

Computer Graphics and Visualization

Contents

Scenario.....	3
Introduction	4
Line Detection	6
Rotate image	14
Boxes	18
Identify word and signature.....	21
Store attendance in database	25
Visualization	29
Challengers faced.	31

Scenario

You are asked to create a student attendance management system based on the signing sheets. The signing sheets have a specific layout which is static. Students can sign the sheet at a given space using different color pens.

- The admin staff would take snapshots using their smart phones of these signing sheets and provide you the image files.
- They also provide you a text file containing the student indices and subject related information.
- Once you receive those files, your program should process the image and text content and identify whether a student is present or absent based on the appearance of a signature.

The main focus is with two key technologies:

- Image processing
- Data visualization

Introduction

This project is regarding the day to day problem of university attendance sheet, Which is printed by university with proper format. Lot of universities still use this kind of a signing sheet because it is classic way and it can do easily. Problem is, end of the day those informations need to record in the system. Currently it is inserted in the system by a person. It take too much time to count all the signatures according to the student. Some time signature not match with the student original signature. This kind of a situation need to identify.

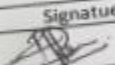
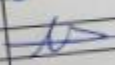


In this project there a several major image processing parts.

At the beginning of the execution it detect vertical lines and horizontal lines of the image, then identifies cross points to detect edges of the boxes. After Identifying of boxes processing take one box and get 2 edges to detect angle.

L 104

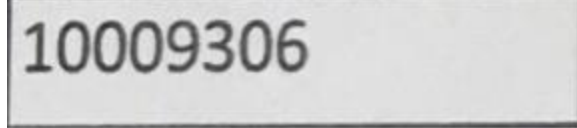
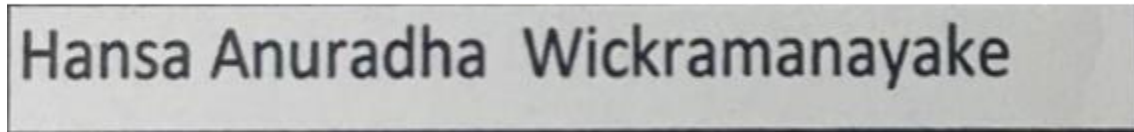
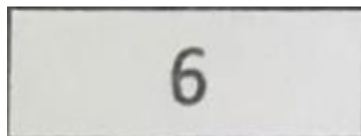
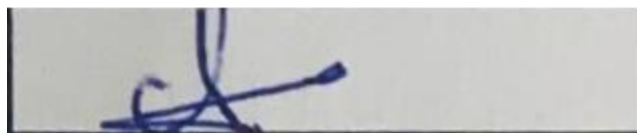
Signing Sheet
NSBM Green University Town
School of Computing
BSc (Hons) in Software Engineering - 2016.1

C.C.V.

Date		Lecturer's Name		Signature
05/07/2019		Mr. Rasika Ranawirera		
No	Student No	Title	Student Name	Signature
1	10000409	Ms	M S Dilshanika Perera	
2	10009301	Mr	C W M A Shehan Abeyrathne	
3	10009302	Mr	B A K M Chithrananda	
4	10009303	Ms	W Shashini Minosha De Silva	
5	10009304	Mr	K L Udara Maduranga Uyanage	
6	10009306	Mr	Hansa Anuradha Wickramanayake	

An arrow labeled "Angle" points to the intersection of the "Lecturer's Name" and "Student Name" columns.

Then rotate image. After rotation again identifies boxes to get new coordinates of ages. After that segmentation is began. In segmentation all the boxes cropped and create segment images.

A rectangular box containing the number 10009306 in a black, sans-serif font.A rectangular box containing the name Hansa Anuradha Wickramanayake in a black, sans-serif font.A rectangular box containing the number 6 in a black, sans-serif font.A rectangular box containing a handwritten signature in blue ink.

We use google vision API to identify numbers and letters. It is pretty much accurate API. Then those words and characters are returned by API. After that we use logical function to identify if it is student Id, student name or sign. This function create an array which saves as 2 dimensions for rows and column. This arrays passes to the database and it stores information about that student.

```
C:\Python3\python.exe C:/Users/pasindu/PycharmProjects/TableDataExtractor/Main.py
[(771, '10000409', 'Ms', 'MS Dilshanika Perera', '0'), (772, '10009301', 'Mr', 'CWM A Shehan Abeyrathne', '0'),
Process finished with exit code 0
```

Then visualize the data from barcharts.

Line Detection

The technique is mainly based on horizontal and vertical black runs processing as well as on image / text areas estimation in order to remove line segments that belong to these areas. Initially, a series of morphological operations with appropriate structuring elements is performed in order to link potential line breaks and to enhance line segments.

The following are the distinct steps of the suggested line detection technique:

- (I) horizontal and vertical line estimation
- (ii) enhancement of line estimation using the removal of image / text areas.

Horizontal and vertical line estimation

After refinement of this result, this calculation of lines will be done by eliminating line segments belonging to image / text areas. The proposed line detection algorithm is based on the processing of horizontal and vertical black runs and a collection of morphological operations with appropriate structuring elements to connect potential line breaks and improve line segments.

The average character height AH that was measured in Section 2.1 depends on all the parameters used in this step. Starting with the IM binary image (1s corresponding to text regions and 0s corresponding to context regions).

We take the following steps:

STEP 1 : We proceed to a series of image IM morphological operations with appropriate structuring elements. Our target is to connect line breaks or dotted lines, but not to connect neighboring characters. We measure IMH and IMV images for horizontal and vertical line detection.

STEP 2 : Both 1s of IMH and IMV images that belong to high-length and small-width line segments are translated to L mark values. In the case of horizontal lines, all IMH 1s belonging to horizontal black length ranges larger than AH and vertical black length ranges smaller than AH are translated to L. In the case of vertical lines, all IMV 1s belonging to vertical black lengths greater than AH and to horizontal black lengths less than AH are converted to L lengths.

STEP 3 : IMH and IMV images are smoothed accordingly in horizontal and vertical directions in order to set all short runs that have a value other than L to L. Horizontal runs of IMH pixels with values not equal to L and lengths less than AH are set to L in the case of horizontal lines. Vertical runs of IMV pixels with values not equal to L and a length less than AH are set to L in the case of vertical lines.

