PROJECT REPORT

ON

"DRISTI with Stereo Vision"

Submitted in partial fulfilment of the requirements for the partial completion of

PROJECT FOR COMMUNITY SERVICE [16EC7DCPW1]

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

SUBMITTED BY:

Samanth S Mokshagundam	**1BM14EC096**
Suhas G	**1BM14EC122**
Suhas T Shanbhogue	**1BM14EC123**
Suraj S	**1BM14EC127**

Under the Guidance of

Dr. B. S. Naghabhushana

(Professor, ECE, BMSCE)

August - December 2017

Department of Electronics and Communication Engineering

B.M.S COLLEGE OF ENGINEERING

(Autonomous College Affiliated to Visvesvaraya Technological University, Belgaum)

Bull Temple Road, Basavanagudi, Bangalore-560019

DECLARATION

We undersigned students of final semester B.E in Electronics and Communication Engineering, BMS College of Engineering, Bangalore, hereby declare that the dissertation entitled "DRISTI with Stereo vision", embodies the report of our project work carried out independently by us under the guidance of Dr. B.S. Nagabhushana, Professor, E&C Department, BMSCE, Bangalore in partial fulfilment for the award of Bachelor of Engineering in Electronics and Communication from Visvesvaraya Technological University, Belgaum during the academic year 2017-2018.

We also declare that to the best of our knowledge and belief, this project has not been submitted for the award of any other degree on earlier occasion by any student.

Place: Bangalore
Date:

Samanth S Mokshagundam	1BM14EC096
Suhas G	1BM14EC122
Suhas T Shanbhogue	1BM14EC123
Suraj S	1BM14EC127

B.M.S COLLEGE OF ENGINEERING

(Autonomous College under VTU)

Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that the project entitled **"DRISTI with STEREO VISION"** is a bonafide work carried out by **Samanth S Mokshagundam** (USN:1BM14EC096), **Suhas G** (USN:1BM14E122), **Suhas T Shanbhogue** (USN:1BM12EC123) and **Suraj S** (USN:1BM14EC127) in partial fulfillment for the partial completion of PROJECT FOR COMMUNITY SERVICE [**16EC7DCPW1**] during the academic year 2017-2018.

Dr. B. S. Nagabhushana	**Dr. G. Poornima**	**Dr. K.Mallikharjuna Babu**
Professor., ECE, BMSCE	HOD, ECE, BMSCE	Principal, BMSCE

External Examination: **Signature with date:**

1.

2.

DEDICATION

WE DEDICATE OUR PROJECT TO ALL THE VISUALLY IMPAIRED PEOPLE OF OUR COUNTRY

ABSTRACT

Vision is the single most point of information input for a multitude of animals including humans. But it is estimated that 253 million people are visually impaired and 36 million people are blind. These people are faced with a multitude of challenges even in completing normal tasks in life. They are dependent on other factors or people for daily life. The advancement of science and technology in the recent years has enabled significant progress in providing support to disabled people. This project is one such attempt to improve the quality of life for the visually impaired by restoring the ability to self-navigate. It attempts to provide an effective and efficient platform to enhance the perspective of the surrounding to the visually impaired. This is achieved by design of an obstacle detecting system based on disparity maps to perceive the proximity of the obstacle. The implementation consists of a Raspberry Pi to capture images and give feedback to the user. Raspberry Pi is connected locally to a laptop wirelessly to send images and then processed on laptop to find obstacles. Feedback control signal is sent back to the Raspberry Pi. According to this, suitable audio feedback is given to the user.

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We express profound gratitude to respected principal **Dr. K. Mallikharjuna Babu,** BMS College of Engineering for providing a congenial environment to work in. Our sincere gratitude to **Dr. G. Poornima ,** Head of the Department, Electronics and Communication Engineering for encouraging and providing this opportunity to carry out the project in the department.

We would like to thank our guide **Dr. B.S Nagabhushana,** Professor, Department of ECE who helped us in all the ways to carry out the project work. He stood beside and guided us in every step..

We thank all our professors for providing the basic knowledge without which this project wouldn't have been possible. Last but not the least we thank our family and friends, who made their valuable support compelled us to maintain a standard throughout our endeavour.

-

Samanth S Mokshagundam

Suhas G

Suhas T Shanbhogue

Suraj S

LIST OF FIGURES

LIST OF ABBREVIATIONS

SGBM: Semi Global Matching

WLS Filter: Weighted Least Square Filter

CONTENT

CHAPTER 1: INTRODUCTION

1.1 Introduction

Autonomous navigation is an extremely important factor for people suffering from visual impairment. The traditional and common mobility aids for such people have been walking cane (also called white cane or stick) and guide dogs. The major drawbacks of these are obstacle range coverage and high maintenance respectively. With the rapid development in technology in computing, network, and sensors, pervasive technology has achieved significant progress and is being put into use gradually. This has opened up new avenues and potential for intelligent navigation capabilities. Past decade has brought a lot of interesting products as Electronic Travel Aids devised to help the blind people to navigate safely and independently as much as possible. Different possible approaches for Obstacle Detection include Vision, Ultrasound, Laser and RADAR, for Warning system, tactile and acoustic. In this project we implement a stereo vision based approach to detect obstacles, and an acoustic warning system. Situations like holes and stairs are not considered specifically in this project. The project is implemented on a raspberry pi connected locally to a laptop via Wi-Fi.

Though Stereo Analysis is a powerful tool, its computational complexity has necessitated the use of processing on a laptop, within a local network with the Raspberry Pi. Being a near-real time system and having used depth estimation and segmentation, our project is aptly named DRISTI: Dynamic Ranging by Image Segmentation and Terrain Imaging.

1.2 Problem Definition

Let's look at an ordinary day with the eyes of a blind person. There is so much to take for granted which is not given to others, for example, because they can't see. What happens with people who are totally blind? All too frequently, blindness affects a person's ability to self -navigate outside well known environments and even simply walking down a crowded street. Many things are done very differently and they are dependent for others for their basic tasks such as organizing toiletries, separating medicines etc. Sometimes, accessible technologies will solve the problem. This includes any technology which can be used equally well by those who can see and those who can't. In other cases, special technologies need to be used to achieve the same that others do just by using their sight.

Dynamic Ranging by Image Segmentation and Terrain Imaging (DRISTI), is one such project specially designed which aims at assisting the visually impaired for navigation in simple environment. With DRISTI we aim equip the blind to detect obstacles. They will be able to gauge the proximity between themselves and the object.

1.3 Objectives

The objectives of the project can be subdivided as:

1. Obstacle detection: Segregate the foreground and the background.

By utilizing the principle of depth estimation based on stereo imaging, the foreground and background of a given scene are segregated. A single scene is captured in two different perspectives with the help of two cameras. The shift in perspective of the two images is estimated. Based on this estimation, foreground and background can be separated because the foreground will have a relatively larger shift when compared to the background.

2. Localization of obstacles: Determines the region where the object is present.

After determining the presence of an obstacle, the region where the obstacle is present is determined. This is a very crucial step for giving feedback to the user.

3. Warning system: After the region of an obstacle is determined, appropriate feedback must be given to the user. We plan on providing audio feedback to the user purely because it is the most primary sense a visually impaired individual can possess. Instead of utilizing a human voice for feedback, we plan on providing an intuitive acoustic feedback consisting of tones of varying frequencies.

CHAPTER 2: LITERATURE SURVEY

In literature, there are many different approaches for obstacle detection and warning system, such as:

- For obstacle detection: Vision, Ultrasound, Laser, and RADAR or a fusion of these.
- For warning system: Acoustic (TTS, Sound), Tactile (TDU, Vibratory motors etc.)

Below are some of the most relevant to our approach:

In [1], authors have used ground-plane detection algorithm using RANSAC and obstacles detected on this ground plane is intimated to the user via acoustic feedback. A ground plane estimation algorithm based on RANSAC plus filtering techniques allows the robust detection of the ground in every frame. A polar grid representation is proposed to account for the potential obstacles in the scene. The design is completed with acoustic feedback to assist visually impaired users while approaching obstacles. Beep sounds with different frequencies and repetitions inform the user about the presence of obstacles. Audio bone conducting technology is employed to play these sounds without interrupting the visually impaired user from hearing other important sounds from its local environment.

In [2], Obstacle detection is achieved by using Optical Flow with Stereo Vision which gives good results even for moving obstacles, unlike the previous one which is more concerned with static results. Accurate detection of moving obstacles from a moving vehicle is at the core of safe autonomous driving research. Stereo vision based sensors have been extensively used for this task as they are passive and provide a large amount 3D and 2D data. However, since no motion information is revealed, in intersections or crowded urban areas, static

and dynamic objects immediately next to each other, or closely positioned obstacles moving in different directions are often merged into a single obstacle leading to dangerous misinterpretations.

The authors of [5], use a similar approach as us, using a Kinect device to obtain a RGBD image which is used to detect pre-defined objects and this is related to the user using Tongue Electrode matrix. This system consists of two main components: environment information acquisition and analysis and information representation. The first component aims at capturing the environment by using a mobile Kinect and analysing it in order to detect the predefined obstacles for visually impaired people, while the second component tries to represent obstacle's information under the form of electrode matrix.

In [3], authors implement an autonomous automobile navigation based on obstacle detection using Unscented Kalman Filter. It's a very comprehensive detection system, but also requiring extensive computation which is suitable for an automobile but not for a wearable such as ours.

