令和 4 年度

長岡工業高等専門学校　電子制御工学科

卒業論文

選手の 3 次元位置を追跡する
バレーボール分析支援システム

長岡工業高等専門学校

電子制御工学科

制御工学研究室

佐野　裕馬

指導教員　外山　茂浩

目次

第 1 章
序論

1.1　スポーツ指導の現状

指導者は、効果的な指導に向けて、自分自身のこれまでの実践、経験に頼るだけでなく、指導の内容や方法に関して、大学や研究機関等での科学的な研究により理論づけられたもの、研究の結果や数値などで科学的根拠のえられたもの、新たに開発されたものなど、スポーツ医・科学の研究の成果を積極的に習得し、指導において活用することが重要である[1]。

近年、選手の動きをデータ化するにあたって、様々なアプローチが行われてきた。この「データ」には、球技におけるボールの位置、選手の位置、選手の姿勢などが含まれるが、本研究ではバレーボールの試合中における選手の位置に特に注目する。

1.2　バレーボールにおける分析支援システム

1.2.1　Data Volley

バレーボールの解析に用いるツールとして、Data Volley[2]を紹介する。これはイタリアの Data Project 社が開発したバレーボール専用のソフトウェアである。Data Volley は世界中で使われており、国際大会に出場するナショナルチームのほとんどが利用している[3]。また、国内であっても、プロチームや全日本バレーボール大学男女選手権大会に参加する大学の多くがこのソフトウェアを活用している[4]。

図 1.1 に Data Volley の GUI を示す。これはアナリストと呼ばれる分析専門家やコーチが用いるソフトウェアであり、各選手のポジションをあらかじめ登録し、コマンドで選手がボールに触れた際の行動を、試合を見ながらリアルタイムで入力する。それに基づき、プレ

イの決定率の確認や、選手の位置をあとから見直すことが可能である。

図 1.1　Data Volley の GUI

しかし、Data Volley には次に示すような欠点が挙げられる[5]。

①記録される位置の精度の低さ

②複雑な入力コマンド

③瞬間的なプレイの判断の要求

④人的入力ミス

1 つ目は「記録される位置の精度の低さ」である。Data Volley では、コートの片面を 6×6 に 36 等分にして扱う。図 1.2 にコート分割の様子を示す。バレーボールコートの片面は 9m 四方であるため、Data Volley で記録される位置の最小単位は 1.5m 四方ということになる。また Data Volley では、ボールに触れていない選手の位置は入力しないため、ボールに触れた瞬間でしか選手の位置を見直せないということも同様に欠点である。

2 つ目は「複雑な入力コマンド」である。例として「7SQ65.4#」というコマンドがあるが、このコマンドは「7 番がゾーン 6 から打ったジャンプサーブを相手 4 番がゾーン 5 で完璧に返球した」という意味である。このような複雑なコマンド入力の方法を覚えるだけでなく、選手のプレイを見ながら入力を行う必要があるため、ある程度の練習が必要である。

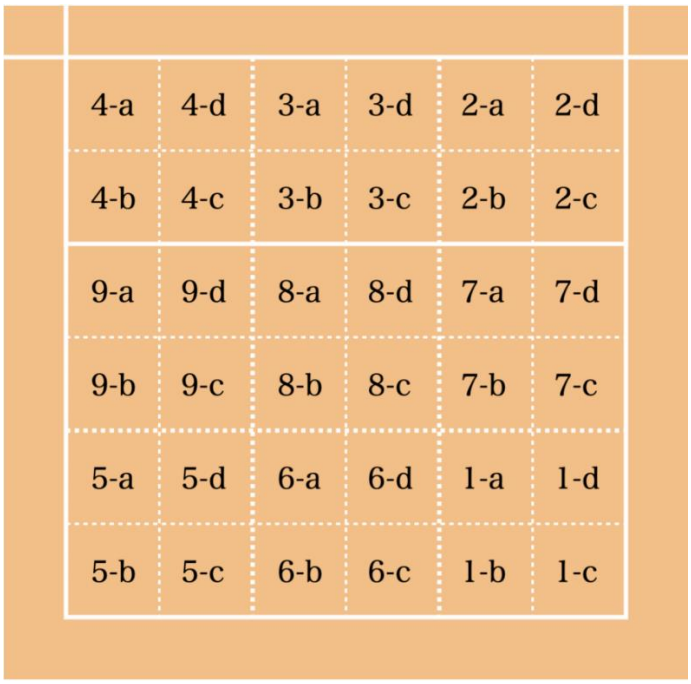

図 1.2　Data Volley におけるコートの分割

3 つ目は「瞬間的なプレイの判断の要求」である。Data Volley はバレーボールの試合中のプレイを分析するため、入力を行うユーザーにはバレーボールの知識が要求される。さらに、入力にはプレイの種類だけではなく、ユーザーから見たプレイの評価も含まれる。そのため、バレーボールのルールを知っているだけでなく、プレイの評価もできることがユーザーに求められる。これは先述した欠点である「複雑な入力コマンド」に重ねて、Data Volley の使用の難易度を高めている。これによって、Data Volley のユーザーをアナリストやコーチに限定してしまっている。

4 つ目に「人的入力ミス」である。これは、ユーザーがコマンドを入力する際に起こすミスのことである。例えば、背番号の 4 と 5 や、レシーブの一種を表す D と返球の一種を表す F は、最も一般的な QWERTY 配列のキーボード上では隣り合っているが、これらが間違って入力された場合は大きく意味が異なってしまう。

1.2.2　平田（2021）の先行研究

平田は、1 台のカメラを用いて撮影したバレーボールの試合映像から、自動で選手のコート平面位置を推定するシステムを開発した[5]。システムによって得られる推定結果の例を図 1.3 に示す。これにより、Data Volley で挙げた欠点である「記録される位置の精度の低さ」、「瞬間的なプレイの判断」、「人的入力ミス」の解決に取り組んだ。

しかし先行研究では、選手が床上にいるという前提で選手位置の推定が行われている。そのため、選手が床上から離れてしまう跳躍時には、映像と比較しても全く異なった位置を推定してしまう。バレーボール競技は、選手が頻繁に跳躍を行うスポーツであるため、跳躍時の選手位置を正しく出来ないことは、重大な欠点である。

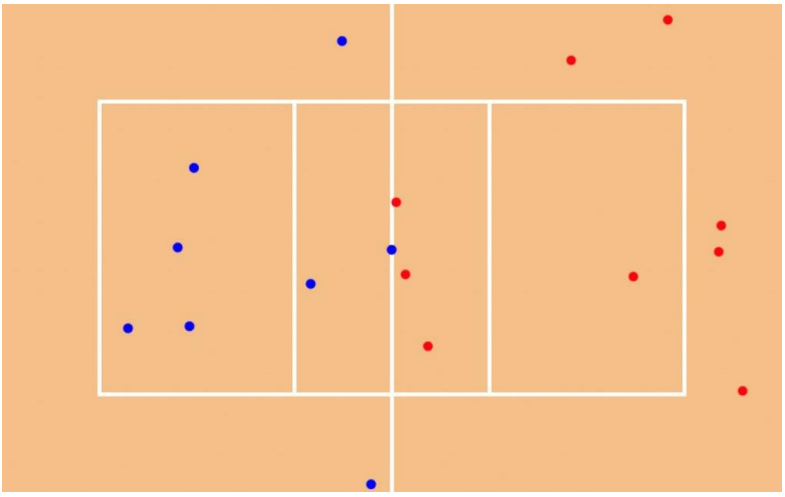

図 1.3　先行研究によって得られる選手位置推定結果の例[5]

