

Document Type:

Application Note - VisionIP DOF HIL/SIL

Project Name:

Vision IP Dense Optical Flow

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>).

Responsible Department	Global ADAS Solution Group
Approved by	
Checked by	
Prepared by	A. Schulz

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavours to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Table of Contents

Table of Contents	3
List of Figures.....	5
List of Tables	6
1 Introduction	7
1.1 Purpose and Scope	7
1.2 Prerequisites	7
1.3 Terms, Abbreviations	7
2 Dense Optical Flow	8
2.1 Overview	8
2.2 Applications.....	8
2.2.1 Movement of Objects.....	8
2.2.2 Distance of Objects	11
2.3 Process	14
3 Application	17
3.1 Content	17
3.2 Integration	17
3.3 Usage	17
3.4 Operation	17
3.4.1 Flow definition.....	18
3.4.2 Configuration preparation	21
3.4.3 Configuration execution	21
3.4.4 Post processing	22
3.4.4.1 Raw data.....	22
3.4.4.2 Colorized single component flow fields	22
3.4.4.3 Level maps.....	24
3.4.4.4 Confirmation maps	24
3.5 Re-Use / Extension	24
3.5.1 Additional configuration parameters	24
3.5.2 Dedicated configuration	24
3.5.3 Partially fixed configuration	25
3.5.4 Fully fixed configuration	25
3.5.5 Streamed ring buffer processing.....	25
4 Open Issues	28
5 Appendix	29
5.1 Configuration Details.....	29
5.1.1 Separation	29
5.1.2 Scatter / Gather DMA usage.....	30
5.1.3 Flow field fusion.....	30
5.1.4 Functions and Parameters.....	31
5.1.4.1 Input images	31
5.1.4.2 Input flow field	32
5.1.4.3 Output flow field	33
5.1.4.4 Search ranges.....	34
5.1.4.5 Image pre filtering	35

5.1.4.6 Flow vector shifting 35

6 References..... 36

7 Revision History 37

PRELIMINARY

up C

List of Figures

Figure 2-1 relation of input images to flow field	8
Figure 2-2 single camera setup – object motion.....	9
Figure 2-3 single camera setup – real world scenario (input image (1/2)).....	10
Figure 2-4 single camera setup – real world scenario (flow field (horizontal)).....	10
Figure 2-5 single camera setup – real world scenario (flow field (vertical)).....	11
Figure 2-6 dual camera setup – real world scenario (input image (1/2))	13
Figure 2-7 dual camera setup – real world scenario (flow field (horizontal))	13
Figure 2-8 dual camera setup – real world scenario (flow field (vertical)).....	14
Figure 2-9 search window	14
Figure 2-10 search window at different resolutions	15
Figure 2-11 optical flow and fusion.....	16
Figure 3-3 DOF test.....	18
Figure 3-4 separate buffers, no L1 to L2 downscaling.....	19
Figure 3-5 no dedicated L1 flow field buffer, with L1 to L2 downscaling	20
Figure 3-6 multiple similar configuration objects	26
Figure 3-7 configuration objects assigned to ring buffer.....	27
Figure 5-1 shifted pyramid processing approach	29
Figure 5-2 tile processing example	30
Figure 5-2 fusion stage.....	31

List of Tables

Table 3-1 Flow field vector output 22
Table 6-1 Revision History 37

PRELIMINARY



1 Introduction

1.1 Purpose and Scope

This application note serves as a quick guide into Dense Optical Flow (DOF) using Vision IP drivers. Background information of DOF is detailed in chapter 2 and example code is described in chapter 3.

The example code also demonstrates the compatibility between the two major XIL frameworks,

- SIL – simulator in the loop
- HIL – hardware in the loop

Apart from a few UDFs mostly for file I/O and memory management, no other dependencies regarding which framework is used exists in the code. The interaction with the configuration library and the system driver is identical for both frameworks.

1.2 Prerequisites

Hardware:

- V4H Starter Kit development board or similar

Software:

- Linux OS / BSP
- later-Car SDK1 for Windows and Linux

1.3 Terms, Abbreviations

Term / Abbreviation	
BSP	board support package
DOF	dense optical flow
HIL	hardware in the loop
MIL	model in the loop
PIL	processor in the loop
ROI	region of interest
SIL	simulator in the loop
UDF	user defined function
XIL	umbrella for HIL, MIL, PIL and SIL

2 Dense Optical Flow

2.1 Overview

The Dense Optical Flow is a process to extract pixel movement information between two related images. The output of this process are at max, two flow fields indicating where each pixel in the 1st image can be found in the 2nd one and vice versa.

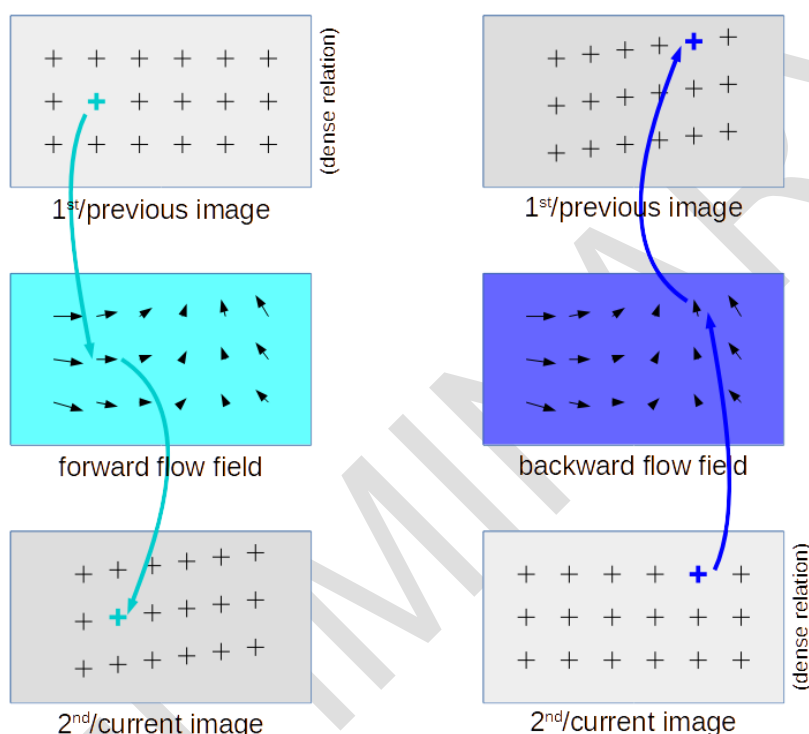


Figure 2-1 relation of input images to flow field

2.2 Applications

2.2.1 Movement of Objects

Using one and the same input source (e.g. a single camera setup), the result of DOF of two consecutive images interprets to movements of objects over time.

