POTHOLE DETECTION

Project Report

Eklavya Mentorship Programme

At

SOCIETY OF ROBOTICS AND AUTOMATION, VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE, MUMBAI

SEPTEMBER 2022

ACKNOWLEDGEMENT

We are thankful to our mentors Toshan Luktuke, Dhruv Kunjadiya and Rishabh Bali who provided us with the professional advice we needed in order to successfully explore the new field of Computer Vision and Stereo Vision.

We have the utmost gratitude for our mentors' perseverance, inspiration, passion, and vast expertise that they generously shared with us during the course of the project.

We also want to express our gratitude to all of the other SRA VJTI mentors for their unwavering encouragement and for providing us with the chance to participate in Eklavya 2022.

Dhruvanshu Joshi

dhruvanshu.joshi@gmail.com

+91 7506523660

Prachi Doshi

prachidoshi24@gmail.com

+91 9137063968

Table of Contents

1.Project Overview

1.1 Description of Project and Use Case

Potholes are bowl-shaped openings in the pavement that can be up to several inches deep. They are caused by the weathering and wear-and-tear of the road. When a pothole forms, its depth can increase by several inches, and rainwater speeds up the process, making it one of the prime causes for car accidents. Potholes are a major contributing factor in both vehicle and motorbike accidents. When driving at a high pace, potholes on the road are rarely visible. Additionally, if a car passes a pothole rapidly, the impact could damage the tyres. Even though drivers may notice the pothole before passing it, it is frequently too late for them to stop. Cars can roll over or rear-end each other with any abrupt turn or halt.

Between 2015 and 2017, accidents involving potholes killed 3,000 people per year on average in India. In addition to this, between 2013 and 2018, there were 639 occurrences of drowning in open manholes, gutters, and the sea in Mumbai, resulting in 328 fatalities. More than 900 potholes were counted on Mumbai's roads by the BMC last year. However, according to data provided by the Delhi PWD, there are 1,357 similar pits in Delhi.

Hence, to prevent such hazardous and dangerous accidents, we have made a detector which will detect any pothole present on the road and inform the driver so that he can handle the situation in a better way.

1.2 Technology Used

Technologies that made this project possible:

- Opencv- OpenCV is a library of programming functions mainly aimed at real-time computer vision. It is an open-source computer vision and machine learning software library. This library will allow us to read the depth image taken from the Oak-d camera stored in our local repository.

- Numpy- NumPy is the fundamental package for scientific computing in Python. It is a Python library used for working with arrays. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

- Matplotlib - Python's Matplotlib toolkit provides a complete tool for building static, animated, and interactive visualisations. Matplotlib makes difficult things possible and simple things easy. It makes interactive figures that can zoom, pan, and update, customises visual style and layout, exports to numerous file formats, and is embedded in JupyterLab and graphical user interfaces. It also produces publication-quality charts. We use this library to visualize various 3-dimensional co-ordinates and surfaces through graph.

- Scikit Learn - Scikit-learn is a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license. We use this library to develop surface fit using optimal least square wherever required.

- DepthAI - Robots and computers can sense the environment like humans do, identifying things or features and where they are in the actual world, with the help of the Spatial AI platform called DepthAI. We use this module to capture the stereo images of a pothole region and obtain the depth map corresponding to it.

- Jupyter notebook - Project Jupyter is a project with goals to develop open-source software, open standards, and services for interactive computing across multiple programming languages. We use the Jupyter notebook to write our code.

2. Introduction

2.1 General

The main idea of this project is to detect potholes present on the road and label them individually. In cases where there are more than one potholes, we label them individually while considering them to be separate potholes and also display their relative individual distance in terms of pixels.

We use the Oak-D camera and DepthAPI to get the stereo images as well as the depth map of the road surface.

Using these we will chart the surface fit for the road in case there was no pothole using bi-square weighted robust least-squares.

On comparing this surface we recognise the candidate pothole region in the image and use image processing to eliminate noise.

Finally, we use connected component labelling to label the road potholes and draw a rectangle around the potholes.

All these tasks are performed using DepthAPI functions provided by DepthAI, OpenCV functions, functions provided by SciKit Learn, MatplotLib and also Numpy.

2.2 Basic Project domains

- OpenCV
- Depth-AI
- Numpy
- SkLearn
- Matplotlib
- SciPy

2.3 Theory

In this project, we first get the stereo images of the environment using the OAK-D camera. Then we rectify these images using the concept of stereo rectification and also preprocess them to make them easier to implement. Next we get the disparity image using these rectified images and then we convert the disparity image to depth image. We generate a scatter plot of points using the depth values from the depth image. Using bi-square weighted least squares robust regression methods, we perform surface fitting on this scatter plot and find a most suitable replica of the surface. The points from the plot which are way below this surface are considered to be potholes. This plot is then masked on the depth image. The depth image is then converted to binary image which is used to perform pothole labelling. Thus, potholes are detected and labeled.

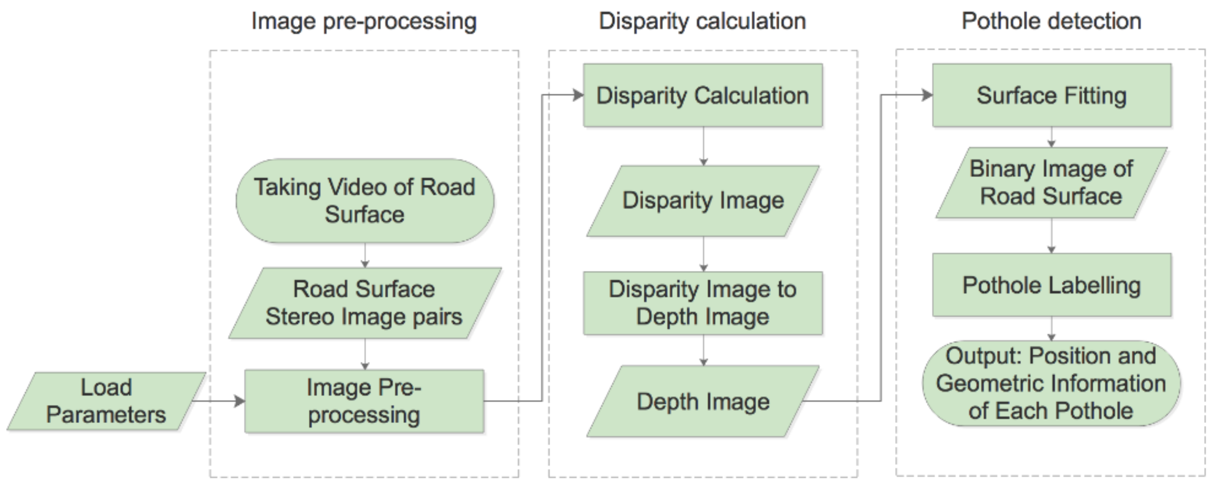


Image pre-processing
Disparity calculation
Pothole detection
Taking Video of Road Surface
Road Surface Stereo Image pairs
Load Parameters
Image Pre-processing
Disparity Calculation
Disparity Image
Disparity Image to Depth Image
Depth Image
Surface Fitting
Binary Image of Road Surface
Pothole Labelling
Output: Position and Geometric Information of Each Pothole

Fig. 2. On-line Flowchart of the Pothole Detection System.

- **Linear Algebra**

  The height, width and depth of the image are all considered to be vectors. Hence all the operations done on them are performed using the concepts of Linear Algebra.

  The camera calibration process also uses the concepts of linear algebra. But the OAK-D camera is factory calibrated so we do not need to apply these concepts for that purpose. Also, since we are using the Depth map generated by the camera itself, using DepthAI methods, the theory pertaining to it is also not required.

- **Triangulation**

  Method of triangulation is used to reproject the disparity image. In this method we consider a point in one of the images and then search for the same point in the other image. These points in both these 2-D images are then pulled behind in the third axis to get the depth. These depth values are stored to get the depth image and also the depth map.

