Scene Obstruction Detection using TI’s TMS320C66x DSP

User Guide

June 2017

IMPORTANT NOTICE

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) Scene Obstruction Detection Module implemented on TI's TMS320C66x DSP. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.
TI's Scene Obstruction Detection Module implementations are based on IVISION interface. IVISION interface is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

1.2 Intended Audience

This document is intended for system engineers who want to integrate TI's vision and imaging algorithms with other software to build a high level vision system based on C66x DSP.
This document assumes that you are fluent in the C language, and aware of vision and image processing applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) standard will be helpful.

1.3 How to Use This Manual

This document includes the following chapters:

- **Chapter 2 - Introduction**, provides a brief introduction to the XDAIS standards. It also provides an overview of Scene Obstruction Detection and lists its supported features.
- **Chapter 3 - Installation Overview**, describes how to install, build, and run the algorithm.
- **Chapter 4 - Sample Usage**, describes the sample usage of the algorithm.
- **Chapter 5 - API Reference**, describes the data structures and interface functions used in the algorithm.
- **Chapter 6 - Frequently Asked Questions,** provides answers to frequently asked questions related to using Scene Obstruction Detection Module.

1.4 Related Documentation from Texas Instruments

This document frequently refers TI's DSP algorithm standards called XDAIS. To obtain a copy of document related to any of these standards, visit the Texas Instruments website at www.ti.com.

1.5 Abbreviations

The following abbreviations are used in this document.

Table 1 List of Abbreviations

Abbreviation	Description
API	Application Programming Interface

Abbreviation	Description
CIF	Common Intermediate Format
DMA	Direct Memory Access
DMAN3	DMA Manager
DSP	Digital Signal Processing
EVE	Embedded Vision Engine
EVM	Evaluation Module
IRES	Interface for Resources
LANDET	Scene Obstruction Detection Module
ROI	Region Of Interest
QCIF	Quarter Common Intermediate Format
QVGA	Quarter Video Graphics Array
RMAN	Resource Manager
SQCIF	Sub Quarter Common Intermediate Format
VGA	Video Graphics Array
XDAIS	eXpressDSP Algorithm Interface Standard

1.6 Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (‘‘) represents pseudo-code.
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

1.7 Product Support

When contacting TI for support on this product, quote the product name (SCENE_OBSTRUCTION_DETECT post processing Module on TMS320C66x DSP) and version number. The version number of the SCENE_OBSTRUCTION_DETECT post processing Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, Scene Obstruction Detection Module are trademarks of Texas Instruments.

2 Introduction

This chapter provides a brief introduction to XDAIS. It also provides an overview of TI's implementation of Post Processing on the C66x DSP and its supported features.

2.1 Overview of XDAIS

TI's vision analytics applications are based on IVISION interface. IVISION is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS). Please refer documents related to XDAIS for further details.

2.1.1 XDAIS Overview

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. In order to facilitate these functionalities, the IALG interface defines the following APIs:

- `algAlloc()`
- `algInit()`
- `algActivate()`
- `algDeactivate()`
- `algFree()`

The `algAlloc()` API allows the algorithm to communicate its memory requirements to the client application. The `algInit()` API allows the algorithm to initialize the memory allocated by the client application. The `algFree()` API allows the algorithm to communicate the memory to be freed when an instance is no longer required.
Once an algorithm instance object is created, it can be used to process data in real-time. The `algActivate()` API provides a notification to the algorithm instance that one or more algorithm processing methods is about to be run zero or more times in succession. After the processing methods have been run, the client application calls the `algDeactivate()` API prior to reusing any of the instance's scratch memory.
The IALG interface also defines three more optional APIs `algControl()`, `algNumAlloc()`, and `algMoved()`. For more details on these APIs, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

2.2 Overview of the algorithm

The Scene Obstruction Detection module is used to detect the presence of foreign deposits on the camera lens' surface. The input of the module is the output of the TDA3x H3A's auto-focus module. The output is a single 32-bit integer taking the value 0 or 1. 0 means the scene is unobstructed and 1 means the scene is obstructed.

The TDA3x's H3A must be independently programmed in order to produce the focus values that can be consumed by the scene obstruction detection module. Please refer to the TDA3x TRM for further details.
The algorithm uses machine-learning technique to classify the scene. It first computes the relevant features from the H3A AF results to form a feature vector, which is fed into a linear classifier. The linear classifier's coefficients are set at algorithm's creation time and were obtained through an offline learning phase. Please contact TI for details on training the classifier.
When integrated in the system, the algorithm generally does not need to be executed every frame, especially when the capture rate is 30 fps or 60 fps. The parameter frameSkipInterval is used to control when the algorithm is actually fully executed.
Also in order to avoid large objects moving in front of the camera from triggering too many 'scene obstruction' event, the parameter numBlockedFramesThreshold is used to specify how many consecutive frames must be obstructed before an actual 'scene obstructed' event is output.

2.3 Supported Services and Features

This user guide accompanies TI's implementation of Scene Obstruction Detection Algorithm on the TI's C66x DSP.
This version of the Scene Obstruction Detection has the following supported features of the standard:

- Supports TDA3x H3A output.
- Two version sof the library available, one built for DSP and the other one for M4.
- Independent of any operating system.

-

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing Scene Obstruction Detection module. It also provides information on building and running the sample test application.

3.1 System Requirements

This section describes the hardware and software requirements for the normal functioning of the algorithm component.

3.1.1 Hardware

This algorithm has been built and tested TI's C66x DSP and ARM Cortex M4 on TDA3x platform. The algorithm shall work on any future TDA platforms hosting C66x DSP or Cortex M4.

