令和４年度卒業研究報告書

Web カメラによる視線トラッキングに関する研究

情報技術科　小松　篤志

指導教員　　ソソラ

目次

第 1 章　はじめに

近年，非接触の操作デバイスの需要が拡大しつつある．そこに，音声認識や体を使ったジェスチャーを画像認識するなどの多くの手法が使われている．一方，ある病気の場合は眼だけは動くが他の身体の部位は動かすことができないという事例があり，ジェスチャーだけでは操作できない．ここで，「眼の動きを使った表現」ができれば聴覚に障害を持った人 でも使えると思った．

そこで，コミュニケーション手段の一つして「眼の動き」に注目し，眼の動きをトラッキングして何かを操作できるツールを製作したいと考えた．

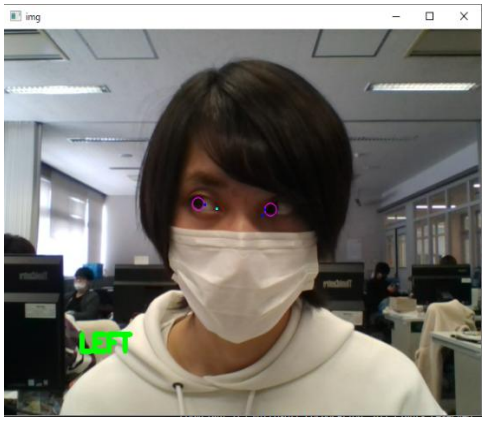
研究は以下の通りに進めた．

OpenCV や MediaPipe（FaceMesh）などの使い方を学習しながら，開発環境を設定した．

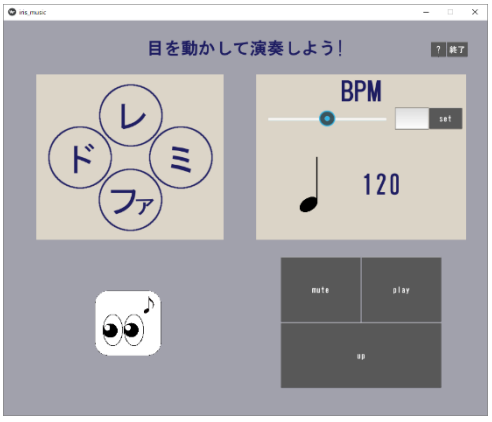
市販の WEB カメラを使用し，リアルタイムで眼の輪郭を含むキーポイントを取得し，EXCEL ファイルに書き出した．

キーポイントの座標データの解析による視線方向（右左上下）を判定するアルゴリズムを考えた．

取得した視線方向を活用し，音楽を演奏することができるデスクトップアプリケーションを作成した．



左を見ている様子



アプリメイン画面

第 2 章　研究概要

2.1 概要

本研究では, 特殊なハードウェアを必要とせずに, WEB カメラを使用してリアルタイムで虹彩, 瞳孔, および眼の輪郭を含むキーポイントにより, 眼の動きを追跡するアルゴリズムを開発する.

さらに眼の動きを利用したアプリケーションを作成する.

2.2 開発環境

本研究の開発環境を以下の表に示す. また, 開発環境のインストールおよびライブラリの説明は付録にて解説する.

OS	Windows 10
統合開発環境	Visual Studio Code
開発言語	Python 3.8.10
使用ライブラリ	OpenCV MediaPipe faceMesh Pandas Numpy Kivy Concurrent.ftures Winsound
カメラ	Integrated Camera(標準装備, 720p 30fps) C920n pro hd webcam(1080p, 30fps)

表 1　開発環境

第 3 章　眼の検出について

3.1 カメラの入力・画面表示方法

本研究のカメラ認識には「Opencv」を使用している．カメラの入力および画面出力の流れは下図に示す．

```python
import cv2 as cv

cap = cv.VideoCapture(0)    #接続している力メラの選択

while True:
    ret, frame = cap.read() #1フレーム読み込み
    if not ret:
        break
    frame = cv.flip(frame, 1) #表示される画像を鏡向きにする


    cv.imshow('img', frame) #画面表示
    key = cv.waitKey(1)     #キー入力待ち
    if key == ord('q'):     #終了条件
        break
```

図 1　カメラ入力から表示までのソースコード

VideoCapture()の引数で内蔵カメラ(標準カメラ)または後付けカメラの選択となる．対応引数は以下の表に示す．

引数に渡す値	認識する画像
0	内蔵カメラ (内蔵カメラがない場合)後付けカメラ
1	内蔵カメラ以外の後付けカメラ
動画のファイルパス	指定した動画

表 2　VIdeoCapture()の引数対応表

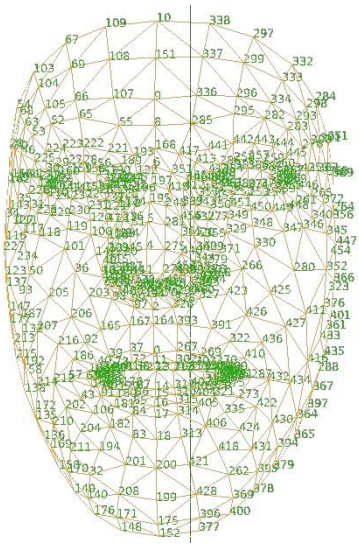
cap.read()は入力画像を1フレームしか読み込むことができないのでwhileの無限ループで読み込み，フレームを更新している．

waitKey(1)以降(図1: 13行目以降)などを使用し，無限ループを終了させる方法を記述しなければならない．

3.2 眼の検出

mediaPipe FaceMesh を利用し目の検出をする．顔に割り当てられている468点のポイントのうち以下の図に示す40点のポイントを利用する(以降これらのポイントをキーポイントと呼ぶ)．

参考文献【1】参照



```
5   #目に割り当てられているキーポイントの値
6   LEFT_EYE = [362, 382, 381, 380, 374, 373, 390, 249, 263, 466, 388, 387, 386, 385, 384, 398]
7   RIGHT_EYE = [33, 7, 163, 144, 145, 153, 154, 155, 133, 173, 157, 158, 159, 160, 161, 246]
8   LEFT_IRIS = [474, 475, 476, 477]
9   RIGHT_IRIS = [469, 470, 471, 472]
```

図 2　眼に割り当てられたポイントの値

Facemesh を利用するために python の with 構文を使用する. With 構文はファイルのオープンからクローズまでを簡単にするものである.

Facemesh で画像を扱うためには cv2 で読み込んだ画像値(BGR)を標準値(RGB)に変換する必要がある.

虹彩の中心座標は図 3 の 41, 42 行目で計算している.

```
14  mp_face_mesh = mp.solutions.face_mesh
15
16  with mp_face_mesh.FaceMesh(
17      max_num_faces=1,
18      refine_landmarks=True,
19      min_detection_confidence=0.5,
20      min_tracking_confidence=0.5
21  ) as face_mesh:
22      while True:
23          ret, frame = cap.read()
24          if not ret:
25              break
26          frame = cv.flip(frame, 1)
27          rgb_frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)    #画像形式を変更(BGR → RGB)
28          img_h, img_w = frame.shape[:2]                      #画像の高さ，幅を取得
29          results = face_mesh.process(rgb_frame)              #facemeshに画像データを与える
30          if results.multi_face_landmarks:
31              #画像の中の眼の位置とキーポイントの対応付け
32              mesh_points = np.array([np.multiply([p.x, p.y], [img_w, img_h]).astype(int)
33                                      for p in results.multi_face_landmarks[0].landmark])
34              #目全体
35              #cv.polylines(frame, [mesh_points[LEFT_EYE]], True, (0, 255, 0), 1, cv.LINE_AA)
36              #cv.polylines(frame, [mesh_points[RIGHT_EYE]], True, (0, 255, 0), 1, cv.LINE_AA)
37              #瞳孔のみ(四角)
38              # cv.polylines(frame, [mesh_points[LEFT_IRIS]], True, (0, 255, 0), 1, cv.LINE_AA)
39              # cv.polylines(frame, [mesh_points[RIGHT_IRIS]], True, (0, 255, 0), 1, cv.LINE_AA)
40              #瞳孔のみ(〇)表示
41              (l_cx, l_cy), l_radius = cv.minEnclosingCircle(mesh_points[LEFT_IRIS])
42              (r_cx, r_cy), r_radius = cv.minEnclosingCircle(mesh_points[RIGHT_IRIS])
43              center_left = np.array([l_cx, l_cy], dtype=np.int32)
44              center_right = np.array([r_cx, r_cy], dtype=np.int32)
45              cv.circle(frame, center_left, int(l_radius), (255,0,255), 1, cv.LINE_AA)
46              cv.circle(frame, center_right, int(r_radius), (255,0,255), 1, cv.LINE_AA)
47
48          cv.imshow('img', frame) #画面表示
49          key = cv.waitKey(1)     #キー入力待ち
50          if key == ord('q'):     #終了条件
51              break
```

図 3　眼を認識する操作のソースコード

以上でカメラから眼の認識を行う処理の完成である.この時点で紫色の円が目の動きに合わせて動いている.

