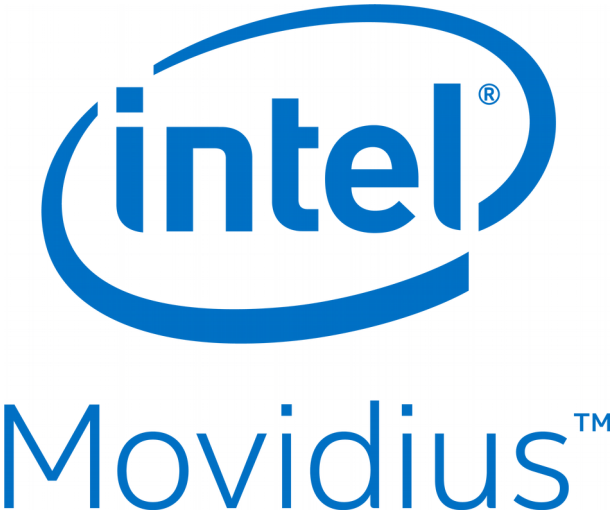


MA2xxx vTrack Software Module

Specification

v1.0 / May 2018

Copyright and Proprietary Information Notice

Intel® Movidius™
2200 Mission College Blvd
M/S SC11-201
Santa Clara, CA 95054
http://www.movidius.com/

Contents

1 Introduction

1.1 Scope

This document is intended for Myriad developers and provides the software specification and implementation details for the vTrack software module offered as an addition to the standard Myriad Development Kit (MDK).

1.2 Licensing Information

Use of this software is covered by a specific Intel® Movidius™ Software License Agreement.

1.3 Disclaimer

The information in this document and any document referenced herein is provided for informational purposes only, is provided AS IS AND WITH ALL FAULTS and cannot be understood as substituting for customized service and information that might be developed by Intel® Movidius™ for a particular user based upon that user's particular environment. RELIANCE UPON THIS DOCUMENT AND ANY DOCUMENT REFERENCED HEREIN IS AT THE USER'S OWN RISK.

1.4 Support Information

For MDK support you can post your questions on tickets at https://www.movidius.org/.

2 Overview

vTrack is a feature tracking algorithm. The goal is to detect keypoints (features) on a frame and track them on the next frames.

2.1 Features

2.1.1 Overview

The vTrack algorithm has the following key features:

- Detects a high number of features on a single or multiple input frame(s).
- Finds the new position of the previously detected features, on the current frame.
- Tracks large feature displacement with good precision.
- Outputs the current and previous position of the features together with the feature ID.
- Estimates the features new position based on a gyro rotation matrix (gyro assist).
- Tracks features for a very long time.
- Configurable shave number to minimize latency in different use-cases.
- Configurable output by using different vTrack modes.
- Configurable features (i.e. distance, strength, number).
- Automatic adaptation to different scenes (adaptation to textureness).

2.1.2 Compatible Silicon

The algorithm is implemented, tested and optimized for the MA2x5x and MA2x8x silicon.

2.2 References

2.2.1 Literature

[1] "*Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm*", Intel Corporation, J.Y. Bouguet, 2001.

[2] "*ORB: An efficient alternative to SIFT or SURF.*" Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary R. Bradski: ICCV 2011: 2564-2571

[3] "*Rapid Online Analysis of Local Feature Detectors and Their Complementarity*", Shoaib Ehsan *, Adrian F. Clark and Klaus D. McDonald-Maier, Sensors 2013

