SmartAR™ SDK 概要

1.1 版

変更履歴

版数	区分	項目	変更内容	日付
1.0	新規	-	新規作成	2016/01/27
1.1	更新	-	ライセンス認証について、一部記述を追記	2016/03/28

目次

1 概要

はじめに

SmartAR™ SDK は Android, iOS を対象としたマルチプラットフォーム対応の AR（Augmented Reality：拡張現実感）アプリケーション開発環境です。
開発者は本 SDK が提供する API 群を使用することにより、拡張現実を実現する技術である SmartAR™ を使用したアプリケーションをマルチプラットフォーム対応で容易に開発することができます。
このドキュメントでは SmartAR™ SDK の概要について説明します。

主な機能と用途

SmartAR™ SDK は、認識処理を行う SarRecognizer クラス、プラットフォームのカメラデバイスを抽象化した SarCameraDevice クラス、プラットフォームのセンサーデバイスを抽象化した SarSensorDevice クラス、カメラ画像の描画を行う SarCameraImageDrawer クラスといったクラス群を中心に構成されており、各クラスは Android, iOS に対して共通のインターフェースを提供します。

図 1 は SmartAR™ SDK と SmartAR™ SDK を使用するアプリケーションの典型的な構成を示しています。本 SDK と OpenGL を使用した AR アプリケーションのコア部分を各プラットフォーム共通の実装とすることで、マルチプラットフォーム対応アプリケーションを効率的に開発できます。

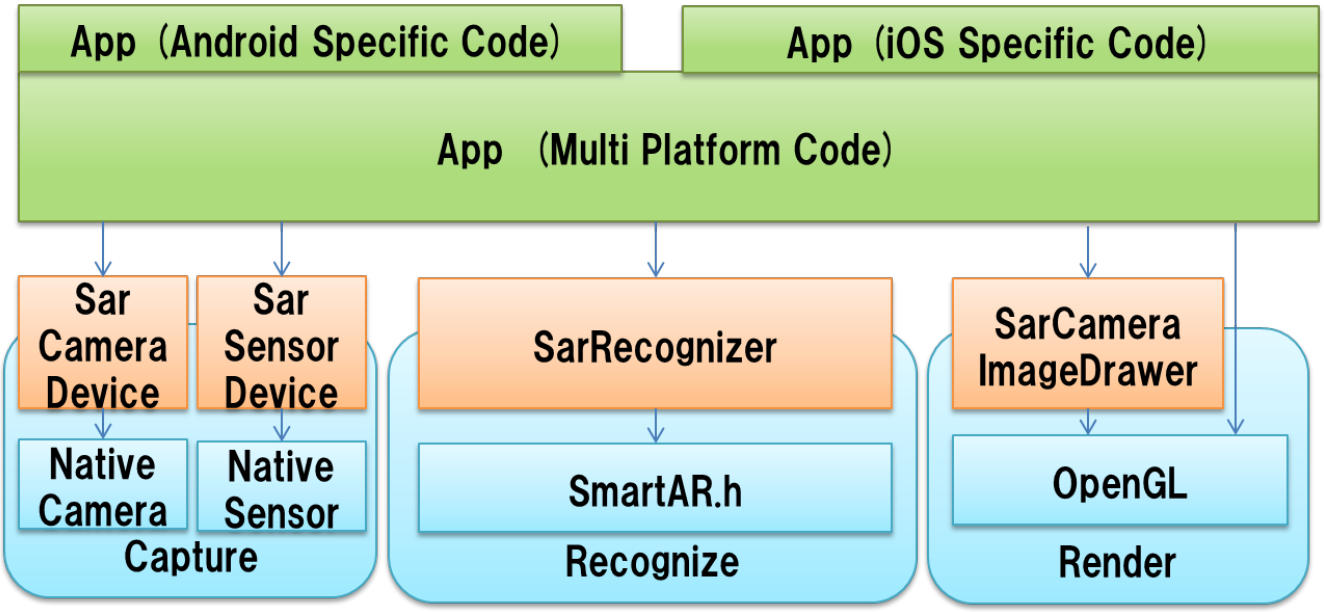

図 1 アプリケーションと SmartAR™ SDK の構成図

SmartAR™ SDK を用いた典型的なアプリケーションは、図 2 に示すような流れで処理を行います。

アプリケーションは SarCameraDevice と SarSensorDevice からカメラ画像とセンサー情報を取得し、これらを SarRecognizer へ渡して認識処理を行います。
その後、カメラ画像と、SarRecognizer から取得した認識結果に基づいた AR コンテンツを重ね合わせて描画することにより、AR を実現します。

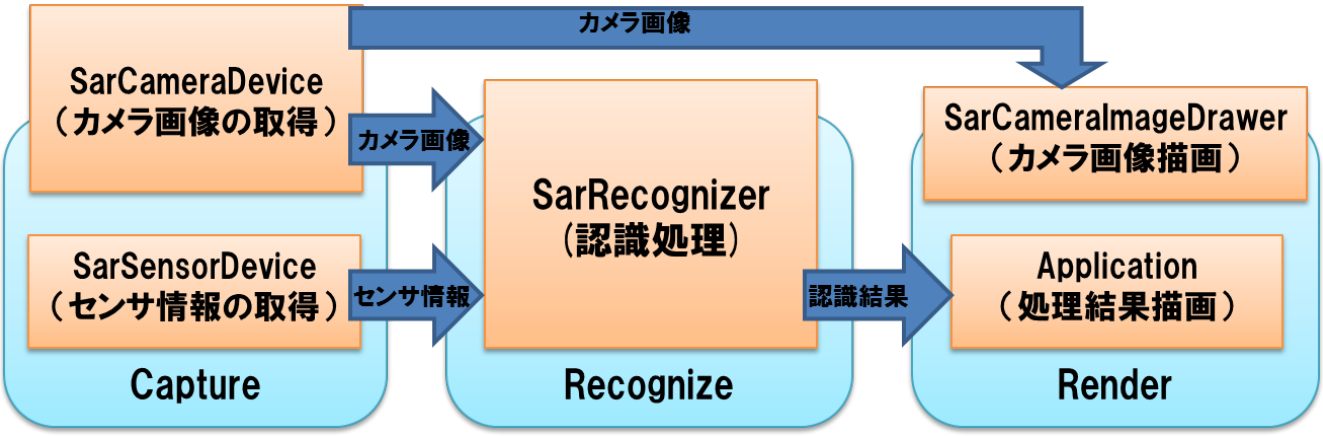

図 2　SmartAR™ SDK による処理の流れ

SarRecognizer には 2 つの動作モードがあり、それぞれ TargetTracking モード、SceneMapping モードと呼ばれます。
これらのモードの概要は以下の通りです。