1.3　本研究の内容

そこで本研究は、先行研究[5]の発展研究と位置づけ、バレーボールの試合映像から選手の 3 次元位置を求めることにより、先行研究の課題であった跳躍時の推定を正しく行うことを目的とする。また、「(できる限り）人の手を介さない」、さらに「扱いやすい」システムを目指す。

ただし、先行研究とは異なり、1 台のカメラでの撮影ではなく、複数台のカメラで試合を

同時撮影することにより、3 次元の選手位置を推定する。また、選手位置に関して、先行研究では両足の中間点としていたが、選手の重心位置を考慮して、本研究では選手の腰の位置を選手位置と設定した。

本論文の構成として、第 1 章では本研究の背景及び目的を述べた。第 2 章から第 5 章では、選手の 3 次元位置を推定するために必要な手順を述べる。そして、第 6 章では本研究の結果として、それぞれの手順において得られる途中の結果、最終的に得られる選手位置の推定結果、そしてそれらの結果に対する考察を述べる。最後に、第 7 章に本研究のまとめと今後の課題及び発展について述べる。

また、付録 A では本研究で作成したプログラムのソースコードを紹介している。

第 2 章
カメラキャリブレーション

2.1　カメラパラメータ

選手の位置を 3 次元的に表すためには、実空間上に 3 次元座標系を設定し、時間同期させた複数台のカメラで同時撮影した後、撮影した映像から 3 次元空間を再構築する。3 次元空間を再構築するためには、実空間における物体がどのようにカメラ画面に映るかを知る必要があり、そのためにはカメラパラメータと呼ばれる情報が必要である。

カメラパラメータは、カメラの焦点距離と光学的中心の情報を含むカメラ内部パラメータと、カメラの位置と姿勢の情報を含むカメラ外部パラメータに分けられ、これらを求めることをカメラキャリブレーションと呼ぶ。

2.2　カメラ内部パラメータの推定

カメラ内部パラメータは、キャリブレーションパターンを様々な画角から撮影することによって推定する。キャリブレーションパターンとは、既知のサイズの円や正方形が並んでいるパターンのことである。図 2.1 にチェッカーボードと呼ばれるキャリブレーションパターンの一種を示す。本研究では、チェッカーボードを利用してカメラ内部パラメータを推定した。

カメラ内部パラメータの推定について、本研究では OpenCV[6]の calibrateCamera 関数を用いた。この関数は複数の点について、実空間上での 3 次元位置と、その点の映像上での画像位置の対応を撮影した画像の枚数分、引数として渡すことでカメラ内部パラメータであるカメラ行列の推定を行うことができる。関数のアルゴリズムについて以下に示す。

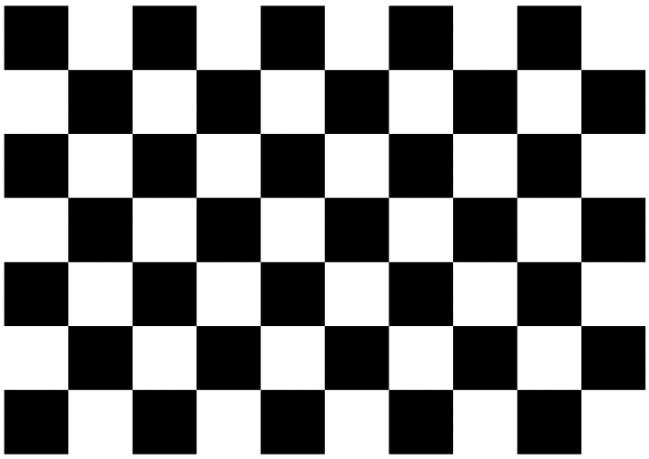

図 2.1　チェッカーボード

①まず、カメラ内部パラメータ（カメラ行列）の初期値を設定する。また、カメラの歪み係数を全て 0 に設定する。歪み係数についても、0 でない値を設定することができる。

②カメラ内部パラメータが既知であるかのようにカメラの初期姿勢を推定する。

③再投影誤差を最小にするように、大域的な Levenberg-Marquardt 法を実行する。再投影誤差は、全ての点に対して、計算によって求められる画像上の位置と実際の画像上の位置の距離の二乗和である。

この関数を実行することによって、カメラ内部パラメータと撮影画像分のカメラ外部パラメータ、そしてカメラの歪み係数を推定することができる。ここで得られるカメラ外部パラメータは、チェッカーボード平面に X,Y 軸、そしてそれらに垂直な Z 軸で構成される実空間座標系であるため、選手位置を求めるためには使用しない。

カメラ内部パラメータの推定を行うと、先ほど示した通り、カメラの歪み係数も同時に推定することができる。OpenCV では、「半径方向歪み」と「円周方向歪み」の二つに関して実装が成されている[7]。

「半径方向歪み」とはレンズの形状に起因し、レンズの中心から離れた場所を通過する光は中心付近を通過する光よりも大きく曲げられることによる歪みである。この補正パラメータはテイラー級数で表されるが、最初の 3 項程度を考えると十分な場合が多い。

「円周方向歪み」は、撮像素子がピンホールの平面に対して平行になっていないことに起因する。この補正パラメータは 2 つの変数で表現することができる。

これらの歪み係数を用いると、図 2.2 と図 2.3 に示すように画像の歪みを除去することができる。

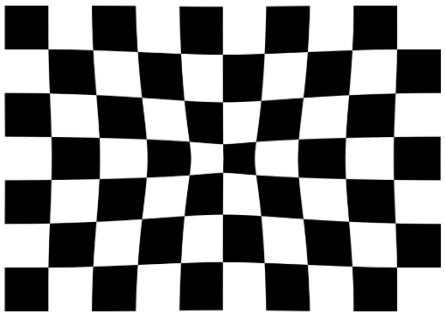

図 2.2　歪み除去前の画像

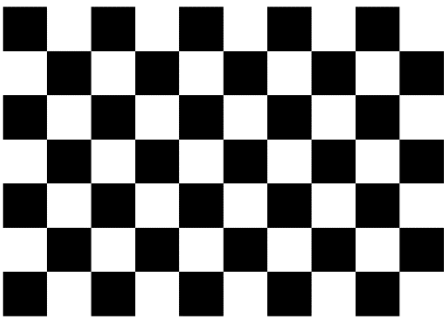

図 2.3　歪み除去後の画像

2.3　カメラ外部パラメータの推定

カメラ外部パラメータは、バレーボールコートの既知点と、その映像上での画像位置を対応付けることによって推定する。中井らの研究[8]では、バレーボールコートの既知点として 14 点が紹介されており、本研究でも同様の既知点を用いた。用いたコートの既知点を図 2.4 に示す。本研究では、実空間座標系を図 2.4 のように設定した。

カメラ外部パラメータの推定について、本研究では OpenCV の solvePnP 関数を用いた。この関数には、calibrateCamera 関数と同様に、実空間上の物体の位置と、それらの映像上での画像位置の対応を渡す。また、2.2 節で推定したカメラ行列と歪み係数も渡すことによって、カメラ外部パラメータを推定する。

得られる外部パラメータは、カメラ位置を示す並進ベクトルとカメラ姿勢を示す回転ベ

クトルである。しかし、ここで得られるパラメータはカメラ外部パラメータを示しているが、直接的にカメラ位置や姿勢を表していない。そのため、以下のように変換を行う。

まず、カメラ姿勢を示す回転ベクトルに対して、OpenCV の Rodrigues 関数を用いて、3 行 1 列の回転ベクトルを、3 行 3 列の回転行列に変換する。そして、その回転行列の転置を取ることによって、カメラ座標系から実空間座標系へと変換する行列となる。

