SYMPHONY

PROJECT ID: IEEE BPIT

INTRODUCTION

Symphony is an interactive music experience that utilizes computer vision technology to detect leg gestures made by the participant using ArucoMarkers. The project utilizes the OpenCV library and ArucoMarkers to detect specific markers placed on the floor, and through the use of boolean expressions in conditional statements, produces corresponding musical notes or sounds. This project also incorporates the Tkinter library to provide a user-friendly interface for the participant, making the experience more engaging and interactive. Written in Python, Symphony utilizes various other libraries such as NumPy, threading, time, OS, Tkinter and PyGame to facilitate image processing, threading, and sound output for a seamless and exciting experience.

OBJECTIVES

- To detect markers placed on the floor using OpenCV.
- To produce sounds when markers are detected using boolean expressions in conditional statements.
- To provide a better user interface for the participant using Tkinter.
- To make the game more responsive by using threading and the time module.

METHODOLOGY AND RESEARCH

- Symphony employs advanced computer vision technology, specifically the OpenCV library, to detect markers placed on the floor.
- The markers are accurately identified through the use of the ArucoMarkers module within the OpenCV library.
- To enhance performance and responsiveness, the project incorporates the threading module.
- Audio files are played in response to marker detection with the use of the Pygame library.
- The NumPy library is utilized to handle arrays and matrices within the code for efficient data manipulation.
- A user-friendly interface for the participant is provided through the use of the Tkinter library.
- The code employs the random module to randomly shuffle the audio files played, adding an element of unpredictability to the experience.
- The function "find_aruco" is used to detect markers by taking an image as input and utilizing the Aruco module within OpenCV, providing accurate and real-time detection.
- The function "func" is called by the "find_aruco" function, which checks for marker detection and plays corresponding audio files using the mixer module, providing a dynamic and interactive experience.
- The code also includes a function "counter()" which is used to increment a count variable, to track the number of marker detections, and provide feedback to the user.

FUTURE SCOPE

- Symphony can be expanded to include a greater variety of markers and more intricate gesture recognition, providing even more opportunities for the participant to control and interact with the music.
- Integrating Symphony with other music production software can open up a wide range of creative possibilities, allowing for even more dynamic and unique musical performances.
- The project's unique and interactive nature makes it an ideal fit for various performance settings, such as dance shows, concerts, and other live events.
- The technology behind Symphony can also be applied in other fields, such as gaming, fitness, and rehabilitation, providing new opportunities for interactive and engaging experiences.
- Symphony could be enhanced with the integration of additional sensors and hardware to increase the precision and accuracy of marker detection and gesture recognition.
- Machine learning techniques could be implemented to improve the adaptability and versatility of the system.
- The project could be adapted for use in virtual and augmented reality environments, providing even more immersive and interactive experiences.
- Symphony could be utilized in educational settings as a tool for teaching music and programming.
- Symphony can be integrated with other digital music instruments and used in various fields such as music production, therapy, gaming, advertising, and entertainment.
- It can also be adapted for use in virtual and educational settings, providing immersive and interactive experiences.

