Stereo Vision Post Processing using TI's TMS320C66x DSP

User Guide

May 2017

IMPORTANT NOTICE

1.1 About This Manual

1 Read This First

This document describes how to install and work with Texas Instruments' (TI) StereoVision Postprocessing 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 StereoVision Postprocessing 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 StereoVision Postprocessing 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 StereoVision Postprocessing 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	StereoVision Postprocessing 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 (Stereovision post processing Module on TMS320C66x DSP) and version number. The version number of the Stereovision post processing Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, StereoVision Postprocessing 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 Post Processing

The stereovision postprocessing module can be used to enhance the disparity map produced by a stereovision algorithm running on DSP or a hardware coprocessor such as EVE. The algorithm bundles edge detection and line detection modules. It assumes that the inputs are an 8-bits disparity map, an optional 8-bits auxiliary disparity map, three 16-bits

cost maps and the input intensity frame for left camera or right camera. The post-processing steps carried out are:

- Sub-pixel interpolation of the disparity map to increase the resolution of the disparity values. It is enabled with any disparity step values. We can use up to 4 bits for fractional disparity. During this stage, 16-bit curvature is also generated for every pixel. It will be used for 5-bit confidence calculation.
- Texture computation: texture value of each pixel is computed, which will be used by the next step for the disparity map cleaning and 5-bit confidence calculation.
- Disparity map cleaning. Custom thresholds are applied to reject pixels that have low texture or low curvature. Also at the same time if the optional 8-bit auxiliary disparity map is provided, left-right check is performed to get rid of pixels that have inconsistent disparity values between two search directions: left-to-right and right-to-left. Finally a segment-based cleaning process is applied, in which each image row is divided into segments of different length. A new segment starts whenever the absolute difference between the current pixel's disparity value and the previous pixel's disparity value exceeds by an amount given by the parameter `maxDispDissimilarity`. Then confidence value of each pixel within a segment is compared with the parameter `minConfidenceThreshold`. The algorithm counts how many exceeds this threshold `minConfidenceThreshold`. If that number is greater than the parameter `minConfidentNSegment` then the entire segment is labeled as valid, otherwise the entire segment is labeled as invalid.
  The invalid pixels's disparity value is reset to 0 so they appear black when the disparity map is displayed.

  While cleaning, 16-bit disparity value is generated for every pixel by packing 7-bit integer-pel disparity, 4-bit sub-pel disparity and 5-bit confidence value. It is stored in the same buffer for 16-bit curvature by overwriting it. Confidence value, conf5bits, is calculated based on texture, curvature, matching cost and L-R check as follows.

  ```
  conf5bits = conf_curvature + conf_texture + conf_cost + conf_lrd
  ```

  - `conf_curvature` is a confidence value by curvature, which is calculated by `(curvature >> 3) / maxCurvature`, where `curvature` is the curvature of a pixel and `maxCurvature` is the maxmum curvature among all pixels in a picture. Its range is 0 ~ 7.
  - `conf_texture` is a confidence value by `texture`, which is the amount of texture at a pixel. It is calculated by `(texture >> 5)`. Its range is 0 ~ 7.
  - `conf_cost` is the confidence by matching cost. Let's say `cost` is the matching cost of a pixel and `maxCost` is the maximum matching cost among all pixels in a picture. `conf_cost` is calculated by `(7 – cost / (maxCost >> 3))`. Its range is 0 ~ 7.
  - `conf_lrd` is the confidence by the difference between left-to-right disparity and right-to-left disparity, which is denoted as `lr_diff`. For each pixel, `inBoundary` is defined to specifiy whether a pixel is in image boundary, for which the auxiliary disparity cannot be calculated. `inBoundary` is 0 or 1. Then `conf_lrd` is calculated by `((inBoundary << 3) – (MIN(lr_diff,32) >> 2))`. Its range is 0 ~ 8.

  Therefore, `conf5bits` can have value between 0 ~ 29. The higher the value, the higher confidence we have for a disparity. Please note that left-right check should be enabled for use cases that produce 16-bit disparity map.

- Hole filling: if desired post-processing can fill the holes produced by the disparity map cleaning step. Note: hole filling can introduce unwelcome artifacts.

