ss project phase2 q1

Mr.Amirreza Zamani
Hosein Asghari Hosouri

July 4, 2023

1 Introduction

Hello. thank you so much for this interesting project. as an explanation about our Cooperation in brief. I did the first part related to Prof.Khalaj in MATLAB but because of a ridiculous error in cv2 library in Python i couldn't continue. The musical part and github part is done by effort of Mr.Zameni and I filled our report in Latex. the repository of github is:
github and the address of my latex files is: latex thank you so much .

2 Q1

We create and upload file Q1.m in MATLAB for finding Prof.Khalaj picture as template in background that both pictures is available in this latex . for solving this question We used two approaches

2.1 1'st approach

we ourselves write code:
the concept of template matching is as follows: we want to see where is the maximum resemblance between background image frequency content and template's .

for this concept we use correlation . it is similar to convolution except of we do not need to make time reversal into one if the signals. but we have a problem that the correlation answer is related to the amplitude of the signals and it affects on the output of correlation . to make the answer independent of amplitude we divide it by the amplitude of the signals and finally we introduce normal correlation . after that we find the place of maximum element and then insert a rectangle according to size of tempelate.
some notes that should be mentioned: finally we have a 2*2 matrix and we want to find the maximum element and insert a rectangle as big as our template . an important bug occurs during our code was that the template size has trash pixels that made it very large but we fix it by **"imcrop"** command.
the code is

```
clc;clear all;
b=(imread('background.JPG'));% read background image
t=(imread('template.JPG'));% read template image
[ir,ic]=size(b);%
[tr,tc]=size(t);
t=imcrop(t,[0,0,tr,tr]);% to fix this image to real size we need to omit some trash pixe
R=normxcorr2(t(:,:,3),b(:,:,3));% finding normal cross correlation
R=imcrop(R,[tr,tr,ic,ir]);% crop the extra pixels
[r,c,v]=find(R==max(max(R)));%finding coordinates where maximum correlation occurs
Result=insertShape(b,'Rectangle',[c,r,tr,tr],'LineWidth',5);%create a box around maximum
figure(1)
subplot(2,2,1)
imshow(b)
```

Figure 1: background

Figure 2: template

```
title('background')
subplot(2,2,2)
imshow(t)
title('template')
subplot(2,2,3)
imshow(R)
title('C')
subplot(2,2,4)
imshow(Result)
title('single shot detective')
```

finally our results in brief is as follows:

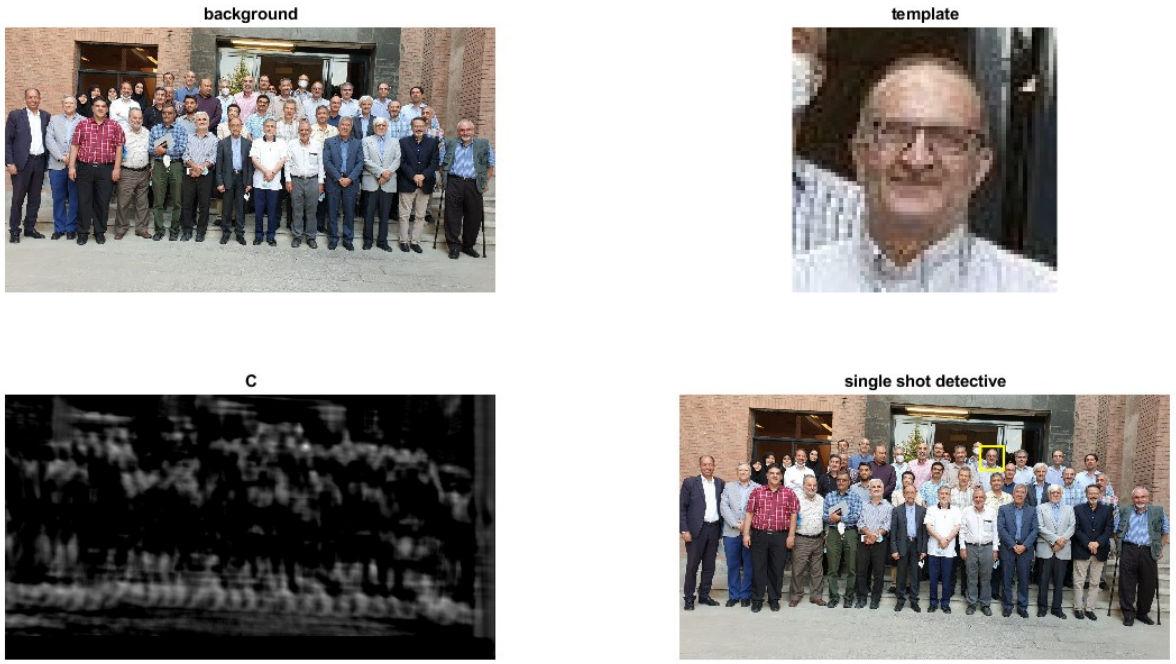


Figure 3: our main results in in down and right side as titled single shot detective

2.2 2'st approach

this part is additional
in this approach we use matlab defined function called as :

```
tMatcher=vision.TemplateMatcher;
x=tMatcher(b(:,:,3),t(:,:,3));
```

and it returns to us the center of Prof.Khalaj image and then we should insert a rectangle as big as size of template and concentric with Prof.Khalaj image. the code is:

```
clc;clear all;
b=(imread('background.JPG'));% read background image
t=(imread('template.JPG'));% read template image
[ir,ic]=size(b);%
[tr,tc]=size(t);
```

```
t=imcrop(t,[0,0,tr,tr]);% to fix this image to real size we need to omit some trash pixe
tMatcher=vision.TemplateMatcher;
x=tMatcher(b(:,:,3),t(:,:,3));
Result=insertShape(b,'Rectangle',[x(1),x(2),tr,tr],'LineWidth',5);%create a box around m
figure(1)
imshow(Result)
```

ss project phase2 q2

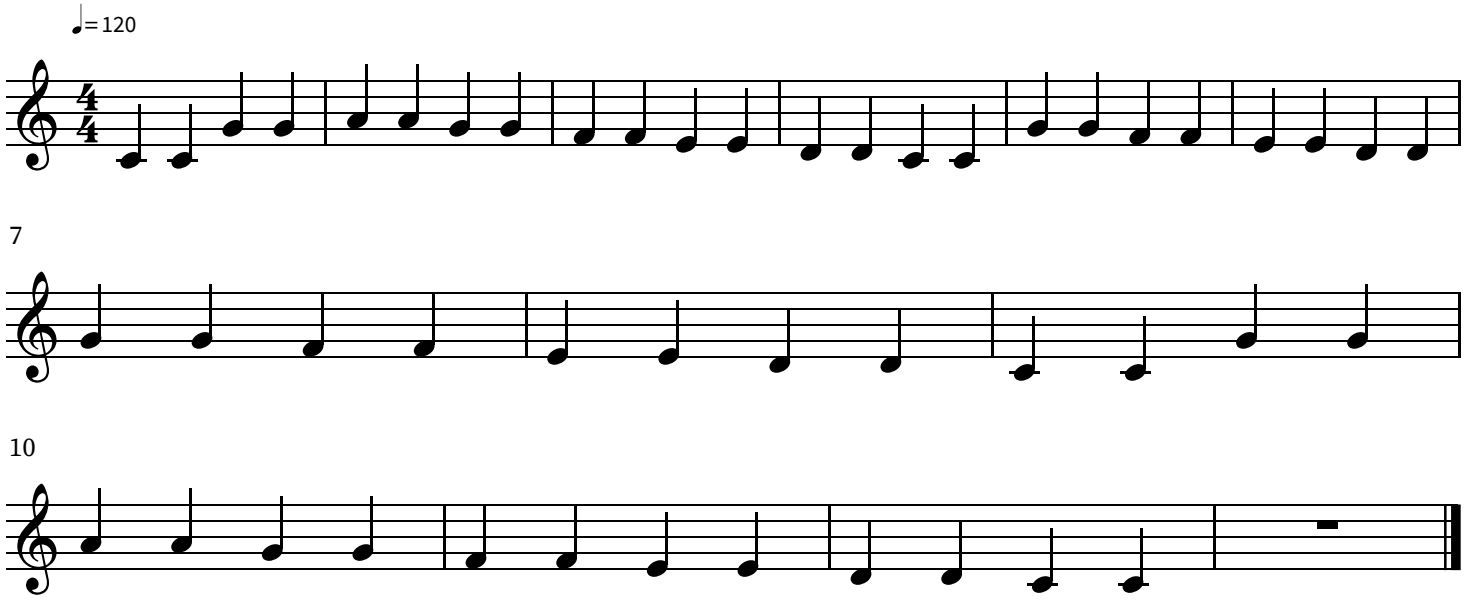
Hosein Asghari Hosouri
Mr.Amirreza Zamani