続いて、カメラ位置を示す並進ベクトルに対して、カメラ座標系から実空間座標系へと変換する行列との積を取ることによって、実空間座標系でのカメラ位置を示すベクトルとなる。

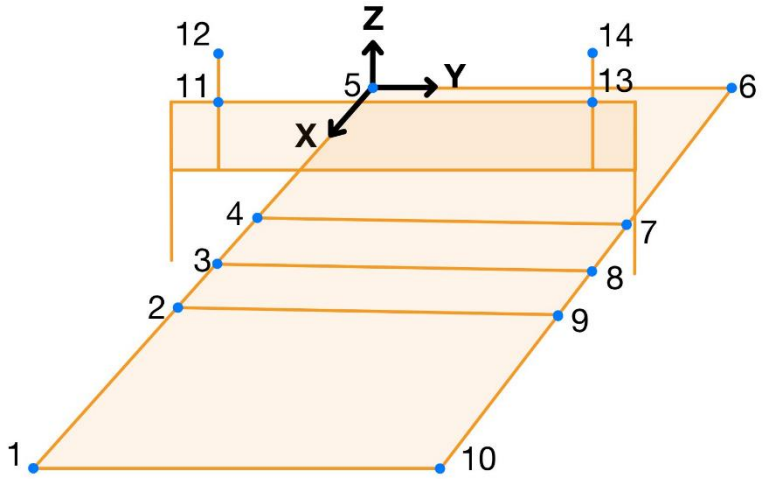

図 2.4　バレーボールコートの既知点 14 点

第 3 章
選手に対する姿勢推定と追跡

3.1　姿勢推定

従来の姿勢推定に関する研究は，身体にマーカーを取り付けて撮影を行うモーションキャプチャシステムの利用が大半で合った。しかし近年では、マーカーを使用せずに、カメラで撮影した映像から、人工知能技術を用いて姿勢を推定する技術が確立されている。本研究では、その例として OpenPose[9][10][11][12]と、本研究で使用した AlphaPose[14][15][16][17]を紹介する。

3.1.1　OpenPose

OpenPose とは、複数人のキーポイントを同時に推定することが出来る姿勢推定アルゴリズムであり、姿勢推定アルゴリズムの中で最も有名なアルゴリズムの 1 つである。

推定できるキーポイントは、モデルによって異なるが、例として COCO のキーポイントを図 3.1 に示す。

OpenPose では、姿勢推定を行う際に、「ボトムアップ型」の姿勢推定モデルを用いている。ボトムアップ型とは、画像中のキーポイントを全て抽出した後、人物ごとにそれらをマッチングさせて繋ぎ合わせていく方法である。ボトムアップ型と異なるモデルとして「トップダウン型」があるが、その手法と比べて計算量を抑えられるという利点がある。しかし、一人に対して姿勢推定を行っているわけではないため、部位間の繋ぎ合わせの精度が低く、異なる人物の部位を繋いでしまうという欠点もある。

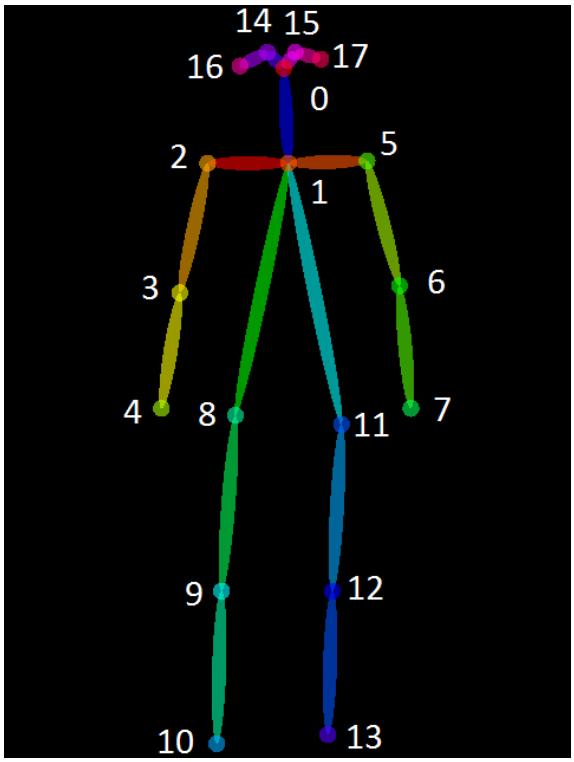

図 3.1　COCO のキーポイント[13]

3.1.2　AlphaPose

AlphaPose とは、上海交通大学の研究チームが公開している人物追跡機能も持つ高精度姿勢推定ライブラリである。OpenPose と比較した AlphaPose の特徴を以下に示す。

①トップダウン型の姿勢推定モデル

②人物追跡機能

1 つ目は「トップダウン型の姿勢推定モデル」である。トップダウン型とは、画像に映る人物を検出した後、それぞれの人物に対して一人用の姿勢推定を行う方法である。トップダウン型は、人物に比例して計算量が増加してしまうという欠点があるが、ボトムアップ型と比べて推定精度が高いという利点がある。

2 つ目は「人物追跡機能」である。AlphaPose の人物検出には YOLO（You Only Look Once）が用いられている。YOLO では検出した人物に ID を振り分けて追跡することがで

き、AlphaPose では、その機能を用いている。

本研究では、5 章における処理の関係上、精度の高い姿勢推定が求められる。また、4 章において映像間の選手を対応付ける際、ID が選手に振られていることで、映像に映る選手の指定が簡単になる。以上の理由から本研究では OpenPose ではなく、AlphaPose を用いて姿勢推定を行った。

AlphaPose は OpenPose と同様に、推定できるキーポイントは様々であるが、本研究では図 3.1 に示した COCO のキーポイントを用いた。

3.2　選手位置の設定

平田の先行研究[5]では、選手位置をその選手の両足の中間位置に設定した。しかし、選手のレシーブ時での体の重心を考えた際、腰の位置が選手位置としてより適切だと考えた。そのため、本研究では推定する選手の位置をその選手の腰の位置に設定した。

図 3.1 にて COCO のキーポイントを示したが、腰の位置を直接得られないため、腰の位置は部位 8 と部位 11 の中点を計算することによって得ることにした。

第 4 章
映像間の選手の対応付け

4.1　手動による対応付け

AlphaPose では、映像に映る人物に ID を振り分けて追跡することができる。しかし、映像を独立に処理するため、同じ選手に対してカメラごとに異なる ID を振り分けてしまう。図 4.1、図 4.2 にその様子を示す。

図 4.1　カメラ 1 での ID 振り分けの様子

図 4.2　カメラ 2 での ID 振り分けの様子

この後、第 5 章における「選手の 3 次元位置の推定」の際に、映像間の選手の対応が必要になる。処理された映像を見比べ、検出人数が多いあるフレームにおいて、目視によって、同じ選手を対応付ける。これを「手動による対応付け」と呼ぶ。

4.2　自動による対応付け

「手動による対応付け」とは異なる方法で、映像間の選手の対応付けを行うことを考える。

方法として、選手の腰を通る直線を各カメラで考えた際、カメラごとに選べる直線の組み合わせの中から距離が近い組を順に対応付ける。対応付けは、どちらかの映像で対応付けら

