COMPUTER VISION APPLICATIONS

A SUMMER TRAINING REPORT

Submitted by

CHIRAG SHARMA
Enrollment Number: 00414807320
ELECTRONICS AND COMMUNICATION ENGENEERING

Under the supervision of

Prof.Nitin Sharma
MAIT(GGSIPU,New Delhi)

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY
ROHINI, NEW DELHI

Maharaja Agrasen Institute of Technology

To Whom It May Concern

I, **CHIRAG SHARMA**, Enrollment No. **00414807320**, a student of **Bachelor of technology (ECE), a class of 2019-2023, Maharaja Agrasen Institute of technology, Delhi** hereby declare that the summer training project entitled
"**Computer Vision Applications"** is an original work and the same has not been submitted to any other institute for the award of any other degree.

Date:
Place:

CHIRAG SHARMA
Enrollment No: 00414807320
Electronics and Communication Engineering
E123(E1)

Vision is really hard

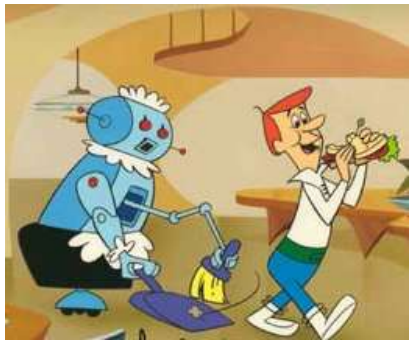
- Vision is an amazing feat of natural intelligence
  - Visual cortex occupies about 50% of Macaque brain
  - More human brain devoted to vision than anything else

Why computer vision matters

Safety

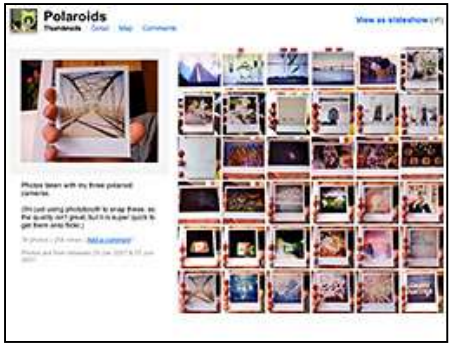
Health

Security



Comfort

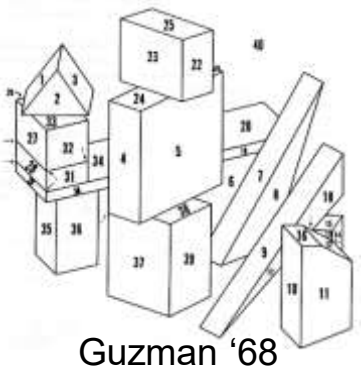
Fun



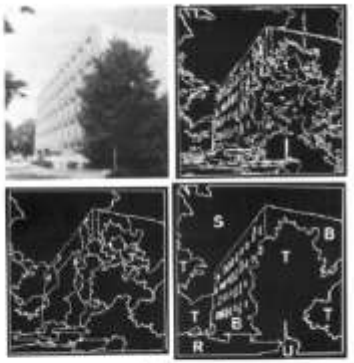
Access

Ridiculously brief history of computer vision

- 1966: Minsky assigns computer vision as an undergrad summer project
- 1960's: interpretation of synthetic worlds
- 1970's: some progress on interpreting selected images
- 1980's: ANNs come and go; shift toward geometry and increased mathematical rigor
- 1990's: face recognition; statistical analysis in vogue
- 2000's: broader recognition; large annotated datasets available; video processing starts

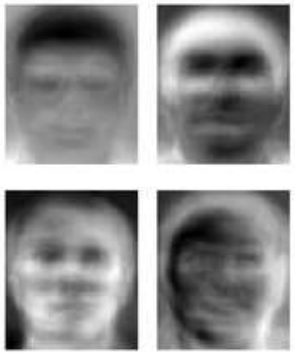


Guzman '68



Ohta Kanade '78

- Examples of state-of-the-art- Here are some examples which I have observed closely.
- We can clearly see the bifurcation of computer vision here.
- Computer vision is used in industries ranging from energy to automotive to information technology services.
- Vision employs different levels of Images