TargetTracking モード

TargetTracking モードは、指定された認識対象をカメラ画像内から検出・追跡し、認識対象に対する SmartAR™動作端末のポーズ（位置と向き）を推定するためのモードです。任意の自然画像を認識対象とすることができます。TargetTracking モードの処理の詳細については、「3 参考情報」の「Target Tracking モードの処理の仕組み」を参照してください。

SceneMapping モード

SceneMapping モードは、SmartAR™動作端末を空間中で動かすことで、カメラ画像の変化とセンサー情報から、カメラに映った未知の空間の 3D 構造と、その空間内での SmartAR™動作端末のポーズ（位置と向き）を推定するモードです。SLAM（Simultaneous Localization and Mapping）と呼ばれる技術を使用しています。SceneMapping モードの処理の詳細については、「3 参考情報」の「SceneMapping モードの処理の仕組み」を参照してください。

対応環境について

SmartAR™ SDK が対応するプラットフォームは以下のとおりです。

OS 名	バージョン
Android	4.0.3 以降
iOS	8.0 以降

プログラムへの組み込み

SmartAR™ SDK を使用するために必要なファイルは次のとおりです。

ファイル名	説明
SarCameraDevice.h	SarCameraDevice クラス及びその関連クラスを定義するヘッダファイル
SarCameraImageDrawer.h	SarCameraImageDrawer クラスを定義するヘッダファイル
SarCommon.h	基本的な定数・クラスなどを定義するヘッダファイル
SarPlatformDef.h	SDK 内部で使用されるヘッダファイル。 アプリケーション開発者が直接使用することはありません。
SarRecognizer.h	SarRecognizer クラス及びその関連クラスを定義するヘッダファイル
SarSensorDevice.h	SarSensorDevice 及びその関連クラスを定義するヘッダファイル
SarScreenDevice.h	SarScreenDevice クラスを定義するヘッダファイル
SarSmart.h	SarSmart クラスを定義するヘッダファイル
SarUtility.h	ユーティリティ機能を提供するヘッダファイル
libsmartar.a	iOS 用のスタティックリンクライブラリ
libsmartar.so	Android 用のダイナミックリンクライブラリ
smartar.jar	Android 用の jar ファイル

「SmartAR™ SDK-Reference.pdf」を参照し、アプリケーションが必要とするヘッダファイルをインクルードして使用してください。

リンクするライブラリはプラットフォームごとに異なります。
Android を実行環境とする場合は smartar.jar をプロジェクトに追加し、libsmartar.so を次のようにプログラム中からロードしてください。

例）libsmartar.so のロード

```
public class Sample {
    ～(省略)～
    static {
        System.loadLibrary("smartar");
    }
    ～(省略)～
}
```

iOS を実行環境とする場合は libsmartar.a をプロジェクトに追加してください。

ライブラリのアップデート方法

ヘッダファイルとライブラリファイルを上書きコピーし、再度ビルドしなおしてください。

ライブラリについて

各ファイルには、以下の役割があります。

	AR コア機能	API 機能	デバイス制御機能
Android	libsmartar.so		smartar.jar
iOS	libsmartar.a		

Android、iOS は OS が標準で提供している API を使ってカメラ、センサーをサポートします。

サンプルプログラム

SmartAR™ SDK を使ったサンプルプログラムとして、Android 版、iOS 版がありますので必要に応じて参照してください。
サンプルプログラムは 2 種類あります。
sample_api_simple は必要最小限の実行構成を示したプログラムです。
sample_api_full はライブラリが持つ各種 API の動作実行を行うことができるプログラムです。

dictool

任意の自然画像を認識対象とする辞書を作成するためのツールです。
詳細については、次の章の「認識対象画像用辞書作成ツールの使い方」節を参照してください。

ライセンス認証について

SmartAR™ SDK はライセンス認証を行います。
ライセンスファイルを、Android の場合は assets フォルダ以下に配置、iOS の場合はプロジェクトに追加して、アプリケーションのリソースとして含まれるようにし、SarSmart の初期化時にファイルパスを指定することで認証を行うことができます。
ライセンスが認証されていない状態でも SmartAR™ を使用することが出来ますが、以下の機能が制限されます。

- SarCameraDevice から取得した以外の画像を認識処理に渡すことができません
- SarCameraDevice から取得したカメラ画像を描画するためには SarCameraImageDrawer を使用する必要があります
- カメラ画像描画には Watermark が付与されます
- ライブラリの画像キャプチャ機能を使用してカメラ画像を取得することができません

具体的な API の使用制限については「SmartARSDK-Reference_j.pdf」を参照して下さい。

サンプルプログラムではライセンスファイルとして license.sig というファイルをリソースとして持ち、ライセンス認証を行っています。（ダミーファイルのため、実際には認証処理に失敗します。）
SarSmart の初期化処理は SDK/Sample/mobile_common/sample_simple/sample_simple.cc の初期化部分を参考にすることが出来ます。

コード例)

```
SarSmart* smart = new SarSmart(nativeContext, nativeEnv, "license.sig");
```

2 使用方法

TargetTracking モードの処理の概要

TargetTracking モードは、認識対象として登録した自然画像をカメラ画像内から探索し、検出した自然画像から SmartAR™動作端末のポーズ(位置、向き)を推定するモードです。カメラ画像内に自然画像が存在し、正常に認識されている間、継続的に SmartAR™動作端末のポーズをリアルタイムに推定し続けることが可能です。

自然画像の認識に成功すると、SarRecognizer から取得した認識結果(SarRecognitionResult)の isRecognized_メンバの値が true になります。

TargetTracking モードの使用方法

認識処理はシングルスレッドまたはマルチスレッドの何れでも行えます。
一般に、マルチコアのデバイスではマルチスレッド動作の方が良好なパフォーマンスを得ることができます。
以下に、それぞれの実装の疑似コードを示します。