STEP 4 : Horizontal and vertical lines in IMH and IMV images, respectively, are re-defined from all linked components with a length greater than $2 AH$ of L-valued pixels.

Enhancement of line estimation using the removal of image / text areas.

The measurement of image / text areas is achieved by performing horizontal and vertical image IMn smoothing for pixels that do not belong to the horizontal or vertical lines detected. All related components of great height ($> 3AH$) belong to graphics, images or text after this smoothing. In this point, as vertical and horizontal lines are omitted, tables will not appear as individual connected components in the final smooth picture.

We take the following steps:

STEP 1 : By setting all horizontal runs with 0's that have a duration of less than $1.2AH$ to L, we lead to a horizontal smoothing of image IMV.

STEP 2 : By setting all vertical runs with 0's that have a duration of less than $1.2AH$ to L, we lead to vertical smoothing.

STEP 3 : IT image / text areas are specified in the resulting IMV image from L-valued linked components with a rectangle height greater than 3 AH.

Read the file from the correct direction as the first step, using the threshold to convert the input image to a binary image and invert it to get a black background and white lines and fonts.

```
dir = "D:\\nsbm pasindu\\Sem8\\CGV\\Project\\Course work\\"
dirSegments = "D:\\nsbm pasindu\\Sem8\\CGV\\Project\\Course work\\segments\\"

file = dir+'\\4.jpeg'
img = cv2.imread(file,0)

# thresholding the image to a binary image
thresh, img_bin = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

# inverting the image
img_bin = 255 - img_bin
cv2.imwrite(dir+'cv_inverted.png', img_bin)
# Plotting the image to see the output
#plotting = plt.imshow(img_bin, cmap='gray')
plt.show()
```

Screen Shot :

L 104







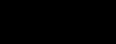
Signing Sheet

NSBM Green University Town

School of Computing

BSc (Hons) in Software Engineering - 2016.1

CCV.

Date		Lecture's Name		Signature
05/07/2019		13.03 14.03	Dr. Rasika Ranawake	
No	Student No	Title	Student Name	Signature
1	10000409	Ms	M S Dilshanika Perera	
2	10009301	Mr	C W M A Shehan Abeyrathne	
3	10009302	Mr	B A K M Chithrananda	
4	10009303	Ms	W Shashini Minosha De Silva	
5	10009304	Mr	K L Udara Maduranga Liyanage	
6	10009306	Mr	Hansa Anuradha Wickramanayake	

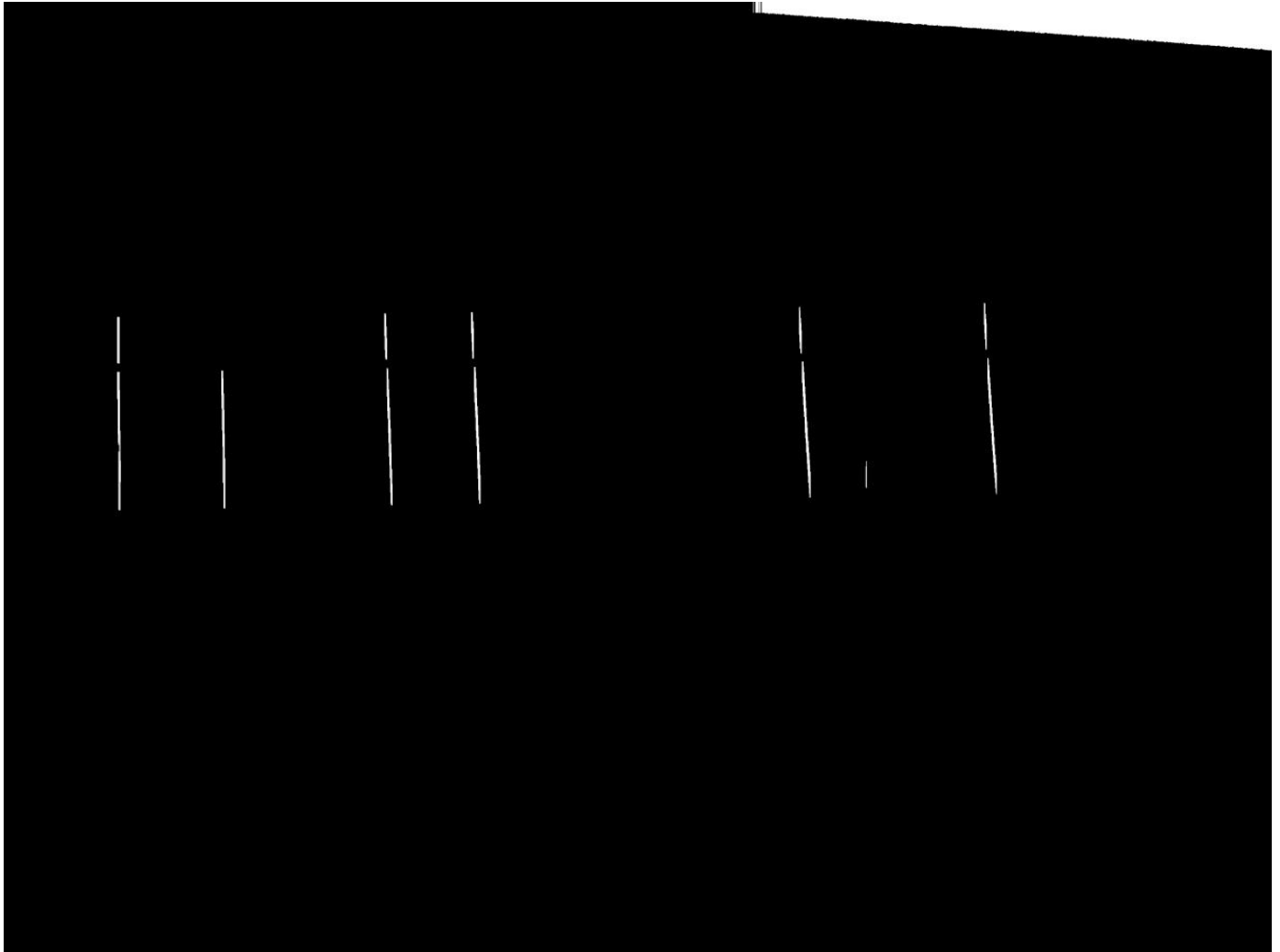
The next step is to describe the kernel for the detection of rectangular boxes, and the tabular structure afterwards. First, we determine the length of the kernel and then the vertical and horizontal kernels, so that all vertical lines and all horizontal lines are detected later.

```
# countcol(width) of kernel as 100th of total width
kernel_len = np.array(img).shape[1] // 100
# Defining a vertical kernel to detect all vertical lines of image
ver_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, kernel_len))
# Defining a horizontal kernel to detect all horizontal lines of image
hor_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_len, 1))
# A kernel of 2x2
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
```