第 4 章　視線方向検出

4.1 全体構造について

本研究では上下左右の 4 方向の視線を判定することにした. 判定結果を返すアルゴリズムが 3 つあり, 左右の判定, 上の判定, 下の判定の 3 つを製作した. 左右の判定は同じアルゴリズムで解決することができたが, 上下の判定は目の大きさ, 各キーポイントの移動距離など, 判定に影響する部分の共有ができなかった. これらのことから上下の判定はアルゴリズムを別のものにした. 構造のイメージ図を以下に示す.

図 4　視線判定アルゴリズムの構造

4.2 左右の判定

4.2.1　初期案

まずは右を向いた時の判定を作成しようとした. その時に用いた方法が虹彩(黒目)の中心座標の変化を過去の時間との比較で判定するものだ. 虹彩の中心座標を利用し, 「50ms 前の x 座標より現在の x 座標が 5 以上増えていたら右を見ている.」というような判定を作成した. すると, 右を見た一瞬は判定が出るが継続してその方向を見ていると, 判定が出ないというようなことが起きてしまった. 加え

て，この段階では 1 つのポイントしか使用しておらず，顔全体が移動したときも同様の判定が出てしまうことから初期構想は使用することができなかった．

4.2.2　正式アルゴリズム案

まず考えなければならないのが，顔が横にずれてしまっても目の判定に影響が及ばないようにすることである．これを実現するために目の動きを相対的な距離で判定できるように設計した．目頭と虹彩の中心との間の距離を利用する方法である．このようにすることで顔が横にずれてしまっても判定に影響する値に変化が生じにくくなった．さらにこの値を使用することにより初期案で問題になっていた継続的な判定の出力ができない問題を解決できる．また，左右それぞれの眼のデータを使うことでさらに精度の良い判定結果が得られる．

4.2.3　アルゴリズムの詳細

(1)　キーポイント取得

眼のキーポイントのうち図 5 に示す①～⑤（①―左目頭，②―右目頭，③―左虹彩の中心，④―右虹彩の中心，⑤―目頭と目頭の中心）のキーポイントの座標を取得する．

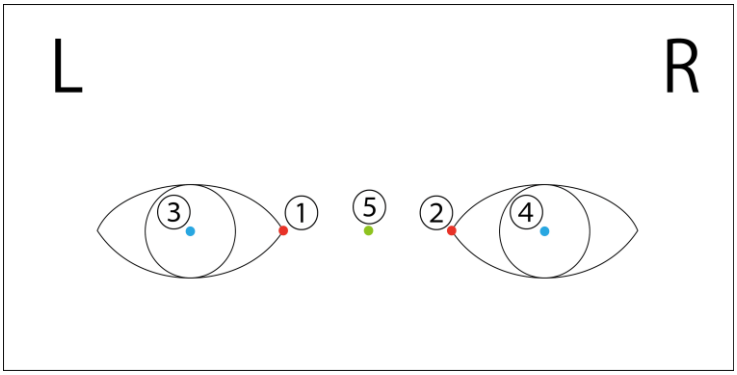
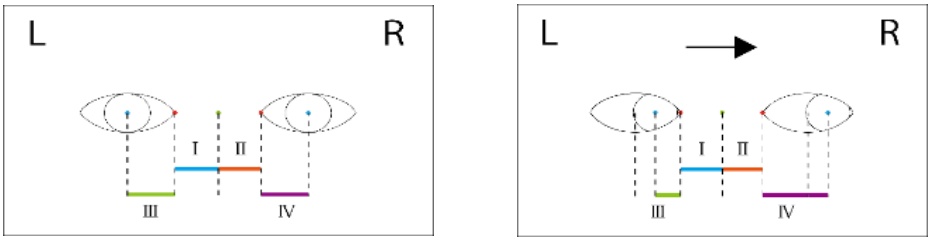


図 5　左右判定に必要なキーポイント

(2) 距離測定

キーポイント①～⑤の座標から異なるキーポイント同士の距離を求める．ここで，i 番目のキーポンの座標 Pi(xi,yi)，j 番目のキーポイントの座標を Pj(xj,yj)とすると *i,j* の距離 *Di-j* を式 1 により求める．そして，図 5.2（ア）には前へ向いている初期状態，（イ）には眼球を右に動かす状態の距離をそれぞれ色別で示した．

$$D_{i\text{-}j} = \sqrt{(Xi - Xj)^2 + (Yi - Yj)^2} \qquad (1)$$



（ア）　　（イ）

図 5.2　眼を右へ動かしたときの変化の様子

(3) 距離の変化の様子

頭を動かさない状態で，眼を右に動かす場合は $D_{3\text{-}1}$ と $D_{2\text{-}4}$ の値が変化される．

つまり，眼球を右に動かすと $D_{3\text{-}1}$（III）の値がある閾値より小さくなり，$D_{2\text{-}4}$（IV）の値もある閾値より大きくなる．

4.2.4　実験・検証

(1)　実験内容

4.2.3 で提案したアルゴリズムのIIIとIVの距離の変化をグラフ化して比較する.

(2)　実験条件

実験に用いたカメラは PC に標準で装備されている Integrated Camera(内蔵カメラ)を使用. 測定の流れは, ①カメラの前に座る. ②カメラから 40cm 離れる. ③指定の動きをする. これらの流れで測定する. この条件はすべてのアルゴリズムの実験で使用する.

(3)　検証した動き

左右の判定に検証した動き(視線移動プロセス)は以下の通りだ.

図 6　視線移動プロセス

(4)　実験結果

以下が実験によって得られたデータの一つである.

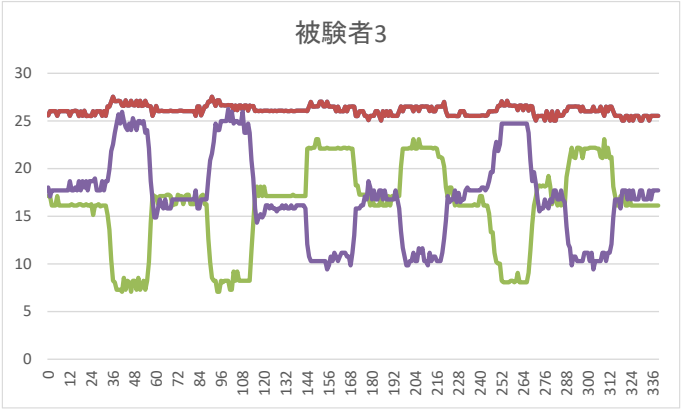


図 7 実験結果 1

横軸に時間，縦軸にそれぞれの間の距離を取っている．赤い線のデータは目頭間の値であり，カメラと顔との距離の指標として使用.

グラフと目の動きの対応を以下の図に示す.

正面を向いているときは互いの距離はおよそ同じなので直線状になる.

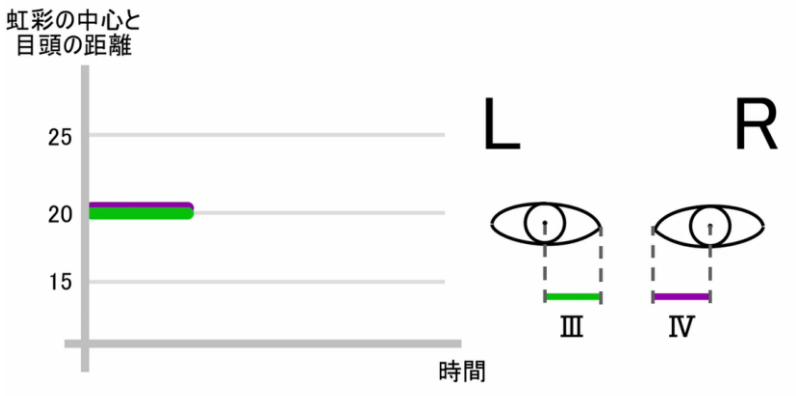


図 8　正面を向いているときのグラフと眼の対応

右を向くとそれぞれの値が変化する.

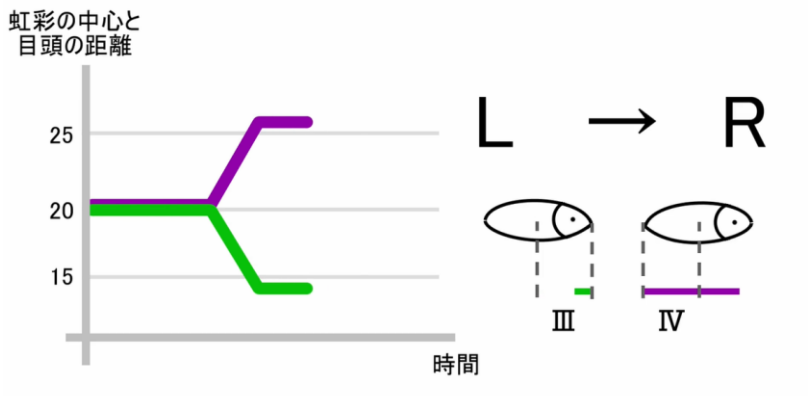


図 9　右を向いた時のグラフと眼の対応

右を見たときはⅢの値が少なくなり，グラフでも緑の線が低い値を取る．逆にⅣの値は増加するのでグラフ上でも高い値を取る.