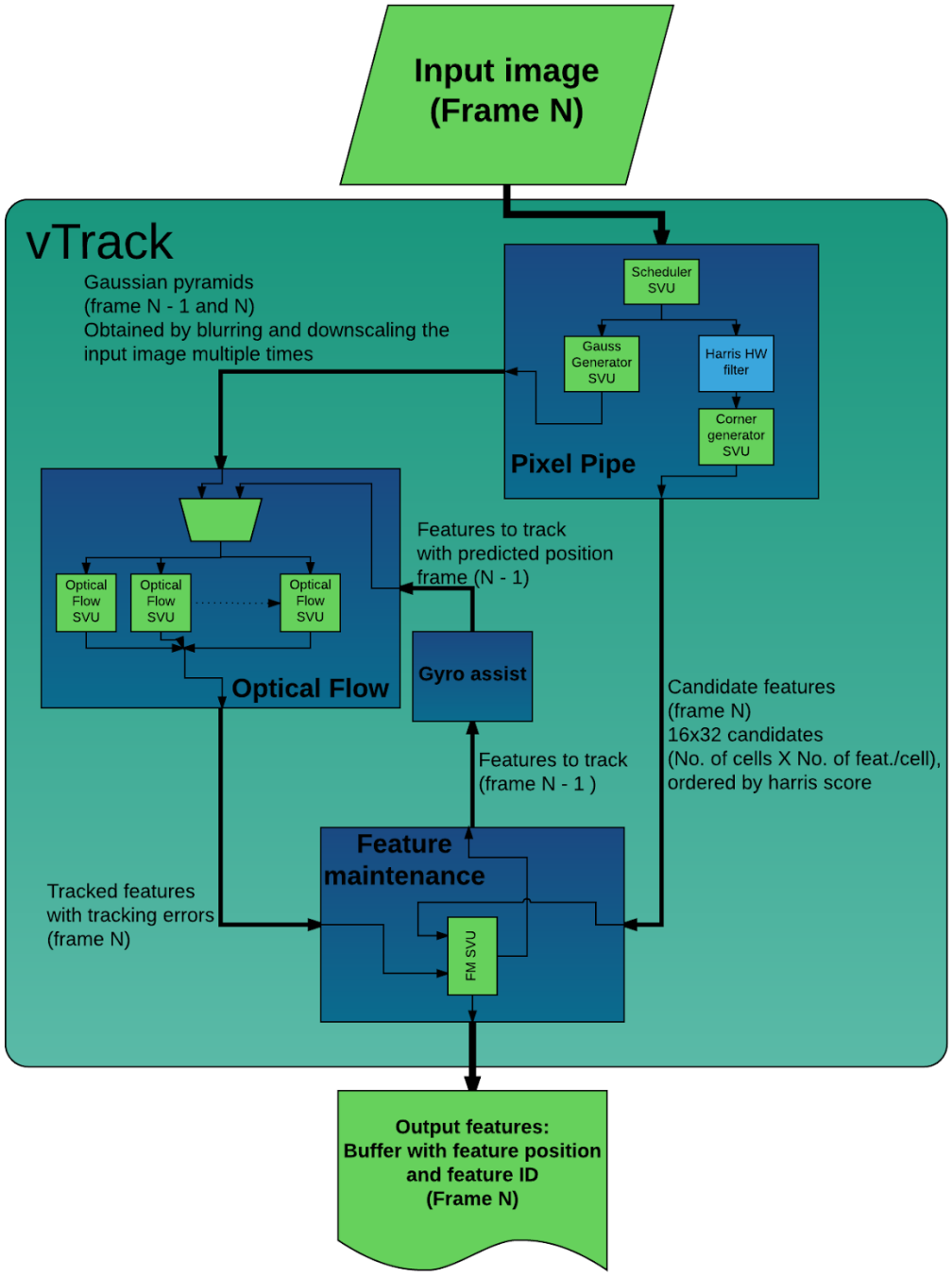
2.2.2 Open Libraries

The source code provided with this component is wholly owned by Movidius. Similar functionality is encapsulated by the following code found in other libraries:

- http://docs.opencv.org/modules/legacy/doc/motion_analysis.html#calcopticalflowlk

3 Functional Description

3.1 Architecture



The algorithm has four major components:

1. **vPipe**: This is the interface of vTrack with the application. It is responsible to schedule the other components of the algorithm based on the user setting.
2. **pixelPipe**: Detect keypoints (features) on the input frame and generate the gaussian pyramid for optical flow.

3. **motionEstimation**: Find the new position of the previous features.
4. **featureMaintenance**: Maintaining the list of the tracked features. The component will assign id and age for the features, drop the features which weren't tracked correctly by the opticalFlow. If needed, it will also add the strongest features coming from pixelPipe. While the method employed here is unique, similar ideas are employed in “good feature learning” as described in [2].

3.2 Component description

3.2.1 Vpipe

This component is the scheduler of the vTrack. It will start the above components based on the user configuration. This component will run on Leon only. No shaves are used by the component. vPipe has several execution modes, configurable dynamically at initialization phase. They are described in section 3.4.

3.2.2 PixelPipe

This component has a double purpose: it will generate a gaussian pyramid and a list of good features to track. The good features are obtained from the Harris score or Shi-Tomasi (Myriad X only). The gaussian pyramid is padded with a configurable number of pixels (default is 8). To have features all around the image, pixelPipe will divide the image into cells (default is 16) and process each cell separately.

3.2.3 Gyro Assist

This is an optional component in vTrack that predicts the displacement of previously tracked features from some external input. We predict the new feature positions using a rotation matrix. The algorithm accepts a rotation matrix at each frame to predict the new feature positions. If no rotation matrix is provided, feature prediction is disabled. The rotation matrix can be calculated by using (for example) a gyroscope, outside of the vTrack algorithm. Using feature prediction greatly improves runtime and power consumption.

The feature prediction is fed into the optical flow component for feature tracking. If no rotation matrix is provided, the component assumes that there is no feature displacement, so the search of the feature is started from the previous positions. This is acceptable while the feature displacement is not too big though not ideal.

3.2.4 Motion Estimation

This component receives as input the previous image pyramid, the current image pyramid, list of features and new feature prediction. After execution, it will provide the position of the previous features on the current image.

The motion of the points can be calculated in two possible ways:

1. Using the pyramidal Lucas-Kanade optical flow method.
2. Using a dense motion estimation hardware block on the myriad (Block matcher).

