一般社団法人 人工知能学会
Japanese Society for
Artificial Intelligence

人工知能学会研究会資料
JSAI Technical Report
SIG-Challenge-054-1 (8/17)

RoboCup SSL Humanoid のための ポイントクラウドサーバの提案

A Proposal of Point Cloud Server for the RoboCup SSL Humanoid

平井 雅之 [1*]　升谷 保博 [1]
HIRAI Masayuki[1]　MASUTANI Yasuhiro[1]

[1] 大阪電気通信大学
[1] Osaka Electro-Communication University

Abstract: 本稿では RoboCup SSL Humanoid の次世代の競技形態のための外部視覚共有システムとして「ポイントクラウドサーバ」を提案する．外部視覚としてフィールドを囲む複数の深度カメラを用いる．深度カメラに 1 台ごとにサーバプログラムを設け深度カメラから得られた深度画像を 3 次元の色付き点群に変換し，フィールド座標系に座標変換する．そこからフィルタリング処理を行い不要な点を除去し，さらに，データを圧縮してマルチキャストの UDP 通信でネットワークに送出する．このデータを各チームのプログラムが受け取り，そこから先は独自に処理を行う．深度カメラとして Kinect v2 を使って提案方法を実装し検証を行った．

1　はじめに

RoboCup サッカーの小型ロボットリーグ (Small Size Robot League, 以下 SSL)[1] とそのサブリーグである SSL Humanoid[2] のシステムでは，他のリーグと異なり，ロボット自身にはカメラを搭載する必要はなく，外部カメラを用いてフィールド上のロボットやボールの位置を認識しロボットを制御することができる．

現状では，図 1 のようにフィールド上空のカメラでロボットの頭頂部に取り付けたマーカのパターンを撮影し，2 次元的な画像処理を経て，その番号や座標，方向を算出している．しかし，この方式には多くの欠点がある．ロボットの頭頂部のマーカ板は，ロボットを動作させる際に邪魔になったり，ロボットが転倒してマーカが見えなくなると認識されなくなったりする．また，得られるのはロボットの座標と方向のみなので，ロボットの 3 次元的な状態を知ることはできない．これは，多様なポーズが可能なヒト型ロボットにとって大きな問題である．そこで，SSL Humanoud の提案当時からこの競技の最終段階の外部視覚システムの条件として図 2 のように 3 次元・マーカなしを想定している．このようなシステムであれば，複数の外部カメラで撮影したカメラに基づいて，複数のロボットの 3 次元的な形状を取得し，それに基づきロボットを制御することができ，外部視覚の特徴を活かした新たなアイディアの創出が期待できる．

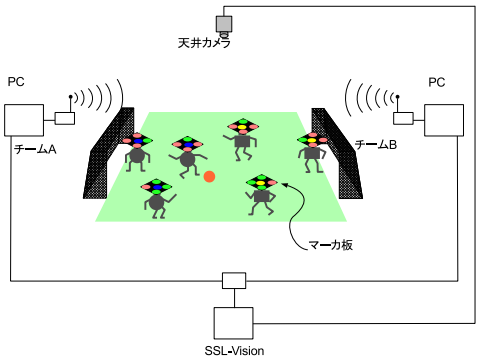

図 1: 現在の SSL Humanoid システム（2 次元・マーカあり）

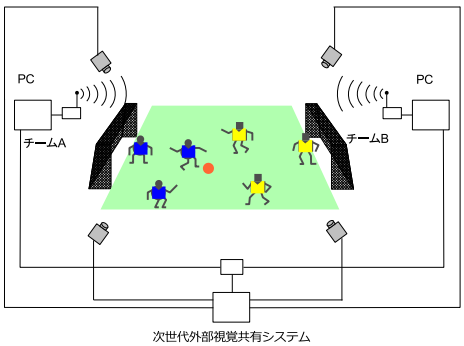

図 2: 次世代の SSL Humanoid システム（3 次元・マーカなし）

*連絡先：大阪電気通信大学 総合情報学部 情報学科
〒 575-0063 大阪府四條畷市清滝 1130-70
E-mail: odens.oecu@gmai.com

これを受けて，木村らはフィールド上の 3 次元空間のボクセルデータを共有することを提案し，それを具体化する「ボクセルサーバ」を開発した [3]．このサーバは，8 台のカメラを用い，3025[mm]×4050[mm]×500[mm] の領域を 10[mm] 立方のボクセルで表現し，それをネットワークを介してクライアントへ配信している．また，クライアント側では，ボクセルデータからロボットのポーズ推定を行い，その結果からロボットの行動決定ができることを示している [4]．さらに，植田らはこのボクセルサーバに色の情報を付け加えたカラーボクセルサーバを提案した [5]．これにより，ロボットとボールが接触すると 1 つのオブジェクトとして認識する問題が解決された．しかし，これらの方式では，十分な 3 次元情報を得ようとすると少なくとも 8 台のカメラを必要としており，設置やキャリブレーションに時間と手間がかかるという問題点がある．

そこで，RGB カメラではなく，最近手軽に使えるようになった深度カメラを 利用して次世代 SSL Humanoid のための外部視覚システムを実現する「ポイン トクラウドサーバ」を提案する．以下の節では，その考え方や実装について述べる．次に，深度カメラとして Kinect v2 を 4 台用いて，SSL Humanoid のフィールド（4.5[m] × 3[m]）に置かれたオブジェクトを撮影し，実装したシステムの性能を評価し，実際の競技に使えるかどうか検証する．

2　ポイントクラウドサーバの提案

複数の深度カメラから得られた情報をネットワークを介して参加チームが共有することについて考察する．まず決めなければいけないので，どこまでを共有してサーバ側で処理し，何をクライアント（チーム側）へ提供するかということである．これについて，以下のような三つの選択肢を考える．

1. 深度画像をそのまま提供．
2. 一般的な前処理を行ってから提供
3. 3 次元オブジェクトの認識を行ってから提供．

1 の場合，サーバ側の処理は非常に簡潔であるが，クライアント側の負担が重く，データの送信量が大きくなり処理速度が下がってしまう．2 の場合，データの送信量をある程度削減でき，処理速度も確保しやすいが，カメラのキャリブレーションなどの課題がある．3 の場合，一番データの送信量が少ないが，サーバ側の処理が多く，クライアント側での工夫を行う余地が少ない．そこで，本稿では，2 を選択し，深度画像から得られた点群データを提供する「ポイントクラウドサーバ」を提案する．

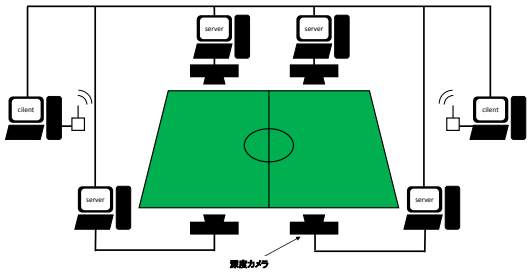

図 3: ポイントクラウドサーバの概念図

ポイントクラウドサーバの概念図を図 3 に示す．複数の深度カメラでフィールドを囲い撮影する．深度カメラ 1 台につきサーバプログラム 1 個を割り付ける．共有するデータはマルチキャストの UDP 通信でネットワークに送出する．

サーバプログラムでは深度画像を 3 次元の色付き点群データに変換し，共通の座標系に変換する．次に，フィールド外や床といった不要な点を除去と点のデータ量の圧縮を行い送信量を削減する．複数のサーバのデータを区別するために，それぞれに別々のマルチキャストアドレスとポート番号を割り当てる．

現在の共有視覚システムである SSL-Vision では，60fps 程度の頻度で処理をしている．一方，一般的な深度カメラでは，30fps 程度の頻度でデータの取得ができるので，提案のシステムでもそれと同じ処理頻度にできるのが理想的である．しかし，車輪型ロボットに比べてヒト型ロボットは動作が遅いため，10fps 程度の処理頻度でも実用できるのではないかと考えている．

厳密な計測のためには，センサ間の同期ができた方が望ましいが，機材的な制約でそれは難しいかもしれない．そこで，競技としては非同期のデータに基づくことを前提としても良いのではないかと考える．ただし，サーバ間での時刻は同期しておき，点群データにタイムスタンプを付与する．

3　ポイントクラウドサーバの実装

2 節で提案したポイントクラウドサーバを検証するために，実装したシステムについて述べる．

3.1　機材・開発環境

実装した PC はサーバ側は Windows10，クライアント側は Windows8.1 である．開発環境には Visual Stu-

dio 2015，使用言語は C++である．

深度カメラは Kinect v2 を用いた．深度画像の解像度は 512 × 424 画素である．また，Kinect の画角は水平方向に 70 [deg]，垂直方向に 60 [deg] ，計測範囲は 0.5[m] から 8[m] までである．

本研究では，点群を表示させ操作するため，およびキャリブレーションに使用した点群内での平面検出や不要な点群のフィルタリングに，Point Cloud Library を使用している．実装に利用したバージョンは，1.8.1 である．

3.2 サーバの実装

まず，Kinect の深度画像は KinectGrabber[6] を使用して色付き点群に変換する．

Kinect の深度画像から得られた点群の座標値は，カメラ座標系で表現されている．クライアント側での点群のマージが容易になるように，サーバ側で点群を共通のフィールド座標系へ変換する．図 4 に示すように，フィールド座標系は，フィールドの平面上の中心を原点とし，フィールド平面の法線方向を z 軸，ゴールとゴールを結ぶ方向を x 軸とする．

ある点 $\boldsymbol{p}$ を，カメラ座標系で表現したものを ${}^{c}\boldsymbol{p}$，フィールド座標系で表したものを ${}^{f}\boldsymbol{p}$ とすると．両者の関係は，式 (1) のように表現できる．

$$ {}^{f}\boldsymbol{p} = {}^{f}R_{c}\,{}^{c}\boldsymbol{p} - {}^{f}\boldsymbol{p}_{c} \tag{1} $$

ここで，${}^{f}R_{c}$ はフィールド座標系に対するカメラ座標系の回転を表す行列，${}^{f}\boldsymbol{p}_{c}$ はフィールド座標系に対するカメラ座標系の並進を表すベクトルである．本稿の実装では，別のキャリブレーションプログラムで，この回転行列と並進ベクトルを推定し，その結果をファイルを介してサーバプログラムにが受け取って処理に利用する．

Kinect v2 の深度画像から得られる点群の点の数は最大で 512 × 424 =217088 個である．このデータをそのまま送出するとデータの送信量が多く処理速度の低下に繋がるためデータの送信量を削減する必要がある．

点群の中には不要な点が数多く存在するためフィルタリングを行う．まずフィールドから大きく外れた箇所の点群を除去する．SSL Humanoid のフィールドは 3025 × 4050[mm] であるので，フィールドの原点を中心とし，3600 × 5000[mm] に含まれない点を除去する．次に床面は不要なので z 軸方向に 5[mm] 以下の点群を除去する．これにより，フィールド上のオブジェクトの点群だけが残る．

さらに，点群の点 1 個当たりのデータ量も減らす．PCL における色付き点群の 1 点のデータは．座標値 (X,Y,Z) がそれぞれ [m] 単位の浮動小数点数で 4byte，

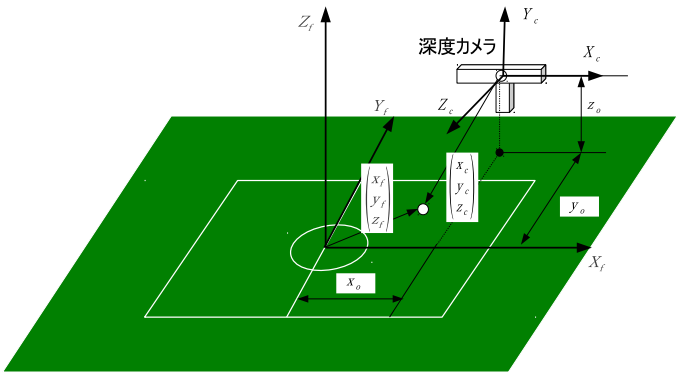

図 4: フィールド座標系とカメラ座標系

色情報 (R,G,B,α) がそれぞれ 1 byte，合計で 16byte である．これを，座標値はそれぞれ [mm] 単位の整数 2byte に，色情報はそれぞれ 5bit に変換し合計で 8byte に圧縮する．

3.3 テストクライアント

テスト用のクライアントプログラムを作成する．全てのサーバから新しいデータが届いた場合，点群をマージし表示する機能のみが実装されている．今後は，マージした点群に対して様々な処理を行う処理の実装が必要である．

3.4 キャリブレーション

キャリブレーションによって求めるべきものはカメラ座標系からフィールド座標系に変換するための回転行列と並進ベクトルである．この手法はフィールド座標系の xy 面と xz 面と yz 面に一致する平板を設置し，それを撮影した点群データから，回転行列と並進ベクトルを求める．

xy 面は床を使用し，xz 面と yz 面に相当する平板として衝立などで壁を作る．その 2 つの壁と床を Kinect で撮影し，その点群を使い PCL で平面検出を行う．その際の深度カメラと平面と座標系のおおよその位置関係を図 5 に示す．この例ではフィールドの第 4 象限に Kinect を設置している．

3.4.1 キャリブレーションの実装

点群から平面を取得する方法は PCL のモデルタイプは SACMODEL_PLANE で，検出方法は SAC_RANSAC

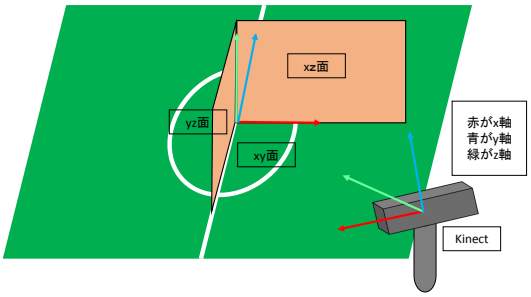

図 5: 深度カメラ，平面，座標系の位置関係

を用いて検出する．閾値は 0.05[m] で試行回数は 50 回である．これにより点群内の平面の式の係数が得られる．

この方法で平面を検出すると点群内で一番大きな平面が求められる．その後元の点群から求めた平面に含まれる点群を取り除き再度平面検出を行うことで複数の平面を検出する．これを 3 つの平面が検出されるまで繰り返す．

検出された 3 平面のパラメータから，座標変換に必要な回転行列と並進ベクトルを推定し結果をファイルに保存する．

3.4.2 キャリブレーションの原理

xy 平面，xz 平面，yz 平面の係数をそれぞれ a_z, b_z, c_z, d_z，a_y, b_y, c_y, d_y，a_x, b_x, c_x, d_x とする．ただし以下の様な関係を満足しているとする．

$$\sqrt{a_i^2 + b_i^2 + c_i^2} = 1 (i = x, y, z) \tag{2}$$

xy 平面の式を式 (4)，xz 平面の式を式 (5)，yz 平面の式を式 (6) と表す．

$$a_z x + b_z y + c_z z + d_z = 0 \tag{3}$$
$$a_y x + b_y y + c_y z + d_y = 0 \tag{4}$$
$$a_x x + b_x y + c_x z + d_x = 0 \tag{5}$$

3 つの平面の式を x, y, z に対する連立方程式とする．これを解くと，その解の x, y, z はカメラ座標系におけるフィールド座標系の原点である．この点を以下の様に表す．

$${}^c\boldsymbol{p}_f = \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix} \tag{6}$$

平面の式の a, b, c はそのまま平面の法線ベクトルを表しており，フィールド座標系の座標軸と平行である．これを以下の様に書くことにする．

$$\boldsymbol{n}_i = \begin{pmatrix} a_i \\ b_i \\ c_i \end{pmatrix} (i = x, y, z) \tag{7}$$

ただし，$\boldsymbol{n}_i$ の向きに任意性があるが，Kinect がフィールド座標系の原点の方向を向いているという前提で，Kinect がフィールド座標系のどの象限にあるかによって適切に設定する．推定した平面のパラメータは誤差を含んでいるため，各法線ベクトルは直交しているとは限らない．そこで，床面の法線を基準に補正する．

$$\boldsymbol{n}'_x = \boldsymbol{n}_x - (\boldsymbol{n}_x \cdot \boldsymbol{n}_z)\boldsymbol{n}_z \tag{8}$$
$$\boldsymbol{n}''_x = \frac{\boldsymbol{n}'_x}{|\boldsymbol{n}'_x|} \tag{9}$$
$$\boldsymbol{n}'_y = \boldsymbol{n}_z \times \boldsymbol{n}''_x \tag{10}$$

ここで $\boldsymbol{x} \cdot \boldsymbol{y}$ はベクトル $\boldsymbol{x}$ と $\boldsymbol{y}$ の内積，$\boldsymbol{x} \times \boldsymbol{y}$ はベクトル $\boldsymbol{x}$ と $\boldsymbol{y}$ の外積を表す．カメラ座標系におけるフィールド座標系の姿勢は回転行列を使って式 (11) のように表される．

$${}^cR_f = (\boldsymbol{n}''_x \boldsymbol{n}'_y \boldsymbol{n}_z) \tag{11}$$

これらを使うと，フィールド座標系からカメラ座標系の変換は，式 (12) のように表される．

$${}^c\boldsymbol{p} = {}^cR_f {}^f\boldsymbol{p} + {}^c\boldsymbol{p}_f \tag{12}$$

この式を ${}^f\boldsymbol{p}$ について解くと式 (13) になる．

$${}^f\boldsymbol{p} = ({}^cR_f)^{\mathrm{T}}({}^c\boldsymbol{p} - {}^c\boldsymbol{p}_f) = {}^fR_c {}^c\boldsymbol{p} - {}^f\boldsymbol{p}_c \tag{13}$$

ここで，

$${}^fR_c = ({}^cR_f)^{\mathrm{T}} \tag{14}$$
$${}^f\boldsymbol{p}_c = {}^fR_c {}^c\boldsymbol{p}_f \tag{15}$$

であり，これらがキャリブレーション結果そのものである．

4 評価実験

この節では，本稿で提案したシステムや手法の評価実験について述べる．SSL Humanoid のルールで定められた 3025 × 4050[mm] のフィールドに対して 4 台の Kinect を設置した．Kinect を設置したフィールド座標系での位置を表 1 に示す．単位は [mm] である．取得した点群データはテストクライアントで表示し，評価に使用する．実験に使用したサーバ PC とクライアント PC の仕様を表 2 に示す．

表 1: Kinect の配置座標 [mm]

	x[mm]	y[mm]	z[mm]
第 1 象限	2800	2000	1000
第 2 象限	-2900	1700	1100
第 3 象限	-2800	-2100	1000
第 4 象限	2800	-2000	1100

表 2: 使用した PC の仕様

	CPU	RAM
server	Intel Celeron@1.70GHz	4GB
cilent	Intel Core i7@2.30GHz	8GB

4.1 複数カメラの統合の確認

本稿で提案したキャリブレーションを用いて Kinect の位置姿勢を推定し，その結果を用いて各 Kinect から取得した点群をマージする．3 つの平面を用意するために使用した衝立は 1000 × 600[mm] と 1100 × 600[mm] の 2 枚のプラスチックダンボールを固定したものである．これをコートの印に合わせて立てることで床を合わせ，互いに直交する三つの平面を用意する．キャリブレーションを行う際の様子を図 6 に示す．統合確認実験の際のフィールド上のオブジェクトの配置を図 7 に示す．フィールドには大きさ約 260 × 140 × 170[mm] のロボットが 3 台と大きさ約 350 × 160 × 190[mm] のロボットが 3 台と直径 60[mm] のボールを 1 つ設置してある．マージ後の点群を図 8 に示す．図 8 より，キャリブレーションにより 4 つの Kinect から送られてくる点群データがマージされ，SSL Humanoid のフィールド全域の点群を取得できていることがわかる．しかし Kinect 間の座標に少なからずずれが見られ，キャリブレーションの際の平面の設置方法の改善や基となる原理の見直しが必要であることがわかる．

フィールドの平面とフィールド上のオブジェクトでキャリブレーションやマージの処理の良し悪しを判断するために，フィールド外の点を削除した結果を図 9 から図 11 に示す．図 9 よりフィールド内の白線にずれがあることがわかり，図 10 と図 11 より，床の点群が一致していることがわかる．これらの結果からキャリブレーションの際に使用した平面として，床はかなりの精度で平面が検出できている．これにより，平面を大きくするなどの設置方法を改善すると他の平面での精度も向上すると考えられる．

図 6: キャリブレーションの様子

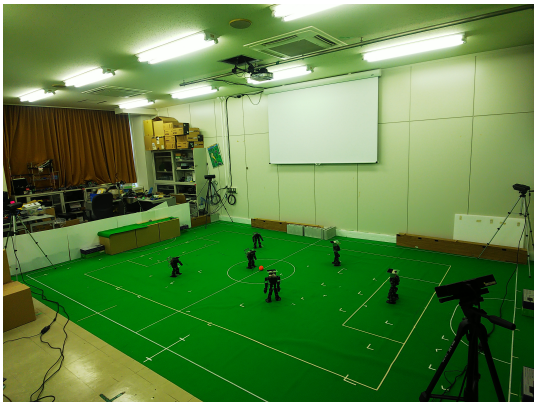

図 7: 点群取得時のフィールド

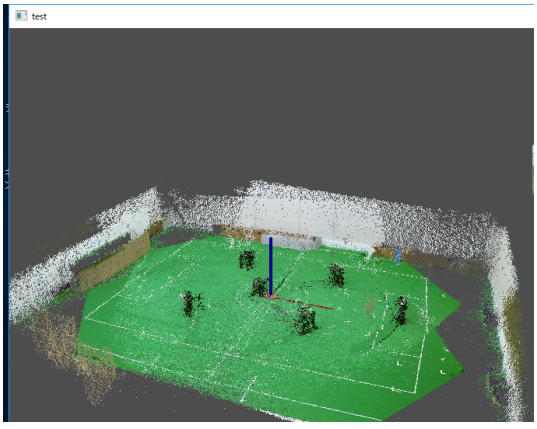

図 8: マージされた点群を全て表示したもの

4.2 フィルタリングの効果の確認

本稿で提案したフィルタリングを行い，実際のデータの削減量，処理速度の変化を確認する．床面のフィルタリングを行った点群を図 12 に示す．フィルタリン

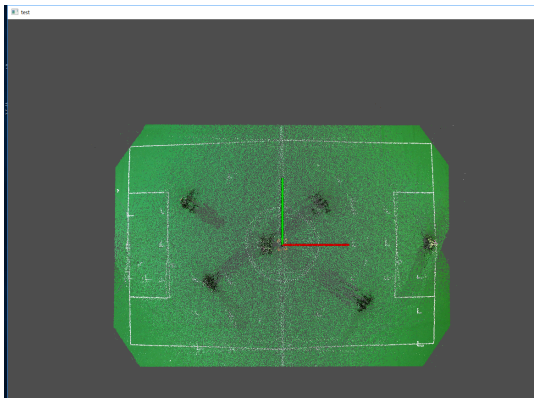

図 9: フィールド外の点を削除した点群（XY 面）

図 10: フィールド外の点を削除した点群（XZ 面）

図 11: フィールド外の点を削除した点群（YZ 面）

表 3: フィルタリング前後のデータ

	フィルタリング前	フィルタリング後
点数	217088 × 4	約 5500
データ量	1736704[byte] × 4	約 43200[byte]
処理速度	1〜3[fps]	6〜9[fps]

グを行う前後での点数，データの大きさ，処理速度を表 3 に示す．

図 12 より，フィルタリングにより床面がほとんど除去されていることがわかる．表 3 より，フィルタリングを行う前と後で点数，データの大きさが大きく削減され，処理速度も向上していることがわかる．目標としていた 10fps には届いていないため，さらに改善は必要であるがこのフィルタリングが十分に有効であることがわかる．

図 12: フィルタリングで床面を削除した後の点群

表 4: 使用したロボットと点数

使用ロボット	RIC-30
大きさ	260 × 140 × 170[mm]
中心での点群	約 800
四隅での点群	約 2200
ボールの大きさ	直径 60mm
中心での点群	約 100
四隅での点群	約 300

4.3 得られたオブジェクトの点群の確認

本稿で提案したキャリブレーションとフィルタリングを行い得られた点群データはオブジェクトを認識するのに使用できるかを確認し，ロボットに複数の姿勢を取らせ点群での変化を確認する．検証に使用したロボットとボールの写真を図 13 に示す．オブジェクトをフィールド座標系の (0,0) 付近に設置し，写真とほぼ同じ角度から見た点群を図 14 に示す．真上から見た点群を図 15 に示す．オブジェクトをフィールド座標系の (1800,1300) 付近に設置し，写真とほぼ同じ角度から見た点群を図 16 に示す．使用したロボットとボールのそれぞれについての大きさと得られた点の数を表 4 に示す．なお，点数は Kinect4 台分の合計である．

図 14 と図 15 より，立っているロボットと倒れているロボットに明らかな違いがあることがわかる．重心

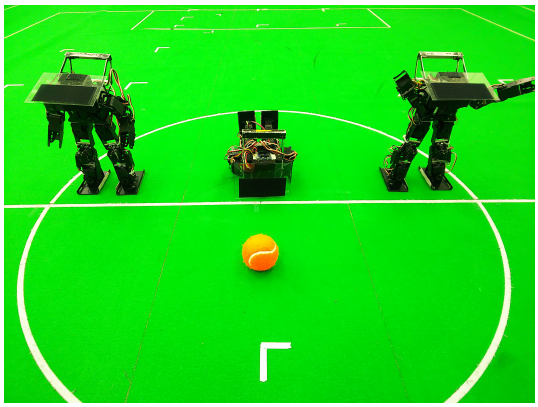

図 13: オブジェクトの点群の確認実験の対象

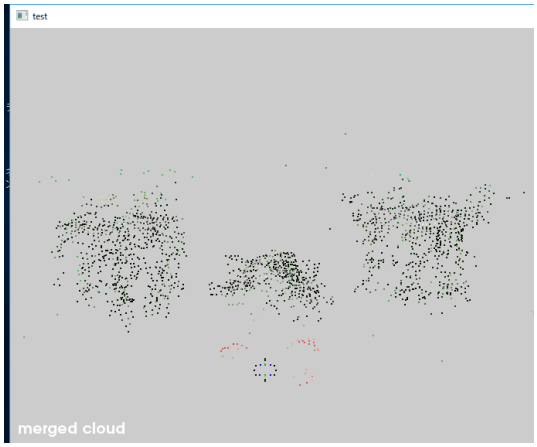

図 14: オブジェクトの点群（フィールドの中心，斜め上から）

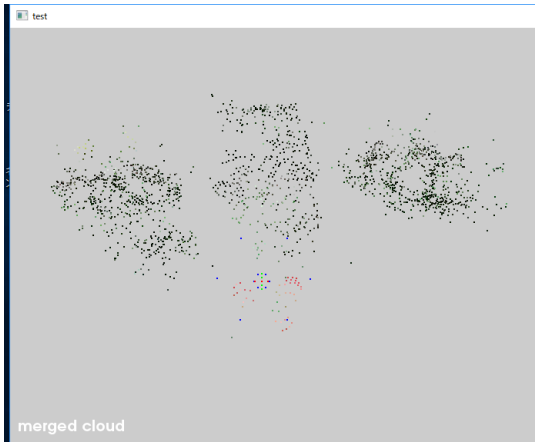

図 15: オブジェクトの点群（フィールドの中心，真上から）

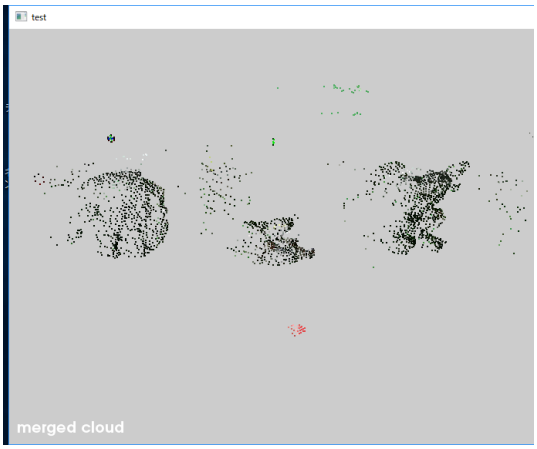

図 16: オブジェクトの点群（フィールドの隅，斜め上から）

の位置が低くなっており，点群が前後に広がっているのがわかる．これを利用すればロボットが立っているのか倒れているのかの判断が可能である．しかし，点数が少ないため，左のロボットと右のロボットに大きな違いは見られない．大きな変化は現状でも判断が可能であるが細かな姿勢の違いを判断するにはカメラの解像度を上げるなどを行い点数を増やす必要がある．