Reference [4], is another project which is aimed at autonomous navigation for automobiles. It computes a V-disparity map, from which obstacles are determined. Many roads are not totally planar and often present hills and valleys because of environment topography. Nevertheless the majority of existing techniques for - road obstacle detection using stereovision assumes that the road is planar. This can cause several issues : imprecision as regards the reel position of obstacles as well as false obstacle detection or obstacle detection failures. In order to increase the reliability of the obstacle detection process, this paper proposes an original, fast and robust method for detecting the obstacles without using the flat-earth geometry assumption; this method is able to cope with uphill and downhill gradients as well as dynamic pitching of the vehicle. Our approach

is based on construction and investigation of the "v-disparity"' image which provides a good representation of the geometric content of the road scene**.**

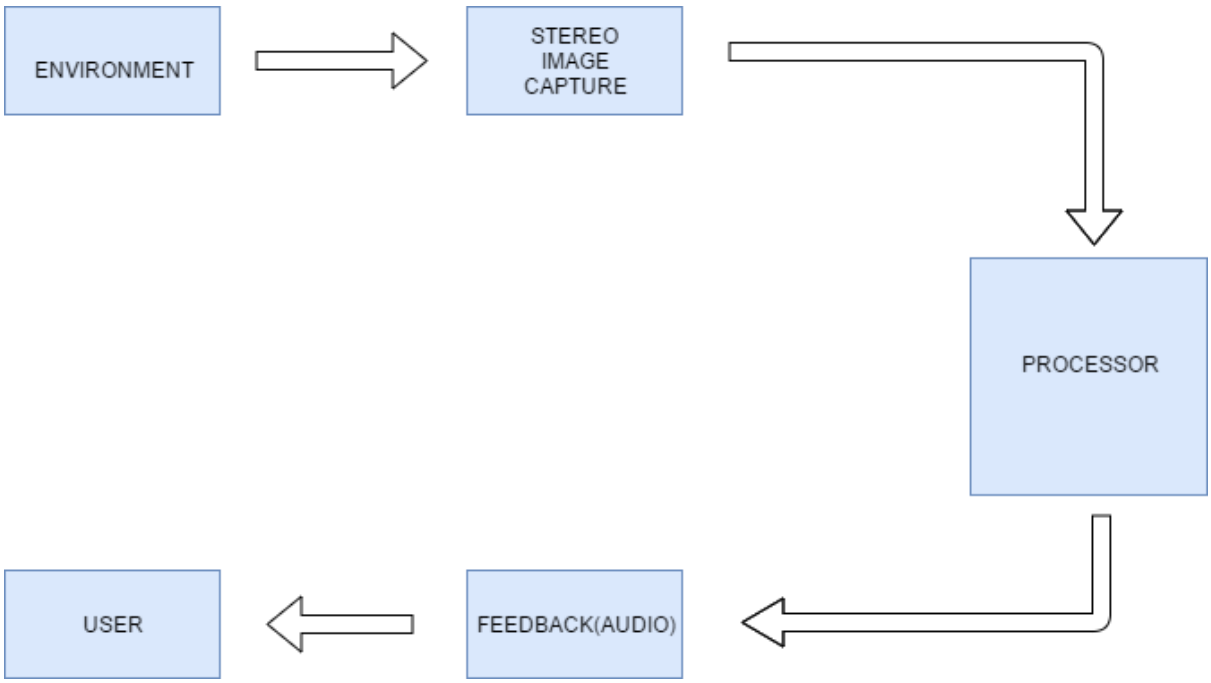
The authors of [6] describe an obstacle detection and mapping technique for mobile robot, based on depth matrix and drop-off detection using a laser based 3-DOF local SLAM. A mobile robot operating in an urban environment has to navigate around obstacles and hazards. Though a significant amount of work has been done on detecting obstacles, not much attention has been given to the detection of drop-offs, e.g., sidewalk curbs, downward stairs, and other hazards where an error could lead to disastrous consequences. This paper proposes algorithms for detecting both obstacles and drop-offs (also called negative obstacles) in an urban setting using stereo vision and motion cues. We propose a global colour segmentation stereo method and compare its performance at detecting hazards against prior work using a local correlation stereo method They implemented a U-disparity map to find the depth and applied a modified particle filter to detect multiple obstacles and track them. Vision systems provide a large functional spectrum for perception applications and, in recent years, they have demonstrated to be essential in the development of Advanced Driver Assistance Systems (ADAS) and Autonomous Vehicles. In this context, this paper presents an on-road objects detection approach improved by our previous work in defining the traffic area and new strategy in obstacle extraction from U-disparity

One more [8], approach for Unmanned Ground Vehicles using a “Horopter-based Stereo vision” and terrain imaging by 3D model formed based on real-time video processing. Sarnoff's next-generation video processor, the PVT-200, is used to demonstrate real-time algorithms for stereo processing, obstacle detection, and terrain estimation from stereo cameras mounted on a moving vehicle. Sarnoff's stereo processing and obstacle detection capabilities are currently being used in several Unmanned Ground Vehicle (UGV) programs,

including MDARS-E and DEMO III. Sarnoff's terrain estimation capabilities are founded on a "model-based directed stereo" approach. A collaborative research is demonstrated between Sarnoff and Universitat der Bundeswehr Miinchen, where we are studying vision processing for autonomous off-road navigation as part of the AUTONAVprogram.

CHAPTER 3: IMPLEMENTATION AND METHODOLOGY

3.1 BLOCK DIAGRAM



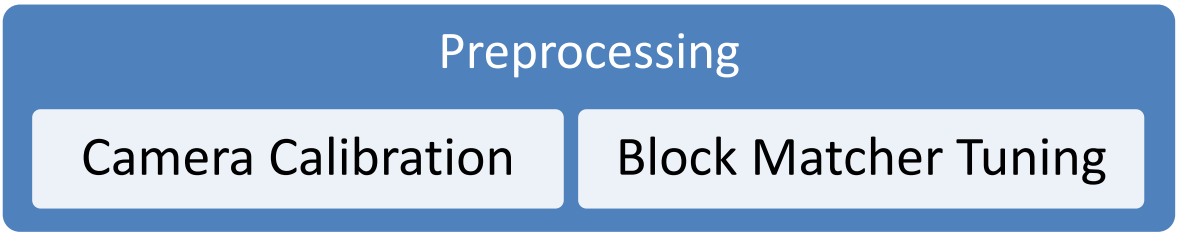
With the help of two Logitech C170 web-cameras, two different perspectives of the same scene are captured. Several image processing techniques are then applied to the image pair obtained by the processor to identify the relative proximities of potential obstacles in the captured scene. The proximities of the obstacles are then appropriately classified and based on the region of occurrence of the obstacles, audio feedback are given to the user which help in autonomous navigation.

3.2 PROJECT FLOW

Our project can be subdivided into 2 flows:

- Pre-processing
- Implementation Process

Pre-processing:



Preprocessing involves 2 steps:

1. Camera Calibration:

The Logitech c170 web cameras predominantly suffer from two types of distortions: Radial distortion in which straight lines will appear curved and tangential distortion which occurs because image taking lenses are not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected.

In order to remove these undesired irregularities from the two images, the individual camera parameters are required. The camera properties include the camera matrix, distortion coefficients, Rotation and Translation vectors, Projection matrix etc. To find all these parameters, we provided 100 images of a 7X10 chessboard in different orientations. One such orientation is shown below.

We find some specific points in it (square corners in chess board). We know its coordinates in real world space and we know its coordinates in image.

With these data, some mathematical problem is solved in background to get these parameters. When corresponding points in a set of images is known, its possible to find the translation vector and rotation vector, which gives rise to the distortion. Hence now we know all the parameters required to rectify the images.

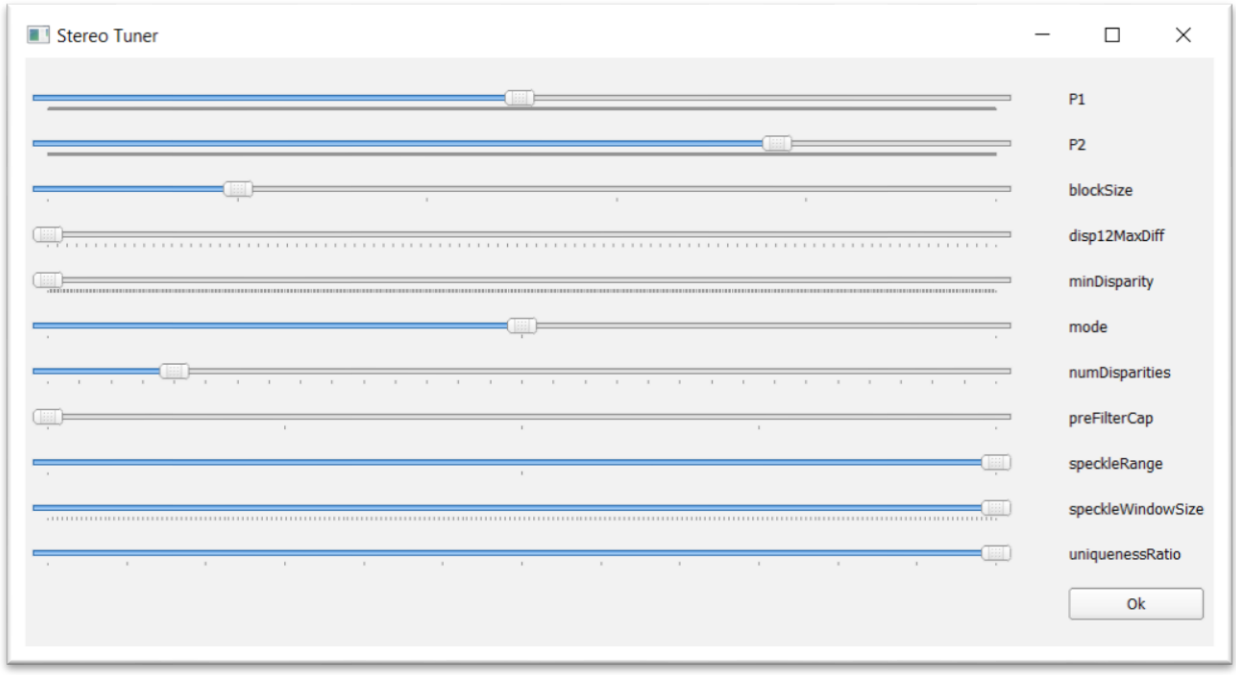
Figure 1 - Left Image

Figure 2 - Right Image

2. Block Matcher Tuning:

Semi Global Matching is a method which we are using to compute disparity. But it has a lot of parameters to be tuned for a particular hardware setup. So, to help in tuning we wrote an application to tune the parameters on the fly while observing the disparity.