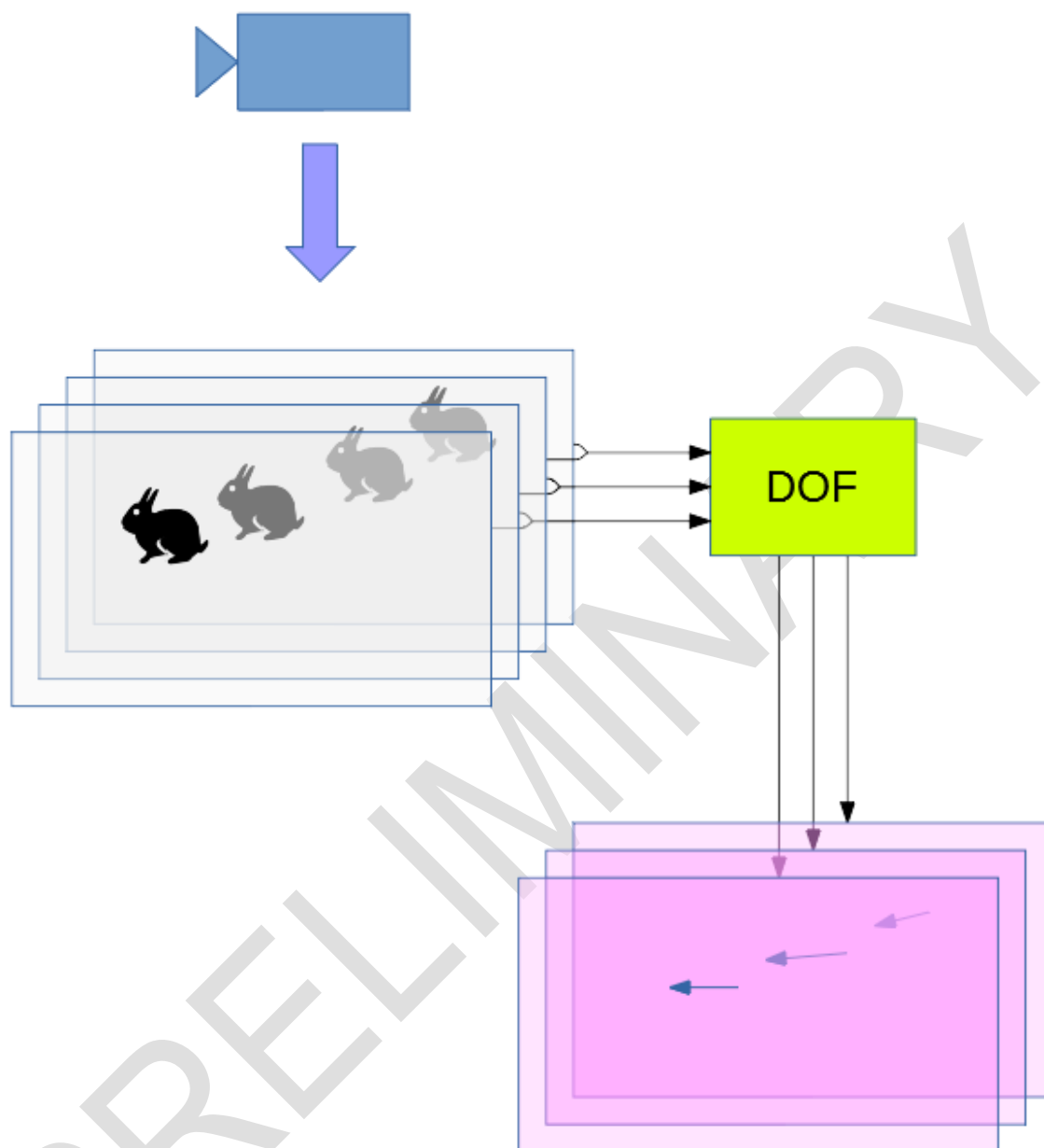


Figure 2-2 single camera setup – object motion



Figure 2-3 single camera setup – real world scenario (input image (1/2))

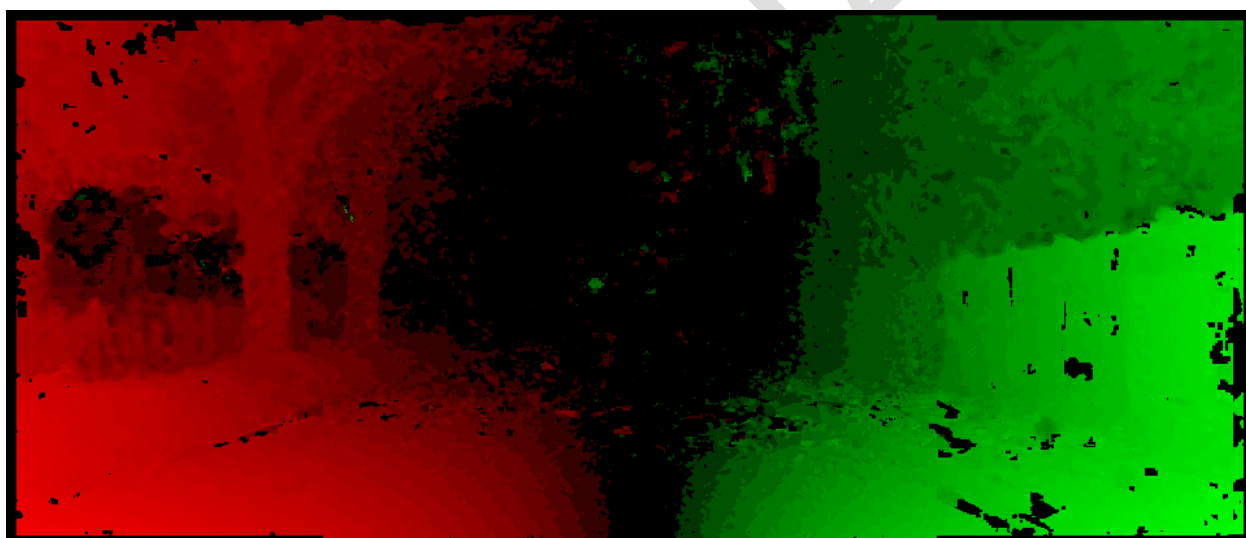


Figure 2-4 single camera setup – real world scenario (flow field (horizontal))

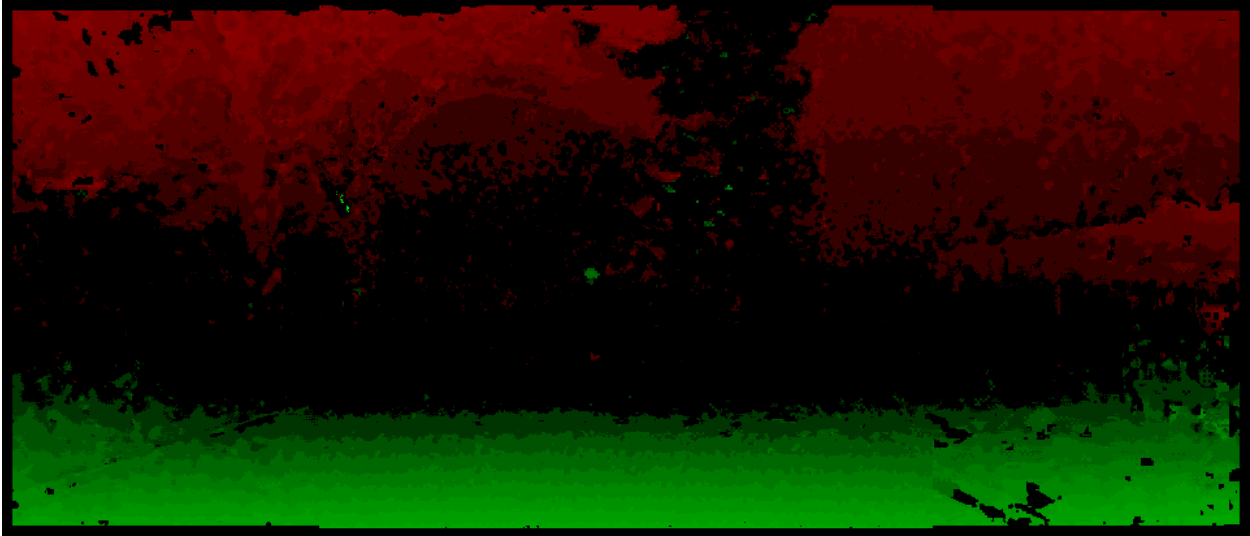


Figure 2-5 single camera setup – real world scenario (flow field (vertical))

2.2.2 Distance of Objects

With a synchronized dual camera setup, processing DOF on their images, taken at the same time, produces a flow field which is related to the distance of objects.

Hereby, for horizontally aligned cameras, only the horizontal flow field vector component carries the desired data. The vertical component, in best case, is zero for all vectors.

