令和４年度卒業研究報告書

MediaPipe を用いたスポーツデータ分析

情報技術科　高橋　優也

指導教員　　石舘　勝好

目次

第 1 章　はじめに 2

第 2 章　研究概要 3

2.1 概要 3

2.2 使用したライブラリ 4

第 3 章　環境構築 6

3.1 システムの環境構築 6

3.2 バッチファイルの作成 9

第 4 章　骨格の検出 10

4.1 MediaPipe による Landmark の取得 10

4.2 静止画による骨格検出方法 11

4.3 動画による骨格検出方法 16

第 5 章　骨格座標データの書き出し 21

第 6 章　可視化の課題と分析方法の検討 32

6.1 骨格データ可視化における課題 32

6.2 分析方法の検討 34

6.3 Matplotlib を用いたグラフの作成 36

第 7 章　実験及び考察 45

7.1 実験方法 45

7.2 実験の結果と考察 46

7.3 課題 51

第 8 章　終わりに 52

第 9 章　参考文献 53

第1章　はじめに

スポーツをする際に、プレーの質を向上させるためには客観的な視点からプレーを振り返ることが必要不可欠である。プロスポーツの分野では客観的に動きをとらえるために複数のカメラやモーションキャプチャツールを用いてデータの収集を行い、分析までを行っている。しかし、データを集める際の機器は高価なものが多く、一般の人や中高生には手が出せない状況となっており、普及が進んでいない状況である。また、一番導入すべきと考えている、中高生などのスポーツ指導の場面での普及も進んでおらず、中高生のスポーツ指導においてデータの分析をしている学校としていない学校で差が出てしまうこともある。

そこで,身近なカメラからデータ収集を行うことでデータ分析の際にかかる費用を抑え、中高生の指導に生かせるようにしたいと考えた。そこで、MediaPipe を用いて動画から骨格データや関節座標を取得し、座標データを用いてスポーツプレイヤーの動きを分析できると考えこの研究を進めることとした。

第 2 章　研究概要

2.1 概要

本研究は Python のライブラリである MediaPipe や Matplotlib の学習を行うとともに、開発環境の構築、プログラムの作成、取得データの可視化、比較を行う。今回はバレーボールのサーブの動きをカメラを用いて撮影し、撮影した動画から骨格座標データを取得し、グラフによる可視化までを目標とすることにした。

本研究には以下の開発環境と機材を使用する。

表 1　開発環境と機材

OS	Windows10 Pro
使用言語	Python3.9.13
統合開発環境	Visual Stadio Code（以下 VSCode とする）
ライブラリ	OpenCV
	MediaPipe
	NumPy
	Pandas
	Matplotlib
カメラ	ZV-E10L(B)

2.2 使用したライブラリ

2.2.1 OpenCV

OpenCV は、画像処理・画像解析および機械学習等の機能を持つ C/C++、Java、Python、MATLAB 用ライブラリである。Intel が開発したライブラリで、OSS として提供されているため、誰でも無料で使うことができる。趣味範囲の利用はもちろん、商業目的の利用も問題ありません。画像解析処理の前段階で使うことが多く、今回の研究でも土台となっている。また、本研究では動画の処理やカメラでの関節の動きのキャプチャーに使用している。

2.2.2 MediaPipe

MediaPipe とは、Google で開発を主導しているオープンソースの機械学習ライブラリで、ライブストリーミングでの使用に特化している。この MediaPipe を利用すると高性能な AI 画像処理アルゴリズムを利用した AR アプリケーション等を簡単に作成できる。

本研究では、Python 環境で利用可能な骨格の検出、関節座標の取得を利用している。MediaPipe は Python 環境では他にも以下の機能が利用可能である。

- 手の認識
- ポーズの認識
- フェイスメッシュ（表情の認識）
- ホリスティック（フェイスメッシュとポーズの両認識）
- 顔の認識
- オブジェクトロン（物体の認識）
- セルフィーセグメンテーション（背景の合成）

2.2.3 NumPy

プログラミング言語 Python において数値計算を効率的に行うための拡張ライブラリである。効率的な数値計算を行うための型付きの多次元配列（例えばベクトルや行列などを表現できる）のサポートを Python に加えるとともに、それらを操作するための大規模な高水準の数学関数ライブラリを提供する。本研究では画面上の座標を計算する処理に利用する。

2.2.4 Pandas

Pandas は、プログラミング言語 Python において、データ解析を支援する機能を提供するライブラリである。特に、数表および時系列データを操作するためのデータ構造と演算を提供する。Pandas は BSD ライセンスのもとで提供されている。本研究では、取得した座標データを配列に格納し、CSV ファイルに書き出しを行っている。

2.2.5 Matplotlib

Matplotlib は、プログラミング言語 Python およびその科学計算用ライブラリ NumPy のためのグラフ描画ライブラリである。オブジェクト指向の API を提供しており、様々な種類のグラフを描画する能力を持つ。描画できるのは主に 2 次元のプロットだが、3 次元プロットの機能も追加されてきている。描画したグラフを各種形式の画像（各種ベクトル画像形式も含む）として保存することもできるし、wxPython、Qt、GTK といった一般的な GUI ツールキット製のアプリケーションにグラフの描画機能を組みこむこともできる。本研究では、配列に格納した座標データを散布図として表示するのに使っている。

第 3 章　環境構築

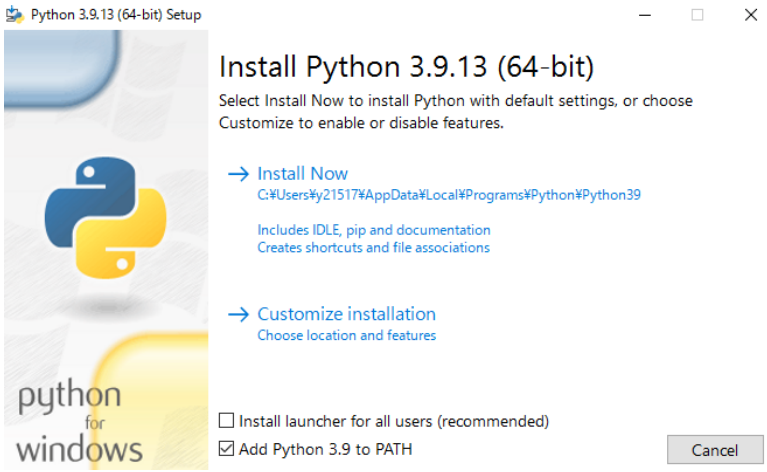
3.1 システムの環境構築

3.1.1　環境のインストール

（1） Python のインストール

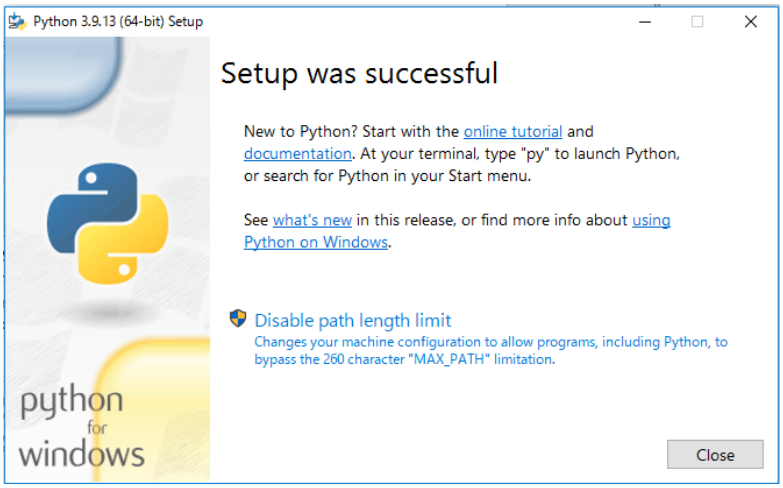
本研究では Python3.9.13 を利用する。

まずは、Python 公式ホームページ（https://www.python.org/downloads/）から使いたいバージョンの Python のインストーラーをダウンロードする。今回は「python-3.9.13-amd64.exe」を起動する。



ウィンドウ下にある「Add Python 3.9 to PATH」にチェックマークを付ける。このチェックマークをつけ忘れると Python をインストールしてもアプリを実行できない。

インストールが完了すると下記の画面が表示される。「Close」クリックし、インストーラーを閉じる。



(2) VSCode の環境構築

次に統合開発環境である VSCode をインストールする。

まずは、Visual Studio Code 公式ホームページ（https://code.visualstudio.com/download）から VSCode のインストーラーをダウンロードする。

この時も Python 同様に「Path への追加」にチェックを付ける。

3.1.2　使用ライブラリをインストール

Python をインストール後、コマンドプロンプトか VSCode のターミナルで以下のコマンドを実行する。校内でなければプロキシの設定を行う必要はない。

表 1　コマンド入力する処理

コマンド	行う処理
set HTTP_PROXY=http://172.16.0.2:8080	校内プロキシの設定(HTTP)
set HTTPS_PROXY=http://172.16.0.2:8080	校内プロキシの設定(HTTPS)
pip install opencv-python	OpenCV のインストール
pip install mediapipe	Mediapipe のインストール
pip install numpy	NumPy のインストール
pip install pandas	pandas のインストール
pip install matplotlib	Matplotlib のインストール

3.2 バッチファイルの作成

バッチファイルの作成を行っておくと、ライブラリが不足した場合にライブラリを追加でインストールする際にプロキシの設定をする必要がなくなる。(VSCode のターミナルは作業状況を保存してくれるのでそちらで開発する場合は不要)

メモ帳を開き、下記の内容を入力する。

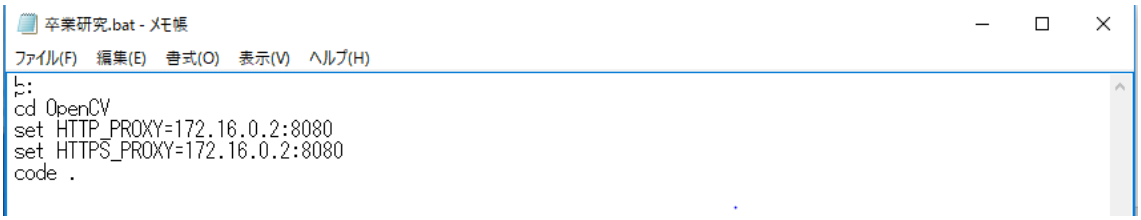


図 1　BAT ファイルの内容

コードの内容は以下のようになる。

(1)　C:（1 行目）

C ドライブに階層を移動

(2)　cd OpenCV (2 行目)

OpenCV のフォルダに階層を移動

(3)　3、4 行目

校内プロキシの設定を行う。3 行目(HTTP 用)、4 行目(HTTPS 用)。それぞれ行う必要がある。学校外での作業（学内 LAN の使用）でなければ設定する必要はない。

(4)　Code .

