

**A
PROJECT
ON
ROAD MARKER LABELLING TOOL**

By:

Shivam (60176802714/CSE-3/2014)



**Department of Computer Science
Guru Tegh Bahadur Institute of Technology**

**Guru Gobind Singh Indraprastha University
Dwarka, New Delhi
Year 2016-2017**

Swaayatt Robots Pvt. Ltd.

42-Abhinav Real Homes, Awadhouri, Piplani, Bhopal, MP 462021

Letter of Completion of Internship

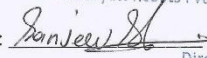
Date: 06-08-2016

To whomever it may concern,

This letter is to inform that Shivam Sardana worked as an intern at Swaayatt Robots in computer vision and artificial intelligence, from June 14 to August 5, 2016. Shivam worked on many topics during his intern, among which the main topics were: (i) processing the point cloud representation of the robot's surroundings, for robot perception; and (ii) working on an algorithmic architecture for automatically labeling the road markers for a "road marker" data-set preparation. During his intern, Shivam had to read fairly mathematical papers in several areas in computer vision, for example, visual odometry, visual inertial navigation, and several papers describing interest point detectors and descriptors.

Let me know if you have further questions.

Sanjeev Sharma
Founder and Director
Swaayatt Robots Pvt. Ltd.
Phone: 0755 2980307

For : Swaayatt Robots Pvt. Ltd.
Signature: 
Director

Sanjeev Sharma, Founder and Director, Swaayatt Robots
Email: sanjeevs@swaayatt-robots.com, sanjeev.sharma.iitr@gmail.com
Phone: 8989715787

AKNOWLEDGEMENT

I express my deep sense of gratitude and obligation to **Prof. Munshi Yadav** who provided me with his expert guidance, support and suggestions about my project. Without his help, I wouldn't have been able to present this piece of work to the present standard. I also take this opportunity to give thanks to **Mr. Sanjeev Sharma** who gave me support for the project and other aspects of my study at **SWAAYATT ROBOTS, BHOPAL** allowing me to perform to the best of my abilities.

Date

Shivam

(60176802714 /CSE-3/ 2014)

shivamsardanaavi@gmail.com

ABSTRACT

The project namely “**ROAD MARKER LABELLING TOOL**” is a tool implemented in python. The main objective of this tool is to create labeled data set to train the model for self-driving car. The tool has been improved by research techniques developed and replace the existing manual system to mark images. The project has a very user friendly interface. It also provides the user with quick response and very accurate marking of labels. This help autonomous cars to train them to identify markers on road. It is a research problem and it will be further developed.

The entire database has been collected by autonomous car of Swaayatt Robots. All the major disadvantages of the other old computer vision techniques is that they are slow and produce semi-automatic labels on images and my project has overcome it and provide automatic labelling of images.

CONTENTS

<u>CHAPTER</u>	<u>PAGE NO.</u>
Title Page	i
Certificate	ii
Acknowledgement	iii
Abstract	iv
1. List of figures	1
2. Introduction	3
3. Requirement Analysis (SRS)	4
4. System Design	9
1. Client Server Architecture	10
2. Data Flow Diagram	11
5. Results	15
6. Summary and Conclusions	16
7. References	18
8. Appendixes	20

CHAPTER 1
LIST OF FIGURES

LIST OF FIGURES

Fig no.	Figure Name	Page no
1.1	Input Image	21
1.2	Bilaterally Filtered Image	21
2.1	Sample Input Image	22
2.2	Harris Corner Image	22
2.3	Watershed Image	23
2.4	bitwise_and Image	23
2.5	Final Output	24

CHAPTER 2

INTRODUCTION

INTRODUCTION

This project introduces the **ROAD MARKER LABELLING TOOL** which is a python based program. It explains the automatic labelling of road markers on roads for self-driving car. There is an input database of images of Indian roads obtained from autonomous car and there is output labeled data set to train the model. The database contains the images of Indian roads with markers. This project reduces the time in the manual labelling of images by humans to train the model.

The project has total two sections. One for filtering and one for segmentation technique. The project is very simple and useful which allows the company to easily label their road marker data set. In Bilateral Filter.py, images are filtered out in which edges are preserved and noise is removed. In Road Marker Tool.py labelling is done.

