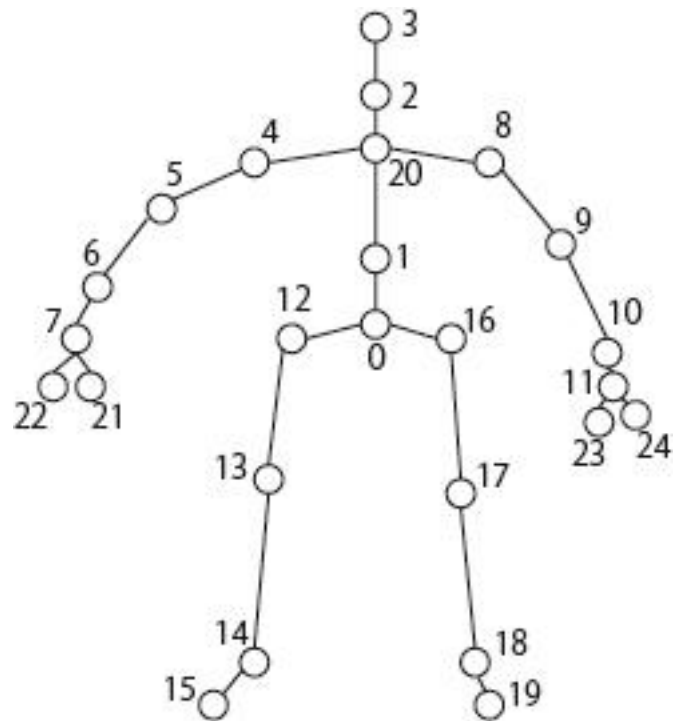


BODY

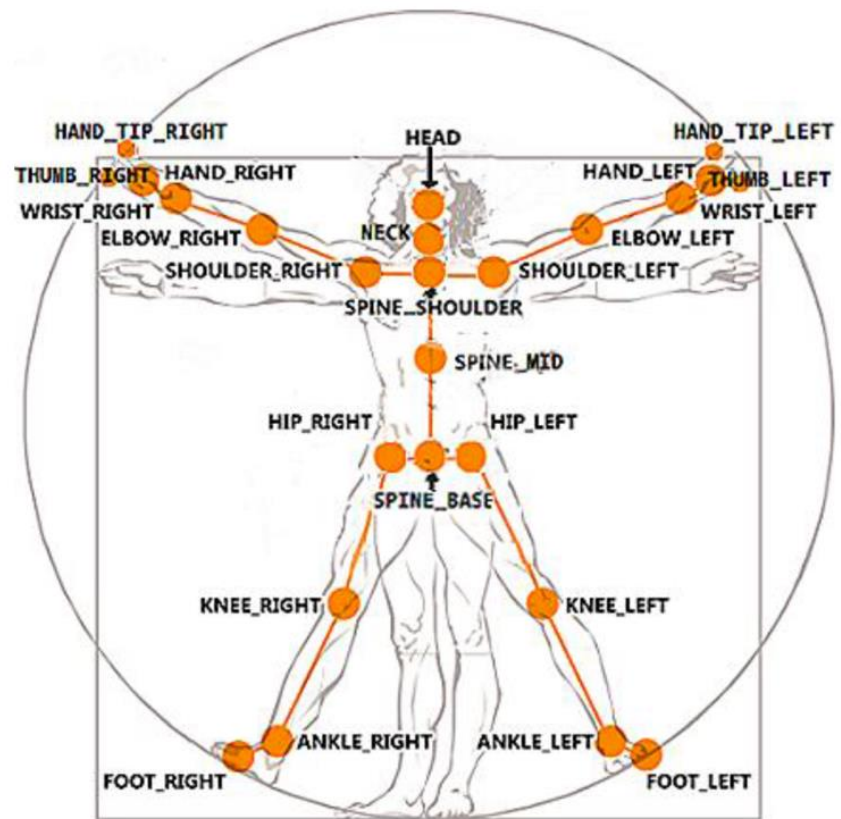
プレゼミ (田中)

