

令和7年度 卒業論文

光学式モーションキャプチャ補助デバイスの開発

学籍番号 15611
氏名 中野晃聖
所属学科 制御情報工学科
指導教員 内堀晃彦
日付 令和8年1月30日

目次

第1章	緒言	3
1.1	背景	3
1.2	目的	3
1.3	研究方針	3
1.4	論文の構成	4
第2章	概要	5
第3章	光学式モーションキャプチャについて	6
3.1	カメラとマーカー	6
3.2	キャリブレーション	6
3.3	グランドプレーン	7
3.4	剛体トラッキング	7
第4章	慣性式モーションキャプチャ	8
4.1	回転の表し方	8
4.1.1	角速度ベクトル	8
4.1.2	オイラー角	8
4.1.3	回転行列	9
4.1.4	クオータニオン	10
4.1.5	オイラー角(ZYX)からクオータニオンへの変換	11
4.2	6軸IMUから角度を取得する基本的な計算手法	12
4.2.1	ジャイロセンサによる姿勢推定	12
4.2.2	加速度センサによる姿勢推定	13
第5章	Seeed Studio XIAO nRF52840を用いた角度の取得方法	14
5.1	キャリブレーション	14
5.2	バイアス補正, 単位変換	14
5.3	姿勢推定	15
5.3.1	高周波成分(ジャイロ)の姿勢更新	15
5.3.2	低周波成分(加速度)の姿勢更新	15
5.3.3	クオータニオン化	16
5.4	相補フィルタで融合	16
5.5	加速度の信頼性判定	16
第6章	補助デバイスの仕様	17
6.1	全体概要	17
6.2	OptiTrack eSync2	17
6.3	Seeed Studio XIAO nRF52840受信機	17
6.3.1	Bluetooth通信に関する仕様	18
6.3.2	シリアル通信に関する仕様	19
6.4	Seeed Studio XIAO nRF52840送信機	19
6.4.1	ジャイロ設定(CTRL2_G)	19
6.4.2	加速度設定(CTRL1_XL)	19
6.4.3	ジャイロフィルタ設定 (CTRL7_G)	20

6.4.4	加速度フィルタ設定(CTRL8_XL)	20
第7章	レンダリングコンピュータ	21
7.1	パケット受信	21
7.2	スケーリング変換	21
7.3	クオータニオン姿勢表現	21
7.4	ジャイロ積分	22
7.5	相補フィルタの適用	22
7.6	キャリブレーション	23
7.7	データ出力	23
第8章	可視化ツール	24
8.1	クオータニオンから回転行列への変換	24
8.2	3D表示機能	24
8.3	軸変換機能	25
8.4	キーボード操作	25
第9章	実験・評価	26
9.1	実験環境	26
9.2	評価項目	26
9.2.1	姿勢推定精度	26
9.2.2	レンダリング可能性	26
9.3	実験方法	26
9.4	結果	26
9.5	考察	27
第10章	結論	29
10.1	まとめ	29
10.2	今後の課題	29
第11章	謝辞	30
第12章	参考文献	31
第13章	付録	33
13.0.1	受信機のソースコード	33
13.0.2	送信機のソースコード	40
13.0.3	レンダリング用コンピュータで動作させる姿勢推定プログラムのソースコード	50
13.0.4	姿勢推定プログラムのライブラリ	66
13.0.5	姿勢推定プログラムの結果をプレビューするためのプログラムのソースコード	66

第1章 緒言

1.1 背景

近年、モーションキャプチャ技術は、映像制作、スポーツ科学、医療・リハビリテーション、ロボティクス、ヒューマンインターフェースなど、多様な分野において広く利用されている。人体の動作を三次元空間上で計測し、骨格モデルや三次元モデルへ反映することにより、アニメーション生成や動作の定量的な評価、訓練・治療支援の高度化が可能となる。

一方、近年の動作解析では、人体の運動だけでなく、道具操作や手作業を伴う動作の理解が重要視されている。例えば、作業支援ロボットの研究、リハビリテーション動作の評価においては、人体の動きに加え、手に持つ工具や操作対象といった小型物体の運動を同時に計測することが求められる。このような小型物体を含めた動作計測は、動作全体の因果関係や操作意図を理解する上で重要な要素である[1] [2]。

モーションキャプチャには複数の方式が存在するが、代表的なものとして光学式モーションキャプチャが挙げられる。光学式モーションキャプチャは、複数台のカメラによりマーカーを撮影し、その三次元座標を算出することで、高精度な絶対座標および姿勢情報を取得できるという利点を有する[3] [4]。しかし、小型物体を対象とした場合、十分な数のマーカーを配置することが困難であることや、手や身体による遮蔽の影響を受けやすいことから、安定した計測が困難となる。その結果、小型物体の姿勢情報が欠損しやすく、連続的な動作解析が妨げられるという課題が存在する[4]。

これに対し、慣性式モーションキャプチャは加速度センサおよびジャイロセンサを内蔵したIMUを対象に装着し、センサの情報から姿勢変化を推定する方式である。慣性式は遮蔽物の影響を受けにくく、センサを装着可能であれば連続的に姿勢変化を取得できるため、小型物体の計測に適している。一方で、角速度の時間積分による姿勢推定では誤差が蓄積しやすく、長時間計測において姿勢の信頼性が低下するという問題がある[5]。

光学式モーションキャプチャと慣性式モーションキャプチャを併用する研究事例[6] [7]は存在するものの、多くは人体動作の補完を主目的としたものであり、小型物体の動作計測を主目的として統合的に設計されたシステムは十分に検討されていない。

1.2 目的

光学式モーションキャプチャによって得られる高精度な絶対座標・姿勢情報を基準とし、慣性式モーションキャプチャによって得られる相対的な回転情報を組み合わせることで、両方式の欠点を相互に補完する統合的な計測環境を構築できれば、従来手法よりも高精度かつ安定したモーションキャプチャの実現が期待される。

本研究では、この考え方に基づき、光学式モーションキャプチャを基準とし、慣性式モーションキャプチャの手法を組み合わせた小型補助デバイスの開発を目的とする。

1.3 研究方針

本研究では、光学式モーションキャプチャシステムである OptiTrack と連携可能な小型補助デバイスを Seeed Studio XIAO nRF52840 を用いて開発する。本デバイスは小型物体への装着を想定した小型・軽量構成とし、搭載した IMU から小型物体の姿勢変化を推定する。

取得された小型物体の相対的な姿勢情報を、光学式モーションキャプチャによって得られる人体の絶対座標・姿勢情報を統合することで、小型物体を含めた動作を同一座標系上で扱う計測システムを構築する。これにより、遮蔽やマーカー制約によって光学式のみでは取得が困難であった小型物体の動作情報を補完する。

1.4 論文の構成

本論文の構成を以下に記す。

本研究で開発するシステムの全体概要とデータフローに関して、第 第2章で述べる。

光学式モーションキャプチャの原理に関して、カメラとマーカーの仕組み、キャリブレーション、グランドプレーン、剛体トラッキングの観点から第 第3章で説明する。

慣性式モーションキャプチャに必要な回転の表現方法に関して、角速度ベクトル、オイラー角、回転行列、クオータニオンの各表現法と、6軸IMUを用いた姿勢推定の基本的な計算手法について第 第4章で述べる。

Seeed Studio XIAO nRF52840を用いた角度取得の具体的な実装方法に関して、キャリブレーション、バイアス補正、姿勢推定、相補フィルタによる融合処理を第 第5章で説明する。

開発した補助デバイスの仕様に関して、OptiTrack eSync2との同期機能、Bluetooth通信およびシリアル通信の仕様を第 第6章で詳述する。

レンダリング用コンピュータ上で動作する姿勢推定プログラムの実装に関して、第 第7章で述べる。

VPythonを用いたリアルタイム3D可視化ツールに関して、第 第8章で説明する。

開発したシステムの実験および評価結果に関して、第 第9章で述べる。

本研究のまとめと今後の課題に関して、第 第10章で述べる。

第2章 概要

本研究で開発するシステムは、光学式モーションキャプチャシステム OptiTrackと、IMUを搭載した小型補助デバイスを統合した構成である。光学式モーションキャプチャから取得される絶対座標・姿勢情報を親データ、補助デバイスから取得される相対的な姿勢変化を子データとして扱い、両者を同一の時間軸上で結合することで、小型物体を含めた動作計測を実現する。

システム全体のデータフローを図 第2章.1に示す。



図 第2章.1: データフロー

補助デバイスには、3軸加速度センサおよび3軸ジャイロセンサを内蔵したSeed Studio XIAO nRF52840を採用した。本デバイスは小型・軽量であり、小型物体への装着に適している。センサから取得した角速度および加速度データに対し、相補フィルタを用いた姿勢推定処理を施すことことで、ジャイロセンサのドリフト誤差を抑制しつつ、動的な姿勢変化を追従可能とした。

OptiTrackと補助デバイス間のデータ同期には、OptiTrackのeSync2から出力される同期パルスを利用した。受信側デバイスがパルスを検知するたびにシーケンス番号をカウントアップし、このシーケンス番号を補助デバイスとの通信に付加することで、光学式と慣性式の両データを同一フレーム上で対応付ける。補助デバイスとの通信にはBluetooth Low Energy (BLE) を使用し、取得したデータはシリアル通信を介してPCへ転送される。

PC上では、OptiTrackから取得したモーションデータと補助デバイスから取得した姿勢データをシーケンス番号に基づいて統合し、Unityを用いて三次元空間上に可視化する。これにより、光学式モーションキャプチャでは取得困難であった小型物体の姿勢情報を、人体の動作と同一座標系上で表示することが可能となる。

第3章 光学式モーションキャプチャについて

3.1 カメラとマーカー

光学式モーションキャプチャは、複数台の専用カメラを用いて対象の動きをキャプチャする。光学式モーションキャプチャのカメラは、赤外線を発光するストロボライトが内蔵されている。対象に取り付けるマーカーは、入射光を光源方向へ高効率に反射する再帰反射材が用いられる。これは、光は通常乱反射するため光源に到達する光量は小さくなるため、入射する光源の方向にのみ光を返すことで、強い光量を保つためである[8]。

カメラから赤外線を発光し、反射した光をカメラで撮影することで、カメラから平面として見えるマーカーの座標が二次元座標で取得される。キャリブレーションにより、複数台のカメラの互いの座標と角度が定義され、キャリブレーション情報と各カメラの各マーカーに対する二次元座標をもとに、三次元座標を計算する。そのため、各マーカーは複数台のカメラのうち少なくとも2台以上のカメラから見えていることが必要となる[9]。

3.2 キャリブレーション

光学式モーションキャプチャは三角測量の原理を用いて座標推定を行う。ある時刻において、カメラAおよびカメラBの両方に同一のマーカーが観測されたとする。このとき、各カメラからマーカーに向かってレイAおよびレイBが空間中に投射される。理想的にはこれら2本のレイは一点で交差するが、実際には観測誤差の影響により完全には交差しない。そのため、レイAとレイBの距離が最も近くなる点をマーカーの3次元座標として推定する。しかし、この手法を適用するためには、各カメラの座標および姿勢が既知である必要がある。そこで、事前にキャリブレーションを行う[10]。

キャリブレーションとは、複数台のカメラ間の相対的な座標および姿勢、焦点距離などの内部パラメータ、ならびに空間座標系における原点および座標軸の定義を行う処理を指す。本研究で使用する OptiTrack Prime 17w は、ダイナミックキャリブレーションと呼ばれる手法によってキャリブレーションが行われる[10]。

動作範囲内でマーカーを捉えることのできる座標にカメラを複数台設置し、キャリブレーションワンドと呼ばれる3つのマーカーが一列に配置されている機器(図 第3章.1)を用いてキャリブレーションを行う。キャリブレーションワンドに設置されたマーカーには中央のマーカーと、それぞれ距離の異なるマーカーが左右に1つずつ設置されている。機器を動作範囲内の空間中で動かすことで、各カメラのマーカーに関する2D座標が記録されていく。この時取得されたデータをもとに、各カメラのマーカーに対するレイが求められる。キャリブレーションワンド上にあるマーカー間の既知の距離関係を使用し、複数時刻にわたるデータ全体に対して再投影誤差が最小となるように最適化を行うことで、3D空間上におけるカメラ同士の相対的な座標および姿勢が推定される。この手法のことをバンドル調整(Bundle Adjustment)という[11] [12] [9] (図 第3章.2)。



図 第3章.1: キャリブレーションワンド

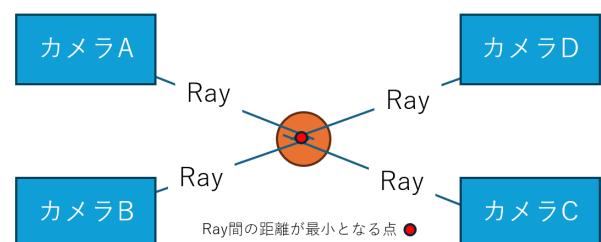


図 第3章.2: 三次元座標の求め方

3.3 グランドプレーン

グランドプレーンとは、計測空間における基準平面を定義する処理であり、通常は床面に対応する平面が設定される。グランドプレーンを設定することで、ワールド座標系の原点座標および各座標軸の向きが決定され、取得された三次元座標データを物理空間と対応づけることが可能となる[10] [9].

グランドプレーンスクエアと呼ばれる機器(図 第3章.3)を原点としたい座標に設置し、複数のマーカーの三次元座標情報を用いることで、床面を一つの平面として推定する。この平面は、各マーカー座標に最も一致するように計算される。このとき、グランドプレーンスクエア上のマーカーが示す方向をZ軸（床面に対する垂直方向）として定義される。ここで定義された原点および座標軸に基づき、キャリブレーションによって得られたカメラ間の相対座標関係が再調整される。これにより、モーションキャプチャの環境が整備される。[12].



図 第3章.3: グランドプレーンスクエア

3.4 剛体トラッキング

光学式モーションキャプチャでは、対象物体に3点以上のマーカーを取り付け、それらを一つの剛体 (Rigid Body) (図 第3章.4) として定義することで、物体の座標および姿勢を推定する手法が用いられる。

まず、各マーカーの三次元座標は複数カメラによる三角測量によって算出される。次に、マーカー間の相対座標関係が時間的に変化しないという仮定のもと、これらのマーカー群を一つの剛体として扱う。各時刻におけるマーカー配置と基準配置との対応関係から、剛体の並進および回転が推定される。これを剛体トラッキングと呼ぶ[13].