A screenshot of the application is given below:



The importance of parameter tuning can be seen by the difference in disparity images obtained:

Figure 3 – Before Tuning

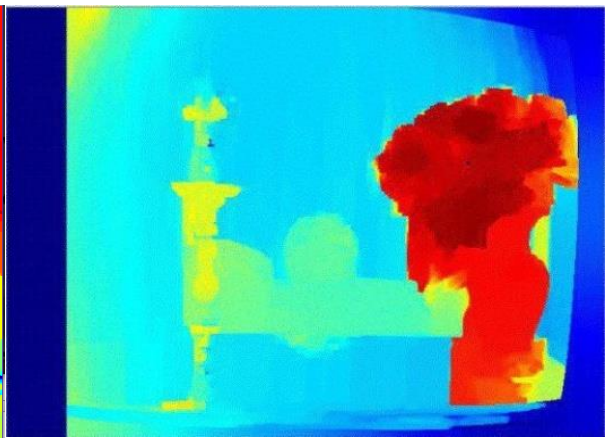
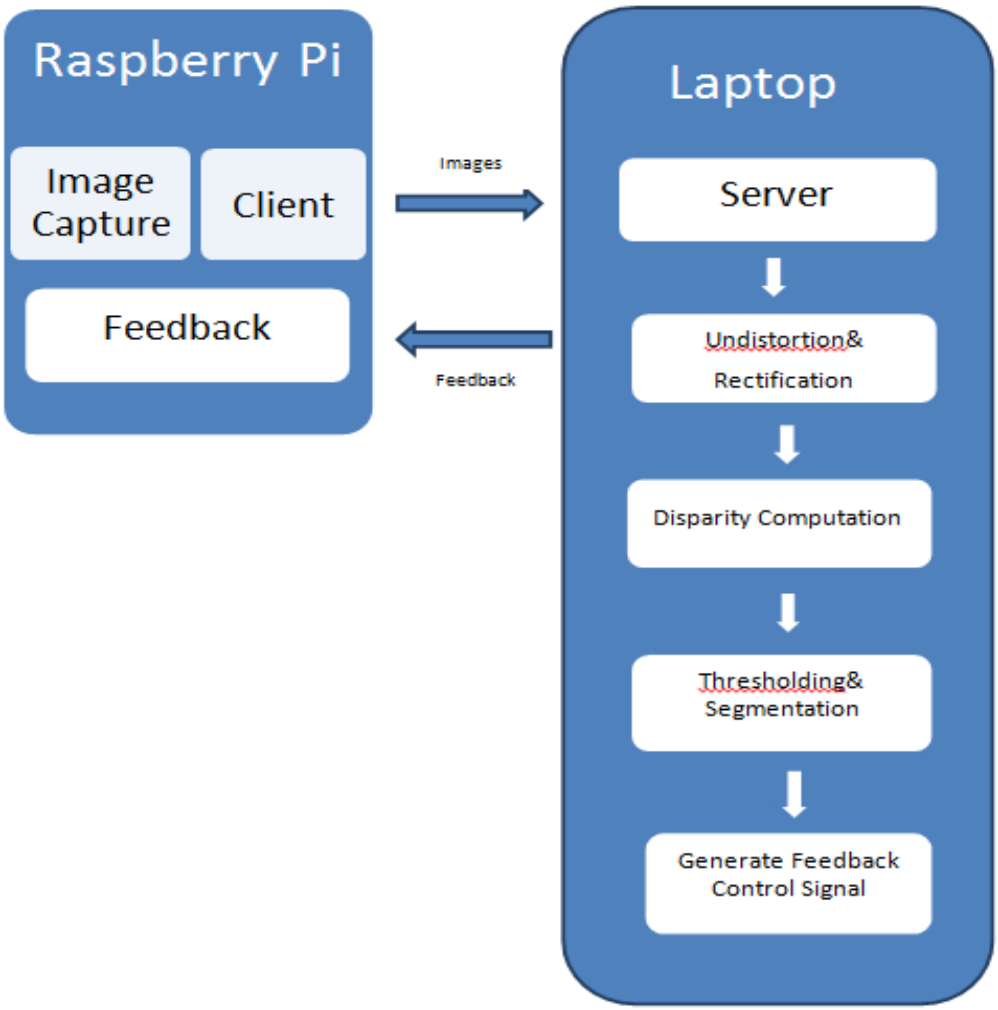


Figure 4 – After Tuning

Implementation Process



1. Image Capture

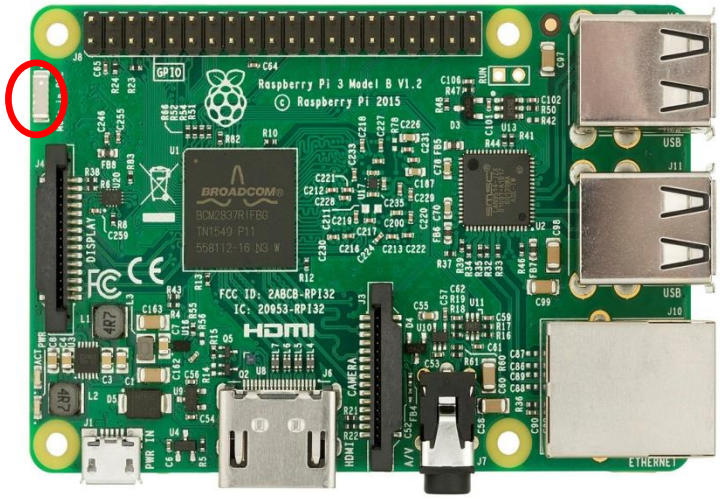
Using two Logitech C170 web cameras, two images each representing different perspectives of the same scene are captured. For the purpose of illustration of the concepts detailed below, we make use of a stereo image pair we have captured with the help of the web cameras we have used. The stereo pair is shown below.

Figure 5 - Image from left camera

Figure 6 - Image from right camera

2. Client

We are using Raspberry Pi as the client to capture image and then convert the image to gray-scale and send it to the laptop over wireless LAN. Raspberry Pi 3 Model B is having on board BCM43438 wireless LAN, which we can use to connect to a mobile hotspot, also to which the laptop is connected as well. Since this is a real-time application, streaming protocol based on UDP is used.



The Wi-Fi antenna module is shown by the red marking.

3. Server

Laptop acts as the server which receives the image from the raspberry pi and processes on it. After processing, laptop sends the feedback control signal.

4. Undistortion And Rectification

With the help of the camera parameters, the images captured from both the left and right cameras are subjected to image undistortion. The undistorted image is then subjected to rectification, where the image planes of both the captured images are made parallel to each other. The rectified image pair for the above test images is shown below.

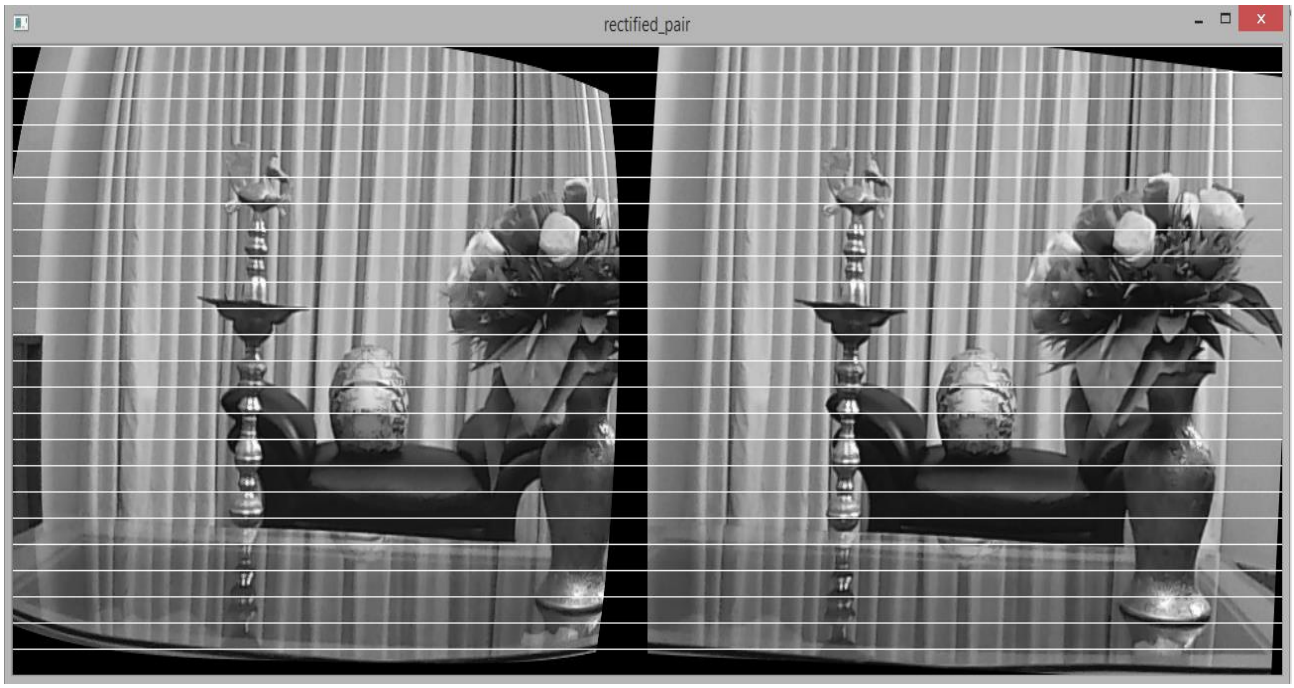


Figure 8 - Rectified Image Pair

5. Disparity Computation

The two rectified images are now used to compute the disparity corresponding to both the images. Disparity is formally defined as the differences in *x*-coordinates on the image planes of the same feature viewed in the left and right cameras: $x_{left} - x_{right}$. The two algorithms which are widely used for disparity computation are BM (Block Matching) and SGBM (Semi Global Block Matching). We have employed SGBM for disparity computation because SGBM offers relatively more accuracy and clarity. The disparity in both grey scale and RGB is shown below.

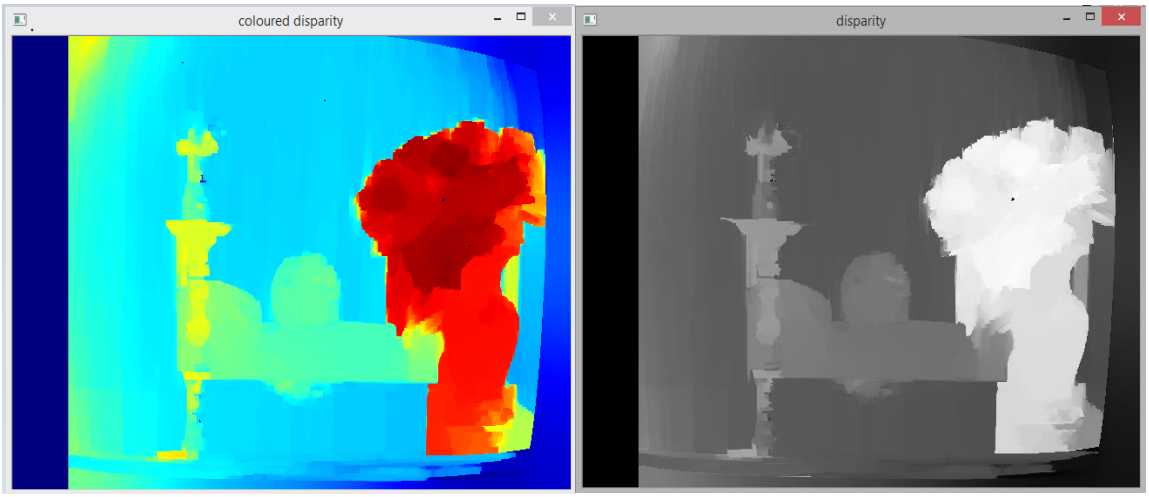


Figure 9 - Disparity

6. Disparity Thresholding And Segmentation

The disparity obtained is then subjected to 2-level binary thresholding indicating two levels of proximities. The proximity range for dividing the objects into these two regions is determined by the disparity intensity value.