PROJECT SOURCE CODE AND RESULTS

Code for Marker Detection

```
import threading

import time

import cv2

import cv2.aruco as aruco

import numpy as np

from pygame import mixer

import imutils

import random

check = False

class ntm():

  def __init__(self, img):

      threading.Thread.__init__(self)

      self.find_aruco = threading.Thread(target=find_aruco,
args=(img))

      self.find_aruco.daemon = True

  def run(self, img):

      print("????????")

      find_aruco(self, img)

id = []

# dict = {103: '1_part.mp3', 1: '2_part.mp3', 2: '3_part.mp3', 3:
'4_part.mp3', 4: '1.mp3', 5: '2.mp3', 6: '3.mp3', 8: '4.mp3'}

dict = {4: '1_part.mp3', 5: '2_part.mp3', 6: '3_part.mp3', 8:
'4_part.mp3'}

mixer.init()
```

```
def find_aruco(img, markersize=6, totalmarkers=250, draw=True):
  # gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # gray2 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
  thresh1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # cv2.threshold(gray, 175, 255, cv2.THRESH_BINARY)[1]
  #     thresh2 = cv2.threshold(gray2, 100, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)[1]
  #     imgray=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  key = getattr(aruco,
f'DICT_{markersize}X{markersize}_{totalmarkers}')
  arucodict = aruco.Dictionary_get(key)
  arucoparam = aruco.DetectorParameters_create()
  # bboxs, ids, rejected = aruco.detectMarkers(thresh1, arucodict,
parameters=arucoparam)
  bboxs, ids, rejected = aruco.detectMarkers(thresh1, arucodict,
parameters=arucoparam)
  #     bboxs1, ids1, rejected1 = aruco.detectMarkers(thresh2,
arucodict, parameters=arucoparam)
  # chnone(ids, ids1)
  id = np.unique(ids)
  # + np.unique(ids1)
  print(id)
  func(id, check)
  # id.append(ids)
  if draw:
      aruco.drawDetectedMarkers(thresh1, bboxs)
  cv2.imshow("threaded image", thresh1)
count = 0
def one():
```

```
  mixer.music.play()
def check_all(id):
  a = id.size
  if a == 4:
      return True
def counter():
  global count
  if count < 50:
      count += 12.5
  print(count)
def chnone(id):
  id1 = [4, 5, 6, 8]
  id = id1
  id0 = np.random.shuffle(id1)
  i = id[0]
  k = random.randint(4, 8)
  if k != 7:
      if id1[0] != 4 or id1[0] != 5 or id1[0] != 6 or id1[0] != 8:
          mixer.music.load(dict[k])
          mixer.music.play()
      time.sleep(4)
      counter()
# def chnone(ids, ids1):
#         mixer.music.load('1.wav')
#         if ids is None or ids1 is None:
#             ids = 0 if ids is None else ids
#             ids1 = 0 if ids1 is None else ids1
```

```
#         elif ids[0] != [0] or ids1[0] != [0]:
#             mixer.music.play()
#             counter()
#             time.sleep(10)
#             b = False
#             mixer.music.load('2_part.mp3')
#             return
#         elif ids[1] != [1] or ids1[1] != [1]:
#             one()
#             counter()
#             b = False
#             mixer.music.load('3_part.mp3')
#             return
#         elif ids[2] != [2] or ids1[2] != [2]:
#             one()
#             counter()
#             b = False
#             mixer.music.load('4_part.mp3')
#             return
#         elif ids[3] != [3] or ids1[3] != [3]:
#             one()
#             counter()
#             b = False
#             # mixer.music.load('music/5_part.mp3')
#             return
# elif ids[4] != [4] or ids1[4] != [4]:
#     one()
```

```
#     counter()
#     b = False
#     mixer.music.load('music/6_part.mp3')
#     return
# elif ids[5] != [5] or ids1[5] != [5]:
#     one()
#     time.sleep(10)
#     counter()
#     b = False
#     mixer.music.load('music/7_part.mp3')
#     return
# elif ids[6] != [6] or ids1[6] != [6]:
#     one()
#     time.sleep(10)
#     counter()
#     b = False
#     mixer.music.load('music/8_part.mp3')
#     return
# elif ids[7] != [7] or ids1[7] != [7]:
#     one()
#     time.sleep(10)
#     counter()
#     b = False
#     return
def func(id, check):
  if id.size != 4:
      print("True")
```

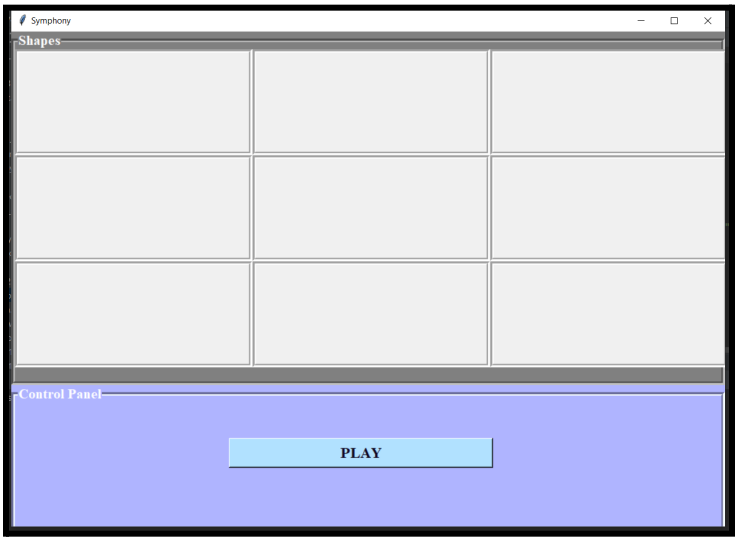
```
        mixer.init()
        check = True
        chnone(id)
    # if check:
    #
    #     print("Running")
    # else:
    #     print(" ??? ")
    return check
def main():
    cap = cv2.VideoCapture(1)
    while True:
        success, img = cap.read()
        find_aruco(img)
        cv2.waitKey(1)
if __name__ == "__main__":
    main()



Code for GUI
from tkinter import *
from random import *
class MusicPlayer:
    def __init__(self, root):
        self.root = root
        self.root.title("Symphony")
        self.root.geometry("1000x700+100+0")
```

