CL 494 – B. Tech Project
Spring 2023

Image Processing Based Digitalization

of Displays in Process Industry

Report

Supervisor
Prof. Mani Bhushan

Submitted by

Divyansh Natani
Roll No: 190020043

Department of Chemical Engineering
Indian Institute of Technology Bombay

Acknowledgement

I would like to take this opportunity to thank and express heartfelt gratitude Prof. Mani Bhushan, Dept. of Chemical Engineering IIT Bombay for giving me this opportunity to work on this project and for his consistent support and guidance throughout.

I would also like to thank Ms Priti, for allowing me to take pictures from her 4-tank lab setup and providing necessary help and support.

I would also like to thank our parents, who provided us with the necessary resources to undertake and work on this project, and our friends, who always helped us whenever we needed it, in every possible way.

Table of Contents

Chapter 1 - Introduction to Problem Statement

Small scale industries (where the capital investment is one-time and the process of manufacturing, production and servicing are done on a small scale) generally have screen displays outside the machinery and a dedicated labor is employed to note the reading periodically. This record keeping is a monotonous task and is a waste of human resources. Further, since record keeping is done physically, generally on record registers, the power of data analytics is not used to optimize the process, nor smart safety instruments are used, leading to high loss of opportunity cost.

Medium and Large Scale industries generally tackle this situation by installing centralized control rooms with manual/digital record keeping, optimizing the labor cost. This setup is possible while installing the plant as it needs high interconnection of wires and LANs or in major restructuring of plants. In small scale industries, the plant has limited space and does not have space for centralized systems.

Furthermore, in a working plant, installation of wires and a centralized system will require partial/full shutdown, as it would require construction work, replacement of equipment which do not allow digital reading transfer. Plants cannot afford to close the production as it leads to huge losses and thus, stick to manual mode of reading collection.

Therefore, we propose an image-processing based solution, which is less capital intensive as well as can be installed in a working plant, occupying very little space and no human power. In this system, we will periodically take pictures of the readings using a camera and will use image processing techniques to detect the numerical value. Further, we can use a server to store this value and use this data for further plant optimisation procedures.

Chapter 2 - Digital Displays

Digital displays used in instruments and machinery are seven segment displays, which are used to display numerical values between 0-9.

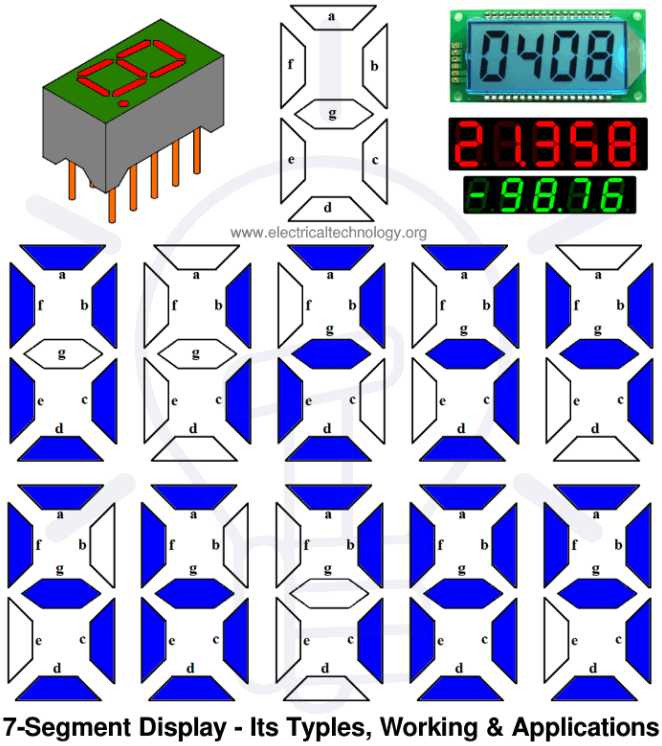


Fig 1. Seven Segment display of digits 0-9

The seven segments can be said to be turned on(denoted by 1), and off (denoted by 0) and we can represent digits 0-9 from a set of on and off segments. We can use this knowledge to detect a number from the photo of the display by deconstructing the seven segments of the digitally displayed number.

	Segments						
Digits	0	1	2	3	4	5	6
0	1	1	1	0	1	1	1
1	0	0	0	0	0	1	0
2	1	0	1	1	1	0	1
3	1	0	1	1	0	1	1
4	0	1	1	1	0	1	0
5	1	1	0	1	0	1	1
6	1	1	0	1	1	1	1
7	1	0	1	0	0	1	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

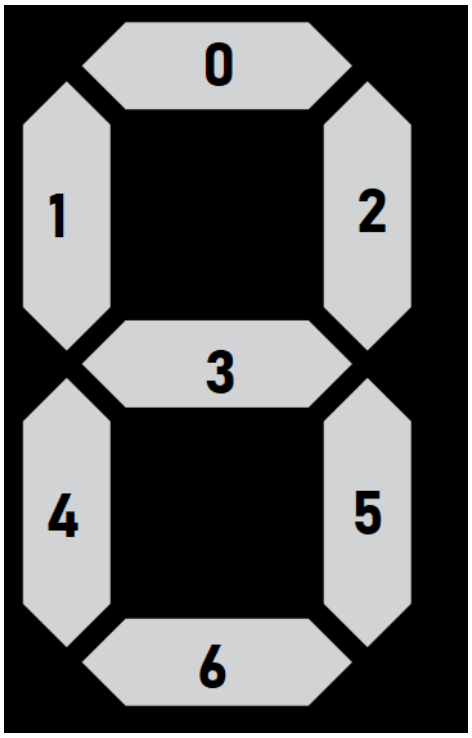


Fig 2 - Seven Segment display bits

Chapter 3 - Algorithm

The most important part of conversion of displays into numerical reading is the image processing algorithm. This algorithm takes a reading image and converts it into a set of numbers that can be stored. Firstly, the algorithm identifies the boundaries of the reading devices and identifies the display area that is to be used for further processing. This can be done either by using a blurring image and using Canny function from Computer Vision Library (openCV, an open source library maintained by Intel Corporations) or by manually selecting the bounding box, as both the equipment display and camera are stationary. If the equipment skin does not have sharp boundaries or clear image boundaries, cv2 library might give garbage results and thus we can use the bounding box method for display selection.

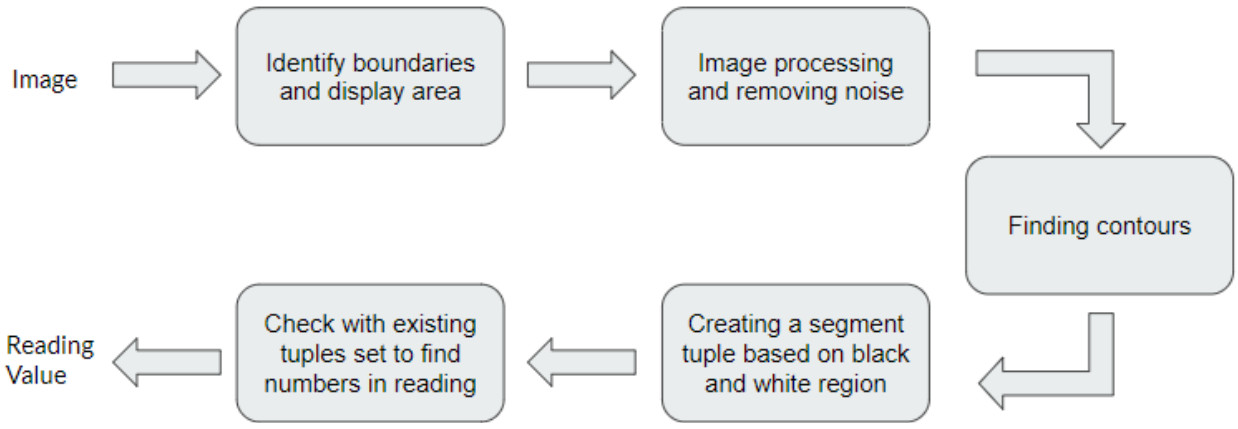


Fig 3 - Algorithm for converting reading image to readings

Secondly, we need to reduce noise from the image of the display so that we can convert the image into a black and white image, with black as the segments which are on (i.e segment-bit=1), or segments which are off (i.e. segment-bit=0). Various functions are available in the openCV library, which can help us achieve the goal and choosing these algorithms/functions for image processing will be case to case dependent.

Some of the functions are:

- cv2.THRESH_OTSU
- cv2.MORPH_ELLIPSE and cv2.MORPH_OPEN
- cv2.medianBlur()
- cv2.convertScaleAbs()

Afterwards, we can specifically find contours for each number on the readings and create a 7 bit tuple with each bit turned on or off based on whether the segment-bit is turned on or off identified on a confidence interval (if 50% or more of a segment is filled with black, then segment-bit is on, otherwise off). Afterwards, we can check the tuple with the existing set to identify the number.