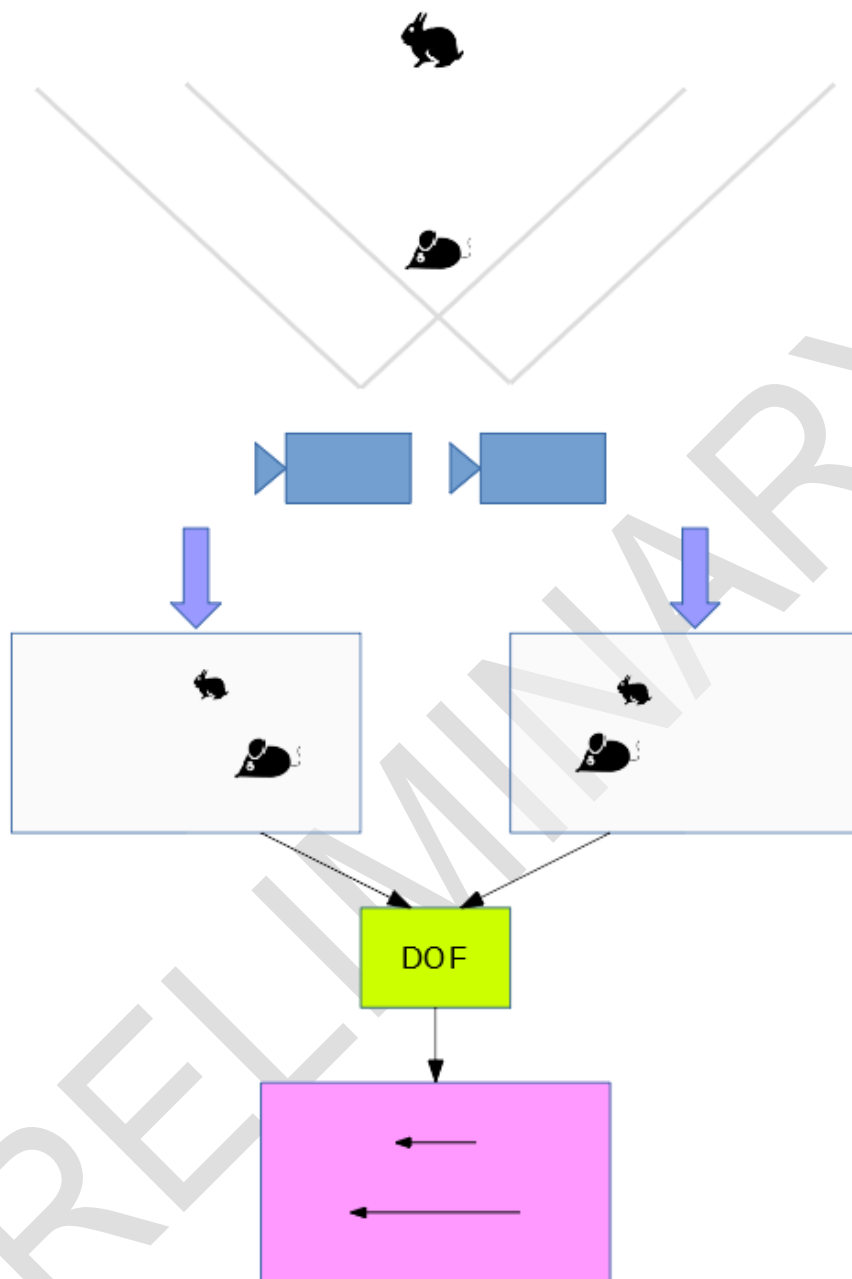




Figure 2-6 dual camera setup – real world scenario (input image (1/2))

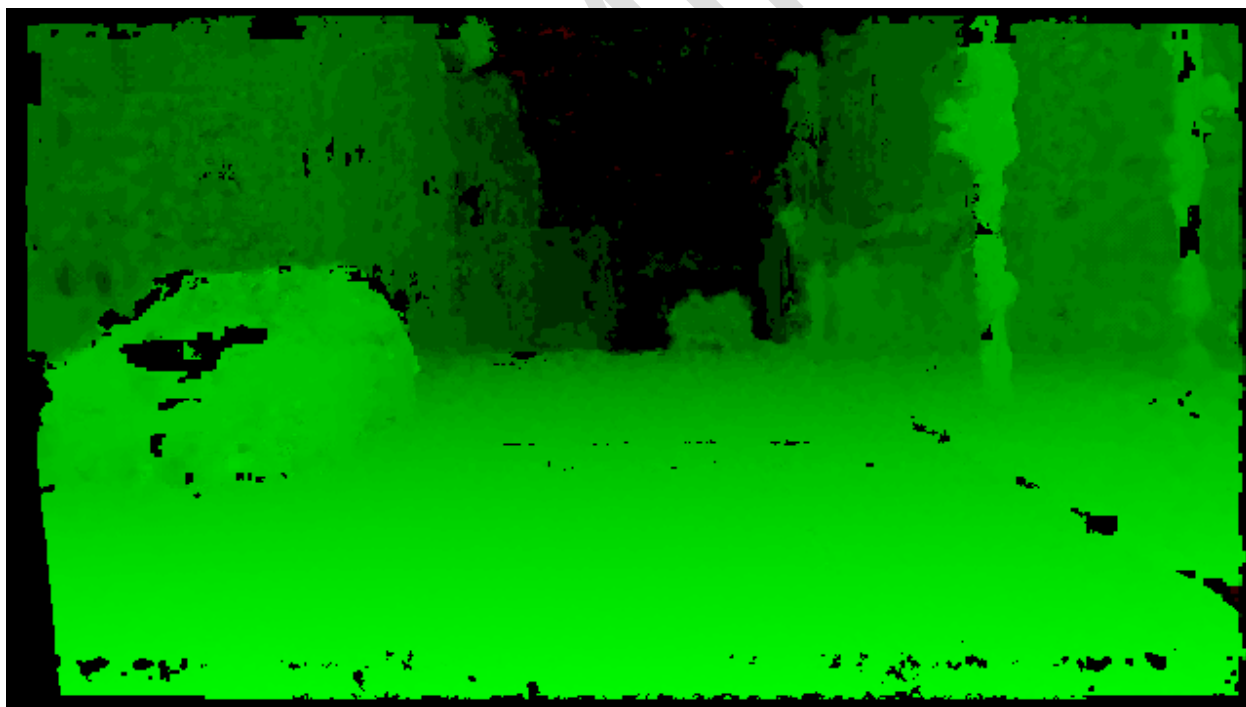


Figure 2-7 dual camera setup – real world scenario (flow field (horizontal))

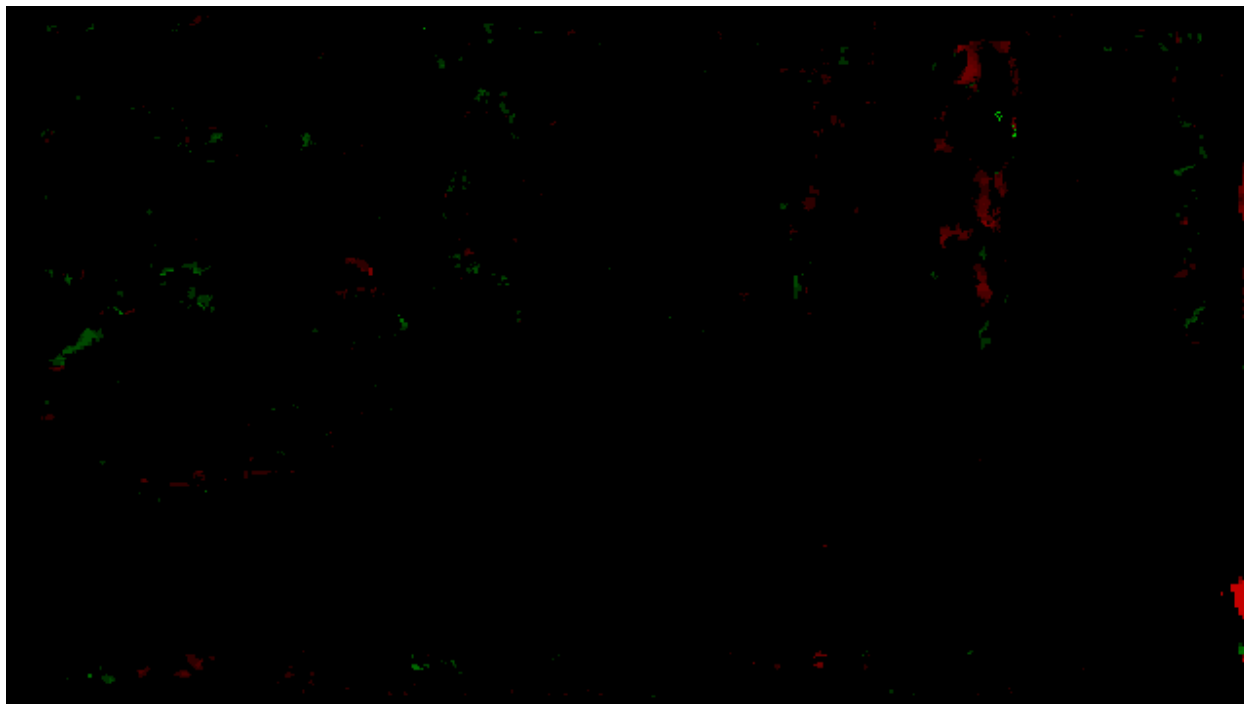


Figure 2-8 dual camera setup – real world scenario (flow field (vertical))

2.3 Process

Described below is the processing for the forward flow. Regarding backward flow, the process is the same only with 1st and 2nd image exchanged.

For **each** pixel in the 1st image, the DOF process uses a search window as an area to search for the matching pixel in the 2nd image.

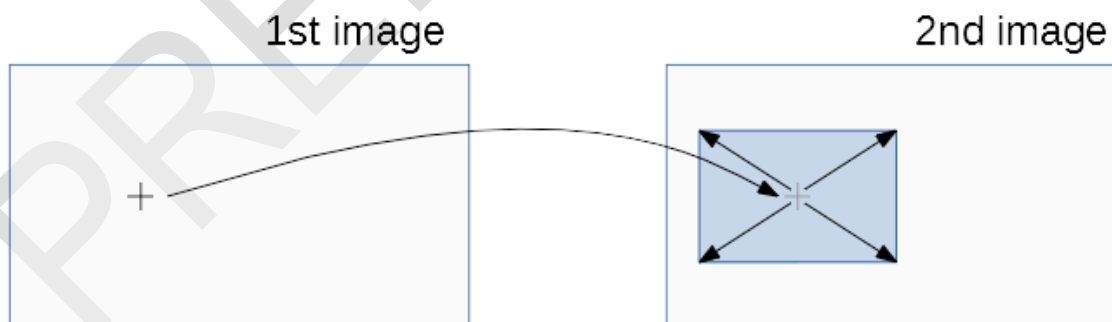


Figure 2-9 search window

The difference in the coordinates of the matching pixel between the 1st and the 2nd image yields the flow vector result.

