1. **APPENDICES**

**APPENDIX 1: Excel Trial Results**

This table shows the trial result about the vestibular exercises, after 50 trying. The success rate of the program was found to be 79.15% and is shown below the table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test No** | **Motion Number** | **Correct Move Time (seconds)** | **Incorrect Move Time (seconds)** | **Expected Correct Percentage** | **Expected Incorrect Percentage** | **Obtained Correct Percentage** | **Obtained Incorrect Percentage** |
| 1 | 1a | 15 | 15 | 50,00% | 50,00% | 65,00% | 35,00% |
| 2 | 1a | 20 | 10 | 66,67% | 33,33% | 97,00% | 3,00% |
| 3 | 1a | 13 | 17 | 43,33% | 56,67% | 64,20% | 35,80% |
| 4 | 1a | 30 | 0 | 100,00% | 0,00% | 92,36% | 7,64% |
| 5 | 1a | 6 | 24 | 20,00% | 80,00% | 46,82% | 53,18% |
| 6 | 1a | 19 | 11 | 63,33% | 36,67% | 87,58% | 12,42% |
| 7 | 1a | 14 | 16 | 46,67% | 53,33% | 74,20% | 25,80% |
| 8 | 1a | 14 | 16 | 46,67% | 53,33% | 67,30% | 32,70% |
| 9 | 1a | 25 | 5 | 83,33% | 16,67% | 97,54% | 2,46% |
| 10 | 1a | 17 | 13 | 56,67% | 43,33% | 78,62% | 21,38% |
| 11 | 1a | 13 | 17 | 43,33% | 56,67% | 72,63% | 27,37% |
| 12 | 1a | 11 | 19 | 36,67% | 63,33% | 77,98% | 22,02% |
| 13 | 1a | 18 | 12 | 60,00% | 40,00% | 86,30% | 13,70% |
| 14 | 1a | 21 | 9 | 70,00% | 30,00% | 93,80% | 6,20% |
| 15 | 1a | 16 | 14 | 53,33% | 46,67% | 74,86% | 25,14% |
| 16 | 1a | 7 | 23 | 23,33% | 76,67% | 45,63% | 54,37% |
| 17 | 1a | 15 | 15 | 50,00% | 50,00% | 65,45% | 34,55% |
| 18 | 1a | 23 | 7 | 76,67% | 23,33% | 88,63% | 11,37% |
| 19 | 1a | 28 | 2 | 93,33% | 6,67% | 95,66% | 4,34% |
| 20 | 1a | 9 | 21 | 30,00% | 70,00% | 44,25% | 55,75% |
| 21 | 1a | 11 | 19 | 36,67% | 63,33% | 51,30% | 48,70% |
| 22 | 1a | 23 | 7 | 76,67% | 23,33% | 84,36% | 15,64% |
| 23 | 1a | 17 | 13 | 56,67% | 43,33% | 85,10% | 14,90% |
| 24 | 1a | 12 | 18 | 40,00% | 60,00% | 59,30% | 40,70% |
| 25 | 1a | 15 | 15 | 50,00% | 50,00% | 74,26% | 25,74% |
| 26 | 2b | 12 | 18 | 40,00% | 60,00% | 68,00% | 32,00% |
| 27 | 2b | 14 | 16 | 46,67% | 53,33% | 61,41% | 38,59% |
| 28 | 2b | 10 | 20 | 33,33% | 66,67% | 52,36% | 47,64% |
| 29 | 2b | 25 | 5 | 83,33% | 16,67% | 76,25% | 23,75% |
| 30 | 2b | 9 | 21 | 30,00% | 70,00% | 48,66% | 51,34% |
| 31 | 2b | 23 | 7 | 76,67% | 23,33% | 99,42% | 0,58% |
| 32 | 2b | 6 | 24 | 20,00% | 80,00% | 38,22% | 61,78% |
| 33 | 2b | 21 | 9 | 70,00% | 30,00% | 92,30% | 7,70% |
| 34 | 2b | 28 | 2 | 93,33% | 6,67% | 94,25% | 5,75% |
| 35 | 2b | 3 | 27 | 10,00% | 90,00% | 38,69% | 61,31% |
| 36 | 2b | 27 | 3 | 90,00% | 10,00% | 90,21% | 9,79% |
| 37 | 2b | 14 | 16 | 46,67% | 53,33% | 74,20% | 25,80% |
| 38 | 2b | 1 | 29 | 3,33% | 96,67% | 82,35% | 17,65% |
| 39 | 2b | 12 | 18 | 40,00% | 60,00% | 62,85% | 37,15% |
| 40 | 2b | 8 | 22 | 26,67% | 73,33% | 47,30% | 52,70% |
| 41 | 2b | 6 | 24 | 20,00% | 80,00% | 35,25% | 64,75% |
| 42 | 2b | 18 | 12 | 60,00% | 40,00% | 87,52% | 12,48% |
| 43 | 2b | 17 | 13 | 56,67% | 43,33% | 86,34% | 13,66% |
| 44 | 2b | 4 | 26 | 13,33% | 86,67% | 84,58% | 15,42% |
| 45 | 2b | 9 | 21 | 30,00% | 70,00% | 48,91% | 51,09% |
| 46 | 2b | 12 | 18 | 40,00% | 60,00% | 56,80% | 43,20% |
| 47 | 2b | 25 | 5 | 83,33% | 16,67% | 74,60% | 25,40% |
| 48 | 2b | 29 | 1 | 96,67% | 3,33% | 95,60% | 4,40% |
| 49 | 2b | 2 | 28 | 6,67% | 93,33% | 38,63% | 61,37% |
| 50 | 2b | 17 | 13 | 56,67% | 43,33% | 84,58% | 15,42% |
|  |  |  |  |  |  |  |  |
|  |  |  | **Avarages:** | **50,93%** | **49,07%** | **71,79%** | **28,21%** |
|  |  |  |  |  |  |  |  |
|  |  |  | **Percentage Of Program Success:** | | | **79,15%** |  |