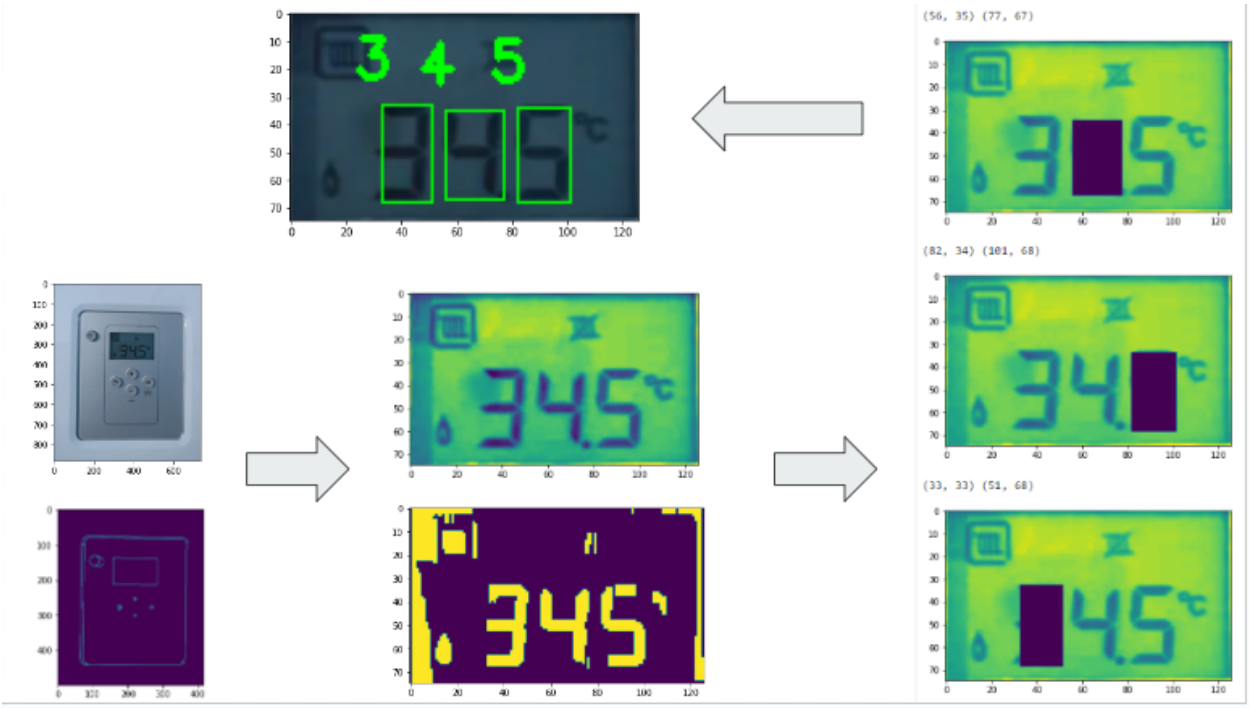


Fig 4 - Converting image of temperature reading device to numerical reading using above algorithm

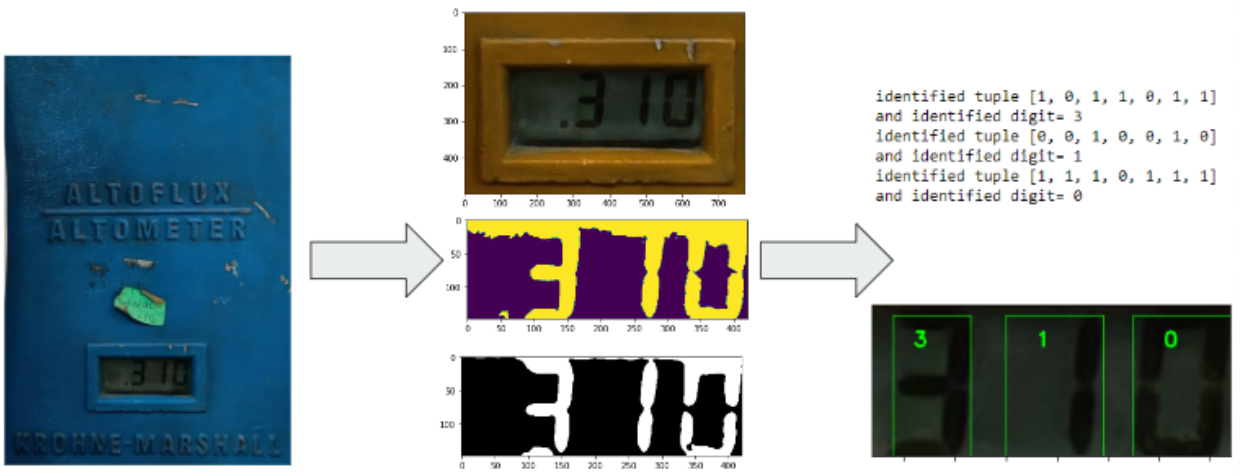


Fig 5 - Applying above algorithm to lab equipment present in Chemical Engineering Lab, Department of Chemical Engineering, IIT Bombay

Chapter 4 - Working System

We can use the above algorithm to develop a full fledged working system of identifying numbers from images and storing them in a database. Our system will comprise three major parts: the camera, a local server, and a hosted database.

4.1 - IP Camera

The IP Camera industry has boomed in the last 5 years due to increasing use of camera technology in domestic and shop security. The prices have dipped and added features like high resolution, 360° view and wireless control have been added for a competitive environment. The prices of these low to medium use cameras range from Rs 1,000 to Rs 2,500, with good resolution and added features. Higher price range cameras (up to Rs 10,000) have extremely high resolutions and can be used in case of high disturbance and high number of readings in single frame cases. In our case, we will focus on low price cameras with reasonable resolution.

4.1.1 - IP Camera - How does it work?

IP Camera comprises an internal router system, which works as a host and broadcasts the livestream view on a particular port. So, anyone who is able to access the local area network will be able to see and use this broadcast stream. Further, this stream can be protected by passwords for increased security.

These IP cameras provide APIs for connecting to streams and taking live pictures with given resolution and view properties. These APIs can be connected to servers for periodically taking pictures and processing them to identify readings.

For this project, we have used an android mobile application, “IP Camera”, made by *Pavel Khlebovich* available on Google Play Store. This application replicates the behavior of a physical IP camera and provides a live stream captured from the camera of the mobile in which it is installed. Furthermore, this livestream can be accessed by computers connected to the same wifi network as the mobile device.

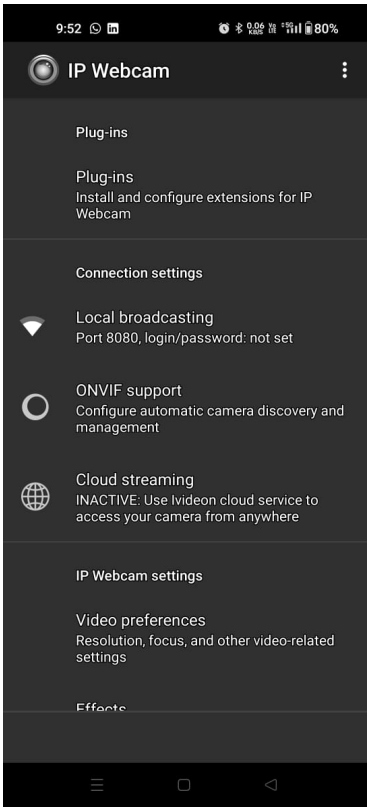


Fig - Screenshot of “IP Camera” app by *Pavel Khlebovich*

Fig - IP Camera available in markets (Source: amazon.com)

4.2 - Local Server

Local server could be any computer device connected on the same local area networks as the camera. In our project, we have used a python script as a server, which runs periodically and connects to the camera to catch a snapshot of the livestream and then process it with an OCR algorithm to identify the numbers and then send this reading to a local/cloud based database system.

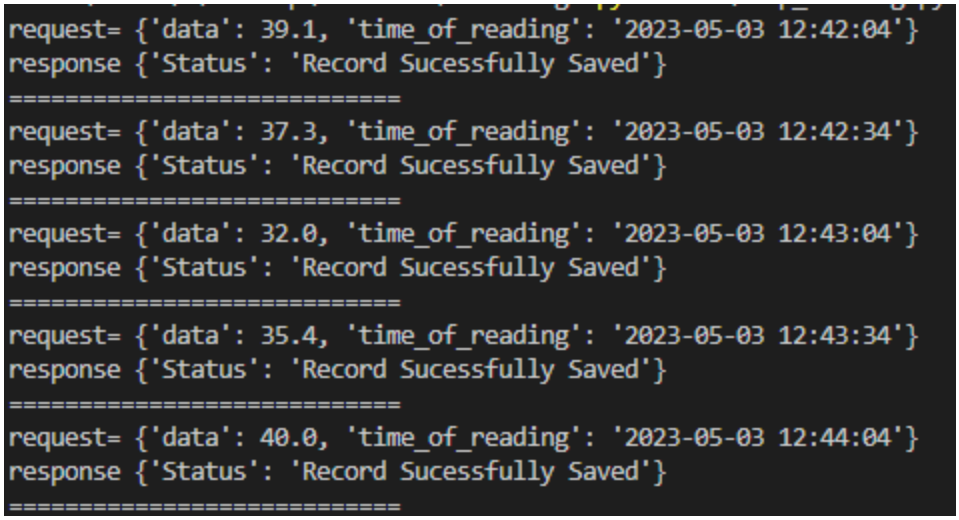


Fig - Snapshot of Lab reading and response of python server

In case of implementation of this project in a large-scale industry, where connecting all the cameras to a single local area network is not possible, we can use a Raspberry Pi system, which can host this python script, convert the image into numbers and send these readings to the database. Or, we can use an low capacity arduino system, which will simply collect the image from the camera and dump it into a local flask based high capacity system, which can run the python script and convert the images into reading and then send the readings to the database.

