ニューラルソフト有限会社

（公開資料）	~~草案・企画書~~ 計画~~・規格書~~ ~~検査計画・規格書~~ ~~検査成績書~~	検認		照査	作成
		―	―	―	市来 博記

表　　題

リアルノイド オリジン 運動制御原始モジュール
順/逆運動学演算ブロック 開発計画書

副　　題

RealNoids-Origin Motion Control Primitive Module
FK/IK Arithmetic Block Development Plan

キ ー ワ ー ド

リアルノイド/オリジン/運動制御/原始モジュール/順運動学/逆運動学/
RealNoids/Origin/Motion Control/Primitive Module/FK/IK

参照/添付資料

管理番号	件名	改定履歴
000-10-05-0101	ヒューマノイド研究開発	~~A B~~ C ~~D E F G H I J K L M N O P Q R S T U V W X Y Z~~

目次

1. はじめに

本書はロボバイオ リアルノイド オリジン（等身大ヒューマノイド）の全身の運動制御を実現するための運動制御原始モジュールの順/逆運動学演算ブロックの開発範囲とスケジュールを定義するものである。

2. 用語と略語の定義

本書で使用する用語と略語を表 2-1 に示す。

表 2-1 用語と略語の一覧

用語・略語	説明
原始モジュール	論理的・感情的解釈による制御のパラメータと経路の最適化を行わずに特定の機能を実現するハードウェアー又はソフトウェアー
フレーム	ヒューマノイドの骨組み
リンク	フレームを構成する節
リンク チェーン	フレーム全体又は特定部位を構成する節の繋がり
ジョイント	リンクを連結する部分
FK	リンク チェーンのジョイントの変位から姿勢を求める数学的プロセス
IK	リンク チェーンの姿勢からジョイントの変位を求める数学的プロセス
IK チェーン	IK の対象となるリンク チェーン
始端(BJ)	IK チェーンの根本となるジョイント
終端(EJ)	IK チェーンの末端となるジョイント
支端(FJ)	IK チェーンの屈曲部分の根本となるジョイント
屈曲基準(RJ)	IK チェーンの屈曲方向の基準となるジョイント
尖端(AJ)	IK チェーンの屈曲部分の頂点となるジョイント
モデル	3D 空間内の固有の座標系を定義する原点と XYZ 直交軸のベクトル、及び形状の情報
軸モデル	空形状のモデル
ジョイント モデル	XYZ 直交軸の角度制限の情報を持つモデル
コントローラ(C)	IK チェーンの終端の位置を特定する軸モデル
ポール(P)	IK チェーンの屈曲方向を特定する軸モデル
ポール ロータ(PR)	ポールを回転させるための IK ポールの親となる軸モデル
CCD	Cyclic-Coordinate-Descent 法（逆運動学の解法）
思考ブロック(T-BLK)	意志と行動を対応付ける最小単位のデータとプログラム(Thinking Block)
思考ブロック チェーン(T-BLKC)	複数の思考ブロックを連結したもの
順動力学演算	リンク チェーンのジョイントに発生するトルクに対する終端の加速度を求める数学的プロセス。
逆動力学演算	リンク チェーンの終端の加速度を実現するために必要となるジョイントのトルクを求める数学的プロセス。

3. 運動制御原始モジュール 順/逆運動学演算ブロックの概要

順/逆運動学演算ブロックはヒューマノイドの運動における体幹の末端（左右内側の肩関節と左右の股関節)、頭部、四肢の先端の動きを分割した位置における各関節の角度を算出することを目的とするソフトウェアーである。

4. 基本方針

順/逆運動学演算ブロックの開発における基本方針を示す。

- 実機環境（ARM 64BIT 4Core 以上 Linux 系 OS）及びシミュレーション環境（Windows 10 で動作する CINEMA 4D のプラグイン）で動作するネイティブ プログラムとし、C++言語で記述する。
- 空間座標系は Y UP/左手系とする。(座標系のスケールは使用しない。)
- ~~リンク チェーンの根本以外のジョイントは親ジョイントの Z 軸上に配置されるものとする。~~
- ジョイントの動作は回転（HPB 順）のみ可能とする。(リンクの伸縮は不可とする。)
- ジョイントは可動範囲の制限を設けることができるものとする。(-180.0 と+180.0 を跨ぐ範囲制限は不可とする。)
- ジョイントの座標変換は固定できるものとする。(任意の角度を 0 度とすることができる。)

5. 構成

順/逆運動学演算ブロックは実機環境及びシミュレーション環境下において、上位機能モジュールからの要求と指令に従って動作する。実機環境とシミュレーション環境における関連モジュールの構成を図 5-1 と図 5-2 に示す。

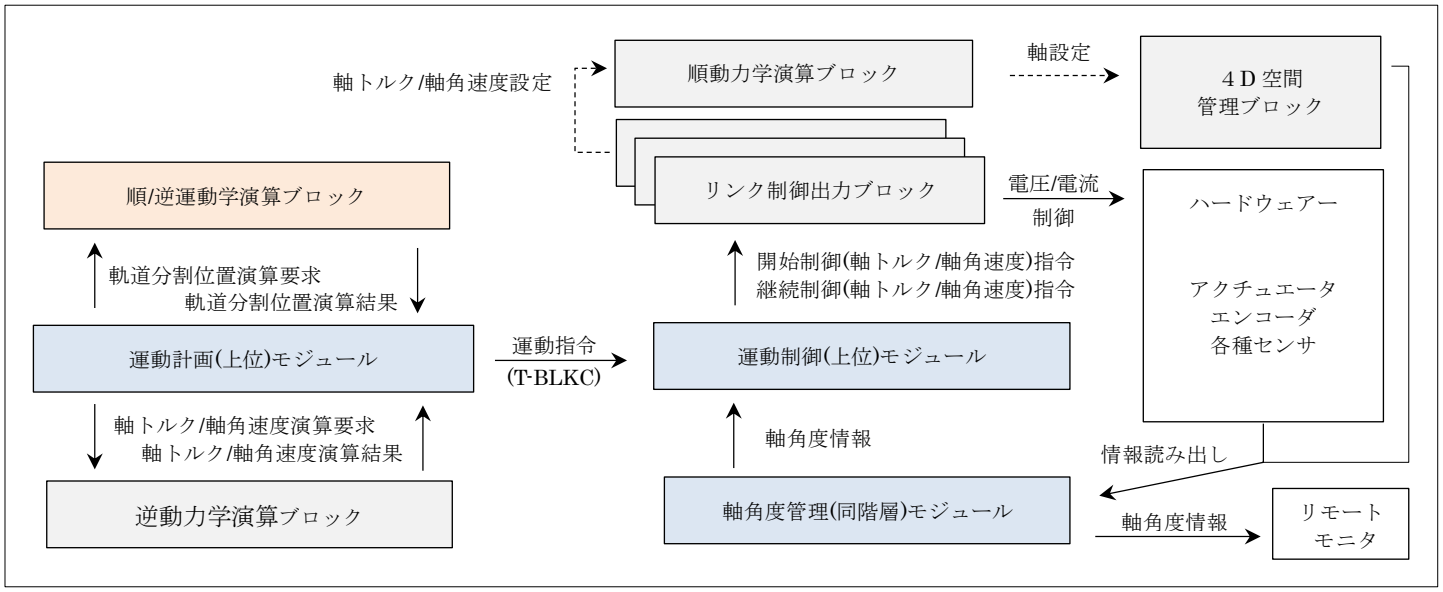

図 5-1 実機環境における関連モジュールの構成

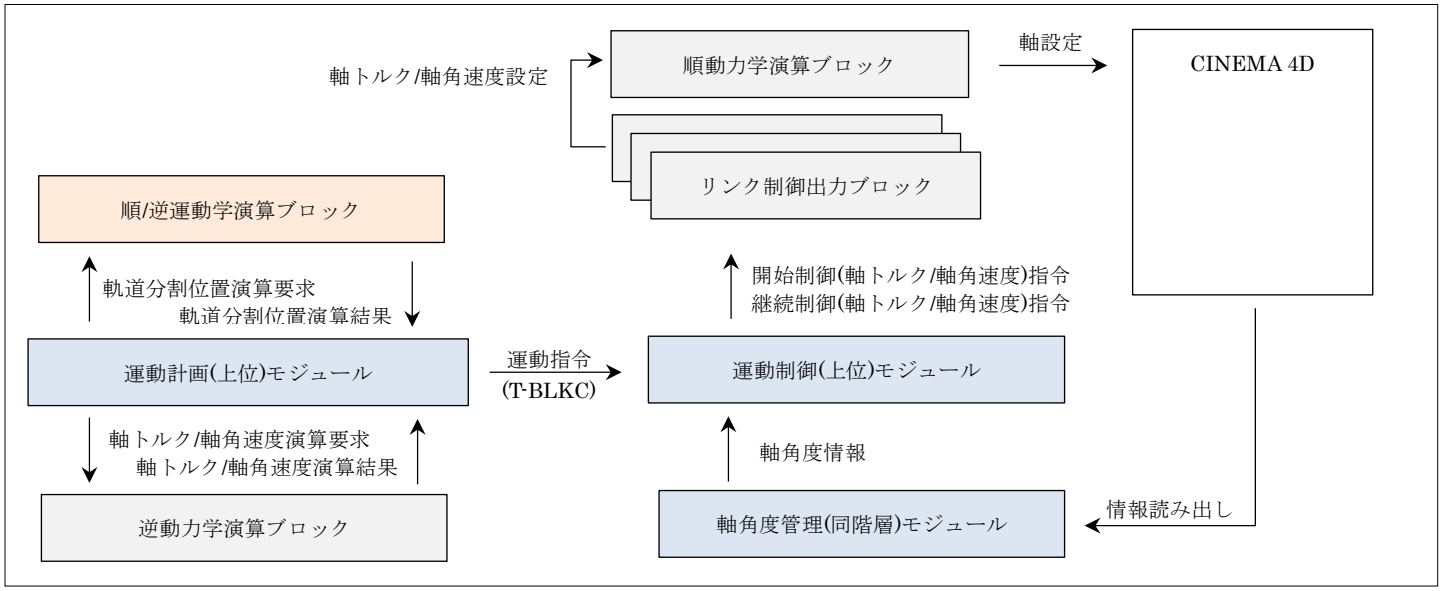

図 5-2 シミュレーション環境における関連モジュールの構成

6. 機能

順/逆運動学演算ブロックは以下の演算モードを有し、順運動学と逆運動学の演算結果を合成したリンクチェーンの姿勢におけるジョイントの角度を算出することができる。

- 完全 FK（シミュレーション環境のみ）
  IK 演算を実行しない。
- 2D ソルバ
  IK を解く際、IK チェーン内のジョイントを IK 平面（始端、ポール、コントローラの原点を含む平面）に投影した像（以降、2D 仮想ジョイントと記す。）の位置と軸角度を演算する。
- 3D ソルバ
  IK を解く際、IK チェーン内のジョイントの位置と軸角度を直接演算する。

逆運動学演算（FK 成分を考慮）の過程における IK チェーンの姿勢を表 6-1 に示す。

表 6-1 逆運動学演算過程おける IK チェーンの姿勢

姿勢の名称	説明
初期姿勢	一般的に T スタンス（ポーズ）や A スタンス（ポーズ）におけるリンク チェーンの姿勢。 （支端から終端まで直線上に配置することができる。）
照準姿勢	2D ソルバ 初期姿勢に FK 成分を適用した姿勢の始端を回転して終端をコントローラに向けた姿勢。 3D ソルバ 角度制限に従って可能な限り始端とコントローラを結ぶ直線上に全てのジョイントを配置した姿勢。
初期屈曲姿勢	2D ソルバ 照準姿勢における 2D 仮想ジョイントの繋がりを屈曲させた姿勢。 3D ソルバ 照準姿勢を屈曲させて FK 成分を適用した姿勢。 （屈曲方法は後述する。）
CCD N 回演算完了姿勢	CCD 演算を N 回実行した後の IK チェーンの姿勢。

次章以降に演算モード毎の詳細機能を示す。

6.1 ソルバ共通

(1) 演算スキップ（シミュレーション環境のみ）
IK 演算を無効化（＝完全 FK モード）することができる。

(2) IK チェーン指定
ジョイント モデルの配列によって IK チェーンを指定できる。

(3) 初期姿勢指定
IK チェーンを構成するジョイントの軸角度の配列によって IK チェーン初期姿勢を指定できる。

(4) FK 成分指定
IK チェーンを構成するジョイントの軸回転角度（IK チェーン初期姿勢が基準）の配列によって IK 演算時の FK 成分を指定できる。

(5) IK/FK 演算結果の混合指定

IK 演算と FK 演算による IK チェーンの姿勢の混合比を 0.0~1.0 の範囲（無段階）で指定できる。

0.0：リンク チェーンは IK 演算のみの姿勢となる。

0.5：リンク チェーンは IK 演算と FK 演算の中間の姿勢となる。

1.0：リンク チェーンは FK 演算のみの姿勢となる。

(6) FK 成分の強度指定

IK 演算に対する FK 成分の強度を 0.0~1.0 の範囲（無段階）で指定できる。

0.0：FK 成分はリンク チェーン姿勢に影響しない。

0.5：FK 成分はリンク チェーン姿勢に 50%の強度で影響を与える。

1.0：FK 成分はリンク チェーン姿勢に 100%の強度で影響を与える。

(7) コントローラ（終端目標位置）指定

軸モデルによって IK チェーンの終端となるジョイントの目標位置を指定できる。シミュレーション環境では、以下に示すコントローラの属性を指定できる。

(A) コントローラの位置を終端の到達可能範囲に制限する。

(B) 終端位置とコントローラの位置が閾値以上離れている場合であっても、コントローラの位置に変更がなければ IK 演算を実施しない。

(C) コントローラの軸の向きを特定の要素（始端/終端/「始端→コントローラ」ベクトル）に合わせる。（「始端→コントローラ」ベクトルを選択した場合は、コントローラの Z 軸を合わせる。）

(D) 完全 FK モード時にコントローラを保持する。（コントローラの位置を終端位置に保つ。）

(8) 逆運動学演算

IK を解くアルゴリズムに Cyclic-Coordinate-Descent（CCD）法（角度制限付き）を用いる。以下に 3 リンク構成の IK チェーンを例に、CCD 法の概要（処理ステップ）を示す。

(A) リンク 3 の始端側ジョイントを回転して終端をコントローラに向ける。

(B) リンク 2 の始端側ジョイントを回転して終端をコントローラに向ける。

(C) リンク 1 の始端側ジョイントを回転して終端をコントローラに向ける。

(D) 終端とコントローラの距離が閾値内であれば完了とする。そうでない場合は(A)から(D)を繰り返す。

図 6-1 に(A) から(C) のステップにおける IK チェーンの状態を示す。但し、図中のリンクの回転方法は 2D ソルバと 3D ソルバで異なる。

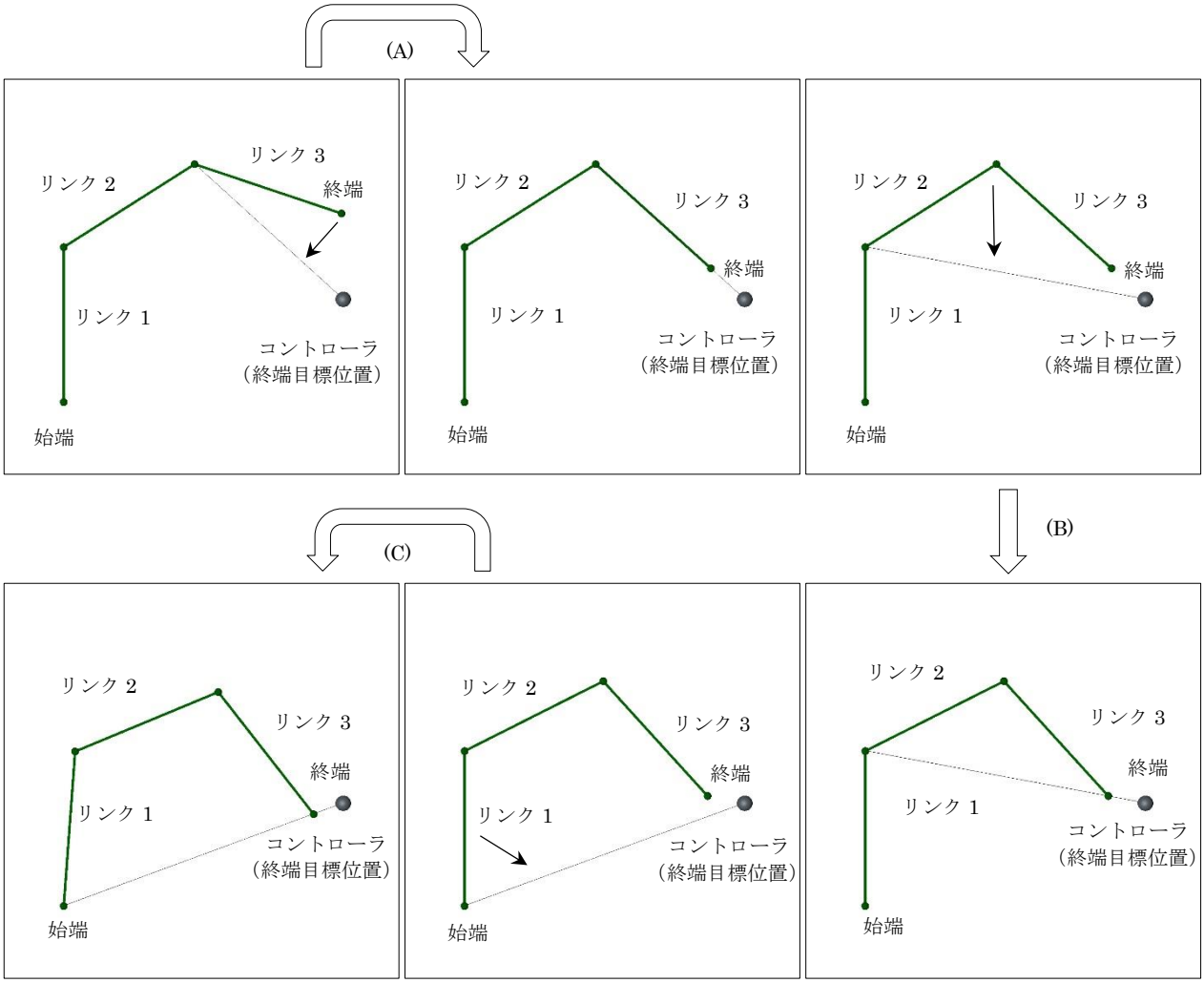

図 6-1 CCD アルゴリズムのステップにおける IK チェーンの状態

(9) CCD 法の属性指定

CCD 法の属性を指定できる。

(A) 最大繰り返し回数

-1：照準姿勢

0：初期屈曲姿勢

1~：CCD N 回演算完了姿勢

(B) 閾値（終端とコントローラの位置を同一と見なす距離）

(C) 最大連続角度超過回数（0 は無限）

角度変更で最大連続角度超過回数を超える角度超過が発生したジョイントを CCD 処理の対象から除外する。

(10) 固定ジョイント指定

始端及び終端からのジョイント数を指定することで、初期屈曲姿勢以降の姿勢における IK チェーンの末端に位置するジョイントの角度を固定することができる。（デバッグ・調査用の機能）