The project is based on the core concepts of **PYTHON** programming language and **OpenCV** library. Feature of Python that are included are encapsulation, abstraction, inheritance, and reusability of the code is also done properly. The source code also includes general programming features like if-else and various string methods. The simplicity of this program is using Python scripting in which you just enter the address of dataset and just run the script and it will label images. It is a research project and the methods used in this are currently a research problem for many robotics and car companies working on self-driving car and artificial intelligence.

CHAPTER 3
REQUIREMENT ANALYSIS
SOFTWARE REQUIREMENT SPECIFICATION

SOFTWARE REQUIREMENT SPECIFICATION (SRS)

Introduction About SRS

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements, we need to have clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

Goals of SRS

The Software Requirements Specification (SRS) is a communication tool between stakeholders and software designers. The specific goals of the SRS are:

- Facilitating reviews
- Describing the scope of work
- Providing a reference to software designers (i.e. navigation aids, document structure)
- Providing a framework for testing primary and secondary use cases
- Including features to customer requirements
- Providing a platform for ongoing refinement (via incomplete specs or questions).

SRS ABOUT HOTEL MANAGEMENT SYSTEM

The Software Requirement Specification will provide a detailed description of the requirements of the hotel management system. This SRS will allow for the complete understanding of what is to be expected from the newly introduced system which is to be constructed. The clear understanding of the system and its functionality will allow for the correct software to be developed for the end user and will be used for the development of the future stages of the project.

Purpose

This specification document describes the capabilities that will be provided by the software program “**ROAD MARKER LABELLING TOOL**”. It also takes the various required constraints by which the system will abide. The intended audience for this document are the development team, testing team and end users of the product.

Scope

The software product “**ROAD MARKER LABELLING TOOL**” is a Python based application used for automatic labelling of road markers for training of autonomous car of robotics company, “**SWAAYATT ROBOTS**”. The tool manages the input image data set. The user is also allowed to change kernel size and other mathematical figures. The application provides the user with quick response and very accurate results.

Functional Requirements

Functional requirements specify the business requirements of the project in detail. Usually business requirements are specified in terms of the actions that user performs on the software tool. This is known as the use case model. Functional requirements

should contain a combination of use cases and plain textual description of system features. System features are specified at a higher level and use cases attempt to translate into user actions.

Non-Functional Requirements

Non-functional or technical requirements specify how the software system should operate. In contrast, functional requirements specify what a software system should do. Some of the non-functional requirements are derived from the functional requirements. Non-functional requirements captured include performance requirements, application scalability, application security, maintainability, usability, availability, logging and auditing, data migration requirements, multi lingual support etc.

Functional Requirements for Road Marker Labelling Tool

1. Input Images

The user has to enter the address and name of image.

2. Kernel Size in filtering

After entering the images, user has to enter kernel size for processing of images

3. Selection of the mathematical inputs

The user has to select mathematical figures for index, sobel operator size for processing.

Non-Functional Requirements for Road Marker Labelling Tool

4. Reliability

Accurate figures must be given for highly reliable mathematical figures in it.

5. Portability

The application is built using the concepts of core python. The application will run on any OS having Python or Ipython notebook in it.