Each of the proximity image planes obtained are then subjected to segmentation. This is done by dividing each of these image planes into appropriate number of subdivisions. Presence of objects in these subdivisions is determined by contouring. The two regions of proximities are shown below.

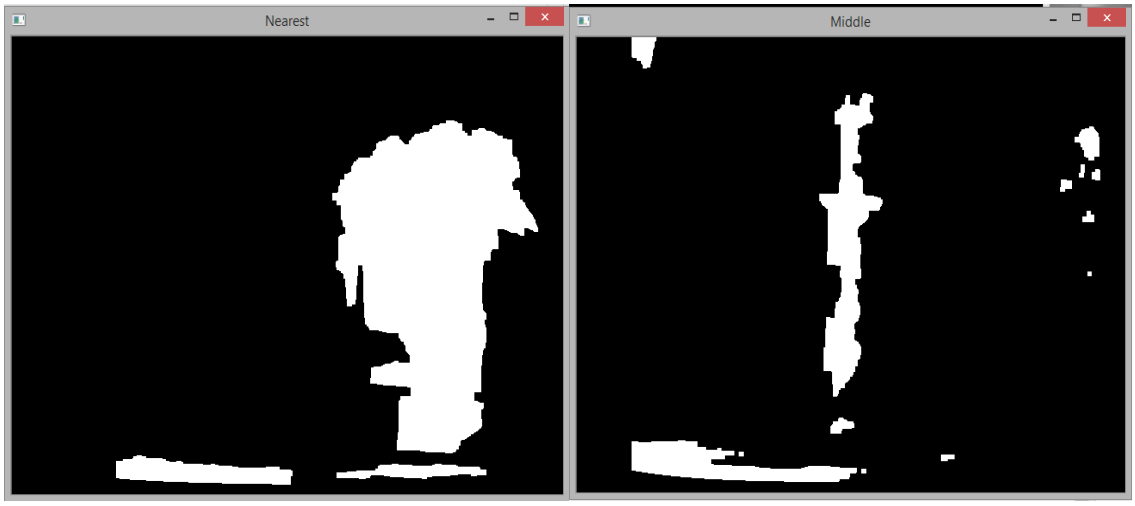


Figure 10 - Nearest

Figure 11 - Middle

7. Generating Feedback Control Signals

Based on the results of segmentation, appropriate audio signals are generated to indicate obstacles present in the "Near" region .For this, 3 frequencies are taken and the concept of stereo audio is used to differentiate the position of obstacles. The algorithm used to do this is explained in the next chapters.

8. Feedback

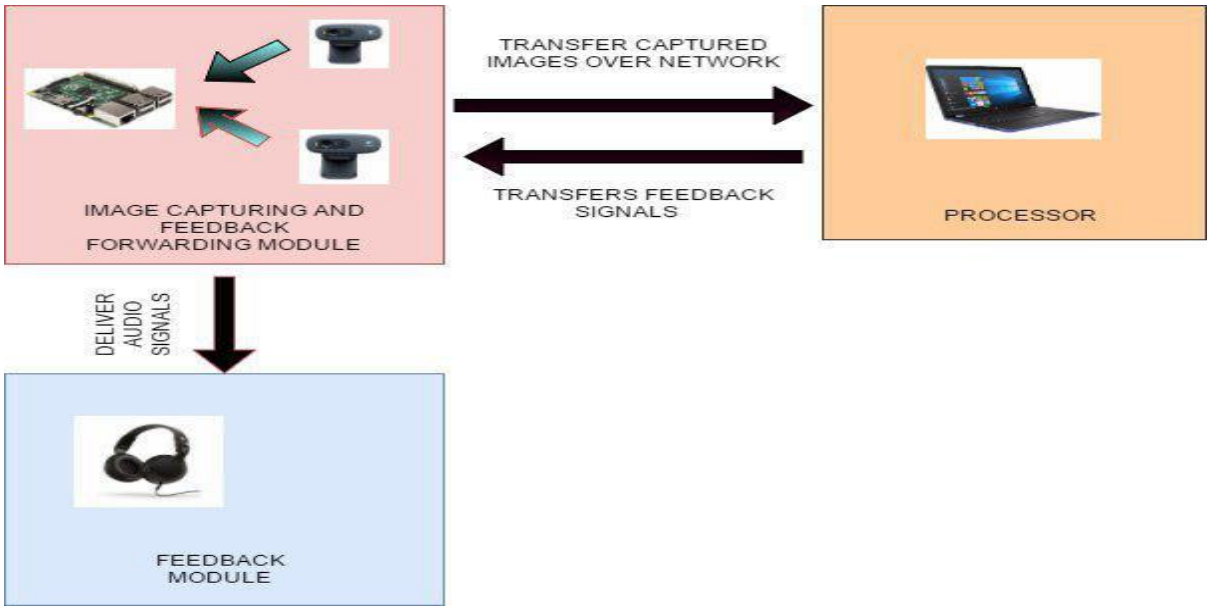
Based upon the feedback control signal, appropriate frequencies are generated and given to the user through earphones. Here the concept of Stereo sound is used.

3.3 HARDWARE ARCHITECTURE

3.3.1 Circuit Diagram

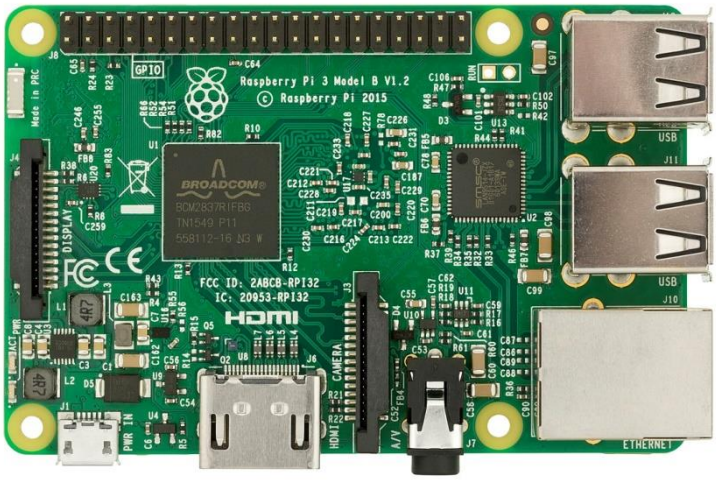
The hardware architecture of our project is comprised of:

- Raspberry Pi
- Stereo Camera Setup – c170 webcams × 2
- Earphone
- Laptop



3.3.2 Components Description

1. **Raspberry Pi:**



The Raspberry Pi is a single board computer featuring a Broadcom System on Chip (SoC), with an ARM compatible CPU and also an on-chip GPU.

Technical Specifications:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touch screen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

The Raspberry Pi is loaded with Raspbian Stretch OS.

2. **Stereo Camera Setup – C170 webcams × 2**

Figure 12 - Stereo camera setup

2 webcams are taken and fixed to a box such that it is immobile. The baseline of the stereo rig is fixed. The cameras used are Logitech c170. The technical specifications of the camera are:

- Optical Resolution: 640×480
- Diagonal Field of View (FOV) : 58°
- Focal Length: 2.3 mm
- Frame Rate (max): 640×480@30

3. **Earphone:**

Figure 13 - Earphones

Samsung Earphones are used for providing acoustic feedback based on proximity of object. These earphones are plugged into raspberry pi.

Some of the technical specifications are:

- Sound Output Mode: Stereo
- Minimum Frequency Response: 20 Hz,
- Impedance: 32 Ohm

4. **Laptop.**

Figure14- Laptop

The laptop used is a HP laptop with Intel i5 core processor, with clock speed 2.30 GHz to 2.40 GHz. It has 8 GB RAM. It is running on a 64 bit Windows 10 Operating System.

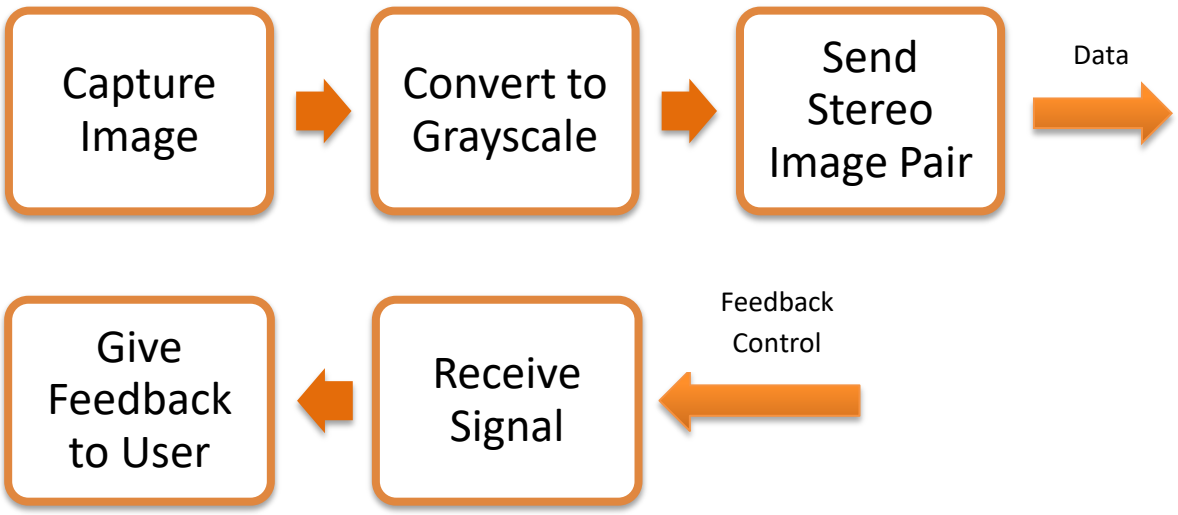
3.4 SOFTWARE ARCHITECTURE

Software Architecture of our project can be divided into 2 parts:

1. Input-Output and Pre-processing
2. Image Processing for Disparity and Generation of Feedback Control Signal

Part (1) is implemented on Raspberry Pi and Part (2) is implemented on Laptop.

