libsmart Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

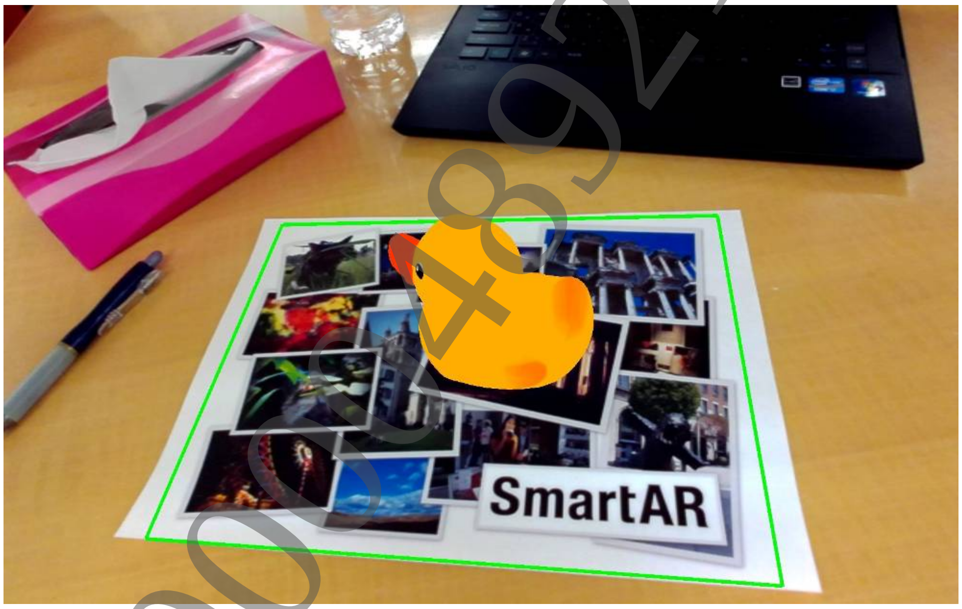
Purpose and Characteristics

libsmart is a library for easily implementing augmented reality (AR), which superimposes computer graphics (CG) of a virtual character onto a camera image of a real-life environment to make it appear as though the character really exists in the scene. libsmart uses image recognition technologies whose core is SmartAR™, a technology capable of achieving AR.

libsmart is composed of the TargetTracking library and the SceneMapping library. The TargetTracking library uses an arbitrary natural image (planar object image, such as, a book cover or tarot card) and AR Play Card as recognition targets and searches for them in a camera image. Upon succeeding with the search, the library estimates the camera pose (position and orientation) for each recognized target object. Its main characteristic is that because corner points within a natural image are detected to recognize/track targets, there is no need to print a special "marker" on the recognition target planar objects.

With this technology, it is possible, for example, to easily realize an AR with a CG character superimposed on top of a planar object captured in the camera image (Figure 1).

Figure 1 Recognition of a Planar Object and CG Superimposition by the TargetTracking Library

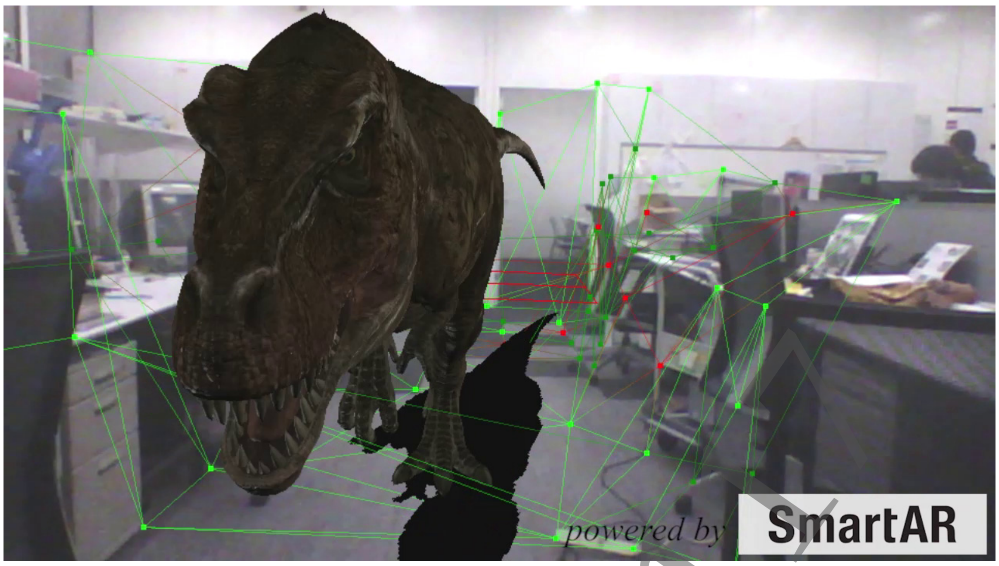


The SceneMapping library estimates the 3D structure of an unknown environment (scene map) and the camera pose within that 3D structure based on the changes in camera images and measurements taken by the motion sensors as the camera moves across space. It realizes a technology called Simultaneous Localization and Mapping (SLAM).

With this technology, it is possible to recognize the 3D structure of the surrounding environment in which the camera exists, without having to recognize/track a specific recognition target, and to render a virtual character as if it really exists in the environment (Figure 2).

Note that it is possible to use the TargetTracking library and SceneMapping library together.

Figure 2 Recognition of a 3D Structure in the Surrounding Environment and CG Superimposition by the SceneMapping Library



Used Resources

libsmart uses the following system resources.

Resource	Description
Work memory	When loading the dictionary of one object, the application explicitly allocates 650 KiB When creating one recognition target with `sceSmartCreateInstantImageTarget()`, the application explicitly allocates 400 KiB When using a natural image as a recognition target with TargetTracking library, the application explicitly allocates 16 MiB When using the AR Play Card as a recognition target with the TargetTracking library, the application explicitly allocates 500 KiB When using Scene Mapping library, the application explicitly allocates 18 MiB

Embedding into a Program

The following files are required in order to use libsmart.

File	Description
libsmart.h	Header file
libSceSmart_stub.a	Stub library file
libSceSmart_stub_weak.a	Weak import stub library file

Include libsmart.h in the source program (various header files will be automatically included as well).

Load the PRX module in the program, as follows.

```
if ( sceSysmoduleLoadModule(SCE_SYSMODULE_SMART) != SCE_OK ) {
    // Error handling
}
```

Upon building the program, link libSceSmart_stub.a.

Copyright Notice

"SmartAR" is the augmented reality technology developed by Sony Corporation. This is a trademark or registered trademark of Sony Corporation in Japan and/or other countries. When displaying the logo and/or the trademark on a game screen, package, distributed item, etc., follow the "SmartAR Name & Logo Guideline" included in the Brand Guidelines And Logos package provided on the PlayStation®Vita Developer Network website (https://psvita.scedev.net/).

Sample Programs

The following programs are provided as libsmart sample programs for reference purposes.

sample_code/engines/api_libsmart/target_tracking/

This sample shows the basic usage of the libsmart TargetTracking library.

sample_code/engines/api_libsmart/instant_tracking/

This sample exemplifies use of the feature of the libsmart TargetTracking library to create a recognition target on the spot.

sample_code/engines/api_libsmart/scene_mapping/

This sample shows the basic usage of the libsmart SceneMapping library.

sample_code/engines/api_libsmart/scene_mapping_custom/

This sample exemplifies use of the custom listener feature of the libsmart SceneMapping library.

sample_code/engines/api_libsmart/smart_tracking/

This sample exemplifies a case where libsmart's SceneMapping library and TargetTracking library are used together.

Host Tool

dictool

dictool is a tool to create a dictionary file from a recognition target image. dictool is installed in the following directory:

- %SCE_PSP2_SDK_DIR%/host_tools/smart/

For usage of dictool, refer to the "5: How to Use dictool" chapter.

2 Recognition/Estimate Processing

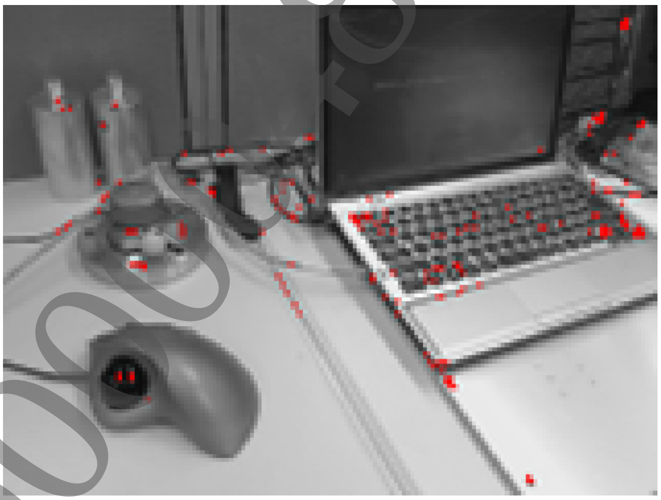
This chapter indicates the processing flow by which libsmart (TargetTracking library and SceneMapping library) recognizes recognition target objects and the 3D structure of the surrounding environment, and estimates the camera pose.

Terminology

The terms used in explaining recognition and estimate processing are summarized below.

Term	Description
Corner point (feature point)	Corner points in an image. In libsmart, these are used as "feature points" to recognize/track registered recognition targets.
Natural image	A general image where no special patterns, such as markers, are used
Recognition target	The recognition/tracking target planar object
Pose	A combination of position and orientation. In libsmart, the camera's location and angle are expressed in six degrees of freedom.
Scene map	The 3D structure of the surrounding environment that is estimated by the library. It is composed of multiple landmarks.
Landmarks	Points on the surface of an environment's 3D structure whose 3D position is estimated. In libsmart, the results of tracking selected corner points in an image and estimating the 3D position from the parallax of multiple camera images that have been taken from differing viewpoints are recognized as landmarks.

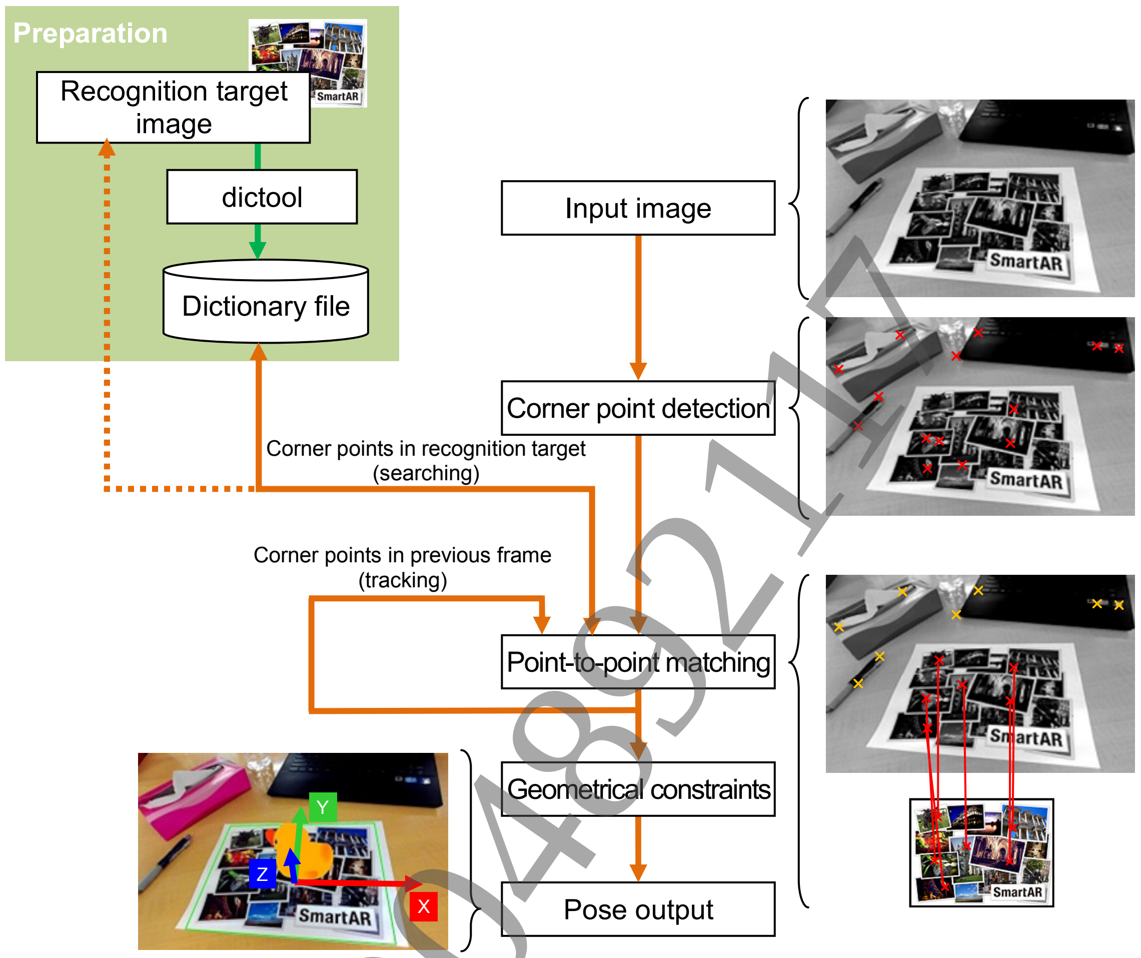
Figure 3 Corner Points Within an Image



Structure of TargetTracking Library Processing

The recognition processing flow of the TargetTracking library is indicated below.