Calculated for all pixels in the 1st image, a 2-dimensional flow field is created for the entire input image.

As the size of the search window is directly related to the processing effort/time, the DOF process is intended to be executed on up to 3 different resolutions, usually with 1:1, 1:2 and 1:4 scaling.

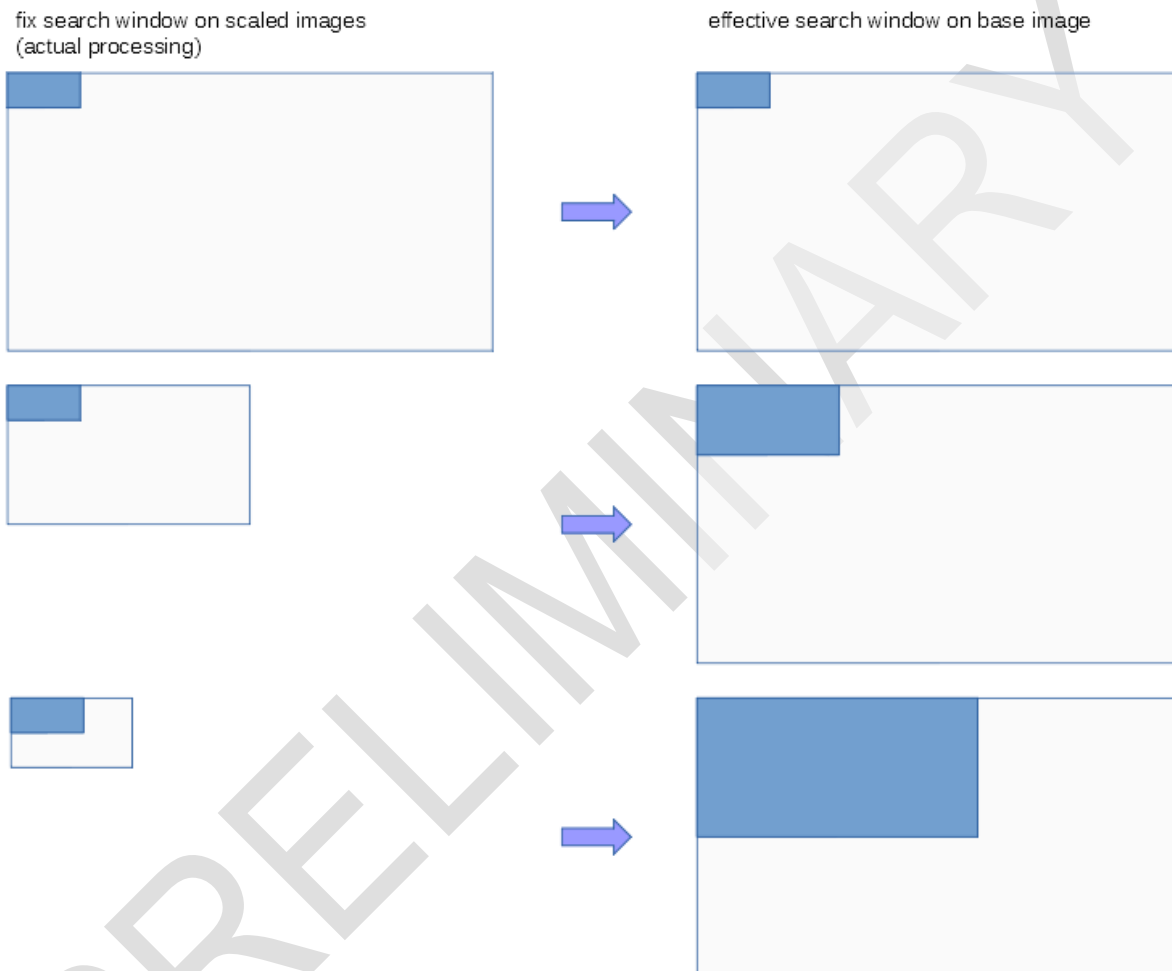


Figure 2-10 search window at different resolutions

Using 3 different resolutions hereby reduces the computational effort:

single execution with required search window:

- resolution factor: 1
 - search window factor: 1
- ⇒ overall factor: $1 * 1 = 1$

3 executions with scaled resolutions and scaled search windows

- 1st execution
 - resolution factor: $(1/4)^2$
 - search window factor: $(1/4)^2$
 - 2nd execution
 - resolution factor: $(1/2)^2$
 - search window factor: $(1/4)^2$
 - 3rd execution
 - resolution factor: 1
 - search window factor: $(1/4)^2$
- ⇒ overall factor: $\left(\frac{1}{4}\right)^2 * \left(\frac{1}{4}\right)^2 + \left(\frac{1}{2}\right)^2 * \left(\frac{1}{4}\right)^2 + 1 * \left(\frac{1}{4}\right)^2 \sim 0.082$

In order to merge the result of the 3 processing's, the DOF hardware IP contains a fusion stage which merges the result of a previous processing on the fly.

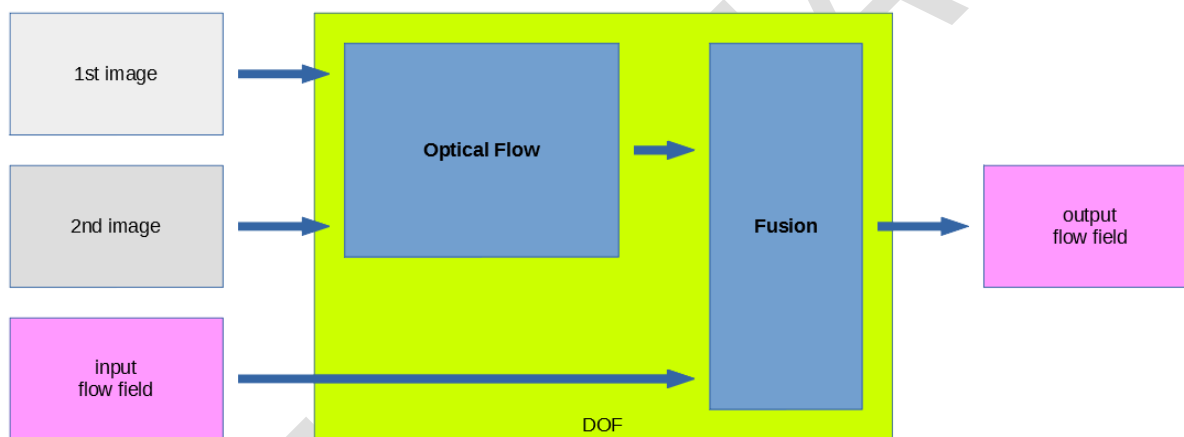


Figure 2-11 optical flow and fusion

Since input and output flow field are usually at different resolutions, the fusion stage is equipped with on-the-fly up & down scaling blocks, usable as necessary.

3 Application

3.1 Content

t.b.d.

3.2 Integration

t.b.d.

3.3 Usage

t.b.d.

