情報理工学特別研究報告書

題目

手のジェスチャー認識を用いた
spotify 操作システムの開発

学生証番号　153731

氏名　清次 駿斗

提出日　令和 6 年 1 月 31 日

指導教員　蚊野　浩

京都産業大学
情報理工学部

要約

spotify や apple music をはじめとする音楽サブスクリプションサービスの会員数は増加傾向にある. このことから，音楽再生サブスクリプションサービスの需要が高まっていることがわかる. 特に世界市場シェアの約 33.27%(2022 年)を占めており，最も利用者の多い spotify の操作に着目し，研究を行った. 現在の音楽再生のアプリの操作方法は主にタッチ操作，音声操作であり，特にタッチ操作が主流である. また，ジェスチャーによる操作は普及していない. しかし，実際にタッチ操作ができない状況や，音声操作は音が発せられている状況では、音声認識の精度が落ちるといった問題点がある. これらの状況時にジェスチャーによる操作は有効であると考え，ジェスチャーによる音楽再生アプリの操作を可能とするシステムを開発した.

手の検出では mediapipe を使用し，手の 21 個のランドマークを検出する. また mediapipe は手の検出だけでなく，検出したランドマークの座標も求めることができる. この座標を用いて，手のランドマーク間の距離や，特定のランドマークの座標の移動距離を求め，手続き的処理を行うことで，手の形状やジェスチャーの動きを認識する. spotify の操作は spotify api を用いることで，外部のアプリーションから spotify のデータへのアクセスや，再生操作等の一般的操作を行うことが可能である. 本実験ではこの spotify api を使用し，python で spotify の一般的操作を行う.

本研究の実験では，spotify の一般的な操作をジェスチャーによって実現するシステムの有効性を検証した. 被験者 6 名を対象に，音量調整，楽曲の再生・一時停止，ミュート・ミュート解除，楽曲のスキップ・戻し，10 秒飛ばし・10 秒戻し，再生位置の調整の 6 種類の操作を行い，正常動作率と誤認識数を測定した. 実験の結果，ジェスチャーの種類によって正常動作率および誤認識数に差が見られた. ジェスチャー同士の形状の類似性や、認識アルゴリズムの閾値設定が影響していると考えられる. また，動きを伴うジェスチャーでは，被験者の手の動かし方に個人差があり，システムへの適合度にばらつきが見られた.

これらの結果から，ジェスチャー間の境界をより明確にする必要や，システムの認識条件をより柔軟にする，またはジェスチャーのデザインを個人差に対して寛容なものにするなど，改良の余地がある.

目次

1章　序論

spotify や apple music をはじめとする音楽サブスクリプションサービスの会員数は増加傾向にある.日本国内に着目すると、2021 年末時点で，音楽配信サービスの利用者は約 2,590 万人，2025 年末には 3,250 万人に達する見込みである[1].これらの情報から，音楽再生サブスクリプションサービスの需要が高まっていることがわかる.

現在の音楽再生アプリの操作方法は主にタッチ操作，音声操作であり，特にタッチ操作が主流である.一方で，他の分野の例として，VR や AR デバイスにおける直感的な操作など，ジェスチャー認識技術は近年さまざまな分野で応用が進んでいる.しかし，音楽再生アプリにおける操作方法としてはジェスチャー認識による操作は普及していない.

ここで，ジェスチャー認識による操作が普及していない理由は，照明条件や背景の複雑さをはじめとする環境の変化に対する，精度の問題があると考えられる.実際に，現在普及している音声認識も誤認識が珍しくはなく，ユーザが慣れているタッチ操作が主流となっている.

しかし，デバイスまでの距離が遠い場合には，タッチ操作ができない.そのような時に利用される音声による遠隔操作は，音楽再生アプリによる音の影響で認識精度が落ちる可能性や，音楽の再生が中止される(siri を用いて操作した場合)といった問題があった.このような状況では，ジェスチャーによる操作が有効であると考えられる.

そのため、本研究ではジェスチャーによる音楽再生アプリの操作を行うことができるシステムを Python で開発した.特に，音楽サブスクリプションサービスの中で，世界市場シェアの約33.27%(2022年)[2]を占めており，最も利用者の多いspotifyの操作に着目し，研究を行った.

2章　mediapipeとspotify api

2.1 mediapipeについて

Mediapipeは Google が開発したクロスプラットフォームの機械学習フレームワークであり，画像や動画の処理が可能なツールである．Mediapipe はオープンソースソフトウェアとして公開されており，そのソースコードは無償で利用可能である．これにより，誰でも自由にソフトウェアを使用・複製・改良，さらには再配布することができる．このオープン性は，幅広いユーザーや開発者にとって大きな利点であり，さまざまな分野での応用が可能となっている．

機能面の特徴として手のジェスチャー認識や顔メッシュ，姿勢推定などの機能がある．本研究では，これらの機能の中で，手の検出の機能にに着目する．

次に，mediapipe の手の検出における機能等について説明していく．Mediapipe は手のひらや指に対応する 21 個のランドマークを検出することができる．Mediapipe が検出する手のランドマークを図 2.1 に示す．図 2.1 のように指の関節や手の付け根など，手の構造に沿ったポイントをランドマークとして検出する．例えば，mediapipe で検出されるランドマークの 1 つである 8．INDEX_FINGERTIP は人差し指の先端を表しており、ポイント 8 として管理することができる．