剛体が定義されたとき、どのマーカーがどの剛体に対応するかに加え、各マーカー間の距離や相対的な配置から求められる剛体の形状が保存される。それ以降は、このマーカー間の距離と剛体の形状に一致するものをフレームごとに探索することで、どのマーカーが剛体に対応するかを決定する。その後、剛体の重心を計算し、この重心をもとに各フレームごとに剛体をどれほど移動させ、回転させるべきかを計算する[14].

これにより、対象物体の座標と姿勢が推定される。

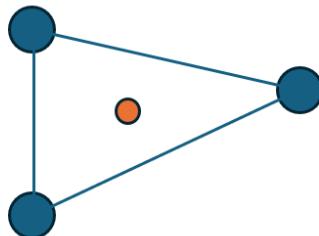


図 第3章.4: 剛体のイメージ図

第4章 慣性式モーションキャプチャ

4.1 回転の表し方

4.1.1 角速度ベクトル

角速度ベクトル $\vec{\omega}$ とは、物体の瞬間的な回転軸方向と回転角速度の大きさを同時に表すベクトル（式 第4章.1）であり、回転運動を3つの軸成分に分解して表現したものである。[15] その大きさは回転角速度を、向きは瞬間回転軸を表す（図 第4章.1）。

例えば、 $\vec{\omega} = [1, 0, 0]$ のとき、x軸周りに 1[rad/s] で回転していることを意味する。慣性式モーションキャプチャなどでは、IMUの出力データを表現する際に用いられる。

$$\vec{\omega} = [\omega_x, \omega_y, \omega_z] \text{ [rad/s]} \quad (\text{第4章.1})$$

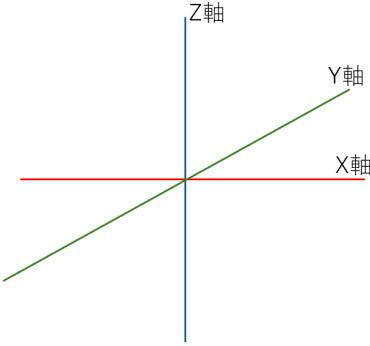


図 第4章.1: 角速度ベクトルのイメージ図

4.1.2 オイラー角

オイラー角とは、回転を3つの角度によって表現する方法であり、物体の姿勢を直感的に理解しやすいという特徴を持つ姿勢表現法である。一般に回転は、あらかじめ定められた3軸に対して順番に行われ、各回転角の組によって姿勢が定義される。

代表的な表現として Z-Y-X 系（図 第4章.2）があり、これはz軸周りの回転をYaw（図 第4章.3）、y軸周りの回転をPitch（図 第4章.4）、x軸周りの回転を Roll（図 第4章.5）とし、この順に回転を適用するものである[15]。

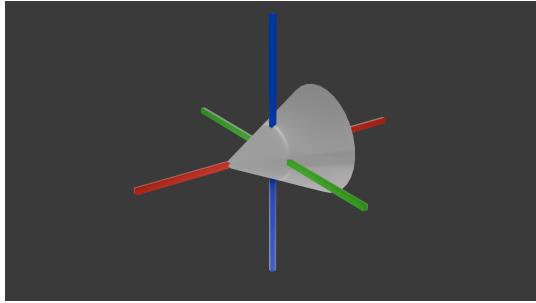


図 第4章.2: 初期状態

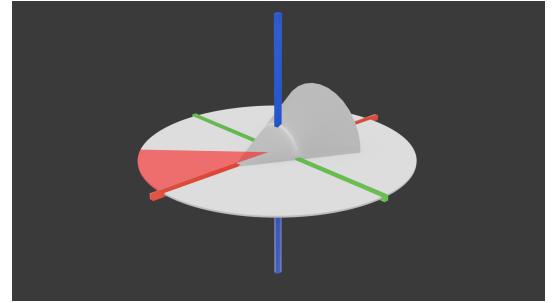


図 第4章.3: 1.Z-Yaw軸に回転

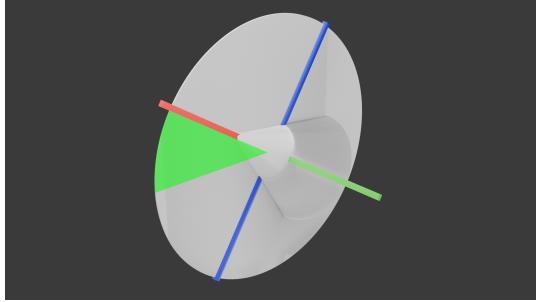


図 第4章.4: 2.Y-Pitch軸に回転

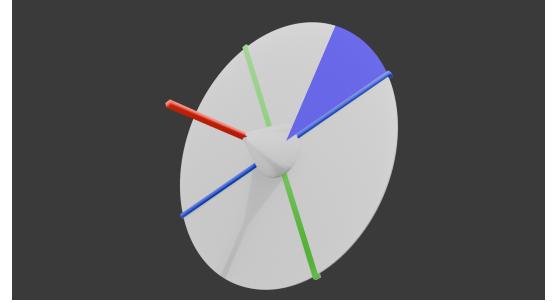


図 第4章.5: 3.X-Roll軸に回転

オイラー角は、3つのパラメータのみで姿勢を表現できるため計算量が少ないという利点を有する。一方で、特定の姿勢において自由度が失われるジンバルロックと呼ばれる問題が発生するという欠点がある。

オイラー角は

$$\Theta = (\phi, \theta, \psi) [\text{rad}]$$

として表す。ここで、 ϕ は Roll 角、 θ は Pitch 角、 ψ は Yaw 角を表す [15]。

$\Theta = (0, \frac{\pi}{2}, 0)$ のとき、Roll と Yaw の回転軸が一致し、2つの回転が独立に定義できなくなる。このように回転の自由度が1つ失われる姿勢を 特異姿勢 と呼び、この現象をジンバルロックという。

4.1.3 回転行列

回転行列とは、姿勢を線形変換として表現する方法であり、座標変換の観点から姿勢を厳密に記述できる姿勢表現法である。

回転行列は 3×3 の正方行列で表され、物体座標系から固定座標系への変換に用いられる。また、直交行列の性質より逆変換は

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

によって与えられる [15]。

回転行列 \mathbf{R} は以下の性質を満たす直交行列である（式 第4章.2）。ここで、 \mathbf{I} は単位行列を表す。

$$\mathbf{R}^T \mathbf{R} = \mathbf{I}, \quad \det(\mathbf{R}) = 1 \quad (\text{第4章.2})$$

x, y, z 各軸周りの回転行列は、右手系座標系において正方向から見て反時計回りに回転させる変換として、次のように定義される [15]。

x軸周りの回転行列（式 第4章.3）

$$\mathbf{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{pmatrix} \quad (\text{第4章.3})$$

y軸周りの回転行列（式 第4章.4）

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix} \quad (\text{第4章.4})$$

z軸周りの回転行列（式 第4章.5）

$$\mathbf{R}_z(\psi) = \begin{pmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (\text{第4章.5})$$

Z-Y-X 系オイラー角を用いる場合、回転行列は各軸回転行列の積として次式で表される（式 第4章.6）。

$$\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \quad (\text{第4章.6})$$

このとき、回転の適用順序は右から左であり、まず x-Roll 軸回転、次に y-Pitch 軸回転、最後に z-Yaw 軸回転が適用される。

回転行列自体には特異点は存在しないが、オイラー角との相互変換においては、不安定性が生じることがある。

4.1.4 クオータニオン

クオータニオンは、三次元空間における回転を4つの実数成分によって表現する姿勢表現法である（式 第4章.7）。オイラー角や回転行列と同様に物体の姿勢を表すことができるが、特異点（ジンバルロック）を持たず、数値的に安定であるという特徴を有する [15]。

$$q = w + xi + yj + zk \quad (\text{第4章.7})$$

ここで、 w は実部、 x, y, z は虚部、 i, j, k は四元数における虚数単位である。虚数単位の性質は次式で与えられる（式 第4章.8）。

$$i^2 = j^2 = k^2 = ijk = -1 \quad (\text{第4章.8})$$

クオータニオンは次のようにスカラー部とベクトル部に分けて表すことができる。

$$q = (w, \vec{v}_q), \quad \vec{v}_q = (x, y, z) \quad (\text{第4章.9})$$

回転を表す単位クオータニオンは、回転角 θ と回転軸 \vec{u} を用いて次式で表される。

$$q = \left(\cos \frac{\theta}{2}, \vec{u} \sin \frac{\theta}{2} \right) \quad (\text{第4章.10})$$

ここで、 \vec{u} は大きさ1の単位ベクトルである。

三次元空間ベクトル $\vec{a} = (a_x, a_y, a_z)$ を回転させる場合、これを実部0の純虚クオータニオン

$$v = (0, a_x, a_y, a_z)$$

として表し、次式により回転を行う。

$$v' = q v q^{-1} \quad (\text{第4章.11})$$

単位クオータニオンにおいて、回転角 θ は

$$\theta = 2 \arccos(w) \quad (\text{第4章.12})$$

より求められ、回転軸は

$$\vec{u} = \frac{(x, y, z)}{\sqrt{1 - w^2}} \quad (\text{第4章.13})$$

として得られる [15]。

このことから、クオータニオン回転は「回転軸 \vec{u} の周りに角度 θ 回転する操作」に対応していることが分かる。

さらに、クオータニオンを用いることで、球面線形補間 (SLERP) により角速度の不連続を生じることなく連続的な姿勢遷移を実現できる。

一方、クオータニオンによる回転は図 第4章.6～図 第4章.8 に示すように、一つの回転軸を基準として行われる。

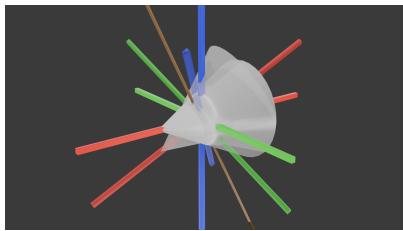


図 第4章.6: 初期状態

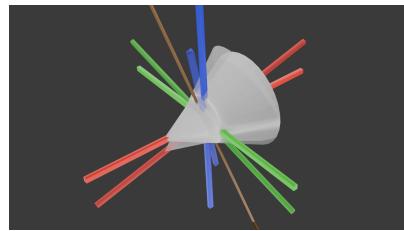


図 第4章.7: 回転途中

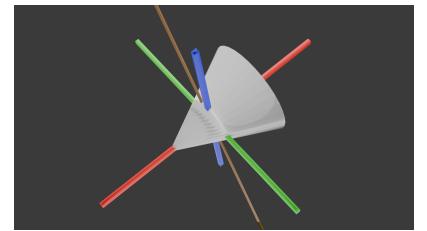


図 第4章.8: 回転終了

4.1.5 オイラー角(ZYX)からクオータニオンへの変換

今回の座標系は右手系で、回転順序をZYX(Yaw→Pitch→Roll)である。この時、YawはZ軸回り、PitchはY軸回り、RollはX軸回りである。合成回転を(式 第4章.14)とする。

$$R = R_z(\psi) R_y(\theta) R_x(\phi) \quad (\text{第4章.14})$$

半角の三角関数は(式 第4章.15),(式 第4章.16),(式 第4章.17)のように示される.

$$c_\phi = \cos\left(\frac{\phi}{2}\right), \quad s_\phi = \sin\left(\frac{\phi}{2}\right) \quad (\text{第4章.15})$$

$$c_\theta = \cos\left(\frac{\theta}{2}\right), \quad s_\theta = \sin\left(\frac{\theta}{2}\right) \quad (\text{第4章.16})$$

$$c_\psi = \cos\left(\frac{\psi}{2}\right), \quad s_\psi = \sin\left(\frac{\psi}{2}\right) \quad (\text{第4章.17})$$

この時、クオータニオンは(式 第4章.18)で求められる.

$$\begin{aligned} w &= c_\psi c_\theta c_\phi + s_\psi s_\theta s_\phi \\ x &= c_\psi c_\theta s_\phi - s_\psi s_\theta c_\phi \\ y &= c_\psi s_\theta c_\phi + s_\psi c_\theta s_\phi \\ z &= s_\psi c_\theta c_\phi - c_\psi s_\theta s_\phi \end{aligned} \quad (\text{第4章.18})$$

4.2 6軸IMUから角度を取得する基本的な計算手法

6軸IMU (Inertial Measurement Unit) は、3軸ジャイロセンサと3軸加速度センサから構成されており、角速度および並進加速度を同時に計測できる.

慣性式モーションキャプチャでは、これらの出力を統合することで各センサの姿勢推定を行う.

4.2.1 ジャイロセンサによる姿勢推定

ジャイロセンサは、各軸周りの角速度ベクトル

$$\vec{\omega}(t) = (\omega_x, \omega_y, \omega_z)$$

を出力する.

この角速度を時間積分することで姿勢の時間変化を推定できる.

クオータニオンを用いる場合、姿勢の時間微分は次式で表される.

$$\dot{q}(t) = \frac{1}{2}q(t) \otimes \Omega(t)$$

ここで、

$$\Omega(t) = (0, \omega_x, \omega_y, \omega_z)$$

は角速度を純虚クオータニオンとして表したものである.