Figure 4 TargetTracking Library Recognition Process Flow



Recognition Target Image

This is the planar object image serving as the recognition target. It must be converted to a dictionary file in advance using dictool or loaded onto memory upon application execution. There are objects that are easy to recognize and objects that are difficult to recognize (refer to the "Natural Images for Obtaining Good Recognition Results" item)

The above preparation is not required when using an AR Play Card as a recognition target.

Input Image

Input a 640 x 480 pixel grayscale image such as a camera image. A photo taken in a bright environment with a high shutter speed and little blur is suitable.

Corner Point Detection

Points with high contrast that form corners are detected from the input image.

Point-to-Point Matching

During a search, the correspondence relationships between the corner points of the registered recognition target and the corner points detected in the input image are determined.

During tracking, the correspondence relationships between the corner points whose correspondence relationships were found in the previous frame and the corner points detected in the latest input image are determined.

Geometrical Constraints

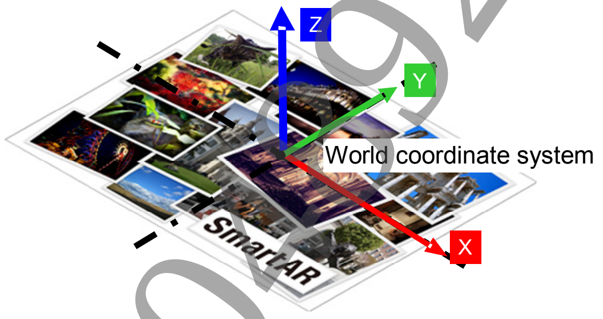
The geometrical consistency (positional relationships such as up-down, left-right, etc.) of the correspondence relationships found through point-to-point matching is verified.

Pose Output

The camera pose for the recognized target is output. Once recognition is successful, the library will transition from the search state to the tracking state, and processing from the next frame onwards will accelerate.

The camera "pose" is expressed with the camera position and orientation in a coordinate system fixed to an individual recognition target. This coordinate system is defined with the recognition target's center as the origin, the direction from the recognition target's back to its front as the Z-axis, and the direction from the recognition target's left to right when seen from the front as the X-axis.

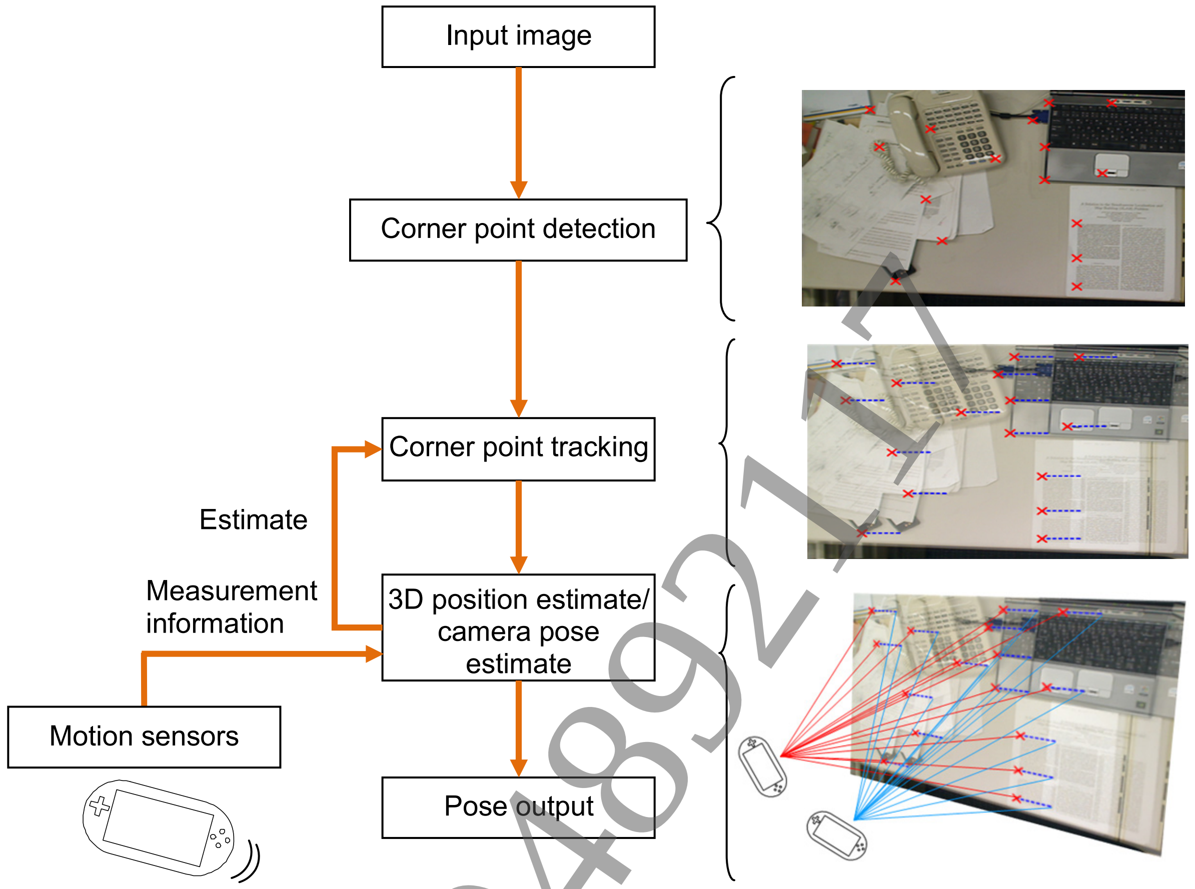
Figure 5 Coordinate Axes That Indicate the Pose



SceneMapping Library Processing

The recognition processing flow of the SceneMapping library is indicated below.

Figure 6 Recognition/Estimate Processing Flow of the SceneMapping Library



Input Image

Input a 640 x 480 pixel grayscale image such as a camera image. A photo taken in a bright environment with a high shutter speed and little blur is suitable.

Motion Sensor Measurement Information

Enter measurement results of motion sensors obtained from libmotion

Corner Point Detection

Corner points are detected in the camera image to add a new landmark to the scene map.

Corner Point Tracking

Corner points are tracked to search for landmarks in the camera image.

3D Position Estimate/Camera Pose Estimate

3D positions of landmarks in the scene map are estimated from the parallax of camera images taken from different viewpoints to construct the scene map. At the same time, the camera pose, by which landmarks in the scene map are visible at the positions of the tracked corners, is estimated.

Pose Output

The camera pose (position and orientation) is output based on the scene map.

3 Using the TargetTracking Library

The TargetTracking library is a library that searches to see if planar objects registered as recognition targets appear in a camera image or not, then outputs the camera pose (position and orientation) for the objects that appear. By performing tracking according to the search results, objects that appear in a video can be efficiently recognized.

Recognition Targets of the TargetTracking Library

A natural image registered in advance as a dictionary file, a natural image that has not been registered in advance as a dictionary file, and an AR Play Card can be recognition targets of the TargetTracking library. Usage of the TargetTracking library differs according to these recognition targets as follows.

When the Recognition Target is a Natural Image Prepared in Advance

A recognition target natural image can be converted into a dictionary file with dictool so that it can be registered to and used by libsmart. In such a case, you will need to create the dictionary file for the recognition target in advance with dictool. The TargetTracking library can recognize multiple recognition targets; however, each recognition target must be registered in separate dictionaries.

The TargetTracking library's performance will vary depending on input images and recognition target states. The size of a dictionary file is about 500 KB.

For more information on dictool, refer to the "5: How to Use dictool" chapter.

When the Recognition Target is a Natural Image without Advance Preparation

Even if a dictionary file is not prepared in advance, it is possible to register a downloaded image or camera image upon application execution as a recognition target to recognize it using the TargetTracking library. However, there are demerits to this, including the deceleration of the search speed compared to when a dictionary file is used. Thus, the creation of a dictionary file is recommended when using an image that can be obtained in advance as a recognition target. For details, refer to the "For Better TargetTracking Library Performance" section of the "6: Reference Information" chapter.

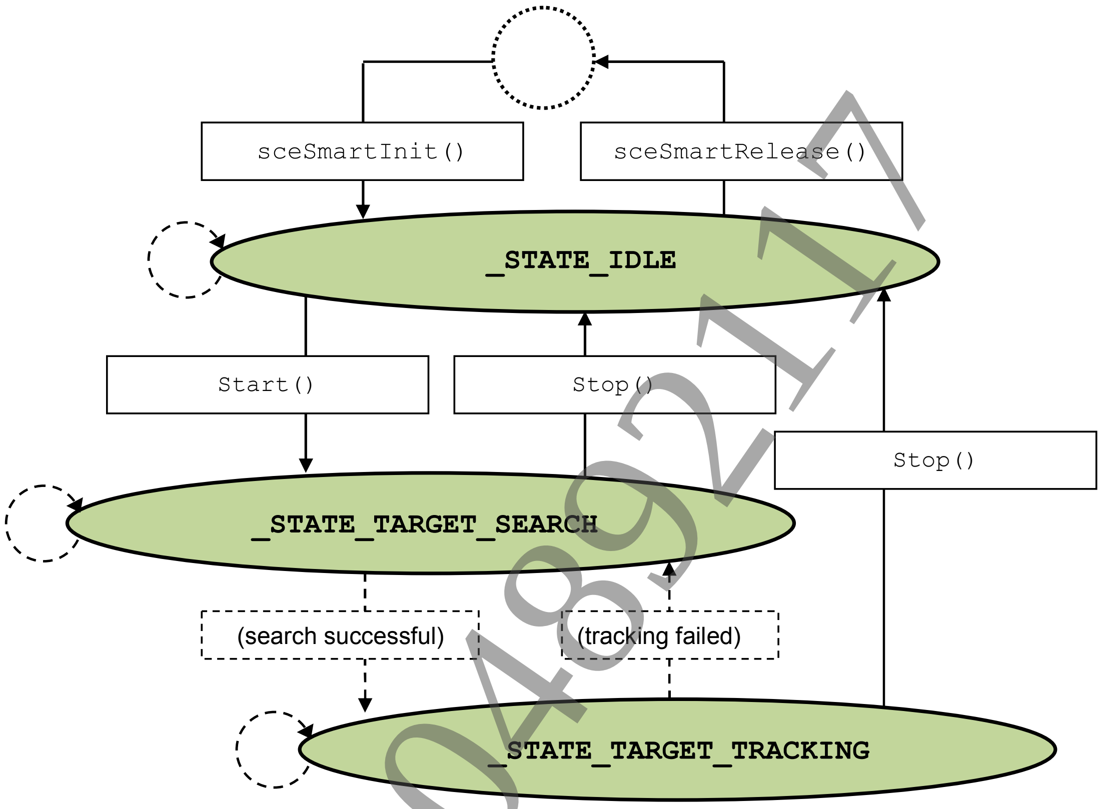
When the Recognition Target is an AR Play Card

When using an AR Play Card as a recognition target, recognition can be executed by registering the ID allocated per AR Play Card to libsmart. Up to 6 AR Play Cards can be recognition targets simultaneously.