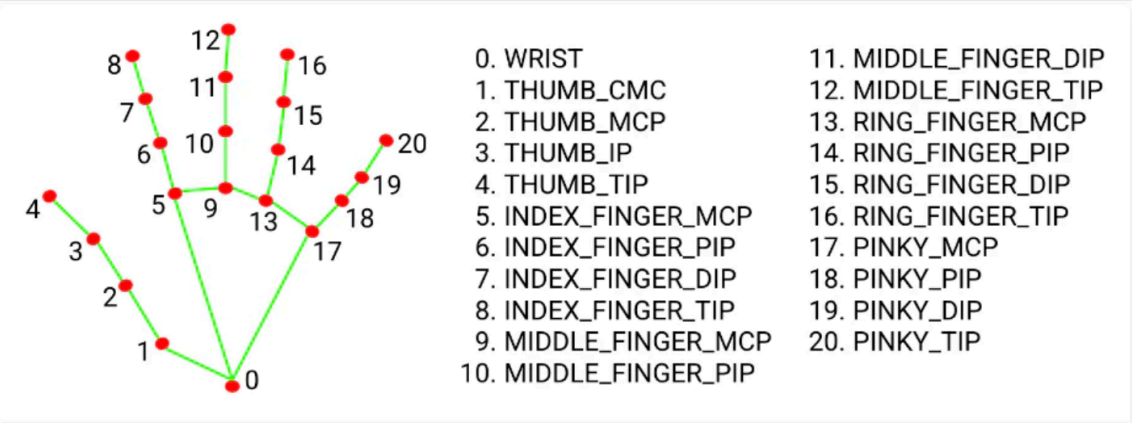

図 2.1 mediapipe における手のランドマークポイント

また，mediapipe では，21 個のランドマークポイントを検出するだけでなく，これらのランドマークポイントの座標を求めることも可能である．図 2.2 のように，カメラ画像の左上座標（x,y）=（0,0）、右下座標（x,y）=（1,1）とし，で手のひらや指の 21 個のランドマークの座標を検出することができる．これらの座標は，画像内の特定の位置を示しており，各ランドマークには x,y,z の 3 次元座標が割り当てられている．z 座標

は手の付け根（ポイント 0）を基準とした相対的な深度情報である. x, y 座標は 0 から 1 の範囲で正規化されているが，z 座標は正規化されていない. 今回の研究では手の形やジェスチャーを認識する際に, 手のひらがカメラに向かっていることを前提としているため，深度情報を示す z 座標は使用せず，画像内での位置情報を示す x, y 座標のみ利用する. これらの情報の活用方法として，mediaPipe を使用して特定のランドマークの座標を取得する場合，python コード上では hand_landmarks.landmark[8].x のように記述することで，人差し指の先端（ポイント 8）の x 座標を簡単に取得できる. このようにして取得した座標データをもとに、手の形状や動きのパターンを解析し，特定のジェスチャーを認識することが可能となる. これらの機能は，本研究においてジェスチャーを活用した操作システムの基盤を支える重要な要素となっている.

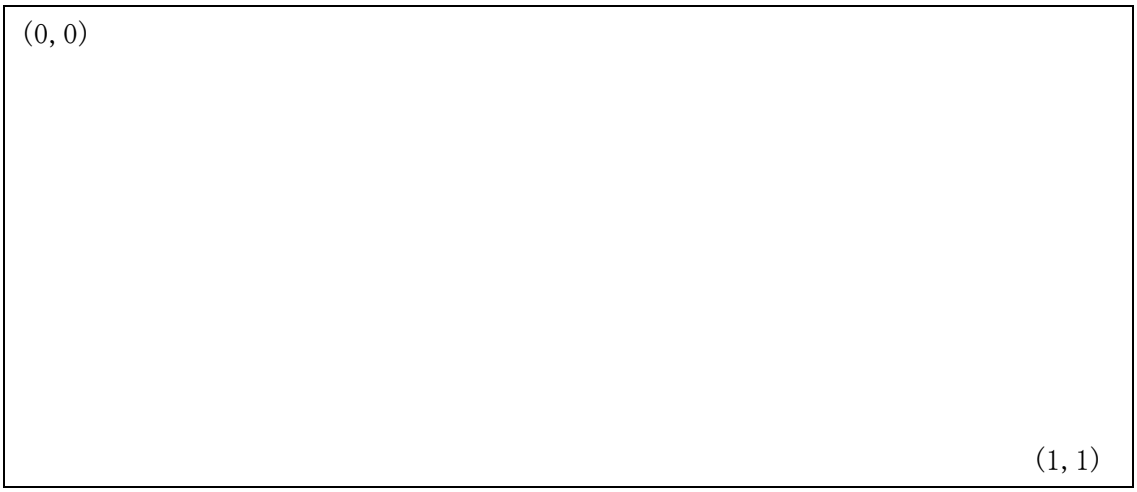

図 2.2 ランドマークポイントの座標値の範囲

図 2.3 に mediapipe で手を検出した様子を示す.図 2.1 で示したように,21 個のランドマークを認識していることが確認できる.