3.2.5 FeatureMaintenance

This component will get the list of features from optical flow, along with the measured tracking error. It will

also receive new features from the pixelPipe. The component then decides which features will be dropped and which will be kept for the subsequent frames.

Tracked features have priority over new features from pixelPipe. The algorithm will sort this list first. It will then drop the features which:

- Have too large tracking error (wasn't tracked correctly).
- Have too small Harris score (configurable threshold).

For the first frame this list is empty. After this, it will select the strongest features from the candidate list (from pixelPipe) to achieve the target number of features. The entry conditions of the features are the following:

- No feature in the list which is too close to the candidate features (minimum distance criteria).
- Harris score of the feature is high enough.
- Room in the cell for the feature (target feature number is not achieved).

3.3 Supported Parameters

3.3.1 General configuration parameters

3.3.1.1 Frame spec

720p and 480p resolutions are supported.

3.3.1.2 Output Type

There are two options for output type:

Tracked + pixel pipe: vTrack will output features tracked from the previous frame and pixel pipe features from the current frame.

Tracked only: vTrack will only ouput the features tracked from the previous frame.

3.3.2 Vpipe

3.3.3 Pixel Pipe

3.3.3.1 Number of target features

Total number of features desired. The algorithm will try to keep the output feature number around this target. The default is 320

3.3.3.2 Maximum number of features

Maximum number of features desired. The algorithm will try to keep the output feature number strictly below this number. Buffers are allocated based on this number so It cannot be overrun. The default is 480

3.3.3.3 Number of image cells

Image is divided into cells. Each cell has a target feature count = frame target features / number of cells.

This mechanism ensures that we have distributed features in the image and not only from one part of it. Because different regions of the image can contain different amounts of texture, the algorithm will update feature thresholds for each cell separately.

The features can move from one image cell to another in time. If a cell is full, but an already tracked point has a new coordinate in this cell, the algorithm will not drop the feature. Because of this we can have more features in a cell than the target number.

The number of cells can be configured in horizontal and in vertical direction. The default number of cells is 4 (horizontal) x 4 (vertical). This means that the default number of target features/cell is 20

3.3.3.4 Initial Harris threshold

This threshold controls the minimum strength of a feature which will be detected. This is needed, because features below a certain strength (regions without texture) can't be tracked correctly.

vTrack supports automatic threshold update, which can be activated or deactivated using this value. Setting the threshold to 0 when calling the vTrack run function will enable the automatic thresholds. The benefit of having variable threshold is that the algorithm can adapt to different scenes.

If there is a lot of texture, we need to increase the threshold to limit the number of detected points which are propagating through the vTrack modules. This is mostly a runtime optimization. Each cell has its own threshold.

3.3.3.5 Automatic Harris threshold adaption speeds and limits

To control the automatic threshold adaption we can use four values:

1. **Minimum Harris threshold**: The threshold will not go under this limit. Even if there are not enough points, we don't go below this limit because the points are not strong enough to be trackable.
2. **Maximum Harris threshold**: The threshold will not go below this limit. If the threshold goes too high in a certain time, it can take too much to get back to a normal value. Imagine a chessboard in front of the camera which increases the thresholds. If the board disappears from one frame to another, we would not detect enough points on the next frames.
3. **Harris threshold increase velocity**: the speed of threshold increase.
4. **Harris threshold decrease velocity**: the speed of threshold decrease.

3.3.3.6 Number of Pyramid levels

The number of pyramid levels generated by pixel pipe. Since the pyramids are used by optical flow (see section 3.3.5.2), this parameter is shared between pixel pipe and optical flow. It is also possible to disable

3.3.3.7 Corner Detection algorithm

Pixel pipe can use different configurations to detect features. This parameter allows the selection of the feature detector.

There is an option to choose Harris corner detector or the Shi-Tomasi corner detector. When computing the

gradient for either of the corner detectors, the sobel operator is used to smoothen the image whose gradient is be computed. There is an option to disable the sobel operator and simply use a 1D row/column differentiator

Another variable in the algorithm configuration is in the insertion of features. It can be set so that (i)the first N features are added or (ii) the best N features by corner score are generated. N is the target number of features. The rest of the features are discarded.

3.3.3.8 Pyramid detection

The pyramid detection in pixel pipe can be turned off/on. This will be done depending on whether the pyramids are needed for motion estimation (optical flow) or not (Mest).

3.3.4 Gyro Assist

3.3.4.1 Camera parameters for gyro assist

Gyro assist will predict the feature positions before running optical flow based on a rotation matrix (usually calculated from gyro samples). For this, it needs some of the camera parameters: camera center, FOV. These values can be filled without camera calibration, too (i.e. camCenterX = WIDTH /2). Even if the gyro assist is not perfect, it still helps optical flow to start from a much closer coordinate. Of course, for best performance a good camera calibration must be filled.