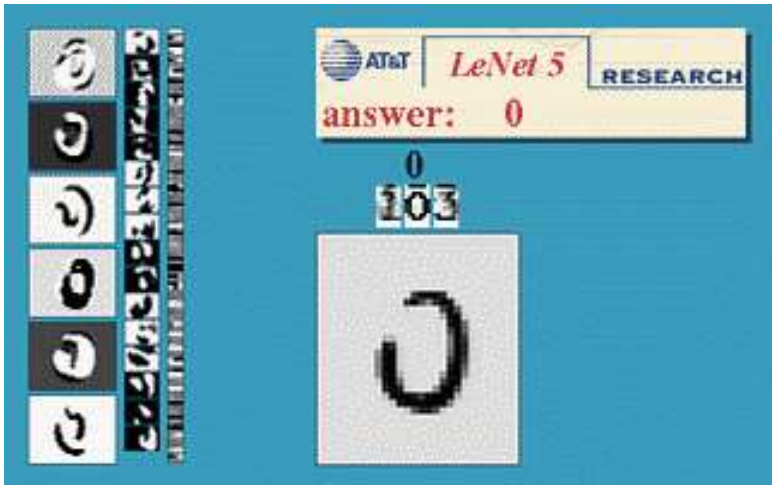


Turk and Pentland '91

Optical character recognition (OCR)

Technology to convert scanned docs to text

- If you have a scanner, it probably came with OCR software



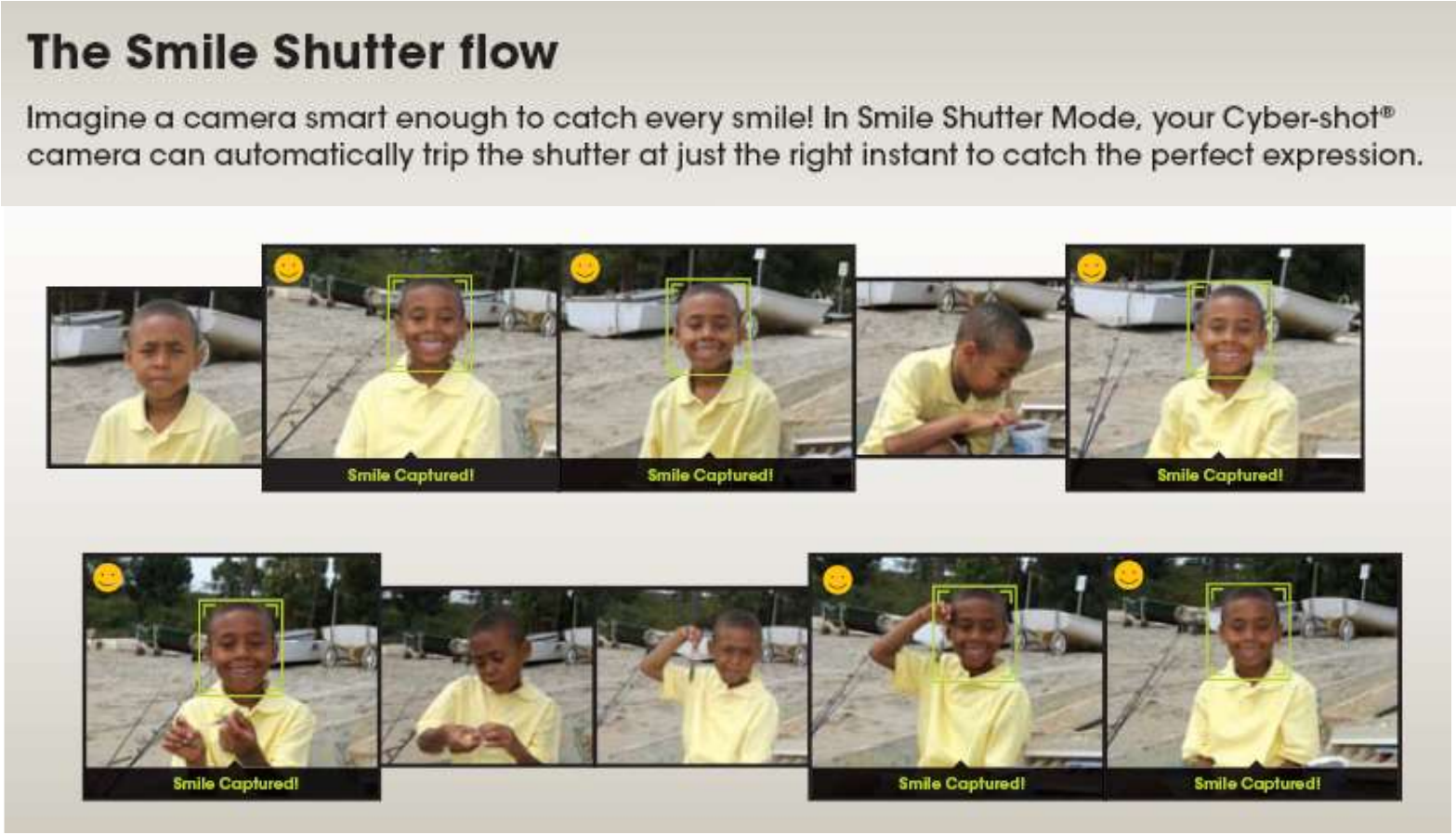
Digit recognition, AT&T labs
http://www.research.att.com/~yann/