(11) 初期屈曲姿勢演算法指定

$\mathrm{ACos}\left(\frac{\text{支端からコントローラの長さ}}{\textit{IKチェーン全体の長さ}}\right)$を支端の屈曲角度（支端回転角度）とする初期屈曲姿勢の解法を以下の演算法から選択できる。

(A) n 乗根

支端と終端を除くジョイント数乗が 3 になる根を使用して、「終端の前->終端」の角度が「支端->支端の次」の角度×－1 になるようにジョイントの角度を設定する。

(B) T.B.D

(12) 初期屈曲係数基礎値指定

初期屈曲姿勢演算時のジョイント回転角度に対する係数の基礎値を 0.0 から 2.0 の範囲で指定できる。
初期屈曲係数は以下の式で求める。

$$\text{初期屈曲係数} = \left(1.0 - \frac{\text{支端回転角度(ラジアン)}}{0.5\pi}\right) \times \left(1.0 - \text{初期屈曲係数基礎値}\right) + \text{初期屈曲係数基礎値}$$

$$\text{支端回転角度(ラジアン)} = \mathrm{ACos}\left(\frac{\text{支端から IK コントローラの長さ}}{\text{IK チェーン長}}\right)$$

(13) 初期屈曲ウェイト基礎値（最大・最小値）指定

初期屈曲姿勢演算時におけるウェイト基礎値の最大値と最小値を-1.0 から 1.0 の範囲で指定できる。
初期屈曲ウェイトは以下の式で求める。

$$\text{初期屈曲ウェイト} = 1.0 + \left(\mathrm{Min} + C^3 \times (\mathrm{Max} - \mathrm{Min})\right)$$

$$\mathrm{C} = 1.0 - \frac{\text{ジョイント位置からウェイト最大位置までの長さ}}{\text{支端から終端の長}}$$

Min = ウェイト基礎最小値
Max = ウェイト基礎最大値

(14) IK 成分の可視化（シミュレーション環境のみ）

IK 解決による IK チェーンのジョイントの位置を結ぶ直線を指定の色で表示することができる。

(15) FK 成分の可視化（シミュレーション環境のみ）

FK による IK チェーンのジョイントの位置を結ぶ直線を指定の色で表示することができる。

(16) ハンドル ラインの可視化（シミュレーション環境のみ）

始端と終端を結ぶ直線を指定の色で表示することができる。

6.2 2D ソルバ

(1) 屈曲方向指定

ポール（軸モデル（原点のみ参照する。））の位置を指定することによって IK チェーンの姿勢を屈曲させる方向を指定できる。

(2) 屈曲基準と屈曲基準軸指定

IK チェーンの姿勢を屈曲させる際にポールに向ける基準ジョイントの基準軸を指定できる。

基準ジョイントは始端からの番号（始端=0）で指定する。

基準軸はフリー：軸指定なし/Y 軸正方向/ Y 軸負方向/X 軸正方向/ X 軸負方向から一つを選択できる。

(3) 屈曲ツイスト角度指定

IK チェーンの姿勢を屈曲させる際に始端の Z 軸をツイストする角度を指定できる。

(4) 屈曲方向変更補助（ポール ロータ）

IK チェーンの姿勢変更に追従するようにポールの位置を自動的に回転させるための基準となる軸モデル（ポールの親要素）を指定できる。

(5) IK 空間座標変換と 2D 仮想ジョイント化

IK を解く際、IK チェーンのジョイントの位置と軸角度を図 6-2 に示す IK 空間座標系に変換する。

更に、IK を解くプロセスでは、ジョイントを IK 平面（図 6-3 の三角平面を無限に拡張した平面。）に投影した像を 2D 仮想ジョイントとし、その位置と角度を演算してジョイントの軸角度を算出する。

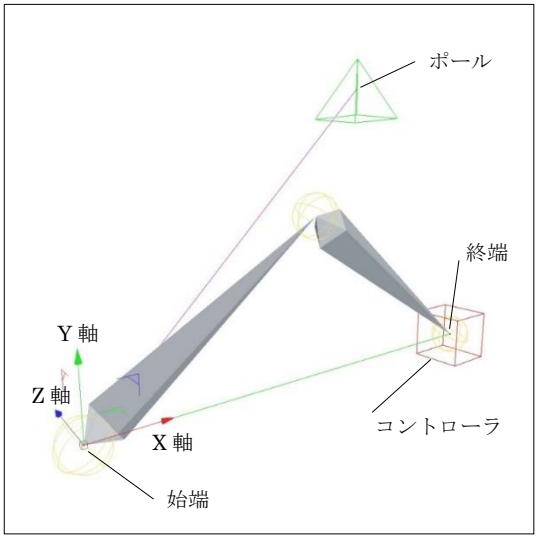

図 6-2 2D ソルバの IK 空間

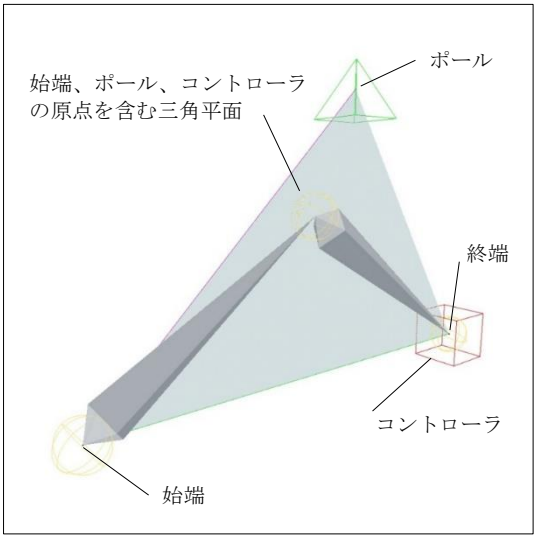

図 6-3 2D ソルバの IK 平面

IK 空間座標系の原点と軸の向きは以下の通りとする。

- 原点：始端の原点
- X 軸：始端からコントローラへの単位ベクトル
- Z 軸：X 軸と始端からポールへのベクトルの両方に垂直な単位ベクトル
- Y 軸：X 軸と Z 軸の両方に垂直な単位ベクトル

(6) 角度制限指定

IK チェーンを構成する全てのジョイントの角度制限の配列（要素：有効/無効、最小角度、最大角度）によって指定できる。

角度制限の最小/最大角度は IK 平面に投影された像の 2D 仮想ジョイントの絶対角度（親 2D 仮想ジョイントの角度に対する相対角度）を指定する。

(7) コントローラの属性指定

ソルバ共通機能のコントローラの属性指定に加えて、以下の設定が可能とする。

(A) コントローラの軸の向きをポール ロータに合わせる。
(B) コントローラの Z 軸とポール ロータの Z 軸を同期する。

(8) CCD 事前演算

CCD 法による IK と解く演算の前に IK チェーンに施す演算を以下に示す。

(A) 初期姿勢設定
ジョイントに初期姿勢の角度を設定する。
(B) 照準姿勢設定
始端を回転して終端をコントローラに向ける。（IK 平面を確定する。）
屈曲基準の基準軸がポールに向くように始端の Z 軸を回転する。
ジョイントに屈曲ツイスト角度と FK 成分を適用する。
(C) 初期屈曲姿勢設定
ジョイントの初期屈曲ウェイト値を算出する。
ジョイントに初期屈曲姿勢演算法で算出される角度＋FK 成分の角度を角度制限内で設定する。

(9) CCD におけるジョイント回転方法

CCD 法におけるジョイント（IK 平面に投影された像の 2D 仮想ジョイント）の回転軸は IK 平面の法線と平行で、ジョイントの原点を通るベクトルとする。

(10) ポール ベクタの可視化（シミュレーション環境のみ）

始端から IK ポールを結ぶ直線を指定の色で表示することができる。

6.3 3D ソルバ

(1) 屈曲基準と支端の指定

屈曲基準（屈曲方向の基準となるジョイント）を始端ジョイントからの番号で指定できる。支端（屈曲の根本となるジョイント=屈曲根）は屈曲基準の一つ前のジョイントとする。

(2) 屈曲方向指定

以下の方法で屈曲方向（屈曲ベクタ）を指定することができる。

(A) 屈曲根基準：手動

照準姿勢における支端の Y 軸からの角度（Z 軸回転角度）を指定する。支端の Y 軸を Z 軸周りに指定角度分回転したベクトルを屈曲ベクタとする。

(B) 屈曲根基準：自動

支端の Z 軸をコントローラに向けた座標系の XY 平面に投影した「コントローラから終端」へのベクトルを屈曲ベクタとする。

(C) グローバル

グローバル座標系におけるベクトルを指定する。指定されたグローバル座標系のベクトルを IK 空間座標系（(6)の項を参照のこと。）に変換したベクトルを屈曲ベクタとする。

(3) 屈曲根制御指定

屈曲根（支端）の P 軸が-90 から+90 度を超える可動範囲を有する場合、以下に示す屈曲根に対する制御を選択できる。

(A) 制御なし

屈曲ベクタから算出する屈曲根の角度をそのまま採用する。

(B) 反転抑制制御

屈曲ベクタから算出する屈曲根の P 軸の角度が-90~+90 度の範囲を超えないように P 軸の回転を停止する。

(C) 反転優先制御

屈曲根の P 軸の角度が 90 度を跨いで反対側の角度に遷移しやすいように、始端から屈曲根（支端）の H 軸までの角度を固定する。（反転優先制御が解除されるまで継続する。）

(4) 屈曲基準軸指定

IK チェーンの姿勢を屈曲させる際に屈曲方向に向ける屈曲基準の軸（フリー：軸指定なし/Y 軸正方向/ Y 軸負方向/X 軸正方向/ X 軸負方向）を指定することができる。

(5) 屈曲ツイスト角度指定

IK チェーンの姿勢を屈曲させる際に支端の Z 軸をツイストする角度を指定できる。

(6) IK 空間座標変換

IK を解く際、IK チェーンのジョイントの位置と軸角度を図 6-4 に示す IK 空間座標系に変換する。

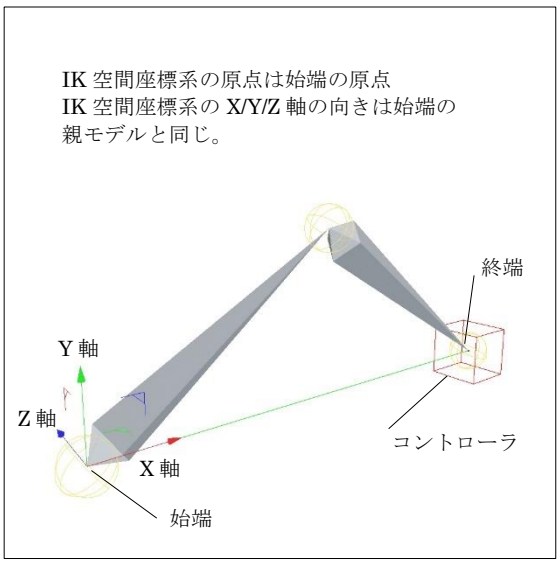

図 6-4 3D ソルバの IK 空間

(7) 角度制限指定

角度制限の配列（要素：有効/無効、最小角度、最大角度）によって IK チェーンを構成するジョイントの角度制限を指定できる。

角度制限の最小/最大角度はジョイントの絶対角度（座標変換固定角度からの相対角度）を指定する。

(8) CCD 事前演算

CCD 法による IK と解く演算の前に IK チェーンに施す演算を以下に示す。

(A) 初期姿勢設定

ジョイントに初期姿勢の角度を設定する。

(B) 照準姿勢設定

ジョイントの Z 軸をコントローラに向ける。（角度制限を適用する。）

(C) 屈曲方向算出

(D) 初期屈曲姿勢設定

ジョイントの初期屈曲ウェイト値を算出する。

ジョイントに初期屈曲姿勢演算法で算出される角度＋FK 成分の角度を角度制限内で設定する。（支端の角度設定は屈曲基準の屈曲基準軸を屈曲方向に向けるための Z 軸回転角度と屈曲ツイスト角度の設定を含む。）

(9) CCD 法の最大繰り返し回数指定

ソルバ共通機能の最大繰り返し回数指定値に加えて、以下の設定が可能とする。

-2：初期姿勢

(10) CCD におけるジョイント回転方向

CCD 法におけるジョイント回転時の軸は、当該ジョイントの原点を通り、当該ジョイントから終端へのベクトルと当該ジョイントからコントローラへのベクトルの両方に垂直なベクトルとする。

(11) IK 演算の詳細条件指定

以下の詳細条件を指定できる。

(A) 屈曲基準以降のバンク角（Z 軸の角度）の固定の有無

(B) 屈曲ベクタの固定の有無（屈曲方向指定が屈曲根基準：自動以外の場合のみ）

(C) 屈曲ベクタ算出時の屈曲根のバンク角項除外の有無（屈曲方向指定が屈曲根基準：手動の場合のみ）

(D) 屈曲ベクタ先端の侵入禁止領域と侵入時の動作（屈曲方向指定が屈曲根基準：手動の場合のみ）

侵入禁止領域は屈曲根の原点を頂点とする円錐の頂点部の角度を指定する。（0~180 度）

侵入時の動作は、侵入禁止領域の回り込み、又は停止を選択できる。

(E) 自動屈曲の感度と強度（屈曲方向指定が屈曲根基準：自動の場合のみ）

感度は、支端の座標系の XY 平面に投影した「コントローラから終端」へのベクトルの長さを指定する。（この長さに満たない場合は、屈曲ベクタを変更しない。）（0.001~10.0 システム単位）

強度は、屈曲ベクタ変更角度に対する係数（0.0~1.0）を指定する。

(F) 最大連続角度超過回数

最大連続角度超過回数を超えたジョイントは CCD 処理の対象外とする。

(G) B 軸回転抑制角度

「ジョイントからコントローラ」へのベクトル、又は「ジョイントから終端」へのベクトルと、ジョイントの Z 軸の角度が B 軸回転抑制角度未満のジョイントの B 軸回転を抑制する。

(H) B 軸回転抑制時の H 軸角度の補正の有無

(I) 屈曲ベクタ平滑化角度

屈曲ベクタの角度を一度に変更できる最大角度を指定できる。（0=平滑化なし~45 度）

(J) CCD 平滑化係数（0.1~1.0=平滑化なし）

(K) 非 Z 軸配置

子となるジョイントを Z 軸上に配置しないことを指定できる。

(L) B 軸回転抑制を全フェーズに適用

B 軸回転抑制を CCD N 回演算完了姿勢にのみ適用するか、その他の姿勢（初期姿勢、照準姿勢、初期屈曲姿勢）にも適用するかを指定できる。

(M) 照準姿勢の屈曲根（支端）へのツイストの適用

CCD の最大繰り返し回数が－1（照準姿勢）時に屈曲根（支端）をツイストするかどうかを指定できる。

(12) 屈曲方向の可視化（シミュレーション環境のみ）

任意のモデルを指定することでIKチェーンの屈曲方向を確認できる。（モデルのZ軸を屈曲方向に向ける。）

7. 実装の指針

順/逆運動学演算ブロックは二つの詳細ブロックに分割し、各詳細ブロックを C++言語の 2 クラス+1 共通クラス+2 共通データ構造+3D ソルバ専用データ構造で実現する。

(1) 詳細ブロック構成

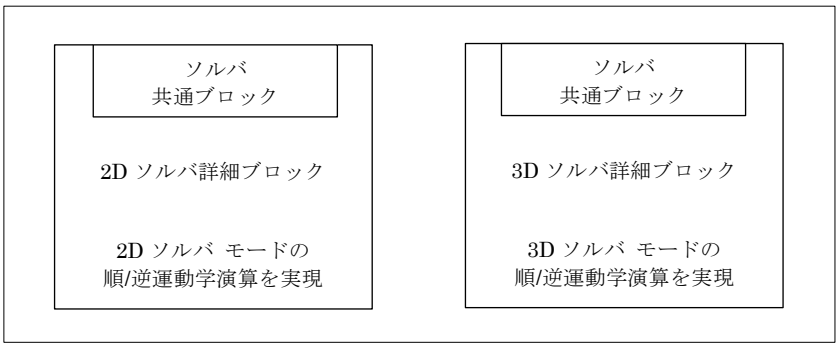

図 7-1 順/逆運動学演算ブロックの詳細ブロック構成

(2) クラス構成

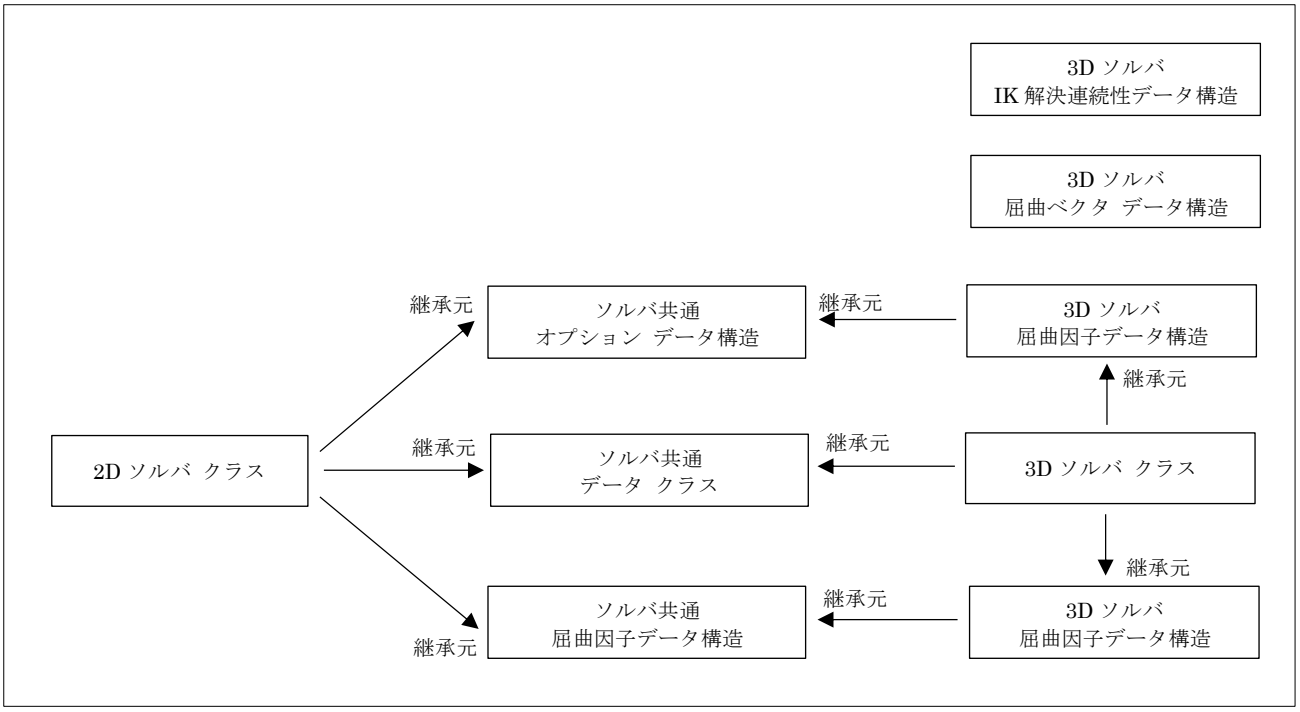

図 7-2 順/逆運動学演算ブロックの詳細ブロックのクラス構成

次章以降に各クラスの実装の指針を示す。

7.1 実装の指針の記載に関する規約

実装の指針の記載に関する規約と、使用する記号及び略語を以下に示す。

データのアクセス保護レベルに関する文言の定義

- 外部データ：アクセス保護のないデータ
- 保護データ：当該データと当該データを継承するデータの付帯関数のみアクセス可能なデータ
- 内部データ：当該データに付帯する関数のみアクセス可能なデータ