3.1.2 Software

The following are the software requirements for the stand alone functioning of the Scene Obstruction Detection module:

- **Development Environment:** This project is developed using TI's Code Generation Tool. Other required tools used in development are mentioned in section 3.3
- The project are built using g-make (GNU Make). GNU tools comes along with CCS installation.

3.2 Installing the Component

The algorithm component is released as install executable. Following sub sections provided details on installation along with directory structure.

3.2.1 Installing the compressed archive

The algorithm component is released as a compressed archive. To install the algorithm, extract the contents of the exe file onto your local hard disk. The exe file extraction creates a top-level directory called 200.V.SOD.C66X.01.00 . Folder structure of this top level directory is shown in below figure.

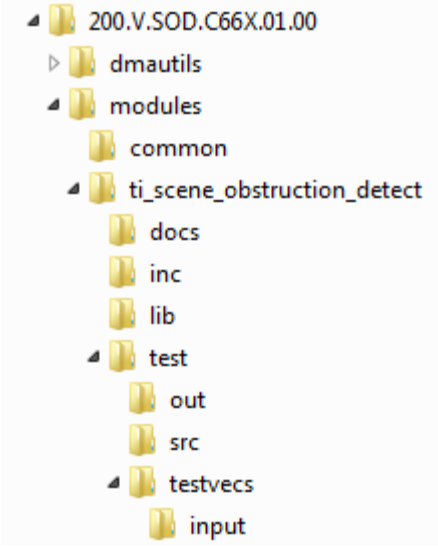

Figure 1 Component Directory Structure

Table 2 Component Directories

Sub-Directory	Description
\modules	Top level folder containing different DSP app modules
\common	Common files for building different DSP modules
\modules \ti_scene_obstruction_detect	Scene Obstruction Detection module for C66x DSP
\modules\ ti_scene_obstruction_detect \docs	User guide and Datasheet for Scene Obstruction Detection module
\modules\ ti_scene_obstruction_detect \inc	Contains iSCENE_OBSTRUCTION_DETECT_TI.h interface file
\modules\ ti_scene_obstruction_detect \lib	Contains Scene Obstruction Detection algorithm library for C66x and Cortex M4
\modules\ ti_scene_obstruction_detect \test	Contains standalone test application source files
\modules\ ti_scene_obstruction_detect \test\out	Contains test application .out executable
\modules\ ti_scene_obstruction_detect \test\src	Contains test application source files

Sub-Directory	Description
\modules\ ti_scene_obstruction_detect \test\testvecs	Contains config, input, output, reference test vectors
\modules\ ti_scene_obstruction_detect \test\testvecs\input	Contains sample input *.buff file

3.3 Object libraries for c66x and ARM cortex M4

Object libraries are available for both c66x and ARM cortex M4. Test applications can be built for c66x or host PC by following the instructions below. The current release does not contain any test application that targets ARM cortex M4 due to lack of makefile.

3.4 Building Sample Test Application

This Scene Obstruction Detection library has been accompanied by a sample test application.
It is possible to build the test application to run on host (PC) mode or to run on the DSP target.
This version of the Scene Obstruction Detection library has been validated with XDAIS tools containing IVISION interface version. Other required components (for test application building) version details are provided below.

- XDAIS
- Code Generation tools
- EDMA3 LLD
- For host building, Visual Studio 2012

3.4.1 Installing XDAIS tools (XDAIS)

XDAIS version can be downloaded from the following website:
http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/xdais/
Extract the XDAIS zip file to the same location where Code Composer Studio has been installed. For example:
C:\CCStudio5.0
Set a system environment variable named "`xdais_PATH`" pointing to <install directory>\<xdais_directory>

3.4.2 Installing Code Generation Tools

Install Code generation Tools from the link

https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm
After installing the CG tools, set the environment variable named "`DSP_TOOLS`" to the installed directory like <install directory>\<cgtools_directory>

3.4.3 Building the Test Application Executable through GMAKE for target execution on DSP

The sample test application that accompanies Scene Obstruction Detection module will run in TI's Code Composer Studio development environment on the target platform. To build and run the sample test application through gmake, follow these steps:

1) Verify that you have installed code generation tools version mentioned
2) Verify that you have installed XDAIS tools version mentioned
3) Verify that appropriate environment variables have been set as discussed in this above sections.
4) Build the sample test application project by gmake
   a) modules\ti_SCENE_OBSTRUCTION_DETECT\test> gmake clean
   b) modules\ti_SCENE_OBSTRUCTION_DETECT\test> gmake all
5) The above step creates an executable file, test_SCENE_OBSTRUCTION_DETECT_algo.out in the modules\ti_SCENE_OBSTRUCTION_DETECT\test\out sub-directory.
6) Open CCS with TDA3x platform selected configuration file. Select Target > Load Program on C66x DSP, browse to the *modules*\ *ti_scene_obstruction_detect* *test**out* sub-directory, select the executable created in step 5, and load it into Code Composer Studio in preparation for execution.
7) Select Target > Run on C66x DSP window to execute the sample test application.
8) The test application reads H3A output data stored in the file *test**testvecs**input**H3A_AF.buff* and executes the scene obstruction detection algorithm.
9) On successful completion, the test application writes pass or fail on the console window.

