAMERA

TO ESTABLISH A CONNECTION TO YOUR CAMERA USING A SMARTPHONE OR TABLET:

- (1) ENSURE THAT WIFI AND BLUETOOTH ARE ENABLED WITHIN THE SETTINGS MENU ON YOUR DEVICE.
- (2) LAUNCH THE V.360° COMPANION APPLICATION. YOUR DEVICE WILL AUTOMATICALLY SEARCH FOR AVAILABLE CAMERAS.
- (3) TOUCH THE CAMERA'S ICON TO ESTABLISH A CONNECTION VIA BLUETOOTH. THE CAMERA LED WILL RAPIDLY BLINK GREEN TO INDICATE THAT IT IS POWERING ON.
- (4) WHEN CAMERA IS ON AND IN READY MODE, THE LED WILL SLOWLY FLASH GREEN.
- (5) WHEN PROMPTED, ENTER THE DEFAULT PIN (0000) FOR YOUR CAMERA THE FIRST TIME YOU CONNECT TO YOUR CAMERA.
- (6) ONCE CONNECTED, THE APP WILL GUIDE YOU THROUGH INSTRUCTIONS TO ESTABLISH A CONNECTION TO YOUR CAMERA AS A WIFI HOST.
- (7) RE-LAUNCH THE V.360° APP AND TOUCH THE CAMERA ICON TO ESTABLISH A CONNECTION VIA WIFI

## HOME SCREEN LAYOUT



\* IMAGES ARE FOR REFERENCE ONLY.

- 1 BACK BUTTON – TOGGLE TO PREVIOUS SCREEN
- 2 SHOTGLASS VIEW FINDER – TOGGLE LEFT/RIGHT TO ROTATE, UP/DOWN TO SWITCH TO FLAT VIEW
- 3 MODE CAROUSEL – SCROLL LEFT/RIGHT TO SWITCH BETWEEN VIDEO, PHOTO, BURST, TIME LAPSE, & SURVEILLANCE MODES
- 4 SHUTTER BUTTON – START/STOP RECORDING, CAPTURE PHOTOS
- 5 MODE SPECIFIC SETTINGS TAB
- 6 REMOTE CONTROL TAB – ACCESS TO CAMERA CONTROLS/FEATURES
- 7 MEDIA GALLERY – DISPLAY STORED CONTENT (VIDEOS & PHOTOS)
- 8 SD CARD - REMAINING PHOTOS/VIDEO RECORD TIME
- 9 SD CARD - STORED PHOTOS/VIDEO COUNTER
- 10 LIVE STREAM VIEWER – ON/OFF
- 11 CAMERA BATTERY LIFE INDICATOR
- 12 GENERAL CAMERA SETTINGS