  The method used to get the depth values is called the triangulation method. It is a completely geometric based approach.
  The steps to get the depth are as follows:

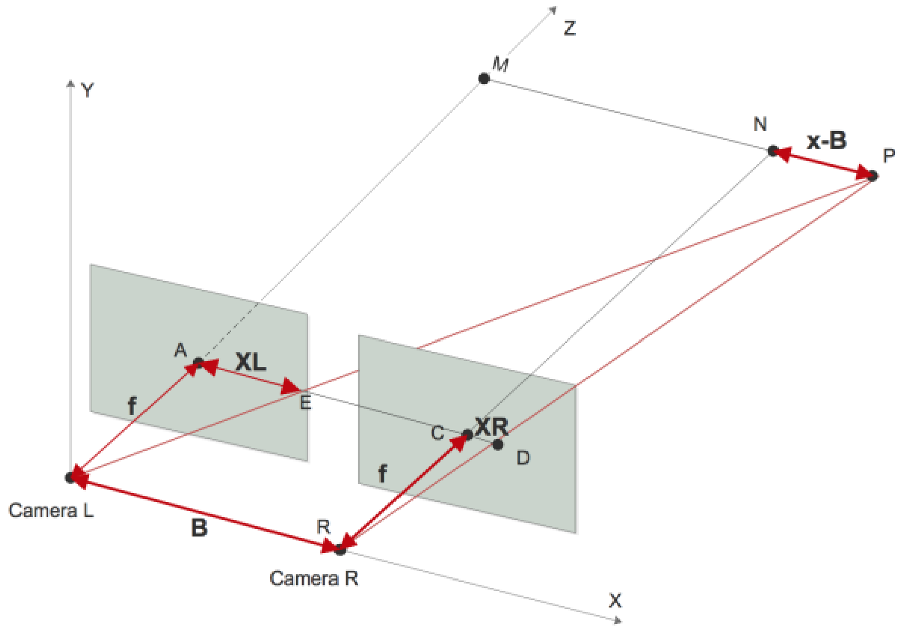


Figure 3.7. Triangulation Method to Reproject Disparity Image

Let optical center of left camera = origin
f = focal length
z = depth point at X
d = disparity

Along the X-axis:
For the left camera:

$$\Delta\, OAE \;\sim\; \Delta\, OMP$$

$$\therefore \frac{z}{f} = \frac{x}{XL} \qquad \ldots (1)$$

For the right camera:

$$\Delta\, OCD \;\sim\; \Delta\, ONP$$

$$\therefore \frac{z}{f} = \frac{x-B}{XR} \qquad \ldots (2)$$

Similarly, along the Y-axis:

$$\frac{z}{f} = \frac{y}{YL} = \frac{y}{YR} \qquad \ldots (3)$$

From the stereo camera calibrations, we know

$$z = \frac{B*f}{d} \qquad \ldots (4)$$

$\therefore$ from (1), (2), (3) and (4), we get

$$x = \frac{B*XL}{d} \quad \text{and} \quad y = \frac{B*YL}{d}$$

- **Robust Regression**

  To minimise and eliminate the errors in road surface fitting, we are using Bi-Square weighted Least Square method. This method minimizes a weighted sum of squares, where the weight given to each data point depends on how far the point is from the fitted line. Points near the line get full weight. Points farther from the line get reduced weight. Points that are farther from the line than would be expected by random chance get zero weight.

  For this project we have used the function scipy.optimize from SciPy open source library. SciPy optimize provides functions for minimizing (or maximizing) objective functions, possibly subject to constraints. It includes solvers for nonlinear problems (with support for both local and global optimization algorithms), linear programing, constrained and nonlinear least-squares, root finding, and curve fitting.

This function uses the Cauchy-Lorentz regression. The Cauchy distribution $f(x;\ x0,\ \gamma)$ is the distribution of the x-intercept of a ray issuing from $(x0,\ \gamma)$ with a uniformly distributed angle. It is also the distribution of the ratio of two independent normally distributed random variables with mean zero.

(1) Fit the road surface model with bi-square weighted robust least-squares:

$$y = a_1 + a_2 x + a_3 z + a_4 x^2 + a_5 xz + a_6 z^2 \tag{3.21}$$

(2) Minimize the residuals $r_i = (y_i - \widehat{y})^2$ by differentiating the sum with respect to the coefficients. The resulting equation in matrix form is illustrated as follow:

$$\begin{bmatrix} n & S_x & S_z & S_{x^2} & S_{xz} & S_{y^2} \\ S_x & S_{x^2} & S_{xz} & S_{x^3} & S_{x^2 z} & S_{xz^2} \\ S_z & S_{xz} & S_{z^2} & S_{x^2 z} & S_{xz^2} & S_{z^3} \\ S_{x^2} & S_{x^3} & S_{x^2 z} & S_{x^4} & S_{x^3 z^2} & S_{x^3 z} \\ S_{xz} & S_{x^2 z} & S_{xz^2} & S_{x^3 z} & S_{x^2 z^2} & S_{xz^3} \\ S_{z^2} & S_{xz^2} & S_{z^3} & S_{x^2 z^2} & S_{xz^3} & S_{z^4} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} = \begin{bmatrix} S_y \\ S_{xy} \\ S_{zy} \\ S_{x^2 y} \\ S_{xzy} \\ S_{z^2 y} \end{bmatrix}, \tag{3.22}$$

where, $S_{xz} = \sum_{i=1}^{n} \omega_i x_i z_i$

(3) Compute the adjusted residuals as following:

$$r_{adj} = \frac{r_i}{\sqrt{1 - h_i}}, \tag{3.23}$$

h_i are leverages that adjust the residuals by reducing the weight of high-leverage data. Then the standardized adjusted residuals are calculated as following:

$$u = \frac{r_{adj}}{Ks}, \tag{3.24}$$

K is the tuning constant equal to 4.685. s is the robust variance given by the median absolute deviation of the residual (MAD)/0.6745.

(4) Compute the bi-square weights as the function of u_i for the i_{th}points as following:

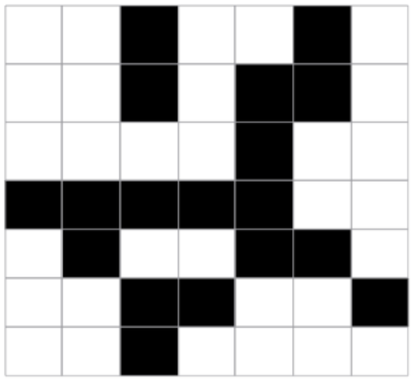
$$w_i = \begin{cases} (1 - (u_i)^2)^2 & |u_i| < 1 \\ 0 & |u_i| \geq 1 \end{cases} \tag{3.25}$$

(5) If the fit converges, then we are done. Otherwise, calculate the next iteration of the fitting processes from the first step until it converges.

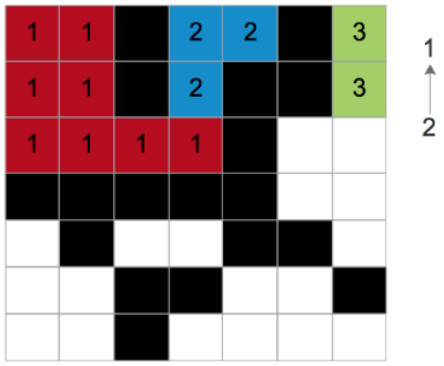
- **Two-Pass Method for Labelling**

  For identifying and labelling the pothole, we use the two pass method based on the connected component labelling method which is as follows:

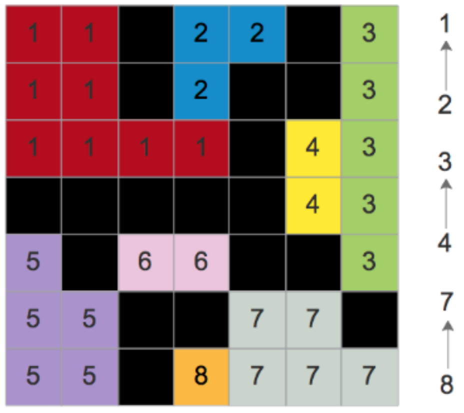
  Taking Figure as an example, we start off at the top left corner. The first pixel (1,1) is not background (black in this example) and there is not any label around this pixel, so we create a new label named 1(marked as red) for this pixel. Then we move to pixel (1, 2), it is not background and there is a label to its left, so we copy the label of its left pixel. Then the next pixel (1, 3) is the background, so we skip it. Next comes the pixel (1, 4), the situation of this pixel is the same as pixel (1, 1), so we create a new label name 2 (marked as blue). Repeat these steps for each pixel until pixel (3, 4). There are 2 labels around this pixel. We label this pixel as 1 (the smaller number). Then mark that label 2 (the larger number) is a child of label 1 as shown in Figure. Following the above rule, we can label the whole image illustrated as Figure. The flowchart of the first pass of judgement is illustrated in Figure.