離散時間系において、サンプリング周期を Δt とすると、姿勢更新は

$$q_{k+1} = q_k \otimes \left(\cos \frac{|\vec{\omega}_k| \Delta t}{2}, \frac{\vec{\omega}_k}{|\vec{\omega}_k|} \sin \frac{|\vec{\omega}_k| \Delta t}{2} \right)$$

として近似的に計算される。

この方法は短時間では高精度であるが、ジャイロバイアスやノイズにより時間経過とともに誤差が蓄積するドリフト問題を有する。

4.2.2 加速度センサによる姿勢推定

加速度センサは

$$\vec{a} = (a_x, a_y, a_z)$$

を出力する。

静止状態または等速直線運動時には、観測される加速度は重力加速度のみとなる。

この性質を利用することで、重力方向を基準とした姿勢推定が可能となる。

Roll角およびPitch角は次式で与えられる。

$$\phi = \arctan 2(a_y, a_z)$$

$$\theta = \arctan 2(-a_x, \sqrt{a_y^2 + a_z^2})$$

加速度センサによる姿勢推定はドリフトを生じないが、運動中には並進加速度の影響を強く受けるため、動的環境下では精度が低下する。

また、Yaw角は重力情報のみからは推定できない。

なお、ジャイロセンサは短時間(高周波成分)の精度が高いが、ドリフトが発生し得る。一方、加速度センサは長時間(低周波成分)の精度が高いが、高速な運動時に不安定となる。そのため相補フィルタを使用する必要がある。

第5章 Seeed Studio XIAO nRF52840を用いた角度の取得方法

今回使用するSeeed Studio XIAO nRF52840は、ジャイロセンサと加速度センサが搭載されている。そのため、以下のような手法で現在の角度を取得することとした。

5.1 キャリブレーション

ジャイロセンサを使用するにあたり、現在のジャイロに対するバイアスを測定し、除かなければならない。ジャイロセンサから取得できるデータを $gyro_data = (g_x, g_y, g_z)$ としたとき、バイアスは(式 第5章.1)のように計算し推定する。

$$gyro_bias = (\Sigma g_x / count, \Sigma g_y / count, \Sigma g_z / count) \quad (\text{第5章.1})$$

Seeed Studio XIAO nRF52840のサンプリングレートに関して、今回使用する光学式モーションキャプチャのサンプリングレートである120Hzに合わせる。また、ジャイロセンサのキャリブレーションに関しては2.0秒行うものとするため、理論上のサンプル数 $count$ は、240となる。ただし、通信によるデータロスなどが起こりサンプル数が10を下回った場合、十分なサンプルがなく誤ったバイアスを推定する可能性があるため、この場合はバイアスなしとして処理を行う。

Roll並びにPitchの初期姿勢を推定する。加速度センサから取得できるデータを $accel_data = (a_x, a_y, a_z)$ としたとき、加速度の平均値は(式 第5章.2)で求められる。

$$accel_mean = (\Sigma a_x / count, \Sigma a_y / count, \Sigma a_z / count) \quad (\text{第5章.2})$$

初期姿勢は(式 第5章.3)、(式 第5章.4)、(式 第5章.5)で求められる。

$$roll_0 = \arctan 2(accel_mean_y, accel_mean_z) \quad (\text{第5章.3})$$

$$pitch_0 = \arctan 2(-accel_mean_x, \sqrt{accel_mean_y^2 + accel_mean_z^2}) \quad (\text{第5章.4})$$

$$yaw_0 = 0 \quad (\text{第5章.5})$$

なお、 $\arctan 2(y, x)$ は $-\pi < \theta \leq \pi$ の範囲で $x = r \cos \theta$, $y = r \sin \theta$, $r = \sqrt{x^2 + y^2}$ (ただし $r > 0$)となる単一の値 θ を返す関数である。

なお、この時Seeed Studio XIAO nRF52840は停止しているものとし、サンプル数 $count$ はジャイロセンサのキャリブレーションと同様のものであるとする。

5.2 バイアス補正、単位変換

バイアスを毎回除去するとき(式 第5章.6)のように行なう。

$$gyro_current = (g_x - gyro_bias.x, g_y - gyro_bias.y, g_z - gyro_bias.z) \quad (\text{第5章.6})$$

また、単位を変換させるとき(式 第5章.7)のように計算する。

$$\omega = gyro_current \cdot \frac{\pi}{180} [\text{rad/s}] \quad (\text{第5章.7})$$

5.3 姿勢推定

5.3.1 高周波成分(ジャイロ)の姿勢更新

ジャイロセンサから取得できるデータを $\frac{1}{\text{サンプリングレート}}$ で更新する。まず、角速度から微小回転角を(式 第5章.8)で計算する。

$$\Delta\vec{\theta} = \vec{\omega} \cdot dt \quad (\text{第5章.8})$$

回転角の大きさは(式 第5章.9)で求められる。

$$\theta = \sqrt{(\Delta\theta_x)^2 + (\Delta\theta_y)^2 + (\Delta\theta_z)^2} \quad (\text{第5章.9})$$

回転軸ベクトルは(式 第5章.10)で求められる。

$$\vec{u} = \frac{1}{\theta} \begin{bmatrix} \Delta\theta_x \\ \Delta\theta_y \\ \Delta\theta_z \end{bmatrix} \quad (\text{第5章.10})$$

微小回転クオータニオンは(式 第5章.11)で定義される。

$$dq = \begin{bmatrix} \cos(\theta/2) \\ u_x \sin(\theta/2) \\ u_y \sin(\theta/2) \\ u_z \sin(\theta/2) \end{bmatrix} \quad (\text{第5章.11})$$

高周波成分のクオータニオンは(式 第5章.12)で得られる。

$$q_{\text{gyro}} = \text{normalize}(q_{\text{current}} \otimes dq) \quad (\text{第5章.12})$$

5.3.2 低周波成分(加速度)の姿勢更新

加速度から取得できるデータを $\frac{1}{\text{サンプリングレート}}$ で更新する。Roll角は(式 第5章.13), Pitch角は(式 第5章.14), Yaw角は(式 第5章.15)で計算される。

$$roll_accel = \arctan 2(a_y, a_z) \quad (\text{第5章.13})$$

$$pitch_accel = \arctan 2(-a_x, \sqrt{a_y^2 + a_z^2}) \quad (\text{第5章.14})$$

$$yaw_accel = yaw_gyro \quad (\text{第5章.15})$$

5.3.3 クォータニオン化

加速度によって得られるオイラー角をクォータニオンへ(式 第5章.16)変換する.

$$q_{\text{accel}} = \text{Quaternion}(roll_{\text{accel}}, pitch_{\text{accel}}, yaw_{\text{accel}}) \quad (\text{第5章.16})$$

5.4 相補フィルタで融合

高周波の場合はジャイロセンサからのデータを、低周波の場合は加速度センサからのデータを採用するため(式 第5章.17)，以下のようなフィルタで融合させる(図 第5章.1). 今回 $\alpha = 0.98$ とする.

$$q = \text{nlerp}(q_{\text{gyro}}, q_{\text{accel}}, t = 1 - \alpha) \quad (\text{第5章.17})$$

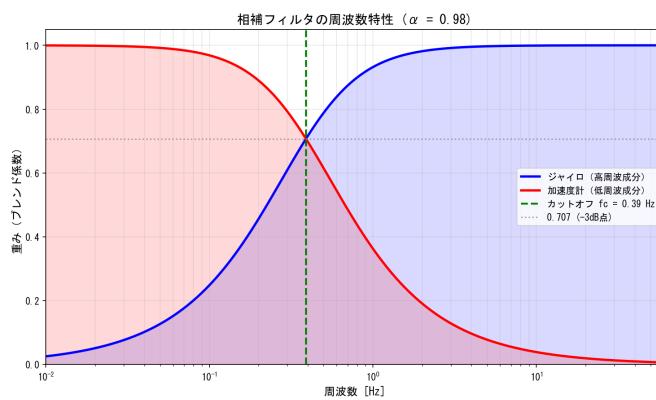


図 第5章.1: 相補フィルタの周波数特性

5.5 加速度の信頼性判定

加速度センサから得られる姿勢情報は、静止時または等速直線運動時には重力方向を正しく示すが、動的な運動中には並進加速度の影響を受け信頼性が低下する。そこで、加速度のノルムを用いた信頼性判定を行う。加速度ノルムは(式 第5章.18)で計算される。

$$\|\vec{a}\| = \sqrt{a_x^2 + a_y^2 + a_z^2} \quad (\text{第5章.18})$$

$0.5 < \|\vec{a}\| < 1.5$ [g]のとき加速度データを使用し、そうでない場合はジャイロのみで姿勢更新を行う。

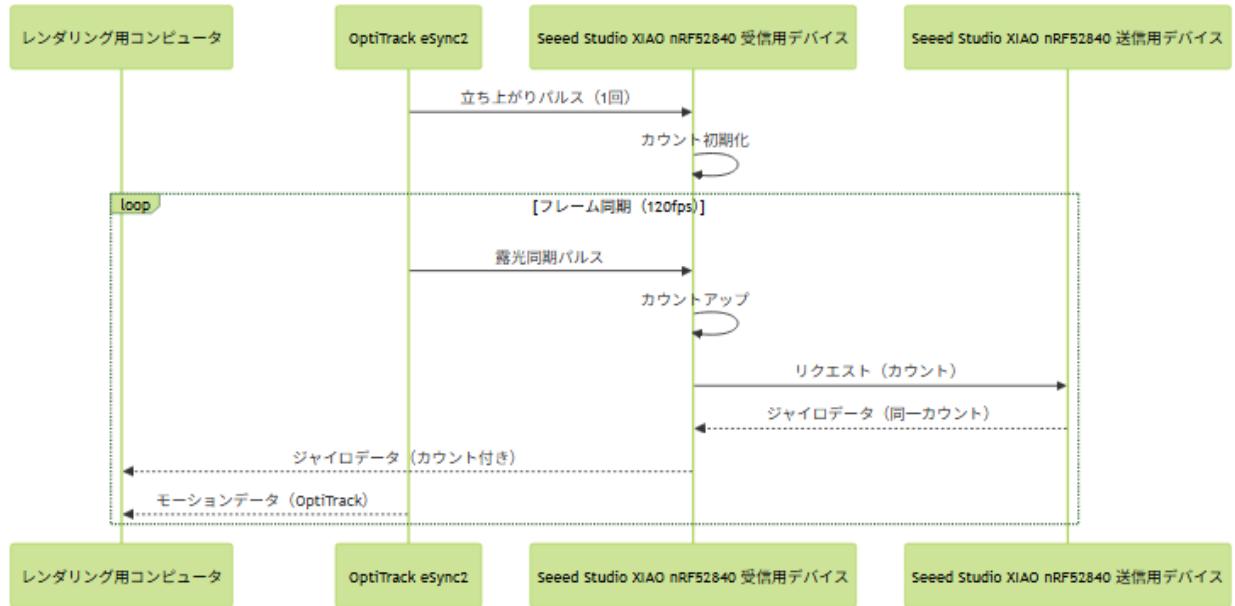
なお、静止時の加速度ノルムは重力加速度の1.0[g]となる。しきい値を0.5~1.5[g]とすることで、急激な運動や衝撃が加わった際には加速度センサの情報を無視し、ジャイロセンサのみによる姿勢更新を行う。これにより、動的環境下における姿勢推定の安定性を向上させる。

以上の処理により、120HzでSeeed Studio XIAO nRF52840の現在の姿勢を推定することができる。

第6章 補助デバイスの仕様

今回使用しているOptiTrackとSeeed Studio XIAO nRF52840は別のデバイスで、同期機能を実装しなければ収録されたデータを同期して再生することができないため、以下のように同期機能を実装した。

6.1 全体概要



6.2 OptiTrack eSync2

eSync2とは、OptiTrackと連携しトリガー信号を入出力するためのインターフェースのことである[16]。今回、OptiTrackに補助デバイスを同期させるため、トリガー信号の出力を受け付けることとする。eSync2では、カメラが露光したタイミング(マーカーの情報を取得したタイミング)でパルスを出力する[17]。このパルスを補助デバイスの受信機で受け取ることで、OptiTrackとの同期を行う。

6.3 Seeed Studio XIAO nRF52840受信機

今回、キャプチャ対象へつけるSeeed Studio XIAO nRF52840(以降送信用デバイス)とは別に、送信用デバイスのデータを受け取るSeeed Studio XIAO nRF52840(以降受信用デバイス)を開発する。理由として、送信用デバイスが無線接続でデータを送れる必要があること、コンピュータのOSがWindowsの場合Bluetooth通信を行う際の制限があることが挙げられる。

まず、モーションキャプチャ中は範囲内を大きく動くことが多く、今回対象としている物体などにデバイスを有線で接続した場合、行動範囲が非常に限られるため、無線接続でデータをレンダリング用コンピュータへ送信できなければならない。今回採用したSeeed Studio XIAO nRF52840は、Bluetooth接続が可能なため、これを採用することとした。

Seeed Studio XIAO nRF52840を使用してWindowsコンピュータへBluetooth接続を行った際、接続間隔やMTU等のプロパティを細かく制御できないという問題や、遅延が大きく出るという問題があった。加えて、OptiTrackからのパルスを受け取り送信用デバイスへ送信しなければならないため、受信用デバイスを開発することになった。

受信用デバイスの主な役割は、シーケンス管理を行いながら、送信用デバイスへデータのリクエストを行い、その後送信されてくるデータをシリアル通信でレンダリング用コンピュータへ送信することである。

6.3.1 Bluetooth通信に関する仕様

今回、UUIDを表 第6章.1のように定めた。

表 第6章.1: Bluetooth接続における必要なUUID

項目	UUID
データサービス	0x1234
データCharacteristic	0x5678
リクエストCharacteristic	0x5679

通信パラメータを以下のように設定した。
- MTU : 50バイト（交換要求）
- 接続間隔 : 6（最小接続間隔）
- スキャン間隔 : 160 (100ms)
- スキャンウィンドウ : 80 (50ms)

パケットに関して、表 第6章.2のようにし通信を行った。

表 第6章.2: Bluetooth通信におけるパケットの構成(24byte)

オフセット	サイズ	内容	フィールド名
0-1	2	ヘッダ (0x55, 0xAA)	header
2-5	4	シーケンス番号	seq
6-9	4	リクエストシーケンス番号	requestSeq
10-11	2	ジャイロX (int16)	gx
12-13	2	ジャイロY (int16)	gy
14-15	2	ジャイロZ (int16)	gz
16-17	2	加速度X (int16)	ax
18-19	2	加速度Y (int16)	ay
20-21	2	加速度Z (int16)	az
22-23	2	チェックサム	checksum

今回チェックサムはXORチェックサムを採用した。具体的な計算方法を以下に示す。

```
checksum = header
    ^ (seq & 0xFFFF) ^ ((seq >> 16) & 0xFFFF)
    ^ (requestSeq & 0xFFFF) ^ ((requestSeq >> 16) & 0xFFFF)
    ^ gx ^ gy ^ gz
    ^ ax ^ ay ^ az
```

また、データサイズ削減のため、使用したIMUライブラリの出力であるfloatをint16_t型に変換しデータを送信することとした。ジャイロセンサのスケーリング係数は16.384、加速度センサのスケーリング係数は8192である。これにより、int16_tの範囲に最大レンジをマッピングでき、データサイズを効率化することができる。

6.3.2 シリアル通信に関する仕様

シリアル通信に関して、ボーレートを921600bps、出力形式をバイナリとした。データ構造に関して、受信したデータを加工せず、受信したパケットをそのままレンダリング用コンピュータへ送信している。