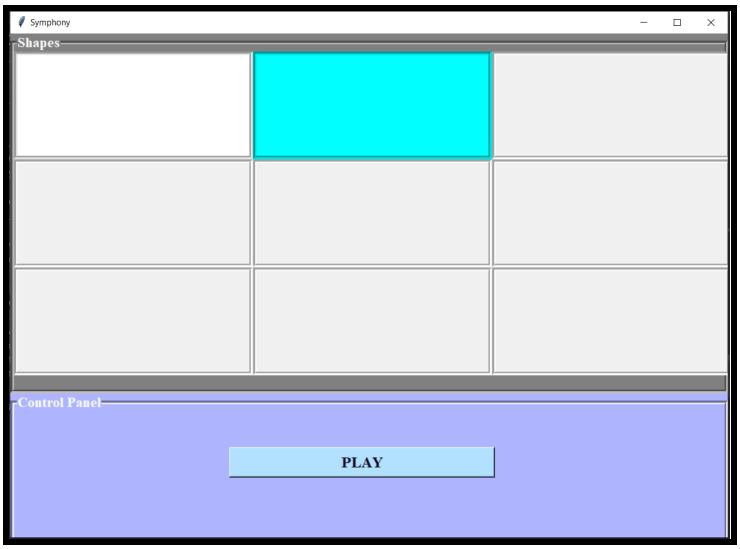
```
        self.track = StringVar()

        self.status = StringVar()

        self.shapeframe =
LabelFrame(self.root,text="Shapes",font=("times new
roman",15,"bold"),bg="grey",fg="white",bd=5,relief=GROOVE)

        self.shapeframe.place(x=0, y=0, width=1000, height=500)

        #matrix Labels

        self.a00 = LabelFrame(self.shapeframe, bd=5)

        self.a00.place(x=0, y=0, width=333, height=150)

        self.a01 = LabelFrame(self.shapeframe, bd=5)

        self.a01.place(x=333, y=0, width=333, height=150)

        self.a02 = LabelFrame(self.shapeframe, bd=5)

        self.a02.place(x=666, y=0, width=333, height=150)

        self.a10 = LabelFrame(self.shapeframe, bd=5)

        self.a10.place(x=0, y=150, width=333, height=150)

        self.a11 = LabelFrame(self.shapeframe, bd=5)

        self.a11.place(x=333, y=150, width=333, height=150)

        self.a12 = LabelFrame(self.shapeframe, bd=5)

        self.a12.place(x=666, y=150, width=333, height=150)

        self.a20 = LabelFrame(self.shapeframe, bd=5)

        self.a20.place(x=0, y=300, width=333, height=150)

        self.a21 = LabelFrame(self.shapeframe, bd=5)

        self.a21.place(x=333, y=300, width=333, height=150)

        self.a22 = LabelFrame(self.shapeframe, bd=5)

        self.a22.place(x=666, y=300, width=333, height=150)
```

```
#ButtonFrame

      buttonframe = LabelFrame(self.root,text="Control
Panel",font=("times new
roman",15,"bold"),bg="#AFB4FF",fg="white",bd=5,relief=GROOVE)

      buttonframe.place(x=0, y=500, width=1000, height=330)

      plybtn = Button(buttonframe, text="PLAY",command=self.playsong,
width=30, height=1, font=("times new roman", 16, "bold"),
fg="#100720", bg="#B1E1FF").grid(row=0, column=0, padx=300, pady=50)

      #song frame

      #scrolling and lust

  def change(self,color,ls, i=0):

          if(i<8):

              ls[i].config(bg=color)

              ls[i].after(1500, self.change, 'white',ls, i)

              ls[i].after(1500, self.change,'aqua',ls,  i+1)

      #shape = Shape(shapeframe)

  def playsong(self):

      ls = [self.a00, self.a01, self.a02, self.a10, self.a11,
self.a12, self.a20, self.a21, self.a22]

      shuffle(ls)

      self.change('aqua', ls)

root = Tk()

MusicPlayer(root)

root.mainloop()
```