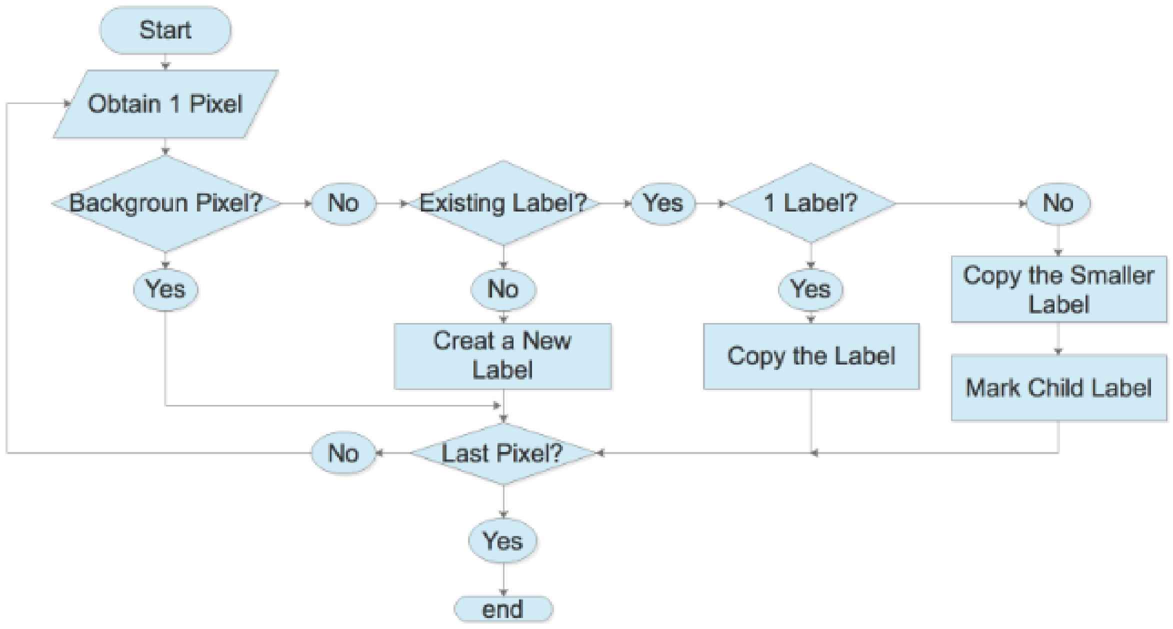
Binary Image



Two labels around one label

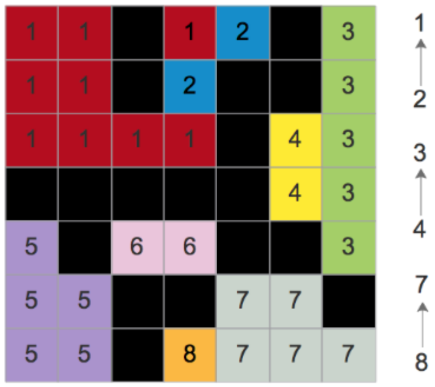


Labels after 1st pass

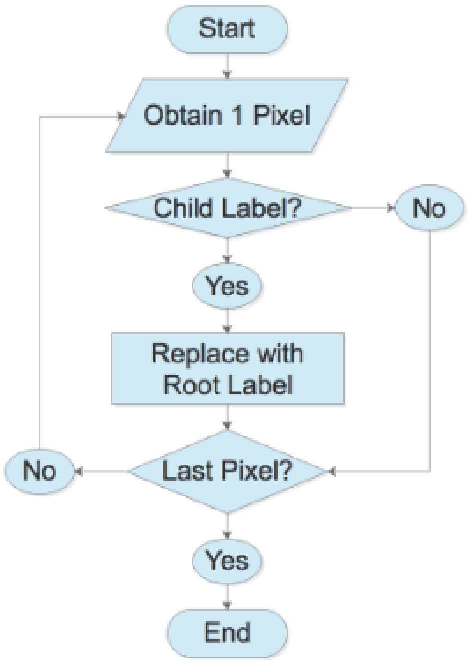


First Pass Flow Chart

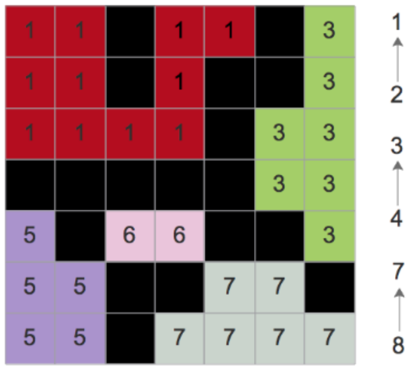
Then we start the second pass of labelling the image. We start at the top left corner as well and check whether its label is a child of any other label. If it is a child label of a smaller label, replace the child label with the smaller label as shown in Figure. Repeat these judgements for each pixel in the image. Finally, the labelling result is shown in Figure.



2nd Pass Image



Second Pass Flow Chart



Labelling Result

3. Methods and Stages of Progress

3.1 Prerequisite

- Download and install Jupyter notebook. (click here to know more)
- Install Numpy
- Install Open cv-python
- Install scikit learn
- Install matplotlib
- Install DepthAI (refer the read.me section here)

3.2 Pothole Detection and It's code

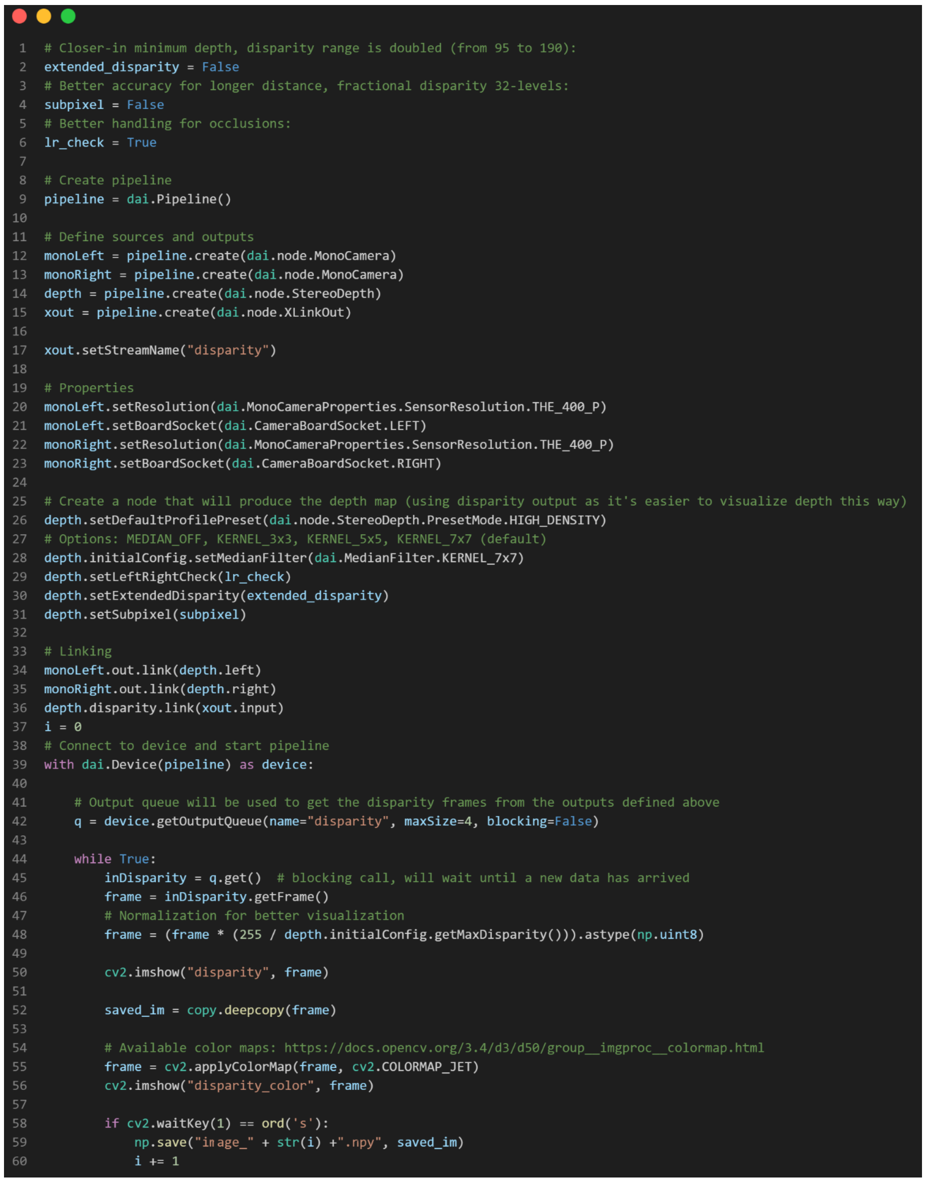
We adopt extensive use of Oak-D camera and opencv to detect the potholes on road surface.