July 4, 2023

1 Q2

In this question we should implement Twinkle, "Twinkle, Little Star". it means we have to convert it from sheet music into music by template matching method the related music sheet is attached below:

Twinkle, Twinkle, Little Star



some additional information about this music sheet:
as you can see,we only have Crotchet that means 1x and we have 120BPM and 7 means that our blocking of notes(kind of assortment) starts at 7 for better expression look at the the illustration below:

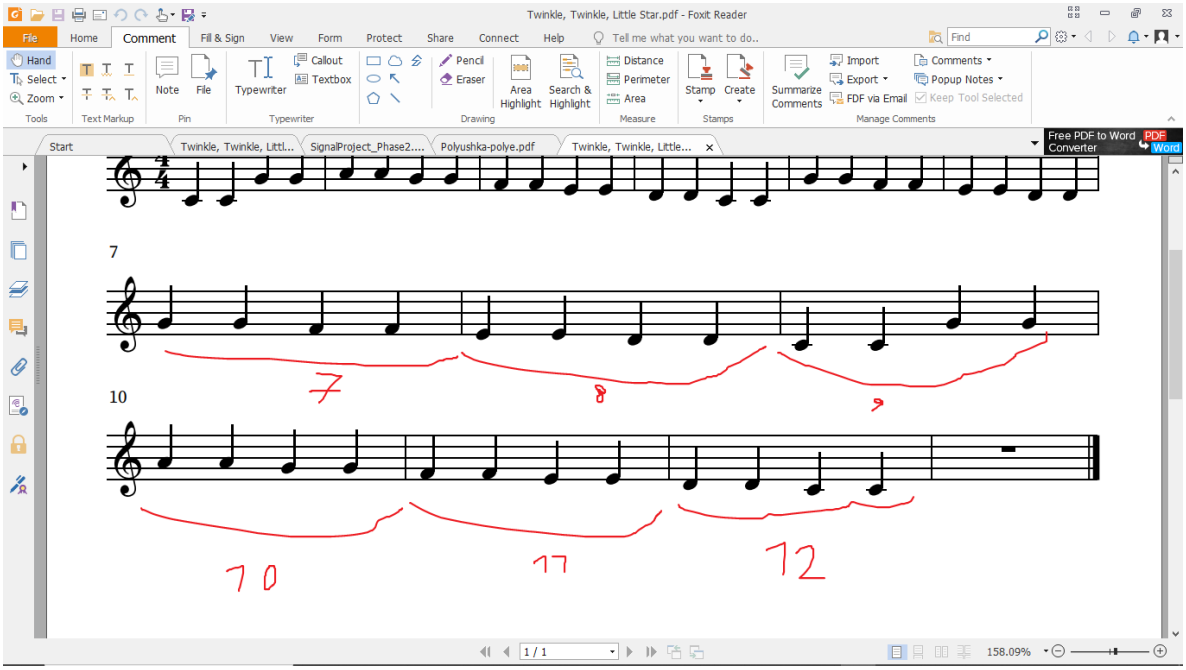


Figure 1: blocking

to prescribe this code :
1- we defined a function named as "LineDetector(im-input)" that recieves an image from imread and find the horizontal lines.
2-then we defined a function named as "detectNotes(img, template, threshold)" that recieves main image and then template that we want to find it throughout the main image(we cropped the musicsheet and find quarter as template) with a coefficient of sensitivity called "threshold" that the smaller it is the more sensitive our function is . so we have to determine it by iterative method for best performance and in this sample we set it as 0.69.

Figure 2: quarter

3- we defined two objects named as "note" and "blunt". **blunt** seperated musical sets(in fact each five line is a musical set) and includes either notes , coordinates of their lines or a function named "findNotesStr(self)" that correspond the lines and their own lines . and finally assigns each notes an **ID** and a parameter that determines their pitch modulation.

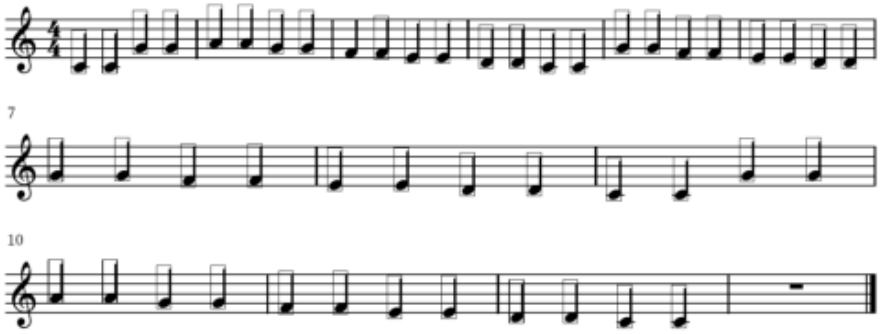


Figure 3: output

4- we defined **make notes** that it merges notes together. at first by means of the lineDetector findes vertical coordinates,y, and then ceates **"bluntN=int(len(lines)/5)"**(due to each five line is a musical set) and we have a listarray named "blunts" and an auxiliary variable named "temp-blunts" and by using a for loop we fill out "blunts" if we want to understand that weather a note is included of the blunt or not , it should be around its midline. to determine midline we average 5 line inside a blunt and find **temp-blunt.average** (then we compare the distance between the center of note and midline of each blunt is less than 75(a parameter used for this question) then this notes belongs to this blunt). for example in this question we only need to investigate Crotchet as template. so we find in throughout the Twinkle and find its rects(x ,y coordinates) and for each rect it generates a new notes and save new coordinates and because it is left and up coordinates so we need to make y,61 pixel downward and x,13 pixel forward(according to the size of "quarter" as template" . finally we call findNotesStr() in the class of blunt. in this question we only have 5 lines with one line above and one line below of them that means totally we have 7 lines on the other hand , uppermost line that is used is +1 and the lowermost is somewhere between -1 and -2 . it says we must only show 14 notes.then according to notelevel definition it determine notelevel for every positions between lines and saves them(by the way HLD(half line distance) is average of distance between lines in a blunt).then by indexing we try to find which level is closer to our notes, and the closest one is our level. and then by means of an if statement we determine the octave and ID of each notes.
5- so far , we have some blunts and in each blunt some notes exist with determined ID octave. so we only need the order of notes. as you know we identify the order of each note according to its x coordinate . by using **sort function of python** we sort them according to their own x coordinate and simultaneously it generate a **" music-to-play"** by three parameters (**"[id,octave,duration]"**)