6.4 Seeed Studio XIAO nRF52840送信機

受信機からシーケンス番号とともにリクエストが送られてくるため、受信したタイミングでセンサの値を読み取り、パケットを作成してBluetoothで送信する。

マイコンボードに搭載されているIMUセンサはLSM6DS3である。このセンサから、I2C通信によってデータを取得し、マイコンボード上で処理を行うこととなる。

なお、LSM6DS3に対し、以下のようなレジスタ設定を行っている。

6.4.1 ジャイロ設定(CTRL2_G)

```
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL2_G, 0x8C);
```

表 第6章.3: CTRL2_Gのレジスタ構成

bit	名称	内容
0-1	-	未使用
2-3	FS_G	フルスケール
4-7	ODR_G	ジャイロ出力データレート

0x8C=1000 1100

ODR_G=1000

FS_G=11

より、ジャイロセンサを有効にし、出力データレートを1.66kHz、測定レンジを±2000 dpsと設定している[18]。

6.4.2 加速度設定(CTRL1_XL)

```
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL1_XL, 0x8A);
```

表 第6章.4: CTRL1_XLのレジスタ構成

bit	名称	内容
0-1	BW_XL	帯域設定
2-3	FS_XL	フルスケール
4-7	ODR_XL	加速度出力データレート

0x8A=1000 1010

ODR_XL=1000

FS_XL=10

BW_XL=10

より、加速度センサを有効にし、出力データレートを1.66kHz、測定レンジを±4 g、内部ローパスフィルタ(400Hz)を有効に設定している。このローパスフィルタは、ノイズに対する対策のものである[18]。

6.4.3 ジャイロフィルタ設定 (CTRL7_G)

```
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL7_G, 0x00);
```

ジャイロに対する追加のフィルタは設定していない。

6.4.4 加速度フィルタ設定(CTRL8_XL)

```
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL8_XL, 0x09);
```

表 第6章.5: CTRL8_XLのレジスタ構成

bit	名称	内容
0-2	LOW_PASS_ON_6D	LPF設定
3	INPUT_COMPOSITE	データ合成
4	HP_REF_MODE	参照モード
5-6	HPCF_XL	HPFカットオフ
7	LPF2_XL_EN	LPF2有効

0x09=0000 1001

INPUT_COMPOSITE=1

LOW_PASS_ON_6D=1

より、加速度信号にLPF2を適用している[18]。

このように、センサ側でフィルタ用いてノイズに対する対策を遅延が発生しないよう低遅延で行っている。

第7章 レンダリングコンピュータ

受信機からシリアル通信で送られてきたデータをクオータニオン化し、各プログラムへと送信する。本システムでは、Rust言語を使用し、クオータニオン化処理を高速で行う。

7.1 パケット受信

受信機からは921600bpsのシリアル通信でIMUデータが送信される。パケット構造は表 第7章.1の通りである。

表 第7章.1: パケット構造 (24バイト)

オフセット	サイズ	内容
0-1	2バイト	ヘッダ (0x55, 0xAA)
2-5	4バイト	シーケンス番号
6-9	4バイト	リクエストシーケンス番号
10-15	6バイト	ジャイロ (X, Y, Z) 各2バイト
16-21	6バイト	加速度 (X, Y, Z) 各2バイト
22-23	2バイト	予約

パケットはヘッダ0x55, 0xAAで同期を取り、パケット境界を検出する。ヘッダ同期にはタイムアウト処理を設け、1秒以内にヘッダが検出できない場合はパケットを破棄する。

7.2 スケーリング変換

受信した生データは以下の式でSI単位系に変換する。

ジャイロの角速度変換は式 第7章.1で行う。

$$\omega_{dps} = \frac{raw}{16.384} \quad (\text{第7章.1})$$

ここで ω_{dps} は角速度[deg/s], raw は受信した16ビット符号付き整数である。スケールファクタ16.384はLSM6DS3のジャイロスコープ設定±2000 dpsに対応する。

加速度変換は式 第7章.2で行う。

$$a_g = \frac{raw}{8192} \quad (\text{第7章.2})$$

ここで a_g は加速度[G]である。スケールファクタ8192は加速度センサ設定±4Gに対応する。

7.3 クオータニオン姿勢表現

姿勢はクオータニオン $q = (w, x, y, z)$ で表現する。クオータニオンはオイラー角と比較して以下の利点がある。

- ・ジンバルロックが発生しない
- ・補間計算が容易
- ・計算効率が良い

クオータニオンの積は式 第7章.3で計算する.

$$q_1 \otimes q_2 = \begin{pmatrix} w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2 \\ w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2 \\ w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2 \\ w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2 \end{pmatrix} \quad (\text{第7章.3})$$

7.4 ジャイロ積分

ジャイロの角速度($\omega_x, \omega_y, \omega_z$)[rad/s]から微小回転クオータニオンを生成し、現在の姿勢に乗算することで姿勢を更新する。

まず角速度ベクトルの大きさを計算する。

$$\theta = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \cdot dt \quad (\text{第7章.4})$$

ここで dt はサンプリング周期である。回転軸の単位ベクトルは以下となる。

$$(u_x, u_y, u_z) = \frac{1}{\theta/dt}(\omega_x, \omega_y, \omega_z) \quad (\text{第7章.5})$$

微小回転クオータニオン Δq は式 第7章.6で計算する。

$$\Delta q = \left(\cos \frac{\theta}{2}, u_x \sin \frac{\theta}{2}, u_y \sin \frac{\theta}{2}, u_z \sin \frac{\theta}{2} \right) \quad (\text{第7章.6})$$

姿勢の更新は式 第7章.7で行う。

$$q_{new} = q_{current} \otimes \Delta q \quad (\text{第7章.7})$$

7.5 相補フィルタの適用

ジャイロ積分のみでは長期的にドリフトが蓄積するため、加速度計からの姿勢推定と融合する相補フィルタを適用する。

加速度から推定したロール・ピッチ角と、ジャイロから推定したヨー角を組み合わせてクオータニオン q_{acc} を生成する。ジャイロ積分で得たクオータニオン q_{gyro} と正規化線形補間(nlerp)で融合する。

$$q_{fused} = nlerp(q_{gyro}, q_{acc}, 1 - \alpha) \quad (\text{第7章.8})$$

ここで $\alpha = 0.98$ である。この値により、短期的にはジャイロの応答性を維持しつつ、長期的には加速度計でドリフトを補正する。

加速度の大きさが $0.5G < |a| < 1.5G$ の範囲外の場合は動的加速度が加わっていると判断し、相補フィルタの補正を行わずジャイロ積分のみを使用する。

7.6 キャリブレーション

起動時に2秒間のキャリブレーションを実行する。キャリブレーション中はセンサを静止させ、以下の処理を行う。

1. ジャイロバイアスの推定：静止状態での角速度出力を平均化し、バイアスとして記録する。以降の計測値からこのバイアスを減算する。
2. 初期姿勢の推定：加速度から重力方向を検出し、初期のロール・ピッチ角を決定する。ヨー角は0に設定する。

キャリブレーションはプログラム実行中にも標準入力またはUDP通信で”CALIBRATE”コマンドを送ることで再実行できる。

7.7 データ出力

姿勢データは2つの形式で出力する。

1. 標準出力：クォータニオン形式で出力する。フォーマットは以下の通りである。

DATA_Q,seq,request_seq,qw,qx,qy,qz

2. UDP通信：Unityなどのレンダリングアプリケーション向けに、回転行列のX軸・Y軸ベクトルを送信する。送信先は127.0.0.1:50005である。

DATA,seq,request_seq,ex_x,ex_y,ex_z,ey_x,ey_y,ey_z

回転行列のX軸・Y軸ベクトルから、受信側でZ軸ベクトルを外積で復元し、完全な回転行列を再構成できる。この形式はUnityのTransform.LookAt()などで直接利用可能である。

第8章 可視化ツール

本システムでは、姿勢推定結果を視覚的に確認するため、VPythonを使用したリアルタイム3D可視化ツールを開発した。本ツールはRustプログラムをサブプロセスとして起動し、標準出力から受信したクオータニオンデータを3D表示する。

8.1 クオータニオンから回転行列への変換

クオータニオン $q = (q_w, q_x, q_y, q_z)$ から回転行列 R への変換は(式 第8章.1)で行う。

$$R = \begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{pmatrix} \quad (\text{第8章.1})$$

この回転行列を用いて、デバイスのローカル座標系における単位ベクトルをワールド座標系へ変換し、3Dオブジェクトの向きを更新する。

8.2 3D表示機能

可視化ツールでは、以下の要素を画面上に表示する。実際の画面を図 第8章.1に示す。

1. 3Dボックス: デバイスの向きを示す青色半透明の直方体
2. 座標軸矢印: X軸（赤）、Y軸（緑）、Z軸（青）の矢印
3. 姿勢情報: Roll, Pitch, Yaw角度をリアルタイムで数値表示
4. シーケンス番号: 内部シーケンス番号とOptiTrack同期パルス番号を表示

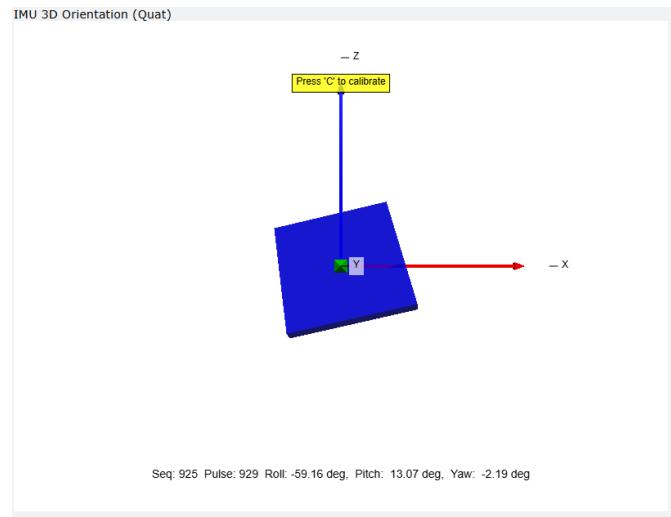


図 第8章.1: VPythonによる可視化画面

表示更新レートは120Hzとし、センサのサンプリングレートと一致させている。

8.3 軸変換機能

センサ座標系と表示座標系が異なる場合に対応するため、軸の入れ替えおよび符号反転機能を実装した。軸変換は(式 第8章.2)により行う。

$$\vec{v}_{display} = \begin{pmatrix} v_{remap[0]} \cdot sign[0] \\ v_{remap[1]} \cdot sign[1] \\ v_{remap[2]} \cdot sign[2] \end{pmatrix} \quad (\text{第8章.2})$$

ここで、*remap*は軸の入れ替え順序、*sign*は各軸の符号を指定する。

8.4 キーボード操作

可視化ツールでは、キーボードによる操作を受け付ける。`'C'`キーを押下することで、Rustプロセスへ”CALIBRATE”コマンドを送信し、キャリブレーションを再実行できる。キャリブレーション中は画面上のラベルが変化し、完了時に通知される。

第9章 実験・評価

9.1 実験環境

本研究で使用した実験環境を以下に示す.

- ・光学式モーションキャプチャ : OptiTrack Prime 17w
- ・同期インターフェース : OptiTrack eSync2
- ・補助デバイス : Seeed Studio XIAO nRF52840 (送信用・受信用各1台)
- ・パルス発生器 : Raspberry pi 4
- ・レンダリング用PC : Dell XPS8950
- ・開発環境 : Arduino IDE, Rust, Python (VPython)

9.2 評価項目

本システムの性能を評価するため、以下の項目について検証を行う.

9.2.1 姿勢推定精度

疑似パルス発生器により補助デバイスを動作させ、ジンバルロックが発生しないことを確認し、実際の動きと同様の動きを推定できているかを確認する.

9.2.2 レンダリング可能性

UnityでOptiTrackのデータと補助デバイスのデータを同時再生可能かを確認する.

9.3 実験方法

Raspberry pi 4を使用し、疑似的に120Hzのパルスを発生させる.

ジンバルロックが発生する角度へ送信機を傾け、ジンバルロックが発生しないことを確認する。また、激しい動作、ゆっくりとした動作で傾け、その後停止させることで加速度による補正が正しく行われているかを確認。

UnityでOptiTrackのデータと補助デバイスからのデータを再生し、互いに影響なく再生でき、動作するかを確認する。OptiTrackからリアルタイムストリームでデータを取得することがライセンスの問題上不可能であったため、fbxファイル形式での録画データを使用。補助デバイスはcsvファイル形式でデータを保存し、Unity上でシーケンス番号を同期して再生した。

9.4 結果

ジンバルロックが発生するPitch=±90°に送信機を傾けたとき、ジンバルロックが発生しないことを確認できた。

激しく動作させたとき、図 第9章.1 図 第9章.2 のように、実際の姿勢と異なる姿勢で停止し、その後数秒にかけて実際の姿勢と同じ姿勢に補正されていった。

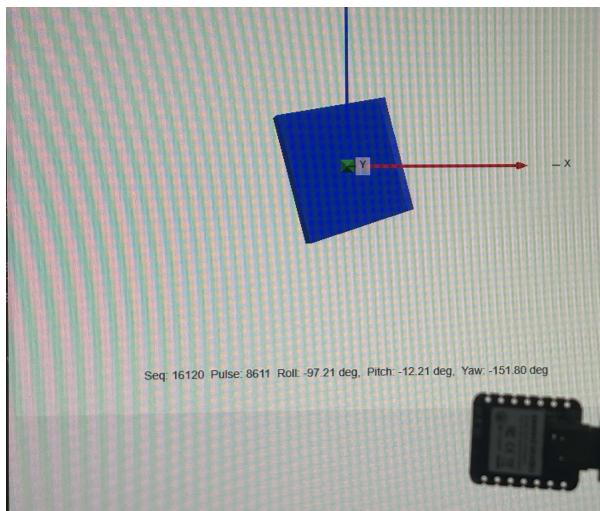


図 第9章.1: 激しく動作させた後に停止したときの
プレビュー

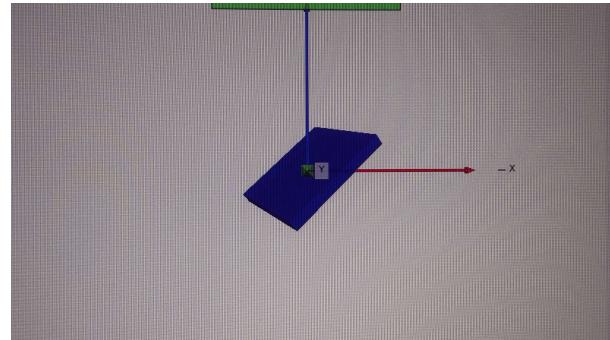


図 第9章.2: 激しく動作させた後に卓上で固定した
ときのプレビュー

Unity上でデータを同時再生したところ、図 第9章.3のように問題なく再生され、補助デバイスの実際の姿勢と同様の姿勢をとっていた。



図 第9章.3: Unity上での動作プレビュー

9.5 考察

ジンバルロックが発生しなかったことから、クォータニオンで計算することでジンバルロックが発生していないとわかる。

激しく動作させたのち停止したとき、実際の姿勢と異なる姿勢で停止することから、高速移動中はジャイロのみで積分するため、応答性は高いものの誤差が蓄積してしまい、停止したときその誤差が顕在化されたものだと思われる。静止後実際の姿勢に近づく現象に関して、静止後は加速度による補正が開始するため、今回使用する相補フィルタの $\alpha = 0.98$ より、2%ずつ元の姿勢に戻る。