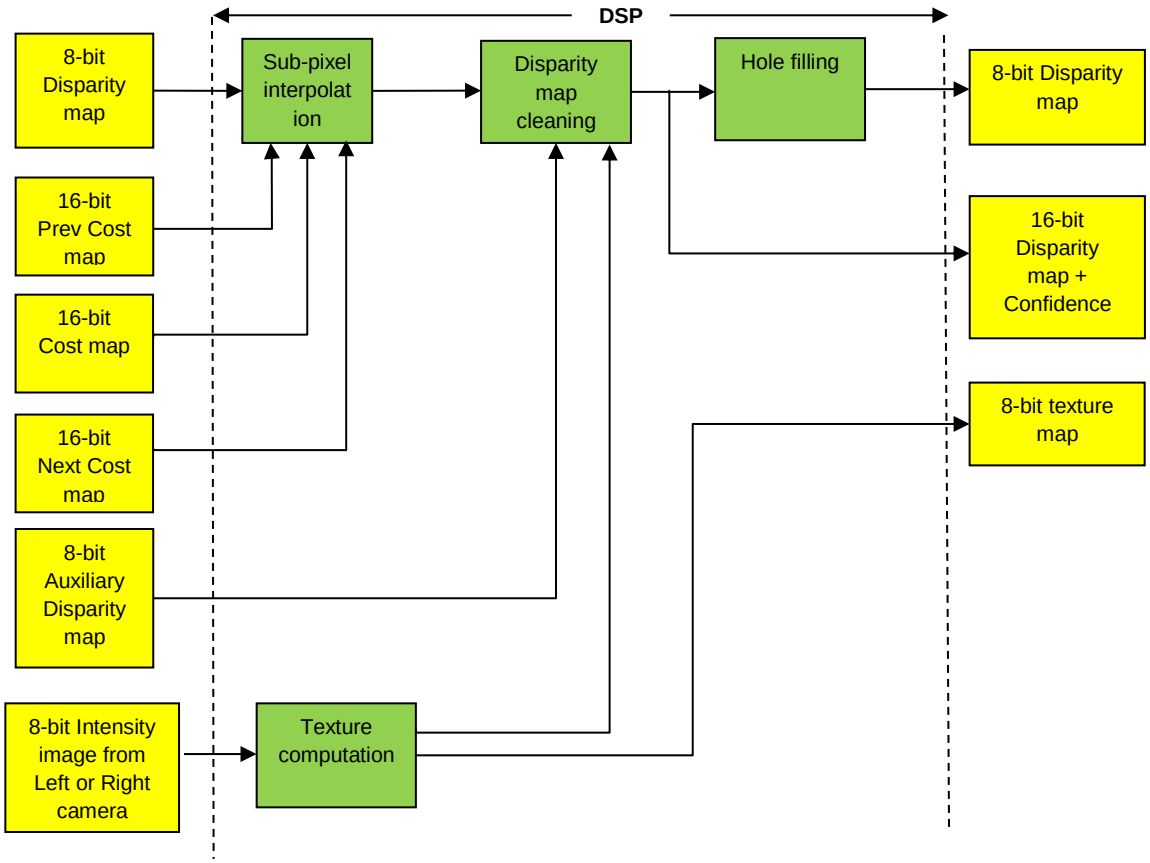


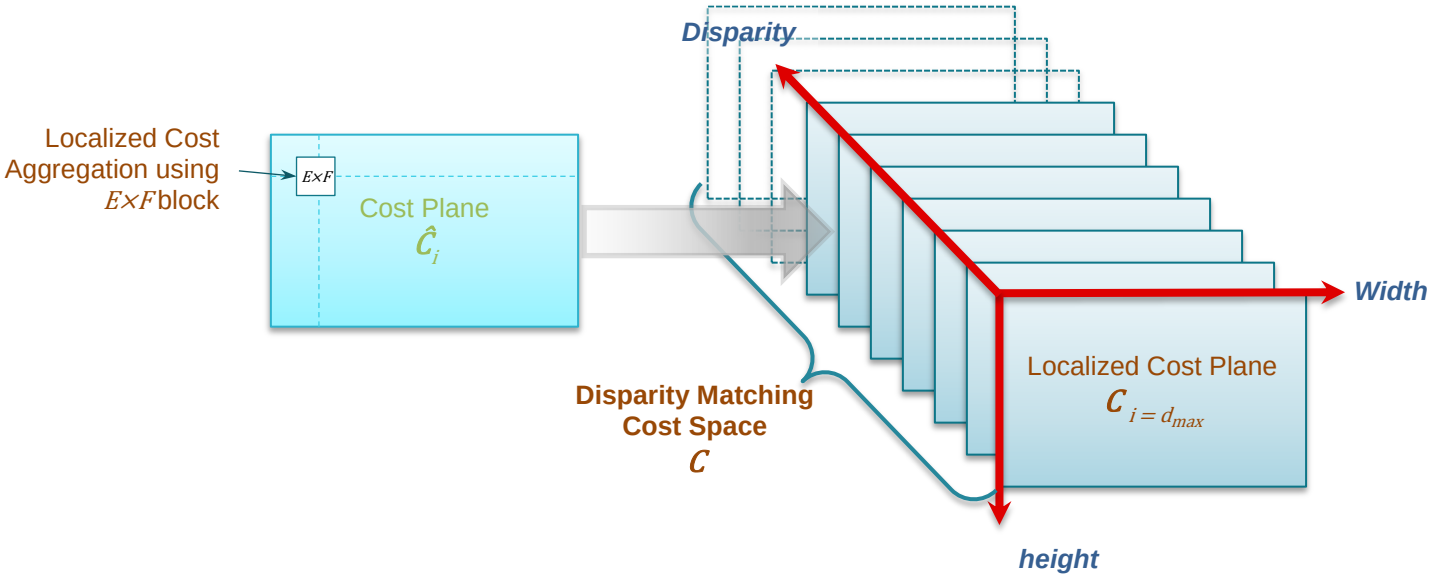
Figure 1 Fundamental blocks of StereoVision Postprocessing

The input 8-bits disparity map was produced by the stereovision algorithm which was executed prior to this post-processing algorithm. It is obtained either by conducting a left-to-right or right-to-left correspondence search. The post-processing algorithm needs to know which direction was used as well as other parameters used to configure the stereovision module, such as census transform kernel dimensions, local block matching window dimensions, etc. These parameters are conveyed to the post-processing module through custom parameter structures.