CHAPTER 4

SYSTEM DESIGN

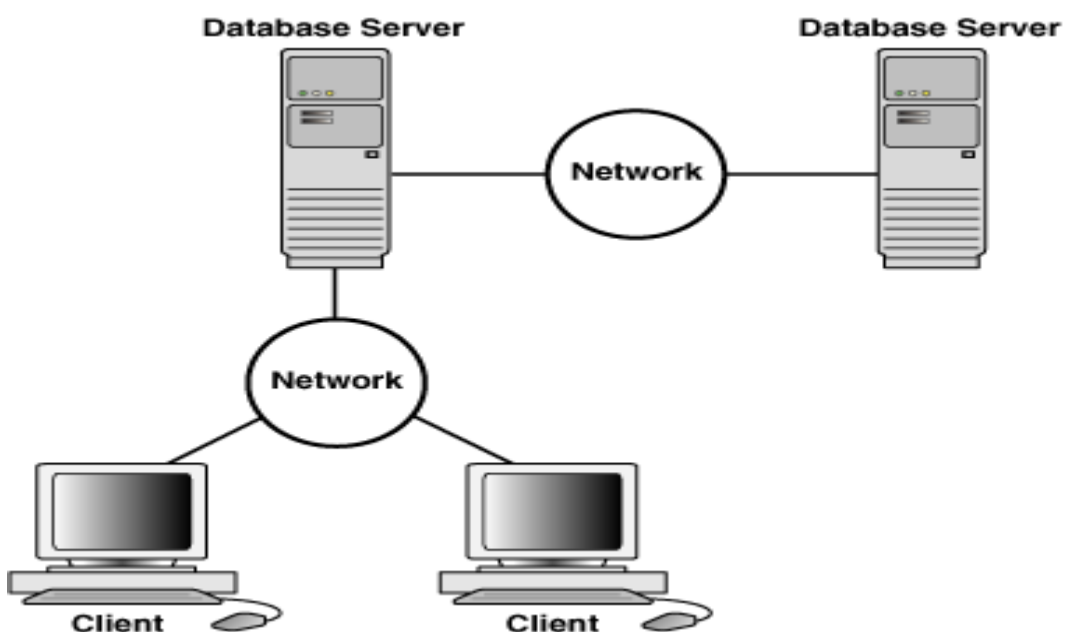
CLIENT SERVER ARCHITETURE

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



DATA FLOW DIAGRAM

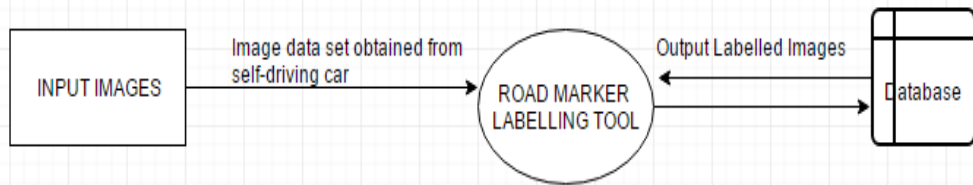
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top-down approach to Systems Design. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

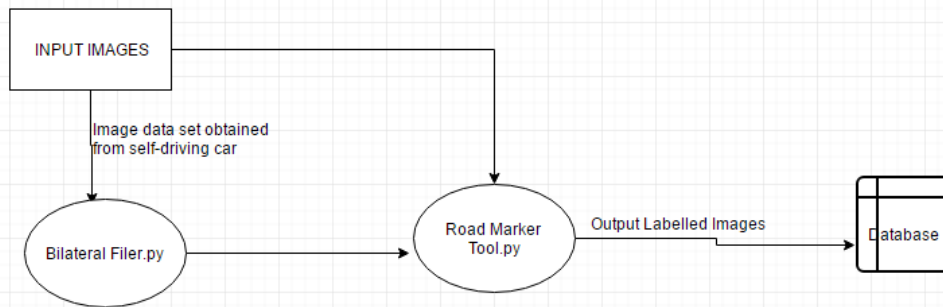
Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram model.

LEVEL-0 DFD



Level-0 DFD

LEVEL-1 DFD



Level 1 DFD

CHAPTER 5

RESULT

RESULTS

A very necessary Python based application for the robotics company “**SWAAYATT ROBOTS**” is generated for self-driving car. The project is a comprehensive software suite consisting of integrated modules for various aspects of computer vision. The software includes all the features required in a labelling tool, object recognition, and segmentation techniques. This designed for labelling various types of data set. It emphasizes the highest-level research activities through the comprehensive features, mathematical inputs. The software is fast. The software functionality is based on the core concepts of python programming language. This project reduces the time in the manual labelling of images by humans to train the model. It is a research project and the methods used in this are currently a research problem for many robotics and car companies working on self-driving car and artificial intelligence.