License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

Face detection

- Many new digital cameras now detect faces
  - Canon, Sony, Fuji, …

Smile detection



Sony Cyber-shot® T70 Digital Still Camera

Object recognition (in supermarkets)

LaneHawk by EvolutionRobotics

“A smart camera is flush-mounted in the checkout lane, continuously watching for items. When an item is detected and recognized, the cashier verifies the quantity of items that were found under the basket, and continues to close the transaction. The item can remain under the basket, and with LaneHawk,you are assured to get paid for it… “

Login without a password…

Fingerprint scanners on many new laptops, other devices

Face recognition systems now beginning to appear more widely
http://www.sensiblevision.com/

Object recognition (in mobile phones)

COLOUR CONVERSION PHASE

There are different types of color models.

The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system.

```
void main()
 { string path = "Resources/test.png";
 Mat img = imread(path);
Mat imgGray;
Mat  imgBlur, imgCanny, imgDia, imgerode ;
cvtColor(img, imgGray, COLOR_BGR2GRAY); //converts input image color to gray.
```

- This code here converts image of blue, green
- And red to the gray colour.
- The command cvtColor works here. It first
- Takes the image (img) as destination and the
- Second parameter as (imgGray) which stores
- The converted image and last parameter does
- The work of changing the color of the image.

A)VIRTUAL PAINTER

Steps that I implemented :-

A)Importing libraries, assigning BGR values for color.

B)Creating a window that provides various options.

C)Capturing a video using a webcam and grabbing frames.

While Using different morphinig techniques.

D)Displaying output

bindex = 0,gindex = 0,rindex = 0,yindex = 0

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]

colorIndex = 0

- Step-1 – Importing required libraries
- import numpy as np
- import cv2
- from collections import deque
- Using the below code snippet, we will declare the BGR values of the red color, which detects the red color and helps us to draw.
- Lower_red = np.array([161, 155, 84])
- Upper_red = np.array([179, 255, 255])
- Here i used red colour lower and upper ranges, you can try with your own color by setting BGR values
- we will use a deque function, which is used to store separate colors that helps in creating buttons on the window.
- bpoints = [deque(maxlen=512)]
- gpoints = [deque(maxlen=512)]
- rpoints = [deque(maxlen=512)]
- ypoints = [deque(maxlen=512)]

A)VIRTUAL PAINTER

Step-2 – Creating a window, that provides various options.

Using the below code snippet, we will create a window that displays the different color options to draw and also it creates the buttons using cv2.putText.

Window = np.zeros((471,636,3)) + 255

Window = cv2.rectangle(paintWindow, (40,1), (140,65), (0,0,0), 2)

Window = cv2.rectangle(paintWindow, (160,1), (255,65), colors[0], -1)

Window = cv2.rectangle(paintWindow, (275,1), (370,65), colors[1], -1)

Window = cv2.rectangle(paintWindow, (390,1), (485,65), colors[2], -1)

Window = cv2.rectangle(paintWindow, (505,1), (600,65), colors[3], -1)

- cv2.putText(Window, "CLEAR ALL", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
- cv2.putText(Window, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
- cv2.putText(Window, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
- cv2.putText(Window, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
- cv2.putText(Window, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (150,150,150), 2, cv2.LINE_AA)
- cv2.namedWindow('Paint', cv2.WINDOW_AUTOSIZE)
- As you can see in the output we created a normal window now we need to set the webcam and place this template on the webcam video.