Recognition Target State Transitions

The following diagram shows each of the state transitions for recognition targets when performing recognition with the TargetTracking library. The state transitions that occur with function calls are shown with a solid line and arrow, and the transitions that automatically occur depending on the recognition state are shown with a dotted line and arrow.

Figure 7 State Transitions of the TargetTracking Library



*The function prefix `sceSmartTargetTracking` and the macro name prefix `SCE_SMART_TARGET_TRACKING` are omitted.

SCE_SMART_TARGET_TRACKING_STATE_IDLE (idle state)

In this state, the TargetTracking library has not been started yet.

When the TargetTracking library is started up, the state will transition between the search state and the tracking state depending on whether a recognition target is found and tracked successfully.

SCE_SMART_TARGET_TRACKING_STATE_TARGET_SEARCH (search state)

In this state, no registered recognition targets have been found in the image and the library continues the search.

SCE_SMART_TARGET_TRACKING_STATE_TARGET_TRACKING (tracking state)

In this state, a registered recognition target has been recognized in the image and the library is successfully tracking the recognition target.

Basic Procedure for Recognition

The basic procedure for carrying out recognition using the TargetTracking library is explained below.

(1) Initialize libsmart

Call `sceSmartInit()` to initialize libsmart. The `SceSmartMemoryAllocator` structure storing a function pointer to a memory allocator can be passed to `sceSmartInit()`. When the argument of `sceSmartInit()` is NULL, libsmart will be initialized so that `malloc()` and `free()` will be used for memory allocation.

(2) Create a recognition target

When using an image prepared in advance as the recognition target, load the dictionary file with `sceSmartCreateLearnedImageTarget()`. The recognition target will be created and a recognition target ID will be allocated.

When using an image that has not been prepared in advance as the recognition target, use `sceSmartCreateInstantImageTarget()` to create a recognition target from an image placed on memory.

This procedure will not be necessary if the recognition target is an AR Play Card. Because a unique ID is assigned to each card, use this ID as the recognition target ID.

(3) Register the recognition target

Call `sceSmartTargetTrackingRegisterTarget()` with the assigned recognition target ID specified as an argument. When handling multiple recognition targets, repeat steps (2) and (3) as many times as required.

(4) Set a search policy

Call `sceSmartTargetTrackingSetSearchPolicy()` to set a search policy that either prioritizes search speed or prioritizes search sensitivity. Note that the specification of a search policy is only valid when a recognition target is created from a dictionary file.

Prioritize search speed:

This search policy focuses on executing a search rapidly. Although it is easy to obtain high recognition performance because the time required for one search is short, the recognition target within the input image needs to appear at a certain size.

Prioritize search sensitivity:

This search policy focuses on search sensitivity. It shows high recognition performance even in cases where the recognition target's dimensions within the input image are small, and it can recognize even with an area approximately 1/4 the size in comparison with when search speed is prioritized, but the processing time increases approximately 4 times.

(5) Start recognition

Call `sceSmartTargetTrackingStart()` to start the TargetTracking library. For the argument, specify the maximum number of recognition targets for which tracking processing will be performed with a single call of `sceSmartTargetTrackingRun()` or `sceSmartTargetTrackingRun2()`. For details, refer to step (7).

Note
The maximum number of recognition targets when using AR Play Cards cannot be limited by `sceSmartTargetTrackingStart()`.

(6) Obtain the input image

Use libcamera, etc. to obtain the input image. It must be a grayscale image with 8 bits per pixel, the width must be `SCE_SMART_IMAGE_WIDTH` pixels, and the height must be `SCE_SMART_IMAGE_HEIGHT` pixels. Therefore, processing where an `SCE_CAMERA_FORMAT_YUV422_PACKED` image is unpacked with just the Y component extracted, or just the Y component calculated from an `SCE_CAMERA_FORMAT_RAW8` image will also be required.

(7) Execute recognition

Call `sceSmartTargetTrackingRun()` or `sceSmartTargetTrackingRun2()` to execute recognition.

The input data for `sceSmartTargetTrackingRun()` is the input image only, and the input data for `sceSmartTargetTrackingRun2()` is the input image, the timestamp, and information on motion sensor measurements. Information on motion sensor measurements must consist of libmotion's measurement values, listed without intervals or duplications and in order from the newest measured time to older measurements.

Note
If the recognition target is a natural image, you can use either without any change in performance.
If the recognition target is an AR Play Card, we recommend using `sceSmartTargetTrackingRun2()`. Moreover, use `sceSmartTargetTrackingRun2()` with the timestamp when there is a need to clarify for which frame the recognition result is for; for example, when rendering CG to superimpose over the camera image on a separate thread.

Even when multiple recognition targets are registered, the search processing will be performed for one recognition target only for each call. Therefore, there will be a delay of at least the time required for the registered targets until the search processing is complete for all registered recognition targets. On the other hand, the tracking processing will be performed for multiple targets that have already been discovered at the same time up to the maximum number specified with `sceSmartTargetTrackingStart()` in a single call. Note that if the maximum number of tracking targets has been reached, search processing will not be performed.

Of the registered recognition targets, `sceSmartTargetTrackingRun()`/`sceSmartTargetTrackingRun2()` returns the number of recognition targets that have been successfully searched or tracked in the image. Recognition targets that have been successfully searched transition to the `SCE_SMART_TARGET_TRACKING_STATE_TARGET_TRACKING` state; those that didn't will remain in the `SCE_SMART_TARGET_TRACKING_STATE_TARGET_SEARCH` state. Recognition targets in the `SCE_SMART_TARGET_TRACKING_STATE_TARGET_TRACKING` state will continue to be tracked in the next frame and onwards.

(8) Obtain the results of recognition

With `sceSmartTargetTrackingGetResults()`, it is possible to obtain the camera pose (position and orientation) for each of the recognized recognition targets.

With `sceSmartTargetTrackingQuery()`, it is possible to obtain whether or not a specific recognition target was recognized or not and obtain the camera pose for the target that was recognized. With `sceSmartTargetTrackingQuery2()`, it is possible to obtain the timestamp for the input image in addition to the recognition results for a specific recognition target.

Perform appropriate processing using the obtained recognition results, and return to step (6) and repeat as required.

(9) Terminate recognition

Call `sceSmartTargetTrackingStop()` to terminate the TargetTracking library.

(10) Unregister the recognition target

Call `sceSmartTargetTrackingUnregisterTarget()` to unregister the recognition target registered to the TargetTracking library.

(11) Delete the recognition target

Call `sceSmartDestroyTarget()` to delete the recognition targets.

(12) Terminate libsmart

Call `sceSmartRelease()` to terminate libsmart.

API Summary

The APIs used when performing recognition using the TargetTracking library are as follows.

Main APIs Used in Basic Recognition Processing

API	Description
`SceSmartMemoryAllocator`	Memory allocator
`sceSmartInit()`	Initializes libsmart
`sceSmartCreateLearnedImageTarget()`	Creates a recognition target from the dictionary file that has been prepared in advance from an image
`sceSmartCreateInstantImageTarget()`	Creates a recognition target on the spot from an image
`sceSmartTargetTrackingRegisterTarget()`	Registers the recognition target
`sceSmartTargetTrackingSetSearchPolicy()`	Sets the specified search policy
`sceSmartTargetTrackingStart()`	Starts the TargetTracking library
`sceSmartTargetTrackingRun()`	Executes recognition processing
`sceSmartTargetTrackingRun2()`	Executes recognition processing
`sceSmartTargetTrackingGetResults()`	Obtains recognition results
`sceSmartTargetTrackingQuery()`	Obtains specific recognition results
`sceSmartTargetTrackingQuery2()`	Obtains specific recognition results with the timestamp
`sceSmartTargetTrackingStop()`	Stops the TargetTracking library
`sceSmartTargetTrackingUnregisterTarget()`	Unregisters the recognition target
`sceSmartDestroyTarget()`	Delete recognition targets
`sceSmartRelease()`	Stops the use of libsmart

4 Using the SceneMapping Library

The SceneMapping library creates a 3D structure of an unknown environment (scene map) and estimates the camera pose (position and orientation) within that 3D structure based on the changes in camera images and measurements taken by the motion sensors as the camera moves across space.

Characteristics of Recognition/Estimate Processing by the SceneMapping Library

World Coordinate System Definition and Initialization Mode

In the SceneMapping library, the world coordinate system (coordinate system fixed to a scene) must be defined to create a scene map before carrying out recognition/estimate processing. For this, the application must specify how to define the world coordinate system upon library initialization. The definition types are referred to as "SceneMapping library initialization modes". Regarding the world coordinate definition for each initialization mode, refer to the "World Coordinate System Definition According to the Initialization Mode" section of the "6: Reference Information" chapter.

There are initialization modes such as, the "feature point location mode" that only locates the positions of initialization points (corner points recognized upon library initialization) without carrying out recognition/estimate processing, and the "scene map loading mode" that performs initialization by reading a scene map that has been saved in advance. The world coordinate system will not be defined in these initialization modes.

Custom Listener

When the application is able to determine the camera pose and scene map from camera images and motion data without using features of the SceneMapping library, the application must prepare a unique callback function to pass this information to the SceneMapping library. This scheme is referred to as a "custom listener".

For example, by using the TargetTracking library as the customer listener callback, the recognition target created with `sceSmartCreateInstantImageTarget()` can be used to initialize the SceneMapping library. Moreover, because the scene map is already determined from the blueprint or prior measurements for a diorama or event space, this precise scene map can be provided to the SceneMapping library to obtain output that is more stable than the estimate by the library.

For details of the custom listener, refer to the "Use a custom listener" item and the "For Better SceneMapping Library Performance" section of the "6: Reference Information" chapter.