4.3 - Database implementation

For this project, we have used a python based REST Django framework with SQLite as a database for hosting the server. Django is a powerful framework with both backend and frontend development capabilities and further provides features like ‘Django Admin’ to manage and maintain data.

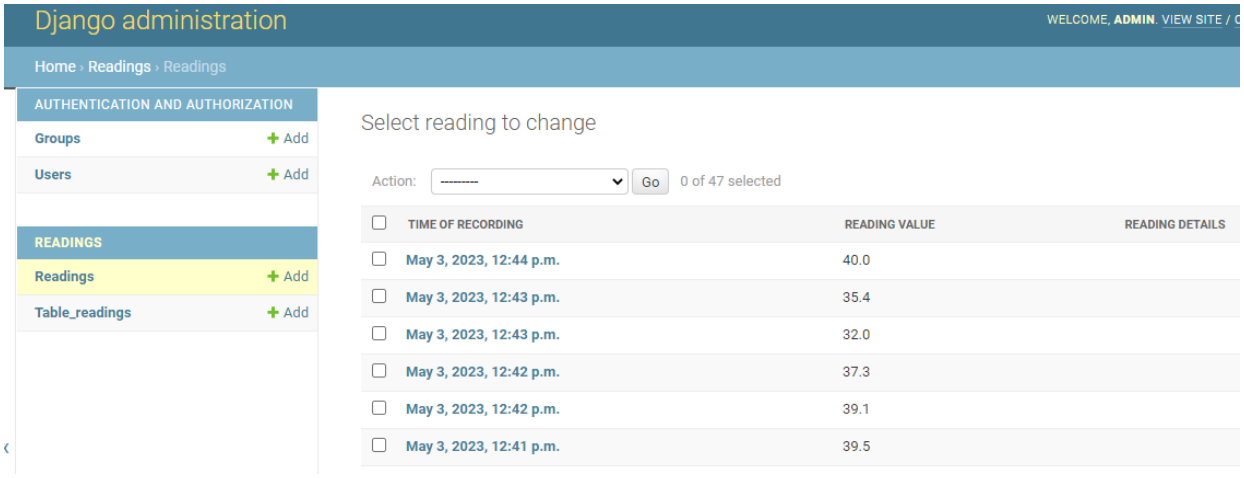


Fig - Screenshot of Django admin with data of 'Reading' table

Alternate options for this technology could be Python based Flask framework or php based Laravel framework.

Chapter 5 - Future Scope

First of all, a full scale lab testing will have to be done on the lab equipment, to certify the working of the system in a physical working environment. This would ensure that all the corner cases, like obstruction of image by some object, change in lighting conditions of the room, change in camera or equipment angle due to physical disturbance, failure of server to send data to database and more. This corner case can be easily handled by ignoring the reading, if it is a one time bad reading, or by notifying the owner (using digital emails or notifications or physical sirens) for continuous bad reading.

Furthermore, additions can be done on backend framework to include graphical representation of data, like live graphs and histograms, making it a full fledged system and alert notifications can be added using services available like Amazon Web Services or Google Firebase to make it a small IOT based smart reading system,

References

1.) *Home* (2023) *OpenCV*. Available at: https://opencv.org/

2.) Rosebrock, A. (2023) *Recognizing digits with opencv and python*, *PyImageSearch*. Available at: https://pyimagesearch.com/2017/02/13/recognizing-digits-with-opencv-and-python/

3.) *Buy Imou WIFI Wireless Security Camera 256GB SD card support, 1080p full HD, audio recording, night vision, abnormal sound alarm, Alexa Google assistant compatible with cameras (white) online at low price in India: Imou Camera Reviews & Ratings* (no date) *Buy Imou WiFi Wireless Security Camera 256GB SD Card Support, 1080P Full HD, Audio Recording, Night Vision, Abnormal Sound Alarm, Alexa Google Assistant Compatible with Cameras (White) Online at Low Price in India | Imou Camera Reviews & Ratings - Amazon.in*. Available at: https://www.amazon.in/Imou-Security-Detection-Cue-2C/dp/B08HF1FJ6Q/

4.) *Buy conbre multiplexer 2 pro {upgraded} HD SMART WIFI wireless IP CCTV security camera: Night vision: 2-way audio: Support 64 GB Micro SD card slot online at low price in India: Conbre Camera Reviews & Ratings* (no date) *Buy Conbre MultipleXR2 Pro {Upgraded} HD Smart WiFi Wireless IP CCTV Security Camera | Night Vision | 2-Way Audio | Support 64 GB Micro SD Card Slot Online at Low Price in India | Conbre Camera Reviews & Ratings - Amazon.in*. Available at: https://www.amazon.in/gp/product/B07ZMKFXVL/ref=ask_ql_qh_dp_hza

5.) *Seven segment displays* (2022) *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/seven-segment-displays/

6.) Technology, E. (2020) *Seven segment display: 7-segment display types and working*, *ELECTRICAL TECHNOLOGY*. Available at: https://www.electricaltechnology.org/2020/05/7-seven-segment-display.html

Appendix

This appendix contains Jupyter Notebook codes of various Image processing codes written for instrument, lab and dummy images. Further codes of python files can be accessed using this GitHub link.

https://github.com/DivyanshNatani/OCR-Display-Digitalisation

Further, REST Django based Database application code can be accessed using this GitHub repository.

https://github.com/DivyanshNatani/OCR-Display-Database

Image processing of a Temprature reading device

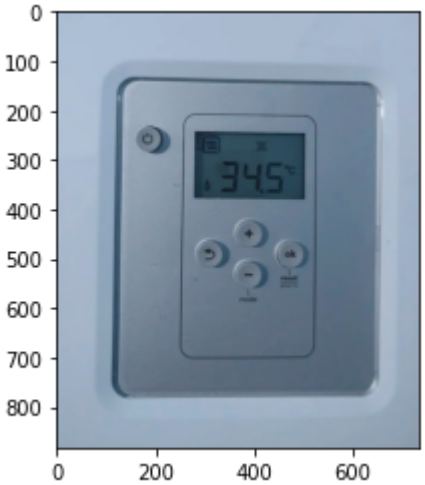
In [1]:
```
!pip install imutils
```

```
Requirement already satisfied: imutils in c:\users\a\appdata\local\programs\python\python39\lib\site-packages (0.5.4)
[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [1]:
```
# import the necessary packages
from imutils.perspective import four_point_transform
from imutils import contours
import imutils
import cv2
from matplotlib import pyplot as plt
# define the dictionary of digit segments so we can identify
# each digit on the thermostat
DIGITS_LOOKUP = {
        (1, 1, 1, 0, 1, 1, 1): 0,
        (0, 0, 1, 0, 0, 1, 0): 1,
        (1, 0, 1, 1, 1, 1, 0): 2,
        (1, 0, 1, 1, 0, 1, 1): 3,
        (0, 1, 1, 1, 0, 1, 0): 4,
        (1, 1, 0, 1, 0, 1, 1): 5,
        (1, 1, 0, 1, 1, 1, 1): 6,
        (1, 0, 1, 0, 0, 1, 0): 7,
        (1, 1, 1, 1, 1, 1, 1): 8,
        (1, 1, 1, 1, 0, 1, 1): 9
}
```

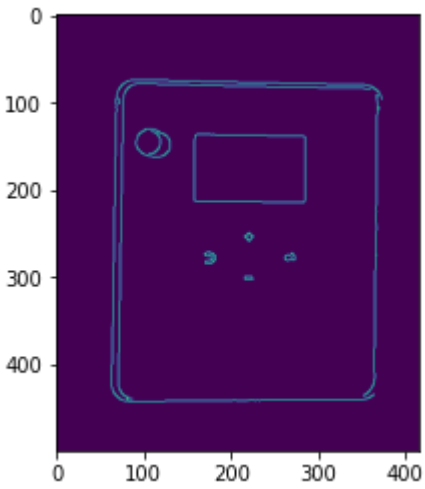
In [2]:
```
# load the example image
image = cv2.imread("example.jpg")
plt.imshow(image)
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x1ead55b5940>
```



In [3]:
```
# pre-process the image by resizing it,
# converting it to graycale, blurring it, and computing an edge map
image = imutils.resize(image, height=500)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
edged = cv2.Canny(blurred, 50, 200, 255)
plt.imshow(edged)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x1ead56ba790>
```



In [4]:
```
# find contours in the edge map, then sort them by their
# size in descending order
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
displayCnt = None
# loop over the contours
for c in cnts:
        # approximate the contour
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)
        # if the contour has four vertices, then we have found
        # the thermostat display
        if len(approx) == 4:
                displayCnt = approx
                break
```

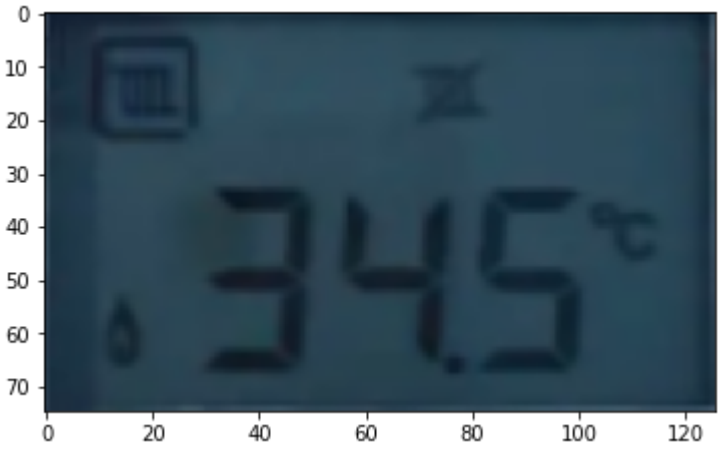
In [5]:
```
displayCnt
```

Out[5]:
```
array([[[159, 138]],