A)VIRTUAL PAINTER

Step-3 – Capturing a video using a webcam and grabbing frames.

Here we are capturing the video, using a webcam so we set videoCapture to zero.

camera = cv2.VideoCapture(0)

Using the below code snippet, we will run a loop to grab the frames from the webcam and to display the dynamic results.

while True:

(grabbed, frame) = camera.read()

frame = cv2.flip(frame, 1)

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

if not grabbed:

Break

- Step-4 – Using different morphing techniques
- Using the below code snippet, we will preprocess using the input(the text drawn on-air) using morphing techniques erode, morphologyEx, dilate.
- kernel = np.ones((5, 5), np.uint8)
- Mask = cv2.inRange(hsv, Lower_red, Upper_red)
- Mask = cv2.erode(Mask, kernel, iterations=2)
- Mask = cv2.morphologyEx(Mask, cv2.MORPH_OPEN, kernel)
- Mask = cv2.dilate(Mask, kernel, iterations=1)
- Using the below code snippets, we will use counters, it holds the data in an unordered collection it allows you to count the items in the iterable list.
- (cnts, _) = cv2.findContours(blueMask.copy(), cv2.RETR_EXTERNAL,
- cv2.CHAIN_APPROX_SIMPLE)
- center = None

A) VIRTUAL PAINTER

Using the below code snippet, when it detects the red color and around that color, I try to create a bounding circle.

```
if len(cnts) > 0:
    cnt = sorted(cnts, key = cv2.contourArea, reverse = True)[0]
    ((x, y), radius) = cv2.minEnclosingCircle(cnt)
    cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
    M = cv2.moments(cnt)
    center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00'])
```

- In the below code snippet, if counters are stored then it runs the loop. using the deque function.

```
- if center[1] <= 65:
-     if 40 <= center[0] <= 140: # Clear All
-         bpoints = [deque(maxlen=512)]
-         gpoints = [deque(maxlen=512)]
-         rpoints = [deque(maxlen=512)]
-         ypoints = [deque(maxlen=512)]
-         bindex = 0, gindex = 0, rindex = 0,yindex = 0

- paintWindow[67:,:,:] = 255
-     elif 160 <= center[0] <= 255:
-         colorIndex = 0
-     elif 275 <= center[0] <= 370:
-         colorIndex = 1
-     elif 390 <= center[0] <= 485:
-         colorIndex = 2
-     elif 505 <= center[0] <= 600:
-         colorIndex = 3
-
```

A)VIRTUAL PAINTER

```
else :
        if colorIndex == 0:
            bpoints[bindex].appendleft(center)
        elif colorIndex == 1:
            gpoints[gindex].appendleft(center)
        elif colorIndex == 2:
            rpoints[rindex].appendleft(center)
        elif colorIndex == 3:
            ypoints[yindex].appendleft(center)
```

- Step-5 – Displaying output
- In the below code snippet, we are implementing the draw option by taking our designed window and applying it to the webcam video.

```
points = [bpoints, gpoints, rpoints, ypoints]
    for i in range(len(points)):
        for j in range(len(points[i])):
            for k in range(1, len(points[i][j])):
                if points[i][j][k - 1] is None or points[i][j][k] is None:
                    continue
                cv2.line(frame, points[i][j][k - 1], points[i][j][k], colors[i], 2)
                cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)
```

- In the below code snippet, we are displaying the output video

```
cv2.imshow("Web_cam_video", frame)
cv2.imshow("Paint", paintWindow)
```

A) VIRTUAL PAINTER

In the below code snippet, we are displaying the output video

cv2.imshow("Web_cam_video", frame)

cv2.imshow("Paint", paintWindow)

Conclusion

In the above implementation, we demonstrated how to create a virtual paint application using OpenCV tool.

We had given lower and upper ranges of red color. so we can use red color as a pencil to draw virtually and it reflects on your system screen.

Although my drawings are bad, it's fun to implement.