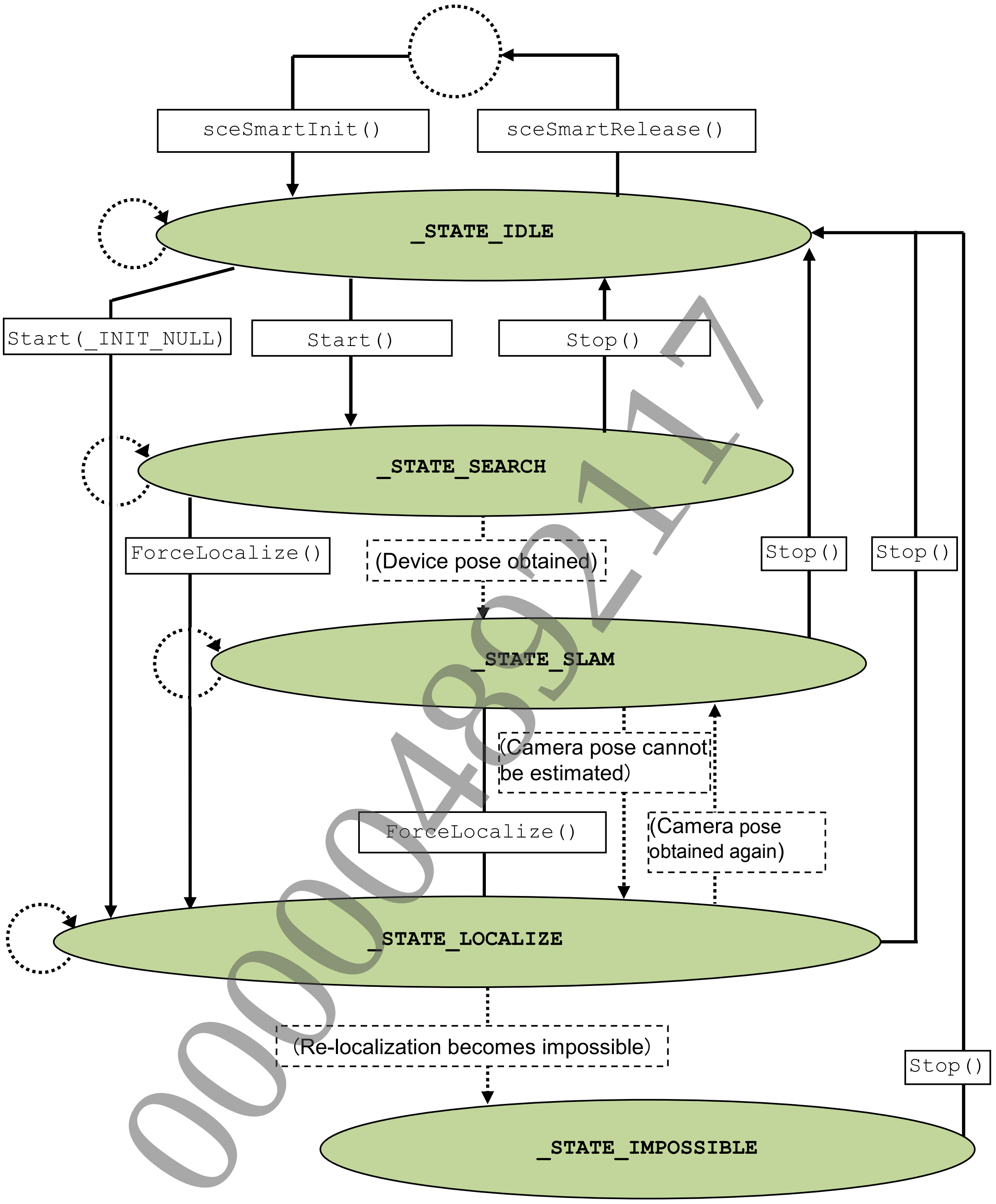
Combined Usage with the TargetTracking Library

The SceneMapping library can be used in combination with the TargetTracking library while excluding the recognition target area within the screen with mask processing. In this case, you will need to create mask information by using the results of recognition obtained from the TargetTracking library (information on camera position and orientation against the object to be recognized), and input it in the SceneMapping library. The mask area will be represented as a combination of multiple triangular patches, where triangular patch data is composed of an array of vertex position information within the image.

Library State Transitions

The library state transitions when carrying out recognition/estimate processing with the SceneMapping library are indicated below. The state transitions that occur with function calls are shown with a solid line and arrow, and arrows with dotted lines indicate automatic transitions occurring due to the state of recognition.

Figure 8 State Transitions of the SceneMapping Library



*The `sceSmartSceneMapping` function prefix and `SCE_SMART_SCENE_MAPPING` macro prefix are omitted here

SCE_SMART_SCENE_MAPPING_STATE_IDLE (idle state)

This state indicates that the SceneMapping library is ready for start-up.

SCE_SMART_SCENE_MAPPING_STATE_SEARCH (initialization state)

In this state, the SceneMapping library is initializing the camera's pose and the scene map in the specified initialization mode. The SceneMapping library is attempting to recognize the recognition targets and initialization points based on the input image and motion sensor measurements. When initialization succeeds, the library state will automatically transition to the next one, `SCE_SMART_SCENE_MAPPING_STATE_SLAM`.

SCE_SMART_SCENE_MAPPING_STATE_SLAM (SLAM state)

This state is the SceneMapping library's main state. While in this state, the SceneMapping library can satisfactorily estimate the camera's pose and the scene map in relation to the scene. The scene map will be created gradually as the camera moves through the scene (if the map has not been fixed with `sceSmartSceneMappingFixMap()`). When the library is in this state, the application will instruct the user to move the camera back and forth, up and down, left and right as opposed to simply rotating it in the same spot. If the camera's pose cannot be estimated due to tracking fails or the camera moving too rapidly, etc. the state will automatically transition to `SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE`.

SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE (localization state)

In this state, the SceneMapping library is attempting to re-localize (restore camera pose estimate) using the scene map created in the `SCE_SMART_SCENE_MAPPING_STATE_SLAM` state. When the library is in this state, the application will instruct the user to return the camera to the pose it was in during the `SCE_SMART_SCENE_MAPPING_STATE_SLAM` state.

SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE_IMPOSSIBLE (localization impossible state)

This state indicates that re-localization is not possible because sufficient scene map information required for the SceneMapping library to attempt restoring the camera pose in the `SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE` state could not be obtained in the `SCE_SMART_SCENE_MAPPING_STATE_SLAM` state. When the library is in this state, the application will instruct the user to initialize it again.

SceneMapping Library Initialization Modes

SLAM estimate must be initialized with `sceSmartSceneMappingStart()` before carrying out recognition/estimate processing with the SceneMapping library. The initialization modes that can be specified as an argument of `sceSmartSceneMappingStart()` are indicated below.

`SCE_SMART_SCENE_MAPPING_INIT_LEARNED_IMAGE`	Initialization with a learned natural image
`SCE_SMART_SCENE_MAPPING_INIT_WAAR`	Initialization with an AR Play Card
`SCE_SMART_SCENE_MAPPING_INIT_HFG`	HFG (horizontal from gravity) initialization
`SCE_SMART_SCENE_MAPPING_INIT_VFG`	VFG (vertical from gravity) initialization
`SCE_SMART_SCENE_MAPPING_INIT_SFM`	SFM (structure from motion) initialization
`SCE_SMART_SCENE_MAPPING_INIT_CUSTOM`	Initialization with a custom listener
`SCE_SMART_SCENE_MAPPING_INIT_DRYRUN`	Feature point location mode for HFG/VFG/SFM
`SCE_SMART_SCENE_MAPPING_INIT_NULL`	Scene map loading mode

In initialization modes other than `SCE_SMART_SCENE_MAPPING_INIT_DRYRUN/SCE_SMART_SCENE_MAPPING_INIT_NULL`, a world coordinate system will be defined upon library initialization. For the world coordinate system, refer to the "World Coordinate System Definition According to the Initialization Mode" section of the "6: Reference Information" chapter.

Details of each initialization mode are explained below.

SCE_SMART_SCENE_MAPPING_INIT_LEARNED_IMAGE

In this initialization mode, SceneMapping library initialization is carried out with the recognition target (natural image) learned in advance using dictool.

Before executing `sceSmartSceneMappingStart()`, you will need to load the dictionary file and register it to the SceneMapping library. First, load the dictionary file with `sceSmartCreateLearnedImageTarget()`. Then register the obtained recognition target's ID to the SceneMapping library with `sceSmartSceneMappingRegisterTarget()` and call `sceSmartSceneMappingStart()`. Initialization will complete when the recognition target natural image is correctly recognized.

Although there is no library feature to initialize the SceneMapping library with a natural image that has not been learned in advance with dictool as the recognition target, this can be realized on the application-side by using a custom listener.

SCE_SMART_SCENE_MAPPING_INIT_WAAR

In this initialization mode, initialization is carried out using an AR Play Card.

Note that AR Play Card registration to the SceneMapping library is required before executing `sceSmartSceneMappingStart()`. Register the unique ID assigned per AR Play Card as the recognition target ID with `sceSmartSceneMappingRegisterTarget()`, and then call `sceSmartSceneMappingStart()`.

Initialization will complete when the recognition target AR Play Card is correctly recognized.

SCE_SMART_SCENE_MAPPING_INIT_HFG

In this initialization mode, horizontal from gravity (HFG) initialization is carried out using motion sensors. A horizontal plane is calculated based on the gravity direction detected by the motion sensors, and the world coordinate system is then defined based on the horizontal plane.

Initialization will complete after calling `sceSmartSceneMappingStart()` when it is confirmed within the library that sufficient initialization points are in the image.

Note
Use `sceSmartSceneMappingDispatchAndQuery()` and `sceSmartSceneMappingRunCore()` to pass motion sensor information. The photographed object will be processed as a horizontal plane. If initialization points are insufficient (that is, if a featureless object is photographed) initialization will not be completed.

SCE_SMART_SCENE_MAPPING_INIT_VFG

In this initialization mode, vertical from gravity (VFG) initialization is carried out using motion sensors. A vertical plane is calculated based on the gravity direction detected by the motion sensors and on the premise that the camera is directly confronting the vertical plane; the world coordinate system is then defined based on the vertical plane.

At initialization, the camera and the vertical plane need to be directly confronting each other. Initialization will complete after calling `sceSmartSceneMappingStart()` when it is confirmed within the library that sufficient initialization points are in the image.

Note
Use `sceSmartSceneMappingDispatchAndQuery()` and `sceSmartSceneMappingRunCore()` to pass motion sensor information. The photographed object will be processed as a vertical plane. If initialization points are insufficient (that is, if a featureless object is photographed) initialization will not be completed.

SCE_SMART_SCENE_MAPPING_INIT_SFM

In this initialization mode, structure from motion (SFM) initialization is carried out using the camera's motion parallax. A 3D structure composed of initialization points is calculated based on camera images from two viewpoints with different focus position. Then, the dominant plane of the section photographed at the center of the camera image is calculated based on the 3D structure, and the world coordinate system is defined based on the dominant plane.

After calling `sceSmartSceneMappingStart()`, the SceneMapping library will check whether the image shows sufficient initialization points. If sufficient initialization points are shown, the SceneMapping library will continue tracking the initialization points until parallax is sufficient. When parallax becomes sufficient, the SceneMapping library will complete the initialization process.

Note
For example, the library will not be initialized if parallax is insufficient, such as when the camera is rotated in the same spot. If initialization points are insufficient - that is, if a featureless object is photographed - initialization will not be completed.
If no motion sensor measurement value is passed (that is, if you do not use `sceSmartSceneMappingDispatchAndQuery()` and `sceSmartSceneMappingRunCore()`), the photographed object will be processed by assuming that it is a flat plane.

SCE_SMART_SCENE_MAPPING_INIT_CUSTOM

In this initialization mode, custom initialization is carried out using a custom listener that is uniquely defined by the application.

To carry out custom initialization, prepare a callback function defined with `SceSmartSceneMappingCustomListenerOnLocalizationRequested` on the application-side and call `sceSmartSceneMappingSetCustomListener()`. After `sceSmartSceneMappingStart()` is called, this callback function will be called per frame. For custom listener use, refer to the "Use a custom listener" section, and the "For Better SceneMapping Library Performance" of the "6: Reference Information" chapter.

SCE_SMART_SCENE_MAPPING_INIT_DRYRUN

This is a test mode for checking how many initialization points there are inside an image. Actual initialization will not be carried out in this mode and the world coordinate system will not be defined

After calling `sceSmartSceneMappingStart()`, the SceneMapping library will continue checking the number and location of initialization points within the image. Obtain the number and position of initialization points with `sceSmartSceneMappingGetNumInitializationPoints()` and `sceSmartSceneMappingGetInitializationPointInfo()`.