by means of "helper" that was given to us we developed our code as follows:

```
import pydub
import numpy as np
import sounddevice as sd
from matplotlib import pyplot as plt
from scipy.io.wavfile import read, write
import cv2 as cv
import pandas as pd
```

```
# read image
TwinkleIm = cv.imread('Twinkle.png')
# detect lines
def LineDetector(input_im):
# convert image to gray image
    Gray_IM = cv.cvtColor(input_im,cv.COLOR_BGR2GRAY)
# detect edges
    t_lower = 50  # Lower Threshold
    t_upper = 200  # Upper threshold
#apertureSize is an optional parameter that is used to specify the order of the Sobel fi
#to calculate the gradient in the Canny algorithm. The default value is 3 and its value
#be odd between 3 and 7. You can increase the Aperture size when you want to detect more
    edges = cv.Canny(Gray_IM,t_lower,t_upper,apertureSize=3)

    IM_lines = cv.HoughLinesP(
                edges,
                1, # Distance resolution in pixels
                np.pi/180, # Angle resolution in radians
                threshold=50, # Min number of votes for valid line
                minLineLength=40, # Min allowed length of line
                maxLineGap=10 # Max allowed gap between line for joining them
                )
    lines_Y=[]
    for line in IM_lines:
        xx,ys,xf,yf=line[0]
        if(abs(xx-xf)>50):
            if(abs(ys-yf)<5):
                a=1
                for i in lines_Y:
                      if(abs(i-ys)<5):
                        a=0
                if(a or lines_Y==[]):
                    lines_Y.append(ys)
    for i in lines_Y:
        cv.line(Gray_IM, (0, i), (1600, i), (0,0,255), 2, cv.LINE_AA)
    lines_Y.sort()
    return Gray_IM,lines_Y

test_IM=TwinkleIm
Detected_IM,linesY=LineDetector(test_IM)
_ = plt.imshow(Detected_IM[:,:])
#_ = plt.xticks([])
#_ = plt.yticks([])
#_ = plt.box(0)
print(linesY)
###################
def detectNotes(input_image, template, threshold):
    w, h = template.shape[::-1]
    res_image = input_image.copy()
    matches = cv.matchTemplate(input_image, template, cv.TM_CCOEFF_NORMED)
    match_loc = np.where( matches >= threshold)
    rects = list(zip(*match_loc[::-1]))
    rects = sorted(rects, key = lambda x: (x[0], x[1]))
    rects = np.asarray(rects)
    prev_x = rects[0][0]
    prev_y = rects[0][1]
```

```
    x_margin = 20
    y_margin = 20
    stay = [0]
    for i in range(1, rects.shape[0]):
        if (abs(rects[i][0] - prev_x) > x_margin or abs(rects[i][1] - prev_y) > y_margin
            stay.append(i)
            prev_x = rects[i][0]
            prev_y = rects[i][1]

    rects = rects[stay]
    rect2=[]
    for i in rects:
       # if(rect2==[]):
        #    rect2.append(i)
        a=1
        for j in range (len(rect2)):
            if(    ((i[0]-rect2[j][0])**2)+((i[1]-rect2[j][1])**2)<20  ):
              a=0
        if(a):
            rect2.append(i)
        for pt in rects:
            cv.rectangle(res_image, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)

    return rect2,res_image
###################
template = cv.imread("quarter (1).PNG", cv.IMREAD_GRAYSCALE)
gray = cv.cvtColor(TwinkleIm, cv.COLOR_BGR2GRAY)
threshold = 0.69

rects, img2 = detectNotes(gray, template, threshold)

print(len(rects))
_ = plt.imshow(img2, cmap = plt.cm.gray)
_ = plt.xticks([])
_ = plt.yticks([])
_ = plt.box(0)
################
# note class
class note :
    def __init__(self):
        self.x=0
        self.y=0
        self.it=0
        self.octave=0
        self.dur=0
class blunt :
    def __init__(self):
        self.lines=[]
        self.average=0
        self.notes=[]
    def findNotesStr(self):
       notelevel=[
            self.lines[0]-15,
            self.lines[0]-7,
            self.lines[0],
            self.lines[1]-7,
```

```
        self.lines[1],
        self.lines[2]-7,
        self.lines[2],
        self.lines[3]-7,
        self.lines[3],
        self.lines[4]-7,
        self.lines[4],
        self.lines[4]+7,
        self.lines[4]+15,
        self.lines[4]+23
    ]
    index=pd.Index(notelevel)
    for note in self.notes:
         a=index.get_indexer([note.y], method="nearest")
         if(a==0):
            note.octave=5
            note.id=0
         elif (a==1):
            note.octave=5
            note.id=10
         elif (a==2):
            note.octave=5
            note.id=8
         elif (a==3):
            note.octave=5
            note.id=7
         elif (a==4):
            note.octave=5
            note.id=5
         elif (a==5):
            note.octave=5
            note.id=3
         elif (a==6):
            note.octave=5
            note.id=2
         elif (a==7):
            note.octave=4
            note.id=0
         elif (a==8):
            note.octave=4
            note.id=10
         elif (a==9):
            note.octave=4
            note.id=8
         elif (a==10):
            note.octave=4
            note.id=7
         elif (a==11):
            note.octave=4
            note.id=5
         elif (a==12):
            note.octave=4
            note.id=3
         elif (a==13):
            note.octave=3
            note.id=2
```

```
a=note()
a.y=10
a.x=5
print(a.x+a.y)
###########
import pandas as pd
print(min([2,3,4]))
#############
#detect notes
img = cv.imread('Twinkle.png')
def MakeNotes(input_img):
    result=[]
    ## line
    junk1,lines =LineDetector(input_img)
    # create blunts

    bluntN=int(len(lines)/5)
    blunts=[]
    for i in range (bluntN):
        temp_blunt=blunt()
        temp_blunt.lines=[lines[5*i],lines[5*i+1],lines[5*i+2],lines[5*i+3],lines[5*i+4]
        temp_blunt.average=sum(temp_blunt.lines)/len(temp_blunt.lines)
        print("Average",temp_blunt.average)
        blunts.append(temp_blunt)

    ## note
    template = cv.imread("quarter (1).PNG", cv.IMREAD_GRAYSCALE)
    gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.69
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+13
        temp_note.y=point[1]+61
        temp_note.dur=3
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
    for i in blunts:
        print(len(i.notes))
        i.findNotesStr()
        return image2,blunts


out,blunts=MakeNotes(img)
_ = plt.imshow(out, cmap = plt.cm.gray)
music_to_play=[]
for i in blunts:
    i.notes.sort(key=lambda x: x.x, reverse=False)
    for j in i.notes:
        music_to_play.append([j.id,j.octave,j.dur])
final=get_music(music_to_play,44100)
```

```
len(music_to_play)
sd.play(final)
## play function
notes_base = 2**(np.arange(12)/12)*27.5
notes_duration = np.array([3200, 1600, 800, 400, 200, 100])*0.7
notes_ann = ['A', 'A#', 'B', 'C', 'C#', 'D', 'Eb', 'E', 'F', 'F#', 'G', 'G#']

def sin_wave(f, n, fs):
    x = np.linspace(0, 2*np.pi, n)
    ring = 30
    xp = np.linspace(0, -1*(n*ring/fs), n)
    y = np.sin(x*f*(n/fs))*np.exp(xp)
    z = np.zeros([n, 2])
    z[:, 0] = y
    z[:, 1] = y
    return z

def play_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    sd.play(y, fs)
    sd.wait()
    return

def put_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    return y

def get_music(music_notes, fs):
    m = []
    for item in music_notes:
        y = put_note(item[0], item[1], item[2], fs)
        m.append(y)
    m = np.concatenate(m, 0)
    return m

fs1 = 44100
music = [[8, 5, 3], [10, 5, 4], [10, 5, 4], [10, 5, 3], [10, 5, 3], [10, 5, 3], [10, 5,
        [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4],
        [8, 5, 4], [8, 5, 4], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8,
        [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4],

y = get_music(music, fs1)
sd.play(y, fs1)
##
Scale = [[3,4,3], [5,4,3], [7,4,3], [8,4,3], [10,4,3], [0,4,3], [2,4,3], [3,5,3],
        [2,4,3], [0,4,3], [10,4,3], [8,4,3], [7,4,3], [5,4,3], [3,4,3]]

y = get_music(Scale, fs1)
sd.play(y, fs1)
write("twinkle.wav", 44100, final.astype(np.float32))
```

ss project phase2 q2

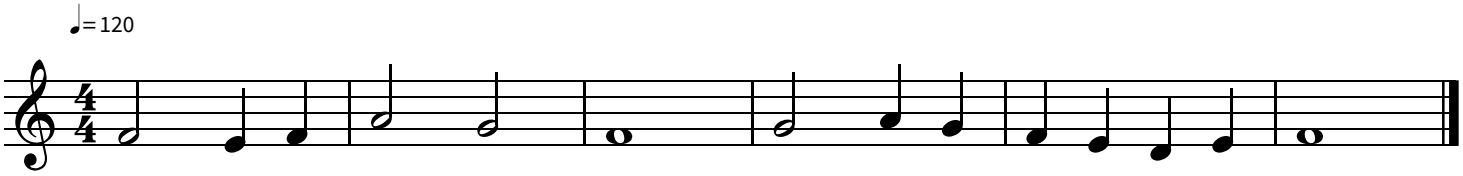
Hosein Asghari Hosouri
Mr.Amirreza Zamani

July 4, 2023

1 Q3

In this question we should implement "Ave Maria". it means we have to convert it from sheet music into music by template matching method the related music sheet is attached below:

Ave Maria



some additional information about this music sheet:

as you can see,we have Crotchet ,Minim ,Semibreve that means 1x ,2x ,4x .and we have 120BPM . we follow previous approach but in this sample we add some new notes with different distances .the main changes occur in "get-music function" we must add the related template of ,Minim ,Semibreve too.for more explanation we must consider duration of each note. because in the previous section we have only one type of note but here we have three types with different duration . furthermore we need to check for each template new appropriate coefficients of threshold and we found the magnitude of displacement with respect to found rects that in this sample is 63 for y and 14 for x.
by means of previous section we developed our code as follows:

```
import pydub
import numpy as np
import sounddevice as sd
from matplotlib import pyplot as plt
from scipy.io.wavfile import read, write
import cv2 as cv
import pandas as pd

###############
# read image
TwinkleIm = cv.imread('Ave Maria.png')
# detect lines
def LineDetector(input_im):
# convert image to gray image
    Gray_IM = cv.cvtColor(input_im,cv.COLOR_BGR2GRAY)
# detect edges
    t_lower = 50  # Lower Threshold
    t_upper = 200  # Upper threshold
#apertureSize is an optional parameter that is used to specify the order of the Sobel fi
#to calculate the gradient in the Canny algorithm. The default value is 3 and its value
#be odd between 3 and 7. You can increase the Aperture size when you want to detect more
    edges = cv.Canny(Gray_IM,t_lower,t_upper,apertureSize=3)

    IM_lines = cv.HoughLinesP(
                edges,
                1, # Distance resolution in pixels
                np.pi/180, # Angle resolution in radians
                threshold=50, # Min number of votes for valid line
                minLineLength=40, # Min allowed length of line
                maxLineGap=10 # Max allowed gap between line for joining them
                )
    lines_Y=[]
    for line in IM_lines:
        xx,ys,xf,yf=line[0]
        if(abs(xx-xf)>50):
            if(abs(ys-yf)<5):
                a=1
                for i in lines_Y:
                    if(abs(i-ys)<5):
                        a=0
                if(a or lines_Y==[]):
                    lines_Y.append(ys)
    for i in lines_Y:
        cv.line(Gray_IM, (0, i), (1700, i), (0,0,255), 2, cv.LINE_AA)
    lines_Y.sort()
    return Gray_IM,lines_Y
```

```
    ####################
    def detectNotes(input_image, template, threshold):
    w, h = template.shape[::-1]
    res_image = input_image.copy()
    matches = cv.matchTemplate(input_image, template, cv.TM_CCOEFF_NORMED)
    match_loc = np.where( matches >= threshold)
    rects = list(zip(*match_loc[::-1]))
    rects = sorted(rects, key = lambda x: (x[0], x[1]))
    rects = np.asarray(rects)
    prev_x = rects[0][0]
    prev_y = rects[0][1]
    x_margin = 20
    y_margin = 20
    stay = [0]
    for i in range(1, rects.shape[0]):
        if (abs(rects[i][0] - prev_x) > x_margin or abs(rects[i][1] - prev_y) > y_margin
            stay.append(i)
            prev_x = rects[i][0]
            prev_y = rects[i][1]

    rects = rects[stay]
    rect2=[]
    for i in rects:
       # if(rect2==[]):
        #    rect2.append(i)
        a=1
        for j in range (len(rect2)):
            if(    ((i[0]-rect2[j][0])**2)+((i[1]-rect2[j][1])**2)<20  ):
              a=0
        if(a):
            rect2.append(i)
        for pt in rects:
            cv.rectangle(res_image, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)

    return rect2,res_image
    #############
    Ave_image = cv.imread('Ave.png',cv.IMREAD_GRAYSCALE)
Ave_image1 = cv.imread('Ave.png')
template2 = cv.imread("whole.PNG", cv.IMREAD_GRAYSCALE)
threshold2 = 0.55

rects_part2, img3 = detectNotes(Ave_image, template2, threshold2)
linesim,lines=LineDetector(Ave_image1)
print(len(rects_part2))
for i in rects_part2:
    cv.circle(Ave_image1, (i[0]+14,i[1]+11), radius=0, color=(0,0 , 255), thickness=2)

_ = plt.imshow(Ave_image1, cmap = plt.cm.gray)
_ = plt.xticks([])
_ = plt.yticks([])
_ = plt.box(0)
```

so far the result is as follows:

rest of code :

```
# note class
```

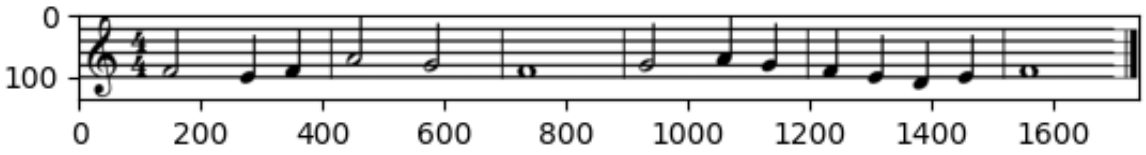


Figure 1: detect Ave lines

```
class note :
    def __init__(self):
        self.x=0
        self.y=0
        self.id=0
        self.octave=0
        self.dur=0
class blunt :
    def __init__(self):
        self.lines=[]
        self.average=0
        self.notes=[]
    def findNotesStr(self):
       HLD=int(np.rint((self.lines[4]-self.lines[0])/8))
       notelevel=[
            self.lines[0]-2*HLD,
            self.lines[0]-HLD,
            self.lines[0],
            self.lines[1]-HLD,
            self.lines[1],
            self.lines[2]-HLD,
            self.lines[2],
            self.lines[3]-HLD,
            self.lines[3],
            self.lines[4]-HLD,
            self.lines[4],
            self.lines[4]+HLD,
            self.lines[4]+2*HLD,
            self.lines[4]+(4*HLD+1)
             ]
       index=pd.Index(notelevel)
       for note in self.notes:
           a=index.get_indexer([note.y], method="nearest")
           if(a==0):
               note.octave=5
               note.id=0
           elif (a==1):
               note.octave=5
               note.id=10
           elif (a==2):
               note.octave=5
               note.id=8
           elif (a==3):
               note.octave=5
               note.id=7
           elif (a==4):
               note.octave=5
```

```
                note.id=5
            elif (a==5):
                note.octave=5
                note.id=3
            elif (a==6):
                note.octave=4
                note.id=2
            elif (a==7):
             note.octave=4
                note.id=0
            elif (a==8):
                note.octave=4
                note.id=10
            elif (a==9):
                note.octave=4
                note.id=8
            elif (a==10):
                note.octave=4
                note.id=7
            elif (a==11):
                note.octave=4
                note.id=5
            elif (a==12):
                note.octave=4
                note.id=3
            elif (a==13):
                note.octave=3
                note.id=2


a=note()
a.y=10
a.x=5
print(a.x+a.y)
#detect notes2
img2 = cv.imread('Ave.png')
def MakeNotes2(input_img):
    result=[]
    ## line
    junk1,lines =LineDetector(input_img)
    # create blunts

    bluntN=int(len(lines)/5)
    blunts=[]
    for i in range (bluntN):
        temp_blunt=blunt()
        temp_blunt.lines=[lines[5*i],lines[5*i+1],lines[5*i+2],lines[5*i+3],lines[5*i+4]
        temp_blunt.average=sum(temp_blunt.lines)/len(temp_blunt.lines)
        print("Average",temp_blunt.average)
        blunts.append(temp_blunt)

    ## note
    ## quarters
    template = cv.imread("quarter (1).PNG", cv.IMREAD_GRAYSCALE)
    gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.69
```

```
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+14
        temp_note.y=point[1]+63
        temp_note.dur=3
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
              i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,

    ## half
    template = cv.imread("half.PNG", cv.IMREAD_GRAYSCALE)
   # gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.75
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+14
        temp_note.y=point[1]+69
        temp_note.dur=2
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
    ## whole
    template = cv.imread("whole.PNG", cv.IMREAD_GRAYSCALE)
    threshold = 0.55
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+14
        temp_note.y=point[1]+11
        temp_note.dur=1
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,

    for i in blunts:
        print(len(i.notes))
        i.findNotesStr()




    return image2,blunts


out2,blunts2=MakeNotes2(img2)
_ = plt.imshow(out2, cmap = plt.cm.gray)
music_to_play2=[]
for i in blunts2:
```

```
    i.notes.sort(key=lambda x: x.x, reverse=False)
    for j in i.notes:
        music_to_play2.append([j.id,j.octave,j.dur])
final2=get_music(music_to_play2,44100)
len(music_to_play2)
sd.play(final2)
####################
notes_base = 2**(np.arange(12)/12)*27.5
notes_duration = np.array([3200, 1600, 800, 400, 200, 100])*0.7
notes_ann = ['A', 'A#', 'B', 'C', 'C#', 'D', 'Eb', 'E', 'F', 'F#', 'G', 'G#']

def sin_wave(f, n, fs):
    x = np.linspace(0, 2*np.pi, n)
    ring = 30
    xp = np.linspace(0, -1*(n*ring/fs), n)
    y = np.sin(x*f*(n/fs))*np.exp(xp)
    z = np.zeros([n, 2])
    z[:, 0] = y
    z[:, 1] = y
    return z

def play_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    sd.play(y, fs)
    sd.wait()
    return

def put_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    return y
    def get_music(music_notes, fs):
    m = []
    for item in music_notes:
        y = put_note(item[0], item[1], item[2], fs)
        m.append(y)
    m = np.concatenate(m, 0)
    return m

fs1 = 44100
music = [[8, 5, 3], [10, 5, 4], [10, 5, 4], [10, 5, 3], [10, 5, 3], [10, 5, 3], [10, 5,
        [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4],
        [8, 5, 4], [8, 5, 4], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8,
        [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4],

y = get_music(music, fs1)
sd.play(y, fs1)
write("Ave_result.wav", 44100, final2.astype(np.float32))
```