The 8-bit auxiliary disparity map is used for the purpose of comparing the disparity outputs produced by both directions: left-to-right and right-to-left correspondence search. This process called *left-right consistency check,* aims at identifying pixels of which disparity value is likely to be incorrect due to occlusions. One consequence is that the auxiliary disparity map must have been obtained using the opposite direction that was used to configure the stereovision module to obtain the original disparity map, described in the previous paragraph. For instance, when the original 8-bits disparity map is produced using

left to right direction then the auxiliary disparity map must be produced in the opposite direction: right to left. To save computation cycles, the auxiliary disparity map could have been produced from a lower resolution image, either downsampled horizontally or vertically. This information is communicated to the post-processing algorithm through the proper parameters `auxDisparityHorzDsFactor` and `auxDisparityVertDsFactor` in the structure `STEREOVISION_TI_PostProcOptions`.

The three cost maps are used by the sub-pixel interpolation step: previous cost map, cost map and next cost maps. To understand what each one represents, we show that the cost space is a 3-D space *C*.



For each pixel, the best disparity is the one that minimizes the cost and its value is within the range [0, d_{max}]. The cost map contains each pixel's minimum cost, which coincides with the best disparity of value d_{best}. The previous cost map corresponds to the cost of the previous disparity d_{best}-1 and the next cost map corresponds to the cost of the next disparity d_{best}+1. For instance if for a certain pixel, the stereo vision module found that its disparity is d_{best}=50 then the previous cost would be equal to the cost at disparity 49. Likewise the next cost would be equal to the cost at disparity 51.

2.3 Supported Services and Features

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

- Supports 8 bit luma only input image, 8-bit disparity maps and 16-bit disparity maps.
- Supports image resolution of up to 720xH where H can be arbitrarily large.
- Support for user control thresholds.
- Independent of any operating system.

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

3 Installation Overview

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 on TDA2x platform. The algorithm shall work on any future TDA platforms hosting C66x DSP.

3.1.2 Software

The following are the software requirements for the stand alone functioning of the StereoVision Postprocessing 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.ST.C66X.01.00 . Folder structure of this top level directory is shown in below figure.

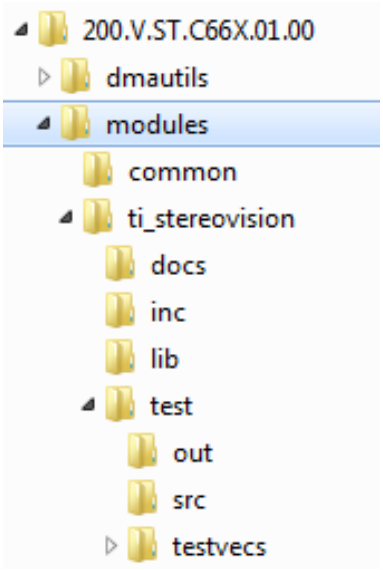


Figure 2 Component Directory Structure

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_stereovision	StereoVision postprocessing module for C66x DSP
\modules \ti_stereovision \docs	User guide and Datasheet for StereoVision postprocessing module
\modules \ti_stereovision \inc	Contains istereovision_ti.h interface file
\modules \ti_stereovision \lib	Contains StereoVision postprocessing algorithm library
\modules\ti_ stereovision \test	Contains standalone test application source files
\modules\ti_ stereovision \test\out	Contains test application .out executable
\modules\ti_ stereovision \test\src	Contains test application source files

Sub-Directory	Description
\modules\ti_ stereovision \test\testvecs	Contains config, input, output, reference test vectors
\modules\ti_ stereovision \test\testvecs\config	Contain config file to set various parameters exposed by StereoVision postprocessing module
\modules\ti_ stereovision \test\testvecs\input	Contains sample input image .pgm file

3.3 Building Sample Test Application

This StereoVision postprocessing 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 StereoVision Postprocessing library has been validated with XDAIS tools containing IVISION interface version. Other required components are provided below.

- Code Generation tools
- EDMA3 LLD
- DMA utility library
- For host building, Visual Studio
- For host building, c6xsim

For version numbers of each component that is compatible with this stereo-vision release, please refer to the release notes.

3.3.1 Installing XDAIS tools (XDAIS)

XDAIS 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>

3.3.2 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>

3.3.3 DMA Utility Library

Install DMA utility library for DSP from this link

https://cdds.ext.ti.com/ematrix/common/emxNavigator.jsp?objectId=28670.42872.62652.37497

The DMA utility library is also available in processor SDK – Vision package. After installing DMA Utility Library, Set a system environment variable named "DMAUTILS_PATH" pointing to <install_directory>\dmautils.