VSCode を起動する

※ライブラリのインストールやプログラムのビルドもここで行うこともできる。

第 4 章　骨格の検出

本研究では骨格の動きを動画から取得するプログラムを作成した。このプログラムでは MediaPipe で定義されている骨格（landmark）の座標を取得することで骨格の様子を検出している。

4.1 MediaPipe による Landmark の取得

MediaPipe では姿勢推定のランドマークの位置を以下のように保持している。(図３)

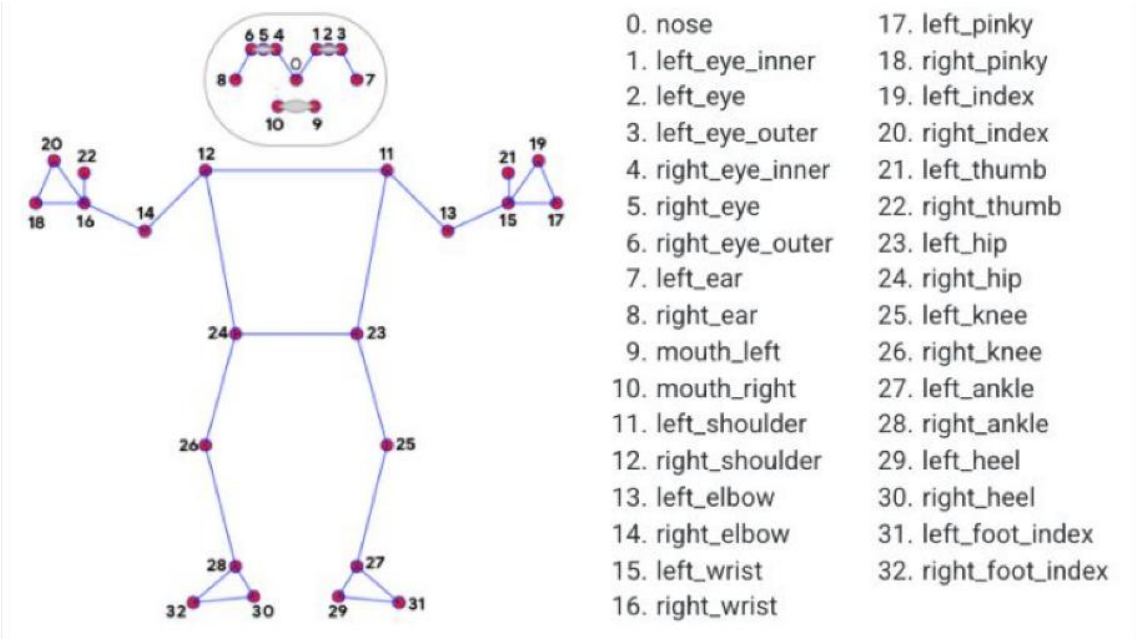


図 2　MediaPipe で定義される Landmark

図 3 は姿勢推定で検出されたランドマークがどのように保持されているかを示す図になっている。MediaPipe では一人の姿勢推定から合計 33 個のランドマークを検出できる。各ランドマークには ID が割り振られており、鼻であれば ID 番号 0 番、右つま先であれば 32 番といったように ID 番号で指定することで、特定の Landmark の座標を取得することができる。

4.2 静止画による骨格検出方法

はじめは実験的に静止画による骨格検出を行った。下記が骨格推定前の画像である。(図 4)

あえて一人が画面内に映っている画像ではなく、複数人が写っている画像を使用することで複数人の時にどのような処理をするのか調べることにした。

図 3　骨格検出前の画像（実験用）

静止画による骨格検出のプログラムを以下に示す。

リスト 1　静止画による骨格検出プログラム(test1.py)

```
#ライブラリのインポート
import mediapipe as mp
import cv2

#姿勢推定ソリューションのインスタンス化
mp_pose = mp.solutions.pose
#描画ソリューションの呼出しと描画時の設定
mp_drawing = mp.solutions.drawing_utils
mesh_drawing_spec = mp_drawing.DrawingSpec(thickness=2,color=(0,255,0))
mark_drawing_spec = mp_drawing.DrawingSpec(thickness=3,circle_radius=3,color=(0,0,255))
```

```
#入力データ（画像パス）を変数に格納
img_path='bb.jpg'

#骨格検出および描画に関する設定
with mp_pose.Pose(
    #検出時の信頼性の高さの設定
    min_detection_confidence=0.5,
    static_image_mode=True) as pose_detection:
    #画像の読み込み
    image = cv2.imread(img_path)
    #画像のサイズ調整
    image = cv2.resize(image,dsize=None,fx=0.3,fy=0.3)
    #色チャンネルを MediaPe 用に調整
    rgb_image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    #画像の縦、横のサイズ
    height=rgb_image.shape[0]
    width=rgb_image.shape[1]
    #MediaPipe を使った骨格の検出処理
    results=pose_detection.process(rgb_image)
    #元画像のコピー
    annotated_image=image.copy()
    #左肩と右足首のランドマーク座標の出力
    print('x:',results.pose_landmarks.landmark[11].x)
    print('y:',results.pose_landmarks.landmark[11].y)
    print('x:',results.pose_landmarks.landmark[28].x)
    print('y:',results.pose_landmarks.landmark[28].y)
#コピーした画像にランドマークを描画
mp_drawing.draw_landmarks(
    image=annotated_image,
    landmark_list=results.pose_landmarks,
    connections=mp_pose.POSE_CONNECTIONS,
    landmark_drawing_spec=mark_drawing_spec,
    connection_drawing_spec=mesh_drawing_spec
    )
```

```
#書き出し処理
cv2.imwrite('result.jpg',annotated_image)
```

4.2.1　静止画による骨格検出プログラムの解説

```
#ライブラリのインポート
import mediapipe as mp
import cv2
#姿勢推定ソリューションのインスタンス化
mp_pose = mp.solutions.pose
```

初めに OpenCV と Mediapipe をインポートする。環境作成の際にインストールはしているが、インポートの処理を行わないと作成したスクリプト（関数など）が path に書き込まれず、処理が行われなくなってしまうため、注意が必要である。

その後 MediaPipe に定義されている姿勢推定ソリューションを呼び出すインスタンスを作成する。

```
#描画ソリューションの呼出しと描画時の設定
mp_drawing = mp.solutions.drawing_utils
mesh_drawing_spec = mp_drawing.DrawingSpec(thickness=2,color=(0,255,0))
mark_drawing_spec = mp_drawing.DrawingSpec(thickness=3,circle_radius=3,color=(0,0,255))

#入力データ（画像パス）を変数に格納
img_path='bb.jpg'
```

次に骨格を書き出すための処理を作成する。描画ソリューションの呼出しと骨格描画時の色や関節の位置を表示する円の大きさなどをここで指定する。また、読み込む画像データのパスを変数に格納しておくと後々、処理がしやすくなる。

```
#骨格検出および描画に関する設定
with mp_pose.Pose(
    #検出時の信頼性の高さの設定
    min_detection_confidence=0.5,
    static_image_mode=True) as pose_detection:
    #画像の読み込み
```

```
    image = cv2.imread(img_path)
    #画像のサイズ調整
    image = cv2.resize(image,dsize=None,fx=0.3,fy=0.3)
    #色チャンネルを MediaPe 用に調整
    rgb_image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    #画像の縦、横のサイズ
    height=rgb_image.shape[0]
    width=rgb_image.shape[1]
    #MediaPipe を使った骨格の検出処理
    results=pose_detection.process(rgb_image)
    #元画像のコピー
    annotated_image=image.copy()
```

with を処理の最初につけることで処理の開始と終了を考慮する必要がなくなる。min_detection_confidence では、骨格検出の正確性を 0〜1 で設定することができる。1 に近いほど正確に検出できるが少しでも違えばほとんど検出されなくなる可能性がある。逆に 0 に近いほど骨格のようなものはほとんど表示できるようになる。しかし、誤検出も増えるといったトレードオフの関係が成立している。今回はデフォルトの 0.5 を利用する。また、OpenCV の色チャンネルが BGR で保持しているのに対し、MediaPipe は RGB で処理するため、MediaPipe に合わせる必要がある。

```
#コピーした画像にランドマークを描画
mp_drawing.draw_landmarks(
    image=annotated_image,
    landmark_list=results.pose_landmarks,
    connections=mp_pose.POSE_CONNECTIONS,
    landmark_drawing_spec=mark_drawing_spec,
    connection_drawing_spec=mesh_drawing_spec
    )
#書き出し処理
cv2.imwrite('result.jpg',annotated_image)
```

コピーしてきた画像にランドマークの座標を表示する。また、各ランドマークとそれらを結ぶメッシュを描画するオプションも指定する。メッシュを描くためのは connection を指定する必要があるが、mp_face_mesh インスタンスの中の FACEMESH_TESSELATION を使用している。

4.2.2　実行結果

このプログラムを実行した結果は以下のように骨格を検出する。

また、複数人映り込んだ場合でも一人分しか検出することはできないようだ。また複数人が重なっている場合は、骨格が正しく表示されなくなってしまうことがある。

図 4　プログラム実行後に生成される画像

またメイン処理内の以下の処理で座標取得の様子を描画した。

```
#左肩と右足首のランドマーク座標の出力
print('x:',results.pose_landmarks.landmark[11].x)
print('y:',results.pose_landmarks.landmark[11].y)
print('x:',results.pose_landmarks.landmark[28].x)
print('y:',results.pose_landmarks.landmark[28].y)
```

プログラムで設定した左肩と右足首の座標は以下のようになる。

```
PS C:\Users\y21517\Desktop\OpenCV> python test2.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
x: 0.856561005115509
y: 0.41206133365631104
x: 0.7867127656936646
y: 0.8069928884506226
```

図5　左肩と右足首の座標

座標は画面のどの位置にあるのか割合的に算出するものになっている。

4.3 動画による骨格検出方法

次に動画による骨格検出を行った。

バレーボールのサーブの動きをカメラで撮影し、その動画から骨格の動きを検出する。(図7)

図6　プログラム実行前の動画

動画による骨格検出のプログラムの全体を以下に示す。

リスト2　動画による骨格検出プログラム

```
import mediapipe as mp
import cv2

mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils
mesh_drawing_spec = mp_drawing.DrawingSpec(thickness=2,color=(0,225,0))
mark_drawing_spec = mp_drawing.DrawingSpec(thickness=2,circle_radius=2,color=(0,0,255))

cap_file = cv2.VideoCapture('C0324.mp4')
```

```
with mp_holistic.Holistic(
    #動画なので static_image_mode の処理は false を返す
     min_detection_confidence=0.5,
     static_image_mode=False) as holistic_detection:

  while cap_file.isOpened():
     success, image = cap_file.read()
     if not success:
          print("empty camera frame")
          break
     image = cv2.resize(image,dsize=None, fx=1.0,fy=1.0)
     rgb_image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

     results = holistic_detection.process(rgb_image)
     mp_drawing.draw_landmarks(
          image=image,
          landmark_list=results.pose_landmarks,
          connections=mp_holistic.POSE_CONNECTIONS,
          landmark_drawing_spec=mesh_drawing_spec
          )
     cv2.imshow('holistic detection',image)
     if cv2.waitKey(5) & 0xFF==27:
          break
cap_file.release()
```