Unity上でデータを同時再生できていたため、OptiTrackのリアルタイムストリーム機能を使用しても同様に動作すると思われる。

第10章 結論

10.1 まとめ

本研究では、光学式モーションキャプチャシステムOptiTrackと連携可能な小型補助デバイスの開発を行った。本研究で達成した内容を以下に示す。

1. 小型補助デバイスの開発：Seeed Studio XIAO nRF52840を用いた小型・軽量なIMU搭載デバイスを開発した。
2. 姿勢推定システムの構築：6軸IMU（LSM6DS3）から取得した角速度および加速度データを用いて、相補フィルタによる姿勢推定を実装した。
3. 無線データ転送：Bluetooth Low Energy（BLE）を使用し、センサデータを無線で転送するシステムを構築した。
4. 同期機能の実装：パルス信号を利用し、光学式モーションキャプチャとの同期機能を実現した。
5. リアルタイム可視化：VPythonを用いた3D可視化ツールを開発し、姿勢推定結果をリアルタイムで確認可能とした。

これらにより、光学式モーションキャプチャでは取得困難であった小型物体の姿勢情報を補完するためのシステム基盤を構築した。

10.2 今後の課題

本研究で開発したシステムには、以下の課題が残されている。

1. 複数デバイスへの対応：現在は1台の補助デバイスのみに対応しているが、複数の小型物体を同時に計測するための拡張が求められる。
2. ヨー角ドリフトの補正：6軸IMUではヨー角のドリフトを完全に補正することが困難であるため、磁気センサの追加による9軸化が有効と考えられる。
3. デバイスの小型化：より小型・軽量なデバイス設計により、適用可能な対象物体の範囲を拡大できる。
4. 統合アプリケーションの開発：Unity等を用いた統合的な可視化・解析アプリケーションの開発が今後の展望として挙げられる。
5. 激しく動作した後の誤差の蓄積に関して、相補フィルタのプロパティ変更や動的補正係数を導入することで解消されると考えられる。

第11章 謝辞

研究および論文の執筆にあたり、多忙な中ご指導をいただいた指導教員の内堀教授に深謝いたします。
また、実験環境を整備するにあたり、協力してくれた知能ロボット実験室の皆様に感謝申し上げます。

第12章 参考文献

- [1] Vanani, Poojan, Patel, Darsh, Khorami, Danyal, Munaganuru, Siva, Reddy, Pavan, Reddy, Varun, Raghunath, Bhargav, Lallmamode, Ishrat, Patel, Romir, Kidané, Assegid と others, Mesquite Mo-Cap: Democratizing Real-Time Motion Capture with Affordable, Bodyworn IoT Sensors and WebXR SLAM, <https://arxiv.org/html/2512.22690v2>, 2025
- [2] Ze, Yanjie, Chen, Zixuan, Araújo, Joao Pedro, Cao, Zi-ang, Peng, Xue Bin, Wu, Jiajun と Liu, C Karen, Twist: Teleoperated whole-body imitation system, <https://arxiv.org/html/2505.02833v1>, 2025
- [3] Inc., Acuity, モーションキャプチャの基礎知識, <https://www.motioncapture.jp/optitrack/knowledge/how/20161028.html>, 2026-01-27
- [4] SPICE, モーションキャプチャとは? 原理や事例、種類によるメリットデメリットをわかりやすく解説, <https://mocap.jp/about-mocap/>, 2026-01-27
- [5] Benjamin R Hindle, Anna V Lorimer, Justin W L Keogh, Inertial-Based Human Motion Capture: A Technical Summary of Current Processing Methodologies for Spatiotemporal and Kinematic Measures, <https://pubmed.ncbi.nlm.nih.gov/33859720/>, 2021
- [6] Hailey N. Hicks, Sara A. Harper, Howard Chen, Sensor Fusion for Enhancing Motion Capture: Integrating Optical and Inertial Motion Capture Systems, <https://doi.org/10.3390/s25154680>,
- [7] Geise Santos, Tiago Tavares, Marcelo Wanderley, A multi-sensor human gait dataset captured through an optical system and inertial measurement units, <https://arxiv.org/abs/2111.15044>,
- [8] Inc., Acuity, OptiTrackカメラによる計測店の認識の原理, <https://www.motioncapture.jp/optitrack/knowledge/how/20161114.html>, 2026-01-27
- [9] Inc., Acuity, OptiTrackカメラでの計測なら3次元化が可能な原理, <https://www.motioncapture.jp/optitrack/knowledge/how/20161220.html>, 2026-01-27
- [10] Inc., Acuity, OptiTrack クイックスタートガイド_2.1:2:2, https://www.acuity-inc.co.jp/wordpress/wp-content/uploads/2023/11/OptiTrack_QuickStartGuide2.1_compressed.pdf, 2026-01-27
- [11] Triggs, Bill, McLauchlan, Philip F, Hartley, Richard I と Fitzgibbon, Andrew W, Bundle adjustment—a modern synthesis, International workshop on vision algorithms, https://link.springer.com/chapter/10.1007/3-540-44480-7_21, 1999
- [12] Inc., Acuity, OptiTrack Documentation v3.0, <https://docs.optitrack.com/v3.0/motive/calibration>, 2026-01-27
- [13] Inc., Acuity, OptiTrack Documentation v3.0, <https://docs.optitrack.com/v3.0/motive/rigid-body-tracking>, 2026-01-27
- [14] Escalona, José L., Kinematics of motion tracking using computer vision, <https://doi.org/10.48550/arXiv.2008.00813>, 2020
- [15] Diebel, James と others, Representing attitude: Euler angles, unit quaternions, and rotation vectors, Matrix, 2006
- [16] SPICE, esync2, <https://mocap.jp/optitrack/accessory/esync2/>, 2026-01-27
- [17] OptiTrack, External Device Sync Guide: eSync 2, <https://docs.optitrack.com/synchronization/synchronization-hardware/external-device-sync-guide-esync-2>, 2026-01-27

- [18] STMicroelectronics, LSM6DS3TR-C エントリ・レベル / ミッドティア・スマートフォンおよびポータブル PC プラットフォーム向け iNEMO 6 軸慣性測定ユニット (IMU) ドキュメント, <https://www.st.com/ja/mems-and-sensors/lsm6ds3tr-c.html#documentation>, 2026-01-27

第13章 付録

13.0.1 受信機のソースコード

```
#include <bluefruit.h>
void data_notify_callback(BLEClientCharacteristic* chr, uint8_t* data, uint16_t len);

#define DATA_SERVICE_UUID      0x1234
#define DATA_CHAR_UUID        0x5678
#define REQUEST_CHAR_UUID     0x5679 //    Characteristic

#define ENABLE_DEBUG true

//  

#define AUTO_REQUEST_MODE true // true:           , false:  

#define AUTO_REQUEST_INTERVAL 100 //           (ms)

// GPIO
#define SYNC_RESET_PIN 6 // GPIO6:          0
#define SYNC_COUNT_PIN 7 // GPIO7:  
  

BLEClientService      dataService(DATA_SERVICE_UUID);  

BLEClientCharacteristic dataChar(DATA_CHAR_UUID);  

BLEClientCharacteristic requestChar(REQUEST_CHAR_UUID);  
  

void scan_callback(ble_gap_evt_adv_report_t* report);
void connect_callback(uint16_t conn_handle);
void disconnect_callback(uint16_t conn_handle, uint8_t reason);  
  

//  

volatile uint32_t currentSequence = 0;
volatile bool syncResetTriggered = false;
volatile bool syncCountTriggered = false;  
  

// : GPIO6 -
void syncResetISR() {
    currentSequence = 0;
    syncResetTriggered = true;
}  
  

// : GPIO7 -
void syncCountISR() {
```

```

currentSequence++;
syncCountTriggered = true;
}

void setup()
{
    Serial.begin(921600);
#if ENABLE_DEBUG
    for (int i = 0; i < 100 && !Serial; i++) {
        delay(10);
    }
    Serial.println("== XIAO nRF52840 Central (data receiver with sync) ===");
    Serial.println("ENABLE_DEBUG is ON");
#else
    delay(100);
#endif

pinMode(SYNC_RESET_PIN, INPUT_PULLUP);
pinMode(SYNC_COUNT_PIN, INPUT_PULLUP);

// attachInterrupt(digitalPinToInterrupt(SYNC_RESET_PIN), syncResetISR, RISING);
// attachInterrupt(digitalPinToInterrupt(SYNC_COUNT_PIN), syncCountISR, RISING);

#if ENABLE_DEBUG
    Serial.println("GPIO interrupts configured:");
    Serial.print(" GPIO");
    Serial.print(SYNC_RESET_PIN);
    Serial.println(": Sync reset (sequence = 0) - RISING edge");
    Serial.print(" GPIO");
    Serial.print(SYNC_COUNT_PIN);
    Serial.println(": Sync count (sequence++) - RISING edge");
#endif

Bluefruit.begin(0, 1);
Bluefruit.setName("XIAO Central");

dataService.begin();

dataChar.setNotifyCallback(data_notify_callback);
dataChar.begin();

```

```

requestChar.begin();

Bluefruit.Central.setConnectCallback(connect_callback);
Bluefruit.Central.setDisconnectCallback(disconnect_callback);

Bluefruit.Scanner.setRxCallback(scan_callback);
Bluefruit.Scanner.restartOnDisconnect(true);
Bluefruit.Scanner.setInterval(160, 80);
Bluefruit.Scanner.filterUuid(dataService.uuid);
Bluefruit.Scanner.useActiveScan(false);
Bluefruit.Scanner.start(0);

#if ENABLE_DEBUG
Serial.println("Scanning for peripheral...");
#endif
}

void loop()
{
#if AUTO_REQUEST_MODE
// :
static uint32_t lastAutoRequest = 0;
uint32_t now = millis();

if (Bluefruit.connected() && (now - lastAutoRequest >= AUTO_REQUEST_INTERVAL)) {
    lastAutoRequest = now;
    currentSequence++;

#if ENABLE_DEBUG
    static uint32_t lastDebugPrint = 0;
    if (now - lastDebugPrint >= 1000) {
        lastDebugPrint = now;
        Serial.print("[AUTO] Sequence: ");
        Serial.println(currentSequence);
    }
#endif

    sendSequenceRequest();
}

```

```

#else
//    : GPIO
#if ENABLE_DEBUG
// GPIO7
static uint32_t lastPinCheck = 0;
static int lastPinState = -1;
uint32_t now = millis();

if (now - lastPinCheck >= 100) { // 100ms
    lastPinCheck = now;
    int currentPinState = digitalRead(SYNC_COUNT_PIN);
    if (currentPinState != lastPinState) {
        Serial.print("[GPIO] Pin ");
        Serial.print(SYNC_COUNT_PIN);
        Serial.print(" state changed to: ");
        Serial.println(currentPinState == HIGH ? "HIGH" : "LOW");
        lastPinState = currentPinState;
    }
}
#endif

//
if (syncCountTriggered) {
    syncCountTriggered = false;

#if ENABLE_DEBUG
Serial.print("[PULSE] Count pulse detected! Sequence: ");
Serial.println(currentSequence);
#endif

if (Bluefruit.connected()) {
    sendSequenceRequest();
} else {
    #if ENABLE_DEBUG
    Serial.println("[WARN] Not connected, cannot send request");
    #endif
}
}

if (syncResetTriggered) {
    syncResetTriggered = false;
}

```

```

#if ENABLE_DEBUG
Serial.print("[PULSE] Reset pulse detected! Sequence reset to: ");
Serial.println(currentSequence);
#endif

if (Bluefruit.connected()) {
    sendSequenceRequest();
} else {
    #if ENABLE_DEBUG
    Serial.println("[WARN] Not connected, cannot send request");
    #endif
}
}

#endif
}

//      Sender
void sendSequenceRequest() {
    if (!requestChar.discovered()) {
        #if ENABLE_DEBUG
        Serial.println("Request characteristic not discovered");
        #endif
        return;
    }

    //      :
    uint32_t seq = currentSequence;

    #if ENABLE_DEBUG
    Serial.print("Sending request with sequence: ");
    Serial.println(seq);
    #endif

    // Write
    requestChar.write32(seq);
}

void scan_callback(ble_gap_evt_adv_report_t* report)
{
    #if ENABLE_DEBUG

```

```

Serial.println("Found device, trying to connect...");
#endif

Bluefruit.Central.connect(report);
}

void connect_callback(uint16_t conn_handle)
{
#if ENABLE_DEBUG
Serial.println("Connected");
Serial.println("Discovering data service ...");
#endif

if (!dataService.discover(conn_handle)) {
#if ENABLE_DEBUG
Serial.println("Data service NOT found, disconnect");
#endif
Bluefruit.disconnect(conn_handle);
return;
}

#if ENABLE_DEBUG
Serial.println("Data service found");
Serial.println("Discovering data characteristic ...");
#endif

if (!dataChar.discover()) {
#if ENABLE_DEBUG
Serial.println("Data characteristic NOT found, disconnect");
#endif
Bluefruit.disconnect(conn_handle);
return;
}

#if ENABLE_DEBUG
Serial.println("Data characteristic found");
#endif

// Characteristic
if (!requestChar.discover()) {
#if ENABLE_DEBUG