The next step is the detection of the vertical lines.

```
# Use vertical kernel to detect and save the vertical lines in a jpg
image_1 = cv2.erode(img_bin, ver_kernel, iterations=3)
vertical_lines = cv2.dilate(image_1, ver_kernel, iterations=3)
cv2.imwrite(dir+"vertical.jpg", vertical_lines)
# Plot the generated image
#plotting = plt.imshow(image_1, cmap='gray')
#plt.show()
```

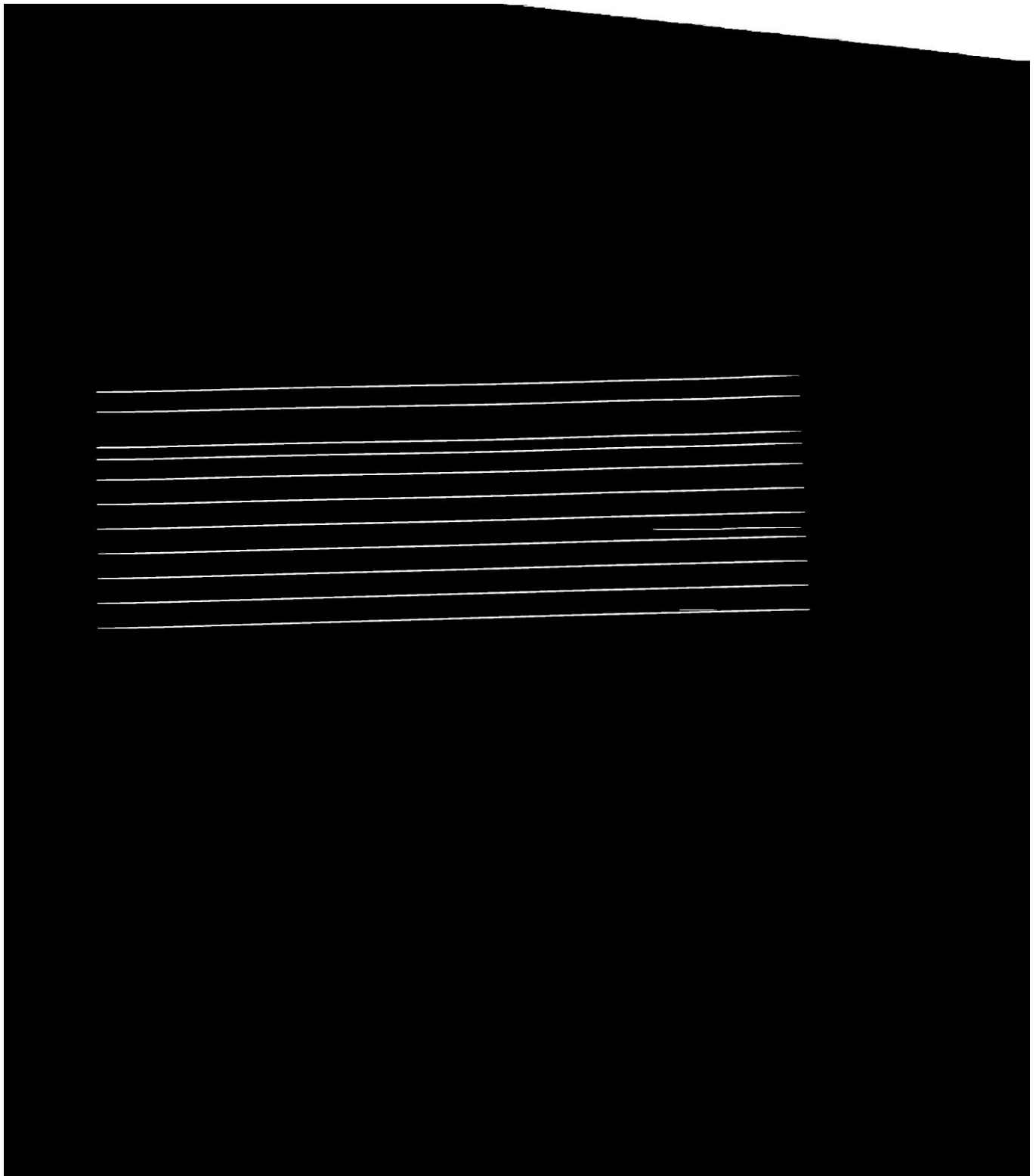
Screen Shot :



And now the same for all horizontal lines.

```
# Use horizontal kernel to detect and save the horizontal lines in a jpg
image_2 = cv2.erode(img_bin, hor_kernel, iterations=3)
horizontal_lines = cv2.dilate(image_2, hor_kernel, iterations=3)
cv2.imwrite(dir+"horizontal.jpg", horizontal_lines)
# Plot the generated image
#plotting = plt.imshow(image_2, cmap='gray')
#plt.show()
```

Screen Shot :



As a next step, by weighting both with 0.5, we combine the horizontal and vertical lines into a third picture. To detect each cell, the goal is to get a simple tabular structure.

```
# Combine horizontal and vertical lines in a new third image, with both having same weight.
img_vh = cv2.addWeighted(vertical_lines, 0.5, horizontal_lines, 0.5, 0.0)
# Eroding and thresholding the image
img_vh = cv2.erode(~img_vh, kernel, iterations=2)
thresh, img_vh = cv2.threshold(img_vh, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
cv2.imwrite(dir+"img_vh.jpg", img_vh)
bitxor = cv2.bitwise_xor(img, img_vh)
bitnot = cv2.bitwise_not(bitxor)
# Plotting the generated image
#plotting = plt.imshow(bitnot, cmap='gray')
#plt.show()
```