3.3.5 Optical Flow

3.3.5.1 Search window size

Defines the image patch size used for tracking the feature. Increasing this value will allow tracking higher motions, but it will cause a decrease in runtime. The algorithm supports sizes between 11x11 and 23x23. As 11x11 and 19x19 are assembly optimized, we suggest using one of these two configurations.

NxN configuration means that the algorithm will not be able to track bigger motion than (N-1)/2 pixels in a certain direction on a certain pyramid level. (i.e. 11x11 configuration will enable tracking up to 5 pixels in each direction in a pyramid level).

Default configuration is 11x11.

3.3.5.2 Number of image pyramid levels (only for OF)

The image pyramid is used to enable tracking bigger motions than the window size. Each added pyramid level, in theory, doubles the maximum motion.

The default value depends on resolution: 4 levels for VGA, 5 levels for 720p.

NOTE: OpenCV implementation of Optical Flow does not count the base level as a pyramid level. The configured number is N-1 where n is the total number of levels. Following the OpenCv practice, our algorithm has the same input value.

3.3.5.3 Epsilon (only for OF)

Feature tracking termination criteria. Optical flow will refine the feature position on each pyramid level until

the displacement between two refinements is smaller than this value. The default value is 0.01. Note that OpenCV squares this value which means that the same value 0.01 means 0.0001 in OpenCV. Decreasing this number increases Optical Flow runtime.

3.3.5.4 Max OF iterations (only for OF)

Feature tracking termination criteria. Optical flow will refine the feature position maximum this many times on each pyramid level. If the Epsilon criteria described in the previous chapter is not met after this number of iterations, the algorithm will continue with the current calculated value.

The default is 9 iterations. Increasing this number increases the OF runtime.

3.3.5.5 Motion Estimation Algorithm

The motion estimation component can use either optical flow or the hardware block matching component on the myriad.

3.3.6 Feature Maintenance

3.3.6.1 Lost feature error threshold

Optical flow measures the tracking error for every feature. If the point can't be tracked or it's out of the image it will set this error to a maximum value. This threshold defines the level where the tracking accuracy is considered too bad to keep the point.

3.3.6.2 Minimum distance between features

Corner detectors tend to detect many points close to each other which can contain redundant information. In most of the cases we want to prevent a high textured object stealing all our feature points leaving the rest of the cell without features.

The minimum distance criteria will ensure that any feature pair is at a bigger euclidean distance than this threshold. The default value is 50, the unit of measurement being squared euclidean distance in pixels.

3.3.6.3 Harris threshold for tracked features

The Harris thresholds described in previous sections would prevent the detection of a feature. Once a feature was detected and we started tracking it, we need to update its Harris score on each image.

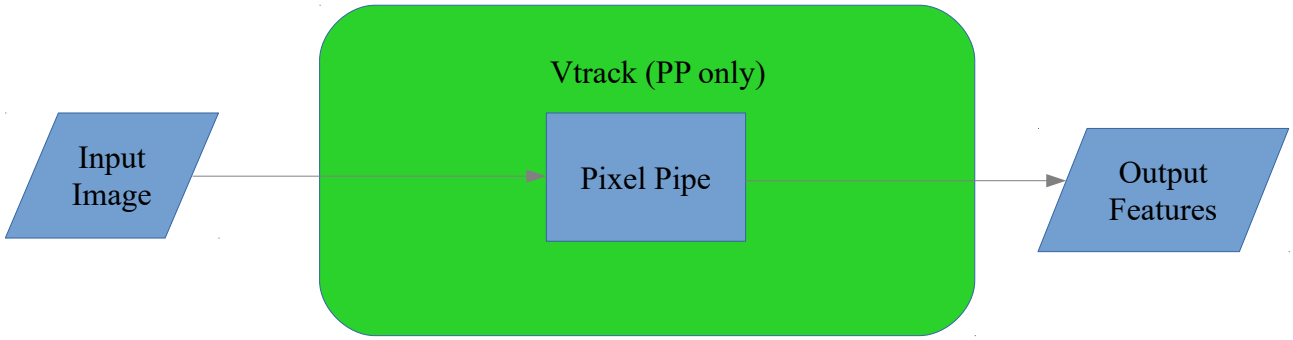
This is needed because a feature point can disappear, or it can become too weak to be tracked. This threshold defines the point where such a feature must be dropped.

As the goal of the algorithm is to provide longer tracks, we try to add strong points and track them until they are absolutely untrackable. This is why, this value is usually smaller than the detection threshold.