CHAPTER 6
SUMMARY AND CONCLUSION

SUMMARY AND CONCLUSION

It has been immense pleasure, honor and challenge to have this opportunity to take up this research project and complete it successfully. While developing this project, I have learnt a lot about computer vision, image processing and machine learning, I have also learnt it to make user friendly and error free. I have also learnt how to make the program robust and how to give a proper design to the system. During the development process, I studied carefully and understood the software should be more demanding, I also realized the importance of maintaining a minimal margin for error. As the project is purely based on the concepts of python programming, mathematics, artificial intelligence, computer vision so it also helped me to clear my concepts of Python and also helped me in building up of my logic and increased my research interest in Artificial Intelligence. The project shows the full functionality and how the famous research works and hard implementation can be used for marking markers for self-driving car.

CHAPTER 7
REFERENCES

REFERENCES

1. <http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=morphologyex>
2. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm>
3. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm>
4. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm>
5. http://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html#gsc.tab=0
6. C. Harris and M. Stephens (1988). "A combined corner and edge detector" (PDF). *Proceedings of the 4th Alvey Vision Conference*. pp. 147–151
7. L. Najman and M. Schmitt. Geodesic saliency of watershed contours and hierarchical segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, Num. 12 (1996), pages 1163–1173.
8. Jianbo Shi and Jitendra Malik (1997): "Normalized Cuts and Image Segmentation", IEEE Conference on Computer Vision and Pattern Recognition, pp 731–737
9. J. Winn, N. Jojic. Locus: Learning object classes with unsupervised segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, Beijing, 2005.
10. http://docs.opencv.org/3.1.0/d3/db4/tutorial_py_watershed.html#gsc.tab=0

APPENDIX A

SCREENSHOTS



Figure 1.1

This is input sample to Bilateral Filter.py. This image will be input from data set obtained from self-driving car and it will be filtered from bilateral filter which will preserve edges and remove noise from image.

Figure 1.2



This is output image from Bilateral Filter.py. This image is filtered and as edge are preserved and noise is removed. This is sample where kernel size(k) = 19.



Figure 2.1

This is same input sample to Road Marker Tool.py which was input to Bilateral Filter.py. In this features will be detected using Harris Corner Detection Technique.

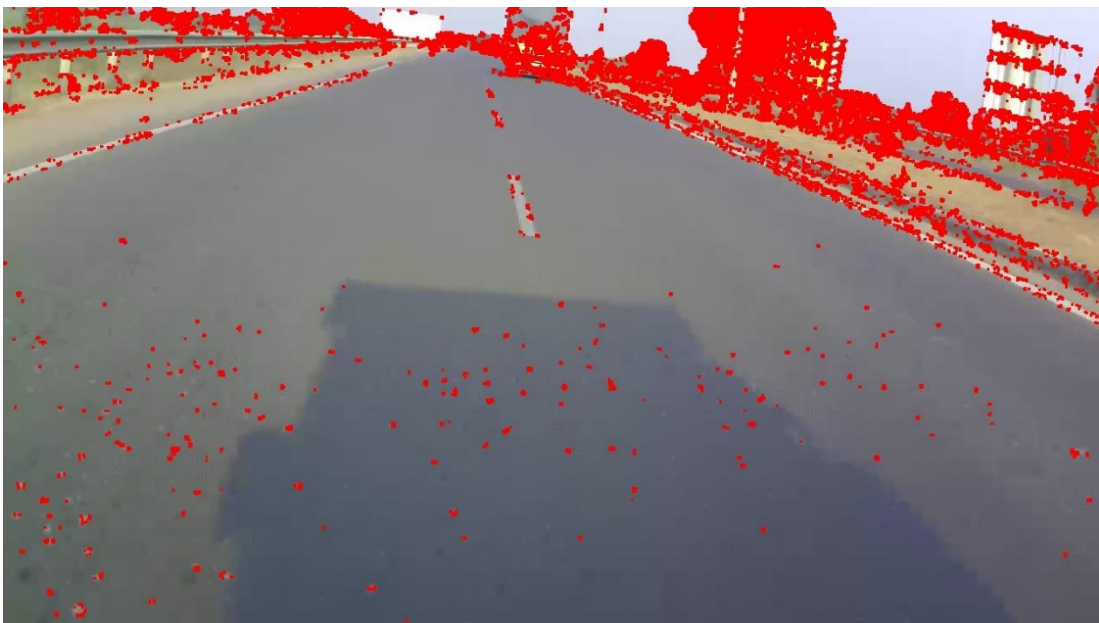


Figure 2.2

This is output image of Harris corner function which is named as `cv2.cornerHarris`. The red dots represent feature points as in figure there are many feature points which needed to be filtered out. This output will act as input to watershed function.