3.4 Operation

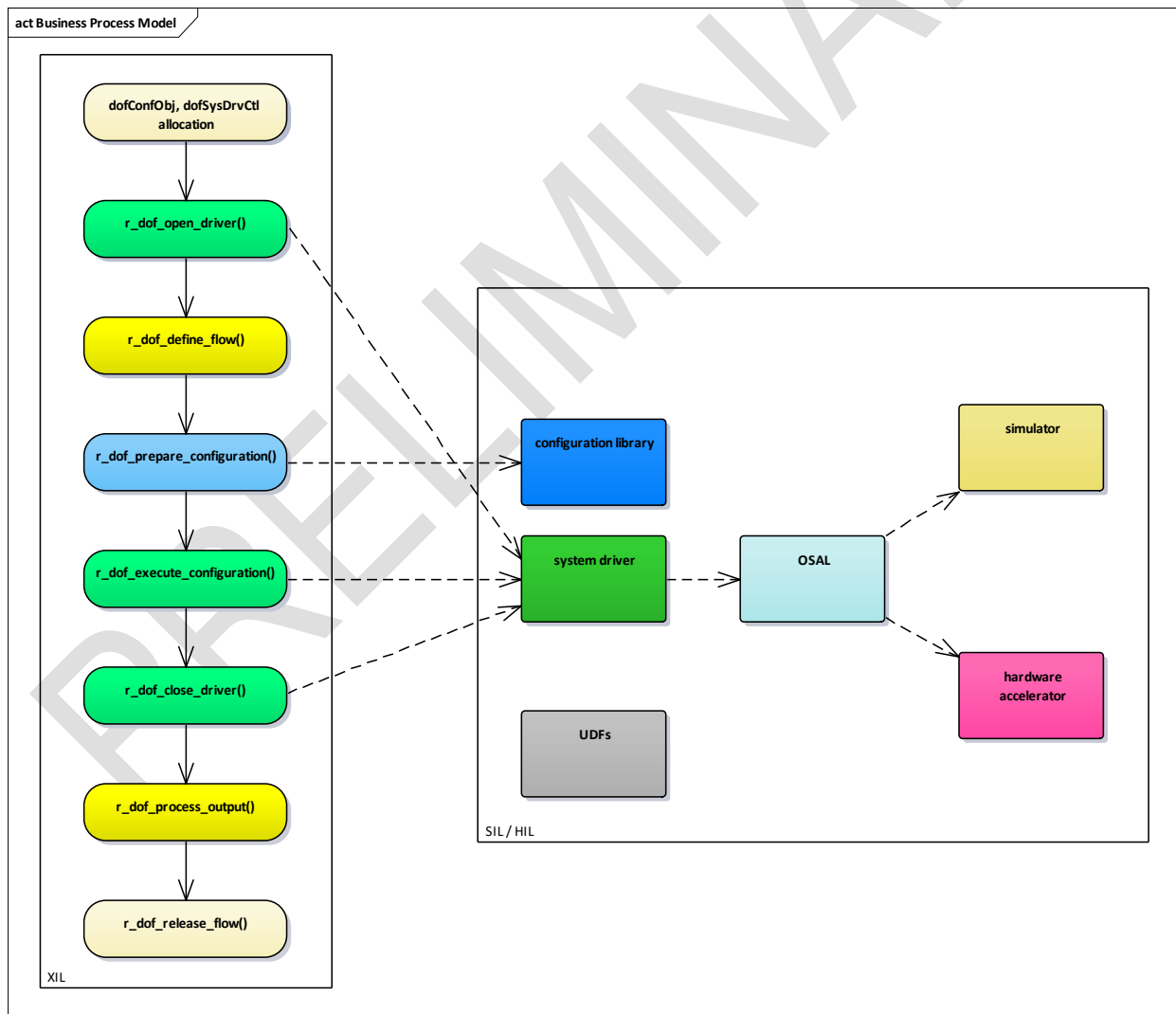


Figure 3-1 DOF test

Both, SIL and HIL implementation of this application note start with the main() function which calls the actual DOF test.

The DOF test, executes various functional blocks. In addition indicated, the majority of the test is assigned to the XIL layer and therefore identical for both SIL and HIL variant of this test. The differences are limited to the caller of this test (i.e. the main() function) and the implementation of the OSAL and layers below.

3.4.1 Flow definition

Prior to flow definition, the application defines the following few key points:

- preferred L1 input image width
- preferred L1 input image height
- downscaling of L1 to L2
- backward flow, forward flow or both in parallel

With these parameters, the flow definition hereby determines all remaining parameters automatically.

i.e. for all levels L3, L2 and L1 and for each level-based input image, input flow (L2 & L1 only) and output flow:

- resolution of frame
- resolution of ROI
- position of ROI within frame

Based on the option for downscaling L1 to L2, the two scenarios exist:

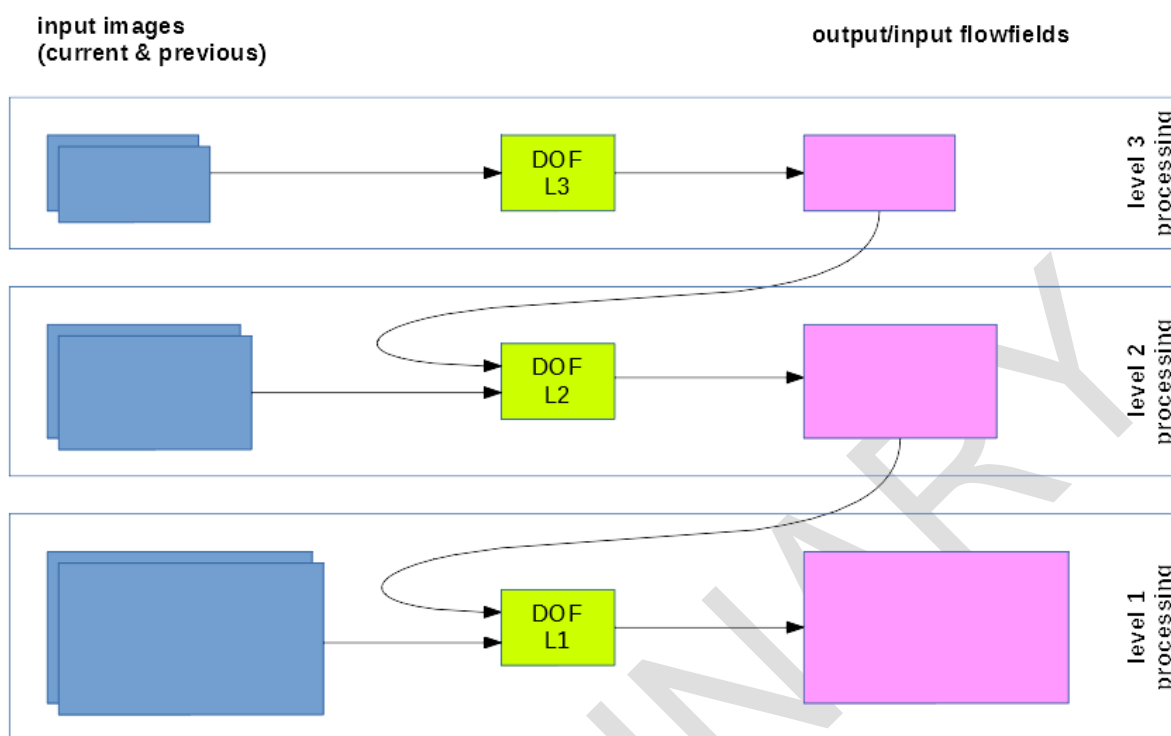


Figure 3-2 separate buffers, no L1 to L2 downscaling

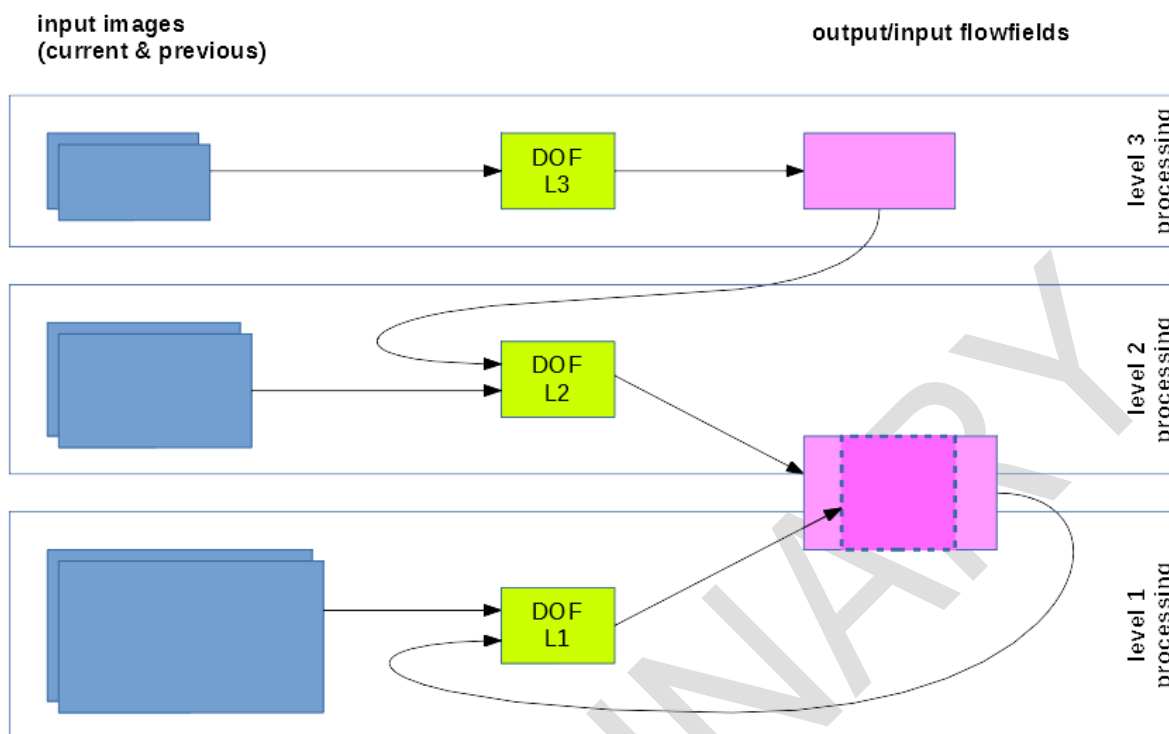


Figure 3-3 no dedicated L1 flow field buffer, with L1 to L2 downscaling

Along with the considerations of the hardware accelerator limitations (e.g. alignments, min/max resolutions) and dependencies among the levels, the target for solving is to achieve the optimal case for the complete flow with minimum buffer sizes without dropping any valid input image pixels.

After successful solving, the flow definition prints the buffer requirements.