[illegible]

Rotate image

First identifies cross points to detect edges of the boxes. After Identifying of boxes processing take one box and get 2 edges to detect angle. Because we need to know the existing angles and the angles we want to get. The table should be rotate as usual

```
countSignature = 0
w, h = 5, 20;
AttendentSheet = [[0 for x in range(w)] for y in range(h)]
arrRowCount = 0
arrColCount = 0
previousCol = 0
SignMinWidth = 0
SignMaxWidth = 0
SignMinHeight = 0
SignMaxHeight = 0
AngleX1 = 0
AngleY1 = 0
AngleX2 = 0
AngleY2 = 0
i = 0
j = 0
for b, centroid in box:
    draw.line(b + [b[0]], fill='green')
    cx, cy = centroid
    draw.ellipse((cx - 2, cy - 2, cx + 2, cy + 2), fill='red')
    x1, y1 = b[0]
    x2, y2 = b[1]
    x3, y3 = b[2]
    x4, y4 = b[3]

    if x3 - x1 <= 1000:

        img3 = img.crop((x1, y1, x3, y3))
        path = dir+"segments\\" + str((x1, y1, x3, y3)) + " aa.jpg"
        img3.save(path)

        client = vision.ImageAnnotatorClient()
        with io.open(path, 'rb') as image_file:
            content = image_file.read()

        image = vision.types.Image(content=content)
        response = client.text_detection(image=image)
        texts = response.text_annotations

        if texts:
            texts[0].description = texts[0].description.rstrip('\n')

            if texts[0].description == "Student Name":
                #print("Student Name coordinate", x1,y1," - ", x2,y2," - ", x3,y3," - ", x4,y4,)
                AngleX1 = x4
                AngleY1 = y4

            if texts[0].description == "Signature":
                AngleX2 = x4
                AngleY2 = y4

            if AngleX1 != 0 and AngleX2 !=0:
                break
```

```
if response.error.message:
    raise Exception(
        '{}\nFor more info on error messages, check: '
        'https://cloud.google.com/apis/design/errors'.format(response.error.message))

angle = degrees(atan2(AngleY1-AngleY2, AngleX2-AngleX1))
```

Then rotate image according to angle

Reade image

`img = cv2.imread(file)`

Rotate image using imutils function

`img = imutils.rotate(img, angle=360 - angle)`

Write rotated image as rotated.png

`cv2.imwrite(dir+"rotated.png",img)`

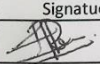
Rotated Image :

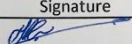





L 104

Signing Sheet

NSBM Green University Town
School of Computing
BSc (Hons) in Software Engineering - 2016.1

Cov.

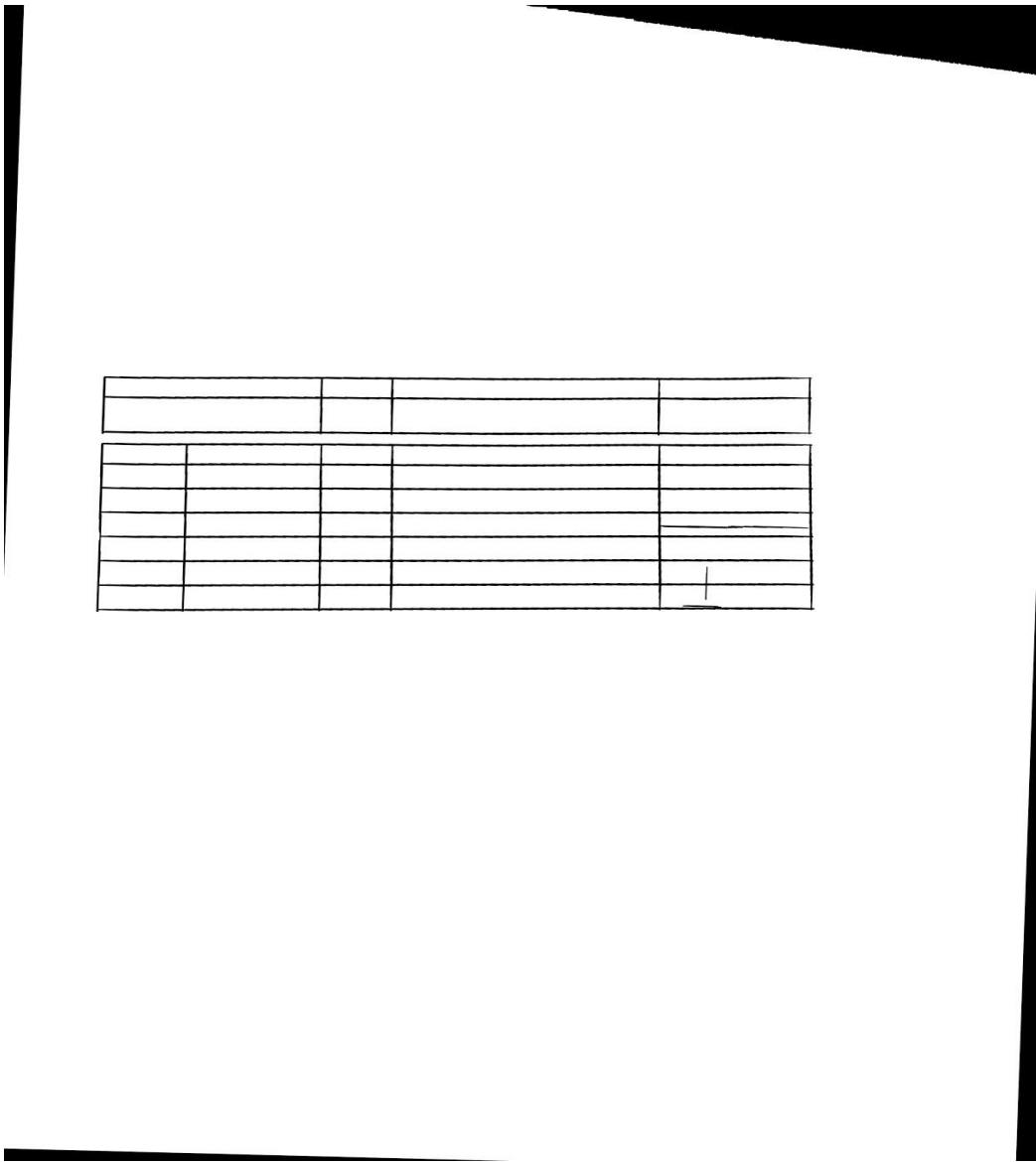
Date		Lecture's Name	Signature
05/07/2019	13.00 16.00	Dr. Rasika Ranaweera	