Figure 2.3

This is output image from `cv2.watershed()` in which as seen in screenshot is segmented by red borders. Now task is to extract the largest segment.

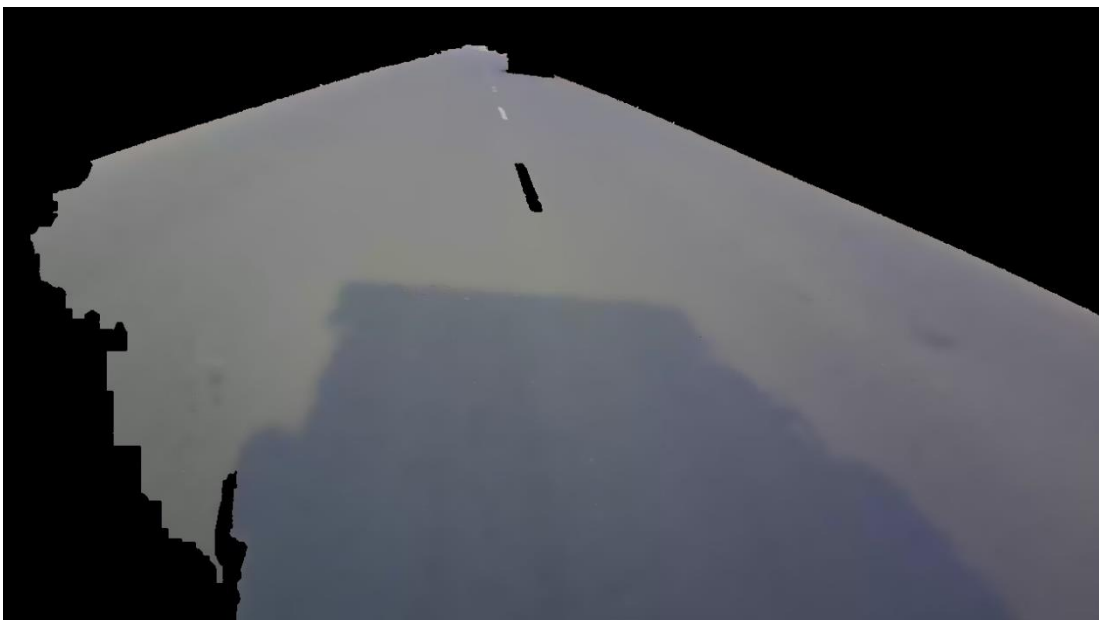


Figure 2.4

This is output of `cv2.bitwise_and()`. Through in this input is Figure 2.1. In this largest segment of road is extracted.



Figure 2.5

This is final output of tool. This is still a research problem. As seen in screenshot on first marker strip there are yellow dots which are features which to be detect. As seen there are few features but there are 100's of features there. This will form labeled dataset to feed to CNN for training of model.

APPENDIX B

SOURCE CODE

Bilateral Filter.py

```
import cv2
import numpy as np

DELAY_CAPTION = 1500;
DELAY_BLUR = 500;

img = cv2.imread('ts4.jpg')
i=19;
# Remember, bilateral is a bit slow, so as value go higher, it takes long time
bilateral_blur = cv2.bilateralFilter(img,i, i*2,i/2)
string = "
cv2.putText(bilateral_blur,string,(20,20),cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(255,255,255))
cv2.imshow('Blur',bilateral_blur)
cv2.waitKey(DELAY_BLUR)

cv2.waitKey(0)
cv2.imwrite('blur4(k=19).jpg',bilateral_blur)
```