データに付帯する関数の実行保護レベルに関する文言の定義

- 外部関数：実行保護のない関数
- 保護関数：当該データと当該データを継承するデータの付帯関数からのみ実行可能な関数
- 内部関数：当該データに付帯する関数からのみ実行可能な関数

ブロック（データと付帯する関数の組み合わせ）の概念の名称に関する定義

- データの全項目が外部データの場合、名称の最後に「構造」又は「データ」を付加する。
- データの全項目が外部データでない場合、名称の最後に「クラス」を付加する。
- ブロックの名称の最後が「クラス」の場合、データの全項目は、特に指定がない限り内部データとする。

データの構造に関する記載で使用する記号を表 7-1 に示す。

表 7-1 データの構造に関する記載で使用する記号

記号	意味
＜－＞	データの型で定義される初期値
＜・＞	コンストラクタの入力で定義される初期値
Ⓔ	外部データ（public）
Ⓟ	保護データ（protected）
Ⓘ	内部データ（private）

データの構造と付帯する機能に関する記載で使用する略語を表 7-2 に示す。

表 7-2 データの構造と付帯する機能に関する記載で使用する略語

略語	意味
M	マトリクス
Ml	ローカル座標系変換マトリクス
Mg	グローバル座標系変換マトリクス
L 座標／L 原点	ローカル座標系の座標／ローカル座標系におけるモデル又は座標系の原点
G 座標／G 原点	グローバル座標系の座標／グローバル座標系におけるモデル又は座標系の原点
J 座標	ジョイントのローカル座標系の座標

データに付帯する機能に関する記載で使用する記法を表 7-3 に示す。

表 7-3 データに付帯する機能に関する記載で使用する記法

<table>
<tr><th>記法</th><th>説明／例</th></tr>
<tr><td>I.項目名又は項目番号</td><td>当該機能の入力項目</td></tr>
<tr><td>#.項目名又は項目番号</td><td>当該データの項目</td></tr>
<tr><td>#.(^).項目名又は項目番号</td><td>当該データの継承元データの項目</td></tr>
<tr><td>~.*</td><td>当該データの全項目</td></tr>
<tr><td>~.$</td><td>当該データの特定の項目（対する辺の項目に対応する項目）</td></tr>
<tr><td>~.[A,B,･･･]／~.[A~B]</td><td>当該データの A 項目と B 項目／当該データの A 項目から B 項目</td></tr>
<tr><td>@、@1、･･･、@n</td><td>可変データの定義</td></tr>
<tr><td>A.項目名</td><td>当該機能内で定義される項目値</td></tr>
<tr><td>代入先 ＝ 条件 ？ 値 1： 値 2</td><td>C 言語の三項演算子と同じ</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>外部機能の呼び出し（括弧内は呼び出す機能への入力）</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>保護機能の呼び出し</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>内部機能の呼び出し</td></tr>
<tr><td>＜機能名＞(入力値、出力領域、･･･)</td><td>本書の記載範囲外のブロックの外部機能の呼び出し</td></tr>
<tr><td>Ⓢ機能名</td><td>静的関数</td></tr>
<tr><td>Ⓥ機能名／Ⓥ機能名</td><td>仮想関数／純粋仮想関数</td></tr>
<tr><td>機能名Ⓒ</td><td>コンスト属性付き関数</td></tr>
<tr><td>〔○○○○○〕〔○○○○○〕</td><td>略語の定義、コメント等</td></tr>
<tr><td>場合分け</td><td>二つの場合の例
1 行で条件判定と処理を記載する記法
“条件判定”、“処理”
“そうでない”、“処理”
条件判定と処理を改行して記載する記法
“条件判定の処理”
処理（字下げする）
“そうではない”
処理（字下げする）

三つ以上の場合の例
<table>
<tr><th>条件判定</th><th>処理</th></tr>
<tr><td>1 の条件判定</td><td>1 の条件が満たされた場合の処理</td></tr>
<tr><td>・
・
・</td><td>・
・
・</td></tr>
<tr><td>n の条件判定</td><td>n の条件が満たされた場合の処理</td></tr>
</table>
</td></tr>
</table>

7.2 ソルバ共通オプション データ

ソルバ共通オプション データ（IkOptionData）の構成を示す。

(1) 管理データ

表 7-4 ソルバ共通オプション データの管理データ

番号	項目	初期値	型
1	初期姿勢演算方式型	－	enum class InitType
2	CCD 固定データ型（表 7-5 を参照のこと。）	－	struct CcdFix
3	CCD 閾値	0.01	Float
4	CCD 最大繰り返し回数	20	Int32
5	最大連続角度超過回数（0：無限）	0	Int32
6	初期屈曲姿勢演算法（n 乗根/T.B.D）	n 乗根	InitType
7	初期屈曲係数基礎値	1.0	Float
8	初期屈曲ウェイト基礎値（最大値・最小値）	(0,0)	rbxSys::Span<Float>
9	CCD 平滑化係数（0.1~1.0=平滑化なし）	0.1	Float
10	CCD 固定データ	＜－＞	CcdFix

表 7-5 CCD 固定データ型

番号	項目	初期値	型
1	固定数（始端側と終端側）	(0,0)	rbxSys::Shrink<Int32>
2	始端側適用指定（true：適用する／false：適用しない）	false	Bool
3	終端側適用指定（true：適用する／false：適用しない）	false	Bool

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-4 から表 7-5 の初期値

機能	コンストラクタ（無処理）	
入力	無処理指定	ENUM_DONT_INITIALIZE
処理	無処理	

7.3 ソルバ共通屈曲因子データ

ソルバ共通屈曲因子データ（IkBendFactorData）の構成を示す。

(1) 管理データ

表 7-6 ソルバ共通屈曲因子データの管理データ

番号	項目	初期値	型
1	屈曲基準軸指定型ⓔ （FREE:基準軸なし/X+/X-/Y+/Y-）	－	enum class BendRefAxis
2	屈曲基準番号	1	Int32
3	屈曲基準軸	Y+	BendRefAxis
4	屈曲基準ツイスト角度	0.0	Float

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-6 の初期値

7.4 ソルバ共通データ クラス

2D ソルバと 3D ソルバの共通データの管理を実現するソルバ共通データ クラス（IkCoreData）の構成を示す。

(1) 管理データ

表 7-7 ソルバ共通データ クラスの管理データ

番号	項目	初期値	型
1	ジョイント データ型Ⓔ（表 7-8 を参照のこと。）	－	struct IkJointData
2	3 の n 乗根テーブル（要素数は IK チェーンが脊柱の場合を想定。）Ⓟ	3のn乗根	const Float[28]
3	ソルバの状態（初期/IK チェーン接続/IK 解決済み）Ⓟ	初期	enum class SolveState
4	バインド タイプ（0：直線バインド／1：非直線バインド）Ⓟ	0	enum class BindType
5	CCD 終了ステータスⓅ	－	－
5.1	CCD 演算回数	-1	Int32
5.2	演算誤差（終端とコントローラの距離）	0	Float

表 7-8 ジョイント データ型

番号	項目	初期値	型
1	モデル	nullptr	BaseObject*
2	ユニーク ID	0	UInt64
3	角度制限データ	－	VectorSpan
3.1	H 軸制限状況（true:制限あり/false:制限なし）	False	Bool
3.2	P 軸制限状況（true:制限あり/false:制限なし）	False	Bool
3.3	B 軸制限状況（true:制限あり/false:制限なし）	False	Bool
3.4	最小角度	(0,0,0)	Vector
3.5	最大角度	(0,0,0)	Vector
4	座標変換固定 Ml と相対角度	－	rbxSys::RelRotFrozen

(2) 外部関数

機能	CCD 演算回数取得	
処理	出力値=#.CCD 演算回数	
出力	IK 解決時に実行した CCD 演算の回数	Int32

機能	CCD 演算誤差取得	
処理	出力値=#.CCD 演算誤差	
出力	IK 解決終了時の終端とコントローラの距離	Float

(3) 保護関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-7 の初期値

機能	ⓥデストラクタ
処理	無処理

機能	バインド タイプ確定	
入力	ジョイント データ配列	const BaseArray<IkJointData>&
処理	〔始端(0 番)→1 番ジョイントへの G 座標系ベクトルを基準ベクトルとして求める @=I.ジョイント データ配列[0]、@+1=I.ジョイント データ配列[1]と定義する〕 A.Mg=@.モデル.親の Mg×@.座標変換固定 Ml と相対角度.＜絶対 Ml 取得＞() A.始点=A.Mg.原点 A.Mg=A.Mg×@+1.座標変換固定 Ml と相対角度.＜絶対 Ml 取得＞() A.基準ベクトル={正規化}←(A.Mg.原点－A.始点) 〔2 番目以降のジョイントが A.基準ベクトルの延長線上にあるかどうかを確認〕 @=I.ジョイント データ配列[2~末尾]の処理 　A.始点=A.Mg.原点 　A.Mg=A.Mg×@.座標変換固定 Ml と相対角度.＜絶対 Ml 取得＞() 　A.ベクトル={正規化}←(A.Mg.原点－A.始点) 　rbxSys::isEqual(A.ベクトル, A.基準ベクトル) == false 　　#.バインド タイプ=非直線バインド 　　処理を終了 #.バインド タイプ=直線バインド	

7.5 2D ソルバ クラス

2D ソルバ モードの順/逆運動学演算を実現する 2D ソルバ クラス(Ik2DSolver：public IkCoreData, protected IkOptionData, protected IkBendFactorData)の構成を示す。

(1) 管理データ

表 7-9 2D ソルバ クラスの管理データ

番号	項目	初期値	型
1	IK 空間座標系 Mg	無変換 M	Matrix
2	IK 空間座標系逆 Mg	無変換 M	Matrix
3	ジョイント データ配列（ジョイント データは表 7-10 から表 7-14 の内容で構成する。）	要素数=0	BaseArray<Ik2dJointData>
4	IK チェーンの 2D 長	0.0	Float
5	最初の回転可能ジョイントの番号	-1	Int32
6	最後の回転可能ジョイントの番号	-2	Int32
7	回転可能ジョイント数	0	Int32
8	コントローラの位置（IK 空間座標系）	(0,0,0)	Vector
9	ポールの位置（IK 空間座標系）	(0,0,0)	Vector

表 7-10 基本データ

番号	項目	初期値	型
1	ジョイント番号	－	Int32
2	モデル	－	BaseObject*
3	座標変換固定 Ml	－	Matrix
4	座標変換固定逆 Ml	－	Matrix
5	初期姿勢の角度	－	Vector
6	角度制限状況（true:制限あり/false:制限なし）	－	Bool
7	角度制限値（最小値/最大値）	－	Span<Float>

表 7-11 FK データ

番号	項目	初期値	型
1	FK の角度（初期姿勢からの相対角度）	－	Vector
2	FK の実効角度（強度を考慮した FK の角度）	－	Vector

表 7-12 演算データ

番号	項目	初期値	型
1	相対 Ml	－	Matrix
2	IK 空間座標系 Ml	－	Matrix

表 7-13 2D 演算データ

番号	項目	初期値	型
1	2D 位置(演算前)	－	Vector
2	2D 位置(演算後)	－	Vector
3	2D ジョイント間長（IK 平面に投影した長さ）	－	Float
4	2D 始端からの長さ（IK 平面に投影した長さ）	－	Float
5	2D 絶対角度(演算前)(基準=IK 空間座標系の X 軸)	－	Float
6	2D 絶対角度(演算後)(基準=IK 空間座標系の X 軸)	－	Float
7	2D 相対角度（2D 仮想ジョイント相対）	－	Float
8	2D 初期屈曲ウェイト値	－	Float
9	CCD 演算毎の 2D 相対角度の前値	－	Float
10	FK 成分なしの 2D 初期位置	－	Vector
11	FK 成分の 2D 相対角度	－	Vector
12	連続角度超過回数	0	Int32

表 7-14 演算結果データ

番号	項目	初期値	型
1	Mg	－	Matrix
2	相対角度	－	Vector

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-9 の初期値

機能	IK オプション設定	
入力	IK オプション データ	const IkOptionData&
処理	#.(^)=I.IK オプション データ	

機能	IK チェーン接続	
入力	ジョイント データ配列（角度制限値は X 軸の設定値のみ参照する。）	const BaseArray<IkJointData>&
処理	ソルバの状態を「初期」に設定 I.ジョイント データ配列.要素数 ＜2 || I.ジョイント データ配列.要素数 ＞T.B.D、終了(出力値=false) #.ジョイント データ配列.要素数 != I,ジョイント データ配列.要素数 　#.ジョイント データ配列.＜領域確保＞(I.ジョイント データ配列.要素数) 　領域確保 != 正常完了、終了(出力値=false) #.(^).＜バインド タイプ確定＞(I.ジョイント データ配列) ＜基本データ設定＞(I.ジョイント データ配列) #.(^).CCD 終了ステータス=初期値 #.(^).ソルバの状態=「IK チェーン接続」 出力値=true	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	屈曲因子設定	
入力	屈曲因子データ	const IkBendFactorData&
処理	#.(^).ソルバの状態 == 初期、終了(出力値=false) #.(^).屈曲因子データ=I.屈曲因子データ 出力値=true	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	IK 解決	
入力	FK 成分配列	const BaseArray<Vector>&
	FK の強度	Float
	コントローラ	const BaseObject*
	ポール	const BaseObject*
処理	#.(^).ソルバの状態 == 初期、終了(出力値=false) #.ジョイント データ配列.要素数 != FK 成分配列.要素数、終了(出力値=false) #.(^).CCD 終了ステータス=初期値 #.(^).ソルバの状態 == IK 解決済み、＜連続角度制限超過回数クリアー＞() ＜IK 空間座標系確定＞(#.ジョイント データ配列[0].モデル，I.コントローラ，I.ポール) ＜FK 成分確定＞(I.FK 成分配列，I.FK の強度) ＜IK 平面確定＞() ＜FK 成分適用＞() ＜2D 演算初期データ確定＞() #.CCD 最大繰り返し回数 == -1〔照準姿勢〕、＜IK 解決をスキップ＞()；終了(出力値=true) ＜初期屈曲姿勢設定＞() CCD 繰り返し処理 〔繰り返し数の上限：#.(^).CCD 最大繰り返し回数、 繰り返し終了条件：終端とコントローラ間の距離 ＜#.(^).CCD 閾値〕 　＜CCD＞(Min(処理回数×0.1，1.0)) #.(^).CCD 終了ステータスを設定 ＜IK 解決演算結果設定＞() #.(^).ソルバの状態=「IK 解決済み」 出力値=true	
出力	処理結果（true：正常終了／false：異常終了）	Bool

<table>
<tr><td>機能</td><td colspan="2">IK チェーン姿勢更新</td></tr>
<tr><td>入力</td><td>FK 混合比</td><td>Float</td></tr>
<tr><td>処理</td><td colspan="2">#.(^).ソルバの状態 ==「IK 解決済み」
FK 混合比別の処理
<table>
<tr><th>FK 混合比</th><th>処理</th></tr>
<tr><td>0.0
(IK+Rel FK)</td><td>@=#.ジョイント データ配列[0~末尾]の処理
@.モデル.相対角度=@.(演算結果データ).相対角度</td></tr>
<tr><td>1.0
(Abs FK)</td><td>@=#.ジョイント データ配列[0~末尾]の処理
@.モデル.相対角度=
@.初期姿勢の角度+@.FK の角度</td></tr>
<tr><td>0.0<FK 混合比<1.0
(IK+Rel FK + Abs FK)</td><td>@=#.ジョイント データ配列[0~末尾]の処理
@.モデル.相対角度=
@.(演算結果データ).相対角度×(1.0-I.FK 混合比) +
(@.初期姿勢の角度+@.FK の角度)×I.FK 混合比</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>機能</td><td colspan="2">IK チェーン G 位置取得</td></tr>
<tr><td>入力</td><td>IK チェーン G 位置配列</td><td>BaseArray<Vector>&</td></tr>
<tr><td>処理</td><td colspan="2">#.(^).ソルバの状態 ==「IK 解決済み」
I.IK チェーン G 位置配列.要素数=#.ジョイント データ配列.要素数
@1=I.IK チェーン G 位置配列[*]、@2=#.ジョイント データ配列[*]の処理
@1 = @2.(演算結果データ).Mg.位置
そうでない
I.IK チェーン G 位置配列.要素数=0</td></tr>
</table>

(3) 内部関数マップ

IK チェーン接続系	IK 解決系
IK チェーン接続 ├(ソルバ共通データ.バインド タイプ確定) └基本データ設定	IK 解決 ├連続角度制限超過回数クリアー ├IK 空間座標系確定 ├FK 成分確定 ├IK 平面確定 │ ├初期姿勢設定 │ │ └IK 空間座標系 Ml 確定 │ └簡易照準姿勢設定 ├FK 成分適用 │ ├IK 空間座標系 Ml 確定 │ └照準姿勢設定 ├2D 演算初期データ確定 ├初期屈曲姿勢設定 │ ├2D 角度変換(絶対角度→相対角度) │ ├2D 初期屈曲ウェイト確定 │ ├2DFK 成分適用 │ ├2D 角度制限適用 │ ├2D 角度変換(相対角度→絶対角度) │ └2D 位置更新 ├CCD │ ├2D 角度変換(相対角度→絶対角度) │ └2D 位置更新 ├IK 解決をスキップ └IK 解決演算結果設定

(4) 内部関数

機能	基本データ設定	
入力	ジョイント データ配列	const BaseArray<IkJointData>&
処理	@1=#.ジョイント データ配列[*]、@2=I.ジョイント データ配列[*] @1.$=@2.* @1.連続角度超過回数=0 #.最初の回転可能ジョイントの番号=始端に最も近い回転可能なジョイントの番号 #.最後の回転可能ジョイントの番号=終端に最も近い回転可能なジョイントの番号	