1. Input-Output and Pre-processing



a. Capture Image

Images are captured from 2 webcams connected to Raspberry pi, on a separate thread from the main thread.

```
left_capture = cv2.VideoCapture(LEFT_CAM)
right_capture = cv2.VideoCapture(RIGHT_CAM)
```

Two instances of Video Capture are created which are next used to read frames from the webcams as continuously. Implementation of this in separate threads help in obtaining the most recently captured frames.

```
left_frame = left_capture.read()
right_frame = right_capture.read()
```

b. Convert to Grayscale

```
left_img = cv2.cvtColor(left_frame, cv2.COLOR_BGR2GRAY)
right_img = cv2.cvtColor(right_frame,cv2.COLOR_BGR2GRAY)
```

Images captured are converted from 3 channel RGB to single channel grayscale image.

c. Send Stereo Image pair

These images are next sent to the laptop via Wireless LAN. A streaming socket is implemented and connection is established with the laptop initially.

```
connection = socket.socket()
connection.connect((addr, port))
```

The images are converted to binary data and transmitted over this connection. A minimalistic protocol is implemented, total_packet = data_size + data, is sent.

```
connection.send(struct.pack('<L', sys.getsizeof(data)))
connection.send(data)
```

d. Receive Signal

After processing, laptop sends back the feedback control signal. It is received in a similar way as sending.

```
signal_len =    struct.unpack('<I',connection.recv(struct.calcsize('<I')))[0]

signal = connection.recv(signal_len)
```

e. Give Feedback to user

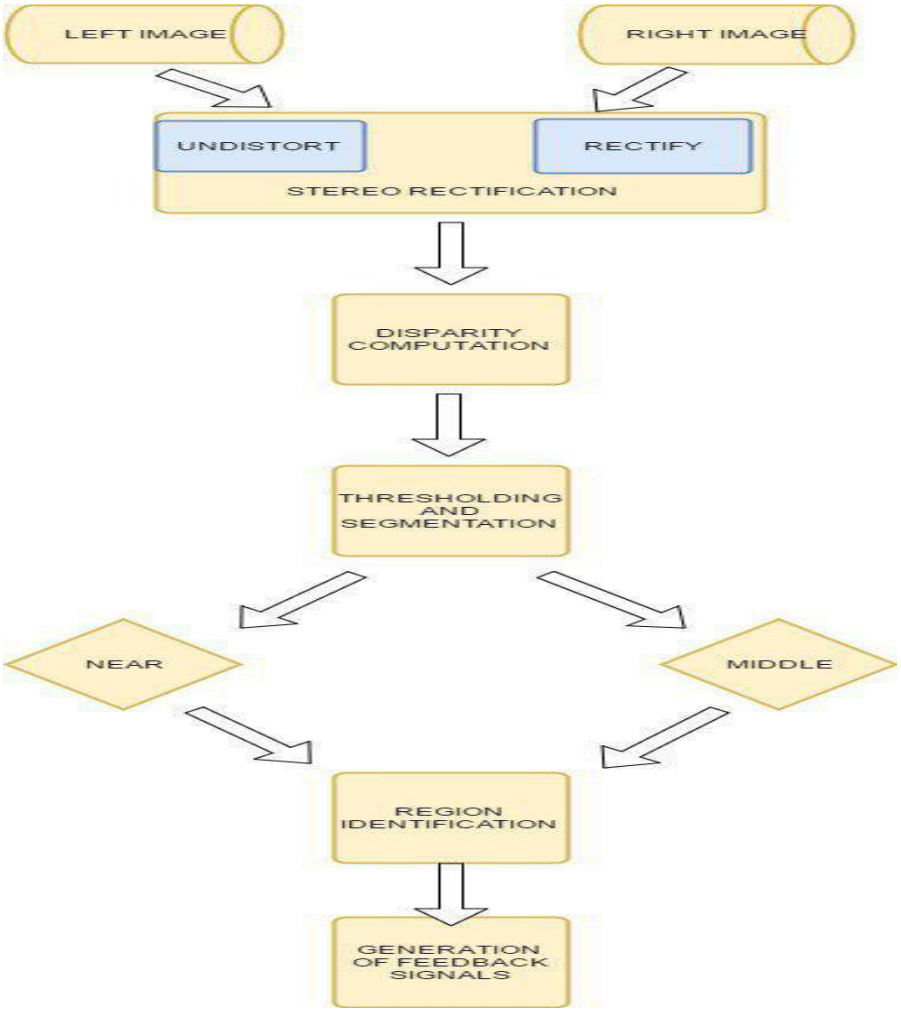
According to signal received, appropriate frequency is generated at particular earphone side. The truth table for audio feedback delivery is given in the table below.

IMAGE REGIONS			OUTPUT
LEFT REGION	MIDDLE REGION	RIGHT REGION	
0	0	0	No Feedback
0	0	1	F1, Right Earphone
0	1	0	F1, Both Earphone
1	0	0	F1, Right Earphone
1	1	0	F2, Left Earphone
0	1	1	F2, Right Earphone
1	0	1	F2, Both Earphone
1	1	1	F3, Both Earphone

Table 1 – Feedback Generation Truth Table

```
sd.play(self.volume* sound, loop = True, mapping = [1, 2])
```

2. Image Processing for Disparity and Generation of Feedback Control Signal



a. Stereo Rectification

Once the calibration parameters are known, stereo rectification can be performed. The main aim of stereo rectification is to make the left image plane and the right image plane parallel to each other. This ensures that the epipolar lines corresponding to both the images will be horizontal. Stereo rectification can be done sequentially in Opencv python with the help of the following functions.

1. Using "cv2.init UndistortRectifyMap()" function.

The function computes the joint undistortion and rectification transformation(maps) and represents the result in the form of maps for remap(). We called this function twice in order to get the undistortion and rectification maps for both the left image and the right image. The code snippet is given below.

```
1. mapx1,mapy1=cv2.initUndistortRectifyMap(calibration.cam_mats['left'],calibration.di
   st_coefs['left'],calibration.rect_trans['left'],calibration.proj_mats['left'],(w,h)
   ,cv2.CV_32FC1)
2. mapx2,mapy2=cv2.initUndistortRectifyMap(calibration.cam_mats['right'],calibration.d
   ist_coefs['right'],calibration.rect_trans['right'],calibration.proj_mats['right'],(
   w,h),cv2.CV_32FC1)
```

where calibration is the an object of class "Calibration" consisting of the calibration folder which contains both the extrinsic and intrinsic parameters of both the cameras. The outputs of this functtion call are the undistortion and rectification maps denoted bymapx and mapy respectively.

2.Using "cv2.remap()" function.

After obtaining the undistortion and rectification maps, by using cv2.remap () function, rectified image pair is obtained. The code snippet is given below.

```
1. rectified_left = cv2.remap(left_img,mapx1,mapy1,cv2.INTER_LINEAR)
2. rectified_right = cv2.remap(right_img,mapx2,mapy2,cv2.INTER_LINEAR)
```

where "rectified_left" and " rectified_right " represent the rectified left image and rectified right image respectively.

b. Disparity Computation

After obtaining the rectified image pair, disparity is obtained by applying Stereo Correspondence algorithm. OpenCV provides a few stereo correspondence algorithms like Graph cut, BM and SGBM . We have used SGBM because it offers the best performance in terms of accuracy and clarity. The code snippet for computing disparity using SGBM is given below.

```
def get_filtered_disparity(left_img, right_img):
    block_matcher = cv2.StereoSGBM_create(
                            minDisparity = 0,
                            numDisparities = 64,
                            blockSize = 3,
                            P1 = 72,
                            P2 = 2600,
                            disp12MaxDiff = 0,
                            uniquenessRatio = 4,
                            speckleRange = 1,
                            preFilterCap = 1,
                            speckleWindowSize = 10,
                            mode = 1
                            )
    wls_filter = cv2.ximgproc.createDisparityWLSFilter(block_matcher)
    right_matcher = cv2.ximgproc.createRightMatcher(block_matcher)
    left_disparity = block_matcher.compute(left_img, right_img)
    cv2.imshow('disparity without wls filter',left_disparity/left_disparity.max())

    right_disparity = right_matcher.compute(right_img, left_img)
    wls_filter.setLambda(8000.0)
    wls_filter.setSigmaColor(1.0)
    filtered_disparity = wls_filter.filter(left_disparity, left_img, None, right_di
sparity)
    filtered_disparity[filtered_disparity > 1008] = 1008
    filtered_disparity[filtered_disparity < -16] = -16
    disparity = (((filtered_disparity + 16)/1008)*255).astype(np.uint8)
    cv2.imshow('disparity', disparity)
    kernel = np.ones((3,3),np.uint8)
    disparity = cv2.morphologyEx(disparity,cv2.MORPH_OPEN,kernel, iterations = 2)
    coloured_disparity = cv2.applyColorMap((disparity).astype(np.uint8), cv2.COLORM
AP_JET)
    cv2.imshow('coloured disparity',coloured_disparity)
    return disparity
```

The SGBM parameters were obtained by utilizing a stereo tuner program. In the snippet given above, "block_matcher" is an object of SGBM class and contains all the required parameters needed for SGBM computation.

The disparity obtained by only utilizing stereo correspondence based on SGBM resulted in disparity having a lot of sharpness. In order to obtain a more refined disparity, we utilized a special type of filter called WLS filter which

uses Weighted Least Squares algorithm to refine the disparity. It takes the disparities obtained with both left and right images being references and computes a more refined disparity.

c. Thresholding And Segmentation

The disparity obtained is further subjected to 2-level binary thresholding in order to segregate obstacles into two regions based on their disparity values(distance). The code snippet is given below.

```
def apply_thresholds(disparity):
    NEAREST_LOWER_THRESH = 180
    NEAREST_HIGHER_THRESH = 255
    MIDDLE_LOWER_THRESH = 140
    MIDDLE_HIGHER_THRESH = 180
    nearest = np.full(disparity.shape, 255, dtype = np.float32)
    middle = np.full(disparity.shape, 255, dtype=np.float32)
    nearest[disparity > NEAREST_HIGHER_THRESH] = 0
    nearest[disparity < NEAREST_LOWER_THRESH] = 0
    middle[disparity > MIDDLE_HIGHER_THRESH] = 0
    middle[disparity < MIDDLE_LOWER_THRESH] = 0
    kernel = np.ones((5,5),np.uint8)
    nearest = cv2.morphologyEx(nearest, cv2.MORPH_OPEN, kernel)
    nearest = cv2.morphologyEx(nearest, cv2.MORPH_CLOSE, kernel)
    middle = cv2.morphologyEx(middle, cv2.MORPH_OPEN, kernel)
    middle = cv2.morphologyEx(middle, cv2.MORPH_CLOSE, kernel)
    cv2.imshow('Nearest', nearest)
    cv2.imshow('Middle', middle)
    return nearest, middle
```

where "nearest" and "middle" contain binary images indicating obstacles in their respective proximity regions. The Image represented by "nearest" has only the pixel values in disparity greater than "NEAREST_LOWER_THRESH " represented by 255. Similarly, the image represented by "middle", has only pixel values in disparity lying between "MIDDLE_HIGHER_THRESH " and "MIDDLE_LOWER_THRESH " represented by 255.