       [[158, 213]],

       [[284, 215]],

       [[285, 140]]], dtype=int32)
```

In [6]:
```
# extract the thermostat display, apply a perspective transform
# to it
warped = four_point_transform(gray, displayCnt.reshape(4, 2))
output = four_point_transform(image, displayCnt.reshape(4, 2))
```

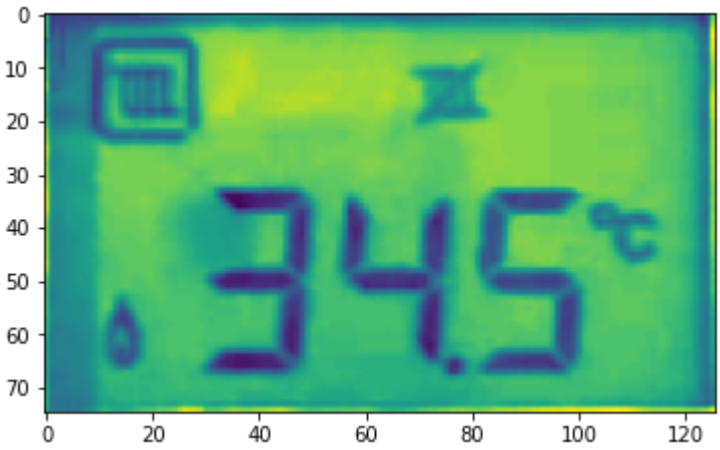
In [7]:
```
plt.imshow(output)
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x1ead5736820>
```



In [8]:
```
plt.imshow(warped)
```

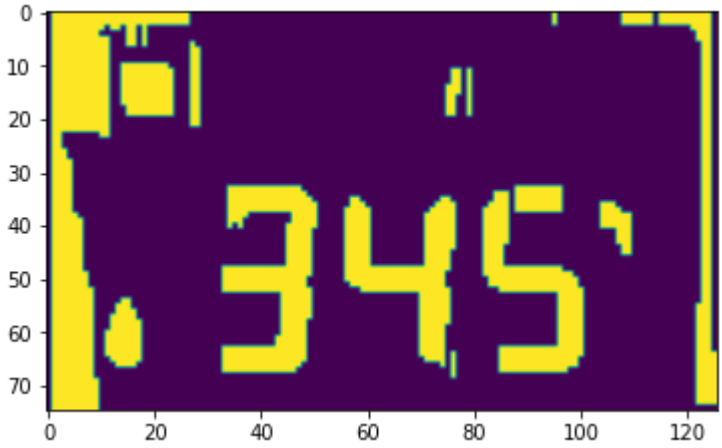
Out[8]:
```
<matplotlib.image.AxesImage at 0x1ead579cac0>
```



In [9]:
```
# threshold the warped image, then apply a series of morphological
# operations to cleanup the thresholded image
thresh = cv2.threshold(warped, 0, 255,
        cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (1, 5))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
```

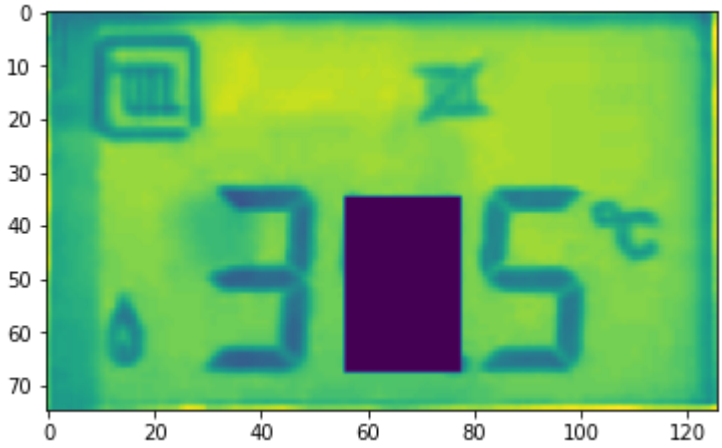
In [10]:
```
plt.imshow(thresh)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x1ead5810850>
```

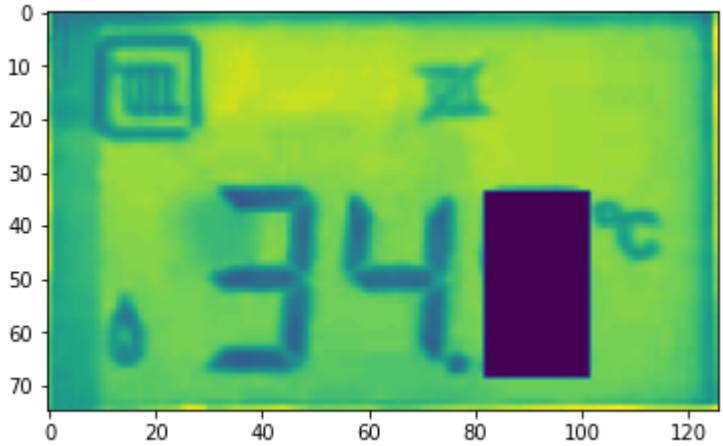


In [11]:
```
# find contours in the thresholded image, then initialize the
# digit contours lists
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
digitCnts = []
# loop over the digit area candidates
for c in cnts:
    # compute the bounding box of the contour
    (x, y, w, h) = cv2.boundingRect(c)

#     if the contour is sufficiently large, it must be a digit
    if w >= 15 and (h >= 30 and h <= 40):
        start_point=(x, y)
        end_point=(x+w, y+h)
        print(start_point,end_point)
        color = (0, 0, 0)
        thickness = -1
        img1 = cv2.rectangle(warped.copy(), start_point, end_point, color, thickness)
        plt.imshow(img1)
        plt.show()
        digitCnts.append(c)
```

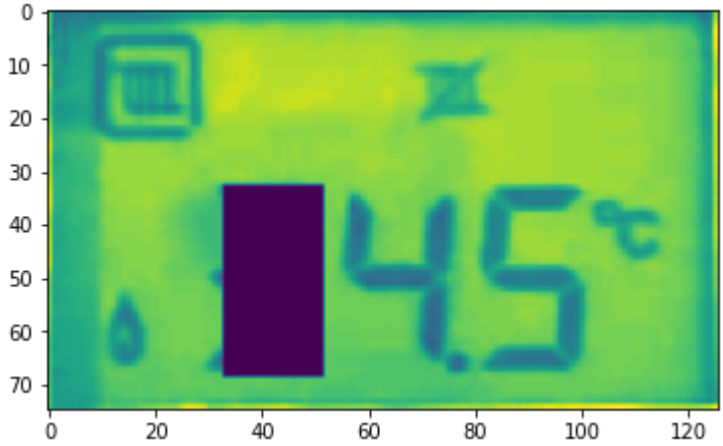
```
(56, 35) (77, 67)
```