3.4.4 Building the Test Application Executable through GMAKE for host PC execution (only available in source release).

The sample test application that accompanies Scene Obstruction Detection module will run from command line on PC. To build and run the sample test application through gmake, follow these steps:

1) Verify that you have installed code generation tools version mentioned
2) Verify that you have installed XDAIS tools version mentioned
3) Verify that appropriate environment variables have been set as discussed in this above sections.
4) Verify that the Visual Studio 2012 path *"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin"* has been included in the system path.
5) Build the sample test application project by gmake
   a) modules\ti_scene_obstruction_detect > *gmake TARGET_BUILD=release TARGET_PLATFORM=PC clean*

b) modules\ ti_scene_obstruction_detect > *gmake TARGET_BUILD=release TARGET_PLATFORM=PC all*

6) The above step creates an executable file, *ti_scene_obstruction_detect _algo.out.exe* in the *modules**ti_scene_obstruction_detect**test**out* sub-directory.
7) Open the command window in directory *modules*\ *ti_scene_obstruction_detect**test**out* and type *ti_scene_obstruction_detect_algo.out.exe* to run the text.
8) The test application reads H3A output data stored in the file *test**testvecs**input**H3A_AF.buff* and executes the scene obstruction detection algorithm.
9) During execution, the data is read from file system, processed and results are written to files.

3.5 Uninstalling the Component

To uninstall the component, delete the algorithm directory from your hard disk.

This chapter provides a detailed description of the sample test application that accompanies this Scene Obstruction Detection component.

4 Sample Usage

4.1 Overview of the Test Application

The test application exercises the `IVISION` and extended class of the Scene Obstruction Detection library. The source files for this application are available in the \test\src sub-directories.

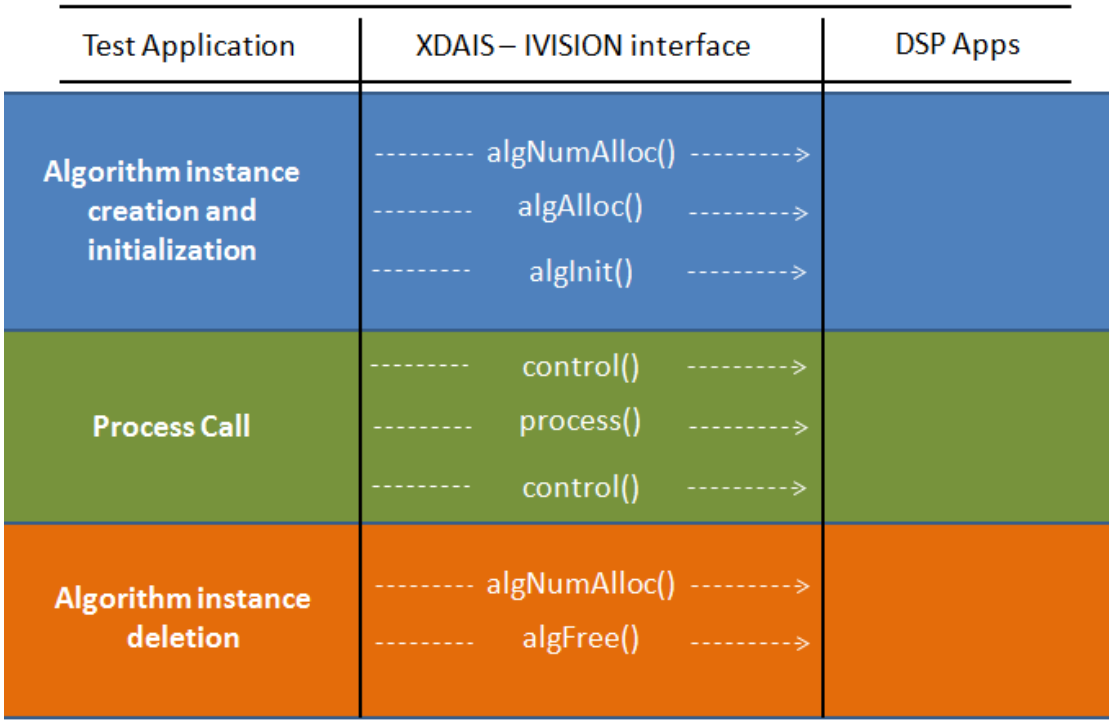

Table 3 Test Application Sample Implementation

The test application is divided into four logical blocks:

- Parameter setup
- Algorithm instance creation and initialization
- Process call
- Algorithm instance deletion

4.2 Parameter Setup

Each algorithm component requires various configuration parameters to be set at initialization. For example, Scene Obstruction Detection requires parameters such as maximum image height, maximum image width, and so on. The test application obtains the required parameters from the Algorithm configuration files.
In this logical block, the test application does the following:

1) Opens the configuration file, listed in SCENE_OBSTRUCTION_DETECT.cfg and reads the various configuration parameters required for the algorithm.

For more details on the configuration files, see Section **Error! Reference source not ound.**.

Sets the `CENE_OBSTRUCTION_DETECT_TI_CreateParams` structure based on the values it reads from the configuration file.
Does the algorithm instance creation and other handshake via. control methods
For each frame reads the image into the application input buffer and makes a process call
For each frame dumps out the disparity map to specified output file.

4.3 Algorithm Instance Creation and Initialization

In this logical block, the test application accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs implemented by the algorithm are called in sequence by `ALG_create()`:

1) `algNumAlloc()` - To query the algorithm about the number of memory records it requires.
   `algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.
   `algInit()` - To initialize the algorithm with the memory structures provided by the application.