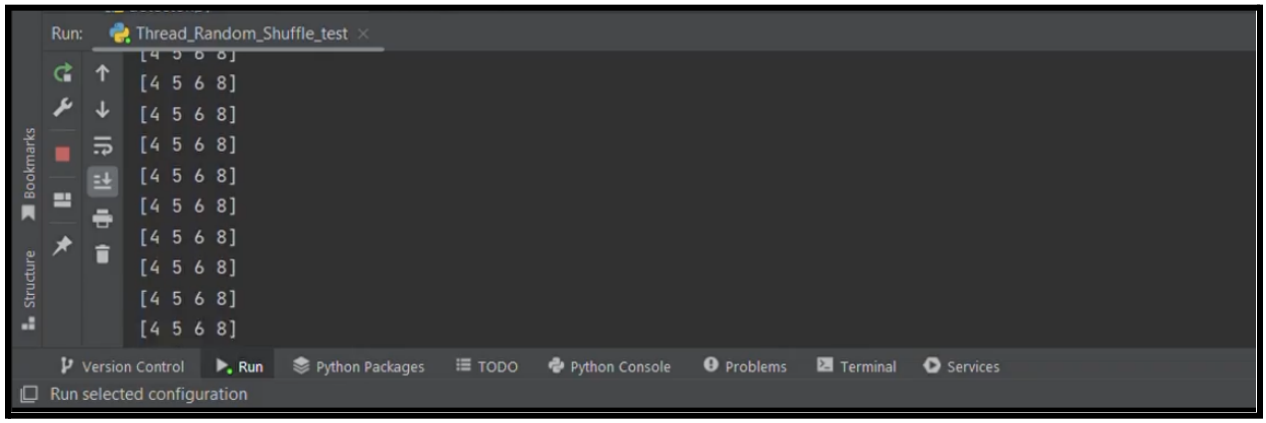
SNAPSHOTS OF CODE

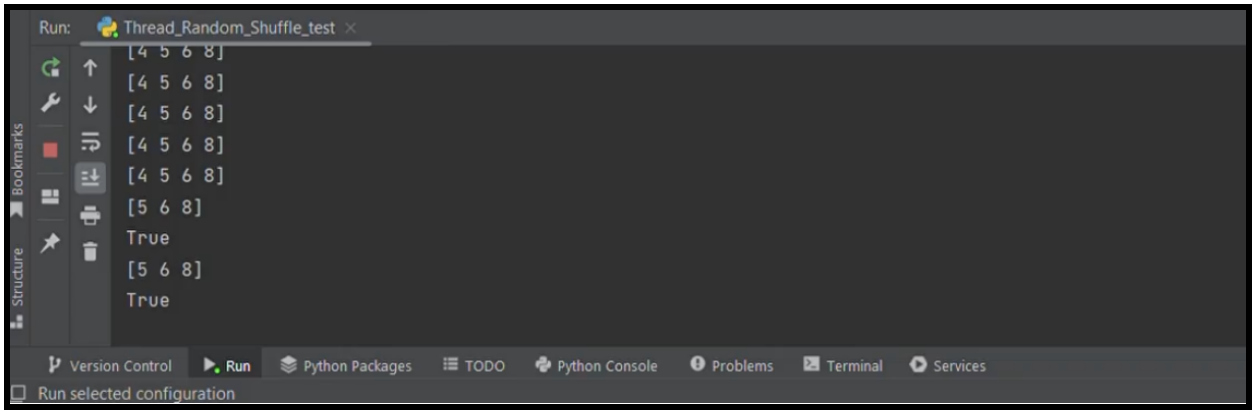
Symphony UI





Output Window Showing The IDs Of Detected ArucoMarkers





IDs of ArucoMarkers being triggered for detection and playing sounds

CONTRIBUTORS

S.No.	Name	Year
1	Vineet Thakur	4th Year EEE
2	Himanshu Bhenwal	2nd Year CSE
3	Jai Tyagi	2nd Year ECE
4	Nakul Singh	2nd Year CSE
5	Harshit Kaushik	2nd Year CSE
6	Aditya sharma	2nd Year EEE
7	Krishna Agarwal	2nd Year CSE
8	Archita	3rd Year CSE
9	Arushi	3rd Year CSE