3.4 Run Modes

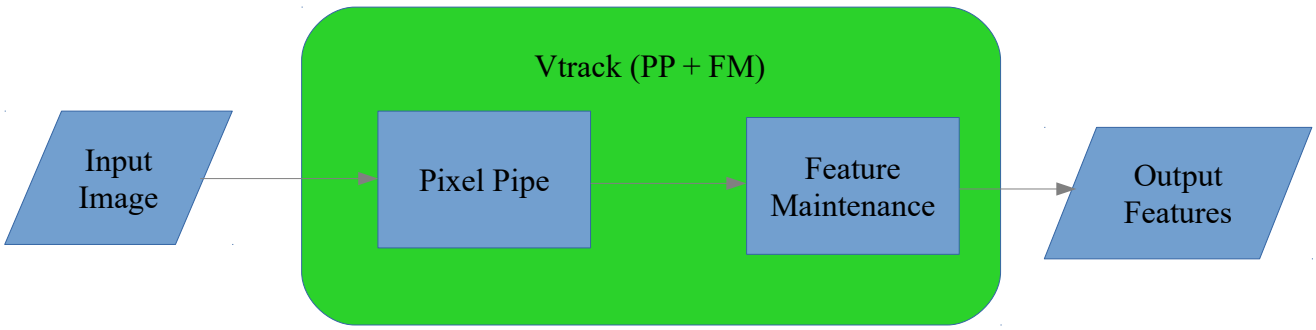
3.4.1 PixelPipe

Vtrack will only generate a list of features and the image pyramid.



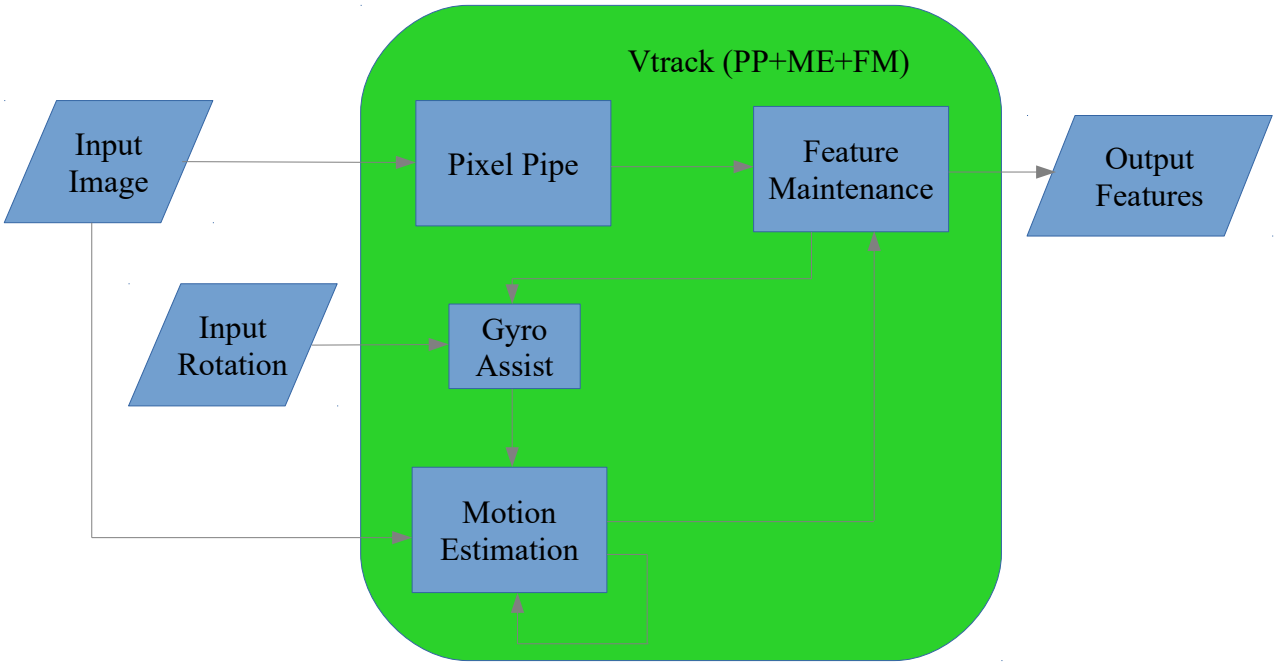
3.4.2 PixelPipe + Feature Maintenance

Vtrack will generate a list of features and keep only the features which fulfill the entry criteria. This mode will not trigger optical flow, therefore it will not track the features across frames.



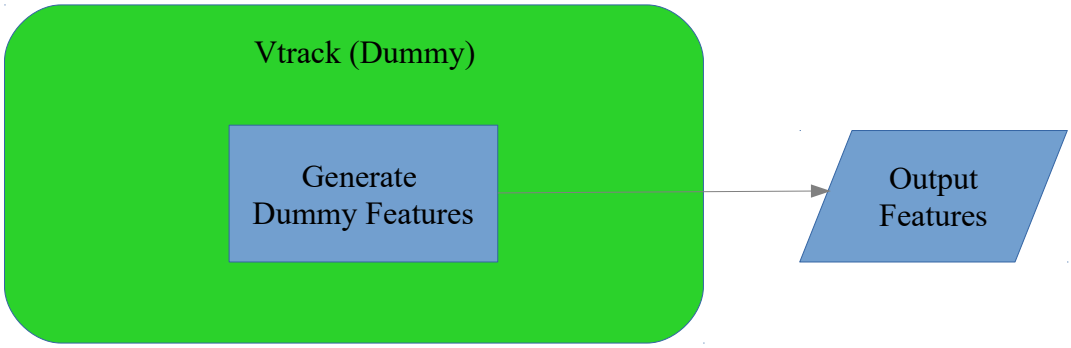
3.4.3 PixelPipe + Motion Estimation + Feature Maintenance

Vtrack will run all the components together (pixel pipe, optical flow, feature maintenance).