```
(82, 34) (101, 68)
```




```
(33, 33) (51, 68)
```



In [12]:
```
# print(digitCnts)
```

In [13]:
```
# sort the contours from left-to-right, then initialize the
# actual digits themselves
digitCnts = contours.sort_contours(digitCnts,
        method="left-to-right")[0]
digits = []
```

In [14]:
```
len(digitCnts)
```

Out[14]:
```
3
```

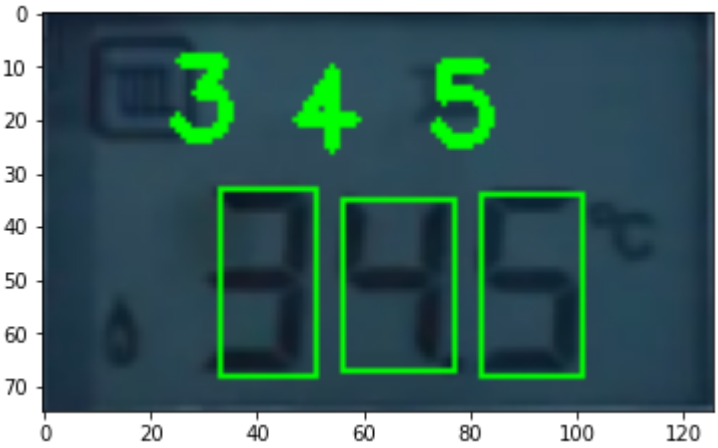
In [15]:
```
# loop over each of the digits
for c in digitCnts:
        # extract the digit ROI
        (x, y, w, h) = cv2.boundingRect(c)
        roi = thresh[y:y + h, x:x + w]
        # compute the width and height of each of the 7 segments
        # we are going to examine
        (roiH, roiW) = roi.shape
        (dW, dH) = (int(roiW * 0.25), int(roiH * 0.15))
        dHC = int(roiH * 0.05)
        # define the set of 7 segments
        segments = [
                ((0, 0), (w, dH)),      # top
                ((0, 0), (dW, h // 2)), # top-left
                ((w - dW, 0), (w, h // 2)),     # top-right
```

```
                ((0, (h // 2) - dHC) , (w, (h // 2) + dHC)), # center
                ((0, h // 2), (dW, h)), # bottom-left
                ((w - dW, h // 2), (w, h)),     # bottom-right
                ((0, h - dH), (w, h))   # bottom
        ]
        on = [0] * len(segments)
        # loop over the segments
        for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
                # extract the segment ROI, count the total number of
                # thresholded pixels in the segment, and then compute
                # the area of the segment
                segROI = roi[yA:yB, xA:xB]
                total = cv2.countNonZero(segROI)
                area = (xB - xA) * (yB - yA)
                # if the total number of non-zero pixels is greater than
                # 50% of the area, mark the segment as "on"
                if total / float(area) > 0.5:
                        on[i]= 1
        # lookup the digit and draw it on the image
        digit = DIGITS_LOOKUP[tuple(on)]
        digits.append(digit)
        cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
        cv2.putText(output, str(digit), (x - 10, y - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
        print(digit)
```

```
3
4
5
```

In [16]:
```
plt.imshow(output)
```

Out[16]:
```
<matplotlib.image.AxesImage at 0x1ead8b709d0>
```



In [17]:
```
# display the digits
print(digits)
print(u"{}{}.{} \u00b0C".format(*digits))
# cv2.imshow("Input", image)
# cv2.imshow("Output", output)
# cv2.waitKey(0)
```

```
[3, 4, 5]
34.5 °C
```

In [ ]:

Image processing of a dummy image

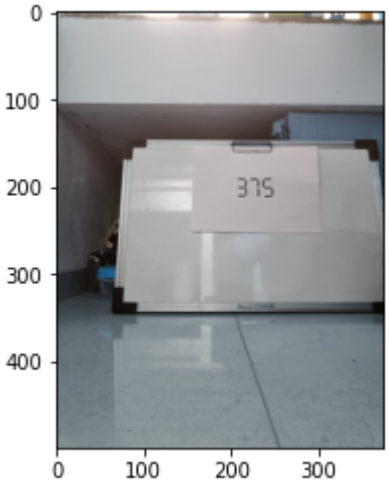
In [1]:
```
!pip install imutils
```

```
Requirement already satisfied: imutils in c:\users\a\appdata\local\programs\python\python39\lib\site-packages (0.5.4)
[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [13]:
```
# import the necessary packages
from imutils.perspective import four_point_transform
from imutils import contours
import imutils
import cv2
from matplotlib import pyplot as plt
import numpy as np
# define the dictionary of digit segments so we can identify
# each digit on the thermostat
DIGITS_LOOKUP = {
        (1, 1, 1, 0, 1, 1, 1): 0,
        (0, 0, 1, 0, 0, 1, 0): 1,
        (1, 0, 1, 1, 1, 0, 1): 2,
        (1, 0, 1, 1, 0, 1, 1): 3,
        (0, 1, 1, 1, 0, 1, 0): 4,
        (1, 1, 0, 1, 0, 1, 1): 5,
        (1, 1, 0, 1, 1, 1, 1): 6,
        (1, 0, 1, 0, 0, 1, 0): 7,
        (1, 1, 1, 1, 1, 1, 1): 8,
        (1, 1, 1, 1, 0, 1, 1): 9
}
```

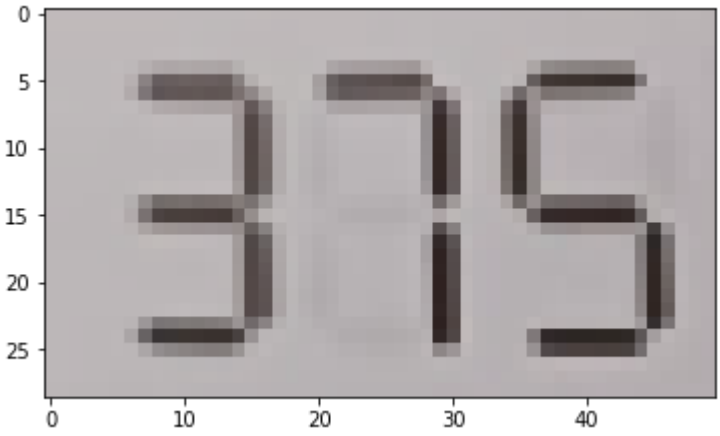
In [35]:
```
# load the example image
image = cv2.imread("375.jpg")
image = imutils.resize(image, height=500)
plt.imshow(image)
```

Out[35]:
```
<matplotlib.image.AxesImage at 0x1da2e172f40>
```

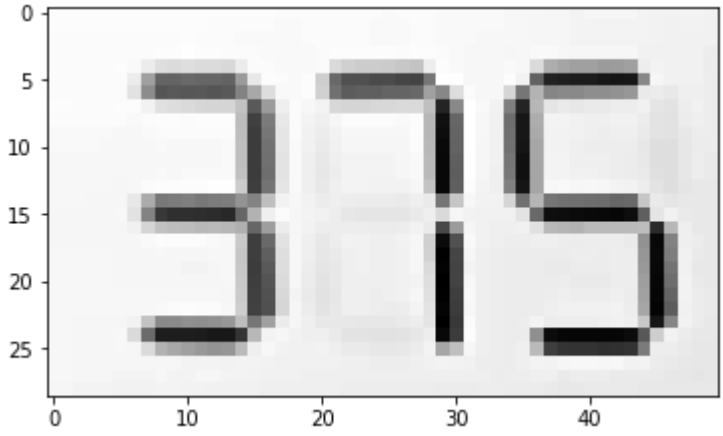


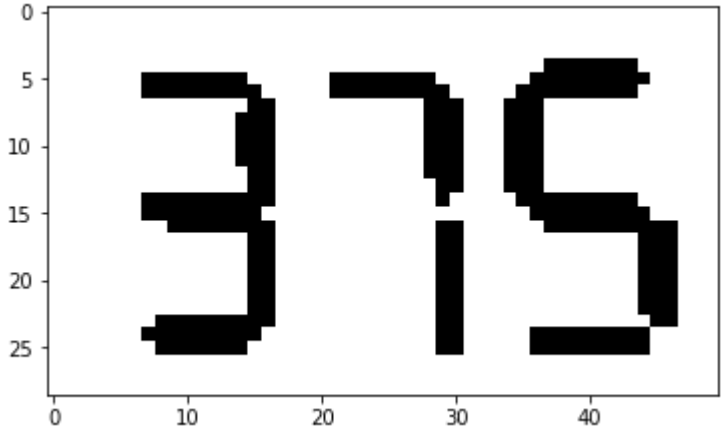
In [36]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

displayCnt=np.array([[200, 186],
       [200, 215],
       [250, 215],
       [250, 187]])
warped = four_point_transform(gray, displayCnt.reshape(4, 2))
output = four_point_transform(image, displayCnt.reshape(4, 2))
plt.imshow(output)
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x1da2e1ea130>
```

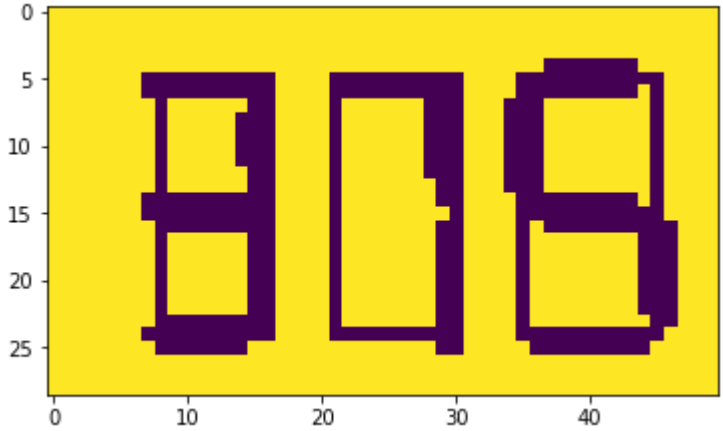


In [37]:
```
gray = warped
plt.imshow(gray, cmap="gray")
plt.show()

