Lane Detection using TI's TMS320C66x DSP

User Guide

May 2017

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive & Transportation	www.ti.com/automotive
Communications & Telecom	www.ti.com/communications
Computers & Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energyapps
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics & Defense	www.ti.com/space-avionics-defense
Video & Imaging	www.ti.com/video
TI E2E Community	e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

1 READ THIS FIRST 6

1.1 About This Manual 6

1.2 Intended Audience 6

1.3 How to Use This Manual 6

1.4 Related Documentation from Texas Instruments 6

1.5 Abbreviations 6

1.6 Text Conventions 7

1.7 Product Support 7

1.8 Trademarks 7

2 INTRODUCTION 8

2.1 Overview of XDAIS 8

2.1.1 XDAIS Overview 8

2.2 Overview of Lane Detection and departure warning 9

2.3 Supported Services and Features 9

3 INSTALLATION OVERVIEW 10

3.1 System Requirements 10

3.1.1 Hardware 10

3.1.2 Software 10

3.2 Installing the Component 10

3.2.1 Installing the compressed archive 10

3.3 Building Sample Test Application 12

3.3.1 Installing XDAIS tools (XDAIS) 12

3.3.2 Installing Code Generation Tools 12

3.3.3 DMA Utility Library 12

3.3.4 Building the Test Application Executable through GMAKE 12

3.4 Configuration File 13

3.4.1 Test Application Configuration File 13

3.5 Host emulation build for source package 15

3.5.1 Installing Visual Studio 15

3.5.2 Building source in host emulation 15

3.5.3 Running host emulation executable 16

3.6 Uninstalling the Component 16

4 SAMPLE USAGE 17

4.1 Overview of the Test Application 17

4.2 Parameter Setup 17

4.3 Algorithm Instance Creation and Initialization 18

4.4 Process Call 18

4.5 Algorithm Instance Deletion 19

4.6 Frame Buffer Management 20

4.6.1 Input and Output Frame Buffer 20
4.6.2 Input Buffer Format 20
4.6.3 Output Buffer Format 20

5 API REFERENCE 21

5.1.1 IVISION_Params 21
5.1.2 IVISION_Point 21
5.1.3 IVISION_BufPlanes 21
5.1.4 IVISION_BufDesc 22
5.1.5 IVISION_BufDescList 23
5.1.6 IVISION_InArgs 23
5.1.7 IVISION_OutArgs 24
5.1.8 Lane Detection Enumerations 24
5.1.9 Lane Detection Data Structures 28

5.2 Recommended and Supported Values of Parameters 31

5.3 Interface Functions 34

5.4 Creation APIs 35

5.5 Initialization API 36

5.6 Control API 37

5.7 Data Processing API 39

5.8 Termination API 42

6 APPENDIX 43

7 FREQUENTLY ASKED QUESTIONS 45

7.1 Code Build and Execution **45**
7.1.1 Algorithm Related 45

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) Lane 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 Lane 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 Lane 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 Lane 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
CIF	Common Intermediate Format
DMA	Direct Memory Access
DMAN3	DMA Manager
DSP	Digital Signal Processing
EVM	Evaluation Module
IRES	Interface for Resources
LANDET	Lane 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 (Lane Detection Module on TMS320C66x DSP) and version number. The version number of the Lane Detection Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, Lane 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 Lane Detection 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 Lane Detection and departure warning

The lane detection module can be used to detect lane markings on the road. The algorithm bundles an edge detection and line detection modules. It assumes that the input is an 8-bit luma image and ROI co-ordinates of the image pertaining to the road are provided. The edge detection method used is Canny and Hough transform is used to detect lines from these edge points. Also, lane departure warning information is found using the rho and theta parameters of the detected lanes.

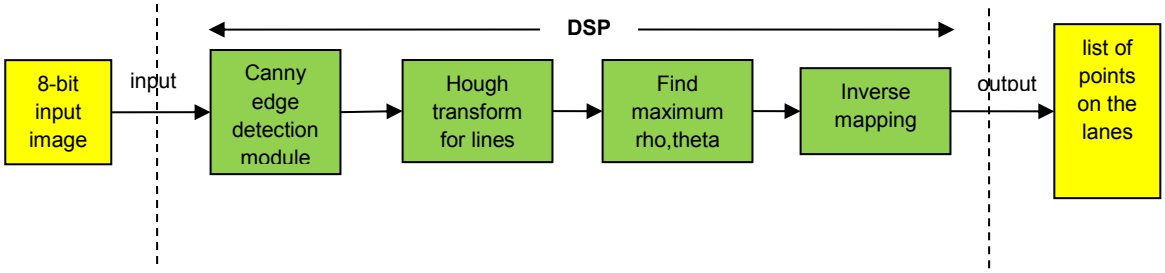

Figure 1 Fundamental blocks of Lane Detection

2.3 Supported Services and Features

This user guide accompanies TI's implementation of Lane Detection Algorithm on the TI's C66x DSP.