Bodyデータとは

Kinect v2で検知できるのは25点



関節固有の番号



名前

```

#include <iostream>
#include <sstream>

#include <Kinect.h>
#include <openv2%openov.hpp>

#include <atlbase.h>

// 次のように使います
// ERROR_CHECK ( ::GetDefaultKinectSensor( &kinect ) );
// 書籍での解説のためにマクロにしています。実際には展開した形で使うことを検討してください。
#define ERROR_CHECK( ret ) *
if ( (ret) != S_OK ) { *
    std::stringstream ss; *
    ss << "failed " #ret " << std::hex << ret << std::endl; *
    throw std::runtime_error( ss.str().c_str() ); *
}

class KinectApp
{
private:
    CComPtr<IKinectSensor> kinect = nullptr;

    CComPtr<IBodyFrameReader> bodyFrameReader = nullptr;
    [Body* bodies[6];

public:
    ~KinectApp()
    {
        // Kinectの動作を終了する
        if ( kinect != nullptr ) {
            kinect->Close();
        }
    }

    // 初期化
    void Initialize()
    {
        // デフォルトのKinectを取得する
        ERROR_CHECK( ::GetDefaultKinectSensor( &kinect ) );
        ERROR_CHECK( kinect->Open() );

        // ボディリーダーを取得する
        CComPtr<IBodyFrameSource> bodyFrameSource;
        ERROR_CHECK( kinect->get_BodyFrameSource( &bodyFrameSource ) );
        ERROR_CHECK( bodyFrameSource->OpenReader( &bodyFrameReader ) );

        for ( auto& body : bodies ) {
            body = nullptr;
        }
    }

    void run()
    {
        while ( 1 ) {
            update();
            draw();

            auto key = cv::waitKey( 10 );
            if ( key == 'q' ) {
                break;
            }
        }
    }

private:
    // データの更新処理
    void update()
    {
        updateBodyFrame();
    }

```

```

}

// ボディフレームの更新
void updateBodyFrame()
{
    // フレームを取得する
    CComPtr<IBodyFrame> bodyFrame;
    auto ret = bodyFrameReader->AcquireLatestFrame( &bodyFrame );
    if ( FAILED( ret ) ) {
        return;
    }

    // 前回のBodyを解放する
    for ( auto& body : bodies ) {
        if ( body != nullptr ) {
            body->Release();
            body = nullptr;
        }
    }

    // データを取得する
    ERROR_CHECK( bodyFrame->GetAndRefreshBodyData( 5, &bodies[0] ) );
}

void draw()
{
    drawBodyIndexFrame();
}

void drawBodyIndexFrame()
{
    // 関節の座標をDepth座標系で表示する
    cv::Mat bodyImage = cv::Mat::zeros( 424, 512, CV_8UC4 );

    for ( auto body : bodies ) {
        if ( body == nullptr ) {
            continue;
        }

        BOOLEAN isTracked = false;
        ERROR_CHECK( body->get_IsTracked( &isTracked ) );
        if ( !isTracked ) {
            continue;
        }

        // 関節の位置を表示する
        Joint joints[JointType::JointType_Count];
        body->GetJoints( JointType::JointType_Count, joints );
        for ( auto joint : joints ) {
            // 手の位置が追跡状態
            if ( joint.TrackingState == TrackingState::TrackingState_Tracked ) {
                drawEllipse( bodyImage, joint, 10, cv::Scalar( 255, 0, 0 ) );
            }
            // 手の位置が推測状態
            else if ( joint.TrackingState == TrackingState::TrackingState_Inferred ) {
                drawEllipse( bodyImage, joint, 10, cv::Scalar( 255, 255, 0 ) );
            }
        }
    }

    cv::imshow( "Body Image", bodyImage );
}

void drawEllipse( cv::Mat& bodyImage, const Joint& joint, int r, const cv::Scalar& color )
{
    // カメラ座標系をDepth座標系に変換する
    CComPtr<ICoordinateMapper> mapper;
    ERROR_CHECK( kinect->get_CoordinateMapper( &mapper ) );

    DepthSpacePoint point;
    mapper->MapCameraPointToDepthSpace( joint.Position, &point );

    cv::circle( bodyImage, cv::Point( point.X, point.Y ), r, color, -1 );
}