*** シングルスレッドで認識処理を行う疑似コード**

```
void main_thread()
{
    // (1) 初期化処理
    SarSmart* smart = new SarSmart();
    recognizer = new SarRecognizer(smart);

    target = new SarLearnedImageTarget(stream);
    recognizer->setTargets(target);

    cameraDevice = new SarCameraDevice(smart);
    cameraDevice->setVideoImageListener(myVideoImageListener);
    cameraDevice->getCameraDeviceInfo(&cameraDeviceInfo);
    recognizer->setCameraDeviceInfo(&cameraDeviceInfo);

    sensorDevice = new SarSensorDevice(smart);
    sensorDevice->setSensorListener(mySensorListener);
    sensorDevice->getSensorDeviceInfo(&sensorDeviceInfo);
    recognizer->setSensorDeviceInfo(&sensorDeviceInfo);

    cameraDevice->start();
    sensorDevice->start();
}

// センサ情報取得
class MySensorListener : public SarSensorListener
{
public:
    virtual void onSensor(sensorState)
    {
        sensorStates.add(sensorState);
    }
};
```

```
//画像取得と認識処理の実行
class MyVideoImageListener : public SarCameraImageListener
{
public:
    virtual void onImage(imageHolder)
    {
        SETUP_REQUEST(imageHolder, sensorStates, request);

        recognizer->run(request);
    }
};

//終了処理
void end_app()
{
    cameraDevice->stop();
    sensorDevice->stop();

    delete cameraDevice;
    delete sensorDevice;

    delete recognizer;
    delete target;
    delete smart;
}

//描画処理
void render_thread()
{
    cameraImageDrawer = new SarCameraImageDrawer(smart);
    cameraImageDrawer->start();
    while (...)
    {
        cameraImageDrawer->draw(image);

        recognizer->getResults(&results, maxResults);
        if (results[0].isRecognized_)
        {
            DRAW_AR_CONTENTS();
        }
    }
    cameraImageDrawer->stop();
    delete cameraImageDrawer;
}
```

*** マルチスレッドで認識を行う疑似コード**

```
// 初期化処理
void start_app()
{
    Smart* smart = new SarSmart();
    recognizer = new SarRecognizer(smart);
    recognizer->setWorkDispatchedListener(myWorkDispatchedListener);

    target = new SarLearnedImageTarget(stream);
    recognizer->setTargets(target);

    cameraDevice = new SarCameraDevice(smart);
```

```
    cameraDevice->setVideoImageListener(myVideoImageListener);
    cameraDevice->getCameraDeviceInfo(&cameraDeviceInfo);
    recognizer->setCameraDeviceInfo(cameraDeviceInfo);

    sensorDevice = new SarSensorDevice(smart);
    sensorDevice->setSensorListener(mySensorListener);
    sensorDevice->getSensorDeviceInfo(&sensorDeviceInfo);
    recognizer->setSensorDeviceInfo(sensorDeviceInfo);

    cameraDevice->start();
    sensorDevice->start();
    START_WORKER_THREAD();
}

// センサ情報取得
class MySensorListener : public SarSensorListener
{
public:
    virtual void onSensor(sensorState)
    {
        sensorStates.add(sensorState);
    }
};

//画像取得と認識処理の要求
class MyVideoImageListener : public SarCameraImageListener
{
public:
    virtual void onImage(imageHolder)
    {
        SETUP_REQUEST(imageHolder, sensorStates, request);

        recognizer->dispatch(request);
    }
};

//終了処理
void end_app()
{
    FINISH_WORKER_THREAD();
    sensorDevice->stop();
    cameraDevice->stop();

    delete cameraDevice;
    delete sensorDevice;

    delete recognizer;
    delete target;
    delete smart;
}

//認識処理の開始
class MyWorkDispatchedListener : public SarWorkDispatchedListener
{
public:
    virtual void onWorkDispatched()
    {
        RAIZE_EVT_TO_WORKER_TRHEAD();
    }
};
```

```
//認識処理の実行
void worker_thread()
{
    while (...)
    {
        WAIT_EVT();

        recognizer->runWorker();
    }
}

//描画処理
void render_thread()
{
    cameraImageDrawer = new SarCameraImageDrawer(smart);
    cameraImageDrawer->start();
    while (...)
    {
        cameraImageDrawer->draw(image);

        recognizer->getResults(&results, maxResults);
        if (results[0].isRecognized_)
        {
            DRAW_AR_CONTENTS();
        }
    }
    cameraImageDrawer->stop();
    delete cameraImageDrawer;
}
```

(1) 初期化処理

SarSmart、SarRecognizer、SarLearnedImageTarget などのインスタンスを生成し初期化します。
その後 SarRecognizer::setTargets()を呼び出し、SarLearnedImageTarget を SarRecognizer に登録します。
マルチスレッドで動作させる場合は、ワーカースレッドのトリガーとなる SarWorkDispatchedListener を SarRecognizer に登録します。

(2) 認識処理の実行

シングルスレッドでは SarRecognizer::run()を使用し 1 つのスレッドで同期的に認識処理を行うのに対し、
マルチスレッドでは SarRecognizer::dispatch()を呼び出し、認識処理要求を発行する 1 つのスレッドと、SarRecognizer::runWorker()を呼び出し、要求された認識処理を実行する 1 つ以上のワーカースレッドで処理を行います。

SarRecognizer::run()または SarRecognizer::dispatch()の引数に指定する SarRecognitionRequest 構造体には認識処理に必要となる入力データを格納します。
SarRecognitionRequest 構造体の image_メンバには画像データを格納した SarImage クラスを指定する必要があります。SarImage クラスのフォーマットには SAR_IMAGE_FORMAT_L8, SAR_IMAGE_FORMAT_YCBCR420, 又は SAR_IMAGE_FORMAT_YCRCB420 を指定することが可能です。
sensorStates_メンバに SarSensorDevice クラスより取得したセンサー情報を指定することにより、デバイスの動きに対する追従性が向上します。

認識に成功していると、SarRecognizer::getResult()またはSarRecognizer::getResults()で取得した認識結果を格納したSarRecognitionResult構造体のisRecognized_メンバがtrueになります。

(3) 終了処理

SarSmart、SarRecognizer、SarLearnedImageTarget などのインスタンスをデストラクトします。

SceneMapping モードの処理の概要

SceneMapping モードはカメラ画像の変化とセンサー情報からカメラに映った未知の空間の 3D 構造とその空間内での SmartAR 動作端末のポーズ(位置、向き)を推定するモードです。アプリケーションは特定のマーカー画像がカメラに映っていない状態でも、周囲の環境に対応した AR コンテンツを表示することが可能です。
空間の 3D 構造は SmartAR™動作端末の運動視差とセンサー情報により計算されるため、SmartAR™動作端末を空間中で動かす必要があります。

認識する 3D 構造空間内での座標定義方法には複数の選択肢があり、SarRecognizer に初期化モードを指定することで任意のものを選択することができます。

SAR_SCENE_MAPPING_INIT_MODE_TARGET	カメラ画像に写る自然画像の中心点を原点とする
SAR_SCENE_MAPPING_INIT_MODE_HFG	カメラ画像に写る水平面の中心点を原点とする
SAR_SCENE_MAPPING_INIT_MODE_VFG	カメラ画像に写る垂直面の中心点を原点とする
SAR_SCENE_MAPPING_INIT_MODE_SFM	3D 構造から計算した主平面の中心個所を原点とする