This version of the Lane Detection has the following supported features of the standard:

- Supports 8 bit luma only input image.
- Supports Lane detection using Canny edge and Hough transform.
- Supports image resolution of upto 1920x1080.
- Support for user control performance and quality knobs.
- Independent of any operating system.

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing Lane 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 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 Lane Detection module:

- **Development Environment:** This project is developed using TI's Code Generation Tool 7.4.2. Other required tools used in development are mentioned in section 3.3
- The project are built using g-make (GNU Make version 3.81). 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.LD.C66X.00.02 . Folder structure of this top level directory is shown in below figure.

Figure 2 Component Directory Structure

Table 2 Component Directories

Sub-Directory	Description
\common	Common files for building different DSP modules
\makerules	Make rule files
\modules	Top level folder containing different DSP app modules
\modules \ti_lane_detection	Lane detection module for C66x DSP
\modules \ti_lane_detection \docs	User guide and Datasheet for Lane detection module
\modules \ti_lane_detection \inc	Contains ild_ti.h interface file
\modules \ti_lane_detection \lib	Contains Lane detection algorithm library
\modules \ti_lane_detection \test	Contains standalone test application source files
\modules \ti_lane_detection \test\out	Contains test application .out executable
\modules \ti_lane_detection \test\src	Contains test application source files
\modules \ti_lane_detection \test\testvecs	Contains config, input, output, reference test vectors
\modules \ti_lane_detection \test\testvecs\config	Contain config file to set various parameters exposed by Lane detection module
\modules \ti_lane_detection \test\testvecs\input	Contains sample input image .yuv file
\modules \ti_lane_detection \test\testvecs\output	Contains output .csv file with a list of lane points detected.
\modules \ti_lane_detection \test\testvecs\reference	Contains reference .csv file with a list of lane points detected.

3.3 Building Sample Test Application

This Lane detection library has been accompanied by a sample test application. To run the sample test application XDAIS tools are required.

This version of the Lane Detection library has been validated with XDAIS tools containing IVISION interface version. Please refer to the Release Notes for other required components (for test application building) version details.

3.3.1 Installing XDAIS tools (XDAIS)

XDAIS version 7.24.00.04 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 version 7.4.2 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 the 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 Building the Test Application Executable through GMAKE

The sample test application that accompanies Lane Detection module will run in TI's Code Composer Studio development environment. 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 you have installed DMA Utility Library version mentioned

4) 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_lane_detection\test> gmake clean

   b) modules\ti_lane_detection\test> gmake all

5) The above step creates an executable file, test_lane_detection_algo.out in the modules\ti_lane_detection\test\out sub-directory.

6) Open CCS with TDA2x platform selected configuration file. Select Target > Load Program on C66x DSP, browse to the modules\ti_lane_detection\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) Sample test application takes the input files stored in the \test\testvecs\input sub-directory, runs the module.

9) The reference files stored in the \test\testvecs\reference sub-directory can be used to verify that the lane detection is functioning as expected.

10) On successful completion, the test application writes the information regarding the detected lane points in the \test\testvecs\output sub-directory. If ENABLE_TRACES macro is enabled, then it also writes the output frame with the lane points marked in \test\testvecs\output sub-directory.

11) User should compare with the reference provided in \test\testvecs\reference directory. Both the content should be same to conclude successful execution.

3.4 Configuration File

This algorithm is shipped along with:

- Algorithm configuration file (ld.cfg) – specifies the configuration parameters used by the test application to configure the Algorithm.

3.4.1 Test Application Configuration File

The algorithm configuration file, ld.cfg contains the configuration parameters required for the algorithm. The ld.cfg file is available in the \test\testvecs\config sub-directory.