Road Marker Tool.py

```
import cv2
import numpy as np
#Harris

filename = 'ts4.jpg'
img1 = cv2.imread(filename)          #img1 is input image after reading

gray = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY) #gray is rgb2gray conversion
                                           #of img1

gray = np.float32(gray)
print(gray.shape)

dst = cv2.cornerHarris(gray, 2, 3, 0.04)
#2 is Neighborhood size ,3 is 3x3 sobel operator 0.04 threshold for Harris detector
#free parameter

#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)
a=np.array(dst)
x,y=a.shape
print(x,y)

c=dst>0.00001*dst.max() #threshold to consider harris points use in gray threshold

# Threshold for an optimal value, it may vary depending on the image.

img1[dst>0.00001*dst.max()]=[0,0,255]  #provides red color to harris corner

cv2.imshow('dst',img1)

cv2.imwrite('harris1.jpg',img1)  #saving image

cv2.waitKey(0)

q=np.unravel_index(dst.argmax(),dst.shape)
print(q)

#Harris completes here
```

```

img = cv2.imread('blur4(k=19).jpg')          #bilateral image from bilateral filter part

gray2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #gray2 is rgb2gray of img

ret, thresh =
cv2.threshold(gray2,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    #threshold considering factors threshold binary inversion + otsu binarization

# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.9*dist_transform.max(),255,0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers+1

# Now, mark the region of unknown with zero
markers[unknown==255] = 0
markers = cv2.watershed(img,markers)

print "Number of segments:", np.max(markers) #printing number of markers

a=np.array(markers)
print(a.shape)

img[markers == -1] = [0,0,255]    #giving background boundary red color
cv2.imshow('output',img)
cv2.imwrite('wh1(k=19).jpg',img)
print(markers)
cv2.waitKey(0)
# watershed on whole image completes here

#NOW firstly extracting foreground area

```

```

m = cv2.convertScaleAbs(markers)      #On each element of the input array, the
function convertScaleAbs performs three
print(m)                             # operations sequentially: scaling, taking an
absolute value, conversion to an unsigned 8-bit type:
print(m.shape)
print(m)
    #using otsu binarization + threshold binary inverse
ret,thresh = cv2.threshold(m,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
res = cv2.bitwise_and(img,img,mask = thresh)  #this is for retrieval of foreground
area from not sure background area

cv2.imshow('bitwise',res)
cv2.imwrite('bitwise(k=19).jpg',res)      #extracted foreground image ie road
part

cv2.waitKey(0)
print(res)

x,y,z=res.shape
print(x)
print(y)
print(z)

gray1 = cv2.cvtColor(res,cv2.COLOR_BGR2GRAY)      #converting it into gray

gray1 = np.float32(gray1)

m,n=gray1.shape
print(m)
print(n)

for i in range(1,m):
    for j in range(1,n):
        if gray1[i,j]!=0:
            if(j>=582 and j<=654 and i>=171 and i<=255):
                gray1[i,j]=gray1[i,j]*c[i,j]      #getting harris (c variable see
above) getting harris corner in that bitwise segmented road part
            else:
                gray1[i,j]=0

filename1 = 'ts4.jpg'                      #original image
img13 = cv2.imread(filename1)              # img13 variable

img13[gray1>0]=[0,255,255]                 #condition to show harris on
conditioned part
cv2.imshow('dst',img13)
print(gray1)

```

```

cv2.waitKey(0)
b=0
abc=cv2.imread(filename1)
harris_points=[]
for i in range(1,m):
    for j in range(1,n):
        #getting harris corner and this if
        condition is we have consider 4 points near
        if gray1[i,j]!=0:
            # white markers these are those co
            ordinates
            if(j>=582 and j<=654 and i>=171 and i<=255):
                z=(i,j)
                harris_points.append(z)
                b=b+1

harris_pointS=np.array(harris_points)
print(b)
print(harris_points)
abcd=[]
for i in range(b):
    #now again playing watershed along
    each harris point by considering 32x32
    s,d=harris_points[i]
    #window along each point , applying
    same watershed that is applied above. Here watershed is applied on extracted part
    above.
    xc=abc[s-16:s+16,d-16:d+16]
    grayxc = cv2.cvtColor(xc,cv2.COLOR_BGR2GRAY)
    ret, threshxc =
cv2.threshold(grayxc,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    # noise removal
    kernelxc = np.ones((3,3),np.uint8)
    openingxc = cv2.morphologyEx(threshxc,cv2.MORPH_OPEN,kernel, iterations =
2)

    # sure background area
    sure_bgxc = cv2.dilate(openingxc,kernelxc,iterations=3)

    # Finding sure foreground area
    dist_transformxc = cv2.distanceTransform(openingxc,cv2.DIST_L2,5)
    ret, sure_fgxc =
cv2.threshold(dist_transformxc,0.9*dist_transformxc.max(),255,0)

    # Finding unknown region
    sure_fgxc = np.uint8(sure_fgxc)
    unknown = cv2.subtract(sure_bgxc,sure_fgxc)
    # Marker labelling
    ret, markersxc = cv2.connectedComponents(sure_fgxc)

    # Add one to all labels so that sure background is not 0, but 1
    markersxc = markersxc+1