左を向いた時は図 9 と逆のようになり，その様子を図 10 とする.

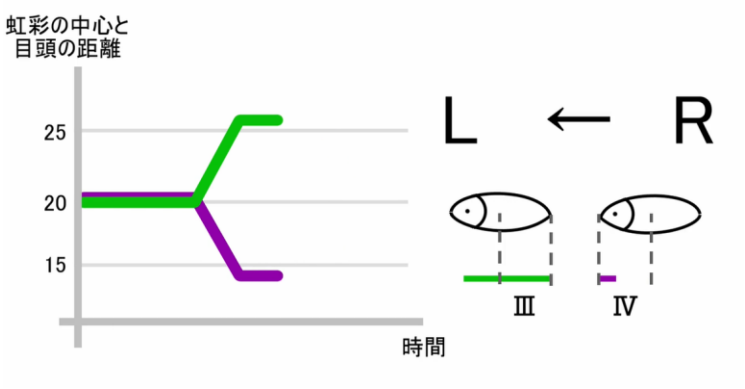


図 10　左を向いた時のグラフと眼の対応

(5)　閾値の設定

実験結果から同じ条件下であればある一定の値をそれぞれとることが分かった．そこで，左右判定に用いる閾値を図 11 の通り計測値の差で設定することにした．

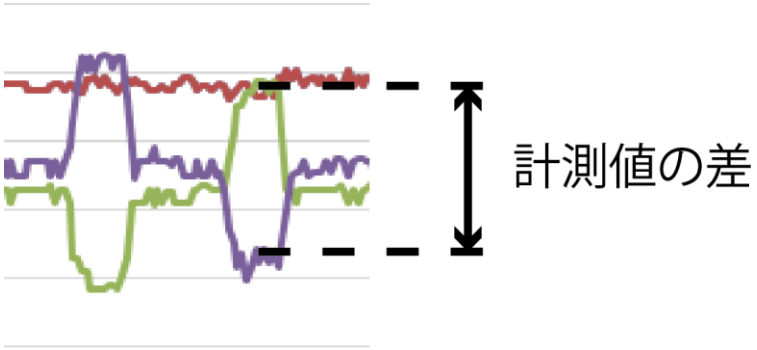


図 11　閾値に利用する値

実験データをもとに閾値を 10 と設定し，その値を超えたとき判定結果を返す．

(6)　閾値に利用する値の設定理由

初めはこのデータの左右それぞれ上の値を取ったとき，つまり正面を向いているときよりⅢが伸びるまたはⅣが伸びるときのように値が大きくなったときに判定を出そうとしたが，より目や斜視を考慮した結果この値が正確な左右を取れることが分かったので採用した．

(7) 実装(事前準備)

座標取得は指定された点の位置に画面サイズを掛けることで得られる．また，x 座標と y 座標を一つの配列にまとめることで一つのキーポイントとして扱うことができ，管理しやすくなる．

座標取得メソッドを用意した．座標の取り方の例としてキーポイントを一つだけ抽出したソースコードを図 12 に示す．他のキーポイントを取得したソースコードはすべてのアルゴリズムをまとめて後ほど掲載する．

```
48  #目頭のキーポイント座標抽出
49  def get_keypoint(results, img_h, img_w):
50      left_inner_corner_eye_x = int(results.multi_face_landmarks[0].landmark[362].x * img_w)
51      left_inner_corner_eye_y = int(results.multi_face_landmarks[0].landmark[362].y * img_h)

                                  省略

68      eye_inner_L = np.array([left_inner_corner_eye_x, left_inner_corner_eye_y])
```

図 12　座標取得方法例

次に配列「data」を用意する．最終的な配列の中身は以下の図の通りになる．

```
["中心から左目頭", "中心から右目頭", "左目頭から虹彩の中心", "右目頭から虹彩の中心",
 "上下右", "上下左", "左右判定", "目の面積右", "目の面積左", "右目下距離", "左目下距離"]
```

図 13　配列「data」の中身

次に視線の方向を取得するメソッド「eye_direction()」を作成する．引数は図 14 の通り．

eye_direction()は返値を持っており，正面を見ている状態(どの判定も出ていない状態)は 0 を返す．

```
101  #4方向計算
102  def eye_direction(frame,
103                    data,
104                    center_right,
105                    center_left,
106                    lineToPoint_R,
107                    lineToPoint_L,
108                    lineToPoint_R_dist,
109                    lineToPoint_L_dist,
110                    eye_area,
111                    R_low_dist,
112                    L_low_dist):
```

図 14　eye_direction()の引数

(8) 実装(左右アルゴリズム)

最後に eye_direction()に左右の判定プログラムを実装し完成する．左右の返値はそれぞれ【右:5 左:-5】となっている．図 15 が判定プログラムである．

```
147     #右方向
148     if data[3] - data[2] > LR_THRESHOLD:
149         cv.putText(frame,
150                    text="RIGHT",
151                    org=(100, 400),
152                    fontFace=cv.FONT_HERSHEY_SIMPLEX,
153                    fontScale=1.0,
154                    color=(0,255,0),
155                    thickness=5)
156         return 5
157     #左方向
158     if data[3] - data[2] < -LR_THRESHOLD:
159         cv.putText(frame,
160                    text="LEFT",
161                    org=(100, 400),
162                    fontFace=cv.FONT_HERSHEY_SIMPLEX,
163                    fontScale=1.0,
164                    color=(0,255,0),
165                    thickness=5)
166         return -5
```

図 15　左右判定プログラム部分のソースコード

cv.putText()は opencv の機能で出力画面の中に文字列を表示させることができる．

実装が完了して判定結果を表示させるようにした結果を図 16, 17 に示す．

図 16 左を判定している様子

図 17　右を判定している様子

4.3 下向きの判定

4.3.1　初期案

左右の判定と同様に虹彩の中心と他のどこかとの距離変化を利用しようと考えた．しかし，眼の上下の動きを観察したところ，眼球のみが変化するのではなく，瞼なども同時に動いてしまうため左右の判定同様の仕様では解決できなかった．

4.3.2　正式アルゴリズム案

眼の動きを観察したところ，下を向いた時は白目を含む目の面積が大きく変化することが分かった．通常時(前を向いているとき)より値が下がったら下を見ている判定が返る．しかし，これだけでは問題がある．それは，人によって目の大きさが異なってしまう点である．そこで，判定条件をもう一つ加える．

さらに目の動きを観察したところ，目じりの動きは上下瞼に比べ大きな動きをすることはなかった．さらに左右の目と目の間の中心点(目全体の中心)もまた，固定点として使用することが可能であった．これら 2 点を結んだ直線を境界線(以降「下向き基準線」と呼ぶ)として，その線より虹彩の中心座標が低くなった時に判定を出すように考案した．

4.3.3　アルゴリズムの詳細

(1)　キーポイント取得

眼のキーポイントのうち図 18 に示す（③―左虹彩の中心，④―右虹彩の中心，⑤―目頭と目頭の中心，⑩―左目じり，⑪―右目じり）キーポイントの座標を取得する．

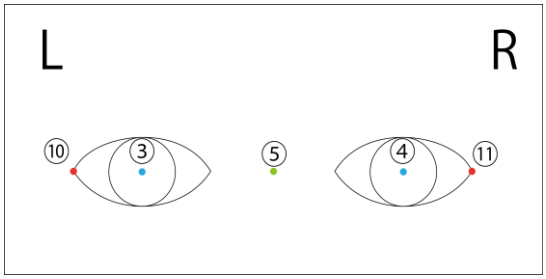


図 18　下判定に必要なキーポイント

（2） 距離，面積の測定

2 点を通る直線から離れた位置にある点までの距離を求める．基本的な計算はベクトルを使用したものである．参考文献【3】を参照.

```
171 def pointToLine(line_a, line_b, point):
172     # a(x1, y1) = (2, 2)
173     x1 = line_a[0]
174     y1 = line_a[1]
175     # b(x2, y2) = (6, 4)
176     x2 = line_b[0]
177     y2 = line_b[1]
178     # c(x3, y3) = (4, 5)
179     x3 = point[0]
180     y3 = point[1]
181     u = np.array([x2 - x1, y2 - y1])
182     v = np.array([x3 - x1, y3 - y1])
183     L = abs(np.cross(u, v) / np.linalg.norm(u))
```

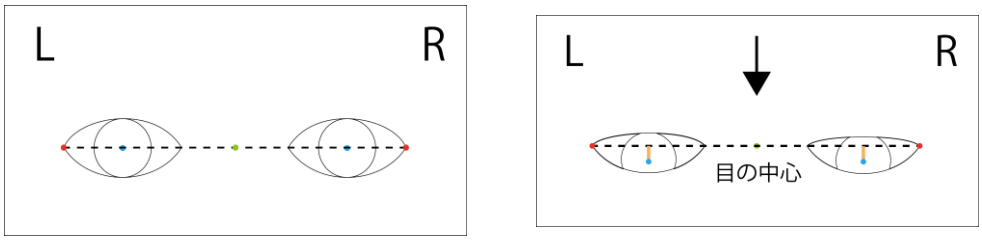
図 18　線と点の距離

加えて目の面積を取得する．目の面積は Opencv に搭載されている contourArea()を使用する．キーポイントの配列を渡すことでその点によって囲われた部分の面積を取得できる.

```
eye_area_L = cv.contourArea(np.array(mesh_points[LEFT_EYE]))
eye_area_R = cv.contourArea(np.array(mesh_points[RIGHT_EYE]))
```

図 19　目の面積の取得

眼を動かしたときの変化の様子を以下の図に示す.