# median = cv2.medianBlur(warped, 3)
# # plt.imshow(output)
# adjusted = cv2.convertScaleAbs(gray, alpha=9, beta=20)

# median = cv2.medianBlur(adjusted, 9)
# plt.imshow(median, cmap="gray")
# plt.show()

# thresh_with_grey = cv2.threshold(median, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_
# plt.imshow(thresh_with_grey, cmap="gray")
# plt.show()

# kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (1, 1))
# thresh_with_grey = cv2.morphologyEx(thresh_with_grey, cv2.MORPH_OPEN, kernel)
# plt.imshow(thresh_with_grey)
# plt.show()


# gray = cv2.threshold(median, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
# plt.imshow(gray, cmap="gray")
# plt.show()

(thresh, blackAndWhiteImage) = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
plt.imshow(blackAndWhiteImage, cmap="gray")
plt.show()
```





In [38]:
```
import numpy

contours = [numpy.array([[8,5],[8,24],[16,24], [16,5]], dtype=numpy.int32),numpy.array

drawing = blackAndWhiteImage.copy()
for cnt in contours:
    cv2.drawContours(drawing,[cnt],0,(0,255,0),1)

plt.imshow(drawing)
digitCnts=contours
thresh=blackAndWhiteImage
```

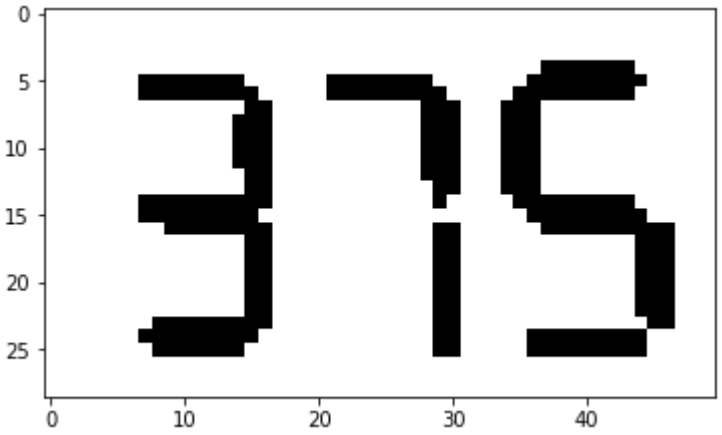


In [39]:
```
digits = []
len(digitCnts)
```

Out[39]: 3

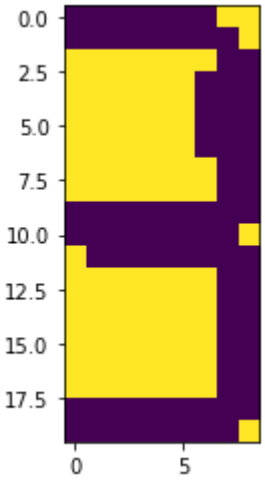
In [40]:
```
plt.imshow(thresh, cmap='gray')
```

Out[40]: <matplotlib.image.AxesImage at 0x1da2e2cd3d0>

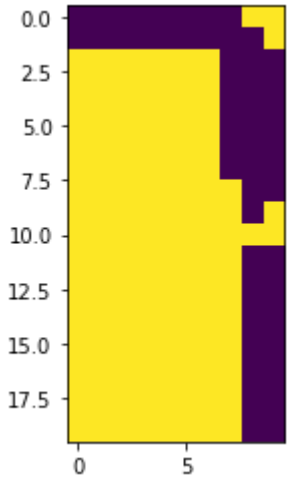


In [41]:
```
# loop over each of the digits
for c in digitCnts:
    # extract the digit ROI
    (x, y, w, h) = cv2.boundingRect(c)
    roi = thresh[y:y + h, x:x + w]
    plt.imshow(roi)
    plt.show()
    # compute the width and height of each of the 7 segments
    # we are going to examine
    (roiH, roiW) = roi.shape
    (dW, dH) = (int(roiW * 0.15), int(roiH * 0.1))
    dHC = int(roiH * 0.05)
    # define the set of 7 segments
    segments = [
        ((0, 0), (w-dW, dH)),   # top
        ((0, dH), (dW, h // 2 - dHC)),  # top-left
        ((w - dW, 0), (w, h // 2)),     # top-right
        ((0, (h // 2) - dHC) , (w, (h // 2) + dHC)), # center
        ((0, h // 2), (dW, h)), # bottom-left
        ((w - dW, h // 2), (w, h)),     # bottom-right
        ((0, h - dH), (w, h))   # bottom
    ]
    on = [0] * len(segments)
    # loop over the segments
    for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
        # extract the segment ROI, count the total number of
        # thresholded pixels in the segment, and then compute
        # the area of the segment
        segROI = roi[yA:yB, xA:xB]
        total = cv2.countNonZero(segROI)
        area = (xB - xA) * (yB - yA)
#         print(area, total)
#         cnt1=[numpy.array([[xA,yA],[xA,yB],[xB,yB], [xB,yA]], dtype=numpy.int32)]
# #         print(cnt1)
#         drawing2=roi.copy()
#         cv2.drawContours(drawing2,cnt1,0,(0,255,0),1)
#         plt.imshow(drawing2)
#         plt.show()
```

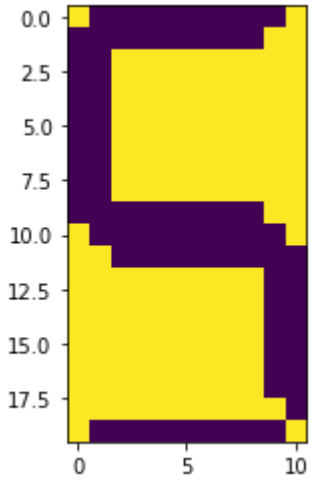
```
        # if the total number of non-zero pixels is greater than
        # 50% of the area, mark the segment as "on"
        if 1- total / float(area) > 0.5:
            on[i]= 1
#             print("has a digit!")
#         print("==========Moving to next seggment==========")
    # lookup the digit and draw it on the image
    try:
        digit = DIGITS_LOOKUP[tuple(on)]
        digits.append(digit)
        cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
        cv2.putText(output, str(digit), (x + w -10 , y + h +2),
            cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
        print(on)
        print(digit)
    except:
        print("Checking again...")
        for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
        # extract the segment ROI, count the total number of
        # thresholded pixels in the segment, and then compute
        # the area of the segment
            segROI = roi[yA:yB, xA:xB]
            total = cv2.countNonZero(segROI)
            area = (xB - xA) * (yB - yA)
            # if the total number of non-zero pixels is greater than
            # 30% of the area, mark the segment as "on"
            if 1-total / float(area) > 0.4:
                on[i]= 1
        print(on)
        try:
            digit = DIGITS_LOOKUP[tuple(on)]
            digits.append(digit)
            cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
            cv2.putText(output, str(digit),  (x + w -10 , y + h +2),
                cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
            print(digit)
        except:
            print('No digit found')
#     break
```