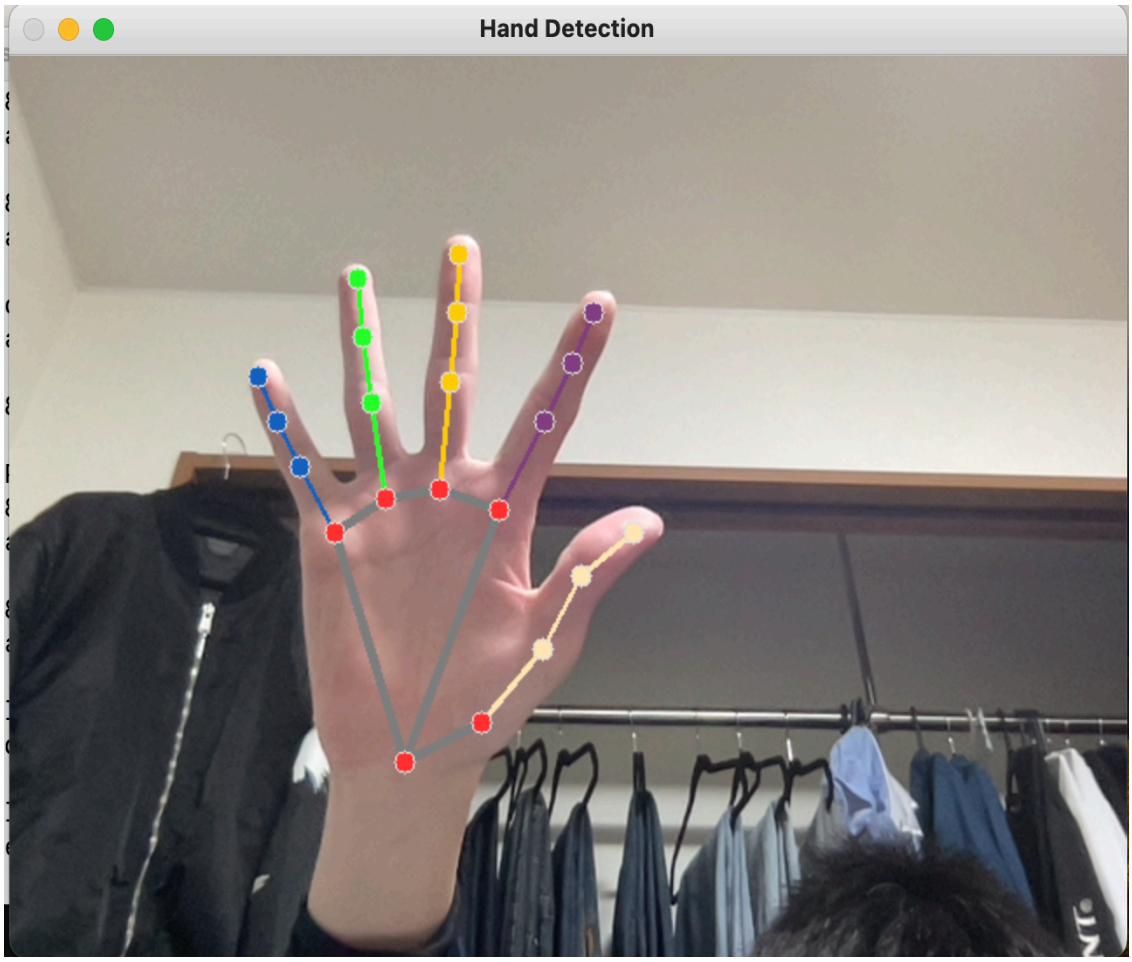

図 2.3 mediapipe 手を検出した際の様子

図 2.4 に mediapipe から取得した図 2.3 の画像に対する手のランドマークの座標を示す. x, y は 0〜1 の値で表示されている. z 座標はポイント 0 が 0 となっており，他の座標はポイント 0 を基準とした相対的な座標を示していることが確認できる.

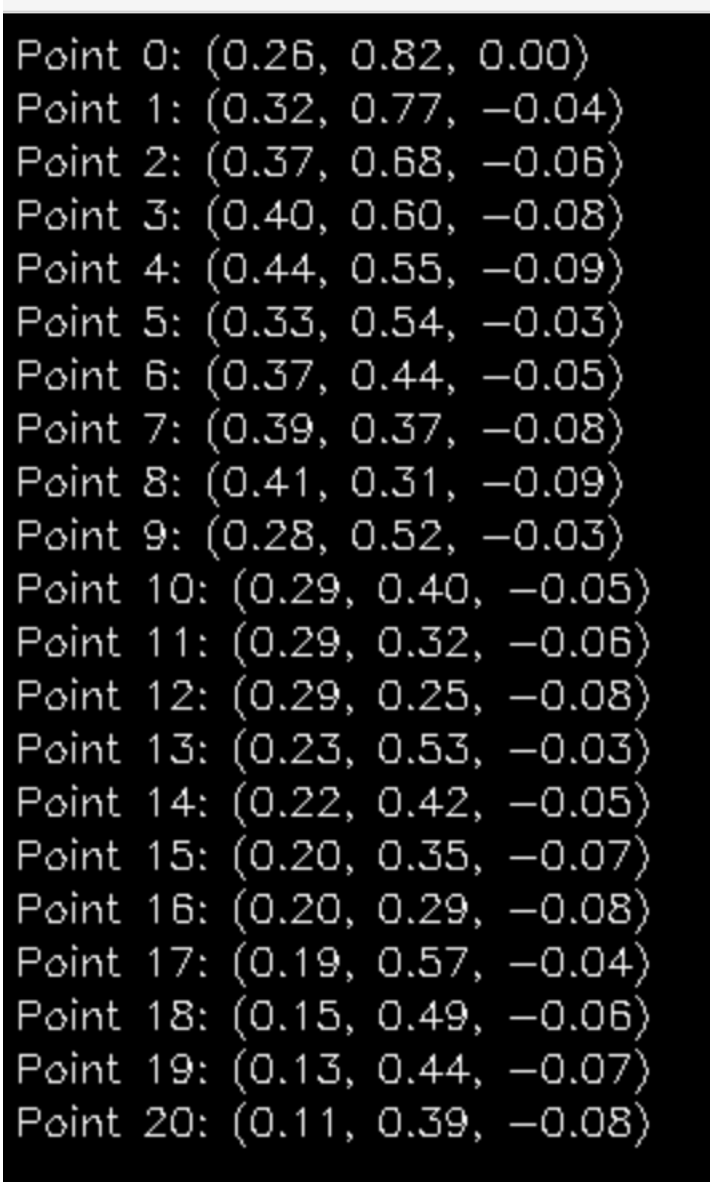

図 2.4 mediapipe から取得した手のランドマークの座標（図 2.3 と対応）

図 2.5 に mediapipe を用いて手の検出を行った際の処理速度を示す. この図から処理速度は約 20fps であることがわかる. ここで, fps とは 1 秒間に何回手の検出を行う処理が行われているかを表す指標であり, システムのリアルタイム性を評価する際に重要な要素である. 実際に iphone14Pro のカメラは最大 60fps でビデオを撮影することが可能であるが, 動画の容量が大きくなるため, 設定を変更しない場合は 30fps での撮影が推奨されている. 比較すると 20fps は少し低いが, 手の検出やジェスチャー認識の用途においては十分スムーズな動作が期待できる.