**APPENDIX 2: Code Blocks (prediction.py)**

*#IMPORT LIBRARIES***import** sys *#FOR PYTHON VERSION PROCESSES***import** os *#FOR PROCESSES OF FOLDERS AND FILES***import** cv2 *#FOR IMAGE PROCESSING***import** numpy **as** np *#FOR ARRAYS OF IMAGE***import** time *#FOR TIME PROCESSES***from** playsound **import** playsound *#FOR PLAYING SOUNDS***from** PyQt5.QtWidgets **import** \* *#FOR USER INTERFACE***from** keras.models **import** load\_model *#FOR LOADING THE SAVED MODEL***import** glob *#FOR PROCESSES OF FOLDERS AND FILES***from** keras.preprocessing **import** image *#FOR PREDICTION OF MODEL***from** playsound **import** playsound *#FOR PLAYING SOUNDS*model = load\_model(**"lbp-relu-adam.h5"**) *#LOADING MODEL***global** result\_number *#PREDICTION VALUE OF MODEL***global** result\_total *#TOTAL OF PREDICTION VALUE***global** result\_avg *#AVARAGE OF TOTAL PREDICTION VALUE*camera = cv2.VideoCapture(0) *#FOR USING COMPUTER WEBCAM*

**class** Window(QWidget): *#IT CREATES CLASS FOR FORM SCREEN*

**def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.init\_ui()

**def** init\_ui(self):  
 self.button1 = QPushButton(self) *#CREATE A BUTTON* self.button1.setText(**"START"**) *#SET BUTTON TEXT* self.button1.resize(200,200) *#SET BUTTON RESIZE* self.button1.move(575,250) *#SET BUTTON LOCATION IN SCREEN* self.button1.clicked.connect(self.open\_frame) *#BUTTON CLICK METHOD* self.setWindowTitle(**"Motion Recognize"**) *#SET WINDOW TITLE* self.showMaximized() *#SET WINDOW GEOMETRY*