3.4.2 Configuration preparation

The preparation of the configuration object is separated in three major blocks:

- initialization of the configuration object
 - setup of default configuration values for all levels L3, L2 & L1
note: this also predefines configuration values not explicitly set in later steps
- conditioning of configuration object
 - apply area information, calculated by the flow definition in 3.4.1
 - allocation and assignment of output flow fields
 - input buffer setup
 - allocation, loading (file I/O) and assignment of input images
 - assignment of input flow fields by reusing output flow fields
- finalizing of the configuration object
 - execution of the configuration libraries finalize function for post checking and ready-for-execution marking

As the result of this configuration preparation, a configuration object is available, ready for execution by simulator / hardware accelerator.

3.4.3 Configuration execution

With the prepared configuration object, the simulator / hardware accelerator is invoked for callback-based execution.

Upon DOF processing end, the assigned callback function is executed, testing for the status / result of the execution and notifying the main thread of the application to continue processing.

3.4.4 Post processing

After execution has finished successfully, export files are created by the post processing functions.

3.4.4.1 Raw data

For each level, a dump of the entire flow field (at frame resolution, including the ROI) is exported as-is without any additional file header or footer.

Therefore, the content of this export is an area of flow field vectors, each 32 bit with the bit utilization as following:

Table 3-1 Flow field vector output

bit	description
31:22	horizontal motion component (data encoding, see FLE)
21:12	vertical motion component (data encoding, see FLE)
11:10	FLE – flow encoding specifies data encoding used for horizontal and vertical motion component: 0: sQ5.4 (range: -32...31.9375, accuracy: 1/32) 1: sQ6.3 (range: -64...63.875, accuracy: 1/16) 2: sQ7.2 (range: -128...127.75, accuracy: 1/8) 3: sQ8.1 (range: -256...255.5, accuracy: 1/4)
9:3	reserved
2	PCD – pyramid confirmed: 0: flow vector at Ln conflicts with or has not result at Ln+1 1: flow vector at Ln corresponds to result at Ln+1 (best case)
1:0	PLO – pyramid level origin: 0: no level, flow vector is invalid 1: L1 (original resolution) (best case) 2: L2 (1/2 resolution) 3: L3 (1/4 resolution)

3.4.4.2 Colorized single component flow fields

For visualization, a horizontal and a vertical image file is created, showing each component's directional magnitude by color brightness.

Hereby red and green are used to indicate negative resp. positive component values with the brightness referring to the magnitude of the component.

The overall brightness of the images is logarithmically upscaled to a maximum value. The maximum value used for this upscaling is chosen from the maximum of both horizontal and vertical component values of all vectors for a single level to keep both resulting images (horizontal and vertical component image) visually comparable regarding their brightness.

up C

PRELIMINARY

3.4.4.3 Level maps

The level maps for each level provide a visual indication of the validity and the origin of the flow vector. This map basically indicates the selections the fusion stage of the DOF has taken.

The color encoding is as following:

- black – flow vector is invalid, no result exists
- white – flow vector from L3
- light gray – flow vector from L2
- dark gray – flow vector from L1

3.4.4.4 Confirmation maps

Confirmation maps for L2 and L1 provide quality information of the flow fields based on the PCD attribute of each flow vector:

- black – flow vector is invalid (after fusion)
- yellow – flow vector at L_n is valid but not at L_{n+1} or vector at L_n does not correspond to vector at L_{n+1}
- green – flow vector at L_n and L_{n+1} are valid and correspond to each other

3.5 Re-Use / Extension

3.5.1 Additional configuration parameters

As noted in 3.4.2, not all possible parameters are set in the given example. In order to change any of the remaining default values (e.g. search ranges), code adaptations targeting the "dofConfObj" can be made anywhere between calls for "r_initialize_configuration_object()" and "r_finalize_configuration_object()" using the configuration libraries function "R_DOF_ConfLibChangeParam()" as exercised in "r_apply_flow()".

3.5.2 Dedicated configuration

For a complete dedicated configuration, using own flow parameter, input and output buffers,

- r_dof_define_flow()
- r_dof_prepare_configuration()

can be completely replaced.

The minimum requirement here is to construct a valid configuration object "dofConfObj" (including the finalization via configuration library) before starting execution of it.

Optionally when setting up the "flowDescriptor" as well, describing buffer properties, the post processing function can be used as-is for visualization purposes.

3.5.3 Partially fixed configuration

In case all flow parameters are known (excluding buffer pointers), the configuration object can be serialized to a non-volatile memory and reloaded later for execution.

Assuming the post processing is not needed, it is only necessary to continue with the assignments of all input and output buffers, finalize the configuration object and start execution.

The complete flow definition, initialization of the configuration object as well as applying any buffer size information can be skipped.

3.5.4 Fully fixed configuration

In addition to 3.5.3, if also buffer locations are statically fixed (virtual & physical address are fix), using a serialized **finalized** configuration object allows execution of the simulator / hardware accelerator without any interacting with the configuration library. This scenario is the fastest possible regarding configuration setup **and** execution.

3.5.5 Streamed ring buffer processing

The example code for this application note is a one-shot case.

A single configuration is created from scratch, executed while the necessary driver interactions are made.

For a use case involving multiple different executions, the process of creating configuration objects is different but can benefit from the various stages as shown next.

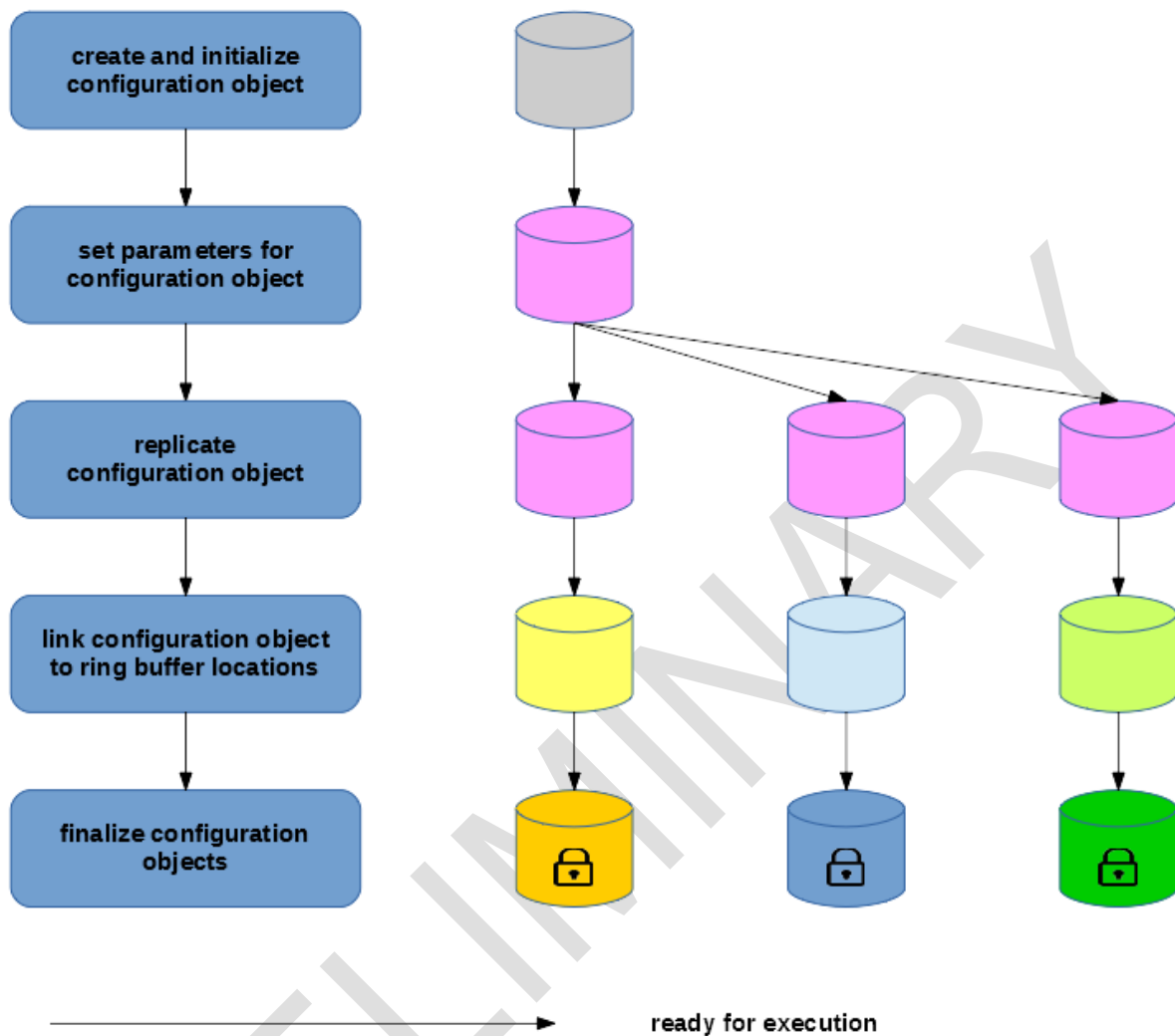


Figure 3-4 multiple similar configuration objects

For a partially fixed configuration, the configuration object can be replicated and afterwards the single instances can be assigned to different buffer locations (e.g. addresses within one and the same ring buffer). Beneficial hereby is to reduce the number of interactions with the configuration library and its overhead to process one and the same task multiple times.