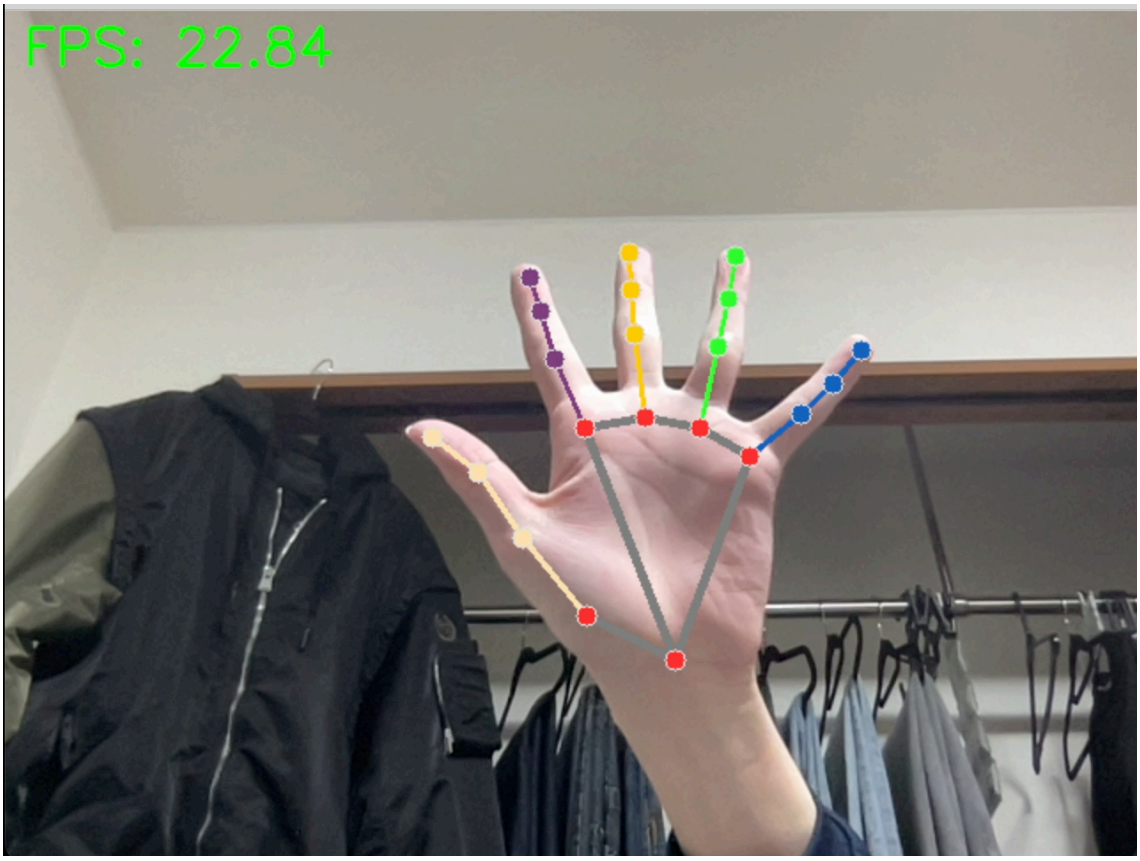

図 2.5 mediapipe によって手を検出した際の処理速度

2.2 spotify API について

API とは Application Programming Interface の略で，ソフトウェア同士が情報のやり取り・機能を利用し合うためのインターフェースのことである.つまり，spotify api とは spotify の音楽ストリーミングサービスと連携するための API のことである.これを利用することで，外部のアプリーションから spotify のデータへのアクセスや，再生操作等の一般的操作を行うことができる.本実験ではこの spotify api を使用し，python で spotify の一般的操作を行う.

3 章　ジェスチャー認識と spotify api の認証・利用

3.1　認識する手のジェスチャー

本研究では mediapipe から得られる 21 個のランドマークの座標を用いてジェスチャーの認識を行う.今回の研究で行う spotify の操作は，音量調整，音楽の再生・一時停止，ミュート・ミュート解除，楽曲のスキップ・楽曲を戻す，10 秒飛ばし・10 秒戻し，再生位置の調整の 6 つの操作である.これらは spotify のタッチ操作でも可能な操作である.この 6 つの操作を行うため，これに対応した 6 種類のジェスチャーの認識を行う.ジェスチャーと spotify の一般的操作の対応関係を表 3.1 に示す.

表 3.1　ジェスチャー内容と spotify 操作の対応関係

	ジェスチャー	Spotify操作
①	人差し指と親指を摘んだ状態で上下させる	音量調整
②	手を広げた状態を1秒間維持	再生・一時停止
③	手を握った状態を0.5秒間維持	ミュート・ミュート解除
④	人差し指を素早く横にスワイプ	楽曲スキップ・楽曲を戻す
⑤	人差し指と中指を揃えた状態で素早く横スワイプ	10秒飛ばし・10秒戻し
⑥	音楽再生バーを人差し指で調整	再生位置の調整

3.2　手のジェスチャー認識のシステム

本研究ではジェスチャーを認識する方法として，「機械学習による処理」と，「手続き的処理」の 2 つの方法を試行し，認識精度の高い手法を採用することとした．認識精度に加えて，実装の迅速性や開発リソースの最適化といった実用性の観点から，効率的かつスケーラブルな手法を選定した.

一つ目の機械学習による処理とは，大量のデータからパターンを学習し、未知のデータに対して予測や分類を行う方法である.データの収集に mediapipe から取得される 21 個のランドマークの座標を活用する.この座標データには，手の形状や，動きを表すポイント間の距離や座標の移動距離が含まれている.この情報を学習データとして使用し，ジェスチャーを認識するモデルを構築した.機械学習の実装には pytorch を用いた.

まず,ジェスチャーを行った際の手の形状の座標データを収集し，これをトレーニングデータとして使用した.モデルは手の形状や動きのパターンを学習し，これに基づいてジェスチャーを認識する.実験の結果，手が静止している再生・一時停止，ミュート・ミュート解除のジェスチャーを認識することはできた.しかし，　手の動きが伴うジェスチャーは認識させることができなかった.

二つ目の手続き的処理とは，明確なルールや手順に基づいて処理を実行する方法である.Mediapipe から得られる 21 個のランドマークの座標データを活用する.具体的には，各ランドマークのポイント間の座標の距離や，特定のポイントの座標の移動距離を計算し，これらの情報を基にジェスチャー認識を行う.まず手の形状や動きを捉えるために，mediapipe からジェスチャー認識を行う際に必要なポイントの座標を取得する.次に，これらの座標を基に，ジェスチャー認識を行うための計算を行う.例えば，手の形状を求める際には，親指と人差し指の間の距離や，手首から指先までの距離などが含まれる.また、指先のランドマークの座標を 0.3 秒毎に比較し，移動距離を計算することで、手の動きのパターンを捉えることができる.この方法では，一つ目の方法とは異なり，手の動きが伴うジェスチャーも認識させることができた.