We can break down the process into 5 major steps:

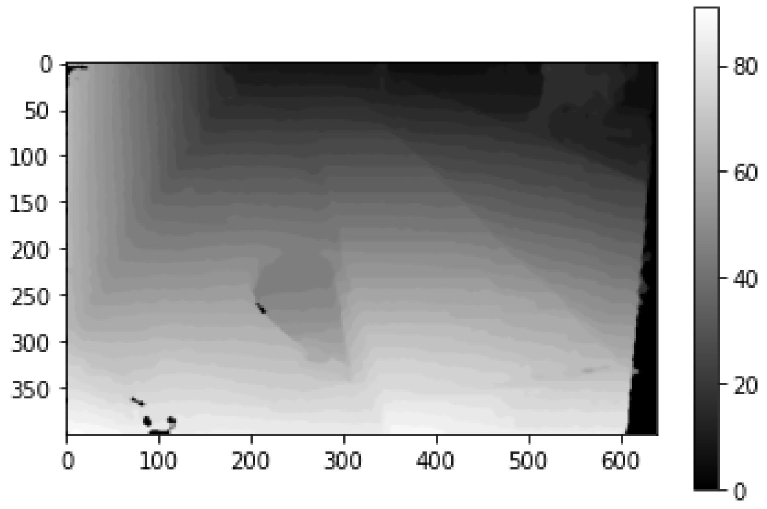
- **Depth-map saving and mono-left/right image saving using the oak-D camera**

  We obtain the depth map of the interested scene by attaching the Oak-D camera to the host pc and running the Depth saving code to save the depth map in .npy format.

  Then we use the os module to navigate the saved depth in .npy format and use numpy library to load the data. The generated depth has a lot of garbage values at some places due to problems caused by rectification, occlusion , etc.



```python
# Closer-in minimum depth, disparity range is doubled (from 95 to 190):
extended_disparity = False
# Better accuracy for longer distance, fractional disparity 32-levels:
subpixel = False
# Better handling for occlusions:
lr_check = True

# Create pipeline
pipeline = dai.Pipeline()

# Define sources and outputs
monoLeft = pipeline.create(dai.node.MonoCamera)
monoRight = pipeline.create(dai.node.MonoCamera)
depth = pipeline.create(dai.node.StereoDepth)
xout = pipeline.create(dai.node.XLinkOut)

xout.setStreamName("disparity")

# Properties
monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

# Create a node that will produce the depth map (using disparity output as it's easier to visualize depth this way)
depth.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_DENSITY)
# Options: MEDIAN_OFF, KERNEL_3x3, KERNEL_5x5, KERNEL_7x7 (default)
depth.initialConfig.setMedianFilter(dai.MedianFilter.KERNEL_7x7)
depth.setLeftRightCheck(lr_check)
depth.setExtendedDisparity(extended_disparity)
depth.setSubpixel(subpixel)

# Linking
monoLeft.out.link(depth.left)
monoRight.out.link(depth.right)
depth.disparity.link(xout.input)
i = 0
# Connect to device and start pipeline
with dai.Device(pipeline) as device:

    # Output queue will be used to get the disparity frames from the outputs defined above
    q = device.getOutputQueue(name="disparity", maxSize=4, blocking=False)

    while True:
        inDisparity = q.get()  # blocking call, will wait until a new data has arrived
        frame = inDisparity.getFrame()
        # Normalization for better visualization
        frame = (frame * (255 / depth.initialConfig.getMaxDisparity())).astype(np.uint8)

        cv2.imshow("disparity", frame)

        saved_im = copy.deepcopy(frame)

        # Available color maps: https://docs.opencv.org/3.4/d3/d50/group__imgproc__colormap.html
        frame = cv2.applyColorMap(frame, cv2.COLORMAP_JET)
        cv2.imshow("disparity_color", frame)

        if cv2.waitKey(1) == ord('s'):
            np.save("image_" + str(i) +".npy", saved_im)
            i += 1
```

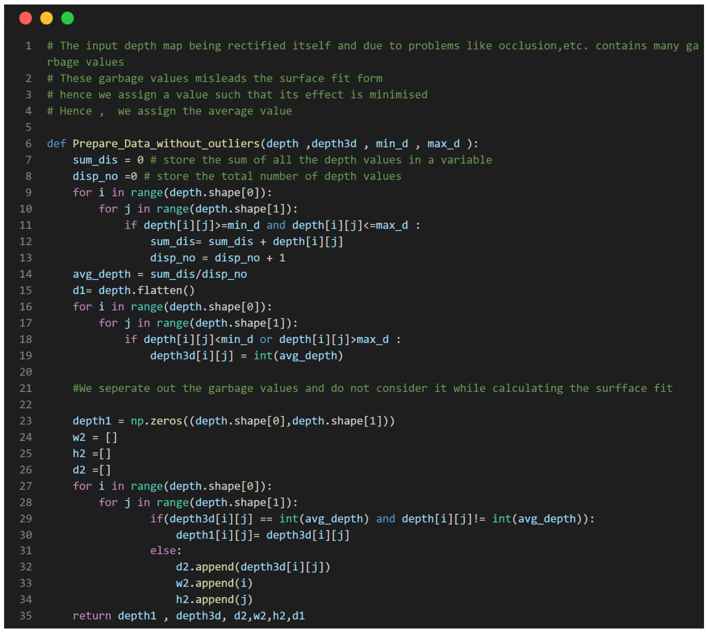


- **Using this depth map to generate a surface fit of the road**

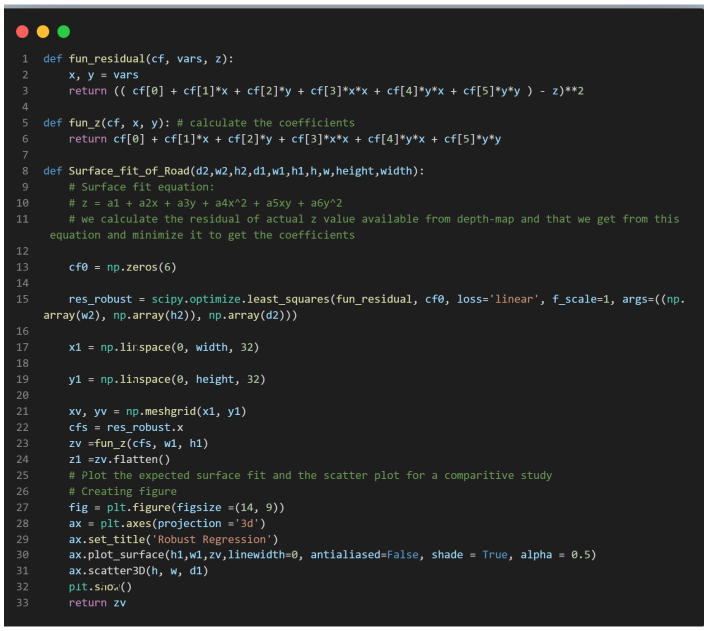
Once we have the Depth-map, we now need the Surface fit of the road. When using the least square method to fit the points to a road surface, all the points are typically considered to be of identical quality. Potholes or noise-related spots beneath the road's surface could affect the precision of the appropriate road surface. This road pothole identification technology minimises the impact of outliers during fitting adding an additional scale factor to the process (the weight).
Hence , to get the surface, we use the quadratic equation:
$z = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy + a_6 y^2$

We minimize the residuals $r_i = (z_i - z_b)^2$ by differentiating the sum with respect to the coefficients. Boiling it down to matrix form and then we perform the equations given under robust regression section.