The two images obtained by thresholding are then subjected to Segmentation in which, the corresponding image is divided into regions vertically and individual segments are further used for processing. The image corresponding to the "nearest" proximity range is divided into 3 regions while the other image is divided into 4 regions. The code snippet for segmentation is given below.

```
def apply_segmentation(nearest, middle):
    signals = {'nearest': [], 'middle':[]}
    nearest_parts = np.array_split(nearest, 3, axis = 1)

    for each in nearest_parts:
        loc = find_locations(each)
        if loc:
            signals['nearest'].append(True)
```

```
        else:
            signals['nearest'].append(False)

    middle_parts = np.array_split(middle, 4, axis = 1)
    for each in middle_parts:
        loc = find_locations(each)
        if loc:
            signals['middle'].append(True)
        else:
            signals['middle'].append(False)

    return signals

def find_centroid(contour):
    M = cv2.moments(contour)
    try:
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        return (cx, cy)
    except:
        return None

def find_locations(img):
    contour_img, contours, _ = cv2.findContours(img.astype(np.uint8), cv2.RETR_EXTE
RNAL,
    cv2.CHAIN_APPROX_NONE)
    contours = [contour for contour in contours if cv2.contourArea(contour) >
    (img.shape[0]*img.shape[1]/3)]
    locations = [find_centroid(contour) for contour in contours]
    locations = [loc for loc in locations if loc not None]

    return locations
```

d. Region Identification

Based on the signals obtained after segmentation, the regions are identified. This is a very simple procedure which just converts the signals obtained into a more understandable form to generate feedback. The functionality is included in the above code snippet.

e. Generation Of Feedback Control Signal

Based on the region identified, appropriate signal is generated and sent to the Raspberry Pi over the wireless network. For eg, if 2 objects are present nearby to the camera, one to the left and one to the right, then the signal generated is [1,0,1]. This functionality is included in the above code snippet as well.

Reception of Data and sending of signal from laptop to Raspberry Pi is also similar to the communication procedure implemented on Raspberry pi.

CHAPTER 4: RESULTS AND DISCUSSIONS

4.1 Results

Project DRISTI works as proposed**;** two cameras are used to obtain live visual feed of the surroundings of a blind person. This is given as input to the Raspberry Pi, which computes the binary of the image and sends it to the laptop over a Local Area Network for further computation. The laptop is used to calculate disparity from the two camera using SGBM algorithm. When the blind person encounters an object the disparity calculated by the laptop will be utilised by the Raspberry Pi to give an acoustic feedback based on the proximity of the said object. Depending upon which ear gets the audio signal and also upon the frequency of the signal the number of objects and the region/s in which the object is present can be determined. The user can use this data to avoid the obstacle.

4.2 Discussion

In this project, visual assistance system for the blind ,based on stereo imaging was proposed. Since the traditional methods such as walking stick and guide dog are not efficient and reliable, there was a need for new technology which overcomes these drawbacks.

For the purpose of assisting the visually impaired, in order to gauge the distance, stereo imaging technique has been utilised. It is advantageous over other techniques which use sensors for measuring distance. Eg: in the case of ultrasonic sensors objects need to be present in the line of sight of the sensor, In addition they are not able to distinguish between a single object and multiple objects and also the area in which the object is present in. In order to determine

these using a sensor like ultrasonic sensor, multiple sensors along many directions will be required which will increase the cost significantly and the arrangement of sensors will again not be feasible, so the best approach would be to use two cameras and obtain stereo images for object detection as well as depth mapping of the objects.

For every scene two images were obtained using the cameras. In order to obtain the depth mapping of the scene, different algorithms were tested upon the images to determine the best possible algorithm.

Block Matching algorithm (BM)

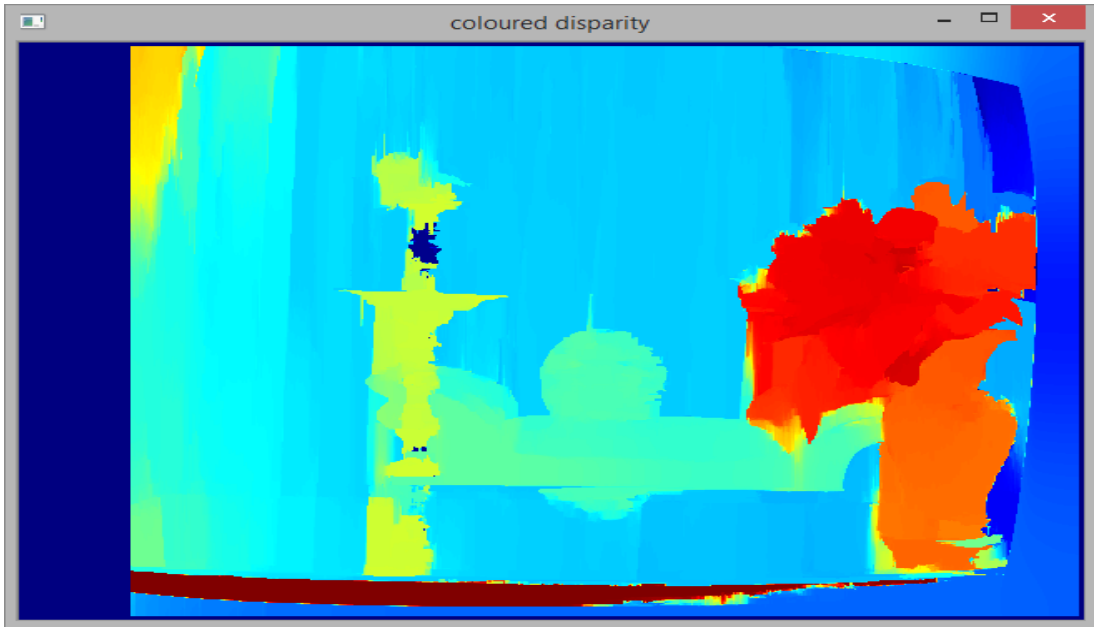


Figure15- BM

Semi Global Block Matching (SGBM)

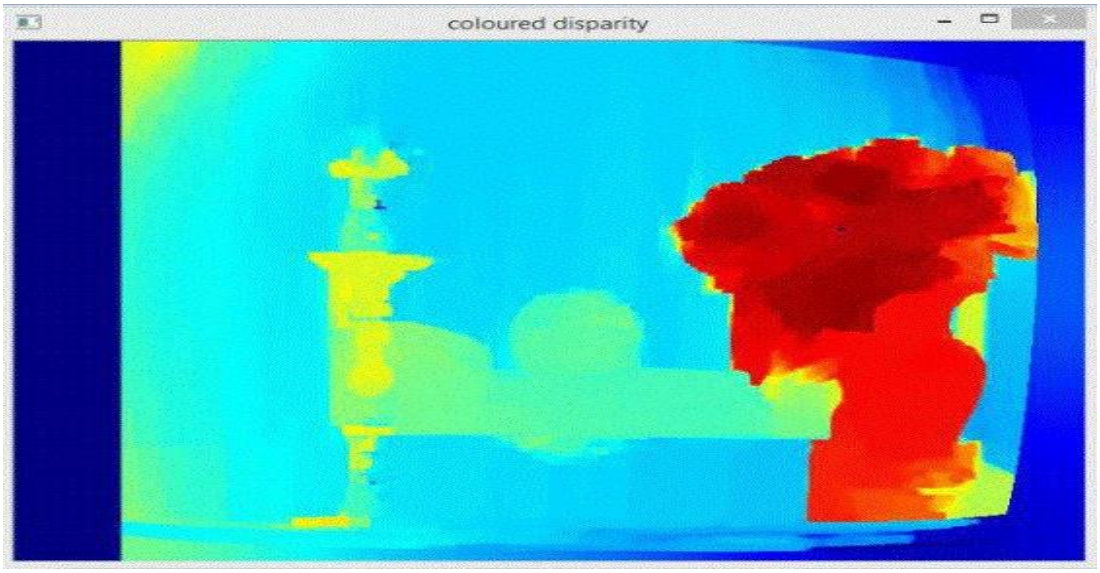


Figure16-SGBM

From the outputs for the sample set presented above, we can conclude that SGBM provides a smoother image with less distortions. For this reason SGBM algorithm was chosen in order to calculate the disparity.

Technology is to be used for the welfare of all mankind, we would like to equip all the visually impaired individuals with this new tool for their navigational aid, but the cost of the project makes it less affordable for some, However with proper government funding this can be overcome. Project DRISTI will impede with the normal ear function as the user will have to wear earphones, but the new research field which uses bone conduction technology will be able to liberate the ear. Apart from these shortcomings of the system there are no major flaws that may give us a thought to drop the idea or lower our interests in such systems. The perks and gains from this system are far beyond their flaws.

The benefits of the proposed system are:

1. Obstacle recognition at a distance: One of the major advantages is that the blind person will be able to identify the presence of an object from a distance without having to come in contact with the obstacle.
2. Reduced human error: as the primitive walking stick is replaced with more accurate technology there is reduction in error, both false positives and false negatives, for obstacle detection.
3. Ease of navigation: Indoor navigation as well as outdoor navigation is made easier.
4. Improves the self-esteem of the blind as they become more independent in their navigation and day to day activities.
5. Saves time: Navigation time from one place to another is reduced with the help of this tool.

Since the orientation of the object with respect to themselves is known, they will be able to determine the necessary course to be taken in order to avoid the obstacle.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

5.1 CONCLUSIONS

At the end of this project, the conclusion drawn, was that a new technology of obstacle detection could be brought into practise using stereo imaging for helping the blind navigate through to their destination without having to take too much of a risk in reaching their destination.

DRISTI (Dynamic Ranging by Image Segmentation & terrain Imaging), would be of good help to the blind for object detection and autonomous navigation. A noteworthy advantage of this project is that it does not depend on which type of obstacle(vehicle/human/animal, etc.) is approaching towards the visually impaired person or is being approached to by the person, due to the feedback, the person knows about the danger ahead and is able to avoid collision with these obstacle thereby saving himself/herself from any sort of injury/damage that could be caused otherwise.

DRISTI also provides hands free navigation to the blind and is a reliable means of feedback system to ensure that they are well informed of any obstacles ahead. The person does not have to constantly try to gauge for obstacles with the help of any aid such as walking stick, cane, etc. There also will not be any restrictions in any well-lit environments for the blind to have this device help them unlike other aids such as guide dogs and won't get stuck in pavements, pits, etc. like walking sticks & canes.

The feedback is reliable as it not only helps detect obstacle, but also helps to know in which direction the obstacle ,depending upon from which direction(left or right the audio feedback is obtained and helps to also get a gauge of the approximate distance of the obstacle depending on the intensity of the feedback.