手続き的処理は明確なルールに基づくシンプルで高速な実行が可能であり、実装の容易さと処理の即時性を重視して採用することとした。

3.3 ジェスチャー認識の処理内容

ここでは 6 種類のジェスチャー認識の際の手続き的処理の内容を説明する.

3.3.1 音量調整 / 人差し指と親指を摘んだ状態で上下

音量調整をする際のジェスチャーとして，人差し指と親指を摘んだ状態で上下に動かす方法を採用した. 上に動かすと音量が増加，したに動かすと音量が減少する.このジェスチャーを認識するために,親指の先（ポイント 4）と人差し指の先（ポイント 8）の座標を取得し，距離を計算する.その値が 0.03 以下である場合，親指と人差し指を摘んでいる状態と認識する.その状態が検出されると，ポイント 8 の y 座標の移動距離を求める.その距離が 0.03 以上減少していれば，音量を増加させる.逆に 0.03 以上増加していれば，音量を減少させる.実際のジェスチャーの様子を図 3.1 に示す.

この認識手順の問題点として，音量を調整した後に，上下させた手を元の位置に戻す動きが，再びジェスチャーとして認識される場合があった.例えば音量を増加させた後に，手を下に戻す際に，音量を減少させる操作が誤って実行されてしまうといった，ユーザの意図に反した逆の操作が実行されてしまう場合がある.そのため，音量を増加させた直後は一秒間音量を下げる操作を無効化し，逆に，音量を減少させた直後は一秒間音量を上げる操作を無効化するようシステムに制御を加えた.これによりユーザの意図した操作だけが実行できるようになった.

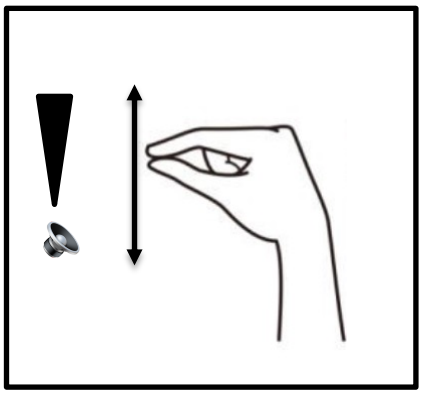

図 3.1 音量調整を行う際のジェスチャー

3.3.2 音楽の再生・一時停止 / 手を広げた状態を一秒間維持

音楽の再生･一時停止をする際のジェスチャーとして，手を広げた状態を一秒間維持する方法を採用した．このジェスチャーは，mediapipe を使用し，手の付け根（ポイント 0）と小指の先（ポイント 20）の距離を計算し，その値が 0.2 以上の場合に「手を広げた状態」と認識する.その状態が 1 秒間キープされた場合に，spotify から現在の再生状態を取得し，再生中であれば一時停止，停止中であれば音楽を再生する.実際のジェスチャーの様子を図 3.2 に示す.

しかし，この方法には問題点があった.1 秒間のジェスチャー認識で spotify の操作が行われるため，短時間で操作がスムーズに行われる一方，音楽を再生させる場合に再生直後に再び同じジェスチャーが認識され，音楽が停止されてしまうことや，その逆の現象が発生することがあった.この問題の解決策として，ジェスチャーを認識し，spotify が動作した際に，その後 3 秒間はジェスチャーを認識しないようにシステムに制御を加えた.この追加システムにより，誤認識による再生・一時停止の頻繁な切り替えを防ぎ，より安定した操作が可能となった.

図 3.2 再生・一時停止の際のジェスチャー

3.3.3 ミュート・ミュート解除 / 拳を握った状態を一秒間維持

音量のミュート･ミュート解除をする際のジェスチャーとして，手を握った状態を一秒間維持する方法を採用した.手の付け根の（ポイント 0）と手の 5 本の指の先（ポイント 4，8，12，16，20）の距離をそれぞれ求め，その値の全てが 0.2 以下の場合に「手を握った状態」と解釈し，ミュート中であればミュート解除，ミュートさせていなければミュートする.実際のジェスチャーの様子を図 3.3 に示す.

この操作も 3.3.2 で挙げた問題と同様に，ミュート･ミュート解除が頻発する問題が発生した.そのため,ジェスチャーを認識した際に，その後 3 秒間はジェスチャーを認識させないという制御をシステムに加えた.

図 3.3 ミュート・ミュート解除の際のジェスチャー

3.3.4 曲を 1 曲スキップ・1 曲戻す / 人差し指を素早くスワイプ

曲を 1 曲スキップ・1 曲戻すの操作を行うジェスチャーとして，人差し指を素早くスワイプする方法を採用した.左から右にスワイプした際には 1 曲進める，右から左にスワイプした際には 1 曲戻す.このジェスチャーを認識するために mediapipe から人差し

指の先（ポイント 8）の x 座標を取得し，0.2 秒毎に比較することで座標の変化を調べる.ポイント 8 の x 座標が 0.25 以上増加した場合，曲を 1 曲スキップする.また，x 座標が 0.25 以上減少していた場合は，曲を 1 曲戻す.実際のジェスチャーの様子を図 3.4 に示す.

しかし，問題点としてこの操作はスワイプした後に手を戻す際，ジェスチャーが認識され，本来行いたい操作とは逆の操作が実行されることあった.この問題を解決するため，スキップの操作を行った際は戻す操作を，戻す操作を行った際はスキップの操作を，1 秒間は行うことができないようシステムに制御を加えた.