SceneMapping モードの仕組み及び、それぞれの初期化モードにおける座標定義方法の詳細については 3 章の参考情報を参照してください。

SceneMapping モードによる認識処理中に何らかの問題が発生した場合、SarRecognizer はローカライズ不可状態に遷移することがあります。その場合にはアプリケーションは、それぞれのアプリケーションのユースケースに応じてユーザへの通知や SarRecognizer のリセットなど、何らかの対応を行う必要があります。このため、アプリケーションは SarRecognizer から取得した認識結果が格納されている SarRecognitionResult 構造体の sceneMappingState_メンバの値が SAR_SCENE_MAPPING_STATE_LOCALIZE_IMPOSSIBLE になっていないか確認することが推奨されます。

SceneMapping モードの使用方法

SceneMapping モードの使用方法は TargetTracking モードを使用した場合とほぼ同様です。
ただし、以下の例のように SarRecognizer の初期化時に初期化モードを指定する必要があります。

例）“SAR_SCENE_MAPPING_INIT_MODE_TARGET”で初期化する場合

```
SarRecognizer* recognizer = new SarRecognizer(
                smart,
                SAR_RECOGNITION_MODE_SCENE_MAPPING,
                SAR_SCENE_MAPPING_INIT_MODE_TARGET);
```

以下に、それぞれの初期化モードについて説明します。

SAR_SCENE_MAPPING_INIT_MODE_TARGET

事前に認識対象画像用辞書作成ツール(dictool)で学習した認識対象（自然画像）で初期化します。世界座標系の定義は、「3 参考情報」の「座標系」を参照してください。
この初期化モードでは SarLearnedImageTarget クラスにより辞書を読み込んだ後、SarRecognizer に SarLearnedImageTarget を登録します。

Note
SarRecognizer へ登録することが可能な SarTarget には、SarLearnedImageTarget の他に SarSceneMappingTarget、SarCompoundTarget があります。詳細は「SmartAR™ SDK-Reference.pdf」を参照してください。

SAR_SCENE_MAPPING_INIT_MODE_HFG

加速度センサーを利用した HFG (Horizontal from Gravity)初期化です。加速度センサーから検出される重力方向をもとに水平面を計算し、水平面から世界座標系を定義します。世界座標系の定義は、「3 参考情報」の「座標系」を参照してください。
SarRecognizer の認識処理開始後、画像に初期化点(2 次元画像内でトラッキングされる点)が十分写っていることを確認できれば、すぐに初期化が完了します。

Note
本モードを使用する際には、必ずセンサー情報を SarRecognitionRequest 構造体に指定してください。

SAR_SCENE_MAPPING_INIT_MODE_VFG

加速度センサーを利用した VFG (Vertical from Gravity)初期化です。
加速度センサーから検出される重力方向と「SmartAR™動作端末は鉛直面に正対する」という前提をもとに鉛直面を計算し、鉛直面から世界座標系を定義します。世界座標系の定義は、「3 参考情報」の「座標系」を参照してください。
初期化時、SmartAR™動作端末と鉛直面は正対している必要があります。SarRecognizer の認識処理開始後、画像に初期化点(2 次元画像内でトラッキングされる点)が十分写っていることを確認できれば、すぐに初期化が完了します。

Note
本モードを使用する際には、必ずセンサー情報を SarRecognitionRequest 構造体に指定してください。

SAR_SCENE_MAPPING_INIT_MODE_SFM

SmartAR™動作端末の運動視差を利用した SFM (Structure from Motion)初期化です。焦点の位置の異なる 2 視点のカメラ画像から、初期化点(2 次元画像内でトラッキングされる点)で構成された 3D 構造を計算します。その後、カメラ画像の中心に写っている局所の主平面(Dominant Plane)を 3D 構造から計算し、主平面から世界座標系を定義します。世界座標系の定義は、「3 参考情報」の「座標系」を参照してください。
認識処理開始後、SarRecognizer は画像に初期化点が十分写っているか確認します。初期化点が十分に写っていたら、SarRecognizer は視差が十分になるまで初期化点をトラッキングし続けます。視差が十分な状態になると初期化処理が完了します。

Note
例えばその場で回転した場合など、視差が不十分だと初期化しません。初期化点が不十分な状態、つまり特徴のない被写体を写した状態では初期化は完了しません。
センサー情報を使用しない場合、被写体が平面だと仮定して処理します。

SAR_SCENE_MAPPING_INIT_MODE_DRY_RUN

このモードは、画像内にいくつの初期化点(2 次元画像内でトラッキングされる点)があるかを調べるためのテストモードです。世界座標系の定義は行いません。
認識処理開始後、SarRecognizer は画像内の初期化点の数と場所を調べ続けます。初期化点の個数と位置は、SarRecognizer::getResults()または SarRecognizer::getResult()で返される SarRecognitionResult 構造体の numInitPoints_と initPoints_メンバで取得してください。初期化点

を使った初期化には、SAR_SCENE_MAPPING_INIT_MODE_HFG か SAR_SCENE_MAPPING_INIT_MODE_VFG、SAR_CENE_MAPPING_INIT_MODE_SFM で対応してください。

Note
このモードでは実際の初期化は行われません。
SmartAR™ SDK は SAR_MAX_NUM_INITIALIZATION_POINTS 以上の初期化点を生成しません。

認識対象画像用辞書作成ツールの使い方

認識対象画像用辞書作成ツール dictool は、認識対象の画像から SmartAR™SDK で利用するための辞書データ(xxxx.dic) を生成するためのツールです。生成された辞書データを SmartAR™ SDK に読み込ませることで、対象物体の検出および追跡（トラッキング）が可能になります。ひとつの認識対象につきひとつの辞書データが必要です。

現在 SmartAR™SDK で利用可能な認識対象物は、平面形状の画像（写真、雑誌、ポスターなど）です。認識対象の画像の模様（テクスチャ）によって、認識しやすいものと認識しにくいものがあり、SmartAR™ SDK の性能や利用時の使用感が大きく変化します。テクスチャの加工や認識対象の選別が可能な場合には 「良い認識対象を選ぶためのポイント」を参考にしてください。

動作環境

- 32-bit/64-bit Windows
- SSE2 対応プロセッサ

利用可能な画像形式

- PNG 形式　　：24bit カラー/ 8bit グレー
- BMP 形式　　：24bit カラー/ 8bit グレー
- PGM(P5)形式：8bit グレー
- PPM(P6)形式：24bit カラー