図 16 では図 14 と違い，左のロボットと右のロボットの姿勢の違いが見て取れる．フィールドの隅だと深度カメラから近く，オブジェクトの点数が増えるため，ロボットの状態が詳細にわかる．

図 14 と図 15 より，ロボットの点群を取得した際は問題にはならなかったがボールの点群を取得すると座標変換の結果の不一致がはっきりと確認できる．ボールはロボットに比べ小さなオブジェクトであるためずれが大きく現れてしまう，これではボールの認識に影響があるため精度の改善が必要である．

表 4 より，オブジェクトの点数がフィールドの位置によって約 3 倍程違うことがわかる．フィールドの中心はどの Kinect からも遠いからである．

5　クライアント側の処理の検討

この節では評価実験の結果を経て，クライアント側の処理として求められるものを検討する．

5.1　クラスタリング

まずロボットやボールを認識するにはクラスタリングが必要である．個々のオブジェクトが孤立しているならクラスタリングするのは難しくないがオブジェクト同士が接触した際のクラスタリングが課題である．ロ

ボットとボールの場合は色などで判別できるがロボット同士の接触は判別がつきにくいと考えられる

5.2 ロボットの位置と向き

ロボットの認識ができたと仮定すると次の問題はロボットの位置と向きの認識である．クラスタリングした点群データの重心を求めれば位置はわかる．クラスタリングした点群データから主軸方向を認識すれば縦の向きがわかる．多くのロボットは上から見ると横に長いことが殆どであるのでそれも利用できるが，それだけでは前後の判断が難しい．ロボットの胴体の前面にのみ色の違うものを張るなど何かしらの対処が必要である．

5.3 ロボットのチームと号機の区別

各ロボットに指令を送る際にロボットの識別は必須である．敵味方の区別も必要である．マーカ板を撤廃したがチームの認識のため，色のついたラベルなどを張ることが必要だろう．

6 おわりに

本稿では，RoboCup SSL Humanoid の共有視覚システムとして利用することを想定した「ポイントクラウドサーバ」を提案した．

4 台の Kinect でフィールドを全域を撮影することができた．フィルタリングなどの処理でデータを削減すると処理頻度が向上した．もう少し処理速度の向上ができればヒト型ロボット用としては十分実用が可能である．点群のデータはオブジェクトを認識するのに十分利用できる．

今後の課題は，キャリブレーションの精度の向上，ポイントクラウドサーバシステムの処理周期の向上，クライアント側のプログラム作成である．

参考文献

[1] 長坂保典: ロボカップの道しるべ : 第 3 回 小型ロボットリーグ，情報処理, vol.52 No.1, pp.95–110, 2011.

[2] 升谷保博，成瀬正: 外部カメラを用いたヒト型ロボットによるサッカー競技 RoboCup Soccer SSL Humanoid，人工知能学会誌, Vol.25, No.2, pp.213–219, 2010.

[3] 木村堯海，升谷保博: 複数の外部カメラ画像に基づくヒト型ロボットの 3 次元形状の実時間取得 RoboCupSoccer SSL Humanoid の 3 次元情報サーバをを目指して, 第 31 回人工知能学会 AI チャレンジ研究会, pp.33–38, 2010.

[4] 植田康生，木村尭海，勢川友樹，升谷保博: 外部カメラ画像から得られた複数のヒト型ロボットのボクセルデータに基づく行動決定, 第 33 回人工知能学会 AI チャレンジ研究会, pp.7–12, 2011.

[5] 植田康生，升谷保博: RoboCup SSL Humanoid のためのカラーボクセルサーバの提案と応用, 第 37 回人工知能学会 AI チャレンジ研究会, pp.9-14, 2013

[6] 杉浦司: GitHub UnaNancyOwen/KinectGrabber `https://github.com/UnaNancyOwen/KinectGrabber/tree/Kinect2Grabber`