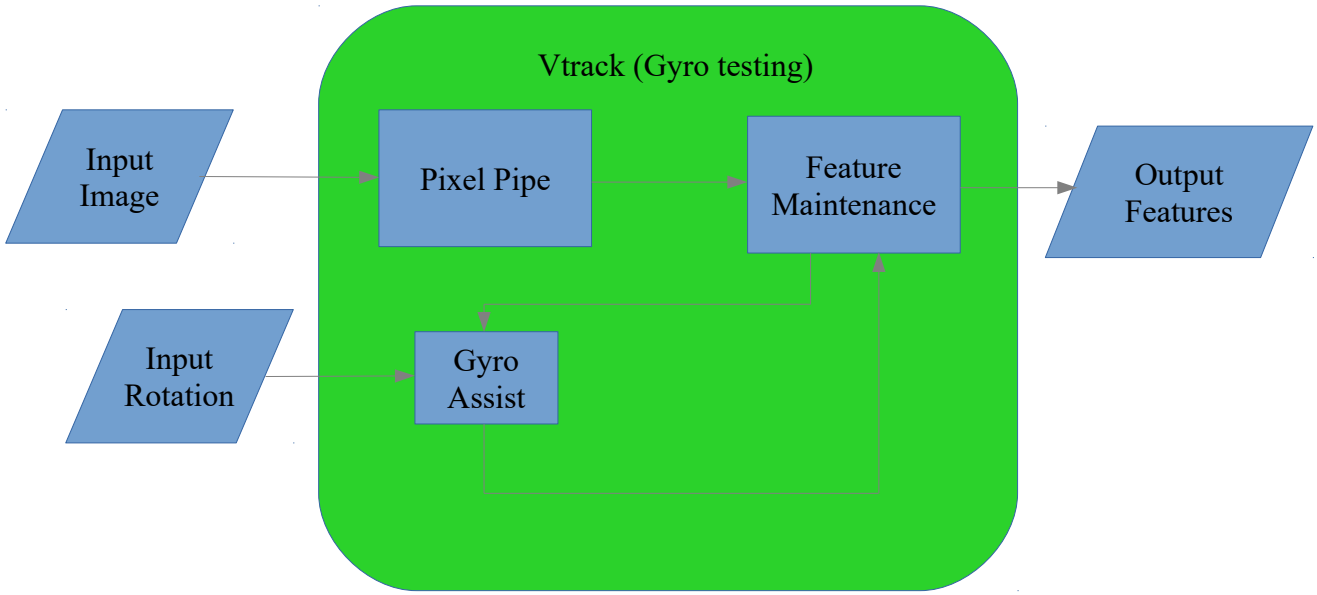
3.4.4 Dummy Mode

Vtrack will generate dummy features in the form of a grid to test the dataflow (Myriad-PC host data transmission check).



3.4.5 Gyro Testing Mode

Disables optical flow and tracks the features based on the gyro assist only. Used to check if the gyro samples were correctly integrated and/or camera parameters are correctly set.



3.5 Configuration Presets

There are a few preset configuration parameter settings that exist for ease of integration of vTrack.

3.5.1 Algorithm Configuration

- **`vPipeInitDefaultAlgoCfg`: This is the default configuration that is used for vTrack**

Parameter	Value
General vTrack cfg	
Input Frame Spec	Passed in as argument
Output Type	Pixel Pipe + Tracking
Pixel Pipe cfg	
Num Pyramid levels	Based on frame size
Harris Init threshold	10000000
Corner Alg type	Harris with Sobel for gradients (best[by score] N features picked)
Pyramid Alg on/off	ON

Parameter	Value
Cell config	4x4 cells
Target features/frame	320
Max features/frame	480
Gyro Assist cfg	–
Cam properties	Centre: (312.417, 245.933), FOV: 45
Optical Flow cfg	–
Window size	11x11
Max iterations	9
epsilon	0.01
Alg Type	LK optical flow

- **`VpipeInitDefaultAlgoShiTomasiCfg`: This configuration is the default one used when using Shi-Tomasi for feature generation (Myriad X only)**