SCE_SMART_SCENE_MAPPING_INIT_NULL

This initialization mode uses a scene map that has been created in advance. The world coordinate system will not be defined in this mode.

After initializing the SceneMapping library with `sceSmartSceneMappingStart()`, you will need to load a previously created scene map with `sceSmartSceneMappingLoadMap()`, and to perform re-localization with `sceSmartSceneMappingForceLocalize()` after it is loaded. The initialization state will transition to `SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE` and re-localization will be carried out. A transition will be made to `SCE_SMART_SCENE_MAPPING_STATE_SLAM` when re-localization is successful.

The scene map will become fixed, and searches for new landmarks will stop. Fixed scene maps can be unfixed by calling `sceSmartSceneMappingFixMap(SCE_FALSE)`; however, landmarks added after unfixing will not disappear even if `sceSmartSceneMappingReset()` is called.

Note
Scene map files do not have backward compatibility. Refer to the "Compatibility with Scene Map Files of the SceneMapping Library" section below for more information on scene map file versions. If the scene map is invalid and recognition processing is executed with `sceSmartSceneMappingRun()` or `sceSmartSceneMappingDispatchAndQuery()`, the library state will transition to `SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE_IMPOSSIBLE`.

Dispatch Mode

Because recognition processing by the SceneMapping library takes time, it will be efficient to execute the thread performing recognition processing separate from the main thread of the application. In such a case, specify the "dispatch mode" in `sceSmartSceneMappingSetDispatchMode()` for specifying whether or not to asynchronously execute `sceSmartSceneMappingDispatchAndQuery()`, which prepares input data, and `sceSmartSceneMappingRunCore()`, which executes recognition processing based on the prepared input data.

The default dispatch mode is the synchronous mode (`SCE_SMART_SCENE_MAPPING_DISPATCH_MODE_SYNC`). While recognition processing is being executed by `sceSmartSceneMappingRunCore()` in this mode, the next input data will not be prepared even if `sceSmartSceneMappingDispatchAndQuery()` is called on a separate thread. On the other hand, when the asynchronous mode (`SCE_SMART_SCENE_MAPPING_DISPATCH_MODE_ASYNC`) is specified as the dispatch mode, input data will be prepared immediately when `sceSmartSceneMappingDispatchAndQuery()` is called regardless of the recognition processing state. Processing will be more efficient as there will no wait time for synchronization between the `sceSmartSceneMappingDispatchAndQuery()` thread and `sceSmartSceneMappingRunCore()` thread.

Basic Procedure for Recognition/Estimate Processing

The basic procedure for carrying out recognition/estimate processing using the SceneMapping library is explained below.

(1) Initialize libsmart

Call `sceSmartInit()` to initialize libsmart. The `SceSmartMemoryAllocator` structure storing a function pointer to a memory allocator can be passed to `sceSmartInit()`. When the argument of `sceSmartInit()` is NULL, libsmart will be initialized so that `malloc()` and `free()` will be used for memory allocation.

(2) Set the mask feature

When using the SceneMapping library in combination with the TargetTracking library, call `sceSmartSceneMappingEnableMask()` to enable the mask feature before initializing the SceneMapping library. As an argument, specify the maximum number of triangular patches required for the mask area to specify upon obtaining recognition processing results.

(3) Set the dispatch mode

When carrying out multithread processing, call `sceSmartSceneMappingSetDispatchMode()` to specify the timing at which to transfer input data between the thread preparing input data with `sceSmartSceneMappingDispatchAndQuery()` and the thread executing recognition processing with `sceSmartSceneMappingRunCore()`. For details on the dispatch mode, refer to the "Dispatch Mode" section, and the "`SceSmartSceneMappingDispatchMode`" section of the "libsmart Reference" document.

(4) Initialize the SceneMapping library

Initialize the SceneMapping library and prepare for recognition/estimate processing. Call `sceSmartSceneMappingStart()` with the SLAM initialization mode represented by the `SceSmartSceneMappingInitMode` enumerator type specified as the argument.

Note
Depending on the initialization mode, recognition targets must be registered before calling `sceSmartSceneMappingStart()`. For details on the initialization mode, refer to the "SceneMapping Library Initialization Modes" section.

(5) Execute recognition/estimate processing

Below is an explanation of two execution examples: one using only images with single-thread processing, and one using images and motion sensors with multi-thread processing.

Note
Although it is possible to use both images and motion sensors in a single-thread processing specifications-wise, high recognition performance cannot be expected in such cases since there will be a time difference between image input and motion sensor measurement.

Execution Using Only Images with Single-thread Processing

The data input for executing recognition/estimate processing consists of timestamped grayscale images. Grayscale images must be 8-bit, with a width of `SCE_SMART_IMAGE_WIDTH` pixels and a height of `SCE_SMART_IMAGE_HEIGHT` pixels.

In single-thread processing, execute `sceSmartSceneMappingRun()`. When you execute `sceSmartSceneMappingRun()`, it will return the structure `SceSmartSceneMappingResult`, which stores the results of processing. The camera's pose and timestamp, as well as the SceneMapping library's internal state, will return to the `SceSmartSceneMappingResult` structure.

Note
Input images will be deep-copied in the SceneMapping library.

A code example using only images with single-thread processing is indicated below.

```
{
    sceSmartInit();
    id = sceSmartCreateLearnedImageTarget();
    sceSmartSceneMappingRegisterTarget(id);
    sceSmartSceneMappingEnableMask(max);
    sceSmartSceneMappingStart(SCE_SMART_SCENE_MAPPING_INIT_LEARNED_IMAGE);

    while (true) {
        GET_SENSOR_MEASUREMENT();
        SETUP_MASK();
        sceSmartSceneMappingRun(&result, &image, time);
    }

    sceSmartSceneMappingUnregisterTarget(id);
    sceSmartSceneMappingStop();
    sceSmartDestroyTarget(id);
    sceSmartRelease();
}
```

Execution Using Images and Motion Sensors with Multi-thread Processing

The data input for execution processing consists of timestamped grayscale images and information on measurements by the motion sensors. Grayscale images must be 8-bit, with a width of `SCE_SMART_IMAGE_WIDTH` pixels and a height of `SCE_SMART_IMAGE_HEIGHT` pixels. Information on motion sensor measurements must consist of libmotion's measurement values, listed without intervals or duplications and in order from the newest measured time to older measurements.

In multi-thread processing, use the three functions `sceSmartSceneMappingDispatchAndQuery()`, `sceSmartSceneMappingRunCore()` and `sceSmartSceneMappingPropagateResult()`. When employing the masking functionality, use `sceSmartSceneMappingDispatchAndQueryWithMask()`. `sceSmartSceneMappingDispatchAndQuery()` will prepare input data for the next recognition process, and obtain the results of the previous one. `sceSmartSceneMappingRunCore()` will perform the actual processing. `sceSmartSceneMappingPropagateResult()` will propagate the recognition results of a given time to a specified time. Propagation will, to a certain extent, compensate for delays in calculation.

The processing volumes of `sceSmartSceneMappingDispatchAndQuery()` and `sceSmartSceneMappingPropagateResult()` are small, but that of `sceSmartSceneMappingRunCore()` is large; therefore, perform processing by assigning these functions appropriately to the threads. Also refer to the Figure 14 for a multi-thread timeline.

A code example using images and motions sensors with multi-thread processing is indicated below.

```
void game_thread()
{
   // game thread
   sceSmartInit();
   id = sceSmartCreateLearnedImageTarget();
   sceSmartSceneMappingRegisterTarget(id);
   sceSmartSceneMappingEnableMask(max);
        sceSmartSceneMappingSetDispatchMode(
              SCE_SMART_SCENE_MAPPING_DISPATCH_MODE_ASYNC);

   sceSmartSceneMappingStart(SCE_SMART_SCENE_MAPPING_INIT_LEARNED_IMAGE);
   while (true) {
      GET_SENSOR_MEASUREMENT();
      SETUP_MASK();
      // when the masking process is enabled.
      sceSmartSceneMappingDispatchAndQueryWithMask(&result...);
      // when the masking process is disabled.
      // sceSmartSceneMappingDispatchAndQuery(&result...);
      RAISE_EVT();
      // Compensate for the computation delay.
      sceSmartSceneMappingPropagateResult(&result...)
   }
   sceSmartSceneMappingUnregisterTarget(id);
   sceSmartSceneMappingStop();
   sceSmartDestroyTarget(id);
   sceSmartRelease();
}

void slam_thread()
{
   // SLAM thread
   while (true) {
         WAIT_EVT();
      {
         sceSmartSceneMappingRunCore();
      }
   }
}
```

(6) Obtain results

The camera's pose and timestamp, as well as the SceneMapping library's internal state, are stored in the structure `SceSmartSceneMappingResult`, which is obtained with `sceSmartSceneMappingRun()` or `sceSmartSceneMappingDispatchAndQuery()`. If the initialization mode was `SCE_SMART_SCENE_MAPPING_INIT_LEARNED_IMAGE`, the ID of the target recognized at initialization will also be stored.

You can also obtain landmark information with `sceSmartSceneMappingGetLandmarkInfo()`. Obtain the number of landmarks on a map with `sceSmartSceneMappingGetNumLandmarks()`.

Note
The SceneMapping library expands the scene map while processing is executed. Scene map expansion will stop when the number of landmarks comprising the scene map reaches `SCE_SMART_SCENE_MAPPING_MAX_NUM_LANDMARKS`.

(7) Save a scene map

To save a scene map, call `sceSmartSceneMappingSaveMap()` when the library is in the SLAM state (`SCE_SMART_SCENE_MAPPING_STATE_SLAM`).

Note
Scene map files do not have backward compatibility. Refer to the "7: Notes" for more information on scene map file versions.

(8) Stop the SceneMapping library

If you wish to stop processing, call `sceSmartSceneMappingStop()` to stop the SceneMapping library's processing. The SceneMapping library's memory for calculation will be released.

(9) Unregister and delete the recognition target

If a recognition target has been registered, unregister it from the library by calling `sceSmartSceneMappingUnregisterTarget()`, and then delete the recognition target by calling `sceSmartDestroyTarget()`.

(10) Terminate libsmart

Call `sceSmartRelease()` to terminate libsmart.

Re-initialize the SceneMapping library

If you want to re-initialize the SceneMapping library with an initialization mode that differs from the current mode, call `sceSmartSceneMappingStop()` and then call `sceSmartSceneMappingStart()`.

Note
The scene map will not be deleted and will remain only when the current initialization mode is `SCE_SMART_SCENE_MAPPING_INIT_NULL`.

When you wish to re-initialize the library with the same initialization mode, call `sceSmartSceneMappingReset()`. This returns the state of the library to what it was immediately after the call of `sceSmartSceneMappingStart()`. Initialization will be completed faster than when `sceSmartSceneMappingStop()` and `sceSmartSceneMappingStart()` are executed because the SceneMapping library's memory for calculation will not be released.

Use a custom listener

To enable a custom listener, prepare callback functions defined by `SceSmartSceneMappingCustomListenerOnLocalizationRequested` and `SceSmartSceneMappingCustomListenerOnLandmarkDetected` on the application side and execute `sceSmartSceneMappingSetCustomListener()`.

`SceSmartSceneMappingCustomListenerOnLocalizationRequested` is the callback that is called when the SceneMapping library is initialized in the custom initialization mode (`SCE_SMART_SCENE_MAPPING_INIT_CUSTOM`). Moreover, this callback will be called for every frame after library initialization; estimate the pose in the callback.

When this callback succeeds in estimating the pose in the `SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE` state, the library will start SLAM again with that pause as the initial value. Moreover, when pose estimation succeeds in the `SCE_SMART_SCENE_MAPPING_STATE_SLAM` state, the SceneMapping library will overwrite the internally-held pose. For details, refer to the "For Better SceneMapping Library Performance" of the "6: Reference Information" chapter.