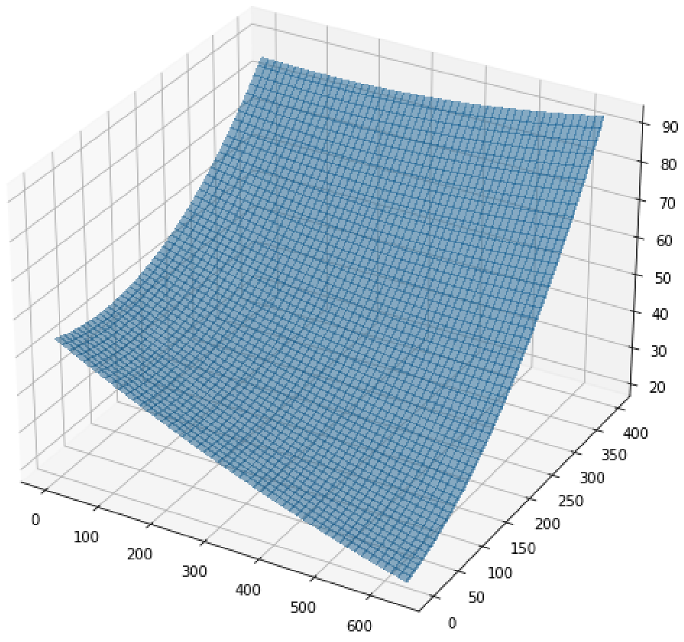
The data that's to be fed to the above equation must be free of any error present due to inaccuracy of the Oak-D camera. Now, we employ the scipy.optimise module to manage all of these calculations. SciPy optimise offers methods for minimising (or maximising), potentially constrained, objective functions. It features linear programming, restricted and nonlinear least-squares, root finding, and curve fitting methods, as well as solvers for nonlinear problems (with support for both local and global optimization procedures). Then, using constraints on the variables, we solve this nonlinear least-squares problem. The least-squares algorithm locates a local minimum of the cost function F(x) such that the impact of outliers is minimised, given the residuals f(x) (an m-D real function of n real variables) and the loss function (a scalar function). Here we utilise linear loss and a scale factor of 1 to optimise the time. Now by feeding the coefficients obtained to the quadratic equation we are able to obtain the surface chart of the road. We plot this using the matplotlib module.



```python
# The input depth map being rectified itself and due to problems like occlusion,etc. contains many garbage values
# These garbage values misleads the surface fit form
# hence we assign a value such that its effect is minimised
# Hence ,  we assign the average value

def Prepare_Data_without_outliers(depth ,depth3d , min_d , max_d ):
    sum_dis = 0 # store the sum of all the depth values in a variable
    disp_no =0 # store the total number of depth values
    for i in range(depth.shape[0]):
        for j in range(depth.shape[1]):
            if depth[i][j]>=min_d and depth[i][j]<=max_d :
                sum_dis= sum_dis + depth[i][j]
                disp_no = disp_no + 1
    avg_depth = sum_dis/disp_no
    d1= depth.flatten()
    for i in range(depth.shape[0]):
        for j in range(depth.shape[1]):
            if depth[i][j]<min_d or depth[i][j]>max_d :
                depth3d[i][j] = int(avg_depth)

    #We seperate out the garbage values and do not consider it while calculating the surfface fit

    depth1 = np.zeros((depth.shape[0],depth.shape[1]))
    w2 = []
    h2 =[]
    d2 =[]
    for i in range(depth.shape[0]):
        for j in range(depth.shape[1]):
                if(depth3d[i][j] == int(avg_depth) and depth[i][j]!= int(avg_depth)):
                    depth1[i][j]= depth3d[i][j]
                else:
                    d2.append(depth3d[i][j])
                    w2.append(i)
                    h2.append(j)
    return depth1 , depth3d, d2,w2,h2,d1
```



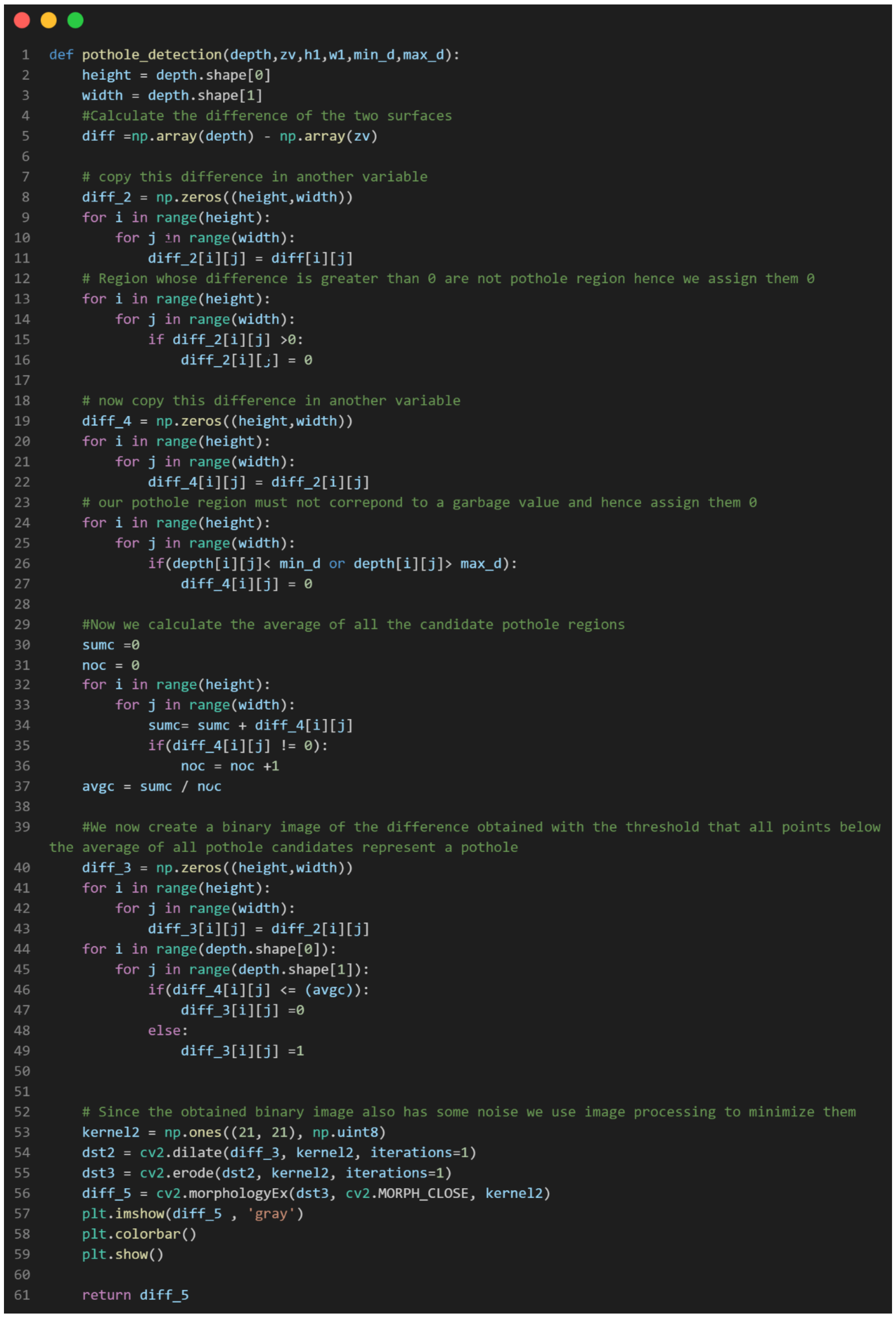
```python
def fun_residual(cf, vars, z):
    x, y = vars
    return (( cf[0] + cf[1]*x + cf[2]*y + cf[3]*x*x + cf[4]*y*x + cf[5]*y*y ) - z)**2

def fun_z(cf, x, y): # calculate the coefficients
    return cf[0] + cf[1]*x + cf[2]*y + cf[3]*x*x + cf[4]*y*x + cf[5]*y*y

def Surface_fit_of_Road(d2,w2,h2,d1,w1,h1,h,w,height,width):
    # Surface fit equation:
    # z = a1 + a2x + a3y + a4x^2 + a5xy + a6y^2
    # we calculate the residual of actual z value available from depth-map and that we get from this
 equation and minimize it to get the coefficients

    cf0 = np.zeros(6)

    res_robust = scipy.optimize.least_squares(fun_residual, cf0, loss='linear', f_scale=1, args=((np.
array(w2), np.array(h2)), np.array(d2)))

    x1 = np.linspace(0, width, 32)

    y1 = np.linspace(0, height, 32)

    xv, yv = np.meshgrid(x1, y1)
    cfs = res_robust.x
    zv =fun_z(cfs, w1, h1)
    z1 =zv.flatten()
    # Plot the expected surface fit and the scatter plot for a comparitive study
    # Creating figure
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    ax.set_title('Robust Regression')
    ax.plot_surface(h1,w1,zv,linewidth=0, antialiased=False, shade = True, alpha = 0.5)
    ax.scatter3D(h, w, d1)
    plt.show()
    return zv
```