ss project phase2 q4

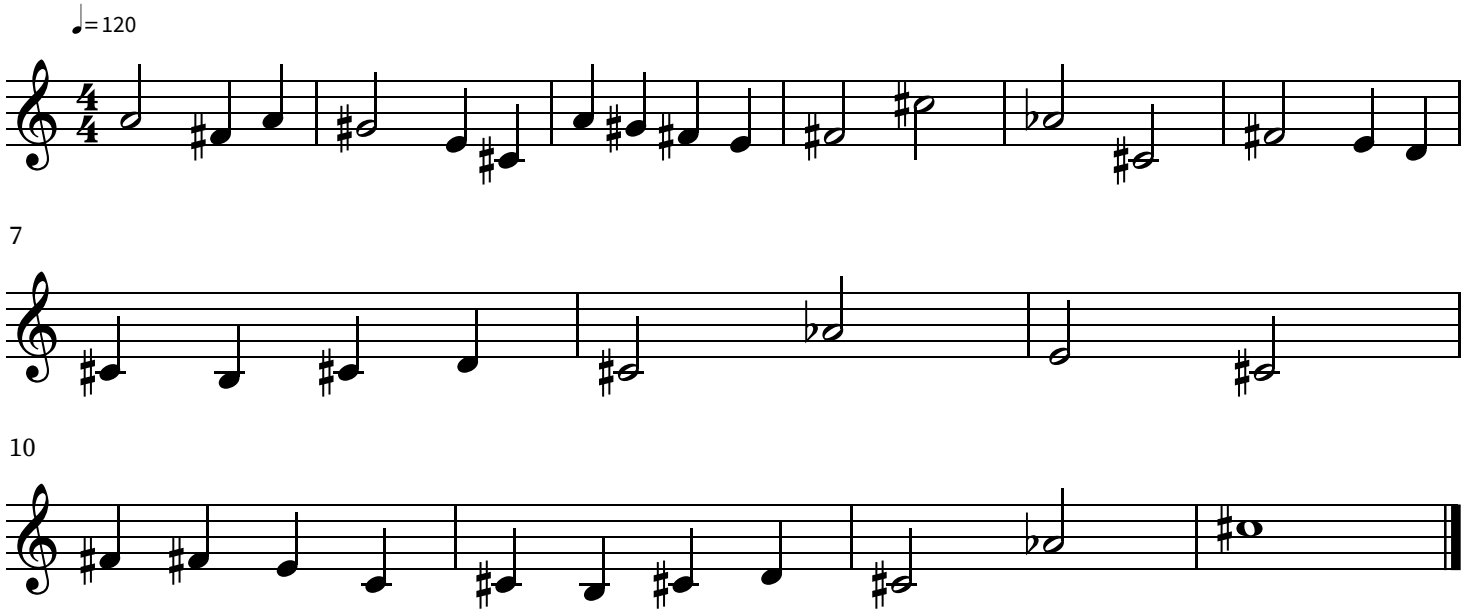
Hosein Asghari Hosouri
Mr.Amirreza Zamani

July 4, 2023

1 Q4

In this question we should implement "Polyushka-Ploye". it means we have to convert it from sheet music into music by template matching method the related music sheet is attached below:

Polyushka-polye



some additional information about this music sheet:
as you can see,we have Crotchet ,Minim ,Semibreve that means 1x ,2x ,4x and diese and bemol .and we have 120BPM and 7 means that our blocking of notes(kind of assortment) starts at 7 . we follow previous approach but in this sample we add some new notes with different distances such as diese , bemol and some stem down notes and b new generated templates we detected them .the main changes occur in "get-music function" we must add the related template of diese , bemol and some stem down notes too.for more explanation we must consider duration of each note. because in the previous section we have only one type of note but here we have three types with different duration . furthermore we need to check for each template new appropriate coefficients of threshold and we found the magnitude of displacement with respect to found rects that in this sample is 62 for y and 14 for x.
as well,we defined new calsses for diese and bemel in the name of **" DorB"** that has three parameter such as "x , y , and dboolean(when it is 1 means diese and when it is 0 means bemol.
when we are grouping our note we add new classes diese and bemol.when diese and bemol notes went to theri blunt we add a function(called **"AddExtraToNote()"**) that finds the closest note to the diese or bemol note and make that not one fourth or half pitch forward and relate that note to the diese or bemol.acctually this function adopts two different approaches . for diese notes 1–¿ 2 , 2–¿3 , 3–¿4 and ...,10–¿11 and finally 11–¿0 with an octave higher. for bemol this approach is exactly reversed.
by means of previous section we developed our code as follows:

```
import pydub
import numpy as np
import sounddevice as sd
from matplotlib import pyplot as plt
from scipy.io.wavfile import read, write
import cv2 as cv
import pandas as pd

def detectNotes(input_image, template, threshold):
    w, h = template.shape[::-1]
    res_image = input_image.copy()
    matches = cv.matchTemplate(input_image, template, cv.TM_CCOEFF_NORMED)
    match_loc = np.where( matches >= threshold)
    rects = list(zip(*match_loc[::-1]))
    rects = sorted(rects, key = lambda x: (x[0], x[1]))
    rects = np.asarray(rects)
    prev_x = rects[0][0]
    prev_y = rects[0][1]
    x_margin = 20
    y_margin = 20
    stay = [0]
    for i in range(1, rects.shape[0]):
        if (abs(rects[i][0] - prev_x) > x_margin or abs(rects[i][1] - prev_y) > y_margin
            stay.append(i)
            prev_x = rects[i][0]
            prev_y = rects[i][1]

    rects = rects[stay]
    rect2=[]
    for i in rects:
       # if(rect2==[]):
        #    rect2.append(i)
        a=1
        for j in range (len(rect2)):
            if(    ((i[0]-rect2[j][0])**2)+((i[1]-rect2[j][1])**2)<20  ):
              a=0
        if(a):
```

```
            rect2.append(i)
        for pt in rects:
            cv.rectangle(res_image, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)

    return rect2,res_image
# detect lines
def LineDetector(input_im):
# convert image to gray image
    Gray_IM = cv.cvtColor(input_im,cv.COLOR_BGR2GRAY)
# detect edges
    t_lower = 50  # Lower Threshold
     t_upper = 200  # Upper threshold
#apertureSize is an optional parameter that is used to specify the order of the Sobel fi
#to calculate the gradient in the Canny algorithm. The default value is 3 and its value
#be odd between 3 and 7. You can increase the Aperture size when you want to detect more
    edges = cv.Canny(Gray_IM,t_lower,t_upper,apertureSize=3)

    IM_lines = cv.HoughLinesP(
                edges,
                1, # Distance resolution in pixels
                np.pi/180, # Angle resolution in radians
                threshold=50, # Min number of votes for valid line
                minLineLength=40, # Min allowed length of line
                maxLineGap=10 # Max allowed gap between line for joining them
                )
    lines_Y=[]
    for line in IM_lines:
        xx,ys,xf,yf=line[0]
        if(abs(xx-xf)>50):
            if(abs(ys-yf)<5):
                a=1
                for i in lines_Y:
                    if(abs(i-ys)<5):
                        a=0
                if(a or lines_Y==[]):
                    lines_Y.append(ys)
    for i in lines_Y:
        cv.line(Gray_IM, (0, i), (1700, i), (0,0,255), 2, cv.LINE_AA)
    lines_Y.sort()
    return Gray_IM,lines_Y

Ave_image = cv.imread('polyushka.png',cv.IMREAD_GRAYSCALE)
Ave_image1 = cv.imread('polyushka.png')
template2 = cv.imread("whole.PNG", cv.IMREAD_GRAYSCALE)
threshold2 = 0.55

rects_part2, img3 = detectNotes(Ave_image, template2, threshold2)
linesim,lines=LineDetector(Ave_image1)
print(len(rects_part2))
for i in rects_part2:
    cv.circle(Ave_image1, (i[0]+14,i[1]+11), radius=0, color=(0,0 , 255), thickness=2)

_ = plt.imshow(Ave_image1, cmap = plt.cm.gray)
_ = plt.xticks([])
_ = plt.yticks([])
_ = plt.box(0)
```

```
# note class
class DorB:
    def __init__(self,x,y,Dbolean):
        self.x=x
        self.y=y
        self.D=Dbolean

class note :
    def __init__(self):
        self.x=0
        self.y=0
        self.id=0
        self.octave=0
        self.dur=0
class blunt :
    def __init__(self):
        self.lines=[]
        self.average=0
        self.notes=[]
        self.extra=[]
    def AddExtraToNote(self):
        for i in self.extra:
            min=100
            index=0
            for j in range(len(self.notes)):
                if( (abs(i.x-self.notes[j].x ))<min):
                    min=(abs(i.x-self.notes[j].x ))
                    #print(min)
                    index=j

            #print(index)
            if(i.D==1):
                #print("dies")
                if(self.notes[index].id==11):
                    self.notes[index].id=0
                    self.notes[index].octave=self.notes[index].octave+1
                else:
                    self.notes[index].id=self.notes[index].id+1
            else:
                #print("bemol")
                if(self.notes[index].id==0):
                    self.notes[index].id=11
                    self.notes[index].octave=self.notes[index].octave-1
                else:
                    self.notes[index].id=self.notes[index].id-1
    def findNotesStr(self):
       HLD=int(np.rint((self.lines[4]-self.lines[0])/8))
       notelevel=[
            self.lines[0]-2*HLD,
            self.lines[0]-HLD,
            self.lines[0],
            self.lines[1]-HLD,
            self.lines[1],
            self.lines[2]-HLD,
            self.lines[2],
            self.lines[3]-HLD,
```

```
            self.lines[3],
            self.lines[4]-HLD,
            self.lines[4],
            self.lines[4]+HLD,
            self.lines[4]+2*HLD,
            self.lines[4]+(4*HLD+1)
        ]
        index=pd.Index(notelevel)
        for note in self.notes:
            a=index.get_indexer([note.y], method="nearest")
            if(a==0):
                note.octave=5
                note.id=0
            elif (a==1):
                note.octave=5
                note.id=10
            elif (a==2):
                note.octave=5
                note.id=8
            elif (a==3):
                note.octave=5
                note.id=7
            elif (a==4):
                note.octave=5
                note.id=5
            elif (a==5):
                note.octave=5
                note.id=3
            elif (a==6):
                note.octave=4
                note.id=2
            elif (a==7):
                note.octave=4
                note.id=0
            elif (a==8):
                note.octave=4
                note.id=10
            elif (a==9):
                note.octave=4
                note.id=8
            elif (a==10):
                note.octave=4
                note.id=7
            elif (a==11):
                note.octave=4
                note.id=5
            elif (a==12):
                note.octave=4
                note.id=3
            elif (a==13):
                note.octave=3
                note.id=2


a=note()
a.y=10
```

```
a.x=5
print(a.x+a.y)
##################
#detect notes2
img2 = cv.imread('polyushka.png')
def MakeNotes2(input_img):
    result=[]
    ## line
    junk1,lines =LineDetector(input_img)
    # create blunts

    bluntN=int(len(lines)/5)
    blunts=[]
    for i in range (bluntN):
        temp_blunt=blunt()
        temp_blunt.lines=[lines[5*i],lines[5*i+1],lines[5*i+2],lines[5*i+3],lines[5*i+4]
        temp_blunt.average=sum(temp_blunt.lines)/len(temp_blunt.lines)
        print("Average",temp_blunt.average)
        blunts.append(temp_blunt)

    ## note
    ## quarters
    template = cv.imread("quarter (1).PNG", cv.IMREAD_GRAYSCALE)
    gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.7
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+14
        temp_note.y=point[1]+62
        temp_note.dur=3
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,

    ## half
    template = cv.imread("half.PNG", cv.IMREAD_GRAYSCALE)
   # gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.75
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+13
        temp_note.y=point[1]+69
        temp_note.dur=2
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
    #half2
    template = cv.imread("half2.PNG", cv.IMREAD_GRAYSCALE)
   # gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.71
```

```
 rects,junk2= detectNotes(gray, template, threshold)
 print(len(rects))
 for point in rects:
     temp_note=note()
     temp_note.x=point[0]+13
     temp_note.y=point[1]+12
     temp_note.dur=2
     for i in blunts:
         if(abs(i.average-temp_note.y)<75):
             i.notes.append(temp_note)
     image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
 ## dies
 template = cv.imread("dies.PNG", cv.IMREAD_GRAYSCALE)
# gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
 threshold = 0.7
 rects,junk2= detectNotes(gray, template, threshold)
 print(len(rects))
 for point in rects:
     temp_d=DorB(point[0]+25,point[1]+27,1)
     for i in blunts:
         if(abs(i.average-temp_d.y)<75):
             i.extra.append(temp_d)
             image2 = cv.circle(input_img, (temp_d.x,temp_d.y), radius=0, color=(0, 0
 ## bemol
 template = cv.imread("bemol.PNG", cv.IMREAD_GRAYSCALE)
# gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
 threshold = 0.7
 rects,junk2= detectNotes(gray, template, threshold)
 print(len(rects))
 for point in rects:
     temp_d=DorB(point[0]+25,point[1]+40,0)
     for i in blunts:
         if(abs(i.average-temp_d.y)<75):
             i.extra.append(temp_d)
             image2 = cv.circle(input_img, (temp_d.x,temp_d.y), radius=0, color=(0, 0

 ## whole
 template = cv.imread("whole.PNG", cv.IMREAD_GRAYSCALE)
 threshold = 0.55
 rects,junk2= detectNotes(gray, template, threshold)
 print(len(rects))
 for point in rects:
     temp_note=note()
     temp_note.x=point[0]+14
     temp_note.y=point[1]+11
     temp_note.dur=1
     for i in blunts:
         if(abs(i.average-temp_note.y)<75):
             i.notes.append(temp_note)
 image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0, 255)




 for i in blunts:
```

```
        # print(len(i.notes),"dcecc")
          i.findNotesStr()
          i.AddExtraToNote()



    return image2,blunts


out2,blunts2=MakeNotes2(img2)
_ = plt.imshow(out2, cmap = plt.cm.gray)
music_to_play2=[]
for i in blunts2:
    i.notes.sort(key=lambda x: x.x, reverse=False)
    for j in i.notes:
        music_to_play2.append([j.id,j.octave,j.dur])


final2=get_music(music_to_play2,44100)
len(music_to_play2)
sd.play(final2)
#################
```

another result that is considerable has brought below:

```
print(music_to_play2)
[[0, 4, 2], [9, 4, 3], [0, 4, 3], [11, 4, 2], [7, 4, 3], [4, 4, 3], [0, 4, 3], [11, 4, 3
```

rest of code:

```
notes_base = 2**(np.arange(12)/12)*27.5
notes_duration = np.array([3200, 1600, 800, 400, 200, 100])*0.7
notes_ann = ['A', 'A#', 'B', 'C', 'C#', 'D', 'Eb', 'E', 'F', 'F#', 'G', 'G#']

def sin_wave(f, n, fs):
    x = np.linspace(0, 2*np.pi, n)
    ring = 30
    xp = np.linspace(0, -1*(n*ring/fs), n)
    y = np.sin(x*f*(n/fs))*np.exp(xp)
    z = np.zeros([n, 2])
    z[:, 0] = y
    z[:, 1] = y
    return z

def play_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    sd.play(y, fs)
    sd.wait()
    return

def put_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    return y
```

```
def get_music(music_notes, fs):
    m = []
    for item in music_notes:
        y = put_note(item[0], item[1], item[2], fs)
        m.append(y)
    m = np.concatenate(m, 0)
    return m

fs1 = 44100
music = [[8, 5, 3], [10, 5, 4], [10, 5, 4], [10, 5, 3], [10, 5, 3], [10, 5, 3], [10, 5,
        [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4],
        [8, 5, 4], [8, 5, 4], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8,
        [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4],

y = get_music(music, fs1)
sd.play(y, fs1)
write("polyushka.wav", 44100, final2.astype(np.float32))
```