```

```

Serial.println("Request characteristic NOT found (optional)");
#endif
} else {
#if ENABLE_DEBUG
Serial.println("Request characteristic found");
#endif
}

BLEConnection* conn = Bluefruit.Connection(conn_handle);
if (conn) {
#if ENABLE_DEBUG
Serial.println("Requesting MTU exchange...");
#endif

conn->requestMtuExchange(50);
delay(100);

#if ENABLE_DEBUG
uint16_t mtu = conn->getMtu();
Serial.print("MTU: ");
Serial.println(mtu);
#endif

conn->requestConnectionParameter(6);
}

#if ENABLE_DEBUG
Serial.println("Enable notifications ...");
#endif

if (dataChar.enableNotify()) {
#if ENABLE_DEBUG
Serial.println("Notification enabled, ready to receive data");
Serial.println("--- DATA START ---");
delay(100);
#endif
} else {
#if ENABLE_DEBUG
Serial.println("Failed to enable notification, disconnect");
#endif
Bluefruit.disconnect(conn_handle);
}

```

```

    }

}

void disconnect_callback(uint16_t conn_handle, uint8_t reason)
{
    (void)conn_handle;

#if ENABLE_DEBUG
    Serial.print("Disconnected, reason=0x");
    Serial.println(reason, HEX);
#endif
}

static uint8_t packet_buffer[24];
static uint16_t buffer_pos = 0;
static const uint16_t PACKET_SIZE = 24;

void data_notify_callback(BLEClientCharacteristic* chr, uint8_t* data, uint16_t len)
{
    (void)chr;

    for (uint16_t i = 0; i < len; i++) {
        if (buffer_pos < PACKET_SIZE) {
            packet_buffer[buffer_pos++] = data[i];
        }
    }

    if (buffer_pos == PACKET_SIZE) {
        Serial.write(packet_buffer, PACKET_SIZE);

        buffer_pos = 0;
    }
}
}

```

13.0.2 送信機のソースコード

```

#include <bluefruit.h>
#include <Adafruit_NeoPixel.h>
#include <LSM6DS3.h>
#include <Wire.h>

#define TARGET_FPS 120

```

```

#define SEND_INTERVAL_MS (1000 / TARGET_FPS)

#define ENABLE_DEBUG true
#define ENABLE_STATS true

#define PACKET_HEADER 0xAA55

#define REQUEST_CHAR_UUID 0x5679 //      Characteristic

//  

struct __attribute__((packed)) IMUPacket {
    uint16_t header;      // 0xAA55 ( )
    uint32_t seq;         //
    uint32_t requestSeq; //           Receiver
    int16_t gx;           // X ( : -32768 ~ 32767)
    int16_t gy;           // Y ( )
    int16_t gz;           // Z ( )
    int16_t ax;           // X ( : -32768 ~ 32767)
    int16_t ay;           // Y ( )
    int16_t az;           // Z ( )
    uint16_t checksum;    //
};

LSM6DS3 imu(I2C_MODE, 0x6A);

#ifndef PIN_NEOPIXEL
#define PIN_NEOPIXEL 11
#endif

Adafruit_NeoPixel pixel(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);

BLEService      customService(0x1234);
BLECharacteristic txChar(0x5678);
BLECharacteristic requestChar(REQUEST_CHAR_UUID);

void cccd_callback(uint16_t conn_hdl, BLECharacteristic* chr, uint16_t cccd_value);
void request_write_callback(uint16_t conn_hdl, BLECharacteristic* chr, uint8_t* data, uint16_t len);

void startAdv();

//
```

```

volatile uint32_t requestedSequence = 0;
volatile bool hasNewRequest = false;

enum LedMode {
    LEDMODE_ADV,
    LEDMODE_CONNECTED,
    LEDMODE_CCCD_PULSE,
    LEDMODE_NOTIFY_ERROR,
    LEDMODE_REQUEST_RECEIVED
};

LedMode ledMode      = LEDMODE_ADV;
uint32_t ledModeExpire = 0;
bool ledState       = false;
uint32_t ledBlinkPrev = 0;

void setPixelColor(uint8_t r, uint8_t g, uint8_t b) {
    pixel.setPixelColor(0, pixel.Color(r, g, b));
    pixel.show();
}

void showColor(bool on, uint8_t r, uint8_t g, uint8_t b) {
    if (on) {
        setPixelColor(r, g, b);
    } else {
        setPixelColor(0, 0, 0);
    }
}

void updateLed() {
    static uint32_t lastUpdate = 0;
    uint32_t now = millis();

    if (now - lastUpdate < 10) return;
    lastUpdate = now;

    if (ledMode == LEDMODE_CCCD_PULSE || ledMode == LEDMODE_NOTIFY_ERROR || ledMode == LEDMODE_REQUEST_RECEIVED)
        if (now > ledModeExpire) {
            if (Bluefruit.connected()) {
                ledMode = LEDMODE_CONNECTED;
            } else {

```

```

        ledMode = LEDMODE_ADV;
    }
    ledState = false;
    setPixelColor(0, 0, 0);
}
}

switch (ledMode) {
    case LEDMODE_ADV: {
        const uint32_t interval = 500;
        if (now - ledBlinkPrev >= interval) {
            ledBlinkPrev = now;
            ledState = !ledState;
        }
        showColor(ledState, 0, 0, 255);
    } break;

    case LEDMODE_CONNECTED: {
        ledState = true;
        showColor(true, 0, 255, 0);
    } break;

    case LEDMODE_CCCD_PULSE: {
        ledState = true;
        showColor(true, 255, 255, 0);
    } break;

    case LEDMODE_NOTIFY_ERROR: {
        const uint32_t interval = 100;
        if (now - ledBlinkPrev >= interval) {
            ledBlinkPrev = now;
            ledState = !ledState;
        }
        showColor(ledState, 255, 0, 0);
    } break;

    case LEDMODE_REQUEST_RECEIVED: {
        ledState = true;
        showColor(true, 0, 255, 255);
    } break;
}

```

```

}

void setup() {
    Serial.begin(115200);
    #if ENABLE_DEBUG
    for (int i = 0; i < 100 && !Serial; i++) {
        delay(10);
    }
    #else
    delay(100);
    #endif

    #if ENABLE_DEBUG
    Serial.println("== XIAO Peripheral (TX) - 120Hz Gyro BLE Sender with Sync ==");
    Serial.print("Target FPS: ");
    Serial.println(TARGET_FPS);
    Serial.print("Send interval: ");
    Serial.print(SEND_INTERVAL_MS);
    Serial.println(" ms");
    Serial.print("IMUPacket size: ");
    Serial.print(sizeof(IMUPacket));
    Serial.println(" bytes");
    #endif

    // I2C
    Wire.begin();
    Wire.setClock(400000);
    delay(100);

    // IMU
    imu.settings.gyroRange = 2000;
    imu.settings.accelRange = 4;

    #if ENABLE_DEBUG
    Serial.println("Initializing IMU sensor...");
    #endif

    uint16_t result = imu.begin();
    if (result != 0) {
        #if ENABLE_DEBUG
        Serial.print("IMU initialization failed! Error code: ");

```

```

Serial.println(result);

Serial.println("Scanning I2C bus...");

for (uint8_t addr = 1; addr < 127; addr++) {
    Wire.beginTransmission(addr);
    if (Wire.endTransmission() == 0) {
        Serial.print("Found I2C device at 0x");
        if (addr < 16) Serial.print("0");
        Serial.println(addr, HEX);
    }
}
#endif

while (1) {
    delay(100);
}

// IMU SETTINGS
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL2_G, 0x8C);
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL1_XL, 0x8A);
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL7_G, 0x00);
imu.writeRegister(LSM6DS3_ACC_GYRO_CTRL8_XL, 0x09);

#if ENABLE_DEBUG
Serial.println("IMU initialized successfully!");
#endif

pixel.begin();
pixel.setBrightness(32);
setPixelColor(0, 0, 0);
ledMode      = LEDMODE_ADV;
ledBlinkPrev = millis();

Bluefruit.begin();
Bluefruit.setName("XIAO_TX_120Hz");

Bluefruit.Periph.setConnInterval(6, 6);
Bluefruit.Periph.setConnSupervisionTimeout(400);

Bluefruit.Periph.setConnectCallback([](uint16_t conn_hdl) {

```

```

#if ENABLE_DEBUG
Serial.println("Central connected");

uint16_t mtu = Bluefruit.Connection(conn_hdl)->getMtu();
Serial.print("Negotiated MTU: ");
Serial.println(mtu);
#endif

ledMode = LEDMODE_CONNECTED;
});

Bluefruit.Periph.setDisconnectCallback([](uint16_t conn_hdl, uint8_t reason) {
#if ENABLE_DEBUG
Serial.print("Central disconnected, reason = ");
Serial.println(reason);
#endif
ledMode = LEDMODE_ADV;
//  

requestedSequence = 0;
hasNewRequest = false;
});

customService.begin();

// Characteristic (Notify)
txChar.setProperties(CHR_PROPS_NOTIFY);
txChar.setPermission(SECMODE_OPEN, SECMode_NO_ACCESS);
txChar.setMaxLen(sizeof(IMUPacket));
txChar.setCccdWriteCallback(cccd_callback);
txChar.begin();

// Characteristic (Write)
requestChar.setProperties(CHR_PROPS_WRITE | CHR_PROPS_WRITE_WO_RESP);
requestChar.setPermission(SECMODE_OPEN, SECMode_OPEN);
requestChar.setMaxLen(sizeof(uint32_t));
requestChar.setWriteCallback(request_write_callback);
requestChar.begin();

startAdv();

#endif

```

```

Serial.println("Setup complete. Ready to send gyro data at 120Hz via BLE.");
Serial.println("Waiting for sequence requests from Central...");
#endif
}

void loop() {
    static uint32_t counter = 0;
    static uint32_t lastLogTime = 0;
    static uint32_t sentCount = 0;
    static uint32_t failCount = 0;

    uint32_t now = millis();

    // :
    if (Bluefruit.connected() && hasNewRequest) {
        hasNewRequest = false;

        // IMU DATA READ
        float gyroX_f = imu.readFloatGyroX();
        float gyroY_f = imu.readFloatGyroY();
        float gyroZ_f = imu.readFloatGyroZ();

        float accelX_f = imu.readFloatAccelX();
        float accelY_f = imu.readFloatAccelY();
        float accelZ_f = imu.readFloatAccelZ();

        int16_t gyroX = constrain((int16_t)(gyroX_f * 16.384), -32768, 32767);
        int16_t gyroY = constrain((int16_t)(gyroY_f * 16.384), -32768, 32767);
        int16_t gyroZ = constrain((int16_t)(gyroZ_f * 16.384), -32768, 32767);

        int16_t accelX = constrain((int16_t)(accelX_f * 8192), -32768, 32767);
        int16_t accelY = constrain((int16_t)(accelY_f * 8192), -32768, 32767);
        int16_t accelZ = constrain((int16_t)(accelZ_f * 8192), -32768, 32767);

        //

        uint32_t currentRequestSeq = requestedSequence;

        IMUPacket packet;
        packet.header = PACKET_HEADER;
        packet.seq = counter;
        packet.requestSeq = currentRequestSeq;
}

```

```

packet.gx = gyroX;
packet gy = gyroY;
packet.gz = gyroZ;
packet.ax = accelX;
packet.ay = accelY;
packet.az = accelZ;

//      requestSeq
packet.checksum = packet.header ^ (packet.seq & 0xFFFF) ^ ((packet.seq >> 16) & 0xFFFF)
                  ^ (packet.requestSeq & 0xFFFF) ^ ((packet.requestSeq >> 16) & 0xFFFF)
                  ^ packet.gx ^ packet.gy ^ packet.gz
                  ^ packet.ax ^ packet.ay ^ packet.az;

bool ok = txChar.notify((uint8_t*)&packet, sizeof(packet));

if (!ok) {
    failCount++;
    ledMode = LEDMODE_NOTIFY_ERROR;
    ledModeExpire = millis() + 1000;
    ledBlinkPrev = millis();

#if ENABLE_DEBUG
    Serial.println("Failed to send data packet");
#endif
} else {
    sentCount++;
    counter++;

#if ENABLE_DEBUG
    Serial.print("Sent packet #");
    Serial.print(counter - 1);
    Serial.print(" for request seq: ");
    Serial.println(currentRequestSeq);
#endif
}

#if ENABLE_STATS
if (now - lastLogTime >= 1000) {
    lastLogTime = now;
    if (Bluefruit.connected()) {

```

```

        Serial.print("Sent: ");
        Serial.print(sentCount);
        Serial.print(" pkts/s | Failed: ");
        Serial.print(failCount);
        Serial.print(" | Total: ");
        Serial.print(counter);
        Serial.print(" | ReqSeq: ");
        Serial.println(requestedSequence);
    }
    sentCount = 0;
    failCount = 0;
}
#endif

updateLed();
}

//  

void request_write_callback(uint16_t conn_hdl, BLECharacteristic* chr, uint8_t* data, uint16_t len) {
    (void)conn_hdl;
    (void)chr;

    if (len >= sizeof(uint32_t)) {
        uint32_t newSeq;
        memcpy(&newSeq, data, sizeof(uint32_t));
        requestedSequence = newSeq;
        hasNewRequest = true;

        #if ENABLE_DEBUG
        Serial.print("Received sequence request: ");
        Serial.println(newSeq);
        #endif
    }

    ledMode = LEDMODE_REQUEST_RECEIVED;
    ledModeExpire = millis() + 100;
}
}

void cccd_callback(uint16_t conn_hdl, BLECharacteristic* chr, uint16_t cccd_value) {
    #if ENABLE_DEBUG
    Serial.print("CCCD updated: conn=");
}

```

```

Serial.print(conn_hdl);
Serial.print(" value=0x");
Serial.println(cccd_value, HEX);
#endif

ledMode      = LEDMODE_CCCD_PULSE;
ledModeExpire = millis() + 200;
}

void startAdv() {
    Bluefruit.Advertising.stop();

    Bluefruit.Advertising.addFlags(BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE);
    Bluefruit.Advertising.addTxPower();
    Bluefruit.Advertising.addService(customService);
    Bluefruit.Advertising.addName();

    Bluefruit.Advertising.restartOnDisconnect(true);

    Bluefruit.Advertising.setInterval(32, 244);
    Bluefruit.Advertising.setFastTimeout(30);
    Bluefruit.Advertising.start(0);

#if ENABLE_DEBUG
    Serial.println("Advertising started - Device name: XIAO_TX_120Hz");
#endif

    ledMode      = LEDMODE_ADV;
    ledBlinkPrev = millis();
}