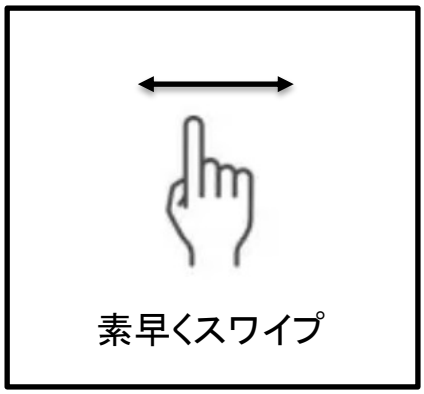

図 3.4 曲のスキップ・戻すの際のジェスチャー

3.3.5 曲を 10 秒進める･10 秒戻す / 人差し指と中指を揃えた状態で素早くス. ワイプ

曲を 10 秒進める･10 秒戻すの操作を行うジェスチャーとして人差し指と中指を揃え，素早くスワイプする方法を採用した.この操作は mediapipe から人差し指の先（ポイント 8）と中指の先（ポイント 12）の座標を取得し，この 2 つのポイント間の距離を計算する.その距離が 0.1 以下の場合， 人差し指と中指が揃っていると判断し， x 座標を 0.2 秒毎に比較することで座標の変化を調べる.ポイント 12 の x 座標が 0.25 以上増加した場合，曲を 10 秒進める.また，x 座標が 0.25 以上減少していた場合は，曲 10 秒戻す.このジェスチャーの様子を図 3.5 に示す.

しかし，問題点として 3.3.4 と同様に,この操作はスワイプした後に手を戻す際，ジェスチャーが認識され，本来行いたい操作とは逆の操作が実行されることがあった.この問題を解決するため，10 秒進める操作を行った際は 10 秒戻す操作を，10 秒戻す操作を行った際は 10 秒進める操作を，1 秒間は行うことができないようシステムに制御を加えた.

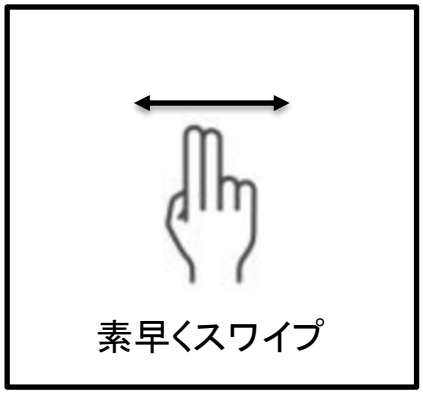

図 3.5 10 秒飛ばし・戻しの際のジェスチャー

3.3.6 再生位置の調整/音楽再生バーを人差し指に調整

spotify api から曲の再生時間の情報を取得し，それに基づいて，カメラ画像に表示するウィンドウ上に音楽再生バーを表示しする.音楽の再生位置バーはカメラのウィンドウの下部に設置する.次に，mediapipe からポイント 8 の座標を取得する.ポイント 8 の y 座標が再生バー上にある場合，ポイント 8 の x 座標に応じて，音楽の再生位置を更新する.実際のジェスチャーの様子を図 3.6 に示す.

しかし，リアルタイムで spotify api から曲の再生状況を取得し続けるため，処理の負担が大きく，手の認識の速度が遅くなるといった問題が発生した.そのため，10 秒毎に spotify api から音楽の再生状況の情報を取得し，更新することにした.その間，音楽再生バーは一定速度で進むと仮定し，音楽再生バーの描画を続ける.しかし， 音楽の再生・一時停止，曲のスキップや 10 秒進めるなどの曲の再生時間に変化する操作が行われた場合は音楽の再生状況に変化が生じるため，これらの操作が行われた際は spotify api からから音楽の最新の再生状況を取得し，音楽再生バーを更新する.

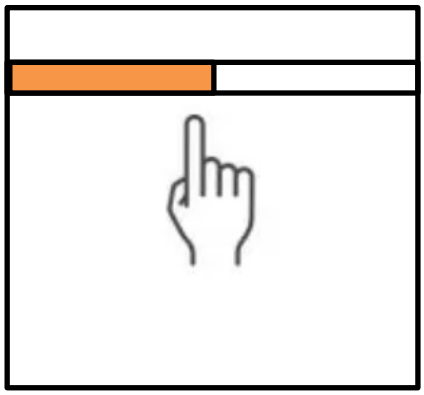

図 3.6 再生位置の調整を行うジェスチャー

3.4 spotify api 認証方法

spotify api を利用するには，spotify アカウントを使って API の認証を行う必要がある.まずSpotify Developer Dashboard にアプリケーションを登録する必要がある.そのために，（https://developer.spotify.com/dashboard） にアクセスする.次に，spotify アカウントでログインする.ログイン後，表示されるダッシュボード画面の中の「Create an App」をクリックし，アプリケーションを作成する.アプリケーション作成の際に App Name（任意の名前），App Description（アプリについての任意の説明）を入力する.作成後，アプリの Client ID と Client Secret が表示される.これらの情報は認証の際に必要となるので，保存しておく.その後，「Edit Settings」から Redirect URLs を登録する.

次に上記で得た情報を python のコードに組み込むことで，アプリケーションから spotify の認証ページにリダイレクトする URL を生成する.これにより，spotify api を利用するための認証が完了し，アプリケーションで spotify のデータや機能を操作できるようになる.spotify api の認証の全体の流れを図 3.7 に示す.

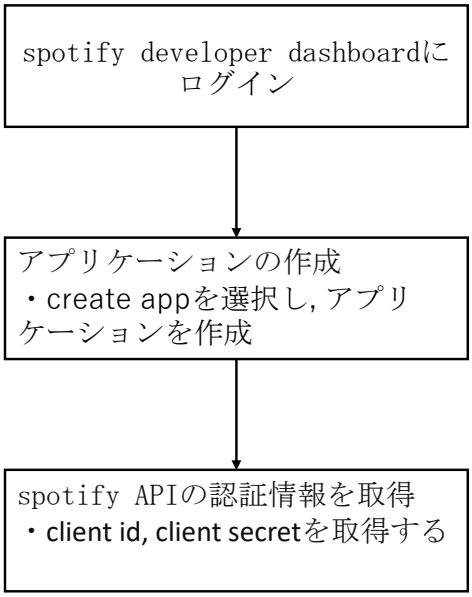

図 3.7 spotify api 認証手順

Python で spotify api の認証を行う際のコードを図 3.8 に示す.

```
sp = spotipy.Spotify(auth_manager=SpotifyOAuth(
    client_id='クライアントID ',
    client_secret='クライアントシークレット',
    redirect_uri='http://localhost:8888/callback',  # リダイレクトURI
    scope='user-read-playback-state user-modify-playback-state'  # 必要なスコープ
))
```

図 3.8 python で spotify api の認証を行う際のコード

spotify api を python で利用するには認証を行う必要がある.認証には OAuth2.0 を使用し，適切なスコープを指定することで，spotify のプレイヤーの状態の取得や，再生を制御することができる.