れる選手がいなくなるまで続ける。これを「自動による選手の対応付け」と呼ぶ。

既に対応を付けた選手に対しては、後から距離の近い組として現れた場合においても、その組を対応付けないことにより、選手の対応の重複を防ぐ。

また、直線同士が実際の選手位置とは異なる位置で最も近くなるケースも考えられるため、高さ方向に、選手が到達しないであろう領域を除くことで制限をかけている。

選手の腰を通る直線の定義方法と直線の距離の計算方法は、第 5 章で詳しく述べる。

第 5 章
選手の 3 次元位置の推定

5.1　理論

まず、選手の 3 次元位置推定の理論について説明する。ある選手の 3 次元位置を求めるには、カメラからその選手の腰を通る直線を複数本考え、それらの直線の最近点を選手位置とみなして求める。理論のイメージを図 5.1 として示す。

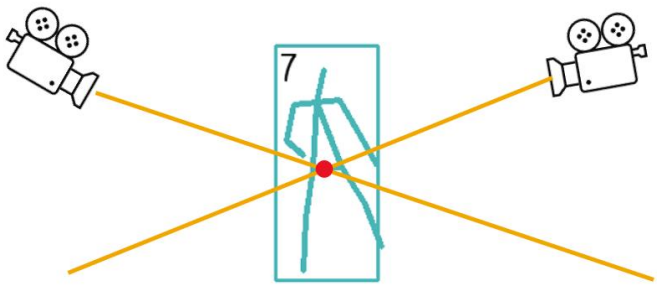

図 5.1　選手の 3 次元位置推定理論のイメージ

5.2　カメラ座標系における直線定義

5.1 節で説明した理論の通り、選手の腰を通る直線を定義する必要がある。まず、2.2 節で得られるカメラの焦点距離と、3.2 節で得られる選手の腰の画像座標を組み合わせることで、カメラから選手の腰に向かうベクトルをカメラ座標系で定義することができる。

カメラ座標系とは、実空間に存在するカメラの位置を原点として、注視する方向に z 軸、z 軸に垂直な面における右方向を x 軸、下方向を y 軸とした座標系である。カメラ座標系のイメージを図 5.2 に示す。

カメラから選手の腰に向かうベクトルに対して、媒介変数をかけることによって選手を通る直線をカメラ座標系で定義できる。

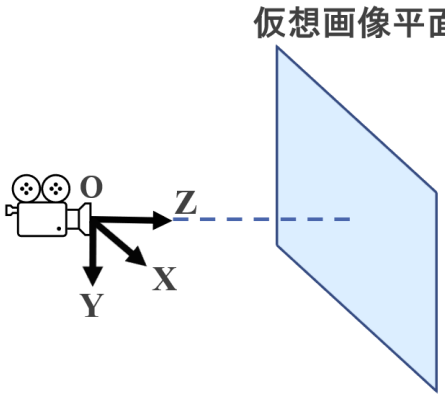

図 5.2 カメラ座標系

5.3　カメラ座標系から実空間座標系への変換

5.2 節において選手の腰を通る直線を定義できたが、カメラ座標系はカメラごとに存在する。そのため、映像ごとに直線の座標系が異なってしまい、同じ座標系で考えなければ直線の最近点を求めることができない。

そこで、2.3 節で得られたカメラ外部パラメータを用いて、カメラ座標系から実空間座標系へと変換する。カメラ外部パラメータより、カメラ座標系から実空間座標系へ回転させる行列を得ることができる。この行列とカメラ座標系のベクトルで積を取ることで、そのベクトルの向きを実空間座標系で表せるようになる。また、実空間座標系のカメラ位置も考慮して、その並進ベクトルを、先ほど変換したベクトルに足し合わせることで、カメラ座標系から実空間座標系への変換を行う。

5.4　反復による選手位置の算出

直線の最近点を求める方法として、反復による算出方法を説明する。直線はベクトルに媒介変数をかけて表現することができるため、媒介変数を任意の値に設定すれば、直線上のある一点の位置を表すことができる。

これを 2 直線 L1、L2 について考えて、得られる 2 点の距離を計算する。この距離は 2 つ

の媒介変数によって決まる。そこで、その距離を最小にする媒介変数の値を得ることで、例として、直線 L2 に最も近づく、直線 L1 上の点 k_1 が求まる。同様に逆の場合も考えて、点 k_2 を得る。そして点 k_1 と点 k_2 の中点を最近点 k として考えられる。直線の最近点の導出イメージを図 5.3 として示す。

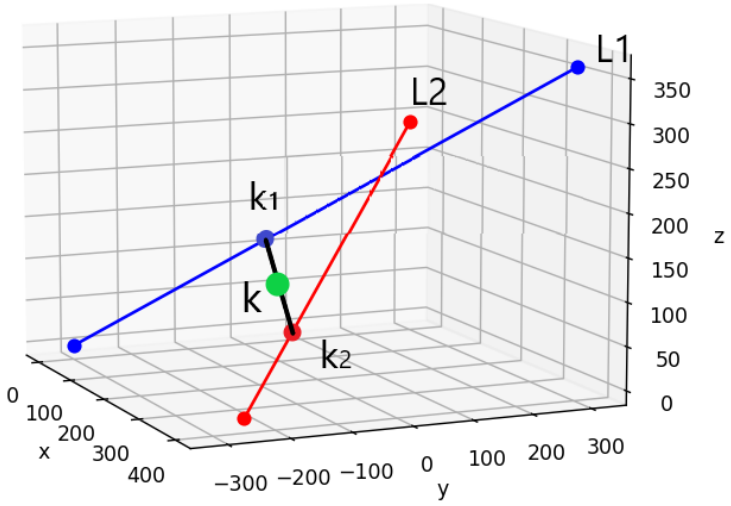

図 5.3　直線の最近点の導出イメージ

その媒介変数の値を得るために、SymPy[18]という数式処理を行うライブラリを用いた。SymPy を用いることで、距離を最小にする媒介変数を反復処理によって求められる。

5.5　解析的な選手位置の算出

5.4 節で、反復による選手位置の算出について説明したが、別の方法として、式を用いることで解析的に直線の最近点を算出することができる。

まず、5.4 節で説明した直線 L1 と直線 L2 を以下のように定義する。

直線 L1　　$s\boldsymbol{v}_1 + \boldsymbol{t}_1$

直線 L2　　$u\boldsymbol{v}_2 + \boldsymbol{t}_2$

s、u は媒介変数、$\boldsymbol{v}_1$、$\boldsymbol{v}_2$ は直線の単位方向ベクトル、$\boldsymbol{t}_1$、$\boldsymbol{t}_2$ は直線上のある一点である。

このように 2 本の直線を定義した場合、直線 L2 に最も近づく、直線 L1 上の点 $\boldsymbol{k_1}$ の位置は以下の式によって求められる。

$$\boldsymbol{k_1} = \boldsymbol{t_1} + \frac{\{(\boldsymbol{v_1} - (\boldsymbol{v_1} \cdot \boldsymbol{v_2})\boldsymbol{v_2}) \cdot (\boldsymbol{t_2} - \boldsymbol{t_1})\}}{1 - (\boldsymbol{v_1} \cdot \boldsymbol{v_2})^2}$$

同様に直線 L1 に最も近づく、直線 L2 上の点 $\boldsymbol{k_2}$ の位置は以下の式によって求められる。

$$\boldsymbol{k_2} = \boldsymbol{t_2} + \frac{\{(\boldsymbol{v_2} - (\boldsymbol{v_2} \cdot \boldsymbol{v_1})\boldsymbol{v_1}) \cdot (\boldsymbol{t_1} - \boldsymbol{t_2})\}}{1 - (\boldsymbol{v_2} \cdot \boldsymbol{v_1})^2}$$