A sample ld.cfg file is as shown.

```
#------------------------------------------------------------------#
# Common Parameters                                                #
#------------------------------------------------------------------#

numTestCases     = 1   # Number of test cases to be run
inFileName       = "../testvecs/input/img"
outFileName      = "../testvecs/output/ldoutput"
maxImageWidth    = 640 # Maximum width of the input image.
maxImageHeight   = 360 # Maximum height of the output image.
actualImageWidth = 640 # Actual width of the input image.
actualImageHeight= 360 # Actual height of the output image.
maxFrames        = 10  # Maximum number of input frames.
roiWidth         = 582 # ROI width of the image to be processed
roiHeight        = 200 # ROI height of the image to be processed
startX           = 29  # Start X co-ordinate of the ROI
startY           = 120 # Start Y co-ordinate of the ROI

#------------------------------------------------------------------#
# LD Parameters                                                    #
#------------------------------------------------------------------#
cannyHiThresh        = 30     # High threshold for Canny edge detection
cannyLoThresh        = 20     # Low threshold for Canny edge detection
houghNmsThresh       = 20     # Threshold for NMS on the hough space
maxRho               = 624    # Maximum Rho, this should be equal to
sqrt((roiWidth)^2+(roiHeight)^2)

startThetaLeft       = 0     # Start of theta range for left lane
endThetaLeft         = 150     # End of theta range for left lane
startThetaRight      = 0    # Start of theta range for right lane
endThetaRight        = 150    # End of theta range for right lane
thetaStepSize        = 1      # Theta increment step within the range

#------------------------------------------------------------------#
# Tracking and Lane departure Parameters                           #
#------------------------------------------------------------------#
trackMethod = 1   # Flag to indicate tracking method to use, Kalman filter supported
enableWarning = 1 # Flag to enable lane departure warning
numHoughMaximasDet = 6 # Number of maximas for each lane in the Hough space to be
detected. Maximum supported is 10

numHoughMaximasTrack = 3 # Number of maximas for each lane in the hough space to be
tracked. Maximum supported is 5. Also, it has to be <= numHoughMaximasDet

departThetaLeftMin = 125 # Min theta for left lane, below which contributes to left
lane switch

departThetaLeftMax = 135 # Max theta for left lane, above which contributes to right
lane switch

departRhoLeftMin = 350 # Min rho for left lane, below which contributes to right lane
switch
```

```
departRhoLeftMax = 373 # Max rho for left lane, above which contributes to left lane
switch

departThetaRightMin = 20 # Min theta for right lane, below which contributes to left
lane switch

departThetaRightMax = 30 # Max theta for right lane, above which contributes to right
lane switch

departRhoRightMin = 382 # Min rho for right lane, below which contributes to left lane
switch

departRhoRightMax = 385 # Max rho for right lane, above which contributes to right
lane switch

visType = 1 # Marks the trapezoidal lane region. If set to 0, then it marks the lane
points detected
```

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

3.5 Host emulation build for source package

The source release of Lane Detection module can be built in host emulation mode. This option speeds up development and validation time by running the platform code on x86/x64 PC.

3.5.1 Installing Visual Studio

Building host emulation for Lane Detection module requires Microsoft Visual Studio 11.0 (2012) which can be downloaded from below link.

http://www.microsoft.com/en-in/download/details.aspx?id=34673

3.5.2 Building source in host emulation

After installing the required components, navigate to Circular Light Recognition install path and run `vcvarsall.bat` to setup the required environment variables

```
{ld_install_path} > {…\Microsoft Visual Studio
11.0\VC\vcvarsall.bat}
```

Once the environment variables are setup build the Lane Detect source in host emulation mode

```
{ld_install_path} > gmake all TARGET_BUILD=debug TARGET_PLATFORM=PC
```

This will build the host emulation executable under the path

```
{ld_install_path}\test\out\ test_lane_detection_algo.out.exe
```

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 in the common folder

3.5.3 Running host emulation executable

Launch Microsoft Visual Studio 11.0 and open file
`test_lane_detection_algo.out.exe`

This will load the host emulation program which can be used for development and validation purpose.