3.3.4 Installing EDMA3 LLD

Install EDMA3 LLD Tools from the link
http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/psp/edma3_lld/index.html
After installing the EDMA3 LLD, set the environment variable named "EDMA3_LLD_ROOT" to the installed directory like
<install_directory>\<edma3lld_directory>

3.3.5 Installing c6x sim

To build the example in host emulation mode for c6x DSP, you will need to install the c6xsim which is available at
http://processors.wiki.ti.com/index.php/Run_Intrinsics_Code_Anywhere
Install this package into the top-level common folder.

3.3.6 Building the Test Application Executable through GMAKE for target execution

The sample test application that accompanies StereoVision Postprocessing 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
2) Verify that you have installed XDAIS tools
3) Verify that you have installed EDMA3 LLD
4) Verify that appropriate environment variables have been set as discussed in this above sections.
5) Build the sample test application project by gmake
   a) modules\ti_stereovision\test> gmake clean
   b) modules\ti_stereovision\test> gmake all
6) The above step creates an executable file, test_stereovision_algo.out in the modules\ti_stereovision\test\out sub-directory.
7) Open CCS with TDA2x platform selected configuration file. Select Target > Load Program on C66x DSP, browse to the modules\ti_stereovision\test\out sub-directory, select the executable created in step 5, and load it into Code Composer Studio in preparation for execution.
8) Select Target > Run on C66x DSP window to execute the sample test application.
9) The test application takes the static image, cost and disparity maps input arrays defined in files \test\src\inputFrame.c, inputCost.c, inputDisparity.c, runs the module.

10) The reference disparity maps output array is defined in \test\src\outputDisparityPostProcEqual0.c (smoothing filter strength set to 0) or \test\src\outputDisparityPostProcEqual3.c (smoothing filter strength set to 3) and is used to verify that the stereoVision postprocessing is functioning as expected.
11) On successful completion, the test application writes pass or fail on the console window.
12) There are actually two tests that can selectively be performed depending on the value of the #ifdef at line 398 of \test\src\stereovision_tb.c . By default the first test is performed, which sets the smoothing filter strength to 0.
13) Note that there is also a mode in which the test program, instead of doing validation, simply reads the input files specified in stereovision.cfg and writes out the output in the output files. To enable this file I/O mode, comment out line #43:

```
#if (!HOST_EMULATION)
#define _STATIC_INPUT // comment this line out
                      // to enable file I/O.
#endif
```

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

The sample test application that accompanies StereoVision Postprocessing 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
2) Verify that you have installed XDAIS
3) Verify that you have installed EDMA3 LLD
4) Verify that you have installed DMA utility library.
5) Verify that you have installed the c6xsim.
6) Verify that appropriate environment variables have been set as discussed in this above sections.
7) Verify that the Visual Studio path *"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin"* has been included in the system path.
8) Build the sample test application project by gmake
   a) modules\ti_stereovision > *gmake TARGET_BUILD=release TARGET_PLATFORM=PC clean*
   b) modules\ti_stereovision > *gmake TARGET_BUILD=release TARGET_PLATFORM=PC all*
9) The above step creates an executable file, *test_stereovision_algo.out.exe* in the *modules\ti_stereovision\test\out* sub-directory.
10) Open the command window in directory *modules\ti_stereovision\test\out* and type *test_stereovision_algo.out.exe* to run the text.

11) The test application will parse the configuration file located in *modules\ti_stereovision\test\testvecs\config\stereovision.cfg* for location of the input data, output data and algorithm's parameters.
12) During execution, the data is read from file system, processed and results are written to files.

3.4 Configuration File

This algorithm is shipped along with:

- Algorithm configuration file (stereovision.cfg) – specifies the configuration parameters used by the test application to configure the Algorithm.
- This configuration file is only used when the test application is compiled with file I/O enabled.

3.4.1 Test Application Configuration File

The algorithm configuration file, stereovision.cfg contains the configuration parameters required for the algorithm. The stereovision.cfg file is available in the \test\testvecs\config sub-directory.
A sample stereovision.cfg file is as shown. More details on the meaning of each parameter follows.

```
#--------------------------------------------------------------------#
# Common Parameters                                                  #
#--------------------------------------------------------------------#
numTestCases     = 1   # Number of test cases to be run
inImageFileName       = "../testvecs/input/img.pgm" # Image corresponding to the left
                                                     # or right camera
inCostFileName      = "../testvecs/output/cost.bin" # Cost map generated by the
                                                     # stereovision module
# It is assumed that the three cost maps: previous cost maps, minimum cost map, next
cost maps are concatenated in the order mentioned above.

inDisparityFileName   = "../testvecs/input/disparity.pgm" # Disparity map generated
                                                          # by the stereovision module

inAuxDisparityFileName   = "../testvecs/input/auxDisparity.pgm" # Aux disparity map
                                                  #generated by the stereovision module
outFileName      = "../testvecs/output/disparity.pgm" # Output disparity map
inputBitDepth    = 8 # Bit depth of the input image, only 8 is supported now
maxImageWidth    = 640 # Maximum width of the input image
maxImageHeight   = 480 # Maximum height of the output image
startX0        = 136 # X Offset of the point in the input image that coincides with
                     # the upper left corner of the disparity map
startY0        = 9   # Y offset of the point in the input image that coincides with
                     # the upper left corner of the disparity map
censusWinWidth0  = 9 # width of the census kernel, only 9 is supported for now
censusWinHeight0 = 9 # width of the census kernel, only 9 is supported for now
censusWinHorzStep0 = 2 # horizontal step within the census kernel
censusWinVertStep0 = 2 # vertical step within the census kernel
costSupportWinWidth0 = 11 # width of the cost window used for disparity local block
```

```
matching
costSupportWinHeight0 = 11 # height of the cost window used for disparity local block
                          #matching
numDisparities0 = 128    # maximum number of disparities
disparityStep0   = 1     # disparity step
searchDir0       = 0      # 0: left frame to right frame, 1: right frame to left frame
smoothingStrength0 = 0     # Only 0 supported
disparityNumFracBits0 = 0 # max value = 4
disparityMinThreshold0 = 0 # Threshold for minimum disparity value
disparityMaxThreshold0 = 127 # Threshold for maximum disparity value
costMaxThreshold0 = 90 # Normalized threshold 0-100 for maximum cost value
minConfidenceThreshold0 = 98 # Normalized threshold 0-100 for minimum confidence value
holeFillingStrength0 = 0 # Only 0 supported
textureLumaLoThresh0 = 0 # Normalized lower bound threshold 0-100 for luma
textureLumaHiThresh0 = 100 # Normalized higher bound threshold 0-100 for luma
textureThreshold0 = 95 # Normalized threshold 0-100 for texture
lrMaxDiffThreshold0 = 3 # Maximum disparity difference between left-right
maxDispDissimilarity0 = 4 # Maximum disparity difference between neighbor pixels
minConfidentNSegment0 = 6 # Length of support segment used in disparity map cleaning
```