In addition, having as many configuration objects as different buffer locations exist within a ring buffer allows to completely avoid reconfiguration during processing.

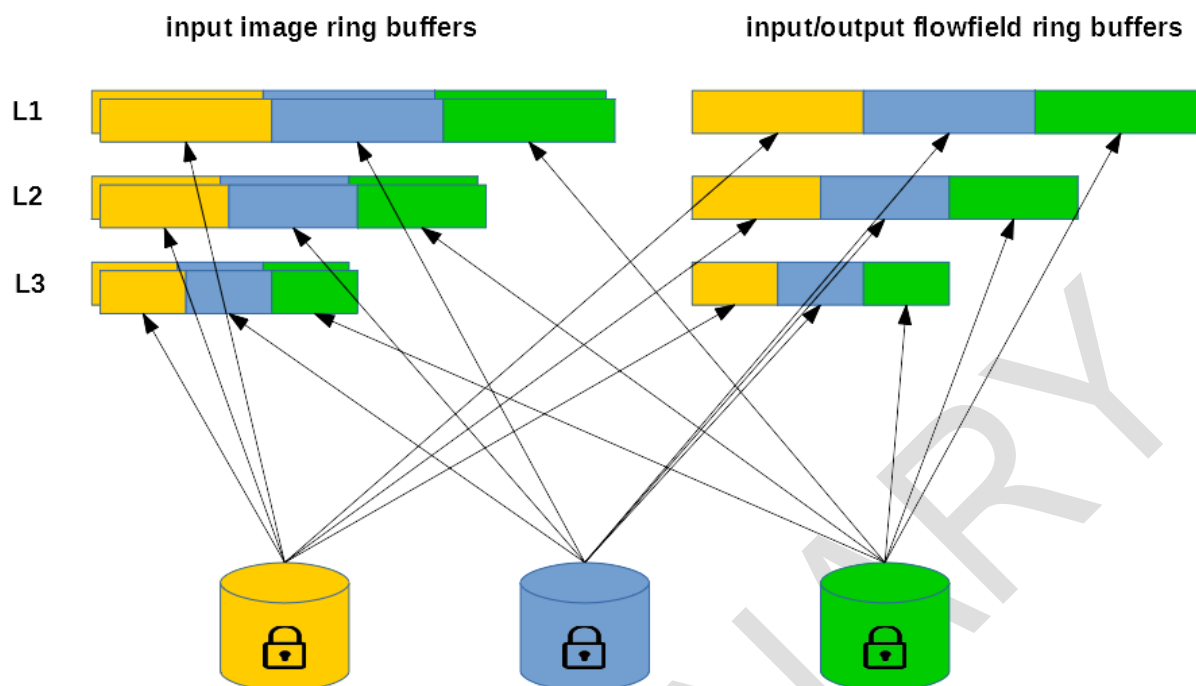


Figure 3-5 configuration objects assigned to ring buffer

Building blocks of this application note which might be usable for this scenario are:

- system driver handling
 - r_dof_open_driver()
 - r_dof_close_driver
- setup of base configuration object
 - r_dof_define_flow()
 - r_initialize_configuration_object()
 - r_apply_flow()
- completion of replicated configuration objects
 - r_prepare_output()
 - r_prepare_input()
 - r_finalize_configuration_object()

4 Open Issues

- Application note in preliminary state.
- functions and parameter types deviate from current implementation

PRELIMINARY

5 Appendix

5.1 Configuration Details

5.1.1 Separation

The DOF driver supports at max 3 executions in a row. Usually, these executions refer to one flow and cover the level L3, L2 and L1, processing one input frame in pyramidal resolutions.

Since the driver and the IP keep the executions separated from each other, deviations from the default use case are possible.

e.g.

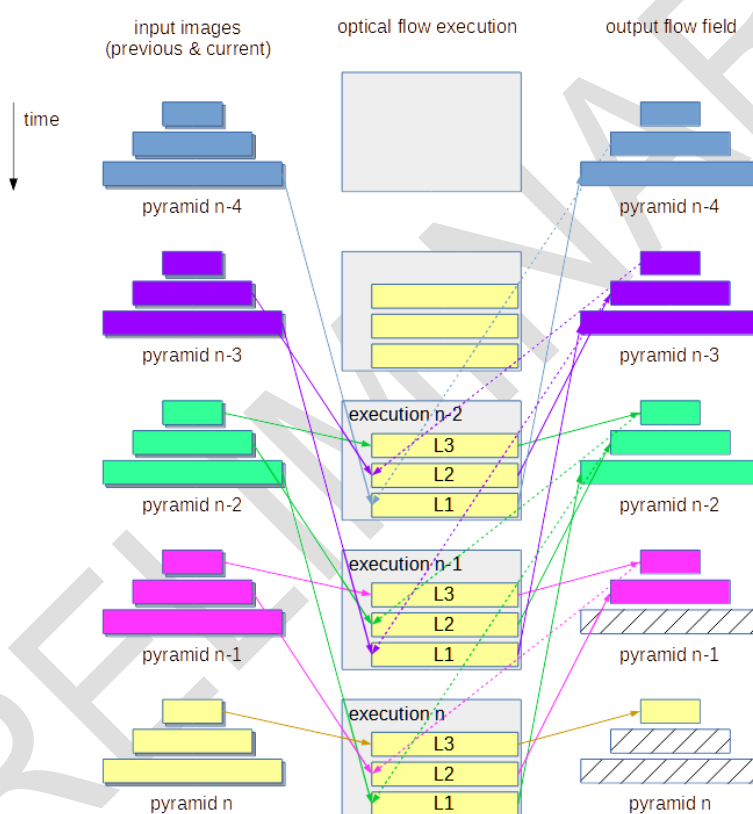


Figure 5-1 shifted pyramid processing approach

Though the latency of having the final result increases, this flow allows adapting the configuration parameters after each execution for a certain pyramid level separately.

To support the execution separation, all executions are configured independently from each other via dedicated execution level parameter for all configuration functions.

5.1.2 Scatter / Gather DMA usage

Input images, input flow field and output flow field are streamed via 2D scatter/gather DMA.

Providing the overall frame resolution as well as ROI configuration allows processing images/flow fields which extend DOF core hardware limits.

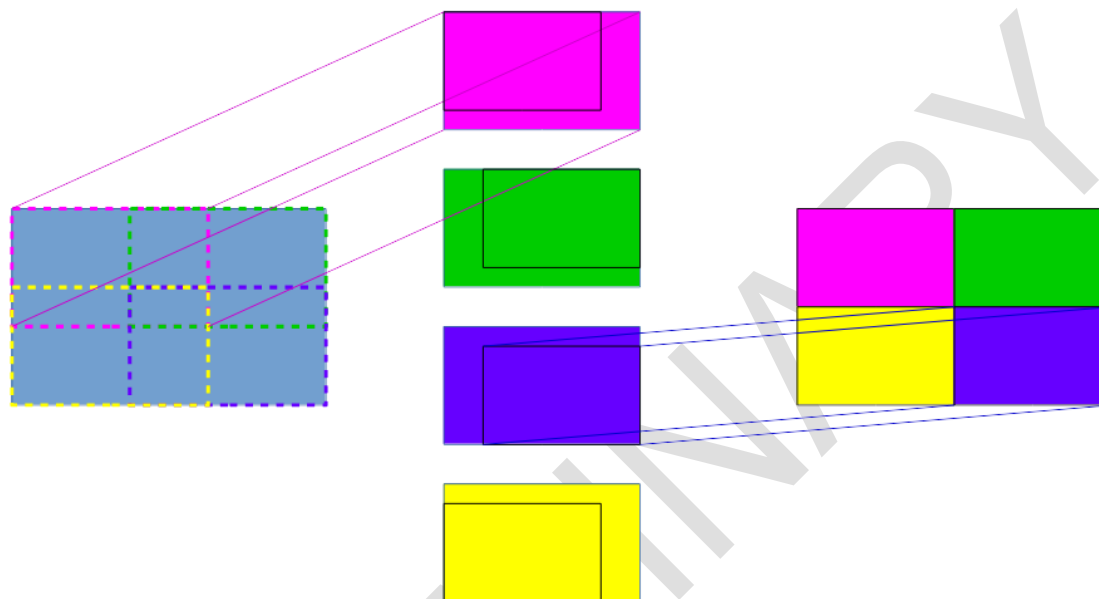


Figure 5-2 tile processing example

Note: For tile processing to be accurate, tiles need to overlap with their search ranges.