3.6 Uninstalling the Component

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

4 Sample Usage

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

4.1 Overview of the Test Application

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

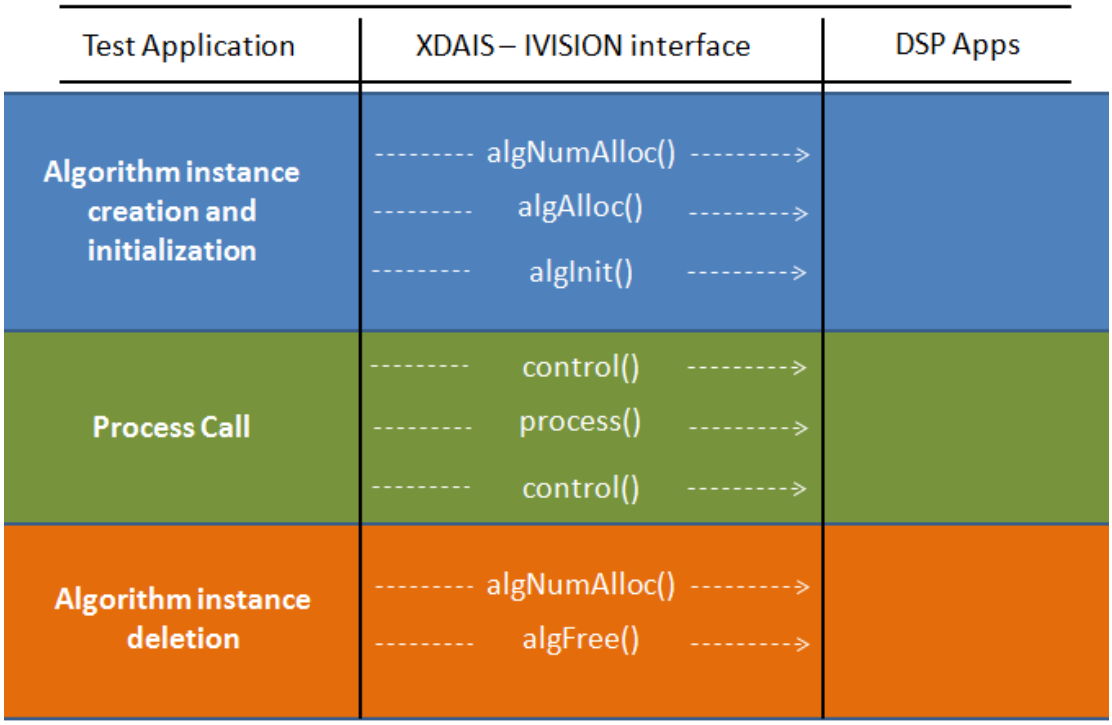

Table 3 Test Application Sample Implementation

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, lane 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 ld.cfg and reads the various configuration parameters required for the algorithm.

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

Sets the `LD_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 detected lane point co-ordinates 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 detect lane in the provided image. 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.

`control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the eight available control commands.

`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. 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 lane detection module outputs the XY co-ordinates of lane points detected via `LD_TI_output` structure defined in ild_ti.h interface. The co-ordinates are comprising of 16-bit Y and 16-bit X. The structure provides the list of left lane points detected followed by right lane points detected. The number of left lane points and right lane points detected are provided as part of the `LD_TI_OutArgs`. Please refer to section 5.1.9.4 for more details.

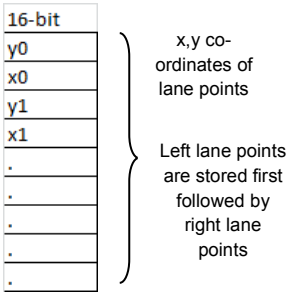

Figure 3 Output Buffer format

5 API Reference

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

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

Field	Data Type	Input/ Output	Description
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 points in the polygon
bufPlanes[IVISION_MAX_NUM_PLANES]	IVISION_BufPlanes	Input	Description of each plane

Field	Data Type	Input/ Output	Description
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
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 Lane Detection Enumerations

This section includes the following Lane Detection specific enumerations:

- `ILD_ErrorType`
- `ILD_InBufOrder`
- `ILD_OutBufOrder`
- `LD_TI_TRACKMETHOD`

- LD_TI_WARNING_INFO
- LD_TI_LANE_INFO

5.1.8.1 ILD_ErrorType

‖ Description

This enumeration defines all the error codes returned by the Lane detection algorithm.

‖ Fields

Field	Data Type	Input/ Output	Description
ILD_ERRORTYPE_MAXNUMEDGES_EXCEEDED	enum	Output	The number of edges detected is more than the maximum supported given by LD_TI_MAXNUMEDGES
ILD_ERRORTYPE_MAXNUMTHETA_EXCEEDED	enum	Output	The maximum Theta value provided by user is more than what is supported, which is given by LD_TI_MAXNUMTHETA
ILD_ERRORTYPE_MAXNUMRHO_EXCEEDED	enum	Output	The maximum Rho value provided by user is more than what is supported, which is given by LD_TI_MAXNUMRHO
ILD_ERRORTYPE_INVALID_IMAGE_DIMS	enum	Output	Image dimensions are beyond what is supported, given by LD_TI_MAXWIDTH and LD_TI_MAXHEIGHT
ILD_ERRORTYPE_INVALID_ROI_DIMS	enum	Output	ROI dimensions are greater than image dimensions
ILD_ERRORTYPE_INVALID_THETA_RANGE	enum	Output	Theta range is invalid. Could be because start theta is less than end theta
ILD_ERRORTYPE_INVALID_RHO_RANGE	enum	Output	The Rho range specified by user is invalid, example if departRhoMax/Min is greater than LD_TI_MAXNUMRHO
ILD_ERRORTYPE_INVALID_CANNY_THRESHOLD	enum	Output	Canny threshold is invalid because high threshold is less than low threshold
ILD_ERRORTYPE_LD_CREATE_FAIL	enum	Output	Failure while creating Lane detection algorithm
ILD_ERRORTYPE_INVALID_KF_PARAMS	enum	Output	Failure to indicate invalid Kalman Filter params