機能	IK 空間座標系確定	
入力	始端	const BaseObject* op
	コントローラ	const BaseObject* op
	ポール	const BaseObject* op
処理	A.ポール ベクタ=ポール.G 原点－始端.G 原点 A.X 軸=コントローラ.G 原点－始端.G 原点 A.Z 軸=Cross(A.X 軸, A.ポール ベクタ) A.Y 軸=Cross(A.Z 軸, A.X 軸) #.IK 空間座標系 Mg={正規化}←Matrix(始端.G 原点, A.X 軸, A.Y 軸, A.Z 軸) #.IK 空間座標系逆 Mg=Inv(#.IK 空間座標系 Mg) #.コントローラの位置=#.IK 空間座標系逆 Mg×コントローラ.G 原点 #.ポールの位置=#.IK 空間座標系逆 Mg×ポール.G 原点	

機能	FK 成分確定	
入力	FK 成分配列	const BaseArray<Vector>&
	FK の強度（0.0~1.0）	Float
処理	@1=#.ジョイント データ配列[*]、@2=I.FK 成分配列[*] @1.(FK データ).FK の角度=@2 @1.(FK データ).FK の実効角度=@2×FK の強度	

機能	IK 平面確定
処理	＜初期姿勢設定＞() ＜簡易照準姿勢設定＞()〔始端、屈曲基準、終端のみをコントローラに向ける〕 #.(^).屈曲基準軸 != FREE 〔屈曲基準の基準軸をポールに向ける〕 〔基準軸ベクトルを取得〕 A.基準軸ベクトル= rbxSys::getMatrixAxis(#.ジョイント データ配列[#.(^).屈曲基準の番号].IK 空間座標系 Ml, (#.(^).屈曲基準軸÷2) + 1) #.(^).屈曲基準軸 % 2 != 0〔基準軸の方向が負方向の場合〕、A.基準軸ベクトル *= -1 〔基準軸ベクトルと「始端→ポール」ベクトルの角度 + 屈曲ツイスト角度分、 始端を IK 空間座標系 X 軸回転 ZY 平面上での基準軸ベクトルから atan2 で角度を求めると Z 軸からの角度となる為、 基準軸ベクトルの Z 値と Y 値を入れ替えて計算する〕 A.回転角度=ATan2(A.基準軸ベクトル.z, A.基準軸ベクトル.y) + #.(^).屈曲ツイスト角度 〔ジョイント データ配列[0]を始端データと定義する〕 始端データ.IK 空間座標系 Ml=MatrixRotX(A.回転角度)×始端データ.IK 空間座標系 Ml そうではない 〔ジョイント データ配列[0]を始端データと定義する〕 始端データ.IK 空間座標系 Ml=MatrixRotX(#.(^).屈曲ツイスト角度)×始端データ.IK 空間座標系 Ml

機能	初期姿勢設定
処理	@=#.ジョイント データ配列[*]の処理 　@.相対 Ml=HPBToMatrix(@.初期姿勢の角度，ROTATIONORDER::HPB) 〔ジョイント データ配列[0]を始端データと定義する〕 #.(^).屈曲基準軸 == FREE 　A.Ml = #.IK 空間座標系逆 Mg× 　　(始端データ.モデル.親モデル.Mg×始端データ.座標変換固定 Ml×始端データ.相対 Ml) 　始端データ.座標変換固定 Ml.原点=Vector(0) 　始端データ.座標変換固定逆 Ml=Inv(始端データ.座標変換固定 Ml) 　始端データ.相対 Ml=始端データ.座標変換固定逆 Ml×A.Ml そうではない 　始端データ.座標変換固定 Ml.原点=Vector(0) 　始端データ.座標変換固定逆 Ml=Inv(始端データ.座標変換固定 Ml) ＜IK 空間座標系 Ml 確定＞()

機能	簡易照準姿勢設定（始端、屈曲基準、終端のみをコントローラに向ける）
処理	〔#.ジョイント データ配列[0]を始端データ、 #.ジョイント データ配列[#.(^).屈曲基準の番号]を基準データ、 #.ジョイント データ配列[末尾]を終端データと定義する〕 A.終端の位置〔IK 空間座標系〕=終端データ.IK 空間座標系 Ml.原点 〔IK チェーンを IK 空間座標系 Y 軸回転して終端をコントローラに向ける〕 A.回転 M=MatrixRotY(－ATan2(A.終端の位置.z，A.終端の位置.x)) 始端データ.IK 空間座標系 Ml=A.回転 M×始端データ.IK 空間座標系 Ml 基準データ.IK 空間座標系 Ml=A.回転 M×基準データ.IK 空間座標系 Ml 終端データ.IK 空間座標系 Ml=A.回転 M×終端データ.IK 空間座標系 Ml 〔IK チェーンを IK 空間座標系 Z 軸回転して終端をコントローラに向ける〕 A.回転 M=MatrixRotZ(ATan2(A.終端の位置.y，A.終端の位置.x)) 始端データ.IK 空間座標系 Ml=A.回転 M×始端データ.IK 空間座標系 Ml 基準データ.IK 空間座標系 Ml=A.回転 M×基準データ.IK 空間座標系 Ml 終端データ.IK 空間座標系 Ml=A.回転 M×終端データ.IK 空間座標系 Ml

機能	FK 成分適用
処理	〔#.ジョイント データ配列[0]を始端データと定義する〕 始端データ.相対 Ml=始端データ.IK 空間座標系 Ml×HPBToMatrix(始端データ.FK の実効角度) @=#.ジョイント データ配列[1~末尾]、@$^{-1}$=@の前のジョイント データの処理 　A.前の FK 実効角度=@.座標変換固定 Ml×@$^{-1}$.FK の実効角度 　@.相対 Ml=HPBToMatrix(　　@.初期姿勢の角度－A.前の FK 実効角度＋@.FK の実効角度，ROTATIONORDER::HPB) ＜IK 空間座標系 Ml 確定＞() ＜照準姿勢設定＞()

機能	照準姿勢設定（全てのジョイントをコントローラに向ける）
処理	A.終端の位置〔IK 空間座標系〕=#.ジョイント データ配列[末尾].IK 空間座標系 Ml.原点 〔IK チェーンを IK 空間座標系 Y 軸回転〕 A.回転 M=MatrixRotY(－ATan2(A.終端の位置.z，A.終端の位置.x)) @=ジョイント データ配列[*]の処理 　@.IK 空間座標系 Ml=A.回転 M×@.IK 空間座標系 Ml 〔IK チェーンを IK 空間座標系 Z 軸回転〕 A.回転 M=MatrixRotZ(ATan2(A.終端の位置.y，A.終端の位置.x)) @=ジョイント データ配列[*]の処理 　@.IK 空間座標系 Ml=A.回転 M×@.IK 空間座標系 Ml

機能	2D 演算初期データ確定
処理	〔ジョイントの 2D 位置を設定〕 @=ジョイント データ配列[*]の処理 @.2D 位置(演算前)=@.IK 空間座標系 Ml.原点 @.2D 位置(演算前).z = 0 〔始端から終端の前のジョイントの 2D 角度と 2D 長を設定〕 A.IK チェーン長=0 @=#.ジョイント データ配列[1~末尾]、@$^{-1}$=@の前のジョイント データの処理 A.ベクトル=@.2D 位置(演算前)－@$^{-1}$.2D 位置(演算前) @$^{-1}$.2D ジョイント間長=A.ベクトル.長さ @$^{-1}$.2D 始端からの長さ=A.IK チェーン長 A.IK チェーン長 += A.ベクトル.長さ @$^{-1}$.2D 絶対角度(演算前)=ATan2(A.ベクトル.y，A.ベクトル.x) 〔終端の 2D 角度と 2D 長を設定、 #.ジョイント データ配列[末尾]を終端データ、 #.ジョイント データ配列[末尾－1]を終端$^{-1}$データと定義する〕 終端データ.2D ジョイント間長=0 終端データ.2D 始端からの長さ=A.IK チェーン長 終端データ.2D 絶対角度(演算前)=終端$^{-1}$データ.2D 絶対角度(演算前)

機能	初期屈曲姿勢設定
処理	＜2D 角度変換(絶対角度→相対角度)＞() A.屈曲ジョイント数=#.ジョイント データ配列.要素数－2－#.最初の回転可能ジョイントの番号 A.屈曲ジョイント数 ＞0 〔初期屈曲データ（A.BtoC、A.始端回転角度、A.屈曲係数、A.n 乗根、A.n 乗値）を確定〕 初期屈曲データ確定処理〔初期屈曲姿勢設定：初期屈曲データ確定処理のシートを参照のこと〕 ＜2D 初期屈曲ウェイト確定＞(A.BtoC) 〔初期屈曲姿勢を設定するジョイントの範囲（A.first、A.lastNext）を確定〕 設定範囲確定処理〔初期屈曲姿勢設定：設定範囲確定処理のシートを参照のこと〕 〔ジョイントの角度を変更〕 A.前のジョイントの回転角=0 @=#.ジョイント データ配列[A.first～A.lastNext－1]の処理 A.回転角度=A.始端回転角度－A.始端回転角度×(A.n 乗値－1.0)×@.2D 初期屈曲ウェイト値 @.2D 相対角度{演算子※1}A.回転角度－A.前のジョイントの回転角)×A.屈曲係数 A.前のジョイントの回転角=A.回転角度 A.n 乗値=A.n 乗根 #.(ソルバ共通).バインド タイプ != 直線バインド ＜2DFK 成分適用＞() ＜2D 角度制限適用＞() ＜2D 角度変換(相対角度→絶対角度)＞() ＜2D 位置更新＞() ※1 #.(ソルバ共通).バインド タイプが直線バインドの場合は「+=」演算子、そうではない場合は「=」

－	初期屈曲姿勢設定：初期屈曲データ確定処理
処理	〔初期屈曲データ（BtoC、始端回転角度、屈曲係数、n 乗根、n 乗値）を確定、 #.ジョイント データ配列[0]を始端データと定義する〕 A.BtoC=(#.コントローラの位置(IK 空間座標系)－始端データ.2D 位置(演算前)).長さ A.始端回転角度=ACos(A.BtoC，Inv(#.IK チェーンの 2D 長)) A.屈曲係数=(1.0－A.始端回転角度×Inv(PI05))×(1.0－#.(^).初期屈曲係数基礎値) +#.(^).初期屈曲係数基礎値 A.n 乗根=A.屈曲ジョイント数 ≦ #.(^).n 乗根テーブル.要素数 ？ #.(^).n 乗根テーブル[I.ジョイント数－1]： Pow(3.0，Inv(Float(A.屈曲ジョイント数))) A.n 乗値=1.0

―	初期屈曲姿勢設定：設定範囲確定処理
処理	〔初期屈曲姿勢を設定するジョイントの範囲を確定、 #.(^).CCD 固定データを CCD 固定データと定義する〕 A.first = 　CCD 固定データ.始端側適用指定 == false || 　#.最初の回転可能ジョイントの番号＞CCD 固定データ.固定数.始端側 ? 　　0：CCD 固定データ.固定数.始端側 A.lastNext=#.ジョイント データ配列.要素数－1 CCD 固定データ.終端側適用指定 == false 　A.lastNext －= CCD 固定データ.固定数.終端側

機能	2D 初期屈曲ウェイト確定	
入力	BtoC（始端からコントローラまでの長さ）	Float
処理	〔#.ジョイント データ配列[#.最初の回転可能ジョイントの番号]を支端データと定義する〕 A.StoE〔支端から終端までの 2D 長〕= 　#.IK チェーンの 2D 長－支端データ.2D 始端からの長さ A.StoA〔支端からウェイト最大位置までの 2D 長〕= 　Max(I.BtoC－支端データ.2D 始端からの長さ，A.StoE×0.5) A.BtoA〔始端からウェイト最大位置までの 2D 長〕=支端データ.2D 始端からの長さ ＋A.StoA A.InvStoE=Inv(StoE) A.ウェイト範囲=#.(ソルバ共通).初期屈曲ウェイト基礎値.＜範囲取得＞() @=#.ジョイント データ配列[#.最初の回転可能ジョイントの番号~末尾の前]の処理 　A.ウェイト最大位置までの長さ=Abs(A.BtoA－@.2D 始端からの長さ) 　A.C=1.0－(A.ウェイト最大位置までの長さ×A.InvStoE) 　@.2D 初期屈曲ウェイト値= 　　1.0 + (#.(ソルバ共通).初期屈曲ウェイト基礎値.最小値 ＋A.C³×A.ウェイト範囲)	

機能	2DFK 成分適用	
処理	#.ジョイント データ配列[*].相対 Ml を設定 #.ジョイント データ配列[*].相対 Ml の FK 成分を除去 #.ジョイント データ配列[*].FK 成分なしの 2D 初期位置を設定 #.ジョイント データ配列[*].FK 成分の 2D 相対角度を設定 #.ジョイント データ配列[*].2D 相対角度を設定	〔2DFK 成分適用：1 を参照のこと。〕 〔2DFK 成分適用：2 を参照のこと。〕 〔2DFK 成分適用：3 を参照のこと。〕 〔2DFK 成分適用：4 を参照のこと。〕 〔2DFK 成分適用：5 を参照のこと。〕

―	2DFK 成分適用：1：相対 Ml 設定
処理	A.upMg=#.IK 空間座標系 Mg @=ジョイント データ配列[0~末尾]の処理 　A.Mg=#.IK 空間座標系 Mg×@.IK 空間座標系 Ml 　A.Ml=Inv(A.upMg)×A.Mg 　@.相対 Ml=@.座標変換固定逆 Ml×A.Ml 　A.upMg=A.Mg

―	2DFK 成分適用：2：相対 Ml の FK 成分除去
処理	〔始端の FK 成分を除去〕 @=ジョイント データ配列[0]〔@は始端データ〕 A.回転 M=HPBToMatrix(－@.FK の実効角度，ROTATIONORDER::ZXYLOCAL) 始端データ.相対 Ml=A.回転 M 〔始端の次から終端までの FK 成分を除去〕 @=#.ジョイント データ配列[1~末尾]、@$^{-1}$=@の前のジョイント データの処理 　A.前の FkRot=@.座標変換固定 Ml×@$^{-1}$.FK の実効角度 　A.回転 M=HPBToMatrix(A.前の FkRot－@.FK の実効角度，ROTATIONORDER::ZXYLOCAL)

―	2DFK 成分適用：3：FK 成分なしの 2D 初期位置設定
処理	A.upMg=#.IK 空間座標系 Mg @=ジョイント データ配列[0~末尾]の処理 　A.Mg=A.upMg×@.座標変換固定 Ml×@.相対 Ml 　@.FK 成分なしの 2D 初期位置=(#.IK 空間座標系逆 Mg×A.Mg).原点 　@.FK 成分なしの 2D 初期位置.z = 0 　A.upMg=A.Mg

―	2DFK 成分適用：4：FK 成分の 2D 相対角度設定
処理	A.ジョイント相対角度=0.0 @=ジョイント データ配列[1~末尾]、@-1=@の前のジョイント データの処理 A.FK 成分なしのベクトル={正規化}←(@.FK 成分なしの 2D 初期位置－@-1.FK 成分なしの 2D 初期位置) A.FK 成分ありのベクトル={正規化}←(@.2D 位置(演算前)－@-1.2D 位置(演算前)) A.Sign=rbxSys::isRevRotXY(A.FK 成分なしのベクトル，A.FK 成分ありのベクトル) ? -1.0 : 1.0 @-1.FK 成分の 2D 相対角度= ACos(Dot(A.FK 成分なしのベクトル，A.FK 成分ありのベクトル))×A.Sign－A.ジョイント相対角度 A.ジョイント相対角度 += @-1.FK 成分の 2D 相対角度 @=ジョイント データ配列[末尾].FK 成分の 2D 相対角度=0

―	2DFK 成分適用：5：2D 相対角度設定
処理	@=ジョイント データ配列[0~末尾の前]の処理 @.2D 相対角度 += @.FK 成分の 2D 相対角度

機能	2D 角度制限適用
処理	@=ジョイント データ配列[1~末尾の前]、@-1=@の前のジョイント データの処理 @.角度制限状況 == 制限あり A.超過角度=@.2D 相対角度－ rbxSys::getSafeRotDeg(0.0，@.2D 相対角度，@.角度制限値[最小値，最大値]，0 許容値) A.超過角度＞0 許容値、 @-1.2D 相対角度 -= A.超過角度 @.2D 相対角度 += A.超過角度 ++@.連続角度超過回数

機能	CCD	
入力	強度（0.0~1.0）	Float
処理	〔CCD 実行前の 2D 相対角度を保存〕 @=ジョイント データ配列[0~末尾の前]の処理 　@.CCD 演算毎の 2D 相対角度の前値=@.2D 相対角度 〔IK チェーンの終端側から CCD 処理の 1 サイクルの処理を実施する、 #.(^).CCD 固定データを CCD 固定データと定義する〕 A.終端の 2D 位置=#.ジョイント データ配列[末尾].2D 位置(演算後) @=#.ジョイント データ配列[　末尾の前－CCD 固定データ.固定数.終端側~ 　CCD 固定データ.固定数.始端側]の処理〔当該ジョイントを Jt と定義する〕 !(#.(^).最大連続角度超過回数＞0 && @.(^).連続角度超過回数＞#.最大連続角度超過回数)、 　A.toE〔Jt から終端への 2D ベクトル〕=A.終端の 2D 位置－@.2D 位置(演算後) 　A.toC〔Jt からコントローラへの 2D ベクトル〕=#.コントローラの位置－@.2D 位置(演算後) 　A.toE 単位={正規化}←A.toE 　A.toC 単位={正規化}←A.toE 　〔Jt から終端への 2D ベクトルと Jt からコントローラへの 2D ベクトルの角度分 Jt を 2D 回転〕 　A.回転方向=rbxSys::isRevRotXY(A.toE 単位，A.toC 単位)？-1.0：1.0 　A.角度=ACos(Dot(A.toE 単位，A.toC 単位))×A.回転方向 　A.角度制限超過状況=超過なし 　A.回転角度=@.角度制限状況 == 制限なし ？A.角度×I.強度 ： 　　rbxSys::getSafeRotDeg(　　0.0, 　　@.2D 相対角度+ A.角度×I.強度, 　　@.角度制限値.[最小角度，最大角度], 　　A.角度制限超過状況, 　　0 許容値 　　)－@.2D 相対角度 　@.2D 相対角度 += A.回転角度 　@.連続角度超過回数=A.角度制限超過状況 == 超過あり ？@.連続角度超過回数+1：0 　〔終端の 2D 位置を更新〕 　Abs(A.回転角度－A.角度)≦0 許容値 　　A.終端の 2D 位置=@.2D 位置(演算後)+A.toC 　そうではない、Abs(A.回転角度)＞0 許容値 　　A.終端の 2D 位置=@.2D 位置(演算後)+MatrixRotZ(－A.回転角度)×A.toE 〔2D 相対角度に変更がある場合は 2D 絶対角度と 2D 位置を更新して処理を終了〕 @=ジョイント データ配列[0~末尾の前]の処理 　Abs(@.2D 相対角度－@.CCD 演算毎の 2D 相対角度の前値) > T.B.D 値 　　＜2D 角度変換(相対角度→絶対角度)＞() 　　＜2D 位置更新＞() 　　終了(出力値=true) 終了(出力値=flase)	
出力	処理結果（true：ジョイント角度の変更あり／false：ジョイント角度の変更なし）	Bool