A sample implementation of the create function that calls `algNumAlloc()`, `algAlloc()`, and `algInit()` in sequence is provided in the `ALG_create()` function implemented in the alg_create.c file.
IMPORTANT! In this release, the algorithm assumes a fixed number of EDMA channels and does not rely on any IRES resource allocator to allocate the physical EDMA channels. This EDMA channel allocation method will be moved to IRES based mechanism in subsequent releases.

4.4 Process Call

After algorithm instance creation and initialization, the test application does the following:

1) Sets the dynamic parameters (if they change during run-time) by calling the `control()` function with the `IALG_SETPARAMS` command.
   Sets the input and output buffer descriptors required for the `process()` function call. The input and output buffer descriptors are obtained by calling the `control()` function with the `IALG_GETBUFINFO` command.
   Calls the `process()` function to post-process the provided disparity map. The inputs to the process function are input and output buffer descriptors, pointer to the `IVISION_InArgs` and `IVISION_OutArgs` structures.
   When the `process()` function is called, the software triggers the start of algorithm.

The `control()` and `process()` functions should be called only within the scope of the `algActivate()` and `algDeactivate()` XDAIS functions, which activate and deactivate the algorithm instance respectively. If the same algorithm is in-use between two process/control function calls, calling these functions can be avoided. Once an algorithm is activated, there can be any ordering of `control()` and `process()` functions. The following APIs are called in sequence:

1) `algActivate()` – To activate the algorithm instance.
   `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the eight control commands.
   `process()` - To call the Algorithm with appropriate input/output buffer and arguments information.
   `algDeactivate()` - To deactivate the algorithm instance.

The do-while loop encapsulates frame level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts.
If the algorithm uses any resources through RMAN, then user must activate the resource after the algorithm is activated and deactivate the resource before algorithm deactivation.

4.5 Algorithm Instance Deletion

Once `process` is complete, the test application must release the resources granted by the IRES resource Manager interface if any and delete the current algorithm instance. The following APIs are called in sequence:

1) `algNumAlloc()` - To query the algorithm about the number of memory records it used.
   `algFree()` - To query the algorithm to get the memory record information.

A sample implementation of the delete function that calls `algNumAlloc()` and `algFree()` in sequence is provided in the `ALG_delete()` function implemented in the alg_create.c file.

4.6 Frame Buffer Management

4.6.1 Input and Output Frame Buffer

The algorithm has input buffers that stores frames until they are processed. These buffers at the input level are associated with a bufferId mentioned in input buffer descriptor. The output buffers are similarly associated with bufferId mentioned in the output buffer descriptor. The IDs are required to track the buffers that have been processed or locked. The algorithm uses this ID, at the end of the process call, to inform back to application whether it is a free buffer or not. Any buffer given to the algorithm should be considered locked by the algorithm, unless the buffer is returned to the application through `IVISION_OutArgs->inFreeBufID[] and IVISION_OutArgs->outFreeBufID[]`.
For example,

Process Call #	1	2	3	4	5
bufferID (input)	1	2	3	4	5
bufferID (output)	1	2	3	4	5
inFreeBufID	1	2	3	4	5
outFreeBufID	1	2	3	4	5

The input buffer and output buffer is freed immediately once process call returns.

4.6.2 Input Buffer Format

Algorithm expects the the input data to represent AF data produced by the H3A hardware engine present in TDA3X ’s ISP. Please refer to the TDA3x’s TRM, paragraph ISS ISP H3A AF for details on the data format.

4.6.3 Output Buffer Format

The ouput buffer is a single 32-bit word having the value 0 or 1. 0 means the scene is clear and 1 means the scene is obstructed.

5 API Reference

This chapter provides a detailed description of the data structures and interfaces functions used by Scene Obstruction Detection.

Disclaimer: although the naming convention uses the prefix SCENE_OBSTRUCTION_DETECT, the algorithm described in this document only concerns the post processing part. It is always assumed that the input disparity map and associated cost maps have been generated prior to calling the algorithm described here, by a different SCENE_OBSTRUCTION_DETECT module possibly running on a different core. The prefix SCENE_OBSTRUCTION_DETECT is used in prevision for future development that merges both processing and post-processing into a single algorithm module.

5.1.1 IVISION_Params

Description

This structure defines the basic creation parameters for all vision applications.

Fields

Field	Data Type	Input/ Output	Description
algParams	IALG_Params	Input	IALG Params
cacheWriteBack	ivisionCacheWriteBack	Input	Function pointer for cache write back for cached based system. If the system is not using cache fordata memory then the pointer can be filled with NULL. If the algorithm recives a input buffer with IVISION_AccessMode as IVISION_ACCESSMODE_CPU and the ivisionCacheWriteBack as NULL then the algorithm will return with error

5.1.2 IVISION_Point

Description

This structure defines a 2-dimensional point

Fields

Field	Data Type	Input/ Output	Description
X	XDAS_Int32	Input	Not used in this algorithm
Y	XDAS_Int32	Input	Not used in this algorithm

5.1.3 IVISION_BufPlanes

Description

This structure defines a generic plane descriptor

Fields

Field	Data Type	Input/ Output	Description
Buf	Void*	Input	Number of points in the polygon
width	XDAS_UInt32	Input	Not used in this algorithm
height	XDAS_UInt32	Input	Not used in this algorithm
frameROI	IVISION_Rect	Input	Not used in this algorithm
subFrameROI	IVISION_Rect	Input	Not used in this algorithm
freeSubFrameROI	IVISION_Rect	Input	Not used in this algorithm
planeType	XDAS_Int32	Input	Not used in this algorithm
accessMask	XDAS_Int32	Input	Indicates how the buffer was filled by the producer, It is IVISION_ACCESSMODE_HWA or IVISION_ACCESSMODE_CPU