No	Student No	Title	Student Name	Signature
1	10000409	Ms	M S Dilshanika Perera	
2	10009301	Mr	C W M A Shehan Abeyrathne	
3	10009302	Mr	B A K M Chithrananda	
4	10009303	Ms	W Shashini Minosha De Silva	
5	10009304	Mr	K L Udara Maduranga Liyanage	
6	10009306	Mr	Hansa Anuradha Wickramanayake	

After rotation again identifies boxes to get new coordinates of ages. cells. that's why we use vhimg.

```
vhimg = cv2.imread(vhPath)
vhimg = imutils.rotate(vhimg, angle=360 - angle)
cv2.imwrite(dir+"img_vh.jpg", vhimg)
```

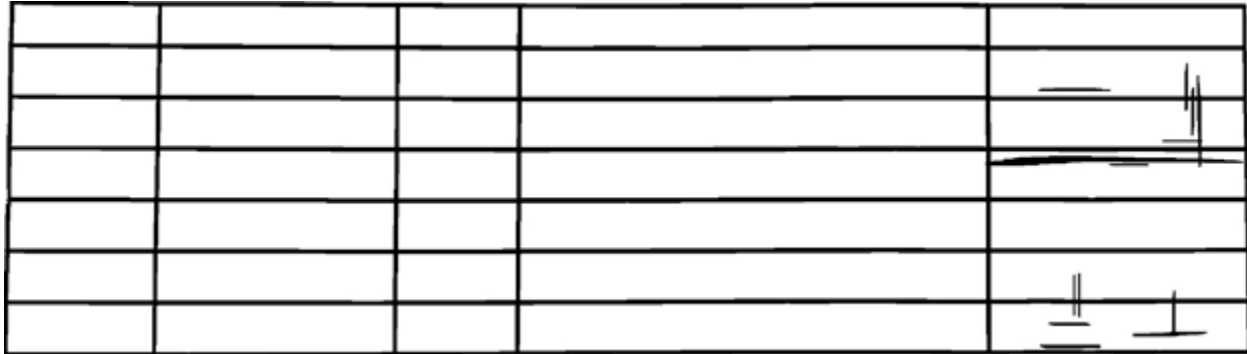
img_vh.jpg



Boxes

Boxes(img_vh_) : method is mainly created to separate and identify the each cell on the given table image and return two arrays.

Input : cropped main table image(this will contain only vertical and horizontal line)



Output : two arrays will be return

1. Binarized image array with replace 1 with 255.

```
[[ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 ...
 [255 255 255 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]
 [ 0  0  0 ... 0  0  0]]
```

2. Coordinates of the identified corners of the boxes with their mean values of x coordinates and Y coordinates (which can be use as coordinates of the centroid of identified boxes)

```
[[[(0, 0), (10, 0), (10, 465), (0, 465)], (5.70954356846473, 234.8941908713693)],
 [[(0, 0), (2157, 0), (2157, 612), (0, 612)], (1261.4228414608401, 320.57926957990287)],
 [[(15, 5), (267, 5), (267, 74), (15, 74)], (140.59968847352025, 40.01246105919003)],
 [[(272, 8), (678, 8), (678, 76), (272, 76)], (475.1951476793249, 41.802742616033754)],
 [[(684, 10), (889, 10), (889, 77), (684, 77)], (786.3290441176471, 43.59375)],
 [[(896, 10), (1699, 10), (1699, 78), (896, 78)], (1297.3352402745995, 43.98054919908467)],
 [[(1706, 11), (2144, 11), (2144, 77), (1706, 77)], (1924.6914682539682, 44.30853174603175)],
 [[(14, 79), (265, 79), (265, 162), (14, 162)], (139.1931137724551, 120.79191616766467)],
 [[(270, 81), (677, 81), (677, 164), (270, 164)], (473.5897959183674, 122.60612244897959)],
 [[(683, 83), (888, 83), (888, 165), (683, 165)], (785.432055749129, 124.27003484320558)],
 [[(895, 83), (1699, 83), (1699, 165), (895, 165)], (1297.0355530474042, 124.07167042889391)],
 [[(1706, 83), (2145, 83), (2145, 164), (1706, 164)], (1947.466560509554, 126.72133757961784)],
 [[(1790, 150), (1914, 150), (1914, 154), (1790, 154)], (1851.96484375, 152.27734375)],
 [[(12, 167), (264, 167), (264, 250), (12, 250)], (137.5776119402985, 208.97910447761194)],
 [[(269, 169), (677, 169), (677, 252), (269, 252)], (472.97556008146637, 210.5733197556008)],
 [[(894, 170), (1700, 170), (1700, 252), (894, 252)], (1298.0488215488215, 211.32323232323233)],
 [[(1706, 170), (2067, 170), (2067, 251), (1706, 251)], (1924.251290877797, 212.460413080895)],
 [[(683, 171), (888, 171), (888, 252), (683, 252)], (785.2762237762238, 211.66783216783216)],
 [[(2071, 171), (2147, 171), (2147, 251), (2071, 251)], (2108.272151898734, 211.65189873417722)].
```

```
img2 = ImageOps.grayscale(img_vh)
im = np.array(img2)
```

here we are converting incoming image into gray scale (img2). And save the image in format of array for manipulate later.

```
im = morphology.grey_dilation(im, (3, 3)) - im
```

in this line we are dilate and inverting the values of the pixels (like 1 to 254, like 255 to 0)

```
# Binarize.
mean, std = im.mean(), im.std()
t = mean + std
im[im < t] = 0
im[im >= t] = 1
```