Field	Data Type	Input/ Output	Description
ILD_ERRORTYPE_INVALID_MAXHOUGHMAXIMASDET	enum	Output	Error to indicate that the user defined parameter numHoughMaximasDet is greater than LD_MAX_DET_HOUGH_MAXIMAS or less than LD_MIN_DET_HOUGH_MAXIMAS
ILD_ERRORTYPE_INVALID_MAXHOUGHMAXIMASTRACK	enum	Output	Error to indicate that the user defined parameter numHoughMaximasTrack is greater than LD_MAX_TRACK_HOUGH_MAXIMAS or less than LD_MIN_TRACK_HOUGH_MAXIMAS
ILD_ERRORTYPE_INVALID_MAXHOUGHMAXIMAS	enum	Output	Error to indicate that the user defined parameter numHoughMaximasTrack is greater than numHoughMaximasDet

5.1.8.2 ILD_InBufOrder

‖ Description

This enumeration defines the purpose of the input buffers.

‖ Fields

Field	Data Type	Input/ Output	Description
LD_BUFDESC_IN_IMAGEBUFFER	enum	Input	This buffer descriptor provides the actual image data required by the algorithm. For lane detection algorithm, only luma data is used.

5.1.8.3 ILD_OutBufOrder

‖ Description

This enumeration defines the purpose of the output buffers.

‖ Fields

Field	Data Type	Input/ Output	Description
LD_BUFDESC_OUT_XY_LIST	enum	Output	This buffer is filled up by Lane detection algorithm with the XY co-ordinates of the detected lanes. The XY co-ordinates are of 16-bit X and 16-bit Y.

Field	Data Type	Input/ Output	Description
			The buffer will contain (X,Y) locations of lane points belonging to ROI-1 which is left ROI (ideally left lane) followed by points belonging to ROI-2 which is right ROI (ideally right lane). The number of lane points detected in each ROI is output as part of the outArgs (numLeftLanePoints and numRightLanePoints).

5.1.8.4 LD_TI_TRACKMETHOD

‖ Description

This enumeration contains information to indicate the tracking method employed. In order to enable tracking, the inArgs – trackingMethod should be set to LD_TI_TRACK_KF.

‖ Fields

Field	Data Type	Input/ Output	Description
LD_TI_TRACK_DISABLE	enum	Input	Tracking disabled
LD_TI_TRACK_KF	enum	Input	Kalman Filter (2x4 model) based tracking enabled.

5.1.8.5 LD_TI_WARNING_INFO

‖ Description

This enumeration contains information to enable lane departure warning.

‖ Fields

Field	Data Type	Input/ Output	Description
LD_TI_WARNING_DISABLE	enum	Input	Lane departure warning is disabled
LD_TI_WARNING_ENABLE	enum	Input	Lane departure warning is enabled

5.1.8.6 LD_TI_LANE_INFO

‖ Description

This enumeration contains information of lane departure when the lane departure warning is enabled. In order to enable warning, the inArgs – warningMethod should be set to LD_TI_WARNING_ENABLE. In case of departure, the outArgs – infoFlag will contain the departure information such as LD_TI_RIGHT_LANE_CROSS or LD_TI_LEFT_LANE_CROSS. In case warning is not enabled, the default value LD_TI_NO_LANE_CROSS is returned as part of the outArgs – infoFlag and departure information is not recorded.

‖ Fields

Field	Data Type	Input/ Output	Description
LD_TI_NO_LANE_CROSS	enum	Output	No lane has been crossed
LD_TI_RIGHT_LANE_CROSS	enum	Output	Right lane has been crossed
LD_TI_LEFT_LANE_CROSS	enum	Output	Left lane has been crossed

5.1.9 Lane Detection Data Structures

This section includes the following Lane Detection specific extended data structures:

- LD_TI_CreateParams
- LD_TI_InArgs
- LD_TI_OutArgs
- LD_TI_output

5.1.9.1 LD_TI_CreateParams

‖ Description

This structure defines the create-time input arguments for Lane detection Algorithm instance object.

‖ Fields

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	See IVISION_Params data structure for details
maxImageWidth	uint16_t	Input	Max input width of image. It should not exceed LD_TI_MAXWIDTH
maxImageHeight	uint16_t	Input	Max input height of image. It should not exceed LD_TI_MAXHEIGHT
maxRho	uint16_t	Input	Max input Rho for Hough space. Traditionally, this should be set to $\sqrt{(roiWidth^2 + roiHeight^2)}$ for a rho resolution of 2. This should not exceed LD_TI_MAXNUMRHO
maxTheta	uint16_t	input	Max input Theta for Hough space. It should not exceed

Field	Data Type	Input/ Output	Description
			LD_TI_MAXNUMTHETA
*edma3RmLldHandle	void	Input	Pointer to the EDMA3 LLD resource manager handle

5.1.9.2 LD_TI_InArgs

‖ Description

This structure contains all the parameters which are given as input to Lane detection algorithm at frame level. The theta information is provided in terms of bin and not degrees. The LD application has pre-computed values for sin and cos in Q8 format for range -80 to +79 degrees (theta increments of 1) and provided as tables sinLUT[LD_TI_MAXNUMTHETA] and cosLUT[LD_TI_MAXNUMTHETA].