5.1.4 IVISION_BufDesc

Description

This structure defines the iVISION buffer descriptor

Fields

Field	Data Type	Input/ Output	Description
numPlanes	Void*	Input	Number of planes
bufPlanes[IVISION_MAX_NUM_PLANES]	IVISION_BufPlanes	Input	Description of each plane
formatType	XDAS_UInt32	Input	Height of the buffer (in lines)
bufferId	XDAS_Int32	Input	Identifier to be attached with the input frames to be processed. It is useful when algorithm requires buffering for input buffers. Zero is not supported buffer id and a reserved value

Field	Data Type	Input/ Output	Description
Reserved[2]	XDAS_UInt32	Input	Reserved for later use

5.1.5 IVISION_BufDescList

Description

This structure defines the iVISION buffer descriptor list. IVISION_InBufs and IVISION_OutBufs is of the same type

Fields

Field	Data Type	Input/ Output	Description
Size	XDAS_UInt32	Input	Size of the structure
numBufs	XDAS_UInt32	Input	Number of elements of type IVISION_BufDesc in the list
bufDesc	IVISION_BufDesc **	Input	Pointer to the list of buffer descriptor

5.1.6 IVISION_InArgs

Description

This structure defines the iVISION input arugments

Fields

Field	Data Type	Input/ Output	Description
Size	XDAS_UInt32	Input	Size of the structure
subFrameInfo	XDAS_UInt32	Input	bit0 - Sub frame processing enable (1) or disabled (0) bit1 - First subframe of the picture (0/1) bit 2 - Last subframe of the picture (0/1) bit 3 to 31 – reserved

5.1.7 IVISION_OutArgs

Description

This structure defines the iVISION output arugments

Fields

Field	Data Type	Input/ Output	Description
`Size`	`XDAS_UInt32`	Input	Size of the structure
`inFreeBufIDs[IVISION_MAX_NUM_FREE_BUFFERS]`	`XDAS_UInt32`	Input	Array of bufferId's corresponding to the input buffers that have been unlocked in the Current process call. The input buffers released by the algorithm are indicated by their non-zero ID (previously provided via IVISION_BufDesc#bufferId A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid inFreeBufIDs within the array hence the application can stop searching the array when it encounters the first zero entry. If no input buffer was unlocked in the process call, inFreeBufIDs[0] will have a value of zero.
`outFreeBufIDs [IVISION_MAX_NUM_FREE_BUFFERS]`	`XDAS_UInt32`	Input	Array of bufferId's corresponding to the Output buffers that have been unlocked in the Current process call. The output buffers released by the algorithm are indicated by their non-zero ID (previously provided via IVISION_BufDesc#bufferId A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid inFreeBufIDs within the array hence the application can stop searching the array when it encounters the first zero entry. If no output buffer was unlocked in the process call, inFreeBufIDs[0] will have a value of zero.
`reserved[2]`	`XDAS_UInt32`		Reserved for future usage

5.1.8 Scene Obstruction Detection Enumerations

This section includes the following Scene Obstruction Detection specific enumerations:

- `SCENE_OBSTRUCTION_DETECT_ErrorType`
- `SCENE_OBSTRUCTION_DETECT_InBufOrder`
- `SCENE_OBSTRUCTION_DETECT_OutBufOrder`

5.1.8.1 SCENE_OBSTRUCTION_DETECT_ErrorType

‖ Description

This enumeration defines all the error codes returned by the Scene Obstruction Detection algorithm.

‖ Fields

Field	Data Type	Input/ Output	Description
SCENE_OBSTRUCTION_DETECT_TI_ERRORTYPE_INVALID_NUM_INPUTS	enum	Output	Incorrect number of input buffers passed to the function.
SCENE_OBSTRUCTION_DETECT_TI_ERRORTYPE_CREATE_FAIL	enum	Output	Algorithm creation failed

5.1.8.2 SCENE_OBSTRUCTION_DETECT_InBufOrder

‖ Description

This enumeration defines the purpose of the input buffers.

‖ Fields

Field	Data Type	Input/ Output	Description
SCENE_OBSTRUCTION_DETECT_TI_BUFDESC_IN_FV_STATS	enum	Input	This buffer descriptor provides the input focus values generated by the H3A engine

5.1.8.3 SCENE_OBSTRUCTION_DETECT_OutBufOrder

‖ Description

This enumeration defines the purpose of the output buffers.

‖ Fields

Field	Data Type	Input/ Output	Description
SCENE_OBSTRUCTION_DETECT_TI_BUFDESC_OUT	enum	Output	This buffer is a single 32-bits word filled up by the algorithm. A value of 0 means the scene is not obstructed and a value of 1 means it is obstructed.

5.1.9 Scene Obstruction Detection Data Structures

This section includes the following Scene Obstruction Detection specific extended data structures:

- SCENE_OBSTRUCTION_DETECT_TI_CreateParams
- SCENE_OBSTRUCTION_DETECT_TI_InArgs
- SCENE_OBSTRUCTION_DETECT_TI_OutArgs
- SCENE_OBSTRUCTION_DETECT_TI_output

5.1.9.1 SCENE_OBSTRUCTION_DETECT_TI_CreateParams

‖ Description

This structure defines the create-time input arguments for Scene Obstruction Detection Algorithm instance object.