```

13.0.3 レンダリング用コンピュータで動作させる姿勢推定プログラムのソースコード

```

use anyhow::[Context, Result];
use serialport::SerialPort;
use std::io::{self, Write, BufRead, BufReader};
use std::sync::{Arc, Mutex};
use std::thread;
use std::time::{Duration, Instant};
use std::net::UdpSocket;

// =====

```

```

const PACKET_SIZE: usize = 24; // 24    requestSeq
const TARGET_FPS: f64 = 120.0;
const DT: f64 = 1.0 / TARGET_FPS;
const DEG2RAD: f64 = std::f64::consts::PI / 180.0;

// ===== IMU =====

#[derive(Debug, Clone, Copy)]
struct ImuPacket {
    seq: u32,
    request_seq: u32, //
    gx: i16,
    gy: i16,
    gz: i16,
    ax: i16,
    ay: i16,
    az: i16,
}

impl ImuPacket {
    fn from_bytes(buf: &[u8]) -> Option<Self> {
        if buf.len() != PACKET_SIZE {
            return None;
        }

        let seq = u32::from_le_bytes([buf[2], buf[3], buf[4], buf[5]]);
        let request_seq = u32::from_le_bytes([buf[6], buf[7], buf[8], buf[9]]);
        let gx = i16::from_le_bytes([buf[10], buf[11]]);
        let gy = i16::from_le_bytes([buf[12], buf[13]]);
        let gz = i16::from_le_bytes([buf[14], buf[15]]);
        let ax = i16::from_le_bytes([buf[16], buf[17]]);
        let ay = i16::from_le_bytes([buf[18], buf[19]]);
        let az = i16::from_le_bytes([buf[20], buf[21]]);

        Some(ImuPacket {
            seq,
            request_seq,
            gx,
            gy,
            gz,
        })
    }
}

```

```

        ax,
        ay,
        az,
    })
}
}

// =====

fn gyro_to_dps(gx: i16, gy: i16, gz: i16) -> (f64, f64, f64) {
    // : ±2000 dps / 16.4 LSB/dps
    let scale = 16.384_f64;
    (gx as f64 / scale, gy as f64 / scale, gz as f64 / scale)
}

fn accel_to_g(ax: i16, ay: i16, az: i16) -> (f64, f64, f64) {
    // : ±4 g / 8192 LSB/g
    let scale = 8192.0_f64;
    (ax as f64 / scale, ay as f64 / scale, az as f64 / scale)
}

// =====

fn clear_serial_buffer(port: &mut dyn SerialPort) -> Result<usize> {
    let mut cleared = 0usize;
    let mut buf = [0u8; 1024];
    let start_time = Instant::now();
    let max_duration = Duration::from_millis(100); // 100ms

    //
    while start_time.elapsed() < max_duration {
        match port.read(&mut buf) {
            Ok(n) if n > 0 => {
                cleared += n;
            }
            Ok(_) | Err(_) => {
                break;
            }
        }
    }
}

```

```

Ok(cleared)
}

// ===== 1 =====

fn read_packet(port: &mut dyn SerialPort) -> Result<Option<ImuPacket>> {
    let mut byte = [0u8; 1];
    let sync_start = Instant::now();
    let sync_timeout = Duration::from_secs(1); //

    //      (0x55, 0xAA) -
    loop {
        if sync_start.elapsed() > sync_timeout {
            return Ok(None);
        }

        match port.read_exact(&mut byte) {
            Ok(_) => {}
            Err(e) => {
                if e.kind() == std::io::ErrorKind::TimedOut {
                    return Ok(None);
                } else {
                    return Err(e.into());
                }
            }
        }
    }

    if byte[0] == 0x55 {
        match port.read_exact(&mut byte) {
            Ok(_) => {}
            Err(e) => {
                if e.kind() == std::io::ErrorKind::TimedOut {
                    return Ok(None);
                } else {
                    return Err(e.into());
                }
            }
        }
    }

    if byte[0] == 0xAA {
        break;
    }
}

```

```

        }
    }

//  

let mut rest = [0u8; PACKET_SIZE - 2];  

match port.read_exact(&mut rest) {  

    Ok(_) => {}  

    Err(e) => {  

        if e.kind() == std::io::ErrorKind::TimedOut {  

            return Ok(None);  

        } else {  

            return Err(e.into());  

        }
    }
}

let mut buf = [0u8; PACKET_SIZE];  

buf[0] = 0x55;  

buf[1] = 0xAA;  

buf[2..].copy_from_slice(&rest);

Ok(ImuPacket::from_bytes(&buf))
}

// =====
#[derive(Debug, Clone, Copy)]
struct Quaternion {
    w: f64,
    x: f64,
    y: f64,
    z: f64,
}

impl Quaternion {
    fn identity() -> Self {
        Self {
            w: 1.0,
            x: 0.0,
            y: 0.0,
            z: 0.0,
        }
    }
}

```

```

        }
    }

fn normalize(self) -> Self {
    let n = (self.w * self.w + self.x * self.x + self.y * self.y + self.z * self.z).sqrt();
    if n == 0.0 {
        Self::identity()
    } else {
        Self {
            w: self.w / n,
            x: self.x / n,
            y: self.y / n,
            z: self.z / n,
        }
    }
}

fn mul(self, other: Self) -> Self {
    Self {
        w: self.w * other.w - self.x * other.x - self.y * other.y - self.z * other.z,
        x: self.w * other.x + self.x * other.w + self.y * other.z - self.z * other.y,
        y: self.w * other.y - self.x * other.z + self.y * other.w + self.z * other.x,
        z: self.w * other.z + self.x * other.y - self.y * other.x + self.z * other.w,
    }
}
}

///      (roll, pitch, yaw) [rad]          (Z-Y-X, yaw-pitch-roll)
fn from_euler(roll: f64, pitch: f64, yaw: f64) -> Self {
    let cr = (roll * 0.5).cos();
    let sr = (roll * 0.5).sin();
    let cp = (pitch * 0.5).cos();
    let sp = (pitch * 0.5).sin();
    let cy = (yaw * 0.5).cos();
    let sy = (yaw * 0.5).sin();

    Self {
        w: cy * cp * cr + sy * sp * sr,
        x: cy * cp * sr - sy * sp * cr,
        y: sy * cp * sr + cy * sp * cr,
        z: sy * cp * cr - cy * sp * sr,
    }
}

```

```

    .normalize()
}

///      +      (roll, pitch, yaw) [rad]
fn to_euler(self) -> (f64, f64, f64) {
    let qw = self.w;
    let qx = self.x;
    let qy = self.y;
    let qz = self.z;

    // roll (x-axis)
    let sinr_cosp = 2.0 * (qw * qx + qy * qz);
    let cosr_cosp = 1.0 - 2.0 * (qx * qx + qy * qy);
    let roll = sinr_cosp.atan2(cosr_cosp);

    // pitch (y-axis)
    let sinp = 2.0 * (qw * qy - qz * qx);
    let pitch = if sinp.abs() >= 1.0 {
        sinp.signum() * (std::f64::consts::FRAC_PI_2)
    } else {
        sinp.asin()
    };

    // yaw (z-axis)
    let siny_cosp = 2.0 * (qw * qz + qx * qy);
    let cosy_cosp = 1.0 - 2.0 * (qy * qy + qz * qz);
    let yaw = siny_cosp.atan2(cosy_cosp);

    (roll, pitch, yaw)
}

///      X Y  Unity
fn to_rotation_vectors(self) -> ((f64, f64, f64), (f64, f64, f64)) {
    let qw = self.w;
    let qx = self.x;
    let qy = self.y;
    let qz = self.z;

    //
    // X      (ex)
    let ex_x = 1.0 - 2.0 * (qy * qy + qz * qz);

```

```

let ex_y = 2.0 * (qx * qy + qw * qz);
let ex_z = 2.0 * (qx * qz - qw * qy);

// Y      (ey)
let ey_x = 2.0 * (qx * qy - qw * qz);
let ey_y = 1.0 - 2.0 * (qx * qx + qz * qz);
let ey_z = 2.0 * (qy * qz + qw * qx);

((ex_x, ex_y, ex_z), (ey_x, ey_y, ey_z))
}

///  (gx,gy,gz) [rad/s]   dt
fn integrate_gyro(self, gx: f64, gy: f64, gz: f64, dt: f64) -> Self {
    let dtheta_x = gx * dt;
    let dtheta_y = gy * dt;
    let dtheta_z = gz * dt;

    let theta = (dtheta_x * dtheta_x + dtheta_y * dtheta_y + dtheta_z * dtheta_z).sqrt();
    if theta < 1e-8 {
        return self;
    }

    let ux = dtheta_x / theta;
    let uy = dtheta_y / theta;
    let uz = dtheta_z / theta;

    let half = theta * 0.5;
    let sh = half.sin();

    let dq = Self {
        w: half.cos(),
        x: ux * sh,
        y: uy * sh,
        z: uz * sh,
    };

    self.mul(dq).normalize()
}

///      nlerp
fn nlerp(self, other: Self, t: f64) -> Self {

```

```

let t = t.clamp(0.0, 1.0);
let inv_t = 1.0 - t;

let dot = self.w * other.w + self.x * other.x + self.y * other.y + self.z * other.z;
let (ow, ox, oy, oz) = if dot < 0.0 {
    (-other.w, -other.x, -other.y, -other.z)
} else {
    (other.w, other.x, other.y, other.z)
};

Self {
    w: self.w * inv_t + ow * t,
    x: self.x * inv_t + ox * t,
    y: self.y * inv_t + oy * t,
    z: self.z * inv_t + oz * t,
}
.normalize()
}

// =====

fn attitude_from_accel(ax: f64, ay: f64, az: f64) -> (f64, f64) {
    let roll = ay.atan2(az);
    let pitch = (-ax).atan2((ay * ay + az * az).sqrt());
    (roll, pitch)
}

// =====

fn update_orientation_quat(
    q: &mut Quaternion,
    gx_dps: f64,
    gy_dps: f64,
    gz_dps: f64,
    ax_g: f64,
    ay_g: f64,
    az_g: f64,
    dt: f64,
    alpha: f64,
) {

```

```

// dps -> rad/s
let gx = gx_dps * DEG2RAD;
let gy = gy_dps * DEG2RAD;
let gz = gz_dps * DEG2RAD;

// 1)
let q_gyro = q.integrate_gyro(gx, gy, gz, dt);

// 2)
let norm_a = (ax_g * ax_g + ay_g * ay_g + az_g * az_g).sqrt();
if norm_a < 0.5 || norm_a > 1.5 {
    *q = q_gyro;
    return;
}

//      roll/pitch
let (roll_acc, pitch_acc) = attitude_from_accel(ax_g, ay_g, az_g);

//      yaw
let (_, _, yaw_gyro) = q_gyro.to_euler();

// roll/pitch      yaw
let q_acc = Quaternion::from_euler(roll_acc, pitch_acc, yaw_gyro);

// 3) ~ 0.98          (nlerp)
*q = q_gyro.nlerp(q_acc, 1.0 - alpha);
}

// =====
//
//      accel
//


fn calibrate(port: &mut dyn SerialPort, duration_sec: f64) -> Result<(Quaternion, (f64, f64, f64))> {
    let start = Instant::now();

    let mut gyro_sum = (0.0_f64, 0.0_f64, 0.0_f64);
    let mut accel_sum = (0.0_f64, 0.0_f64, 0.0_f64);
    let mut count = 0usize;

    while start.elapsed().as_secs_f64() < duration_sec {

```

```

match read_packet(port)? {
    Some(pkt) => {
        let (gx_dps, gy_dps, gz_dps) = gyro_to_dps(pkt.gx, pkt.gy, pkt.gz);
        let (ax_g, ay_g, az_g) = accel_to_g(pkt.ax, pkt.ay, pkt.az);

        gyro_sum.0 += gx_dps;
        gyro_sum.1 += gy_dps;
        gyro_sum.2 += gz_dps;

        accel_sum.0 += ax_g;
        accel_sum.1 += ay_g;
        accel_sum.2 += az_g;

        count += 1;
    }
    None => {}
}

if count < 10 {
    return Ok((Quaternion::identity(), (0.0, 0.0, 0.0)));
}

let inv_n = 1.0 / (count as f64);
let gyro_bias_dps = (
    gyro_sum.0 * inv_n,
    gyro_sum.1 * inv_n,
    gyro_sum.2 * inv_n,
);
let accel_mean_g = (
    accel_sum.0 * inv_n,
    accel_sum.1 * inv_n,
    accel_sum.2 * inv_n,
);

let (roll0, pitch0) = attitude_from_accel(accel_mean_g.0, accel_mean_g.1, accel_mean_g.2);
let yaw0 = 0.0_f64;

let q0 = Quaternion::from_euler(roll0, pitch0, yaw0);

Ok((q0, gyro_bias_dps))

```

```

}

// ====
// 
// "CALIBRATE"
// "CALIBRATION_DONE"
// 1. : "DATA_Q,seq,request_seq,qw,qx,qy,qz" ( )
// 2. UDP: "DATA,seq,request_seq,ex_x,ex_y,ex_z,ey_x,ey_y,ey_z" (Unity )

fn main() -> Result<()> {
    //
    // : imu_orientation_quat.exe COM8
    let args: Vec<String> = std::env::args().collect();
    let port_name = if args.len() >= 2 {
        args[1].clone()
    } else {
        //
        let ports = serialport::available_ports()
            .context(" ")?;
        if ports.is_empty() {
            anyhow::bail!(" ");
        }
        ports[0].port_name.clone()
    };
    let baud = 115200;

    let mut port = serialport::new(&port_name, baud)
        .timeout(Duration::from_millis(200))
        .open()
        .with_context(|| format!(" {} ", port_name))?;

    port.write_data_terminal_ready(true)?;
    port.write_request_to_send(true)?;

    //
    std::thread::sleep(Duration::from_millis(500));

    //
    let (mut q, mut gyro_bias_dps) = calibrate(&mut *port, 2.0)?;
}