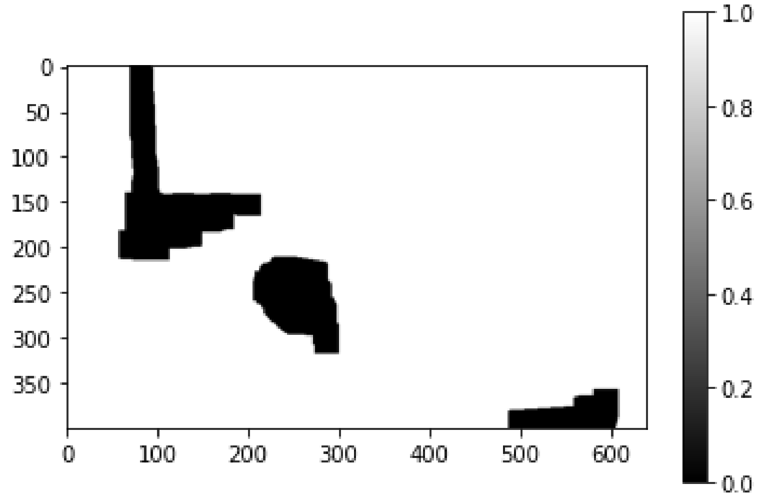
- ## Comparing the points on actual surface with this fit to detect candidate pothole regions

Once the chart is made , its now time to compare the actual 3 dimensional coordinates of the depth image to the ones predicted from the surface fit equation. Any point below this is considered to be a candidate pothole region.

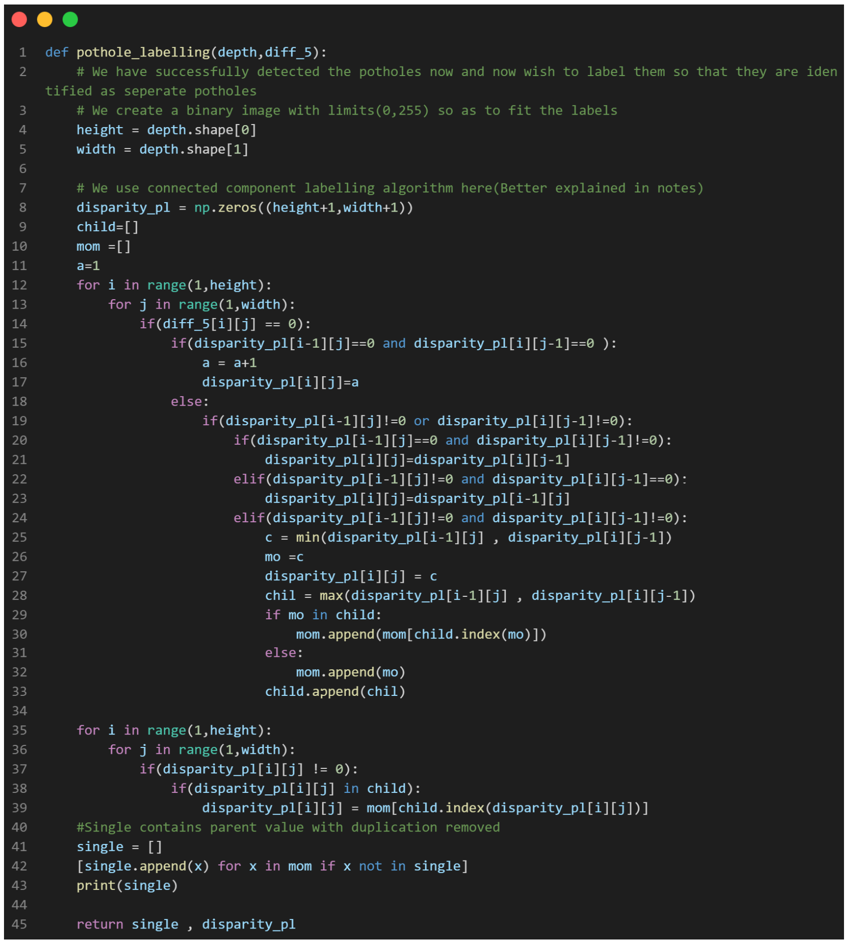
This process detects a lot of garbage values also as pothole. Therefore we discard them. Now we make a binary image corresponding to the depth image with the average pixel value of all candidate potholes being the threshold. Now we have minimised the error to a better extent but to further eliminate the noise we carry out image processing. We first dilate this binary image(to eliminate small black noise) and then dilation(to make our actual pothole region prominent) followed by the reverse of the above process i.e. opening to finally boil our problem down to only areas which are potentially pothole.



```python
def pothole_detection(depth,zv,h1,w1,min_d,max_d):
    height = depth.shape[0]
    width = depth.shape[1]
    #Calculate the difference of the two surfaces
    diff =np.array(depth) - np.array(zv)

    # copy this difference in another variable
    diff_2 = np.zeros((height,width))
    for i in range(height):
        for j in range(width):
            diff_2[i][j] = diff[i][j]
    # Region whose difference is greater than 0 are not pothole region hence we assign them 0
    for i in range(height):
        for j in range(width):
            if diff_2[i][j] >0:
                diff_2[i][j] = 0

    # now copy this difference in another variable
    diff_4 = np.zeros((height,width))
    for i in range(height):
        for j in range(width):
            diff_4[i][j] = diff_2[i][j]
    # our pothole region must not correpond to a garbage value and hence assign them 0
    for i in range(height):
        for j in range(width):
            if(depth[i][j]< min_d or depth[i][j]> max_d):
                diff_4[i][j] = 0

    #Now we calculate the average of all the candidate pothole regions
    sumc =0
    noc = 0
    for i in range(height):
        for j in range(width):
            sumc= sumc + diff_4[i][j]
            if(diff_4[i][j] != 0):
                noc = noc +1
    avgc = sumc / noc

    #We now create a binary image of the difference obtained with the threshold that all points below
the average of all pothole candidates represent a pothole
    diff_3 = np.zeros((height,width))
    for i in range(height):
        for j in range(width):
            diff_3[i][j] = diff_2[i][j]
    for i in range(depth.shape[0]):
        for j in range(depth.shape[1]):
            if(diff_4[i][j] <= (avgc)):
                diff_3[i][j] =0
            else:
                diff_3[i][j] =1


    # Since the obtained binary image also has some noise we use image processing to minimize them
    kernel2 = np.ones((21, 21), np.uint8)
    dst2 = cv2.dilate(diff_3, kernel2, iterations=1)
    dst3 = cv2.erode(dst2, kernel2, iterations=1)
    diff_5 = cv2.morphologyEx(dst3, cv2.MORPH_CLOSE, kernel2)
    plt.imshow(diff_5 , 'gray')
    plt.colorbar()
    plt.show()

    return diff_5
```