Note
When in the `SCE_SMART_SCENE_MAPPING_STATE_SEARCH` state and upon first estimating the pose, the callback must estimate both the pose and the landmark points. Note that `SCE_SMART_SCENE_MAPPING_STATE_LOCALIZE_IMPOSSIBLE` will result if there are too few custom landmark points to which coordinates have been set.

`SceSmartSceneMappingCustomListenerOnLandmarkDetected` is the callback that will be called for every frame when the library is in the `SCE_SMART_SCENE_MAPPING_STATE_SLAM` state. 2D coordinates will be passed from the library for feature points in the image; estimate 3D coordinates of the landmarks within the callback. However, note that estimate result 3D coordinates may be deleted in the library. `SceSmartSceneMappingCustomListenerOnLandmarkDetected` is a feature independent from `SceSmartSceneMappingCustomListenerOnLocalizationRequested` and can be combined with another initialization feature provided by the SceneMapping library. For details, refer to the "For Better SceneMapping Library Performance" of the "6: Reference Information" chapter.

API Summary

Main APIs Used in Basic Recognition/Estimate Processing

API	Description
`sceSmartInit()`	Initializes libsmart
`SceSmartMemoryAllocator`	Memory allocator
`sceSmartSceneMappingEnableMask()`	Enables the masking feature
`sceSmartSceneMappingSetDispatchMode()`	Sets the dispatch mode
`SceSmartSceneMappingInitMode`	SLAM's initialization mode
`sceSmartSceneMappingStart()`	Starts up the SceneMapping library
`sceSmartSceneMappingRun()`	Executes recognition processing on the input image
`SceSmartSceneMappingResult`	Results of scene mapping
`sceSmartSceneMappingDispatchAndQuery()`	Prepares input data and obtains the results of recognition processing
`sceSmartSceneMappingDispatchAndQueryWithMask()`	Prepares input data and obtains the results of recognition processing (using the masking feature)
`sceSmartSceneMappingRunCore()`	Executes recognition processing on input data
`sceSmartSceneMappingPropagateResult()`	Propagates recognition results
`sceSmartSceneMappingGetLandmarkInfo()`	Returns landmark information
`sceSmartSceneMappingGetNumLandmarks()`	Returns the number of landmarks
`sceSmartSceneMappingSaveMap()`	Saves the current scene map
`sceSmartSceneMappingStop()`	Stops the SceneMapping library
`sceSmartSceneMappingUnregisterTarget()`	Unregisters recognition targets
`sceSmartDestroyTarget()`	Deletes a recognition target
`sceSmartRelease()`	Stops using libsmart
`sceSmartSceneMappingReset()`	Resets the SceneMapping library
`SceSmartSceneMappingCustomListenerOnLocalizationRequested`	Callback function for custom initialization of a custom listener
`SceSmartSceneMappingCustomListenerOnLandmarkDetected`	Callback function for landmark detection of a custom listener
`sceSmartSceneMappingSetCustomListener()`	Sets a custom listener

5 How to Use dictool

dictool Overview

dictool is a Windows command tool for creating dictionary files (*.dic) to be used with libsmart from images of recognition targets. Target objects can be detected and tracked by having libsmart load created dictionary files. One dictionary data will be necessary for each recognition target.

Note
Approximately 650 KiB of memory is used to read one dictionary file.

At present, the recognition targets that can be used with libsmart are flat printed materials (photographs, magazines, posters, etc.). Depending on the texture of the recognition target's image, some targets will be easier to identify, and libsmart's performance and usability may vary considerably. For cases allowing processing the texture or selecting the recognition target, refer to the "For Better TargetTracking Library Performance" of the "6: Reference Information" chapter.

Runtime Environment

- Windows 7 64bit
- SSE2-compatible processor

Usable Image Formats

- PNG format: 24-bit color/8-bit grayscale
  (the alpha channel and indexed color are not supported)
- BMP format: 24-bit color/8-bit grayscale
- PGM (P5) format: 8-bit grayscale
- PPM (P6) format: 24-bit color

Use image files scanned from recognition target objects or use image files created from print masters of recognition target objects.

Make sure that the image aspect ratio is the same as the aspect ratio of the recognition target object.

If the shorter side of an image is less than 400 pixels, a warning will appear since it will have negative effects on recognition performance. If the shorter side of an image is less than 200 pixels, creation of a dictionary file will fail since the possibility of gaining sufficient recognition performance is extremely low.

Creating and Evaluating a Dictionary

Procedure to Create a Dictionary

(1) Prepare an image file of the recognition target.

(2) If you do not have an actual recognition target image, print the image. In doing so, take care to ensure that there is no variance between the image's aspect ratio and the printout's.

(3) Measure the physical width (`physicalWidth`) of the actual image or image printout. This value will be used by libsmart for determining the distance from the camera to the object. Accuracy with three digits after decimal point for measuring in units of meters (millimeter accuracy) is required.

(4) Execute dictool with the command below (refer to the "Build Options" item below concerning build options).
A dictionary file of the V9 version (*.v9.dic) will be created in a minute to a few minutes.

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

(5) When the dictionary file is created, the dictionary will be evaluated, and the evaluated score for recognition performance (recognition score) will be displayed. Regarding dictionary evaluation and recognition scores, also refer to the "Dictionary Evaluation" item.

Execution Example

The execution of dictool is exemplified below.

```
> 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]
```

Dictionary Evaluation

When a dictionary is created with dictool, the recognition score will be output onscreen (displayed as `"the score of this target = …"`). This is a simple indicator of the created dictionary's recognition performance; in other words, how much the recognition target - in various orientations and in various environments - will be recognized.

The recognition score is a value between 0 and 100 and a good recognition performance can be expected with a recognition score of about 70 or more. If the value is extremely low (50 or less), libsmart may not recognize the target correctly. For cases allowing processing or selecting the recognition target, refer to the "For Better TargetTracking Library Performance" of the "6: Reference Information" chapter. If an error occurs, refer to the "Error Messages" section.

Moreover, the evaluation option can be specified as follows when executing dictool to evaluate an already created dictionary file. The recognition score that can be obtained will be more accurate than the score obtained upon creation of the dictionary.

```
>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]
```

Build Options

The following build options can be used for dictool:

Option	Format	Description
`-image`	`-image` *`<fileName>`*	Specifies the input image to be recognized. Supports PGM, PPM, BMP and PNG formats
`-help`	`-help`	Displays help. When this option is specified, all other options will be ignored.
`-physicalWidth`	`-physicalWidth` *`<width>`*	Specifies the recognition target's physical lateral size in metric units; this is not its size in pixels. This value is used for scaling the coordinate values output when libsmart recognizes a target. The default value is 1.0 [meter]

Option	Format	Description
`-outimage`	`-outimage` *`<fileName>`*	Saves as a PPM-format image the feature point information stored in the dictionary file detected from the input image. Refer to the "Feature Point Information" item on how to view the results
`-mask`	`-mask` *`<fileName>`*	Specifies the mask image (PGM-format) for specifying the valid area to be recognized within the recognition target image. For details on masking specification, refer to the "Masking Specification" item.

Evaluation Option

The following evaluation option can be used for dictool:

Option	Format	Description
`-i`	`-i` *`<dictionary>`*	Specifies the dictionary file to evaluate

Masking Specification

Upon dictionary creation, an image for the masking (PGM format) can be specified with the `-mask` option to create a dictionary using only a specific area of the input image. The masking image must be of the same size as the input image. Areas with a pixel value of 255 within the masking image will be used as recognition target areas, while other areas will not be used as recognition targets.

This masking feature is intended to be used to exclude a single part of an input image in cases where you wish to accurately register the shape of a recognition target that is not rectangular, such as a coaster; or to register the same recognition targets with single differing parts as one recognition target.

If the area to be excluded from the registered image area is enlarged, recognition performance may deteriorate; check the recognition score when specifying a masking image.

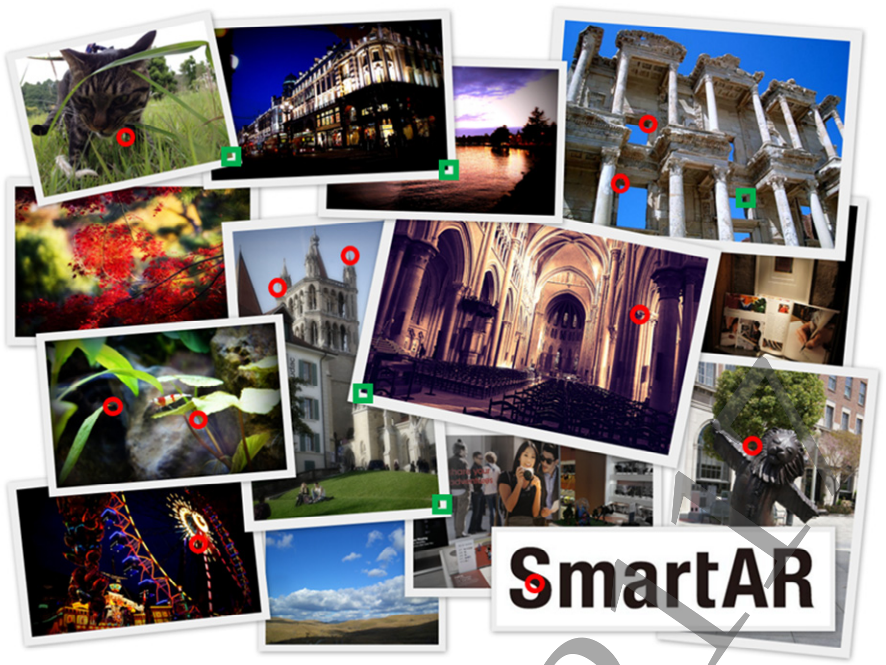
dictool Execution Example When Using the Masking Feature

```
> 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]
        <-NOTE : this score is a brief version of target evaluation. Use EVALUATE
option of this tool for better estimation.
```

Feature Point Information

Save the feature point image onto which features points to be used for recognition have been rendered as follows by specifying the `-outimage` option when creating a dictionary. It would be difficult to pinpoint detection and tracking performance based on this feature point image; however, it may serve as reference for corrections in cases where recognition target detection/tracking performance is extremely poor, or for comparisons with other recognition targets.

Figure 9 Feature Point Image



○: Feature points used for detecting

□: Feature points used for tracking

Viewing the Feature Point Image

The symbols in the feature point image mean the following.

Symbol	Description
Red circle	Feature points used for detecting recognition targets
Green square	Feature points used for tracking recognition targets

Points to note in viewing a feature point image are described below.

- The more uniformly feature points are distributed, the more stable detection and tracking will be.
- If the contrast of the area surrounding these feature points is low, detection rate and tracking performance will be poor.
- If feature points for detection (red circle) are distributed unevenly, that location may be concealed or become unclear, thus causing detection performance to decline greatly.

Error Messages

`"The size of the input image, (<width>, <height>), is too small as a target image"`

The size of the input image is too small. Use an image of the appropriate size, making reference to "Usable Image Formats" item.

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

The input image's aspect ratio is unbalanced. Ensure that the ratio of the longer side to the shorter side is 4:1 or less.

6 Reference Information

World Coordinate System Definition According to the Initialization Mode

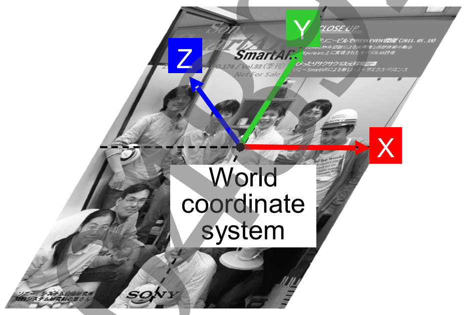
All coordinate systems are right-handed in libsmart. Position and orientation are represented by position vectors and orientation quaternions. Position vectors are camera-centered position vectors in a scene-fixed coordinate system. Orientation quaternions represent rotation of the device-fixed coordinate system from the scene-fixed coordinate system.

