Empower-4

Abstract of the idea:

Using laboratory equipment and performing experiments becomes very challenging for the visually impaired. This write-up presents an attempt made by our team to ease this problem for them. This report presents an *Image Processing* based model which can inform the user about the liquid level filled in a beaker. It deploys some basic physics principles along with its implementation using OpenCV.

Background on the Problem Statement & How the Solution Addresses the Problem:

Whenever we talk of higher education, especially in science, we think first of the labs and scientific equipment. A majorly underrated problem is how the visually impaired are forced to be dependent even for taking readings using the most straightforward equipment, like a beaker. This vulnerability not only depreciates their self-confidence but also targets their independence. It results in a sharply falling number of people with special abilities in higher education.

Our solution will not only guide the visually impaired person when the beaker is filled, but it will also accurately tell the height of different liquid columns in the beaker.

Approach Taken to the Technology Problem Formulation and How the Technology Addresses the Real-Life Problem.

The main problem the people face is the lack of “vision”. They can’t see the readings they’ve taken. It’s also difficult for them to observe whether the liquid has started spilling out. So, it becomes very challenging for them to participate in the experiments. This model eases this for them, employing *Image Processing* techniques acting as their eyes. This model takes in the requisite amount as input and alerts the user when to stop.
Moreover, the gadget is straightforward and handy as it requires just striped cardboard and a camera (which might be the user's smartphone). The entire model is deployed on a plain Web App and enables the user access as and when required.

The description of the methodology of the solution.

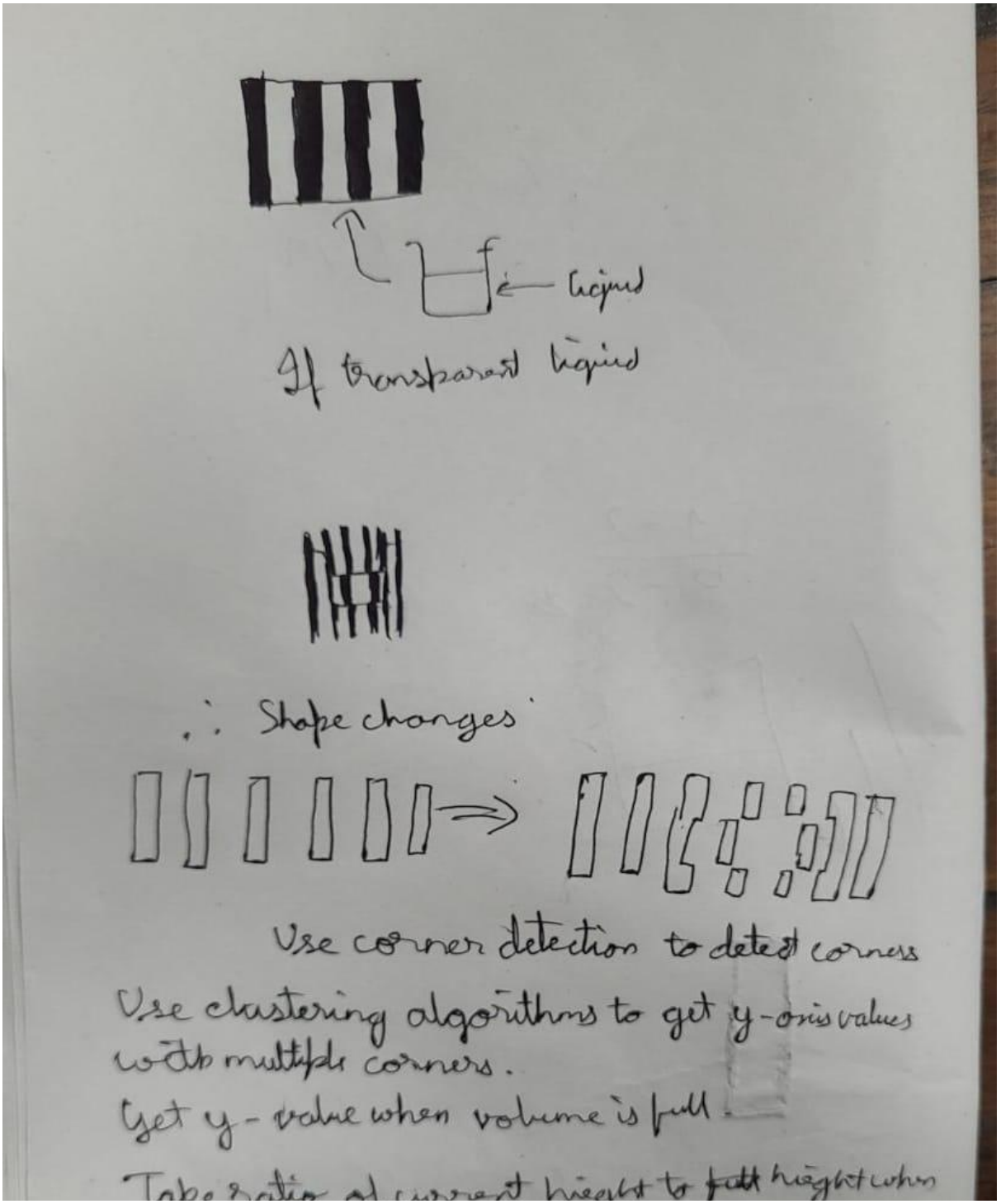
Using the different optical properties of an empty and a filled beaker