※短片が 200pixel 以上の画像を利用してください。サイズが小さすぎる場合には警告が表示されます。

辞書の作成方法

(1) 認識対象の画像ファイル（「利用可能な画像形式」を参照）を用意してください。

(2) 画像をプリンターで紙に印刷してください。この際、画像と印刷物のアスペクト比が変わらないように注意してください。

(3) 印刷した画像の横のサイズ(physicalWidth)を測ってください(単位はメートル）。

(4) dictool を以下のようなコマンドで実行してください。ビルドオプションについては後述の「ビルドオプション」を参照してください。１分ほどで辞書データ(xxxx.v9.dic)が作成されます。

```
> dictool.exe build -image smartar01.pgm -physicalWidth 0.19
```

(5) 辞書データが作成されると、辞書の評価が行われ認識対象の認識性能の評価値が表示されます。評価に関しては「辞書の評価方法」も参照してください。

dictool 実行後の表示例

```
> dictool.exe build -image smartar01.pgm -physicalWidth 0.19
     Build dictionary:
     input        : smartar01.pgm
     physicalWidth : 0.190000 [m]
     serialID     : 0
     vendor       : 0-0
     dicfile      : smartar01.pgm.v9.dic
     autoresize (576, 768) -> (150, 200)
     points: 33 31
     store the dictionary file to smartar01.pgm.v9.dic'
```

```
        ..............................................................
..............................
        the score of this target = 82.90 [0.00:bad - 100.00:excellent]
```

″Recognition Score″が低い(70 未満)の場合には、SmartAR™SDK で正しく認識されないことがあります。認識対象の加工、選択が可能な場合には、「良い認識対象を選ぶためのポイント」を参考にしてください。より厳密な辞書の評価を行う場合は、「辞書の評価方法」を参照してください。
エラーが出た場合は「エラーメッセージ」を参照してください。

ビルドオプション

dictool は以下のビルドオプションが使用できます。

ビルドオプション	書式	説明
-image	-image <*image.pgm*>	認識対象となる入力画像を指定します。PGM, PPM, BMP, PNG 形式をサポートしています。このオプションは必須です。
-help	-help	ヘルプを表示します。
-physicalWidth	-physicalWidth <*float*>	認識対象物体の″物理的な″横の大きさをメートル単位で指定してください。ピクセルサイズではありません。この値は、SmartAR™ SDK において対象物が認識された際に出力される座標値のスケールリングに利用されます。デフォルト値は 1.0[m]です。
-outimage	-outimage <*string*>	入力画像から検出された辞書データに格納されている特徴点情報を画像として保存します。形式は PPM です。結果の見方については「特徴点情報の見方」を参照してください。
-mask	-mask <*maskfile.pgm*>	認識対象画像内で認識対象となる有効領域を指定するためのマスク画像を指定します。マスク画像は、認識対象画像と同じサイズでなくてはなりません。画素値 255 の領域が辞書作成に利用されます。PGM 形式をサポートしています。

マスク

マスク用画像（PGM 形式）を入力することで、入力画像の特定の領域のみを利用する辞書の作成が可能です。マスク画像のサイズは入力画像のサイズと同じでなくてはなりません。マスク画像内の画素値の 255 の領域が認識対象領域として利用され、それ以外の領域は認識対象として利用されません。
このマスク機能は、コースターなど矩形ではない認識対象物の形状を正確に登録したい場合に、入力画像中のごく一部を除外する目的での利用を想定しています。登録画像領域のうち除外される領域が大きくなると、認識性能が低下することがあります。認識スコアを確認してください。

マスク機能を利用する場合の例

```
> dictool.exe build -image smartar01.pgm -physicalWidth 0.19 -mask
smartar01.mask.centeronly.pgm
        Build dictionary:
               input       : smartar01.pgm
              physicalWidth : 0.190000 [m]
               mask       :  smartar01.mask.centeronly.pgm
               serialID    : 0
               vendor      : 0-0
               dicfile     : smartar01.pgm.v9.dic
        autoresize (576, 768) -> (150, 200)
        points: 33 31
        store the dictionary file to 'smartar01.pgm.v9.dic'
.....................................................................
.....................
        the score of this target = 83.40 [0.00:bad - 100.00:excellent]
```

良い認識対象を選ぶためのポイント

SmartAR™SDK では、辞書作成時に抽出された特徴点を利用して認識対象の特定・追跡を行っています。特徴点としては、輝度の濃淡画像（テクスチャ）の中でコントラストが高く角を成している部分（コーナー点）が選択されます。テクスチャによって認識しやすい対象があり、SmartAR™ SDK の性能および利用時の使用感が大きく変化します。後述の「辞書の評価方法」や「特徴点情報の見方」を参考にしてください。

認識に適している対象物

- 凹凸のない、平面形状の印刷物（写真、雑誌、ポスターなど）
- 複雑なテクスチャが多く含まれているもの
- 角（コーナー点）が多く含まれるもの
- グレー画像にした時のコントラストが強いもの

認識に適していない対象物

- テクスチャが少ない、もしくは単純なもの（無地の壁、企業ロゴや文字など）
- テクスチャが局在しているもの
- テクスチャのコントラストが低いもの
- 同じテクスチャが繰り替えされるもの
- 縦横比が著しく偏っているもの
- 反射するもの（鏡、ラミネート加工されたもの）

辞書の評価方法

dictool で作成した辞書は、dictool を用いて評価することができます。ここで得られる辞書の評価は、様々な姿勢の認識対象が様々な環境内に置かれた場合、どの程度認できるかの目安になります。概ね70%以上あれば良い体感性能が期待できます。評価を行うには、以下のように-i オプションに続けて辞書ファイルのファイル名を入力して dictool を実行してください。

```
>dictool.exe evaluate -i smartar01.png.v9.dic
       input : smartar01.pgm.v9.dic
   ......................
```

```
the score of this target = 81.37 [0.00:bad - 100.00:excellent]
```

評価が終わると ″the score of this target =″ に辞書の性能が表示されます。

特徴点情報の見方

辞書作成の際に-outimage オプションを付けることで、認識に利用される特徴点が描画された特徴点画像が保存されます。この特徴点画像から検出・追跡の性能を一概に特定することは困難ですが、認識対象の検出・追跡の性能が著しく悪い場合の改善や、他の認識対象との比較の参考になることがあります。

特徴点

- 赤○：認識対象物の検出に利用される特徴点
- 緑□：認識対象物の追跡に利用される特徴点

特徴点が一様に分布しているほど、検出および追跡が安定します。
これら特徴点周辺のコントラストが低いと、検出率および追跡性能が低くなります。
検出用の特徴点（赤○）が偏在している場合、その場所が隠蔽されたり鮮明に写らなくなったりした際に検出性能が著しく低下します。

エラーメッセージ

- ″The size of the input image, (*<width>*, *<height>*), is too small as a target image″

入力画像のサイズが小さすぎます。「利用可能な画像形式」を参考に、適切なサイズの画像を利用してください。

- ″The aspect of the input image, (*<width>*, *<height>*), is out of range [0.25 - 4.0]″