‖ Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	IVISION_ InArgs	Input	See IVISION_InArgs data structure for details.
cannyHighThresh	uint16_t	Input	High threshold for Canny edge detection
cannyLowThresh	uint16_t	Input	Lower threshold for Canny edge detection
houghNmsThresh	uint16_t	Input	Threshold for NMS operation on the Hough space
startThetaLeft	uint16_t	Input	Start angle bin to be considered for left lane. Hough transform for lines operates in the range startThetaLeft to endThetaLeft
endThetaLeft	uint16_t	Input	End angle bin to be considered for left lane. Hough transform for lines operates in the range startThetaleft to endThetaLeft
startThetaRight	uint16_t	Input	Start angle bin to be considered for right lane. Hough transform for lines operates in the range startThetaRight to endThetaRight
endThetaRight	uint16_t	Input	End angle bin to be considered for right lane. Hough transform for lines operates in the range startThetaRight to endThetaRight
thetaStepSize	uint8_t	Input	Angle increment step to be considered for computing Hough transform. Typically set to 1. Only integer values supported

Field	Data Type	Input/ Output	Description
numHoughMaximasDet	uint8_t	Input	Number of maximas to be detected in the Hough space for each ROI. It should not exceed *LD_MAX_DET_HOUGH_MAXIMAS* or be less than *LD_MIN_DET_HOUGH_MAXIMAS*
numHoughMaximasTrack	uint8_t	Input	Number of Hough space maximas to be tracked. It should not exceed *LD_MAX_TRACK_HOUGH_MAXIMAS* or be less than *LD_MIN_TRACK_HOUGH_MAXIMAS*. In case tracking is disabled, *numHoughMaximasTrack* is used for inverse Hough mapping. So, it can be set equal to *numHoughMaximasDet*, but it should be less than equal to *LD_MAX_TRACK_HOUGH_MAXIMAS*
trackingMethod	uint8_t	Input	Parameter to indicate the tracking method employed. For supported values, refer to *LD_TI_TRACKMETHOD* enumeration
warningMethod	uint8_t	Input	Parameter to indicate if lane departure warning is enabled or not. For supported values, refer to *LD_TI_WARNING_INFO* enumeration
departThetaLeftMin	uint16_t	Input	Minimum angle bin of left lane to be considered for lane departure. For more details, refer to appendix
departThetaLeftMax	uint16_t	Input	Maximum angle bin of left lane to be considered for lane departure. For more details, refer to appendix
departRhoLeftMin	uint16_t	Input	Minimum rho of left lane to be considered for lane departure. For more details, refer to appendix
departRhoLeftMax	uint16_t	Input	Maximum rho of left lane to be considered for lane departure. For more details, refer to appendix
departThetaRightMin	uint16_t	Input	Minimum angle bin of right lane to be considered for lane departure. For more details, refer to appendix
departThetaRightMax	uint16_t	Input	Maximum angle bin of right lane to be considered for lane departure. For more details, refer to appendix
departRhoRightMin	uint16_t	Input	Minimum rho of right lane to be considered for lane departure. For more details, refer to appendix
departRhoRightMax	uint16_t	Input	Maximum rho of right lane to be considered for lane departure. For more details, refer to appendix

5.1.9.3 LD_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.
`numLeftLanePoints`	`uint16_t`	Output	Number of points on the left lane, detected in the image
`numRightLanePoints`	`uint16_t`	Output	Number of points on the right lane, detected in the image
`infoFlag`	`uint32_t`	Output	Information flag to indicate lane departure condition. See *LD_TI_LANE_INFO* enumeration for possible values.

5.1.9.4 LD_TI_output

‖ Description

This is the output structure given out by lane detection module. It contains the co-ordinates of the points belonging to lanes detected in the image. The number of points detected is governed by numLanePoints which is part of the LD_TI_OutArgs.