```

```

//  

clear_serial_buffer(&mut *port)?;  
  

let alpha = 0.98_f64;  
  

// ===== UDP =====  

let unity_ip = "127.0.0.1:50005"; // Unity UDP  

let command_port = "127.0.0.1:50006"; // Unity  
  

let udp_socket = UdpSocket::bind(command_port)  

    .context("UDP")?;  

udp_socket.set_nonblocking(true)  

    .context("UDP")?;  
  

eprintln!("[INFO] UDP : {}", unity_ip);  

eprintln!("[INFO] UDP : {}", command_port);  
  

// ( )  

let calibrate_flag = Arc::new(Mutex::new(false));  

let calibrate_flag_clone = Arc::clone(&calibrate_flag);  

let calibrate_flag_udp = Arc::clone(&calibrate_flag);  
  

//  

thread::spawn(move || {  

    let stdin = io::stdin();  

    let reader = BufReader::new(stdin);  

    for line in reader.lines() {  

        if let Ok(line) = line {  

            let trimmed = line.trim().to_string();  

            if trimmed == "CALIBRATE" {  

                let mut flag = calibrate_flag_clone.lock().unwrap();  

                *flag = true;  

            }  

        }  

    }  

});  
  

// ===== stdout & =====  

let stdout = io::stdout();  

let mut out = stdout.lock();
```

```

// Python           request_seq
writeln!(out, "# seq,request_seq,qw,qx,qy,qz")?;
out.flush()?;

// 
let mut last_packet_time: Option<Instant> = None;

loop {
    // UDP
    {
        let mut buf = [0u8; 1024];
        match udp_socket.recv_from(&mut buf) {
            Ok((len, _src)) => {
                if let Ok(msg) = std::str::from_utf8(&buf[..len]) {
                    if msg.trim() == "CALIBRATE" {
                        let mut flag = calibrate_flag_udp.lock().unwrap();
                        *flag = true;
                        eprintln!("[INFO] Received CALIBRATE command via UDP");
                    }
                }
            }
            Err(ref e) if e.kind() == std::io::ErrorKind::WouldBlock => {
                //
            }
            Err(e) => {
                eprintln!("[WARN] UDP receive error: {}", e);
            }
        }
    }

    // 
    {
        let mut flag = calibrate_flag.lock().unwrap();
        if *flag {
            *flag = false;
            drop(flag); //
            drop(out); // stdout

            //
            let (new_q, new_gyro_bias) = calibrate(&mut *port, 2.0)?;
            q = new_q;
        }
    }
}

```

```

gyro_bias_dps = new_gyro_bias;

//  

clear_serial_buffer(&mut *port)?;  

//  

last_packet_time = None;  

// stdout  

let stdout = io::stdout();  

out = stdout.lock();  

//  

writeln!(out, "CALIBRATION_DONE")?;  

out.flush()?;
//      UDP  Unity  

let calibration_done_msg = "CALIBRATION_DONE";  

if let Err(e) = udp_socket.send_to(calibration_done_msg.as_bytes(), unity_ip) {  

    eprintln!("[WARN] Failed to send CALIBRATION_DONE via UDP: {}", e);  

}  

}  

}  

if let Some(pkt) = read_packet(&mut *port)? {  

    let now = Instant::now();  

//  

let actual_dt = if let Some(last_time) = last_packet_time {  

    now.duration_since(last_time).as_secs_f64()  

} else {  

    DT //  

};  

last_packet_time = Some(now);  

let (gx_dps_raw, gy_dps_raw, gz_dps_raw) = gyro_to_dps(pkt.gx, pkt.gy, pkt.gz);  

let (ax_g, ay_g, az_g) = accel_to_g(pkt.ax, pkt.ay, pkt.az);  

let gx_dps = gx_dps_raw - gyro_bias_dps.0;  

let gy_dps = gy_dps_raw - gyro_bias_dps.1;  

let gz_dps = gz_dps_raw - gyro_bias_dps.2;

```

```

// dt
update_orientation_quat(
    &mut q,
    gx_dps,
    gy_dps,
    gz_dps,
    ax_g,
    ay_g,
    az_g,
    actual_dt,
    alpha,
);

```

// 1. request_seq

```

writeln!(
    out,
    "DATA_Q,{},{},{}{:.9}{:.9}{:.9}{:.9}",
    pkt.seq, pkt.request_seq, q.w, q.x, q.y, q.z
)?;
out.flush()?;

```

// 2. Unity UDP request_seq

```

let ((ex_x, ex_y, ex_z), (ey_x, ey_y, ey_z)) = q.to_rotation_vectors();
let udp_msg = format!(
    "DATA{},{}{:.9}{:.9}{:.9}{:.9}{:.9}{:.9}",
    pkt.seq, pkt.request_seq, ex_x, ex_y, ex_z, ey_x, ey_y, ey_z
);

if let Err(e) = udp_socket.send_to(udp_msg.as_bytes(), unity_ip) {
    // UDP Unity
    if cfg!(debug_assertions) {
        eprintln!("[WARN] UDP send error: {}", e);
    }
}
}
}

```

13.0.4 姿勢推定プログラムのライブラリ

```
[package]
name = "imu_orientation_rust"
version = "0.1.0"
edition = "2021"

[dependencies]
serialport = "4.3"
anyhow = "1.0"
socket2 = "0.5"
```

13.0.5 姿勢推定プログラムの結果をプレビューするためのプログラムのソースコード

```
import math
import subprocess
import os
from vpython import canvas, vector, box, arrow, color, rate, label

# ===== Rust =====
RUST_BIN_PATH = r"./target/release imu_orientation_rust.exe" #

# =====
PORT_NAME = "COM10" #

# =====
AXIS_REMAP = [0, 2, 1] #
AXIS_SIGN = [1, -1, -1] # Y Z

def apply_R(R, v):
    x = R[0][0] * v[0] + R[0][1] * v[1] + R[0][2] * v[2]
    y = R[1][0] * v[0] + R[1][1] * v[1] + R[1][2] * v[2]
    z = R[2][0] * v[0] + R[2][1] * v[1] + R[2][2] * v[2]
    return (x, y, z)

def remap_axes(vec, axis_remap, axis_sign):
    return (
        vec[axis_remap[0]] * axis_sign[0],
        vec[axis_remap[1]] * axis_sign[1],
        vec[axis_remap[2]] * axis_sign[2],
```

```

)

def quat_to_rotmat(qw, qx, qy, qz):
    """ -> 3x3"""

n = math.sqrt(qw * qw + qx * qx + qy * qy + qz * qz)
if n == 0.0:
    qw, qx, qy, qz = 1.0, 0.0, 0.0, 0.0
else:
    qw /= n
    qx /= n
    qy /= n
    qz /= n

R = [[0.0] * 3 for _ in range(3)]

R[0][0] = 1 - 2 * (qy * qy + qz * qz)
R[0][1] = 2 * (qx * qy - qz * qw)
R[0][2] = 2 * (qx * qz + qy * qw)

R[1][0] = 2 * (qx * qy + qz * qw)
R[1][1] = 1 - 2 * (qx * qx + qz * qz)
R[1][2] = 2 * (qy * qz - qx * qw)

R[2][0] = 2 * (qx * qz - qy * qw)
R[2][1] = 2 * (qy * qz + qx * qw)
R[2][2] = 1 - 2 * (qx * qx + qy * qy)

return R


def quat_to_euler(qw, qx, qy, qz):
    """ -> (roll, pitch, yaw) [rad]"""

# roll (x-axis)
sinr_cosp = 2 * (qw * qx + qy * qz)
cosr_cosp = 1 - 2 * (qx * qx + qy * qy)
roll = math.atan2(sinr_cosp, cosr_cosp)

# pitch (y-axis)

```

```

sinp = 2 * (qw * qy - qz * qx)
if abs(sinp) >= 1:
    pitch = math.copysign(math.pi / 2, sinp)
else:
    pitch = math.asin(sinp)

# yaw (z-axis)
siny_cosp = 2 * (qw * qz + qx * qy)
cosy_cosp = 1 - 2 * (qy * qy + qz * qz)
yaw = math.atan2(siny_cosp, cosy_cosp)

return roll, pitch, yaw

def main():
    #
    axis_names = ["X", "Y", "Z"]
    remap_str = f"{axis_names[AXIS_REMAP[0]}}, {axis_names[AXIS_REMAP[1]}}, {axis_names[AXIS_REMAP[2]]}"
    sign_str = (
        f"+ if AXIS_SIGN[0] > 0 else -}X, "
        f"+ if AXIS_SIGN[1] > 0 else -}Y, "
        f"+ if AXIS_SIGN[2] > 0 else -}Z"
    )
    print("=" * 50)
    print("  :")
    print(f"  AXIS_REMAP: {AXIS_REMAP} -> (X,Y,Z) -> ({remap_str})")
    print(f"  AXIS_SIGN: {AXIS_SIGN} -> {sign_str}")
    print("=" * 50)

    # Rust
    print(f"[DEBUG] Starting Rust process: {RUST_BIN_PATH}")
    print(f"[DEBUG] Port: {PORT_NAME}")

try:
    proc = subprocess.Popen(
        [RUST_BIN_PATH, PORT_NAME],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True,
        encoding="utf-8",

```

```

        errors="ignore",
        bufsize=1,  #
    )
    print(f"[DEBUG] Rust process started with PID: {proc.pid}")
except Exception as e:
    print(f"[ERROR] Failed to start Rust process: {e}")
    return

calibrating = False
calibration_requested = False

# VPython
scene = canvas(
    title="IMU 3D Orientation (Quat)",
    width=800,
    height=600,
    center=vector(0, 0, 0),
    background=color.white,
)
scene.range = 2

board = box(
    pos=vector(0, 0, 0),
    length=1.5,  # X
    height=1.0,  # Y
    width=0.1,   # Z
    color=color.blue,
    opacity=0.7,
)

x_axis = arrow(
    pos=vector(0, 0, 0),
    axis=vector(1.5, 0, 0),
    color=color.red,
    shaftwidth=0.03,
)
y_axis = arrow(
    pos=vector(0, 0, 0),
    axis=vector(0, 0, 1.5),
    color=color.green,
    shaftwidth=0.03,
)

```

```

)
z_axis = arrow(
    pos=vector(0, 0, 0),
    axis=vector(0, 1.5, 0),
    color=color.blue,
    shaftwidth=0.03,
)

label(pos=vector(1.7, 0, 0), text="X", xoffset=10, height=12, box=False)
label(pos=vector(0, 0, 1.7), text="Y", xoffset=10, height=12, box=False)
label(pos=vector(0, 1.7, 0), text="Z", xoffset=10, height=12, box=False)

text_label = label(
    pos=vector(0, -1.7, 0),
    text="Roll: 0.0, Pitch: 0.0, Yaw: 0.0",
    height=14,
    box=False,
    color=color.black,
)

calib_label = label(
    pos=vector(0, 1.5, 0),
    text="Press 'C' to calibrate",
    height=12,
    box=True,
    color=color.black,
    background=color.yellow,
    opacity=0.8,
)

#
#  

def on_keydown(evt):
    nonlocal calibration_requested
    if evt.key == "c" or evt.key == "C":
        calibration_requested = True

scene.bind("keydown", on_keydown)

if proc.stdout:
    try:
        os.set_blocking(proc.stdout.fileno(), False)

```

```

except Exception as e:
    print(f"[WARN] os.set_blocking for stdout failed: {e}")

if proc.stderr:
    try:
        os.set_blocking(proc.stderr.fileno(), False)
    except Exception as e:
        print(f"[WARN] os.set_blocking for stderr failed: {e}")

try:
    while True:
        rate(120)

        #
        if calibration_requested and not calibrating:
            calibration_requested = False
            calibrating = True
            calib_label.text = "Calibrating... Keep sensor still!"
            calib_label.background = color.red

        if proc.stdin:
            print("[DEBUG] Sending CALIBRATE...")
            proc.stdin.write("CALIBRATE\n")
            proc.stdin.flush()

        if proc.stdout is None:
            break

# stderr
if proc.stderr:
    try:
        while True:
            err_line = proc.stderr.readline()
            if not err_line:
                break
            print(f"[RUST STDERR] {err_line.strip()}")
    except (BlockingIOError, IOError):
        pass

lines = []
try:

```

```

while True:
    line = proc.stdout.readline()
    if not line:
        break
    lines.append(line.strip())
except (BlockingIOError, IOError):
    pass

if not lines:
    continue

latest_data_line = None

for line in reversed(lines):
    if not line:
        continue

    # DATA_Q
    if not line.startswith("DATA_Q,") and not line.startswith("#"):
        print(f"[DEBUG] from Rust: '{line}'")

    # DATA_Q
    if line.startswith("DATA_Q,"):
        if not hasattr(main, 'data_q_debug_count'):
            main.data_q_debug_count = 0
        if main.data_q_debug_count < 5:
            print(f"[DEBUG] DATA_Q line: '{line}'")
            main.data_q_debug_count += 1

    if line == "CALIBRATION_DONE":
        print("[DEBUG] Calibration done!")
        calibrating = False
        calib_label.text = "Calibration Complete! Press 'C' to recalibrate"
        calib_label.background = color.green

    if latest_data_line is None and line.startswith("DATA_Q,"):
        latest_data_line = line

    if latest_data_line is None:
        continue

```

```

# DATA_Q, seq, request_seq, qw, qx, qy, qz
parts = latest_data_line.split(",")

# :
if len(parts) != 7:
    print(f"[DEBUG] Unexpected packet format: {len(parts)} fields, expected 7")
    print(f"[DEBUG] Data: {latest_data_line}")
    continue

_, seq_s, request_seq_s, qw_s, qx_s, qy_s, qz_s = parts
try:
    seq = int(seq_s)
    request_seq = int(request_seq_s)
    qw = float(qw_s)
    qx = float(qx_s)
    qy = float(qy_s)
    qz = float(qz_s)
except ValueError as e:
    print(f"[DEBUG] Parse error: {e}")
    print(f"[DEBUG] Data: {latest_data_line}")
    continue

R = quat_to_rotmat(qw, qx, qy, qz)

ex_w = apply_R(R, (1.0, 0.0, 0.0))
ey_w = apply_R(R, (0.0, 1.0, 0.0))

ex_remapped = remap_axes(ex_w, AXIS_REMAP, AXIS_SIGN)
ey_remapped = remap_axes(ey_w, AXIS_REMAP, AXIS_SIGN)

board.axis = vector(*ex_remapped)
board.up = vector(*ey_remapped)

roll, pitch, yaw = quat_to_euler(qw, qx, qy, qz)
roll_deg = math.degrees(roll)
pitch_deg = math.degrees(pitch)
yaw_deg = math.degrees(yaw)

text_label.text = (
    f"Seq: {seq} "
    f"Pulse: {request_seq} "

```

```
f"Roll: {roll_deg:6.2f} deg, "
f"Pitch: {pitch_deg:6.2f} deg, "
f"Yaw: {yaw_deg:6.2f} deg"
)

finally:
    try:
        proc.terminate()
    except Exception:
        pass

if __name__ == "__main__":
    main()
```