このように点 $\boldsymbol{k_1}$ と点 $\boldsymbol{k_2}$ の位置を解析的に求め、5.4 節のようにそれらの中点 $\boldsymbol{k}$ を選手位置として推定する。

第 6 章
研究結果

6.1　カメラキャリブレーション

本研究では、試合の撮影には Apple 社の iPhone SE を 2 台使用し、それに搭載されるアウトカメラを用いた。iPhone SE に搭載されるアウトカメラのビデオ撮影技術仕様について表 6.1 に示す[19]。本研究では画質 1080p、フレームレート 30fps、光学倍率 1 倍で撮影した。

表 6.1　iPhone SE アウトカメラのビデオ撮影技術仕様

画質	フレームレート	その他特徴
4K	24, 25, 30, 60	最大3倍のデジタルズーム 光学的手振れ補正
1080p	25, 30, 60	
720p	30	

初めにカメラ内部パラメータの推定結果を示す。推定によって得られるカメラ内部パラメータを示すカメラ行列の整数部は以下のようになった。

$$\begin{pmatrix} 1794 & 0 & 933 \\ 0 & 1800 & 551 \\ 0 & 0 & 1 \end{pmatrix}$$

続いて、カメラ外部パラメータの推定結果を示す。本研究ではカメラキャリブレーションの評価を行うためにカメラ位置をあらかじめ測ったうえで撮影を行った。実際に測定した各座標値と推定によって得られたカメラ位置の各座標値を表 6.2 に示す。

表 6.2　カメラ位置の実測値と推定値の比較

	1台目のカメラ		2台目のカメラ	
	実測値	推定値	実測値	推定値
X[mm]	22400	22594	22530	22425
Y[mm]	12500	23960	-3640	-3659
Z[mm]	3890	4000	3890	3958

推定結果と測定結果を比較すると、その誤差は 1 台目のカメラで約 293mm、2 台目のカメラで約 127mm であった。

6.2　AlphaPose による選手の姿勢推定と追跡結果

撮影した映像の 1 フレーム目における処理結果を図 6.1 に示す。続いて、10 秒後におけるフレームの処理結果を図 6.2 に示す。また、映像に映る選手 12 人を A から M とし、各フレームにおける選手ごとに振り分けられる ID について表 6.3 に示す。

どちらのフレームにおいても、おおよそ正しく選手を検知し、その姿勢を推定できていることが図 6.1，6.2 から読み取れる。しかし、表 6.3 においてフレーム間の選手 ID を比較すると、同じ選手に対して異なる ID が振り分けられていることが分かる。

これは、選手の交差によって AlphaPose の追跡が途切れることによって、新たな ID を振り分けられることが原因と考えられる。

図 6.1　1 フレーム目における処理結果

図 6.2　10 秒後におけるフレームの処理結果

表 6.3　各フレームにおける選手 ID

選手	A	B	C	D	F	G
1フレーム目 振り分けID	13	11	1	2	3	5
10秒後のフレーム 振り分けID	13	11	1	2	未検知	5
選手	H	I	J	K	L	M
1フレーム目 振り分けID	9	10	12	4	6	21
10秒後のフレーム 振り分けID	10	50	26	20	6	12

6.3　選手位置の算出における処理時間

処理時間を計測したのは、AlphaPose による処理時間と、直線の最近点を求める処理の 2 つである。中でも、直線の最近点を求める処理については、5.4 節の反復による方法と 5.5 節の解析的な方法について比較を行った。

まず、AlphaPose の処理時間を確認した。AlphaPose は表 6.4 に示す実行環境で処理を行った。本研究の環境では、AlphaPose の処理時間は 10 秒の動画に対して 30 秒ほど処理時間を要した。

表 6.4　AlphaPose 実行環境

CPU	AMD Ryzen 3900X
GPU	NVIDIA GeForce RTX3090 Ti
メモリ	32 [GB]

続いて、直線の最近点を求める処理時間について、124 組の 2 直線の組について処理を行った結果を示す。

・反復算出　　24.38145 秒

・解析的算出　 0.01036 秒

処理時間を比較すると、5.5 節にて示した解析的な手法が計算量の面で優れていることが分かる。また、求めた直線の最近点位置についても差はほとんど見られなかった。

6.4　手動による選手対応付けを行った際の推定結果

本研究では、3 次元で得られる選手位置をコート平面に落として結果を示す。選手の手動での対応付けを行った場合の、図 6.1 における選手位置の推定結果を図 6.3 に示す。また、図 6.2 においても同様にその結果を図 6.4 に示す。

図 6.3 に示した推定結果と図 6.1 を比較すると、選手位置をおおかた推定できていることが分かる。また、跳躍中の推定結果についても確認すると、位置が大きくずれることなく推定できていることを確認できた。

図 6.3 において推定できていない選手については、どちらかの映像で人物検出できなかったことが原因と考えられる。また、図 6.3 と図 6.4 を比較すると、推定できている選手が極端に減っていることが読み取れる。これは、選手の交差などによって人物検出ができなくなることで、AlphaPose による追跡が途切れ、それ以降の選手の対応が取れなくなったことが原因だと考えられる。

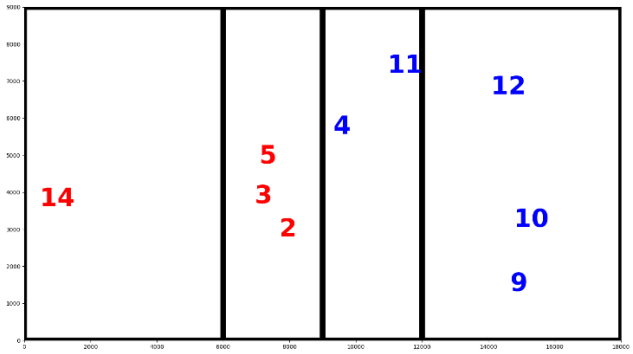

図 6.3　図 6.1 における選手位置の推定結果（手動）

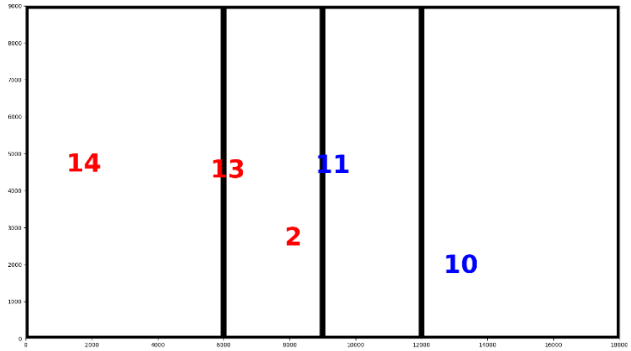

図 6.4　図 6.2 における選手位置の推定結果（手動）

6.5　自動による選手対応付けを行った際の推定結果

選手の自動での対応付けを行った場合の、図 6.1 における選手位置の推定結果を図 6.5 に示す。

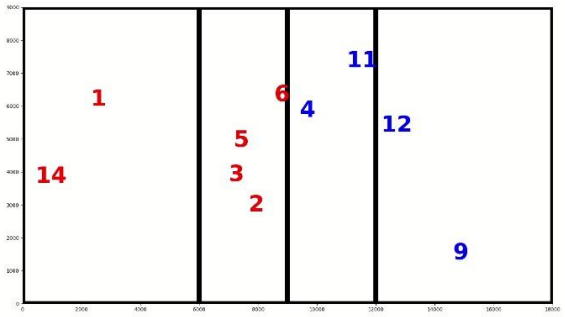

図 6.5　図 6.1 における選手位置の推定結果（自動）

図 6.3 に示した推定結果と比較すると、推定できている人数が多いことが読み取れる。しかし図 6.1 と見比べると、実際の位置とは異なる推定結果になっている選手が少数いることも分かる。これは映像間の選手の対応付けが正しく行われていないことを意味する。