```

```

# Now, mark the region of unknown with zero
markersxc[unknown==255] = 0
markersxc = cv2.watershed(xc,markersxc)

abc[s-16:s+16,d-16:d+16]=xc

nSegs = np.max(markersxc)
print(nSegs)

g=(0,0)
#this type for mulitplication
h=(1,1)

if(nSegs!=0 and nSegs<=3):          #considering condition that no of
segments must not be zero and one and must be less than 3
    if(nSegs%3==0):                # for no. of segments to be 3

        axc=xc[markersxc==1]
        bxc=xc[markersxc==2]
        cxc=xc[markersxc==3]

        if((axc>=91).all() and (axc<=230).all()):          #condition for pixel for
all in that segment
            if((bxc>=90).all() and (bxc<=158).all()):
                if((cxc>=90).all() and (bxc<=158).all()):
                    abcd.append(h)

            else:
                abcd.append(g)
        else:
            abcd.append(g)
    elif((axc>=90).all() and (axc<=158).all()):
        if((bxc>=90).all() and (bxc<=158).all()):
            if((cxc>=91).all() and (cxc<=255).all()):
                abcd.append(h)
            else:
                abcd.append(g)
        elif((bxc>=91).all() and (bxc<=230).all()):
            if((cxc>=90).all() and (cxc<=158).all()):
                abcd.append(h)
            else:
                abcd.append(g)
        else:
            abcd.append(g)
    else:
        abcd.append(g)

```

```

else:
    abcd.append(g)

elif(nSegs%3==2):                                     #condition for no. of segments to be 2

    axc=xc[markersxc==1]
    bxc=xc[markersxc==2]
    print(bxc)
    if((axc>=91).all() and (axc<=255).all()):
        if((bxc>=90).all() and (bxc<=158).all()):
            abcd.append(h)
        else:
            abcd.append(g)

    elif((axc>=90).all() and (axc<=158).all()):
        if((bxc>=91).all() and (bxc<=255).all()):
            abcd.append(h)
        else:
            abcd.append(g)

    else:
        abcd.append(g)

else:
    abcd.append(g)
else:
    abcd.append(g)

cvb=np.array(abcd)                                     #numpy array
cv2.waitKey(0)

print(abcd)

vbn=harris_pointS*cvb                                #multiplying both array as cvb will give result in 0,1
whether it is part of marker or not

print(vbn)

m,n=vbn.shape                                         #shape of vbn

print(m)
print(n)

```

```

for e in range (0,m-1):                                # here m is size(img,1) as we have our numpy
array of size mx2
    if(vbn[e,0]==0 and vbn[e,1]==0):v                    #checking condition here and getting
co-ordinates of harris gray1 image
        c=harris_pointS[e,0];
        d=harris_pointS[e,1];
        gray1[c,d]=0;
        print("true")

    else:
        c=harris_pointS[e,0];
        d=harris_pointS[e,1];
        gray1[c,d]=gray1[c,d]
        print("false")

print(gray1)

img13a=cv2.imread(filename1)                            #again reading filename so not have confusion again by
searching

img13a[gray1>0]=[0,255,255]                             #condition for showing harris on real image yellow color
as opencv works on bgr vale
cv2.imshow('fnally',img13a)
cv2.imwrite('wedfeedfsddffrgrgdcdce1.jpg',img13a) #saving our final image
cv2.waitKey(0)

```