If you specify additional fields in the stereovision.cfg file, ensure that you modify the test application appropriately to handle these fields.

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 StereoVision Postprocessing component.

4 Sample Usage

4.1 Overview of the Test Application

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

Test Application	XDAIS – IVISION interface	DSP Apps
Algorithm instance creation and initialization	algNumAlloc() algAlloc() algInit()	
Process Call	control() process() control()	
Algorithm instance deletion	algNumAlloc() algFree()	

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, stereoVision postprocessing 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 stereovision.cfg and reads the various configuration parameters required for the algorithm.

For more details on the configuration files, see Section 3.4.

Sets the STEREOVISION_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.

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 input image to be 8 bit data, luma only image, disparity maps in 8 bits, cost map in 16-bits. The details about the input image regarding, maximum image width, maximum image height, ROI width, ROI height, startX and startY etc comes from the config file.

4.6.3 Output Buffer Format

The stereoVision postprocessing module outputs 3 buffers:

- 8-bit disparity map which is the result of post-processing the input disparity map.
- 16-bit disparity map which consists of 7-bit integer-pel disparity, 4-bit sub-pel disparity and 5-bit confidence. This output buffer stores curvature value in sub-pel interpolation step. While cleaning disparity map, this buffer is overwritten by 16-bit disparity map.
- 8-bit texture map: which measures the amount of texture around each pixel. Usually pixels that have low texture also have low confidence value. The opposite however is not always true.

This chapter provides a detailed description of the data structures and interfaces functions used by StereoVision Postprocessing.

5 API Reference

Disclaimer: although the naming convention uses the prefix STEREOVISION, 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 stereovision module possibly running on a different core. The prefix STEREOVISION 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	X (horizontal direction offset)
`Y`	`XDAS_Int32`	Input	Y (vertical direction offset)

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	Width of the buffer (in bytes), This field can be viewed as pitch while processing a ROI in the buffer
height	XDAS_UInt32	Input	Height of the buffer (in lines)
frameROI	IVISION_Rect	Input	Region of the interest for the current frame to be processed in the buffer. Dimensions need to be a multiple of internal block dimenstions. Refer application specific details for block dimensions supported for the algorithm. This needs to be filled even if bit-0 of IVISION_InArgs::subFrameInfo is set to 1
subFrameROI	IVISION_Rect	Input	Region of the interest for the current sub frame to be processed in the buffer. Dimensions need to be a multiple of internal block dimenstions. Refer application specific details for block dimensions supported for the application. This needs to be filled only if bit-0 of IVISION_InArgs::subFrameInfo is set to 1
freeSubFrameROI	IVISION_Rect	Input	This ROI is portion of subFrameROI that can be freed after current slice process call. This field will be filled by the algorithm at end of each slice processing for all the input buffers (for all the output buffers this field needs to be ignored). This will be filled only if bit-0 of IVISION_InArgs::subFrameInfois set to 1
planeType	XDAS_Int32	Input	Content of the buffer - for example Y component of NV12
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
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

Field	Data Type	Input/ Output	Description
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 StereoVision Postprocessing Enumerations

This section includes the following StereoVision Postprocessing specific enumerations:

- STEREOVISION_ErrorType
- STEREOVISION_InBufOrder
- STEREOVISION_OutBufOrder
- STEREOVISION_TI_CostMethod
- STEREOVISION_TI_SearchDir
- STEREOVISION_TI_ProcessingMode
- STEREOVISION_TI_DisparityOptions
- STEREOVISION_TI_SmoothingStrength
- STEREOVISION_TI_PostProcOptions
-

5.1.8.1 STEREOVISION_ErrorType

‖ Description

This enumeration defines all the error codes returned by the StereoVision postprocessing algorithm.

