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
—WebSocket によるサーバ・プッシュ機能の実現—

茂木　昂士

(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

本プロジェクトでは，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，同大学院に所属する高橋伸准教授から受注したものである．

受注元の教員からの依頼で Microsoft Kinect センサーを利用し，取得したカメラや深度センサーのデータ，骨格データを配信するサーバ，および提供されたデータを利用するサンプルクライアントを作成した．本プロジェクトの目標は作成したシステムによって，Kinect センサーを用いた開発を支援することである．

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博士前期課程 2 年に所属する，筆者を含む 4 名のチーム Kineco によって進められた．

開発において筆者は Kinect サーバの機能のうち，WebSocket を利用したサーバ・プッシュ機能の検討，実現と，クライアントとの連携を行うための設計，サンプルポーズの作成を行った．

サーバ・プッシュ機能の検討では，問題点を解決するためにサーバ・プッシュ技術の調査及び比較を行った．HTTP 通信のみで実装した場合，骨格データ取得の際に HTTP リクエストが増大してしまう，トラッキング開始のタイミングが取得できないといった問題が発生してしまう．この問題を解決するために Comet，WebSocket というサーバ・プッシュ技術の調査，比較を行った．また設計，実装工程では WebSocket の仕様変更があった場合に対応すること，独自通信方式を採用することを考慮して設計，実装を行った．

クライアントとの連携，サンプルポーズの作成では，クライアント連携部分で利用される通知方式の設計を行った．また独自ポーズ作成を行う利用者向けに，ポーズ作成に必要なユーザのトラッキング，骨格データ取得を行う機能を持ったスーパークラスと，そのクラスを利用したサンプルポーズの作成を行った．

本報告書では本プロジェクトの内容及び,本プロジェクトにおいて筆者が担当した役割と，筆者が開発を担当したサーバ・プッシュ機能及び，当機能を利用したクライアントとの連携について報告する．

目次

図目次

表目次

第1章　はじめに

1.1　プロジェクト概要

Microsoft 社から発売されている Xbox360 向けコントローラ Kinect は，搭載されている RGB カメラ，深度センサーにより人物の位置や動きを認識することができる[1]．Kinect が取得した情報は USB 接続により PC でも利用が可能であり，Kinect を用いた様々なプログラムが開発されてきた．

また Web ブラウザの進化により，3 次元コンピュータグラフィックスを高速に描画するための WebGL や，サーバとブラウザ間で双方向通信を行うための WebSocket などの技術が開発されている．これらの技術により，Web ブラウザでより高度な処理を行うことが可能になっている．

本プロジェクトでは，Kinect を用いて取得した情報をライブカメラの用にリアルタイムで配信するサーバを構築し，WebGL などの技術を用いて Web ブラウザで閲覧可能にすることを目的とする．Kinect からのデータを Web ブラウザで扱うことができるようにすることで，3D データを用いた研究や Kinect を利用したアプリケーションの開発を容易にすることができる．

1.2　本報告書の構成

本報告書は 9 章で構成されている．

第 2 章では本プロジェクトに必要な前提知識として，Microsoft Kinect センサーとそのドライバソフトウェアである OpenNI について記述する．

第 3 章では顧客からの要望の詳細，及び分析内容について記述する．

第 4 章では本プロジェクトの全体像及び開発するシステムの各機能，システムのソフトウェア，ハードウェア構成について記述する．

第 5 章では筆者が担当した WebSocket によるサーバ・プッシュ機能の実現について，及びサーバ・プッシュ機能の検討について記述する．

第 6 章ではサーバ・プッシュ機能を用いた，クライアントとの連携について記述する．

第 7 章では本プロジェクトの体制，進め方及びスケジュールについて記述する．

第 8 章ではプロジェクトを振り返り，各工程とプロジェクト全体に対して良かった点と改善点について記述する．

第 9 章では本プロジェクトのまとめについて記述する．

第2章　前提知識

2.1　Kinectセンサー

Kinect センサー（図 2-1）とは，Microsoft 社から発売されている Xbox360 向けのコントローラである．RGB カメラ，深度センサー，マルチアレイマイクロフォンといったハードウェアと，専用のソフトウェア，プロセッサを搭載している[2]．これらのプレイヤーの動きや姿勢をリアルタイムに認識し，コントローラを持たずにゲームをプレイすることができる．また, Kinect センサーが取得した情報は USB 接続により PC でも利用が可能であり, Kinect を用いた様々なプログラムが開発されてきた．

図 2-1. Microsoft Kinect センサー[3]

2.2　OpenNI

OpenNI（Open Natural Interaction）は Kinect センサーなどの３D センサーデバイスを利用するためのインターフェースであり, Kinect センサーに搭載されている赤外線深度センサーを開発した Prime Sense 社からオープンソースソフトウェアとして提供されている．デバイスのコントロールをする Hardware Device 部分とそのデータに対して画像処理を行い，ジェスチャー認識などを行う Middleware Components 部分からなる（図 2-2）．

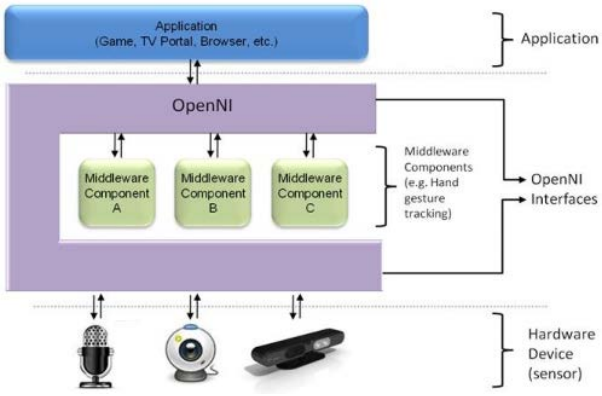


図 2-2. OpenNI の構成[5]

第3章　　顧客からの要望

この章では，このプロジェクトでの顧客からの要望を記述する．次に，顧客との話し合いを通じて決定した本システムが対象とする利用者，システムの要件を記述する．

3.1　　顧客からの要望

顧客からの要望は，ネットワークを通じて Kinect センサーのデータを配信するサーバを構築するというものである．詳細な内容を図 3-1 に示す．

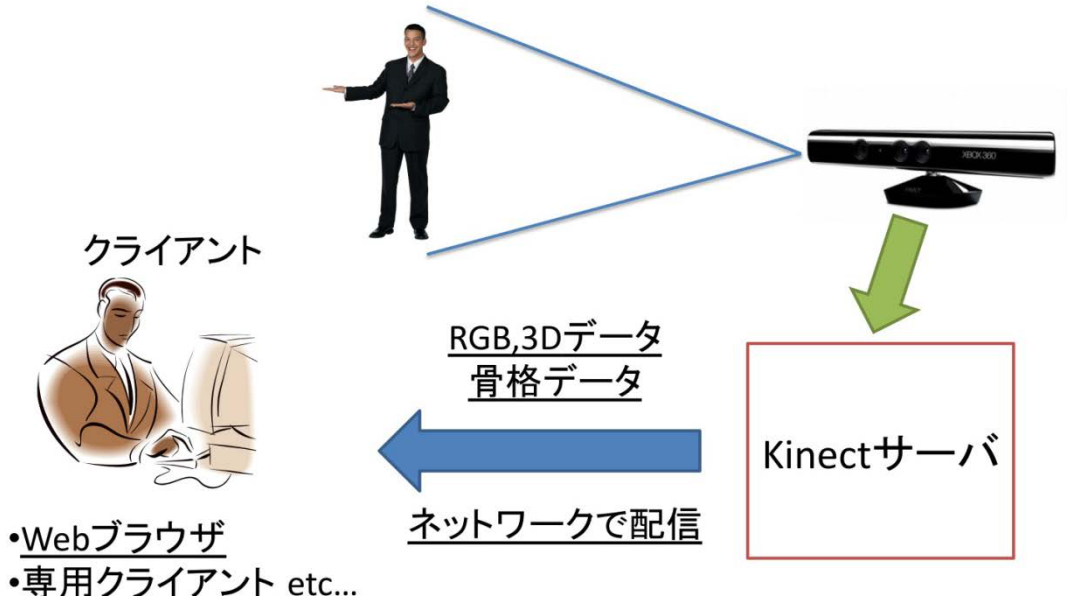


図 3-1．顧客からの要望

3.1.1　　Kinectサーバ

Kinect センサーから取得したデータを，リアルタイムでネットワーク配信する．配信するデータは RGB カメラデータ，深度センサーによる 3D データ，骨格データなど．

3.1.2　　クライアント

Kinect サーバから取得したデータは，専用のクライアントや Web ブラウザから利用する．特に，Web ブラウザでは WebGL[6]などを用いて 3D データの表示を行う．

3.2　　対象とする利用者

本システムは以下の二種類の利用者を対象とした．

3.2.1　　研究用途での利用者

研究用途での利用者は主に解析などにデータを利用するため，Kinect から取得したそのままのデータが必要になる．そのため，jpeg などの非可逆圧縮は利用せず，ピクセルのデータ配列を送信する．

3.2.2　　アプリケーションの開発者

アプリケーション制作を行う利用者は研究用途での利用者と違い，サーバから取得したデ

ータをそのまま利用する．そのため，サーバ側で Kinect のデータに画像化などの処理を加えたものを提供する．これは，クライアント側での処理を減らすことで，性能が低いクライアントでもデータを利用しやすくするためである．

3.3 機能要件

本システムは，利用者に対して以下に示すデータの提供を行う．

3.3.1 RGBカメラ，深度センサーのバイナリデータ

Kinect から取得した RGB カメラのデータ，深度センサーのバイナリデータ，または 1 ピクセルごとのデータを送信する．また，通信帯域が狭い場合を考慮して，それぞれ低解像度のデータも用意する．提供するデータのピクセル数，及び 1 ピクセルあたりのビット数の一覧を表 3-1 に示す．

表 3-1．提供するバイナリデータのサイズ一覧

	サイズ（横×縦）[pixel]	1[pixel]あたりのサイズ[bit]
RGB	640×480	24
深度	320×240	10
RGB 低解像度	320×240	24
深度低解像度	160×120	10

3.3.2 RGB，深度の画像データ

Kinect から取得した RGB カメラ，深度センサーのデータを jpeg 形式に変換した画像データを送信する．RGB カメラのデータは 24bit カラー，深度センサーのデータは 10bit のデータを 8bit に丸め込んだグレイスケールカラーの画像を提供する．

3.3.3 ユーザの骨格データ

Kinect から取得した骨格情報のジョイントポイントの座標，距離を送信する．取得できるジョイントポイントの一覧を図 3-2 に示す．

図 3-2．Kinect センサーで取得できる骨格情報のジョイントポイント一覧[7]

送信するデータはブラウザでの利用を考慮して JSON[8]を利用する．提供する JSON データの形式を図 3-3．提供する骨格データの形式に，データの座標範囲を表 3-2 に示す．

```
{
    "part": "head",
    "coordinate": {"x":20, "y":180, "z":20 }
}
```

図 3-3．提供する骨格データの形式

表 3-2．骨格データの座標範囲

座標	範囲
x	-320 < x < 320
y	-240 < y < 240
z	0 < z < 1000

3.3.4 ユーザ検出データ

Kinect のセンサー範囲内に入ったユーザの ID および，ユーザの状態(Detect，Lost)の情報をクライアントに通知する．

3.3.5 ポーズ検出データ

トラッキング中のユーザが指定したポーズを行った際の情報を通知する．ポーズは利用者が独自に作成することが可能であり，ポーズ作成を支援するためのクラス，サンプルを作成する．

第4章　　プロジェクトの概要

この章では，開発する Kinect サーバの概要を記述する．まず，プロジェクトの全体像について記述する．次に Kinect サーバの機能，及びサーバから提供されるデータ形式の検証内容，複数サーバ連携，サンプルクライアントについて記述する．最後にシステムの構成について記述する．

4.1　　プロジェクトの全体像

プロジェクトの全体像と筆者の担当した部分を図 4-1 に示す．

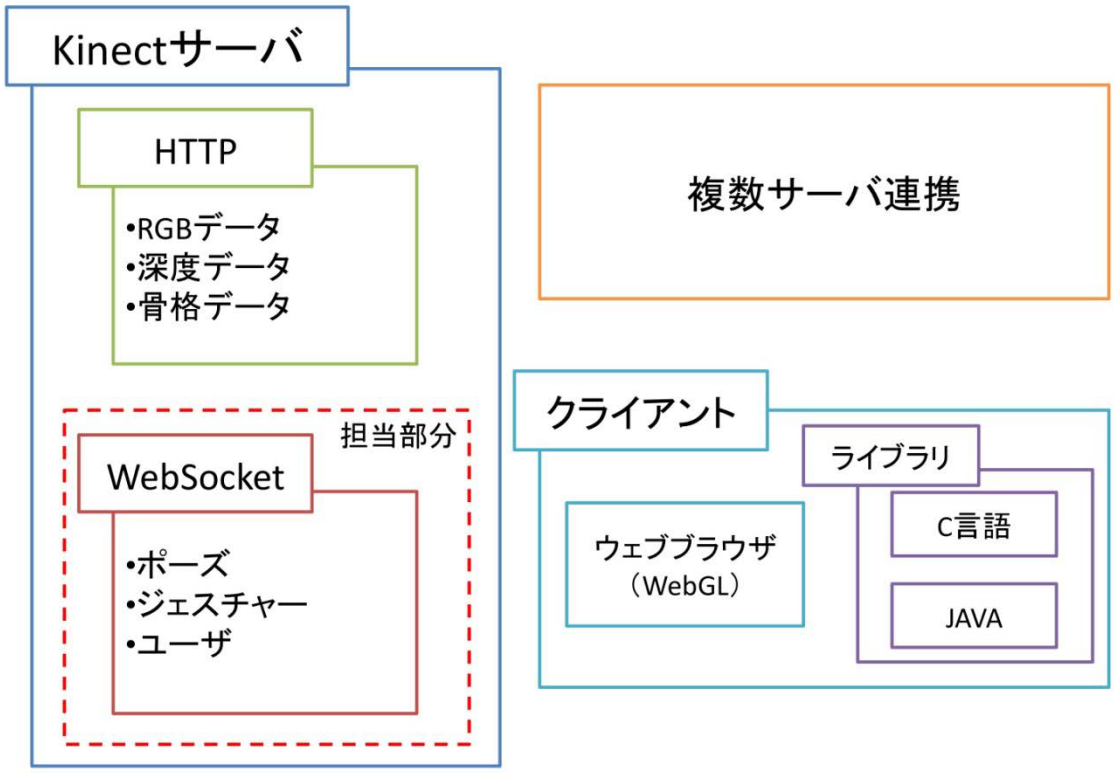


図 4-1．システムの全体像と筆者の担当部分

各部分に関して以下に記述する．

4.2　　Kinectサーバ

HTTP と WebSocket により，Kinect センサーの情報をクライアントに提供する．RGB カメラ，深度センサー，骨格のデータは HTTP 通信を用いて送信し，ポーズやユーザの認識，ジェスチャーといったサーバ側で発生するイベントは WebSocket を用いてクライアントへ通知する．

RGB カメラ，深度センサーのデータに関してはデータ量が多いため，データ量削減のために提供するデータの形式を検討し，プロトタイプを用いて通信速度の検証を行った．検討したデータ形式と，検証結果を以下に記述する．

4.2.1　　JSON形式でのRGBデータ送信

Kinect から取得した RGB カメラのデータを JSON 形式に格納し，クライアントへの送信

を検証した．RGB カメラのデータは，RGB の各データを数値化して JSON に格納した（図 4-2）．

```
{
    "pixels": [
        {"R": 255,"G": 150,"B": 230},
        …,
        {"R": 20,"G": 10,"B": 0}
    ]
}
```

図 4-2. JSON 形式に格納した RGB のデータ

4.2.2 バイナリ形式でのRGBデータ送信

4.2.1 の方法ではデータ量が増加してしまうため，基のバイナリデータを Base64 エンコード[9]で文字列化することで，JSON に格納できるようにした(図 4-3).

```
{
"pixels": “QUJDRE … GRw=”
}
```

図 4-3. JSON にバイナリデータを格納

4.2.3 画像（jpeg形式）データ送信

Kinect センサーから取得した RGB カメラのデータを jpeg 形式で送信した．Kinect センサーのデータから jpeg 形式への変換は OpenCV の imencode 関数を用いた．しかし，画像データの形式では Ajax などで利用することができず，リロードでしか最新情報が取得できない．そこで，Data scheme URI[10]というスキームを利用して，img タグや canvas タグに直接画像データを描画する方法を使用した．

```
<img
src=”<data>”>
```

図 4-4. Data Scheme URI の利用例(img タグ)

Data Scheme URI では，Base64 エンコードされた画像データを図 4-4 の形式で描画することができる．この方法を使用することで，Ajax で取得した画像データを描画することができるようになる(図 4-5).

```
var data = "" + recv;
$("#image").attr("src",data);
```

図 4-5. クライアント側での利用例

4.2.4 通信速度の比較

各形式でのデータサイズ，データ生成時間，通信速度の比較を　表　に示す.

表 4-1. データサイズ，データ生成時間，通信速度の比較

	サイズ[MB]	データ生成時間[s]	通信時間[s]
JSON 形式	7～10	約 4	約 6
バイナリ形式	約 1.4	0.02～0.04	0.5～0.7
jpeg 形式	約 0.01	0.02～0.04	0.05～0.07

この結果から，バイナリ形式と jpeg 形式を採用した.

4.3 複数サーバ連携

複数台の Kinect サーバを連携させることで，1 台の Kinect センサーが取得できるデータよりも詳細なデータを取得できる.

4.4 サンプルクライアント

サンプルクライアントとして WebGL を利用した 3 D データの表示，骨格データの表示，ポーズ作成のサンプルを作成した.

4.5 構成

本システムの開発におけるハードウェア，ソフトウェアの構成を以下に示す.

4.5.1 ハードウェア構成

本プロジェクトで使用する各サーバのハードウェア仕様を表 4-2 に示す.

表 4-2. システムのハードウェア構成

サーバ	Dell　Vostro200
CPU	Intel（R）Core 2Duo
メモリ	4GB
ハードディスク	450G

4.5.2 ソフトウェア構成

本プロジェクトで構築した Kinect サーバのソフトウェア構成を表 4-3 に示す.

表 4-3. サーバのソフトウェア仕様

サーバ OS	Ubuntu 11.2
ライブラリ	libmicrohttpd 0.9.11 Boost C++ Library 1.47 (thread, system) OpenNI 1.3
開発言語	C++

第5章　WebSocketによるサーバ・プッシュ機能の実現

この章ではサーバ・プッシュ機能に実現について記述する．まず，サーバ・プッシュ機能の概要と Kinect サーバにおいて，筆者が担当したユーザ，ジェスチャー，ポーズのサーバ・プッシュ機能の必要性について記述する．次にサーバ・プッシュ手法の比較と，WebSocket について記述する．最後にその実装と工夫点について記述する．

5.1　HTTP通信とサーバ・プッシュの違い

通常の HTTP 通信の手順について図 5-1 に示す．

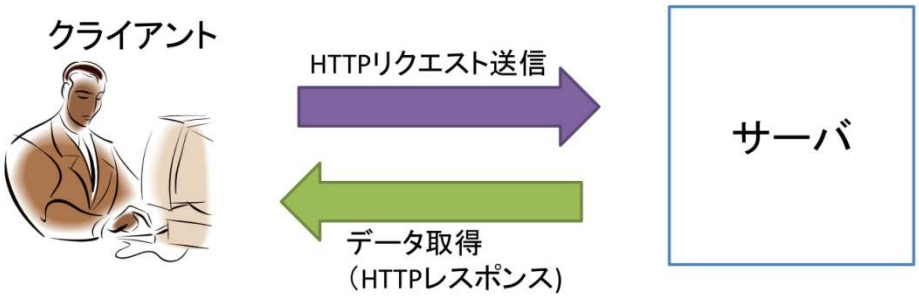


図 5-1．HTTP による通信の手順

HTTP 通信ではクライアントから HTTP リクエストを送信し，そのレスポンスとしてデータを取得する．そのため，データ取得のタイミングはクライアント側の実装方法に依存する．一方，サーバ・プッシュ通信では図 5-2 の様にサーバ側のタイミングでデータを送信することができる．

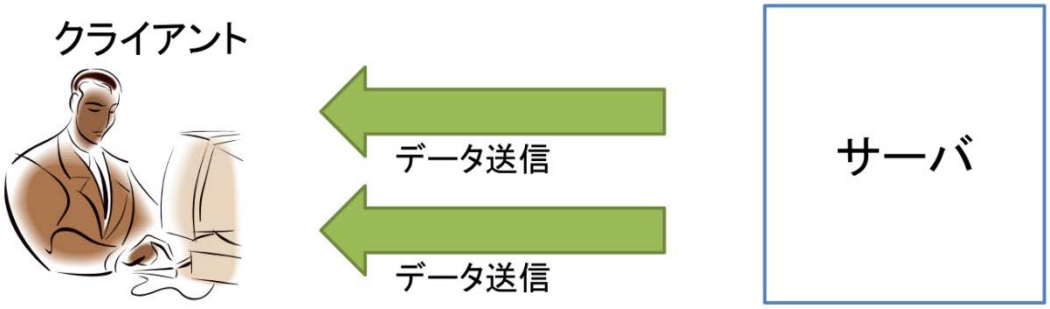


図 5-2．サーバ・プッシュによる通信

このためサーバ・プッシュ通信では，サーバ側で発生したイベントをクライアントに通知することが可能になる．

5.2　　サーバ・プッシュ機能の必要性

5.2.1　　Kinect＋OpenNI環境での骨格データ取得までの流れ

Kinect＋OpenNI の環境でユーザの骨格データを取得するまでの流れを図 5-3 に示す.

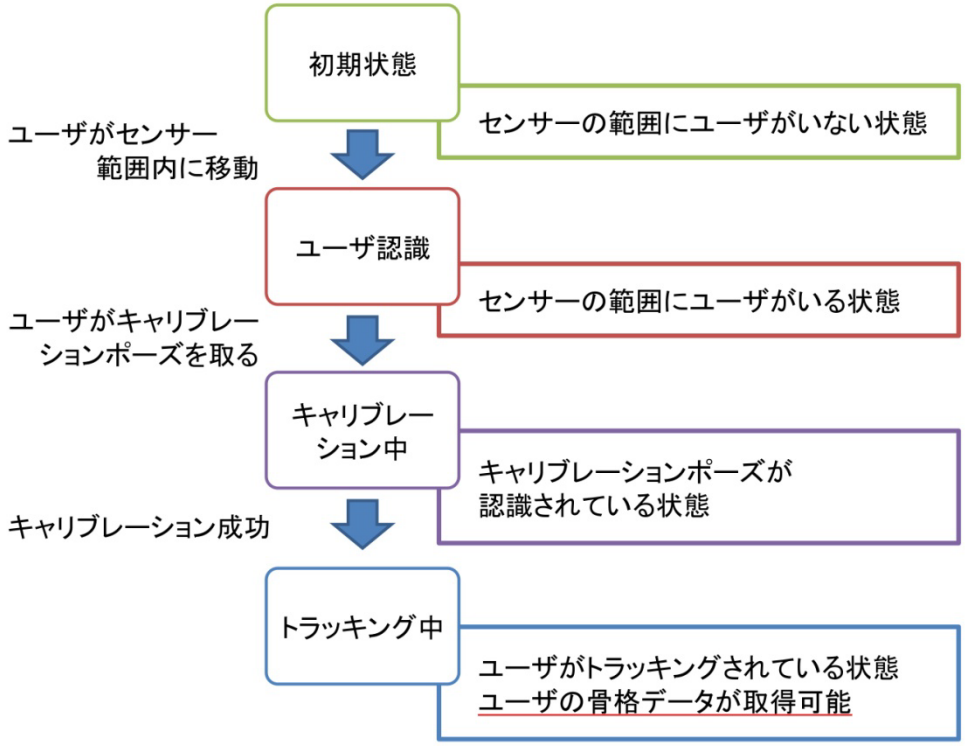


図 5-3. ユーザの骨格データ取得までの流れ

◆　初期状態

Kinect センサーの範囲内にユーザがいない状態．この状態でユーザの ID や位置情報，ユーザの骨格データを取得することはできない．センサーの範囲内にユーザが移動することでユーザ認識状態に遷移する．

◆　ユーザ認識

Kinect センサーの範囲内にユーザがいる状態．この状態ではユーザの ID や位置情報を取得することができる．ユーザがセンサーの範囲外に出ると，初期状態に戻る．ユーザがキャリブレーションポーズを取ると，キャリブレーション中の状態に遷移する．特に設定を行わない場合，キャリブレーションポーズは図 5-4 に示す Psi ポーズ[11]になる．

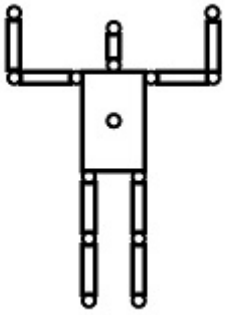


図 5-4. Psi ポーズ

- キャリブレーション中
  ユーザがキャリブレーションポーズを取っている状態．一定時間ポーズを取り続け，キャリブレーションが成功すると，トラッキング中の状態に遷移する．キャリブレーションが失敗した場合は，ユーザ認識状態に戻る．
- トラッキング中
  ユーザのキャリブレーションが成功し，トラッキングを開始した状態．この状態になることで，ユーザの骨格情報が取得可能になる．

5.2.2 HTTP通信のみで骨格データを取得する

HTTP 通信のみを利用した場合の骨格データ取得の流れを図 5-5 示す．

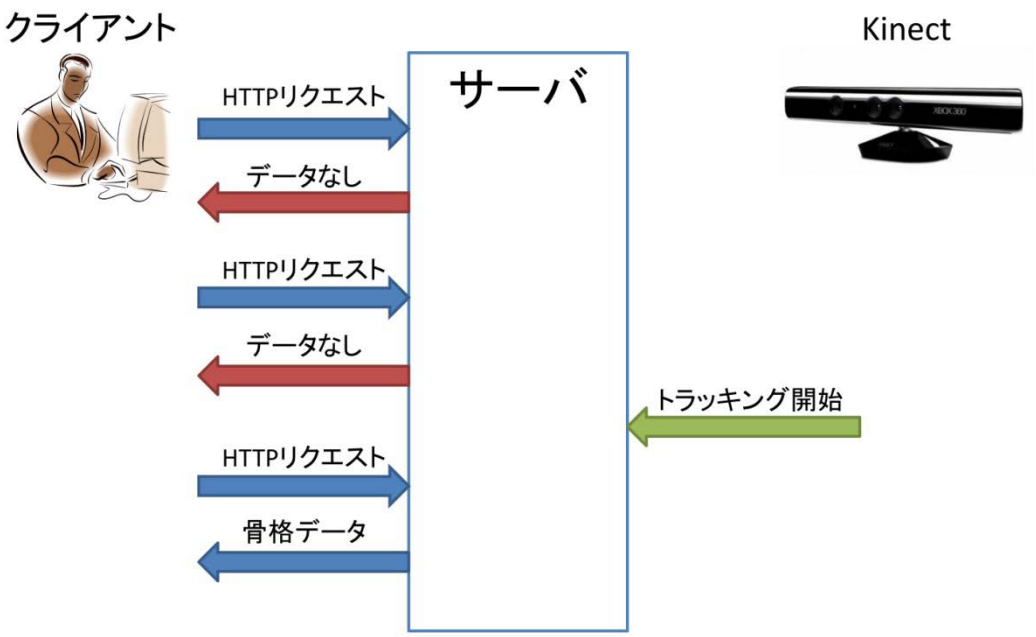


図 5-5. HTTP 通信のみを利用した骨格データ取得の流れ

この流れでは以下の 2 つの問題が発生する．

- HTTP リクエストの増大
  クライアント側ではサーバ側の状態を取得できないため，サーバ側の状態に関わらず HTTP リクエストを送信し続ける必要がある．

◆ トラッキング開始のタイミングが取得できない

HTTP リクエストのタイミングはクライアント側に依存するため，クライアント側がトラッキング開始のタイミングを取得することができない．HTTP リクエストの間隔を短くすることでトラッキング開始のタイミングに近づけることはできるが，その分 HTTP リクエストが増大してしまう．

5.2.3 サーバ・プッシュ機能による改善

HTTP 通信のみを利用した場合に発生する問題は，サーバ・プッシュ機能によって図 5-6 の様に解決できる．

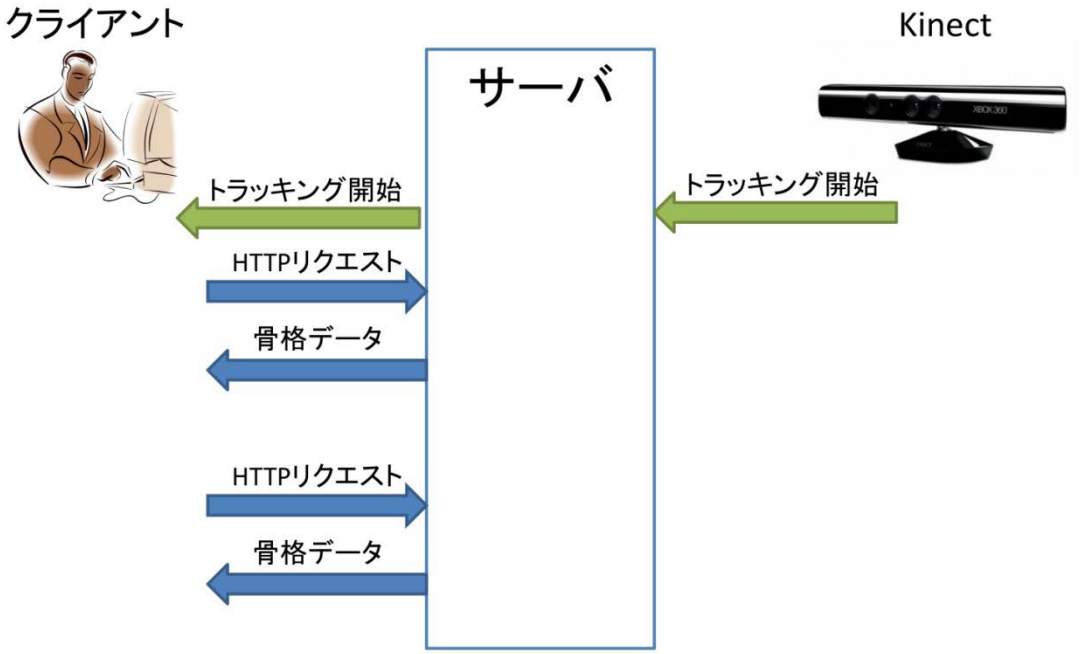


図 5-6. サーバ・プッシュによる改善

サーバ側はトラッキング開始をクライアント側に通知し，通知を受けたクライアントが骨格データの取得を開始するため，HTTP リクエストの量を軽減することができる．

5.3 サーバ・プッシュ手法の検討

サーバ・プッシュの主な手法として Comet，WebSocket がある．以下，この 2 つについて記述する．

5.3.1 Comet

Comet とはクライアントからの HTTP リクエストを送信した後に，タイムアウトまたはイベントが発生するまでリクエストを長引かせるという手法である[12]．Comet によるサーバ・プッシュの実現を図 5-7 に示す．

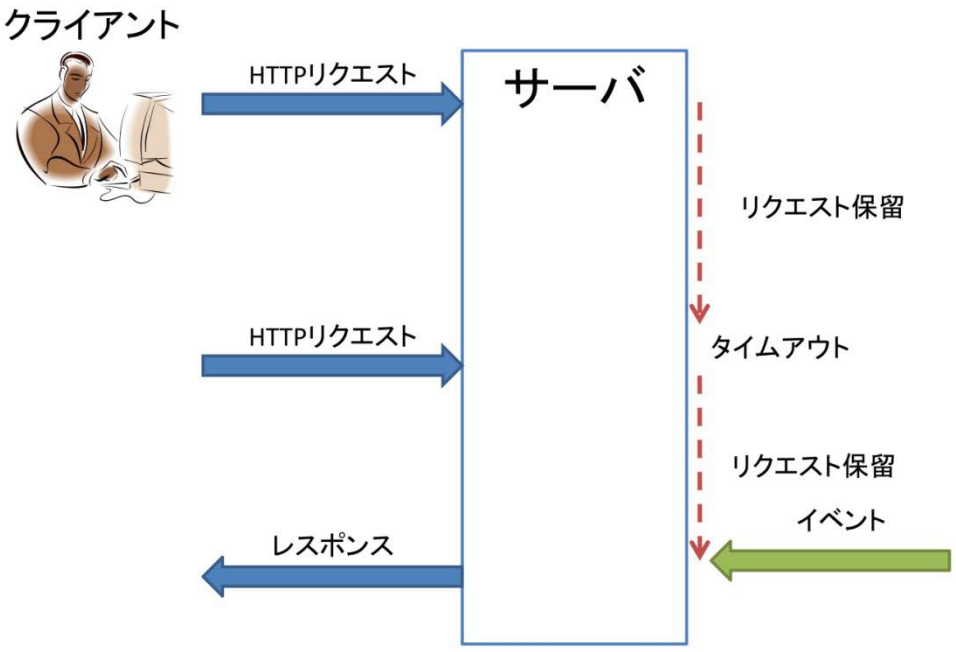


図 5-7. Comet によるサーバ・プッシュの実現

Comet ではリクエストを保留している間に HTTP リクエストを占有してしまうため，CPU やメモリといったリソースを消費し，サーバのパフォーマンスに影響を与えてしまうといった問題がある．

5.3.2 WebSocket

WebSocket とはウェブサーバとウェブブラウザの間で双方向通信を行うための通信規格であり，HTML5 の一部として規格策定が行われたが，現在は個別の仕様として W3C と IETF が規格策定をおこなっている．WebSocket はサーバとクライアントの間で専用のコネクションを作成し，データの通信はそのコネクションを通じて行う．WebSocket の通信を図 5-8 に示す．

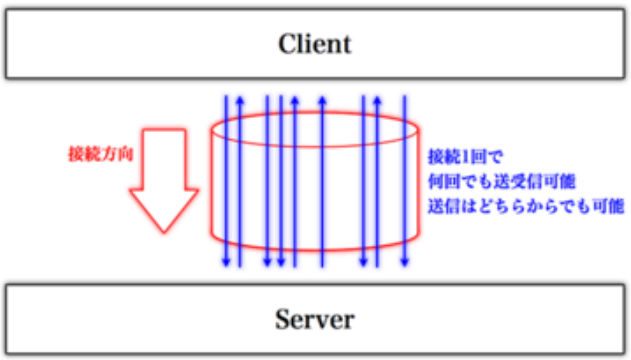


図 5-8. WebSocket による通信[13]

WebSocket の仕様は現在策定中であり，ブラウザによって対応している WebSocket プロトコルのバージョンが異なっている．各ブラウザの対応状況は表 5-1 のようになっている．

表 5-1. WebSocket プロトコルの対応状況[14]

ブラウザ	Version-76	Version 7	Version 10
Chrome	6	-	14
Firefox	4.0	6.0	7.0
IE	-	-	HTML5Labs
Safari	5.0.1	-	-
Opera	11.0	-	-

2 つの手法を比較して，公式の仕様であること，通信時のロスが少ないことなどから WebSocket を採用した．

5.4 WebSocket通信の手順

WebSocket の通信にはコネクションの開始（ハンドシェイク），データの送受信，コネクションの維持，コネクションの終了が必要になる．WebSocket 通信の手順を図 5-9 に示す．

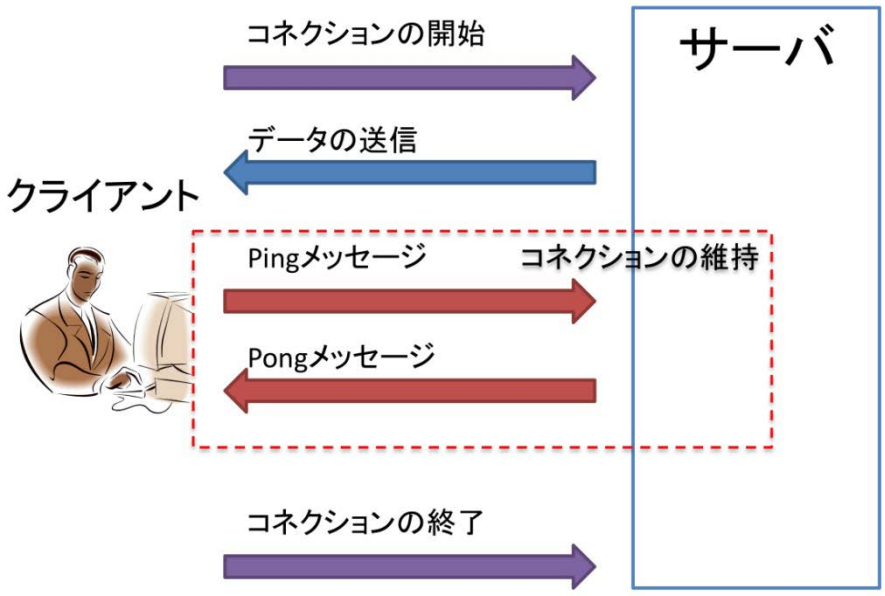


図 5-9. WebSocket 通信の手順

それぞれの手順について，WebSocket Version 10 を例にして以下に記述する．

5.4.1 コネクションの開始

クライアントとサーバで WebSocket のコネクションを開始するためにはクライアント，サーバ間でハンドシェイクを行う必要がある．ハンドシェイクを行うためには，まずクライアント側から図 5-10 の HTTP リクエストを送信する．

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

図 5-10. クライアント側からの接続要求[15]

サーバ側はこのリクエストの内容から図 5-11 のレスポンスを作成し，クライアントに送信する．

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

図 5-11. サーバからの接続要求応答[15]

クライアント側でこのレスポンスの妥当性を確認することで，WebSocket のコネクションが開始される．

5.4.2 データの送受信，コネクションの維持

WebSocket では専用のフレームを利用して通信を行う(図 5-12)．このフレームは HTTP と比べてヘッダの内容が小さく，通信ロスが少なくなる．

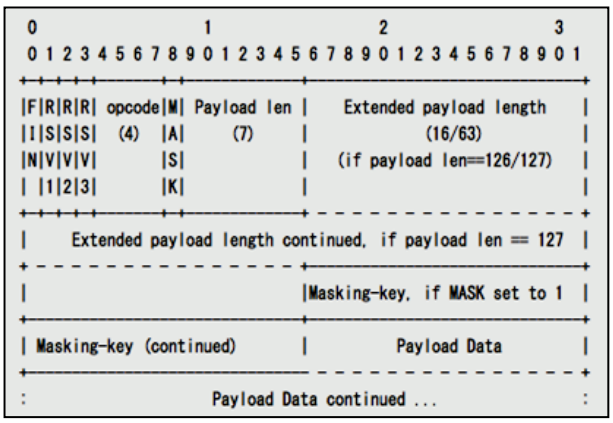


図 5-12. WebSocket のフレーム[15]

一定時間データの送受信がない場合，コネクションの維持のため Ping メッセージと Pong メッセージのやりとりが行われる．データフレームと Ping，Pong メッセージの区別は 4bit の opecode で行われる．opecode の一覧を表 5-2 に示す．

表 5-2. WebSocket フレーム，オペコードの一覧[15]

オペコード(4bit)	フレームの内容
%x0	継続フレーム
%x1	テキストフレーム

%x2	バイナリフレーム
%x3-7	予約
%x8	クローズフレーム
%x9	Ping フレーム
%xA	Pong フレーム
%xB-F	予約

5.4.3 コネクションの終了

WebSocket コネクション終了時には，opecode が%x8 のクローズフレームを送信する．

5.5 WebSocketサーバ開発時の工夫点

本システムでは WebSocket の複数あるバージョンのうち，Version 10 と draft 76 に対応している．この 2 つのバージョンには互換性が無いため，2 種類のコネクションクラス(Connection_ver10 と Connection_draft76)を作成した．この 2 つのコネクションの作成，判別を隠匿するために，SimpleFactory メソッドを利用してコネクションを作成する ConnectionFactory クラスを実装した．WebSocket の仕様は現在策定中であり，仕様が変更される可能性もあるが，ConnectionFactory クラスによって変更後の仕様にも少ない変更で対応できる．Server クラス，Connection クラス，ConnectionFactory クラスの関係を示したクラス図を図 5-13 に示す．

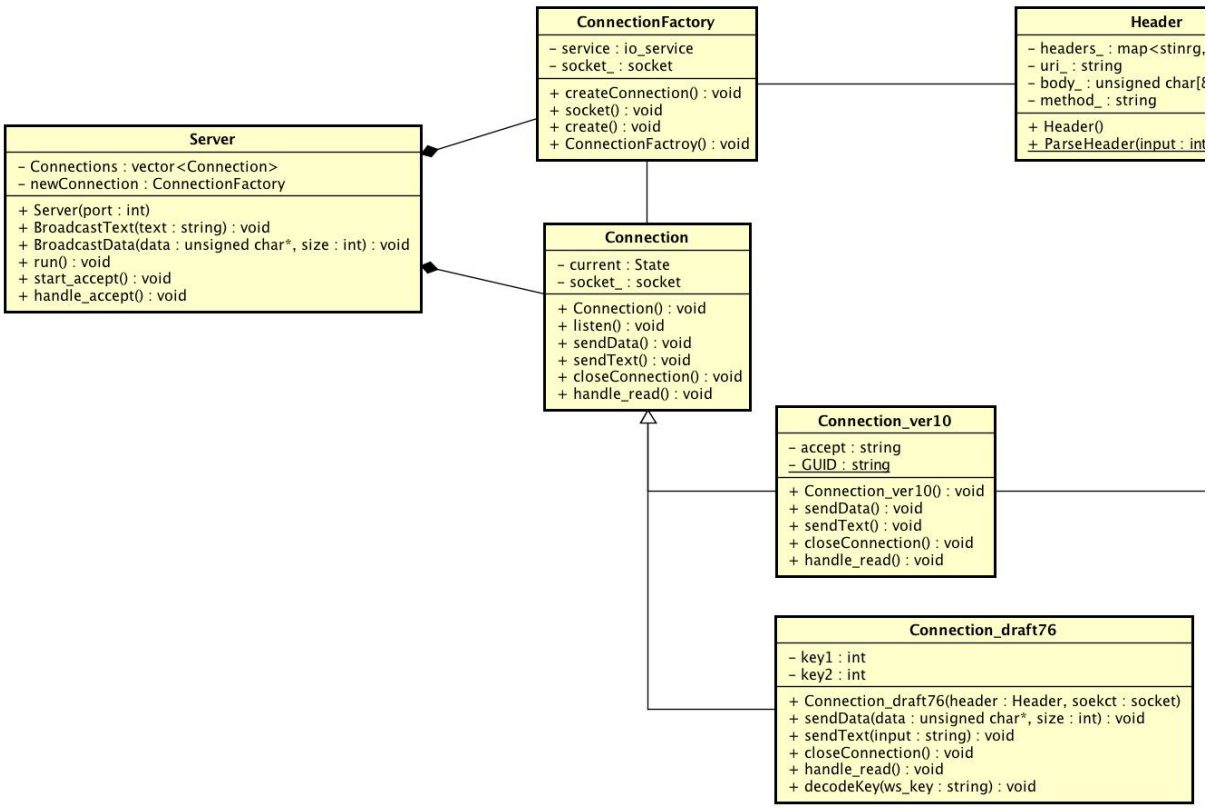


図 5-13. WebSocket サーバのクラス図

5.6 WebSocket導入後のデータ取得までの流れ

WebSocket 導入により可能になったデータ取得の流れを図 5-14 に示す.

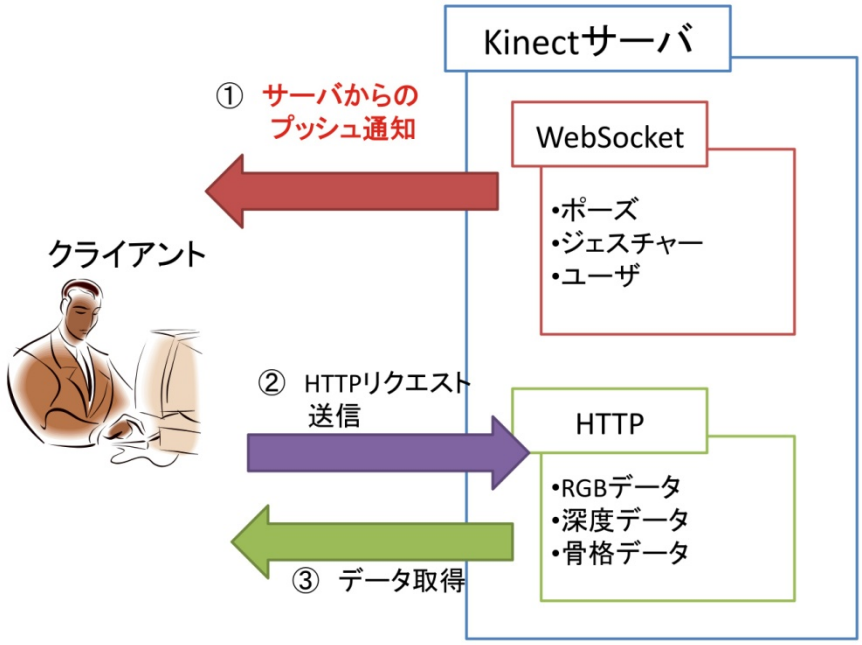


図 5-14. WebSocket 導入後のデータ取得の流れ

流れの詳細を，骨格データ取得を例にして以下に記述する.

1. サーバからのプッシュ通知
   ユーザのキャリブレーションが成功し，トラッキング中に遷移したことをクライアントに通知する.
2. HTTP リクエストの送信
   サーバからの通知を受けて，サーバに骨格データ取得の HTTP リクエストを送信する.
3. データ取得
   サーバはクライアントからのリクエストを受けて，取得したユーザの骨格データを送信する.

5.7 開発の実績

5.7.1 成果物

WebSocket の成果物として, サーバからクライアントへの通知形式を指定した WebSocket 通知形式仕様書と，全体のクラス図を作成した.

5.7.2 ソースコード

開発したソースコードの一覧を表 5-3 に示す.

表 5-3. WebSocket ソースコード一覧

```
WebSocketServer/
├── WebSocket
│   ├── Connection.cpp
│   ├── Connection.h
│   ├── ConnectionFactory.cpp
│   ├── ConnectionFactory.h
│   ├── Connection_draft76.cpp
│   ├── Connection_draft76.h
│   ├── Connection_ver10.cpp
│   ├── Connection_ver10.h
│   ├── Frame.cpp
│   ├── Frame.h
│   ├── Header.cpp
│   ├── Header.h
│   ├── Server.cpp
│   └── Server.h
├── PolarSSL
│   ├── base64.c
│   ├── base64.h
│   ├── config.h
│   ├── md5.c
│   ├── md5.h
│   ├── sha1.c
│   └── sha1.h
├── kinect
│   ├── AbstractUserDetector.cpp
│   ├── AbstractUserDetector.h
│   ├── HandDetector.cpp
│   ├── HandDetector.h
│   ├── InterfacePoseDetector.cpp
│   ├── InterfacePoseDetector.h
│   ├── PoseDetector.cpp
│   ├── PoseDetector.h
│   ├── UserDetector.cpp
│   ├── UserDetector.h
│   └── Vector3D.hpp
├── config
│   └── Config.xml
├── config.h
├── config.h.in
└── main.cpp
```

- WebSocket
  WebSocket 接続を行う部分. WebSocket のサーバ, コネクション及び, HTTP ヘッダのパーサ, WebSocket Ver.10 のフレームが含まれる.
- kinect
  Kinect からデータの取得, 及び WebSocket への送信を行う部分. ユーザ認識, 手の検出, ポーズの検出が含まれる.
- PolarSSL
  暗号化プロトコルを実装したオープンソース PolarSSL(http://polarssl.org/)から, WebSocket のハンドシェイクに必要な base64, MD5, SHA-1 のソースを利用した.

第6章　サンプルクライアントとの連携

この章では筆者が作成したサーバ・プッシュ機能と，サンプルクライアントの連携について記述する．

6.1　サンプルクライアント概要

作成したサーバ・プッシュ機能の利用例として，ポーズの通知によって動作するサンプルクライアントを作成した．トラッキング中のユーザがポーズを取ると，サーバ・プッシュ機能によってサーバからクライアントへ通知が行われ，その通知によってクライアントを操作する．作成したサンプルクライアントの画面を図 6-1 に示す．

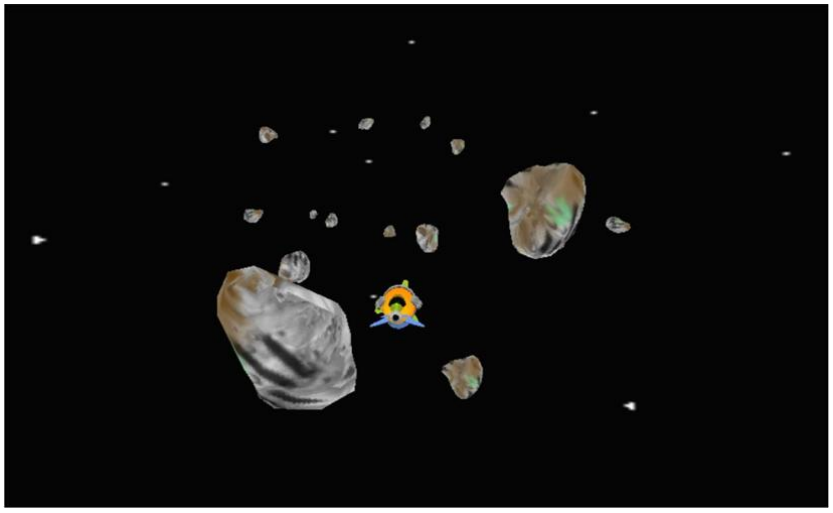


図 6-1．作成したサンプルクライアント

筆者はサンプルクライアントと連携するためのサンプルポーズの作成，サーバからクライアントへの通知形式の設計を行った．

6.2　作成したポーズ

サンプルクライアントと連携するために，サーバからの通知形式の決定と，ポーズ検出部の作成を行った．サーバからの通知形式と，作成したポーズの詳細について以下に記述する．

6.2.1　通知形式

サーバからクライアントへの通知は，Web ブラウザからの利用を考慮して JSON 形式とした．ポーズ検出(detect)とポーズ消失(lost)のタイミングで，WebSocket コネクションを通してサーバからクライアントに通知される．クライアントへの通知に利用する JSON の形式を示す．

表 6-1 に示す．

表 6-1．サーバからの通知形式(Pose)

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

例として，「ユーザ 1 の”front”ポーズが検出された」場合の通知内容を図 6-2 に示す.

```
{ “kineco” :
        { “config” :
                { “version” : 1.0 ,
                  “type” : “pose”
                } ,
        { “pose” :
                { “action” : “detect”,
                  “id” : 1,
                  “name” : “front”
                }
        }
}
```

図 6-2. 通知の例（Pose）

6.2.2 作成したサンプルポーズ

サンプルとして以下に示す 4 つのポーズを作成した.

- front
- back
- left
- right

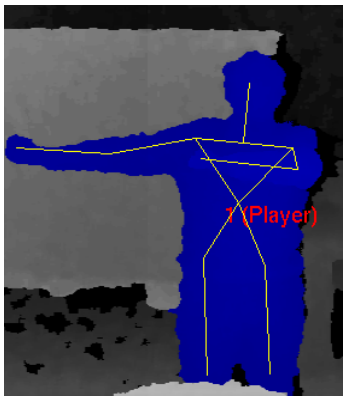


図 6-3. "left"のポーズ

6.3 ポーズ検出の工夫点

ポーズ検出を行うには骨格データが必要となるが，図 5-3 で示したように骨格データ取得までは複雑な手順が必要となる．そこで独自にポーズ作成を行う利用者向けに，骨格データ取得までの手順を行う AbstructUserDetector クラスを作成した．また，骨格データの更新や骨格データからのベクトル取得，ポーズ検出，消失時にサーバへ通知する処理を行う AbstructPoseDetector クラスの作成も行った．独自にポーズ作成を行う利用者は，

AbstructUserDetector クラス，AbstructPoseDetector クラスを継承することで，骨格データ計算の記述のみでポーズ作成を行えるようになる．AbstructPoseDetector クラス，AbstructUserDetector クラスと，サンプルとして作成した PoseDetector クラスの関連を示したクラス図を図 6-4 に示す.

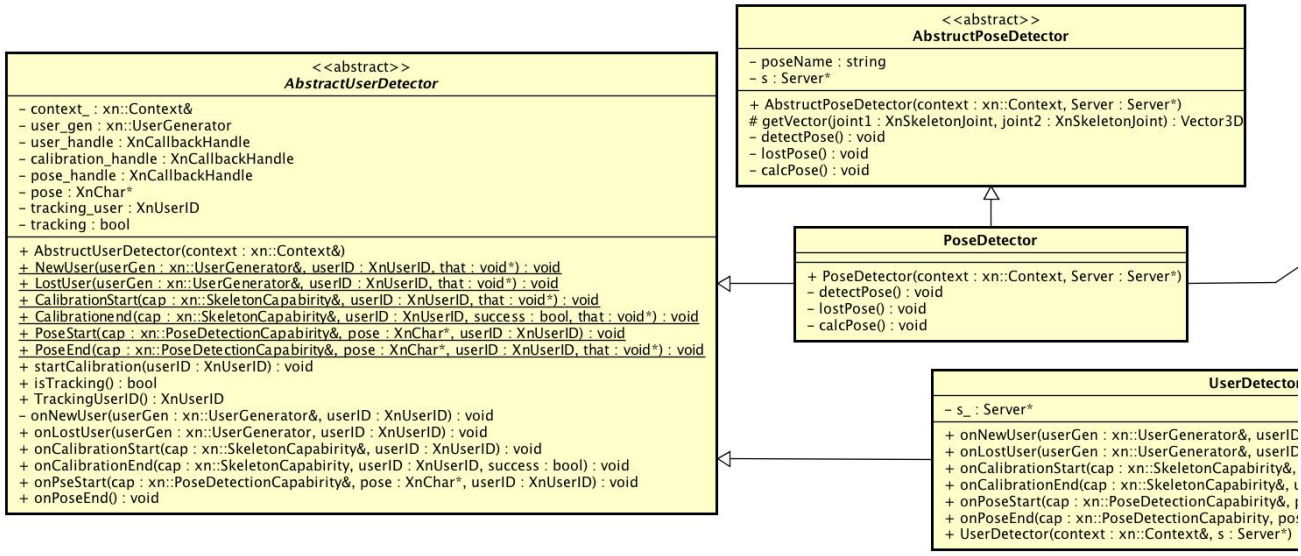


図 6-4. ポーズ検出部のクラス図

第7章　　プロジェクト体制について

この章では，本システムの開発を行ったプロジェクトの体制について記述する．

7.1　　プロジェクトメンバー

システムの開発は以下のメンバーで行った．

- ◆ 茂木　昂士（リーダー）
- ◆ 小菅　拓真（ドキュメント管理）
- ◆ 朱　明（クライアント側技術担当）
- ◆ 劉　斌（サーバ側技術担当）

7.2　　プロジェクトの進め方

本システムを開発するにあたり，開発モデルとしてとして反復型開発手法，プロトタイピングを採用した．

7.2.1　　反復型開発

開発計画には反復型開発手法を採用した（図 7-1）．全体で２回の反復を行う計画を立てたが，スケジュールの遅れが影響して第一反復が完了しないまま第二反復の開発を行った．

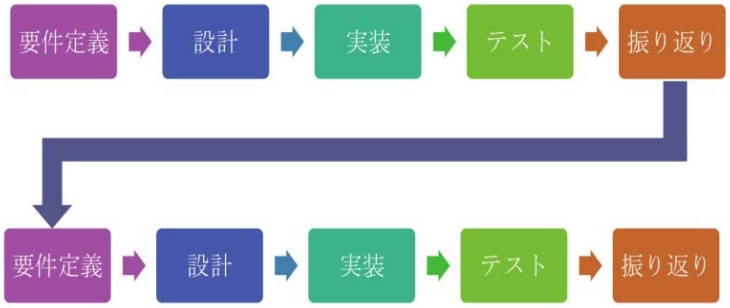


図 7-1. 反復型開発手法

7.2.2　　プロトタイピング

受注元からの要求には詳細な機能要求がなかったため，システムの機能仕様を決定する段階では要求の詳細化を行う方法としてソフトウェア・プロトタイピング手法を取り入れた．ソフトウェア・プロトタイピングに従い，以下の手順で仕様の決定，改善を行った．

1. 顧客の要求を分析
   顧客の要求を分析し，プロトタイプを作成する機能，プロトタイプからの出力形式を決定する．
2. プロトタイプ作成
   エラー処理などを省き，出力が得られるレベルのコードを作成する．
3. レビュー
   顧客とのレビューを行い，改善点のフィードバックをもらう．
4. 仕様の改善
   フィードバックから，仕様の改善，追加を行う．仕様が定まっていない点については再びプロトタイプの作成を行った．

３回程度のプロトタイプ作成で，サーバから取得するデータ形式や WebSocket によるサ

ーバ・プッシュ機能の追加などを決定した.

7.3 プロジェクト内での情報共有方法

プロジェクトの管理には Redmine[16]を利用した. Redmine はオープンソースのプロジェクト管理ソフトウェアで, プロジェクトのタスク管理, 進捗管理, 情報共有が行える. また, Subversion や Git などのバージョン管理システムとの連携機能も備えている.

7.4 スケジュール

プロジェクト開始時に設定したスケジュールと, ２回目の中間発表後に変更したスケジュールについて以下に記述する.

7.4.1 プロジェクト開始時のスケジュール

プロジェクト開始時に設定したスケジュールと, 10 月末時点での進捗状況を図 7-2 に示す.

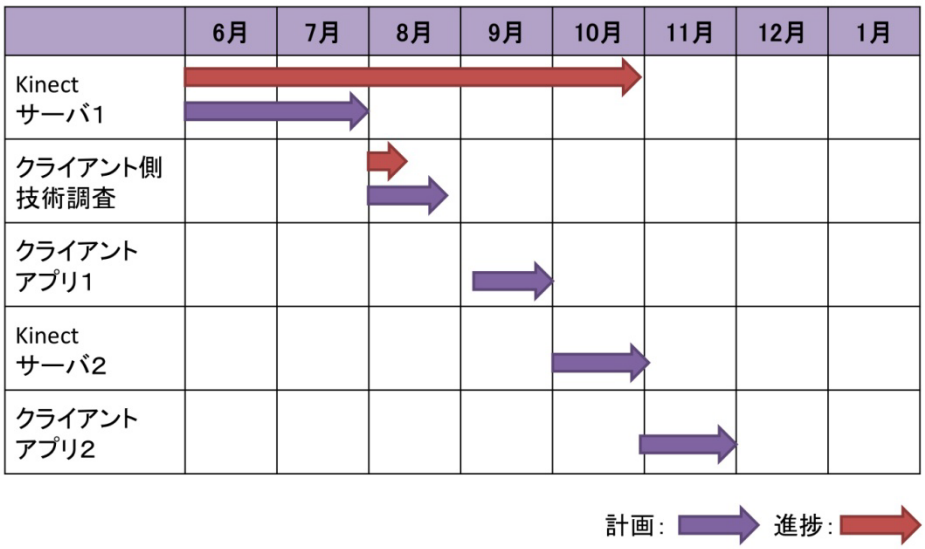


図 7-2. プロジェクト開始時のスケジュール

10 月末の時点で進捗が大きく遅れてしまっていた. 進捗が遅れた原因として以下の 3 つが挙げられる.

- 夏季休業中の作業不足

  夏季休業中はチームメンバーの予定が合わず, ミーティングを行えなかったが, その代わりに検証作業と技術調査を割り当てた. しかし要件が決定していないまま検証作業を行ったため, 作業が進捗に結びつかなかった.

- 進捗の管理不足

  我々のチームでは進捗管理に Redmine を利用しているが, 現状ではほとんど活用出来ていない. 特に途中経過の報告が全く行われていないため, 個人の作業が終了するまで進捗が把握できない. 利用状況を改善するために Redmine の利用マニュアルを作成したがレビューも行っていないだけでなく, マニュアルの存在を知らないメンバーがいるという状況だった.

◆ 個人の作業効率が低い

夏季休業中に割り当てた作業が，指定した期限を過ぎても終わらないといったことがあった．さらに進捗管理や作業報告が出来ていなかったこともあり，期限間近になってからそのことが判明するケースが多かった．

また作業分担が明確になっておらず，個々人の作業量の差が大きい，無駄な作業が多くなってしまう，するべき作業がなくなってしまうなど多くの問題が発生した．

7.4.2 スケジュールの見直し

10 月末時点での進捗の遅れを取り戻すために，図 7-3 に示すスケジュールの変更を行った．

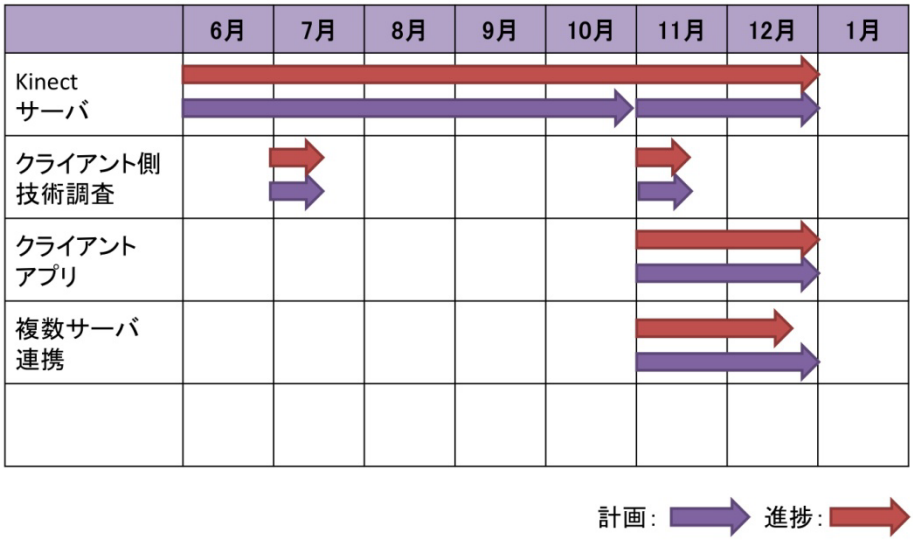


図 7-3. 見直し後のスケジュール

またスケジュールの変更に伴って各メンバーの担当も明確に分担し，設計工程から実装工程までの責任を与えた．また，担当の部分に関しての進捗管理もメンバー個人に行わせた．このことにより，以下に示す改善がみられた．

◆ 作業効率の向上

作業分担を明確にし，各メンバーに設計から実装までの責任を与えたことで個人の作業効率が改善した．

◆ コミュニケーションコストの削減

各担当の機能を分割し，各機能の連携部分を小さくすることで，設計以降のミーティング時間，回数が減少した．また，ミーティング時間が減少した分を開発に充てることで，進捗の遅れを多少ではあるが回収することができた．

7.5　開発担当

11 月のスケジュール変更後，プロジェクト全体を分割して作業分担を行った．
メンバーそれぞれの担当を表 7-1 に示す．

表 7-1．各メンバーの担当

機能		担当
サーバ	HTTP 通信部作成	劉
	WebSocket 通信部作成	茂木
	複数サーバ連携	小菅
クライアント	サンプルクライアント作成	朱
	クライアント用ライブラリ作成	劉

第8章　プロジェクトの振り返り

この章ではプロジェクトの振り返りとして，筆者の観点から要件定義，プロトタイプ作成，システム設計，実装工程，そしてプロジェクト全体について記述する．

8.1　要件定義

- 良かった点

  Kinect や WebSocket など，プロジェクト開始前から興味を持っていた分野での開発出会ったため，自分のアイディアを多く取り入れることができた．

- 改善点

  顧客との打ち合わせが少なかった点やシステムの評価を行えなかったため，顧客の要望をすべて満たせているか確認出来ていない点に問題がある．また技術調査が十分でなく，何度も仕様変更をしたために，仕様決定に時間がかかってしまった．

8.2　プロトタイプ作成

- 良かった点

  自分の知識や経験を活かして，プロトタイプの作成には大きく貢献できた．また，プロトタイピングによって顧客の要望を引き出すことに成功した．

- 改善点

  プロトタイプ作成時点では役割分担が曖昧で，個人の作業量に大きく差ができてしまった．また情報共有が機能せず，作業の重複が多かった点も問題がある．

8.3　システム設計

- 良かった点

  WebSocket の部分やポーズ作成の部分では，拡張性を十分に考慮した設計ができた．

- 改善点

  設計が十分でない部分があるまま実装を行ったことで，何度も設計書の修正が必要になってしまった．

8.4　実装工程

- 良かった点

  スケジュールの大幅な変更や役割分担の明確化などで開発効率を改善し，多少ではあるが進捗の遅れを取り戻すことができた．

- 改善点

  コミュニケーションを減らした結果，全員がシステムの全体像を理解していないという大きなリスクを抱えたまま開発を行うことになってしまった．また，コーディング規約を作成しなかったことにより，一貫性のないコードになってしまった．

8.5　プロジェクト全体

- 良かった点

  設計以降の工程では個人の能力を活かして，効率的な開発を行えた．

- 改善点

  スケジュールの見積が甘く，進捗管理も出来ていなかったため，全体のスケジュールが大幅に遅れてしまい，品質の確保やシステム評価が行えなかった．また，プロジェク

トの初期段階にチームビルディングを怠ったために，チーム全体での情報共有が機能しなかった，チームとしての開発効率が低くなってしまった，などの問題が発生してしまった.

第9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 センサーのデータを提供する Kinect サーバ及び，Kinect サーバのデータを利用するサンプルクライアントの開発を行った．顧客の要望に対してシステム構成の提案や，プロトタイプを用いての検証作業によって仕様を確定した．

筆者はシステムの開発において，Kinect サーバの HTTP 通信における問題点を解決するために，サーバ・プッシュ機能の検討及び設計，実装を行った．またサーバ・プッシュ機能を用いて，クライアントとの連携及びサンプルポーズの設計，実装を行った．

スケジュールの都合上，品質の確保や評価が行えなかったため，今後の課題として品質の向上や評価，機能の充実などが挙げられる．

謝辞

本プロジェクト委託元教員である高橋伸准教授には，テーマの提供やプロジェクト遂行に協力して頂いたことに深く感謝いたします.

指導教員である田中二郎教授には，2 年間にわたり大学生活やプロジェクトに関して指導していただきましたことに心より感謝いたします.

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担当教員である山戸昭三教授，中沢研也教授には，授業だけでなくプロジェクト遂行のための指導，アドバイスをしていただいたことに心より感謝いたします.

本プロジェクト遂行のために大変有益な技術や知識を指導していただいた駒谷昇一教授，菊池純男教授をはじめとした，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教員の方々，本プログラムに様々な支援をしていただいた特定非営利活動法人　高度情報通信人材育成支援センターの方々に深く感謝いたします.

本プロジェクトのチームメンバーである小菅拓真，朱明，劉斌にはプロジェクト成功のために様々な協力をしていただきました．共にプロジェクトを遂行できたことに深く感謝いたします.

最後に，大学院で学ぶ機会を与えてくれた家族，様々な面で支えてくれた友人，先輩，後輩，大学生活でお世話になったすべての方々に心より感謝いたします.

参考文献

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

[2] “Xbox 360 用モーションコントローラ—Microsoft Kinect の仕様と骨格トラッキング動画.” http://takao.asaya.ma/article_701.html .

[3] “Kinect Xbox.com” http://www.xbox.com/ja-JP/kinect

[4] PrimeSense 社ドライバーを公開，オープンソースコミュニティ OpenNI を開始. http://willowgarageja.blogspot.com/2010/12/primesense-releases-drivers-open-source.html .

[5] OpenNI > About. http://www.openni.org/About.aspx.

[6] WebGL OpenGL ES 2.0 for the Web http://www.khronos.org/webgl/

[7] 世界で最初の Kinect+OpenNI+NITE の本を書きました. http://d.hatena.ne.jp/kaorun55/20110518/1305722932 .

[8] Introducing JSON JSON. http://www.json.org/

[9] RFC3548 The Base16, Base32, and Base64 Data Encodings. http://www5d.biglobe.ne.jp/~stssk/rfc/rfc3548j.html.

[10] RFC2397 The “data” URL scheme. http://tools.ietf.org/html/rfc239.

[11] MATLAB Central Simulink for Natural Interaction Devie(NID): NID Skeleton http://www.mathworks.com/matlabcentral/fileexchange/32318-simulink-for-natural-interaction-device-nid/content/slnid/Lib/doc_ja/slnid_skeleton.html

[12] IBM developerWorks “リバース Ajax:第 1 回” http://www.ibm.com/developerworks/jp/web/library/wa-reverseajax1/

[13] WebSocket 登場までの歴史. http://gihyo.jp/dev/feature/01/WebSocket/0001 .

[14] WebSockets – MDN. https://developer.mozilla.org/en/WebSockets.

[15] The WebSocket protocol draft-ietf-hybi-theWebSocketprotocol-17. http://tools.ietf.org/html/draft-ietf-hybi-theWebSocketprotocol-17 .

[16] Redmine.JP. http://redmine.jp/overview/.

付録目次

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
- 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．

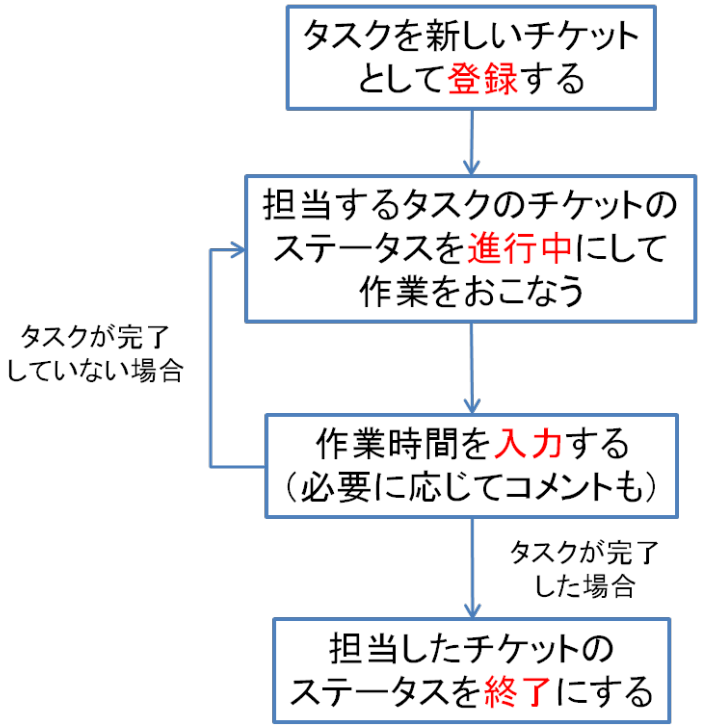


図 4. チケットの運用プロセス

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．

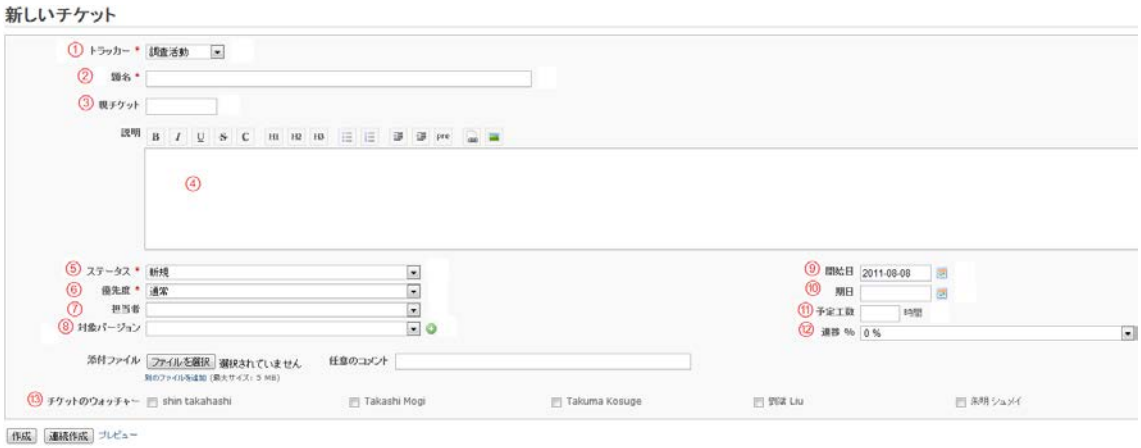


図 5. チケット登録画面

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 2. 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 3. 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 補完してしまい必要なくなった場合 若しくは トラッカーがバグのチケットにおいて 他のバグを修正した際に，このチケットの 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 消化する必要がなくなった場合 若しくは トラッカーがバグのチケットにおいて 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 4. 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 5. 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 6. 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 7. 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 8. 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 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

4. Kinect HTTPサーバAPI概要

4.1. 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。

4.2. 機能説明

◆

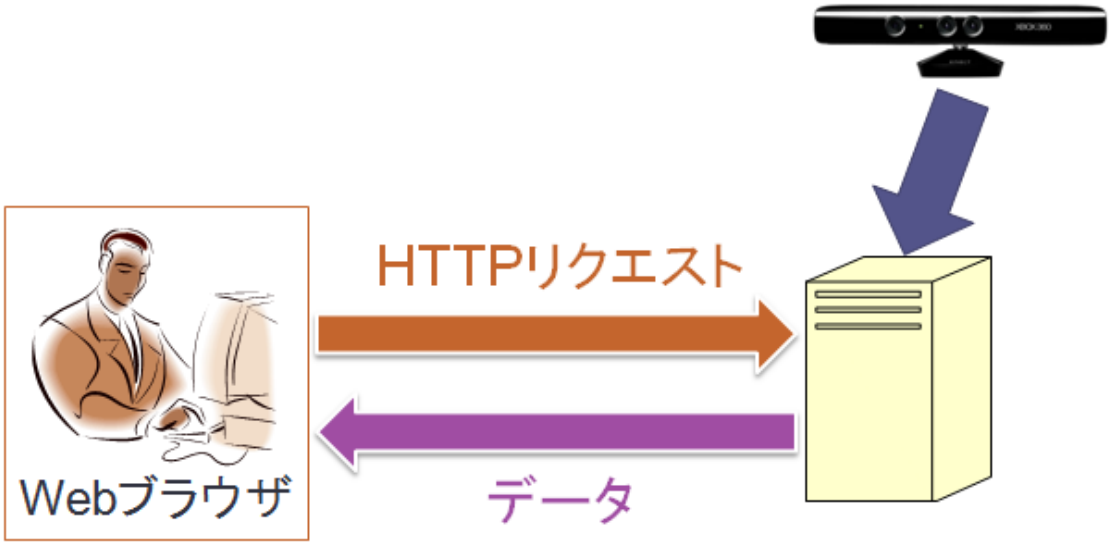
(1) システム条件

- 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

◆

5. Web API インタフェース仕様

5.1. 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

5.2. 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	

5.3. 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 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

5.4. 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	-

5.5. 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

5.6. 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
          }
        }
      ]
    }
```

```
18    ]
19 }}
```

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

5.7. 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

5.8. 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】

5.9. 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】

5.10. 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】

5.11. 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】

5.12. 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

メソッド: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】

5.13. 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

5.14. 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

5.15. 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

5.16. 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】

5.17. 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

6. 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。

7. HAND データ

手を検出した際の通知形式

表 6. 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 -320 < x < 320, -240 <y<240 0 < z < 1000
			y	数値	
			z	数値	

8. USER 検出データ

ユーザを検出した際の通知形式

表 7. 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

9. POSE 検出データ

ポーズを検出した際の通知形式

表 8. 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

10. 概要

10.1. 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。

10.2. 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。

10.3. 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。

10.4. 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			

10.5. 定義（用語、略語）

ID	用語・略号	正式表記	意味

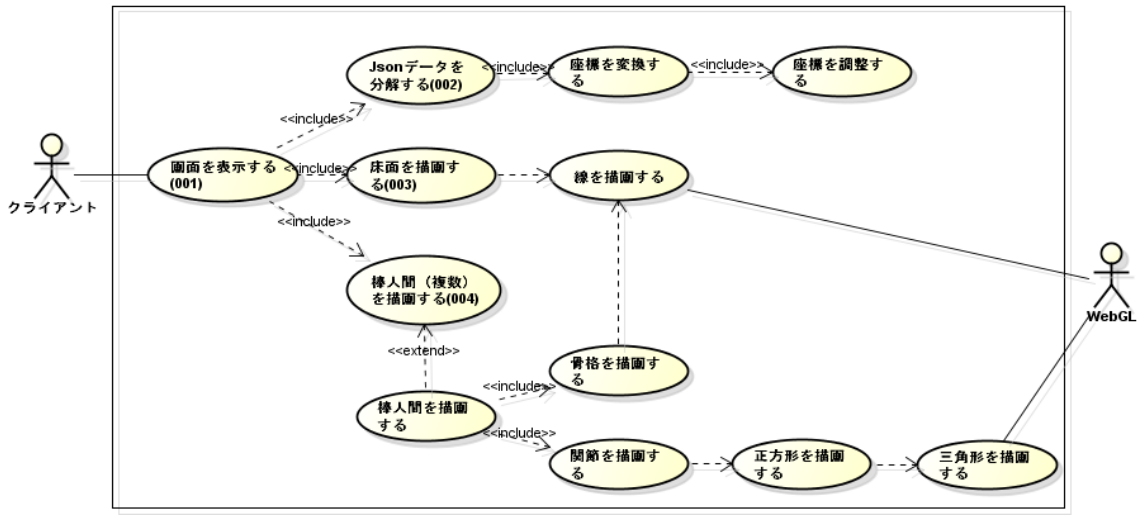
11. 棒人間

11.1. 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。

11.2. ユースケース図



11.3. 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 2　WebGL コンテンツのサイズを設定する 3　プログラマブルシェーダーを設定する 　3.1　頂点シェーダーを初期化する 　3.2　フラグメントシェーダーを初期化する 4　視体積の投影面のサイズを設定する 5　視体積の投影面の背景色を設定する 6　Json データを読み込む 　6.1　床のサイズを設定する 　6.2　棒人間に該当する関節座標値を設定する 7　視体積を定義する 8　アフィン変換の変換行列（4×4）を定義する 9　床を描画する 10　棒人形を描画する 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 2　床オブジェクトを初期化する 3　床を描画する 4　棒人間オブジェクトを初期化する 5　座標データを引数として、棒人間を描画する 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 2　床面の Y 方向の平行移動距離、線の色を初期化する 3　横線・縦線の本数を計算する 4　横線・縦線の頂点配列に両端点の XYZ 座標を設定する 5　床面の頂点バファに頂点配列をロードする 6　横線・縦線の色配列に線の両端点の RGBA 値を設定する 7　床面の色バファに色配列をロードする 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 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 2　現実座標を棒人間の座標に変換するα値を設定する 3　Json データによって、ユーザー数を抽出する 4　ユーザーID 配列を初期化する 5　棒人間の座標を設定する 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 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 5.3　ボン配列（関節の関係）を初期化する 5.4　1 つの関節の X・Y・Z 座標を変換する 5.5　関節の頂点配列にこの関節の XYZ 座標を設定する 5.6　関節の最小・最大 Z 軸値を記録する 5.7　関節の座標処理は終わるかどうかを判断する。 終わらない場合は、5.3 へ戻り 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 5.9　ボンの配列から該当する不適切な関節を取り除く 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 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 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 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 5.14　ボンの色配列にボンの両端点の RGBA 値を設定する 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 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 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

	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 5.19　ボンの色バファにボンの色配列をロードする 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 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 終わらない場合は、5 へ戻り 7　WebGL コンテンツで関節（正方形）を描画する 8　WebGL コンテンツでボンを描画する 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	

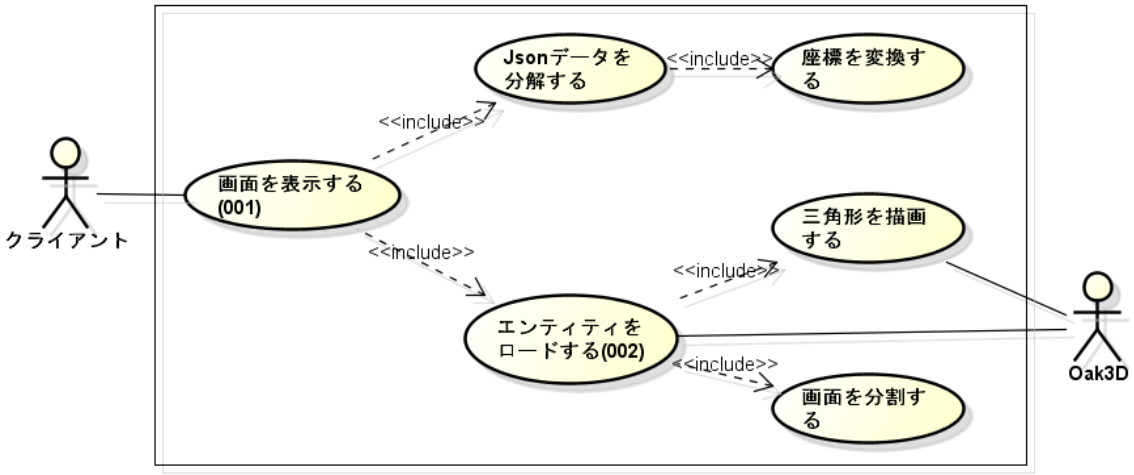
12. 3D 空間

12.1. 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。

12.2. ユースケース図



12.3. 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 2　initScene というコールバックファクションを定義する 2.1　Oak3D のエンジンを初期化する 2.2　エンジンと Canvas を結び付ける 2.3　シーンを初期化する 2.4　シーンのバウンディングボリュームを設定する 2.5　カメラを初期化する 2.6　視体積を定義する 2.7　視体積の投影面のサイズを設定する 2.8　視体積の投影面のバックグランド色を設定する 2.9　カメラの位置を設定する 2.10　視点の座標を設定する 2.11　サーバーの URL：Port から RGB＋D Json データを取得する 2.12　3D 空間のエンティティをロードする 2.13　2.11 へ戻り、繰り返す 3　マウスのコールバックイベントを設定する 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 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 3　子画像の頂点配列と色配列を初期化する 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 5　深度値を 3D 空間の Y 軸に、画像の幅を Z 軸に変換する 6　子画像の頂点配列に各ピクセルの XYZ 値を設定する 7　子画像の色配列に各ピクセルの RGB 値を設定する 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 9　子画像の頂点インデックス配列を初期化する 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 11　シーンにエンティティを新規にする 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 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 14　エンティティの色属性に子画像の色配列を設定する 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 16　エンティティを一定比率でスケーリングする 17　動的な光源をクロースする 18　子画像の処理を全てし終わるかどうかを判断する。 終わらない場合は、3 へ戻り 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	

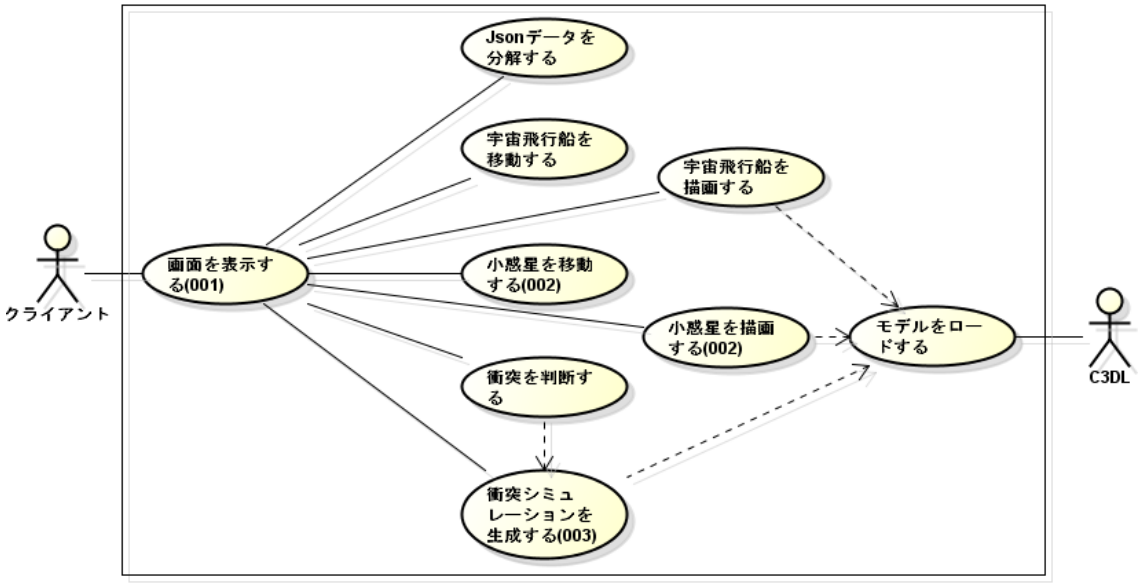
13. 宇宙飛行

13.1. 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。

13.2. ユースケース図



13.3. 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 2　シーンを初期化する 　2.1　シーンのサイズを設定する 　2.2　シーンと Canvas を結び付ける 　2.3　シーンのバックグランド色を設定する 3　WebGL コンテンツを初期化する 　3.1　WebGL コンテンツでシーンをレンダーする 4　カメラを初期化する 　4.1　カメラの位置を設定する 　4.2　カメラの視点を設定する 　4.3　シーンにカメラオブジェクトをロードする 5　小惑星配列を初期化する 6　小惑星を初期化する 7　宇宙飛行船を初期化する 　7.1　宇宙飛行船 collada ディレクトリを初期化する 　7.2　dae モデルをロードする 　7.3　宇宙飛行船の位置を設置する 　7.4　宇宙飛行船をスケーリングする 　7.5　シーンに宇宙飛行船オブジェクトをロードする 8　衝突シミュレーションを初期化する 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 　9.1　一定時間を経って、新しい小惑星を生成する 　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 　9.3　前後移動する場合は、カメラが連動する 　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 9.5　小惑星の判断が全て終わらない場合は、9.4 へ戻り 10　星空を初期化する 10.1　星空 collada ディレクトリを初期化する 10.2　dae モデルをロードする 10.3　シーンに星空オブジェクトをロードする 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 2　dae モデルをロードする 3　png 図でテクスチャする 4　小惑星をスケーリングする 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 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 7　小惑星が進行するスピードを設定する 8　小惑星の軌道を設定する 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 10　小惑星の中心軸に回転角速度を任意設置する 11　小惑星配列に小惑星を設定する 12　シーンに新しい小惑星オブジェクトをロードする 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 2　粒子毎に最大・最小スピードを設置する 3　粒子毎に現れる最大・最小時間帯（秒）を設置する 4　粒子毎に現れる色の幅を設置する 5　指定された図でテクスチャする 6　粒子システムに粒子の総数を設置する 7　シーンに粒子システムオブジェクトをロードする 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 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	

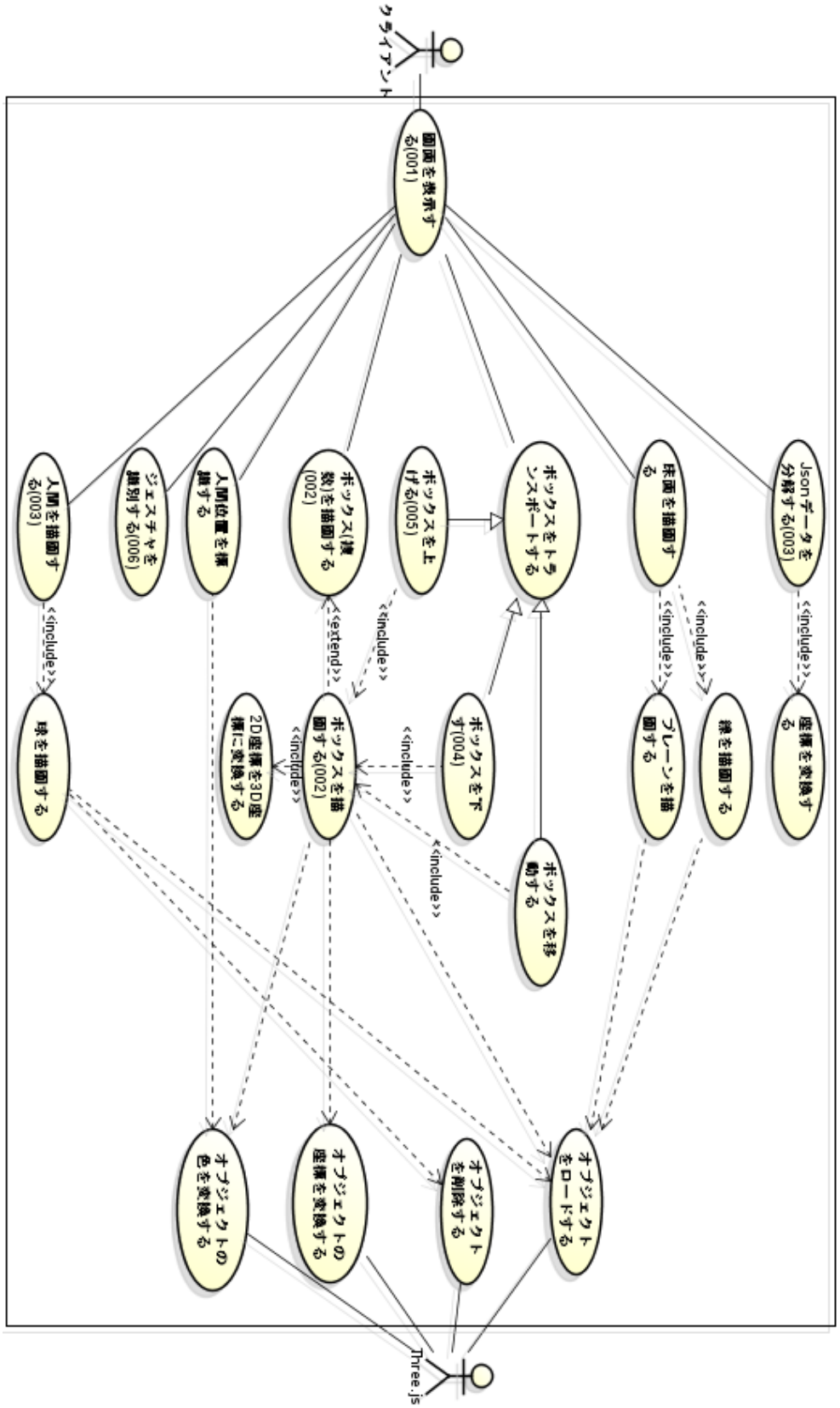
14. ボックスのトランスポート

14. 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。

14.2. 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(建数)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)
ボックスを移動する
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
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>>

14.3. 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 2　レンダラーを初期化する作成 2.1　レンダラーのサイズを設定する 2.2　div コンテナにレンダラーエレメントを設定する 3　カメラを初期化する 3.1　視体積を定義する 3.2　カメラの位置を設定する 3.3　視点を設定する 4　シーンを初期化する 5　床を描画する 5.1　横線・縦線の座標と色を設定する 5.2　シーンに横線・縦線オブジェクトを設定する 5.3　プレーンのサイズと色を設定する 5.4　シーンにプレーンオブジェクトを設定する 6　プロジェクタを初期化する 7　レンダラーにシーンとカメラを設定する 8　ボックスを描画する 9　アニメのコールバックファクションを定義する 9.1　Json データを取得する 9.2　球人間を描画する 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 2　生成するボックスの個数を設定する 3　スクリーンに 1 つの点を任意選ぶ 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 5　視体積にカメラと点の延長線を定義する 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 　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 7　ボックスを描画する 　7.1　ボックスのサイズと色を設定する 　7.2　メッシュで立方体を描画する 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 9　シーンにボックスオブジェクトを設定する 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 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 2　座標データによって、球人間を描画する 　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 　2.2　球人間の高さを定義する 　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 　2.4　球人間の各関節の座標・色を設定する 　2.5　メッシュで球のような関節を描画する 　2.6　シーンに球人間オブジェクトを設定する 　2.7　シーンをレンダーする 3　プレーヤの身ぶりを判断する 　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 　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 　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 　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 4　シーンをレンダーする 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 　ない場合は、戻り 2　球人間の頭の座標を取得する 3　視体積にカメラと頭の延長線を定義する 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 　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。 