4.3.1　動画による骨格検出プログラムの解説

```
import mediapipe as mp
import cv2

mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils
mesh_drawing_spec = mp_drawing.DrawingSpec(thickness=2,color=(0,225,0))
mark_drawing_spec = mp_drawing.DrawingSpec(thickness=2,circle_radius=2,color=(0,0,255))

cap_file = cv2.VideoCapture('C0324.mp4')
```

「4.2.1 静止画による骨格検出プログラムの解説」で行った処理と同様にインポートと描画時の設定を行い、動画のパスも変数に格納する。今回は動画なので VideoCapture クラスを使用する。

```
with mp_holistic.Holistic(
    #動画なので static_image_mode の処理は false を返す
     min_detection_confidence=0.5,
     static_image_mode=False) as holistic_detection:
```

今回も骨格検出の正確性を設定することができる。こちらも静止画による骨格検出と同じである。しかし、static_image_mode は動画のため引数を False にする必要がある。

```
while cap_file.isOpened():
    success, image = cap_file.read()
    if not success:
        print("empty camera frame")
        break
    image = cv2.resize(image,dsize=None, fx=1.0,fy=1.0)
    rgb_image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    results = holistic_detection.process(rgb_image)
```

動画データが問題なく開けたときは処理を続け、何らかの原因で動画が開けなくなった場合は自動的に処理を終了する仕組みになっている。動画データの読み込みに成功したか否かを isOpened 関数を使う。

また、read 関数を使用し 1 枚 1 枚の画像として連続的に読み込み処理を行う。read 関数の返り値として読み込み結果（True または False）を保持するためのものと画像データそのものである。また読み込み失敗時に繰り返し処理を中断する処理も追加する。

```
    mp_drawing.draw_landmarks(
        image=image,
        landmark_list=results.pose_landmarks,
        connections=mp_holistic.POSE_CONNECTIONS,
        landmark_drawing_spec=mesh_drawing_spec
        )
    cv2.imshow('holistic detection',image)
#描画中断処理
    if cv2.waitKey(5) & 0xFF==27:
        break
cap_file.release()
```

描画時の処理は「4.2.1 静止画による骨格検出プログラムの解説」の場合と同じく処理を入力する。描画処理の際に rgb_image を利用してしまうと画面全体の色がおかしくなってしまうのでここでは使わないようにしている。

また、動画の途中であっても再生が中断できるように処理を追加した。Waitkey でキーボード入力

を受け付ける処理になっている。Waitkey の引数はキーボード入力を受け付ける時間をミリ秒単位で指定する。今回は 5 ミリ秒としている。また 0xFF==27 は Esc キーを表しており、Esc キーを押されてから 5 秒で処理が終了する仕組みになっている。

このプログラムを実行した結果が図 7 のようになっている。

図 7　骨格検出プログラム実行後の動画

第5章　骨格座標データの書き出し

次に4章時点までで作成したプログラムと撮影した動画を利用して座標データを取得することにした。また、4章までで作成したプログラムではMediaPipeで定義されているlandmarkをすべて取得してしまうため必要な部分だけを取得するプログラムに書き換える必要がある。そして、このプログラムで取得した座標データをPandasでexcelファイルに書き出す処理も追加し、どのような座標推移がみられるのか検証することにした。

欲しい骨格データの書き出しをするためにプログラムを作成した。

プログラムの解説は一部4章と内容が同じものがあるのでそこは割愛する。

リスト3　骨格座標データ取得プログラム

```
import mediapipe as mp
import cv2
import numpy as np
import pandas as pd

def get_keypoint(results,height,width):
    left_shoulder_x = int(results.pose_landmarks.landmark[11].x * width)
    left_shoulder_y = int(results.pose_landmarks.landmark[11].y * height)
    l_shoulder_y =  height-left_shoulder_y
    left_shoulder_xy = [left_shoulder_x,left_shoulder_y]

    right_shoulder_x = int(results.pose_landmarks.landmark[12].x * width)
    right_shoulder_y = int(results.pose_landmarks.landmark[12].y * height)
    r_shoulder_y =  height-right_shoulder_y
    right_shoulder_xy = [right_shoulder_x,right_shoulder_y]

    left_elbow_x = int(results.pose_landmarks.landmark[13].x * width)
    left_elbow_y = int(results.pose_landmarks.landmark[13].y * height)
    l_elbow_y =  height-left_elbow_y
```

```
left_elbow_xy = [left_elbow_x,left_elbow_y]

right_elbow_x = int(results.pose_landmarks.landmark[14].x * width)
right_elbow_y = int(results.pose_landmarks.landmark[14].y * height)
r_elbow_y =    height-right_elbow_y
right_elbow_xy = [right_elbow_x,right_elbow_y]

left_wrist_x = int(results.pose_landmarks.landmark[15].x * width)
left_wrist_y = int(results.pose_landmarks.landmark[15].y * height)
l_wrist_y = height - left_wrist_y
left_wrist_xy = [left_wrist_x,left_wrist_y]

right_wrist_x = int(results.pose_landmarks.landmark[16].x * width)
right_wrist_y = int(results.pose_landmarks.landmark[16].y * height)
r_wrist_y = height-right_wrist_y
right_wrist_xy = [right_wrist_x,right_wrist_y]

left_hip_x = int(results.pose_landmarks.landmark[23].x * width)
left_hip_y = int(results.pose_landmarks.landmark[23].y * height)
l_hip_y = height-left_hip_y
left_hip_xy = [left_hip_x,left_hip_y]

right_hip_x = int(results.pose_landmarks.landmark[24].x * width)
right_hip_y = int(results.pose_landmarks.landmark[24].y * height)
r_hip_y = height - right_hip_y
right_hip_xy = [right_hip_x,right_hip_y]

left_knee_x = int(results.pose_landmarks.landmark[25].x * width)
left_knee_y = int(results.pose_landmarks.landmark[25].y * height)
l_knee_y = height - left_knee_y
left_knee_xy = [left_knee_x,left_knee_y]

right_knee_x = int(results.pose_landmarks.landmark[26].x * width)
right_knee_y = int(results.pose_landmarks.landmark[26].y * height)
r_knee_y = height - right_knee_y
```

```
    right_knee_xy = [right_knee_x,right_knee_y]

    left_ankle_x = int(results.pose_landmarks.landmark[27].x * width)
    left_ankle_y = int(results.pose_landmarks.landmark[27].y * height)
    l_ankle_y = height - left_ankle_y
    left_ankle_xy = [left_ankle_x,left_ankle_y]

    right_ankle_x = int(results.pose_landmarks.landmark[28].x * width)
    right_ankle_y = int(results.pose_landmarks.landmark[28].y * height)
    r_ankle_y = height - right_ankle_y
    right_ankle_xy = [right_ankle_x,right_ankle_y]

    data=[left_shoulder_x, l_shoulder_y,right_shoulder_x,r_shoulder_y,left_elbow_x, l_elbow_y,
    right_elbow_x,r_elbow_y,left_wrist_x,l_wrist_y,right_wrist_x,r_wrist_y,left_hip_x,l_hip_y,
    right_hip_x,r_hip_y, left_knee_x,l_knee_y,right_knee_x,r_knee_y,left_ankle_x,l_ankle_y,
    right_ankle_x,r_ankle_y]
    return data,left_shoulder_xy,right_shoulder_xy,left_elbow_xy,right_elbow_xy,left_wrist_xy,
    right_wrist_xy,left_hip_xy,right_hip_xy,left_knee_xy,right_knee_xy,left_ankle_xy,right_ankle_xy

def  draw_keypoint(image,RADIUS, CLR_KP,CLR_LINE,THICKNESS,left_shoulder_xy,
       right_shoulder_xy,left_elbow_xy,right_elbow_xy,left_wrist_xy,right_wrist_xy,left_hip_xy,
       right_hip_xy,left_knee_xy,right_knee_xy,left_ankle_xy,right_ankle_xy):
   cv2.circle(image, (left_shoulder_xy[0], left_shoulder_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image,(right_shoulder_xy[0],right_shoulder_xy[1]),RADIUS,CLR_KP, THICKNESS)
   cv2.circle(image, (left_elbow_xy[0], left_elbow_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (right_elbow_xy[0], right_elbow_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (left_wrist_xy[0], left_wrist_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (right_wrist_xy[0], right_wrist_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (left_hip_xy[0], left_hip_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (right_hip_xy[0], right_hip_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (left_knee_xy[0], left_knee_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (right_knee_xy[0], right_knee_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (left_ankle_xy[0], left_ankle_xy[1]), RADIUS, CLR_KP, THICKNESS)
   cv2.circle(image, (right_ankle_xy[0], right_ankle_xy[1]), RADIUS, CLR_KP, THICKNESS)
```

```
    cv2.line(image,(left_shoulder_xy[0],left_shoulder_xy[1]),(left_elbow_xy[0],left_elbow_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)

    cv2.line(image,(right_shoulder_xy[0],right_shoulder_xy[1]),(right_elbow_xy[0],
    right_elbow_xy[1]),CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(left_elbow_xy[0],left_elbow_xy[1]),(left_wrist_xy[0],left_wrist_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_elbow_xy[0],right_elbow_xy[1]),(right_wrist_xy[0],right_wrist_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(left_shoulder_xy[0],left_shoulder_xy[1]),(left_hip_xy[0],left_hip_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_shoulder_xy[0],right_shoulder_xy[1]),(right_hip_xy[0],right_hip_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(left_hip_xy[0],left_hip_xy[1]),(left_knee_xy[0],left_knee_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(left_shoulder_xy[0],left_shoulder_xy[1]),(right_shoulder_xy[0],
    right_shoulder_xy[1]),CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_hip_xy[0],right_hip_xy[1]),(left_hip_xy[0],left_hip_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_knee_xy[0],left_knee_xy[1]),(right_hip_xy[0],right_hip_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_knee_xy[0],left_knee_xy[1]),(right_ankle_xy[0],right_ankle_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(left_knee_xy[0],left_knee_xy[1]),(left_ankle_xy[0],left_ankle_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)


    return image

if __name__ == "__main__":
    count = 0
    THRESH_SLOPE = 0
    THRESH_DiST_SPINE = 30
    THRESH_ARM = 40
```

```
RADIUS = 5
THICKNESS = 2
CLR_KP = (0,0,255)
CLR_LINE = (255,255,255)

dataPOS=pd.DataFrame()
mp_pose=mp.solutions.pose
cap_file=cv2.VideoCapture('C0324.MP4')

with mp_pose.Pose(
    min_detection_confidence=0.5,
    static_image_mode=False)as pose_detection:

    while cap_file.isOpened:
        success,image = cap_file.read()
        if not success:
            print("empty camera frame")
            break
        image = cv2.resize(image,dsize=None,fx=1.0,fy=1.0)
        rgb_image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        height = rgb_image.shape[0]
        width = rgb_image.shape[1]

        results = pose_detection.process(rgb_image)

        if not results.pose_landmarks:
            print('not results')
        else:
            data,left_shoulder_xy,right_shoulder_xy,left_elbow_xy,right_elbow_xy,
            left_wrist_xy,right_wrist_xy,left_hip_xy,right_hip_xy,left_knee_xy,right_knee_xy,
            left_ankle_xy,right_ankle_xy = get_keypoint(results, height, width)
            rows = pd.Series(data)
            dataPOS=dataPOS.append(rows,ignore_index=True)

        image=draw_keypoint(image,RADIUS,CLR_KP,CLR_LINE,THICKNESS,
```

```
            left_shoulder_xy,right_shoulder_xy,left_elbow_xy,right_elbow_xy,left_wrist_xy,
            right_wrist_xy,left_hip_xy,right_hip_xy,left_knee_xy,right_knee_xy,left_ankle_xy,
            right_ankle_xy)
            #対応は MP4 のみ
            cv2.imshow('AI personal trainer', image)
            if cv2.waitKey(5) & 0xFF == 27:
              dataPOS.columns=['L_shoulderX','L_shoulderY','R_shoulderX','R_shoulderY',
              'L_elbowX','L_elbowY','R_elbowX','R_elbowY','L_wristX','L_wristY','R_wristX',
              'R_wristY','L_hipX','L_hipY','R_hipX','R_hipY','L_kneeX','L_kneeY','R_kneeX',
              'R_kneeY','L_ankleX','L_ankleY','R_ankleX','R_ankleY']
              dataPOS.to_excel("P:¥¥卒業研究¥¥body_pos17_data.xlsx")
              break
cap_file.release()
cv2.destroyAllWindows()
```