- **B) Automatic License/Number Plate Recognition (ANPR) with Python**
- Automatic License/Number Plate Recognition (ANPR/ALPR) is a process involving the following steps:
- **Step #1:** Detect and localize a license plate in an input image/frame
- **Step #2:** Extract the characters from the license plate
- **Step #3:** Apply some form of Optical Character Recognition (OCR) to recognize the extracted characters

- **ANPR tends to be an *extremely* challenging subfield of computer vision, due to the vast diversity and assortment of license plate types across states and countries.**
- License plate recognition systems are further complicated by:
- Dynamic lighting conditions including reflections, shadows, and blurring
- Fast-moving vehicles
- Obstructions

License/Number Plate Recognition (ANPR) with Python

Additionally, large and robust ANPR datasets for training/testing are difficult to obtain due to:

These datasets containing sensitive, personal information, including time and location of a vehicle and its driver

ANPR companies and government entities closely guarding these datasets as proprietary information

Therefore, the first part of an ANPR project is usually to collect data and amass enough example plates under various conditions.

So let's assume I don't have a license plate dataset (quality datasets are hard to come by). That rules out deep learning object detection, which means I am going to have to exercise our traditional computer vision knowledge.

- I agree that it would be nice if we had a trained object detection model, but today I want *you* to rise to the occasion.
- Before long, I'll be able to ditch the training wheels and consider working for a toll technology company, red-light camera integrator, speed ticketing system, or parking garage ticketing firm in which we need 99.97% accuracy.
- Given these limitations, I'll be building a *basic* ANPR system that anyone can use as a *starting point*

License/Number Plate Recognition (ANPR) with Python

We're ready to start implementing our Automatic License Plate Recognition script.

I'll keep the code neat and organized using a Python class appropriately named

PyImageSearchANPR -

. This class provides a reusable means for license plate localization and character OCR operations.

Open

anpr.py

and let's get to work reviewing the script:

- # import the necessary packages
- from skimage.segmentation import clear_border
- import pytesseract
- import numpy as np
- import imutils
- import cv2
- class PyImageSearchANPR:
- def __init__(self, minAR=4, maxAR=5, debug=False):
- # store the minimum and maximum rectangular aspect ratio
- # values along with whether or not we are in debug mode
- self.minAR = minAR
- self.maxAR = maxAR
- self.debug = debug

License/Number Plate Recognition (ANPR) with Python

Debugging our computer vision pipeline –

```
def debug_imshow(self, title, image, waitKey=False):
# check to see if we are in debug mode, and if so, show the
# image with the supplied title
if self.debug:
cv2.imshow(title, image)
# check to see if we should wait for a keypress
if waitKey:
cv2.waitKey(0)
```

- **Locating potential license plate candidates**
- Our first ANPR method helps us to find the license plate candidate contours in an image:

```
def locate_license_plate_candidates(self, gray, keep=5):
# perform a blackhat morphological operation that will allow
# us to reveal dark regions (i.e., text) on light backgrounds
# (i.e., the license plate itself)
rectKern = cv2.getStructuringElement(cv2.MORPH_RECT, (13, 5))
blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, rectKern)
self.debug_imshow("Blackhat", blackhat)
```

- Our
- locate_license_plate_candidates
- expects two parameters:
- gray
- : This function assumes that the driver script will provide a grayscale image containing a potential license plate.
- keep
- : We'll only return *up to this many* sorted license plate candidate contours.

License/Number Plate Recognition (ANPR) with Python

As you can see from above, the license plate characters are clearly visible!

In our next step, we'll find regions in the image that are light and *may contain* license plate characters:

1) next, find regions in the image that are light

squareKern = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

light = cv2.morphologyEx(gray, cv2.MORPH_CLOSE, squareKern)

light = cv2.threshold(light, 0, 255,

cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

self.debug_imshow("Light Regions", light)

- The Scharr gradient will detect edges in the image and emphasize the boundaries of the characters in the license plate:
- 1)compute the Scharr gradient representation of the blackhat
- 2)image in the x-direction and then scale the result back to
- 3) the range [0, 255]
- gradX = cv2.Sobel(blackhat, ddepth=cv2.CV_32F,
- dx=1, dy=0, ksize=-1)
- gradX = np.absolute(gradX)
- (minVal, maxVal) = (np.min(gradX), np.max(gradX))
- gradX = 255 * ((gradX - minVal) / (maxVal - minVal))
- gradX = gradX.astype("uint8")
- self.debug_imshow("Scharr", gradX)