入力画像サイズのアスペクト比が偏っています。長辺と短辺の比が 4:1 以下になるようにしてください。

認識対象用辞書のバージョン互換について

SDK のバージョンが変わると辞書の互換性は保証されません。

3 参考情報

用語の説明

用語	説明
コーナー点	登録された認識対象を認識する際に使用する画像内の角らしい点。
自然画像	マーカー等の特殊なパターンを使用していない一般的な画像。
認識対象	認識およびトラッキングの対象となる平面物体。
ポーズ	位置と姿勢を合わせたもの。3D 空間でのカメラの場所と向きを 6 自由度で表現する。
シーンマップ	推定された環境の 3D 構造。複数のランドマークから構成される。
ランドマーク	環境の 3D 構造の表面にある点で、3D 位置が推定されている点。 画像中のコーナー点を選んでトラッキングし、視差から 3D 位置が推定されている。

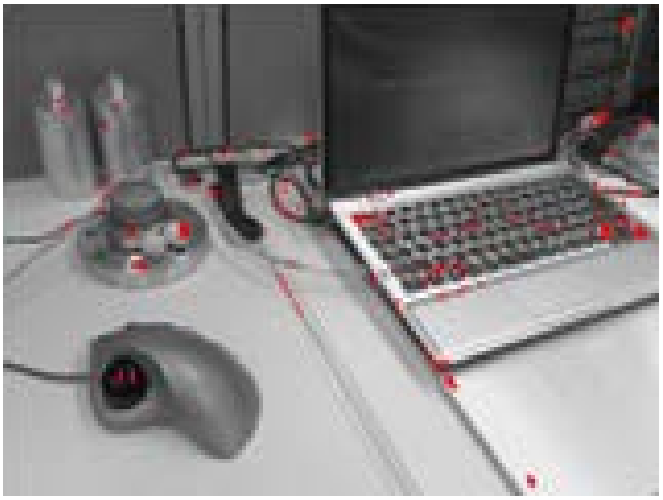

図 3　画像内のコーナー点

TargetTracking モードの処理の仕組み

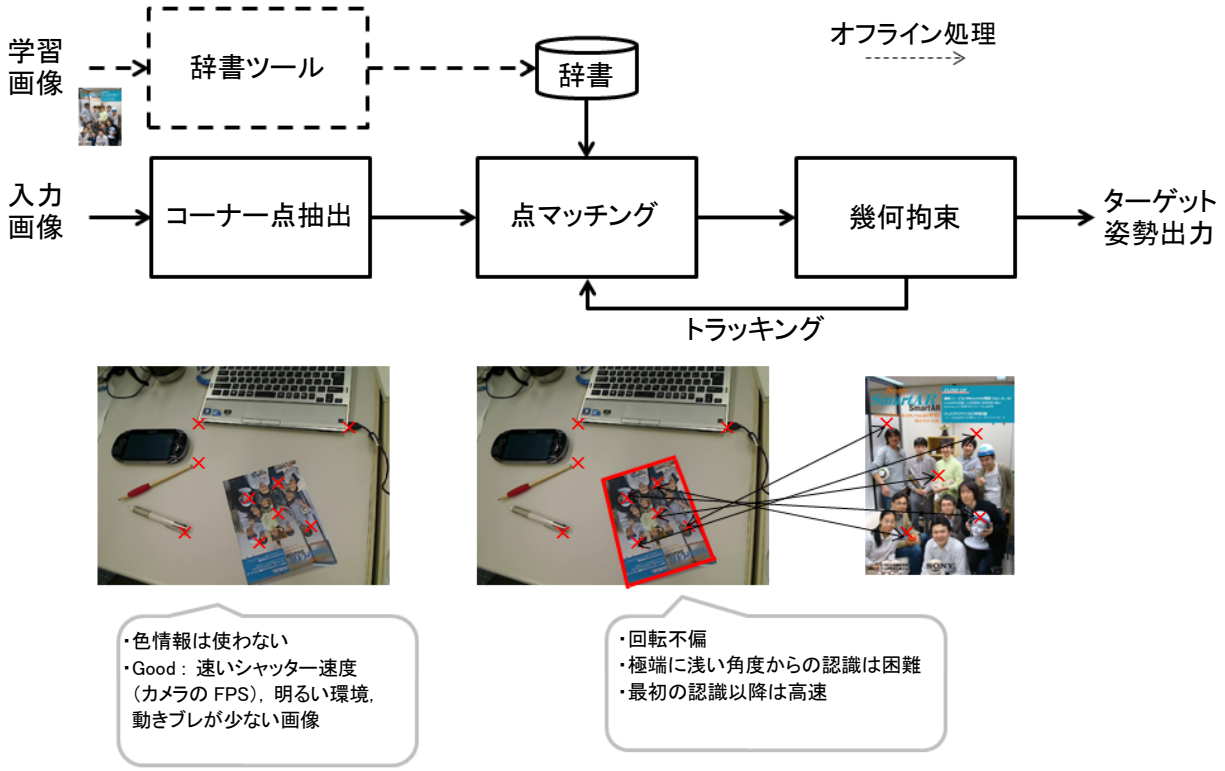

図 4　TargetTracking モードの処理図

(1)　入力画像

グレースケール画像を入力します。

(2)　コーナー点検出

入力された画像から、認識に用いるためのコーナー点を検出します。

(3)　点マッチング

SarRecognizer に登録した認識対象用の辞書ファイルと、検出したコーナー点とのマッチングを行い、対応関係を求めます。

(4)　幾何拘束

点マッチングで求めた対応関係に幾何的な整合性（上下左右などの位置関係）が成り立つかどうかを調べます。

(5)　出力

認識対象の認識結果が出力されます。認識が成功した場合は、認識対象中心を原点とする座標系でのカメラのポーズが取得できます。一度認識が成功するとトラッキング状態に入り、次フレーム以降の処理が高速になります。

TargetTracking モードの性能を引き出すために

(1) 良い認識結果を得るためのポイント

SmartAR™ SDK では、辞書作成時に抽出された特徴点を利用して認識対象の特定・追跡を行っています。特徴点としては、輝度の濃淡画像（テクスチャ）の中でコントラストが高く角を成している部分（コーナー点）が選択されます。テクスチャによって認識しやすい対象があり、SmartAR™ SDK の性能および利用時の使用感が大きく変化します。事前に辞書ファイルを作る場合、dictool の機能を利用して、認識対象についての情報を得ることができます。「認識対象画像用辞書作成ツールの使い方」も参考にしてください。