‖ Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	See `IVISION_Params` data structure for details
`wdrEnable`	uint32_t	Input	Specify whether the sensor capture mode is set to WDR, if wdrEnable=1 then AF data of second/long exposure follows the first exposure.
`paxNumH`	uint32_t	Input	Number of AF paxels in H-direction, must be smaller or equal to `SCENE_OBSTRUCTION_TI_DETECT_MAX_PAXNUMH` and must match the H3A engine's setting.
`paxNumV`	uint32_t	Input	Number of AF pacxels in V-direction, must be smaller or equal to `SCENE_OBSTRUCTION_TI_DETECT_MAX_PAXNUMV` and must match the H3A engine's setting.
`vfEnable`	uint32_t	Input	0: vertical focus disabled, 1: vertical focus enabled. Must match the same setting of the H3A engine.
`classifierCoef`	float[]	Input	Arrays of floats, containing the coefficients of the classifier. The number of elements in the array is given by SCENE_OBSTRUCTION_DETECT_TI_NUM_COEFS.
`scaleFactor`	float	Input	Scale factor that is applied to the feature vector prior to the linear classifier. Must be set to 10.0 if recommended classifier coefficients are used.

Field	Data Type	Input/ Output	Description
`sensitivity`	float	Input	Sensitivity value, -5: less sensitive, 5: most sensitive.
`frameSkipInterval`	*uint32_t*	*Input*	Number of frames to be skipped between each execution of the algorithm 0 means the algorithm runs everyframe 1 means the algorithm runs every other frame. When the algorithm does not execute, the process call returns right away. 2 means the algorithm runs every 2 frames
`numBlockedFramesThreshold`	uint32_t	Input	Number of frames that have to be obstructed before the output is set to 1.
`edma3RmLldHandle`	void*	Input	Pointer to the EDMA3 LLD resource manager handle

5.1.9.2 SCENE_OBSTRUCTION_DETECT_TI_InArgs

‖ Description

This structure contains all the parameters which are given as input to Scene Obstruction Detection algorithm at frame level.

‖ Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	See `IVISION_InArgs` data structure for details.

5.1.9.3 SCENE_OBSTRUCTION_DETECT_TI_OutArgs

‖ Description

This structure contains all the parameters which are given as output by the algorithm.

‖ Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	Output	See IVISION_OutArgs data structure for details.
rsvd1	int32_t	Output	reserved

5.2 Recommended and Supported Values of Parameters

This section provides the supported values for the following data structures:

- SCENE_OBSTRUCTION_DETECT_TI_CreateParams
- SCENE_OBSTRUCTION_DETECT_TI_PostProcOptions
- SCENE_OBSTRUCTION_DETECT_TI_DisparityOptions

Table 4: Supported Values for SCENE_OBSTRUCTION_DETECT_TI_CreateParams

Field	Supported Value
paxNumH	Must match the value used to set up the H3A's H3A_AFPAX2[5:0] PAXHC field. Usually 3.
paxNumV	Must match the value used to set up the H3A's H3A_AFPAX2[5:0] PAXVC field. Usually 3.
vfEnable	Must match the value used to set up the H3A's H3A_PC [20:20] AF_VF_EN field. Usually 1.

classifierCoef	Recommended settings for indoor scene is:
	classifierCoef[0]= 6.010115;
	classifierCoef[1]= 1.562322;
	classifierCoef[2]= -33.339242;
	classifierCoef[3]= 1.963373;
	classifierCoef[4]= -39.755250;
	classifierCoef[5]= 0.0;
	classifierCoef[6]= 0.0;
	classifierCoef[7]= 0.0;
	classifierCoef[8]= 0.0;
	classifierCoef[9]= 0.0;
	classifierCoef[10]= 0.0;
	
	Recommended settings for outdoor scene is:
	classifierCoef[0]= 5.809918
	classifierCoef[1]= 7.272761
	classifierCoef[2]= -194.057086
	classifierCoef[3]= 7.953027
	classifierCoef[4]= -1.078643
	classifierCoef[5]= 0.0
	classifierCoef[6]= 0.0

Field	Supported Value
	classifierCoef[7]= 0.0 classifierCoef[8]= 0.0 classifierCoef[9]= 0.0 classifierCoef[10]= 0.0
scaleFactor	10.0 should be used if the above recommended settings for the coefficients are used.
sensitivity	-2
frameSkipInterval	10
numBlockedFramesThreshold	3

5.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used by Scene Obstruction Detection. The APIs are logically grouped into the following categories:

- **Creation** – `algNumAlloc()`, `algAlloc()`
- **Initialization** – `algInit()`
- **Control** – `control()`
- **Data processing** – `algActivate()`, `process()`, `algDeactivate()`
- **Termination** – `algFree()`

You must call these APIs in the following sequence:

1) `algNumAlloc()`
2) `algAlloc()`
3) `algInit()`
4) `algActivate()`
5) `process()`
6) `algDeactivate()`
7) `algFree()`

`control()` can be called any time after calling the `algInit()` API. `algNumAlloc(), algAlloc(), algInit(), algActivate(), algDeactivate(), and algFree() are standard XDAIS APIs. This document includes only a brief description for the standard XDAIS APIs.` For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

5.4 Creation APIs

Creation APIs are used to create an instance of the component. The term creation mean allocating system resources, typically memory.