License/Number Plate Recognition (ANPR) with Python

We can now smooth to group the regions that may contain boundaries to license plate characters:

blur the gradient representation, applying a closing

operation, and threshold the image using Otsu's method

gradX = cv2.GaussianBlur(gradX, (5, 5), 0)

gradX = cv2.morphologyEx(gradX, cv2.MORPH_CLOSE, rectKern)

thresh = cv2.threshold(gradX, 0, 255,

cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]

self.debug_imshow("Grad Thresh", thresh)

- these results look cluttered. The license plate region is somewhat defined, but there are many other large white regions as well. Let's see if we can eliminate some of the noise:
- # perform a series of erosions and dilations to clean up the
- # thresholded image
- thresh = cv2.erode(thresh, None, iterations=2)
- thresh = cv2.dilate(thresh, None, iterations=2)
- self.debug_imshow("Grad Erode/Dilate", thresh)
- # take the bitwise AND between the threshold result and the
- # light regions of the image
- thresh = cv2.bitwise_and(thresh, thresh, mask=light)
- thresh = cv2.dilate(thresh, None, iterations=2)
- thresh = cv2.erode(thresh, None, iterations=1)
- self.debug_imshow("Final", thresh, waitKey=True)

License/Number Plate Recognition (ANPR) with Python

I'd say it is the second or third largest contour in the image, and I also notice the plate contour is not touching the edge of the image.

Speaking of contours, let's find and sort them:

find contours in the thresholded image and sort them by

their size in descending order, keeping only the largest

ones

cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)

cnts = imutils.grab_contours(cnts)

cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:keep]

return the list of contours

return cnts

- In this next method, our goal is to find the most likely contour containing a license plate from our set of candidates. Let's see how it works:
- def locate_license_plate(self, gray, candidates,
- clearBorder=False):
- # initialize the license plate contour and ROI
- lpCnt = None
- roi = None
- # loop over the license plate candidate contours
- for c in candidates:
- # compute the bounding box of the contour and then use
- # the bounding box to derive the aspect ratio
- (x, y, w, h) = cv2.boundingRect(c)
- ar = w / float(h)

License/Number Plate Recognition (ANPR) with Python

Our

locate_license_plate

function accepts three parameters:

Gray: Our input grayscale image

candidates: The license plate contour candidates returned by the previous method in this class

clearBorder: A boolean indicating whether our pipeline should eliminate any contours that touch the edge of the image.

the aspect ratio is a relationship between the width and height of the rectangle.

- # check to see if the aspect ratio is rectangular
- if ar >= self.minAR and ar <= self.maxAR:
- # store the license plate contour and extract the
- # license plate from the grayscale image and then
- # threshold it
- lpCnt = c
- licensePlate = gray[y:y + h, x:x + w]
- roi = cv2.threshold(licensePlate, 0, 255,
- cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
- Let’s wrap up the locate_license_plate
- method so we can move onto the next phase:
- # check to see if we should clear any foreground
- # pixels touching the border of the image
- # (which typically, not but always, indicates noise)
- if clearBorder:

License/Number Plate Recognition (ANPR) with Python

License/Number Plate Recognition (ANPR) with Python-

roi = clear_border(roi)

display any debugging information and then break

from the loop early since we have found the license

plate region

self.debug_imshow("License Plate", licensePlate)

self.debug_imshow("ROI", roi, waitKey=True)

break

return a 2-tuple of the license plate ROI and the contour

associated with it

return (roi, lpCnt)

- roi = clear_border(roi)
- # display any debugging information and then break
- # from the loop early since we have found the license
- # plate region
- self.debug_imshow("License Plate", licensePlate)
- self.debug_imshow("ROI", roi, waitKey=True)
- break
- # return a 2-tuple of the license plate ROI and the contour
- # associated with it
- return (roi, lpCnt)
- **The central method of the PyImageSearchANPR**
- **class –**
- Our final method brings all the components together in one centralized place so our driver script can instantiate a PyImageSearchANPR
- object, and then make a single function call. Let's implement find_and_ocr
- :

License/Number Plate Recognition (ANPR) with Python

```
def find_and_ocr(self, image, psm=7, clearBorder=False):
# initialize the license plate text
lpText = None
# convert the input image to grayscale, locate all candidate
# license plate regions in the image, and then process the
# candidates, leaving us with the *actual* license plate
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
candidates = self.locate_license_plate_candidates(gray)
(lp, lpCnt) = self.locate_license_plate(gray, candidates,
clearBorder=clearBorder)
# only OCR the license plate if the license plate ROI is not
# empty
if lp is not None:
# OCR the license plate
options = self.build_tesseract_options(psm=psm)
lpText = pytesseract.image_to_string(lp, config=options)
self.debug_imshow("License Plate", lp)
# return a 2-tuple of the OCR'd license plate text along with
# the contour associated with the license plate region
return (lpText, lpCnt)
```