‖ Fields

Field	Data Type	Input/ Output	Description
`Y`	`uint16_t`	Output	Y co-ordinate location of the detected lane points
`X`	`uint16_t`	Output	X co-ordinate location of the detected lane points

5.2 Recommended and Supported Values of Parameters

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

- LD_TI_CreateParams
- LD_TI_inArgs

Table 4: Supported Values for LD_TI_CreateParams

Field	Supported Value
maxImageWidth	Value upto 1920 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	Value upto 1080 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
maxRho	Value upto 2203 is supported, in order to support an image resolution of 1920x1080. This parameter will impact the memory footprint as well as performance. This parameter should be set to $sqrt((roiWidth)^2+(roiHeight)^2)$ for optimal performance
maxTheta	Value upto 160 is supported. This parameter is used for memory allocation. The cos and sin tables have been pre-computed for 160

Table 5 Default and Supported Values for LD_TI_InArgs

Field	Supported Value
cannyHighThresh	16-bit value can be used and acts as higher threshold on magnitude during the Canny Edge detection stage. This value should be higher than cannyLowThresh
cannyLowThresh	16-bit value can be used and acts as lower threshold on magnitude during the Canny edge detection stage. This value should be lower than cannyHighThresh
houghNmsThresh	16-bit value can be used and acts as a threshold in order to find Hough space maximas
startThetaLeft	16-bit value between 0-159 is supported. This is the start theta for Hough transform for left lane. It should be lower than endThetaLeft
endThetaLeft	16-bit value between 0-159 is supported. This is the end theta for Hough transform for left lane. It should be greater than startThetaLeft

Field	Supported Value
startThetaRight	16-bit value between 0-159 is supported. This is the start theta for Hough transform for right lane. It should be lower than endThetaRight
endThetaRight	16-bit value between 0-159 is supported. This is the end theta for Hough transform for right lane. It should be greater than startThetaRight
thetaStepSize	16-bit value between 0-159 supported. This is used as theta increment step size during the Hough transform for line. It should be set to integer values and ideally set to 1
numHoughMaximasDet	8-bit value in the range 1-10 supported. This indicates the number of maximas in the Hough space to be detected. It has to be greater than or equal to numHoughMaximasTrack
numHoughMaximasTrack	8-bit value in the range 1-5 supported. This indicates the number of maximas in the Hough space to be tracked. It should be less than or equal to numHoughMaximasDet. In case tracking is disabled, this can be set equal to numHoughMaximasDet. However, it has to be less than or equal to 5
trackingMethod	0 – tracking disabled 1 – Kalman filter based tracking enabled
warningMethod	0 – lane departure warning disabled 1 – lane departure warning enabled
departThetaLeftMin	16-bit value between 0-159 supported. It has to be less than departThetaLeftMax. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix
departThetaLeftMax	16-bit value between 0-159 supported. It has to be greater than departThetaLeftMin. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix
departRhoLeftMin	16-bit value between 0-2203 supported. It has to be less than departRhoLeftMax. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix
departRhoLeftMax	16-bit value between 0-2203 supported. It has to be greater than departRhoLeftMin. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix

Field	Supported Value
departThetaRightMin	16-bit value between 0-159 supported. It has to be less than departThetaRightMax. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix
departThetaRightMax	16-bit value between 0-159 supported. It has to be greater than departThetaRightMin. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix
departRhoRightMin	16-bit value between 0-2203 supported. It has to be less than departRhoRightMax. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix
departRhoRightMax	16-bit value between 0-2203 supported. It has to be greater than departRhoLeftMin. This parameter is used as threshold to detect lane departure situation. For more details, refer to Appendix

5.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used by Lane 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 could 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 Lane 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, ld_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 lane 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 could 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, ld_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()`

6 Appendix

This chapter provides brief details about the lane departure warning parameters used in the LD algorithm. There are four parameters for each lane that the user has to provide. They are:

1. departThetaLeftMax and departThetaRightMax
2. departThetaLeftMin and departThetaRightMin
3. departRhoLeftMax and departRhoRightMax
4. departRhoLeftMin and departRhoRightMin

The Hough transform principle converts a line in the spacial domain (x,y) into polar co-ordinates given by (rho, theta), theta being the angle of the line with respect to the horizontal and rho being the perpendicular distance of the line from the origin as shown in **Figure *4***. In **Figure *4***, (Θ_1, ρ_1) corresponds to angle and distance of the left lane and (Θ_2, ρ_2) corresponds to angle and distance of the right lane.

During left lane switch as shown in **Figure *5***, the angle and distance of right lane decreases. However, the angle of the left lane decreases while its distance increases. Hence the parameters, departThetaRightMin, departThetaLeftMin, departRhoRightMin and departRhoLeftMax are used as threshold to determine the left lane switch condition.

During right lane switch as shown in **Figure *6***, the angle and distance of right lane increases. However, the angle of the left lane increases while its distance decreases. Hence the parameters, departThetaRightMax, departThetaLeftMax, departRhoRightMax and departRhoLeftMin are used as threshold to determine the right lane switch condition.