```

変換後の座標系

Kinect の使用には 4 フレーム

depth画像サイズ

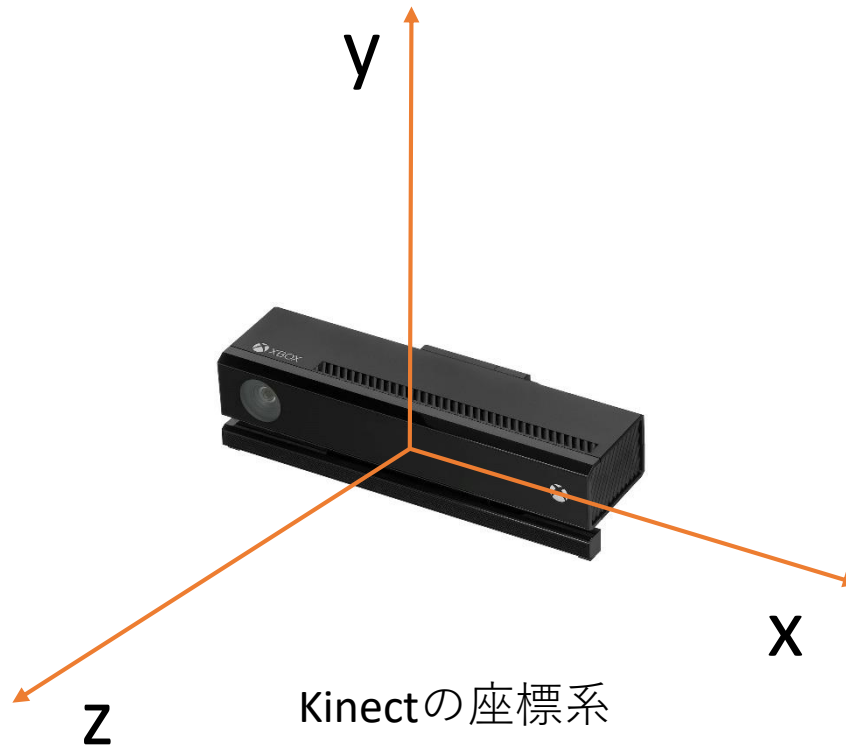
この値は 関節の状態が入る

青

シブシブ

変換された座標

```
};  
}  
void main()  
{  
    try {  
        KinectApp app;  
        app.initialize();  
        app.run();  
    }  
    catch (std::exception& ex){  
        std::cout << ex.what() << std::endl;  
    }  
}
```



3D座標の取得

```
class KinectApp
{
private:
.....

    int joint_no;
    Int gamenx[25], gameny[25];
    double d3x[25], d3y[25], d3z[25];
.....

void drawBodyFrame()
{
    .....
    joint_no = -1;
    // 関節の位置を表示する
    Joint joints[JointType::JointType_Count];
    body->GetJoints( JointType::JointType_Count, joints );
    for ( auto joint : joints ) {
        joint_no++;
        drawEllipse( bodyImage, joint, 10, cv::Scalar( 255, 0, 0 ) );
    }
}
```

```
void drawEllipse( cv::Mat& bodyImage, const Joint& joint, int r, const cv::Scalar& color )
{
    // カメラ座標系をDepth座標系に変換する
    CComPtr<ICoordinateMapper> mapper;
    ERROR_CHECK( kinect->get_CoordinateMapper( &mapper ) );

    DepthSpacePoint point;
    mapper->MapCameraPointToDepthSpace( joint.Position, &point );
    cv::circle( bodyImage, cv::Point( point.X, point.Y ), r, color, -1 );
    //画面座標を保存
    gamenx[joint_no] = point.X;
    gameny[joint_no] = point.Y;
    //3次元座標にする
    d3x[joint_no] = joint.Position.X * 100 ; // [cm]
    d3y[joint_no] = joint.Position.Y * 100;
    d3z[joint_no] = joint.Position.Z * 100;
}
```

点の間に線分を描く

2点の間に線分を描く

```
void line(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
```

パラメータ:

`img` – 画像.

`pt1` – 線分の 1 番目の端点.

`pt2` – 線分の 2 番目の端点.

`color` – 線分の色.

`thickness` – 線分の太さ.

`lineType` –

線分の種類:

8 (あるいは, 省略時) 8連結.

4 4連結.

`CV_AA` アンチエイリアス.

`shift` – 端点の座標において, 小数点以下の桁を表すビット数.

例

```
cv::line(bodyImage, cv::Point(gamenx[20], gameny[20]), cv::Point(gamenx[4], gameny[4]), cv::Scalar(0, 0, 200), 3, 4);
```