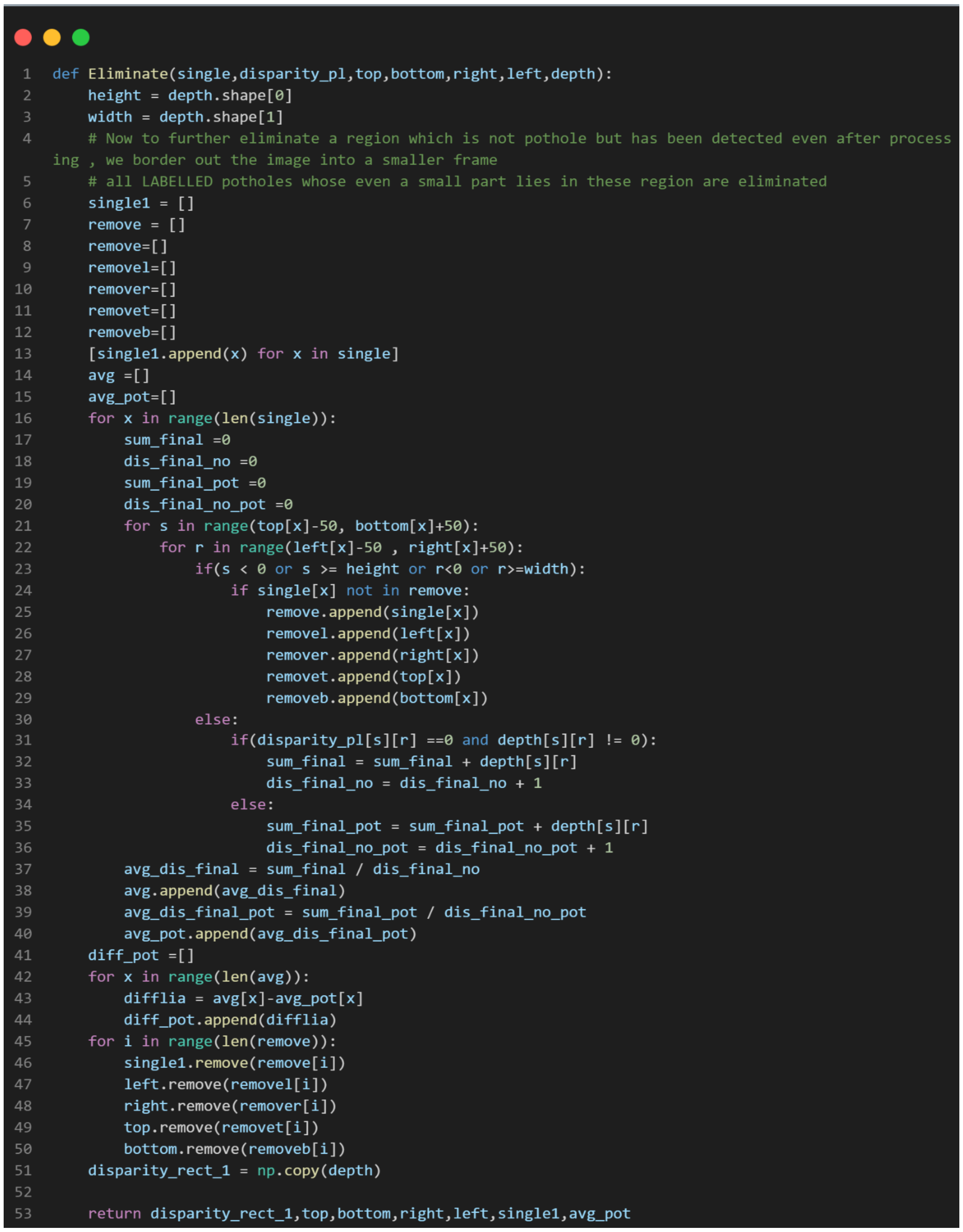


- **Labelling and separating the potholes**
  Once we have all the candidate potholes, we now have to label and separate them. We use connected component labelling and the flowchart as explained above.

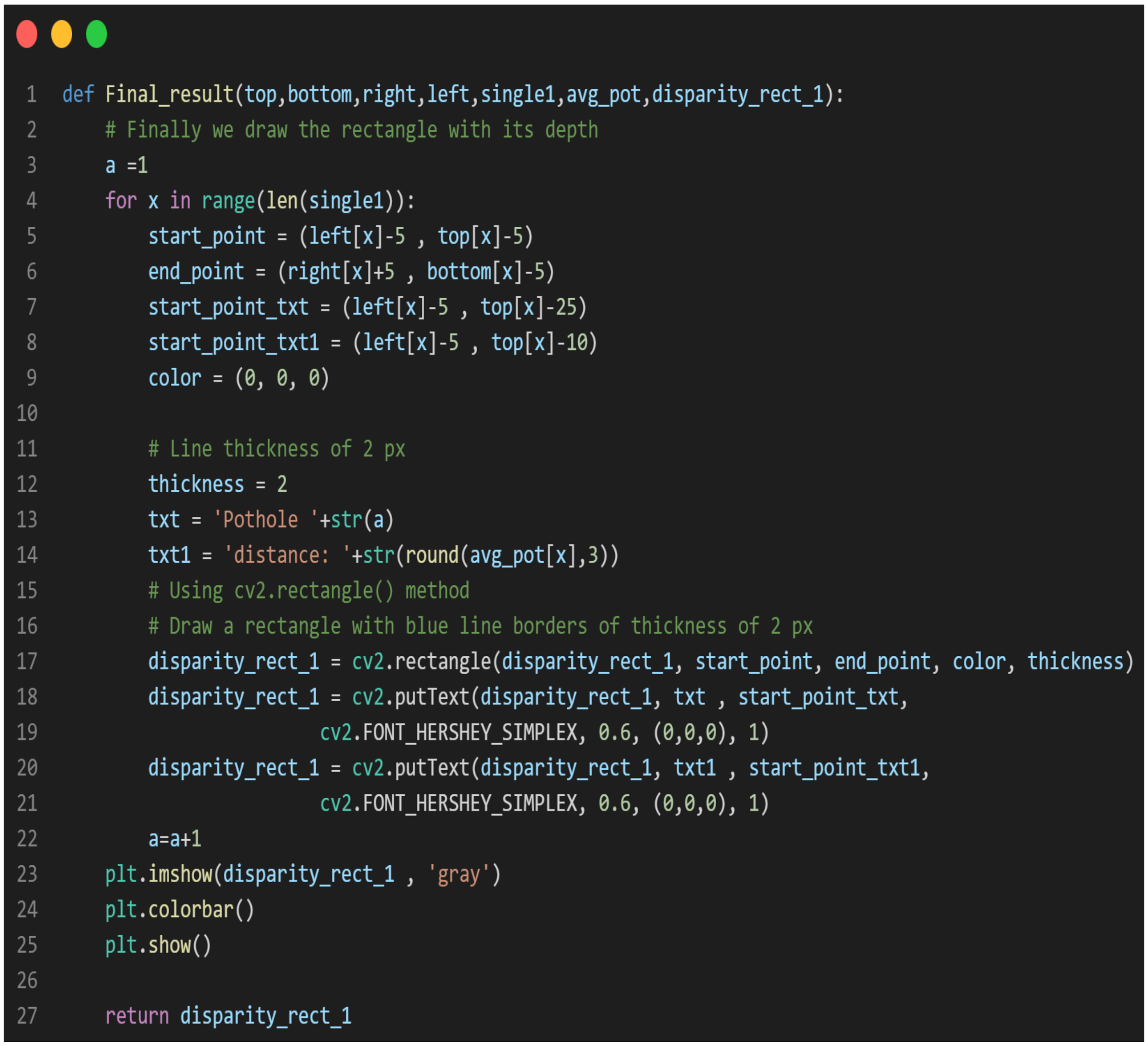


```python
def pothole_labelling(depth,diff_5):
    # We have successfully detected the potholes now and now wish to label them so that they are iden
tified as seperate potholes
    # We create a binary image with limits(0,255) so as to fit the labels
    height = depth.shape[0]
    width = depth.shape[1]

    # We use connected component labelling algorithm here(Better explained in notes)
    disparity_pl = np.zeros((height+1,width+1))
    child=[]
    mom =[]
    a=1
    for i in range(1,height):
        for j in range(1,width):
            if(diff_5[i][j] == 0):
                if(disparity_pl[i-1][j]==0 and disparity_pl[i][j-1]==0 ):
                    a = a+1
                    disparity_pl[i][j]=a
                else:
                    if(disparity_pl[i-1][j]!=0 or disparity_pl[i][j-1]!=0):
                        if(disparity_pl[i-1][j]==0 and disparity_pl[i][j-1]!=0):
                            disparity_pl[i][j]=disparity_pl[i][j-1]
                        elif(disparity_pl[i-1][j]!=0 and disparity_pl[i][j-1]==0):
                            disparity_pl[i][j]=disparity_pl[i-1][j]
                        elif(disparity_pl[i-1][j]!=0 and disparity_pl[i][j-1]!=0):
                            c = min(disparity_pl[i-1][j] , disparity_pl[i][j-1])
                            mo =c
                            disparity_pl[i][j] = c
                            chil = max(disparity_pl[i-1][j] , disparity_pl[i][j-1])
                            if mo in child:
                                mom.append(mom[child.index(mo)])
                            else:
                                mom.append(mo)
                            child.append(chil)

    for i in range(1,height):
        for j in range(1,width):
            if(disparity_pl[i][j] != 0):
                if(disparity_pl[i][j] in child):
                    disparity_pl[i][j] = mom[child.index(disparity_pl[i][j])]
    #Single contains parent value with duplication removed
    single = []
    [single.append(x) for x in mom if x not in single]
    print(single)

    return single , disparity_pl
```