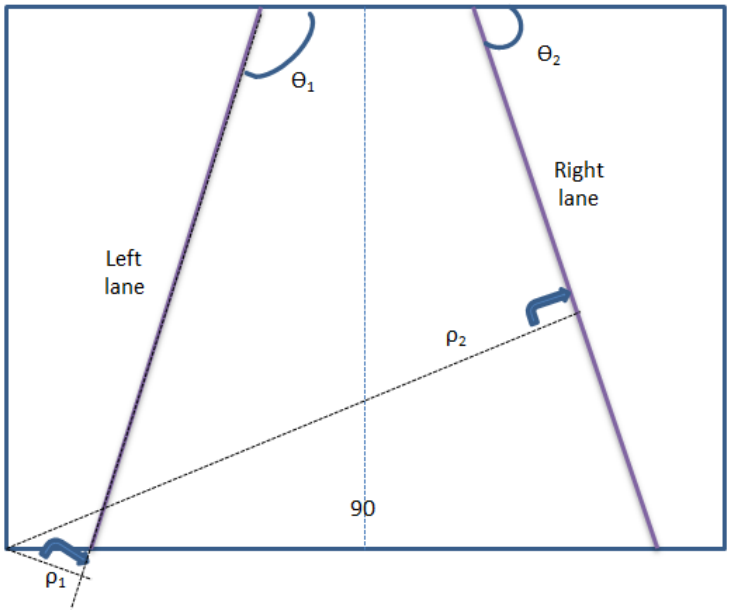

Figure 4: Normal condition – No lane cross

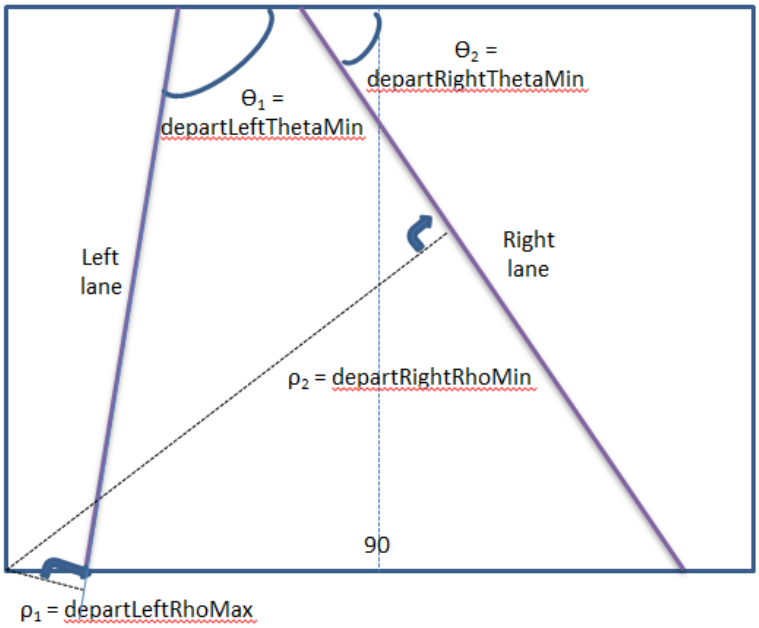

Figure 5: Left lane switch

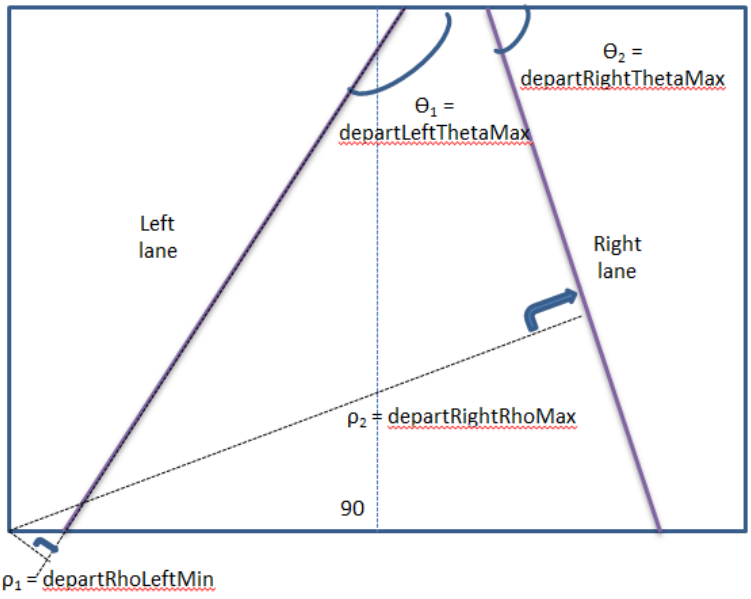

Figure 6: Right lane switch

7 Frequently Asked Questions

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

7.1 Code Build and Execution

Question	Answer

7.1.1 Algorithm Related

Question	Answer