5.1.1　骨格座標データの取得プログラムの解説

```
def get_keypoint(results,height,width):
    left_shoulder_x = int(results.pose_landmarks.landmark[11].x * width)
    left_shoulder_y = int(results.pose_landmarks.landmark[11].y * height)
    l_shoulder_y =   height-left_shoulder_y
    left_shoulder_xy = [left_shoulder_x,left_shoulder_y]
```

このプログラムでは肩、肘、手首、腰、膝、足首の関節を取得するために各関節用の処理を行う。

ランドマーク取得関数を定義する。この関数には 3 つの引数が必要である。姿勢推定で得られた結果を results、さらにはリサイズした画像の高さ height と幅 width で設定する。

result.pose_landmarks.landmark[]で座標を取得している。この座標は正規化された値であるため、それぞれを画像の高さと幅で掛けることで元画像の座標に変換する。x,y 座標はリスト型でペアにしておく。

座標の原点が下記のように設定されている。(図８)

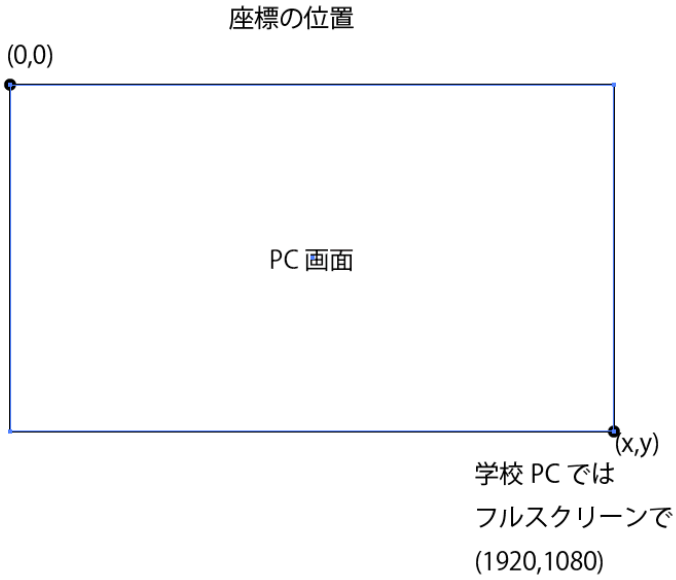


図 8　Python で定義される座標位置

現在は原点が画面左上だが、今後グラフ化する際にグラフが x 軸で反転してしまうため、

以下の処理で原点の位置を左下に修正する。

```
l_shoulder_y =　height-left_shoulder_y
```

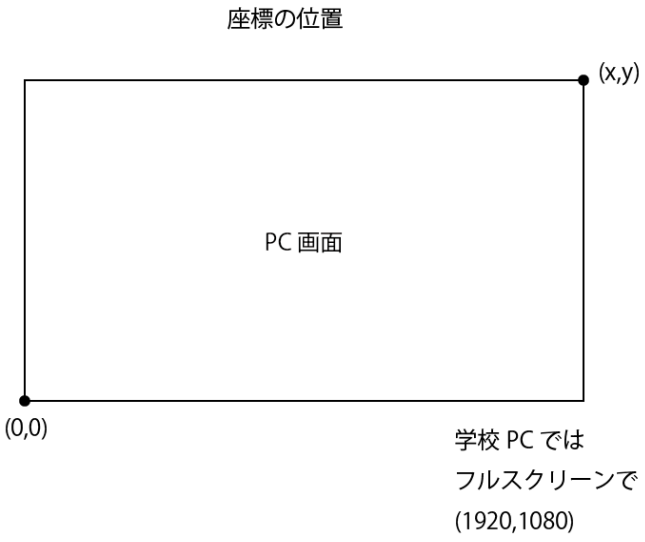


図 9 原点修正後の座標位置

```
data=[left_shoulder_x, l_shoulder_y,right_shoulder_x,r_shoulder_y,left_elbow_x, l_elbow_y,
      right_elbow_x,r_elbow_y,left_wrist_x,l_wrist_y,right_wrist_x,r_wrist_y,left_hip_x,l_hip_y,
      right_hip_x,r_hip_y, left_knee_x,l_knee_y,right_knee_x,r_knee_y,left_ankle_x,l_ankle_y,
      right_ankle_x,r_ankle_y]
      return data,left_shoulder_xy,right_shoulder_xy,left_elbow_xy,right_elbow_xy,left_wrist_xy,
      right_wrist_xy,left_hip_xy,right_hip_xy,left_knee_xy,right_knee_xy,left_ankle_xy,right_ankle_xy
```

先ほど取得したランドマークデータを data に格納する。data へ格納した各関節データは pandas で配列への書き出し処理を行う際に使う。また、格納したデータと各関節の x, y 座標を返り値として返す処理も追加しておく。

```
def   draw_keypoint(image,RADIUS, CLR_KP,CLR_LINE,THICKNESS,left_shoulder_xy,
        right_shoulder_xy,left_elbow_xy,right_elbow_xy,left_wrist_xy,right_wrist_xy,left_hip_xy,
        right_hip_xy,left_knee_xy,right_knee_xy,left_ankle_xy,right_ankle_xy):
    cv2.circle(image, (left_shoulder_xy[0], left_shoulder_xy[1]), RADIUS, CLR_KP, THICKNESS)
```

次に骨格座標を円で描画し、それぞれの座標を線でつなぐ関数を作成する。引数には画像、リスト化した骨格座標データなどを設定する。また、RADIUS や CLR_KP などは描画のために必要な色や半径などを設定したパラメータが入る。こちらはメイン関数の冒頭で定義する。ここでは OpenCV の circle 関数を利用し円を座標上に描画する。引数には画像、各関節の xy 座標、円の半径や色線の太さを指定している。この処理も他に取得したい座標分だけ作成している。

```
    cv2.line(image,(left_shoulder_xy[0],left_shoulder_xy[1]),(left_elbow_xy[0],left_elbow_xy[1]),
    CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)

return image
```

また、同じ関数内に座標をつなぐ処理も追加する。OpenCV の line 関数を利用する。引数には画像、始点と終点の xy 座標、線の色、太さを指定する。また返り値として、読み込んだ画像を指定する。

```
if __name__ == "__main__":   …①
    count = 0
    THRESH_SLOPE = 0
    THRESH_DiST_SPINE = 30
    THRESH_ARM = 40

    RADIUS = 5
    THICKNESS = 2
    CLR_KP = (0,0,255)
    CLR_LINE = (255,255,255)

    dataPOS=pd.DataFrame()
    mp_pose=mp.solutions.pose
    cap_file=cv2.VideoCapture('C0324.MP4')
```

ここではメイン関数を作成し、大枠を作る。Python のメイン関数は①のよう記述する。

またメイン関数冒頭で描画の際に必要な線の長さ、半径、色の設定を追加する。

dataPOS には pandas のデータフレームを用いて 2 次元データに対応させる。

5.1.2　Pandas を用いた座標データの書き出し

```
            if not results.pose_landmarks:
                print('not results')
            else:
                data,left_shoulder_xy,right_shoulder_xy,left_elbow_xy,right_elbow_xy,
                left_wrist_xy,right_wrist_xy,left_hip_xy,right_hip_xy,left_knee_xy,right_knee_xy,
                left_ankle_xy,right_ankle_xy = get_keypoint(results, height, width)
                rows = pd.Series(data)
                dataPOS=dataPOS.append(rows,ignore_index=True)

            image=draw_keypoint(image,RADIUS,CLR_KP,CLR_LINE,THICKNESS,
            left_shoulder_xy,right_shoulder_xy,left_elbow_xy,right_elbow_xy,left_wrist_xy,
            right_wrist_xy,left_hip_xy,right_hip_xy,left_knee_xy,right_knee_xy,left_ankle_xy,
            right_ankle_xy)
            #対応は MP4 のみ
            cv2.imshow('AI personal trainer', image)
            if cv2.waitKey(5) & 0xFF == 27:
              dataPOS.columns=['L_shoulderX','L_shoulderY','R_shoulderX','R_shoulderY',
              'L_elbowX','L_elbowY','R_elbowX','R_elbowY','L_wristX','L_wristY','R_wristX',
              'R_wristY','L_hipX','L_hipY','R_hipX','R_hipY','L_kneeX','L_kneeY','R_kneeX',
              'R_kneeY','L_ankleX','L_ankleY','R_ankleX','R_ankleY']
              dataPOS.to_excel("P:¥¥卒業研究¥¥body_pos17_data.xlsx")
              break
cap_file.release()
cv2.destroyAllWindows()
```

画面から骨格座標データが取得できた場合、各座標の関数に取得したデータを格納していく。一度これらの座標データを rows に 1 次元データとして格納し、dataPOS（2 次元データ）の配列に追加する。この際に ignore_index で配列時の行番号を無視することができる。

waitkey の処理の際に dataPOS.columns に列名を格納している。また、dataPOS.to_excel では書き出した骨格座標データを書き出すパスを指定する。