機能	IK 解決をスキップ
処理	@=ジョイント データ配列[0~末尾]の処理 　@.2D 絶対角度(演算後)=@.2D 絶対角度(演算前) ＜2D 位置更新＞() ＜IK 解決演算結果設定＞()

機能	IK 解決演算結果設定
処理	A.親 Mg=Matrix(#.IK 空間座標系 Mg.原点，ジョイント データ配列[0].モデル.親 Mg.XYZ 直交軸) @=ジョイント データ配列[0~末尾]の処理 〔ジョイントの Mg を算出〕 A.回転 M=MatrixRotZ(－(@.2D 絶対角度(演算後)－@.2D 絶対角度(演算前))) A.移動 M=MatrixMove(Vector(@.2D 位置(演算後).x，@.2D 位置(演算後).y，@.IK 空間座標系 Ml.原点.z)) A.Ml=Matrix(Vector(0),IK 空間座標系 Ml.XYZ 軸) @.(演算結果データ).Mg=#.IK 空間座標系 Mg×A.移動 M×A.回転 M×A.Ml 〔ジョイントの相対角度を算出〕 A.絶対 Ml=Inv(A.親 Mg)×@.(演算結果データ).Mg A.相対 Ml=@.座標変換固定逆 Ml×A.絶対 Ml A.相対角度=rbxSys::getRotPI(MatrixToHPB(A.相対 Ml，ROTATIONORDER::HPB)) @.(演算結果データ).相対角度=rbxSys::getNearRot(A.相対角度，@.モデル.相対角度) 〔親 Mg を更新〕 A.親 Mg=@.(演算結果データ).Mg #.(^).ソルバの状態=「IK 解決済み」

機能	IK 空間座標系 Ml 確定
処理	A.親の Mg=#.IK 空間座標系 Mg @=ジョイント データ配列[0~末尾]の処理 A.Mg=A.親の Mg×@.座標変換固定 Ml×@.相対 Ml @.IK 空間座標系 Ml=#.IK 空間座標系逆 Mg×A.Mg A.親の Mg=A.Mg

機能	2D 角度変換(絶対角度→相対角度)
処理	@=#.ジョイント データ配列[0~末尾]、@$^{-1}$=@の前のジョイント データの処理 @.2D 相対角度=@.2D 絶対角度－@$^{-1}$.2D 絶対角度(演算前)※1 〔※1 #.ジョイント データ配列[－1].2D 絶対角度(演算前)は 0.0 とする〕

機能	2D 角度変換(相対角度→絶対角度)
処理	A.絶対角度=0.0 @=#.ジョイント データ配列[0~末尾]、@$^{-1}$=@の前のジョイント データの処理 A.絶対角度 += @.2D 相対角度 @.2D 絶対角度(演算後)=A.絶対角度

機能	2D 位置更新
処理	A.2D 位置=ジョイント データ配列[0].2D 位置(演算前) @=ジョイント データ配列[0~末尾]の処理 @.2D 位置(演算後)=A.2D 位置 A.2D 位置 += Cos(@.2D 絶対角度(演算後))×2D ジョイント間長 A.2D 位置 += Sos(@.2D 絶対角度(演算後))×2D ジョイント間長

機能	連続角度制限超過回数クリアー
処理	@=#.ジョイント データ配列[*] @.連続角度制限超過回数=0

7.6 3D ソルバ クラス

3D ソルバ モードの順/逆運動学演算を実現する 3D ソルバ クラス(Ik3DSolver：public IkCoreData, protected Ik3dOptionData, protected Ik3dBendFactorData)の構成を示す。

(1) 管理データ

表 7-15 3D ソルバ クラスの管理データ

番号	項目	初期値	型
1	IK 解決連続性データ（構成は 7.6.3 章を参照のこと。）	＜－＞	Ik3dConsecutiveData
2	屈曲方向（屈曲根基準：支端の Y 軸からの Z 軸回転角度）	0.0	Float
3	IK 空間座標系屈曲ベクタ	(0,0,0)	Vector
4	G 座標系屈曲ベクタ	(0,0,0)	Vector
5	IK 空間座標系 Mg	無変換 M	Matrix
6	IK 空間座標系逆 Mg	無変換 M	Matrix
7	FK の強度	1.0	Float
8	ジョイント データ型（構成は 7.6.5 章を参照のこと）	＜・＞	Ik3dJointData
9	ジョイント データ配列	要素数=0	BaseArray<Ik3dJoint Data>
10	IK チェーンの全長	0.0	Float
11	始端→支端長	0.0	Float
12	支端→終端長	0.0	Float
13	始端の G 位置	(0,0,0)	Vector
14	支端の G 位置	(0,0,0)	Vector
15	コントローラの G 位置	(0,0,0)	Vector
16	IkToC（IK 空間座標系の原点からコントローラへの単位ベクトル）	(0,0,0)	Vector
17	初期屈曲データ型	－	struct BendData
17.1	支端回転角度	－	Float
17.2	屈曲係数	－	Float
17.3	n 乗根（可動中間ジョイント数乗が 3 になる根）	－	Float
17.4	n 乗値	－	mutable Float

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-15 の初期値

機能	IK オプション データ設定	
入力	IK オプション データ	const Ik3dOptionData&
処理	#.(^)=I.IK オプション データ	

機能	IK チェーン接続	
入力	ジョイント データ配列	const BaseArray<IkJointData>&
処理	ソルバの状態を「初期」に設定 I.ジョイント データ配列.要素数＜2 || I.ジョイント データ配列.要素数＞T.B.D、終了(出力値=false) #.ジョイント データ配列.要素数 != I,ジョイント データ配列.要素数 #.ジョイント データ配列.＜領域確保＞(I.ジョイント データ配列.要素数) 領域確保 != 正常完了、終了(出力値=false) #.ジョイント データ配列[0].Mg=&IK 空間座標系 Mg @=#.ジョイント データ配列[1~末尾]、@-1=@の前のジョイント データの処理 @.親 Mg =&@-1.Mg #.(^).＜バインド タイプ確定＞(I.ジョイント データ配列) ＜IK 空間座標系確定＞(#.ジョイント データ配列[0].モデル) #.始端の G 位置=IK 空間座標系 Mg.原点の位置 ＜基本データ設定＞(I.ジョイント データ配列) ＜ジョイント間長設定＞() ＜始端→支端長と支端→終端長算出＞() #.(^).CCD 終了ステータス=初期値 #.(^).ソルバの状態=「IK チェーン接続」 出力値=true	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	屈曲因子設定	
入力	屈曲因子データ	const Ik3dBendFactorData&
処理	#.(^).ソルバの状態 == 初期、終了(出力値=false) A.変更前屈曲基準の番号=#.(^).屈曲基準の番号 #.(^).屈曲因子データ=I.屈曲因子データ #.(^).屈曲基準の番号 != A.変更前屈曲基準の番号 ＜始端→支端長と支端→終端長算出＞() ＜角度制限データ更新（全ジョイント）＞() 出力値=true	
出力	処理結果（true：正常終了／false：異常終了）	Bool

機能	前回 IK 解決時の角度設定	
入力	前回 IK 解決時の角度	const PrevBendRootRots&
処理	#.IK 解決連続性データ=I.前回 IK 解決時の角度	

機能	前回 IK 解決時の角度取得	
処理	出力値=#.IK 解決連続性データ	
出力	前回 IK 解決時の角度	const PrevBendRootRots&

<table>
<tr><td>機能</td><td colspan="2">IK 解決</td></tr>
<tr><td rowspan="4">入力</td><td>FK 成分配列</td><td>const BaseArray<Vector>&</td></tr>
<tr><td>FK の強度</td><td>Float</td></tr>
<tr><td>コントローラ</td><td>const BaseObject*</td></tr>
<tr><td>屈曲ベクタ データ</td><td>const Ik3dBendvData&</td></tr>
<tr><td>処理</td><td colspan="2">#.(^).ソルバの状態 == 初期、終了(出力値=false)
#.ジョイント データ配列.要素数 != FK 成分配列.要素数、終了(出力値=false)

#.(^).CCD 終了ステータス=初期値

#.(^).ソルバの状態 == IK 解決済み、＜連続角度制限超過回数クリアー＞()

#.コントローラの G 位置=I.コントローラ.G 位置
＜FK 成分確定＞(I.FK 成分配列)

#.(^).CCD 演算回数=-2
＜初期姿勢設定＞()
CCD 最大繰り返し回数 == -2〔初期姿勢〕
　＜IK 解決連続性データの屈曲ベクター保存＞()
　終了処理

#.(^).CCD 演算回数=-1
A.FtoC=＜照準姿勢設定＞(I.屈曲ベクタ データ)〔A.FtoC は G 座標系（支端→コントローラ〕
CCD 最大繰り返し回数 == -1〔照準姿勢〕
　＜照準姿勢の支端ツイスト＞()
　終了処理

#.(^).CCD 演算回数=0
＜初期屈曲姿勢設定＞(A.FtoC)
CCD 最大繰り返し回数 == 0〔初期屈曲姿勢〕、終了処理

CCD 繰り返し処理
〔繰り返し数の上限：#.(^).CCD 最大繰り返し回数、
　繰り返し終了条件：終端とコントローラ間の距離 ＜#.(^).CCD 閾値〕
　＜CCD＞(Min(処理回数×#.(^).CCD 平滑化係数，1.0))
　++#.(^).CCD 演算回数

終了処理
　#.(^).CCD 終了ステータスを設定
　#.(^).ソルバの状態=「IK 解決済み」
　出力値=true</td></tr>
<tr><td>出力</td><td>処理結果（true：正常終了／false：異常終了）</td><td>Bool</td></tr>
</table>

<table>
<tr><td>機能</td><td colspan="2">IK チェーン姿勢更新</td></tr>
<tr><td>入力</td><td>FK 混合比</td><td>Float</td></tr>
<tr><td>処理</td><td colspan="2">#.(^).ソルバの状態 ==「IK 解決済み」
FK 混合比別の処理
<table>
<tr><th>FK 混合比</th><th>処理</th></tr>
<tr><td>0.0
(IK+Rel FK)</td><td>@=#.ジョイント データ配列[0~末尾]の処理
　@.モデル.相対角度=@.(演算データ).相対角度</td></tr>
<tr><td>1.0
(Abs FK)</td><td>@=#.ジョイント データ配列[0~末尾]の処理
　@.モデル.相対角度=
　　@.初期姿勢の相対角度+@.FK の角度</td></tr>
<tr><td>0.0<FK 混合比<1.0
(IK+Rel FK + Abs FK)</td><td>@=#.ジョイント データ配列[0~末尾]の処理
　@.モデル.相対角度=
　　@.(演算データ).相対角度×(1.0-I.FK 混合比) +
　　(@.初期姿勢の相対角度+@.FK の角度)×I.FK 混合比</td></tr>
</table></td></tr>
</table>

機能	IK チェーン G 位置取得	
入力	IK チェーン G 位置配列	BaseArray<Vector>&
処理	#.(^).ソルバの状態 ==「IK 解決済み」 I.IK チェーン G 位置配列.要素数=#.ジョイント データ配列.要素数 @1=I.IK チェーン G 位置配列[*]、@2=#.ジョイント データ配列[*]の処理 @1 = @2.(演算データ).Mg.位置 そうでない I.IK チェーン G 位置配列.要素数=0	

機能	屈曲方向取得	
処理	出力値=#.屈曲方向	
出力	屈曲方向（支端の Y 軸からの Z 軸回転角度）	Float

機能	G 座標系屈曲ベクタ取得	
処理	出力値=#.G 座標系屈曲ベクタ	
出力	G 座標系屈曲ベクタ	Vector

機能	解決連続性データ設定	
入力	IK 解決連続性データ	const Ik3dConsecutiveData&
処理	#.IK 解決連続性データ.＜複製＞(I.IK 解決連続性データ，ジョイント データ配列.要素数)	

機能	解決連続性データ取得	
処理	出力値=#.IK 解決連続性データ	
出力	IK 解決連続性データ	const Ik3dConsecutiveData&

(3) 内部関数マップ

IK チェーン接続系	IK 解決系
IK チェーン接続 ├(ソルバ共通データ.バインド タイプ確定) ├IK 空間座標系確定 ├基本データ設定 │　├角度制限データ更新（全ジョイント） │　│　└角度制限データ更新（ジョイント指定） │　└Mg 更新 ├ジョイント間長設定 └始端→支端長と支端→終端長算出 　└ジョイント間の直線長を算出	IK 解決 ├連続角度制限超過回数クリアー ├FK 成分確定 ├初期姿勢設定 ├IK 解決連続性データの屈曲ベクター保存 ├Mg 更新 ├照準姿勢設定 │　├角度制限内でジョイントを目標に向ける │　│　└ベクトル間角度回転 │　│　　└角度制限内回転角度確定 │　│　　├B 軸回転抑制要否確認 │　│　　└（2021 年 4 月時点で非公開） │　├Mg 更新 │　├屈曲ベクタ確定 │　│　├Z 軸回転角度(Y 軸基準)からベクトルへ変換 │　│　└自動屈曲ベクタ検出 │　│　　└ベクトルから Z 軸回転角度(Y 軸基準)へ変換 │　└IK 解決連続性データの屈曲ベクター保存 ├照準姿勢の支端ツイスト ├初期屈曲姿勢設定 │　├初期屈曲データ確定 │　├初期屈曲ウェイト確定 │　├FK 成分を適用(指定範囲) │　│　└FK 成分を適用(ジョイント データ指定) │　├支端に初期屈曲姿勢の角度を設定 │　│　├角度制限内でジョイントを目標に向ける │　│　│　└ベクトル間角度回転 │　│　│　　└角度制限内回転角度確定 │　│　│　　├B 軸回転抑制要否確認 │　│　│　　└（2021 年 4 月時点で非公開） │　│　├初期屈曲姿勢の支端制御 │　│　└IkToC 更新 │　├屈曲方向へ屈曲基準の基準軸を向ける │　│　├屈曲基準軸と屈曲ベクタの角度を算出 │　│　└FK 成分を適用(ジョイント データ指定) │　├屈曲基準以降に初期屈曲姿勢の角度を設定 │　│　├角度制限内でジョイントを目標に向ける │　│　├　└ベクトル間角度回転 │　│　├　　└角度制限内回転角度確定 │　│　├　　├B 軸回転抑制要否確認 │　│　├　　└（2021 年 4 月時点で非公開） │　│　├FK 成分を適用(ジョイント データ指定) │　│　└IkToC 更新 │　└Mg 更新 └CCD 　├CCD 回転 　│　├ベクトル間角度回転 　│　│　└角度制限内回転角度確定 　│　│　　├B 軸回転抑制要否確認 　│　│　　└（2021 年 4 月時点で非公開） 　│　├CCD の支端制御 　│　├B 軸回転抑制要否確認 　│　└（2021 年 4 月時点で非公開） 　└Mg 更新

(4) 内部関数

機能	IK 空間座標系確定	
入力	始端のモデル	const BaseObject*
処理	IK 空間座標系 Mg=I.始端のモデル.親モデルの Mg IK 空間座標系 Mg.原点=I.始端のモデル.L 絶対位置 IK 空間座標系逆 Mg=Inv(IK 空間座標系 Mg)	

機能	基本データ設定	
入力	ジョイント データ配列	const BaseArray<IkJointData>&
処理	@1=#.ジョイント データ配列[*]、@2=I.ジョイント データ配列[*] @1.$=@2.* ＜角度制限データ更新（全ジョイント）＞(@1) @1.初期終端ベクタから Z 軸への角度=0.0 @1.連続角度制限超過回数=0 #.ジョイント データ配列[0].座標変換固定 Ml.off=Vector(0,0,0) #.(^).非 Z 軸配置指定 == true ＜Mg 更新＞(0，#.ジョイント データ配列.要素数－1) A.終端の G 座標=#.ジョイント データ配列.[末尾].モデル.Mg.原点 A.z=Vector(0,0,1) @=#.ジョイント データ配列[0~#.ジョイント データ配列.要素数－1] @.初期終端ベクタ={正規化}←Inv(@.Mg)×A.終端の G 座標 @.初期終端ベクタから Z 軸への角度=ACos(Dot(A.z，@.初期終端ベクタ)) @.初期終端ベクタから Z 軸への回転軸=Cross(A.z，@.初期終端ベクタ)	

機能	角度制限データ更新（全ジョイント）
処理	#.(^).屈曲根制御 != 反転優先 @=#.ジョイント データ配列[0~#.(^).屈曲基準の番号－1]の処理 @.実効角度制限データ=@.角度制限データ ＜角度制限データ更新（ジョイント指定）＞(@) @.Z 軸を固定=false そうではない @=#.ジョイント データ配列[0~#.(^).屈曲基準の番号－1－1]の処理 @.実効角度制限データ.[最小角度，最大角度]=@.モデル.相対角度 @.実効角度制限データ.[H/P/B 軸制限状況]=制限あり ＜角度制限データ更新（ジョイント指定）＞(@) @.Z 軸を固定=false @=#.ジョイント データ配列[#.(^).屈曲基準の番号－1]の処理 @.実効角度制限データ=@.角度制限データ @.実効角度制限データ.[最小角度.H，最大角度.H]=@.モデル.相対角度.H @.実効角度制限データ.[H 軸制限状況]=制限あり ＜角度制限データ更新（ジョイント指定）＞(@) @.Z 軸を固定=false @=#.ジョイント データ配列[#.(^).屈曲基準の番号~]の処理 @.実効角度制限データ=@.角度制限データ ＜角度制限データ更新（ジョイント指定）＞(@) @.Z 軸を固定=#.(^).B 軸角度固定指定

機能	角度制限データ更新（ジョイント指定）	
入力	ジョイント データ	Ik3dJointData&
処理	〔I.ジョイント データを J と定義する〕 J.角度制限状況={論理和}←J.角度制限データ.＜[H/P/B]軸制限確認＞() J.可動状況.[H/P/B]=J.角度制限データ.＜[H/P/B]軸固定確認＞() J.固定状況={反転}←J.可動状況 J.固定軸存在状況={論理和}←J.固定状況.[H/P/B]	