Here we binarized our array to do that we use sum of mean and standard deviation value as a threshold value. After this line our array will only have 0's and 1's.

```
# Connected components.
lbl, numcc = label(im)
```

Label and find the number of boxes.

```
# Size threshold.
min_size = 200 # pixels
box = []
for i in range(1, numcc + 1):
    py, px = np.nonzero(lbl == i)
    if len(py) < min_size:
        im[lbl == i] = 0
        continue

    xmin, xmax, ymin, ymax = px.min(), px.max(), py.min(), py.max()
    # Four corners and centroid.
    box.append([
        (xmin, ymin), (xmax, ymin), (xmax, ymax), (xmin, ymax),
        (np.mean(px), np.mean(py))])
```

From this for loop we added the x, y coordinates and centroid coordinates of the boxes to an array called box[].

Identify word and signature

This part mainly focusing on identify words, segmenting, and set identified student number, title, student name, signature to 2 dimension array.

At the beginning this process need to get box which means 4 coordinates and centroid of each boxes.

```
for b, centroid in box:
    cx, cy = centroid
    x1, y1 = b[0]
    x2, y2 = b[1]
    x3, y3 = b[2]
    x4, y4 = b[3]
```

then loop through each boxes and assign edges and centroids to variables. So we can easily crop this segment using edges.

There a multiple sizes of segments comes with boxes, we only need cells of the tables, so we have to select specific sizes of segment. According to samples of signing sheets we select segment which difference between x3 and x1 less than 1000. So we can get what are the necessary segments. After that we saved it for further processing.

```
if x3 - x1 <= 1000:
    img3 = img.crop((x1, y1, x3, y3))
    path = dirSegments + str((x1, y1, x3, y3)) + ".jpg"
    img3.save(path)
```

Next step is to read those segments. First we used tesseract as optical character recognition tool but letter testing we found out lot of issues regarding with identifying character with that tool. So we had to move to Google OCR. It more intelligent than Tesseract. Lot of issues were solved from it.

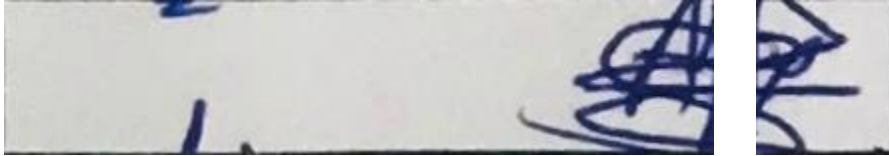
Then we read each boxes by calling Google Vision API.

```
client = vision.ImageAnnotatorClient()
with io.open(path, 'rb') as image_file:
    content = image_file.read()

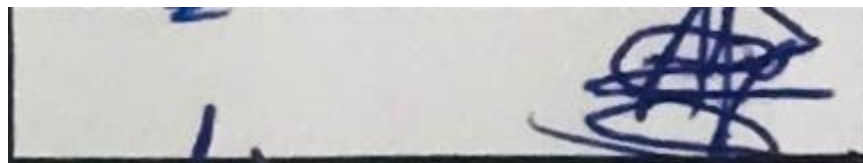
image = vision.types.Image(content=content)
response = client.text_detection(image=image)
texts = response.text_annotations
```

Next step we identify title. We skip titles which is not necessary to include in the array because it is already exist in the Database as column title. But in this step we need to do really necessary thing, it is to get x1 and x3 of Signature title. Student signatures aren't cropped current way by boxes algorithm due to unnecessary cross points. We need this width to crop student signatures. Height get from the student name segment's y1 and y3, from this 2 points, crop of signature can be done.

Before -



After -



```
if texts:
    #remove newLine
    texts[0].description = texts[0].description.rstrip('\n')

    if texts[0].description == "Student Name":
        #print("Student Name coordinate", x1,y1," - ", x2,y2," - ", x3,y3," - ",
x4,y4,)
        AngleX1 = x4
        AngleY1 = y4

    if texts[0].description == "No":
        #print("signature coordinate", x1, y1, " - ", x2, y2, " - ", x3, y3, " - ",
x4, y4, )

    if texts[0].description == "Signature":
        SignMinWidth = x1
        SignMaxWidth = x3
        AngleX2 = x4
        AngleY2 = y4
        countSignature = countSignature + 1

    if countSignature == 1:
```

Next step is go through the identified word and detect is it a student No, title or student name. Assign matrix column for that word or number. We got 0th column as signature, 1st column as student name, 2nd column as title and 3th column as student number.

```

if countSignature == 1:

    if (re.compile('[ABCDEFGHJKLMN]').search(format(texts[0].description)) != None
and len(texts[0].description) >= 6 and arrRowCount != 0):
        arrColCount = 1
        SignMinHeight = y1
        SignMaxHeight = y3
        img3 = img.crop((SignMinWidth, SignMinHeight, SignMaxWidth, SignMaxHeight))
        path = dirSegments+"\\sign" + str(
            (x1, y1, x3, y3)) + ".jpg"
        img3.save(path)
        #sign Recognition
        #sign Recognition
        #sign Recognition

    elif (re.compile('[Mrs]').search(format(texts[0].description)) != None and
len(texts[0].description) <= 3 and arrRowCount != 0):
        arrColCount = 2

    elif (re.compile('[1234567890]').search(format(texts[0].description)) != None and
len(texts[0].description) >= 7 and len(texts[0].description) <= 9 and arrRowCount !=
0):
        arrColCount = 3

    elif (re.compile('[1234567890]').search(format(texts[0].description)) != None and
len(texts[0].description) <= 3 and len(texts[0].description) >= 1 and arrRowCount !=
0):
        arrColCount = 4