この原因の 1 つとして、6.1 節で示したカメラキャリブレーションの誤差が考えられる。カメラ位置や姿勢に誤差を残したまま、選手を通る直線を考えたために、直線の距離の計算結果に誤差が累積したと考えられる。

また、もう 1 つの原因として、直線同士が実際の選手位置とは異なる位置で最も近くなる場合を、今回設定した制限では除けなかったことが考えられる。

第 7 章
結論

7.1　本研究のまとめ

本研究は、先行研究の発展研究と位置づけ、バレーボールの試合映像から選手の 3 次元位置を求めることにより、先行研究の課題であった跳躍時の推定を正しく行うことを目的とした。この課題に対して、本論文で示した第 2 章から第 5 章までの処理を行うことで選手の位置推定に取り組んだ。

結果として 3 次元位置を求めることで、映像間の選手の対応が取れた場合に限るが、選手が跳躍した場合においても位置をおおよそ正しく推定できた。

しかし、選手の対応付けの処理において、課題がいくつか見つかった。キャリブレーションの誤差、選手の交差によって選手の追跡が途切れること、そして直線同士が実際の選手位置とは異なる位置で最も近くなる場合を除けなかったことである。

7.2　今後の展望

最後に、本研究の展望を述べる。

本研究では選手位置を点として捉えたが、処理の過程で選手の姿勢を得られているため、今後の展望として、まずは選手の 3 次元姿勢まで得ることが考えられる。

また Data Volley には選手の行ったスパイクやレシーブといった動作の情報も管理できるため、3 次元の姿勢情報から動作を分類することができれば、開発しているシステムに大きく役立つと考えられる。

付録 A

ソースコード

以下に、本研究で作成した Python のソースコードを処理ごとにまとめて示す。

A.1　カメラキャリブレーション

リスト 1.1 からリスト 1.4 には、カメラキャリブレーションに必要な変数や関数をファイルごとに分けて記載している。リスト 1.5 はそれらをインポートし、カメラキャリブレーションを行うプログラムである。

リスト 1.1　utility.py

```
import csv
import numpy as np

# === 変更箇所 ===
calib_dir = "../utility/calib_images"
data_dir = "./data_camera2"

image_path = "./image/target_camera2.png"
point_path = "./input_data/point_camera2.csv"
OUTPUT_FILE = "{}/output.csv".format(data_dir)
# === ここまで ===

IMG_WIDTH = 1920
IMG_HEIGHT = 1080

netHeightDict = {
    "小学生" : 2000,
    "中学生男子" : 2300,
    "高校生男子" : 2400,
    "一般男子" : 2430,
    "中学生女子" : 2150,
    "高校生女子" : 2200,
    "一般女子" : 2240
}
netHeight = netHeightDict["一般男子"]

objectDict = [
    # コート原点から長辺を回る向き
    [0, 0, 0],
    [6000, 0, 0],
    [9000, 0, 0],
    [12000, 0, 0],
    [18000, 0, 0],
    [18000, 9000, 0],
    [12000, 9000, 0],
```

```
    [9000, 9000, 0],
    [6000, 9000, 0],
    [0, 9000, 0],
    # ネット原点側（下、上、アンテナ先）、逆側（下、上、アンテナ先）
    [9000, 0, netHeight - 1000],
    [9000, 0, netHeight],
    [9000, 0, netHeight + 800],
    [9000, 9000, netHeight - 1000],
    [9000, 9000, netHeight],
    [9000, 9000, netHeight + 800]
]

objectPoints = [
    objectDict[0],
    objectDict[1],
    objectDict[2],
    objectDict[3],
    objectDict[4],
    objectDict[5],
    objectDict[6],
    objectDict[7],
    objectDict[8],
    objectDict[11],
    objectDict[12],
    objectDict[14],
    objectDict[15]
]

def getCoordCSV(fileName):
    table = []
    file = open(fileName, "r")
    reader = csv.reader(file)
    for row in reader:
        table.append(list(map(float, row)))
    file.close()
    return table

def getObjectPointFromIndexList(indexList):
    retList = []
    for index in indexList:
        retList.append(objectDict[index])
    return retList

def writeCameraInfoCSV(cMat, dist, rotMat, transVec, dirVec, csvFileDir = data_dir):
    # 保存ファイル名
    cMatFile     = "{}/camera_mat.csv".format(csvFileDir)
    distFile     = "{}/dist.csv".format(csvFileDir)
    rotMatFile   = "{}/rotation_mat.csv".format(csvFileDir)
    transVecFile = "{}/trans_vec.csv".format(csvFileDir)
    dirVecFile   = "{}/dir_vec.csv".format(csvFileDir)
    # ファイル名とデータの対応付け
    dataDict     = {
        cMatFile     : cMat,
        distFile     : dist,
```

```
        rotMatFile   : rotMat,
        transVecFile : transVec,
        dirVecFile   : dirVec
    }
    # 保存
    for csvFileName in dataDict:
        with open(csvFileName, mode="w") as csvFile:
            np.savetxt(csvFile, dataDict[csvFileName], delimiter=',')

if __name__ == "__main__":
    print(getCoordCSV(OUTPUT_FILE))
```

リスト 1.2　plot_image.py

```
from matplotlib import pyplot as plt
from matplotlib import image as img
import cv2

from utility import *

# テキストファイルに座標を出力
def clickImage(event):
    if event.xdata != None and event.button == 3:
        output_file = open(OUTPUT_FILE, "a")
        print("button={}, x={}, y={}".format(event.button, event.xdata, event.ydata))
        output_file.write("{},{}¥n".format(event.xdata, event.ydata))
        output_file.close()

def getImagePoints(image, changeColor = False):
    # window : 画面 ID
    window = plt.figure(figsize = (16, 9))
    # windowに画像を表示
    if changeColor:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    else:
        plt.imshow(image)
    # クリック関数割り当て
    clickID = window.canvas.mpl_connect("button_press_event", clickImage)
    # 画面表示
    plt.show()
```

リスト 1.3　calibration.py

```
import cv2
import numpy as np
import glob

def getObjectPoints(square_size, nX, nY):
    object_points = np.zeros((nX * nY, 3), np.float32)
    object_points[:,:2] = np.mgrid[0:nY, 0:nX].T.reshape(-1, 2)
    # 正方形辺長の実寸をかけて座標を mm 単位に変更
    object_points = object_points * square_size
    return object_points
```

```
def getImagePoints(image_file, nX, nY):
    image = cv2.imread(image_file)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # チェスボードの内角検知
    success, corners = cv2.findChessboardCorners(gray, (nY, nX), None)
    if success == True:
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        corners_2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        return True, corners_2
    else:
        return False, []

def calibration(
  calib_dir, number_of_squares_X = 10, number_of_squares_Y = 7,
  square_size = 24.2, img_width = 1920, img_height = 1080
):
    image_files = glob.glob(f"{calib_dir}/*.png")

    nX = number_of_squares_X - 1
    nY = number_of_squares_Y - 1

    object_points = getObjectPoints(square_size, nX, nY)

    object_points_list = []
    image_points_list = []

    for image_file in image_files:
        ret, image_points = getImagePoints(image_file, nX, nY)
        if ret:
            object_points_list.append(object_points)
            image_points_list.append(image_points)

    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
    object_points_list, image_points_list, (img_width, img_height), None,None
  )

    optimal_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(
    mtx, dist, (img_width, img_height), 1, (img_width, img_height)
  )

    return mtx, optimal_camera_matrix, dist
```