機能	ジョイント間長設定
処理	#.IK チェーンの全長=0.0 @=#.ジョイント データ配列[1~末尾]、@-1=@の前のジョイント データの処理 @-1.ジョイント間長の符号=@.座標変換固定 Ml.原点 ≧ 0、1.0：－1.0 @-1.ジョイント間長=@.座標変換固定 Ml.原点.原点までの長さ @-1.始端からの長さ=A.全長 #.IK チェーンの全長 += @-1.ジョイント間長 〔#.ジョイント データ配列[末尾]を終端データと定義する〕 終端データ.ジョイント間長の符号=1.0 終端データ.ジョイント間長=0.0 終端データ.始端からの長さ=A.全長

機能	始端→支端長と支端→終端長算出
処理	#.(^).バインド タイプ == 直線バインド #.始端→支端長=始端から支端までのジョイント間長の合算値 #.支端→終端長=#.IK チェーンの全長－#.始端→支端長 そうではない #.始端→支端長=始端と支端を結ぶ直線長※1 #.支端→終端長=支端と終端を結ぶ直線長※1 ※1 <ジョイント間の直線長を算出>(始点ジョイントの番号，終点ジョイントの番号)を使用する

機能	ジョイント間の直線長を算出	
入力	始点ジョイントの番号	Int32
	終点ジョイントの番号	Int32
処理	始点のジョイントの番号 > 終点のジョイントの番号、アサート A.座標変換固定 Ml=Matrix() @=#.ジョイント データ配列[I.始点ジョイントの番号+1~I.終点ジョイントの番号]の処理 A.座標変換固定 Ml = A.座標変換固定 Ml×@.座標変換固定 Ml 出力値=A.座標変換固定 Ml.原点までの長さ	
出力	始点ジョイントと終点ジョイントを結ぶ直線長	Float

機能	FK 成分確定	
入力	FK 成分配列	const BaseArray< Vector>&
処理	@1=#.ジョイント データ配列[*]、@2=I.FK 成分配列[*] @1.(FK データ).FK の角度=@2	

機能	初期姿勢設定
処理	#.(^).屈曲基準の番号 == 1〔始端と支端が同一〕 　始端と支端が同一時の始端/支端の設定〔初期姿勢設定：1 のシートを参照のこと。〕 そうではない 　始端と支端が異なる時の始端から支端までの設定〔初期姿勢設定：2 のシートを参照のこと。〕 〔屈曲基準以降の設定〕 @=#.ジョイント データ配列[#.(^).屈曲基準の番号~末尾]の処理 　@.相対角度=@.初期姿勢の相対角度 　@.相対 Ml=HPBToMatrix(@.相対角度，ROTATIONORDER::HPB)

ー	初期姿勢設定：1：始端と支端が同一時の始端/支端の設定
処理	〔#.ジョイント データ配列[0]を始端/支端データと定義する〕 始端/支端データ.相対角度=#.(^).CCD 最大繰り返し回数 == -2〔初期姿勢〕？ 　　始端/支端データ.初期姿勢の相対角度： 　　#.IK 解決連続性データ.照準姿勢の始端の角度 始端/支端データ.相対 Ml=HPBToMatrix(始端/支端データ.相対角度，ROTATIONORDER::HPB) #.支端の G 位置=始端/支端データ.親 Mg×始端/支端データ.座標変換固定 Ml×始端/支端データ.相対 Ml

ー	初期姿勢設定：2：始端と支端が異なる時の始端から支端までの設定
処理	〔始端の設定／#.ジョイント データ配列[0]を始端データと定義する〕 始端データ.相対角度=#.(^).CCD 最大繰り返し回数 == -2〔初期姿勢〕？ 　　始端/支端データ.初期姿勢の相対角度： 　　#.IK 解決連続性データ.照準姿勢の始端の角度 始端データ.相対 Ml=HPBToMatrix(始端データ.相対角度，ROTATIONORDER::HPB) 始端データ.Mg=始端データ.親 Mg×始端データ.座標変換固定 Ml×始端データ.相対 Ml 〔始端の次のジョイントから支端の前のジョイントの設定〕 @=#.ジョイント データ配列[1~#.(^).屈曲基準の番号－1－1]の処理 　@.相対角度=@.初期姿勢の相対角度 　@.相対 Ml=HPBToMatrix(@.相対角度，ROTATIONORDER::HPB) 　@.Mg=@.親 Mg×始端データ.座標変換固定 Ml×始端データ.相対 Ml 〔支端の設定／#.ジョイント データ配列[#.(^).屈曲基準の番号－1]を支端データと定義する〕 支端データ.相対角度=#.(^).CCD 最大繰り返し回数 == -2〔初期姿勢〕？ 　　支端データ.初期姿勢の相対角度： 　　#.IK 解決連続性データ.照準姿勢の支端の相対角度 支端データ.相対 Ml=HPBToMatrix(支端データ.相対角度，ROTATIONORDER::HPB) #.支端の G 位置=支端データ.親 Mg×支端データ.座標変換固定 Ml×支端データ.相対 Ml

機能	照準姿勢設定	
入力	屈曲ベクタ データ	const Ik3dBendvData&
処理	〔支端の前のジョイントまで照準姿勢の角度を設定〕 A.BtoC=#.コントローラの G 位置－#.始端の G 位置 A.目標=#.始端の G 位置+({正規化}←A.BtoC)×#.IK チェーンの全長 @=#.ジョイント データ配列[0~#.(^).屈曲基準の番号－1－1]の処理 ＜角度制限内でジョイントを目標に向ける＞(@, A.目標) @Mg=@.親 Mg×@.座標変換固定 Ml×@.相対 Ml 〔#.ジョイント データ配列[#.(^).屈曲基準の番号－1]を支端データと定義する〕 #.支端の G 位置=(支端データ.親 Mg×支端データ.座標変換固定 Ml×支端データ.相対 Ml).原点 〔支端から終端の前のジョイントに照準姿勢の角度を設定〕 A.FtoC〔支端→コントロール ベクトル（G 座標系)〕=#.コントローラの G 位置－#.支端の G 位置 A.目標=支端の G 位置+({正規化}←A.FtoC)×#.支端→終端長 @=#.ジョイント データ配列[#.(^).屈曲基準の番号－1~末尾－1]の処理 ＜角度制限内でジョイントを目標に向ける＞(@, A.目標) @.Mg=@.親 Mg×@.座標変換固定 Ml×@.相対 Ml 〔終端の Mg を更新〕 ＜Mg 更新＞(終端の番号, 終端の番号)〔終端の番号=#.ジョイント データ配列.要素数－1〕 ＜屈曲ベクタ確定＞(I.屈曲ベクタ データ) ＜IK 解決連続性データの屈曲ベクター保存＞() 〔#.ジョイント データ配列[0]を始端データと定義する〕 始端データ.＜前回値用相対角度取得＞(#.IK 解決連続性データ.照準姿勢の始端の相対角度) 支端データ.＜前回値用相対角度取得＞(#.IK 解決連続性データ.照準姿勢の支端の相対角度) 出力値=A.FtoC	
出力	支端→コントローラ ベクトル（G 座標系）	Vector

機能	照準姿勢の支端ツイスト
処理	#.(^).照準姿勢の支端ツイスト指定 == true ＜屈曲方向へ屈曲基準の基準軸を向ける＞() ＜Mg 更新＞(#.(^).屈曲基準番号, #.ジョイント データ配列.要素数－1)

機能	屈曲ベクタ確定	
入力	屈曲ベクタ データ	const Ik3dBendvData&
処理	〔I.屈曲ベクタ データを BVD と定義する〕 #.IkToC={正規化}← #.IK 空間座標系逆 Mg×((#.コントローラの G 位置－#.支端の G 位置)+#.始端の G 位置) BVD.屈曲ベクタ算出方式 == グローバル グローバル座標系屈曲ベクタの処理〔屈曲ベクタ確定：1 のシートを参照のこと。〕 そうではない〔屈曲ベクタ算出方式が屈曲根基準：手動／自動の場合〕 屈曲根基準の屈曲ベクタの処理〔屈曲ベクタ確定：2 のシートを参照のこと。〕	

〔屈曲ベクタ データの屈曲ベクタ算出方式と屈曲方向の関係について〕

屈曲ベクタ算出方式	屈曲方向
屈曲根基準：手動	屈曲方向.z が照準姿勢における支端の Y 軸からの角度（Z 軸回転角度）を表す。
屈曲根基準：自動	屈曲方向.[x,y,z]が前回の IK 解決完了時の屈曲方向を表す。（初回は Vector(0)）
体幹基準グローバル	屈曲方向.[x,y,z]が G 座標系における方向を表す。

ー	屈曲ベクタ確定：1：グローバル座標系屈曲ベクタの処理
処理	I.屈曲ベクタ データ.屈曲ベクタを固定 == 固定する A.指定屈曲ベクタ=#.IK 空間座標系逆 Mg×(IK 空間座標系 Mg.原点+BVD.屈曲方向ベクトル) 〔#.IkToC と A.指定屈曲ベクタ間の角度を 90 度に拡げた単位ベクトルを屈曲ベクタとする〕 #.IK 空間座標系屈曲ベクタ=rbxSys::getExpand90UVector(#.IkToC，A.指定屈曲ベクタ) #.G 座標系屈曲ベクタ=#.IK 空間座標系 Mg×#.IK 空間座標系屈曲ベクターIK 空間座標系 Mg.原点 そうではない #.IK 空間座標系屈曲ベクタ=#.IK 空間座標系逆 Mg×(IK 空間座標系 Mg.原点+BVD.屈曲方向ベクトル) #.G 座標系屈曲ベクタ=BVD.屈曲方向ベクトル

ー	屈曲ベクタ確定：2：屈曲根基準の屈曲ベクタの処理
処理	#.屈曲方向(支端の Y 軸からの Z 軸回転角度)=BVD.屈曲方向ベクトル.z A.デフォルト屈曲ベクトル〔支端の L 座標系〕= ＜Z 軸回転角度(Y 軸基準)からベクトルへ変換＞(#.屈曲方向〔(支端の Y 軸からの Z 軸回転角度)〕) 〔#.ジョイント データ配列[#.(^).屈曲基準の番号－1]を支端データと定義する〕 BVD.屈曲ベクタ算出方式 == 屈曲根基準：自動 〔自動屈曲ベクタ検出〕 A.FtoC={正規化}←(#.コントローラの G 位置－支端データ.Mg.原点) ＜自動屈曲ベクタ検出＞(支端データ，A.FtoC，A.デフォルト屈曲ベクトル，BVD.自動屈曲の感度，BVD.自動屈曲の強度) 自動屈曲ベクタ検出完了の場合、終了 〔自動屈曲ベクタ検知が無効、又は屈曲ベクタ検知不可の場合はデフォルト屈曲ベクトルから 屈曲ベクタを求める〕 BVD.屈曲ベクタを固定 == false A.支端 Z 回転 M=BVD.支端バンク角度因子除外 == 除外しない ？ 無変換 M： MatrixRotZ(－支端データ.相対角度.z－支端データ.初期姿勢の相対角度.z) #.IK 空間座標系屈曲ベクタ=#.IK 空間座標系逆 Mg× ((支端データ.Mg×A.支端 Z 回転 M×A.デフォルト屈曲ベクトル －支端データ.Mg.原点)+IK 空間座標系 Mg.原点) #.G 座標系屈曲ベクタ=#.IK 空間座標系 Mg×#.IK 空間座標系屈曲ベクターIK 空間座標系 Mg.原点 ＜屈曲ベクタへの禁止領域とスムージングの適用＞(BVD) そうではない #.IK 空間座標系屈曲ベクタ=#IK 解決連続性データ.IK 空間座標系屈曲ベクタ（前値） #.G 座標系屈曲ベクタ=#IK 解決連続性データ.G 座標系屈曲ベクタ（前値）

機能	Z 軸回転角度(Y 軸基準)からベクトルへ変換	
入力	Z 軸回転角度(Y 軸基準)	Float
処理	出力値=MatrixRotZ(I.Z 軸回転角度)×Vector(0.0, 1.0, 0.0)	
出力	Y 軸ベクトルを Z 軸回転角度分回転したベクトル	Vector

機能	屈曲ベクタへの禁止領域とスムージングの適用	
入力	屈曲ベクタ データ	const Ik3dBendvData&
処理	#IK 解決連続性データ.IK 空間座標系屈曲ベクタ(前値) == (0,0,0) || #IK 解決連続性データ.G 座標系屈曲ベクタ(前値) == (0,0,0)、終了 〔禁止領域の適用〕〔屈曲ベクタ データを BVD と定義する〕 BVD.屈曲ベクタ禁止領域の基準軸 != なし && BVD.屈曲ベクタ禁止領域の角度＞0 && BVD.屈曲ベクタ禁止領域侵入時の動作 != なし A.基準軸=Vector(1,0,0).＜右に回転＞(BVD.屈曲ベクタ禁止領域の基準軸÷2) BVD.屈曲ベクタ禁止領域の基準軸%2 != 0〔基準軸が X－/Y－/Z－の場合は向きを反転する〕 A.基準軸=－A.基準軸 A.基準軸と屈曲ベクタの角度=ACos(Dot(IK 空間座標系屈曲ベクタ, A.基準軸)) A.基準軸と屈曲ベクタの角度＞BVD.屈曲ベクタ禁止領域の角度 A.回転軸=BVD.屈曲ベクタ禁止領域侵入時の動作 == 停止 ？ Cross(#.IK 解決連続性データ.IK 空間座標系屈曲ベクタ(前値)，A.基準軸) : Cross(#.IK 空間座標系屈曲ベクタ，A.基準軸) 回転 M=RotAxisToMatrix(A.回転軸，A.基準軸と屈曲ベクタの角度) #.IK 空間座標系屈曲ベクタ=回転 M×A.基準軸 #.G 座標系屈曲ベクタ=IK 空間座標系 Mg×#.IK 空間座標系屈曲ベクタ－IK 空間座標系 Mg.原点 〔スムージングの適用〕 #.(^).屈曲ベクタ平滑化角度＞0 A.屈曲ベクタ変更角度=ACos(Dot(#.G 座標系屈曲ベクタ，#IK 解決連続性データ.#.G 座標系屈曲ベクタ)) A.屈曲ベクタ変更角度＞#.(^).屈曲ベクタ平滑化角度 A.回転軸=Cross(#.G 座標系屈曲ベクタ，#IK 解決連続性データ.#.G 座標系屈曲ベクタ) A.回転 M=RotAxisToMatrix(A.回転軸，#.(^).屈曲ベクタ平滑化角度) #.G 座標系屈曲ベクタ=A.回転 M×#IK 解決連続性データ.#.G 座標系屈曲ベクタ #.IK 空間座標系屈曲ベクタ=IK 空間座標系逆 Mg×(#.G 座標系屈曲ベクタ+IK 空間座標系 Mg.原点)	

機能	自動屈曲ベクタ検出	
入力	支端データ	const Ik3dJointData&
	FtoC（支端→コントローラ ベクトル）	const Vector&
	デフォルト屈曲ベクトル（支端の L 座標系）	Vector&
	自動屈曲の感度	Float
	自動屈曲の強度	Float
処理	〔支端→コントローラ ベクトルを Z 軸とする座標系（ftocZ 座標系と定義する）の XY 平面に投影した コントローラ→終端 ベクトルが屈曲ベクタ（autoBendStrength=1.0 時）となる。 但し、投影後のベクトルの長さが指定の感度に満たない場合は検出不可とする。〕 〔支端→コントローラ ベクトルを Z 軸とする座標系 M を生成〕 A.ftocM=HPBToMatrix(VectorToHPB(I.FtoC)，ROTATIONORDER::HPB) A.ftocM.原点=I.支端データ.Mg.原点 〔ftocZ 座標系にコントローラ→終端 ベクトルを投影〕 A.終端の G 位置=#.ジョイント データ配列[末尾].モデル.Mg.原点 A.XY 屈曲ベクトル=Inv(A.ftocM)×A.終端の G 位置 A.XY 屈曲ベクトル.z = 0 A.XY 屈曲ベクトルの長さ ≧ I.自動屈曲の感度、〔自動屈曲の感度確認〕 A.デフォルト屈曲ベクタ〔IK 空間座標系〕= #.IK 空間座標系逆 Mg× (I.支端データ.Mg×I.デフォルト屈曲ベクトル−(I.支端データ.Mg.原点−#.IK 空間座標系 Mg.原点)) A.屈曲ベクタ〔IK 空間座標系〕= (A.ftocM×A.XY 屈曲ベクトル−(A.ftocM.原点−#.IK 空間座標系 Mg.原点)) 〔デフォルト屈曲ベクタと屈曲ベクタの向きの差異を確認〕 A.角度差=ACos(Dot(A.屈曲ベクタ，A.デフォルト屈曲ベクタ))×I.自動屈曲の強度 A.角度差 ≧ 許容値〔1.0E-6〕 〔自動屈曲の強度に応じて IK 空間座標系屈曲ベクタを更新〕 A.回転軸=Cross(A.屈曲ベクタ，A.デフォルト屈曲ベクタ) #.IK 空間座標系屈曲ベクタ=RotAxisToMatrix(A.回転軸，A.角度差)×A.デフォルト屈曲ベクタ #.G 座標系屈曲ベクタ= #.IK 空間座標系 Mg×#.IK 空間座標系屈曲ベクタ#.IK 空間座標系 Mg.原点 I.デフォルト屈曲ベクトル={正規化}← Inv(I.支端データ.Mg)× (#.IK 空間座標系 Mg×#.IK 空間座標系屈曲ベクタ+ (I.支端データ.Mg.原点−#.IK 空間座標系 Mg.原点)) #.屈曲方向〔支端の Y 軸からの Z 軸回転角度〕= ＜ベクトルから Z 軸回転角度(Y 軸基準)へ変換＞(I.デフォルト屈曲ベクトル) 終了(出力値=true) 終了(出力値=false)	
出力	処理結果（true：検出完了／false：検出不可）	Bool

機能	ベクトルから Z 軸回転角度(Y 軸基準)へ変換	
入力	ベクトル	Vector
処理	A.回転角度=ACos(Dot(Vector(0.0, 1.0, 0.0)，I.ベクトル)) A.回転角度 < 0.001、終了(出力値=0.0) A.回転軸=Cross(Vector(0.0, 1.0, 0.0)，I.ベクトル) A.回転軸.z > 0、A.回転角度=−A.回転角度 出力値=A.回転角度	
出力	Z 軸回転角度(Y 軸基準)	Float