（ア）　　（イ）

図 18.2　眼を右へ動かしたときの変化の様子

（3）　距離変化の様子

通常時(正面を見ているとき)に比べ下を見たときは下向き基準線より下に来ているのがわかる．さらに目の面積も小さくなっている.

4.3.4　実験・検証

（1）　実験内容

二つの実験を行う．一つは虹彩の中心位置の変化量の測定をすること．もう一つは目の面積の変化量を測定することである.

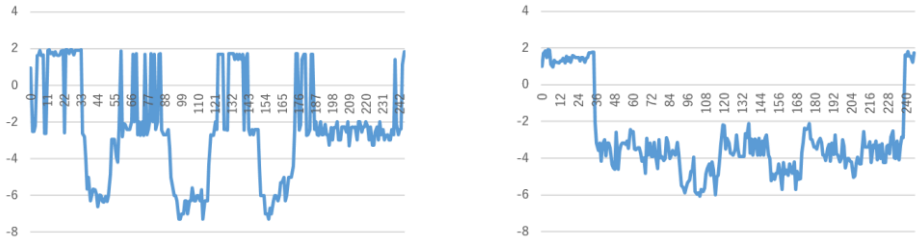
（2）　検証した動き

下の判定に検証した動き(視線移動プロセス)は以下の通りだ.

図 20　下の視線移動プロセス

(3) 実験結果(虹彩の中心と下向き基準線)

以下が虹彩の中心と下向き基準線の距離の変化によって得られたデータの一つである.



左眼　　右眼

図 18.3　虹彩の中心と線の距離の変化

このグラフは横軸に時間, 縦軸に下向き基準線と虹彩の中心の距離を持っている.

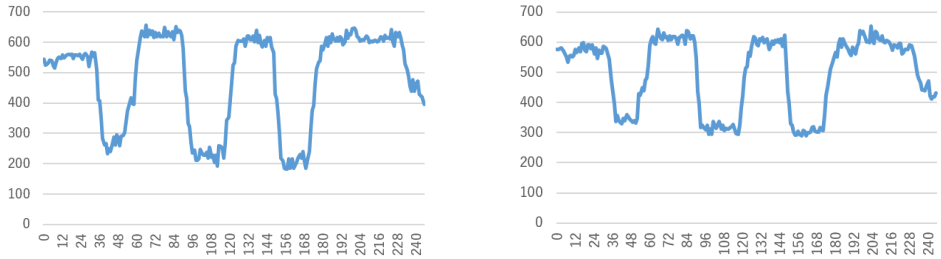
下を向くと虹彩の中心点は下向き基準線より下に動くため値がマイナス値を取得する.

(4) 閾値の設定(虹彩の中心と下向き基準線)

今回は単純にグラフの谷部分(判定がわかりやすい部分)で閾値を設ける. 絶対値へ変換し値は「3」と設定する.

(5) 実験結果(目の面積)

以下が目の面積を利用したデータである.



左眼　　右眼

図 18.4　虹彩の中心と線の距離の変化

このグラフは横軸に時間，縦軸に面積を持っている．

面積の 600 付近が通常時(正面を見ている)の状態で，値がそれよりも下がったときに下を見ていることになる．

(6) 閾値の設定(目の面積)

面積の閾値も単純に，ある値よりも数値が下がったら判定する方式にするため，値を「270」と設定する．この値よりも小さくなったら下を見ている判定が返る．

(7) 実装

視線方向を取得するメソッド eye_direction()に下判定プログラムを実装する．

```
#下方向
if center_right[1] > lineToPoint_R[1] and lineToPoint_R_dist > 3 or center_left[1] > lineToPoint_L[1] and lineToPoint_L_dist > 3:
    if eye_area[0] < 270 or eye_area[1] < 270:
        cv.putText(frame,
                   text="DOWN",
                   org=(100, 400),
                   fontFace=cv.FONT_HERSHEY_SIMPLEX,
                   fontScale=1.0,
                   color=(0,255,0),
                   thickness=5)
        return -10
```

図 21　下判定のソースコード

実装が完了し，判定結果を表示させるようにした結果を図 24 に示す．

図 22　下判定実装完了

4.4 上向きの判定

4.4.1　初期案

上判定のアルゴリズムも下判定と同様に虹彩の中心位置の変化と目の面積の変化を利用しようとした．しかし，虹彩の中心位置を絶対値で取得しているため同じ手法を用いるのは危険だった．加えて，目の面積は変化が非常に小さかった．以上のことから下判定と同じ手法を用いることができなかった．

4.4.2　正式アルゴリズム案

上を向いたときに利用できる見込みがある目の位置が一つだけ確認できた．それが虹彩の下部分と下瞼との間の距離である．基本的に通常時(正面を向いているとき)は虹彩の下部分は下瞼に隠れている．しかし，上を向いたときには白目が露出する．この変化を利用することにした．

4.4.3　アルゴリズムの詳細

(1)　キーポイントの取得

眼のキーポイントのうち図 23 に示す⑥～⑨（⑥―左虹彩の下，⑦―右虹彩の下，⑧―左下瞼の中心，⑨―右下瞼の中心）キーポイントの座標を取得する．

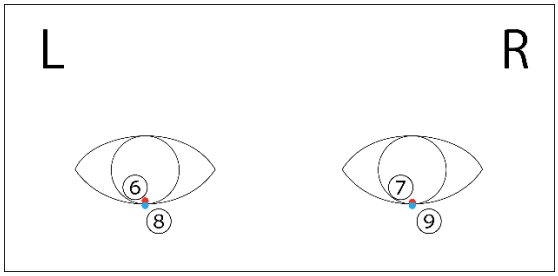
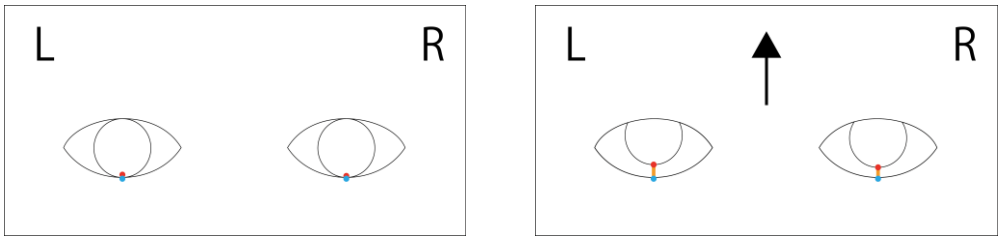


図 23　上判定に必要なキーポイント

(2)　距離の測定

線 6-8, 線 7-9 の y 座標の差で判定を製作する.

眼を動かしたときの変化の様子を以下の図に示す.



(ア)　　(イ)

図 23.2　眼を上へ動かしたときの変化の様子

(3)　距離変化の様子

眼を上方向に動かすことで虹彩の下部分と下瞼の中心部分に距離の差が生まれる.

4.4.4　実験・検証

(1)　実験内容

通常値(正面を向いているとき)の値がどの値をとっているのか検証する. 変化後の値を測定する.

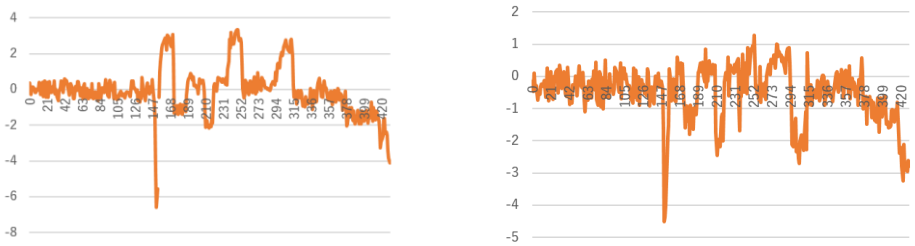
(2)　検証した動き

上の判定に検証した動き(視線移動プロセス)は以下の通りだ.

図 24　視線移動プロセス

(3) 実験結果 1

以下の結果が虹彩の下部分と下瞼の中心部分の距離の変化で得られたデータである.



左眼　　右眼

図 23.3　虹彩の中心と線の距離の変化

このグラフは横に時間軸, 縦に計測した距離の値を所有している.

通常時(正面を向いているとき)の値は 0 を基準に±0.5 付近であることが分かった.