ss project phase2 BONUS

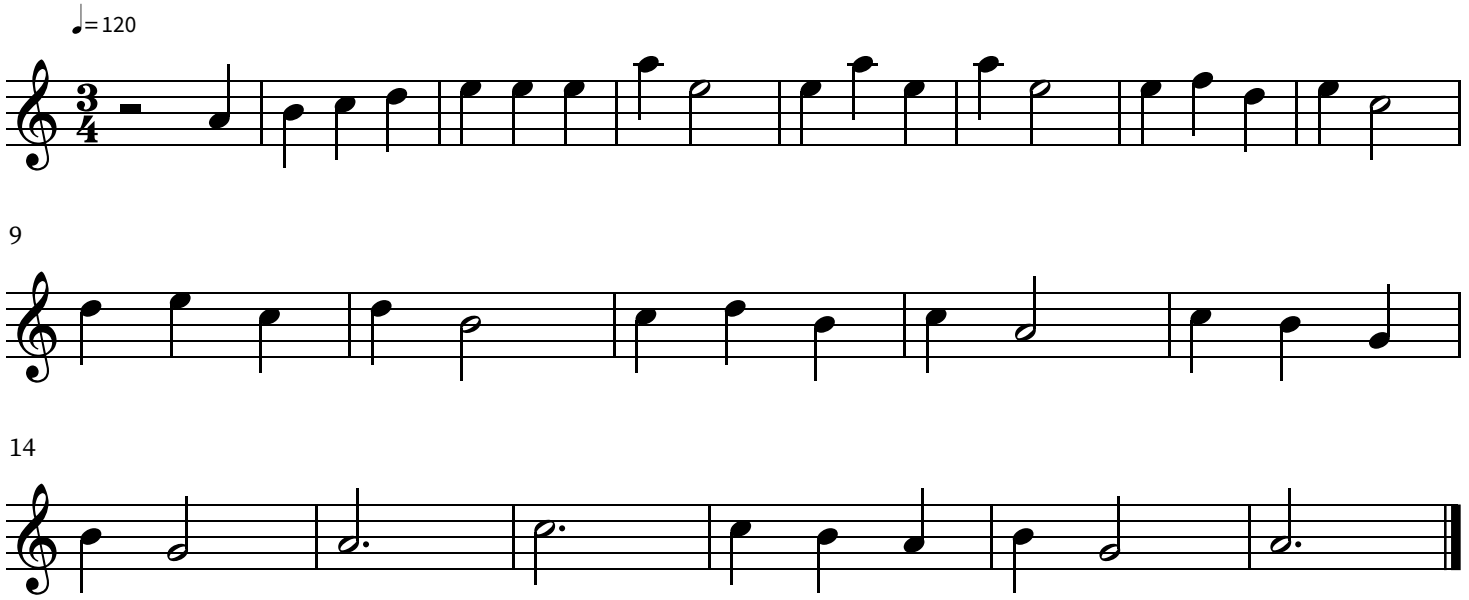
Hosein Asghari Hosouri
Mr.Amirreza Zamani

July 4, 2023

1 BONUS

In this question we should implement "Polyushka-Ploye". it means we have to convert it from sheet music into music by template matching method the related music sheet is attached below:

Jane Maryam



some additional information about this music sheet:
as you can see,we have Crotchet ,Minim ,Semibreve that means 1x ,2x ,4x and diese and bemol .and we have 120BPM and 7 means that our blocking of notes(kind of assortment) starts at 9 . we follow previous approach but in this sample we add some new notes with different distances such asd ot and some stem down notes and b new generated templates we detected them .the main changes occur in "get-music function" we must add the related template of dot and some stem down notes too.for more explanation we must consider duration of each note. because in the previous section we have only one type of note but here we have three types with different duration . furthermore we need to check for each template new appropriate coefficients of threshold and we found the magnitude of displacement with respect to found rects that in this sample is 63 for y and 14 for x.
as well,we defined new calsses for dot in the name of **" dot"** that has three parameter such as "x , y , and self// when we are grouping our note we add new class dot.
by means of previous section we developed our code as follows:

```
import pydub
import numpy as np
import sounddevice as sd
from matplotlib import pyplot as plt
from scipy.io.wavfile import read, write
import cv2 as cv
import pandas as pd
def detectNotes(input_image, template, threshold):
    w, h = template.shape[::-1]
    res_image = input_image.copy()
    matches = cv.matchTemplate(input_image, template, cv.TM_CCOEFF_NORMED)
    match_loc = np.where( matches >= threshold)
    rects = list(zip(*match_loc[::-1]))
    rects = sorted(rects, key = lambda x: (x[0], x[1]))
    rects = np.asarray(rects)
    prev_x = rects[0][0]
    prev_y = rects[0][1]
    x_margin = 20
    y_margin = 20
    stay = [0]
    for i in range(1, rects.shape[0]):
        if (abs(rects[i][0] - prev_x) > x_margin or abs(rects[i][1] - prev_y) > y_margin
            stay.append(i)
            prev_x = rects[i][0]
            prev_y = rects[i][1]

    rects = rects[stay]
    rect2=[]
    for i in rects:
       # if(rect2==[]):
        #    rect2.append(i)
        a=1
        for j in range (len(rect2)):
            if(    ((i[0]-rect2[j][0])**2)+((i[1]-rect2[j][1])**2)<20  ):
              a=0
        if(a):
            rect2.append(i)
        for pt in rects:
            cv.rectangle(res_image, pt, (pt[0] + w, pt[1] + h), (0,0,255), 1)

    return rect2,res_image
```

```
# detect lines
def LineDetector(input_im):
# convert image to gray image
    Gray_IM = cv.cvtColor(input_im,cv.COLOR_BGR2GRAY)
# detect edges
    t_lower = 50  # Lower Threshold
    t_upper = 200  # Upper threshold
#apertureSize is an optional parameter that is used to specify the order of the Sobel fi
#to calculate the gradient in the Canny algorithm. The default value is 3 and its value
#be odd between 3 and 7. You can increase the Aperture size when you want to detect more
    edges = cv.Canny(Gray_IM,t_lower,t_upper,apertureSize=3)

    IM_lines = cv.HoughLinesP(
                edges,
                1, # Distance resolution in pixels
                np.pi/180, # Angle resolution in radians
                threshold=50, # Min number of votes for valid line
                minLineLength=40, # Min allowed length of line
                maxLineGap=10 # Max allowed gap between line for joining them
                )
    lines_Y=[]
    for line in IM_lines:
        xx,ys,xf,yf=line[0]
        if(abs(xx-xf)>50):
            if(abs(ys-yf)<5):
                a=1
                for i in lines_Y:
                    if(abs(i-ys)<5):
                        a=0
                if(a or lines_Y==[]):
                    lines_Y.append(ys)
    for i in lines_Y:
        cv.line(Gray_IM, (0, i), (1700, i), (0,0,255), 2, cv.LINE_AA)
    lines_Y.sort()
    return Gray_IM,lines_Y

Ave_image = cv.imread('Maryam.png',cv.IMREAD_GRAYSCALE)
Ave_image1 = cv.imread('Maryam.png')
template2 = cv.imread("quarter2.PNG", cv.IMREAD_GRAYSCALE)
threshold2 = 0.75

rects_part2, img3 = detectNotes(Ave_image, template2, threshold2)
linesim,lines=LineDetector(Ave_image1)
print(len(rects_part2))
for i in rects_part2:
    cv.circle(Ave_image1, (i[0]+13,i[1]+10), radius=0, color=(0,0 , 255), thickness=2)

_ = plt.imshow(Ave_image1, cmap = plt.cm.gray)
_ = plt.xticks([])
_ = plt.yticks([])
_ = plt.box(0)
```

so far we have result below:

rest of code:

```
# note class
```

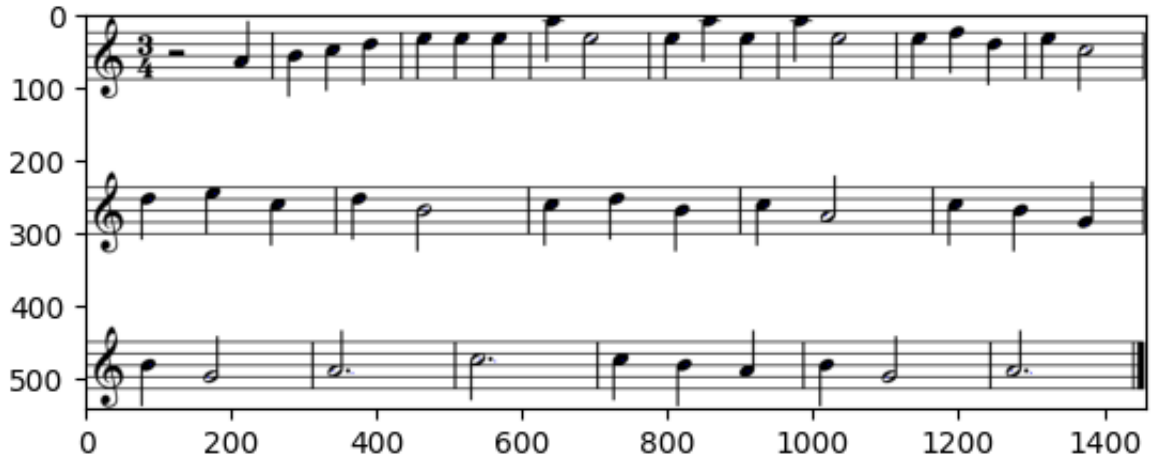


Figure 1: output

```
class dot:
    def __init__(self,x,y):
        self.x=x
        self.y=y
class note :
    def __init__(self):
        self.x=0
        self.y=0
        self.id=0
        self.octave=0
        self.dur=0
class blunt :
    def __init__(self):
        self.lines=[]
        self.average=0
        self.notes=[]
        self.dots=[]
    def AddDots(self):
        for i in self.dots:
            min=100
            index=0
            for j in range(len(self.notes)):
                if( (abs(i.x-self.notes[j].x ))<min):
                    min=(abs(i.x-self.notes[j].x ))
                    #print(min)
                    index=j
            self.notes[index].dur=6


    def findNotesStr(self):
       HLD=int(np.rint((self.lines[4]-self.lines[0])/8))
        notelevel=[
            self.lines[0]-2*HLD,
            self.lines[0]-HLD,
            self.lines[0],
            self.lines[1]-HLD,
            self.lines[1],
```

```
        self.lines[2]-HLD,
        self.lines[2],
        self.lines[3]-HLD,
        self.lines[3],
        self.lines[4]-HLD,
        self.lines[4],
        self.lines[4]+HLD,
        self.lines[4]+2*HLD,
        self.lines[4]+(4*HLD+1)
    ]
    index=pd.Index(notelevel)
    for note in self.notes:
        a=index.get_indexer([note.y], method="nearest")
        if(a==0):
            note.octave=5
            note.id=0
        elif (a==1):
            note.octave=5
            note.id=10
        elif (a==2):
            note.octave=5
            note.id=8
        elif (a==3):
            note.octave=5
            note.id=7
        elif (a==4):
            note.octave=5
            note.id=5
        elif (a==5):
            note.octave=5
            note.id=3
        elif (a==6):
            note.octave=4
            note.id=2
        elif (a==7):
            note.octave=4
            note.id=0
        elif (a==8):
            note.octave=4
            note.id=10
        elif (a==9):
            note.octave=4
            note.id=8
        elif (a==10):
            note.octave=4
            note.id=7
        elif (a==11):
            note.octave=4
            note.id=5
        elif (a==12):
            note.octave=4
            note.id=3
        elif (a==13):
            note.octave=3
            note.id=2
```

```
a=note()
a.y=10
a.x=5
print(a.x+a.y)
####################
#detect notes2
img2 = cv.imread('Maryam.png')
def MakeNotes2(input_img):
    result=[]
    ## line
    junk1,lines =LineDetector(input_img)
    # create blunts

    bluntN=int(len(lines)/5)
    blunts=[]
    for i in range (bluntN):
        temp_blunt=blunt()
        temp_blunt.lines=[lines[5*i],lines[5*i+1],lines[5*i+2],lines[5*i+3],lines[5*i+4]
        temp_blunt.average=sum(temp_blunt.lines)/len(temp_blunt.lines)
        print("Average",temp_blunt.average)
        blunts.append(temp_blunt)

    ## note
    ## quarters
    template = cv.imread("quarter (1).PNG", cv.IMREAD_GRAYSCALE)
    gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.69
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+14
        temp_note.y=point[1]+63
        temp_note.dur=3
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,

    ## half
    template = cv.imread("half.PNG", cv.IMREAD_GRAYSCALE)
   # gray = cv.cvtColor(input_img, cv.COLOR_BGR2GRAY)
    threshold = 0.45
    rects,junk2= detectNotes(gray, template, threshold)
    print(len(rects))
    for point in rects:
        temp_note=note()
        temp_note.x=point[0]+13
        temp_note.y=point[1]+69
        temp_note.dur=2
        for i in blunts:
            if(abs(i.average-temp_note.y)<75):
                i.notes.append(temp_note)
        image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
    ## half2
```

```
template = cv.imread("half2.PNG", cv.IMREAD_GRAYSCALE)
threshold = 0.5
rects,junk2= detectNotes(gray, template, threshold)
print(len(rects))
for point in rects:
    temp_note=note()
    temp_note.x=point[0]+13
    temp_note.y=point[1]+12
    temp_note.dur=2
    for i in blunts:
        if(abs(i.average-temp_note.y)<75):
            i.notes.append(temp_note)
    image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
## quarter 2
template = cv.imread("quarter2.PNG", cv.IMREAD_GRAYSCALE)
threshold = 0.75
rects,junk2= detectNotes(gray, template, threshold)
print(len(rects))
for point in rects:
    temp_note=note()
    temp_note.x=point[0]+13
    temp_note.y=point[1]+10
    temp_note.dur=3
    for i in blunts:
        if(abs(i.average-temp_note.y)<75):
            i.notes.append(temp_note)
    image2 = cv.circle(input_img, (temp_note.x,temp_note.y), radius=0, color=(0, 0,
## dot
template = cv.imread("dot.PNG", cv.IMREAD_GRAYSCALE)
threshold = 0.55
rects,junk2= detectNotes(gray, template, threshold)
print(len(rects))
for point in rects:
    temp_dot=dot(point[0]+13,point[1]+10)
    for i in blunts:
        if(abs(i.average-temp_dot.y)<75):
            i.dots.append(temp_dot)
    image2 = cv.circle(input_img, (temp_dot.x,temp_dot.y), radius=0, color=(0, 0, 25









for i in blunts:
    print(len(i.notes))
    i.findNotesStr()
    i.AddDots()



return image2,blunts
```

```
out2,blunts2=MakeNotes2(img2)
_ = plt.imshow(out2, cmap = plt.cm.gray)
music_to_play2=[]
for i in blunts2:
    i.notes.sort(key=lambda x: x.x, reverse=False)
    for j in i.notes:
        music_to_play2.append([j.id,j.octave,j.dur])
final2=get_music(music_to_play2,44100)
len(music_to_play2)
sd.play(final2)
```

and the results are as follows:
Average 57.0 Average 269.0 Average 481.0 3 5 5 29 3 19 13 10
rest of code:

```
notes_base = 2**(np.arange(12)/12)*27.5
notes_duration = np.array([3200, 1600, 800, 400, 200, 100,1200])*0.7
notes_ann = ['A', 'A#', 'B', 'C', 'C#', 'D', 'Eb', 'E', 'F', 'F#', 'G', 'G#']

def sin_wave(f, n, fs):
    x = np.linspace(0, 2*np.pi, n)
    ring = 30
    xp = np.linspace(0, -1*(n*ring/fs), n)
    y = np.sin(x*f*(n/fs))*np.exp(xp)
    z = np.zeros([n, 2])
    z[:, 0] = y
    z[:, 1] = y
    return z

def play_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    sd.play(y, fs)
    sd.wait()
    return

def put_note(note_id, octave, dur, fs):
    if (note_id < 3) :
        octave += 1
    y = sin_wave(notes_base[note_id]*2**octave, int(notes_duration[dur]*(fs/1000)), fs)
    return y

def get_music(music_notes, fs):
    m = []
    for item in music_notes:
        y = put_note(item[0], item[1], item[2], fs)
        m.append(y)
    m = np.concatenate(m, 0)
    return m

fs1 = 44100
music = [[8, 5, 3], [10, 5, 4], [10, 5, 4], [10, 5, 3], [10, 5, 3], [10, 5, 3], [10, 5,
        [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4], [4, 5, 4], [11, 5, 4],
        [8, 5, 4], [8, 5, 4], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8, 5, 3], [8,
        [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4], [3, 5, 4], [10, 5, 4],
```

```
y = get_music(music, fs1)
sd.play(y, fs1)
write("Ave_result.wav", 44100, final2.astype(np.float32))
fs1=44100
y1 = get_music([[2,3,2]], fs1)
sd.play(y1, fs1)
print(music_to_play2)
```

and the results are as follows:

0, 4, 3!

, [2, 5, 3], [3, 5, 3], [5, 5, 3], [7, 5, 3], [7, 5, 3], [7, 5, 3], [0, 5, 3], [7, 5, 2], [7, 5, 3], [0, 5, 3], [7, 5, 3], [0, 5, 3], [7, 5, 2], [7, 5, 3], [8, 5, 3], [5, 5, 3], [7, 5, 3], [3, 5, 2], [5, 5, 3], [7, 5, 3], [3, 5, 3], [5, 5, 3], [2, 5, 2], [3, 5, 3], [5, 5, 3], [2, 5, 3], [3, 5, 3], [0, 4, 2], [3, 5, 3], [2, 5, 3], [8, 4, 3], [2, 5, 3], [10, 4, 2], [0, 4, 6], [3, 5, 6], [3, 5, 3], [2, 5, 3], [0, 4, 3], [2, 5, 3], [10, 4, 2], [0, 4, 6]]

```
write("Maryam.wav", 44100, final2.astype(np.float32))
```