‖ Fields

Field	Data Type	Input/ Output	Description
STEREOVISION_ERRORTYPE_INDALID_IMAGE_DIMS	enum	Output	The maximum image width and height dimensions exceed the maximum allowable by the algorithm.
STEREOVISION_TI_ERRORTYPE_INVALID_NUM_INPUTS	enum	Output	Incorrect number of input buffers passed to the function.
STEREOVISION_TI_ERRORTYPE_CREATE_FAIL	enum	Output	Algorithm creation failed

5.1.8.2 STEREOVISION_InBufOrder

‖ Description

This enumeration defines the purpose of the input buffers.

‖ Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
STEREOVISION_BUFDESC_IN_IMAGE	enum	Input	This buffer descriptor provides the actual image 8-bits data required by the algorithm. This image either comes from left or right camera and is the same image used to produce the disparity map. If the disparity map was produced by using left to right correspondence search then the image MUST be the left camera, otherwise it MUST be the right camera.
STEREOVISION_TI_BUFDESC_IN_DISPARITY	enum	Input	8-bits disparity map produced by the stereovision module executed before this post-processing module.
STEREOVISION_TI_BUFDESC_IN_AUX_DISPARITY	enum	Input	8-bits auxiliary disparity map used in left-right check.
STEREOVISION_TI_BUFDESC_IN_PREV_COST	enum	Input	16-bits previous cost inputs discussed earlier. Used by the sub-pixel interpolation step.
STEREOVISION_TI_BUFDESC_IN_COST	enum	Input	16-bits cost inputs discussed earlier. Used by the sub-pixel interpolation step.
STEREOVISION_TI_BUFDESC_IN_NEXT_COST	enum	Input	16-bits next cost inputs discussed earlier. Used by the sub-pixel interpolation step.

5.1.8.3 STEREOVISION_OutBufOrder

‖ Description

This enumeration defines the purpose of the output buffers.

‖ Fields

Field	Data Type	Input/ Output	Description
STEREOVISION_TI_BUFDESC_OUT_DISPARITY	enum	Output	This buffer is filled up by StereoVision postprocessing algorithm with the post-processed disparity values.
STEREOVISION_TI_BUFDESC_OUT_CONFIDENCE	enum	Output	This buffer is filled up with the 16-bits curvature and disparity values.

Field	Data Type	Input/ Output	Description
STEREOVISION_TI_BUFDESC_OUT_TEXTURE	enum	Output	This buffer is filled up with the 8-bits texture values.

5.1.8.4 STEREOVISION_TI_CostMethod

Description

This enumeration contains information to indicate the cost method employed by the preceding stereovision module to produce the disparity map. This information is required by the post-processing algorithm to produce a correct output.

Fields

Field	Data Type	Input/ Output	Description
STEREOVISION_TI_SAD	enum	Input	Sum of absolute difference used.
STEREOVISION_TI_HAM_DIST	enum	Input	Hamming distance used.

5.1.8.5 STEREOVISION_TI_SearchDir

Description

This enumeration contains information for the search direction employed by the preceding stereovision module to produce the input disparity map. This information is required by the post-processing algorithm to produce a correct output.

Fields

Field	Data Type	Input/ Output	Description
STEREOVISION_TI_LEFT_TO_RIGHT	enum	Input	Disparity search is done by fixing a pixel in left image and then searching for the minimum cost pixel in the right image, along the same epipolar line.
STEREOVISION_TI_RIGHT_TO_LEFT	enum	Input	Disparity search is done by fixing a pixel in right image and then searching for the minimum cost pixel in the left image, along the same epipolar line.

5.1.8.6 STEREOVISION_TI_ProcessingMode

Description

This enumeration contains information of what processing is performed by this module. Currently only STEREOVISION_TI_POSTPROCESS_ONLY is supported.

Fields

Field	Data Type	Input/ Output	Description
STEREOVISION_TI_ALL	enum	Input	The entire stereovision algorithm is executed: disparity followed by post-processing
STEREOVISION_TI_DISPARITY_ONLY	enum	Input	Only disparity is executed
STEREOVISION_TI_POSTPROCESS_ONLY	enum	Input	Only prost-processing is executed. this is the only option supported at the moment.

5.1.8.7 STEREOVISION_TI_SmoothingStrength

‖ Description

This enumeration contains information on the post processing smoothing filter strength carried out by the module. Recommended setting is STEREOVISION_TI_POST_STRENGTH_NONE.

‖ Fields

Field	Data Type	Input/ Output	Description
STEREOVISION_TI_POST_STRENGTH_NONE	enum	Input	
STEREOVISION_TI_POST_STRENGTH_LO	enum	Input	
STEREOVISION_TI_POST_STRENGTH_MED	enum	Input	
STEREOVISION_TI_POST_STRENGTH_HI	enum	Input	

5.1.9 StereoVision Postprocessing Data Structures

This section includes the following StereoVision Postprocessing specific extended data structures:

- STEREOVISION_TI_CreateParams
- STEREOVISION_TI_DisparityOptions
- STEREOVISION_TI_InArgs
- STEREOVISION_TI_OutArgs
- STEREOVISION_TI_output

5.1.9.1 STEREOVISION_TI_CreateParams

‖ Description

This structure defines the create-time input arguments for StereoVision postprocessing Algorithm instance object.