Different world coordinate systems (coordinate system fixed to a scene) are defined depending on how the SceneMapping library is initialized.

Initialization with a natural image/AR Play Card

When library initialization is carried out with a natural image as the recognition target, the world coordinate system is defined with the center of the recognition target as the origin, the direction from the recognition target's back to its front as axis Z, and the direction from the recognition target's left to right when seen from the front as axis X. In the case of an AR Play Card, define a world coordinate system in the same manner as for natural images.

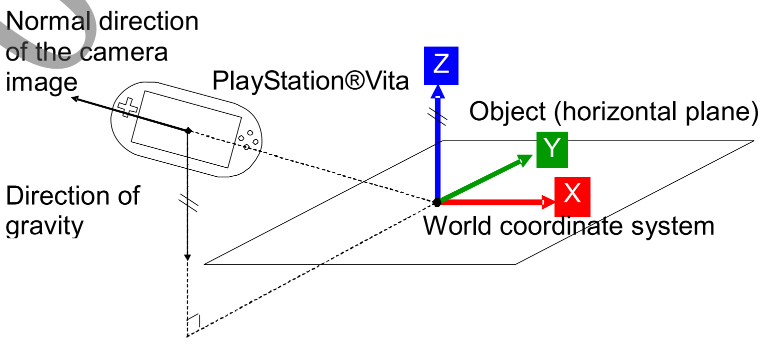
Figure 10 Definition of the Coordinate Axis when Initializing with a Natural Image or an AR Play Card



HFG Initialization

When HFG initialization is carried out, the world coordinate system is defined with the point on the horizontal plane appearing at the camera image's center as its origin, the direction opposite to that of gravity as axis Z, and the direction of the vector projected on the horizontal plane from the camera to the origin as axis Y.

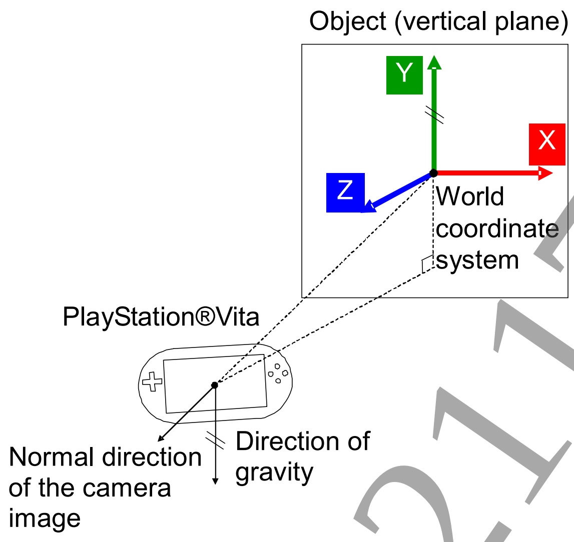
Figure 11 Definition of the Coordinate Axis During HFG Initialization



VFG Initialization

When VFG initialization is carried out, the world coordinate system is defined with the point on the vertical plane appearing at the camera image's center as its origin, the vertical plane's normal line as axis Z, and the direction opposite to that of gravity as axis Y.

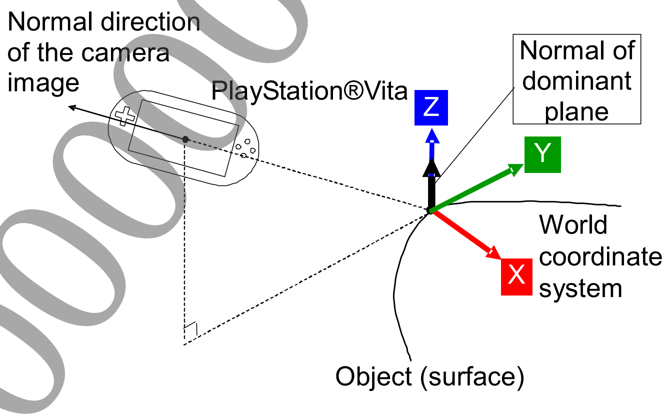
Figure 12 Definition of the Coordinate Axis During VFG Initialization



SFM Initialization

When SFM initialization is carried out, the dominant plane of the camera image's center area is calculated from the 3D structure and the world coordinate system is defined with the point on the dominant plane appearing at the camera image's center as its origin, the dominant plane's normal line as axis Z, and the direction of the vector projected on the dominant plane from the camera to the origin as axis Y.

Figure 13 Definition of the Coordinate Axis During SFM Initialization



Initialization with a Custom Listener

When initialization is carried out with a custom listener, the world coordinate system is provided by the application and there is no specific definition from the library. However, note that the coordinate system to be provided must be right-handed like the others.

For Better TargetTracking Library Performance

Natural Images for Obtaining Good Recognition Results

libsmart identifies and tracks recognition targets by using the feature points extracted during dictionary creation. It chooses parts with high contrast and forming corners (corner points) inside grayscale images (textures) as feature points. Depending on the texture, there are targets that are easy to recognize and those that are difficult to recognize, and libsmart's performance and usability may vary considerably. Consider the following and prepare the natural images to use as recognition targets.

Suitable Targets for Recognition

- Flat, even printed images (photographs, magazines, posters, etc.)
- Images with many complex textures
- Images with many corners (corner points)
- Images with high contrast when turned into gray images

Unsuitable Targets for Recognition

- Images with few textures, or simple images (plain walls, company logos or characters, etc.)
- Images with localized textures
- Images with low texture contrast
- Images on which the same texture is repeated
- Images whose aspect ratio is greatly unbalanced
- Reflective Images (mirrors, laminated images)

Information of recognition targets can be obtained using a feature of dictool. Refer to the "Dictionary Evaluation" item and "Feature Point Information" item of the "5: How to Use dictool" chapter.

How to Select a Natural Image When Creating a Recognition Target on the Spot

When using `sceSmartCreateInstantImageTarget()` to create a recognition target on the spot, there a conditions that must be noted in addition to the above-described points for obtaining good recognition results. It is important that the application be designed to encourage the user to capture an image that satisfies the following conditions.

- Image size is almost the same or slightly larger than the size to be recognized within the camera image
- Image has been captured from the front, directly facing the camera
- Image has been captured without a blur

If the natural image to use as the recognition target dose not sufficiently satisfy the above conditions, the `SCE_SMART_ERROR_BAD_IMAGE` error may return for the call of `sceSmartCreateInstantImageTarget()`.

Note
Generally, a recognition target created on the spot entails longer search time compared to a recognition target created from a dictionary file, and tracking performance will deteriorate if the image is not captured from the front. Thus, the creation of a dictionary file is highly recommended when using an image such as, a package, that can be obtained in advance as a recognition target.

Measures Against Delays in Recognition Processing

Because search time is longer when using a recognition target created on the spot, the recognition result pose will be delayed for the camera image if the recognition target is moving and may cause a misalignment upon superimposing. In the worst case scenario, transition to tracking will fail. To avoid these cases, it is recommended that you only use images with a recognition result timestamp that is close to the timestamp of the image to superimpose onto, and to use recognition results after succeeding for several consecutive frames. All recognition results are displayed in the instant_tracking sample; the above cases can be observed by moving the recognition target.

For the same reasons, when combining use with the SceneMapping library and using a recognition target created on the spot through custom initialization, it is recommended that the recognition success be returned to the SceneMapping library after recognition succeeds for several consecutive frames.

Identifying Problems by Comparing with libsmart Sample Programs

The sample programs distributed with libsmart are made to ensure that libsmart performs fully. Try comparing libsmart's operation with that of sample programs. When recognizing a natural image, run the sample program by replacing its dictionary with a dictionary you have created to achieve a more accurate comparison and to more easily determine whether there is a problem with the program or with the dictionary.

Note
When sample performance greatly deteriorates due to dictionary replacement, confirm that the aspect ratio of the dictionary is the same as that of the printed material in the input image. If the aspect ratio differs, the TargetTracking library cannot effectively function.

For Better SceneMapping Library Performance

Conditions for Obtaining Good Recognition Results

The SceneMapping library will run better under the following conditions. Take care to ensure that image recognition functions as normally as possible.

- Unchanging environments
- Environments rich in corner points when photographed by the camera
- Fast shutter speed
- Bright environments
- Images with little motion blur
- Environments with few repeated patterns
- Smooth and predictable movements of the camera

Points to Note in Processing

Given that the analysis of continuous images carried out by the SceneMapping library becomes more difficult the longer the interval between the images, calculation volume for the analysis may increase, or the accuracy of calculation results may decrease.

To ensure the SceneMapping library's full performance, it is important to continue processing images as fast as possible.

Identifying Problems by Comparing with libsmart Sample Programs

The sample programs distributed with libsmart are made to ensure that libsmart performs fully. Try comparing libsmart's operation with that of sample programs. If initializing with a natural image, run the sample program by replacing its dictionary with a dictionary you have created to achieve a more accurate comparison and to find out more easily whether any problems are caused by the program or by the dictionary. If operation with the sample differs greatly, find out and correct the cause, making reference to the following:

Adjusting the Camera's Frame Rate

Refer to the "libcamera Reference" document and ensure that the camera images are captured at 60 fps. Specifically, check that the value of the `SceCameraInfo` structure's *`wFramerate`* member passed to `sceCameraOpen()` is `SCE_CAMERA_FRAMERATE_60`.

Design buffers and threads so as to ensure that captured images are input in `sceSmartSceneMappingDispatchAndQuery()` as fast as possible. The sample program's buffer management asynchronously requires at least 4 buffers, with 2 buffers for capturing, 1 image buffer for the analyzing thread and 1 image buffer for rendering.

In the same manner, when motion sensors are used, ensure that the values read from the motion sensors are input in `sceSmartSceneMappingDispatchAndQuery()` as fast as possible.

Checking for Chronological Gaps between the Results of the Analysis and Superimposed Display

The output results of a scene map (`SceSmartSceneMappingResult` structure) are calculated based on the time of the input image, and the input image's timestamp is attached to it. Output to the screen is created by superimposing CG on the background of the camera image. Check the difference between the timestamp of the background camera image and the timestamp of the recognition result used in CG rendering. Ideally, this difference should be small or 0, but it is not essential.

Correcting Chronological Gaps between Analysis Results and Superimposed Display

In order to correctly match a camera image with CG on the screen, you will need to use either or both of the methods below. Consider this so as to match the game's features.

(a) Use `sceSmartSceneMappingPropagateResult()` to propagate recognition results to the time of the latest camera image, and output to the screen with CG superimposed according to the propagated result.

(b) Buffer the camera image used in scene mapping and output the buffered camera image to the screen with CG superimposed when the scene mapping results are output.