データからわかる通り, 今までの計測値に比べると取得できる最大値が小さい. このままでは正面を見ているときの誤差値(波打っている部分)と判定値(山の部分が)わかりにくいので改善が必要である.

(4) 改善案

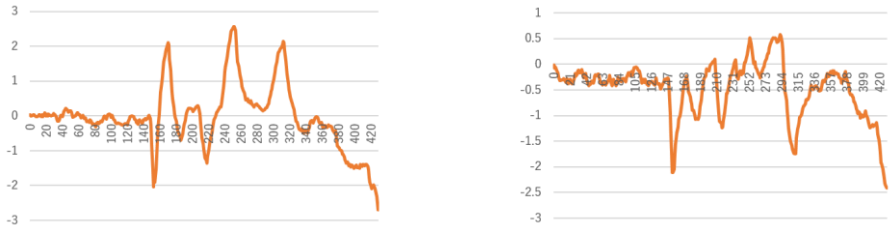
誤差の値を均すために指数移動平均法を用いる. 移動平均法は時系列データを平滑化する手法で, さらに指数関数的に行うことで直近の値に近い形で平均をとることができるアルゴリズムである. この方法をプログラムに落とし込んだものを図に示す.

```
#指数移動平均
def ema(data):
    volume = 15
    ema = np.zeros(len(data))
    ema[:] = np.nan
    ema[volume - 1] = np.mean(data[:volume])

    for section in range(volume, len(data)):
        ema[section] = ema[section - 1] + (data[section] - ema[section - 1]) / (volume + 1) * 2

    return ema[-1]
```

図 25　指数移動平均法

(5) 実験結果(改善後)

以下が移動平均を用いて取得し直したものである.



左眼　　右眼

図 23.4　虹彩の中心と線の距離の変化

改善前に比べて判定結果がわかりやすくなった.

さらに，データを均すことで通常時で誤差が±0.5 を超えることはなくなった.

(6) 閾値の設定

通常時に誤差値が±0.5 を取らない. 加えて，今回は上判定のみを作成することから，マイナスの値は考慮しなくてよい. 以上のことから閾値を「0.5」と設定し，その値を上回ったら上判定が返るようにする.

(7) 実装

視線方向を取得するメソッド eye_direction()に下判定プログラムを実装する.

```
#上方向
if R_low_dist > 0.5 or L_low_dist > 0.5:
    cv.putText(frame,
               text="UP",
               org=(100, 400),
               fontFace=cv.FONT_HERSHEY_SIMPLEX,
               fontScale=1.0,
               color=(0,255,0),
               thickness=5)
    return 10
```

図 26　上判定のソースコード

実装が完了し，判定結果を表示させるようにした結果を図 24 に示す.

図 27　上を判定している様子

4.5 eye_direction()の引数

eye_direction()の引数の内容を表にまとめる.

変数，配列名	内容	[0 番目の内容, 1 番目の内容]
frame	Opencv で読み取った画像情報	
data	図 13 を参照	
center_right	右虹彩の中心座標	[x 座標, y 座標]
center_left	左虹彩の中心座標	[x 座標, y 座標]
lineToPoint_R	目の中心と右目じりの線の中心座標	[x 座標, y 座標]
lineToPoint_L	目の中心と左目じりの線の中心座標	[x 座標, y 座標]
lineToPoint_R_dist	目の中心と右目じりの線から右虹彩の中心までの距離	
lineToPoint_L_dist	目の中心と左目じりの線から左虹彩の中心までの距離	
eye_area	目の面積	[左目の面積，右目の面積]
R_low_dist	右虹彩下部分と下瞼の距離	
L_low_dist	左虹彩下部分と下瞼の距離	

表 3　eye_direction()の引数表

第 5 章　視線方向取得モジュールとしての実装

5.1 外部プログラムからの呼び出し方法

本研究で作成された視線方向取得プログラムは様々な用途で使われることを前提として作る必要があるため，メインクラス(メインファイル)とは別のクラス(ファイル)として扱う．

ここで設計上の問題点として，外部から呼ばれる機能であるにも関わらず，無限ループを使用している点が挙げられる．このループしているメソッド(視線判定メソッド)を外部から単純に呼び出そうとすると視線判定メソッドからメインプログラムに帰ってくることができない．そこで使用される技術として，「マルチスレッド」が必要となる．マルチスレッドとはアプリケーションのプロセス(タスク)を複数のスレッドに分けて並行処理する方式を指す．よってメインプログラムからはマルチスレッドを使用して視線判定メソッドを起動させなければならない．

メインのプログラムによっては視線判定メソッドから直接判定結果を受け取ることができない仕様で作成しなければならないことがある．そのために，値取得用のメソッドも追加で作成する．

すべてのプログラムを終了するときに外部からループを止める必要性がある．ループを止めなければフリーズの原因になることから，ループ停止用プログラムも実装する．

視線方向取得プログラムを外部モジュール化したものを「iris.py」とする．

5.2 外部から呼ばれるメインとなるメソッド

先述している while でのループ処理が視線判定の主要機関である．これを外部から実行するためにメソッドとして扱う．名前を iris_cerector()として作成し，メインのループ部分をネストする．その様子を以下の図に示す.

```
216   #メインプログラムで起動されるモジュール
217   def iris_cerector():
218       #初期化
219       r_cx = 0
220       l_cx = 0
221       r_cy = 0
222       l_cy = 0
223       eye_area = 0
224       R_low_average = [0] * 15
225       L_low_average = [0] * 15
226
227       dataPOS = pd.DataFrame() #データ書き出し用準備
228
229       #メインループ
230       with mp_face_mesh.FaceMesh(
231           max_num_faces=1,
232           refine_landmarks=True,
233           min_detection_confidence=0.5,
234           min_tracking_confidence=0.5
235       ) as face_mesh:
236           while True:
237               ret, frame = cap.read()
238               if not ret:
239                   break
240               frame = cv.flip(frame, 1)
241               rgb_frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
242               img_h, img_w = frame.shape[:2]
243               results = face_mesh.process(rgb_frame)
```

省略

図 28　外部から呼ばれるためのメソッド

5.3 値取得用メソッドの実装

値取得用のメソッド「get_direction()」を作成する．これはメインのプログラムから判定結果を取得できるメソッドである．

get_direction()を利用できるようにするために「set_direction()」を用意する．これは視線判定メソッドで得た値を加工して get_direction()で返せるようにするものである．これは視線判定メソッドで使われる．

以上の二つを実装したものを以下の図で示す．

```
194   #値加工用
195   def set_direction(dir):
196       global direction
197       global direction2
198       dir_val = {5: "e", -5: "c", 10: "d", -10: "f"}
199       dir_val2 = {5: "a", -5: "cc", 10: "g", -10: "b"}
200
201       if dir in dir_val:
202           direction = dir_val[dir]
203           direction2 = dir_val2[dir]
204
205   #メインのプログラムへの受け渡し
206   def get_direction(mode):
207       modes = {0: direction, 1: direction2}
208       if mode in modes:
209           return modes[mode]
```

図 29　メインのプログラムに値を渡す部分

後述する kivy を用いた開発で視線判定メソッドをマルチスレッドで呼び出すと，値の受け渡しを直接行うことができず，別のスレッドで値取得用のメソッドを呼び出さなければならない．本研究では少しメインのプログラムの仕様に寄せているため汎用性は低くなっているが，この部分を調整することで他のプログラムにも対応できるようになる．

5.4 ループ終了用メソッドの実装

現段階のループ終了条件はキーボードの「q」キーを押したときに終了することになっている．このままではメインのプログラムが終了しても視線判定だけ続いている状態となってしまい．メインプログラム終了時にフリーズしてしまう．それを防ぐためにメインプログラム側から終了コマンドを実行する必要がある．

ループ終了メソッド「set_finish()」を作成する．作成したものを以下に示す．

```
211   #ループの終了用
212   def set_finish(comand):
213       global finish_comand
214       finish_comand = comand
```

図 30　終了用コマンドのソースコード

変数finish_comandはbool型であり初期値はFalseである．メインプログラムからTrueが渡されることで以下に示すループ処理中の条件文で判定されループが終了する．