リスト 1.4　external_parameter.py

```
import cv2
import numpy as np
import math
# カメラ外部パラメータを求める
# cameraMatrix : カメラ行列(fx, fy, cx, cy が入ってる)
# distCoef     : 歪みベクトル
# objectPoints : 実空間座標(3 次元)
# imagePoints  : 画像座標(2 次元)
def externalParameter(cameraMatrix, distCoef, points3D, points2D):
    ret, rvec, tvec = cv2.solvePnP(points3D, points2D, cameraMatrix, distCoef)
```

```
    # R_mat : 回転行列(world -> camera)
    # R_raw : 回転行列(camera -> world)
    R_mat, _ = cv2.Rodrigues(rvec)
    R_raw = R_mat.T
    # t_raw : カメラ位置ベクトル
    t_raw = -R_raw @ tvec
    t_raw = t_raw.reshape(1, 3)

    return R_raw, t_raw
```

リスト 1.5　main.py

```
import numpy as np
import cv2
import csv
import sys

import calibration
import plot_image
from external_parameter import *
from utility import *

imagePoints = []
pointIndexList = []

if __name__ == "__main__":
    # コマンドライン引数
    argv = sys.argv

    # カメラ内部パラメータ取得
    cameraMatrix, optimalCameraMatrix, dist = calibration.calibration(calib_dir)
    print("[完了] カメラ内部パラメータ取得")
    print("内部パラメータ¥n", cameraMatrix, "¥n")

    # 画像読み込み
    image = cv2.imread(image_path)
    print("[完了] 較正用画像読み込み")

    # 指定ポイント読み込み
    with open(point_path, "r") as pointFile:
        rows = csv.reader(pointFile)
        for row in rows:
            pointIndexList = list(map(int, row))

    # 画像座標書き込み
    if (len(argv) > 1) and (argv[1] == "click"):
        # 画像座標出力先ファイルの初期化
        output_file = open(OUTPUT_FILE, "w")
        output_file.close()

        # 画像クリックによる CSV 書き込み
        plot_image.getImagePoints(image, True)
        print("[完了] クリックによる画像座標書き込み")
```

```
    else:
        print("[SKIP] クリックによる画像座標書き込み")

    # 画像座標読み込み
    imagePoints = getCoordCSV(OUTPUT_FILE)
    print("[完了] 画像座標読み込み")

    # カメラ外部パラメータ取得
    objectPoints = np.array(
      getObjectPointFromIndexList(pointIndexList), dtype=np.float32
    )
    imagePoints = np.array(
      imagePoints, dtype=np.float32
    )

    camera2worldRotMat, cameraPosition = externalParameter(
      cameraMatrix, dist, objectPoints, imagePoints
    )
    cameraPosition = cameraPosition.reshape(-1)

    # カメラの単位方向ベクトル
    cameraDirection = np.dot(camera2worldRotMat, [0, 0, 1]) * (-1)

    print("[完了] カメラ外部パラメータ取得¥n")
    print("カメラ座標系→世界座標系の回転行列¥n", camera2worldRotMat, "¥n")
    print("世界座標系のカメラ位置¥n",            cameraPosition,     "¥n")
    print("世界座標系のカメラ方向ベクトル¥n",    cameraDirection,    "¥n")

    # CSV 書き込み
    writeCameraInfoCSV(
      cameraMatrix, dist, camera2worldRotMat, cameraPosition, cameraDirection
    )
```

A.2　選手の 3 次元位置推定

リスト 2.1 には、直線の最近点を求める関数を記載している。リスト 2.2 は、リスト 2.1 をインポートし、与えられた選手の画像座標から実空間の 3 次元位置を求めるプログラムである。

リスト 2.1　coord.py

```
import numpy as np

width  = 1920
height = 1080

def getHeadingVector(coord, cameraMat, rotationMat):
    u = coord[0]
    v = coord[1]
    # カメラ座標系での投影面上ポイント
    xC = u - width/2
    yC = v - height/2
    zC = (cameraMat[0, 0] + cameraMat[1, 1])/2 # x, y 焦点距離の中間値
```

```
    # カメラ座標系でのポイント位置ベクトル
    vecC = np.float32([xC, yC, zC])

    # 実空間座標系での方向ベクトルに変換
    vecW = np.dot(rotationMat, vecC) * (-1)

    # 正規化
    vecW /= np.linalg.norm(vecW)

    return vecW

def computeClosestPoint(t1, v1, t2, v2):
    v1 = v1 / np.linalg.norm(v1)
    v2 = v2 / np.linalg.norm(v2)

    part0 = v1 - (np.dot(v2, v1) * v2)
    part1 = t2 - t1
    # 分子
    numerator = np.dot(part0, part1)
    # 分母
    denominator = 1 - (np.dot(v2, v1))**2

    point = t1 + (numerator/denominator) * v1
    return point

# 2 直線の中点と最近点同士の誤差を返す(main 関数で実行)
def compute3dCoord(t1, v1, t2, v2):
    # 最近点
    p1 = computeClosestPoint(t1, v1, t2, v2)
    p2 = computeClosestPoint(t2, v2, t1, v1)

    p = (p1+p2)/2

    return p
```

リスト 2.2　main.py

```
import numpy as np
import cv2
import csv
import json
import glob
from coord import *

# camera1 - camera2 ID 対応 JSON
correspondFile = open("correspond.json", "r")

# data
c1Files = glob.glob("c1_json/*.json")
c2Files = glob.glob("c2_json/*.json")
jsonDataNum = min(len(c1Files), len(c2Files))
correspondData = json.load(correspondFile)
```

```
digit = len(str(jsonDataNum))

# camera1 data file
c1_dir      = "../01_camera_position/data_camera1"
cMat_c1     = np.loadtxt("{}/camera_mat.csv".format(c1_dir), delimiter=",")
dist_c1     = np.loadtxt("{}/dist.csv".format(c1_dir), delimiter=",")
rotMat_c1   = np.loadtxt("{}/rotation_mat.csv".format(c1_dir), delimiter=",")
transVec_c1 = np.loadtxt("{}/trans_vec.csv".format(c1_dir), delimiter=",")
dirVec_c1   = np.loadtxt("{}/dir_vec.csv".format(c1_dir), delimiter=",")

# camera2 data file
c2_dir      = "../01_camera_position/data_camera2"
cMat_c2     = np.loadtxt("{}/camera_mat.csv".format(c2_dir), delimiter=",")
dist_c2     = np.loadtxt("{}/dist.csv".format(c2_dir), delimiter=",")
rotMat_c2   = np.loadtxt("{}/rotation_mat.csv".format(c2_dir), delimiter=",")
transVec_c2 = np.loadtxt("{}/trans_vec.csv".format(c2_dir), delimiter=",")
dirVec_c2   = np.loadtxt("{}/dir_vec.csv".format(c2_dir), delimiter=",")

for i in range(jsonDataNum):
    oldJsonName = "{}.json".format(str(i).zfill(digit))

    c1File = open("c1_json/{}".format(oldJsonName), "r")
    c2File = open("c2_json/{}".format(oldJsonName), "r")

    c1Data = json.load(c1File)
    c2Data = json.load(c2File)

    coordData = {}

    for playerId in c1Data:
        if playerId in correspondData:
            correspondId = correspondData[playerId]
        else:
            continue

        if not correspondId in c2Data:
            continue

        c1Coord = np.array(c1Data[playerId],     dtype = np.float32)
        c2Coord = np.array(c2Data[correspondId], dtype = np.float32)

        vec_c1 = getHeadingVector(c1Coord, cMat_c1, rotMat_c1)
        vec_c2 = getHeadingVector(c2Coord, cMat_c2, rotMat_c2)

        worldCoord = compute3dCoord(transVec_c1, vec_c1, transVec_c2, vec_c2)

        coordData[playerId] = worldCoord.tolist()

    newJsonName = "{}.json".format(str(i).zfill(digit))

    with open("output/{}".format(newJsonName), "w") as f:
        json.dump(coordData, f, indent=4)
```

```
    print("write data in {}".format(newJsonName))

    c1File.close()
    c2File.close()

correspondFile.close()
```