‖ Name

`algNumAlloc()` – determine the number of buffers that an algorithm requires

‖ Synopsis

XDAS_Int32 algNumAlloc(Void);

‖ Arguments

Void

‖ Return Value

XDAS_Int32; /* number of buffers required */

‖ Description

`algNumAlloc()` returns the number of buffers that the `algAlloc()` method requires. This operation allows you to allocate sufficient space to call the `algAlloc()` method. `algNumAlloc()` may be called at any time and can be called repeatedly without any side effects. It always returns the same result. The `algNumAlloc()` API is optional.
For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

algAlloc()
Name
`algAlloc()` – determine the attributes of all buffers that an algorithm requires

‖ Synopsis

XDAS_Int32 algAlloc(const IALG_Params *params, IALG_Fxns **parentFxns, IALG_MemRec memTab[]);

‖ Arguments

IALG_Params *params; /* algorithm specific attributes */
IALG_Fxns **parentFxns;/* output parent algorithm functions */
IALG_MemRec memTab[]; /* output array of memory records */

‖ Return Value

XDAS_Int32 /* number of buffers required */

‖ Description

`algAlloc()` returns a table of memory records that describe the size, alignment, type, and memory space of all buffers required by an algorithm. If successful, this function returns a positive non-zero value indicating the number of records initialized.
The first argument to `algAlloc()` is a pointer to a structure that defines the creation parameters. This pointer may be `NULL`; however, in this case, `algAlloc()` must assume default creation parameters and must not fail.
The second argument to `algAlloc()` is an output parameter. `algAlloc()` may return a pointer to its parent's IALG functions. If an algorithm does not require a parent object to be created, this pointer must be set to `NULL`.
The third argument is a pointer to a memory space of size `nbufs * sizeof(IALG_MemRec)` where, `nbufs` is the number of buffers returned by `algNumAlloc()` and `IALG_MemRec` is the buffer-descriptor structure defined in ialg.h. After calling this function, `memTab[]` is filled up with the memory requirements of an algorithm.
For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

algNumAlloc()
algFree()

5.5 Initialization API

Initialization API is used to initialize an instance of the algorithm. The initialization parameters are defined in the `IVISION_Params` structure (see section 5.1.1 for details).

‖ Name

`algInit()` – initialize an algorithm instance

‖ Synopsis

XDAS_Int32 algInit(IALG_Handle handle, IALG_MemRec memTab[], IALG_Handle parent, IALG_Params *params);

‖ Arguments

IALG_Handle handle; /* algorithm instance handle*/
IALG_memRec memTab[]; /* array of allocated buffers */
IALG_Handle parent; /* handle to the parent instance */
IALG_Params *params; /*algorithm init parameters */

‖ Return Value

IALG_EOK; /* status indicating success */
IALG_EFAIL; /* status indicating failure */

‖ Description

`algInit()` performs all initialization necessary to complete the run time creation of an algorithm instance object. After a successful return `from` algInit(), the instance object is ready to be used to process data.
The first argument to `algInit()` is a handle to an algorithm instance. This value is initialized to the base Field of `memTab[0]`.
The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers allocated for an algorithm instance. The number of initialized records is identical to the number returned by a prior call to `algAlloc()`.
The third argument is a handle to the parent instance object. If there is no parent object, this parameter must be set to `NULL`.
The last argument is a pointer to a structure that defines the algorithm initialization parameters.
For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).
Since there is no mechanism to return extended error code for unsupported parameters, this version of algorithm returns `IALG_EOK` even if some parameter unsupported is set.
But subsequence control/process call it returns the detailed error code

‖ See Also

`algAlloc()`,
`algMoved()`

5.6 Control API

Control API is used for controlling the functioning of the algorithm instance during run-time. This is done by changing the status of the controllable parameters of the algorithm during run-time. These controllable parameters are defined in the `IALG_Cmd` data structure.

‖ Name

`control()` – change run time parameters and query the status

‖ Synopsis

XDAS_Int32 (*control) (IVISION_Handle handle, IALG_Cmd id, IALG_Params *inParams, IALG_Params *outParams);

‖ Arguments

IVISION_Handle handle; /* algorithm instance handle */
IALG_Cmd id; /* algorithm specific control commands*/
IALG_Params *inParams /* algorithm input parameters */
IALG_Params *outParams /* algorithm output parameters */

‖ Return Value

IALG_EOK; /* status indicating success */
IALG_EFAIL; /* status indicating failure */

‖ Description

This function changes the run time parameters of an algorithm instance and queries the algorithm's `status`. `control()` must only be called after a successful `call to algInit()` and must never be called after a call `to algFree()`.
The first argument to control() is a handle to an algorithm instance.
The second argument is an algorithm specific control command. See IALG_CmdId enumeration for details.

‖ Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.
- params must not be NULL and must point to a valid `IALG_Params` structure.

‖ Postconditions

The following conditions are true immediately after returning from this function.

- If the control operation is successful, the return value from this operation is equal to `IALG_EOK;` otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value. If status or handle is NULL then Scene Obstruction Detection returns `IALG_EFAIL`
- If the control command is not recognized or some parameters to act upon are not supported, the return value from this operation is not equal to `IALG_EOK`.
- The algorithm should not modify the contents of params. That is, the data pointed to by this parameter must be treated as read-only.

‖ Example

See test bench file, SCENE_OBSTRUCTION_DETECT_tb.c available in the \test\src sub-directory.

‖ See Also

algInit(), algActivate(), process()

5.7 Data Processing API

Data processing API is used for processing the input data.

‖ Name

`algActivate()` – initialize scratch memory buffers prior to processing.

‖ Synopsis

```
void algActivate(IALG_Handle handle);
```

‖ Arguments

IALG_Handle handle; /* algorithm instance handle */

‖ Return Value

Void

Description

`algActivate()` initializes any of the instance's scratch buffers using the persistent memory that is part of the algorithm's instance object.

The first (and only) argument to `algActivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be initialized prior to calling any of the algorithm's processing methods.