認識に適している対象物

- 凹凸のない、平面形状の印刷物（写真、雑誌、ポスターなど）
- 複雑なテクスチャが多く含まれているもの
- 角（コーナー点）が多く含まれるもの
- グレー画像にした時のコントラストが強いもの

認識に適していない対象物

- テクスチャが少ない、もしくは単純なもの（無地の壁、企業ロゴや文字など）
- テクスチャが局在しているもの
- テクスチャのコントラストが低いもの
- 同じテクスチャが繰り替えされるもの
- 縦横比が著しく偏っているもの
- 反射するもの（鏡、ラミネート加工されたもの）

(2) SmartAR™ SDK のサンプルプログラムとの比較

SmartAR™ SDK と共に配布されているサンプルプログラムは SmartAR™ SDK の本来の性能が発揮されるように作られています。サンプルプログラムと動作を比較してみてください。自然画像の認識を行う場合、サンプルプログラムの辞書を作成した辞書に差し替えて動作させると、より正確な比較ができ、問題がプログラムにあるのか辞書にあるのかの切り分けが容易になります。

Note
辞書の差し替えによってサンプルの性能が大きく低下する場合、辞書と入力画像に映っている印刷物のアスペクト比が同じであることを確認してください。アスペクト比が異なる場合、SarTargetTracking ライブラリは十分な機能を発揮できません。

SceneMapping モードの処理の仕組み

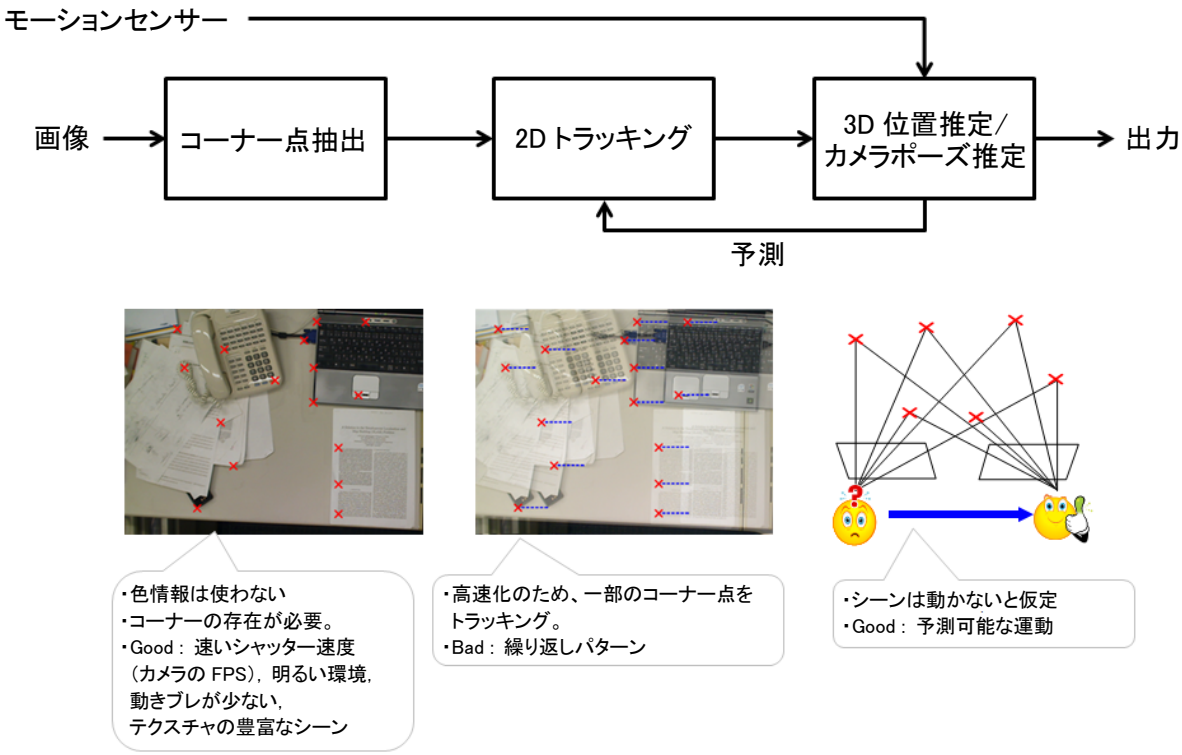

図 5　SceneMapping モードの処理図

(1)　画像とセンサー情報

グレースケール画像と、SarSensorDevice クラスから得られるセンサー情報を入力します。

(2)　コーナー点抽出

新しいランドマークをシーンマップに追加するために、カメラ画像の中からコーナー点を抽出します。

(3)　2D トラッキング

Scene Mapping 機能の内のランドマークがカメラ画像中のどこに見えているか、画像の中から探します。

(4)　3D 位置推定/カメラポーズ推定

カメラ画像中の 2D トラッキングされた位置にシーンマップ内のランドマークが見えるようなカメラのポーズを推定します。同時に、シーンマップ内のランドマークの 3D 位置の推定も行います。

(5)　出力

シーンマップを基準にしたカメラのポーズ（位置と向き）を出力します。

SceneMapping モードの性能を引き出すために

良い認識結果を得るための条件

SceneMapping モードは、以下のような条件でより良く動作します。画像認識をできるだけ正常に動作させるために留意してください。

- 変化しない環境
- カメラで撮影したときに、コーナー点が豊富に存在するような環境
- 速いシャッタースピード
- 明るい環境
- 動きブレの少ない画像
- 繰り返しパターンの少ない環境
- スムーズで予測可能な SmartAR™動作端末の運動

処理のポイント

SceneMapping モードが行っている連続画像の解析では、画像と画像の時間間隔が開けば開くほど解析が難しくなるため、解析にかかる計算量が増加したり、計算結果の精度が悪化したりします。
できるだけ高速に画像を処理し続けることが SceneMapping モードの本来の性能を引き出す上で重要です。

SmartAR™ SDK のサンプルプログラムとの比較

SmartAR™SDK と共に配布されているサンプルプログラムは SmartAR™SDK の本来の性能が発揮されるように作られています。サンプルプログラムと動作を比較してみてください。自然画像で初期化している場合は、サンプルプログラムの辞書を作成した辞書に差し替えて動作させるとより正確な比較ができ、問題がプログラムにあるのか辞書にあるのかの切り分けが容易になります。サンプルとの動作の差異が大きい場合、以降を参考に原因を確認し改善を行ってください。

カメラのフレームレートの確認

カメラが十分なフレームレートで動作していることを確認してください。
キャプチャされた画像が SarRecognizer::dispatch()に入力されるまでにかかる時間が可能な限り短いことを確認してください。
同様に、センサーを使っている場合は、センサーから読み込んだ値が SarRecognizer::dispatch()に入力されるまでにかかる時間が可能な限り短いことを確認してください。