Parameter	Value
General vTrack cfg	–
Input Frame Spec	Passed in as argument
Output Type	Pixel Pipe + Tracking
Pixel Pipe cfg	–
Num Pyramid levels	Based on frame size
Harris Init threshold	2000
Corner Alg type	Shi-Tomasi with 1D differentiator for gradient (best[by score] N features picked)
Pyramid Alg on/off	ON
Cell config	4x4 cells
Target features/frame	320
Max features/frame	480
Gyro Assist cfg	–
Cam properties	Centre: (312.417, 245.933), FOV: 45
Optical Flow cfg	–
Window size	11x11
Max iterations	9
epsilon	0.01
Alg Type	LK optical flow

- **`VpipeInitDefaultAlgoMESTCfg`: This configuration is the default one used when using the MEST hardware block for motion estimation (Myriad X only)**

Parameter	Value
General vTrack cfg	–
Input Frame Spec	Passed in as argument
Output Type	Tracking
Pixel Pipe cfg	–
Num Pyramid levels	Based on frame size
Harris Init threshold	10000000
Corner Alg type	Harris with sobel for gradient (first N features picked)
Pyramid Alg on/off	OFF
Cell config	4x4 cells
Target features/frame	320
Max features/frame	640 x 480 / 4
Gyro Assist cfg	–
Cam properties	Centre: (312.417, 245.933), FOV: 45
Optical Flow cfg	–
Window size	not used
Max iterations	not used
epsilon	not used
Alg Type	MEST HW block

3.5.2 Runtime Parameter Configuration

Parameter	Value
Pixel Pipe cfg	–
Corner thresholds	0 – Thresholds dynamically adjusted from initial value
Threshold decrease velocity	0.9
Threshold increase velocity	1.1
Maximum threshold	1000000000000000.0
Minimum threshold	1000000.0
Feature maintenance cfg	–
Lost feature error	100

Parameter	Value
Min feat distance squared	1
Harris threshold old features	100.0

3.6 Target KPIs

Relevant metrics for assessing this solution are:

- Average number of tracked features.
- Feature persistence – measured via a histogram of Feature track lengths.
- % inliers – number of feature tracks with globally consistent motion on a static scene (TBI).
- % Coverage – spatial distribution of features across scene area.
- Latency in ms.
- Algorithm processing time.

4 Implementation Description

4.1 Overview

Vtrack is implemented primarily on SHAVE processors with Leon control. It also uses up to 3 SIPP filters (Harris, MEST, Minmax) depending on the configuration/platform.

4.2 Memory Usage

vTrack allocates memory from the following buffers that need to be configured. The sizes of the buffers are dependent on image size and maximum number of features to track. The sizes in the table below show the sufficient sizes for the stated image sizes when tracking a maximum of 1024 features per frame

Buffer name	Location	640x480		1280x720		Description
		Type	Size	Type	Size	
vpCmxBuffs	CMX	t_vpCmxInternals	93KB	t_vpCmxInternals	93KB	Feature and metadata buffers
vpCmxHeap	CMX	t_vpVgaCmxHeap	59KB	t_vp720CmxHeap	45.9KB	Lower n images in the image pyramid
vpDdrHeap	DDR	t_vpVgaDdrHeap	800KB	t_vp720DdrHeap	2.41MB	The rest of the image pyramid
vpPpBuffs	CMX	t_vpVgaPpBuffs	25.6KB	t_vp720PpBuffs	40KB	Buffer for pixel pipe objects
vpPpCmxBuffs	CMX	t_vpPpCmxBuffs	128B	t_vpPpCmxBuffs	128B	Pixel pipe dma descriptors

In addition to the buffers mentioned above, when a shave is started, the code for the application it's running is loaded from DMA and the data is dynamically loaded into CMX. There are 5 possible shave modules. Vtrack could use all or some of these modules depending on the configuration. Also, depending on the configuration, some of the shave modules below could be running on multiple shaves. See section 4.3 for more information on shave usage.

Shave Module Name	Code Size	Data Size	Read-only Data Size	Description
pixelPipe	14352B	8B	957B	Pixel pipe master SHAVE.
pp_corners	859B	0B	226B	Pixel pipe corner detector.
pp_gauss	2304B	0B	226B	Pixel pipe Gaussian filter.
of	19449B	36B	308B	Optical flow SHAVE.
fm	3963B	32B	923B	Feature maintenance.

4.3 Shave Usage

The algorithm is highly configurable. The number of shaves can vary greatly depending on the configuration.

As the vTrack modules which are using shaves are not running in parallel, the same shaves can be used for multiple modules. In the chapters below, the optimal shave configuration is described for each module.

By default, the vTrack application is using 3 shaves. However, only during the pixelPipe module we use all 3 shaves. The optical flow module is using 2 shaves, the feature maintenance is using 1 shave by default.

4.3.1 PixelPipe

The pixelPipe component needs at least 1 shave to execute the pixelPipe scheduler. The scheduler will create two types of slave tasks (the gaussian pyramid creation and feature filtering). It is possible to run everything on the scheduling shave, but since each task type is an independent task, they can be given to multiple shaves running in parallel.