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame = cv.flip(frame, 1)
    rgb_frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    img_h, img_w = frame.shape[:2]
    results = face_mesh.process(rgb_frame)
    if results.multi_face_landmarks:
                    省略

    if finish_comand:
        break
```

図 31　finish_comand の使われ方

第 6 章　アプリの作成

6.1 アプリの概要

本研究で作成されたアプリは視線の向く方向によって音程が変化する音楽演奏アプリである．アプリのみで操作の仕方などを知ることができるように操作説明画面をアプリ内に設けた．アプリの全体画面は図の通りである．

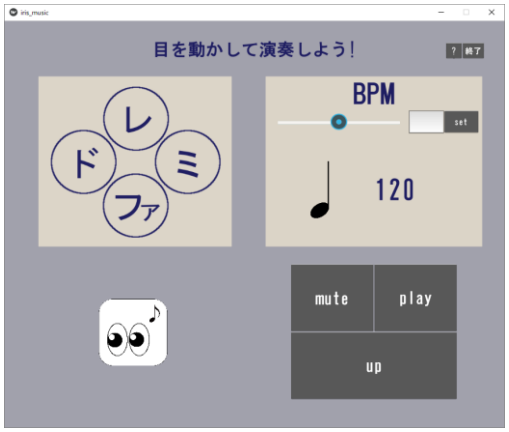


図 32　アプリのメイン画面

図 33　操作説明画面

6.2 アプリ仕様

・視線方向に合わせて音程が変わる

・画面右側が操作オブジェクトである

・画面右上の「BPM」欄でBPM(リズム)の変更が可能

各種ボタンの機能を以下の表に示す.

ボタンの名前	機能
mute	演奏の一時停止
play	演奏再開
up, down	音程の切替
set(BPM 欄)	BPM を確定する
?	操作説明
終了	アプリの終了

表 4　各種ボタンの機能

6.3 ライブラリ「kivy」について

6.3.1　Kivy とは

「kivy」は Python 上で GUI(グラフィカルユーザーインターフェース)を構成するためのオープンソースライブラリである. 対応 OS は Android, iOS, Linux, MacOS X, Windows 等の主要環境で動作させることができる.

6.3.2　選定理由

Python には GUI 開発ライブラリがいくつかある. 今回は研究題材のおまけアプリの作成ということもあり,設計が簡単で短時間で習得できるかつ, 比較的インターネットに情報が多く掲載されていることを理由に kivy を選定することにした.

6.3.3　Kivy の基本

(1)　ソースファイル

Kivy を扱う場合，メインのソースコード(プログラム名.py)一つでも実装できるが，多くの場合 kivy 言語を扱うファイル(kivy プログラム名.kv)があると非常に作業がわかりやすくなる．場合によってはファイル名に注意が必要な場合がある．

(2)　メインファイルの説明

Kivy でアプリケーションを作るときの基盤となる部分を以下の図に示す．

sample1.py

```
#ライブラリのインポート
from kivy.app import App
from kivy.config import Config
from kivy.uix.widget import Widget
from kivy.properties import ObjectProperty
from kivy.lang import Builder

#ウインドウの幅と高さの設定
Config.set('graphics', 'width', 700)
Config.set('graphics', 'height', 500)
#1でサイズ変更可、0はサイズ変更不可
Config.set('graphics', 'resizable', 1)
#kvファイルを指定
Builder.load_file("images.kv")

#各種機能を書くところ
class MyLayout(Widget):
    pass

#アプリの根幹
class DispimageApp(App):
    def build(self):
        self.title = "window"
        return MyLayout()

if __name__ == '__main__':
    DispimageApp().run()
```

図 34　アプリメインファイル

これをもとに構成されていく．ファイル名，各種数値, class 名などは変更可能だが全体の形はおおむねこのままでなければならない．

アプリの根幹である「DispimageApp(App)」(図 33: 21 行目)は「〇〇App(App)」の形にしなければならない．

また，〇〇.kv ファイルの読み込み方は「Builder.load_file(“ファイル名”)」(図 33: 14 行目)でできるが，.kv ファイルを〇〇App(App)の「〇〇」をすべて小文字にした〇〇.kv とすることで, 14 行目を記述しなくても読み込むことができる．ただしメインファイル(〇〇.py)と同じ階層である必要がある．

(3) .kv ファイルの説明

メインファイル(.py)に対応した記述をする必要がある..kv ファイルは HTML と CSS のような関係があり，主に画面の装飾に使われるファイルである．以下に.kv ファイルの例を示す．

dispimage.kv

```
<MyLayout>
    BoxLayout:
        orientation: "vertical"
        size: root.width, root.height
        padding: 20
        spacing: 20

        Image:
            source: "./image/nekko.png"
            allow_stretch: True
            kepp_ratio: True