Hence it is clear that autonomous navigation through audio feedback would be of great help to the blind, especially in a country like India with the huge blind population.

5.2 FUTURE WORK

Our project mainly aimed at detection of obstacles and would mainly help in indoor navigation for the blind people and in roads without too much gradient change and potholes. Following are the advancements that can be made to enhance the perspective of the surroundings even better to the blind people so that they can further navigate without fear and any associated risk.

1. Pothole detection: This methodology would be of utmost help, especially in badly maintained places and roads, where the potholes are left open and the blind people could fall of in case not warned about the same.
2. Gradient detection: Any sudden change of slope is also dangerous for navigation and in this case too, if no proper warning has been given to the people, then there is a high chance of tripping and falling off if hands free.
3. Obstacle recognition: This can be implemented by machine learning wherein the user is not only warned about the type of obstacle in front, so that he or she is further aware of what exactly what is in front of him/her.
4. GPS tracking system: If a GPS tracking system could be applied to the device, then it could help the person know exactly his/her location and help navigate to the destination with help of audio signals. This could also be of great help to anyone who wants to know where exactly the person is, like a friend or family member and could help them find out if the person has got stuck anywhere, by sending the location of the user.

REFERENCES

[1] Alberto Rodríguez, J. Javier Yebes, Pablo F. Alcantarilla, Luis M. Bergasa, Javier Almazan and Andrés Cela, *Assisting the Visually Impaired: Obstacle Detection and Warning System by Acoustic Feedback*, Open Access, Sensors, ISSN 1424-8220

[2] Cosmin D. Pantilie, SilviuBota, Istvan Haller and SergiuNedevschi, *Real-time Obstacle Detection Using Dense Stereo Vision and Dense Optical Flow*, 978-1-4244-8230-6/10/$26.00 ©2010 IEEE

[3] Qian Yu, HelderAraujo,Hong Wang, *A Stereovision Method for Obstacle Detection and Tracking in Non-Flat Urban Environments*, Autonomous Robots 19, 141–157, 2005© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

[4] Raphael Labayrade, Didier Aubert, Jean-Philippe Tarel, *Real Time Obstacle Detection in Stereovision on Non Flat Road Geometry Through "V-disparity" Representation.*

[5] Van-Nam Hoang, Thanh-HuongNguyen,Thi-Lan Le, Thanh-Hai Tran, Tan-PhuVuong, Nicolas Vuillerme, *Obstacle detection and warning system for visually impaired people based on electrode matrix and mobile Kinect*, Open Access at Springer 26 July 2016

[6] AniketMurarka, Mohan Sridharan, Benjamin Kuipers,*Detecting Obstacles and Drop-offs using Stereo and Motion Cues forSafe Local Motion*,published athttps://web.eecs.umich.edu/~kuipers/

[7] Bihao Wang, Sergio Alberto Rodriguez Florez, Vincent Fremont, *Multiple Obstacle Detection and Tracking using StereoVision: Application and Analysis*, published in HAL archives, https://hal.archives-ouvertes.fr/hal-01095618

[8] Robert Mandelbaum, Luke McDowell, Luca Bogoni, Barry Reich, Mogens L. Hansen,*Real-Time Stereo Processing, Obstacle Detection,and Terrain Estimation from Vehicle MountedStereo Cameras*,Proceedings of the 4th IEEE Workshop on the Applications of Computer Vision, October 1998, pages 288-289.

Other websites:

a. https: //erget.wordpress.com/2014/02/01/calibrating-a-stereo-camera-with-opencv/
b. https://erget.wordpress.com/2014/02/28/calibrating-a-stereo-pair-with-python/

c. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
d. https://www.raspberrypi.org/documentation/installation/installing-images/
e. https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

ANNEXURE

Codes used:

1. RASPBERRY CLIENT

```
import sys, os, struct, pickle, socket

import cv2
import numpy as np
import sounddevice as sd

from image_loader import CaptureImage
from audio import AudioFeedback

IMAGE_WIDTH = 640
IMAGE_HEIGHT = 480
LEFT_CAM = 0
RIGHT_CAM = 1

def initialise_network(addr, port):
    connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connection.connect((addr, port))
    return connection

def send(connection, data):
    connection.send(struct.pack('<L', 640*480*2))
    connection.send(data)

def recieve(connection):
    signal_len = struct.unpack('<I', connection.recv(struct.calcsize('<I')))[0]

    signal = b''
    length = 0
    while length < signal_len:
        chunk = connection.recv(int(signal_len/2))
        signal += chunk
        length += sys.getsizeof(chunk)

    signal = pickle.loads(signal)
    return signal

def preprocess(left_img, right_img):
    left_img = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
    right_img = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)

    data = np.hstack((left_img, right_img)).astype(np.uint8)
    return data


def main():
    image_loader = CaptureImage(LEFT_CAM, RIGHT_CAM)
    image_loader.start()
    connection = initialise_network('Suhas-G', 8000)
    feedback = AudioFeedback()
    print('Connection initialised...')
    while True:
        left_img, right_img = image_loader.load_images()
        data = preprocess(left_img, right_img)
        send(connection, data.tostring())
        signal = recieve(connection)
        feedback.update(signal)
```

```
57.
58.
59.     feedback.stop()
60.
61. if __name__ == '__main__':
62.     main()
63.
```

2. LAPTOP SERVER

```
import sys, struct, os, pickle, socket

import numpy as np
import cv2

from load_calibration import Calibration
from disparity import DisparityCreator

CALIBRATION_FOLDER = 'calibration_calibrate_1'
IMAGE_WIDTH = 640
IMAGE_HEIGHT = 480
NEAREST_LOWER_THRESH = 190
NEAREST_HIGHER_THRESH = 255
MIDDLE_LOWER_THRESH = 150
MIDDLE_HIGHER_THRESH = 180


def initialise_connection():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('0.0.0.0', 8000))
    print('Connection initiated...')
    server_socket.listen()
    connection, address = server_socket.accept()
    print('Connection accepted...')
    return connection, server_socket

def recieve(connection):
    data_length = struct.unpack('<L', connection.recv(struct.calcsize('<L')))[0]
    if not data_length:
        sys.exit()
    data = b''
    length = 0
    while length < data_length:
        chunk = connection.recv(int(data_length/2))
        length += len(chunk)
        data += chunk
    data = np.fromstring(data, dtype = np.uint8)
    data = np.reshape(data, (-1, IMAGE_WIDTH * 2))


    left_img, right_img = np.array_split(data, 2, axis = 1)

    return left_img, right_img

def send(connection, signal):
    connection.send(struct.pack('<I', sys.getsizeof(signal)))
    connection.send(pickle.dumps(signal))

def preprocess(calibration, left_img, right_img):

    undistorted_left = cv2.remap(left_img,calibration.undistortion_map['left'],
                                 calibration.rectification_map['left'],
                                 cv2.INTER_LINEAR)
    undistorted_right = cv2.remap(right_img,calibration.undistortion_map['right'],

                                  calibration.rectification_map['right'],
                                  cv2.INTER_LINEAR)


    return undistorted_left,undistorted_right

def apply_thresholds(disparity):
    nearest = np.full(disparity.shape, 255, dtype = np.float32)
```

```
63.     middle = np.full(disparity.shape, 255, dtype=np.float32)
64.
65.     nearest[disparity > NEAREST_HIGHER_THRESH] = 0
66.     nearest[disparity < NEAREST_LOWER_THRESH] = 0
67.
68.     middle[disparity > MIDDLE_HIGHER_THRESH] = 0
69.     middle[disparity < MIDDLE_LOWER_THRESH] = 0
70.
71.     kernel = np.ones((5,5),np.uint8)
72.     nearest = cv2.morphologyEx(nearest, cv2.MORPH_OPEN, kernel)
73.     nearest = cv2.morphologyEx(nearest, cv2.MORPH_CLOSE, kernel)
74.
75.     middle = cv2.morphologyEx(middle, cv2.MORPH_OPEN, kernel)
76.     middle = cv2.morphologyEx(middle, cv2.MORPH_CLOSE, kernel)
77.
78.     # cv2.imshow('Nearest', nearest)
79.     # cv2.imshow('Middle', middle)
80.     return nearest, middle
81.
82. def find_centroid(contour):
83.     M = cv2.moments(contour)
84.     try:
85.         cx = int(M['m10']/M['m00'])
86.         cy = int(M['m01']/M['m00'])
87.         return (cx, cy)
88.     except:
89.         return None
90.
91. def find_locations(img):
92.     contour_img, contours, _ = cv2.findContours(img.astype(np.uint8), cv2.RETR_EXTE
    RNAL, cv2.CHAIN_APPROX_NONE)
93.     contours = [contour for contour in contours if cv2.contourArea(contour) > (img.
    shape[0]*img.shape[1]/3)]
94.     locations = [find_centroid(contour) for contour in contours]
95.     locations = [loc for loc in locations if not loc == None]
96.
97.     return locations
98.
99.
100.        def apply_segmentation(nearest, middle):
101.            signal = {'nearest': [], 'middle':[]}
102.            nearest_parts = np.array_split(nearest, 3, axis = 1)
103.
104.            for each in nearest_parts:
105.                loc = find_locations(each)
106.                if loc:
107.                    signal['nearest'].append(1)
108.                else:
109.                    signal['nearest'].append(0)
110.
111.            middle_parts = np.array_split(middle, 4, axis = 1)
112.            for each in middle_parts:
113.                loc = find_locations(each)
114.                if loc:
115.                    signal['middle'].append(1)
116.                else:
117.                    signal['middle'].append(0)
118.
119.            return signal
120.
121.
122.
123.
124.        def main():
125.            calibration = Calibration((IMAGE_WIDTH, IMAGE_HEIGHT))
126.            calibration.load_calibration_files(CALIBRATION_FOLDER)
```

```
127.            # signal = pickle.dumps({'nearest':[1,2,3], 'middle':[2,3,4]})
128.            disparity_handler = DisparityCreator(16000, 7)
129.            connection, server_socket = initialise_connection()
130.
131.            while True:
132.                left_img, right_img = recieve(connection)
133.                left_img, right_img = preprocess(calibration, left_img, right_img)
134.                #cv2.imshow('images', np.hstack([left_img, right_img]))
135.                data = [left_img, right_img]
136.                for line in range(0, int(data[0].shape[0] / 20)):
137.                    data[0][line * 20, :] = 255
138.                    data[1][line * 20, :] = 255
139.
140.
141.                cv2.imshow('frames', np.hstack(data))
142.
143.                disparity = disparity_handler.get_disparity(left_img, right_img)
144.                cv2.imshow('disparity', cv2.applyColorMap(disparity, cv2.COLORMAP_JE
    T))
145.                nearest, middle = apply_thresholds(disparity)
146.                # cv2.imshow('thresh', np.hstack([nearest, middle]))
147.                signal = apply_segmentation(nearest, middle)
148.                send(connection, pickle.dumps(signal))
149.                cv2.waitKey(10)
150.
151.
152.            # finally:
153.            #     connection.close()
154.            #     server_socket.close()
155.
156.
157.        if __name__ == '__main__':
158.            main()
```