```
[1, 0, 1, 1, 0, 1, 1]
3
```



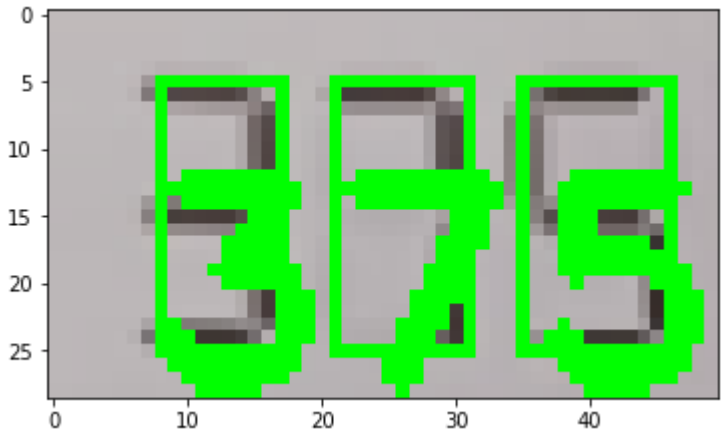

```
[1, 0, 1, 0, 0, 1, 0]
7
```




```
Checking again...
[1, 1, 0, 1, 0, 1, 1]
5
```

In [21]:
```
plt.imshow(output)
```

Out[21]:
```
<matplotlib.image.AxesImage at 0x1da2ce694f0>
```



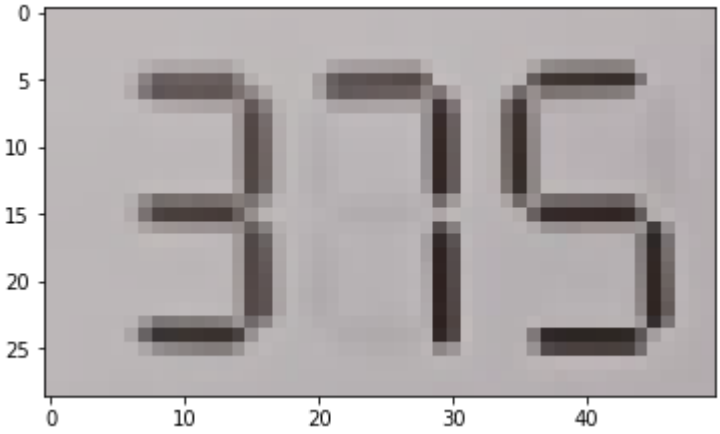
In [48]:
```
def returnDigits(name):
    image = cv2.imread(name)
    image = imutils.resize(image, height=500)
    plt.imshow(image)
```

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    displayCnt=np.array([[200, 186],
           [200, 215],
           [250, 215],
           [250, 187]])
    warped = four_point_transform(gray, displayCnt.reshape(4, 2))
    output = four_point_transform(image, displayCnt.reshape(4, 2))
    plt.imshow(output)

    gray = warped
#     plt.imshow(gray, cmap="gray")
#     plt.show()

    (thresh, blackAndWhiteImage) = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
#     plt.imshow(blackAndWhiteImage, cmap="gray")
#     plt.show()
    contours = [numpy.array([[8,5],[8,24],[16,24], [16,5]], dtype=numpy.int32),numpy.a

#     drawing = blackAndWhiteImage.copy()
#     for cnt in contours:
#         cv2.drawContours(drawing,[cnt],0,(0,255,0),1)

#     plt.imshow(drawing)
    digitCnts=contours
    thresh=blackAndWhiteImage
    digits = []
    # loop over each of the digits
    for c in digitCnts:
        # extract the digit ROI
        (x, y, w, h) = cv2.boundingRect(c)
        roi = thresh[y:y + h, x:x + w]
#         plt.imshow(roi)
#         plt.show()
        # compute the width and height of each of the 7 segments
        # we are going to examine
        (roiH, roiW) = roi.shape
        (dW, dH) = (int(roiW * 0.15), int(roiH * 0.1))
        dHC = int(roiH * 0.05)
        # define the set of 7 segments
        segments = [
            ((0, 0), (w-dW, dH)),       # top
            ((0, dH), (dW, h // 2 - dHC)),      # top-left
            ((w - dW, 0), (w, h // 2)), # top-right
            ((0, (h // 2) - dHC) , (w, (h // 2) + dHC)), # center
            ((0, h // 2), (dW, h)),     # bottom-left
            ((w - dW, h // 2), (w, h)), # bottom-right
            ((0, h - dH), (w, h))       # bottom
        ]
        on = [0] * len(segments)
        # loop over the segments
        for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
            # extract the segment ROI, count the total number of
            # thresholded pixels in the segment, and then compute
            # the area of the segment
            segROI = roi[yA:yB, xA:xB]
            total = cv2.countNonZero(segROI)
            area = (xB - xA) * (yB - yA)
    #         print(area, total)
    #         cnt1=[numpy.array([[xA,yA],[xA,yB],[xB,yB], [xB,yA]], dtype=numpy.int32)
```

```
    # #         print(cnt1)
    #         drawing2=roi.copy()
    #         cv2.drawContours(drawing2,cnt1,0,(0,255,0),1)
    #         plt.imshow(drawing2)
    #         plt.show()
            # if the total number of non-zero pixels is greater than
            # 50% of the area, mark the segment as "on"
            if 1- total / float(area) > 0.5:
                on[i]= 1
    #             print("has a digit!")
    #         print("===========Moving to next seggment===========")
        # lookup the digit and draw it on the image
        try:
            digit = DIGITS_LOOKUP[tuple(on)]
            digits.append(digit)
            cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
            cv2.putText(output, str(digit), (x + w -10 , y + h +2),
                cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
#             print(on)
#             print(digit)
        except:
#             print("Checking again...")
            for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
            # extract the segment ROI, count the total number of
            # thresholded pixels in the segment, and then compute
            # the area of the segment
                segROI = roi[yA:yB, xA:xB]
                total = cv2.countNonZero(segROI)
                area = (xB - xA) * (yB - yA)
                # if the total number of non-zero pixels is greater than
                # 30% of the area, mark the segment as "on"
                if 1-total / float(area) > 0.4:
                    on[i]= 1
#             print(on)
            try:
                digit = DIGITS_LOOKUP[tuple(on)]
                digits.append(digit)
                cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
                cv2.putText(output, str(digit),  (x + w -10 , y + h +2),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
#                 print(digit)
            except:
                print('No digit found')
    return digits
```

In [49]:
```
returnDigits('375.jpg')
```

Out[49]:
```
[3, 7, 5]
```



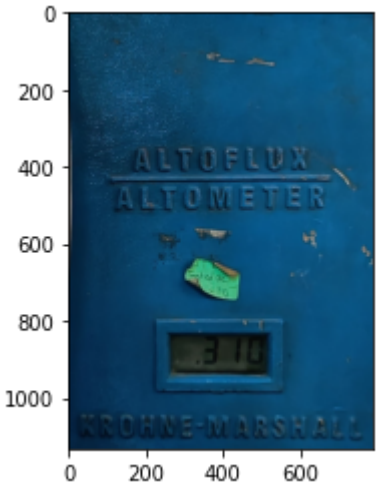
In [ ]:

Image processing of a 4-tank instrument available at Chemical Engineering Lab, Department of Chemical Engineering, IIT Bombay

In [43]:
```
from matplotlib import pyplot as plt
import numpy as np

# import the necessary packages
from imutils.perspective import four_point_transform
from imutils import contours
import imutils
import cv2
# define the dictionary of digit segments so we can identify
# each digit on the thermostat
DIGITS_LOOKUP = {
        (1, 1, 1, 0, 1, 1, 1): 0,
        (0, 0, 1, 0, 0, 1, 0): 1,
        (1, 0, 1, 1, 1, 1, 0): 2,
        (1, 0, 1, 1, 0, 1, 1): 3,
        (0, 1, 1, 1, 0, 1, 0): 4,
        (1, 1, 0, 1, 0, 1, 1): 5,
        (1, 1, 0, 1, 1, 1, 1): 6,
        (1, 0, 1, 0, 0, 1, 0): 7,
        (1, 1, 1, 1, 1, 1, 1): 8,
        (1, 1, 1, 1, 0, 1, 1): 9
}
```

In [44]:
```
orig_image=cv2.imread("../data/lab_image.jpg")

orig_image=cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB)
plt.imshow(orig_image)
```

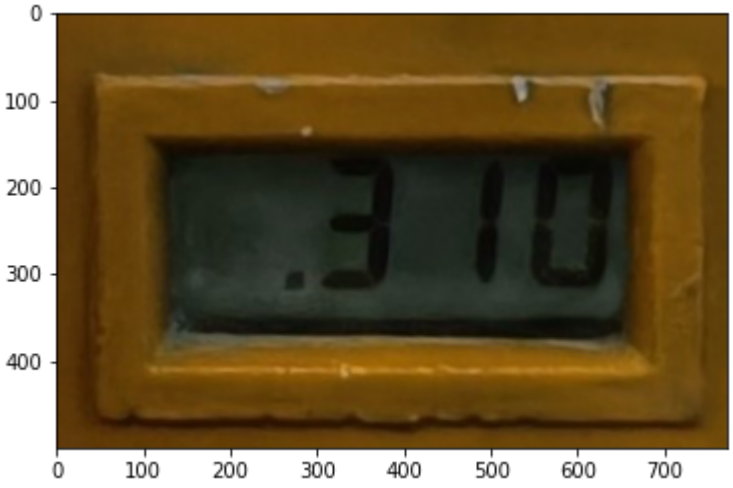
Out[44]:
```
<matplotlib.image.AxesImage at 0x18e782af220>
```



In [45]:
```
# load the example image
image = cv2.imread("num.jpg")
```

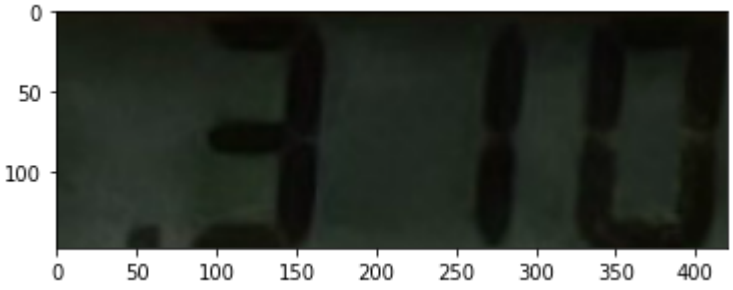
```
# pre-process the image by resizing it
image = imutils.resize(image, height=500)
plt.imshow(image)
```