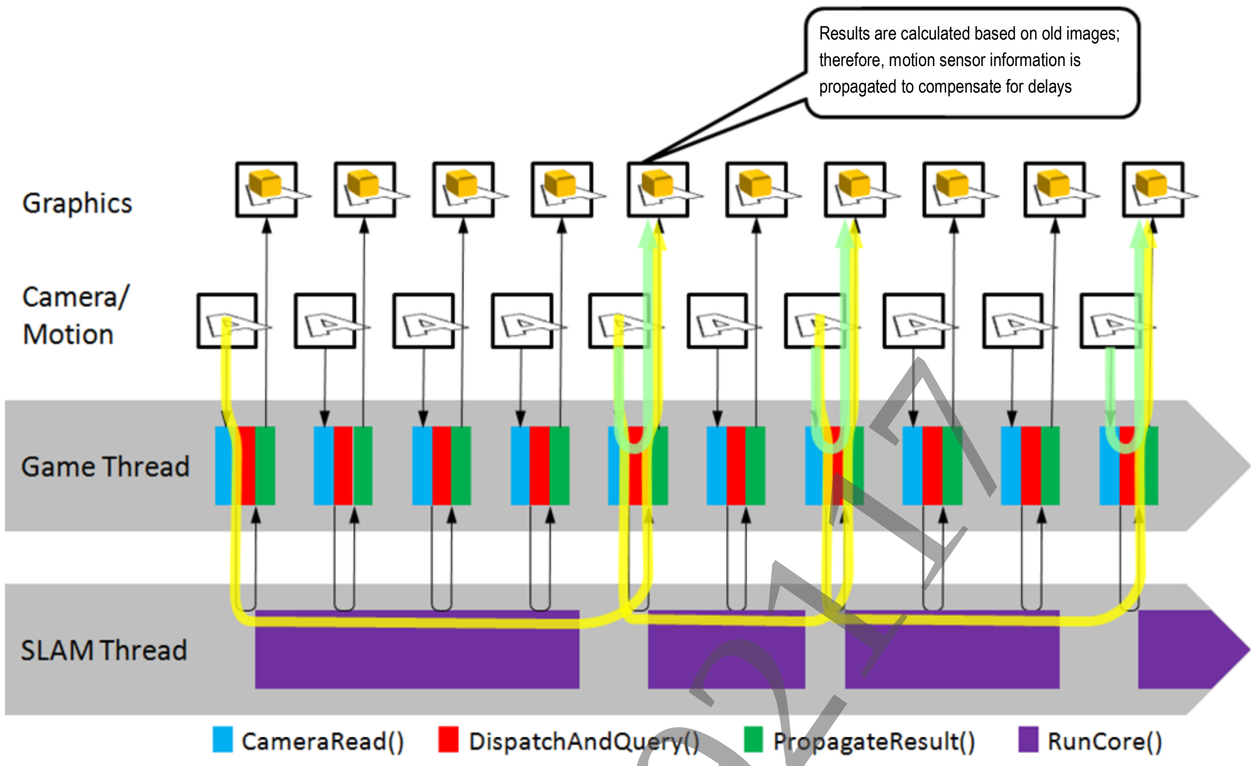
In the sample programs, we have adopted method (a). In this method's case, the screen's delay against the actual movement of the camera will be reduced; however, the error margin will widen due to propagation, and CG may appear slightly out of alignment or to be oscillating.

Enable `ENABLE_SYNCHRONIZATION` and build a sample program to adopt method (b). In this case, the screen's delay against the actual movement of the camera will grow, but the camera image will be matched more accurately to the CG.

Chronological gap improvement using method (a) in multi-thread processing is indicated below.

In the figure, the yellow line indicates the flow of carrying out SLAM estimation based on inputs from the camera or motion sensors and obtaining the results of scene mapping and the green line indicates the flow for the latest camera image. Although inputs from the camera or motion sensors are periodically sent to an application thread, the processing time for scene mapping by `sceSmartSceneMappingRunCore()` generally takes longer than this period and a chronological gap arises between the latest camera image and the scene map to be superimposed onto that image. Thus, the recognition result is propagated to the time of the latest camera image with `sceSmartSceneMappingPropagateResult()` to reduce the gap.

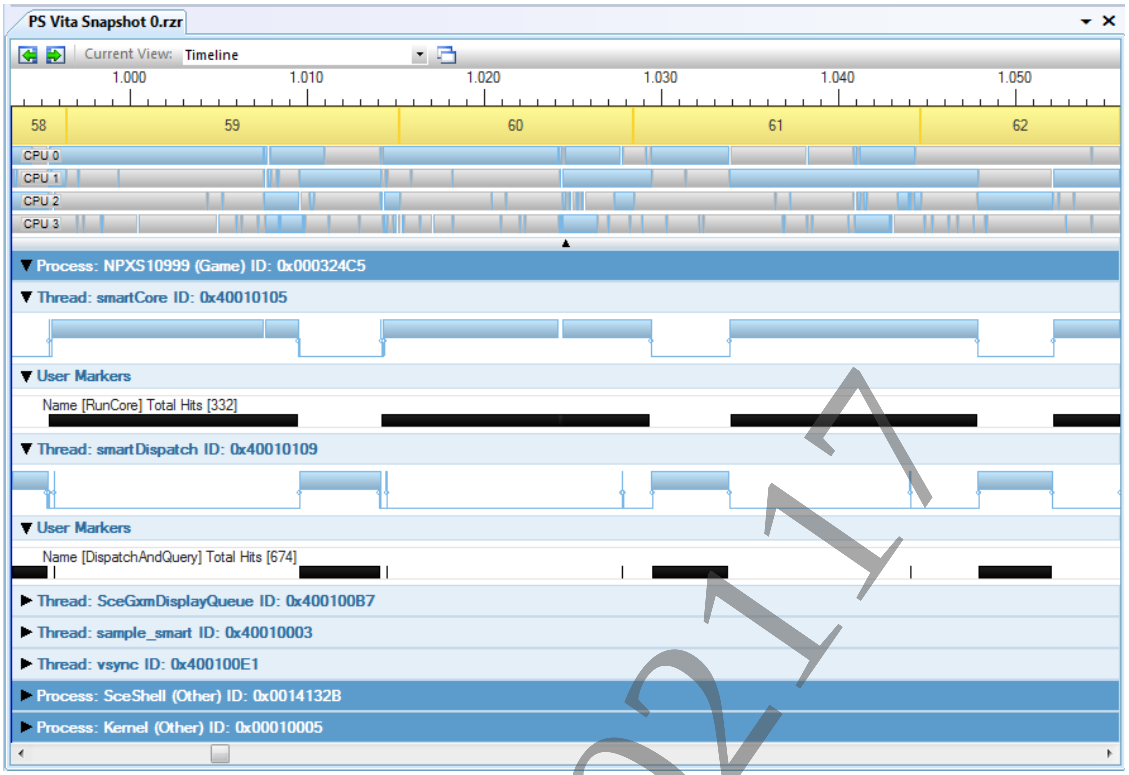
Figure 14 Diagram of the SceneMapping Library's Multi-Thread Timeline


Results are calculated based on old images; therefore, motion sensor information is propagated to compensate for delays
Graphics
Camera/ Motion
Game Thread
SLAM Thread
CameraRead()
DispatchAndQuery()
PropagateResult()
RunCore()

Improving the Operation Status of sceSmartSceneMappingDispatchAndQuery() and sceSmartSceneMappingRunCore()

Profile the program by using libperf and Razor for PlayStation®Vita. Attach a marker to calls of `sceSmartSceneMappingDispatchAndQuery()` and `sceSmartSceneMappingRunCore()`, and confirm that the set markers are packed on the timeline, as in Figure 15. Figure 15 shows the profile result when libsmart's scene_mapping sample is initialized and is in the `SCE_SMART_SCENE_MAPPING_STATE_SLAM` state.

Figure 15 Profile Example of the SceneMapping Library


PS Vita Snapshot 0.rzr
Current View: Timeline
1.000
1.010
1.020
1.030
1.040
1.050
58
59
60
61
62
CPU 0
CPU 1
CPU 2
CPU 3
Process: NPXS10999 (Game) ID: 0x000324C5
Thread: smartCore ID: 0x40010105
User Markers
Name [RunCore] Total Hits [332]
Thread: smartDispatch ID: 0x40010109
User Markers
Name [DispatchAndQuery] Total Hits [674]
Thread: SceGxmDisplayQueue ID: 0x400100B7
Thread: sample_smart ID: 0x40010003
Thread: vsync ID: 0x400100E1
Process: SceShell (Other) ID: 0x0014132B
Process: Kernel (Other) ID: 0x00010005

If the profile result indicates that the call interval between `sceSmartSceneMappingDispatchAndQuery()` and `sceSmartSceneMappingRunCore()` is not appropriate, appropriately set the CPU mask/priority of threads.

Points to Note for Use in Combination with the TargetTracking Library

If an object whose recognition has failed is moved within the screen, it may negatively affect the result of the SceneMapping library's estimation. This is because the mask area corresponding to that object has not been defined, failing to meet the still environment condition on which the estimation is based. Therefore, when using the SceneMapping library together with the TargetTracking library, we recommend creating a scene map in advance by using the SceneMapping library, and fixing the map you obtain. Fixing the scene map will allow you to reliably estimate the position and orientation of the camera and of the objects inside an environment.

Custom Listener Application Examples

Custom listener is a feature to provide information uniquely held by the application to the SceneMapping library. SceneMapping library performance greatly varies depending on the content of this information and the timing at which it is provided. Several custom listener application examples are indicated below.

SceSmartSceneMappingCustomListenerOnLocalizationRequested Callback

As explained in the "4: Using the SceneMapping Library" chapter, the pose information that is stored in the SceneMapping library upon succeeding in estimating the pose will be overwritten by the `SceSmartSceneMappingCustomListenerOnLocalizationRequested` callback. The most basic example of this is custom initialization. By using this overwrite, it is possible, for example, to register a new recognition target with `sceSmartCreateInstantImageTarget()` during SLAM and to move the character onto it.

SceSmartSceneMappingCustomListenerOnLandmarkDetected Callback

Although the basic usage, as explained in the "4: Using the SceneMapping Library" chapter, is to externally provide an already known scene map, this callback can also be used to stabilize SceneMapping library output even when the situation for which to use the scene map has already been largely determined.

Various unique landmark detection processes that demonstrate effectiveness in specific situations are implemented in the scene_mapping_custom sample. The characteristics of each detection processing are introduced below.

Unspecified

Unspecified indicates the same behavior as that of the SceneMapping library under normal circumstances. All feature points passed from the library are not processed in the callback but are passed to the SceneMapping library.

Tiny triangulation

Tiny triangulation uses triangulation to simply estimate 3D coordinates of landmarks based on the pose and deviation of 2D coordinates of the tracked feature points.

Floor

Floor is a callback that registers all feature points to be provided to the SceneMapping library to the z=0 plane of a world coordinate system as landmarks. Unlike HFG initialization and VFG initialization, feature points that are newly detected even during SLAM are passed to the SceneMapping library as landmarks on the z=0 plane. The presumed situation is for a desktop, wall, or floor surface and stable output can be obtained. Moreover, this callback can also be utilized to enable processing such as, to limit landmark registration to the package surface upon recognizing a title logo on a package that has been registered in advance.

Dome

In addition to Floor, Dome is a callback that adds landmarks centered around the pose passed from the SceneMapping library to a spherical surface. When the current pose is passed to the callback, 3D coordinates of landmarks are estimated so that new feature points are placed spherically. A more stable output can be obtained than from usual detection for an application in which PlayStation®Vita is to be spun around on the spot, for example, when a character moves around the user.

Mesh

Mesh is a callback that newly adds feature points, existing on a mesh that is made up of existing landmarks, as landmarks to the plane comprising the mesh.

7 Notes

Compatibility with Scene Map Files of the SceneMapping Library

Scene map files do not have backward compatibility. Check your SDK version to find out whether the version of the scene map files you create can be used.

SDK Version	Scene Map File Version
1.650	1
1.800 or later	3

Version Compatibility of Recognition Target Dictionaries

Recognition target dictionaries come in V8-version and V9-version. V9-version dictionaries are smaller than V8-version dictionaries, and offer better recognition performance. libsmart of the PlayStation®Vita SDK 1.800 or later can also use V8-version dictionaries, but we strongly recommend the use of V9-version dictionaries. Note that libsmart older than the PlayStation®Vita SDK 1.800 cannot load V9-version dictionaries. Compatibility between SDK versions and dictionary versions is as follows:

SDK Version	Dictionary Version
1.650	V8
1.800 or later	V9, V8