‖ Fields

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	See IVISION_Params data structure for details
maxImageRoiWidth	uint16_t	Input	Max input width of image. It should not exceed STEREOVISION_TI_MAXWIDTH
maxImageRoiHeight	uint16_t	Input	Max input height of image. It should not exceed STEREOVISION_TI_MAXHEIGHT
inputBitDepth	Uint8_t	Input	Bit depth of the input image, currently 8-bits is supported.
processingMode	STEREOVISION_TI_ProcessingMode	input	See description of enum type STEREOVISION_TI_ProcessingMode, which defines the processing mode. Currently only STEREOVISION_TI_POSTPROCESS_ONLY is supported.
disparityOptions	STEREOVISION_TI_DisparityOptions	Input	See description of enum type STEREOVISION_TI_DisparityOptions, which defines the disparity options
edma3RmLldHandle	void*	Input	Pointer to the EDMA3 LLD resource manager handle

5.1.9.2 STEREOVISION_TI_DisparityOptions

‖ Description

This structure defines parameters that were used by the stereovision module to produce the input disparity map and associated cost maps. These parameters need to be explicitly communicated to this post-processing module as it cannot infer them from the disparity map neither the cost map.

‖ Fields

Field	Data Type	Input/ Output	Description
censusWinWidth	uint8_t	Input	census transform kernel width
censusWinHeight	uint8_t	Input	census transform kernel height
costSupportWinWidth	uint8_t	Input	Width of the support window, that defines the neighbourhood in which SAD or hamming-distance based cost calculations are performed.
costSupportWinHeight	uint8_t	Input	Height of the support window, that defines the neighbourhood in which SAD or hamming-distance based cost calculations are performed.
minDisparity	uint8_t	Input	Minimum disparity for which the cost is calculated. For each pixel, the disparity corresponding to the minimum cost is returned. The disparity range is [minDisparity, maxDisparity].
maxDisparity	uint8_t	Input	Maximum disparity for which the cost is calculated. For each pixel, the disparity corresponding to the minimum cost is returned. The disparity range is [minDisparity, maxDisparity].
disparityStep	uint8_t	input	Disparity step allows to down-sample the number of disparities for which the cost is calculated, resulting in faster computation in detriment to precision.
costMethod	uint8_t	Input	See description of enum type STEREOVISION_TI_CostMethod, which defines the method employed to calculate each disparity's cost. Currently only STEREOVISION_TI_HAM_DIST is supported
searchDir	uint8_t	Input	See description of enum type STEREOVISION_TI_SearchDir, which defines the direction(s) of the disparity search.

5.1.9.3 STEREOVISION_TI_InArgs

‖ Description

This structure contains all the parameters which are given as input to StereoVision postprocessing algorithm at frame level.

‖ Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	See `IVISION_InArgs` data structure for details.
`postProcOptions`	`STEREOVISION_TI_PostProcOptions`	Input	See description of enum type `STEREOVISION_TI_PostProcOptions`, which defines the post processing options

5.1.9.4 STEREOVISION_TI_PostProcOptions

‖ Description

This structure defines options associated to post-processing for StereoVision postprocessing Algorithm instance object.
Most of the parameters are threshold values that are normalized within the range [0 100] for easy setting. In general a threshold value set to 100 will keep all the disparity values untouched whereas a threshold value set to 0 will label all the disparity values as invalid and will produce an all-zero disparity map.

‖ Fields

Field	Data Type	Input/ Output	Description
`disparityNumFracBits`	uint8_t	Input	Number of fractional bits added to the disparity values by the sub-pixel interpolation. It should be 0 ~ 4.
`textureLumaHiThresh`	uint8_t	Input	Normalized higher bound threshold 0-100 for luma. If image is 8-bits then 0 maps to 0 and 100 maps to 255. This value acts as a saturation bound: any pixel whose luma value is above this threshold will have its texture value set 0.

Field	Data Type	Input/ Output	Description
			So very bright regions automatically get assigned a 0 value texture.
`textureLumaLoThresh`	uint8_t	Input	Normalized lower bound threshold 0-100 for luma. If image is 8-bits then 0 maps to 0 and 100 maps to 255. This value acts as a saturation bound: any pixel whose luma value is below this the threshold will have its texture value set 0. So very dark regions automatically get assign a 0 value texture.
`disparityMinThreshold`	uint8_t	Input	Minimum disparity threshold, Pixels whose disparity is below this threshold are marked invalid.
`disparityMaxThreshold`	uint8_t	Input	Maximum disparity threshold, pixels whose disparity is greater than this threshold are marked invalid.
`costMaxThreshold`	`uint8_t`	input	Normalized threshold value in the range [0 100]. Pixels whose disparity cost are above this threshold are marked invalid.
`minConfidenceThreshold`	`uint8_t`	Input	Normalized threshold value in the range [0 100]. Pixels whose confidence value cost is below (100 – `minConfidenceThreshold)` are marked invalid. In other word, if `minConfidenceThreshold` = 100, all pixels are valid and if `minConfidenceThreshold` = 0, all pixels are invalid.
`textureThreshold`	`uint8_t`	Input	Normalized threshold value in the range [0 100]. Pixels whose texture value is cost is below (100 – `textureThreshold)` are marked invalid. In other word, if `textureThreshold` = 100, all pixels are valid and if `textureThreshold` = 0, all pixels are invalid.
`holeFillingStrength`	`uint8_t`	Input	Normalized value in the range [0 100] expressing the