5.1.3　書き出した骨格座標データ

書き出した座標データが以下のようになっている。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1		L_shoulderX	L_shoulderY	R_shoulderX	R_shoulderY	L_elbowX	L_elbowY	R_elbowX	R_elbowY	L_wristX	L_wristY	R_wristX	R_wristY	L_hipX	L_hipY	R_hipX	R_hipY	L_kneeX	L_kneeY	R_kneeX	R_kneeY	L_ankleX	L_ankleY	R_ankleX	R_ankleY
2	0	534	574	469	580	567	489	500	480	624	488	583	498	541	387	494	385	554	260	516	256	530	137	473	141
3	1	534	574	470	580	567	489	497	483	628	488	582	497	542	387	495	384	555	260	516	255	529	137	473	139
4	2	536	572	470	582	568	487	493	483	630	486	581	494	543	386	497	382	555	260	516	255	527	137	476	136
5	3	540	571	471	582	570	485	492	483	631	484	581	486	544	386	497	382	555	259	516	254	527	137	483	131
6	4	543	570	471	582	572	484	491	482	636	482	581	483	545	385	497	382	555	257	516	254	527	137	492	126
7	5	544	570	472	583	572	484	488	483	635	481	580	482	546	385	498	382	555	257	517	254	527	136	497	124
8	6	544	569	474	583	572	483	488	483	636	478	579	480	546	384	499	380	555	257	520	253	527	136	501	123
9	7	544	569	478	583	569	482	488	483	633	471	576	473	546	383	500	380	555	257	522	253	528	136	507	123
10	8	545	569	480	583	568	482	487	483	628	468	573	463	545	383	501	380	555	257	523	253	529	136	513	122
11	9	546	569	481	584	567	484	485	483	625	465	569	453	545	385	501	381	555	258	525	255	530	135	518	121
12	10	548	569	481	584	568	484	483	482	628	462	567	441	545	385	501	381	557	258	526	255	531	134	521	121
13	11	548	569	483	584	568	484	482	481	629	459	564	429	545	385	502	381	557	258	527	254	531	134	522	121
14	12	548	569	485	583	568	484	481	477	629	459	556	414	545	384	503	380	557	258	527	254	532	134	524	122
15	13	551	569	485	583	568	484	480	475	628	456	547	407	546	384	503	380	558	258	527	253	532	135	524	122
16	14	552	569	485	583	568	484	478	472	627	453	540	393	546	384	503	380	559	258	527	253	532	135	523	122
17	15	555	569	485	583	568	484	474	471	626	451	525	387	548	384	503	380	559	258	526	252	533	136	523	122
18	16	556	569	485	584	568	483	472	471	626	451	512	379	549	384	503	380	561	258	524	251	533	136	523	122
19	17	556	569	485	584	568	483	471	472	624	451	503	377	551	383	505	379	561	257	522	249	533	136	523	122
20	18	556	569	486	584	568	482	469	472	628	452	496	373	552	379	509	376	561	256	522	248	533	136	522	123
21	19	556	569	486	583	568	482	467	472	626	450	491	372	553	377	511	374	560	253	521	246	533	137	522	123
22	20	556	570	486	582	569	482	465	470	626	449	487	370	555	377	512	372	561	252	519	245	534	137	522	123
23	21	556	570	486	581	568	482	462	470	621	446	480	371	555	376	513	372	561	252	518	245	535	137	522	123
24	22	556	571	486	581	566	482	462	469	619	446	472	369	556	377	513	372	561	252	519	245	535	137	521	123
25	23	556	571	486	580	563	482	460	469	612	430	466	368	556	377	513	372	561	252	518	245	536	137	521	123
26	24	556	571	486	580	562	482	459	469	614	435	462	371	556	377	513	372	561	252	518	246	536	137	521	123
27	25	556	571	486	579	562	482	457	469	609	428	459	371	555	377	512	372	561	252	518	246	537	137	521	123
28	26	556	570	486	579	561	482	455	469	610	425	457	371	556	377	512	372	563	252	518	246	537	136	521	123
29	27	556	570	486	579	560	482	454	469	611	427	456	371	556	377	512	372	564	253	518	247	538	136	521	123
30	28	556	570	485	578	559	482	452	469	606	425	455	371	556	377	512	372	565	253	518	247	538	136	521	122
31	29	556	570	484	578	560	482	451	468	606	422	456	371	557	378	512	372	566	252	518	247	538	135	521	122
32	30	556	570	484	578	560	481	451	467	604	425	457	370	557	378	512	372	569	252	518	247	539	135	520	122
33	31	556	569	483	577	560	481	450	467	603	420	458	368	558	378	511	372	571	252	518	247	540	135	520	122
34	32	556	569	483	577	560	481	450	466	603	425	459	369	558	377	511	372	575	253	518	247	540	135	519	122
35	33	556	568	482	577	560	480	449	466	604	428	460	369	558	377	510	372	578	254	517	247	541	136	518	123
36	34	556	568	482	577	560	480	450	466	607	433	460	368	559	377	510	371	582	255	517	247	543	137	518	123
37	35	555	568	482	577	560	480	450	466	610	438	462	368	559	376	510	370	587	257	514	244	544	137	518	123
38	36	555	568	481	576	560	480	450	464	614	442	466	367	560	375	510	368	590	258	514	243	544	137	518	123
39	37	554	567	480	575	560	480	450	464	612	441	470	367	562	375	510	367	593	260	511	243	546	138	515	123
40	38	553	567	480	575	559	480	451	463	609	441	474	367	563	375	511	367	595	261	510	243	551	139	514	123
41	39	553	568	479	575	559	481	451	463	608	442	478	367	564	374	511	366	596	261	510	243	555	139	514	122
42	40	553	567	478	575	560	480	453	463	608	442	481	367	565	374	511	365	595	261	509	242	560	138	514	121

図 10 取得した座標データ

ここではおよそ 40 行程度書き出したが本来はまだデータが存在する。1 回のサーブの動きは、150～200 フレーム程度になる。また取得している関節の量も多くデータ全体の量も莫大な量になっている。このデータは画面が縦横 540px になっているため、それよりも小さい値をとっている。

第 6 章　可視化の課題と分析方法の検討

6.1 骨格データ可視化における課題

(1)　正確な距離や身長の推定

カメラからの距離によって、画面内に表示される人の大きさが異なってしまっている。また、身長によってはカメラからの距離を調節する必要があるため、場合によっては身長 160 cm の人と 180 ㎝の人が同じ身長として判定されてしまうといった問題がある。そのため、正確な身長や移動距離の推定を行うことができないという現状になっている。

(2)　座標のベクトル

スポーツをする際には歩く、走るといった動きを伴うことが多い。しかし、現状の骨格データの取得は 2 次元として行っているため、一定の方向への座標変化になっており、 x 座標のみが増加または減少するという状況になっている。

また、利き手によって撮影の際にカメラを変更する場合、x 座標の変化に違いが生じてしまう。

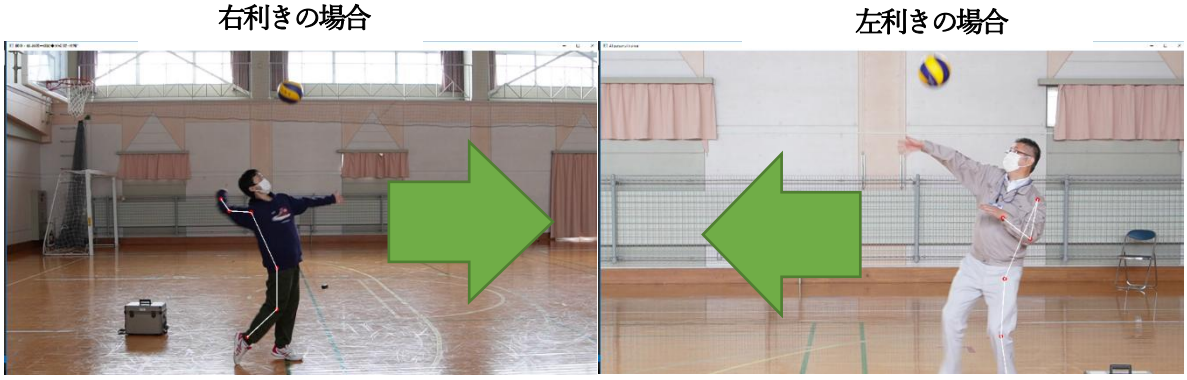


図 11 利き手による撮影方向の違い

(3) フレームごとの変化による比較

5 章で作成したプログラムで書き出した骨格座標データを用いて、エクセル上でフレームごとの座標変化を折れ線グラフとして表示してみた。できたグラフが以下のようになる。(図 10)

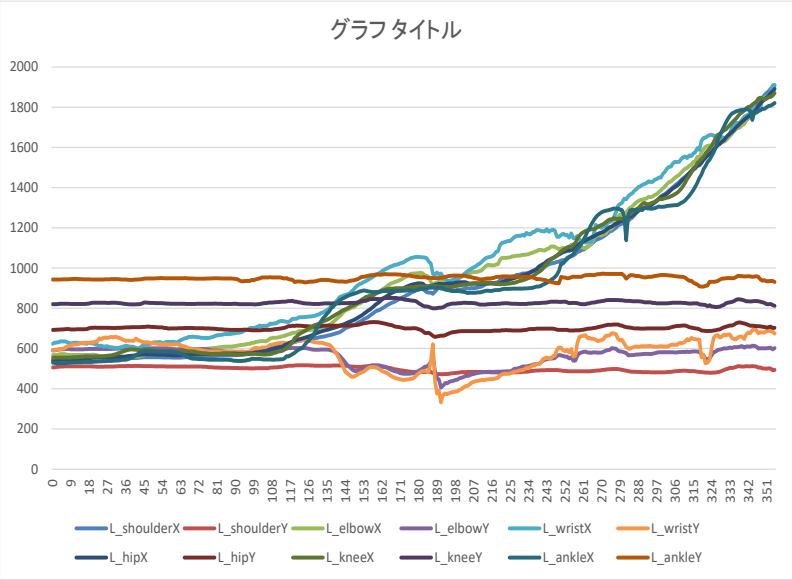


図 12 フレームごとの座標変化

グラフから x 座標のみが右肩上がりに上昇し、y 座標はほとんど変化していないことが分かる。

また、y 座標でも大きく変化しているのは、手首や肘といった、サーブを打つ際に大きく動く部位のみになっている。

このようにどの部分が変化しているのかはグラフから読み取ることができるが、どのタイミングでどのように変化しているのかをグラフから読み取ることは難しくなっている。

そこで、より変化が分かりやすいグラフを作成したいと考えた。

6.2 分析方法の検討

(1) 変化のわかりやすいグラフの検討

フレームごとの座標変化のグラフを作成した際に、変化の様子が分かりにくいという問題があった。走る、歩くといった動作を伴う場合に座標変化が一定に増加、または減少してしまうといった問題もある。