A.3　推定結果のフレーム画像化

リスト 3.1 は、推定した選手の 3 次元座標のうち、x, y 座標の地点に選手 ID を描画した画像を生成するプログラムである。

リスト 3.1　main.py

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import os
import json
import glob

jsonList = glob.glob("input_data/*.json")
count = 0

digit = len(str(len(jsonList)))

fig = plt.figure(figsize=(19.2,10.8))
ax    = plt.axes()

court1 = patches.Rectangle(
    xy=(0, 0), width = 6000, height=9000, facecolor="white", edgecolor="black",
    fill=True, linewidth = 10
)
court2 = patches.Rectangle(
    xy=(6000, 0), width = 3000, height=9000, facecolor="white", edgecolor="black",
    fill=True, linewidth = 10
)
court3 = patches.Rectangle(
    xy=(9000, 0), width = 3000, height=9000, facecolor="white", edgecolor="black",
    fill=True, linewidth = 10
)
court4 = patches.Rectangle(
    xy=(12000, 0), width = 6000, height=9000, facecolor="white", edgecolor="black",
     fill=True, linewidth = 10
)

outDir = "output"
os.makedirs(outDir, exist_ok=True)

textsize = 45

for file in jsonList:
    plt.xlim(0, 18000)
```

```
    plt.ylim(0, 9000)

    ax.add_patch(court1)
    ax.add_patch(court2)
    ax.add_patch(court3)
    ax.add_patch(court4)

    with open(file) as f:
        data = json.load(f)
        for person in data:
            # 座標軸変更の影響で x を 9000 で反転
            x = 9000*2 - data[person][0]
            if data[person][0] > 9000:
                ax.text(
                    x, data[person][1], person, size=textsize,
                    horizontalalignment="center", verticalalignment="center",
                    color="red", fontweight="bold"
                )
            else:
                ax.text(
                    x, data[person][1], person, size=textsize,
                    horizontalalignment="center", verticalalignment="center",
                    color="blue", fontweight="bold"
                )

    figFileName = "output/sanple{}.png".format(str(count).zfill(digit))
    plt.savefig(figFileName)
    count += 1
    ax.clear()
```

A.4　フレーム画像の映像化

リスト 4.1 は、指定した複数枚の画像を結合し、30fps の映像に変換するプログラムである。

リスト 4.1　main.py

```
import sys
import cv2
import glob

# encoder(for mp4)
fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
# output file name, encoder, fps, size(fit to image size)
video = cv2.VideoWriter('output.mp4',fourcc, 30.0, (1920, 1080))

images = glob.glob("input/*.png")

if not video.isOpened():
    print("can't be opened")
    sys.exit()
```

```
for imgFile in images:
    print(imgFile)
    img = cv2.imread(imgFile)

    # can't read image, escape
    if img is None:
        print("can't read")
        break

    # add
    video.write(img)

video.release()
```

参考文献

[1] 運動部活動での指導のガイドライン，文部科学省，http://volleyball-yva.jp/data/mext_guidelines25.pdf，検索 2023.2.13

[2] Data Volley 4, https://www.dataproject.com/Products/EN/en/Volleyball/DataVolley4, 検索 2023.2.13

[3] “データバレーって何？どんなことができるの？”，2022.10.27，https://xn--cck2bb6bwak4kpe6g.com/datavolley/，検索 2023.2.13

[4] データバレー，Facebook，2020.12.4，https://www.facebook.com/DataVolley/photos/a.345358165530652/3640932032639899/?type=3，検索 2023.2.13

[5] 平田，“情報端末の内蔵カメラを用いた運動 再現システム”，R3 長岡高専電子制御工学科卒業論文

[6] OpenCV，https://opencv.org/，検索 2023.2.13

[7] “カメラキャリブレーションについて簡単なまとめ”，https://www.qoosky.io/techs/67a3d876c4，検索 2023.2.13

[8] 中井，村本，栗田，高根，瀧澤，塚本，河合，“バレーボールコート内の既知点を用いた 3 次元座標空間の再構築方法の精度とその特徴”，バレーボール研究 19 巻 1 号

[9] Z. {Cao} and G. {Hidalgo Martinez} and T. {Simon} and S. {Wei} and Y. A. {Sheikh}，” OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”，IEEE Transactions on Pattern Analysis and Machine Intelligence

[10] Tomas Simon and Hanbyul Joo and Iain Matthews and Yaser Sheikh，” Hand Keypoint Detection in Single Images using Multiview Bootstrapping”，CVPR

[11] Zhe Cao and Tomas Simon and Shih-En Wei and Yaser Sheikh，” Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”，CVPR

[12] Shih-En Wei and Varun Ramakrishna and Takeo Kanade and Yaser Sheikh，” Convolutional pose machines”，CVPR

[13] “OpenPose: OpenPose Doc - Output”，https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/md_doc_02_output.html，検索 2023.2.13

[14] Fang，Hao-Shu and Li，Jiefeng and Tang，Hongyang and Xu，Chao and Zhu，Haoyi and Xiu，Yuliang and Li，Yong-Lu and Lu，Cewu，“AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time”，IEEE Transactions on Pattern Analysis and Machine Intelligence

[15] Fang, Hao-Shu and Xie, Shuqin and Tai, Yu-Wing and Lu, Cewu，” {RMPE}: Regional Multi-person Pose Estimation”，ICCV

[16] Li, Jiefeng and Wang, Can and Zhu, Hao and Mao, Yihuan and Fang, Hao-Shu and Lu, Cewu，” Crowdpose: Efficient crowded scenes pose estimation and a new benchmark”,

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition p.3383 – 3393

[17] Xiu, Yuliang and Li, Jiefeng and Wang, Haoyu and Fang, Yinghong and Lu, Cewu, "{Pose Flow}: Efficient Online Pose Tracking", BMVC

[18] SymPy, https://www.sympy.org/en/index.html

[19] iPhone SE(第2世代) - 技術仕様(日本), https://support.apple.com/kb/SP820?locale=ja_JP, 検索 2023.2.13

謝辞

本研究を遂行するにあたり、終始適切な御助言を賜り、また丁寧に御指導をいただきました長岡工業高等専門学校電子制御工学科の外山茂浩教授、一般教育科の市川智之助教に深く感謝の意を表します。撮影に協力していただいた本校男子バレーボール部員に深く感謝の意を表します。また、ご多忙の中、日頃より有益な討論ならびに発表練習等への御助言を頂きました制御工学研究室の本科学生及び専攻科生の方々に深く感謝の意を表します。