**def** open\_frame(self): *#BUTTON CLICK EVENTS* time1 = time.time() *#GIVES THE INSTANT TIME* **while True**: *#INFINITE LOOP IS CREATED TO OPENING THE WEBCAM* ret, frame = camera.read() *#THE IMAGE FROM TAKEN THE WEBCAM IS ASSIGNED TO THE VARIABLE* crop\_image = frame[50:430, 100:540] *# CROPING FRAME FOR RECTANGLE* frame2 = cv2.resize(crop\_image, (224, 224)) *# IMAGE RESIZE FOR PREDICTON OF MODEL* time2 = time.time() *#TAKES THE INSTANT TIME* diff = int(time2-time1) *#THE DIFFERENCE BETWEEN THE TWO TIMES IS FOUND* diff2 = 15-diff *#FOR COUNTDOWN* **if** cv2.waitKey(20) **and** diff < 15: *#THE IMAGE FROM TAKEN THE WEBCAM IS SET 20 FPS AND TO KEEP THE DISPLAY ON FOR 15 SECONDS.* frame = cv2.putText(frame, **"PLEASE TAKE YOUR CORRECT TRAINING"**, (5, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, **"POSITION WITHIN "** + str(diff2) + **" SECONDS."**,(5, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, str(diff2), (240, 240), cv2.FONT\_HERSHEY\_SIMPLEX, 5.0, (0, 255, 0), 6) *#TEXT IS ADDED ON FRAME* cv2.imshow(**'CAMERA'**, frame) *#THE IMAGE IS SHOWN* **else**:  
 **break** *#AFTER COUNTDOWN THE SCREEN CLOSES* result\_total = 0 *#THE FIRST VALUE IS ASSIGNED FOR RESULT\_TOTAL* result\_number = 0 *#THE FIRST VALUE IS ASSIGNED FOR RESULT\_NUMBER* result\_avg = 0 *#THE FIRST VALUE IS ASSIGNED FOR RESULT\_AVG* motion\_number = 0 *#DEFINE MOTION NUMBER DURING EXERCISE TIME AND THE FIRST VALUE IS ASSIGNED FOR MOTION\_NUMBER* correct\_number = 0 *#DEFINE CORRECT MOVE NUMBER DURING EXERCISE TIME AND THE FIRST VALUE IS ASSIGNED FOR CORRECT\_NUMBER* correct\_percent = 0.0 *#DEFINE CORRECT MOVE PERCENTAGE DURING EXERCISE AND THE FIRST VALUE IS ASSIGNED FOR CORRECT\_PERCENT* incorrect\_percent = 0.0 *#DEFINE INCORRECT MOVE PERCENTAGE DURING EXERCISE AND THE FIRST VALUE IS ASSIGNED FOR INCORRECT\_PERCENT*