これにより歩くなどの移動動作が伴う場合に x,y 座標どちらも動いてしまうため、基準にできる点が必要になると考えた。そこで基準点を一つに定め、もう一点の動きをグラフ化することで座標変化が分かりやすくなるのではないかと考えた。

(2) グラフ化する関節の選定

次に基準とする関節と、可視化する関節ついて考えた。グラフ化する関節はできるだけサーブに関連するパーツにするべきだと考えた。また、座標変化が大きい部分の方が人による違いが出やすいのではないかとも考えた。そこでフレームごとの座標変化をグラフ化した際に y 座標の変化が大きかった肘と手首の座標をグラフ化することにした。基準点を肘として、手首の座標がどのように移動しているのかを相対的に取得し、グラフ化を行う。

(3) 計算方法

同じフレーム数の時の肘の位置を基準とした手首の相対的な位置を取得する。座標の位置を計算する際には 2 点間の距離を利用する。(図 11)これを用いて手首の移動量算出する。

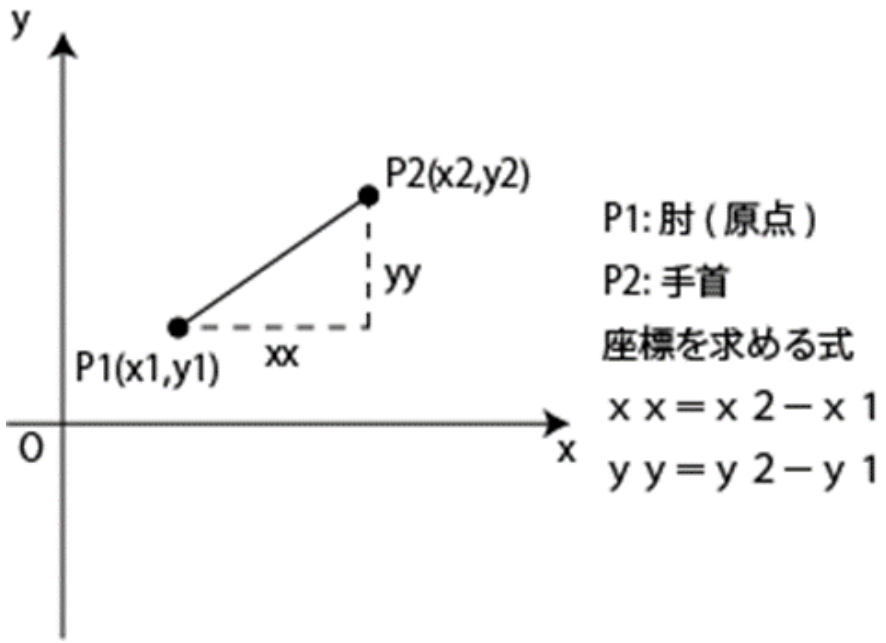


図 13　2 点間の距離の算出

p1 を肘、p2 を手首として座標データを算出する。それぞれの関節データはフレームごとに p1(x1,y1),p2(x2,y2)といったデータが格納される。そこから x 座標、y 座標それぞれの基準点からの距離を算出する。それを求める式が xx,yy の式になっている。

これらで集めたデータを用いて散布図として表示することで座標の移動を算出できると考えた。

6.3 Matplotlib を用いたグラフの作成

リスト 4. 骨格推定からグラフ化まで行うプログラム(right_wrist_trainer.py)

```
＃モジュールのインポート
import mediapipe as mp
import matplotlib.pyplot as plt
import cv2
import numpy as np
import pandas as pd
import csv

#骨格座標取得処理
def get_keypoint(results,height,width):
    right_shoulder_x = int(results.pose_landmarks.landmark[12].x * width)
    right_shoulder_y = int(results.pose_landmarks.landmark[12].y * height)
    shoulder_y =  height-right_shoulder_y
    right_shoulder_xy = [right_shoulder_x,right_shoulder_y]



    right_elbow_x = int(results.pose_landmarks.landmark[14].x * width)
    right_elbow_y = int(results.pose_landmarks.landmark[14].y * height)
    elbow_y =  height-right_elbow_y
    right_elbow_xy = [right_elbow_x,right_elbow_y]

    right_wrist_x = int(results.pose_landmarks.landmark[16].x * width)
    right_wrist_y = int(results.pose_landmarks.landmark[16].y * height)
    wrist_y = height-right_wrist_y
    right_wrist_xy = [right_wrist_x,right_wrist_y]

    right_hip_x = int(results.pose_landmarks.landmark[24].x * width)
    right_hip_y = int(results.pose_landmarks.landmark[24].y * height)
    hip_y = height - right_hip_y
    right_hip_xy = [right_hip_x,right_hip_y]
```

```
    right_knee_x = int(results.pose_landmarks.landmark[26].x * width)
    right_knee_y = int(results.pose_landmarks.landmark[26].y * height)
    knee_y = height - right_knee_y
    right_knee_xy = [right_knee_x,right_knee_y]

    right_ankle_x = int(results.pose_landmarks.landmark[28].x * width)
    right_ankle_y = int(results.pose_landmarks.landmark[28].y * height)
    ankle_y = height - right_ankle_y
    right_ankle_xy = [right_ankle_x,right_ankle_y]

    data=[right_shoulder_x,shoulder_y,right_elbow_x,elbow_y,right_wrist_x,wrist_y,
        right_hip_x,hip_y,right_knee_x,knee_y,right_ankle_x,ankle_y]
    return   data,right_shoulder_xy,right_elbow_xy,right_wrist_xy,right_hip_xy,
        right_knee_xy,right_ankle_xy

#キーポイント取得処理
def draw_keypoint(image,RADIUS, CLR_KP,CLR_LINE,THICKNESS,right_shoulder_xy,
right_elbow_xy,right_wrist_xy,right_hip_xy,right_knee_xy,right_ankle_xy):
    cv2.circle(image,(right_shoulder_xy[0],right_shoulder_xy[1]),RADIUS,CLR_KP,THICKNESS)
    cv2.circle(image, (right_elbow_xy[0], right_elbow_xy[1]), RADIUS, CLR_KP, THICKNESS)
    cv2.circle(image, (right_wrist_xy[0], right_wrist_xy[1]), RADIUS, CLR_KP, THICKNESS)
    cv2.circle(image, (right_hip_xy[0], right_hip_xy[1]), RADIUS, CLR_KP, THICKNESS)
    cv2.circle(image, (right_knee_xy[0], right_knee_xy[1]), RADIUS, CLR_KP, THICKNESS)
    cv2.circle(image, (right_ankle_xy[0], right_ankle_xy[1]), RADIUS, CLR_KP, THICKNESS)

    cv2.line(image,(right_shoulder_xy[0],right_shoulder_xy[1]),(right_elbow_xy[0],
        right_elbow_xy[1]),CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_elbow_xy[0],right_elbow_xy[1]),(right_wrist_xy[0],right_wrist_xy[1]),
        CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_shoulder_xy[0],right_shoulder_xy[1]),(right_hip_xy[0],right_hip_xy[1]),
        CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_hip_xy[0],right_hip_xy[1]),(right_knee_xy[0],right_knee_xy[1]),
        CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
    cv2.line(image,(right_knee_xy[0],right_knee_xy[1]),(right_ankle_xy[0],right_ankle_xy[1]),
        CLR_LINE,THICKNESS,lineType=cv2.LINE_8,shift=0)
```

```
    return image

if __name__ == "__main__":
    THRESH_SLOPE = 0
    THRESH_DiST_SPINE = 30
    THRESH_ARM = 40

    RADIUS = 5
    THICKNESS = 2
    CLR_KP = (0,0,255)
    CLR_LINE = (255,255,255)

    dataPOS=pd.DataFrame()
    mp_pose=mp.solutions.pose
    cap_file=cv2.VideoCapture('C0356.mp4')

#骨格検出など細かい設定
    with mp_pose.Pose(
        min_detection_confidence=0.5,
        static_image_mode=False)as pose_detection:

        while cap_file.isOpened:
            success,image = cap_file.read()
            if not success:
                print("empty camera frame")
                break
            image = cv2.resize(image,dsize=None,fx=1.0,fy=1.0)
            rgb_image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
            height = rgb_image.shape[0]
            width = rgb_image.shape[1]

            results = pose_detection.process(rgb_image)

            if not results.pose_landmarks:
```

```
            print('not results')
        else:
            data,right_shoulder_xy,right_elbow_xy,right_wrist_xy,right_hip_xy,
            right_knee_xy,right_ankle_xy= get_keypoint(results,height,width)
            rows = pd.Series(data)
            dataPOS=dataPOS.append(rows,ignore_index=True)

        image=draw_keypoint(image,RADIUS,CLR_KP,CLR_LINE,THICKNESS,
            right_shoulder_xy,right_elbow_xy,right_wrist_xy,right_hip_xy,
            right_knee_xy,right_ankle_xy)

        #対応は MP4 のみ
        cv2.imshow('肘を基準とした手首の動き', image)
        if cv2.waitKey(5) & 0xFF == 27:
            dataPOS.columns=['R_shoulderX','R_shoulderY','R_elbowX','R_elbowY',
            'R_wristX','R_wristY','R_hipX','R_hipY','R_kneeX','R_kneeY',
            'R_ankleX','R_ankleY']
            dataPOS.to_csv("P:¥¥卒業研究¥¥right_No17.1_data.csv")
            break

i = 0
xx = []
yy = []
x1 = []
y1 = []
x2 = []
y2 = []
MyPath = "P:¥¥卒業研究¥¥right_No17.1_data.csv"
with open(MyPath) as f:
    reader = csv.reader(f)

    for row in reader:
        x1.append(str(row[3]))
        y1.append(str(row[4]))
        x2.append(str(row[5]))
```

```
            y2.append(str(row[6]))

            if(i != 0):
                x = int(x2[i]) - int(x1[i])
                xx.append(x)
                y = int(y2[i]) - int(y1[i])
                yy.append(y)
            i=i+1
plt.scatter(xx,yy)
plt.title("肘を基準とした手首の動き",fontname="MS Gothic")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.xlim(-50, 120)
plt.ylim(-110, 110)
plt.show()
cap_file.release()
cv2.destroyAllWindows()
```

6.3.1　プログラムの解説

```
＃モジュールのインポート
import mediapipe as mp
import matplotlib.pyplot as plt
import cv2
import numpy as np
import pandas as pd
import csv
```

これまでと同様にモジュールのインポート処理を行う。今回はグラフの描画に必要な matplotlib と csv ファイルへの書き出しを行うため csv を追加でインポートしている。

これまでと同じように骨格座標データを取得し、リスト化する処理も書く。今回は骨格データを減らし右半身の骨格だけを取得するようにしている。同様に骨格描画処理、メイン関数での座標描画の細かい設定も書き込んでおく。

```
#動画の中断と書き込み処理
if cv2.waitKey(5) & 0xFF == 27:
        dataPOS.columns=['R_shoulderX','R_shoulderY','R_elbowX','R_elbowY',
        'R_wristX','R_wristY','R_hipX','R_hipY','R_kneeX','R_kneeY',
        'R_ankleX','R_ankleY']
        dataPOS.to_csv("P:¥¥卒業研究¥¥right_No17.1_data.csv")
        break
```