For more details, see *TMS320 DSP Algorithm Standard API Reference.* (literature number SPRU360).

‖ See Also

`algDeactivate()`

‖ Name

`process()` – basic encoding/decoding call

‖ Synopsis

XDAS_Int32 (*process)(IVISION_Handle handle, IVISION_inBufs *inBufs, IVISION_outBufs *outBufs, IVISION_InArgs *inargs, IVISION_OutArgs *outargs);

‖ Arguments

IVISION_Handle handle; /* algorithm instance handle */
IVISION_inBufs *inBufs; /* algorithm input buffer descriptor */
IVISION_outBufs *outBufs; /* algorithm output buffer descriptor */
IVISION_InArgs *inargs /* algorithm runtime input arguments */
IVISION_OutArgs *outargs /* algorithm runtime output arguments */

‖ Return Value

IALG_EOK; /* status indicating success */
IALG_EFAIL; /* status indicating failure */

‖ Description

This function does the basic Scene Obstruction Detection. The first argument to `process()` is a handle to an algorithm instance.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `IVISION_inBufs, IVISION_outBufs` data structure for details).

The fourth argument is a pointer to the `IVISION_InArgs` data structure that defines the run time input arguments for an algorithm instance object.

The last argument is a pointer to the `IVISION_OutArgs` data structure that defines the run time output arguments for an algorithm instance object.

> Note:
> If you are using extended data structures, the fourth and fifth arguments must be pointers to the extended `InArgs` and `OutArgs` data structures respectively. Also, ensure that the `size` Field is set to the size of the extended data structure. Depending on the value set for the `size` Field, the algorithm uses either basic or extended parameters.

‖ Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- `process()` can only be called after a successful return from `algInit()`.
- If algorithm uses DMA resources, `process()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.
- Buffer descriptor for input and output buffers must be valid.
- Input buffers must have valid input data.
- `inBufs->numBufs` indicates the total number of input
- Buffers supplied for input frame, and conditionally, the algorithms meta data buffer.
- `inArgs` must not be NULL and must point to a valid `IVISION_InArgs` structure.
- `outArgs` must not be NULL and must point to a valid `IVISION_OutArgs` structure.
- `inBufs` must not be NULL and must point to a valid `IVISION_inBufs` structure.
- `inBufs->bufDesc[0].bufs` must not be NULL, and must point to a valid buffer of data that is at least `inBufs->bufDesc[0].bufSize` bytes in length.
- `outBufs` must not be NULL and must point to a valid `IVISION_outBufs` structure.
- `outBufs->buf[0]` must not be NULL and must point to a valid buffer of data that is at least `outBufs->bufSizes[0]` bytes in length.
- The buffers in `inBuf` and `outBuf` are physically contiguous and owned by the calling application.

‖ Postconditions

The following conditions are true immediately after returning from this function.

- If the process operation is successful, the return value from this operation is equal to `IALG_EOK;` otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- The algorithm must not modify the contents of `inArgs`.

- The algorithm must not modify the contents of `inBufs`, with the exception of `inBufs.bufDesc[].accessMask`. That is, the data and buffers pointed to by these parameters must be treated as read-only.
- The algorithm must appropriately set/clear the `bufDesc[].accessMask` Field in `inBufs` to indicate the mode in which each of the buffers in `inBufs` were read. For example, if the algorithm only read from `inBufs.bufDesc[0].buf` using the algorithm processor, it couSCENE_OBSTRUCTION_DETECT utilize `#SETACCESSMODE_READ` to update the appropriate `accessMask` Fields. The application may utilize these returned values to manage cache.
- The buffers in `inBufs` are owned by the calling application.

‖ Example

See test application file, SCENE_OBSTRUCTION_DETECT_tb.c available in the \test\src sub-directory.

‖ See Also

algInit(), algDeactivate(), control()

> Note:
> The algorithm cannot be preempted by any other algorithm instance. That is, you cannot perform task switching while filtering of a particular frame is in progress. Pre-emption can happen only at frame boundaries and after `algDeactivate()` is called.

‖ Name

`algDeactivate()` – save all persistent data to non-scratch memory

‖ Synopsis

Void algDeactivate(IALG_Handle handle);

‖ Arguments

IALG_Handle handle; /* algorithm instance handle */

‖ Return Value

Void

‖ Description

`algDeactivate()` saves any persistent information to non-scratch buffers using the persistent memory that is part of the algorithm's instance object.
The first (and only) argument to `algDeactivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be saved prior to next cycle of `algActivate()` and processing.
For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

`algActivate()`

5.8 Termination API

Termination API is used to terminate the algorithm instance and free up the memory space that it uses.

‖ Name

`algFree()` – determine the addresses of all memory buffers used by the algorithm

‖ Synopsis

XDAS_Int32 algFree(IALG_Handle handle, IALG_MemRec memTab[]);

‖ Arguments

IALG_Handle handle; /* handle to the algorithm instance */
IALG_MemRec memTab[]; /* output array of memory records */

‖ Return Value

XDAS_Int32; /* Number of buffers used by the algorithm */

‖ Description

`algFree()` determines the addresses of all memory buffers used by the algorithm. The primary aim of doing so is to free up these memory regions after closing an instance of the algorithm.

`The first argument to algFree() is a handle to the algorithm instance.`

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers previously allocated for the algorithm instance.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

`algAlloc()`

This chapter provides answers to few frequently asked questions related to using this algorithm.

6 Frequently Asked Questions

6.1 Code Build and Execution

Question	Answer

6.1.1 Algorithm Related

Question	Answer