```

Signature recognition step we use colour range to identify blue colour, cv2.inRange will create a mask (binary array) from that pixels, then use that mask to count number of white pixel. That count >1500 we detected as it was signed.

```

#sign Recognition
# Define a range for blue color for remove signatures
img3 = np.array(img3)
hsv = cv2.cvtColor(img3, cv2.COLOR_RGB2HSV)
# Define a range for blue color
hsv_l = np.array([100, 100, 0])
hsv_h = np.array([240, 255, 255])
# Find blue pixels in the image

mask = cv2.inRange(hsv, hsv_l, hsv_h)
img3[mask > 0] = (255, 255, 255)

cv2.imwrite(dir + 'SignColor identify' + str(
    (x1, y1, x3, y3)) + ".jpg", img3)

if np.sum(img3 == 255) > 1500:
    SignDetect = "true"
else:
    SignDetect = "false"

```

Mask :



Not assign items to 0th row matrix

```
if arrRowCount == 0:
    AttendentSheet[i][j] = "0"
    j = j + 1
    if j == 5:
        arrRowCount = 1
```

then set items to columns of matrix, somehow situations like Metrix -column is filled then next item also owns that Metrix -column then we detect that word need to have new Metrix-row. After that all new items goes to that row. In this step also set identified signature to matrix-column with student name.

```
else:
    for k in range(len(AttendentSheet)):
        if AttendentSheet[k][arrColCount] == 0:
            if arrColCount == 1:
                AttendentSheet[k][0] = SignDetect
                AttendentSheet[k][arrColCount] = format(texts[0].description)
                arrRowCount = k
                break
```

then select resize array to remove 0th matrix-row and send to database

```
#remove title column
AttendentSheet = np.delete(AttendentSheet, (0), axis=0)
#Fitted to filled size
AttendentSheet = np.resize(AttendentSheet, (arrRowCount+1,5))

insertRecord(AttendentSheet)
```

Output : [(873, '10000409', 'Ms', 'MS Dilshanika Perera', 'true'), (874, '10009301', 'Mr', 'CWM A Shehan Abeyrathne', 'false'), (875, '10009302', 'Mr', 'BAKM Chithrananda', 'true'), (876, '10009303', 'Ms', 'W Shashini Minosha De Silva', 'false'), (877, '10009304', 'Mr', 'KL Udara Maduranga Liyanage', 'true'), (878, '10009306', 'Mr', 'Hansa Anuradha Wickramanayake', 'true')]

Store attendance in database

The need to visualize real-time (or near real-time) data has been and still is the most important daily driver of many businesses. Microsoft SQL Server has a lot of capabilities to visualize streaming data and in this case, I will address this issue using Python. Also a python Dash package for building web applications and views. Dash is built on Flask, React and Plotly and offers a wide range of capabilities to build interactive web applications, visual and visual connectors.

SQLite is an information-related program. Without 'Lite' in the name it may contain more information than Terabyte 'Lite' part is really related to the fact that the system is 'empty bones'. Provides ways to create and query information with a simple command line interface but not much more. In SQL Study we used the Firefox plugin to provide the GUI (Graphical User Interface) in the SQLite data engine.

It is very simple and often very easy to think of SQL tables. All data manipulation, cutting, price editing, abuse and merging associated with SQL and SQL tables can all be accomplished. The SQL table can always be on the disc and once you have access to it, it is the SQLite data engine that does the job. This allows you to work with tables that are too large for your Python environment that may not have full capture memory at all.

A common use case for SQLite databases is to capture large data sets, using SQL commands from Python to cut and dice and perhaps integrate data within a database system to reduce the size of the Python object that can process it properly and restore results to the Data Center.

Receive data stored on SQLite using Python

We will demonstrate the use of the sqlite3 module by connecting to the SQLite database using both Python and using pandas.

Connects to SQLite database

The first thing we need to do is to import the sqlite3 library, we will import pandas at the same time for convenience.

```
import sqlite3
```

We'll start looking at the sqlite3 library by linking to an existing database and returning the query results.

The first thing we need to do is make a connection to the database. SQLite database is just a file. To make a connection to it we only need to use the sqlite3 connect () function and specify the data file as the first parameter.

```
with sqlite3.connect("AttendanceSheet.db") as db:
```

The next thing we need to do is create a connector cursor and give you a variable. We do this using the link point indicator method.

The interface allows us to transfer SQL statements to the database, processed and returned results. To create a SQL statement using the execute () of the cursor object. The only

parameter we need to pass to `execute()` is a string that contains the SQL query we wish to perform.

```
with sqlite3.connect("AttendanceSheet.db") as db:
    cursor = db.cursor()
    Cursor = db.cursor()

#cursor.execute("DELETE FROM sheet")
```

The `execute()` method does not actually return any data, it simply indicates that we want the data provided using the `SELECT` statement.

```
for i in range(len(AttendentSheet)):
```

The scope of the built-in function `range()` produces whole numbers between the first given number in the stop number. Using a loop, we can measure the sequence of numbers generated by a `range()` function.

In `for i in range()`, `i` iterator variable. To understand what `i` for `range()` means in Python, we first need to understand the `range()` function. The `range()` function uses a generator to generate numbers within a range, it does not create all the numbers at once. Produces the next value only if required for loop iteration. For each loop iteration, Python generates the next value and assigns the iterator variable `i`.

```
insertQuery = """INSERT INTO sheet(StudentNo,Title,StudentName,Signature)
VALUES(?,?,?,?);"""

data_tuple = (AttendentSheet[i][3], AttendentSheet[i][2], AttendentSheet[i][1],
AttendentSheet[i][0])
cursor.execute(insertQuery, data_tuple)
db.commit()
```

The SQL `INSERT INTO` statement is used to add new records to the table. While adding a record to a table, it is not mandatory to provide the total number of columns, we can only add a few columns using the `INSERT INTO` statement and the number of remaining columns to be set to work for that record.

And then insert a row of data and Save (commit) the changes.

```
cursor.execute("SELECT * FROM sheet")
print(cursor.fetchall())
```

Instead of fetching all rows (which may not be feasible), we can fetch row by row. We also fetch the column names.

Before we can apply the query results we need to use the `fetchall()` method for the cursor. The `fetchall()` method returns the list. Each item in the Tuple list contains values from a single row of tables. You can upgrade items in Tuple in the same way you would in a list.