また、中断処理もほとんど同じである。今回は、書き込んだ csv ファイルから肘と手首だけのデータを抽出して配列に格納を行う。そのために、csv ファイルへの書き込みになっている。

```
    i = 0
# 座標データを格納する配列
    xx = []
    yy = []
    x1 = []
    y1 = []
    x2 = []
    y2 = []
    MyPath = "P:¥¥卒業研究¥¥right_No17.1_data.csv"
```

中断処理の後に処理に使う配列や初期値の設定を行う。x1、y1 には肘の座標、x2、y2 には手首の座標を格納する。xx、yy には計算されたデータが格納される。MyPath には、csv ファイルが置かれているパスを設定しておく。

```
#csv ファイルの読み込み
with open(MyPath) as f:
        reader = csv.reader(f)

#配列に読み取った値を入力する。
        for row in reader:
            x1.append(str(row[3]))
            y1.append(str(row[4]))
            x2.append(str(row[5]))
            y2.append(str(row[6]))
```

open でファイルの読み込みを行う。読み込むファイルは reader に格納する。

その後、csv ファイルの各座標データを先ほど作ったか配列に格納していく。この際に for 文で列の取得を繰り返し行い、取得できなくなるまで繰り返すようにしている。

```
                #座標の計算と計算したデータの格納
            if(i != 0):
                x = int(x2[i]) - int(x1[i])
                xx.append(x)
                y = int(y2[i]) - int(y1[i])
                yy.append(y)
             i=i+1
```

同じフレームの時の座標の計算は以上のようになる。i は配列の列数を指す。1 列目(配列は 0 番目)は関節名を設定しているので、計算には用いないように if 文で設定する。計算したデータは x 座標なら xx、y 座標なら yy を格納する。また座標の格納が終了したら次の配列に進むように i に 1 を足していく。

```
#グラフ描画の処理
plt.scatter(xx,yy)
#タイトルとラベル
plt.title("肘を基準とした手首の動き",fontname="MS Gothic")
plt.xlabel("x")
plt.ylabel("y")
グリッドの線や軸の範囲の指定
plt.grid(True)
plt.xlim(-50, 120)
plt.ylim(-110, 110)
#グラフの書き出し
plt.show()

cap_file.release()
cv2.destroyAllWindows()
```

plt.scatter で散布図の描画ができる。先ほど計算した xx と yy を用いて散布図を作成する。タイトルも設定することができるが日本語の場合は fontname で設定しなければ文字化けが起こってしまうため注意が必要。今回は x 軸、y 軸ともに範囲を指定しているが、指定しなくてもグラフ作成の際に自動的に設定してくれる仕様になっている。今回は比較のために軸の範囲を設定している。最後に show でグラフの書き出しを行う。

書き出したグラフと描画処理は以下のようになる。(図 10、11)

図 14　右半身の骨格検出の様子

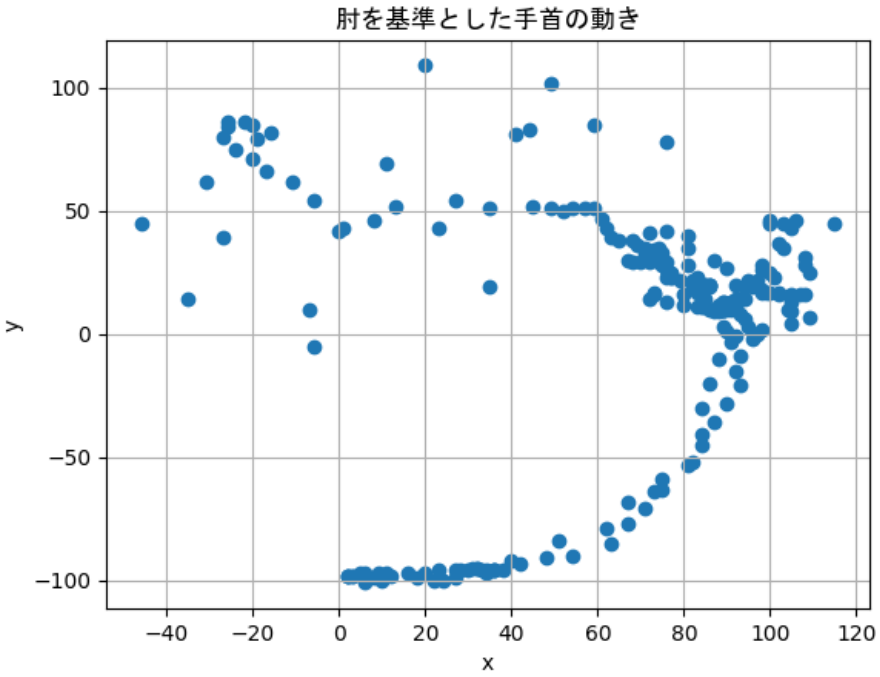


図 15　取得したグラフの様子

第 7 章　実験及び考察

7.1 実験方法

データの分析を行うため複数人のサーブの動きを撮影した。今回は自分を含め 7 人のデータを集めることにした。サーブの成功と失敗は以下のようになる。

今回はサーブを打つ際のカメラからの距離などはほとんど指定せず成功失敗のみのデータを取得する。また、プログラムでは成功、失敗の判定を行うことができないので動画を撮影した際に成功失敗を確認する。今回の成功とはサーブが相手コートに入っていることを意味する。

表 1 複数人によるサーブのデータ

	私(右)	A（右)	B(右)	C(右)	D(右)	E（左)	F(右)
成功(回)	1	1	0	1	3	1	3
失敗（回)	1	1	3	2	0	2	0

また以下の項目において比較を行う

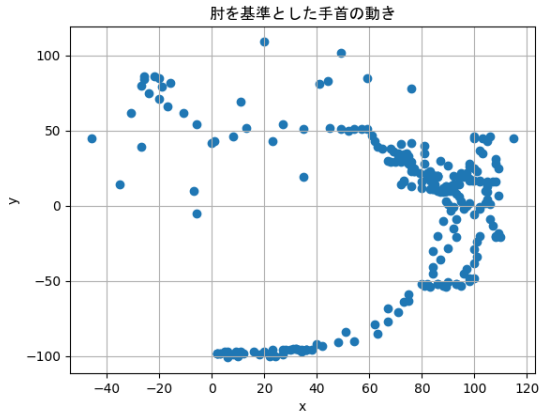
- 成功した人のデータと失敗した人のデータの比較
- 同一人物で成功した場合と失敗した場合の比較