Out[45]:
```
<matplotlib.image.AxesImage at 0x18e785a0970>
```



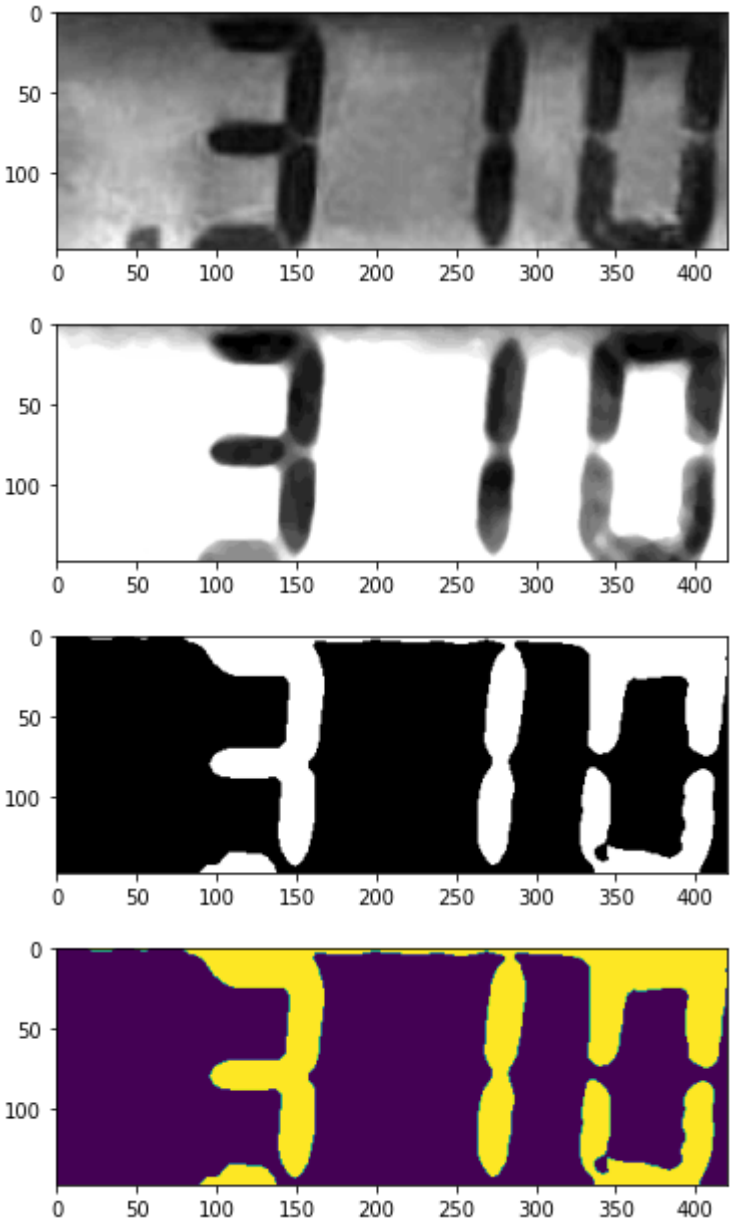
In [46]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

displayCnt=np.array([[220, 162],
       [220, 310],
       [640, 310],
       [640, 162]])
warped = four_point_transform(gray, displayCnt.reshape(4, 2))
output = four_point_transform(image, displayCnt.reshape(4, 2))
plt.imshow(output)
```

Out[46]:
```
<matplotlib.image.AxesImage at 0x18e7874b400>
```



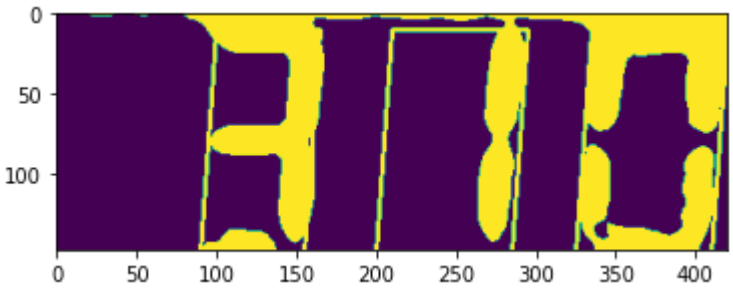
In [47]:
```
# img = cv2.imread(img_path)
# hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# lower_gray = np.array([0, 5, 50], np.uint8)
# upper_gray = np.array([179, 50, 255], np.uint8)
# mask_gray = cv2.inRange(hsv, lower_gray, upper_gray)
# img_res = cv2.bitwise_and(img, img, mask = mask_gray)
gray = warped
plt.imshow(gray, cmap="gray")
plt.show()

adjusted = cv2.convertScaleAbs(gray, alpha=9, beta=20)

median = cv2.medianBlur(adjusted, 9)
plt.imshow(median, cmap="gray")
plt.show()
```

```
thresh_with_grey = cv2.threshold(median, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OT
plt.imshow(thresh_with_grey, cmap="gray")
plt.show()

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (1, 1))
thresh_with_grey = cv2.morphologyEx(thresh_with_grey, cv2.MORPH_OPEN, kernel)
plt.imshow(thresh_with_grey)
plt.show()
```



In [48]:

```
import numpy

contours = [numpy.array([[90,150],[155,150],[165,10], [100,10]], dtype=numpy.int32),nu

drawing = thresh_with_grey.copy()
for cnt in contours:
    cv2.drawContours(drawing,[cnt],0,(255,255,0),2)

plt.imshow(drawing)
digitCnts=contours
thresh=thresh_with_grey
```



In [49]:
```
digits = []
len(digitCnts)
```

Out[49]:
```
3
```

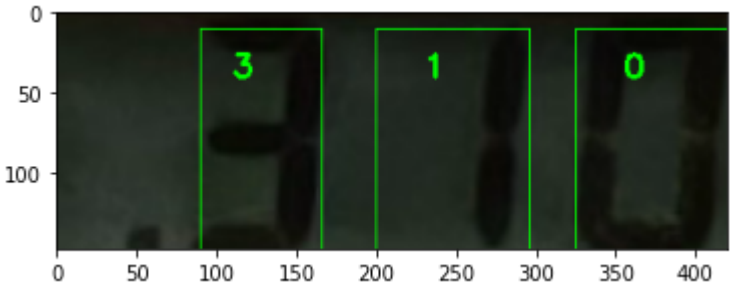
In [56]:
```
# loop over each of the digits
for c in digitCnts:
    # extract the digit ROI
    (x, y, w, h) = cv2.boundingRect(c)
    roi = thresh[y:y + h, x:x + w]
    # compute the width and height of each of the 7 segments
    # we are going to examine
    (roiH, roiW) = roi.shape
    (dW, dH) = (int(roiW * 0.25), int(roiH * 0.15))
    dHC = int(roiH * 0.05)
    # define the set of 7 segments
    segments = [
        ((0, 0), (w, dH)),      # top
        ((0, 0), (dW, h // 2)), # top-left
        ((w - dW, 0), (w, h // 2)),     # top-right
        ((0, (h // 2) - dHC) , (w, (h // 2) + dHC)), # center
        ((0, h // 2), (dW, h)), # bottom-left
        ((w - dW, h // 2), (w, h)),     # bottom-right
        ((0, h - dH), (w, h))   # bottom
    ]
    on = [0] * len(segments)
    # loop over the segments
    for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
        # extract the segment ROI, count the total number of
        # thresholded pixels in the segment, and then compute
        # the area of the segment
        segROI = roi[yA:yB, xA:xB]
        total = cv2.countNonZero(segROI)
        area = (xB - xA) * (yB - yA)
        # if the total number of non-zero pixels is greater than
        # 50% of the area, mark the segment as "on"
        if total / float(area) > 0.4:
            on[i]= 1
    # lookup the digit and draw it on the image
    try:
        digit = DIGITS_LOOKUP[tuple(on)]
        digits.append(digit)
        cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
#         print(x, y)
        cv2.putText(output, str(digit), (x + 30, y + 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
        print("identified tuple",on)
        print("and identified digit=",digit)
    except:
```

```
        for (i, ((xA, yA), (xB, yB))) in enumerate(segments):
        # extract the segment ROI, count the total number of
        # thresholded pixels in the segment, and then compute
        # the area of the segment
            segROI = roi[yA:yB, xA:xB]
            total = cv2.countNonZero(segROI)
            area = (xB - xA) * (yB - yA)
            # if the total number of non-zero pixels is greater than
            # 30% of the area, mark the segment as "on"
            if total / float(area) > 0.3:
                on[i]= 1
#         print(on)
        try:
            digit = DIGITS_LOOKUP[tuple(on)]
            digits.append(digit)
            cv2.rectangle(output, (x, y), (x + w, y + h), (0, 255, 0), 1)
            cv2.putText(output, str(digit), (x + 20, y + 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)
            print("identified tuple",on)
            print("and identified digit=",digit)
        except:
            print('No digit found')
```

```
identified tuple [1, 0, 1, 1, 0, 1, 1]
and identified digit= 3
identified tuple [0, 0, 1, 0, 0, 1, 0]
and identified digit= 1
identified tuple [1, 1, 1, 0, 1, 1, 1]
and identified digit= 0
```

In [51]:
```
plt.imshow(output)
```

Out[51]:
```
<matplotlib.image.AxesImage at 0x18e799b6670>
```



In [ ]: