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－Kinect　HTTP サーバ構築とクライアント用ライブラリの作成－

劉斌

(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、同大学院に所属する教員から受注したものである。筆者を含めた学生 4 名は、その目標を達成するために様々な機能の開発やテストを行った。本報告書では、本プロジェクトの開発背景からシステムの概要、プロジェクトの経過、そして筆者が担当した部分などについてまとめたものである。

本プロジェクトでは、Kinect に搭載している様々なセンサーを利用して、骨格や深度などのデータを提供するとともに、汎用性の高い機能に着目してユーザ向けのより使いやすい、より多くの機能を備える Kinect サーバを提供することを目標とする。筆者は他の 3 名のメンバーと一緒に、その目的を達成するために、Kineco というチームを組んで、プロジェクト管理にわたって Redmine や GIT[20]などのツールを用いてお互いに知識を共有しながら開発からシステムズテストまで一連のプロセスを行った。

本プロジェクトは、主に「Kinect センサーデータ提供機能」「サーバプッシュ機能」「複数 Kinect 連携機能」という三つの機能を提供するのに加えて、更にそれらの機能をいかにして活用できるかを示すためのサンプルクライアントアプリまでも、同時に提供する。筆者はそのなかの「Kinect センサーデータを提供する機能」を担当した。開発に当たっては、どのようなデータをどうやってクライアント側に提供したらユーザの利便性が高まるかに配慮して、仕様を策定した。そしてデータ提供用の手段である Http サーバの構築や他の機能との連携に配慮したモジュール化、さらに、Http サーバにアクセスしてセンサーデータを取ってくれるというコーディング支援用のライブラリの設計と実装を担当した。

目次

第 1 章　はじめに …… 1
1.1　プロジェクトの目的 …… 1
1.2　報告書の構成 …… 1
第 2 章　前提知識及び関連サービス …… 2
2.1　前提知識 …… 2
2.2　関連サービス …… 3
第 3 章　Kinect サーバシステム …… 5
3.1　システムの目標 …… 5
3.2　システム化の範囲 …… 5
3.3　想定する利用者 …… 6
3.4　構成 …… 6
3.4.1　ハードウェア構成 …… 6
3.4.2　ソフトウェア構成 …… 6
3.4.3　システムの構成 …… 7
3.5　要件 …… 7
3.5.1　機能要件 …… 7
3.5.2　非機能要件 …… 8
3.6　前提条件と制約事項 …… 9
3.6.1　前提条件 …… 9
3.6.2　制約事項 …… 9
3.7　利用シーン …… 9
3.8　開発作業の分担 …… 11
第 4 章　センサーデータ提供機能 …… 12
4.1　機能の概要 …… 12
4.2　機能の位置づけ …… 12
4.3　機能の開発概要 …… 13
4.4　機能の設計 …… 14
4.4.1　センサーデータの提供方式 …… 14
4.4.2　Http サーバのモジュール化 …… 16
4.4.3　Http サーバの実装 …… 19
4.4.4　センサーデータの設計 …… 21
4.4.5　ユーザ識別の実現手法 …… 23
4.5　機能詳細 …… 24
4.6　クライアント用のライブラリ …… 30
4.6.1　開発の目的 …… 30
4.6.2　ライブラリの機能要求 …… 30
4.6.3　ライブラリの構成 …… 31
4.6.4　スクリプト言語対応の HTTP クライアントの考察 …… 32
第 5 章　マネージメントの工夫と施策 …… 33

5.1　工夫点 ……33
5.1.1　プロジェクトの一元化管理 ……33
5.2　振り返りと分析 ……36
5.2.1　要件定義における分析 ……36
5.2.2　プロジェクト マネージメントにおける分析 ……36
5.2.3　センサーデータ提供機能の開発における分析 ……37
5.2.4　今後の課題 ……37
第 6 章　結論 ……38
謝辞 ……39
参考文献 ……40
付録 ……41

図目次

図 2-1　OpenNI の構成[5] 3
図 2-2　Ajax の特徴[10] 4
図 2-3　WebSocket の仕組み[10] 4
図 3-1　システム化範囲 5
図 3-2　システムの通信流れ 7
図 3-3　WebGL で人の動きを再現する 10
図 3-4　サーバプッシュ機能の利用シーン 10
図 4-1　センサーデータ提供機能の位置づけ 13
図 4-2　反復型開発の手順 14
図 4-3　委託元の要望 15
図 4-4　Apache Tomcat を採用したアーキテクチャ 15
図 4-5　LibMicroHttpd を採用したアーキテクチャ 16
図 4-6　Http サーバの構成 16
図 4-7　データ取得のシーケンス図 19
図 4-8　Http サーバの一部クラス図 20
図 4-9　違う状態のユーザクラス図 21
図 4-10　JSON のデータ構造[13] 22
図 4-11　JSON と XML 形式の対比[13] 22
図 4-12　JSON 形式の RGB 情報のサンプル 23
図 4-13　JPEG 形式の RGB データ 25
図 4-14　JPEG 形式のデプスマップ 26
図 4-15　カメラ座標系[18] 26
図 4-16　点群データの例 27
図 4-17　JSON 形式の点群座標の例 27
図 4-18 JSON 形式の骨格データ例 28
図 4-19　ライブラリからの利用シーン 30
図 4-20 インタフェースの詳細 31
図 4-21　ライブラリ利用のデータ流れ 32
図 5-1　カレンダーに記載するチケット 33
図 5-2　一覧のタスクリスト 34
図 5-3　チケット詳細 34
図 5-4　ソースコードのバージョン管理一覧 35
図 5-5　ソースコードの差分比較例 35

表目次

表 3-1 システムのハードウェア構成 6
表 3-2 サーバのソフトウェア仕様 6
表 3-3 非機能要件詳細 8
表 3-4 仕事の分担 11
表 4-1 リクエスト明細 17
表 4-2　レスポンス明細 18
表 4-3 エラーコード表 18
表 4-4 Base6 エンコードの効果 23
表 4-5 Kinect センサーの仕様 24
表 4-6 JPEG 画像形式で RGB データ取得状況 25
表 4-7 骨格情報を取得可能なパーツ 28
表 4-8　イメージ情報付き 3 次元点群 JSON 形式 29

第1章　はじめに

筆者は研究開発プロジェクト（以下、本プロジェクト）において、同大学院に所属する教員（以下、委託元教員）から頼まれたシステムの開発を行っていた。筆者は、委託教員が要望したシステムの中、「Kinect を用いて、搭載しているセンサーから取られるデータを提供する機能」という部分を担当して、4 人チームの一員として、システムの開発を請け負った。

成果物として、設計段階では、委託元の要望を踏まえて、提供すべきセンサーデータのタイプを洗い出して機能の仕様書を策定した。また、拡張性や機能性を確保するため、センサーデータを提供する Http サーバをオブジェクト指向でモジュール化した、データの流れとクラス構成を示す用のシーケンス図やクラス図を作成した。

実装段階では、OpenNI のフレームワークを利用して、Kinect に搭載されているセンサーから取得した RGB 画像や深度マップ、そして人の重心、骨格といった基本のデータに加え、さらに、検出また識別された複数人の ID や深度から算出した点群の 3D 座標[4]までの情報を外部へ提供する Http サーバを構築した。また、リクエストを出してデータを取ってくれるコーディング支援用のライブラリも開発した。

1.1　プロジェクトの目的

本プロジェクトは、Kinect に搭載している様々なセンサーを利用して、骨格や深度などのデータを提供した上に、汎用性高い機能に着目してユーザ向けのより使いやすい、より多くの機能を備える Kinect サーバを提供することを目的とする。

1.2　報告書の構成

本報告書は、6 章で構成されている。第 2 章では、本プロジェクトの背景知識と主に調査した関連サービスを取り上げる。第 3 章では、本システムの概要の紹介をはじめ、システムの目的から想定する利用者、利用シーン、そしてシステムの構成や要件および設計について述べる。第 4 章では、筆者が担当した部分について説明する。そして第 5 章では、筆者個人がとらえた､本プロジェクトの進行における工夫､分析および今後の課題について述べる。最後に第 6 章で、結論として、本プロジェクトの開発を通して筆者が体験したことをまとめる。

第2章　　前提知識及び関連サービス

本章では、本プロジェクトの背景となる前提知識および、本プロジェクトで使われている技術を述べる。

2.1　　前提知識

本節は、Kinect の概要、そしてプロジェクトの開発に当たって使われていたオープンソース技術の紹介、また、ほかの関連サービスなど本節では述べる。

近年の情報端末操作に対する新たなアプローチの一つとして NUI(Natural User Interface)が挙げられる。NUI を実現するゲームシステムのアクセサリとして発売されている Kinect は、その利用方法はゲームにとどまらない。例えば、従来のウェブカメラ中心のジェスチャ識別や画像転送などの研究は Kinect を使うことで手法を一変させた。Kinect を PC に繋いで利用するためには OpenNI は登場した。

- Kinect

Kinect は、Microsoft 社から発売されている Xbox 360 向けコントローラである。本体に搭載されている 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。本研究開発プロジェクトでは、そういったセンサーを用い様々な機能を提供するサーバおよびその機能を活用したサンプルクライアントアプリの開発をおこなう。

- OpenNI

OpenNI とは Kinect を開発している PrimeSense 社などが中心となって開発している API 群で、今のところ Kinect の非公式 SDK という位置づけである。　OpenNI は RGB カメラ（画像を取得)、3D センサー（距離を測る)、IR カメラ（3D センサーの為の赤外線出力)、オーディオデバイス（マイク）といった、Kinect で利用可能なカメラ、センサーを使用するためのインタフェースを提供する[5]。具体的な位置づけは以下の図 2-1 で示す。

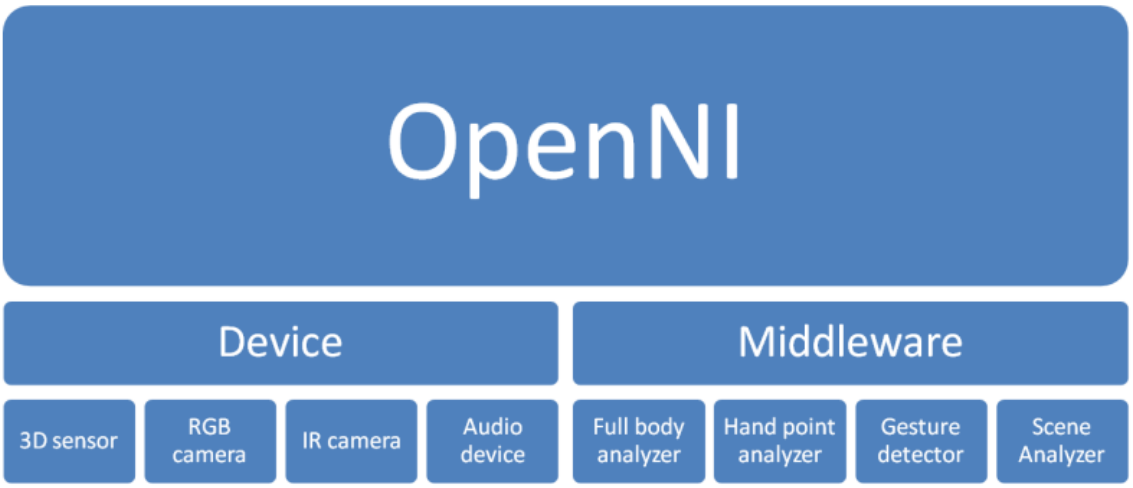


図 2-1　OpenNI の構成[5]

OpenNI は Kinect のデバイスをコントロールする Device 部とそのデータから画像処理を行い，そしてユーザの検出やジェスチャ識別などを行う Middleware 部があり，OpenNI はそれらを統合して扱うインタフェースとなる[5]。

2.2　関連サービス

本節は調査した関連サービスを述べる。まずはクライアントとサーバ間で双方向通信について、以下のような技術を調べた。

- Ajax

Ajax とは、クライアントが非同期にサーバサイドへポーリング(一定間隔でサーバをチェックする)できるようになる仕組みである。そしてサーバからのメッセージを（ほぼ）リアルタイムに配信することができる。しかし、ポーリングの間隔分の遅延が発生するため、厳格なリアルタイム通信は Ajax だけで実現することはできない。また、データ変更があるなしに関わらずチェックを行うため、CPU やメモリを必要以上に使用してしまう[10]。

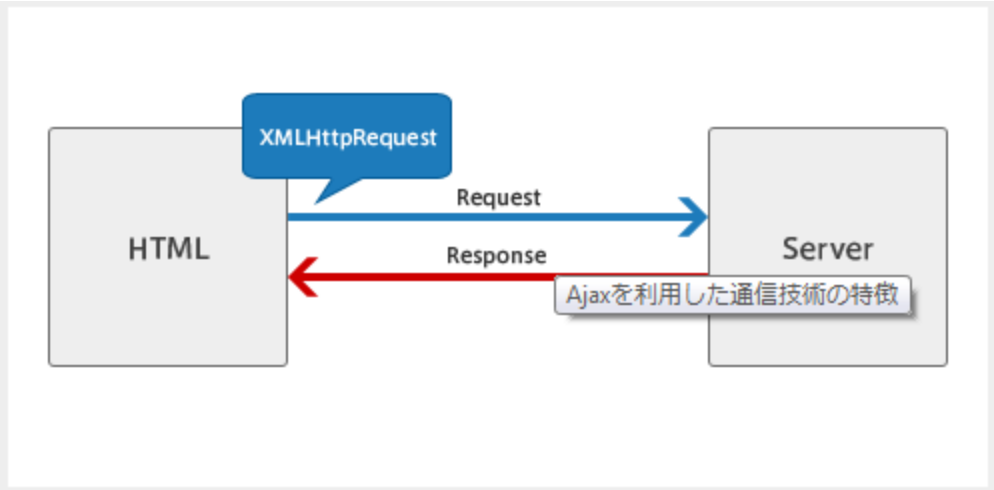


図 2-2　Ajax の特徴[10]

- WebSocket

WebSocket は、クライアントとサーバ間で双方向通信を実現するための仕組みである。通信仕組み以下の図 2-3 で示す。

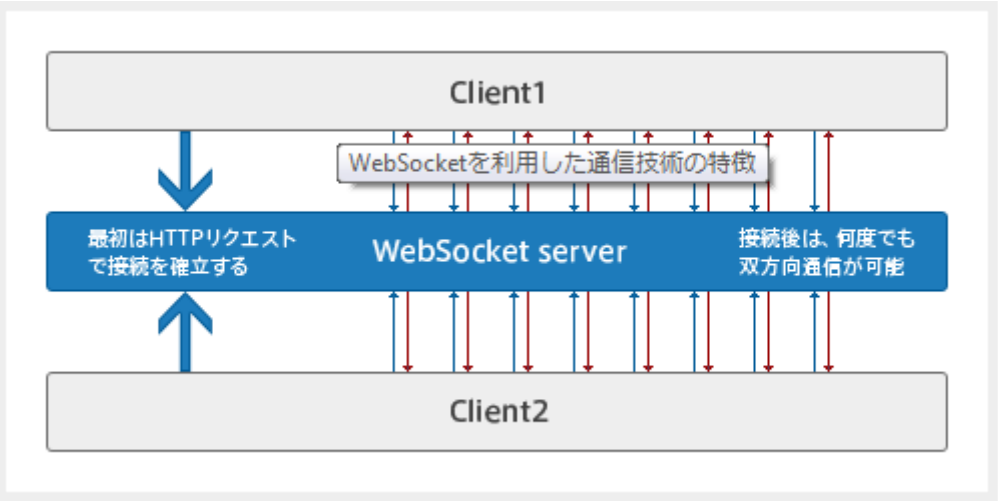


図 2-3　WebSocket の仕組み[10]

WebSocket では、サーバとクライアントが一度コネクションを行った後、その後の通信を全てそのコネクション上で WebSocket 専用プロトコルを使用して行い。Ajax の手法と比較すると以下のようなメリットがある [10][12]。

- 通信ごとに新しいコネクションを張る必要がなくなる。
- HTTP プロトコルではない、専用のプロトコルを使用するため、通信量が少なくてすむ。

これらパフォーマンス面においては他の技術よりリードしているので、今回のサーバプッシュ機能の開発に当たって採用された。

第3章　　Kinect サーバシステム

本章では、本プロジェクトで開発した Kinect サーバシステムの目標、構成及び各機能の詳細を述べる。

3.1　　システムの目標

Kinect に搭載している様々なセンサーを利用して、骨格や深度などのデータを提供した上に、汎用性高い機能に着目してユーザ向けのより使いやすい、より多くの機能を備える Kinect サーバおよびサンプルクライアントアプリの開発をおこなった。目標としては、

- Kinect センサーデータをリアルタイムに提供する Http サーバを構築する。
- 複数台の Kinect を連携させて、より広範囲の 3D データを取得する。
- ジェスチャやポーズなどの識別そしてユーザの検出における WebSocket によるサーバプッシュ機能の実現。
- Kinect サーバ提供する機能を活用したサンプルクライアントアプリを提供する。

上記目標の実現を目指し、筆者らは本システムの開発を行った。

3.2　　システム化の範囲

3.1 節で述べた委託元のニーズや本プロジェクトの目標に従って、リアルタイムに発信する Kinect サーバには、RGB や骨格などのセンサーデータを提供する機能、WebSocket によるサーバプッシュ機能、複数連携機能とサンプルクライアントを提供する機能という 4 つの機能が必要である。システム化範囲は図 3-1 で示す。さらに、より使いやすいため、データの取得には、HttpRequest を出してくれるクライアント用のライブラリもシステム化の範囲に入れるべきと委託元に話し合ってから認識した。

図 3-1　システム化範囲

3.3　想定する利用者

本システムで想定される利用者を、以下に示す。

■　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。

■　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。

3.4　構成

本節では、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およびシステムの構成を示す。

3.4.1　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 3-1 で示す。

表 3-1 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

なお、運用時のサーバの CPU, メモリ、ハードディスク容量は、上記の性能以上を満たすものと想定する。

3.4.2　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 3-2 で示す。

表 3-2 サーバのソフトウェア仕様

サーバ OS	Ubuntu 11.2
Web サーバ	LibMicroHttpd
オープンソース	OpenNI,Base64Encoder,LibcURL
IDE	VI
開発言語	C++
	Java (Java 版のクライアント側用ライブラリ開発)
	Shell Script（コンパイルファイルの作成）

3.4.3　　システムの構成

本システムは大きく二つの部分に分かられ、サーバ側には筆者担当したセンサーデータ機能を始め、サーバプッシュ機能や複数 Kinect 連携機能と三つの部分があり。
そして、今回のシステムの利用には、クライアントからのアクセス手段も限定されている。例えば、サーバプッシュ機能の実現には WebSocket 接続が不可欠であるによって、JavaScript からの利用は前提条件である。機能間の連携や外部とのやり取りについては、図 3-2 で示す。

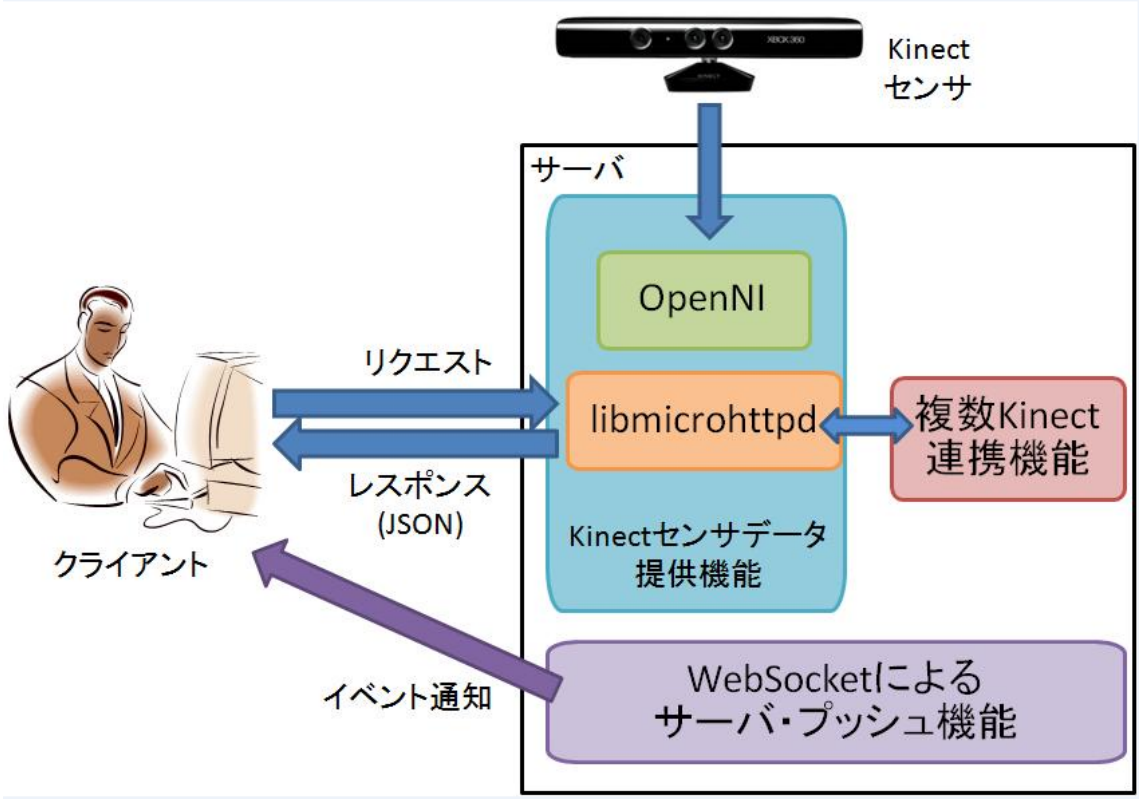


図 3-2　システムの通信流れ

3.5　　要件

本節では、要件定義フェーズで定義された機能要件と非機能要件について述べる。

3.5.1　　機能要件

3.2 節に述べたように、本システムの機能は大きく 3 つに分かれている。具体的には、Http サーバでセンサーデータを提供機能、WebSocket によるサーバプッシュ機能、複数台の Kinect の連携機能及びその 3 つの機能を検証用のサンプルクライアントアプリを提供するという 4 つの仕事分担があり、本節はそれぞれの機能要件について詳しく説明する。

- センサーデータ提供機能

本機能では、委託元の要望に踏まえて、取得したセンサーデータを Http サーバ通じて外部へ提供するという基本要件があり、さらに、ほかのメンバー担当する機能と連携するため、Http サーバをモジュール化して構築した。そのなか、データ取得モジュールでは、センサーから RGB 情報や人の検出する際の骨格、重心といったデータをリアルタイムに取得してメモリに保存する。一方、センサーデータ提供モジュールでは、クライアント側からのリクエストを受け取って、分析した上に常に更新しているメモリからユーザが欲しい情報をユーザによって扱い易い形式でレスポンスとしてクライアント側に転送する。

- WebSocket によるサーバプッシュ機能

WebSocket を利用して、サーバとクライアント側の接続を維持しながら、サーバに繋がる Kinect によってユーザの検出やポーズの識別、手の動きの追跡などのシグナルをクライアント側にプッシュすること。それによって、様々な便利な利用シナリオを実現できる。例えば、ユーザ検出次第、クライアント側にしらせて、そのユーザの情報をもっと知りたい場合、センサーデータ提供機能を利用して詳しい情報を取得可能である。

- 複数 Kinect 連携機能

一台の Kinect は取れる情報の範囲が限られているので、台数を増やして、それぞれ取ってきた情報をあわせてより広い範囲の情報を取れるという機能である。具体的には、一つの独立サーバがあって、そこでネットワークを経由して複数の Kinect から収集してきた情報をあわせている。

3.5.2 非機能要件

本システムでは、非機能要件に関しては表 3-3 の達成を目標とする。

表 3-3 非機能要件詳細

要件	概要
Http サーバの応答時間	1 秒以下
レスポンスタイム	1 秒以下（pointCloud など転送際 5 秒以下）
拡張性	仕様追加を想定して、コーディングのスタイルに拘って、可読性の向上に努める
移植性	オープンソースのライブラリを利用する
	Linux 環境で通用するリリース手順
汎用性	クライアント側用のコーディング支援用のライブラリを提供する（Java, C++両言語版揃い）

3.6 前提条件と制約事項

本節では、本システムの導入と運営における前提条件と制約事項について述べる。

3.6.1 前提条件

本システムの導入に置いて、以下を前提条件とする。

- 対象とするユーザは、PC を用いてネットワーク経由で本システムにアクセスする。
- サーバに Kinect を一台つなげる（複数連携の場合は導入マニュアルに参考してください）。
- C++や Java をコンパイルするツールがインストール済みの Linux 環境である。
- Kinect のドライバを予めサーバにインストール済み。

3.6.2 制約事項

本システムの運用に置いて、以下の制約を設ける。

- サーバプッシュ機能を利用するには、クライアント側は WebSocket 対応のブラウザであることが必要である。
- 検証用サンプルクライアントの一つとして、WebGL を利用して識別されたユーザの骨格情報をブラウザ上で再現するという機能を利用するには、WebGL 対応のブラウザが必須である。
- RGB や深度データを転送しやすいため、予めサーバ側でエンコードすることによって、クライアント側での利用はエンコードする必要がある。

3.7 利用シーン

本節は、本システムの利用シーンを述べる。

- WebGL によってユーザの動きをブラウザ上で再現
  WebGL（ウェブジーエル）は、ウェブブラウザで 3 次元コンピュータグラフィックスを表示させるための標準仕様である[11]。このシーンでは、センサーデータ提供機能の Http サーバにアクセスして、リアルタイムに Kinect で識別された人の骨格の各パーツの 3D 座標を取得してから WebGL によってブラウザ上で図 3-3 示すようなボーディの動きを更新して再現する。

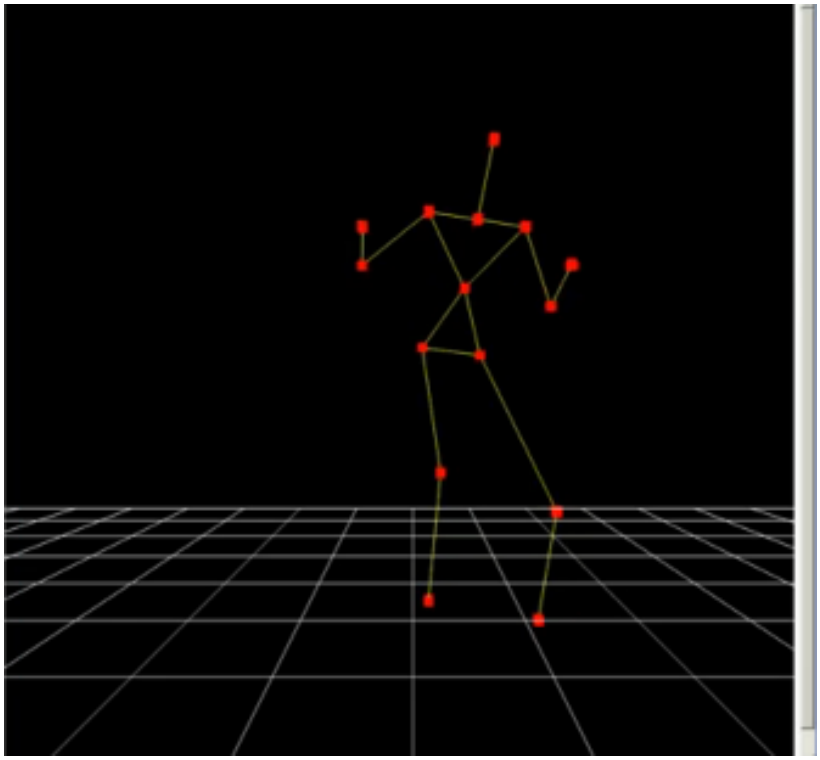


図 3-3　WebGL で人の動きを再現する

さらに、好きなアニメキャラを WebGL の描く段階で代入して、自分の動きを真似てブラウザ上で踊れるアニメキャらも作成可能である。

- リアルタイム人の検出、報告シーン
  このシーンでは、前提条件として、クライアントとサーバの間、WebSocket の接続を維持することが必要である。その後、サーバに繋がっている Kinect のセンサー範囲内で誰か検出されたら、サーバプッシュ機能を利用して、検出シグナルをすぐクライアント側に知らせる。

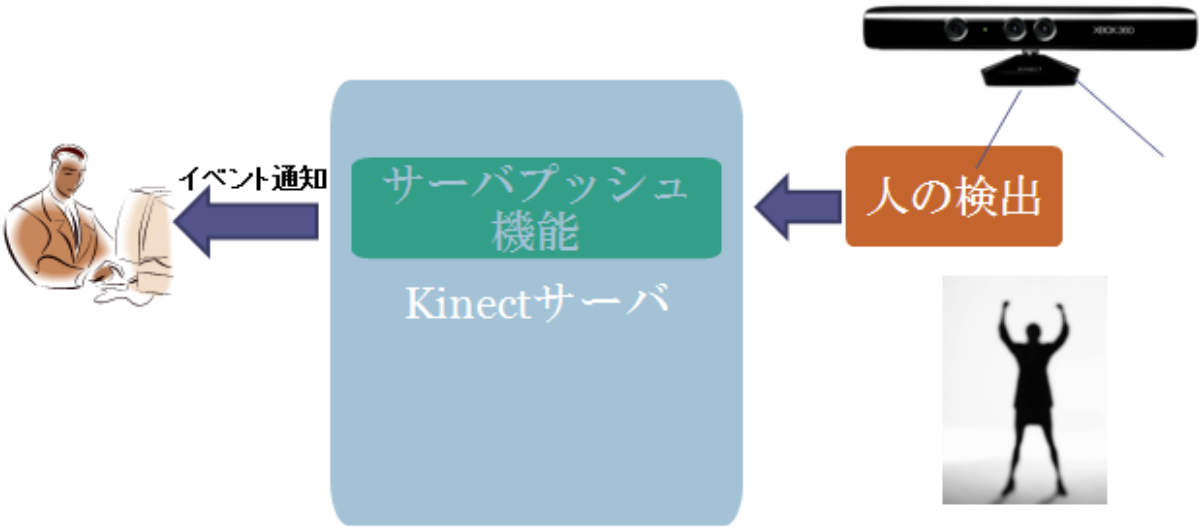


図 3-4　サーバプッシュ機能の利用シーン

こうして人の入退室を監視するようなシーンは利用のシナリオとしてさらなる進化することが可能である。例えば普通防犯カメラの場合、不審者が見つかっても記録することしかできない。一方、Kinect を防犯カメラとして利用して、サーバプッシュ機能を活用して、入出禁止状態の部屋で人を検出したらいち早くクライアントに知らせて、警備員などに報告することは可能になる。

3.8 開発作業の分担

プロジェクトメンバーそれぞれの分担を表したものを表 3-4 に示す。

表 3-4 仕事の分担

モジュール種類	モジュール名	担当者
サーバ部分	WebSocket によってサーバプッシュ機能	茂木
	Kinect センサーデータ提供機能	劉
	複数 Kinect データ連携機能	小菅
クライアント部分	諸機能検証用のサンプルクライアントアプリ	朱
	コーディング支援用クライアントライブラリ	劉
	WebSocket 通信のクライアント側	茂木

第4章　センサーデータ提供機能

本章では、筆者が担当したセンサーデータ提供機能を詳しく述べる。

4.1　機能の概要

本機能では、OpenNI のフレームワークを利用して、Kinect に搭載されているセンサーから取得した RGB 画像や深度マップ、そして人の重心、骨格といった基本のデータに加え、さらに、検出また識別された複数人の ID や深度から算出した点群の 3D 座標[4]までの情報を Http サーバ通じて外部へ提供する。サーバとクライアント間のデータやり取りは主に JSON 形式を採用して行う。一方、RGB 画像などのデータの転送には、そのままバイナリタイプのデータを提供した上に、リアルタイムに画像再現するため、ピクセルごとの RGB 情報を抽出して、エンコーディングした後は JSON 形式に格納してクライアント側へ転送する。そして、ディコーディング後クライアント側では Canvas などのツールを使って、一個一個のピクセルを復元して、RGB 画像を再現するという仕様もサッポートする。

また、機能実現に当たって構築した Http サーバには、複数 Kinect 連携機能にアクセス用のインタフェースがあり、さらに、サーバプッシュ機能との連携ポイントも設けている。

4.2　機能の位置づけ

本システムのシステム化範囲には、「サーバプッシュ機能」、「センサーデータ提供機能」、「複数 Kinect 連携機能」そして「サンプルクライアントアプリ」を含んでいる。筆者が担当した機能では、センサーデータをネット上でユーザに提供するため、四つのモジュールを含める Http サーバを構築した。そこで、複数 Kinect 連携機能であわせてくれる広い範囲の情報を取得する用のインタフェースも用意する。更に、Kinect 初期化モジュールで同じコンテキストを共有することでサーバプッシュ機能と連携して、ユーザの検出次第、そのユーザの情報（重心、骨格など）の取得を可能にする（ユーザ関連の情報を共有するので、ユーザ ID も共通である）という図 4-1 示す位置つけである。

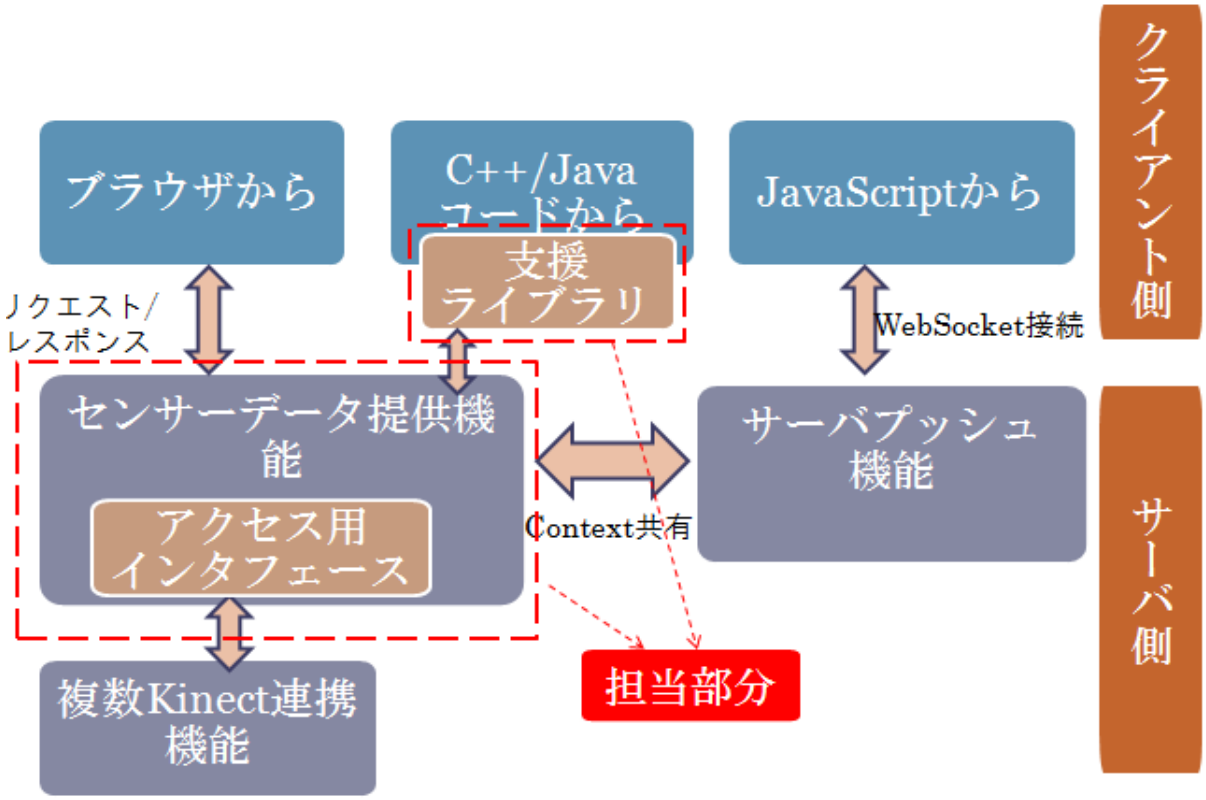


図 4-1 センサーデータ提供機能の位置づけ

4.3 機能の開発概要

本機能の開発は図 4-2 で示す反復型でイテレーションを二回して行われた。

- 第一イテレーションでは、主に、OpenNI を使って、取得可能なセンサーデータを検証しながら、センサーデータ提供機能の仕様を策定した。それに従って、外部のブラウザからのリクエストを受け取って判断してから、要求のセンサーデータを JSON 形式に変更してクライアントへ返すという基本の仕様を満たした Http サーバを構築した。

- 第二イテレーションでは、まず、可読性と拡張性を向上するため、Http 構築に当たって長くなったメーンコードをリファクタリングした。また他のメンバーと連携しやすいため、センサーデータ提供機能を四つのモジュールに分けた。

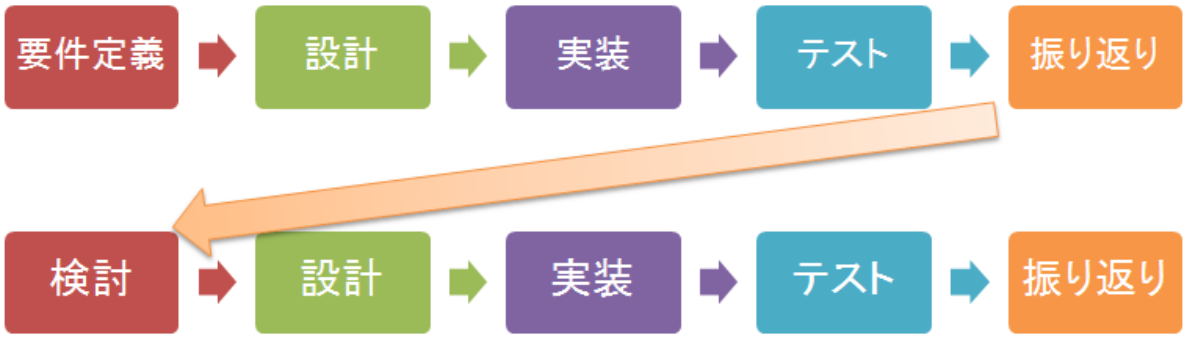


図 4-2　反復型開発の手順

4.4　機能の設計

本機能は、システム全体と外部環境との通信を実現する機能の一つである。Kinect から取得するデータ情報を提供する一方で、担当機能の位置づけという節で述べるように、他のメンバーが担当するサーバプッシュ機能や複数 Kinect の連携機能においても利用される。本機能を実現するために、筆者は以下のような設計を行った。

4.4.1　センサーデータの提供方式

図 4-3 で示す委託元の要望を踏まえて、センサーデータ提供機能は以下の要求を満たすこととした。

- ネットワークを介してデータを提供 。
- リアルタイム性を重視する方が望ましい。
- クライアント側でデータの取得 。
- モバイル端末からでも扱えること。
- Web を用いたサンプルクライアントの実装。

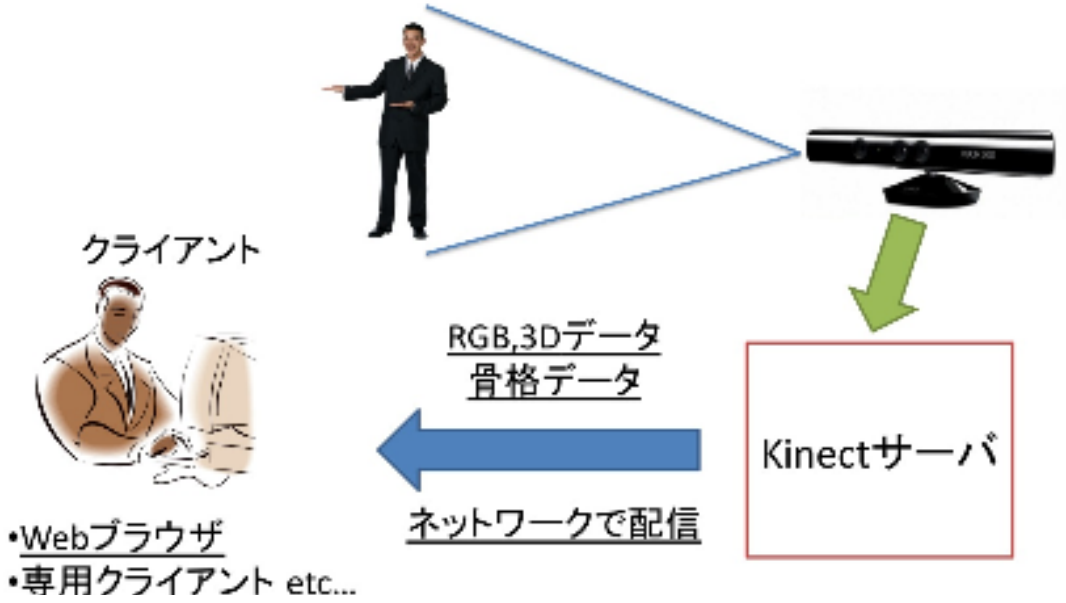


図 4-3　委託元の要望

こうした要望に応じて、 Http サーバによるデータを提供するという提案に辿りついた。そして Http サーバについては、最初に図 4-4 で示すように Apache の Tomcat を Http サーバとして考えて次のようなアーキテクチャを提案した。

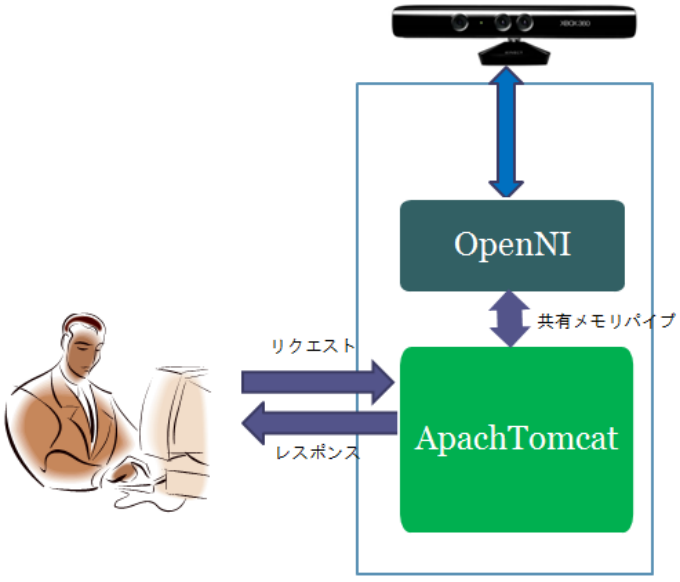


図 4-4　Apache Tomcat を採用したアーキテクチャ

しかし、このような構成案で以下の二つの問題点がある

- Apache と Kinect のプログラム間通信が必要となるシステム全体の構成が複雑になってしまう 。
- 単なるウェブ サーバとして Apache ほど高機能な機能は必要ない 。

それによって、よりシンプルな構成を改めて考えなおしたところ、GNU LibMicroHttpd という C 言語用のライブラリが見つかった。GNU LibMicroHttpd とは軽量なアプリ組込み型 HTTP サーバ作成のためのライブラリであり、HTTP デーモンの作成ことによって、HTTP リクエストの処理 やレスポンスの作成 など基本の HTTP サーバの仕様を満たした上、さらに、C 言語ライブラリなので、高速処理が可能 、また複数のポートを Listen できるといった利点を持っている[6]。 これを使って、C++ソースコード内で簡単に Http サーバを構築できる一方で、直接 OpenNI と連携して、Kinect から取った情報をそのままレスポンスに書き込めるという利点もある。LibMicroHttpd を採用して構築した HTTP サーバの構成は以下の図 4-5 で示す。

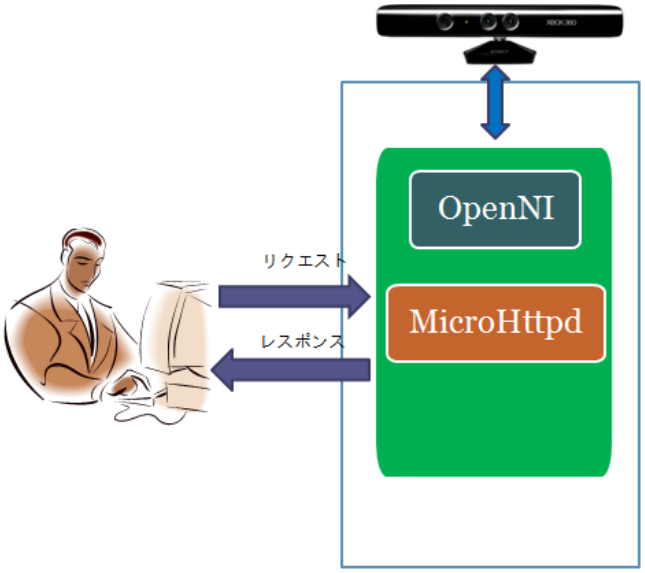


図 4-5 LibMicroHttpd を採用したアーキテクチャ

4.4.2 Http サーバのモジュール化

本サーバでは、四つのモジュールによって構成され、モジュール間のデータ流れやサーバの構成を図 4-6 で示す。

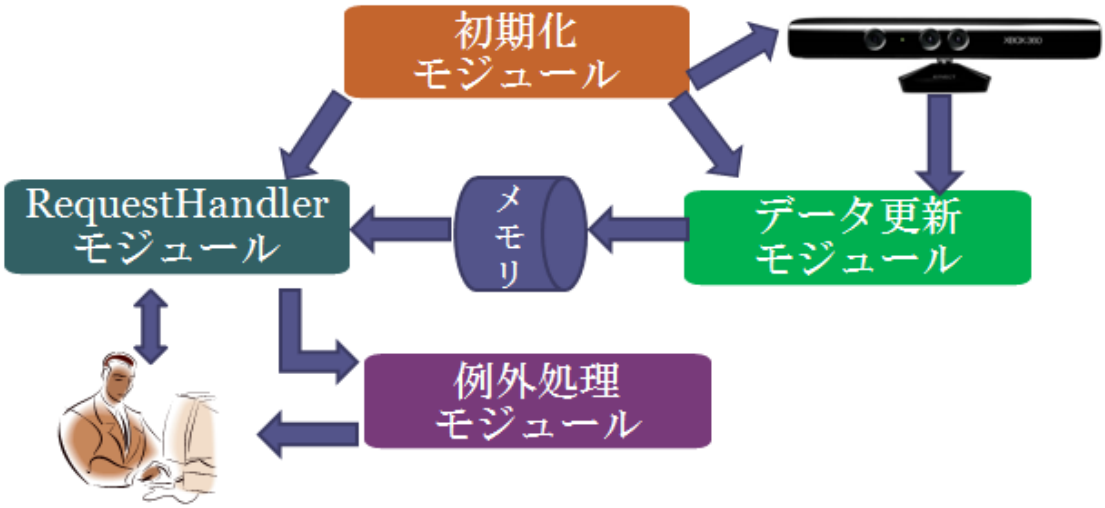


図 4-6 Http サーバの構成

- Kinect を初期化するモジュール

本モジュールでは、各モジュールの初期化作業を行うモジュールである。具体的に、まず Kinect 制御用の設定ファイルを読み込んでコンテキストを新規作成。そして RequestHandler モジュールのデーモンサービスを起動し、外部からの HttpRequest を受け取れるようにする。また、データ更新モジュールのプロセスを始動させる。

- データ更新モジュール：

本モジュールでは、Kinect のセンサーから RGB や深度などのデータを取得してメモリに書き込ませるというプロセスを繰り返す。

- RequestHandler モジュール：

本モジュールでは、コーアの発信役割を担って、外部からのアクセスを分析し、取得したいデータのタイプを判断して、常に更新するメモリから対応のデータを取り出して、そして必要であればいったん JSON 形式に変更してからまた HTTP レスポンスに書きこんでクライアントへ転送する。外部からのアクセスに対する詳しい応答は表 4-1, 表 4-2 で示す。

表 4-1 リクエスト明細

リクエストタイプ	引数	タイプ
getRgbData	NULL	
getJpegImage	NULL	
getDepthImage	NULL	
getDepthImageData	NULL	
getPointCloud	NULL	
getSkeletonJointPosition	NULL	
getCenterPoints	NULL	
getUserIds	NULL	
getCalibratedUserIds	NULL	
getNumberOfDetectedUsers	NULL	
getNumberOfCalibratedUsers	NULL	
getDepth	指定したポイントの座標(x, y)	int
getSkeletonById	取得したいユーザの Id	int
getCenterPointById	取得したいユーザの Id	int
getDepthById	取得したいユーザの Id	int
getMultiPointCloud	NULL	

表 4-2　レスポンス明細

リクエストタイプ	レスポンス	タイプ
getRgbData	JSON 形式のイメージデータを取得する	JSON テキスト
getJpegImage	JPEG 形式のイメージデータを取得する	JPEG バイナリ
getDepthImage	JPEG 形式のデプスマップを取得する	JPEG バイナリ
getDepthImageData	JSON 形式のデプスマップデータを取得する	JSON テキスト
getPointCloud	カメラ座標系における全ての素子の 3D 座標を取る	JSON テキスト
getSkeletonJointPosition	複数ユーザの骨格スケールトンのジョイントポイントを取得する	JSON テキスト
getCenterPoints	複数ユーザの重心の座標を取得する	JSON テキスト
getUserIds	検出されている全てのユーザ ID を取得する	JSON テキスト
getCalibratedUserIds	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	JSON テキスト
getNumberOfDetectedUsers	現在の検出したユーザ数を取得する	JSON テキスト
getNumberOfCalibratedUsers	PSI によってカリブレーションされたユーザ数を取得する	JSON テキスト
getDepth	指定した座標（point.x,point.y）の距離（depth）を取得する	JSON テキスト
getSkeletonById	Id によってユーザの骨格情報を取得	JSON テキスト
getCenterPointById	Id によってユーザの重心の座標を取得	JSON テキスト
getDepthById	Id によってユーザの深度を取得	JSON テキスト
getMultiPointCloud	複数連携機能で合わせた PointCloud を取得	JSON テキスト

- 例外処理モジュール：

本モジュールは、RequestHandler モジュールで誤った URL また追加引数がたりないなどと判断して生成したエラーコードを受け取って、エラーの詳細を生成してクライアント側に知らせる。エラーコードとエラーのディスクリプションは表 4-3 で示す。

表 4-3 エラーコード表

エラーコード	ディスクリプション
E01	URL エラー
E02	未入力の引数がある
E03	入力した引数は形式が間違う
E04	ID に該当するユーザが存在しない
E05	引数のタイプが間違う
E06	引数の値は認定範囲を超える
E07	システムエラー
E08	連携した PointCloud データはまで完成していない

4.4.3 Http サーバの実装

発信サーバの開発には、主に二つの段階において行われた。

第一イテレーションで定められた仕様を満たすため、殆どのデータの取得に関するソースコードは一つのファイルに纏めて書いていた。この場合の問題は二つがある。

一つは、仕様が追加されるたびに、データの取得や発信について関数がどんどん増えていってしまい、同じプロセスを何箇所で繰り返すという無駄なソースコードを生じること。もう一つより重大な問題は、他の機能と連携するたび、既存のソースコードはまず可読性が低い、そして、関数同士の繋がりが多くて、拡張しにくいという点もある。そこで、C++のオブジェクト指向という特徴を生かして、第二イテレーション段階で、既存のソースコードを徹底にリファクタリングした。

具体的には、以下の設計ポイント節で述べる。

■ 設計のポイント：

この節で、センサーデータ提供機能で構築したサーバの実装における設計ポイントと工夫したところを述べる。

- リアルタイム応答性の確保

  Kinect の初期化やセンサーを起動するには時間がかかるので、クライアント側からリクエストを受け取ってからまたセンサーにデータを取ってくるという指示を出したらすでに遅くなる。そういった遅延を解消するため、Http サーバを起動すると同時に、Kinect を初期化して、全てのタイプのセンサーデータの更新を繰り返して指示する。また、センサーから取得した最新のデータを一定のメモリ空間に格納してクライアント側への転送の準備をする。そうすると、リアルタイムでクライアント側からのリクエストを素早く応答してセンサーデータを提供することが可能になる。このような一連の流れは図 4-7 で示す。

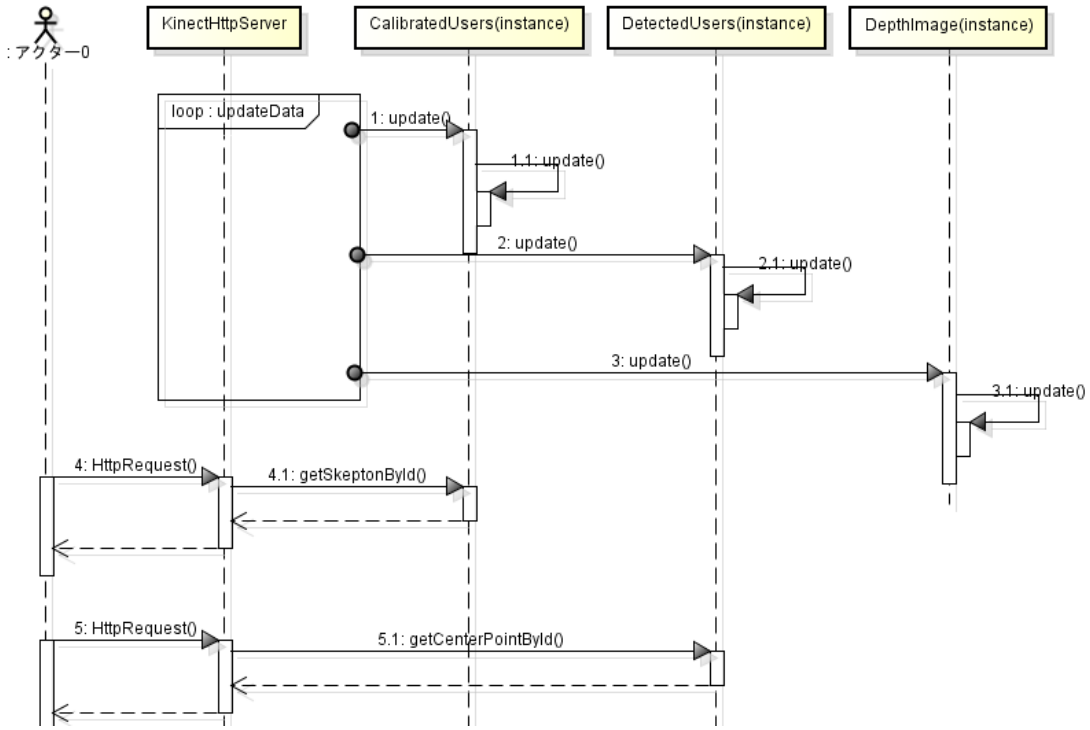
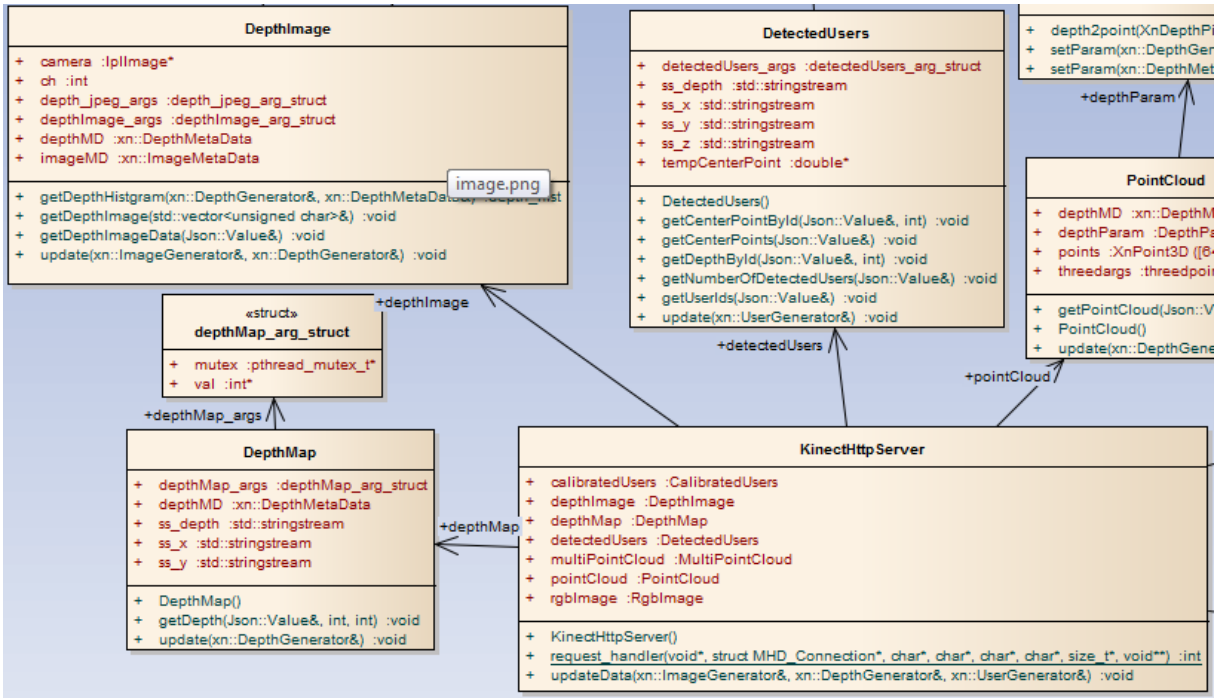


図 4-7　データ取得のシーケンス図

- センサーデータの更新と取得をカプセル化。

  センサーデータの更新方法はデータタイプによって異なる、そして、センサーデータを取得してクライアント側へ転送する前に JSON 形式に変換するルールも違う。そうした相違点をデータごとのインスタンスに隠すため、オブジェクト指向の設計[3]方針から考えて、図 4-8 のようなクラス設計した。メーンソースである KinectHttpServer では、各データタイプのインスタンスを持ち、センサーデータの更新は各インスタンスの更新方法を呼び出すだけである。そして、JSON 形式データの取得に関しては、予め KinectHttpServer で定義した Jason::Value タイプの変数が参照で各インスタンスの取得関数（exp: getPointCloud）に渡される。こうして JSON 形式データの組み立ても各インスタンスに任せた。

図 4-8　Http サーバの一部クラス図



- 仕様によってクラスの中身を決める。
  例えば、ユーザの骨格情報を取得するには、PSI（標準ポーズ）が必要である。つまり、識別（calibrated）されたユーザだけから骨格情報の取得は可能である。一方、ユーザの重心など情報は検出（detected）した全てのユーザ（識別されたユーザももちろん）から取得可能であり、同じユーザだけど、自身の状態によって適応する仕様も限られている。そこに、ユーザには、図 4-9 で示すように、CalibratedUsers と DetectedUsers という 2 つタイプのクラスを作って、それぞれのクラスに適応する仕様の関数を持たせる。

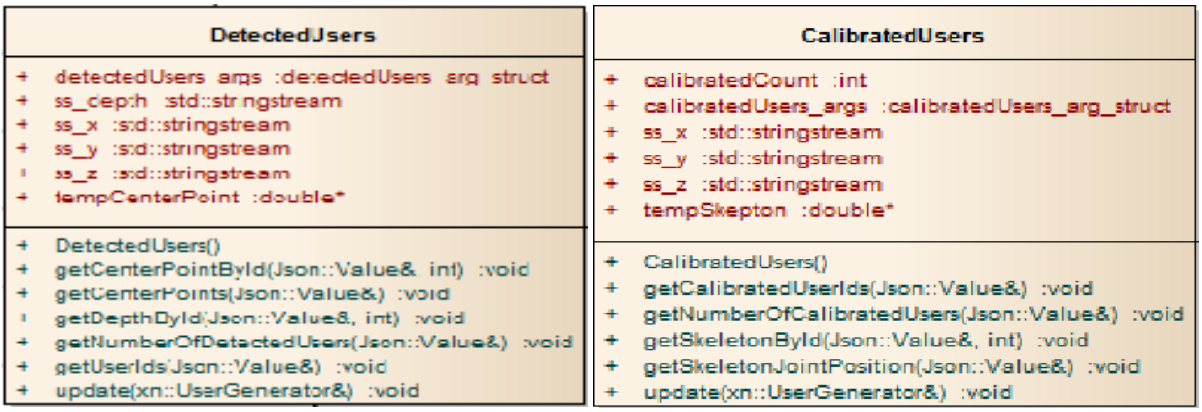


図 4-9　違う状態のユーザクラス図

- 拡張性への配慮
  本機能は他のメンバー担当する複数 Kinect 連携機能やサーバプッシュ機能にも利用されるため、メーンクラスに拡張ポイントを設けている。そこでメンバー変数として複数連携機能を実現するクラスのインスタンス（図 4-8 で示した multiPointCloud）を新規生成して、複数 Kinect から取得する位置情報をあわせるには多少時間がかかって、クライアント側からのリクエストを受け取ると一旦位置情報合わせのフラグをチェックして、無事終了するとまたセンサーデータ提供機能を呼び出してクライアント側へ転送する。そして、サーバプッシュ機能と連携するためには、同じユーザジェネレータを共有する必要があり、というわけで、メーンクラスにはユーザジェネレータがメンバー変数ではなく、独立に定義されている。そうすると、サーバプッシュ側の利用にはメーンクラスのインスタンスを維持する必要もなくなる。システム全体としての構成はより簡潔にした。

4.4.4　　センサーデータの設計

■　データフォーマットの検討

サーバとクライアント間のデータのやり取りは軽量のデータ交換フォーマットにしてほしいという委託元からの要望を踏まえて、かつ将来システムを公開して、オープンソースプロジェクトとしてプログラマに使ってもらいやすいため、C,C++,C#, Java, JavaScript その他多くの C ファイミリーの言語を仕様するプログラマにとっては、馴染み深い規約が使われている JSON を理想的なデータ交換言語にすることになる[8]。

JSON とは JavaScript Object Notation の略で、二つの構造を基にしている。

- 名前/値のペアの集まり、様々な言語で、これはオブジェクト、レコード、構造体、ディクショナリ、ハッシュテーブル、キーのあるリスト、連想配列として実現されている。
- 値の順序付きリスト。殆どの言語で、これは配列として実現されている。

これらは、普通的なデータ構造である。つまり実質的に、現代の全てのプログラミング言語が、いずれの形にせよサポートしている。JSON では、以下の形式をもっている。オブジェクトは、順序付けされない名前/値のペアのセットである。オブジェクトは、{（左の中括弧）で終る。各名前の後ろには、:（コロン）が付く。そして、名前/値のペアは、（コンマ）で区切られる[13]。

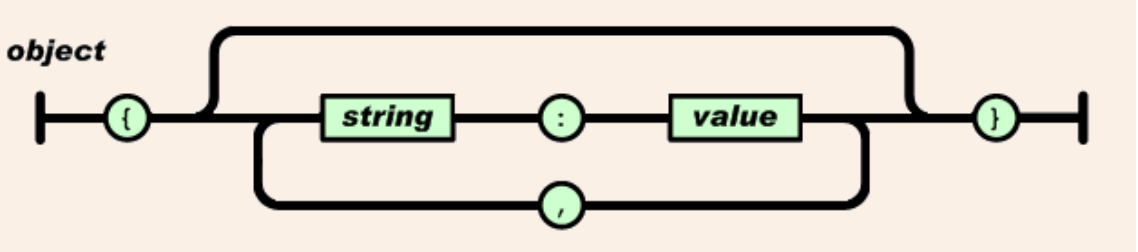


図 4-10　JSON のデータ構造[13]

XML などと同様のテキストベースのデータフォーマットである。XML の形式に比べると図 4-11 のようになる。

リスト1：XMLの例

```
<employees>
  <employee>
    <empId>000001</empId>
    <department>企画部</department>
    <name>山田 太郎</name>
  </employee>
  <employee>
    <empId>000002</empId>
    <department>営業部</department>
    <name>山田 次郎</name>
  </employee>
</employees>
```

リスト2：JSONの例

```
[
  {
    "empId": "000001",
    "department": "企画部",
    "name": "山田 太郎",
  },
  {
    "empId": "000002",
    "department": "営業部",
    "name": "山田 次郎",
  }
]
```

図 4-11　JSON と XML 形式の対比[13]

JSON は XML よりシンプルで、基本的な値などをやりとりするだけであれば、XML よりも JSON の方がずっと簡単である一方、JavaScript を使っているのであれば JSON 形式も自然な形式である。

JSON のメリット：

- 冗長な XML と比べて通信時のデータ量を削減できる。
- JavaScript との親和性の高いので、クライアント側でのデータ解析もしやすい。

以上のようなメリットに対して、今回のサーバ発信機能は提供するデータは殆ど（一部 Jpeg 画像などのバイナリーデータを除き）JSON 形式でクライアント側に転送する。

■　画像データの転送に置いての分析

画像データをどうやって JSON 形式に格納してクライアント側に転送するかをチーム内でよく議論したところ、各ピクセルの RGB 情報を JSON に格納しようとして、図 4-12 で示すように画像データを格納する。

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

図 4-12　JSON 形式の RGB 情報のサンプル

こうして問題になったのは、カメラサイズのデータ量（総計 640＊480 個ピクセル分）が多すぎで、普通のネット回線で通信速度は一段と落ちてしまうことが検証テストで分かった。

そこで通信データの量を減らす方法を検討して、Base64 という エンコード方式を採用した。Base64 では 8 ビットバイトのデータを読み込み、6 ビット単位に区切り、対応する文字を出力するという作業を行い[9]、一つのピクセルに対しての RGB 情報は 3 バイト計 24 ビットであり、Base64 によってエンコードすると、4 つの文字になる。それによって、640*480 個のピクセル持つ RGB 情報は 640*480*4 という長さのストリングに変更する。そして、クライアント側でそのストリングを受けてディコードして、ピクセルごとの RGB 情報を分析してから図を描くようになる。同じ RGB データに対して、Base64 エンコードで文字列化によって効果は表 4-4 で示す。

表 4-4 Base6 エンコードの効果

RGB カメラ画像	エンコード前	エンコード後
データ量	7～10[MB]	1.4[MB]
JSON データ作成時間	約 4[s]	0.02～0.04[s]
通信時間	約 6[s]	0.5～0.7[s]
Canvas で表示時の FPS	約 0.017（1 フレームに一分ほど）	3～5

4.4.5　　ユーザ識別の実現手法

Kinect にとって、ユーザには検出(Detected)と識別(Calibrated)という二つの状態があり、検出とは 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。一方、検出したユーザが両腕を直角に曲げるというポーズを取り続けるとキャリブレーションが成功して，識別された状態に遷移する。その状態だけではユーザの骨格情報を取得可能である。それを解消するため調べたところ、
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，前もってファイルとして保存された普通身長の方のカリブレーション情報を読み込み、新しく検出した方をカリブレーションさせるという手法を採用した[16]。

また、サーバプッシュ機能と連携する場合は、ポーズによって識別イベントが不可欠であるため、その二つのニーズをサポートするため、ソースコードには IsPoseNeeded というフラグを用意して、ユーザ識別に当たって、ポーズを要るかサーバ管理者によって指定できる。

4.5　機能詳細

本節では、Kinect センサーデータ提供機能の仕様を詳しく述べる。
Kinect センサーには複数のセンサーが搭載されている。大まかな仕様は表 4-5 の通りである。

表 4-5 Kinect センサーの仕様

RGB センサー	解像度 VGA(640×480)
距離画像センサー	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

第一イテレーションにおいて、OpenNI で取得可能なセンサーデータをベースに、点群データ（PointCloud）[4]や複数ユーザ場合の骨格、重心などの情報提供を含めて、Http サーバ以下の 14 つのタイプのデータを提供するようにした。

- JSON 形式のイメージデータを取得する
  ピクセルごとの RGB 情報を Base64 でエンコードしてクライアント側に転送する。このようなデータをクライアント側で受け取ってディコーディングした後、Canvas などのツールを用いて、イメージを再現できる。

RGB カメラ画像	Spec
データ量	1.4[MB]
JSON データ作成時間	0.02～0.04[s]
通信時間	0.5～0.7[s]
Canvas で表示時の FPS	3～5

- JPEG 形式のイメージデータを取得する
  カメラから取ったイメージを JPEG 形式でクライアントに転送する。
  本仕様では、Kinect のセンサーデータの中 RGB データを取得して、JPEG 形式としてクライアント側へ提供することが可能である。PC とスマートフォンそれぞれブラウザ上からサーバにアクセスして取得した JPEG 画像は図 4-13 で示す。実際で JPEG 画像データの取得を検証した結果は表 4-6 で示す。結論としては、Web カメラと同程度に表示可能である。

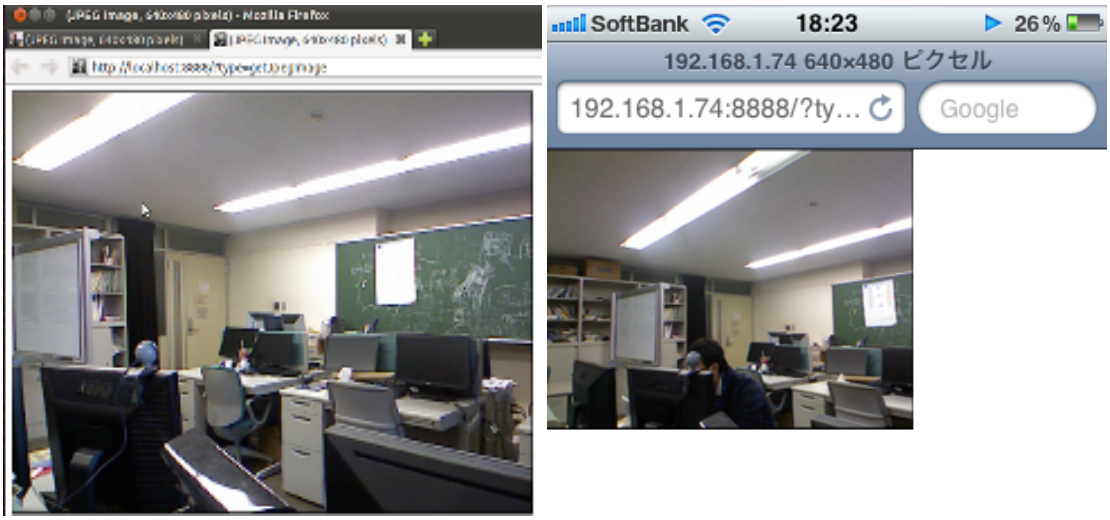


図 4-13 JPEG 形式の RGB データ

表 4-6 JPEG 画像形式で RGB データ取得状況

端末	PC Brower(Firefox)	SmartPhone Brower(Safari)
データ量	約 100[KB]	約 100[KB]
JPEG データ作成時間	約 0.02~0.03[s]	約 0.03~0.05[s]
通信時間	約 0.01~0.03[s]	0.01~0.03[s]
Kinect 設定解像度	640*480	640*480
Kinect 設定 FPS	30	30

- JPEG 形式のデプスマップを取得する
  デプスマップとは[元画像の、どの部分をどれだけ奥まったようにめせたいか]を決める画像である。図 4-14 は図 4-13 と同じ角度で生成したデプスマップを PC とスマートフォンそれぞれのブラウザで表示したものである。そこで示すように、画面手前にあるディスプレーやウェブカメラなどは明るい黄色、奥側にある壁などは暗くなっている。仕組みとしては、本来のイメージデータの RGB データをピクセルごとのデプス（深度）によって上書きする。そして、明るい黄色で塗ってある部分は 3D 表示にしたときに手前に表示される、逆に、黒に近い色で塗ってある部分ほど、3D 表示にしたとき、奥のほうに見える。

図 4-14　JPEG 形式のデプスマップ

- JSON 形式のデプスマップデータを取得する
  デプスマップで上書きされたピクセルごとの RGB 情報を Base64 でエンコードしてクライアント側に転送する、受け取ったデータはクラインと側でディコードする必要がある。
- カメラ座標系における点群の 3D 座標を取得する

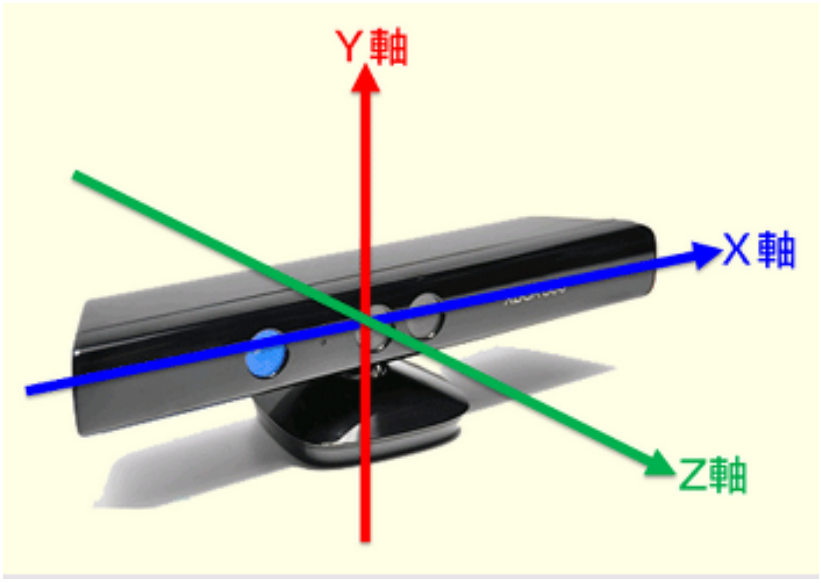


図 4-15　カメラ座標系[18]

カメラ座標系とは、図 4-15 を見て分かるように、Kinect デバイスを中心として、X／Y／Z の軸が構成されている。Kinect 前面方向に Z 軸が伸びているため、Z 軸(デプス)は 0 より増える方向にのみ伸び、マイナスにはならない点や、Y 軸も通常の WPF や Windows フォームの座標系とは反対の方向に伸びている[18]。
点群とは 3 次元座標の集合のことで, Kinect センサーの距離画像センサーによって取得できるデータである。例は図 4-16 で示す。もともとロボット分野の Robot Vision 向けのものであり, Kinect センサーの登場以前は高価な機器でしか 3 次元点群データを取得できなかった。

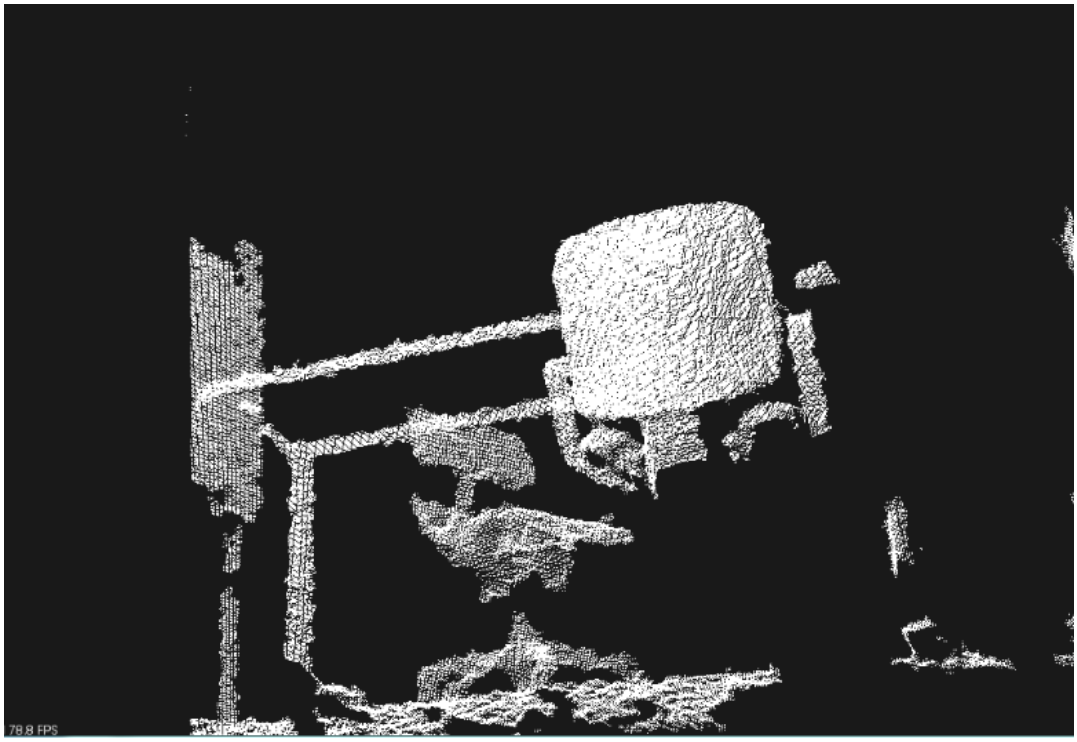


図 4-16　点群データの例

このファンクションでは、デプスデータを利用して、点群の 3D 座標だけを算出してより扱いやすい JSON 形式に変換して、クライアント側に転送する[4][19]。

{"Kineco":{"config":{"height":480,"width":640},"pointCloud":[
{"x":"-179.19","y":"234.254","z":"846.261"},
{"x":"-158.027","y":"-89.0328","z":"938.176"},
{"x":"-26.8865","y":"47.3859","z":"509.522"},
{"x":"-112.342","y":"-239.451","z":"1239.9"},
{"x":"222.934","y":"-927.718","z":"1099.87"},
{"x":"58.583","y":"-590.332","z":"1168.51"},
{"x":"-53.7729","y":"94.7718","z":"1019.04"},
{"x":"40.7659","y":"-54.5648","z":"1159.49"},
{"x":"192.965","y":"-90.1486","z":"959.938"},

図 4-17　JSON 形式の点群座標の例

- 複数ユーザの骨格スケールトンのジョイントポイントを取得する
  本仕様では、標準ポーズでカリブレーションされた人の骨格情報を取得して、JSON 形式に書きこんでクライアント側に転送する。取得可能のパーツ[1][2]は以下の表 4-7 で示す。実際クライアント側に転送したデータ例は図 4-18 で示す。

表 4-7 骨格情報を取得可能なパーツ

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


```
{"Kineco":{"config":{"height":480,"width":640},"user":[{"ejoint":[
{"coordinate":{"x":"-266.957","y":"164.254","z":"1086.01"},"part":"head"},
{"coordinate":{"x":"-251.952","y":"-225.555","z":"1146.89"},"part":"torso"},
{"coordinate":{"x":"-263.023","y":"-26.3778","z":"1117.05"},"part":"neck"},
{"coordinate":{"x":"-171.927","y":"-429.867","z":"1114.5"},"part":"right_hip"},
{"coordinate":{"x":"-91.7601","y":"-1214.6","z":"1150.31"},"part":"right_foot"},
{"coordinate":{"x":"-131.057","y":"-829.925","z":"1132.76"},"part":"right_knee"},
{"coordinate":{"x":"-157.465","y":"-34.6181","z":"1022.13"},"part":"right_shoulder"},
{"coordinate":{"x":"-154.627","y":"-262.728","z":"906.169"},"part":"right_elbow"},
{"coordinate":{"x":"-292.901","y":"-449.303","z":"733.711"},"part":"right_hand"},
{"coordinate":{"x":"-309.931","y":"-419.068","z":"1238.73"},"part":"left_hip"},
{"coordinate":{"x":"-237.312","y":"-1203.78","z":"1266.65"},"part":"left_foot"},
{"coordinate":{"x":"-272.909","y":"-819.116","z":"1252.96"},"part":"left_knee"},
{"coordinate":{"x":"-368.581","y":"-18.1375","z":"1211.97"},"part":"left_shoulder"},
{"coordinate":{"x":"-422.938","y":"-261.445","z":"1430.59"},"part":"left_elbow"},
{"coordinate":{"x":"-404.752","y":"-537.887","z":"1411.33"},"part":"left_hand"}],
"id":3219861112}],"version":0.10}}
```


図 4-18 JSON 形式の骨格データ例

- 複数ユーザの重心の座標を取得する
  検出されていたすべてのユーザの重心の 3D 座標を JSON 形式データに格納してクライアント側に転送する。

- 検出されている全てのユーザ ID を取得する
  検出されていたユーザの ID を JSON 形式データに格納してクライアント側に転送する。

- 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
  標準ポーズで識別されたユーザ、すなわち骨格情報を取れるユーザの ID を JSON 形式データに格納してクライアント側に転送する。

- 現在検出したユーザ数を取得する
  何人が検出されていたのかを JSON 形式データに格納してクライアント側に転送する。

- PSI によってカリブレーションされたユーザ数を取得する
  何人が標準ポーズをして識別されていたのかを JSON 形式データに格納してクライアント側に転送する。

- 指定した座標の距離(depth)を取得する。
  Kinect に写っている点の x, y 座標（640＊480 範囲）を引数として渡され、その点は Kinect までの深度を取る。

- Id によってユーザの骨格情報を取得
  ユーザ Id を引数として渡され、指定したユーザの骨格情報を取得して JSON 形式のデータに格納してクライアント側に転送する。

- Id によってユーザの重心の座標を取得
  ユーザ Id を引数として渡され、指定したユーザの重心の座標を取得して JSON 形式のデータに格納してクライアント側に転送する。

- Id によってユーザの深度を取得
  ユーザ Id を引数として渡され、指定したユーザの重心を取得して深度を算出してから、JSON 形式のデータに格納してクライアント側に転送する。

第二イテレーションでは、他のメンバー担当した機能と連携するため、Http サーバの RequestHanlder モジュールには表 4-8 で示すような JSON 形式の複数連携機能で合わせた 3D PointCloud データも提供できるインタフェースを設けた。

表 4-8 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　　クライアント用のライブラリ

本節では、第二イテレーションにおいて、筆者が担当したクライアント側用のコーディング支援用ライブラリについて述べる。

4.6.1　　開発の目的

第三章で述べたように、ブラウザから HttpRequest を出してデータを取得するというシナリオに加えて、HttpRequest を出してデータを取得してくれるライブラリを提供してほしいという委託元からの要望に応じることは開発の目的である。利用シーンとしては図 4-19 で示す。

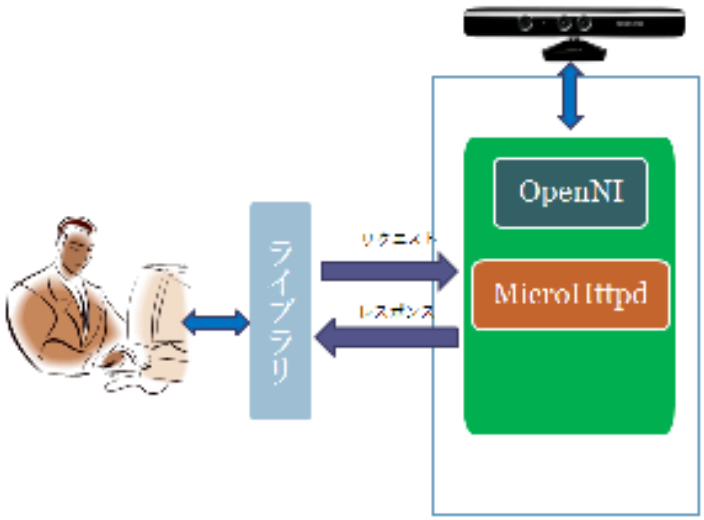


図 4-19　ライブラリからの利用シーン

4.6.2　　ライブラリの機能要求

プログラマの立場で考えたところ、以下のような機能要求を洗い出した。

- ユーザはライブラリをソースコードにインクルードしてサーバのアドレスやポートを引数としてライブラリクラスを初期化する。
- クライアントライブラリ仕様書に書かれている関数を呼ぶことで、サーバにアクセスことができ、そして関数の戻り値としてデータを取得できる。
- 多様性（現時点は C++,Java 用のライブラリを作ってきた）。

4.6.3 ライブラリの構成

本ライブラリは主に 2 つの部分によって構成される。一つの部分は LibCURL である。LibCURL とはフリーで使いやすいクライアントサイド URL 転送ライブラリであり、FTP, Http, Https など様々なネット上のプロトコルをサポートしている[14]。そして、移植性が高いという特徴がもって、複数のプラットフォームで同じようにビルド、稼働させることができる。今回のクライアントライブラリにおいては、Http 発信サーバとやり取りために採用されて、Java や C++両方のクライアントからの利用をサポートする[15]。

そして、もう一つは、外部環境とやり取りをするためのインタフェース。このインタフェースで定義されている関数名はブラウザ上から Http リクエストを出す時追加する引数の値と同じである。つまり、例えばブラウザを経由して全てのユーザの骨格情報を取得するには http://hogeIpAddress:port?type=*getSkeletonJointPosition*.を HttpRequest の URL として入力することが必要であることに対して、インタフェースのほうでは *getSkeletonJointPositon* という関数を呼び出せば同じ骨格情報を取得可能である。インタフェースの詳細は図 4-20 で示す。

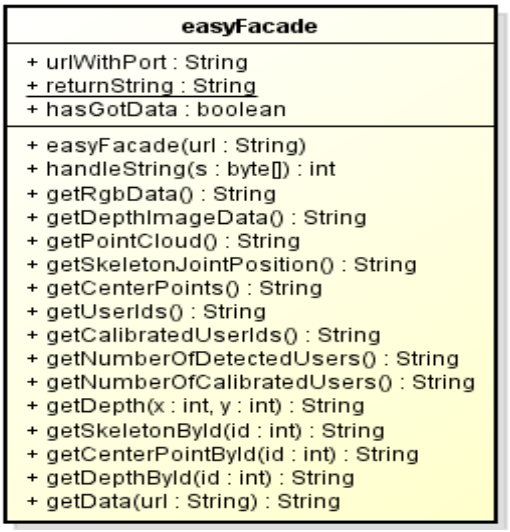


図 4-20 インタフェースの詳細

外部環境では、必要に応じて、インタフェースに書かれている仕様を呼び出して、ライブラリ内部では、呼び出された API によって、適切な URL を組み立てて LibCURL を通じて Http 発信サーバへ Http リクエストを出す。そして、Http 通信を経由して、サーバから送信されてきた JSON 形式のテキストデータをもう一回 LibCURL を通じてインタフェースに渡す、最後は API 関数の戻り値として外部環境へ戻す。こうして、ユーザは面倒な HttpRequest と HttpResponse を管理する手間を省いて、より手軽く Http 発信サーバから Kinect のデータを取得するようになる。クライアントライブラリを利用する場合のデータ転送流れを図 4-21 に示す。

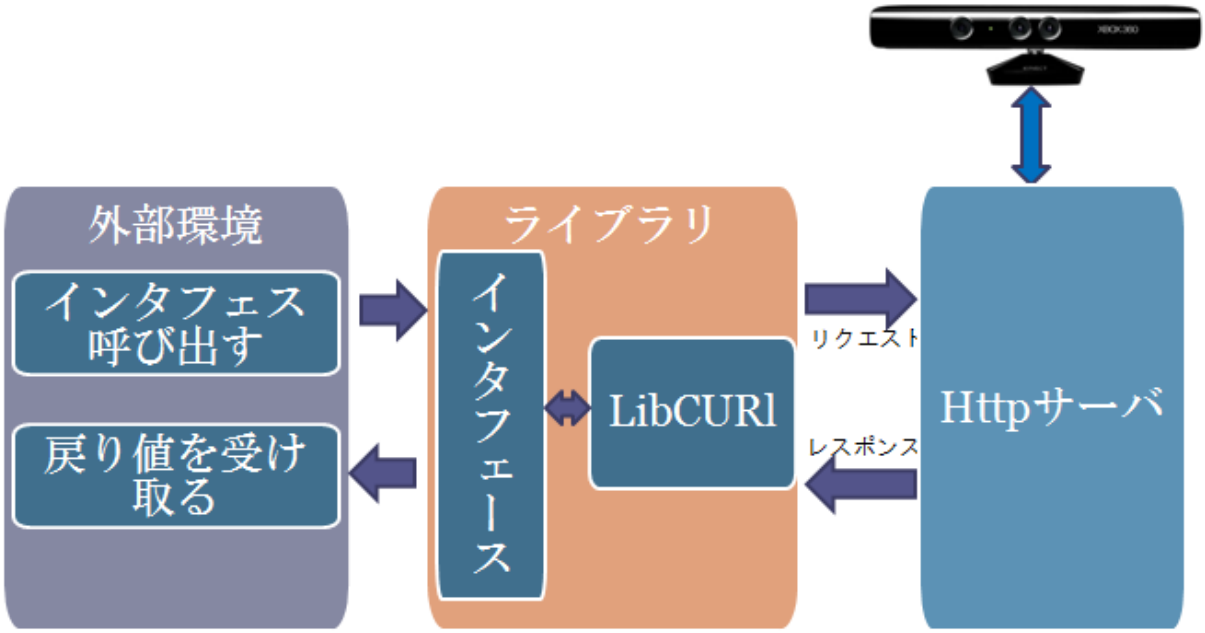


図 4-21　ライブラリ利用のデータ流れ

4.6.4　　スクリプト言語対応の HTTP クライアントの考察

今回のコーディング支援用のクライアント側用のライブラリは Java や C++それぞれ対応のバージョンを開発したが、技術の進化に連れて、スクリプト言語の利用シナリオがどんどん広がっている。この 10 年間スクリプト言語に着目していくと、スクリプト言語の構造が明らかに変化してきたことに気付いているはず。Python、Ruby、Perl、その他多くのスクリプト言語には、C や C++ に見られるようなソケット層が含まれているだけでなく、アプリケーション層のプロトコルの API も含まれている[17]。これらのスクリプト言語には上位レベルの機能が組み込まれているため、例えば HTTP サーバやクライアントを簡単に作成することができると考えてスクリプト言語対応のライブラリの作成を筆者が考察した。具体的に調べたところ、Python ベースの HTTP クライアントも出た。詰まり、LibCURL ライブラリによって C や C++ などの言語と似た機能をスクリプト言語にも追加することが可能である、機能追加の方法はどの言語の場合も共通である。

筆者が LibCURL では HTTP のクライアントの構築がいかに容易であるかを今回のクライアントライブラリの開発で確実に実感して、先導 IT の後輩は次回の研究開発で、アプリケーション層のプロトコルを必要とする Web ブラウザやスパイダー、その他のアプリケーションを作成する際には、LibCURL を試してみてください。LibCURL によって確実に開発時間を短縮することができ、コーディングを楽しめると思う。

第5章　　マネージメントの工夫と施策

本章では、本プロジェクト全体の開発を通して、進捗やタスクの管理、そしてリスク分析などについて述べる。

5.1　　工夫点

本節では本プロジェクトの工夫点を詳しく述べる。

5.1.1　　プロジェクトの一元化管理

本プロジェクトのマネージメントについては主に Redmine[7]の利用によって、一元化管理を行った。具体的に以下のような面でメンバー同士の情報共有を出来ていた。

- スケジュール共有
  Redmine の カレンダー機能をして、定例ミーティングの日付やタスク（チケット）の進行情報などをメンバー間で以下のようなカレンダーで共有して、開発スケジュールの現状を一目で把握することが出来る。

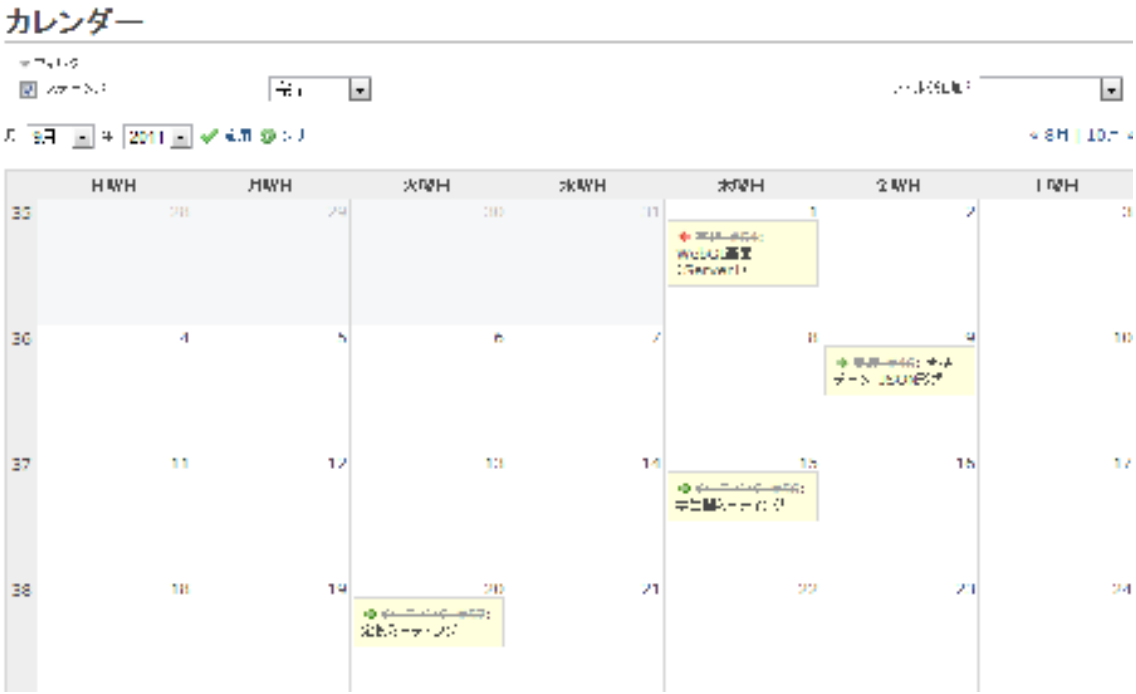


図 5-1　カレンダーに記載するチケット

- タスク管理
  メンバーごと担当している部分の仕事を定量に計るため、そして、自分の作業ログを残ってタスクごとの進捗状況を一目瞭然するためにも以下のようなチケット管理を行った。それによって毎回チームミーティングでメンバーそれぞれ担当している作業をチェックしながら、話を進めていくという形にもなり得る。

図 5-2　一覧のタスクリスト

そして、タスクのディスクリプションや目標そして達成度を記述した上に、そのタスクに関わるソースコードも一緒にリンク付けされている。例えば、一覧リストで出てきた KinectHttpServer.cpp の階層化作業というチケットを開いてみると、図 5-3 で示すように、チケット明細が記述されている。

図 5-3　チケット詳細

- ソースコードのバージョン管理
  今回のプロジェクトの開発に関わるすべてのソースコードは Git というバージョンコントロールツールによって管理されているが、Git プラグインをインストールされたため、ソースコード全体の更新情報を以下の図で示すように Redmine 上で把握できる。

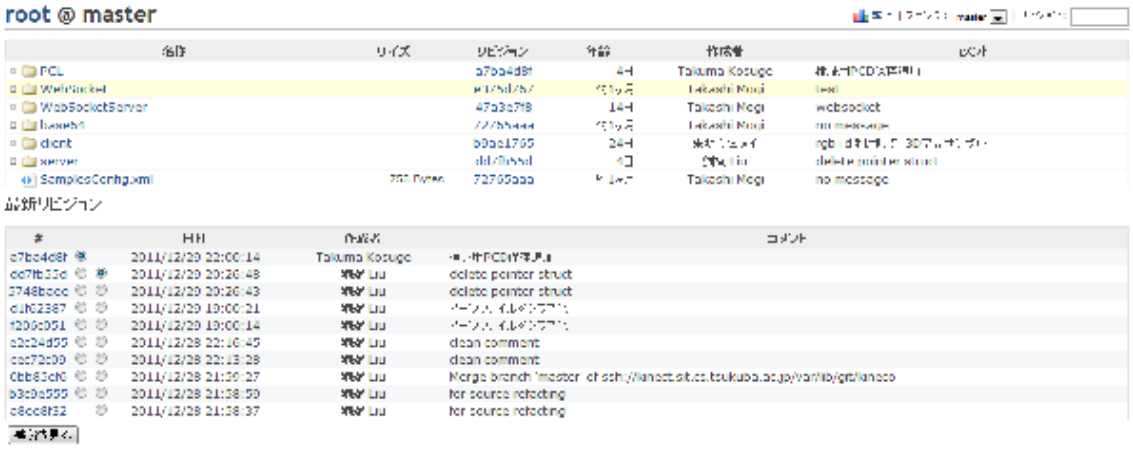


図 5-4　ソースコードのバージョン管理一覧

そして毎回の変更において、どのような変更を行ったのかをチェックしようとすると、差分を見るボタンを押して、以下の詳細を表示する。こうしてソースコードのリファクタリングやレビューなどの時に役に立った。

図 5-5　ソースコードの差分比較例

5.2　　振り返りと分析

この節では、本プロジェクトの進行に関して改善や反省すべきところ、そして今後の方向性について述べる。

5.2.1　　要件定義における分析

- 仕様策定には不厳密なところがある
  中間発表会前に、JPEG など形式のカメラ画像をクライアント側に転送する機能にたいして、具体的な機能性（解像度、fps）ははっきりと決めていなかった、そして、RGB 画像のビットマップを Base64 でエンコードして、クライアント側でディコーディング用のスクリプトや仕様など決めるべきことが考えてなかったため、中間報告会でそういったところが指摘された。

- 仕様を定めること
  システム開発工程の前半で、Kinect データの取得、そしてジェスチャ識別、また様々なクライアントとの連携など複数の技術調査が行われた。Kinect から取得可能なデータを用いて、どのような機能そしてアプリを提供できるかを何度も議論したところ。WebAPI の仕様しか定めなかった。こうして細かいところに時間を費やしてしまって、結局、中間報告段階で、システムの全体像を掴まず、単なる一個一個のファンクションポイントしか見えて来なかった。

5.2.2　　プロジェクト マネージメントにおける分析

- コアタイムの確保について
  メンバーの都合によって、全員揃って作業する時間を確保するには多少難しいところがあり、特に第一イテレーション段階で、仕様などはっきり定めていなかったから、作業の分割もやりにくくて、自分は何をどこまでやり遂げるか分からなくて、あんまり良いチームワークを出来てなかった。中間発表を境に、メンバーそれぞれ担当分を決めてから、必要である分のコミュニケーションを取って、Redmine などのグループウェアを活用しながら、担当分の間の連携を実現した。コアタイムの確保より、メンバーそれぞれ担当する分の作業割をはっきり決めるのはプロジェクトマネージメントの前提であると感じた。

- 委託元教員とのコミュニケーションについて
  開発工程前半で、委託元教員への進捗報告とのコミュニケーションが十分ではなかったため、仕様における細かいところをちゃんと決めていなかった。
  その理由としては、ミーティングの回数が少ないからではなく、ミーティング前に
  先生に何を確認してほしいか一回チーム内で話合ってないため、ミーティングのすすめ方には問題があった。後半では、だいぶ改善した。毎回のミーティングでメンバー一人ずつ先生に報告して、確認してほしいところをヒアリングしていくという形で会議が行われた。

5.2.3 センサーデータ提供機能の開発における分析

- データ発信部分のモジュール化について
  第一段階で基本仕様を満たすと目標して、実装を行った。当時データタイプごとの仕様検証作業はメンバーに割り当てて行われたが、各タイプのデータの取得や転送にあたって、基幹クラスの設計をやらなくて、各自で独立に作業をした。この結果、第二イテレーション段階で筆者はWebAPIに書かれている全ての仕様を担当するようになってから、第一イテレーションで書いたソースコードを纏める作業は非常にやりづらいと感じた。単純に纏めていくと、共通の分のソースコードを何度も繰り返したところ、可読性や拡張性はともかく、一つのC++ファイルに2000行以上のソースコードを書くようになった。そこで、基幹クラスの設計から初め、どうやって他のメンバー担当する機能と連携用のインタフェースを用意したり、メーンプロセスにコンテキストを共有したりするかをミーティングで確認して、データ発信機能全般のソースコードをリファクタリングして整理した。今は振り返ってみると、第一イテレーション段階でちゃんと基幹クラスの設計を先に行なって、詰まり、コーディングのルールや手順を全員に守って貰えば第二段階の機能連携作業の手間を省けることに繋がると思う。

- クライアント用のライブラリの使い方について
  今回提供するJavaやC++向け二つバージョンのライブラリは殆どのデータタイプをと取ってくれるが、JPEG形式のRGB画像といったバイナリーデータを関数の戻り値として戻ってくるのは難しいので、どうやってブラウザからと同じようにバイナリーデータを取得するかを考えて、一対のOutputStreamとInputStreamでバイナリーデータの送受信手法を調べた。結果としては、クライアント側の設定が複雑になる一方で、複数のクライアントからの要求に応じて、複数のOutputStreamを開いてバッファなどの関係でデータの順序が正しくない場合もあり、こうしてあえて不便をユーザに与えてしまう可能性があると考え、バイナリーデータの取得はブラウザからと勧めて、ライブラリからバイナリーデータの取得に関する仕様を除いた。

5.2.4 今後の課題

今回の開発において、チーム内のミーティングで出てきて時間的に納品できるか、そして､技術的に実現できるかを心配して実装に至ってない機能要件がいくつかあった｡例えば、現在のデータ発信機能には満遍なく全てのタイプのデータを常に行進しているという仕組みがあり、その代わり、若しユーザ指定したタイプ（骨格、RGB、深度など）のデータしか更新しないにすれば、サーバに掛ける負担も軽減できるし、転送したくない情報（プライバシー情報が映った画像など）が漏れるリスクも避けるというメリットをもたらすはずだが、実現には、GUIの設定画面を作れば一番使いやすいが、スケジュール的に厳しいと判断した。そのようなユーザにとって喜ぶしかし今回実装しなかった機能や提案など資料として整理する必要がある。そして、システムのリリースにあたり、導入マニュアルと運用マニュアルを細かく書く必要がある。

第6章　　結論

本プロジェクトは、委託元教員の要求を受け、「Kinect サーバおよびサンプルクライアントの開発を行った。本システムの開発要件を決めるにあたり、ミーティングで基本の仕様、そして拡張機能などを話あって定めてシステムを提案した。そして、実際の開発では反復型開発プロセスを採用し、開発工程を 2 段階に分割した。第一段階では、技術調査や仕様を定める、そして基本の仕様を満たすことを目標として実装が行われた。第二段階では、第一段階で検討できていなかった仕様の不足を踏まえて、より使いやすく、そしてより豊富な機能を提供するため、複数連携機能、サーバプッシュ機能をプロトタイプから実用の機能まで実装させて、サーバ発信機能と共に連携して、一つの Kinect サーバシステムの開発を完了した。

筆者は「Kinect センサーデータ提供機能およびクライアントライブラリ」という部分の設計や開発に置いて、技術の調査から、プロトタイプの検証、そして機能性を向上ためのクラス設計などの工程を経験した。プロジェクトマネージメントに当たって、修士一年時に受けた PBL 授業の知識を生かして、様々な管理ツールを使いながら、失敗した点を繰り返さないようにミーティングの進め方やコアタイムを確保する方法などを少しずつ改善してチーム作業の効率性やモチベーションを向上するには何が必要なのかを自ら考えて、プロジェクトマネージメントについて多くのことを学んだ。

技術面については、今回のプロジェクトの実装はほぼ LINUX 下で C++言語を使って行った。実装段階では、特に C++に関する技術的困難に関してメンバー相互の技術的支援を通じて解決できたことが多く、チームによるプロジェクト推進の重要性を認識した。

謝辞

本プロジェクトを行うにあたり、委託元教員である高橋伸先生からは大変貴重な指摘やアドバイスをいただいて非常に感謝いたします。また、同コースの中沢先生から、報告書の構成や書き方など丁寧に細かくチェックしていただき、深く感謝いたします。そして最後は、研究生時代から指導してくださった私の指導教員である田中二郎先生には、本当にお世話になりました、深く感謝いたします。また、チームメンバーから多くの助力と意見をいただき、特にリーダである茂木昴士さんから技術面において全面的にメンバー全員をサポートしていただいて、誠に有難うございます。

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] 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，石塚勝敏（編），（株）カットシステム，東京，2011.

[3] 高橋麻奈　やさしい C++、（株）ソフトバンククリエイティブ

[4] Kinect OpenNI で三次元ポイントクラウド表示
http://tclip.blog38.fc2.com/blog-entry-106.html

[5] Introducing OpenNI
http://www.openni.org/

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

[7] Redmine.
http://redmine.jp/

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

[9] RFC 3548 - The Base16, Base32, and Base64 Data Encodings
http://tools.ietf.org/html/rfc3548

[10] Node.js と WebSocket の利用シーン
http://bizria.jp/technical/nodejs-webssocket.html

[11] WebGL
http://ja.wikipedia.org/wiki/WebGL

[12] The WebSocket API
http://dev.w3.org/html5/websockets/

[13] JSON の紹介
http://www.json.org/json-ja.html

[14] libcURL -the multiprotocol file transfer library
http://cURL.haxx.se/libcURL/

[15] How to use java cURL
http://chimpler.blogspot.com/2009/03/logging-to-ebay-with-java-cURL.html

[16] Skeleton tracking without pose
http://groups.google.com/group/openni-dev

[17] C と Python で libcurl を使う
http://www.ibm.com/developerworks/jp/opensource/library/os-curl/

[18] Kinect 開発入門
http://www.atmarkit.co.jp/fdotnet/kinectsdkbeta/index/index.html

[19] Depth in Meters
http://groups.google.com/group/openni-dev/

[20] Git
http://git-scm.com/

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
- 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 1
2.各プロセスの説明 2
2.1. タスクを新しいチケットとして登録する 2
2.2. 作業時間を入力する 6
2.3. 担当したチケットのステータスを終了にする 7
3.チケットと Git との関連付け 8

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．

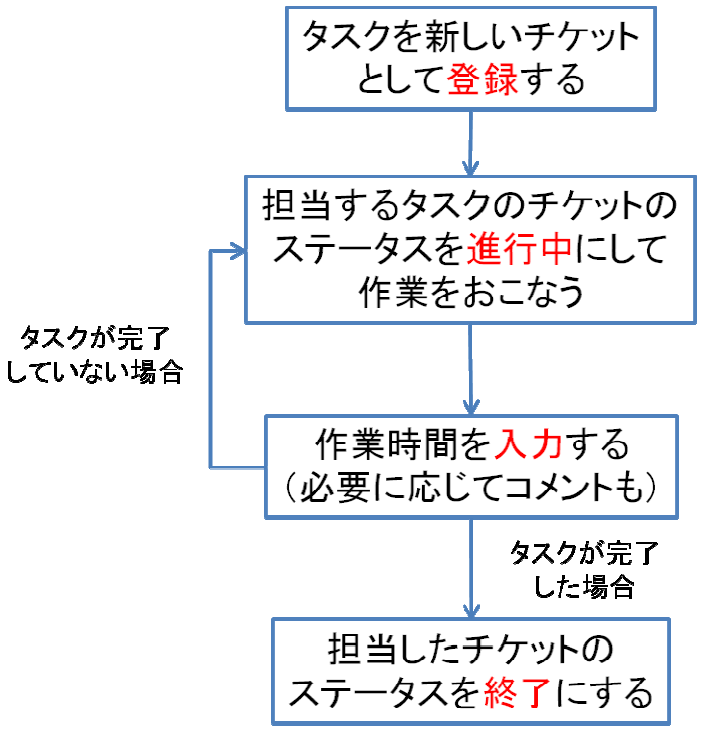


図 1. チケットの運用プロセス

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 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 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．

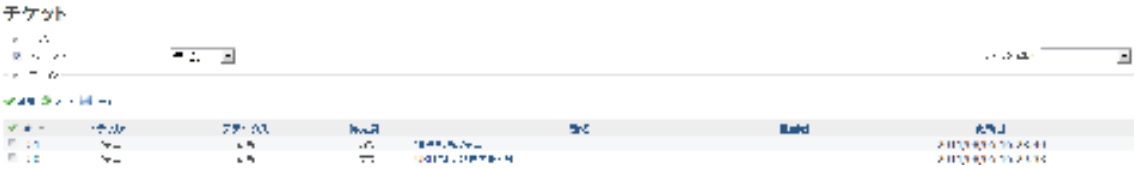


図 3. チケット確認画面

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 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

Ver4.0

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
1.1. 機能概要 1
1.2. 機能説明 1
2.Web API インタフェース仕様　3
2.1. getRgbData 3
2.2. getJpegImage 5
2.3. getDepthImageData 6
2.4. getDepthImage 8
2.5. getPointCloud 9
2.6. getSkeletonJointPosition 11
2.7. getCenterPoints 14
2.8. getUserIds 16
2.9. getCalibratedUserIds 18
2.10. getNumberOfDetectedUsers 20
2.11. getNumberOfCalibratedUsers 21
2.12. getDepth 22
2.13. getSkeletonById 24
2.14. getCenterPointById 26
2.15. getDepthById 28
2.16. getMultiPointCloud 30
2.17. Status Error Code 32

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。

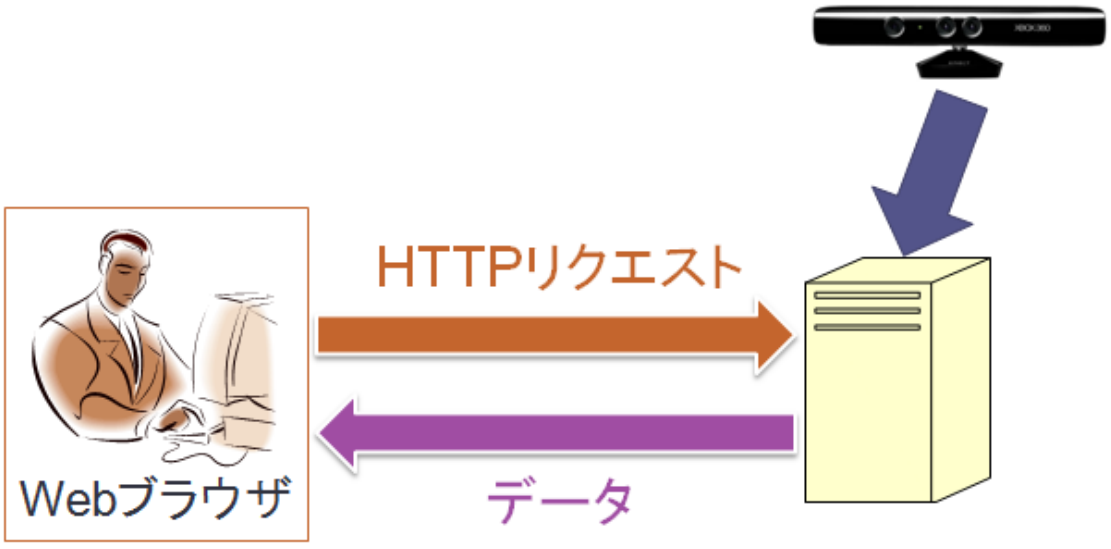
1.2. 機能説明

（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 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】

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

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 -320 < x < 320, -240 <y<240 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

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
1.1　目的 1
1.2　方針 1
1.3　記載範囲 1
1.4　参照ドキュメント 1
1.5　定義（用語、略語） 1
2　棒人間 2
2.1　概説 2
2.2　ユースケース図 2
2.3　ユースケース記述 3
3　3D 空間 8
3.1　概説 8
3.2　ユースケース記述 9
4　宇宙飛行 11
4.1　概説 11
4.2　ユースケース図 11
4.3　ユースケース記述 12
5　ボックスのトランスポート 16
5.1　概説 16
5.2　ユースケース図 17
5.3　ユースケース記述 17

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

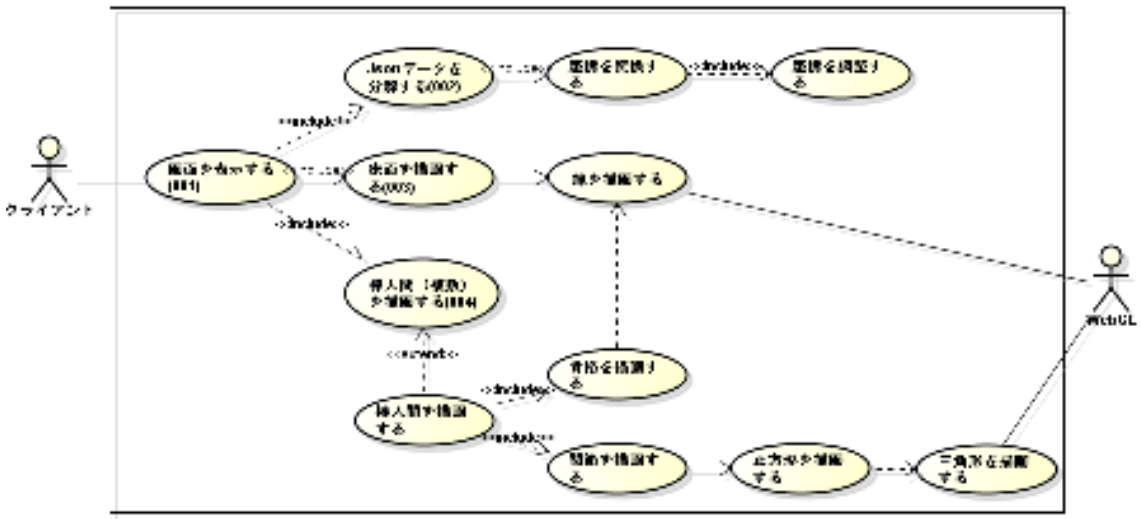
2　棒人間

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 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	

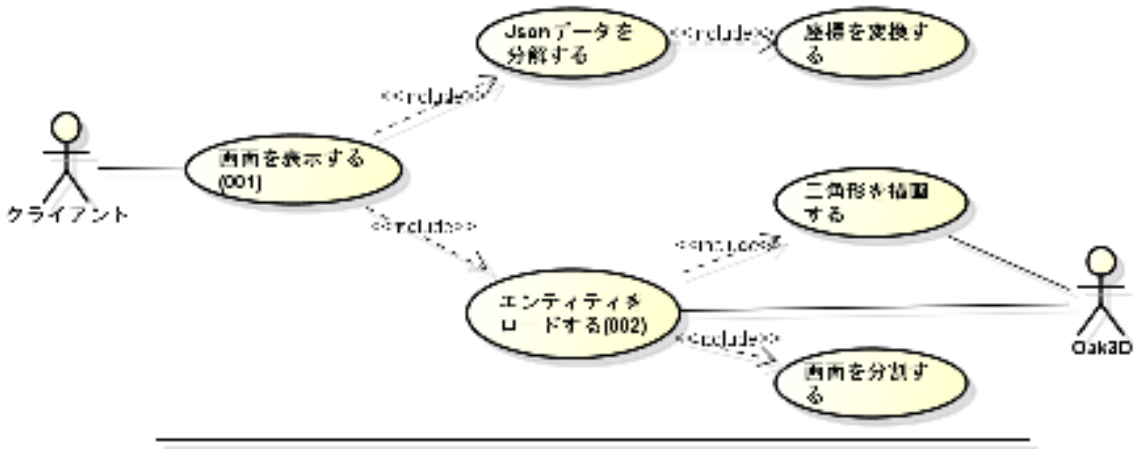
3　3D 空間

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 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	

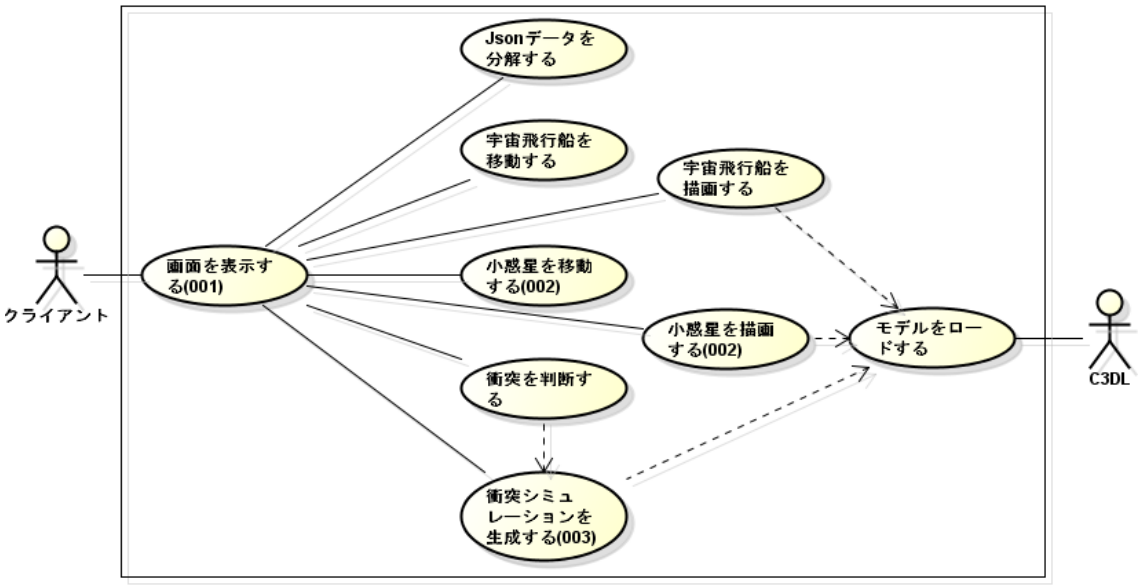
4　宇宙飛行

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 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	

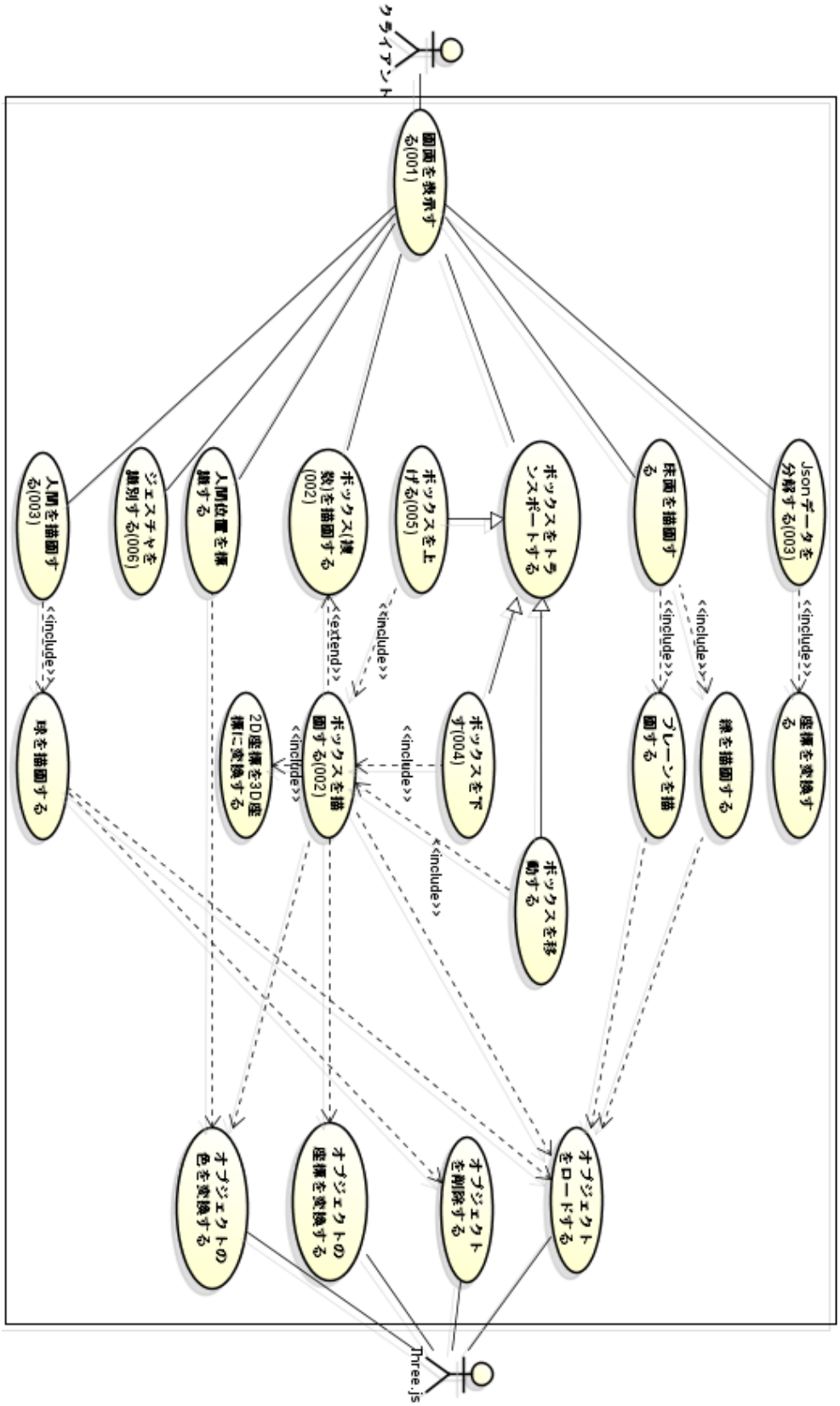
5　ボックスのトランスポート

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

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 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
1.1　目的 1
1.2　方針 1
1.3　記載範囲 1
1.4　参照ドキュメント 1
1.5　定義（用語、略語） 1
2　Http サーバ 2
2.1　シーケンス図 2
2.2　Http サーバクラス図（一部） 3
2.3　クラス図詳細 4

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

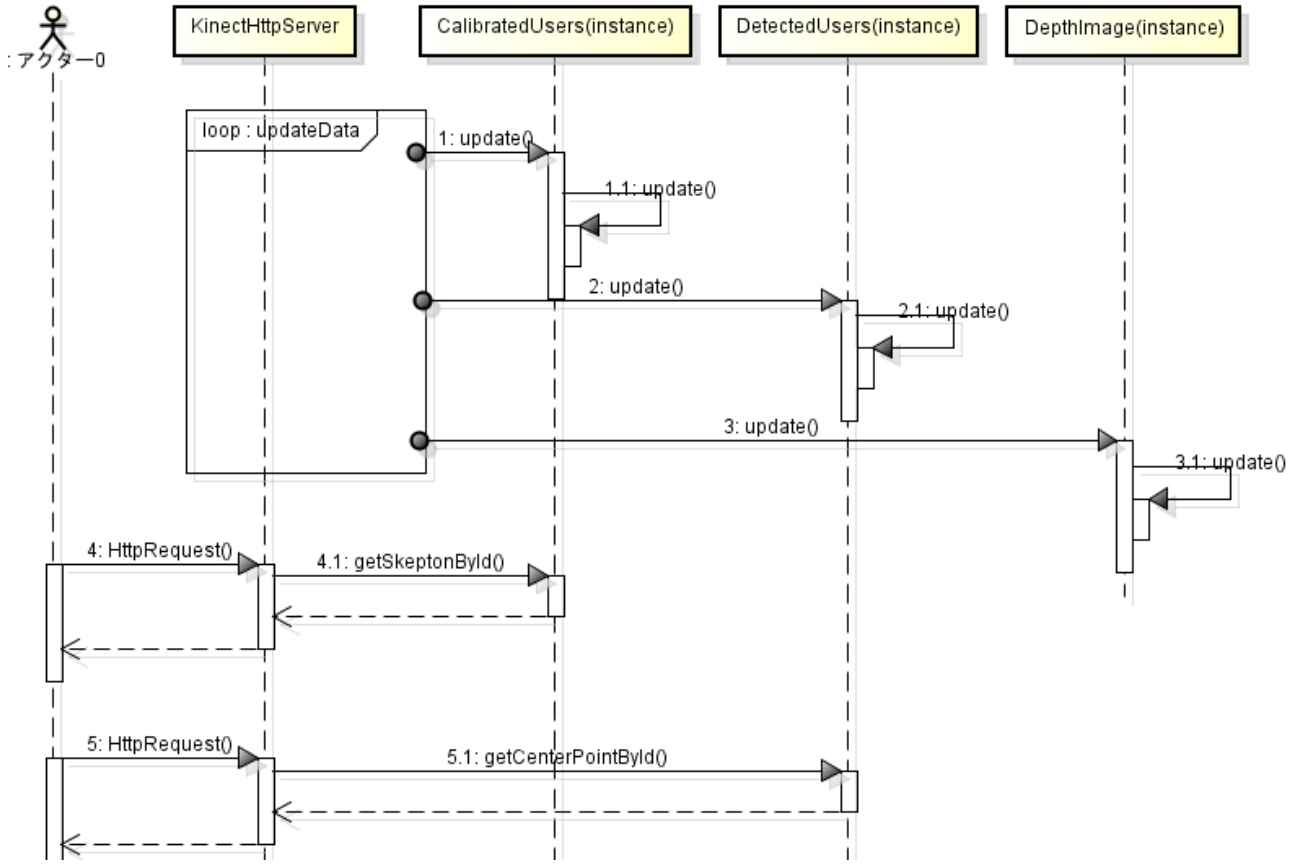


図 2-1 データ取得までのシーケンス図

2.2　Http サーバクラス図（一部）

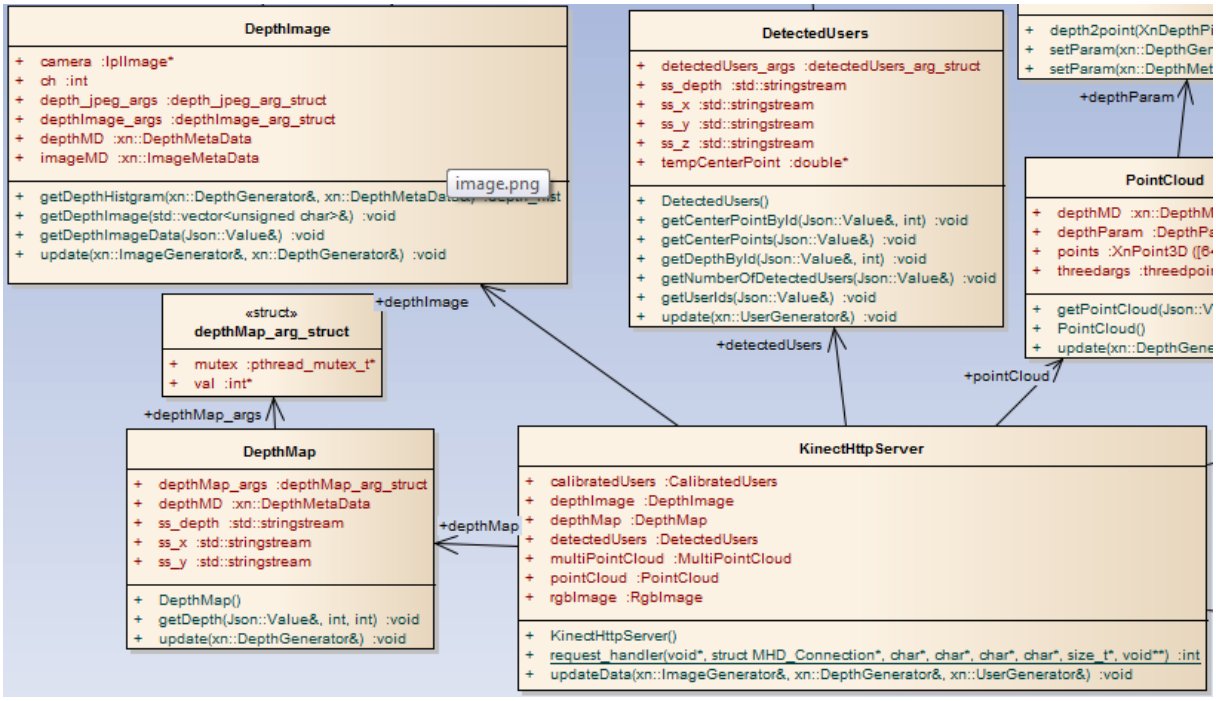


図 2-2　クラス間の関連つけ

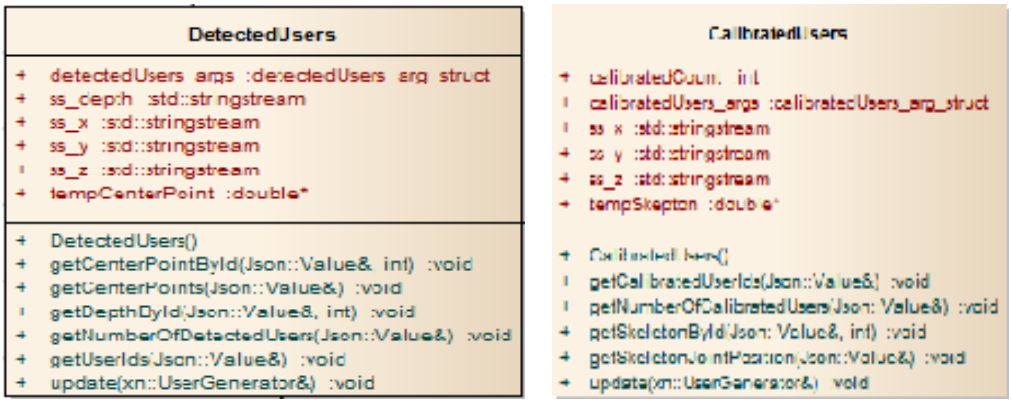


図 2-3　検出・識別ユーザクラス図の相違

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。 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

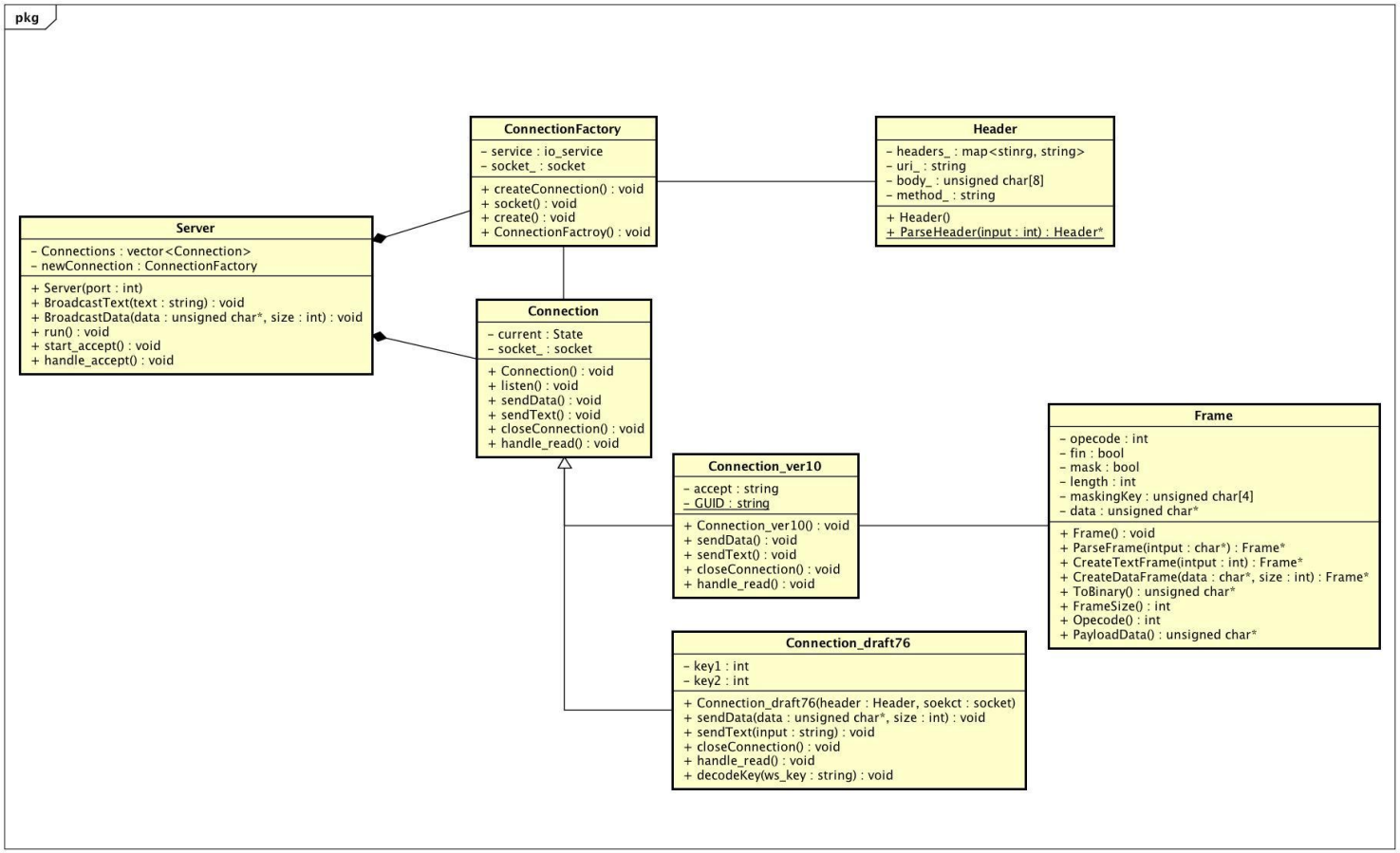
目次

1.はじめに 1
2.WebSocket サーバ 1
3.Kinect データ取得部分 2

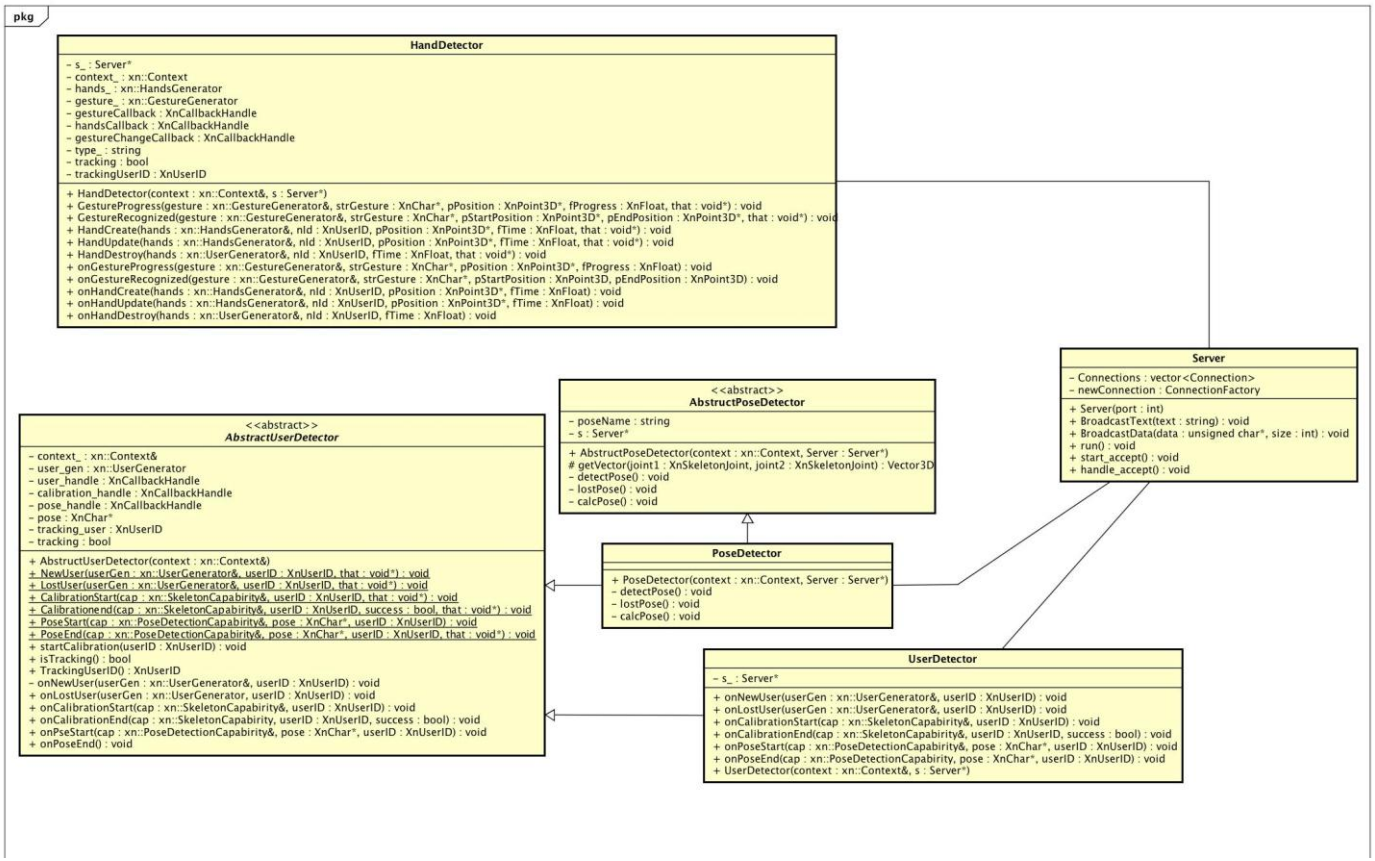
1. はじめに

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

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
1.1 目的 1
1.2 方針 1
1.3 記載範囲 1
1.4 参照ドキュメント 1
1.5 定義（用語、略語） 1
2 棒人間 2
2.1 シーケンス図 2
2.2 クラス図 3
2.3 クラス詳細 4
3 3D 空間 9
3.1 シーケンス図 9
3.2 クラス詳細 10
4 宇宙飛行 11
4.1 シーケンス図 11
4.2 クラス詳細 12
5 ボックスのトランスポート 14
5.1 シーケンス図 14
5.2 クラス図 15
5.3 クラス詳細 16

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

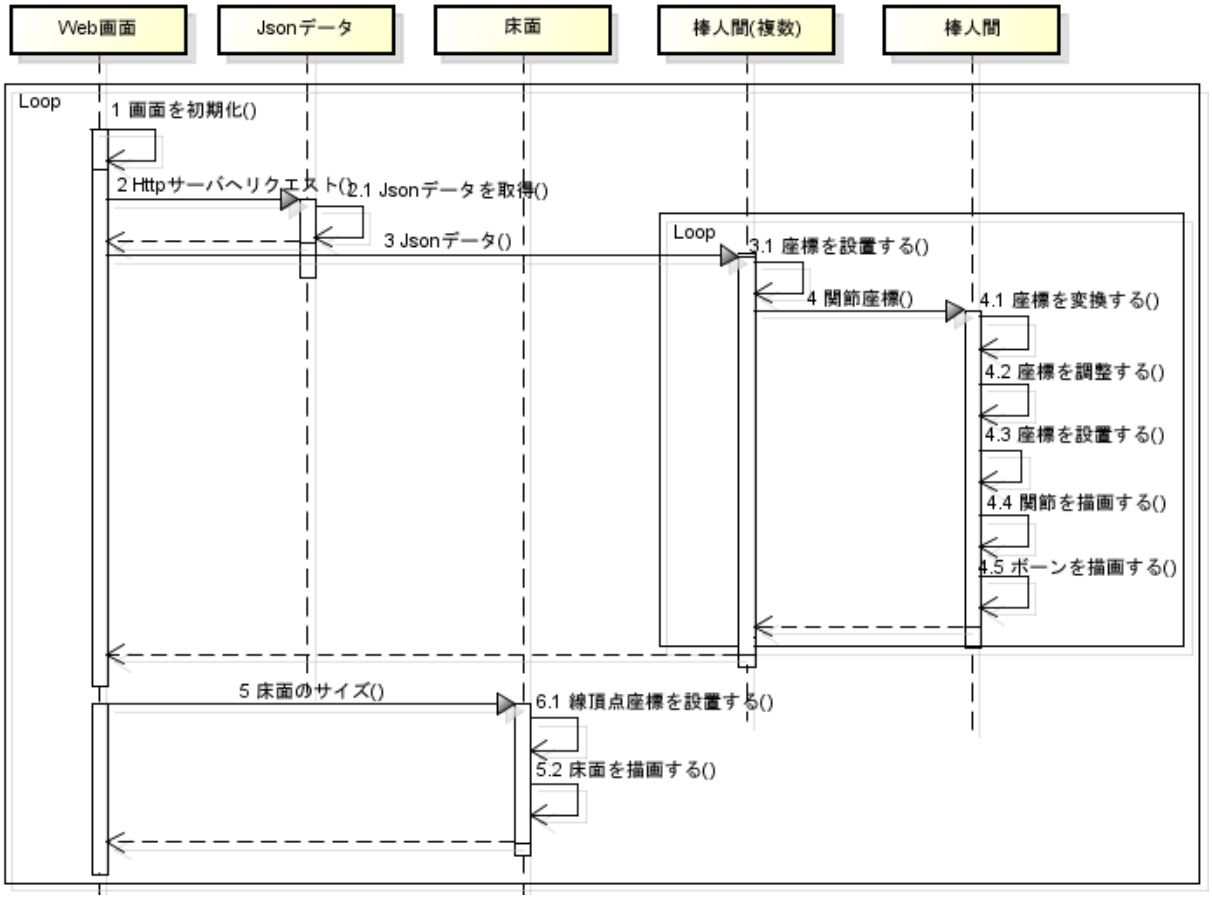
ID	文書名	文書番号	発行年月	備考
	要件定義書			
	基本設計書			

1.5　定義（用語、略語）

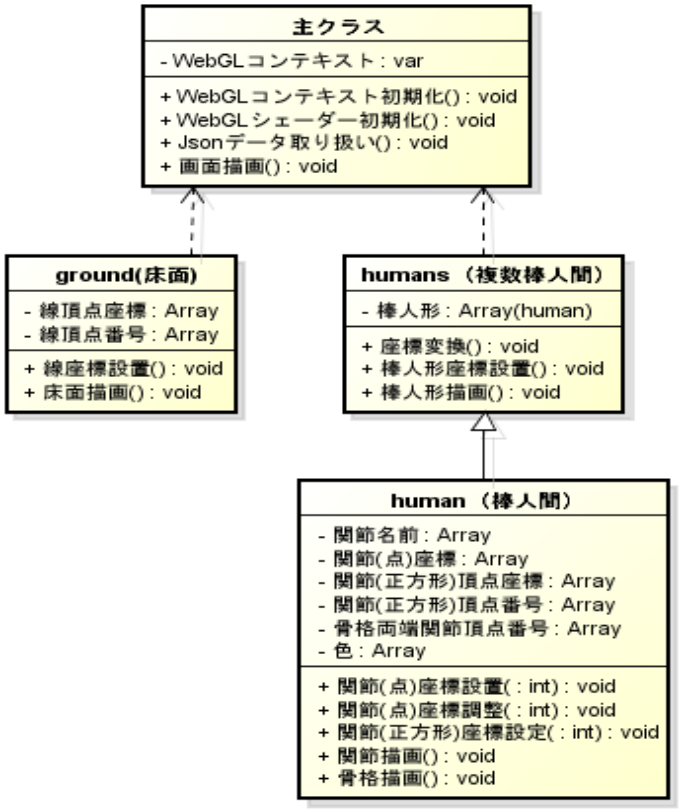
ID	用語・略号	正式表記	意味

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

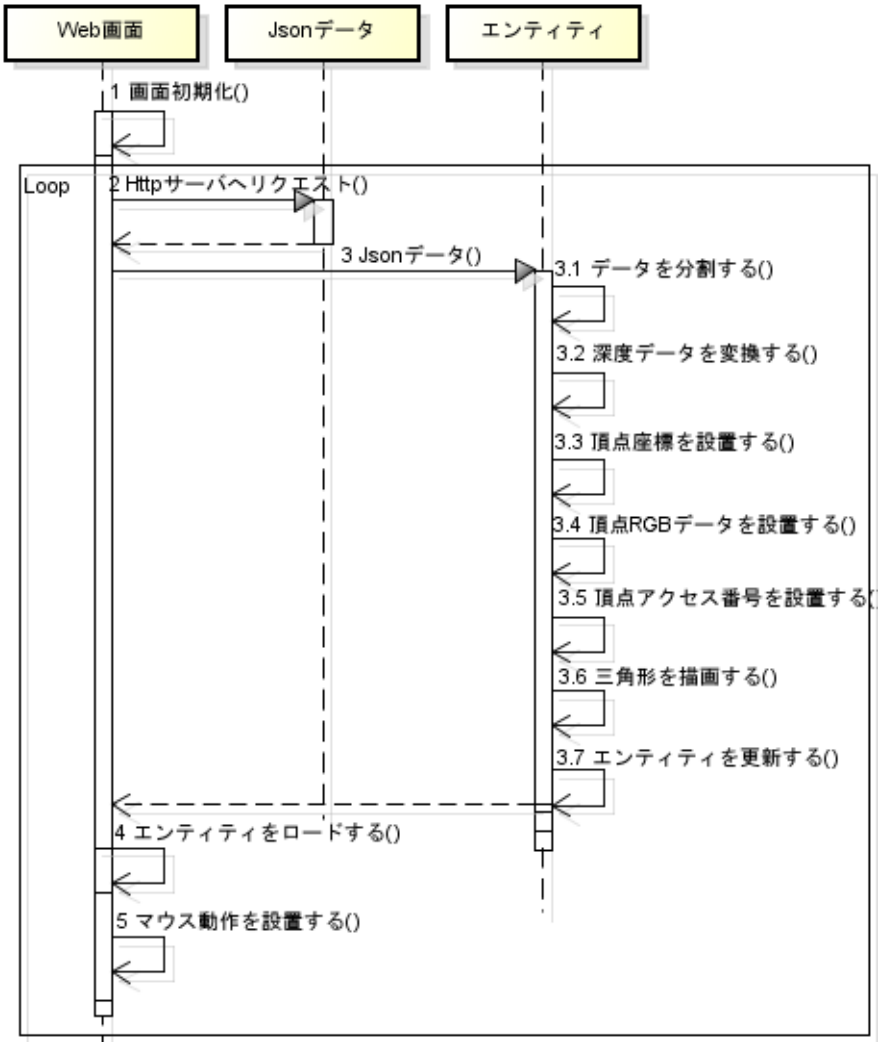
クラス名	humans			
親クラス	なし			
クラス概要				
複数棒人間を描画する				
属性				
可視性	属性・インスタンス名	型	初期値	説明、その他
private	users			userid 配列
private	human			human クラスオブジェクトの配列
メソッド				
可視性	メソッド名	型	引数	説明、その他
public	findUser		userid	userid を存在したかを判断する
public	loadJson		jsonData	関節座標を変換する
public	drawHumans		gl	複数棒人間を描画する

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 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

3　3D 空間

3.1　シーケンス図

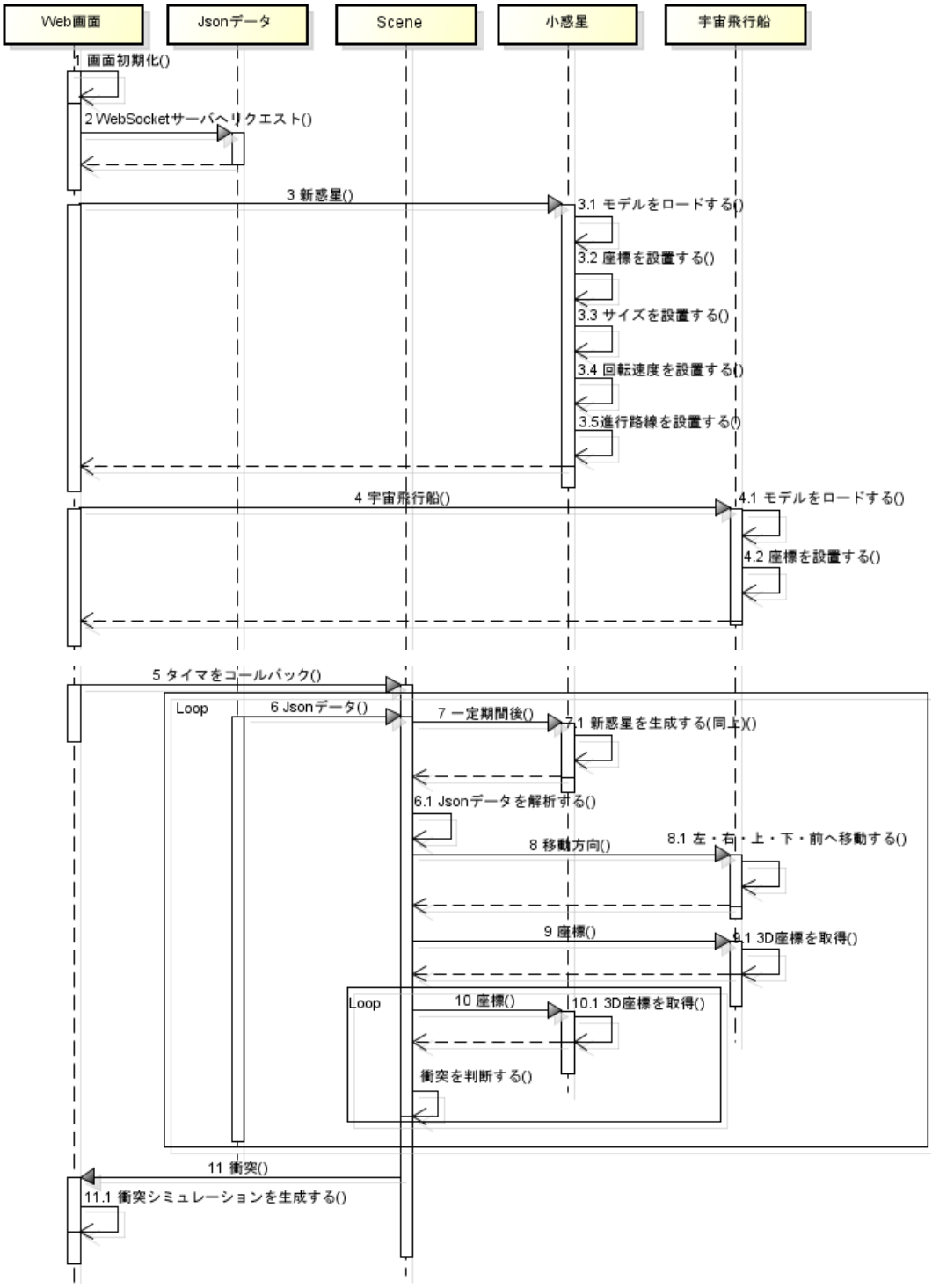


3.2　クラス詳細

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	

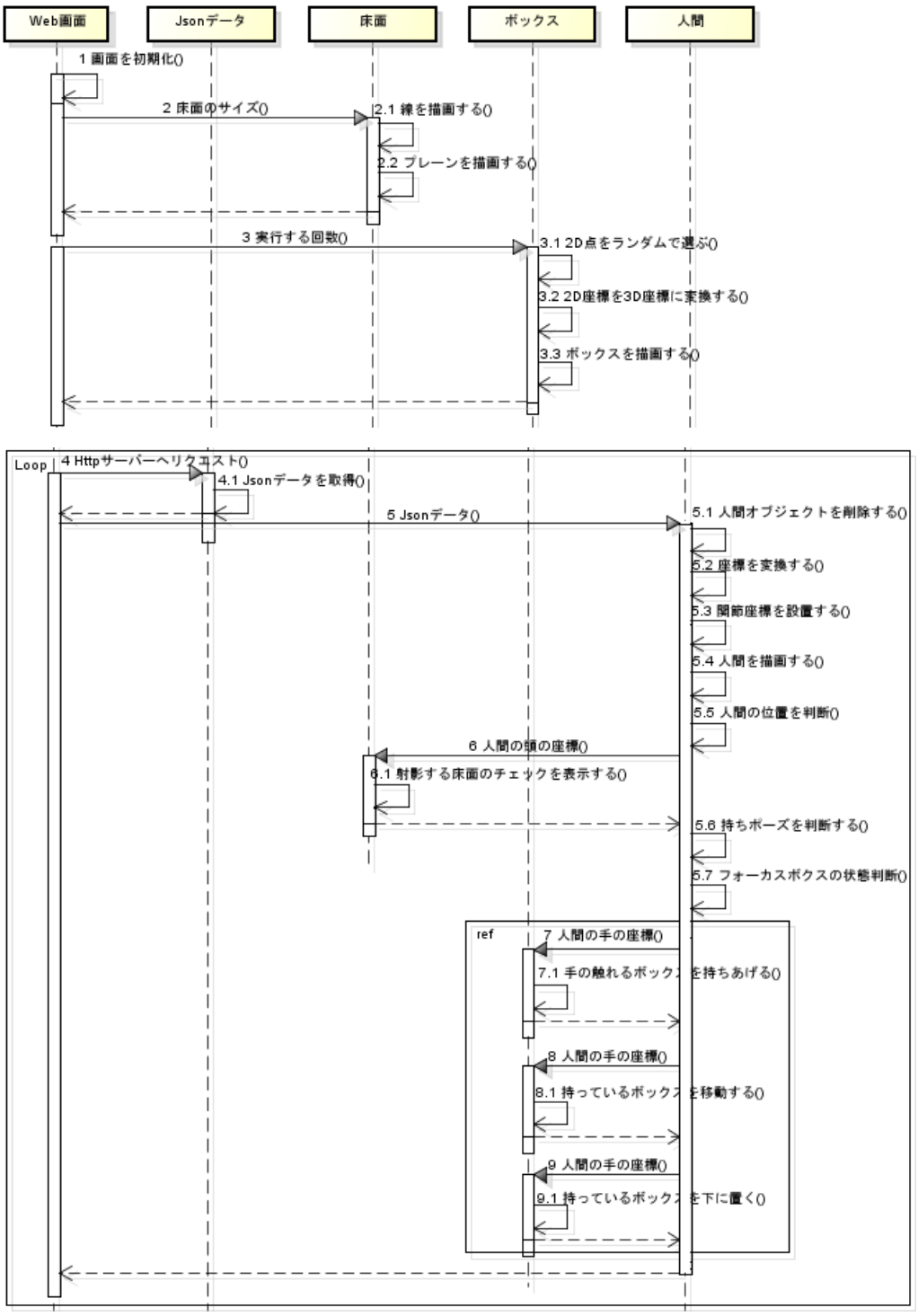
属性				
可視性	属性・インスタンス名	型	初期値	説明、その他
public	screen			シーン
public	planets			小惑星配列
public	plane			宇宙飛行船
public	simulation			シミュレーションオブジェクト

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 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

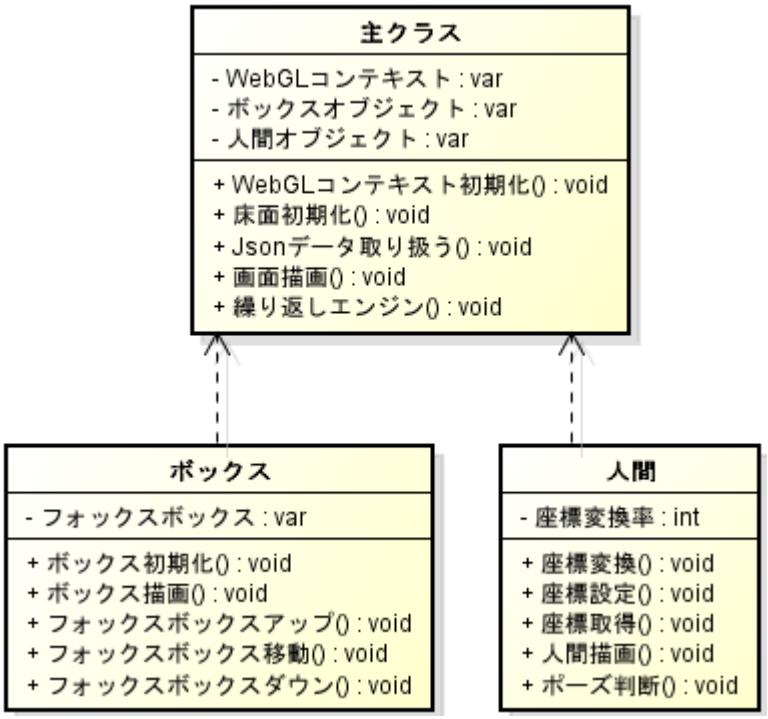
public	initRoll			小惑星の回転速度を設定する
public	loadKinect			Json データを取得する
public	canvasMain			Web ブラウザの起動メソッド

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, 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