Create Table

A SQLite table can be created on an in-memory database or on a disk file based database. Creating a table in SQLite involves first selecting a database file and loading it. And in SQLite, tables can be created in the in-memory databases as well. Here this is the table that we created for our project. The table has 5 columns as Id in integer, StudentsNo in varchar, Title in varchar, StudentName and Signature in varchar. And Id is the primary key of this table.

```
import sqlite3

with sqlite3.connect("AttendanceSheet.db") as db:
    cursor = db.cursor()

    cursor.execute('''
DROP TABLE sheet;
''')

    cursor.execute('''
CREATE TABLE IF NOT EXISTS sheet (
Id INTEGER PRIMARY KEY AUTOINCREMENT,
StudentNo VARCHAR(20) NOT NULL,
Title VARCHAR(20) NOT NULL,
StudentName VARCHAR(40) NOT NULL,
Signature VARCHAR(20) NOT NULL);
''')
```

Database content

[illegible]

Visualization

After identifying the student Id, student name or sign. It creates an array which saves as 2 dimensions for rows and columns. This array passes to the database and it stores information about the student. Using the following code we will create a graph to represent the attendance details that are identified.

In here, we retrieve student's data from the database and plot the graph in order to represent attendance details. It will display the total number of days that the relevant student attended the lectures according to their student id number.

```
cursor.execute("SELECT * FROM sheet")
print(cursor.fetchall())

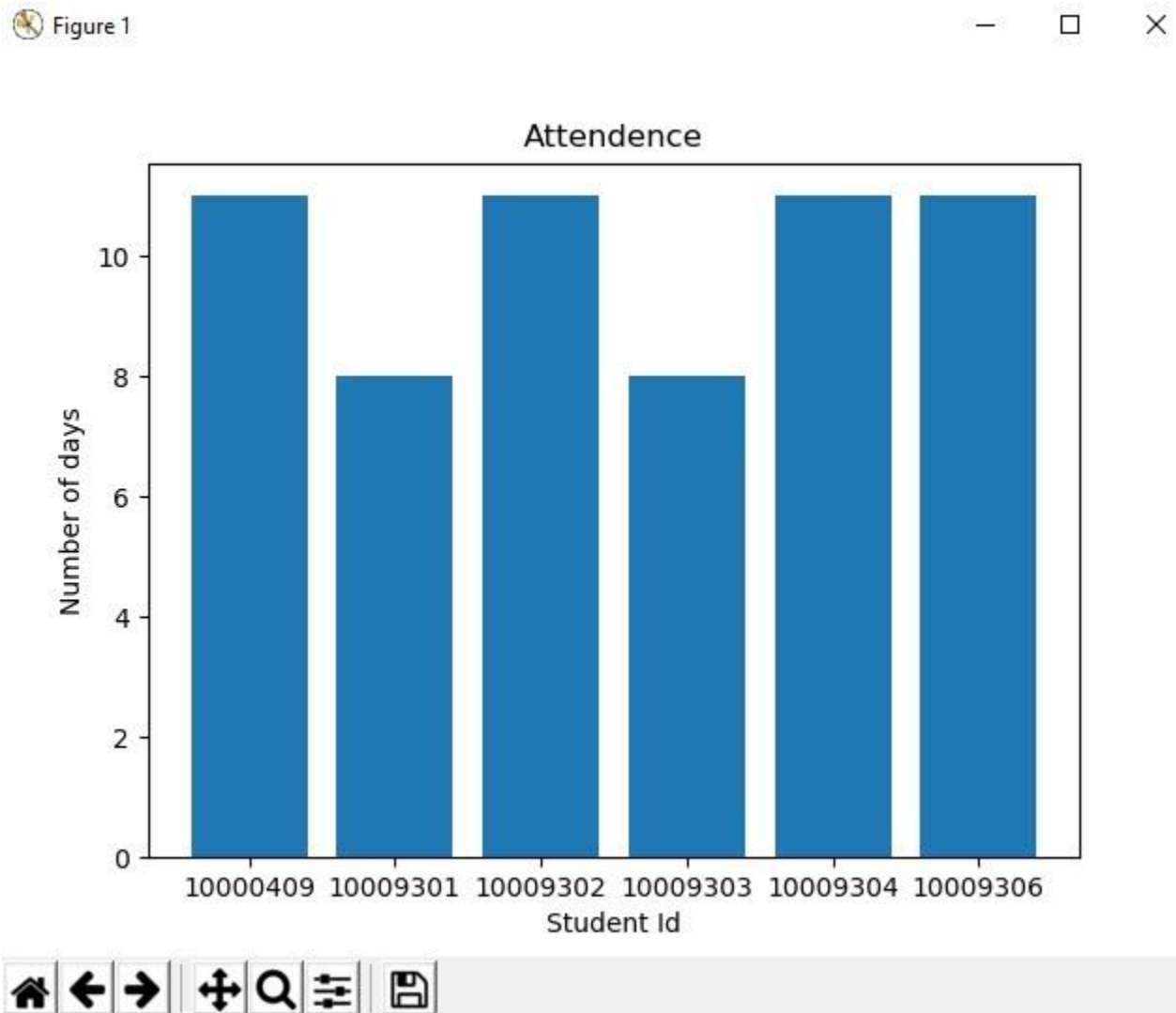
cursor.execute("SELECT StudentNo,COUNT(StudentNo) FROM sheet WHERE Signature = 'true'
GROUP BY StudentNo ORDER BY StudentNo")
rows = cursor.fetchall()

StudenId = []
Days = []
for row in rows:
    StudenId.append(row[0])
    Days.append(row[1])

fig, ax = plt.subplots()
ax.bar(StudenId, Days)
# set title and labels
ax.set_title('Attendance')
ax.set_xlabel('Student Id')
ax.set_ylabel('Number of days')

plt.show()
```

Following graph is displayed when executing the above code. The graph represents the attendance details of students.



Challengers faced.

First issue occurs when segmenting image, In this process there are unnecessary segment generated. So we fixed it by given a specific range of size to select cells of image.

Tesseract tool not word accurately to extract letters so we move to Google AIP, it is pretty much accurate.

We faced issues when crop signatures, because signatures are written out of the box. So we selected right size of that cell by referring other cells.

Words return from google API not in the correct sequence, so we had to create and algorithm to detect which word needed to be in the right column and row.

We had to rotate image to crop signature.