Field	Data Type	Input/ Output	Description
			aggressiveness of the hole filling algorithm.
lrMaxDiffThreshold	uint8_t	Input	The maximum disparity difference between the main disparity map and the auxiliary disparity map, which can be tolerated. A large value means a large tolerance. This is used for left right check. A value of 255 disables the check.
maxDispDissimilarity	Uint8_t	Input	The maximum disparity difference between a pixel and its neighbor before starting a new segment. Pixels within a segment have their confidence values examined in order to declare the segment as valid or invalid.
minConfidentNSegment	uint8_t	Input	Number of pixels within a segment whose confidence value must exceed minConfidenceThreshold to keep the entire segment as valid.
auxDisparityHorzDsFactor	uint8_t	Input	horizontal downsample factor of the auxiliary disparity map
auxDisparityVertDsFactor	uint8_t	Input	vertical downsample factor of the auxiliary disparity map
smoothingStrength	uint8_t	Input	Smoothing filter strength

5.1.9.5 STEREOVISION_TI_OutArgs

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	Output	See IVISION_OutArgs data structure for details.
maxDisparityVal	int16_t	Output	maximum disparity value produced by the sub-pixel interpolation step

Field	Data Type	Input/ Output	Description
minDisparityVal	int16_t	Output	minimum disparity value produced by the sub-pixel interpolation step
maxCostVal	uint16_t	Output	maximum cost value in the input cost map
maxCurveVal	int16_t	Output	maximum confidence (curve) value produced by the sub-pixel interpolation step
rsvd1	int32_t	Output	reserved

5.2 Recommended and Supported Values of Parameters

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

- STEREOVISION_TI_CreateParams
- STEREOVISION_TI_PostProcOptions
- STEREOVISION_TI_DisparityOptions

Table 4: Supported Values for STEREOVISION_TI_CreateParams

Field	Supported Value
maxImageWidth	Value up to 720 is supported. This parameter is only used for memory allocation and does not impact the performance. It is recommended to set this to the image width to be processed in order to reduce memory footprint
maxImageHeight	Any value is supported. This parameter is only used for memory allocation and does not impact the performance. It is recommended to set this to the image height to be processed in order to reduce memory footprint
inputBitDepth	Currently only 8 is supported.

Table 5: Supported Values for STEREOVISION_TI_PostProcOptions

Field	Supported Value
disparityNumFracBits	Between 0 ~ 4.
textureLumaHiThresh	Value up to 100 supported. Generally set it to a value close to 100.

Field	Supported Value
textureLumaLoThresh	Value up to 100 supported. Generally set it to a value close to 0.
disparityMinThreshold	0 is the recommended value unless it is known that the scene does not contain anything below a certain disparity value: all objects are within a certain distance of the camera, that there is no "far" object.
disparityMaxThreshold	Depend on the scene content. The stereovision algorithm was configured to find disparities up to a certain value, which was input as parameter numDisparities of STEREOVISION_TI_DisparityOptions. Consequently disparityMaxThreshold should usually be set to this value unless it is known in advance that all objects in the scene are beyond a certain distance from the camera.
costMaxThreshold	Any value between 0 – 100 but typically a value around 90 is used.
minConfidenceThreshold	Any value between 0 – 100 but typically a value around 98 is used.
textureThreshold	Any value between 0 – 100 but mainly depend on the texture content. Usually a high value is used.
holeFillingStrength	Should be set to 0 in this release.
lrMaxDiffThreshold	Recommended value is 1 to 5.
maxDispDissimilarity	Recommended value is 1 to 5
minConfidentNSegment	Recommended value is 1 to 5. The wider the image is, the larger this value should be.
auxDisparityHorzDsFactor	Must be set to 1 in this release.
auxDisparityVertDsFactor	Must be set to 2 in this release.
smoothingStrength	Must be set to 0 in this release.

Table 6 Default and Supported Values for STEREOVISION_TI_DisparityOptions

Field	Supported Value

Field	Supported Value
censusWinWidth	Set to 9 in this release.
censusWinHeight	Set to 9 in this release.
costSupportWinWidth	Typical values are 11, 13, 15, 17, 19
costSupportWinHeight	Typical values are 11, 13, 15, 17, 19
numDisparities	32, 64, 128
disparityStep	1, 2, 4
costMethod	STEREOVISION_TI_HAM_DIST
searchDir	STEREOVISION_TI_LEFT_TO_RIGHT or STEREOVISION_TI_RIGHT_TO_LEFT

5.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used by StereoVision postprocessing. 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 couSTEREOVISION 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 StereoVision Postprocessing 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, STEREOVISION_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 stereoVision postprocessing. 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 couSTEREOVISION 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, STEREOVISION_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()`