7.2 実験の結果と考察

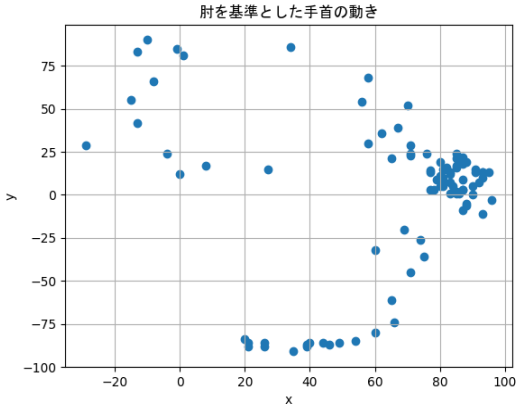
7.2.1　実行の結果

(1)　私の場合

成功時

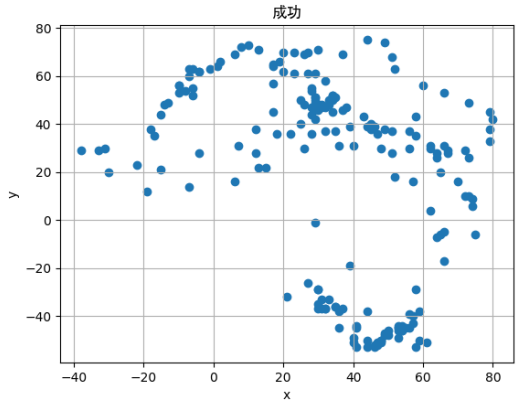


失敗時

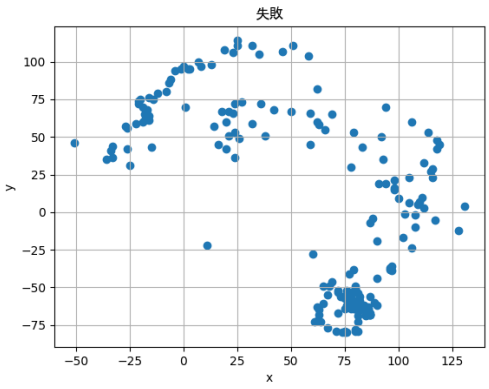


(2)　A さんの場合

成功時

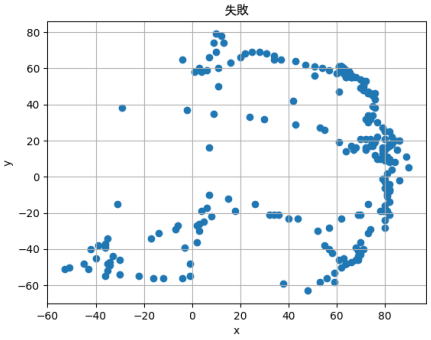


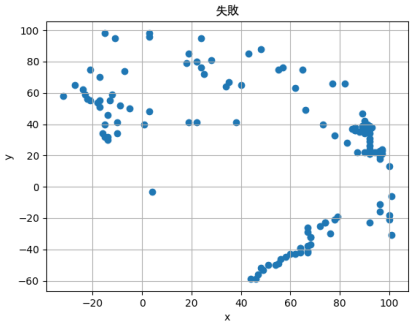
失敗時

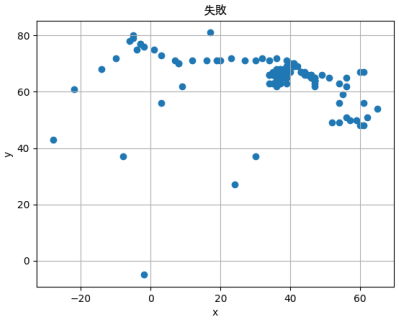


(3) B さんの場合

すべて失敗

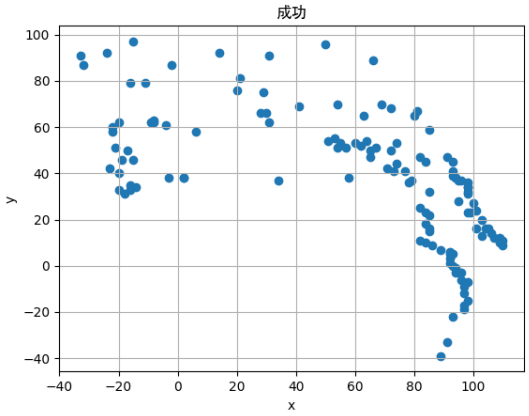




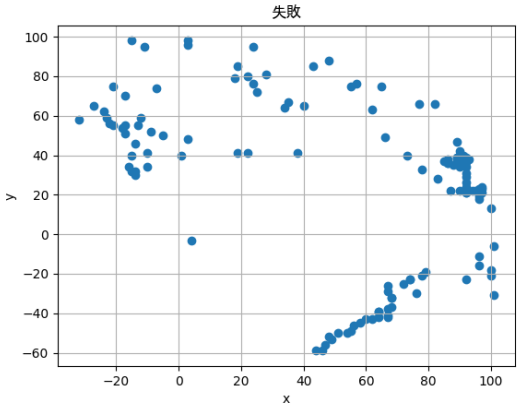


(4) c さんの場合

成功時



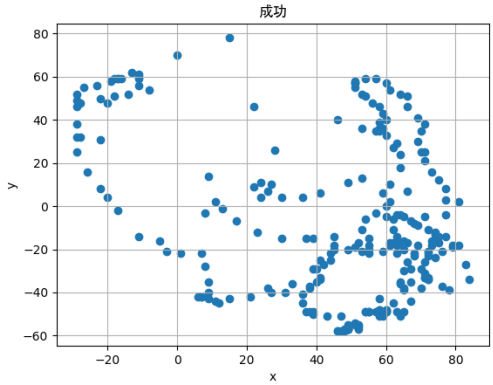
失敗時



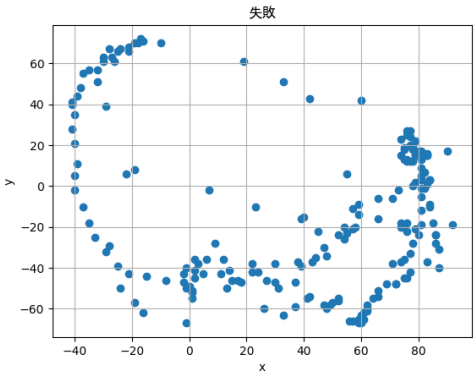
(5) D さんの場合

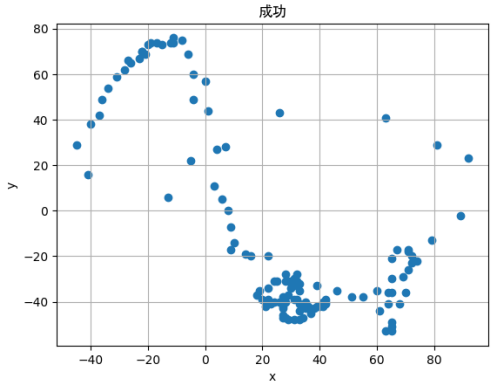
D さんはサーブの動作の際にジャンプをしている

成功時



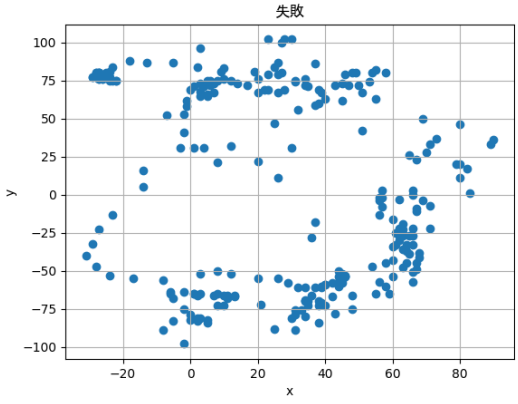
失敗時

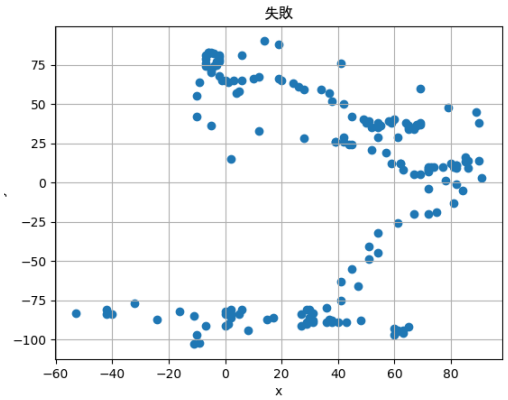




(6) E さんの場合

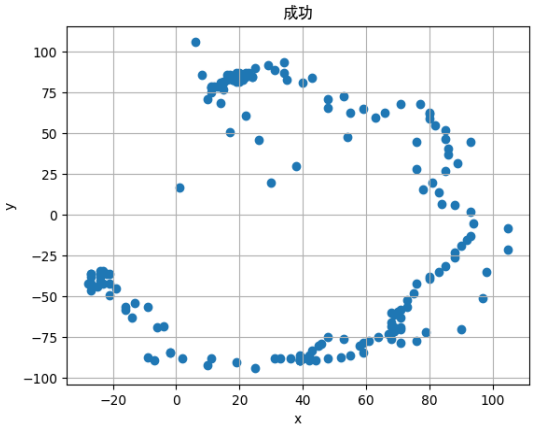
すべて失敗

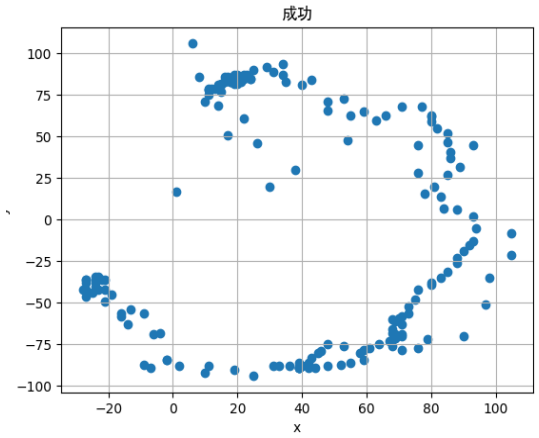


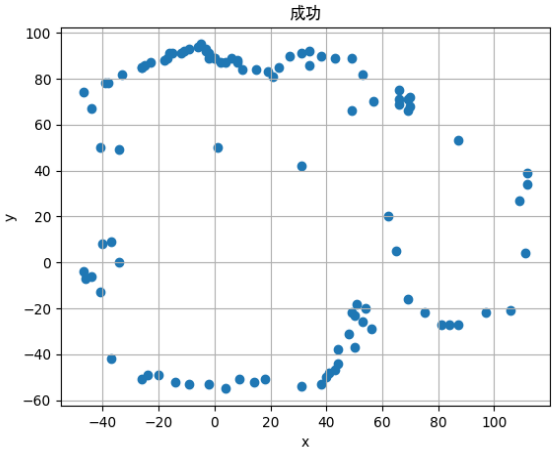


(7) F さんの場合

すべて成功







7.2.2　考察と分析

(1)　同一人物のサーブの比較

ここでは成功失敗の両方のデータがある人（私、A、C）に着目して比較を行った。

- 私の場合

成功時は腕が半楕円を描くように動く。失敗時はｙ座標最高値が若干低い。これはミート（ボールを撃つ）の際に腕が下がっていることが考えられる。

- A さんの場合

A さんの場合は成功時のグラフの目盛を見ると分布範囲が狭くなっている。逆に失敗時は分布範囲が広がっている。この点から考えると、身体に近い部分でサーブを打つ方が力が加わっている、または失敗時は体からボールが離れすぎたため、力をボールに伝えきれていないことが考えられる。

- C さんの場合

C さんの場合はグラフのｘ座標の分布範囲は成功、失敗でほとんど変わらない。失敗のｙ座標の最小座標が成功時よりも低い値をとっている。そのため、サーブの際に腕の振りが遅れているか、サーブトスが低くなってうまくミートできていないことが考えられる。

- 例外:D さん

ジャンプの動きを伴うため成功、失敗ともにデータが安定しなかった。また、ジャンプなどの激しい動きを伴う場合に骨格座標の取得が困難になるという課題がある。

(2) サーブの結果に着目した比較

ここではサーブの成功と失敗に着目して比較を行う。

- 成功時

  成功の際ではグラフが楕円を描くように分布している。また、連続で成功する人は分布範囲が安定しており、最小値、最大値に着目してもほぼ同じ値をとっているため、サーブの動きが固定されており、成功率が高まっていると考えられる。

- 失敗時

  失敗の際にはグラフの形が安定しなかった。また、グラフの座標分布が上下左右のどこかによってしまうと、腕がきちんと振れておらず、ボールに力がうまく伝えられていないことが考えられる。また、データの拡散がみられた場合は腕がサーブトスに合わせようとぶれてしまってボールにうまく力が加えられなかったことが考えられる。

7.3 課題

今後分析をさらに発展させるには以下の課題を解決する必要がある。

- 画面からの距離、身長の正確な推定、Z 座標取得
- 他の関節同士の関係性の比較
- ジャンプなどを伴う動きの座標取得

第8章　終わりに

本研究では python の環境構築からプログラム作成、分析までを行ってきた。本研究を通して OpenCV や Mediapipe などの使い方を理解し,landmark の取得および座標の取得,可視化などに利用することができた.

Mediapipe を用いて,身体の動きに合わせて landmark を取得するモジュールについての理解と,開発環境が必要になった.また、取得した landmark を csv、Excel ファイルに書き出す pandas やそのデータをグラフ化する Matplotlib といったモジュールがあらかじめ Python に定義されていたため、分析自体は進めやすかったと感じている。

スポーツと情報分野を組み合わせた研究は情報科では行われていなかったため,先駆けての研究

だった.そのためカメラからの距離によって、座標データがおかしくなることや、グラフによる可視化の際に変化がつかみづらくなるといった想定外の課題も多く見つかり,プログラムの機能が少なくなってしまった.しかし,骨格推定や身体の動きの可視化などの成果もあったので,今後の研究の基盤となれたなら良いと思う.

第 9 章　参考文献

- MediaPipe・OpenCV・Python で体験する AI パーソナルトレーナーアプリ開発 Kindle 版

https://amzn.asia/d/92pernL

- Home | mediapipe - Google

https://google.github.io/mediapipe/

- 【OpenCV】できることや特徴をわかりやすく解説

https://www.sejuku.net/blog/113292

- Python の拡張モジュール「NumPy」とは？インストール方法や基本的な使い方を紹介！

https://udemy.benesse.co.jp/data-science/ai/python-numpy.html

- Pandas DataFrame を徹底解説！(作成、行・列の追加と削除、index など)

https://ai-inter1.com/pandas-dataframe_basic/

- 【Python】CSV を読み込み配列へ格納：基本テクニックご紹介

https://kirinote.com/python-csv-array/

- Python でグラフ描画する方法を解説。Matplotlib を使えば簡単！

https://udemy.benesse.co.jp/data-science/data-analysis/python-graph.html