**while True**: *#INFINITE LOOP IS CREATED TO OPENING THE WEBCAM* ret, frame = camera.read() *# THE IMAGE FROM TAKEN THE WEBCAM IS ASSIGNED TO THE VARIABLE* crop\_image = frame[50:430, 100:540] *# CROPING FRAME FOR RECTANGLE* frame2 = cv2.resize(crop\_image, (224, 224)) *# IMAGE RESIZE FOR PREDICTON OF MODEL* time3 = time.time() *#TAKES THE INSTANT TIME* diff = int(time3 - time1) *#THE DIFFERENCE BETWEEN THE TWO TIMES IS FOUND* diff2 = 20 - diff *#FOR COUNTDOWN* cv2.imwrite(os.path.join(**r'C:\Users\aydin\Desktop\motion\kamera'**, **'image.jpg'**), frame2) *#PICTURE IS CAPTURED FROM REAL TIME IMAGE FOR PREDICTION* **if** cv2.waitKey(20) **and** diff < 20: *#THE IMAGE FROM TAKEN THE WEBCAM IS SET 20 FPS AND TO KEEP THE DISPLAY ON FOR 5 SECONDS.* **for** img **in** glob.glob(**r"C:\Users\aydin\Desktop\motion\kamera\image.jpg"**): *# THE PICTURE TAKEN IS ASSIGNED TO THE VARIABLE FOR PREDICTION* test\_image = image.load\_img(img, target\_size=(224, 224)) *# THE PICTURE IS RESIZED* test\_image = image.img\_to\_array(test\_image) *# THE PICTURE IS CONVERTED TO ARRAY.* test\_image = np.expand\_dims(test\_image, axis=0) *# THE EXPAND\_DİMS() FUNCTION IS USED TO EXPAND THE SHAPE OF AN ARRAY.* result = model.predict(test\_image) *#THE PREDICTED VALUE THAT THE MODEL DETERMINES FOR THE PICTURE IS ASSIGNED TO A VARIABLE* result\_total = result\_total + int(result[0][0]) *#PREDICTION VALUES ARE SUMMED FOR AVERAGE PREDICTION.* result\_number = result\_number + 1 *#TO INCREASE result\_number*  
 frame = cv2.putText(frame, **"PLEASE WAIT IN THE CORRECT POSITION"**,(5, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, **"WHILE MAKING ADJUSTMENTS."**, (5, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, str(diff2), (240, 240), cv2.FONT\_HERSHEY\_SIMPLEX, 5.0, (0, 255, 0), 6) *#TEXT IS ADDED ON FRAME* cv2.imshow(**'CAMERA'**, frame) *# TO SHOW FRAME* **else**:  
 result\_avg = result\_total / result\_number *# TO GET AVERAGE OF RESULT* **break  
 while True**: *# INFINITE LOOP IS CREATED TO OPENING THE WEBCAM* ret, frame = camera.read() *# THE IMAGE FROM TAKEN THE WEBCAM IS ASSIGNED TO THE VARIABLE* crop\_image = frame[50:430, 100:540] *# CROPING FRAME FOR RECTANGLE* frame2 = cv2.resize(crop\_image, (224, 224)) *# IMAGE RESIZE FOR PREDICTON OF MODEL* time4 = time.time() *# TAKES THE INSTANT TIME* diff = int(time4 - time1) *# THE DIFFERENCE BETWEEN THE TWO TIMES IS FOUND* diff2 = 50 - diff *# FOR COUNTDOWN*cv2.imwrite(os.path.join(**r'C:\Users\aydin\Desktop\motion\kamera'**, **'image.jpg'**), frame2) *#PICTURE IS CAPTURED FROM REAL TIME IMAGE FOR PREDICTION* **if** cv2.waitKey(20) **and** diff < 50: *#THE IMAGE FROM TAKEN THE WEBCAM IS SET 20 FPS AND TO KEEP THE DISPLAY ON FOR 30 SECONDS.* **for** img **in** glob.glob(**r"C:\Users\aydin\Desktop\motion\kamera\image.jpg"**): *# THE PICTURE TAKEN IS ASSIGNED TO THE VARIABLE FOR PREDICTION* test\_image = image.load\_img(img, target\_size=(224, 224)) *# THE PICTURE IS RESIZED* test\_image = image.img\_to\_array(test\_image) *# THE PICTURE IS CONVERTED TO ARRAY.* test\_image = np.expand\_dims(test\_image,axis=0) *# THE EXPAND\_DİMS() FUNCTION IS USED TO EXPAND THE SHAPE OF AN ARRAY.* result = model.predict(test\_image) *# THE PREDICTED VALUE THAT THE MODEL DETERMINES FOR THE PICTURE IS ASSIGNED TO A VARIABLE* **if** (result\_avg\*0.998) <= result[0][0] <= (result\_avg\*1.02): *#A RANGE IS DETERMINED FOR THE ACCURACY OF THE ESTIMATED VALUE* prediction = **'CORRECT'** *#THE PREDICTION VARIABLE IS ASSIGNED A TEXT* frame = cv2.putText(frame, prediction, (20, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 255, 0),2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, str(diff2), (600, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.rectangle(frame, (100, 50), (540, 430), (0, 255, 0),3) *#TEXT IS ADDED ON FRAME* cv2.imshow(**"CAMERA"**, frame) *#THE IMAGE IS SHOWN* motion\_number = motion\_number + 1 *# TO INCREASE motion\_number ONE BY ONE* correct\_number = correct\_number + 1 *# TO INCREASE correct\_number ONE BY ONE* playsound(**'correct.wav'**) *#THE CORRECT SOUND IS PLAYED* **else**:  
 prediction = **'INCORRECT'** *#THE PREDICTION VARIABLE IS ASSIGNED A TEXT* frame = cv2.putText(frame, prediction, (20, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 0, 255), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, str(diff2), (600, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.rectangle(frame, (100, 50), (540, 430), (0, 255, 0), 3) *#TEXT IS ADDED ON FRAME* cv2.imshow(**"CAMERA"**, frame) *#THE IMAGE IS SHOWN* motion\_number = motion\_number + 1*# TO INCREASE motion\_number ONE BY ONE* playsound(**'incorrect.mp3'**) *#THE INCORRECT SOUND IS PLAYED* **else**:  
 correct\_percent = (correct\_number / motion\_number) \* 100 *#CORRECT MOTION PERCENTAGE IS CALCULATED* incorrect\_percent = 100 - correct\_percent *#INCORRECT MOTION PERCENTAGE IS CALCULATED* frame = cv2.putText(frame, **"Correct Exercise Percentes: %"** +str(correct\_percent), (20, 40), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, **"Incorrect Exercise Percentes: %"** +str(incorrect\_percent), (20, 70), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* frame = cv2.putText(frame, **"Exercise Time: 30 Seconds"**, (20, 100), cv2.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 255, 0), 2) *#TEXT IS ADDED ON FRAME* cv2.imshow(**"RESULTS"**, frame) *#THE IMAGE IS SHOWN* **break** camera.release() *#WEBCAM IS RELEASED* cv2.destroyWindow(**"CAMERA"**) *#CAMERA FRAME IS DESTROYED.*