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 6　ボックスの座標に格子の座標を設定する 7　ボックスの色を元に戻す 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 　ない場合は、戻り 2　球人間の両手の中心座標を取得する 3　視体積にカメラと両手の中心の延長線を定義する 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 　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。 5　フォーカスボックスの新座標を設定する 6　フォーカスボックスの色を設定する 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 2　左上腕と右上腕の角度 A を計算する 3　左・右上肢のベクトルを計算する 4　左上肢と右上肢の角度 B を計算する 5　両肩のベクトルを計算する 6　両手の中点・両肘の中点のベクトルを計算する 7　両肩と両手の中点・両肘の中点の角度 C を計算する 8　頭と首のベクトルを計算する 9　両肩と頭・首の角度 D を計算する 10　頭・首と両手の中点・両肘の中点の角度 E を計算する 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 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。

4. リアルタイム応答性の確保
5. センサーデータの更新と取得をカプセル化
6. 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

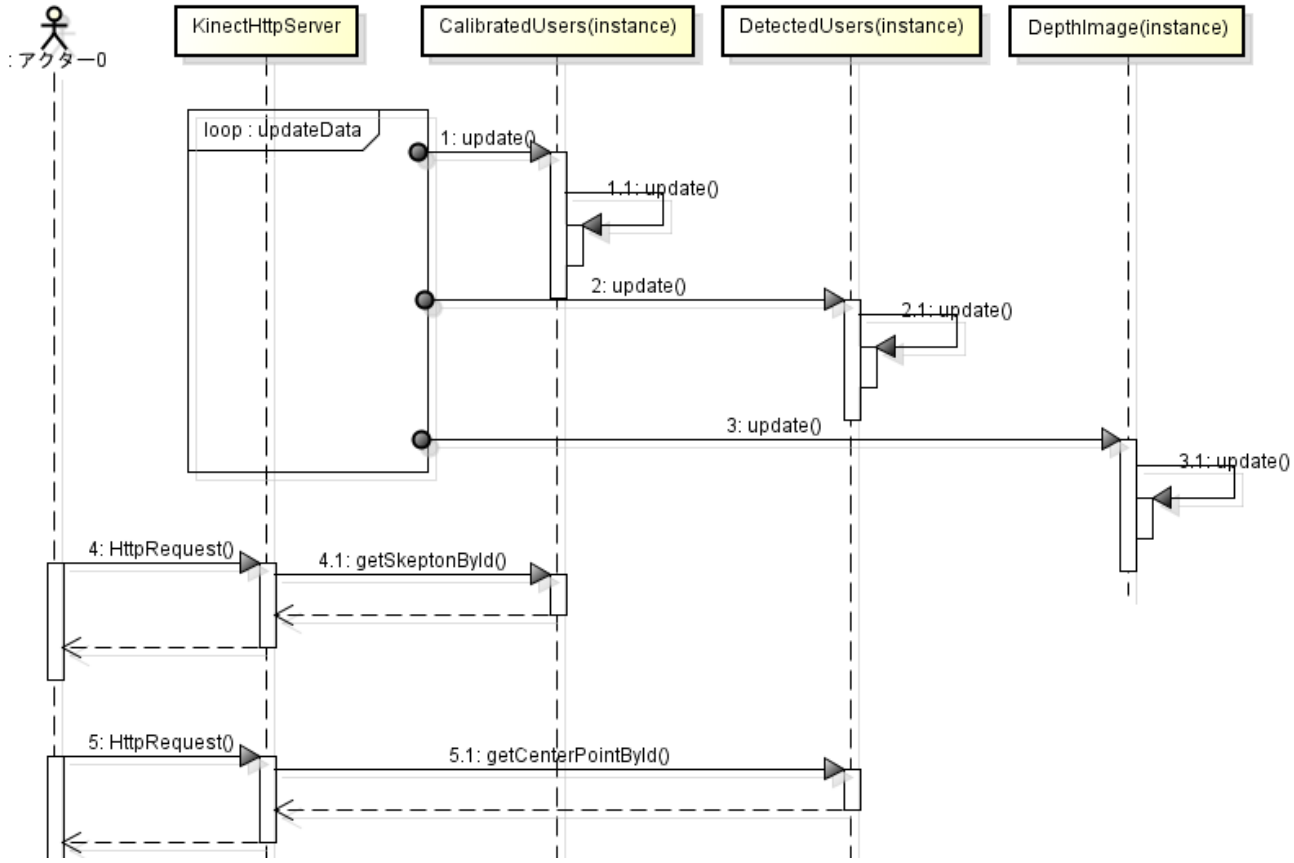


図 2-1 データ取得までのシーケンス図

2.2 Httpサーバクラス図（一部）

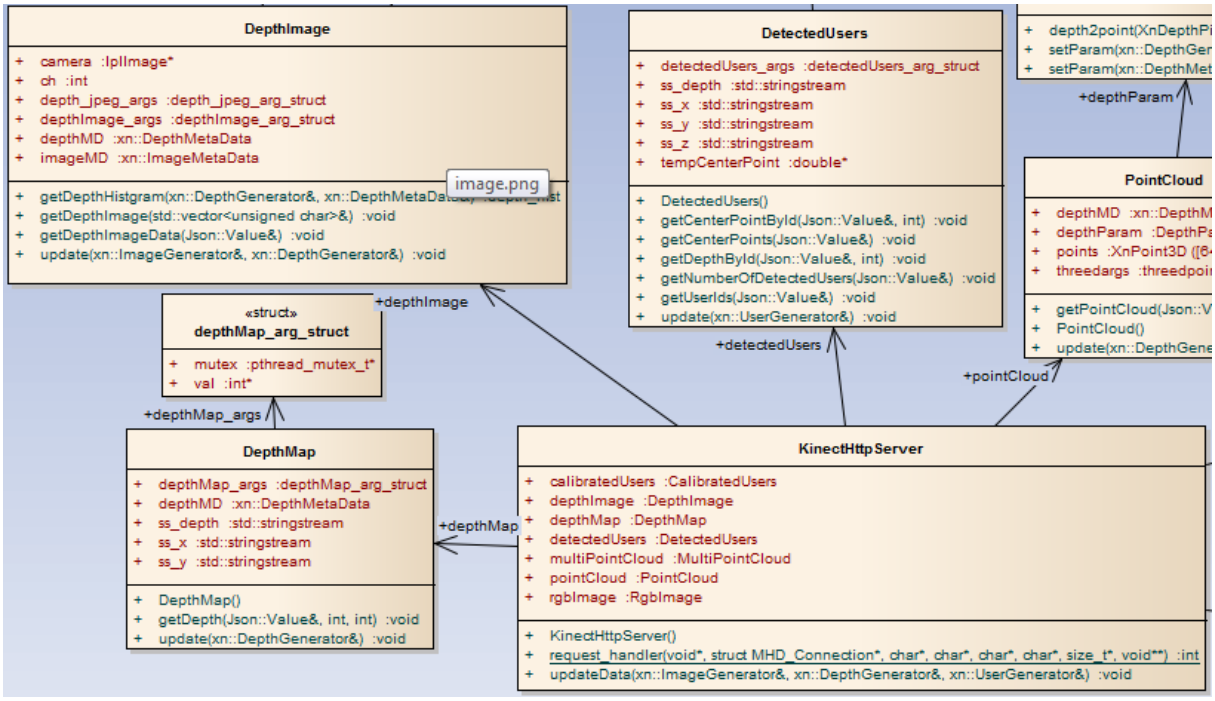


図 2-2　クラス間の関連つけ

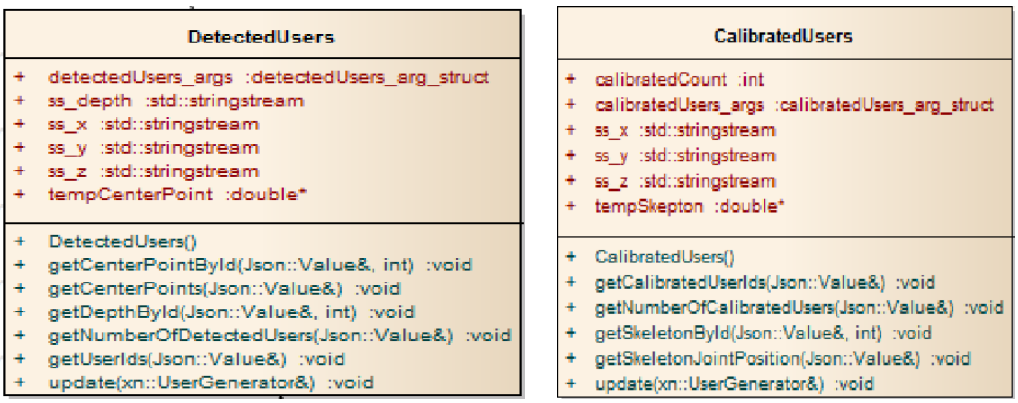


図 2-3　検出・識別ユーザクラス図の相違

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	Ipllmage	camera	カメライインスタンス
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	IpllImage	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。 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

筑波大学大学院博士課程

[文書のサブタイトルを入力してください]

Ver1.0

Team. Kineco

茂木　昂士

小菅　拓真

朱　明

劉　斌

更新履歴

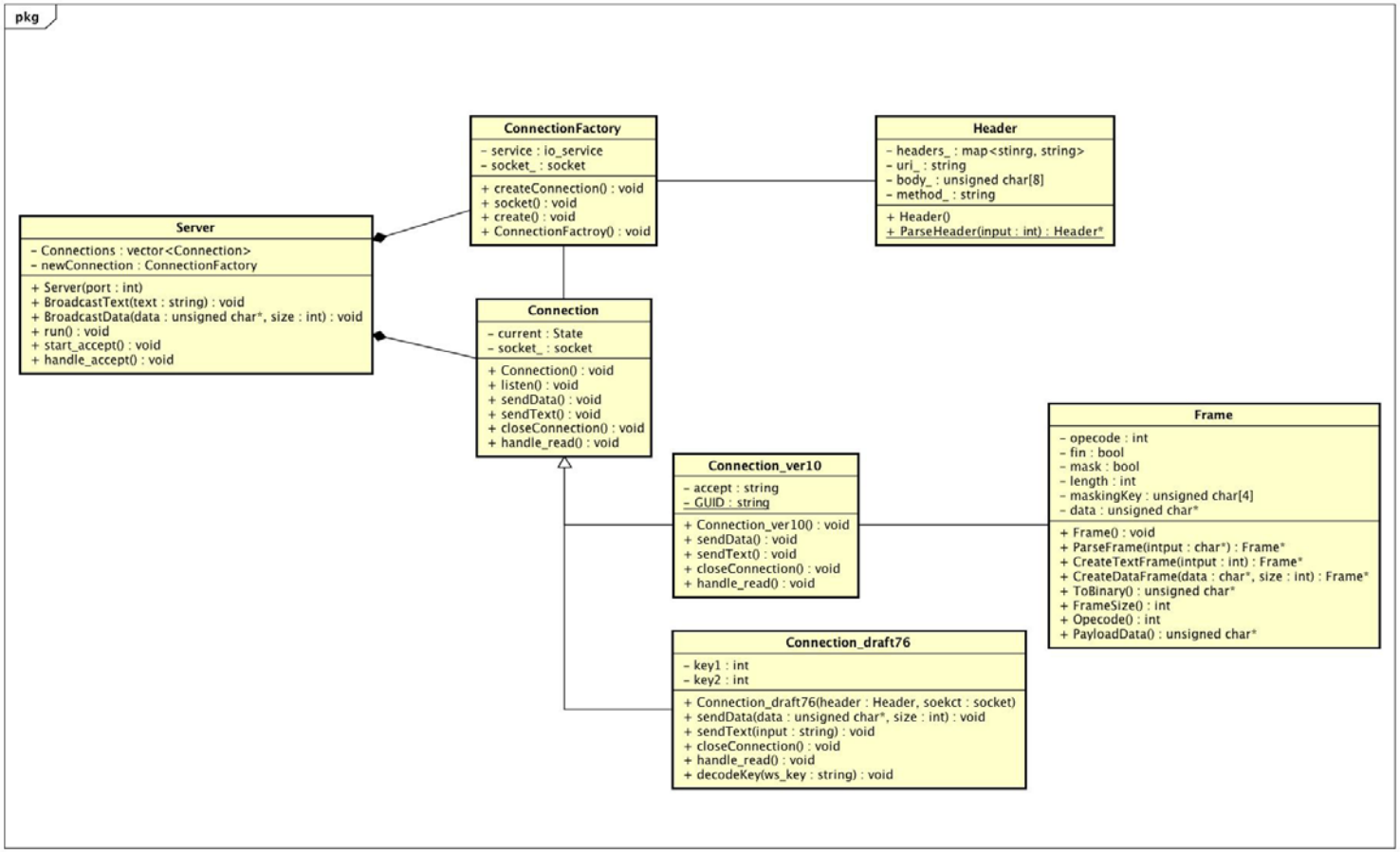
版数	日付	追加・更新箇所	担当
1.0	2011/12/13		茂木
1.1	2011/12/20	PoseDetector の継承元を Interface から Abstract へ変更	茂木

目次

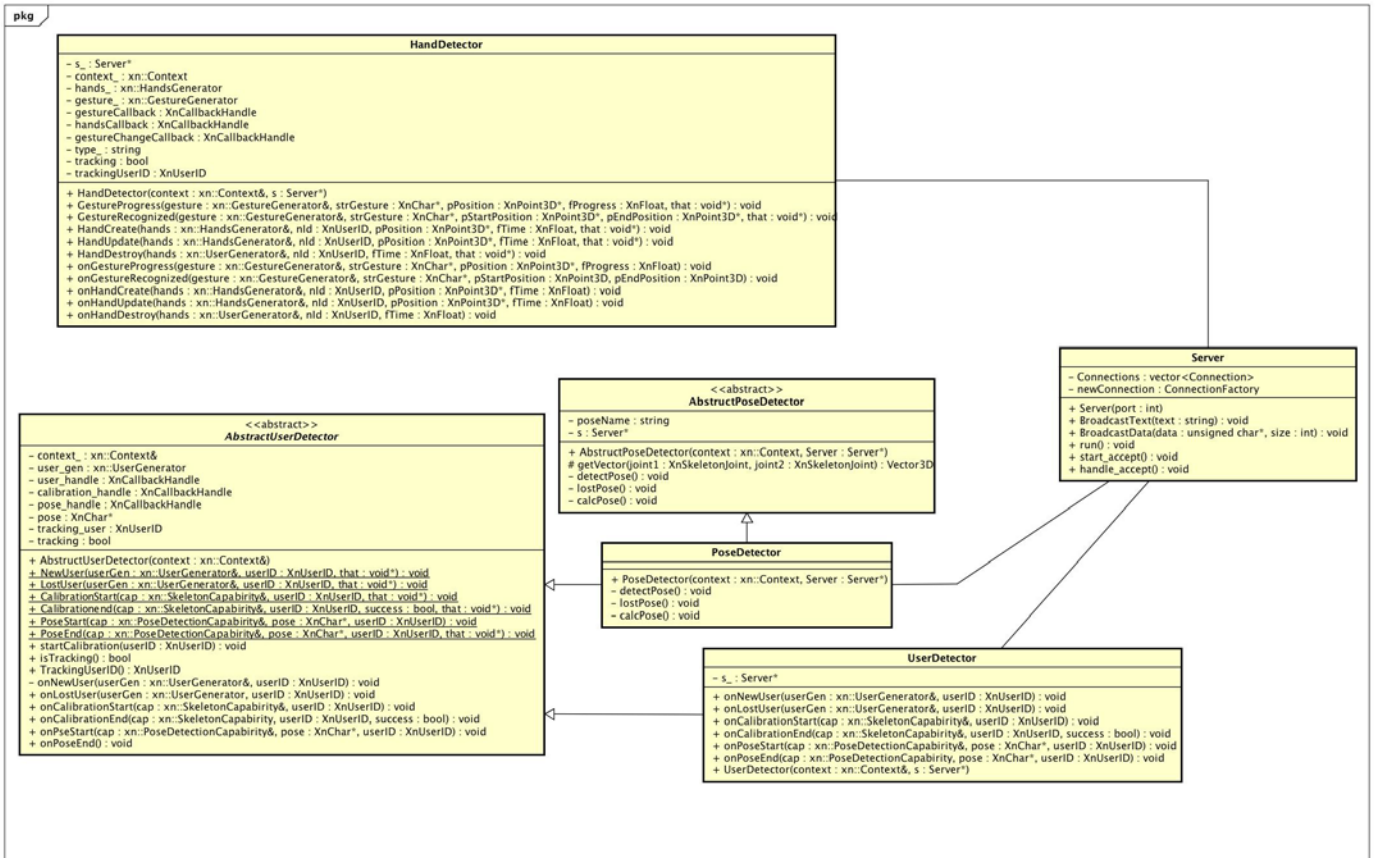
15. はじめに

この資料には、Websocket 通信部分のクラス設計を記述する。

16. WebSocket サーバ



17.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

18. 概要

18.1. 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。

18.2. 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。

7. 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。
8. Kinect によって、提供できるデータに基づいて、設計を行う。
9. 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。

18.3. 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。

18.4. 参照ドキュメント

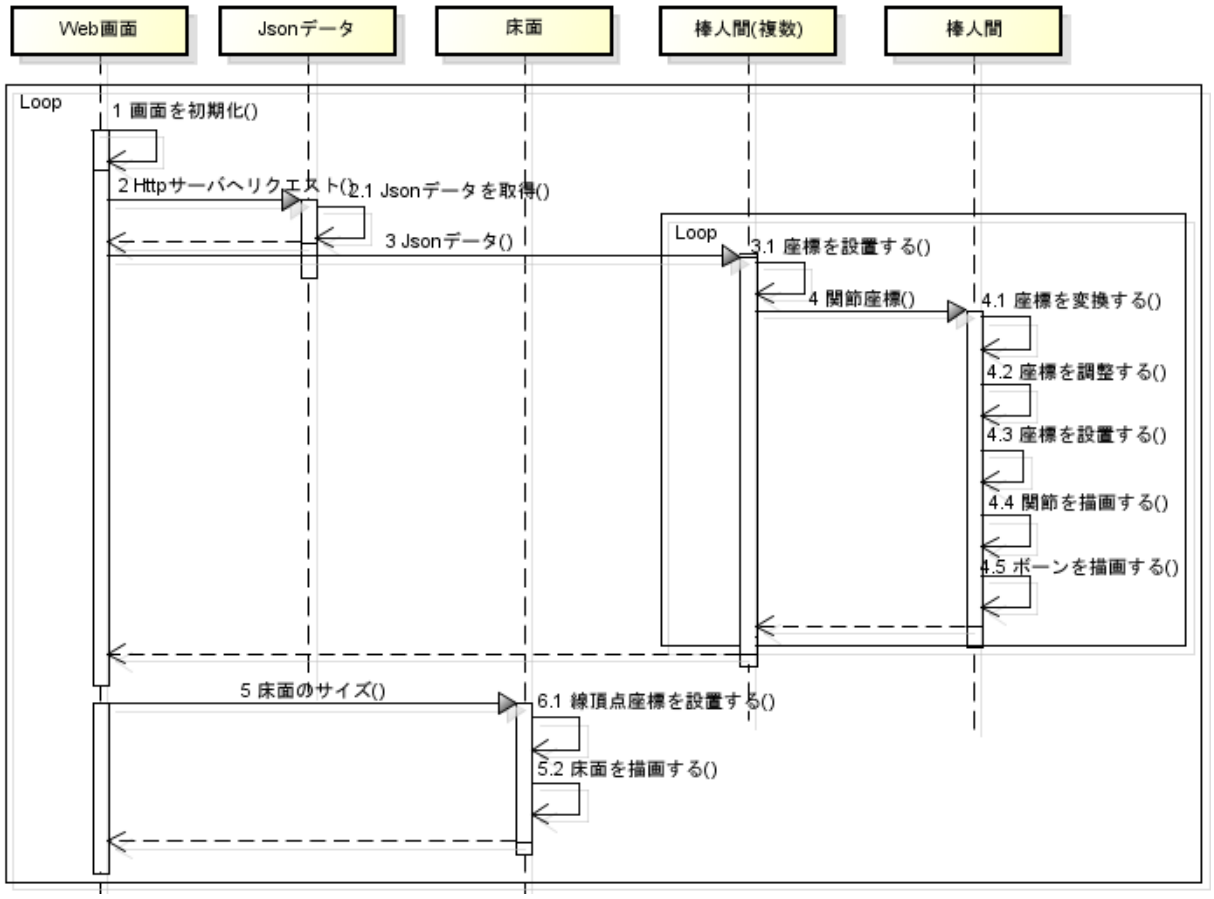
ID	文書名	文書番号	発行年月	備考
	要件定義書			
	基本設計書			

18.5. 定義（用語、略語）

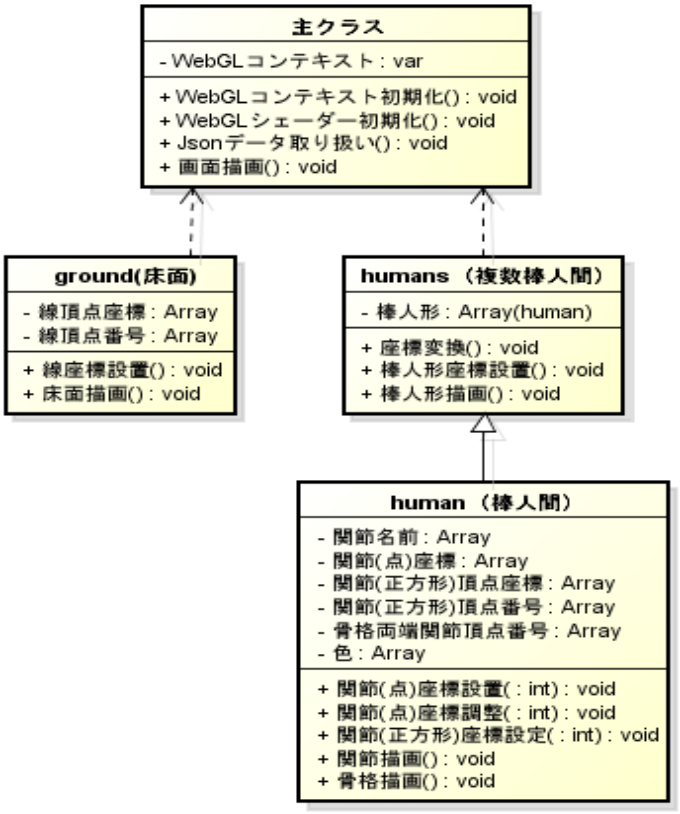
ID	用語・略号	正式表記	意味

19. 棒人間

19.1. シーケンス図



19.2. 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

19.3. 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

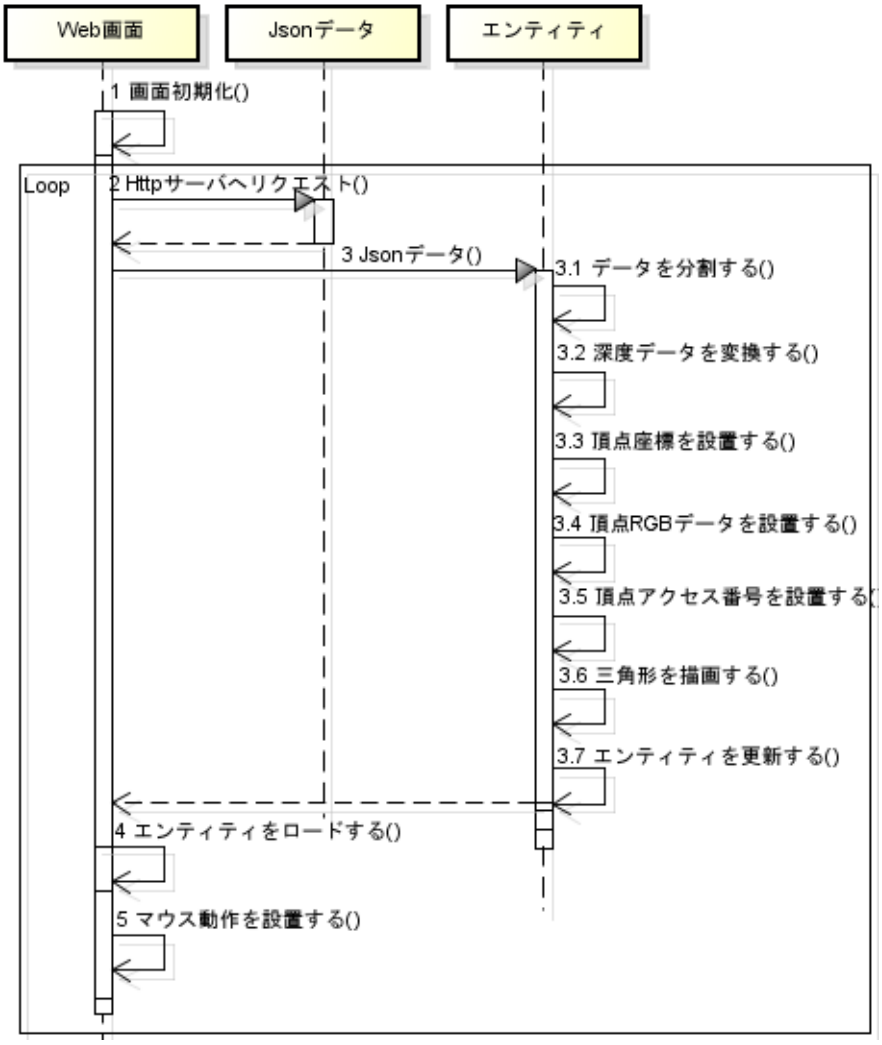
クラス名	humans			
親クラス	なし			
クラス概要				
複数棒人間を描画する				
属性				
可視性	属性・インスタンス名	型	初期値	説明、その他
private	users			userid 配列
private	human			human クラスオブジェクトの配列
メソッド				
可視性	メソッド名	型	引数	説明、その他
public	findUser		userid	userid を存在したかを判断する
public	loadJson		jsonData	関節座標を変換する
public	drawHumans		gl	複数棒人間を描画する

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 xpos ypos 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

20. 3D 空間

20.1. シーケンス図

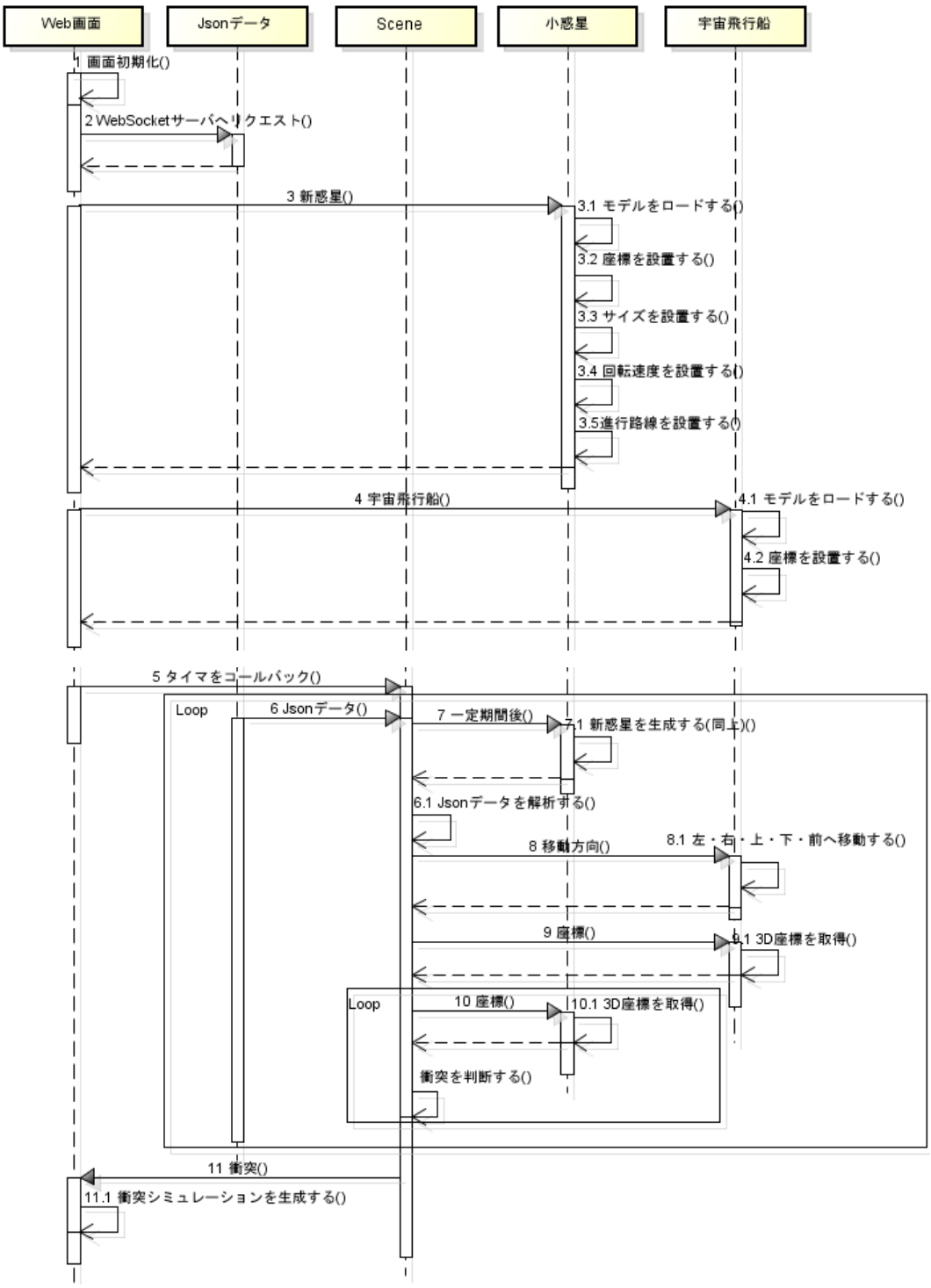


20.2. クラス詳細

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

21. 宇宙飛行

21.1. シーケンス図



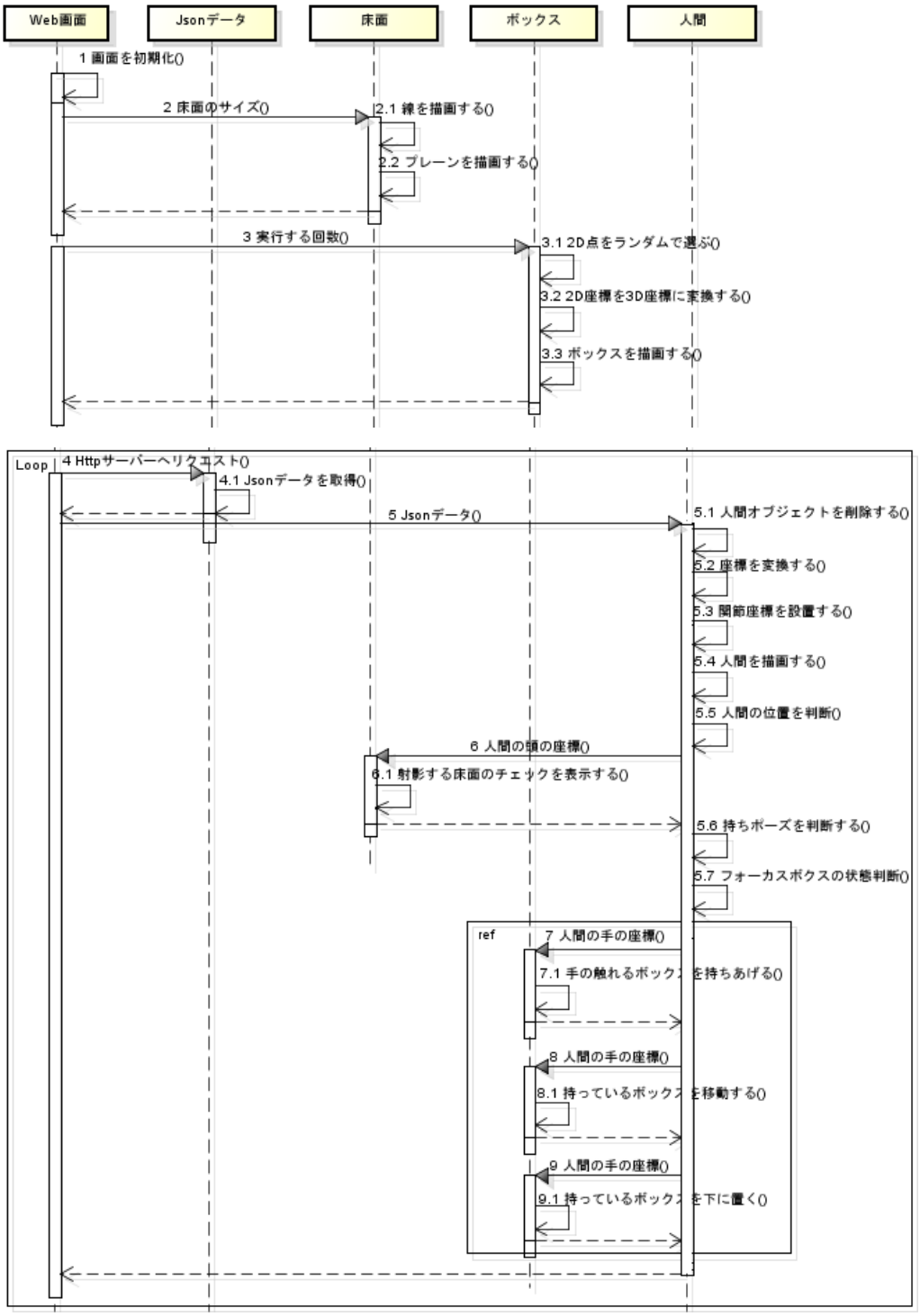
21.2. クラス詳細

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 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

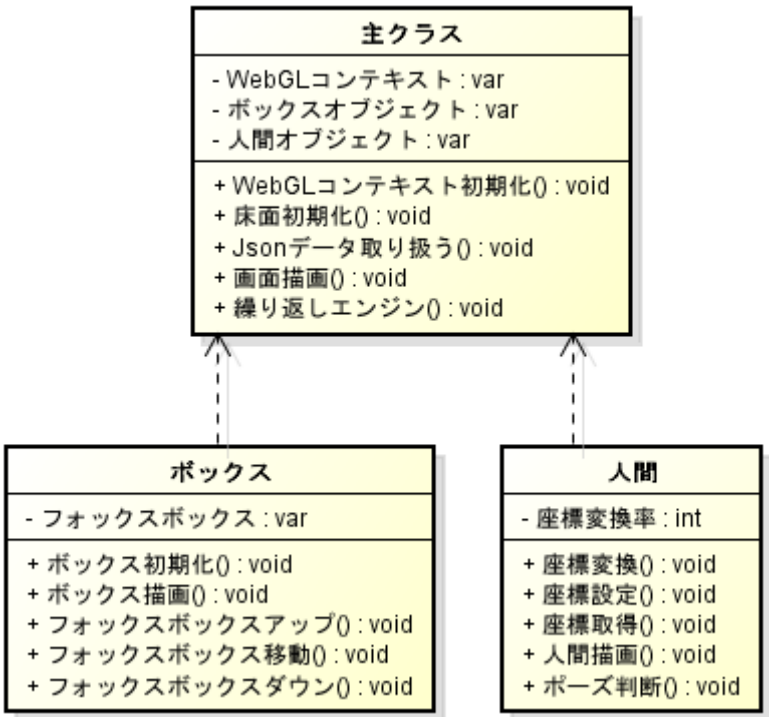
				化
public	initRoll			小惑星の回転速度を設定する
public	loadKinect			Json データを取得する
public	canvasMain			Web ブラウザの起動メソッド

22. ボックスのトランスポート

22.1. シーケンス図



22.2. 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

22.3. 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, x, y, 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 , camera , renderer, 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 dx, dy, 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