5.1.3 Flow field fusion

The configuration of the integrated fusion stage is automatically done by the driver. Forward or backward fusion is active for a certain pyramid level as soon as an input flow field is provided.

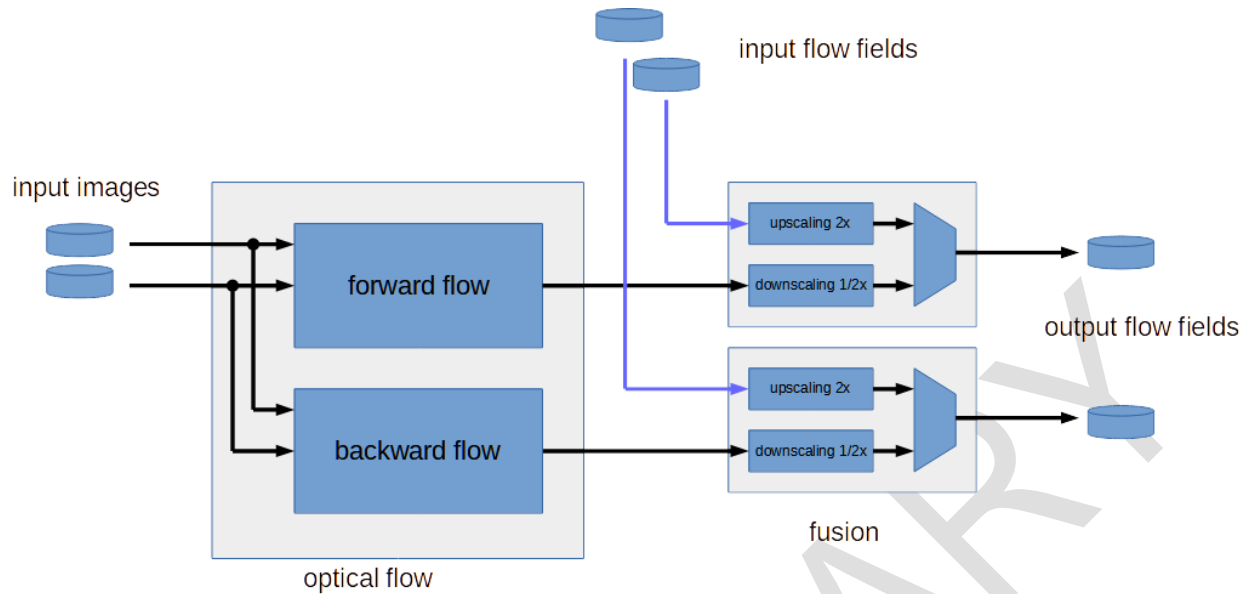


Figure 5-3 fusion stage

The configuration of the optional input flow field upscaling and intermediate flow field downscaling is derived from the resolutions of:

- input images and output flow field => intermediate flow field downscaling
- input flow field and output flow field => input flow field upscaling

5.1.4 Functions and Parameters

5.1.4.1 Input images

buffer setup; address & size:

necessity	mandatory	
related function	R_DOF_ConfLibSetInputImages()	
related arguments	e_dof_pyr_level_t level	pyramid level
	uint16_t* p_curr_frame	ptr to buffer of current image
	uint16_t* p_prev_frame	ptr to buffer of previous image
	uint32_t frame_size	byte size of each image buffer

buffer setup; resolution:

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	
related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter
related parameters types	DOF_PAR_IN_CURR_FRAME_SIZE	pixel count of buffer for current image

	DOF_PAR_IN_PREV_FRAME_SIZE	pixel count of buffer for previous image
	DOF_PAR_IN_ROI_FRAME_WIDTH	width of buffer for current image
	DOF_PAR_IN_ROI_FRAME_HEIGHT	height of buffer for current image

ROI setup:

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	
related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter
related parameters types	DOF_PAR_IN_ROI_WIDTH	width of ROI within buffer for current & previous image
	DOF_PAR_IN_ROI_HEIGHT	height of ROI within buffer for current & previous image
	DOF_PAR_IN_ROI_HORI_OFF	horizontal offset of ROI within buffer for current & previous image
	DOF_PAR_IN_ROI_VERT_OFF	vertical offset of ROI within buffer for current & previous image

5.1.4.2 Input flow field

buffer setup; address & size:

necessity	optional	
related function	R_DOF_ConfLibSetInputFlowfields()	
related arguments	e_dof_pyr_level_t level	pyramid level
	uint32_t* p_fwd_flow_field	ptr to buffer of input forward flow field
	uint32_t* p_bwd_flow_field	ptr to buffer of input backward flow field
	uint32_t flow_size	byte size of input forward and backward flow field buffers

buffer setup; resolution:

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	
related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter

related parameters types	DOF_PAR_INFLW_FRAME_SIZE	flow vector count of buffer for input forward and backward flow field
	DOF_PAR_INFLW_ROI_FRAME_WIDTH	width of buffers for input forward and backward flow field
	DOF_PAR_INFLW_ROI_FRAME_HEIGHT	height of buffers for input forward and backward flow field

ROI setup:

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	
related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter
related parameters types	DOF_PAR_INFLW_ROI_WIDTH	width of ROI within buffers for input forward and backward flow field
	DOF_PAR_INFLW_ROI_HEIGHT	height of ROI within buffers for input forward and backward flow field
	DOF_PAR_INFLW_ROI_HORI_OFF	horizontal offset of ROI within buffers for input forward and backward flow field
	DOF_PAR_INFLW_ROI_VERT_OFF	vertical offset of ROI within buffers for input forward and backward flow field

5.1.4.3 Output flow field

buffer setup; address & size:

necessity	mandatory	
related function	R_DOF_ConfLibSetOutputFlowfields()	
related arguments	e_dof_pyr_level_t level	pyramid level
	uint32_t* p_fwd_flow_field	ptr to buffer of output forward flow field
	uint32_t* p_bwd_flow_field	ptr to buffer of output backward flow field
	uint32_t flow_size	byte size of output forward and backward flow field buffers

buffer setup; resolution:

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	

related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter
related parameters types	DOF_PAR_OUTFLW_ROI_FRAME_WIDTH	width of buffers for output forward and backward flow field
	DOF_PAR_OUTFLW_ROI_FRAME_HEIGHT	height of buffers for output forward and backward flow field

ROI setup:

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	
related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter
related parameters types	DOF_PAR_OUTFLW_FRAME_SIZE	flow vector count of buffer for input forward and backward flow field
	DOF_PAR_OUTFLW_ROI_WIDTH	width of ROI within buffers for output forward and backward flow field
	DOF_PAR_OUTFLW_ROI_HEIGHT	height of ROI within buffers for output forward and backward flow field
	DOF_PAR_OUTFLW_ROI_HORI_OFF	horizontal offset of ROI within buffers for output forward and backward flow field
	DOF_PAR_OUTFLW_ROI_VERT_OFF	vertical offset of ROI within buffers for output forward and backward flow field

5.1.4.4 Search ranges

necessity	optional	
related function	R_DOF_ConfLibChangeParam()	
related arguments	e_dof_pyr_level_t level	pyramid level
	e_dof_param_type_t type	type of parameter
	int32_t val	value of parameter
related parameters types	DOF_PAR_SEARCH_RANGE_LEFT	search window left border distance relative to flow vector coordinates

	DOF_PAR_SEARCH_RANGE_RIGHT	search window right border distance relative to flow vector coordinates
	DOF_PAR_SEARCH_RANGE_UP	search window top border distance relative to flow vector coordinates
	DOF_PAR_SEARCH_RANGE_DOWN	search window bottom border distance relative to flow vector coordinates
	DOF_PAR_VERTICAL_SHIFT	vertical (down) shift of overall search window

5.1.4.5 Image pre filtering

necessity	optional	
related function	R_DOF_ConfLibSetSmcCoeff()	
related arguments	e_dof_pyr_level_t level	pyramid level
	uint8_t *p_ft_c	ptr to 3x3 matrix with filter kernel for current input image
	uint8_t *p_ft_p	ptr to 3x3 matrix with filter kernel for previous input image

5.1.4.6 Flow vector shifting

necessity	optional	
related function	R_DOF_ConfLibSetBitShiftEn()	
related arguments	e_dof_pyr_level_t level	pyramid level
	uint32_t value	digits to left shift each flow vector before fusion stage

Note:

Using a large search range and a large flow vector shifting may lead to overflowing vectors which cannot be encoded in the output anymore.

In this case, the IP will trigger an error interrupt to indicate this issue to the application.

6 References

There are no sources in the current document.

PRELIMINARY

7 Revision History

Table 7-1 Revision History

Version	Details	Chapter	Release Date	Prepared by	Approved by
1.0	initial document release	all	06.12.2019	A. Schulz	
(2.0)	update for OMM 1.1	t.b.d.	n/a	A. Schulz	