- This method accepts three parameters:
- image: The three-channel color image of the rear (or front) of a car with a license plate tag
- Psm: The **Tesseract Page Segmentation mode.**
- clearBorder: The flag indicating whether we'd like to clean up contours touching the border of the license plate ROI
- Given our function parameters, I now:

License/Number Plate Recognition (ANPR) with Python

Creating our license/number plate recognition driver script with OpenCV and Python

Now that our

PyImageSearchANPR

class is implemented, we can move on to creating a Python driver script that will:Load an input image from disk

Find the license plate in the input image

OCR the license plate

Display the ANPR result to our screen

Let's go in the project directory and find our driver file

ocr_license_plate.py:

- # import the necessary packages
- from pyimagesearch.anpr import PyImageSearchANPR
- from imutils import paths
- import argparse
- import imutils
- import cv2
- def cleanup_text(text):
- # strip out non-ASCII text so we can draw the text on the image
- # using OpenCV
- return "".join([c if ord(c) < 128 else "" for c in text]).strip()
- **script's command line arguments:-**
- # construct the argument parser and parse the arguments
- ap = argparse.ArgumentParser()
- ap.add_argument("-i", "--input", required=True,
- help="path to input directory of images")
- ap.add_argument("-c", "--clear-border", type=int, default=-1,

License/Number Plate Recognition (ANPR) with Python

help="whether or to clear border pixels before OCR'ing")

ap.add_argument("-p", "--psm", type=int, default=7,

help="default PSM mode for OCR'ing license plates")

ap.add_argument("-d", "--debug", type=int, default=-1,

help="whether or not to show additional visualizations")

args = vars(ap.parse_args())

Our **command line arguments** include:

--input

: The required path to the input directory of vehicle images.

- --clear-border
- : A flag indicating if we'll clean up the edges of our license plate ROI prior to passing it to Tesseract (further details are presented in the *"Pruning license plate candidates"* section above).
- --psm
- : Tesseract's Page Segmentation Mode; a 7
- indicates that Tesseract should only look for one line of text.
- --debug
- : A boolean indicating whether we wish to display intermediate image processing pipeline debugging images.
- **INITIALIZATION-**
- # initialize our ANPR class
- anpr = PyImageSearchANPR(debug=args["debug"] > 0)
- # grab all image paths in the input directory
- imagePaths = sorted(list(paths.list_images(args["input"])))
- **IMAGEPATHS AND OCR'ING EACH OTHER –**
- # loop over all image paths in the input directory
- for imagePath in imagePaths:
- # load the input image from disk and resize it
- image = cv2.imread(imagePath)
- image = imutils.resize(image, width=600)

License/Number Plate Recognition (ANPR) with Python

```
# apply automatic license plate recognition
(lpText, lpCnt) = anpr.find_and_ocr(image, psm=args["psm"],
clearBorder=args["clear_border"] > 0)
# only continue if the license plate was successfully OCR'd
if lpText is not None and lpCnt is not None:
# fit a rotated bounding box to the license plate contour and
# draw the bounding box on the license plate
box = cv2.boxPoints(cv2.minAreaRect(lpCnt))
box = box.astype("int")
cv2.drawContours(image, [box], -1, (0, 255, 0), 2)
# compute a normal (unrotated) bounding box for the license
# plate and then draw the OCR'd license plate text on the
# image
(x, y, w, h) = cv2.boundingRect(lpCnt)
cv2.putText(image, cleanup_text(lpText), (x, y - 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)
# show the output ANPR image
print("[INFO] {}".format(lpText))
cv2.imshow("Output ANPR", image)
cv2.waitKey(0)
```

- **CONCLUSION-** I've successfully applied ANPR to all of these images, including license/number plate examples on the front or back of the vehicle. But there are limitations due to foreground pixels because they touch the tesserect ocr.

- **REFERENCES- DATASET CURATED BY DEVIKA MISHRA FROM DATATURKS AND I HAVE USED THE SAME. WHICH IS FREELY AVAILABLE.**