各パラメータの説明

・client_id：Spotify 開発者アカウントで発行されるクライアント ID を指定する。

・client_secret：クライアント ID とセットで発行されるクライアントシークレットキー を指定する。

・redirect_uri：Spotify の認証後にリダイレクトされる URI を指定する.ローカル環境で開発する場合は，http://localhost:8888/callback などを設定するのが一般的である.

・scope：Spotify API で実行したい操作に応じて スコープ（権限）を指定する.今回は，user-read-playback-state（現在の再生状態を取得する権限），user-modify-playback-state（再生を変更する権限）の 2 つを指定している.

3.5 spotify api による一般的操作

本研究では、Spotify API を活用し、Python コードを用いて Spotify の操作を実現した.以下の表に spotify api を用いた操作と python コードとの関係を示す.

操作内容	Python コード

音量調整	# 音量を調整する関数
	def adjust_volume(action):
	　　current_volume = int(subprocess.check_output(["osascript", "-e", "output volume of (get volume settings)"]))
	　　if action == "up":
	　　　　subprocess.call(["osascript", "-e", "set volume output volume (output volume of (get volume settings) + 25)"])
	　　elif action == "down" :
	　　　　subprocess.call(["osascript", "-e", "set volume output volume (output volume of (get volume settings) - 25)"])
	
	adjust_volume('up')　# 音量を上げる
	adjust_volume('down')　# 音量を下げる

再生・一時停止	# 再生状態を取得し、再生中なら一時停止、停止中なら再生
	current_playback = sp.current_playback()
	if current_playback and current_playback['is_playing']:
	　　sp.pause_playback()
	else:

	sp.start_playback()

ミュート・ミュート解除	def toggle_mute():
	script = '''
	set currentVolume to output muted of (get volume settings) # 現在の音量のミュート状態を取得
	if currentVolume is true then # ミュートされている場合
	set volume output muted false # ミュートを解除
	else # ミュートされていない場合
	set volume output muted true # ミュートに設定
	end if
	'''
	subprocess.run(["osascript", "-e", script]) # AppleScript を実行
	
	toggle_mute() # ミュート/解除を切り替え

楽曲スキップ・戻す	# 次の曲にスキップ def prevous_track(): sp.next_track() previous_track() #一曲前に戻す def prevous_track(): sp.previous_track() previous_track()

10 飛ばし・10 秒戻し	# 10 秒戻す
	def rewind_10_seconds():
	current_playback = sp.current_playback()
	if current_playback and current_playback['is_playing']:
	current_position = current_playback['progress_ms']
	new_position = max(0, current_position - 10000)
	sp.seek_track(new_position)

	rewind_10_seconds()
	
	# 10秒進める
	def fast_forward_10_seconds():
	    current_playback = sp.current_playback()
	    if current_playback and current_playback['is_playing']:
	        current_position = current_playback['progress_ms']
	        new_position = min(current_playback['item']['duration_ms'], current_position + 10000)
	        sp.seek_track(new_position)
	
	fast_forward_10_seconds()

再生位置の調整	# Spotifyデータを取得する関数
	
	def fetch_spotify_data():
	    """
	    Spotifyの現在の再生情報を取得して、再生位置と曲の長さを更新する。
	    """
	    global current_progress_ms, track_duration_ms, is_playing
	    playback = sp.current_playback()
	    if playback is not None:
	        is_playing = playback['is_playing']
	        if is_playing:
	            # 再生中の場合、再生位置と曲の長さを取得
	            current_progress_ms = playback['progress_ms']
	            track_duration_ms = playback['item']['duration_ms']
	        else:
	            # 再生していない場合、最後の再生位置と曲の長さを保持
	            current_progress_ms = playback['progress_ms']

```
            track_duration_ms = playback['item']['duration_ms']
    else:
        # 再生情報が取得できない場合、初期値を設定
        current_progress_ms = 0
        track_duration_ms = 1
        is_playing = False
# 指が再生バー上にある場合の処理
                if frame_height - 30 <= y_pos <= frame_height:
                    # 指の位置を再生位置にマッピング
                    new_progress_ms = int((x_pos / frame_width) *
track_duration_ms)
                    current_progress_ms = new_progress_ms

                    # Spotifyの再生位置を更新
                    sp.seek_track(new_progress_ms)
            hand_landmarks = results.multi_hand_landmarks[0]
```

4 章　実験と考察

4.1 実験内容

本研究では，ジェスチャー認識による spotify 操作システムの正常動作率を評価するための実験を実施する。被験者には,本研究で開発したシステムを実際に使用してもらい，操作の精度を確認する.具体的には，6 種類のジェスチャー認識による spotify の操作を各 10 回ずつ試みてもらう.その結果として，各操作において何回正しく認識され，意図通りの動作が行われたかを記録する.この実験を行う際に，3.3 のジェスチャー認識で説明した誤認識を防ぐために一定時間特定の操作を制御するシステムを導入しているため，操作を試みる際には操作の間を一定時間空けることとする.また，カメラが手を検出している間は常にジェスチャーを認識できる状態であるため，意図しないタイミングで何かしらのジェスチャーを誤認識する可能性がある.そのため，意図しないタイミングでジェスチャーを認識した際は誤認識の数として記録する.この実験を通して，システムの実用性や改善の余地を明らかにし，ジェスチャー認識の精度向上を目指す.

4.2 実験結果

被験者 6 名を対象に，システムの正常動作率を求めるための実験を実施した.その結果を表 4.1 に示す.表の中の数値は左から[正常動作数/ 不具合の数/ 意図していないタイミングでの誤認識の数]を表している.