app = QApplication(sys.argv) *#FORM SCREEN IS DEFINED*menu = Window() *#CREATING WINDOW FOR FORM SCREEN*sys.exit(app.exec\_()) *#FORM SCREEN IS CLOSED*

**APPENDIX 3: Code Blocks (model\_training.py)**

*#IMPORT LIBRARIES***from** keras.preprocessing.image **import** ImageDataGenerator *#FOR DATA AUGMENTATION***from** keras.models **import** Sequential *#FOR THE MODEL TO BE TRAINED***from** keras.layers **import** Conv2D, MaxPooling2D *#FOR CONVOLUTIONAL LAYERS***from** keras.layers **import** Activation, Dropout, Flatten, Dense *#FOR CONVOLUTIONAL LAYERS***from** keras **import** backend **as** K *#FOR KERAS BACKEND PROCESSES*

train\_data\_dir = **r'C:\Users\aydin\Desktop\svm\dataset\train'** *#FILE PATH OF THE IMAGES TO BE GIVEN TO THE MODEL FOR TRAIN*validation\_data\_dir = **r'C:\Users\aydin\Desktop\svm\dataset\valid'** *#FILE PATH OF THE IMAGES TO BE GIVEN TO THE MODEL FOR VALIDATION*nb\_train\_samples = 720 *#NUMBER OF TRAIN IMAGES*nb\_validation\_samples = 120 *#NUMBER OF VALIDATION IMAGES*epochs = 1 *#TRAINING NUMBER FOR MODEL*batch\_size = 20 *#DURING TRAINING, THE MODEL WORKS WITH GROUPS OF BATCH\_SIZE IMAGES.*img\_width, img\_height = 224, 224 *#SIZE OF TRAIN AND VALIDATION IMAGES  
#ACCORDING TO DATA FORMAT, INPUT\_SHAPE IS DETERMINED. (3, 224, 244) OR (22,4, 224, 3)***if** K.image\_data\_format() == **'channels\_first'**:  
 input\_shape = (3, img\_width, img\_height)  
**else**:  
 input\_shape = (img\_width, img\_height, 3)