機能	初期屈曲姿勢設定	
入力	FtoC（支端→コントローラ ベクトル（G 座標系））	const Vector&
処理	A.ftocJc=#.ジョイント データ配列.要素数－#.(^).屈曲基準の番号－1〔支端の次～終端の前まで〕 A.ftocJc ≦ 0、終了〔屈曲不可〕 A.支端→コントローラ ベクトル長=＜初期屈曲データ確定＞(I.FtoC，A.ftocJc，A.初期屈曲データ) ＜初期屈曲ウェイト確定＞(A.支端→コントローラ ベクトル長) 〔初期屈曲姿勢を設定するジョイントの範囲を確定、 #.(^).CCD 固定データを CCD 固定データと定義する〕 A.first =CCD 固定データ.始端側適用指定 == false ? 0 : CCD 固定データ.固定数.始端側 A.lastNext=#.ジョイント データ配列.要素数－1 #.CCD 固定データ.終端側適用指定 == false、A.lastNext －= CCD 固定データ.固定数.終端側 〔IK チェーンに初期屈曲姿勢の角度を設定〕 A.lastNext < #.(^).屈曲基準の番号 〔支端以降のジョイントが固定されている場合〕 〔始端側固定ジョイントの次から終端側固定ジョイントの前のジョイントまで FK 成分を適用〕 ＜FK 成分を適用(指定範囲)＞(A.first，A.lastNext) そうではない A.first < #.(^).屈曲基準の番号 〔支端が固定されていない場合〕 〔始端側固定ジョイントの次から支端の前のジョイントまで FK 成分を適用〕 ＜FK 成分を適用(指定範囲)＞(A.first，#.(^).屈曲基準の番号－1) A.first=#.(^).屈曲基準の番号－1 ＜支端に初期屈曲姿勢の角度を設定＞(A.初期屈曲データ) A.first += 1 A.lastNext＞#.(^).屈曲基準の番号 ＜屈曲方向へ屈曲基準の基準軸を向ける＞() 〔支端の Z 軸を回転〕 そうではない 〔支端が固定されている場合〕 〔屈曲基準まで n 乗値を更新〕 @=#.(^).屈曲基準の番号－1~A.first－1 の繰り返し処理 A.初期屈曲データ.n 乗値 ×= A.初期屈曲データ.n 乗根 ＜屈曲基準以降に初期屈曲姿勢の角度を設定＞(A.first，A.lastNext，A.初期屈曲データ) 〔終端側の初期屈曲姿勢設定範囲除外部分の Mg を更新〕 ＜Mg 更新＞(A.lastNext，#.ジョイント データ配列.要素数－1)	

機能	初期屈曲データ確定	
入力	FtoC（支端→コントローラ ベクトル（G 座標系））	const Vector&
	ジョイント数（支端の次～終端の前のジョイントまで）	Int32
	初期屈曲データ（格納領域）	BendData&
処理	A.FtoC の長さ=FtoC.長さ I.初期屈曲データ.支端回転角度=ACos(A.FtoC の長さ，Inv(支端→終端長)) I.初期屈曲データ.屈曲係数= (1.0－I..初期屈曲データ.支端回転角度×Inv(PI05))×(1.0－#.(^).初期屈曲係数基礎値) +#.(^).初期屈曲係数基礎値 I.初期屈曲データ.n 乗根= I.ジョイント数 ≦ #.(^).n 乗根テーブル.要素数 ? #.(^).n 乗根テーブル[I.ジョイント数－1]： Pow(3.0，Inv(Float(I.ジョイント数))) I.初期屈曲データ.n 乗値=1.0 出力値=A.FtoC の長さ	
出力	支端→コントローラ ベクトルの長さ	Float

機能	初期屈曲ウェイト確定	
入力	FtoC 長（支端→コントローラ ベクトル長）	Float
処理	@=#.ジョイント データ配列[0~#.(^).屈曲基準の番号－1－1]の処理〔始端から支端まで〕 @.初期屈曲ウェイト値=0.0 〔ウェイト値が最大となる支端からの長さとウェイト値の範囲を確定〕 A.FtoA の長さ=Max(I.FtoC 長，#.支端→終端長×0.5) A.FtoE の逆数=Inv(#.支端→終端長) A.ウェイト値の範囲=#.(^).初期屈曲ウェイト基礎値.＜範囲取得＞() @=#.ジョイント データ配列[#.(^).屈曲基準の番号－1~末尾の前]の処理〔支端から終端の前まで〕 A.ウェイト最大位置までの長さ=Abs(A.FtoA の長さ－(@.始端からの長さ－#.始端→支端長)) A.C=1.0－(A.ウェイト最大位置までの長さ×A.FtoE の逆数) @.初期屈曲ウェイト値= 1.0+(#.(^).初期屈曲ウェイト基礎値.最小値+A.C³×A.ウェイト値の範囲)	

機能	FK 成分を適用(指定範囲)	
入力	開始番号（ジョイント データ配列のインデックス）	Int32
	停止番号（ジョイント データ配列のインデックス）	Int32
処理	@=#.ジョイント データ配列[I.開始番号~I.停止番号－1]の処理 ＜FK 成分を適用(ジョイント データ指定)＞(@) @.Mg=@.親 Mg×@.座標変換固定 Ml×@.相対 Ml	

機能	支端に初期屈曲姿勢の角度を設定	
入力	初期屈曲データ（格納領域）	const BendData&
処理	〔#.ジョイント データ配列[#.(^).屈曲基準の番号－1]を fj、I.初期屈曲データを P と定義する〕 fj.Mg=fj.親 Mg×fj.座標変換固定 Ml×fj.相対 Ml 〔支端の屈曲ベクタを確定〕 A.屈曲角度=(P.支端回転角度－P.支端回転角度×(P.n 乗値－1.0)×fj.初期屈曲ウェイト値×P.屈曲係数 A.支端の屈曲ベクタ= #.IK 空間座標系 Mg×(#.IkToC×Cos(A.屈曲角度) + #IK 空間座標系屈曲ベクタ×Sin(A.屈曲角度)) 〔屈曲ベクタを支端の原点に移動〕 A.支端の屈曲ベクタ += fj.Mg.原点－#.IK 空間座標系 Mg.原点 fj.＜外部参照相対角度接続＞(#.IK 解決連続性データ.初期屈曲姿勢の支端の相対角度) 〔支端の Z 軸を屈曲ベクタの方向へ向ける〕 ＜角度制限内でジョイントを目標に向ける＞(fj，A.支端の屈曲ベクタ) ＜初期屈曲姿勢の支端制御＞(fj) fj.＜外部参照相対角度切断＞() 〔ここでは FK 成分の適用は行わない（屈曲方向へ屈曲基準の基準軸を向ける関数で行う）〕 P.n 乗値 ×= P.n 乗根 fj.Mg=fj.親 Mg×fj.座標変換固定 Ml×fj.相対 Ml #.(^).非 Z 軸配置指定 == true、＜IkToC 更新＞()	

機能	初期屈曲姿勢の支端制御	
入力	支端データ	Ik3dJointData& fj
処理	#.(^).屈曲根制御 == 反転抑制 絶対値(支端データ.相対角度.P 軸角度)＞90 度 支端データ.相対角度=#.IK 解決連続性データ.初期屈曲姿勢の支端の相対角度(前値) 支端データ.相対 Ml=HPBToMatrix(支端データ.相対角度，ROTATIONORDER::HPB) fj.mExcess = －1	

機能	屈曲方向へ屈曲基準の基準軸を向ける
処理	〔支端の Z 軸の回転角度を確定〕 A.回転角度=#.(^).屈曲基準軸 != BendRefAxis::FREE ? ＜屈曲基準軸と屈曲ベクタの角度を算出＞() : #.(^).屈曲ツイスト角度 〔支端の Z 軸の回転角度に角度制限を適用〕 〔#.ジョイント データ配列[#.(^).屈曲基準の番号－1]を fj と定義する〕 fj.実効角度制限データ.B 軸制限状況 == 制限あり A.回転角度=rbxSys::getSafeRotDeg(fj.相対角度.z, A.回転角度, fj.実効角度制限データ.[最小角度, 最大角度].z, 0 許容値) Abs(A.回転角度) ＞ 0 許容値 fj.相対角度.z = rbxSys::getAnglePI(fj.相対角度.z+A.回転角度) fj.相対 Ml = fj.相対 Ml×MatrixRotZ(A.回転角度) ＜FK 成分を適用(ジョイント データ指定)＞(fj) 〔屈曲基準の Mg を更新〕 fj.Mg=fj.親 Mg×fj.座標変換固定 Ml×fj.相対 Ml

機能	屈曲基準軸と屈曲ベクタの角度を算出	
処理	〔屈曲基準軸ベクトルを取得〕 A.屈曲基準軸ベクトル=(#.(^).屈曲基準軸÷2) == BendRefAxis::X ? Vector(1,0,0) : Vector(0, 1, 0) (#.(^).屈曲基準軸%2) != 0、A.屈曲基準軸ベクトル×=－1 〔現在の支端の角度における屈曲基準の Mg を確定〕 〔#.ジョイント データ配列[#.(^).屈曲基準の番号]を rj と定義する〕 rj.Mg=rj.親 Mg×rj.座標変換固定 Ml×rj.相対 Ml 〔屈曲基準の座標系における屈曲ベクタを求める〕 A.屈曲ベクタ=Inv(rj.Mg)×(#.G 座標系屈曲ベクタ+rj.Mg.原点) 〔屈曲基準の座標系の XY 平面における基準軸と屈曲ベクタの角度を求める〕 A.屈曲ベクタ.z=0 A.屈曲ベクタ={正規化}←A.屈曲ベクタ A.基準軸と屈曲ベクタの角度=ACos(Dot(A.屈曲ベクタ, A.屈曲基準軸ベクトル)) Abs(A.基準軸と屈曲ベクタの角度－PI)＜0 許容値、A.基準軸と屈曲ベクタの角度=PI そうではない A.回転軸=Cross(A.屈曲ベクタ, A.屈曲基準軸ベクトル) A.回転軸.z＜0.0、A.基準軸と屈曲ベクタの角度×=－1.0 出力値=A.基準軸と屈曲ベクタの角度	
出力	屈曲基準軸と屈曲ベクタの角度（支端の Z 軸回転角度）	Float

機能	屈曲基準以降に初期屈曲姿勢の角度を設定	
入力	開始番号（ジョイント データ配列のインデックス）	Int32
	停止番号（ジョイント データ配列のインデックス）	Int32
	初期屈曲データ（格納領域）	const BendData&
処理	@=ジョイント データ配列[I.開始番号~I.停止番号－1]の処理 @.Mg=@.親 Mg×@.座標変換固定 Ml×@.相対 Ml 〔支端の屈曲ベクタを確定、I.初期屈曲データを P と定義する〕 A.屈曲角度=(P.支端回転角度－P.支端回転角度×(P.n 乗値－1.0)×fj.初期屈曲ウェイト値×P.屈曲係数 A.屈曲ベクタ= #.IK 空間座標系 Mg×(#.IkToC×Cos(A.屈曲角度) + #IK 空間座標系屈曲ベクタ×Sin(A.屈曲角度)) 〔屈曲ベクタを@の原点に移動〕 A.屈曲ベクタ+=@.Mg.原点－#.IK 空間座標系 Mg.原点 〔@の Z 軸を屈曲ベクタの方向へ向ける〕 @.＜外部参照相対角度接続＞(#.IK 解決連続性データ.初期屈曲姿勢の相対角度[@の番号]) ＜角度制限内でジョイントを目標に向ける＞(@, A.屈曲ベクタ) @.＜外部参照相対角度切断＞() ＜FK 成分を適用(ジョイント データ指定)＞(@) P.n 乗値 ×= P.n 乗根 @.Mg=@.親 Mg×@.座標変換固定 Ml×@.相対 Ml #.(^).非 Z 軸配置指定 == true、＜IkToC 更新＞()	

機能	CCD	
入力	強度	Float
処理	A.ジョイント角度変更状況=変更なし **〔IK チェーンの終端側から CCD 処理の 1 サイクルの処理を実施する、 #.(^).CCD 固定データを CCD 固定データと定義する〕** A.終端の位置〔G 座標系〕=#.ジョイント データ配列[末尾].Mg.原点 A.終端の番号=#.ジョイント データ配列.要素数－1 A.last=A.終端の番号－1－CCD 固定データ.固定数.終端側 A.first=CCD 固定データ.固定数.始端側 @=#.ジョイント データ配列[A.last~A.first]の処理〔当該ジョイントを Jt と定義する〕 〔ジョイントの連続角度超過回数が最大連続角度超過回数以上の場合は 当該ジョイントの CCD をスキップ〕 #.(^).最大連続角度超過回数＞0 && @.連続角度制限超過回数＞#.(^).最大連続角度超過回数 **〔Jt→コントローラ ベクトルと Jt→終端 ベクトルの角度を求める〕** A.JtInvMg〔Jt の逆 Mg〕=Inv(@.Mg) A.eInJt〔Jt の座標系の終端の位置〕={正規化}←A.JtInvMg×A.終端の位置 A.eInJt 単位ベクトル={正規化}←A.JtInvMg A.cInJt 単位ベクトル〔Jt の座標系のコントローラへの単位ベクトル〕= {正規化}←(A.JtInvMg×#.コントローラの G 位置) 〔終端へのベクトルとコントローラへのベクトル間の角度分、角度制限内でジョイントを回転〕 ＜CCD 回転＞(@, A.cInJt 単位ベクトル, A.eInJt 単位ベクトル, I.強度) CCD 回転でジョイントの角度が変更された場合、A.ジョイント角度変更状況=変更あり **〔終端の G 位置を更新〕** A.終端の位置〔G 座標系〕= @.親 Mg×@.座標変換固定 Ml×@.相対 Ml×A.eInJt〔Jt の座標系の終端の位置〕 A.ジョイント角度変更状況 == 変更あり ＜Mg 更新＞(A.first, A.終端の番号) 出力値=A.ジョイント角度変更状況	
出力	処理結果（true：ジョイント角度の変更あり／false：ジョイント角度の変更なし）	Bool

機能	CCD 回転	
入力	ジョイント データ	Ik3dJointData&
	目標ベクトル（ジョイントの座標系単位ベクトル）（ジョイント→コントローラ）	const Vector&
	操作ベクトル（ジョイントの座標系単位ベクトル）（ジョイント→終端）	const Vector&
	強度（0.0~1.0）	Float
処理	A.ジョイント角度変更状況=変更なし 〔I.ジョイント データを J と定義する〕 J.連続角度制限超過回数 != －1 A.相対角度(前値)=J.相対角度 ＜ベクトル間角度回転＞(J，I.目標ベクトル，I.操作ベクトル，I.強度) ベクトル間角度回転でジョイントの角度が変更された J.ジョイント番号 == #.(^).屈曲基準番号－1 ＜CCD の支端制御＞(J，A.相対角度(前値)) J.連続角度制限超過回数 != －1、A.ジョイント角度変更状況=true そうではない、A.ジョイント角度変更状況=true そうではない （2021 年 4 月時点で非公開） 出力値=A.ジョイント角度変更状況	
出力	処理結果（true：ジョイント角度の変更あり／false：ジョイント角度の変更なし）	Bool

機能	CCD の支端制御	
入力	支端データ	Ik3dJointData& fj
	相対角度(前値)	const Vector&
処理	#.(^).屈曲根制御 == 反転抑制 絶対値(支端データ.相対角度.P 軸角度)＞90 度 支端データ.相対角度=I.相対角度(前値) 支端データ.相対 Ml=HPBToMatrix(支端データ.相対角度，ROTATIONORDER::HPB) fj.mExcess = －1	

機能	Mg 更新	
入力	開始番号（設定開始ジョイント番号）	Int32
	終了番号（設定終了ジョイント番号）	Int32
処理	@=#.ジョイント データ配列[I.開始番号~I.終了番号]の処理 @.Mg=@.親 Mg×@.座標変換固定 Ml×@.相対 Ml	

機能	FK 成分を適用(ジョイント データ指定)	
入力	JtD（ジョイント データ）	Ik3dJointData&
処理	A.FK 成分の回転角度=rbxSys::roundzVector(I.JtD.FK の角度)〔0 への丸め〕×#.FK の強度 A.ジョイント角度変更状況=rbxSys::safeRotate(I.JtD.相対角度，A.FK 成分の回転角度，I.JtD.実効角度制限データ， nullptr〔参照角度なし〕，Z 軸を固定しない，0 許容値) A.ジョイント角度変更状況 == ジョイント角度変更あり I.JtD.相対 Ml=HPBToMatrix(I.JtD.相対角度，ROTATIONORDER::HPB)	

機能	角度制限内でジョイントを目標に向ける		
入力	JtD（ジョイント データ）		Ik3dJointData&
	目標（G 座標系）		const Vector&
	強度（0.0~1.0）	省略時は 1.0	Float
処理	〔操作単位ベクトルを確定（Z 軸配置の場合は Z 軸、非 Z 軸配置の場合は初期終端ベクタ〕 A.v=#.(^).非 Z 軸配置指定== false ? Vector(0.0, 1.0, 0.0) : I.JtD.初期終端ベクタ 〔当該ジョイントの座標系の目標への単位ベクトルと求める〕 A.t={正規化}←(Inv(I.JtD.親 Mg×I.JtD.座標変換固定 Ml×I.JtD.相対 Ml)×目標) 〔Z 軸ベクトルと目標へのベクトル間の角度分、角度制限内でジョイントを回転〕 ＜ベクトル間角度回転＞(I.JtD，A.t，A.v，I.強度)		

機能	ベクトル間角度回転	
入力	ジョイント データ	Ik3dJointData&
	目標ベクトル（ジョイントの座標系単位ベクトル）（ジョイント→コントローラ）	const Vector&
	操作ベクトル（ジョイントの座標系単位ベクトル）（ジョイント→終端）	const Vector&
	強度（0.0~1.0）	Float
処理	〔I.ジョイント データを@と定義する〕 A.ジョイント角度変更状況=変更なし A.角度=ACos(Dot(I.目標ベクトル，I.操作ベクトル))×I.強度 A.角度＞0 許容値 　A.回転軸=Cross(I.目標ベクトル，I.操作ベクトル) 　A.回転 M=RotAxisToMatrix(A.回転軸，A.角度) 　A.前回の相対角度参照先=@.外部参照相対角度 == nullptr ? @.相対角度 ：@.外部参照相対角度 　@.角度制限状況 == 制限なし && @.Z 軸を固定 == false 　　@.相対 Ml=@.相対 Ml×A.回転 M 　　@.相対角度=@.角度制限なし相対角度= 　　bxSys::getNearRot(　　rbxSys::getRotPI(MatrixToHPB(@.相対角度，ROTATIONORDER::HPB)), 　　A.前回の相対角度参照先) 　　A.ジョイント角度変更状況=変更あり 　そうではない 　　A.相対 Ml=@.相対 Ml×A.回転 M 　　@.角度制限なし相対角度= 　　bxSys::getNearRot(　　rbxSys::getRotPI(MatrixToHPB(@.相対角度，ROTATIONORDER::HPB)), 　　A.前回の相対角度参照先) 　　A.相対角度差=rbxSys::getRotPI(@.角度制限なし相対角度－@.相対角度) 　　＜角度制限内回転角度確定＞(@，A.相対角度差，I.目標ベクトル，I.操作ベクトル，I.強度) 　　角度制限内回転角度確定で角度変更された場合、A.ジョイント角度変更状況=変更あり そうではない 　@.角度制限なし相対角度=@.相対角度 出力値=A.ジョイント角度変更状況	
出力	処理結果（true：ジョイント角度変更あり／true：ジョイント角度変更なし）	Bool

