筑波大学大学院博士課程

システム情報工学研究科特定課題研究報告書

# Kinect サーバおよび
# サンプルクライアント研究開発
# ―複数 Kinect 連携機能―

小菅 拓真

(コンピュータサイエンス専攻)

指導教員 田中 二郎

２０１２年３月

# 概要

本プロジェクトは, 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻, 高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムにおける研究開発プロジェクトとして，実施したものである．Microsoft 社の製品である Kinect センサで取得した骨格や深度等のセンサデータをネットワークを介して扱えるようことを目的とし, 機能の検討, 技術調査および開発をおこなった．本プロジェクトでは，主な機能となる Kinect センサのセンサデータ送信機能と付加機能である WebSocket によるサーバ・プッシュ機能および複数 Kinect 連携機能の 3 つの機能を有する Kinect サーバシステムの開発と，クライアント側からシステムへのアクセスを支援するためのクライアントライブラリおよび Kinect サーバシステムの機能の使用例を示すためのサンプルクライアントの開発をおこなった．

筆者は広範囲のデータ取得を目的とし，複数の Kinect センサから取得したデータを組み合わせ, 1 つの広範囲のデータとして提供する複数 Kinect 連携機能についての調査およびプロトタイプの開発を担当した．開発したプロトタイプによる検証の結果，いくつかの問題は残るがデータ連携をおこなえることが確認できた．

本報告書では，本プロジェクトの開発背景からシステム概要，筆者が担当した機能，プロジェクト体制等について報告する．

# 目次

# 図目次

# 表目次

# 第1章　はじめに

研究開発プロジェクトにおいて，筑波大学システム情報系情報工学域に所属する高橋伸准教授が提案した「Kinect サーバおよびサンプルクライアントの構築」というテーマのもと，筆者を含む 4 名の学生チームである Team.Kineco によって，システムに必要な機能の立案，検証，開発をおこなった．具体的には，Microsoft 社の Kinect センサから得られる RGB 情報や深度情報等のセンサデータをネットワークを介して提供する Kinect サーバシステムおよび拡張機能の開発である．

筆者は広範囲のデータを取得できるようにする機能をシステムの付加機能の 1 つとして，複数の Kinect センサから取得したデータを組み合わせ，1 つの広範囲のデータとして提供する複数 Kinect 連携機能についての調査およびプロトタイプの開発を担当した．

本報告は次のような構成になっている．第 2 章でプロジェクトの背景と目的，Kinect センサについて説明し，第 3 章で Kinect サーバシステムについて述べる．第 4 章で，開発担当部分である複数 Kinect 連携機能についてと検証作業について説明をおこなう．そして，第 5 章でプロジェクト体制について述べ，第 6 章でプロジェクトの振り返りをおこない，第 7 章で本報告書の結論を述べる．

# 第2章　背景とプロジェクトの目的

本章では，本プロジェクトの背景と，今回利用する Kinect センサについて説明した後に本プロジェクトの目的について述べる．

## 2.1　背景

現実世界では実現することが困難なことを仮想世界という形で表現することは往々にしてある．仮想世界を生み出すものの一つとして仮想現実があるが，近年では現実世界との歩み寄りをみせている．仮想現実を現実世界と融合させた複合現実感と呼ばれるものがあり，メディアアート等の作品に利用されている．一方で，現実世界から仮想世界へのアプローチもみられる．現実世界の映像に仮想世界の情報を重畳表示させる拡張現実感は産業利用され始めたことにより，目にする機会が増えてきた．これらの現実と仮想の歩み寄りにおいては視覚情報だけでなく，ユーザインタフェースも重要となってくる[1].

コンテキストに合う行動を自然な動作によっておこなうことができるインタフェースが利用されるシーンが増えてきている．NI(Natural Interaction)や Microsoft が提唱する NUI(Natural User Interface)という言葉も聞かれるようになってきた．これはセンサ類を搭載した入力デバイスの普及により，従来の情報機器に対する操作とは異なる音声や身体の動き，ジェスチャー等の直観的な操作によって入力をおこなう機会が世の中に広まってきたことも影響しているだろう．

これはテレビゲーム業界において特に顕著である．Nintendo Wii リモコンやバランス Wii ボードの登場によりゲームの遊び方は大きく変わることとなった．ボタン等の押下よる従来型のコントローラによる操作とは異なり，実際に身体を動かして操作をする方が直感的で分かり易かったこともあり人気を博した．そんな中で登場したものが Microsoft Kinect センサ[2]である．

## 2.2　Kinect センサについて

Kinect センサは Microsoft 社がゲーム機の XBOX360 用に発売したコントローラであり，世界初の非接触型ゲームコントローラである．従来何かしらのデバイスを身に付ける必要があったが，Kinect センサによってデバイスを一切身にまとわずに操作が可能となった[3][4].

Kinect センサは距離画像センサ等を備え，3D リアルタイムデータを取得することができるため，複合現実感や拡張現実感への親和性も高い．また，Kinect センサの登場以前の距離画像センサといえば，業務用若しくは研究用途のものでないとリアルタイムでの深度データの取得が困難であり，価格も数十万円から百万円程度していた．そのような状況において Kinect センサは，赤外線パターン照射を普通の CMOS カメラで解析することで実現可能な独自の方式により，一万数千円という圧倒的低価格で誰もが入手可能な距離画像センサを提供できるようになった. そのため, ゲーム機にとどまらず新たなアプリケーションの開発等, 様々な用途で世界中の人々に利用されている．

● Kinect センサの仕様

Kinect センサには複数のセンサが搭載されている[5]. 大まかな仕様は表 2-1 の通りである.

表 2-1 Kinect センサの仕様

| | |
|---|---|
| RGB センサ | 解像度 VGA(640×480) |
| 距離画像センサ | 解像度 VGA(640×480) |
| フレームレート | 30 fps |
| 認識距離 | 0.5 － 9 メートル |
| 垂直視野 | 43 度 |
| 水平視野 | 57 度 |
| チルトレンジ | -30 － +30 度 |
| 音声フォーマット | 16kHz 16bit モノラル PCM |

● 距離画像センサ

距離画像センサは Kinect センサに搭載されているセンサの中で最も重要なセンサといえる．赤外線プロジェクタと赤外線センサの組み合わせで動作しており，赤外線プロジェクタから照射された赤外線パターンの歪みから距離を計測している．

赤外線を利用した距離画像センサは主に，赤外線が行き来する時差を利用した TOF(Time of Flight)方式と，赤外線プロジェクタにより特徴的な赤外線パターンを照射し，赤外線パターンの歪みから距離を計測する方式の 2 種類の方式に分けられる. Kinect センサに搭載されている距離画像センサでは後者の方式を採用しており, Light Coding と呼ばれる独自の方式を使用している．この方式では赤外線照射パターンの各点が一意に識別できるようになっており，複数の Kinect センサを同時に使用しても干渉しない.

● RGB センサ

画像データが取得できるセンサであり，距離画像センサで得られる像とほぼ同じ領域の画像を取得することができる．しかしながら，視野角が距離画像センサと多少異なるため，それぞれの像を重ね合わせるには調整が必要となる.

● 赤外線センサ

距離画像センサにおける赤外線パターンの読み取りに使用しているセンサである．そのため Kinect センサにおいては，赤外線センサから得られるデータを直接扱うことはあまりない. なお, 赤外線センサでのデータの取得と RGB センサのデータとの同時取得はできない.

● アレイマイク

アレイマイクとは, 複数のマイクを並べ, その出力を足し合わせ指向性を得るものであり，音声マイクが 4 つ Kinect センサの下部に搭載されている．これにより，音の発生場所を求める音源位置推定が可能となっている.

● 加速度センサ

3 軸加速度センサを搭載しており，加速度センサの値から重力加速度が得られるため，Kinect センサの傾きが計算できる．なお，チルトモータの角度は地面の水平面に対する角度で制御されている.

## 2.3 プロジェクトの目的

本プロジェクトでは, 様々な用途での利用が考えられる Kinect センサの 3D リアルタイムデータへ，より容易なアクセスを可能にすることを目的としている．具体的には，Kinect センサを用い骨格や深度情報等を提供するサーバおよびそのデータを活用した実装例を示すサンプルクライアントの開発をおこなう． また, 開発したシステムをオープンソースとして一般に公開し，実際にシステムを利用してもらうことも検討している．

# 第3章　Kinect サーバシステム

本章では，顧客の要望をもとに本プロジェクトで開発した Kinect センサから取得したデータをリアルタイムで提供する機能，WebSocket によるサーバ・プッシュ機能および複数 Kinect 連携機能について述べる．本システムで利用するライブラリや技術について紹介し，システムの構成について述べる．

## 3.1　顧客の要望

システムの要件を考えるにあたり，顧客に対しヒアリングをおこなった．顧客からのシステムへの要望は以下の 4 点である．

- ネットワークを介してデータを提供できること
- Kinect センサデータをクライアント側で取得できること
- モバイル端末等の Kinect センサを接続できないデバイスでも利用できること
- 本システムから取得したデータを用いたクライアントの実装例を示すこと

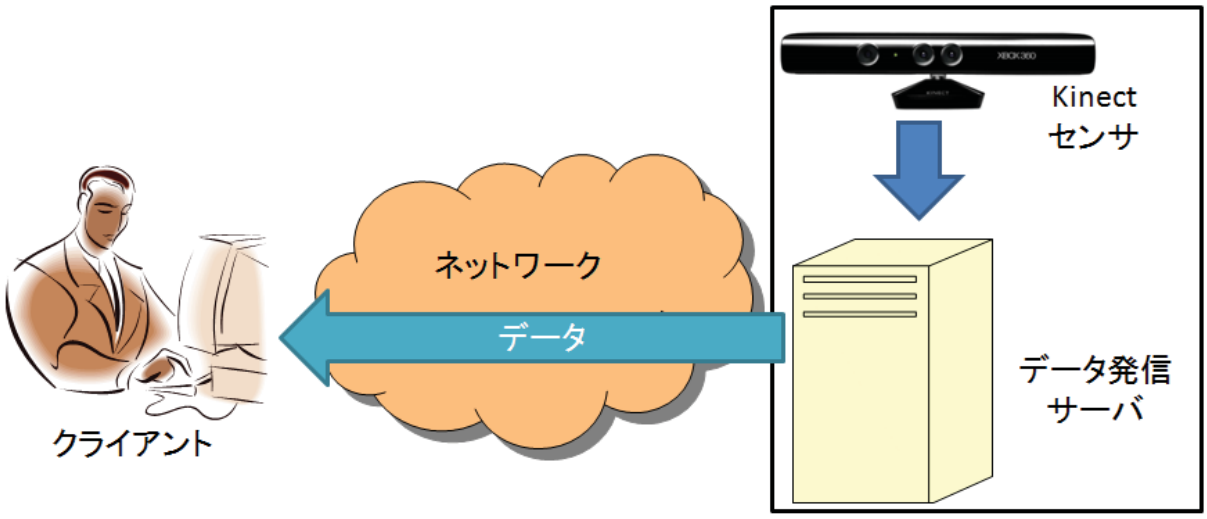



**図 3-1 顧客の大まかなシステムのイメージ**

顧客が考えるシステムのイメージは図 3-1 のようなものであった．Kinect センサを接続したサーバが Kinect センサからデータを取得し，クライアントへデータを送信するというものである．

また，以下の 2 点の付加機能がオプションの要望として挙げられた．

- WebSocket を用いたクライアントへのデータ提供
- 複数の Kinect センサのデータを連携させたデータの提供
  - クライアント側から複数台のセンサがあることを意識させずに利用できること

## 3.2　　データ提供方法

3.1 節の要望を踏まえ，WebAPI によるクライアントへのデータ提供を提案した．WebAPI であれば Web ブラウザからでも利用可能なため，モバイル端末からでも利用が可能であると考えたためである．

当初は WebAPI の提供のみを考えていたが，クライアント側から利用する際の手間を軽減するため，裏で HTTP リクエストを出しデータを取得するクライアント用ライブラリも作成することとした．

## 3.3　　想定する利用者

想定する利用者としては，研究目的で利用するユーザとアプリケーション開発等に利用するユーザである．Kinect センサからのセンサデータについては研究目的で利用されることを，クライアント側で扱いやすいようにセンサデータを加工したデータについてはアプリケーション開発等で利用されることを想定している．

## 3.4　　システムの機能

本プロジェクトでは，図 3-2 に示す Kinect センサデータ提供機能，WebSocket によるサーバ・プッシュ機能，複数 Kinect 連携機能の 3 つのサーバ側の機能，クライアントでの実装例の提示，およびクライアント側からデータ取得を容易にするためのクライアントライブラリを開発することとした．

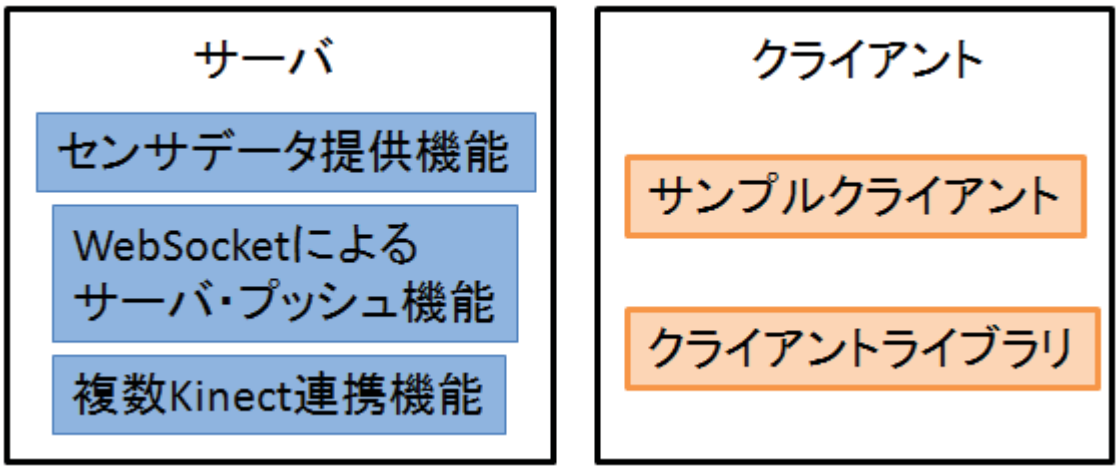



図 3-2 システムの機能

## 3.5　　使用技術

本節では，本システムの基本的な機能を提供するために使用するライブラリや技術等について述べる．

- OpenNI

OpenNI[6]は，サーバ側システムにおいて Kinect センサのデータ処理で利用するライブラ

リである．

OpenNI(Open Natural Interaction)は Kinect センサを開発している PrimeSense 社等が中心となって開発している Kinect センサ等の NI デバイスの共通インタフェースを提供する仕組みである．OpenNI の構成を図 3-3 に示す．

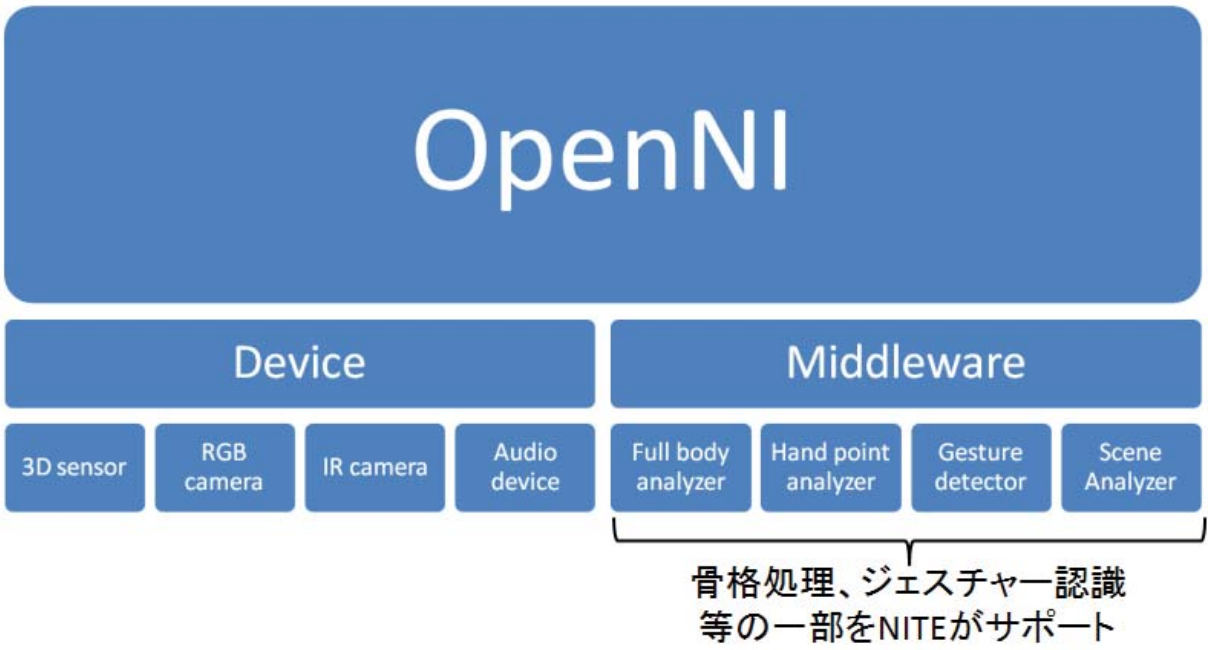



図 3-3 OpenNI の構成*

OpenNI 本体，デバイスのコントロールをするハードウェアドライバ部分，デバイスからのデータに対して画像処理をおこない，ジェスチャー認識等をおこなうミドルウェア部分からなり，複数のデバイスをサポートできるようになっている．ミドルウェアとしてユーザ検出やトラッキングをおこなう NITE が提供されており，NITE を使用することで，様々なジェスチャー検出をすることも可能となる．

Kinect センサ以外のデバイスでも同じインタフェースを利用して開発することできるため，今後増えていくと予想される Kinect センサ以外の OpenNI 対応のデバイスによる将来のシステム拡張にも対応が可能である．

- GNU libmicrohttpd

GNU libmicrohttpd[7]は，サーバ側システムにおいてリクエストおよびレスポンス処理で利用するライブラリである．

HTTP サーバを作成するための C 言語ライブラリであり，デーモンの作成やリクエストの処理等の HTTP サーバを構築する最小限の機能で構成されているため，軽量で高速に動作する．今回 Kinect センサのデータ処理に用いる OpenNI を C++で利用するため，同時に利用可能な本ライブラリを採用することとした．

- JSON

本システムでクライアントへ提供するデータ形式の一つとして JSON(JavaScript Object

* 原典 http://nma.web.nitech.ac.jp/fukushima/openni/openni.html

Notation)[8]を利用する．

JSON は，XML 等と同様のテキストベースのデータフォーマットであり，XML よりシンプルなものとなっている．基本的な値を扱うだけであれば，XML よりも JSON の方が簡単に記述でき，JavaScript ではそのままオブジェクトとして扱うことが可能なデータ構造となっている．オブジェクトの構造は図 3-4 のようになっている．

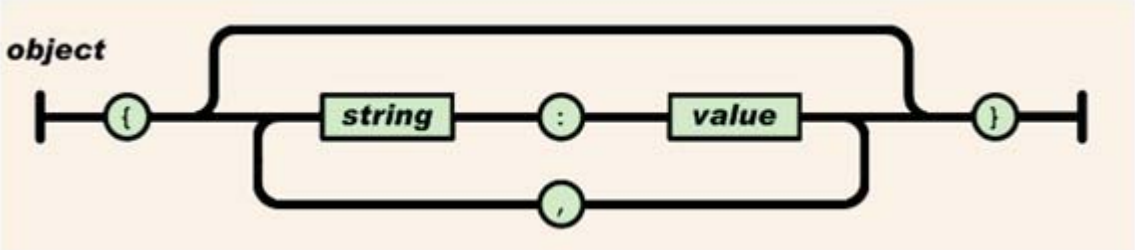



図 3-4 オブジェクトの構造†

- WebSocket

WebSocket[9]は，クライアント・サーバ間の双方向通信を実現するための仕組みである．WebSocket は最初にコネクションを結んでしまえば，後は同じコネクション上でやり取りをするため，通信ロスを減らすことができる．また，HTTP 通信のようにクライアントからリクエストを受けてからレスポンスとしてデータを送信するのではなく，サーバ側のタイミングでデータを送信できるサーバ・プッシュが可能である．そのため，WebAPI 方式では困難であったジェスチャー認識も容易に取り入れることができる．センサデータの提供という点ではサーバ・プッシュ型は都合が良く，汎用性の高い設計にすることで Kinect センサ以外の様々なセンサを利用することができるプラットフォームを構築することが可能となる．さらに，サーバ・プッシュによりクライアントに通知できるため，クライアント側からのサーバに対するポーリングが不要となりネットワークのトラフィック削減が見込める．

なお，開発時点で WebSocket は仕様が未だに確定していない状況にあったため，新しい仕様のもとでは互換性がなく動作しないことが考えられた[10]．現在は仕様が固まり，RFC6455 として定められている[11]．

## 3.6 システム構成

本システムは大きく分けて Kinect センサデータ提供機能，WebSocket によるサーバ・プッシュ機能，複数 Kinect 連携機能の 3 つの機能を持っている．本システムの根幹となる Kinect センサデータ提供機能がクライアントからのリクエストを指定したポートで受付け，リクエストに応じてデータをクライアントへ返すという動きになっている．

WebSocket によるデータ提供機能は，最初にクライアントとコネクションを結べば，ユーザの検出等のイベントが発生した場合にクライアントへ通知をおこなう．これによりクライアント側でユーザ検出状況等を把握することができるため，HTTP リクエストによるポーリングが不要となる．また，ユーザデータ等のコンテキストを Kinect センサデータ提供機能側と共有化しているため，通知内容のデータを利用して Kinect センサデータ提供機能へリクエストを出すことが可能となっている．そのため，データが必要な場合のみに HTTP リク

† 出典 http://www.json.org/json-ja.html

エストを出すことができるようになり，トラフィックを抑えることができる．

複数 Kinect 連携機能は Kinect センサデータ提供機能への拡張機能という位置付けとなっており，クライアントからの要求があれば呼び出されるようになっている．本システムの構成を図 3-5 に示す．

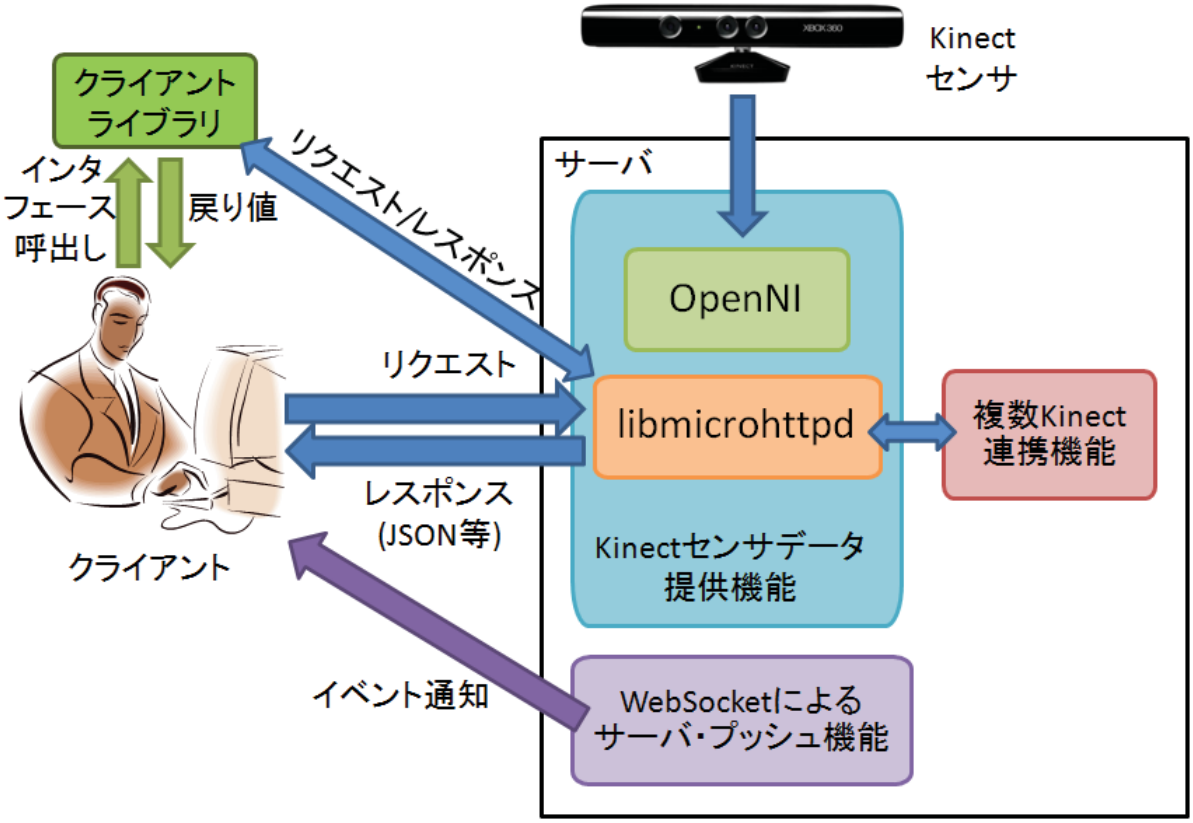



図 3-5 システム構成

# 第4章　複数 Kinect 連携機能

本章では，筆者が開発を担当した複数 Kinect 連携機能について述べる．システムに対する本機能の位置付けおよびデータ連携方針について述べる．その後，今回データ連携処理に用いるライブラリについて説明をおこない，データ連携手順，および本機能の構成について説明する．そして，検証作業について説明をおこなう．

## 4.1　本機能の意義

本機能は，Kinect センサデータ提供機能への付加機能として提供されるものである．データの取得可能範囲を拡張することが主な目的である．単に取得するデータ範囲を広げるだけならば，Kinect センサごとの ID を含めたデータを提供するだけで済む．しかしながら，クライアント側で複数の Kinect センサデータを取得できたとしても扱いにくい．また，顧客からの要望として，クライアントからは複数台の Kinect センサのデータを扱うことを意識させずに利用できるようにすることが挙げられている．そのため，各 Kinect センサから取得したそれぞれのデータをあたかも一台の Kinect センサからのデータとして提供することで，クライアント側では Kinect センサの台数に関係なく同じように扱うことができるようになる．1 つのデータとして提供することにより，クライアント側でデータごとに処理をする必要がなくなり Kinect センサの接続台数が増えてもクライアント側での手間は変わらなくなる．そこで，複数の Kinect センサから取得したデータを連携させたデータを提供することとなった．

なお, 今回は一部のデータのみを連携させ, プロトタイプという形で提供するものである．顧客からも，限定的な状況において一部のデータのみの連携でも構わないと言われている．本プロジェクトでは実現させないが，最終的には Kinect センサデータ提供機能に用意されている機能と同様の機能を本機能でも提供できるようデータ連携をすることが望ましい．本プロトタイプは最終目標への道のりの第一歩として開発をおこなった．

## 4.2　データ連携案

本機能の開発にあたり，データの連携方法について案を出した．案としては大きく分けて次の 2 つ挙がった．

第 1 案として，Kinect センサを固定し，深度データを連携させるという案が挙がった．Kinect センサを固定するため, 最初に位置合わせをおこない座標のずれを補正する変換行列を生成し，データ取得ごとに生成した変換行列によって座標変換をおこなうだけで済む．

次に Kinect センサを固定しない場合の深度データの連携方法を第 2 案として調査をおこなった．大まかな手法としては Kinect センサを固定する場合と同じであるが，データ取得ごとに位置合わせからおこなう必要がある．固定する場合であれば，設置の手間として位置合わせをユーザの補助を伴った半自動でおこなえばよいが，固定されていない場合に毎回ユーザの補助を必要とする位置合わせをおこなうわけにはいかない．自動で位置合わせをおこなうことさえできれば，Kinect センサを固定するといった制約を取り払うことができる．

どちらの案も基準となる Kinect センサデータの座標に対して，他の Kinect センサデータの座標を変換し位置合わせをおこなうといったものである．これにより，全ての Kinect センサデータが基準の Kinect センサデータであるかのように扱える．今回，座標変換により，それぞれのデータを共通の座標系とする位置合わせをもってデータ連携とする．

今回は，プロトタイプとして開発をおこなうため，より自由度の高い Kinect センサを固定しない場合のデータ連携をおこなうべきだと考え，取得したデータごとに位置合わせをおこなう方法を採ることとした．

## 4.3 PCL

PCL(Point Cloud Library)[12]とは 3 次元点群データを扱うためのライブラリである．3 次元点群は 3 次元座標の集合のことで，Kinect センサの距離画像センサによって取得できるデータである．もともと PCL はロボット分野の Robot Vision 向けのものであり，Kinect センサの登場以前は 2.2 節でも述べた通り，高価な機器でしか 3 次元点群データを取得できなかった．安価な Kinect センサの登場により，3 次元点群データの処理を集めたライブラリである PCL に注目が集まり，意欲的に開発が進められている．実際に Kinect センサから取得した 3 次元点群データを図 4-1 に示す．

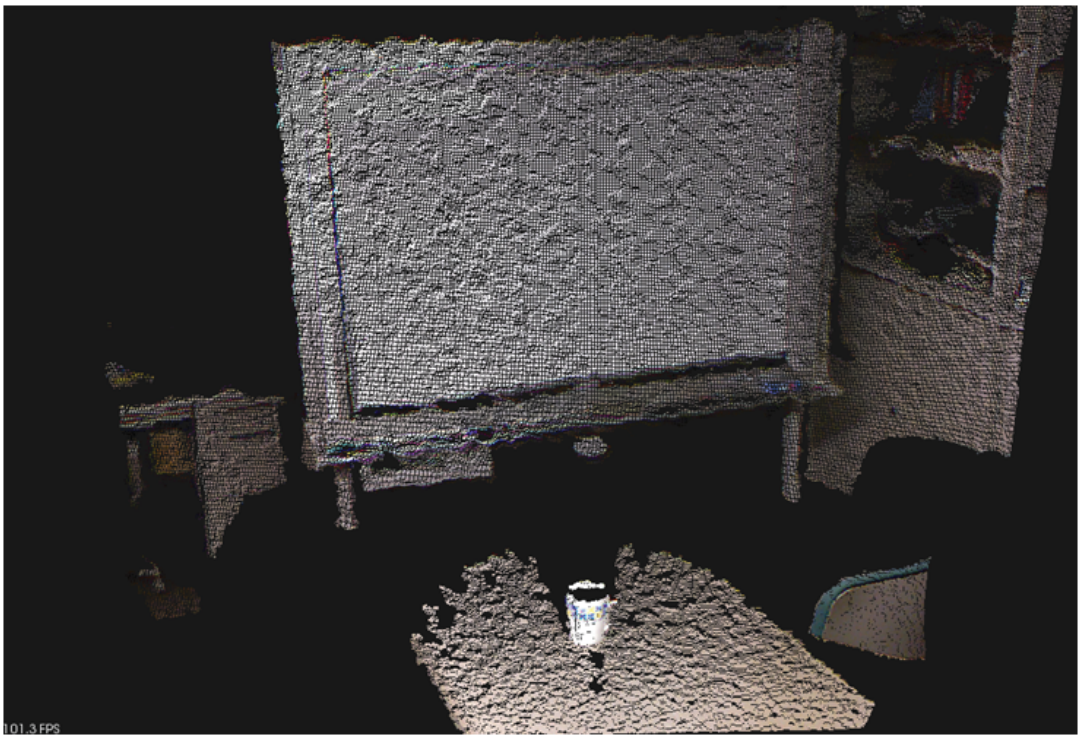



**図 4-1 3 次元点群データ**

Kinect センサ等の距離画像センサから取得した 3 次元点群データと RGB データは，そのセンサの位置・姿勢から見えている範囲の 3D 情報しかない 2.5 次元のデータである．図 4-1 のマグカップの後ろ側等のように，ものによって遮蔽された部分の 3D 情報は取得できない．しかし，2.5 次元であっても 3 次元空間ではあることには違いなく，様々な位置・姿勢からの 2.5 次元のデータを集めることで 3D データとなる．

PCL には OpenNI Grabber Framework と呼ばれる汎用の深度センサデータ取得用のフレ

ームワークがあり，OpenNI に対応しているデバイスであれば OpenNI Grabber 経由でデータを取得することができる．そのため，3 次元点群データの取得から処理まで一括して PCL で扱うことが可能である．

## 4.4　　データ連携手順

データ連携の手順は次のような流れになっている．

① 3 次元点群データを各 Kinect センサから取得する
② それぞれの 3 次元点群データから特徴的な点であるキーポイントを推定する
③ それぞれのキーポイントごとに特徴量を計算する
④ 3 次元点群データごとに特徴量と座標位置の類似度に基づき，対応点を推定する
⑤ 不良対応部分を位置合わせ処理から取り除く
⑥ 残った良い対応部分のデータからずれを推定し，座標変換をおこなう
⑦ それぞれのデータをマージする

これら処理を 1 セットとして，データが更新されるごとに同様の処理をおこなう必要がある[13][14]．この一連の流れを図 4-2 に示す．

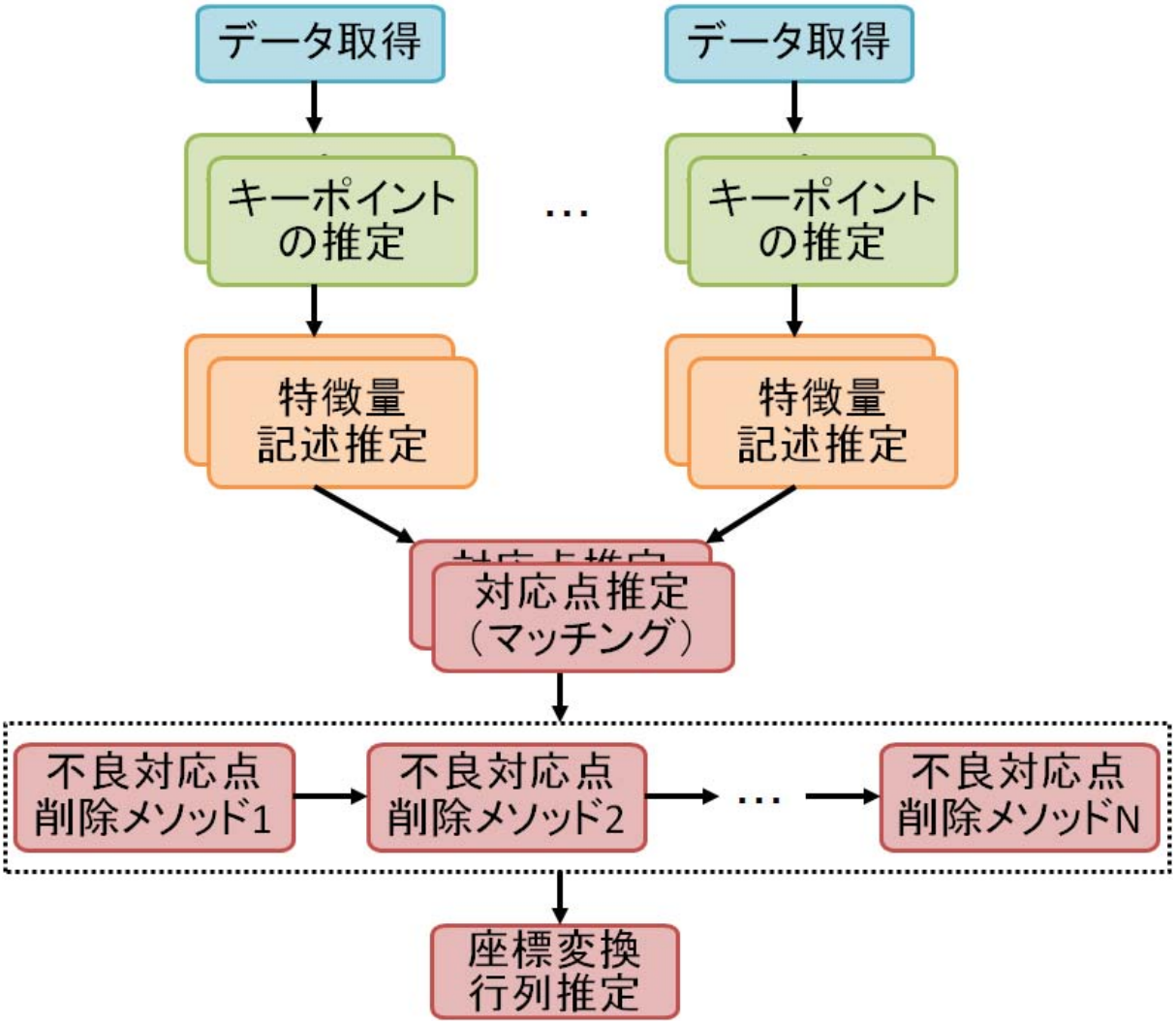



図 4-2 データ連携手順

変換行列を用いて座標変換をおこなった点群データは，位置合わせの対象とした点群データとのマージをおこなうことでデータの連携は完了する．しかしながら，単純にデータ同士マージしただけでは，重なり合う点が発生し，重複する無駄な点により全体のデータ量が増大してしまう．そのため，データのマージ後にダウンサンプリングをおこなうことで重複した点がある部分をなくし，適切なデータ量にした後のデータを提供することが望ましい．

### 4.4.1 特徴量を使った位置合わせ

先に示したデータ連携手順の各段階において，処理方法がいくつかある．今回は，PCL に用意されていたものの内，次のものを使用した．

- キーポイント
  - SIFT
  - コーナー検出法
    - HARRIS
      - HARRIS
      - TOMASI
      - NOBLE
      - LOWE
    - CURVATURE（曲率）
- 特徴量記述
  - PFH
  - FPFH
  - SHOT(RGB)
  - PFH(RGB)

それぞれの手法についての説明は以下の通りである．

- SIFT

SIFT(Scale-Invariant Feature Transform)は，特徴点であるキーポイントの検出と特徴量の記述をおこなうアルゴリズムである[15]．検出したキーポイントに対して，回転やスケール変化，照明変化等に強い特徴量を記述するため画像のマッチングや物体認識，検出に用いられている．

- コーナー検出

コーナー検出法として，5 つの検出手法を用いている．ここでいうコーナーとは，エッジの交わる点と表現でき，また，ある局所近傍で方向の異なる 2 つの特徴的な点ともいえる．コーナーは特徴点の一種であり，これをキーポイントとしている．

- PFH 検出

PFH(Point Feature Histograms)検出[16]は，推定された法線方向間の全ての相互作用から抽出された法線のばらつきを可能な限り捉えようとする手法である．計算量は，与えられる点群データの各点ごとの近くにある点の数によるため，点が密集している部分の PFH 検出は，リアルタイム処理における主要なボトルネックの一つとなる．

- FPFH 検出

FPFH(Fast Point Feature Histograms)検出[17]は，PFH 検出と異なり，近くの点を全て捉えず，対象の点の周りだけで PFH 検出をおこない，それぞれを統合することで高速化を図っている手法である．

- SHOT(RGB)検出

SHOT(Signature of Histograms of Orientations)検出[18]は，3 次元座標と比較し，順序に特徴のある構造のヒストグラムを利用した手法である．今回は，SHOT 検出に RGB 情報を含んだ点と法線を含んだ点群データを用いる．

- PFH(RGB)検出

PFH(RGB)検出は，RGB 情報を含んだ点群データに用いる PFH 検出である．

### 4.4.2 ICP

ICP(Iterative Closest Point)[19]は，入力されたデータの形状を変えずに位置合わせをおこなう剛体位置合わせ手法と呼ばれる代表的な手法である．ICP では 2 つの 3 次元点群の最近傍点を対応点とし，対応点の間の距離を最小にするような変換行列を求め，この変換を一方の 3 次元点群でおこなう．この処理は，誤差が設定した条件を下回るまで繰り返され，位置合わせがおこなわれる．この処理は 3 次元点群の全ての点を用いておこなわれる．具体的には，図 4-3 に示すように 2 つの点群 A，B があるとき，点群 B の各点について，点群 A で最も近い点を対応点とし，各対応点間の距離の 2 乗和が最小となる値を求め，その値を用いて全ての点群を修正するといったものである．反復計算により誤差関数が減少するため，局所解へ単調に収束することが保証されている．

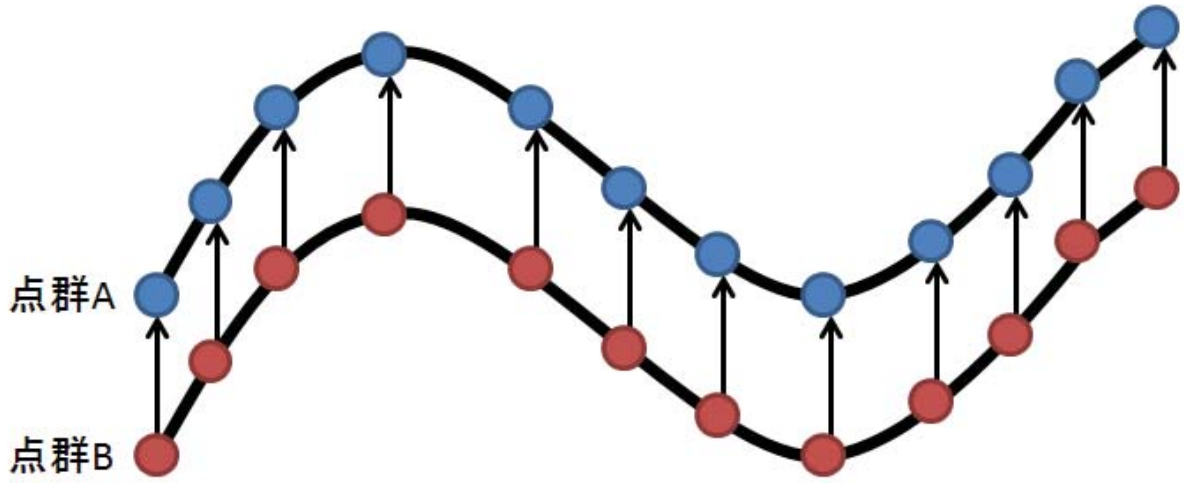



図 4-3 ICP アルゴリズム

ICP では，初期状態において点群データ同士が大まかに位置合わせされていることを仮定している．そのため，最初の大まかな位置合わせとして 4.4.1 項で述べた特徴量を使った位置合わせをおこなった後に，ICP による位置合わせをおこなう必要がある．ただし今回のデータ連携に際しては，データ取得範囲を拡張することが目的であり，位置合わせをおこなうには Kinect センサ同士がある程度同じ領域をカバーする必要があるため，設置状況によっ

ては初期状態における条件は満たしている場合もあるだろう．しかしながら，最近傍点を全ての点に対して計算するため，対応点探索の計算量が問題となる．

## 4.5 出力形式

クライアントへ提供するデータ形式としては，Kinect センサデータ提供機能でクライアントへ提供するデータ形式として主に利用している JSON 形式で提供することとした．提供するデータの JSON 形式を表 4-1 に示す．

3 次元点群データのみの JSON データと，3 次元点群データに加え各点の RGB 情報を付加した JSON データの 2 種類がある．これらの違いは withColor キーの値により判別できるようになっている．PointCloud は配列となっており，1 点に座標データと RGB データを記述する形となっている．

表 4-1 イメージ情報付き 3 次元点群 JSON 形式

| Key | | | | 型 | 説明 |
|---|---|---|---|---|---|
| Kineco | Version | | | 数値 | バージョン情報 |
| | config | points | | 数値 | 3 次元点群の点数 |
| | | withColor | | 真偽値 | 色データの有無 |
| | PointCloud | coordinate | x | 数値 | 点の X 座標 |
| | | | y | 数値 | 点の Y 座標 |
| | | | z | 数値 | 点の Z 座標 |
| | | Color | r | 数値 | 点の色情報 R |
| | | | g | 数値 | 点の色情報 G |
| | | | b | 数値 | 点の色情報 B |

## 4.6 本機能の構成

本機能は大きく分けて，2 つの部分に分かれている．図 4-4 に示すような Kinect センサからデータを取得しメインサーバへデータを送信する Kinect センサデータ送信機能部分，各 Kinect センサから取得したデータを連携させるデータ連携機能部分からなる．

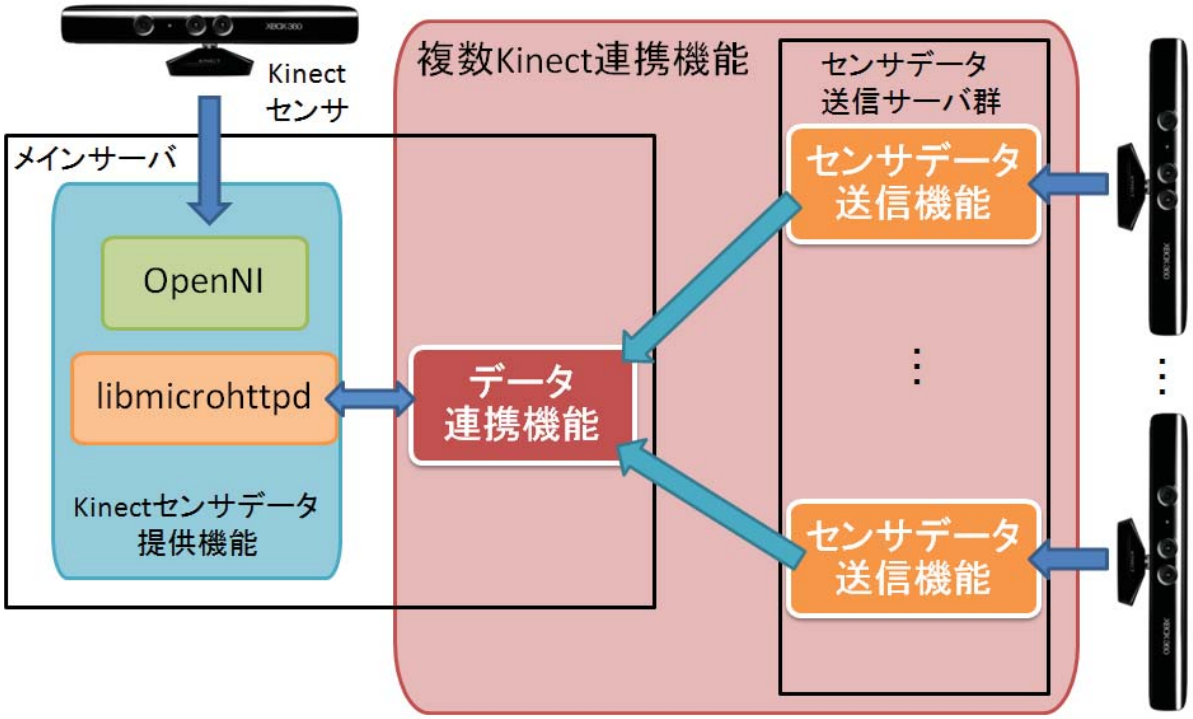



図 4-4 複数 Kinect 連携機能の構成

センサデータ送信サーバにはそれぞれに 1 台ずつ Kinect センサが接続されており，取得したデータをメインサーバに送信するようになっている．そして，メインサーバでそれぞれのデータ連携をする処理をおこない，JSON データを生成する．クライアントからリクエストがあった場合，Kinect センサデータ提供機能が本機能にアクセスし，JSON データを提供するようになっている．

Kinect センサからデータを取得するマシンとデータ連携をおこなうマシンを分けた理由としては，一台のマシンで本機能を網羅しようとするといくつかの問題があったためである．具体的には，マシンと離れた場所に Kinect センサを設置することが困難であったこと，接続する Kinect センサの台数が多数になる場合に問題が発生するといったものである．問題点は次の通りである．

- 設置場所に関する問題
  - Kinect センサとマシンとを接続する USB ケーブルの長さの問題
  - Kinect センサ用の電源数の問題
- 多数の Kinect センサが接続された場合の問題
  - マシン側の USB ポート数の問題
  - システム全体に対する負荷問題

Kinect センサを接続するマシンを分けることで，先に挙げた問題はある程度緩和されると考える．ある程度広い部屋で使用する場合，部屋の端と端といったような離れた場所に，それぞれ Kinect センサを接続したラップトップを配置し，無線ネットワークを用いてデータをメインサーバに送信させるといった利用方法も可能となる．

### 4.6.1 センサデータ送信機能

本機能は，独立したそれぞれのマシンで個々に動作させるものである．実行の際には，メインサーバの IP アドレスおよびデータ連携機能で待ち受けているポート番号が必要となる．また，実行時に取得する点群データについても次の 6 種類から選択できるようにしている．

- 低解像度
  - RGB データ有り
  - RGB データ無し
- 中解像度
  - RGB データ有り
  - RGB データ無し
- 高解像度
  - RGB データ有り
  - RGB データ無し

なお，Kinect センサから取得したデータは，送信前に不要な部分のデータを取り除くことで，転送データ量を削り，TCP のストリーム通信を用いてメインサーバへと送信するようにしている．

### 4.6.2 データ連携機能

本機能は，センサデータ送信サーバから転送される 3 次元点群データを受け取り，それぞれのデータの位置合わせをおこない，JSON 形式として生成するものである．

本機能では，4.4.1 項で示した 6 つのキーポイント推定方法と 4 つの特徴量記述推定手法を用意している．また，特徴量を使った位置合わせだけでは精度が低いため，特徴量を使った位置合わせの後に，4.4.2 項で紹介した ICP による位置合わせをおこなうことで，より高精度な位置合わせを実現している．

実行時には接続する Kinect センサの台数を指定するようになっており，指定した台数のセンサデータ送信サーバから接続があるまで位置合わせはおこなわれない．

また，前後に接続したセンサデータ送信サーバからのデータとある程度同じ領域を含んでいる必要があるという制限がある．これは位置合わせの際に，ある程度同じ領域をカバーしているデータ同士でなければ位置合わせができないためである．

## 4.7 検証

本節では，2 台の Kinect センサから取得した 3 次元点群データを用い，今回用意した手法ごとに検証した結果について述べる．検証環境を表 4-2 に，検証に用いた入力点群データを図 4-5 に示す．

**表 4-2 検証環境**

| | |
|---|---|
| OS | Ubuntu　10.04　LTS |
| プロセッサ | Intel　Core 2 Duo　U9400　1.40GHz |
| メモリ | DDR2　SDRAM　2GB |
| ライブラリ | PCL　1.3.1 |

変換元点群データ　　変換先点群データ

図 4-5 検証用入力点群データ

入力点群データは 4.6.1 項で説明したセンサデータ送信機能によって取得した高解像度，RGB データ有りのデータを用いた．変換元点群データの点数は 172805 点であり，変換先点群データの点数は 120463 点である．

4.4.1 項で挙げた 6 つのキーポイント推定手法ごとの抽出点数および抽出時間の結果をそれぞれ表 4-3，表 4-4，表 4-5，表 4-6，表 4-7，表 4-8 に示す．

表 4-3 キーポイント推定検証結果(SIFT)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント抽出数 | 911 点 | 643 点 |
| キーポイント抽出時間 | 24.22 sec | 18.23 sec |

表 4-4 キーポイント推定検証結果(HARRIS)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント抽出数 | 33938 点 | 20089 点 |
| キーポイント抽出時間 | 28.73 sec | 20.26 sec |

表 4-5 キーポイント推定検証結果(TOMASI)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント抽出数 | 123439 点 | 78272 点 |
| キーポイント抽出時間 | 34.12 sec | 24.39 sec |

表 4-6 キーポイント推定検証結果(NOBLE)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント抽出数 | 44088 点 | 12324 点 |
| キーポイント抽出時間 | 28.14 sec | 20.24 sec |

表 4-7 キーポイント推定検証結果(LOWE)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント抽出数 | 45854 点 | 13053 点 |
| キーポイント抽出時間 | 28.36 sec | 20.28 sec |

表 4-8 キーポイント推定検証結果(CURVATURE)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント抽出数 | 123919 点 | 78932 点 |
| キーポイント抽出時間 | 28.07 sec | 20.11 sec |

検出手法ごとにキーポイントの抽出点数が大きく異なった．抽出時間については，あまり大きな違いはないが, SIFT はキーポイント抽出点数が少ない割に抽出時間がかかっている．これは 4.4.1 項で紹介したように，変化に強いキーポイントを抽出しているためである．キーポイント数が多いと後々の対応点推定において処理時間を有することにもなるため，今回の手法の中では SIFT を用いることが一番良いと考えられる．

次に 4.4.1 項で挙げた 4 つの特徴量記述手法ごとの検証結果を表 4-9，表 4-10，表 4-11，表 4-12 にそれぞれ示す．なお，キーポイント検出は全て SIFT でおこなっており，キーポイント数は全て同じである．

表 4-9 特徴量記述推定検証結果(PFH)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント数 | 911 点 | 643 点 |
| 特徴量抽出時間 | 14.12 sec | 10.07 sec |
| 対応点推定時間 | 0.12 sec | 0.09 sec |
| 不良対応点削除時間 | 811.53 sec | |
| 変換行列推定時間 | 0.5 sec | |

表 4-10 特徴量記述推定検証結果(FPFH)

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント数 | 911 点 | 643 点 |
| 特徴量抽出時間 | 81.69 sec | 66.15 sec |
| 対応点推定時間 | 0.08 sec | 0.07 sec |
| 不良対応点削除時間 | 310.74 sec | |
| 変換行列推定時間 | 0.5 sec | |

表 4-11 特徴量記述推定検証結果(SHOT(RGB))

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント数 | 911 点 | 643 点 |
| 特徴量抽出時間 | 13.19 sec | 9.44 sec |
| 対応点推定時間 | 1.48 sec | 1.3 sec |
| 不良対応点削除時間 | 214.52 sec | |
| 変換行列推定時間 | 0.5 sec | |

表 4-12 特徴量記述推定検証結果(PFH(RGB))

| | 変換元点群データ | 変換先点群データ |
|---|---|---|
| キーポイント数 | 911 点 | 643 点 |
| 特徴量抽出時間 | 14.62 sec | 10.51 sec |
| 対応点推定時間 | 0.46 sec | 0.46 sec |
| 不良対応点削除時間 | 307.26 sec | |
| 変換行列推定時間 | 0.5 sec | |

特徴量記述推定は，位置合わせの精度が大きく影響する部分である．データおよびキーポイントはいずれも同一のものを用いているため，比較が可能である．まず，特徴量抽出推定時間をみてみると FPFH が他と比べて処理に時間がかかっている．他の 3 つはほぼ同じ時間で処理を終えている．しかしながら，対応点推定時間をみると先程と尺度が異なるが，FPFH の処理時間が最も短い．不良対応点削除時間をみると FPFH と PFH(RGB)は同程度の処理時間であり，PFH が最も長く，SHOT(RGB)が最も短い結果となった．今回 RGB データを含んだ点群データを用いたが，SHOT(RGB)と PFH(RGB)は RGB データを含んだ点群データにより推定をおこなっている．重要となる位置合わせの精度については PFH(RGB)が最も良い結果となった．

続いて，位置合わせの精度について一例を挙げて述べる．キーポイント推定に SIFT，特徴量記述推定に FPH(RGB)を用いた対応点推定結果を図 4-6 に，座標変換および JSON データ生成時間の結果を表 4-13 に示す．

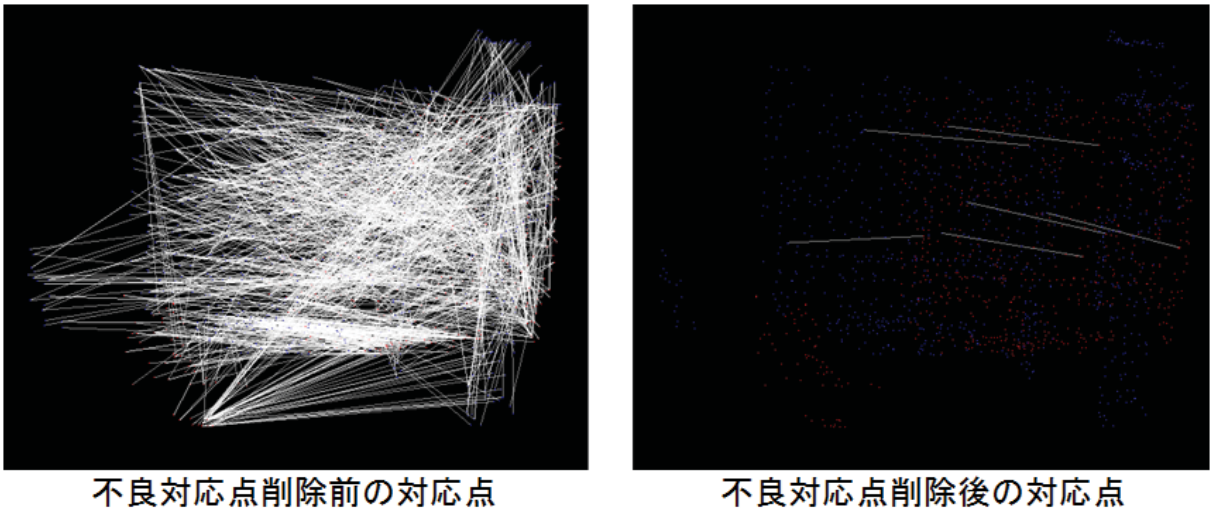



図 4-6 対応点推定結果

表 4-13 特徴量を使った位置合わせにおける座標変換時間・JSON データ生成時間

| | |
|---|---|
| 座標変換時間 | 0.51 sec |
| 点群データのマージ後の点数 | 293268 点 |
| JSON データ生成時間 | 1.86 sec |

不良対応点削除後の対応点を基に変換行列を求めるが，まだ対応点には誤りが含まれているため，完全に一致するようにデータを連携させることは困難である．この例では誤りは少ないため大幅にずれることはないが，対応点が悪いと変換行列が誤ったものとなってしまい，位置合わせの際にずれてしまうことになる．時間はかかってしまうが，特徴量を使った位置合わせをおこなった後に，ICP による位置合わせをおこなうことで，高精度な位置合わせを実現することができた．キーポイント推定に SIFT，特徴量記述推定に PFH(RGB)を用いた特徴量を使った位置合わせと ICP による位置合わせの結果を図 4-7 に示す．


図 4-7 位置合わせ結果

正確な位置合わせをおこなうためには，まず特徴量をつかった位置合わせによりおおまかな位置合わせをおこない，その後 ICP による位置合わせが必要となる．しかし，ICP による位置合わせは点群内の点数によっては非常に計算時間を要するため，あまり実用的ではなさそうである．

# 第5章　　プロジェクト体制

本章では，本プロジェクトにおけるプロジェクト管理およびプロジェクト運用について述べる．

## 5.1　　プロジェクト管理

本プロジェクトでは，プロジェクト管理ツールの Redmine[20]を利用した．また，成果物の共有や管理にはオンラインストレージサービスの Dropbox‡やバージョン管理ツールの Git§を利用した．未完成のドキュメント等は Dropbox で管理し，完成したドキュメントを Redmine 上で管理するようにした．

進捗状況や成果物を Redmine 中心に管理することで，プロジェクトメンバ間での情報共有が容易になることを狙った．なお，Redmine については顧客からも参照できるようにしたため，プロジェクトの状況を確認することが可能である．

### 5.1.1　　進捗管理

進捗管理は Redmine 上でおこなった. チケットと呼ばれる単位で作業管理ができるため，開発する機能をより細かい単位にブレイクダウンし，チケットとして管理することで機能全体の見通しが容易になった．また，各人の進捗状況を把握することが容易にでき，プロジェクト全体の進捗としての確認も可能である．

チケットには作業分類，作業内容，予定工数等を登録できる. 実際のチケットを図 5-1 に，チケット一覧を図 5-2 に示す.


図 5-1 チケットの例

‡ http://www.dropbox.com/
§ http://git-scm.com/


図 5-2 チケット一覧確認画面

### 5.1.2 成果物管理

本プロジェクトにおける成果物であるドキュメント類とソースコードは全て Redmine 上から参照できるようになっている．変更が多いソースコードに関しては，バージョン管理ツールの一つである Git を用いて管理をおこなった．Git による変更登録の際に，Redmine のチケットと紐付けることができるため，Redmine での進捗確認の際にソースコードの確認をすることも可能となっている．また，それぞれの変更による差分も確認することができる．Redmine 上でのバージョン管理画面を図 5-3 に，差分確認画面を図 5-4 に示す．

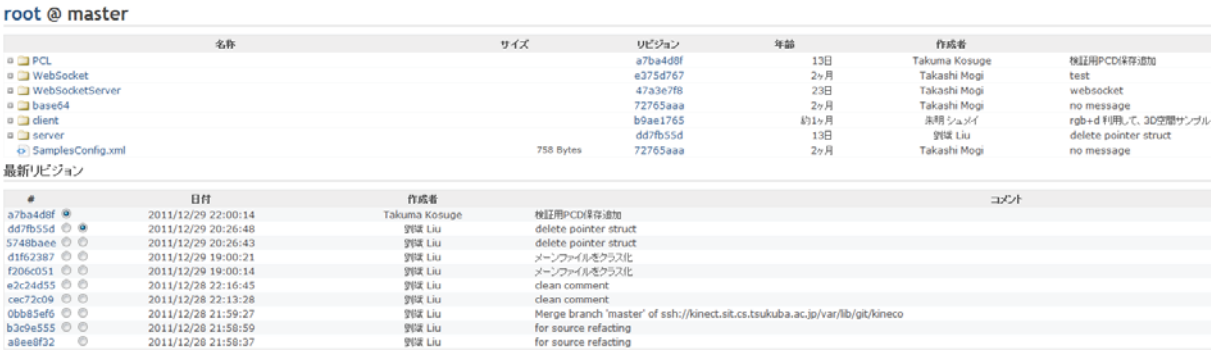



図 5-3 バージョン管理画面

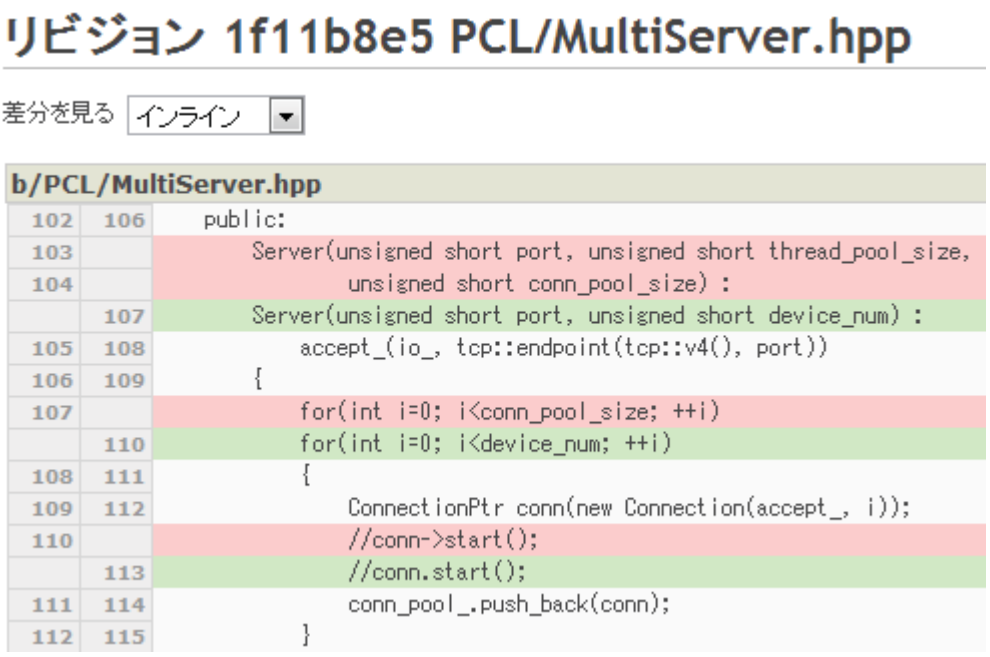



図 5-4 差分確認画面

## 5.2 プロジェクト運用

本節では，本プロジェクトにおける開発手法およびコミュニケーション方法について述べる．

### 5.2.1 開発手法

本プロジェクトで採用した開発手法について述べる．当初，図 5-5 に示すような反復型開発を採用した．理由としては，顧客の要求が大まかなものであったため，機能ごとにプロトタイプを作り，実際に動作するものを顧客に確認してもらうことで，一歩一歩堅実に開発を進めていく狙いがあったからである．また，プロトタイピング手法を導入することにより，当初の実際に動作するプロトタイプを作っていくことで仕様の明確化や技術的な問題を洗い出していく形を採った．

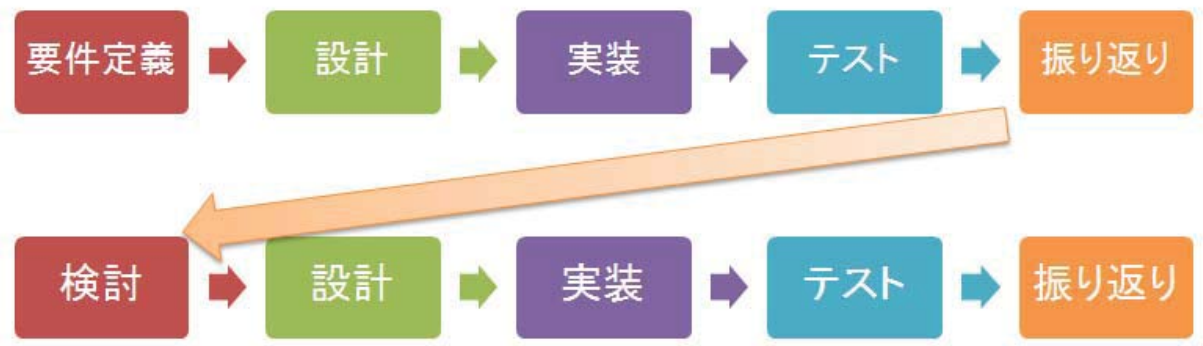



図 5-5 反復型開発手法

しかしながら，各機能が小さく開発効率が悪かったため，より大きな機能単位で作業を割り振ることとなった．割り振る機能が大きくなりすぎたため反復型開発をすることが困難となり，チームとしての反復型開発は取りやめとなった． しかしながら，当初の実際に動作するプロトタイプを作っていくという考えは継承して開発をおこなっていった．

### 5.2.2 コミュニケーション方法

本プロジェクトにおけるコミュニケーション方法を以下のように定めた．

- 週に 2，3 回のミーティングをおこなう
  - 週初めにその週の作業を設定する
  - 週終りにその週の作業の振り返りをおこなう
- コアタイムを設け，疑問点等をその場で相談できるようにする

# 第6章　プロジェクトの振り返り

本章では，プロジェクト全体を振り返り，良かった点，反省点について考察し，今後の課題について述べる．

## 6.1　要件分析の振り返り

システム全体としてのイメージをチームとして早々に固めることができたため，必要となりそうな技術調査や，プロトタイピングへと早い段階で移行することができた．しかし，サンプルクライアントの作成という要望にとらわれ，実際はシステムを利用したクライアントの実装例を示すだけでよかったところに，システムのクライアントでの具体的な利用シーンについての検討にかなりの時間を割いてしまい，時間を浪費してしまったことがあった．また，顧客との打ち合わせが少なかったことや，打ち合わせの際のゴールをしっかりと決めなかったため，具体的な要件を確認できないまま進めてしまった．顧客の求めていることをしっかりと把握するためにも適宜確認をするべきであった．

## 6.2　基本機能のプロトタイプ開発の振り返り

プロトタイピングによって曖昧だった仕様や足りない機能を掴むことができた．プロトタイプを実際に顧客に確認してもらいながら進めていくことができた．しかしながら，プロトタイピングをおこなう機能単位が小さすぎたため，作業分担がうまくできなかった．分担し易いある程度の大きさの機能で作業を割り振るべきだった．

## 6.3　担当機能開発の振り返り

11 月に入ってから調査を始めた機能であり，技術的な不安や問題点も多くあったため，動作するものができない可能性があった．しかしながら，なんとか動作するものを作ることができた．また，開発を楽しみながらできたことは良かったと感じた．しかし，時間が少なかったため，かなり焦って調査をおこない，見落としていた点や勘違いしていたことがいくつかあった．また，クラス設計をおこなうことができなかったことが悔やまれる．

今回のように，時間がないタイミングにもかかわらず急に予定になかった機能や未調査の技術をつかった開発をおこなうことは非常に危険である．スケジュールや開発する機能の規模を見極めるためにも，あらかじめ調査活動をしておくべきだと痛感した．

## 6.4　プロジェクト管理の振り返り

当初設定した開発の予定と 10 月末までの実績を図 6-1 に示す．

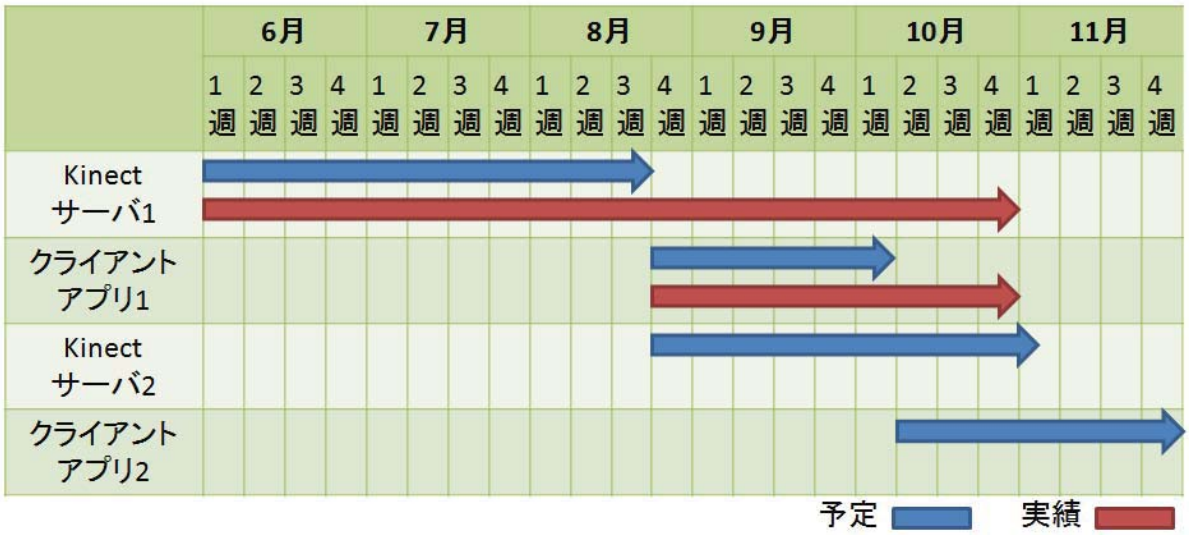



図 6-1 10 月末までの予定と実績

図 6-1 の Kinect サーバ 1 は基本機能のプロトタイプ開発に相当する．途中までは順調に見えたが，夏季休暇に入ってから先行きが不明瞭になってきた．進捗管理用に 5 月末に Redmine の導入をおこない，利用され始めたところで夏季休暇に入った．夏季休暇中は各人の予定が合わず，直接進捗状況を報告できる機会が減少するため，それぞれの進捗状況を確認し合えるように Redmine 利用マニュアルを作成した．しかしながら，既に夏季休暇に入っていたこともあり，利用マニュアルのレビューをおこなえなかっただけでなく，マニュアルに沿った報告を周知させることができなかった．統一された報告体制がなく，夏季休暇に入ったこともあり，Redmine による進捗報告を全員が継続しておこなうというところまで軌道にのせることができなかった． また，コアタイムを設けたのにもかかわらずチームの集まりが悪く，チームが一同に会すことは稀であった．

これらが主な原因となり，Kinect サーバ 1 の予定と実績が大幅にずれる事態につながり，10 月末に新たなスケジュールを立てることとなった．新たなスケジュールでは，大きな機能単位で作業を割り振ることになり，各々作業を進めていくことになった．結果として作業効率は向上したが，進捗報告に関しては対策を打たないままであったため，各人がどういった状況にあるのか余計に把握しづらくなった．

本プロジェクトにおけるプロジェクト管理の在り様は，プロジェクト立ち上げに際して，プロジェクトの体制についてよく検討しないまま進めてきた結果によるものだろう．やはり，プロジェクト立ち上げの際にしっかりと体制作りをしておく必要があるのだと再認識させられた．

## 6.5 今後の課題

複数 Kinect 連携機能はまだプロトタイピング段階であるため，課題としては拡張性の高い設計をおこない，それに沿った実装をおこなうことがまず挙げられる．また，検証作業によって点群データの位置合わせには，計算時間が非常にかかることが分かったが，計算時間だけでなく，位置合わせの精度についても詳しく検証する必要がある．

しかし，位置合わせの精度を求めると計算時間に響いてきてしまう．もちろんリアルタイムに近い計算時間にできればよいが，データ取得ごとに位置合わせをおこなうとなると困難

である．今回は，Kinect センサが移動できることを想定して開発をおこなったが，Kinect センサを固定して利用する場合では，最初の一度だけ位置合わせをするだけで毎回の位置合わせは必要なくなる．変換行列による座標変換をおこなうだけで済むため，リアルタイムに近い時間でデータ連携をおこなうことができるようになると考えられる．

今回の検証作業では 2 台の Kinect センサによるデータ連携しかおこなっていないため，3 つ以上のデータ連携についても検証をおこなう必要がある．また，今回 6 つのキーポイント推定手法と 4 つの特徴量記述推定手法を利用したが, PCL にはまだ他の手法が用意されている．それらの手法についても検証をおこない，将来のデータ連携機能に向けて吟味する必要があるだろう．

システム全体としては，品質の確保や評価をおこなうことができなかったため，品質の向上や評価，機能の充実が今後の課題として挙げられる．

# 第7章　おわりに

研究開発プロジェクトにおいて，「Kinect サーバおよびサンプルクライアントの構築」というテーマのもと，Kinect サーバシステムおよびサンプルクライアントの開発をおこなった．

顧客の要望からシステム案を検討し，プロトタイプを開発しながら機能の提案をおこなった．システムの根幹となる機能に目処がついたところで，各機能のプロトタイプからのリファクタリングおよびシステムへの付加機能の開発に着手した．

開発にあたって筆者は，複数 Kinect 連携機能を担当し，データ連携に関する調査，プロトタイプの開発および検証をおこなった. 検証の結果, ある程度のデータの連携はできたが, Kinect センサが移動できることを想定して開発をおこなったため, 処理時間およびデータ連携の精度が問題として挙がった. Kinect センサを固定して利用する場合の処理を導入すれば, データ取得ごとの処理が減るため，処理時間の問題に関しては改善されると考えられる．

今後の課題として, Kinect センサが固定されている場合のデータ連携をおこなうことでデータごとの提供までの時間を短縮することや, 3 つ以上のデータ連携の検証等が挙げられる．また，システム全体としての評価や品質の確保も挙げられる．

本報告書が，Kinect センサを利用するシステム開発の参考となれば幸いである．

# 謝辞

本プロジェクトの委託元教員である高橋伸准教授には，テーマの提供およびプロジェクト全体を通して多くの学ぶ機会を与えて下さり，深く感謝致します．

指導教員の田中二郎教授には，2 年間に渡り丁寧かつ熱心なご指導ご鞭撻を賜りました．大変お世話になりました．心より感謝致します．

本報告書執筆にあたり，ご指導頂いた中沢研也教授に感謝致します．また，プロジェクトを進めるにあたって大変有益な知識や技術を授けて頂いた山戸昭三教授，前年度上半期まで担当された駒谷昇一教授，菊池純男教授をはじめとする高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムの教員の方々に深く感謝致します．

本プロジェクトに対し，Kinect センサをご提供して下さいました Microsoft 社に感謝致します．複数 Kinect 連携機能の開発にあたり，アドバイスを頂いた金石煥氏に感謝致します．

また，本プロジェクトのチームメンバである朱明氏，茂木昂士氏，劉斌氏には，プロジェクトを進める上で様々な面でのサポートを頂きました．共にプロジェクトを遂行できたことを深く感謝致します．

最後に，様々な面でご支援を頂きました家族や友人，諸先輩方，大学生活でお世話になった全ての方々に心より感謝致します．

# 参考文献


[1] 小菅拓真 , 坂本勝己 , 若月惣太 , 綿貫理明: “自然な複合現実の実現に向けた環境教育絵本-第 1 回川崎国際環境技術展 2009 への出展-“, 専修ネットワーク&インフォメーション, No.16, pp.19-23, 2010.

[2] Kinect - Xbox.com. [Online]. http://www.xbox.com/ja-JP/kinect

[3] 中村薫: 『KINECT センサープログラミング』, (編) 斉藤和邦, （株）秀和システム, 2011.

[4] 谷尻豊寿: 『身体の動きがコントローラ　C++で Kinect プログラミング　KINECT センサー　画像処理プログラミング』, (編) 石塚勝敏, （株）カットシステム, 2011.

[5] 西林孝 , 小野憲史: キネクトハッカーズマニュアル. (編) 榎本統太, （株）ラトルズ, 2011.

[6] Introducing OpenNI. [Online]. http://www.openni.org/

[7] GNU libmicrohttpd a library for creating an embedded HTTP server. [Online]. http://www.gnu.org/software/libmicrohttpd/

[8] JSON. [Online]. http://www.json.org/json-ja.html

[9] The WebSocket API. [Online]. http://www.w3.org/TR/websockets/

[10] BiDirectional or Server-Initiated HTTP (hybi) - Charter. [Online]. https://datatracker.ietf.org/wg/hybi/charter/

[11] RFC 6455 - The WebSocket Protocol. [Online]. http://tools.ietf.org/html/rfc6455

[12] PCL - Point Cloud Library. [Online]. http://pointclouds.org/

[13] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox: “RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments”, Proc. of International Symposium on Experimental Robotics, 2010.

[14] H. Du, P. Henry, X. Ren, M. Cheng, D. B Goldman, S. M. Seitz and D. Fox: “Interactive 3D Modeling of Indoor Environments with a Consumer Depth Camera”, International Conference on Ubiquitous Computing (Ubicomp), 2011.

[15] 藤吉弘亘: Gradient ベースの特徴抽出 - SIFT と HOG -, 情報処理学会 研究報告 CVIM 160, pp. 211-224, 2007.

[16] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz: “Aligning Point Cloud Views using Persistent Feature Histograms”, in Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September, pp.22-26, 2008.

[17] R. B. Rusu, N. Blodow, M. Beet: “Fast Point Feature Histograms (FPFH) for 3D Registration”, The IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May, pp.3212-3217, 2009.

[18] F. Tombari, S. Salti, L. Di Stefano: “Unique Signatures of Histograms for Local Surface Description”, In Proceedings of the 11th European Conference on Computer Vision (ECCV), Heraklion, Greece, September 5-11 2010.

[19] Paul J. Besl, and Neil D. McKay: “A Method for Registration of 3-D Shapes”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No. 2, pp.239-256, 1992.

[20] Redmine.JP. [Online]. http://redmine.jp/

# 付録

- プロジェクト内文書
  - Redmine 利用マニュアル
- 仕様書、基本設計書
  - WebAPI 仕様書
  - WebSocket 通知形式仕様書
  - クライアント基本設計書
- 詳細設計書
  - HTTP サーバ　詳細設計書
  - WebSocket クラス設計書
  - クライアント詳細設計書

# Kinect サーバ

## Redmine 利用マニュアル

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 初版 | 2011/08/08 | Redmine 利用マニュアル草案 | 小菅 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 目次

# 1. 運用プロセス

ここではタスク管理をおこなう単位であるチケットの運用プロセスについて述べる．大まかなチケットの運用プロセスを図 1 に示す．

まず，割り当てられたタスクをチケットとして登録する．チケットとして登録をおこなう者としてはタスク担当者が登録する場合もあれば，違う場合もある．チケットが登録された後に担当するチケットのステータスを「進行中」に変更し，作業を開始する．なお，チケット登録時点で担当者が決められていないタスクをおこなう場合には，チケットの担当者を自身として更新する．作業を終えたら，作業時間を該当タスクに入力する．その時点で作業が完了したときはチケットのステータスを「終了」とする．

以降では各プロセスについて Redmine の具体的な操作方法を示す．

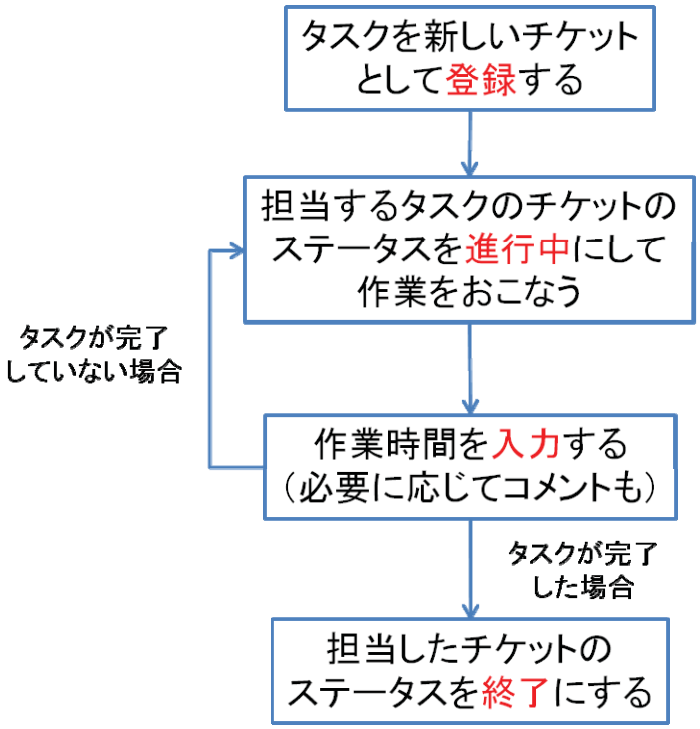



図 1. チケットの運用プロセス

# 2. 各プロセスの説明

## 2.1. タスクを新しいチケットとして登録する

ここでは，チケットを新たに登録する場合についての説明をおこなう．図 2 にチケット登録画面を示す．各項目については以下に詳述する．

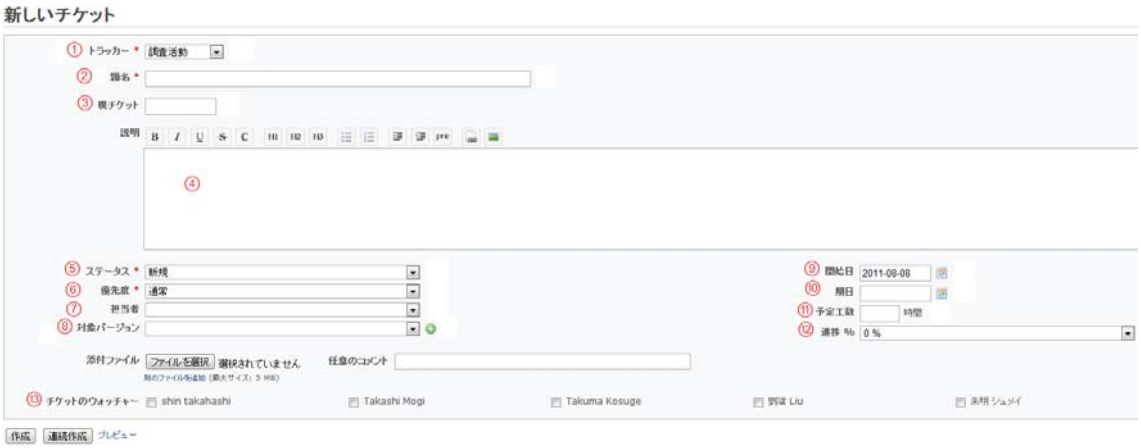



**図 2. チケット登録画面**

① トラッカー

トラッカー(チケットの種類)を入力する．トラッカーの一覧を表 1 に示す．

**表 1. トラッカー一覧**

| トラッカー名 | 意味 |
|---|---|
| 調査活動 | 調査活動に関するチケット |
| 要件定義 | 要件定義工程のチケット |
| 設計 | 設計工程のチケット |
| 実装 | 実装工程のチケット |
| 結合テスト | 結合テスト工程のチケット |
| 総合テスト | 総合テスト工程のチケット |
| ミーティング | ミーティングに関するチケット |
| バグ | バグに関するチケット |
| サポート | プロジェクト運営に関するチケット |

② 題名

チケットの名前を入力する．

③　親チケット

チケット番号を入力することでそのチケットの子チケットとして登録ができる．チケットを階層的に登録することができ，チケットをさらに細かく分ける際に用いる．

④　説明

チケットについての説明について記述する．主に作業内容を記す．

⑤　ステータス

チケットのステータスを入力する．ステータスの一覧を表 2 に示す．ただし、主に利用することになるステータスは「新規」,「進行中」,「終了」の 3 つである．

**表 2. ステータス一覧**

| ステータス名 | 意味 |
|---|---|
| 新規 | 新規にチケットを登録した場合「新規」となる |
| 進行中 | 作業中のチケットは「進行中」とする |
| 解決 | 他のタスクによって該当チケットを<br>補完してしまい必要なくなった場合<br>若しくは<br>トラッカーがバグのチケットにおいて<br>他のバグを修正した際に，このチケットの<br>バグも解決した場合「解決」とする |
| フィードバック | 定義なし |
| 終了 | チケットが完了した場合「終了」とする |
| 却下 | 中止・廃止となり該当チケットを<br>消化する必要がなくなった場合<br>若しくは<br>トラッカーがバグのチケットにおいて<br>修正をおこなわない場合「却下」とする |

⑥　優先度

優先度を入力する．各優先度の一覧を表 3 に示す．

**表 3. 優先度一覧**

| ステータス名 | 意味 |
|---|---|
| 低め | 作業が遅れてもプロジェクトに影響が |

| | |
|---|---|
| | ないものを表す |
| 通常 | 作業が遅れるとプロジェクトに影響が出るものを表す |
| 高め | 作業が遅れるとプロジェクトに大きな影響が出るものを表す |
| 急いで | 急がないとプロジェクトに大きな影響が出るものを表す |
| 今すぐ | 今すぐ作業を行ない，早期に完了しなければならないものを表す |

⑦　担当者

担当者の名前を選択する．タスクを割り当てる人物を選択する．登録時点では割り当てず，作業をおこなう際に担当者として登録する場合には入力しない．

⑧　対象バージョン

対象としているバージョンを選択する．このバージョンとはマイルストーンを表している．バージョンの一覧を表 4 に示す．

**表 4. バージョン一覧**

| バージョン名 | 意味 |
|---|---|
| Kinect サーバ 1 | 反復 1 におけるサーバのチケットはこのバージョンを選択する |
| クライアントアプリ 1 | 反復 1 におけるクライアントのチケットはこのバージョンを選択する |
| Kinect サーバ 2 | 反復 2 におけるサーバのチケットはこのバージョンを選択する |
| クライアントアプリ 2 | 反復 2 におけるクライアントのチケットはこのバージョンを選択する |

⑨　開始日

チケットの開始日を選択する．

⑩　期日

チケットの期日を選択する．

⑪　予定工数

予定している時間を入力する．（単位 hour）

⑫　進捗%
チケットの進捗率を選択する．チケットの進捗率については別途定める必要がある．

⑬　チケットのウォッチャー
チケットを監視する人を選択する．基本的に学生メンバー全員を選択することとする．

## 1.1. 担当するタスクのチケットのステータスを進行中にして作業をおこなう

作業を始めたチケットはステータスを「進行中」とする．ステータスを変更するには，まず「チケット」タブを選択する．このとき、図 3 の画面が表示される．


**図 3. チケット確認画面**

作業をおこなうチケットを右クリックし，図 4 のポップアップを出現させ，「ステータス」から「進行中」を選択することでステータスが「進行中」に更新される．なお，担当者が割り当てられていないチケットを採る場合も同様に「担当者」から自身の名前を選択することでチケットの担当者として登録される．


**図 4. ポップアップ画面**

## 2.2. 作業時間を入力する

作業を終えたら，作業時間の入力をおこなう．入力するためには「チケット」タブを選択し，作業時間の入力をおこなうチケットを右クリックし，2.2 節と同様に図 4 のポップアップを出現させる．ここで，「時間を記録」を選択すると，図 5 の画面が表示される．


図 5. 作業時間の記録画面

ここで，作業した日付，時間，コメント(任意)，活動を入力する．作業時間の入力では，時間単位での入力となるが，「2:00」のように「hour:minute」で入力することも可能である．コメントについては任意ではあるが，どのような作業をおこなったのかを簡単に記述することを想定している．

次に，表 5 に選択できる活動の一覧を示す．

Table 5　活動とその内容

| 活動 | 内容 |
|---|---|
| 設計作業 | 設計に関する作業 |
| 開発作業 | コーディング等の作業 |
| テスト作業 | テストの実施 |
| ドキュメント作成 | ドキュメントの作成 |
| ミーティング | ミーティング<br>若しくは，成果物のレビュー |
| 調査活動 | 該当チケットに関する調査 |

次に，進捗率の更新をおこなう．作業時間を入力したチケットを右クリックすると，2.2 節の図 4 と同様のポップアップが現れる．ここで，「進捗率」からチケットの進捗率を選択することができる．

### 2.3. 担当したチケットのステータスを終了にする

チケットの作業が完了したら，チケットのステータスを「終了」に変更する．ステータスを変更するには，2.2 節と同様に「ステータス」から「終了」を選択することでできる．

なお，2.2 節から 2.4 節までの操作を一度におこないたい場合には，「チケット」タブからチケットの一覧を表示し，該当チケットを開いた後，「更新」から各項目について入力することが可能である．

## 3. チケットと Git との関連付け

チケットと成果物の変更を関連付ける方法について述べる．

成果物を Git でコミットする際にコメントの入力が可能である．その際に，「refs #43」のように「refs #チケット番号」を入力することで，入力したチケット番号のチケットと成果物の変更を関連付けることができる．もちろん，コメントにはチケットとの関連用のキーワードだけでなく，どのような変更をおこなったのかということも記述することとする．複数のチケットと関連付けることも可能であり，その場合には「refs #43, #44」というようにする．

なお，「refs #チケット番号」の前後に空白，改行以外の文字がある場合キーワードとして認識されないため，留意してコメントを記述すること．

# Kinect サーバ

## WebAPI 仕様書

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 初版 | 2011/07/15 | WebAPI 草案 | 小菅 |
| 2 版 | 2011/12/20 | 概要とデータ提供方法の記述および API の追加 | 劉 |
| 3 版 | 2011/12/29 | 複数 Kinect 連携用 API 追加 | 小菅 |
| 4 版 | 2012/1/17 | スタイルや間隔を調整 | 劉 |
| | | | |
| | | | |

# 目次

# 1. Kinect HTTP サーバ API 概要

## 1.1. 機能概要

本 API は、HTTP 通信を用いてクライアントからの指定されたタイプの生データを JSON の形式のテキストデータに格納して返す機能を提供する。

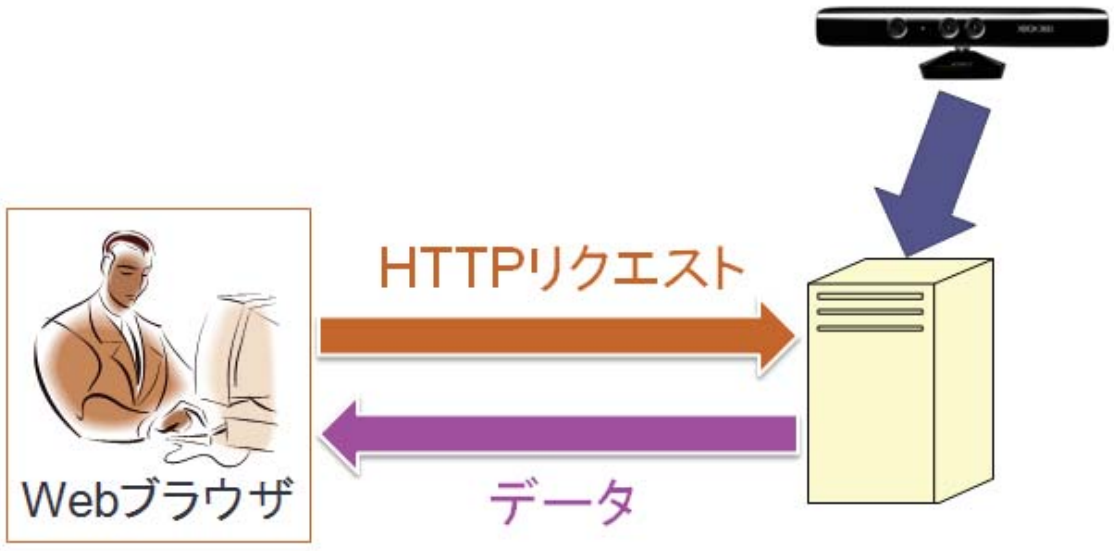
## 1.2. 機能説明

（1） システム条件

- Web サーバ内で動作する。
- クライアント（ブラウザ）からのリクエスト条件に沿ったデータを JSON 形式のテキストデータに編集して送信する。

（2） 処理概念図

データ提供機能は、Web サーバ上に Web API として実装する。以下にクライアントとの処理概念図を示す。



（3） データ提供方法

- 実行環境

  データ提供機能を開発する上で考慮する実行環境および実行形式を以下に示す。

  - サーバ OS：Linux
  - Web サーバ：LibmicroHttp
  - 実行形式：WebAPI としてサーバに実装する。

- 通信方法

  クライアント（ブラウザ）と Web サーバ（CGI）との通信方法及びデータの受け渡し方法を以下に示す。

  - クライアント（ブラウザ）からのデータ取得要求
    - 通信方法： HTTP 通信
    - 送信方式： HTTP POST リクエスト（標準入力）
    - 文字コード：shift_jis
    - データ形式：URL エンコード
    - データ内容：検索条件（データタイプ）

  - Web API からの結果送信
    - 通信方法： HTTP 通信
    - Content-type：text/plain
    - 文字コード： shift_jis
    - データ内容： 結果を JSON 形式のテキストデータ

# 2. Web API インタフェース仕様

## 2.1. getRgbData

【機能】
JSON 形式のイメージデータを取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getRgbData
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | ピクセルごとの色データ | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | version | | | |
| | config | | | |
| | | width | 最大値 | |
| | | height | 最大値 | |
| | pixels | | RGB 情報を Base64 によって encoder されたストリング | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "version":0.1,
  "config":{"width":640,"height":480},
  "pixels":[
    Base64::encode(imageMD.RGB24Data())
  ]
}}
```

失敗:

Status Error Code

【補足】

リクエストに以下のオプションキーを指定可能

| キー | 解説 | フォーマット | 備考 |
|---|---|---|---|
| Format | 取得するデータのフォーマット | "JSON" | 指定しなければ JSON 形式 |

## 2.2. getJpegImage

【機能】
Jpeg 形式の RGB イメージデータを取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getJpegImage
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
Jpeg 画像のバイナリデータ
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | Jpeg 形式の RGB イメージデータを返す | .Jpeg | |

## 2.3. getDepthImageData

【機能】
JSON 形式のデプスマップを取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getDepthImageData
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | ピクセルごとの色データ | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | version | | | |
| | config | | | |
| | | width | 最大値 | |
| | | height | 最大値 | |
| | pixels | | Depth によって色分けをした RGB 情報を<br>Base64 によって encoder されたストリング | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "version":0.1,
  "config":{"width":640,"height":480},
  "pixels":[
    Base64::encode(imageMD.RGB24Data())
  ]
}}
```

失敗:

Status Error Code

【補足】

リクエストに以下のオプションキーを指定可能

| キー | 解説 | フォーマット | 備考 |
|---|---|---|---|
| format | 取得するデータのフォーマット | "JSON" | 指定しなければ JSON 形式 |

## 2.4. getDepthImage

【機能】
Jpeg 形式のデプスマップデータを取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getDepthImage
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
Jpeg 画像のバイナリデータ
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | Jpeg 形式のデプスマップデータを返す | .Jpeg | - |

## 2.5. getPointCloud

【機能】

カメラ座標系における全ての素子の3D座標を取る

【形式】

リクエスト:http:// serverIpAddress:port?type=getPointCloud

メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)

成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | ピクセルごとの深度 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | version | | | |
| | config | | | |
| | | width | 最大値 | |
| | | height | 最大値 | |
| | pointCloud | | | |
| | | | X　単位は m である | |
| | | | Y　単位は m である | |
| | | | Z(depth) 単位は m である | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "version":0.1,
  "config":{"width":640,"height":480},
  "pointCloud":[
     {x:  y: z:}
  ]
}}
```

失敗:

Status Error Code

## 2.6. getSkeletonJointPosition

【機能】
スケルトンのジョイントポイントを取得する

【形式】
リクエスト;http:// serverIpAddress:port?type=getSkeletonJointPosition
メソッド:GET

【引数】(パラメータ)

| キー | 解説 | フォーマット | 備考 |
|---|---|---|---|
| user | 取得するユーザ ID | - | - |
| eJoint | 取得するパーツ | - | - |

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 指定したパーツの座標 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|---|---|
| Kineco | | | | | | |
| | version | | | | | |
| | config | | | | | |
| | | width | | | 最大値 | |
| | | height | | | 最大値 | |
| | | depth | | | 最大値 | |
| | user | | | | | |
| | | id | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | eJoint | | | | |
| | | | Part | | ジョイントポイント | "head" |
| | | | coordinate | | | |
| | | | | x | ジョイントポイントの x 座標 | |
| | | | | y | ジョイントポイントの y 座標 | |
| | | | | z | ジョイントポイントの z 座標 | |
| | | | Part..... | | | |
| | | id | | | | |
| | | ejoint | | | | |
| | | | Part ..... | | | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "version":0.1,
  "config":{"width":640,"height":480,"depth":100},
  "user":[
    {
      "id":1,
      "eJoint":[
        {
          "part":"head",
          "coordinate":{
            "x":1,
            "y":2,
            "z":3
          }
        }
      ]
    }
  ]
}}
```

失敗:

Status Error Code

【補足】

パーツの指定は以下の通り

| neck | right_shoulder | right_elbow | right_foot | right_hand | right_hip | right_knee |
|---|---|---|---|---|---|---|
| 首 | 右肩 | 右肘 | 右足 | 右手 | 右腰 | 右膝 |

| head | torso | left_shoulder | left_elbow | left_foot | left_hand | left_hip | left_knee |
|---|---|---|---|---|---|---|---|
| 頭 | 胴 | 左肩 | 左肘 | 左足 | 左手 | 左腰 | 左膝 |

## 2.7. getCenterPoints

【機能】
複数ユーザの重心を取得する

【形式】
リクエスト;http:// serverIpAddress:port?type=getCenterPoints
メソッド:GET

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 複数ユーザの重心の座標 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|---|---|
| Kineco | | | | | | |
| | user | | | | | |
| | | id | | | | |
| | | Coordinate | | | | |
| | | | | x | ユーザ重心の x 座標 | 2 |
| | | | | y | ユーザ重心の y 座標 | 3 |
| | | | | z | ユーザ重心の z 座標 | 4 |
| | | id | | | | |
| | | coordinate ..... | | | | |

レスポンスフィールドのサンプル


```
{"Kineco":{
    "user":[
      {
        "id":1,
        "coordinate":{
            "x":1,
            "y":2,
            "z":3
          }
       "id":2,
        "coordinate":{
            "x":1,
            "y":2,
            "z":3
          }
        }
      ]
}
}
```


失敗:

Status Error Code

## 2.8. getUserIds

【機能】
検出されている全てのユーザ ID を取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getUserIds
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 検出されている全てのユーザ ID | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | user | | | |
| | | id | ユーザ ID | |
| | | id | ユーザ ID | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "user":[
    {"id":1},
    {"id":2},
    {"id":3}
  ]
}}
```

失敗:

Status Error Code

【補足】

## 2.9. getCalibratedUserIds

【機能】
PSI によってカリブレーションされた全てのユーザ ID を取得する

【形式】
リクエスト:http://serverIpAddress:port?type=getCalibratedUserIds
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | カリブレーションされた | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | user | | | |
| | | id | ユーザ ID | |
| | | id | ユーザ ID | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "user":[
    {"id":1},
    {"id":2},
    {"id":3}
  ]
}}
```

失敗:

Status Error Code

【補足】

## 2.10. getNumberOfDetectedUsers

【機能】
現在のユーザ数を取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getNumberOfDetectedUsers
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 現在ユーザの数 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | user | | | |
| | | detectedUsersCount | 検出したユーザの数 | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "user":
    {"detectedUsersCount":1},
 }
}
```

失敗:
Status Error Code

【補足】

## 2.11. getNumberOfCalibratedUsers

【機能】
現在のユーザ数を取得する

【形式】
リクエスト:http:// serverIpAddress:port?type=getNumberOfCalibratedUsers
メソッド:GET

【引数】(パラメータ)

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 現在ユーザの数 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | user | | | |
| | | calibratedUsersCount | カリブレーションされたユーザの数 | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "user":
    {"calibratedUsersCount":1},
 }
}
```

失敗:
Status Error Code

【補足】

## 2.12. getDepth

【機能】

指定した座標(point.x, point.y)の距離(depth)を取得する

【形式】

リクエスト:http:// serverIpAddress:port?type=getDepth&x=hoge&y=hoge

ﾒｿｯﾄﾞ:GET

【引数】（パラメータ）

| キー | 解説 | フォーマット | 備考 |
|---|---|---|---|
| X | 取得する point の x 座標 | us-ascii | - |
| Y | 取得する point の y 座標 | us-ascii | - |

【戻り値】（レスポンス）

成功：

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 指定した座標の距離 | -application/ json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | coordinate | | | |
| | | x | 取得する point の x 座標 | |
| | | y | 取得する point の y 座標 | |
| | Depth | | 指定した座標から Kinect までの距離（単位は mm です) | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "user":
    {"x":1},
    {"y":1},
   "depth":
 }
}
```

失敗:

Status Error Code

【補足】

## 2.13. getSkeletonById

【機能】
Id によってユーザの骨格情報を取得する

【形式】
リクエスト;http:// serverIpAddress:port?type=getSkeletonById&id=hoge

メソッド:GET

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | ユーザの骨格情報 | application/json | |

レスポンスフィールド:

| フィールド | フィールド | フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|---|---|
| Kineco | | | | | | |
| | user | | | | | |
| | | Id | | | | |
| | | ejoint | | | | |
| | | | part | | | |
| | | | | coodinate | | |

レスポンスフィールドのサンプル

```
{"Kineco":{
    "user":[
      {
        "id":1,
        Ejoint{
         Part:head
        "coordinate":{
             "x":1,
             "y":2,
             "z":3
           }
        Part:neck
        "coordinate":{
             "x":1,
             "y":2,
             "z":3
           }
         .......
        }
    ]
}
}
```

失敗:

Status Error Code

## 2.14. getCenterPointById

【機能】
Id によってユーザの重心の 3D 座標を取得する

【形式】
リクエスト;http:// serverIpAddress:port?type=getCenterPointById&id=hoge
メソッド:GET

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | ユーザの重心の 3D 座標 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|---|
| Kineco | | | | | |
| | user | | | | |
| | | Id | | ユーザ ID | 1 |
| | | Coordinate | | | |
| | | | X | ユーザ重心の X 座標 | |
| | | | Y | ユーザ重心の Y 座標 | |
| | | | Z | ユーザ重心の Z 座標 | |

レスポンスフィールドのサンプル

```
{"Kineco":{
    "user":[
      {
        "id":1,
        "coordinate":{
            "x":1,
            "y":2,
            "z":3
          }
    ]
 }
 }
```

失敗:

Status Error Code

## 2.15. getDepthById

【機能】
Id によってユーザの深度を取得する

【形式】
リクエスト;http:// serverIpAddress:port?type=getDepthById&id=hoge
メソッド:GET

【戻り値】(レスポンス)
成功:

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | ユーザの深度情報 | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|
| Kineco | | | | |
| | user | | | |
| | | Depth | ユーザの深度 | 0.5m |
| | | Id | ユーザ ID | 1 |

レスポンスフィールドのサンプル

```
{"Kineco":{
    "user":[
      {
         “id” :1,
        “depth” :
        }
     ]
 }
 }
```

失敗:

Status Error Code

## 2.16. getMultiPointCloud

【機能】
複数台のデバイスからのイメージ・深度情報を統合したデータを取得する

【形式】
リクエスト:http://serverIpAddress:port?type=getMultiPointCloud
メソッド:GET

【戻り値】(レスポンス)

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 200 | 点群データ | application/json | - |

レスポンスフィールド:

| フィールド | フィールド | フィールド | フィールド | 説明 | 例 |
|---|---|---|---|---|---|
| Kineco | | | | | |
| | version | | | | |
| | config | | | | |
| | | points | | 点群の点数 | |
| | | withColor | | 色情報の有無 | true |
| | PointCloud | | | | |
| | | coordinate | | 座標 | |
| | | | x | | |
| | | | y | | |
| | | | z | | |
| | | color | | | |
| | | | r | | |
| | | | g | | |
| | | | b | | |

レスポンスフィールドのサンプル

```
{"Kineco":{
  "version":0.1,
  "config"{
    "points":153000,
    "withColor":true,
  },
  "PointCloud":[
    {
      "coordinate":{
         "x":1,
         "y":2,
         "z":3
      },
      "color":{
         "r":123,
         "g":221,
         "b":50
      }
    }
  ]
}}
```

失敗：

Status Error Code

【補足】

## 2.17. Status Error Code

レスポンス
失敗：

| ステータスコード | レスポンス | フォーマット | 備考 |
|---|---|---|---|
| 400 | - | - | 不正なリクエスト/パラメータが不足している場合 |
| 403 | - | - | セッションがタイムアウトしたり待ちがでている場合 |
| 500 | - | - | 何らかの原因でエラーが起こった場合 |

※各 API 説明ページに記載がある場合はそちらを優先のこと

# Kinect サーバ

## WebSocket 通知形式仕様書

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 1.0 | 2011/10/16 | 初版 | 茂木 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 1. はじめに

この資料には Kinect サーバの WebSocket 通信によって通知される JSON データの形式を記述する。

# 2. HAND データ

手を検出した際の通知形式

表 1. HAND 検出情報

| key | | | | 型 | 詳細 |
|---|---|---|---|---|---|
| kineco | config | version | | 数値 | バージョン情報 |
| | | type | | 文字列 | データのタイプ、”hand” |
| | hand | action | | 文字列 | create, update, destroy |
| | | id | | 数値 | トラッキングされているユーザの ID |
| | | position | x | 数値 | 手の座標値<br>-320 < x < 320, -240 <y<240<br>0 < z < 1000 |
| | | | y | 数値 | |
| | | | z | 数値 | |

# 3. USER 検出データ

ユーザを検出した際の通知形式

表 2. USER 検出情報

| key | | | 型 | 詳細 |
|---|---|---|---|---|
| kineco | config | version | 数値 | バージョン情報 |
| | | type | 文字列 | データのタイプ、”user” |
| | user | action | 文字列 | “detect”, “lost” |
| | | id | 数値 | ユーザの ID |

## 4. POSE 検出データ

ポーズを検出した際の通知形式

表 3. POSE 検出情報

| key | | | 型 | 詳細 |
|---|---|---|---|---|
| kineco | config | version | 数値 | バージョン情報 |
| | | type | 文字列 | データのタイプ、”pose” |
| | pose | action | 文字列 | “detect”, “lost” |
| | | id | 数値 | ユーザの ID |
| | | name | 文字列 | ポーズの名称 |

# Kinect サーバ

## サンプルクライアント基本設計書

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 初版 | 8/15 | 棒人間 | 朱 |
| 1.0 版 | 10/10 | 3D 空間 | 朱 |
| 2.0 版 | 11/13 | ボックスのトランスポート | 朱 |
| 3.0 版 | 12/18 | 宇宙飛行 | 朱 |
| | | | |
| | | | |

# 目次

# 1. 概要

## 1.1. 目的

本文書では、「Kinect サーバおよびサンプルクライアント研究開発」報告書についての、サンプルクライアントの構築部分の基本設計仕様を記述する。

## 1.2. 方針

WebGL を用いている Web ブラウザで Kinect サーバーからのリアルタイムデータを利用して、サンプルクライアントを開発するため、下記の点に重点を置く。

1. 要求定義書と基本設計書に定めたことに基づいて、設計を行う。
2. Kinect によって、提供できるデータに基づいて、設計を行う。
3. WebGL フレームワークの具体的な特性に基づいて、設計を行う。

## 1.3. 記載範囲

本文書はサンプルクライアントのソフトウェア構成、インタフェース、機能仕様を記載する。

## 1.4. 参照ドキュメント

| ID | 文書名 | 文書番号 | 発行年月 | 備考 |
|---|---|---|---|---|
| | 要件定義書 | | | |
| | | | | |
| | | | | |

## 1.5. 定義（用語、略語）

| ID | 用語・略号 | 正式表記 | 意味 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

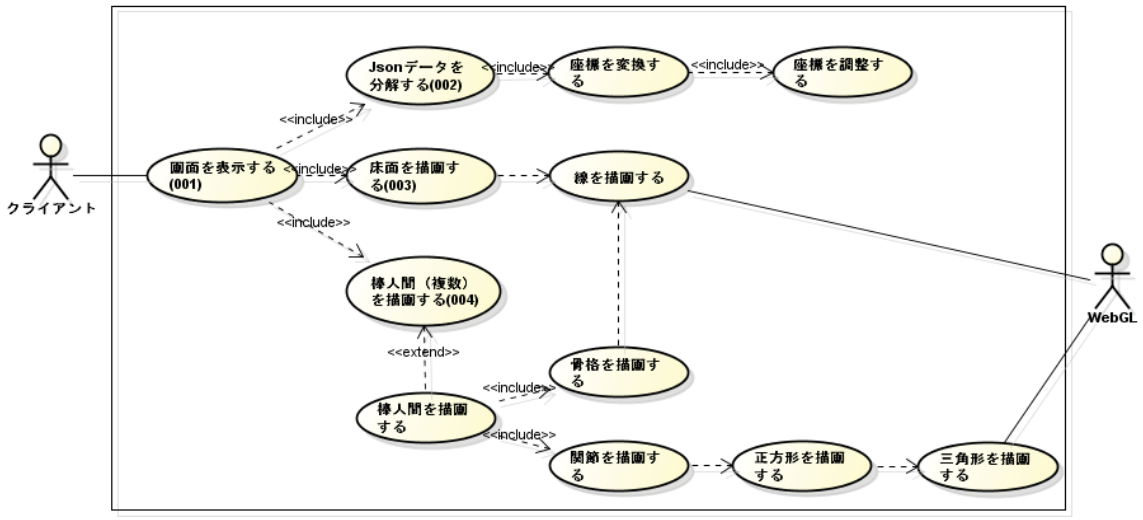
# 2. 棒人間

## 2.1. 概説

横線と縦線が交差している床に、立っている抽象的な人間を描画する。人間の関節が目立つ図形、ボンが関節点と関節点を連結する線を結び付けて、棒人間を表す。

- リアルタイムな骨格データによって、プレーヤのポーズを正確に表す。
- 棒人間をプレーヤの動作と連動して、アニメを生成させる。
- 複数プレーヤの場合は、異なる色で棒人間を区別する。

## 2.2. ユースケース図



## 2.3. ユースケース記述

| | |
|---|---|
| ユースケース ID | 001 |
| ユースケース | 画面を表示する |
| 概要 | Web ブラウザを起動してから、リアルタイムデータによって、棒人間を描画する |
| アクター | クライアント |
| 事前条件 | ユーザーは Web ブラウザを起動している |
| 事後条件 | WebGL のパラメータが正しく設定されている |
| 基本系列 | 1　Canvas オブジェクトのサイズを設定する<br>2　WebGL コンテンツのサイズを設定する<br>3　プログラマブルシェーダーを設定する<br>　3.1　頂点シェーダーを初期化する<br>　3.2　フラグメントシェーダーを初期化する<br>4　視体積の投影面のサイズを設定する<br>5　視体積の投影面の背景色を設定する<br>6　Json データを読み込む<br>　6.1　床のサイズを設定する<br>　6.2　棒人間に該当する関節座標値を設定する<br>7　視体積を定義する<br>8　アフィン変換の変換行列（4×4）を定義する<br>9　床を描画する<br>10　棒人形を描画する<br>11　1 へ戻り、繰り返す |
| 代替系列 | 2A　Web ブラウザが WebGL コンテンツをサポートしない場合は、「"Could not initialise WebGL, sorry :-("」 メッセージを表示する。 |
| 例外系列 | |
| サブユースケース | 002，003，004 |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 002 |
| ユースケース | Json データを分解する |
| 概要 | サーバーに繋がって、リアルタイムデータを取得する |
| アクター | クライアント |
| 事前条件 | WebGL のパラメータが正しく設定されている |
| 事後条件 | 床と棒人間に該当するデータを設置する |
| 基本系列 | 1　サーバーの URL：Port から骨格 Json データを取得する<br>2　床オブジェクトを初期化する<br>3　床を描画する<br>4　棒人間オブジェクトを初期化する<br>5　座標データを引数として、棒人間を描画する<br>6　Web ブラウザで床と棒人間を正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | 003，004 |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 003 |
| ユースケース | 床を描画する |
| 概要 | 床のサイズを設定して、床を描画する |
| アクター | クライアント |
| 事前条件 | Json データを取得している |
| 事後条件 | 床を正確に描画する |
| 基本系列 | 1　床面の横線・縦線の頂点・色のバファを初期化する<br>2　床面の Y 方向の平行移動距離、線の色を初期化する<br>3　横線・縦線の本数を計算する<br>4　横線・縦線の頂点配列に両端点の XYZ 座標を設定する<br>5　床面の頂点バファに頂点配列をロードする<br>6　横線・縦線の色配列に線の両端点の RGBA 値を設定する<br>7　床面の色バファに色配列をロードする<br>8　頂点をアクセスする単位個数を設定し、WebGL コンテンツで線を描画する<br>9　床を正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 004 |
| ユースケース | 棒人間を描画する |
| 概要 | 各棒人間の関節座標を設定して、棒人間を描画する |
| アクター | クライアント |
| 事前条件 | Json データを取得している |
| 事後条件 | 各棒人間を正確に描画する |
| 基本系列 | 1　ボディの 24 関節配列を初期化する<br>2　現実座標を棒人間の座標に変換する α 値を設定する<br>3　Json データによって、ユーザー数を抽出する<br>4　ユーザーID 配列を初期化する<br>5　棒人間の座標を設定する<br>5.1　関節（正方形）の頂点・色・頂点インデックスのバファを初期化する<br>5.2　ボンの頂点・色・頂点インデックスのバファを初期化する<br>5.3　ボン配列（関節の関係）を初期化する<br>5.4　1 つの関節の X・Y・Z 座標を変換する<br>5.5　関節の頂点配列にこの関節の XYZ 座標を設定する<br>5.6　関節の最小・最大 Z 軸値を記録する<br>5.7　関節の座標処理は終わるかどうかを判断する。<br>終わらない場合は、5.3 へ戻り<br>5.8　関節の最小･最大 Z 軸値の 1/3 の所を 2 分法の基準値として、不適切な関節の座標を取り除く<br>5.9　ボンの配列から該当する不適切な関節を取り除く<br>5.10　1 つの関節の XYZ 値を中心としての正方形の頂点座標を定義する<br>5.11　関節（正方形）の頂点配列に正方形の各頂点 XYZ 座標を設定する<br>5.12　関節（正方形）の頂点の色配列に正方形の各頂点 RGBA 値を設定する<br>5.13　関節（正方形）の頂点インデックス配列に各頂点の番号を設定する<br>5.14　ボンの色配列にボンの両端点の RGBA 値を設定する<br>5.15　ボンの頂点インデックス配列に両端点の番号を設定する<br>5.16　関節（正方形）の頂点バファに関節（正方形）の頂点配列をロードする<br>5.17　関節（正方形）の色バファに関節（正方形）の色配列を |

| | |
|---|---|
| | ロードする<br>　5.18　関節（正方形）の頂点インデックスバファに関節（正方形）の頂点インデックス配列をロードする<br>　5.19　ボンの色バファにボンの色配列をロードする<br>　5.20　ボンの頂点インデックスバファにボンの頂点インデックス配列をロードする<br>6　棒人間の座標の設定が全て終わるかどうかを判断する<br>　終わらない場合は、5 へ戻り<br>7　WebGL コンテンツで関節（正方形）を描画する<br>8　WebGL コンテンツでボンを描画する<br>9　棒人間を正しく描画したら、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

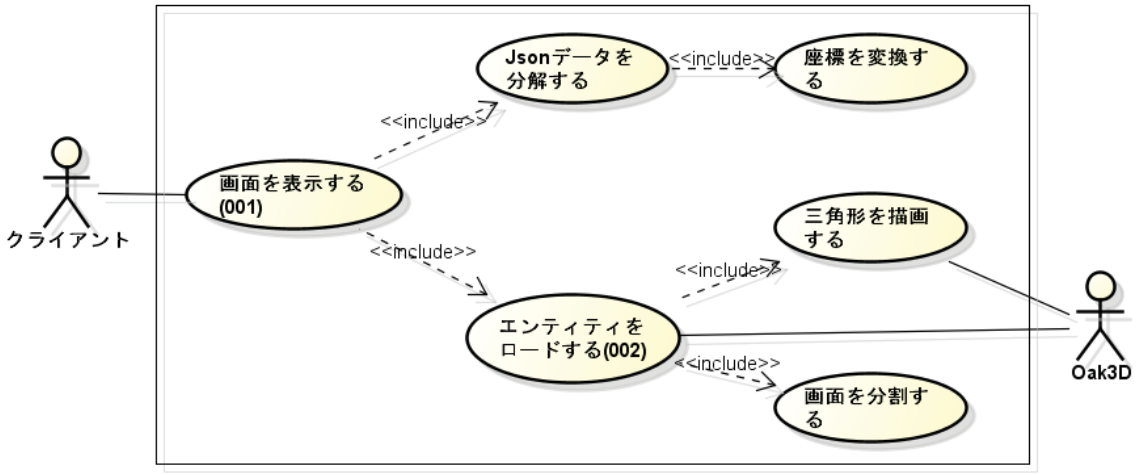
# 3. 3D 空間

## 3.1. 概説

現実の 3D 空間を WebGL で再現する。地理学に等圧線みたいな 3D 空間モデルを生成する。

- 深度が同じ物体は同じ切断面に表す。
- マウスなどの移動に従って、ユーザーは各角度からこのモデルを見る事ができる。

## 3.2. ユースケース図



### 3.3. ユースケース記述

| | |
|---|---|
| ユースケース ID | 001 |
| ユースケース | 画面を表示する |
| 概要 | Web ブラウザを起動してから、リアルタイムデータによって、3D 空間を描画する |
| アクター | クライアント |
| 事前条件 | ユーザーが Web ブラウザを起動している |
| 事後条件 | リアルタイムデータによって、3D 空間の描画を正しく繰り返す |
| 基本系列 | 1　Canvas オブジェクトのサイズを設定し、div タッグと連携する<br>2　initScene というコールバックファクションを定義する<br>2.1　Oak3D のエンジンを初期化する<br>2.2　エンジンと Canvas を結び付ける<br>2.3　シーンを初期化する<br>2.4　シーンのバウンディングボリュームを設定する<br>2.5　カメラを初期化する<br>2.6　視体積を定義する<br>2.7　視体積の投影面のサイズを設定する<br>2.8　視体積の投影面のバックグランド色を設定する<br>2.9　カメラの位置を設定する<br>2.10　視点の座標を設定する<br>2.11　サーバーの URL：Port から RGB＋D Json データを取得する<br>2.12　3D 空間のエンティティをロードする<br>2.13　2.11 へ戻り、繰り返す<br>3　マウスのコールバックイベントを設定する<br>4　タイマを設定し、一定時間を経ってシーンをレンダーするコールバックファクションを定義する |
| 代替系列 | 2.1A　Canvas がエンジンをサポートしない場合は、「"Sorry, your browser doesn't support WebGL!"」メッセージを表示する。 |
| 例外系列 | |
| サブユースケース | 002 |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 002 |
| ユースケース | エンティティをロードする |
| 概要 | Json データにより、各エンティティに 3D 空間のパラメータデータを設定して、最後一体にする |
| アクター | クライアント |
| 事前条件 | Json データを取得している |
| 事後条件 | 各エンティティに Json データが正確に設置されている |
| 基本系列 | 1　現実空間を 3D 空間の座標に変換するα値を設定する<br>2　頂点バファの最大サイズによって、画像を分割する個数を計算する<br>3　子画像の頂点配列と色配列を初期化する<br>4　子画像のスタットピクセルとエンドピクセルを計算する<br>5　深度値を 3D 空間の Y 軸に、画像の幅を Z 軸に変換する<br>6　子画像の頂点配列に各ピクセルの XYZ 値を設定する<br>7　子画像の色配列に各ピクセルの RGB 値を設定する<br>8　子画像のピクセルの設定が全て終わっているかどうかを判断する。終わらない場合は、5 へ戻り<br>9　子画像の頂点インデックス配列を初期化する<br>10　子画像の頂点インデックス配列に各頂点に関する三角形の頂点を設定する<br>11　シーンにエンティティを新規にする<br>12　エンティティの頂点・色・インデックスの属性を削除する<br>13　エンティティの頂点属性に子画像の頂点配列を設定する<br>14　エンティティの色属性に子画像の色配列を設定する<br>15　エンティティのインデックス属性に子画像の頂点インデックス配列を設定する<br>16　エンティティを一定比率でスケーリングする<br>17　動的な光源をクロースする<br>18　子画像の処理を全てし終わるかどうかを判断する。<br>終わらない場合は、3 へ戻り<br>19　3D 空間が正しく描画されたら、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

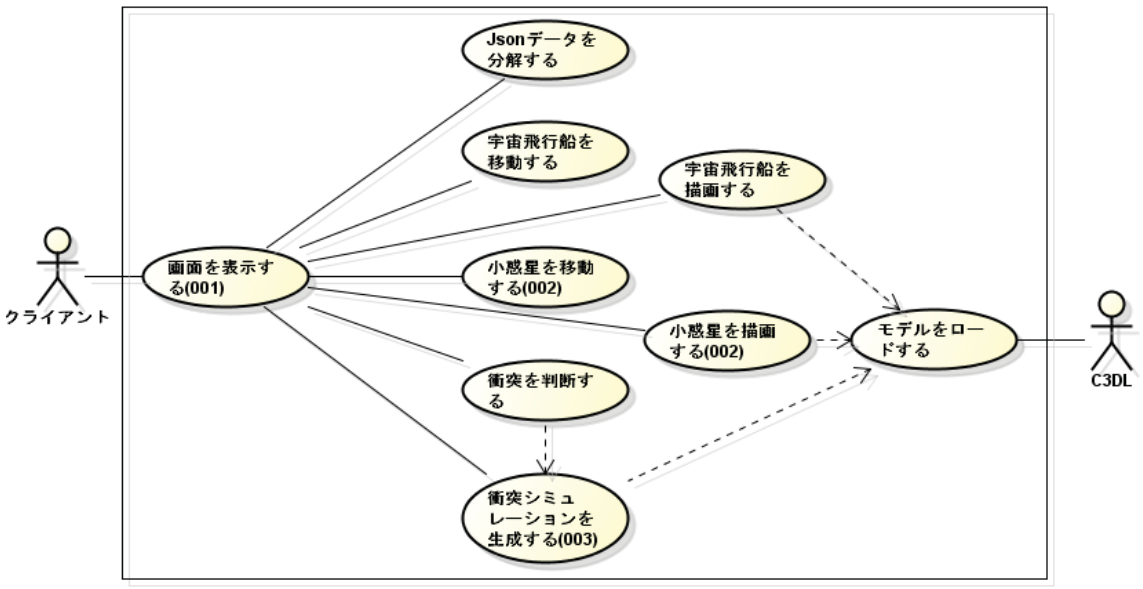
# 4. 宇宙飛行

## 4.1. 概説

宇宙飛行船は飛行船の方向に接近している小惑星が行き渡る宇宙で飛行するアニメを生成する。

- リアルタイムデータの指令に従って、宇宙飛行船が該当する方向へ移動する。
- 一定時間を経ってから、新たな小惑星が飛行船の前に生成される。
- 小惑星が軌道に沿って、飛行船に向かって飛んで来る。
- 宇宙飛行船とある小惑星の距離が限定値に近づいて、衝突シミュレーションを生成する。

## 4.2. ユースケース図



### 4.3. ユースケース記述

| | |
|---|---|
| ユースケース ID | 001 |
| ユースケース | 画面を表示する |
| 概要 | Web ブラウザを起動してから、リアルタイムデータによって、宇宙で飛行するアニメを生成する |
| アクター | クライアント |
| 事前条件 | ユーザーが Web ブラウザを起動している |
| 事後条件 | 宇宙飛行のアニメを正しく描画する |
| 基本系列 | 1　WebSocket サーバーにつながって、URL : Port から Json データを取得する<br>2　シーンを初期化する<br>　2.1　シーンのサイズを設定する<br>　2.2　シーンと Canvas を結び付ける<br>　2.3　シーンのバックグランド色を設定する<br>3　WebGL コンテンツを初期化する<br>　3.1　WebGL コンテンツでシーンをレンダーする<br>4　カメラを初期化する<br>　4.1　カメラの位置を設定する<br>　4.2　カメラの視点を設定する<br>　4.3　シーンにカメラオブジェクトをロードする<br>5　小惑星配列を初期化する<br>6　小惑星を初期化する<br>7　宇宙飛行船を初期化する<br>　7.1　宇宙飛行船 collada ディレクトリを初期化する<br>　7.2　dae モデルをロードする<br>　7.3　宇宙飛行船の位置を設置する<br>　7.4　宇宙飛行船をスケーリングする<br>　7.5　シーンに宇宙飛行船オブジェクトをロードする<br>8　衝突シミュレーションを初期化する<br>9　シーンを更新するコールバックファクションを定義する<br>　9.1　一定時間を経って、新しい小惑星を生成する<br>　9.2　Json データによって、宇宙飛行船を該当する方向へ平行移動する<br>　9.3　前後移動する場合は、カメラが連動する<br>　9.4　1 つの小惑星と宇宙飛行船の距離を判断する |

| | |
|---|---|
| | 限定値より小さい場合は、衝突シミュレーションを生成し、シーンから宇宙飛行船オブジェクトを削除して、本ユースケース記述は終了する<br>9.5　小惑星の判断が全て終わらない場合は、9.4 へ戻り<br>10　星空を初期化する<br>10.1　星空 collada ディレクトリを初期化する<br>10.2　dae モデルをロードする<br>10.3　シーンに星空オブジェクトをロードする<br>11　シーンを描画する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | 002，003 |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 002 |
| ユースケース | 小惑星を初期化する |
| 概要 | 新しい小惑星のパラメータを設定して、シーンに描画する |
| アクター | クライアント |
| 事前条件 | シーンのパラメータの設置が終わっている |
| 事後条件 | シーンに 1 つの新しい小惑星を正確に描画する |
| 基本系列 | 1　小惑星 collada ディレクトリを初期化する<br>2　dae モデルをロードする<br>3　png 図でテクスチャする<br>4　小惑星をスケーリングする<br>5　投影面（PC のスクリーン）で 1 つの点を任意選択して、点の XY 値を取得する<br>6　カメラの座標を結び付け、小惑星が WebGL 空間に XYZ 座標を設定する<br>7　小惑星が進行するスピードを設定する<br>8　小惑星の軌道を設定する<br>9　小惑星の X 軸（pitch）、Y 軸（yaw）、Z 軸（roll）の回転角度を任意設置する<br>10　小惑星の中心軸に回転角速度を任意設置する<br>11　小惑星配列に小惑星を設定する<br>12　シーンに新しい小惑星オブジェクトをロードする<br>13　新しい小惑星を正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 003 |
| ユースケース | 衝突シミュレーションを初期化する |
| 概要 | 衝突シミュレーションを事前に用意して、衝突が起きた場合はシーンに衝突シミュレーションを描画する |
| アクター | クライアント |
| 事前条件 | シーンのパラメータの設置が終わっている |
| 事後条件 | 衝突が起きると、シーンに衝突シミュレーションを正確に描画する |
| 基本系列 | 1　粒子システムを初期化する<br>2　粒子毎に最大・最小スピードを設置する<br>3　粒子毎に現れる最大・最小時間帯（秒）を設置する<br>4　粒子毎に現れる色の幅を設置する<br>5　指定された図でテクスチャする<br>6　粒子システムに粒子の総数を設置する<br>7　シーンに粒子システムオブジェクトをロードする<br>8　衝突を発生する場合は、衝突シミュレーションオブジェクトの座標と持続する時間を設置する<br>9　衝突シミュレーションを正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

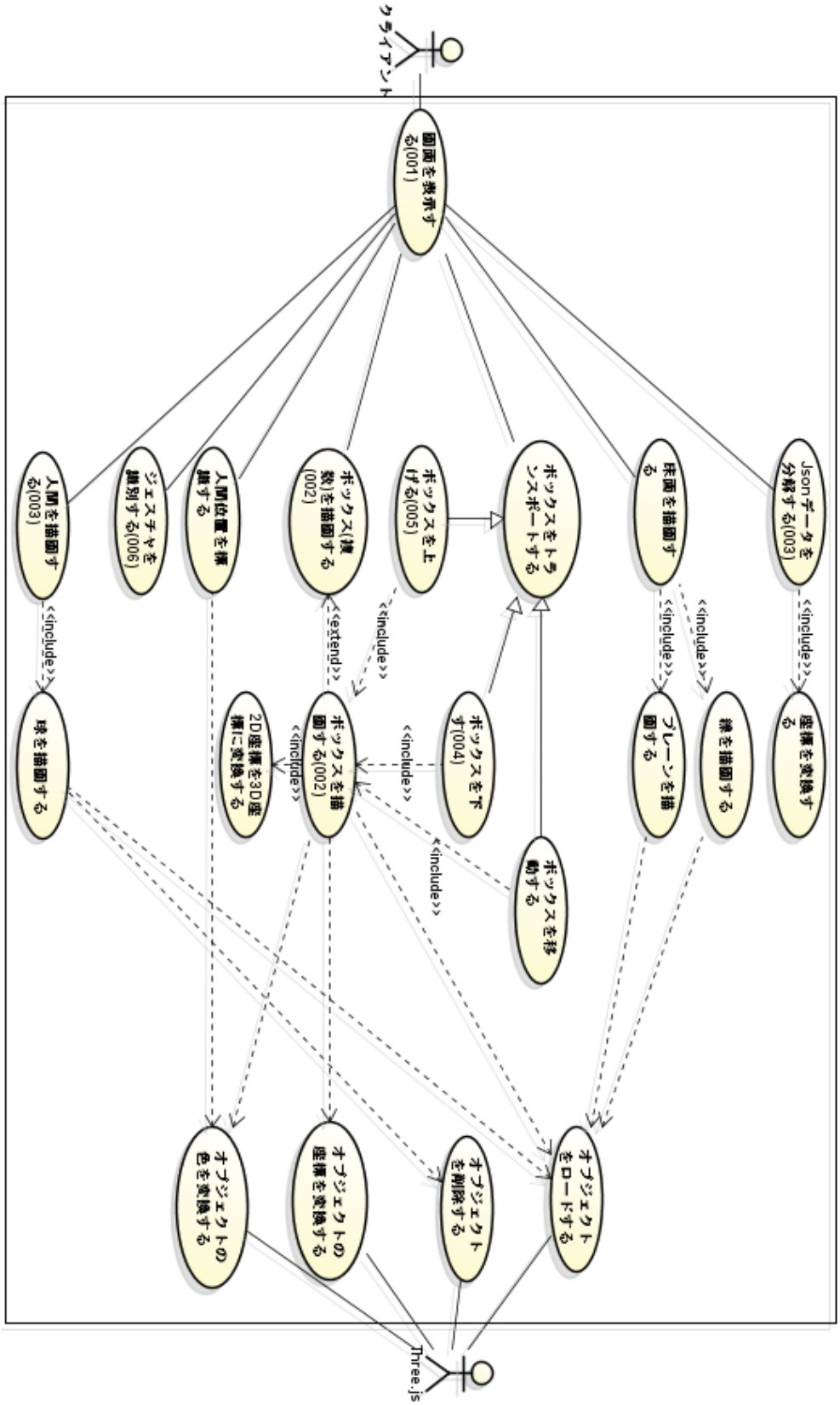
# 5. ボックスのトランスポート

## 5.1. 概説

Kinect を通じて、プレーヤが VR 環境の物体をトランスファすることができるようなアニメを生成する。

- チェック柄の床の中心に、抽象な人間が立っているが、周りの格子にボックスを任意散らす。
- 毎回 Web ページをオープンすると、ボックスが置いている格子を任意選ぶ。
- 球で人間の関節を表す球人間は、プレーヤの身ぶりを正確に表す。
- プレーヤが現実空間に位置を移動すれば、球人間もそれに応じて移動し、立っている格子が目立つ色で表現する。
- 特定な持ち上げポーズを認識すると、目の前に手が触れているボックスが持ち上げられる。
- 持ち上げられたボックスは、プレーヤの手の動きに従って、移動する。
- 特定な持ち上げポーズがロストしたら、持っているボックスが床の格子に落ちる。

## 5.2. ユースケース図


クライアント
画面を表示する(001)
Jsonデータを分解する(003)
床面を描画する
ボックスをトランスポートする
ボックスを上げる(005)
ボックス(複数)を描画する(002)
人間位置を識別する
ジェスチャを識別する(006)
人間を描画する(003)
座標を変換する
ブレーンを描画する
線を描画する
ボックスを下す(004)
ボックスを描画する(002)
2D座標を3D座標に変換する
球を描画する
ボックスを移動する
オブジェクトをロードする
オブジェクトを削除する
オブジェクトの座標を変換する
オブジェクトの色を変換する
Three.js
<<include>>
<<extend>>

### 5.3. ユースケース記述

| | |
|---|---|
| ユースケース ID | 001 |
| ユースケース | 画面を表示する |
| 概要 | Web ブラウザを起動してから、リアルタイムデータによって、ボックスをトランスポートするアニメを生成する |
| アクター | クライアント |
| 事前条件 | ユーザーが Web ブラウザを起動している |
| 事後条件 | ボックスのトランスポートのアニメを正しく描画する |
| 基本系列 | 1　Web ドキュメントに新しい div を追加する<br>2　レンダラーを初期化する作成<br>2.1　レンダラーのサイズを設定する<br>2.2　div コンテナにレンダラーエレメントを設定する<br>3　カメラを初期化する<br>3.1　視体積を定義する<br>3.2　カメラの位置を設定する<br>3.3　視点を設定する<br>4　シーンを初期化する<br>5　床を描画する<br>5.1　横線・縦線の座標と色を設定する<br>5.2　シーンに横線・縦線オブジェクトを設定する<br>5.3　プレーンのサイズと色を設定する<br>5.4　シーンにプレーンオブジェクトを設定する<br>6　プロジェクタを初期化する<br>7　レンダラーにシーンとカメラを設定する<br>8　ボックスを描画する<br>9　アニメのコールバックファクションを定義する<br>9.1　Json データを取得する<br>9.2　球人間を描画する<br>9.3　特定持ち上げポーズを認識する |
| 代替系列 | |
| 例外系列 | 002，003，004，005，006 |
| サブユースケース | |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 002 |
| ユースケース | ボックスを描画する |
| 概要 | 床面の格子にランダムでボックスを描画する |
| アクター | クライアント |
| 事前条件 | シーンのパラメータを設定している、且つ床面を描画している |
| 事後条件 | 床面にボックスを正確に描画する |
| 基本系列 | 1　投影面（スクリーン）と交わるカメラ射線を初期化する<br>2　生成するボックスの個数を設定する<br>3　スクリーンに 1 つの点を任意選ぶ<br>4　点の XY 座標とカメラの XYZ 座標を合わせて、WebGL 空間に該当する XYZ 座標を計算する<br>5　視体積にカメラと点の延長線を定義する<br>6　シーンに延長線と衝突するオブジェクトがあるかどうかを判断する<br>　衝突するオブジェクトが床面である場合は、交差点の座標を取得する<br>7　ボックスを描画する<br>　7.1　ボックスのサイズと色を設定する<br>　7.2　メッシュで立方体を描画する<br>8　ボックス座標に交差点が所属する格子座標を設定する<br>9　シーンにボックスオブジェクトを設定する<br>10　個数処理が全て終わっているかどうかを判断する。終わらない場合は、3 へ戻り<br>11　ボックスを正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 003 |
| ユースケース | アニメのコールバックファクションを定義する |
| 概要 | リアルタイムデータによって、球人間を描画し、特定持ち上げポーズを認識する |
| アクター | クライアント |
| 事前条件 | レンダー ループを定義している |
| 事後条件 | 球人間をリアルタイムで描画し、ジェスチャを判断する |
| 基本系列 | 1　サーバーの URL：Port から骨格 Json データを取得する<br>2　座標データによって、球人間を描画する<br>　2.1　シーンから既存している球人間オブジェクトを削除する<br>　2.2　球人間の高さを定義する<br>　2.3　プレーヤの高さを合わせて、球人間の座標に変換するα値を設定する<br>　2.4　球人間の各関節の座標・色を設定する<br>　2.5　メッシュで球のような関節を描画する<br>　2.6　シーンに球人間オブジェクトを設定する<br>　2.7　シーンをレンダーする<br>3　プレーヤの身ぶりを判断する<br>　3.1　移動する場合は、球人間の頭の座標が所属する格子は赤くなる<br>　3.2　特定な持ち上げポーズを認識する且つ持っているボックスがない場合は、ボックスがアップする<br>　3.3　特定な持ち上げポーズを認識する且つ持っているボックスがある場合は、ボックスが移動する<br>　3.4　特定な持ち上げポーズが認識できない且つ持っているボックスがある場合は、ボックスが落ちる<br>4　シーンをレンダーする<br>5　アニメを正しく生成し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | 004，005，006 |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 004 |
| ユースケース | ボックスを下す |
| 概要 | 持っているボクスを床面の格子に降ろし置き |
| アクター | クライアント |
| 事前条件 | 球人間がフォーカスボックスを持っている |
| 事後条件 | フォーカスボックスが床面の格子に落ちる |
| 基本系列 | 1　持っているボックスがあるかどうかを判断する<br>　ない場合は、戻り<br>2　球人間の頭の座標を取得する<br>3　視体積にカメラと頭の延長線を定義する<br>4　シーンに延長線と衝突するオブジェクトがあるかどうかを判断する<br>　衝突するオブジェクトが床である場合は、交差点の座標を取得する。<br>5　床面の格子の法線ベクトルを合わせて、格子の座標を取得する<br>6　ボックスの座標に格子の座標を設定する<br>7　ボックスの色を元に戻す<br>8　ボックスを正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 005 |
| ユースケース | ボックスを上げる |
| 概要 | 床面の格子に置いているボックスを持ちあげる |
| アクター | クライアント |
| 事前条件 | 特定持ち上げポーズを認識する |
| 事後条件 | フォーカスボックスが手で持ち上げられる |
| 基本系列 | 1　持っているボックスがあるかどうかを判断する<br>　ない場合は、戻り<br>2　球人間の両手の中心座標を取得する<br>3　視体積にカメラと両手の中心の延長線を定義する<br>4　シーンに延長線と衝突するオブジェクトがあるかどうかを判断する<br>　衝突するオブジェクトがボックスである場合は、ボックスオブジェクトを取得する。<br>5　フォーカスボックスの新座標を設定する<br>6　フォーカスボックスの色を設定する<br>7　ボックスを正しく描画し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

| | |
|---|---|
| ユースケース ID | 006 |
| ユースケース | ジェスチャを識別する |
| 概要 | プレーヤの動作によって、特定持ち上げポーズを判断する |
| アクター | クライアント |
| 事前条件 | 球人間がプレーヤのポーズを正確に表示する |
| 事後条件 | 複数の判断条件の結果を表明する |
| 基本系列 | 1　左・右上腕のベクトルを計算する<br>2　左上腕と右上腕の角度 A を計算する<br>3　左・右上肢のベクトルを計算する<br>4　左上肢と右上肢の角度 B を計算する<br>5　両肩のベクトルを計算する<br>6　両手の中点・両肘の中点のベクトルを計算する<br>7　両肩と両手の中点・両肘の中点の角度 C を計算する<br>8　頭と首のベクトルを計算する<br>9　両肩と頭・首の角度 D を計算する<br>10　頭・首と両手の中点・両肘の中点の角度 E を計算する<br>11　角度 A＜10°、角度 B＜10°、角度 C＞80°、角度 D＞85°と角度 E＞25°を全て満たす場合は、特定な持ち上げポーズを認識する<br>12　ジェスチャを正しく認識し、本ユースケースは終了する |
| 代替系列 | |
| 例外系列 | |
| サブユースケース | |
| 備考 | |

# Kinect サーバ

## HTTP サーバ側詳細設計

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 1.0 版 | 1/17 | | 劉 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 目次

## 内容

# 1　概要

## 1.1　目的

本文書では、「Kinect サーバおよびサンプルクライアント研究開発」報告書についての、Http サーバの構築部分の詳細設計仕様を記述する。

## 1.2　方針

センサーデータ提供機能の開発に当たって構築した Http サーバは下記の点に重点を置いて設計した。

1. リアルタイム応答性の確保
2. センサーデータの更新と取得をカプセル化
3. 拡張性への配慮（他機能との連携ポイントを設け）

## 1.3　記載範囲

本文書は Http サーバのソフトウェア構成、インタフェース、機能仕様を記載する。

## 1.4　参照ドキュメント

| ID | 文書名 | 文書番号 | 発行年月 | 備考 |
|---|---|---|---|---|
| | WebAPI 仕様書 | | | |

## 1.5　定義（用語、略語）

| ID | 用語・略号 | 正式表記 | 意味 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

# 2　Http サーバ

## 2.1　シーケンス図

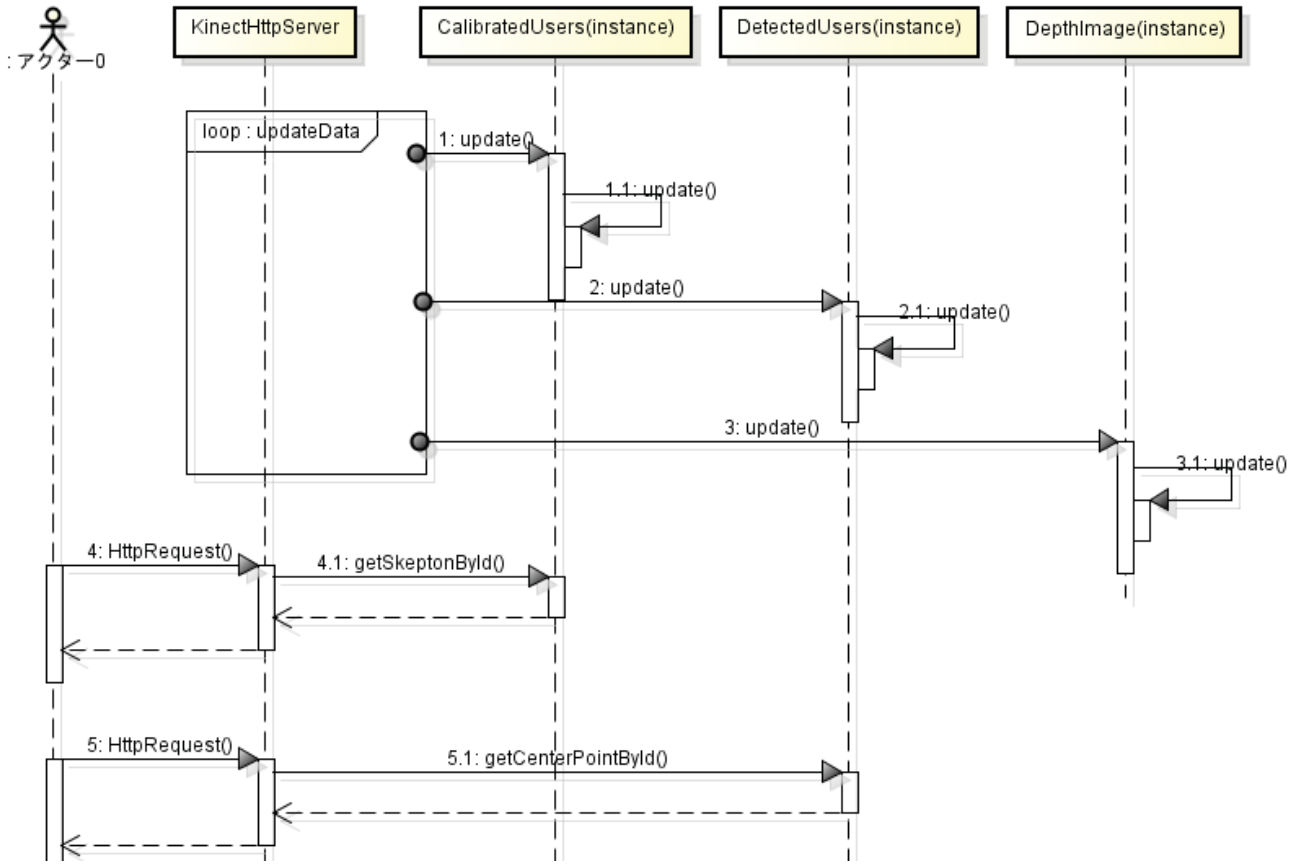



図 2-1 データ取得までのシーケンス図

## 2.2　Http サーバクラス図（一部）

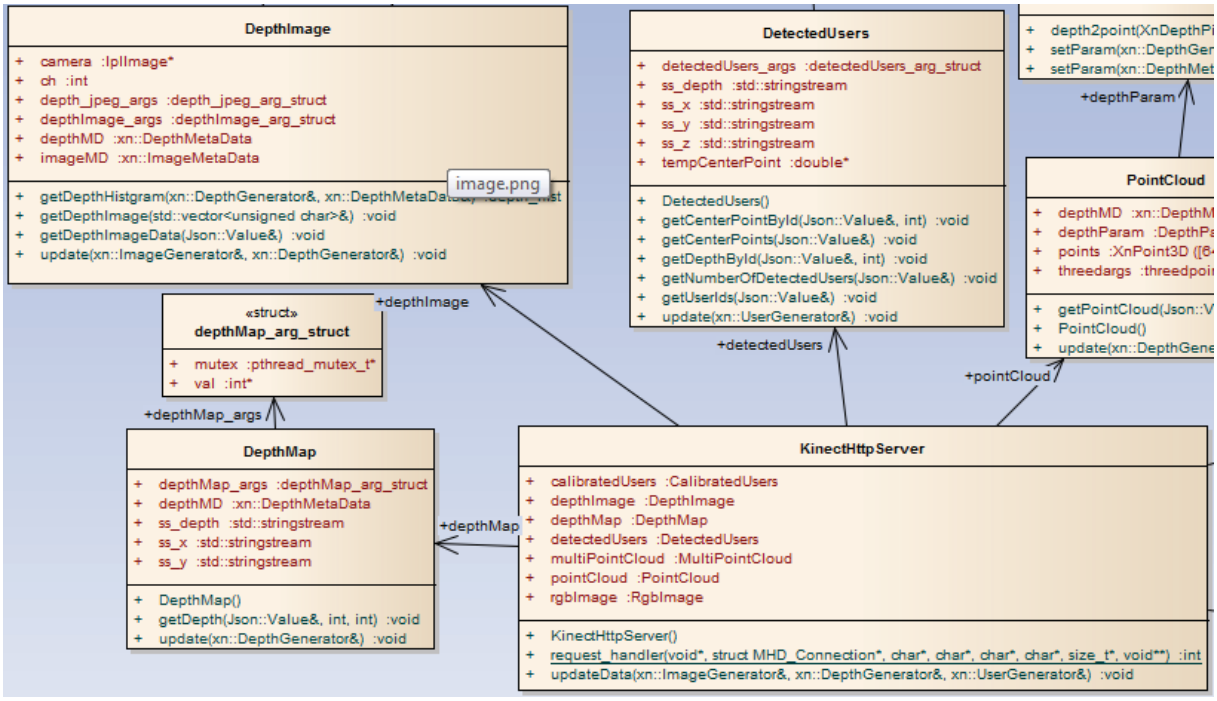



図 2-2　クラス間の関連つけ

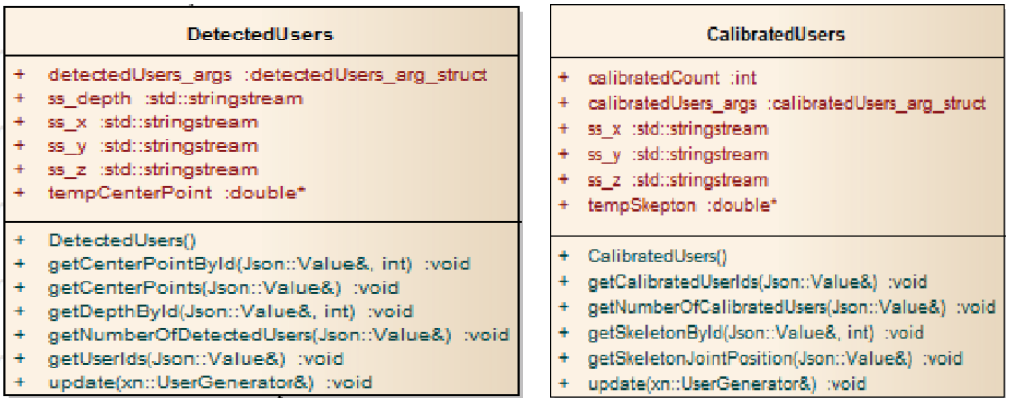



図 2-3　検出・識別ユーザクラス図の相違

## 2.3　クラス図詳細

| クラス名 | KinectHttpServer | | |
|---|---|---|---|
| 親クラス | なし | | |
| クラス概要 | | | |
| センサーデータタイプごとのクラスのインスタンスを持つ、総括にデータの更新や取得をコントロールする。 | | | |
| 外部ライブラリ | | | |
| OpenNI | | | |
| microhttpd | | | |
| cURL | | | |
| OpenCV | | | |
| Base64 | | | |
| JSON | | | |
| 属性 | | | |
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| public | calibratedUsers | CalibratedUsers | 識別されたユーザ |
| public | depthImage | DepthImage | 深度マップ（画像） |
| public | depthMap | DepthMap | 深度マップ（距離） |
| public | detectedUsers | DetectedUsers | システムパラメータ |
| public | multiPointCloud | MultiPointCloud | 連携したポイントクラウド |
| public | pointCloud | PointCloud | 単独ポイントクラウド |
| public | rgbImage | RgbImage | RGB 画像 |
| メソッド | | | |
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | KinectHttpServer | | コンストラクタ関数 |
| public | request_handler | コネクション情報 | 外部リクエスト受け取る関数 |
| public | updateData | Image,User,Depth などのジェネネータ | センサーデータ更新関数 |

| クラス名 | DepthMap | | |
|---|---|---|---|
| 親クラス | なし | | |
| クラス概要 | | | |
| 深度マップの取得や更新するクラス、指定した点の深度を提供 | | | |
| 属性 | | | |
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| public | depthMap_args | depthMap_arg_struct | 深度マップ関連のストラクタ |
| public | depthMD | Xn::DepthMetaData | デプス meta データ |
| public | ss_depth | Std::stringstream | 指定した点の深度 |
| public | ss_x | Std::stringstream | 指定した点の X 座標 |
| public | ss_y | Std::stringstream | 指定した点の Y 座標 |
| メソッド | | | |
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | DepthMap | | コンストラクタ関数 |
| public | getDepth | 指定した点の X,Y 座標 | 深度取得用の関数 |
| public | update | depthGenerator | 全ての点の深度情報を更新する |

| クラス名 | DepthImage | | |
|---|---|---|---|
| 親クラス | なし | | |
| クラス概要 | | | |
| JSON 形式、JPEG 形式のデプスマップの取得や更新用クラス | | | |
| 属性 | | | |
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| public | IpllImage | camera | カメラインスタンス |
| public | ch | int | 画像のチャンネル数(3) |
| public | depth_jpeg_args | depth_jpeg_arg_struct | JPEG 形式データ関連のストラクタ |
| public | depthImage_args | depthImage_arg_struct | JSON 形式データ関連のストラクタ |
| public | depthMD | xn::DepthMetaData | 深度元データ |
| public | imageMD | xn::ImageMetaData | 画像元データ |
| メソッド | | | |
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | getDepthHistgram | depthGenerator,depthMetaData | 全ての点の深度情報を格納するリストを取得 |
| public | getDepthImage | Vector<unsigned char>& | JPEG 形式のデプスマップを取得 |
| public | getDepthImageData | Json::value& | JSON 形式のデプスマップを取得 |
| public | update | imageGenerator,depthGenerator | デプスマップ情報を更新する |

| クラス名 | RgbImage | | |
|---|---|---|---|
| 親クラス | なし | | |
| クラス概要 | | | |
| JSON 形式、JPEG 形式の RGB 画像データ取得や更新用クラス | | | |
| 属性 | | | |
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| private | IplImage | camera | カメライインスタンス |
| private | ch | int | 画像のチャンネル数(3) |
| private | imageMD | xn::ImageMetaData | 画像元データ |
| private | jpeg_args | jpeg_arg_struct | JPEG 形式データ関連のストラクタ |
| private | rgb_args | rgb_arg_struct | JSON 形式データ関連のストラクタ |
| メソッド | | | |
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | getJpegImage | Vector<unsigned char>& | JPEG 形式のデプスマップを取得 |
| public | getRgbData | Json::value& | JSON 形式のデプスマップを取得 |
| public | update | imageGenerator | RGB データ情報を更新する |

| クラス名 | CalibratedUsers | | |
|---|---|---|---|
| 親クラス | なし | | |
| クラス概要 | | | |
| 識別されてユーザクラス | | | |
| 属性 | | | |
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| public | calibratedCount | int | 識別されたユーザの数 |
| public | calibratedUsers_args | calibratedUsers_arg_struct | 識別されたユーザの関連ストラクタ |
| public | ss_x | std::stringstream | X 座標 |
| public | ss_y | std::stringstream | Y 座標 |
| public | ss_z | std::stringstream | Z 座標 |
| public | tempSkepton | double* | メモリに保存する骨格データへのポインタ |
| メソッド | | | |
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | CalibratedUsers | | コンストラクタ関数 |
| public | getCalibratedUserIds | Json::value&, ユーザ ID | 識別されたユーザの ID 取得 |
| public | getNumberOfCalibratedUsers | Json::value& | 識別されたユーザの数を取得 |
| public | getSkeletonById | Json::value&, ユーザ ID | 指定した ID のユーザの骨格情報を取得 |
| public | getSkeletonJointPosition | Json::value& | 識別された全てのユーザの骨格情報を取得 |
| public | update | userGenerator | 識別されたユーザの情報を更新する |

| クラス名 | DetectedUsers | | |
|---|---|---|---|
| 親クラス | なし | | |
| クラス概要 | | | |
| 検出したユーザクラス | | | |
| 属性 | | | |
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| public | detectedUsers_args | detectedUsers_arg_struct | 検出したユーザ関連のストラクタ |
| public | Ss_depth | std::stream | 深度 |
| public | Ss_x | std::stream | X 座標 |
| public | Ss_y | std::stream | Y 座標 |
| public | Ss_z | std::stream | Z 座標 |
| public | tempCenterPoint | double* | メモリに保存する重心情報へのポインタ |
| メソッド | | | |
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | DetectedUsers | | コンストラクタ関数 |
| public | getCenterPointById | Json::value&, ユーザ ID | 指定したユーザの重心情報を取得 |
| public | getCenterPoints | Json::value& | 検出した全てのユーザの重心情報を取得 |
| public | getDepthById | Json::value&, ユーザ ID | 指定したユーザの深度を取得 |
| public | getNumberOfDetectedUsers | Json::value& | 検出したユーザの数を取得 |
| public | getUserIds | Json::value& | 検出したユーザの ID を取得 |
| public | update | userGenerator | 検出したユーザの情報を更新する |

| クラス名 | PointCloud |
|---|---|
| 親クラス | なし |

| クラス概要 |
|---|
| カメラに映っている全ての点の深度情報から点群を算出して提供する。<br>点群とはカメラの全て素子(640*480 個)の 3d 座標の集合 |

| 属性 | | | |
|---|---|---|---|
| 可視性 | 属性・インスタンス名 | 型 | 説明、その他 |
| public | depthMD | Xn::DepthMetaData | 深度元データ |
| public | depthParam | DepthParam | 全ての点の深度情報を格納するリスト |
| public | points | XnPoint3D{[640*480]} | 3d 座標を格納する配列 |
| public | threedargs | Threedpoint_arg_struct | 3d 座標関連のストラクタ |

| メソッド | | | |
|---|---|---|---|
| 可視性 | メソッド名 | 引数 | 説明、その他 |
| public | pointCloud | | コンストラクタ関数 |
| public | getPointCloud | Json::value& | 点群データを取得 |
| public | update | depthGenerator | 点群データを更新 |

# Kinect サーバ

## WebSocket　クラス設計

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

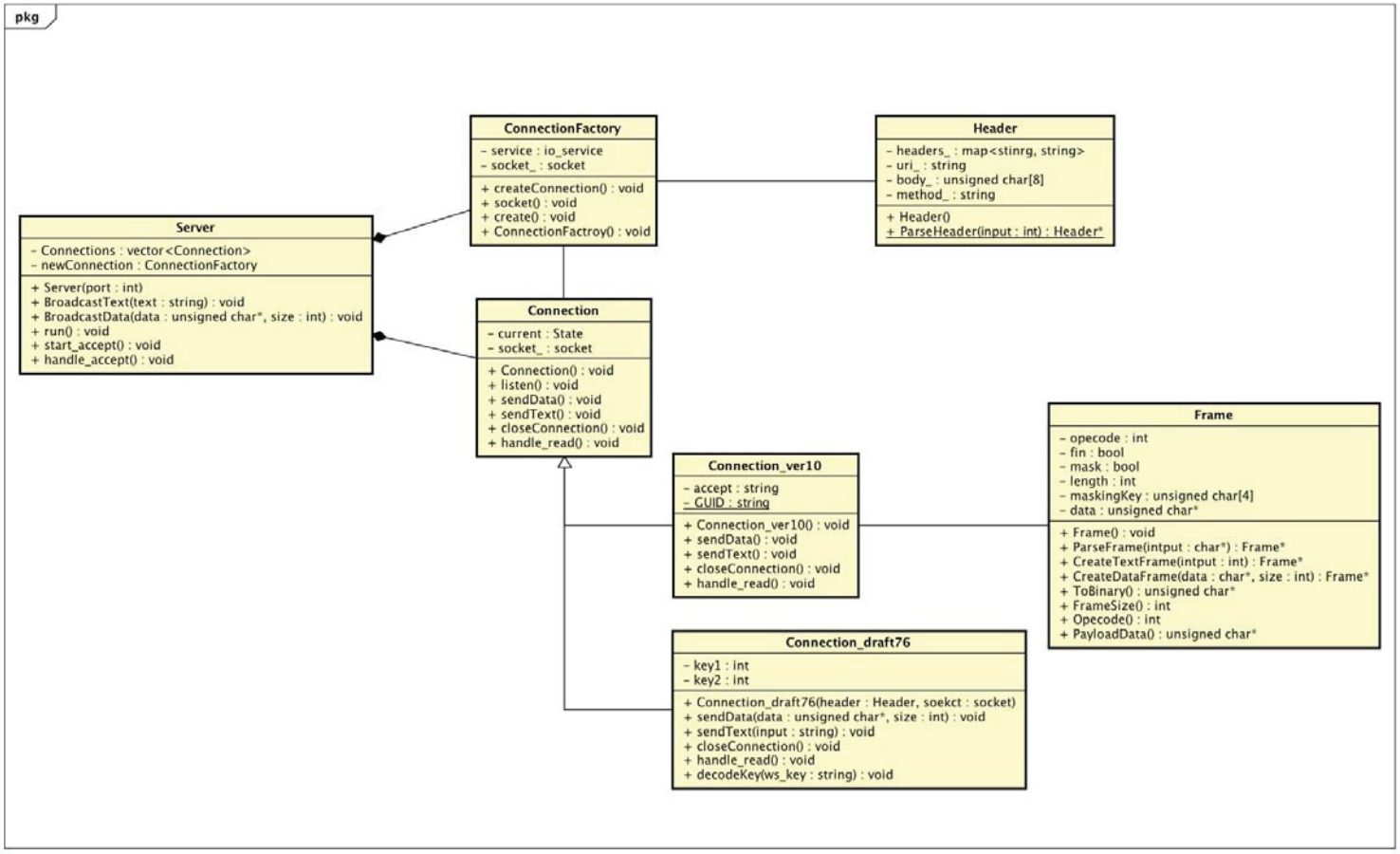
| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 1.0 | 2011/12/13 | | 茂木 |
| 1.1 | 2011/12/20 | PoseDetector の継承元を Interface から Abstract へ変更 | 茂木 |
| | | | |
| | | | |
| | | | |
| | | | |

# 目次

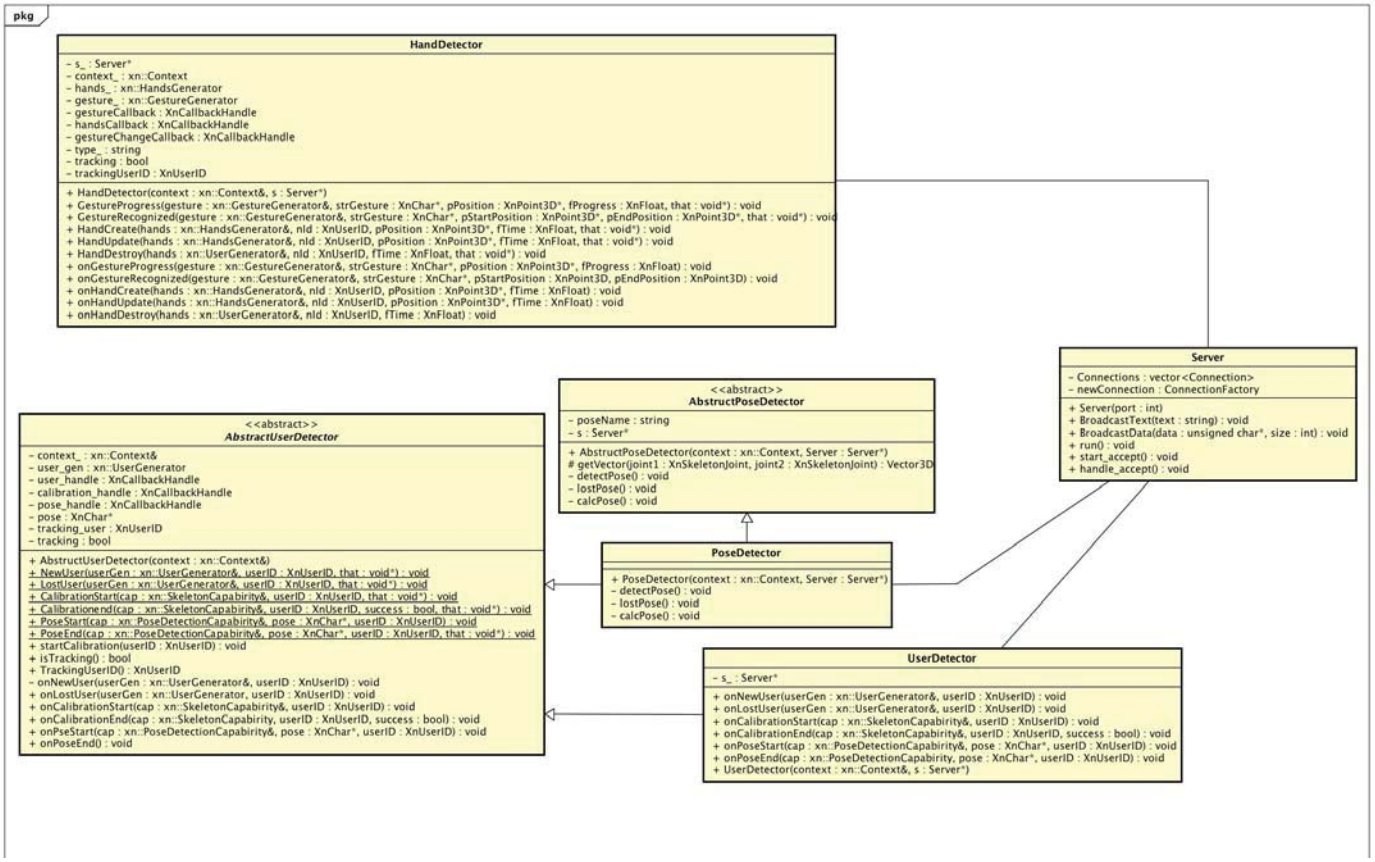
# 1. はじめに

この資料には、Websocket 通信部分のクラス設計を記述する。

# 2. WebSocket サーバ



## 3. Kinect データ取得部分



# Kinect サーバ

## サンプルクライアント詳細設計書

Ver1.0

Team. Kineco
茂木　昂士
小菅　拓真
朱　明
劉　斌

# 更新履歴

| 版数 | 日付 | 追加・更新箇所 | 担当 |
|---|---|---|---|
| 初版 | 8/25 | 棒人間 | 朱 |
| 1.0 版 | 10/15 | 3D 空間 | 朱 |
| 2.0 版 | 11/20 | ボックスのトランスポート | 朱 |
| 3.0 版 | 12/20 | 宇宙飛行 | 朱 |
| | | | |
| | | | |

# 目次

# 1. 概要

## 1.1. 目的

本文書では、「Kinect サーバおよびサンプルクライアント研究開発」報告書についての、サンプルクライアントの構築部分の詳細設計仕様を記述する。

## 1.2. 方針

WebGL を用いている Web ブラウザで Kinect サーバーからのリアルタイムデータを利用して、サンプルクライアントを開発するため、下記の点に重点を置く。

1. 要求定義書と基本設計書に定めたことに基づいて、設計を行う。
2. Kinect によって、提供できるデータに基づいて、設計を行う。
3. WebGL フレームワークの具体的な特性に基づいて、設計を行う。

## 1.3. 記載範囲

本文書はサンプルクライアントのソフトウェア構成、インタフェース、機能仕様を記載する。

## 1.4. 参照ドキュメント

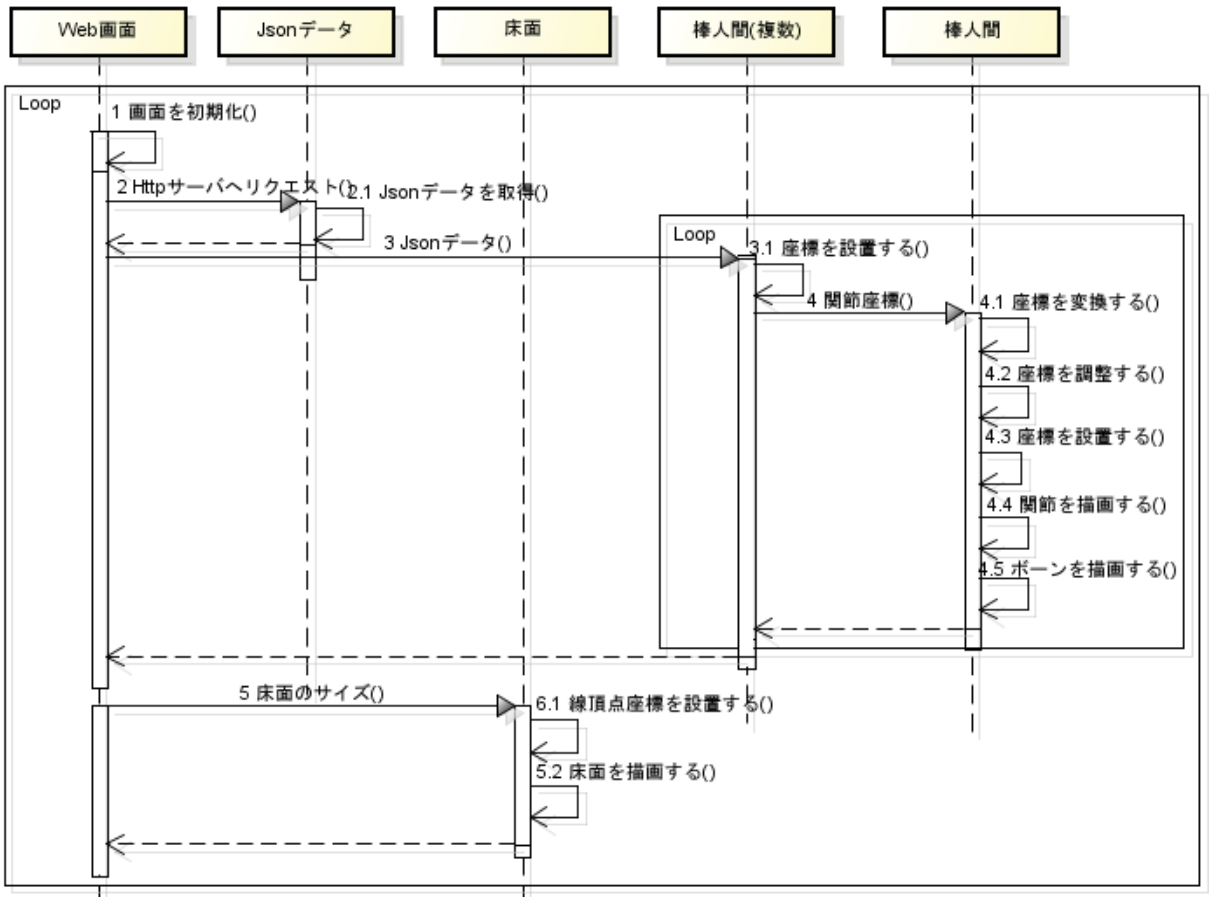
| ID | 文書名 | 文書番号 | 発行年月 | 備考 |
|---|---|---|---|---|
| | 要件定義書 | | | |
| | 基本設計書 | | | |
| | | | | |

## 1.5. 定義（用語、略語）

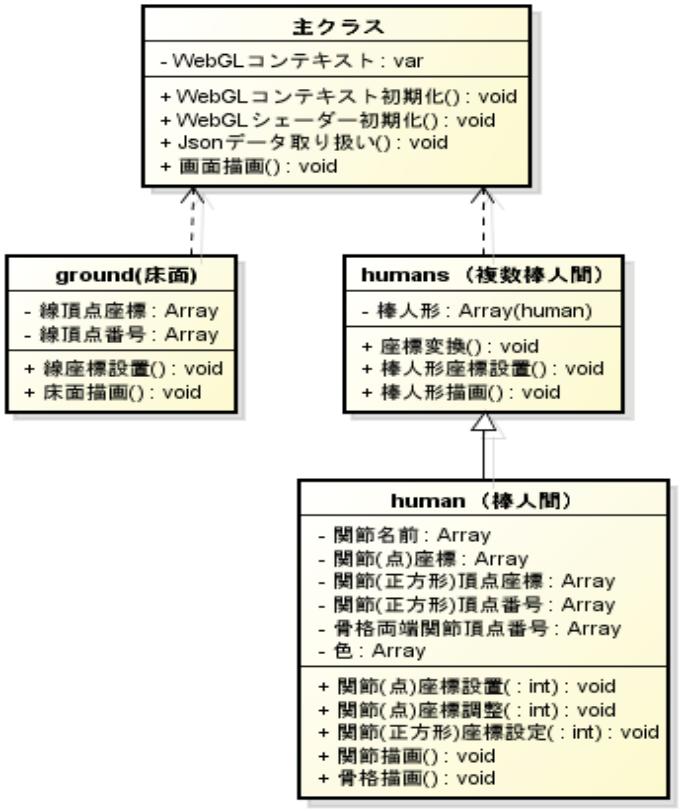
| ID | 用語・略号 | 正式表記 | 意味 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

# 2. 棒人間

## 2.1. シーケンス図



## 2.2. クラス図


主クラス
- WebGLコンテキスト : var
+ WebGLコンテキスト初期化() : void
+ WebGLシェーダー初期化() : void
+ Jsonデータ取り扱い() : void
+ 画面描画() : void
ground(床面)
- 線頂点座標 : Array
- 線頂点番号 : Array
+ 線座標設置() : void
+ 床面描画() : void
humans （複数棒人間）
- 棒人形 : Array(human)
+ 座標変換() : void
+ 棒人形座標設置() : void
+ 棒人形描画() : void
human （棒人間）
- 関節名前 : Array
- 関節(点)座標 : Array
- 関節(正方形)頂点座標 : Array
- 関節(正方形)頂点番号 : Array
- 骨格両端関節頂点番号 : Array
- 色 : Array
+ 関節(点)座標設置( : int) : void
+ 関節(点)座標調整( : int) : void
+ 関節(正方形)座標設定( : int) : void
+ 関節描画() : void
+ 骨格描画() : void

## 2.3. クラス詳細

| クラス名 | 主クラス | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 各システムパラメータの初期化とシーンの描画のコントロール | | | | |
| 外部ライブラリ | | | | |
| glMatrix-0.9.5.min.js | | | | |
| webgl-utils.js | | | | |
| jquery.min.js | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| public | gl | | | WebGL コンテンツ |
| public | shaderProgram | | | プログラマブルシェーダー |
| public | mvMatrix | | | システムパラメータ |
| public | pMatrix | | | システムパラメータ |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | initGL() | | canvas | canvas 初期化 |
| public | getShader | | gl | initShaders の子ファクション |
| public | initShaders | | | プログラマブルシェーダー初期化 |
| public | setMatrixUniforms | | | 変換行列初期化 |
| public | loadKinect() | | | Json データを取得する |
| public | drawScene() | | | シーンをレンダラーする |
| public | webGLStart() | | | Web ブラウザの起動メソッド |

| クラス名 | ground | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 床を描画する | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| private | groundVertexPositionBuffer | | | 頂点バファ |
| private | groundVertexColorBuffer | | | 頂点の色バファ |
| private | yLookAt | | | Y 軸の平行移動距離 |
| private | lineColor | | | 線の色 |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | initGround | | gl | 床の座標を定義する |
| public | drawGround | | gl | 床を描画する |

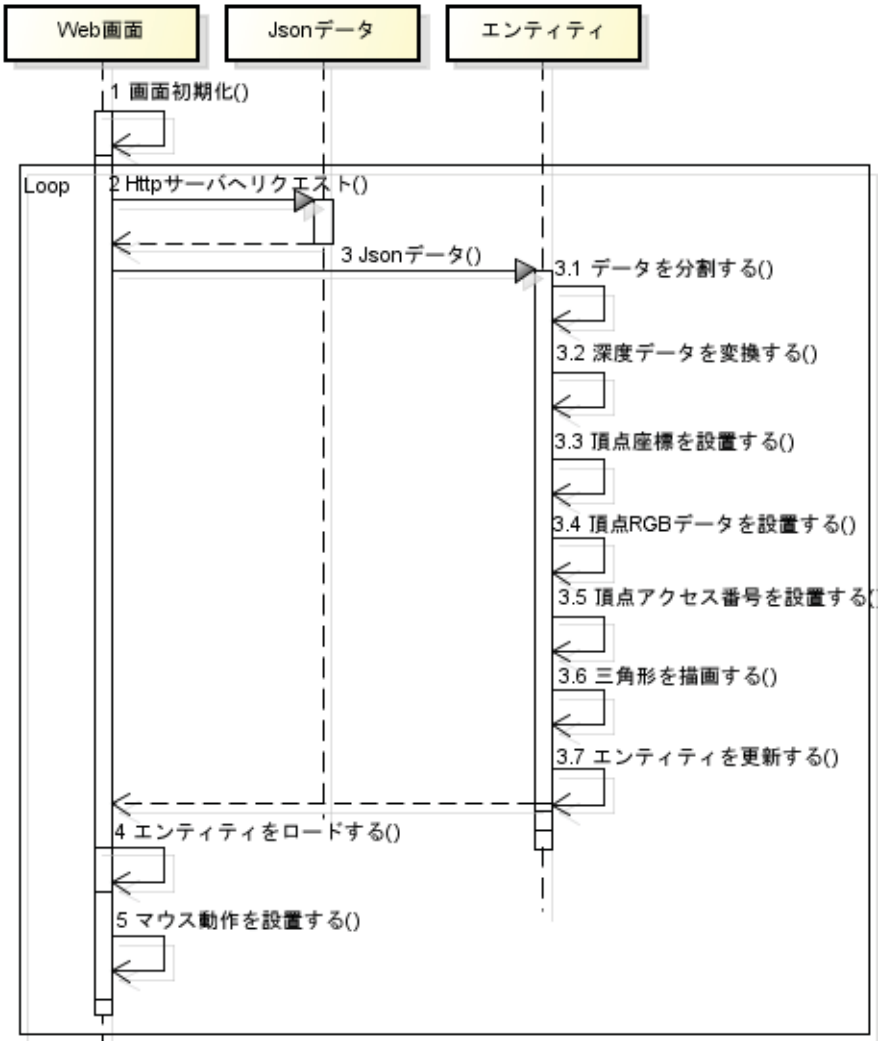
| クラス名 | humans | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 複数棒人間を描画する | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| private | users | | | userid 配列 |
| private | human | | | human クラスオブジェクトの配列 |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | findUser | | userid | userid を存在したかを判断する |
| public | loadJson | | jsonData | 関節座標を変換する |
| public | drawHumans | | gl | 複数棒人間を描画する |

| クラス名 | human | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 1 つの棒人間を描画する | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| private | parts | | | 関節の名前配列 |
| private | partPosition | | | 関節（点）の座標配列 |
| private | partIndices | | | ボンの頂点配列 |
| private | partJoints | | | 関節（四角形）の頂点配列 |
| private | partJointIndices | | | 関節（四角形）の頂点インデックス配列 |
| private | userColor | | | 棒人間の色配列 |
| private | bodyVertexPositionBuffer | | | 関節（四角形）の頂点バファ |
| private | bodyVertexColorBuffer | | | 関節（四角形）の色バファ |
| private | bodyVertexIndexBuffer | | | 関節（四角形）の頂点インデックスバファ |
| private | boneVertexPositionBuffer | | | ボンの頂点バファ |
| private | boneVertexColorBuffer | | | ボンの色バファ |
| private | boneVertexIndexBuffer | | | ボンの頂点インデックスバファ |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | setpartPosition | | ids<br>xpos<br>ypos<br>zpos | 関節（点）ids の XYZ 座標を設定する |
| public | setpartJoints | | | 2 分法で不適切な座標を処理すると関節（四角形）の座標を定義する |
| public | initpartPosition | | | 関節（点）の座標配列初期化 |
| public | initBuffer | | | 頂点・色・インデックスバフ |

| | | | | |
|---|---|---|---|---|
| | | | | ァを設定する |
| public | drawHuman | | | 1 つの棒人間を描画する |

# 3. 3D 空間

## 3.1. シーケンス図

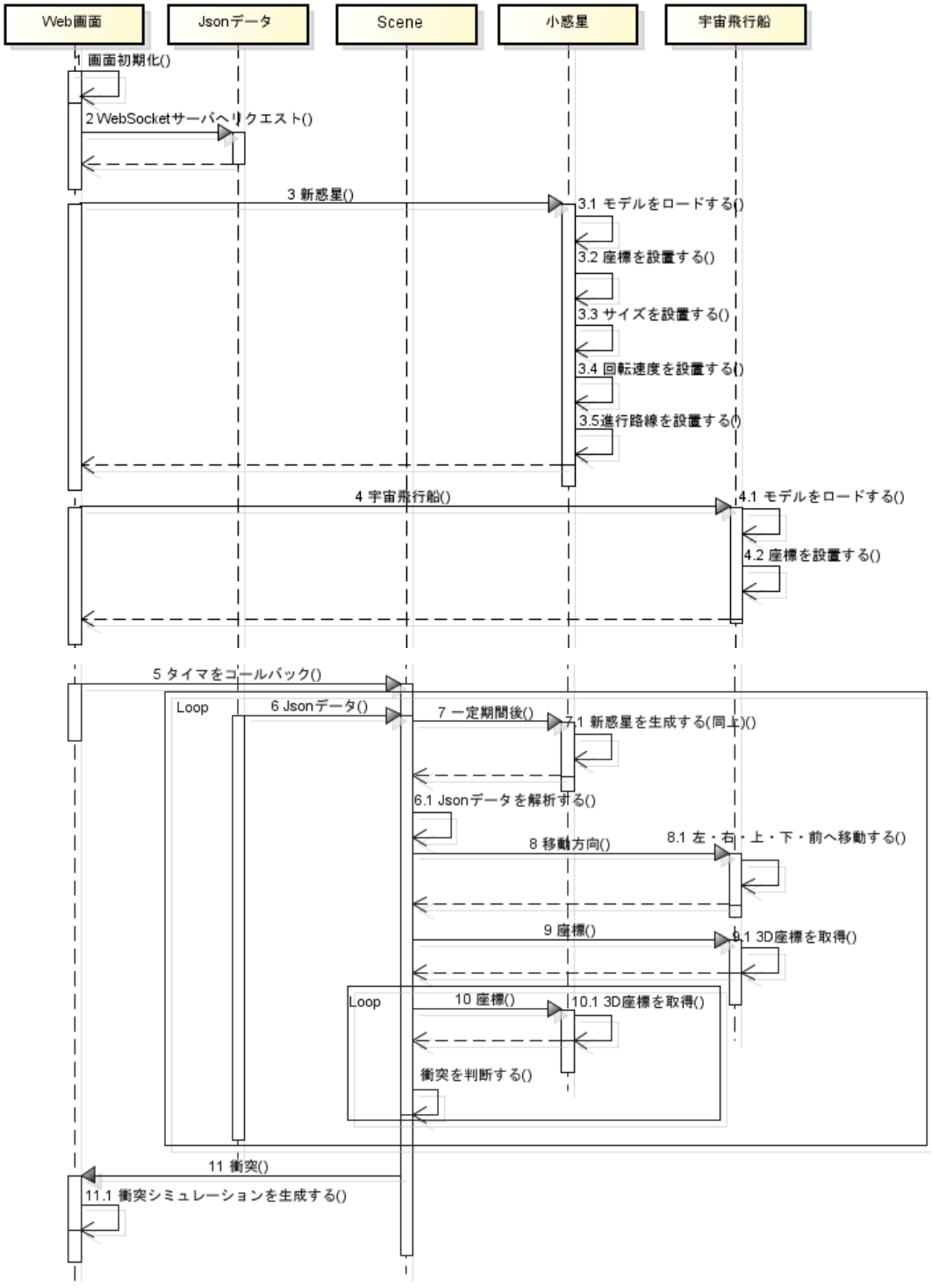


### 3.2. クラス詳細

| クラス名 | 主クラス | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 各システムパラメータの初期化とシーンの描画のコントロール | | | | |
| 外部ライブラリ | | | | |
| Oak3D_v_0_4_6_beta.js | | | | |
| jquery.min.js | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| public | CANVAS_WIDTH | | | canvas の長さ |
| public | CANVAS_HEIGHT | | | canvas の高さ |
| public | canvas | | | canvas オブジェクト |
| public | mouseLastX | | | マウスのイベントのパラメータ |
| public | mouseLastY | | | マウスのイベントのパラメータ |
| public | mouseDown | | | マウスのイベントのパラメータ |
| public | engine | | | Oak3D のエンジン |
| public | scene | | | シーン |
| public | cam | | | カメラ |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | loadJson() | | | Json データを取得する |
| public | drawModel | | jsonObj | エンティティに RGB+D データを設定する |
| public | setCamera | | | カメラを設定する |
| public | initScene | | | シーンを初期化する |
| public | renderScene | | | シーンをレンダラーする |
| public | onMouseDown | | | マウスのイベント |
| public | onMouseUp | | | マウスのイベント |
| public | onMouseMove | | | マウスのイベント |
| public | onLoad() | | | Web ブラウザの起動メソッド |

# 4. 宇宙飛行

## 4.1. シーケンス図



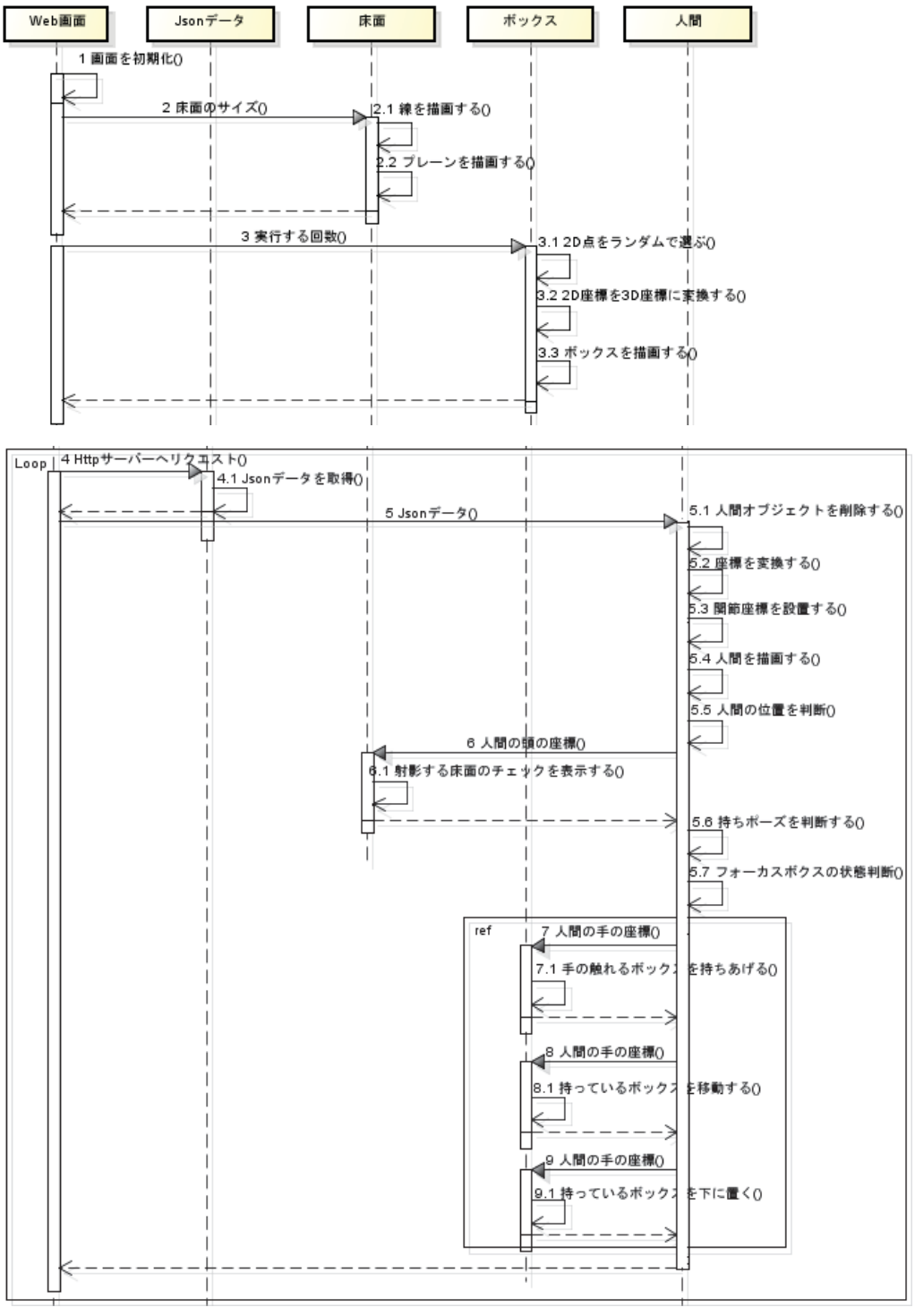
## 4.2. クラス詳細

| クラス名 | 主クラス | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 各システムパラメータの初期化とシーンの描画のコントロール | | | | |
| 外部ライブラリ | | | | |
| c3dapi.js | | | | |
| fly_plane_tri.dae | | | | |
| skysphere.dae | | | | |
| asteroid2.dae | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| public | screen | | | シーン |
| public | planets | | | 小惑星配列 |
| public | plane | | | 宇宙飛行船 |
| public | simulation | | | シミュレーションオブジェクト |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | update | | time | シーンのコールバックファンクション |
| public | collision | | | 衝突を判断する |
| public | distance | | v1<br>v2 | 2 つベクトルのノルムを計算する |
| public | moveLeft | | | 宇宙飛行船が左へ移動する |
| public | moveRight | | | 宇宙飛行船が右へ移動する |
| public | moveUp | | | 宇宙飛行船が上へ移動する |
| public | moveDown | | | 宇宙飛行船が下へ移動する |
| public | moveFront | | | 宇宙飛行船が前【加速】へ移動する |
| public | moveBack | | | 宇宙飛行船が後【後退】へ移動する |
| public | initPlanet | | | 新しい小惑星初期化 |
| public | initPlane | | | 宇宙飛行船初期化 |
| public | initExplosion | | | 衝突シミュレーション初期化 |

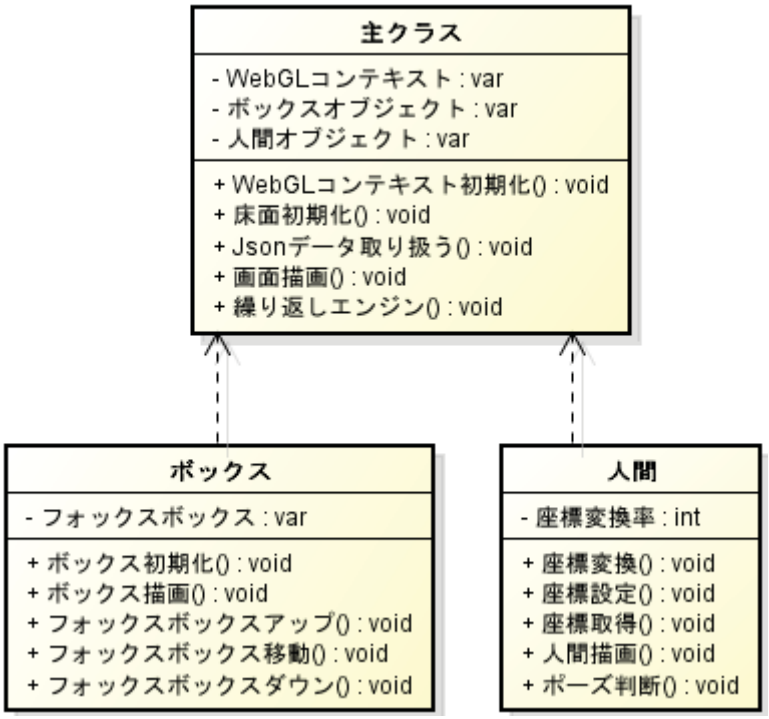
| | | | | |
|---|---|---|---|---|
| public | initRoll | | | 小惑星の回転速度を設定する |
| public | loadKinect | | | Json データを取得する |
| public | canvasMain | | | Web ブラウザの起動メソッド |

# 5. ボックスのトランスポート

## 5.1. シーケンス図



## 5.2. クラス図


主クラス
- WebGLコンテキスト : var
- ボックスオブジェクト : var
- 人間オブジェクト : var
+ WebGLコンテキスト初期化() : void
+ 床面初期化() : void
+ Jsonデータ取り扱う() : void
+ 画面描画() : void
+ 繰り返しエンジン() : void
ボックス
- フォックスボックス : var
+ ボックス初期化() : void
+ ボックス描画() : void
+ フォックスボックスアップ() : void
+ フォックスボックス移動() : void
+ フォックスボックスダウン() : void
人間
- 座標変換率 : int
+ 座標変換() : void
+ 座標設定() : void
+ 座標取得() : void
+ 人間描画() : void
+ ポーズ判断() : void

### 5.3. クラス詳細

| クラス名 | 主クラス |
|---|---|
| 親クラス | なし |
| クラス概要 | |
| 各システムパラメータの初期化とシーンの描画のコントロール | |
| 外部ライブラリ | |
| Three.js | |
| RequestAnimationFrame.js | |
| Stats.js | |
| Detector.js | |
| jquery.min.js | |
| 属性 | |

| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
|---|---|---|---|---|
| public | renderer | | | レンダー |
| public | mygl | | | WebGL コンテンツ |
| public | camera | | | カメラ |
| public | scene | | | シーン |
| public | projector | | | プロジェクタ |
| public | stats | | | リソース監視オブジェクト |
| private | plane | | | 床面プレン |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | initParameter | | | システムパラメータを設定する |
| public | animate | | | レンダーのコールバックファクション |
| public | loadKinect | | | Json データを取得する |
| public | render | | | シーンをレンダラーする |
| public | setStats | | | リソース監視オブジェクト初期化 |
| public | setCamera | | | カメラ初期化 |
| public | drawGrid | | scene | 床面初期化 |
| public | init | | | Web ブラウザの起動メソッド |

| クラス名 | boxes | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| ボックスの描画とフォックスボックスの状態変換 | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| private | voxels | | | ボックス配列 |
| private | voxel | | | ボックスオブジェクト |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | drawCube | | | 1つボックスを描画する |
| public | getPosition | | intersector | 床面との交差点が所属する格子の座標を計算する |
| public | getRealIntersector | | intersects | getFirstObjectの子ファクション |
| public | getFirstObject | | | シーンにカメラ射線と衝突オブジェクトを検査する |
| public | drawCubes | | camera , scene , projector | 複数ボックスを描画する |

| クラス名 | human | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| 球人間の描画と特定持ち上げポーズの認識 | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| private | parts | | | 関節の名前配列 |
| private | posts | | | 関節の座標配列 |
| private | ratio | | | 球人間に変換するα値 |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | setPosition | | name,<br>x,<br>y,<br>z | 関節 name の XYZ 座標を設定する |
| public | setPositions | | | α値を計算して、各関節の座標を設定する |
| public | drawSphere | | scene | 球を描画する |
| public | initHuman | | scene ,<br>camera ,<br>renderer,<br>context | 球人間を描画する |
| public | getHumanPosition | | | 球人間の頭座標を取得する |
| public | getHandPosition | | | 球人間の両手の中心座標を取得する |
| public | getPose | | | 特定持ち上げポーズかどうかを認識する |

| クラス名 | focusVoxel | | | |
|---|---|---|---|---|
| 親クラス | なし | | | |
| クラス概要 | | | | |
| フォックスボックスの状態変換 | | | | |
| 属性 | | | | |
| 可視性 | 属性・インスタンス名 | 型 | 初期値 | 説明、その他 |
| private | focusVoxel | | | フォックスボックス　オブジェクト |
| private | rollOveredFace | | | 球人間が立っている格子 |
| メソッド | | | | |
| 可視性 | メソッド名 | 型 | 引数 | 説明、その他 |
| public | initrollOveredFace | | | 球人間の立っている格子が赤く設置する |
| public | translateCube | | intersector<br>dx,<br>dy,<br>dz | フォックスボックスを移動する親ファクション |
| public | upVoxel | | sobject | フォックスボックスを持ち上げる |
| public | downVoxel | | sobject | フォックスボックスを運搬する |
| public | moveVoxel | | sobject | フォックスボックスを下ろし置く |