*#THE MODEL IS DESIGNED IN THIS PART.*model = Sequential() *#A SEQUENTIAL MODEL IS CREATED.*model.add(Conv2D(32, (3, 3), input\_shape=input\_shape)) *#A CONVOLUTIONAL LAYER WITH 32 NEUROS IS CREATED. KERNEL SIZE 3X3 AND INPUT\_SHAPE IS INDICATED.*model.add(Activation(**'relu'**)) *#THE ACTIVATION FUNCTION IS RELU.*model.add(MaxPooling2D(pool\_size=(2, 2))) *#MAX\_POOLING 2X2 IS ADDED TO CONVOLUTIONAL LAYER.*model.add(Conv2D(32, (3, 3), input\_shape=input\_shape)) *#A CONVOLUTIONAL LAYER WITH 32 NEUROS IS CREATED. KERNEL SIZE 3X3 AND INPUT\_SHAPE IS INDICATED.*model.add(Activation(**'relu'**)) *#THE ACTIVATION FUNCTION IS RELU.*model.add(MaxPooling2D(pool\_size=(2, 2))) *#MAX\_POOLING 2X2 IS ADDED TO CONVOLUTIONAL LAYER.*model.add(Dropout(0.25)) *#THE DROPOUT PROCESS IS IMPLEMENTED*model.add(Flatten()) *#FLATTENING TWO FULLY CONNECTED LAYERS*model.add(Dense(512)) *#ADDING TWO FULLY CONNECTED LAYERS:*model.add(Activation(**'relu'**)) *#THE ACTIVATION FUNCTION IS RELU.*model.add(Dropout(rate=0.5)) *#THE DROPOUT PROCESS IS IMPLEMENTED*model.add(Dense(1)) *#MAKE FULLY CONNECTED MODEL*model.add(Activation(**'sigmoid'**)) *#THE ACTIVATION FUNCTION IS SIGMOID FOR OUTPUT LAYER.*model.compile(loss=**'binary\_crossentropy'**, *#MODEL COMPILED* optimizer=**'adam'**,  *#ADAM IS USED FOR OPTIMIZATION* metrics=[**'accuracy'**]) *#FIND MODEL ACCURACY*

train\_datagen = ImageDataGenerator(rotation\_range=40, width\_shift\_range=0.2,  
 height\_shift\_range=0.2, rescale=1./255, shear\_range=0.2, zoom\_range=0.2,  
 horizontal\_flip=**True**, fill\_mode=**'nearest'**) *#DATA AUGMENTATION FOR TRAIN IMAGES*  
test\_datagen = ImageDataGenerator(rescale=1./255) *#DATA AUGMENTATION FOR VALIDATION IMAGES*train\_generator = train\_datagen.flow\_from\_directory(train\_data\_dir, target\_size=(img\_width, img\_height), batch\_size=batch\_size, class\_mode=**'binary'**) *#TRAIN DATA PREPARE FOR MODEL*  
validation\_generator = test\_datagen.flow\_from\_directory( validation\_data\_dir,  
 target\_size=(img\_width, img\_height),  
 batch\_size=batch\_size, class\_mode=**'binary'**) *#VALIDATION DATA PREPARE FOR MODEL*

history = model.fit\_generator(train\_generator,  
 steps\_per\_epoch=nb\_train\_samples // batch\_size,  
 epochs=epochs, validation\_data=validation\_generator,  
 validation\_steps=nb\_validation\_samples // batch\_size)  
print(model.summary()) *#PRINT MODEL SUMMARY*model.save(**"norm-relu-adam15.h5"**) *#SAVE MODEL TRAINED*