機能	角度制限内回転角度確定	
入力	JtD（ジョイント データ）	Ik3dJointData&
	回転角度	Vector&
	目標 T（ジョイントの座標系単位ベクトル）（ジョイント→コントローラ）	const Vector&
	操作 V（ジョイントの座標系単位ベクトル）（ジョイント→終端）	const Vector&
	強度（0.0~1.0）	Float
処理	＜B 軸回転抑制要否確認＞(I.回転角度，I.目標 T，I.操作 V) B 軸回転抑制要否確認の結果 == 抑制必要 **B 軸回転抑制の処理** A.St〔ジョイント角度変更状況〕=rbxSys::safeRotate(I.JtD.相対角度，I.回転角度，I.JtD.実効角度制限データ，I.JtD.外部参照相対角度， I.JtD.Z 軸を固定，0 許容値) A.St.角度変更の有無 == ジョイント角度変更あり A.St.制限角度超過状況 == 何れかの軸で制限角度超過あり A.St.制限角度超過状況 != 全ての軸で制限角度超過あり #.(^).CCD 演算回数＞0 〔CCD フェーズ〕 **角度変更あり＆CCD フェーズの処理** そうではない、#.(^).CCD 演算回数 == 0〔初期屈曲姿勢フェーズ〕 **角度変更あり＆初期屈曲姿勢フェーズの処理** そうではない 〔初期姿勢と照準姿勢フェーズ〕 **角度変更あり＆初期姿勢と照準姿勢フェーズの処理** そうではない〔全軸制限角度超過〕 I.JtD.相対 Ml=HPBToMatrix(I.JtD.相対角度，ROTATIONORDER::HPB) そうではない 〔全軸制限角度超過なし〕 I.JtD.相対 Ml=HPBToMatrix(I.JtD.相対角度，ROTATIONORDER::HPB) そうではない 〔ジョイント角度変更なし〕 A.St.制限角度超過状況 == 何れかの軸で制限角度超過あり #.(^).CCD 演算回数 == 0〔初期屈曲姿勢フェーズ〕 **角度変更なし&初期屈曲姿勢フェーズの処理** I.JtD.連続角度制限超過回数= A.St.可動ジョイントの制限角度超過の有無 == 制限角度超過あり ？ I.JtD.連続角度制限超過回数+1：0 出力値=A.St.角度変更の有無 ※1 (B 軸回転抑制要否確認の結果 == true && #.(^).全フェーズ B 軸回転抑制指定 == true)	
出力	処理結果（true：ジョイント角度変更あり／false：ジョイント角度変更なし）	Bool

―	B 軸回転抑制（2021 年 4 月時点で非公開）
処理	

―	角度変更あり＆CCD フェーズ（2021 年 4 月時点で非公開）
処理	

―	角度変更あり＆初期屈曲姿勢フェーズ（2021 年 4 月時点で非公開）
処理	

―	角度変更あり＆初期姿勢と照準姿勢フェーズ（2021 年 4 月時点で非公開）
処理	

―	角度変更なし＆初期屈曲姿勢フェーズ（2021 年 4 月時点で非公開）
処理	

機能	B 軸回転抑制要否確認	
入力	回転角度	const Vector&
	目標ベクトル（G 座標系単位ベクトル）	const Vector&
	操作ベクトル（G 座標系単位ベクトル）	const Vector&
処理	A.Z 軸と目標ベクトルの角度=ACos(Dot(目標ベクトル, Z 軸ベクトル)) A.Z 軸と操作ベクトルの角度=ACos(Dot(操作ベクトル, Z 軸ベクトル)) 出力値=(A.Z 軸と目標ベクトルの角度＜#.(^).B 軸回転抑制角度 || A.Z 軸と操作ベクトルの角度＜#.(^).B 軸回転抑制角度) ? true : false	
出力	B 軸回転抑制要否状況（true:抑制必要/false:抑制不要）	Bool

機能	連続角度制限超過回数クリアー
処理	@=#.ジョイント データ配列[*] @.連続角度制限超過回数=0

機能	IK 解決連続性データの屈曲ベクター保存
処理	#.IK 解決連続性データ.IK 空間座標系屈曲ベクタ=#.IK 空間座標系屈曲ベクタ #.IK 解決連続性データ.G 座標系屈曲ベクタ=#.G 座標系屈曲ベクタ

機能	IkToC 更新（IK 空間座標系の原点からコントローラへの単位ベクトル更新）	
入力	JtD（ジョイント データ）	const Ik3dJointData&
処理	I.JtD.初期終端ベクタから Z 軸への角度＞0.001 度 A.G 座標系回転軸=I.JtD.Mg×I.JtD.初期終端ベクタから Z 軸への回転軸－I.JtD.Mg.原点 A.IK 空間座標系回転軸=#.IK 空間座標系逆 Mg×(A.G 座標系回転軸+#.IK 空間座標系 Mg.原点) A.回転 M=RotAxisToMatrix(A.IK 空間座標系回転軸，I.JtD.初期終端ベクタから Z 軸への角度) #.IkToC=A.回転 M×#.IkToC	

7.6.1 IK 3D ソルバ オプション データ

IK 3D ソルバ オプション データ(Ik3dOptionData：IkOptionData)の構成を示す。

(1) 管理情報

表 7-16 IK 3D ソルバ オプション データの管理データ

番号	項目	初期値	型
1	B 軸回転抑制角度（0.0=抑制なし~2.0 度 単位 ラジアン）	0.0	Float
2	屈曲ベクタ平滑化角度（0.0=平滑化なし~45.0 度 単位 ラジアン）	0.0	Float
3	非 Z 軸配置指定	false	Bool
4	全フェーズ B 軸回転抑制指定（true：全フェーズで抑制／CCD フェーズのみ抑制）	false	Bool
5	B 軸回転抑制角度を H 軸で補正指定（true：補正する／false：補正しない）	false	Bool
6	照準姿勢の支端ツイスト指定（true：ツイストする／false：ツイストしない）	false	Bool

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=の初期値

7.6.2 IK 3D ソルバ 屈曲因子データ

IK 3D ソルバ 屈曲因子データ（Ik3dBendFactorData：IkBendFactorData）の構成を示す。

(1) 管理データ

表 7-17 IK 3D ソルバ 屈曲因子データの管理データ

番号	項目	初期値	型
1	屈曲根制御（制御なし／反転抑制／反転優先）	なし	enum class FjCtrl
2	B 軸角度固定指定（屈曲基準以降のジョイント）	false	Bool

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-17 の初期値

7.6.3 IK 解決連続性データ

IK 解決連続性データ(Ik3dConsecutiveData)の構成を示す。

(1) 管理情報

表 7-18 IK 解決連続性データの管理データ

番号	項目	初期値	型
1	照準姿勢の始端の相対角度(前値)	(0,0,0)	Vector
2	照準姿勢の支端の相対角度(前値)	(0,0,0)	Vector
3	初期屈曲姿勢の支端の相対角度(前値)	(0,0,0)	Vector
4	IK 空間座標系屈曲ベクタ(前値)	(0,0,0)	Vector
5	G 座標系屈曲ベクタ(前値)	(0,0,0)	Vector
6	初期屈曲姿勢の相対角度（全ジョイント分の前値）（支端より前のジョイントは未使用） (注) [支端番号]と 1.3 の項は異なる可能性がある。（現仕様では同一）	[*]= (0,0,0)	Vector[T.B.D]

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.*=表 7-18 の初期値

機能	リセット
処理	#.*=表 7-18 の初期値

機能	姿勢データ設定	
入力	IK チェーンのジョイント データ	const BaseArray<IkCoreData::IkJointData>&
	屈曲基準番号	Int32
処理	#.照準姿勢の始端支端の相対角度= I.IK チェーンのジョイント データ[0].座標変換固定 Ml と相対角度.相対角度 #.初期屈曲姿勢の支端の相対角度=#.照準姿勢の支端の相対角度= I.IK チェーンのジョイント データ[屈曲基準番号－1].座標変換固定 Ml と相対角度.相対角度 A.ジョイント数=Min(I.IK チェーンのジョイント データ.要素数, T.B.D) @1=#初期屈曲姿勢の相対角度[0~A.ジョイント数－1]、 @2=I.IK チェーンのジョイント データ[0~A.ジョイント数－1]の処理 @1.初期屈曲姿勢の相対角度= @2.座標変換固定 Ml と相対角度.相対角度	

機能	複製	
入力	IK 解決連続性データ	const Ik3dConsecutiveData&
	ジョイント数	Int32
処理	#.$=I.[6 以外の項目] @=0~I.ジョイント数の処理 #.初期屈曲姿勢の相対角度[@]=I.IK 解決連続性データ.初期屈曲姿勢の相対角度[@]	

7.6.4 屈曲ベクタ データ

屈曲ベクタ データ(Ik3dBendvData)の構成を示す。

(1) 管理情報

表 7-19 屈曲ベクタ データの管理データ

番号	項目	初期値	型
1	屈曲ベクタ算出方式 （屈曲根基準：手動／屈曲根基準：自動／グローバル）	＜・＞	enum class Method
2	屈曲方向（屈曲ベクタ算出方式により内容が異なる） 屈曲根基準：手動／自動：z 値が支端の Y 軸からの Z 軸回転角度 グローバル：xyz 値が G 座標系ベクトル	＜・＞	Vector
3	支端バンク角度因子除外（演算因子から支端のバンク角度を除外） （true：除外する／false：除外しない）	false	Bool
4	屈曲ベクタを固定（true：固定する／false：固定しない） （グローバルの場合のみ有効）	false	Bool
5	屈曲ベクタ禁止領域の基準軸(-1:NONE/0:X+/1:X-/2:Y+/3:Y-/4:Z+/5:Z-)	-1	Int32
6	屈曲ベクタ禁止領域の角度（0=禁止領域なし~180 度）	0	Float
7	屈曲ベクタ禁止領域侵入時の動作（なし:0/停止:1/回り込み:2）	0	enum class ForbOperation
8	自動屈曲の感度（0.0=無効，0.001~10.0）	0.1	Float
9	自動屈曲の強度（0.0~1.0）	0.1	Float

(2) 外部関数

機能	デフォルト コンストラクタ
処理	無処理

機能	コンストラクタ（屈曲根基準：手動用：屈曲ベクタ禁止領域未使用）	
入力	屈曲方向（支端の Y 軸からの Z 軸回転角度）	Float
	支端バンク角度因子除外	Bool
	屈曲ベクタを固定	Bool
	屈曲ベクタ禁止領域の基準軸	Int32
	屈曲ベクタ禁止領域の角度	Float
	屈曲ベクタ禁止領域侵入時の動作	ForbOperation
処理	#.屈曲ベクタ算出方式=屈曲根基準：手動 #.屈曲方向=Vector(0.0，0.0，I.屈曲方向) #.自動屈曲の感度=自動屈曲の強度=#.0.0 #.[その他の項目]=I.$	

機能	コンストラクタ（屈曲根基準：自動用）	
入力	屈曲方向（支端の Y 軸からの Z 軸回転角度）	Float
	自動屈曲の感度	Float
	自動屈曲の強度	Float
処理	#.屈曲ベクタ算出方式=屈曲根基準：自動 #.屈曲方向=Vector(0.0，0.0，I.屈曲方向) #.自動屈曲の感度=I.自動屈曲の感度 #.自動屈曲の強度=I.自動屈曲の強度 #.[その他の項目]=表 7-19 の初期値	

機能	コンストラクタ（グローバル用）	
入力	屈曲方向（G 座標系ベクトル）	const Vector&
	屈曲ベクタを固定	Bool
処理	#.屈曲ベクタ算出方式=グローバル #.屈曲方向=I.屈曲方向 #.自動屈曲の感度=自動屈曲の強度=#.0.0 #.[その他の項目]=表 7-19 の初期値	

機能	コンストラクタ（汎用）	
入力	屈曲ベクタ算出方式（屈曲根基準：手動／屈曲根基準：自動／グローバル）	BendVMethod
	屈曲方向	const Vector& bendDir
	支端バンク角度因子除外	Bool
	屈曲ベクタを固定	Bool
	屈曲ベクタ禁止領域の基準軸	Int32
	屈曲ベクタ禁止領域の角度	Float
	屈曲ベクタ禁止領域侵入時の動作	ForbOperation
	自動屈曲の感度	Float
	自動屈曲の強度	Float
処理	#.$=#.*	

7.6.5 ジョイント データ

3D ソルバ クラス::ジョイント データ(Ik3DSolver::Ik3dJointData)の構成を示す。

(1) 管理情報

表 7-20 から表 7-22 の内容で構成する。

表 7-20 基本データ

番号	項目	初期値	型
1	ジョイント番号	－	Int32
2	モデル	nullptr	BaseObject*
3	座標変換固定 Ml	－	Matrix
4	座標変換固定逆 Ml	－	Matrix
5	初期姿勢の相対角度	－	Vector
6	初期終端ベクタ	－	Vector
7	初期終端ベクタから Z 軸への角度	－	Float
8	初期終端ベクタから Z 軸への回転軸	－	Vector
9	角度制限データ	－	VectorSpan
9.1	H 軸制限状況（true:制限あり/false:制限なし）	false	Bool
9.2	P 軸制限状況（true:制限あり/false:制限なし）	false	Bool
9.3	B 軸制限状況（true:制限あり/false:制限なし）	false	Bool
9.4	最小角度	(0,0,0)	Vector
9.5	最大角度	(0,0,0)	Vector
10	実効角度制限データ（構成は角度制限データと同じ）	－	VectorSpan
11	角度制限状況（true:何れかの軸に制限あり/false: 全軸制限なし）	－	Bool
12	Z 軸を固定（true:固定する/false:固定しない）	false	Bool
13	可動状況（H/P/B 軸の可動状況）（true:可動/false:固定）	－	rbxSys::VecStat<Bool>
14	固定状況（可動状況の逆値）	－	rbxSys::VecStat<Bool>
15	固定軸存在状況（true:固定軸あり/false:固定軸なし）	－	Bool

表 7-21 FK データ

番号	項目	初期値	型
1	FK の角度（初期姿勢からの相対角度）	－	Vector

表 7-22 演算データ

番号	項目	初期値	型
1	ジョイント間長の符号		
2	ジョイント間長	－	Float
3	始端からの長さ	－	Float
4	相対 Ml（座標変換固定からの相対）	－	Matrix
5	相対角度（座標変換固定からの相対）	－	Vector
6	角度制限なし相対角度（座標変換固定からの相対）	－	Vector
7	初期屈曲ウェイト値	－	Float
8	Mg	－	Matrix
9	親 Mg（親ジョイントの Mg）	nullptr	const Matrix*
10	外部参照相対角度（前回 IK 解決時の角度として参照）	nullptr	const Vector*
11	連続角度制限超過回数	－	Int32

(2) 外部関数

機能	デフォルト コンストラクタ
処理	#.モデル=nullptr #.Z 軸を固定=false #.外部参照する前回の相対角度=nullptr #.親 Mg=nullptr

機能	外部参照相対角度接続	
入力	相対角度（前回値領域）	const Vector&
処理	#.外部参照相対角度=&I.相対角度	

機能	外部参照相対角度切断
処理	＜前回値用相対角度取得＞(*#.外部参照相対角度) #.外部参照相対角度=nullptr

機能	前回値用相対角度取得	
入力	相対角度（今回値格納領域）	Vector&
処理	I.相対角度=#.相対角度	

8. 体制

運動制御原始モジュール 順/逆運動学演算ブロックの開発は運動制御系担当者（一人）が行う。

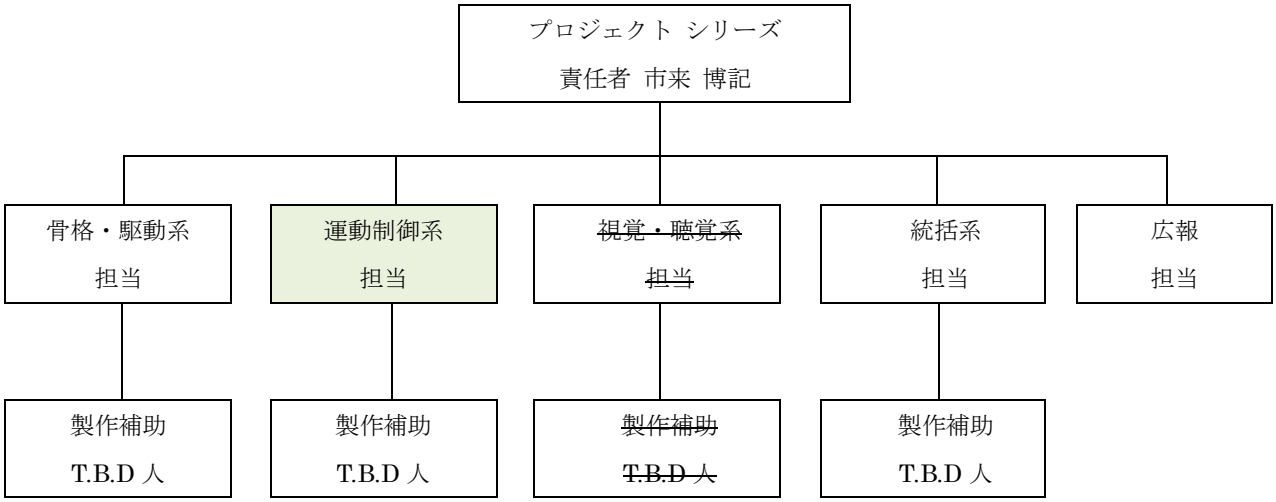

図 8-1 開発人員体制図

9. スケジュール

ロボバイオ リアルノイド オリジン – 運動制御原始モジュール開発計画書の 9 章を参照のこと。

10. 予算

非公開

11. 問題点

現時点（2019 年 3 月）で承知している問題点と対策を表 11-1 に示す。

表 11-1 運動制御原始モジュール 順/逆運動学演算ブロック開発における問題点

区分	内容	対策
姿勢選択	複数存在する IK の解から最も適した解を選択できるか？	ジョイントのウェイト値の指定の追加を検討する。 屈曲方向指定の「屈曲根基準：手動」と「グローバル」をシームレスに切り替える方法を検討する。
処理時間	常に初期姿勢からの IK 解法では処理時間が長くならないか？	現在の姿勢に対して CCD 演算を開始する「継続」モードの追加を検討する。

12. おわりに

本書 7 章に記載していないクラスの内部関数の構成を決定する際は、適切に機能分割する等して可読性・保守性の高いものとする。

ニューラルソフト有限会社

改定履歴	改　　定　　内　　容	検認	照査	作成
初期作成 19/3/31		―	―	市来 博記
B 21/12/25	文書ファイルのプロパティを設定した。	―	―	市来 博記
C 22/4/30	表紙のフォーマットを変更した。 管理番号を採番した。	―	―	市来 博記