- The beaker, whose contents are to be measured, is placed in front of the striped board.

- The board has parallel stripes which will fall behind the beaker and due to refraction, the background will seem to be distorted when seen through the filled beaker.

- Camera (external/user's smartphone), placed on a stand in front of the beaker takes pictures with the striped background.

- The images taken, will pass through our model deployed on the website and the coordinates of the points with the bends would then be calculated using image processing techniques. Therefore, the volume of the liquid would be estimated.



Our Code :-

main2.cc > main()

```
import numpy as np
import cv2

j=0
listy=[]
size=[]
def getContours(img,img1):

    global j
    global listy
    global size
    contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    listy = []
    size = []
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 500:  # remove noise


            approx = cv2.approxPolyDP(cnt, 0.09 * cv2.arcLength(cnt, True), False)  # extract points from contour
            n = approx.ravel()
            i = 0
            #size.append(len(n))
            x=0
            for pt in n:
                if (i % 2 == 0):
                    listy.append(pt)
                else:
                    size.append(pt)
                i = i + 1
            """listy.append(maxy)
            size.append(x1)
            listy.append(miny)
            size.append(x2)"""

            #print(area)
            j+=1
            cv2.drawContours(img1, cnt, 2 , (0,255,0), 3)
            cv2.imshow("f1", img)

cap=cv2.VideoCapture(0)
if not cap.isOpened():
    print("cap not open")
    exit()

while True:


    success, img=cap.read()
    h = img.shape[0]
    w = img.shape[1]
    print(h)
    print(w)
    if not success:
        print("unable to get vid")
        break
    #cv2.imshow("Image", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    #cv2.imshow("i", img)
    hsv_min = (0, 0, 0)  # Lower end of the HSV range
    hsv_max = (100, 70, 150)  # Upper end of the HSV range

    # Transform image to HSV color space
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Threshold based on HSV values
    color_thresh = cv2.inRange(hsv, hsv_min, hsv_max)

```

```
    # Invert the image
    img1 = cv2.bitwise_not(color_thresh)
    #cv2.imshow("i", img)
    #img1=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kernel = np.ones((2,2), np.uint8)
    #img = cv2.erode(img, kernel, iterations=10)
    img1 = cv2.erode(img1, kernel, iterations=8)
    #img1=cv2.Canny(img1,50,150)
    getContours(img1,img)

    l=len(listy)
    l1=len(size)
    for a in range(0,l):
        cv2.circle(img, (listy[a],size[a]),radius=5,color=(0,0,255),thickness=-1)

    cv2.imshow("f", img)
    #cv2.imshow("img", img1)
    print("---------------")
    print(listy)
    print(l)
    print(size)
    print(l1)
    cv2.waitKey(1000)


#def calculate(listy):

"""gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

import cv2
import numpy as np

#img = cv2.imread('sofsk.png', 0)
size = np.size(img)
```

```

#def calculate(listy):

"""gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

import cv2
import numpy as np

#img = cv2.imread('sofsk.png', 0)
size = np.size(img)
skel = np.zeros(img.shape, np.uint8)

ret, img = cv2.threshold(img, 127, 255, 0)
element = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
done = False

while (not done):
    eroded = cv2.erode(img, element)
    temp = cv2.dilate(eroded, element)
    temp = cv2.subtract(img, temp)
    cv2.imshow("skel", skel)
    cv2.imshow("temp", temp)
    skel = cv2.bitwise_or(skel, temp)
    img = eroded.copy()

    """#zeros = size - cv2.countNonZero(img)
    #if zeros == size:
    #   done = True"""

"""cv2.imshow("skel", skel)
print("UwU")"""

cv2.waitKey(0)
```