        Button:
            text: "Delicious Curry!!"
            pos_hint: {"center_x": 0.5}
            size_hint: (None, None)
            width: 200
            height: 50
```

図 35 .kv ファイルの例

図 33 の Widget を継承したクラス(図 33: 17 行目)に対応してコードを記述していく.

.kv ファイルは<クラス名>の下へネスト構造を作りレイアウトを製作していく.

<クラス名>で表記された元のクラスは root という扱いになり各種プロパティでは, root の変数やメソッドを呼び出すことが可能となる.

オブジェクトのタグ名の頭文字は大文字となり, プロパティはすべて小文字で表記される.

注意点として動画を読み込むために「VideoPlayer」というタグを使用するのだが, これを記述するだけでは動画は再生されない. 追加で「ffpyplayer」をインストールする必要がある.

6.4 GUI 設計

6.4.1 レイアウト

まずはウィジェット(ボタンなどのオブジェクト)の配置を考える. 視線の方向と音程を表す画像を一番注目される部分に配置する必要がある. このように考えたときに左上が最適と判断した.

次に操作パネルの配置を考える. 本アプリは基本的にマウス操作が必要になるため, 右手で直感的に操作できるように右半分に操作パネルを配置した.

終了ボタンはほとんどの windows 系アプリが右上で統一されていることから同じように右上に配置することにした.

操作説明用画面の設置も行った. アプリ内で操作方法などすべて完結できればユーザ的にも扱いやすいアプリになると考えた. 終了ボタンの横に配置することで目につきやすいと考えた.

6.4.2　メイン画面のネスト構造

アプリのレイアウトとソースコードのウィジェット(タグなど)を対応付けた図を以下に示す.

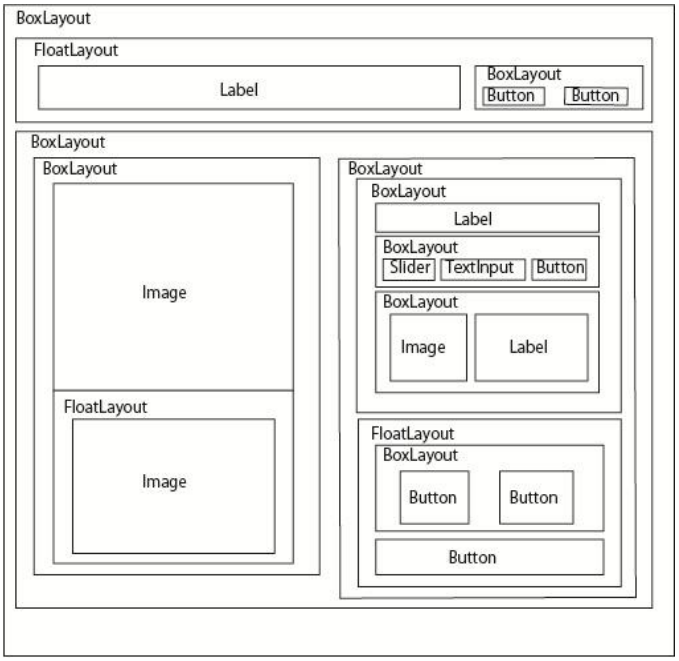


図 36　メイン画面のネスト構造

6.4.3　操作説明画面のネスト構造

操作説明画面のネスト構造を以下の図に示す.

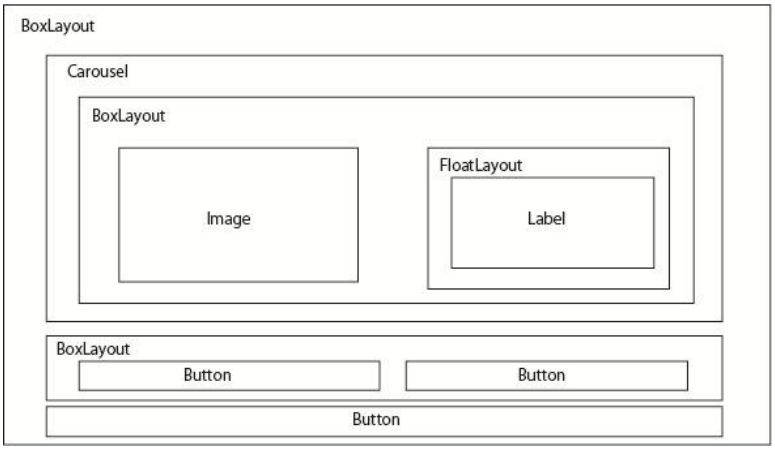


図 37　操作説明画面のネスト構造

6.4.4　すべてのタグに使えるプロパティの説明

（1） id

それぞれのウィジェットに id を付与することが可能

Id 指定で特定の動きをさせるために必要である.

（2） size

それぞれのウィジェットの領域を指定できる.

Size: x(数値), y(数値)

で使用できる.

（3） padding

css と同じ仕様で上下左右に空間を持たせることができる.

（4） color

ウィジェットの領域を塗りつぶす.

color: 数値/256, 数値/256, 数値/256 , 1(透明度 0～1)

の表記で実装

（5） size_hint

ウィジェットの大きさを同階層のネストの割合で指定可能.

size_hin_x でｘ軸方向の調整, size_hint_y でｙ軸方向の調整をする.

size(30, 30)のように固定値で設定したい場合は size_hint(None, None)と指定しなければならない.

（6） pos_hint

主に Floatlayout の子要素で使用することになる. 相対的な配置位置の調整が可能.

pos_hint: {“center_x”: 数値, “center_y”: 数値}が使いやすい.

6.4.5　BoxLayout の縦横切り換え

orientation プロパティで横に配置していくか，縦に配置していくかを切り替えることができる．デフォルト(記述しない状態)では横に配置していく．

縦に配置する場合は，「orientation: “vertical”」と記述する．

6.4.6　Button のクリック処理

基本的にはルートの関数を呼び出す．プロパティの記述は「on_press: root.メソッド()」の形式で行う．

6.4.7　GUI の完成

GUI(アプリ画面)の作成は以上で完了となる．全体のソースコードは付録として後述する．

この後に，視線トラッキングモジュールの呼び出しを行い各種ボタンに機能を割り振る．

ここまでで完成したファイルをそれぞれ「iris_music.py」と「iris_music2.kv」とする．

6.5 視線方向取得モジュールの呼び出し

メインのプログラム(iris_music.py)に kivy ファイル(iris_music2.kv)と視線方向取得モジュール(iris).py を読み込む.

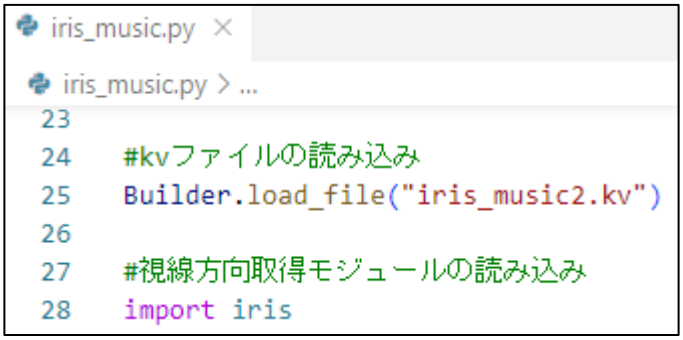


図 38　各種ファイルの読み込み

次に，インスタンスを生成し, ThreadPoolExecuter を利用しマルチスレッドで iris_cerector()を起動させる.

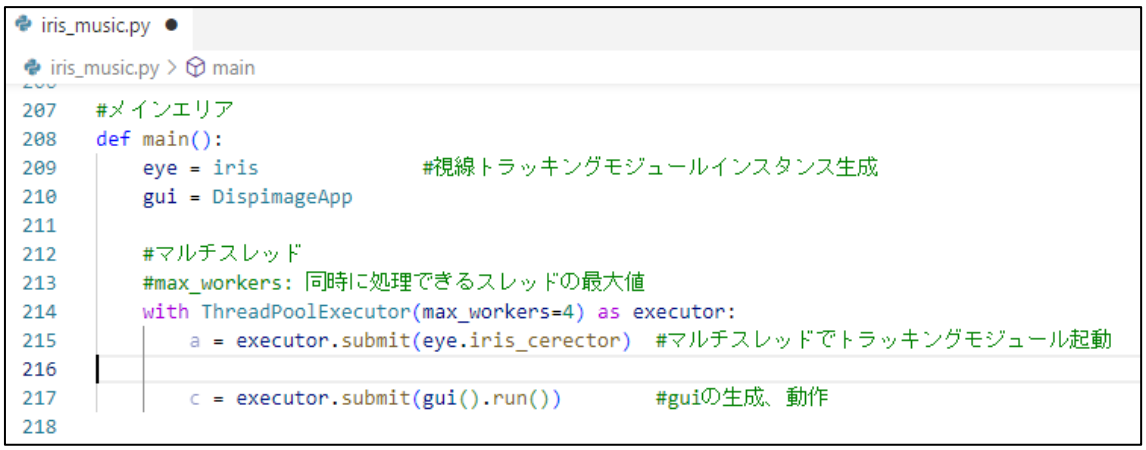


図 39　マルチスレッド処理

マルチスレッドで起動させる場合インスタンスを生成してから executer.submit()の引数に渡さないと起動することができない.

Max_worker : executer.submit()を実行できる最大値を決定する.

値取得用メソッドの「get_direction()」を別のスレッドで起動させる．今回は常に値を取り続ける必要があったため別メソッドでループ処理を行うように作成した．direction()に視線方向取得モジュールのインスタンスを渡すことでget_direction()が使えるようになる．

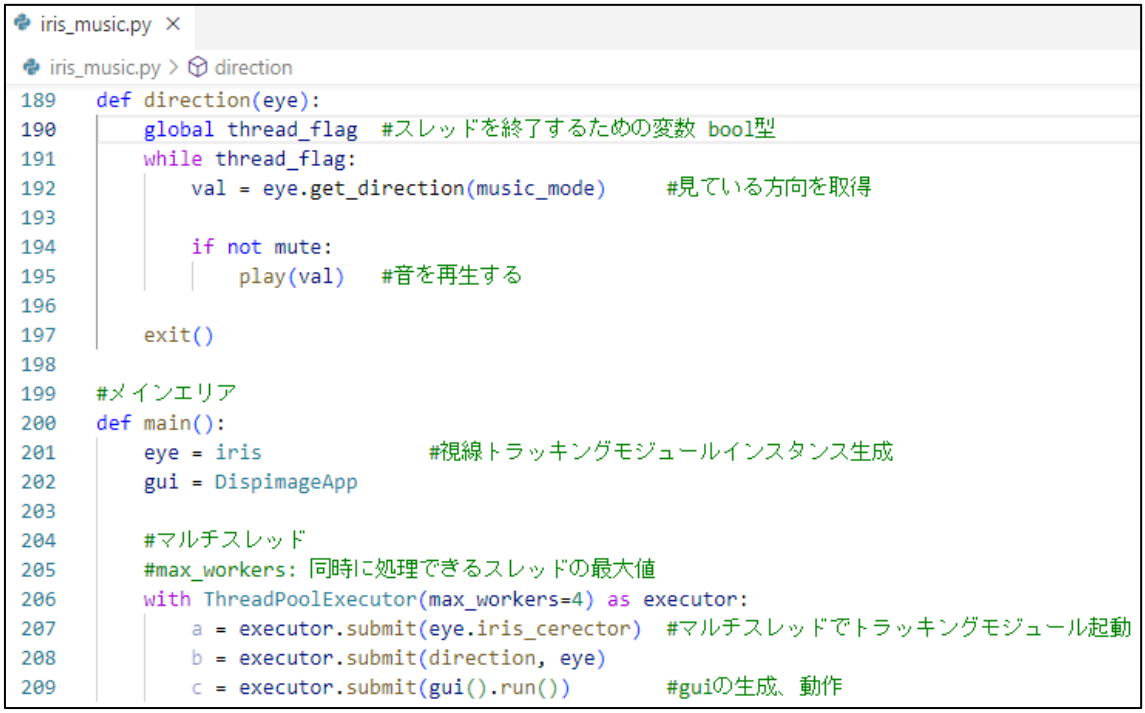


図 40　get_direction()の呼び出し

6.6 音を鳴らす仕組み

各音程の周波数をそれぞれ辞書型配列で定義する．さらに, BPM(テンポ)を設定する. winsound というライブラリの beep()に定義した周波数と BPM を引数として渡すことで, BPM に合わせた音を鳴らすことができる.

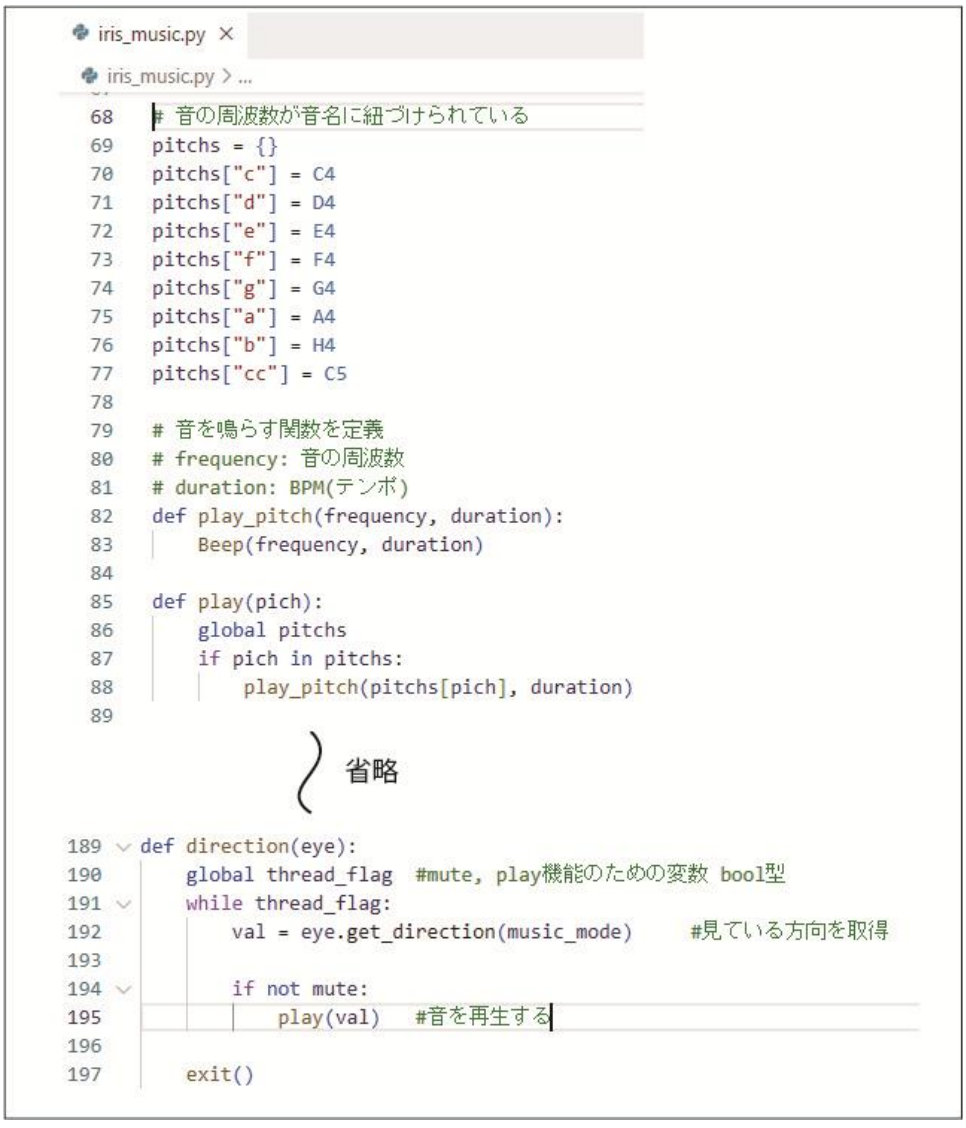


図 41　音を鳴らす仕組み

6.7 再生と停止の処理

direction()の処理内にある play()メソッドを呼び出している部分の if 条件(図 41: 194 行目～195 行目)の bool 型変数「mute」の値を更新することで音の再生，停止を制御できる仕組みをとっている．

条件を変更する処理は画面のボタンをクリックしたときに呼び出される．

ボタンを押したときの処理はクラス「MyLayout(Widget)」内に記述する．二つの処理を以下に示す．