カメラのフレームレートを上げる

キャプチャした画像ができるだけ早く SarRecognizer::dispatch()に入力されるようにバッファやスレッド設計を行ってください。

解析結果と重畳表示の時間ずれの確認

出力される SarRecognitionResult 構造体は、入力した画像の時点を元に計算したものなので、入力画像のタイムスタンプが付けられています。画面出力は、カメラ画像を背景として CG を重畳して作られます。背景カメラ画像のタイムスタンプと、CG 描画に使う SarRecognitionResult 構造体のタイムスタンプの差を確認してください。このタイムスタンプを一致させることが理想的ですが、必須ではありません。

解析結果と重畳表示の時間ずれの改善

カメラ画像と CG を画面上で正確に合わせるためには、以下の 2 つの方法どちらか、または両方を行う必要があります。ゲームの特性に合わせて検討してください。

(a) SarRecognitionResult を最新のカメラ画像の時間まで時間伝播し、時間伝播した result で CG を重畳して画面出力する。

(b) SarRecognitionResult に対応するカメラ画像をバッファしておき、result が出力されたら、対応するカメラ画像に CG を重畳して画面出力する。

サンプルプログラムでは(b)の方法を採用しています。
この方法では、原理的にはカメラ画像と CG のずれは発生しません。
(a)の場合、実際の SmartAR™動作端末の動きに対する画面の時間遅れは小さくなりますが、時間伝播を行ったことで誤差が増幅され、CG が少しずれたり振動したりしているように見えることがあります。

SarRecognizer::dispatch()と SarRecognizer::runWorker()の動作状況の確認

プロファイラを使ってプログラムをプロファイルしてください。SarRecognizer::dispatch()と SarRecognizer::runWorker()が連続して呼び出されていることを確認してください

SarRecognizer::dispatch()と SarRecognizer::runWorker()の動作状況の改善

もし必要であればスレッドの CPU マスクやプライオリティを適切に設定してください。

座標系

座標系はすべて右手系です。位置および向きは位置ベクトルと姿勢クォータニオンで表現されます。位置ベクトルはシーン固定座標系でのカメラ中心の位置ベクトルです。姿勢クォータニオンは、デバイス固定座標系のシーン固定座標系からの回転を表します。
以下の SceneMapping モードの初期化方法によって、それぞれ異なる世界座標系が定義されます。

- 自然画像による初期化
- HFG で初期化
- VFG で初期化
- SFM で初期化

(1) 自然画像初期化時の世界座標系の定義

自然画像（以下、認識対象）による初期化時の世界座標系については、認識対象の中心を原点、認識対象の裏面から表面への方向を Z 軸、認識対象を正面から見て左から右への方向を X 軸、と定義します。

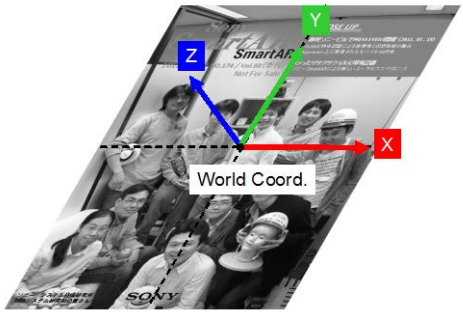

図 6 自然画像初期化時の座標軸の定義

(2) HFG 初期化時の世界座標系の定義

HFG 初期化時の世界座標系については、カメラ画像の中心に写っている水平面箇所を原点、重力方向の逆向きを Z 軸、カメラから原点へのベクトルを水平面上に射影したベクトルの方向を Y 軸、と定義します。

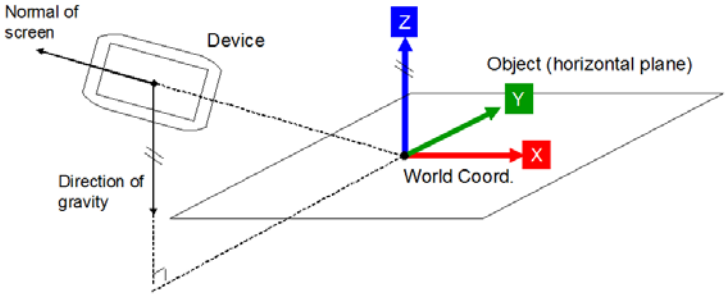

図 7 HFG 初期化時の座標軸の定義

(3) VFG 初期化時の世界座標系の定義

VFG 初期化時の世界座標系については、カメラ画像の中心に写っている鉛直面箇所を原点、鉛直面の法線を Z 軸、重力方向の逆向きを Y 軸、と定義します。

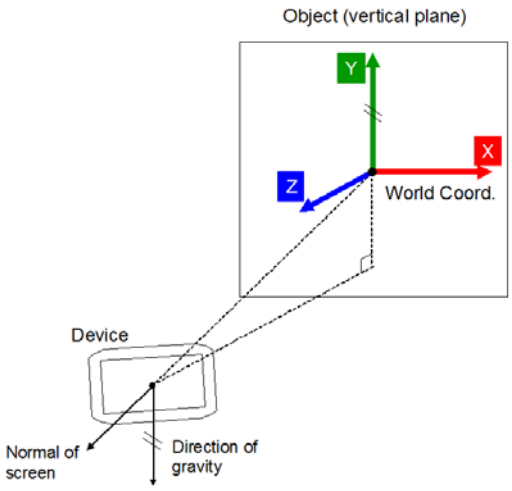

図 8　VFG 初期化時の座標軸の定義

(4)　SFM 初期化時の世界座標系の定義

SFM 初期化時の世界座標系については、3D 構造からカメラ画像中心領域の主平面（Dominant Plane）を計算し、世界座標系が、カメラ画像の中心に写っている主平面箇所が原点、主平面の法線が Z 軸、カメラから原点へのベクトルを主平面上に射影したベクトルの方向を Y 軸、と定義します。

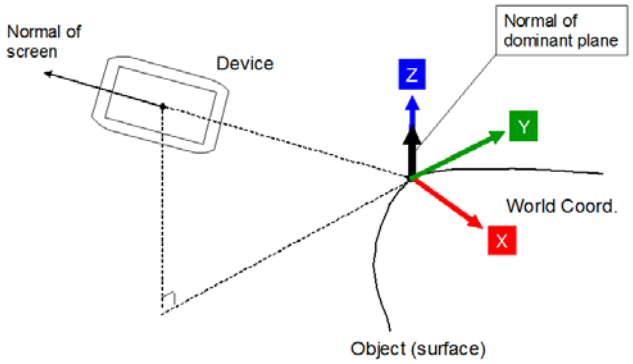

図 9　SFM 初期化時の座標軸の定義