ここでの不具合と誤認識の違いについて説明する，不具合とは 10 回操作を試みた際に，指定されたジェスチャーに対してシステムが正しく反応せず，動作しなかった場合や，他の動作が作動したなど，正常に動かなかったことを指す.一方，誤認識とは，意図的にジェスチャーを「行おう」と思っていない状況下，つまり意図していないタイミングで，手がカメラに写り込んでおり，その手の形状が誤って特定のジェスチャーとして認識され、意図しない動作が発生してしまうことを指す。例えば、操作するつもりのない場面でも、手が偶然あるジェスチャーの形状に近いと判断されて、誤作動が起きるケースを含む。

表 4.1 実験結果［正常動作数/不具合の数/意図していないタイミングでの誤認識］

操作内容 / 被験者	音量調整	再生・一時停止	ミュート ミュート解除	曲のスキップ・戻す	10 秒飛ばし・戻す	再生位置の調整
A	8/ 2/ 2	10/ 0/ 3	5/ 5 / 5	10/ 0/ 0	7/ 3/ 0	7/ 3/ 0
B	6/ 4/ 1	9/ 1/ 2	6/ 4/ 2	9/ 1/ 1	8/ 2/ 1	8/ 2/ 0
C	7/ 3/ 2	10/ 0/ 0	7/ 3/ 5	6/ 4/ 0	6/ 4/ 0	7/ 3/ 0
D	7/ 3/ 3	10/ 0/ 0	6/ 4/ 2	8/ 2/ 0	8/ 2/ 0	9/ 1/ 0
E	7/ 3/ 1	8/ 2/ 0	7/ 3/ 4	9/ 1/ 0	7/ 3/ 1	8/ 2/ 0
F	8/ 2/ 1	9/ 1/ 1	5/ 5/ 4	10/ 0/ 0	7/ 3/ 1	7/ 3/ 0

本実験の結果から各操作の正常動作率と平均誤認識数を求めると以下のようになる.

1. 音量調整
   正常動作率: 71.67%
   平均誤認識数: 1.67 回

2. 再生・一時停止
   正常動作率: 93.33%
   平均誤認識数: 1.0 回

3. ミュート・ミュート解除
   正常動作率: 60%
   平均誤認識数: 3.67 回

4. 曲のスキップ・戻す
   正常動作率: 86.67%
   平均誤認識数: 0.17 回

5. 10 秒飛ばし・戻す
   正常動作率：71.67%
   平均誤認識数：0.5 回

6. 再生位置の調整
   正常動作率：76.67%
   平均誤認識数：0 回

4.3 考察

結果として，ジェスチャーの種類によって正常動作率および誤認識数に差が見られる結果となった. 特に再生・一時停止の操作は精度が高く，逆にミュート・ミュート解除の操作は精度が低かった. 原因として，再生一時停止の操作のジェスチャーは静止している状態であり、加えて他のジェスチャーとあまり類似していないためだと考えられる. ミュート・ミュート解除の操作のジェスチャーも静止している状態であるが，音量調整のジェスチャーの際の手の形状と似ていたため，誤認識数も多く，正常動作率が低かったと考えられる. その他の動作に関しては，全て動きが伴うジェスチャーであり，個人差による影響が顕著に現れた. 被験者の中には，システムの認証条件に適合する動きを自然に行える人がいる一方で，そうでない人もいた. この結果被験者間で正常動作率に対する差が生じる傾向が見られた. 特に手の動きや位置がシステムの期待するパターンから外れていた場合，認識精度が低下する傾向が確認できた.

以上のことから、ジェスチャー間の境界をより明確にする必要や，システムの認識条件をより柔軟にする、またはジェスチャーのデザインを個人差に対して寛容なものにするなど、改良の余地があると考えられる.

5章　結論

本研究ではジェスチャー認識による音楽再生アプリ spotify の操作を行うシステムを構築した. その結果, spotify の一般的操作をジェスチャーによって行うことができた. システムの構築に成功したが, いくつか課題点が残っている. 操作によっては誤認識が多く, 精度が十分でないものが確認された. また, 被験者によってシステムの認証条件への適合に差が生じた. これらの課題を解決するためには, ジェスチャー間の境界をより明確にする必要や, システムの認識条件をより柔軟にする. またはジェスチャーのデザインを個人差に対して寛容なものにするなど, 幅広いユーザーに対応可能なシステムの実現が期待できる.

今後は, 認識アルゴリズムの精度向上やジェスチャーの再設計, さらにはシステムの閾値設定の最適化を通して, 制度や実用性をさらに向上させることを目指していく.

参考文献

[1] https://ictr.co.jp/report/20221111.html/

2022 年 定額制音楽配信サービス利用動向に関する調査

[2] https://deallab.info/music-streaming/

音楽配信サービス業界の世界市場シェアの分析

[3] https://kwcplus.kddi-web.com/blog/what-is-api

API とは

[4] https://qiita.com/bianca26neve/items/116814135739929759a0

Mediapipe による手の形状検出

[5] https://developer.spotify.com

Spotify for developer

[6] https://nehapire.com/web/spotify-api/

Spotify api

謝辞

本論文を作成にあたり,丁寧な御指導を賜りました蚊野浩教授に感謝いたします.

付録　開発したプログラム

[ファイル名]

jspotify.py

[内容]

ジェスチャーを認識し，そのジェスチャーに対応した spotify の操作を行う．

[関数]

関数	内容
last_10rewind_time	10 秒戻しを行なった時刻
last__10skip_time	10 秒飛ばしを個なった時刻
last_spotify_update_time	最後に spotify データを更新した時間
current_progress_ms	現在の再生位置
track_duration_ms	現在再生中の曲の長さ
is_playing	曲の再生状態の有無
is_muted	現在のミュート状態
gesture_start_time_mute	ミュート・解除が開始された時刻
gesture_start_time_playback	再生・停止が開始された時刻
last_action_time_mute	最後のミュート・解除が行われた時刻
last_action_time_playback	最後の再生・停止が行われた時刻
last_x_position	人差し指の x 座標を記録する変数
last_skip_time	曲をスキップした時間
last_rezind_time	最後に曲を戻した時間
last_distance	親指と人差し指の距離
thumb_tip	親指の先の座標
index_finger_tip	人差し指の先の座標
middle_finger_tip	中指の先の座標
finger_point	小指の先のランドマーク