```
#muteボタンを押したとき(停止処理)
def mute_button(self):
    global mute
    mute = True

#playボタンを押したとき(再生処理)
def go_playing(self):
    global mute
    mute = False
```

図 42　再生，停止処理のソースコード

6.8 BPM(テンポ)設定処理

基本的に変数「duration」の値を更新する．画面上の TextInput の値またはスライダーの値を取得して duration の値を更新する．

slider との値を連携する．Slider の使い方は参考文献【7】silider の使い方を参照．

さらに textInput からの入力にも対応するため TextInput の値を「self.ids.(.kv で指定した ID 名).text」で取得する．ここで取得できる値は文字列なので整数型に変換する必要がある．

その後，BPM を以下の式で計算し duration に代入する．今回は最小値を 40，最大値を 200 に設定した．

$$\frac{60\text{ 秒}}{\text{指定したい BPM の数値}}$$

最後に TextInput 欄に残った文字列を空白リセットする．

これらの操作は「set」ボタンが押されたときに実行される．

以下に BPM 設定のソースコードを示す．

```
#setボタンを押したとき(BPMの更新)
def BPM_set(self):
    global duration     #BPMの値
    BPM = self.ids.BPM_input.text
    duration = int(60000 / int(temp_duration))
    if BPM.isdecimal():
        if int(BPM) >= 40 and int(BPM) <= 200:
            duration = int(60000 / int(BPM))

            self.ids.BPM_label.text = BPM   #画面表示を更新
            self.ids.BPM_slider.value = BPM #スライダーの値をTextInputの値と同じにする
            self.ids.BPM_input.text = ""    #TextInputに残った値を空白リセット
        self.ids.BPM_input.text = ""        #数値が範囲外だった時に空白リセットさせる

#スライダーの値取得など
def BPM_slider(self, *args):
    global temp_duration #スライダーの値を格納
    temp_duration = int(args[1])
    self.ids.BPM_label.text = str(int(args[1]))
```

図 43　BPM 設定のソースコード

6.9 音程モード切り替え処理

視線方向取得モジュール(iris.py)の get_direction()に引数としてモード切り替え用の値を渡すことで音程モードの切り替えが可能.

iris_music.py の変数「music_mode」には 0 または 1 が代入される. 0 が代入されたときは get_direction が「ド，レ，ミ，ファ」の音程を鳴らすための文字列を返す, 1 が代入されると「ソ，ラ，シ，ド」の音程を鳴らすための文字列を返す. この部分の iris.py のソースコードは図 29 である.

この処理はモード切り替えボタン(up, down)ボタンがクリックされたときに行われる. 一つのボタンで処理を切り替えるため，表示の切り替えと機能の切り替えをする必要がある. 変数「mode_text」にボタンに表示されている文字列を読み込み，その文字列が「up」「down」のどちらが表示されているかで判定する. この処理も同様にクラス MyLayout(Widget)内に記述する. この部分のソースコードを以下に示す.

```
#music_modeの切り替え(「ソ，ラ，シ，ド」側にする)
def mode_high(self):
    global music_mode
    music_mode = 1

#musi_modeの切り替え(「ド，レ，ミ，ファ」側にする)
def mode_low(self):
    global music_mode
    music_mode = 0

#ボタンの機能とモードの切り替え
def mode_ch(self):
    mode_text = self.ids.mode_change.text
    if mode_text == "up":
        self.ids.my_image.source = "./image/new_gabc.png"
        self.ids.mode_change.text = "down"
        self.mode_high()
    else:
        self.ids.my_image.source = "./image/new_cdef.png"
        self.ids.mode_change.text = "up"
        self.mode_low()
```

図 44　モード切替のソースコード

6.10 説明画面の表示方法

説明画面はpopup要素で出現させる．MyLayout(Widget)のボタンを押した処理で別に定義したクラス「PopUplayout(BoxLayout)」を機能させる popup 処理を行う．詳細の解説は，参考文献【8】https://senablog.com/python-kivy-popup/を参照する．

ソースコードは以下の通り．

```
class PopUpLayout(BoxLayout):
    popup_close = ObjectProperty(None)

    def img_back(self):
        self.ids.carousel.load_previous()

    def img_next(self):
        self.ids.carousel.load_next()

class MyLayout(Widget):
    def popup_open(self):
        content = PopUpLayout(popup_close=self.popup_close)
        self.popup = Popup(title='操作説明', content=content,
                           size_hint=(0.7, 0.7), auto_dismiss=False)
        self.popup.open()

    def popup_close(self):
        self.popup.dismiss()
```

図 45　説明画面を表示するためのソースコード

第7章　終わりに

本研究は，コミュニケーション手段の一つとして「眼の動き」に注目し，眼の動きをトラッキングして操作できるツールに関する今後の研究の基盤になれたらよいと思う．

視線を取得する方法として角膜反射法(参考文献【9】)がある．しかし，この方法では眼に光を当てなければならず目への負担がどうしても避けられない．本研究では，外部からの人体への影響を極力減らして実行することが可能となった．

本研究を通して，眼の動きをトラッキングする専用のハードウェアが無くても眼の動きを検知することが可能であると確認できた．さらに，小さい子供から大人まで「眼の動き(上下左右)」を使って音楽を演奏することができるアプリを完成させた．

今後の課題として

左右判定の精度は十分に作成することができた．しかし，上下の判定はまだ改善の余地がある．特に上の判定は修正が必要である．

再初期段階で考えていた案は視線でPCのマウスのように画面上を操作できるようなツールだった．本研究では，市販のwebカメラから画像認識により目の動きを取得するため，目の細かい動きを検知するのが困難だった．そこで，妥協案として方向を取得するだけなら誤差と区別できる明らかなデータが取得できると考え実装した．

今後この研究がもし，引き継がれることがあるならば，上記に述べたように視線でPCのマウスのように画面上を操作できるつまり，「目の細かい動きを検知できる」機能を実装してほしい．

参考文献

【1】Face Mesh の使い方

https://blog.deepblue-ts.co.jp/image-processing/how_to_use_face-mesh/

【2】2 点間の距離

https://www.higashisalary.com/entry/numpy-linalg-norm

【3】2 点を通る直線から離れた位置にある点までの距離を求める

https://tokibito.hatenablog.com/entry/20121227/1356581559

【4】指数移動平均法

https://bigdata-tools.com/moving-average/

【5】kivy の基本

https://torimakujoukyou.com/category/python/kivy/

【6】kivy のレイアウト関連

https://senablog.com/category/programming/python/kivy/

【7】slider の使い方

https://torimakujoukyou.com/python-kivy-slider/

【8】kivy のポップアップ

https://senablog.com/python-kivy-popup/

【9】角膜反射法

https://www.jstage.jst.go.jp/article/vision/3/2/3_81/_pdf