3. AUDIO FEEDBACK

```
import numpy as np
import sounddevice as sd


class AudioFeedback(object):
    def __init__(self, fs = 44100, duration = 10, volume = 0.3, channels = 2):
        self.fs = fs
        self.duration = duration
        self.volume = volume
        self.channels = channels

        sound_1 = (np.sin(2*np.pi*np.arange(self.fs*self.duration)*440/self.fs)).astype(np.float32)
        sound_2 = (np.sin(2*np.pi*np.arange(self.fs*self.duration)*500/self.fs)).astype(np.float32)
        sound_3 = (np.sin(2*np.pi*np.arange(self.fs*self.duration)*540/self.fs)).astype(np.float32)

        self.frequency_map = {1: sound_1, 2: sound_2, 3: sound_3}
        self.size = sound_1.size
        self.init_playback()

    def init_playback(self):
        sd.default.samplerate = self.fs
        sd.default.channels = self.channels

    def update(self, signal):
        freq = signal.count(1)
        sound = np.zeros([self.size, 2])
        if signal[0] == 0 and signal[1] == 1 and signal[2] == 0:
            sound = np.hstack([self.frequency_map[freq], self.frequency_map[freq]])

        else:
            if signal[0] == 1:
                sound[:, 0] = self.frequency_map[freq]
            if signal[2] == 1:
                sound[:, 1] = self.frequency_map[freq]

        sd.play(self.volume* sound, loop = True, mapping = [1, 2])

    def stop(self):
        sd.stop()


if __name__ == '__main__':
    fs = 44100
    duration = 10.0
    volume = 0.3
    channels = 2
    feedback = AudioFeedback(fs = fs, duration = duration, volume = volume, channels = channels)

    while True:
        try:
            signal = list(map(float, input().strip().split(' ')))
            feedback.update(signal)

        except:
            pass
            break

    feedback.stop()
```

4. IMAGE LOADER

```
import os
import sys
from threading import Thread

import cv2


class CaptureImage(object):
    '''''A class to capture images from webcam in a separate thread. It allows
    access to only the recent most image pairs.
    Methods defined here:
    @public: start()
             stop()
             load_images()
    @private: _capture_frames()
    '''
    def __init__(self, LEFT_CAM, RIGHT_CAM):
        '''''
        Initialise 2 webcams to capture images. Take 20 images to be discared,
        so that a small amount of time is given for webcams to adjust
        @params: LEFT_CAM - Port number corresponding to left webcam
                 RIGHT_CAM - Port number corresponding to right webcam
        '''
        self.left_capture = cv2.VideoCapture(LEFT_CAM)
        self.right_capture = cv2.VideoCapture(RIGHT_CAM)

        if not self.left_capture.isOpened():
            self.left_capture.open()
        if not self.right_capture.isOpened():
            self.right_capture.open()

        self.stopped = False

        for _ in range(20):
            ret, self.left_frame = self.left_capture.read()
            ret, self.right_frame = self.right_capture.read()

    def start(self):
        '''''Start a thread to capture images
        '''
        self.t = Thread(target= self._capture_frames)
        self.t.setDaemon(True)
        self.t.start()

    def _capture_frames(self):
        '''''A method for thread to loop while capturing left and right images
        from webcams. The recent most image pairs are stored.
        '''
        while True:
            if self.stopped :
                self.t.stop()
                self.left_capture.close()
                self.right_capture.close()
                return
            ret, self.left_frame = self.left_capture.read()
            ret, self.right_frame = self.right_capture.read()


    def load_images(self):
        '''''Returns the recent most left and right image pairs.
        '''
        return self.left_frame, self.right_frame
```

```
64.
65.     def stop(self):
66.         '''Assign a thread termination indicating variable to True.
67.         '''
68.         self.stopped = True
69.
70.
71. class ReadImages(object):
72.     '''
73.     A class to load left and right image pairs from the given folder.
74.     Methods defined here:
75.     @public: load_images()
76.              start()
77.              stop()
78.     '''
79.     def __init__(self, folder):
80.         '''Initiales the left and right image file names present in folder.
81.         @params: folder - Name of the folder which contains images in the
82.                           format left<sequence_number> and
83.                           right<sequence_number>.
84.         '''
85.         left_images = [file for file in os.listdir(folder) \
86.                                 if file.startswith('left')]
87.         right_images = [file for file in os.listdir(folder) \
88.                                 if file.startswith('right')]
89.         self.images_folder = []
90.         for file in left_images:
91.             string = 'right'+ file[4:]
92.             if string in right_images:
93.                 self.images_folder.append(os.path.join(folder, file))
94.                 self.images_folder.append(os.path.join(folder, string))
95.
96.     def load_images(self):
97.         '''Loads images 2 at a time corresponding to left and right images.
98.         '''
99.         if self.images_folder:
100.                 left_frame, right_frame = [cv2.imread(file) \
101.                                                 for file in self.images_folder[:2]]

102.                 self.images_folder = self.images_folder[2:]
103.
104.                 return left_frame, right_frame
105.
106.             else:
107.                 sys.exit()
108.
109.         def start(self):
110.             '''A method just to discard initial 3 image pairs.
111.             '''
112.             self.images_folder = self.images_folder[6:]
113.
114.         def stop(self):
115.             '''Dummy method for compatibility.
116.             '''
117.             return
```

5. CALIBRATION LOADER

```
import os
import numpy as np
import cv2

class Calibration(object):
    '''''
    A class to load all calibration files
    Methods defined:
    @public: load_calibration_files(folder_name)
    @private: _modify_undistort_and_rectify_maps()
    '''
    def __init__(self, size):
        '''''
        Initialise all calibration parameters
        Parameters initialised:
        cam_mats, dist_coefs, rot_mat, trans_vec, e_mat, f_mat, rect_trans,
        proj_mats, disp_to_depth_mat, valid_boxes, undistortion_map,
        rectification_map
        @params: size - Size of the image.
        '''
        self.cam_mats = {"left": None, "right": None}
        #: Distortion coefficients (D)
        self.dist_coefs = {"left": None, "right": None}
        #: Rotation matrix (R)
        self.rot_mat = None
        #: Translation vector (T)
        self.trans_vec = None
        #: Essential matrix (E)
        self.e_mat = None
        #: Fundamental matrix (F)
        self.f_mat = None
        #: Rectification transforms (3x3 rectification matrix R1 / R2)
        self.rect_trans = {"left": None, "right": None}
        #: Projection matrices (3x4 projection matrix P1 / P2)
        self.proj_mats = {"left": None, "right": None}
        #: Disparity to depth mapping matrix (4x4 matrix, Q)
        self.disp_to_depth_mat = None
        #: Bounding boxes of valid pixels
        self.valid_boxes = {"left": None, "right": None}
        #: Undistortion maps for remapping
        self.undistortion_map = {"left": None, "right": None}
        #: Rectification maps for remapping
        self.rectification_map = {"left": None, "right": None}
        #: Size of the image
        self.size = size

    def load_calibration_files(self, folder):
        '''''
        Load all calibration files and also modify undistortion_maps and
        rectification_maps
        @params: folder: name of the calibration folder
        '''
        for key, item in self.__dict__.items():
            if key == 'size':
                continue
            if isinstance(item, dict):
                for side in ("left", "right"):
                    filename = os.path.join(folder,
                                            "{}_{}.npy".format(key, side))
                    self.__dict__[key][side] = np.load(filename)
            else:
                filename = os.path.join(folder, "{}.npy".format(key))
                self.__dict__[key] = np.load(filename)
```

```
64.
65.         self._modify_undistort_and_rectify_maps()
66.
67.
68.     def _modify_undistort_and_rectify_maps(self):
69.         (self.undistortion_map['left'],
70.         self.rectification_map['left']) = cv2.initUndistortRectifyMap(
71.                                     self.cam_mats['left'],
72.                                     self.dist_coefs['left'],
73.                                     self.rect_trans['left'],
74.                                     self.proj_mats['left'], self.size,
75.                                     cv2.CV_32FC1)
76.         (self.undistortion_map['right'],
77.          self.rectification_map['right']) = cv2.initUndistortRectifyMap(
78.                                         self.cam_mats['right'],
79.                                         self.dist_coefs['right'],
80.                                         self.rect_trans['right'],
81.                                         self.proj_mats['right'], self.size,
82.                                         cv2.CV_32FC1)
```

6. DISPARITY COMPUTATION

```
1.  import cv2
2.  import numpy as np
3.  # from multiprocessing import process
4.
5.
6.  class DisparityCreator(object):
7.      def __init__(self, Lambda, sigma):
8.          self.left_matcher = cv2.StereoSGBM_create(
9.                                      minDisparity = 0,
10.                                     numDisparities = 64,
11.                                     blockSize = 3,
12.                                     P1 = 73,
13.                                     P2 = 2600,
14.                                     disp12MaxDiff = 43, #initial 43
15.                                     uniquenessRatio = 10,
16.                                     speckleRange = 1,
17.                                     preFilterCap = 0,
18.                                     speckleWindowSize = 10,
19.                                     mode = 1
20.                                     )
21.
22.         self.wls_filter = cv2.ximgproc.createDisparityWLSFilter(self.left_matcher)

23.         self.right_matcher = cv2.ximgproc.createRightMatcher(self.left_matcher)
24.         self.wls_filter.setLambda(Lambda)
25.         self.wls_filter.setSigmaColor(sigma)
26.
27.     def get_disparity(self, left_img, right_img):
28.         left_disparity = self.left_matcher.compute(left_img, right_img)
29.         right_disparity = self.right_matcher.compute(right_img, left_img)
30.         filtered_disparity = self.wls_filter.filter(left_disparity, left_img, None,
     right_disparity)
31.
32.         filtered_disparity[filtered_disparity > 1008] = 1008
33.         filtered_disparity[filtered_disparity < -16] = -16
34.         filtered_disparity = (((filtered_disparity + 16)/ 1008) * 255).astype(np.ui
    nt8)
35.
36.         kernel = np.ones((3,3),np.uint8)
37.         disparity = cv2.morphologyEx(filtered_disparity,cv2.MORPH_OPEN,kernel, iter
    ations = 2)
38.         coloured_disparity = cv2.applyColorMap((disparity).astype(np.uint8), cv2.CO
    LORMAP_JET)
39.         cv2.imshow('disparity', coloured_disparity)
40.         return disparity
```