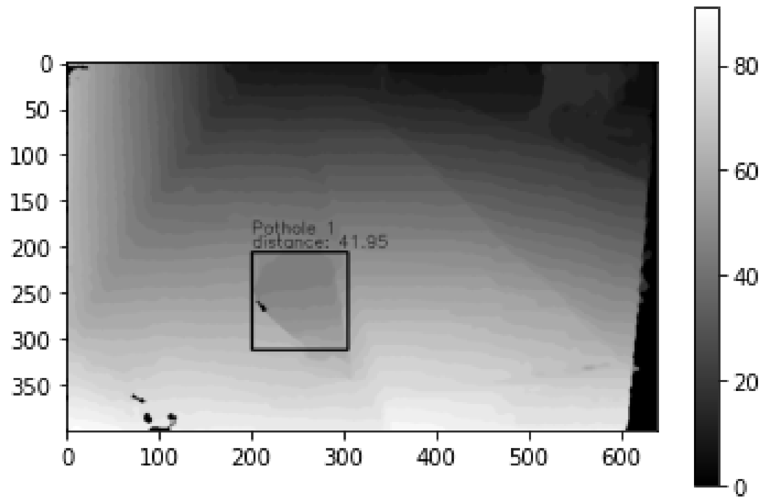
- **Eliminating incorrect potholes identified and displaying the result**
  As seen in the above picture , our algorithm also detects points which are not potholes but just depths(eg: junction of wall and floor) and such regions can be eliminated if we have a proper region of interest defined. Assuming that our ROI is a relatively central part of the image , we eliminate the boundary of thickness 50 from all 4 sides. Hence we are able to eliminate some more garbage potholes identified.



```python
def Eliminate(single,disparity_pl,top,bottom,right,left,depth):
    height = depth.shape[0]
    width = depth.shape[1]
    # Now to further eliminate a region which is not pothole but has been detected even after processing , we border out the image into a smaller frame
    # all LABELLED potholes whose even a small part lies in these region are eliminated
    single1 = []
    remove = []
    remove=[]
    removel=[]
    remover=[]
    removet=[]
    removeb=[]
    [single1.append(x) for x in single]
    avg =[]
    avg_pot=[]
    for x in range(len(single)):
        sum_final =0
        dis_final_no =0
        sum_final_pot =0
        dis_final_no_pot =0
        for s in range(top[x]-50, bottom[x]+50):
            for r in range(left[x]-50 , right[x]+50):
                if(s < 0 or s >= height or r<0 or r>=width):
                    if single[x] not in remove:
                        remove.append(single[x])
                        removel.append(left[x])
                        remover.append(right[x])
                        removet.append(top[x])
                        removeb.append(bottom[x])
                else:
                    if(disparity_pl[s][r] ==0 and depth[s][r] != 0):
                        sum_final = sum_final + depth[s][r]
                        dis_final_no = dis_final_no + 1
                    else:
                        sum_final_pot = sum_final_pot + depth[s][r]
                        dis_final_no_pot = dis_final_no_pot + 1
        avg_dis_final = sum_final / dis_final_no
        avg.append(avg_dis_final)
        avg_dis_final_pot = sum_final_pot / dis_final_no_pot
        avg_pot.append(avg_dis_final_pot)
    diff_pot =[]
    for x in range(len(avg)):
        difflia = avg[x]-avg_pot[x]
        diff_pot.append(difflia)
    for i in range(len(remove)):
        single1.remove(remove[i])
        left.remove(removel[i])
        right.remove(remover[i])
        top.remove(removet[i])
        bottom.remove(removeb[i])
    disparity_rect_1 = np.copy(depth)

    return disparity_rect_1,top,bottom,right,left,single1,avg_pot
```



```python
def Final_result(top,bottom,right,left,single1,avg_pot,disparity_rect_1):
    # Finally we draw the rectangle with its depth
    a =1
    for x in range(len(single1)):
        start_point = (left[x]-5 , top[x]-5)
        end_point = (right[x]+5 , bottom[x]-5)
        start_point_txt = (left[x]-5 , top[x]-25)
        start_point_txt1 = (left[x]-5 , top[x]-10)
        color = (0, 0, 0)

        # Line thickness of 2 px
        thickness = 2
        txt = 'Pothole '+str(a)
        txt1 = 'distance: '+str(round(avg_pot[x],3))
        # Using cv2.rectangle() method
        # Draw a rectangle with blue line borders of thickness of 2 px
        disparity_rect_1 = cv2.rectangle(disparity_rect_1, start_point, end_point, color, thickness)
        disparity_rect_1 = cv2.putText(disparity_rect_1, txt , start_point_txt,
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,0,0), 1)
        disparity_rect_1 = cv2.putText(disparity_rect_1, txt1 , start_point_txt1,
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,0,0), 1)
        a=a+1
    plt.imshow(disparity_rect_1 , 'gray')
    plt.colorbar()
    plt.show()

    return disparity_rect_1
```

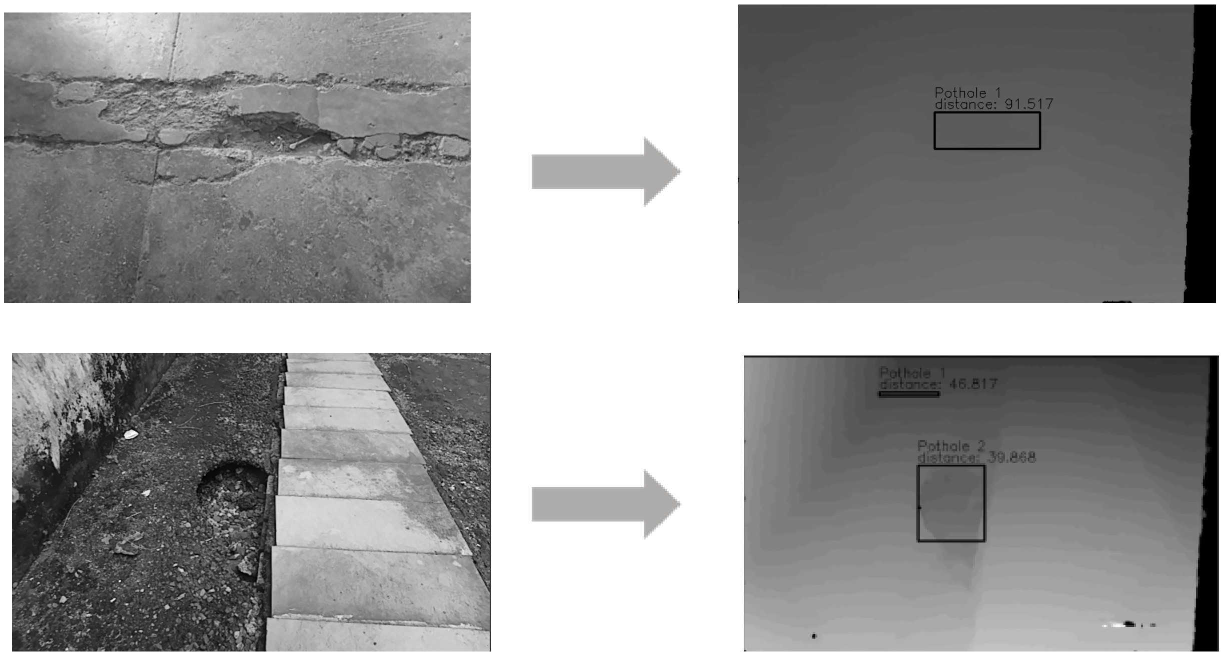


4. Conclusion And Future Work

4.1 Conclusion

The key project learning includes :- Handling of Oak-D camera and DepthAPI, OpenCV and image processing and least square regression analysis as well as connected component labelling algorithm.

Hence, we have here discussed a method that detects road potholes based on stereo vision. The accuracy is satisfactorily good. The time for computation is: secs.



4.2 Future work

1). We would like to extend this setup to generate a real time pothole detection system using the Oak-D camera which would also require to further improve the time efficiency of the code.

5.Useful resources and links:

1. https://docs.luxonis.com/projects/api/en/latest/samples/StereoDepth/depth_preview/ : Depth Generation from Oak-D camera
2. https://www.geeksforgeeks.org/3d-surface-plotting-in-python-using-matplotlib/ : 3 dimensional surface plotting
3. https://www.geeksforgeeks.org/3d-scatter-plotting-in-python-using-matplotlib/#:~:text=A%203D%20Scatter%20Plot%20is,to%20enable%20three%20dimensional%20plotting. : Scatter plot
4. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html : Implementation of least square regression to chart the Surface fit of the road
5. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html : image processing operations
6. https://www.geeksforgeeks.org/python-opencv-cv2-rectangle-method/ : enclosing the detected pothole in a box