Task types:

- Master shave: executes the pixel pipe scheduler. Creates tasks for the slave SHAVEs.
- Corner shave: executes the non maxima suppression that follows the Harris filter.
- Gauss shave: creates the image pyramid which is to be used for optical flow.

If the algorithm has only one input image, there will be two tasks which can be executed in parallel. This means that for maximum pixelPipe performance three shaves should be used : 1 for the scheduler, 1 for creating the Gaussian pyramid and 1 for corner filtering. In this case there is no point to configure more than three shaves for pixelPipe.

In case of multiple input images (n – number of input images), we will have n * 2 tasks which can be processed in parallel. This means that the maximum number of shaves which should be used in this case is 2 * n + 1 shaves: n shaves for Gauss, n shaves for corner filtering, and 1 shave for the scheduler. As the gaussian processing is much faster than the corner filtering, the optimal shave configuration is as follows: n / 2 shaves for gaussian pyramid, n shaves for corner filtering and 1 shave for the scheduler.

4.3.1.1 PixelPipe on Myriad X

On Myriad X, there is a specialised SIPP hardware filter for non maxima suppression. Therefore, there is no corner shave used.

If the motion estimation mode is set to MEST rather than Optical flow, an image pyramid is not needed. Therefore, there would be no gauss shave. In that case, pixelPipe will only use 1 SHAVE for scheduling.

4.3.2 OpticalFlow

The number of shaves for the opticalFlow module is configurable. To minimize the latency, the application should use as many shaves as it can. This will not cause a consumption increase, because the shave power islands are turned off, when the shaves finish their processing.

The optimal number of shaves in the case of this module is dependent on the number of frame cells (by default = 16). If the number of frame cells is n, the number of shaves used should be **k**, where **n % k = 0**. If this is not true, there will be a time slot, where n % k shaves are processing and the rest of the shaves are turned off.

4.3.2.1 MEST on Myriad X

In MEST mode, vTrack uses the SIPP filter for MEST. Therefore, the optical flow shaves are not used.

4.3.3 Feature Maintenance

The number of shaves of the feature maintenance module is not configurable. It is fixed to 1. This is because the runtime of this module is very short.

4.4 SIPP Filter Usage

Vtrack can use a number of SIPP filter depending on configuration and platform.

4.4.1 Harris Filter

Harris corner filter is used for getting Harris or Shi-Tomasi scores for each pixel in the image. The scores are then passed for thresholding and non maxima suppression to pick out the best features in the image.

To use the Harris filter, pixelPipe populates a circular buffer of image lines. The Harris filter is called line-wise and circles through the lines in the circular buffer. The line-wise operation allows us to more efficiently use each fetch of a line (once fetched, it's processed by all the parts of pixelPipe).

The Harris SIPP filter is configurable. However, in its usage within vTrack, some of the parameters are determined from the inputs provided to vTrack and others are set within vTrack.

Parameter	Myriad 2	Myriad X
Harris Kernel size	5	5
Circular line buffer size	6	6
Line Stride, Plane Stride	From input spec	From input spec
Output line buffer size	4	4
Harris K parameter	0.04	0.04
Output type	fp32	fp16
Fp16 scaling subtrahend	N/A	16 (Shi-Tomasi) / 24 (Harris)
Use Shi-Tomasi	N/A	From input config
Use Sobel	N/A	From input config

4.4.2 MinMax filter (Myriad X only)

On the Myriad X, rather than thresholding and performing non maxima suppression on a slave shave, there is a specialized hardware filter for the task. Like the Harris filter, it performs the task line by line. The MinMax filter requires an fp16 input. As a result, this induces the requirement for he fp16 configuration in the Harris filter on the Myriad X.

The Threshold used by the minmax hardware filter is fixed for all cells. Therefore, we use the lowest threshold from all the cells in the minmax filter. The other cells are further filtered in the feature maintenance with the correct threshold.

Like the Harris SIPP filter, this is configurable. The configuration used for vTrack is as follows:

Parameter	Myriad X
Circular line buffer size	4
Line Stride, Plane Stride	From input spec
Output line buffer size	3
Threshold	Adaptive – Lowest of all cells (see 3.3.3.5)
Extrema Threshold	0

4.4.3 MEST filter (Myriad X only)

A dense motion estimation filter is available on the Myriad X that can be used in the Motion Estimation stage instead of Optical Flow. This filter performs an 8x8 block matching using SAD performed on hardware.

The MEST filter (unlike the previous two) cannot be called on a line by line basis. A single run is called on the entire frame and the results are output.

The configuration used for the MEST filter is as follows:

Parameter	Myriad X
Image size	From input spec
Base image	Previous image ptr
Source image	Image ptr
Motion vector resolution	Integer

4.5 API

There are two possible variants of the API that can be used for vTrack. The recommended API to use is the FLIC vTrack API. In this case, vTrack is wrapped as a plugin for the FLIC framework. This allows for easy integration in multi-threadded FLIC pipelines and simple memory diagnosis features, etc. FLIC interface is currently supported for the LEON.

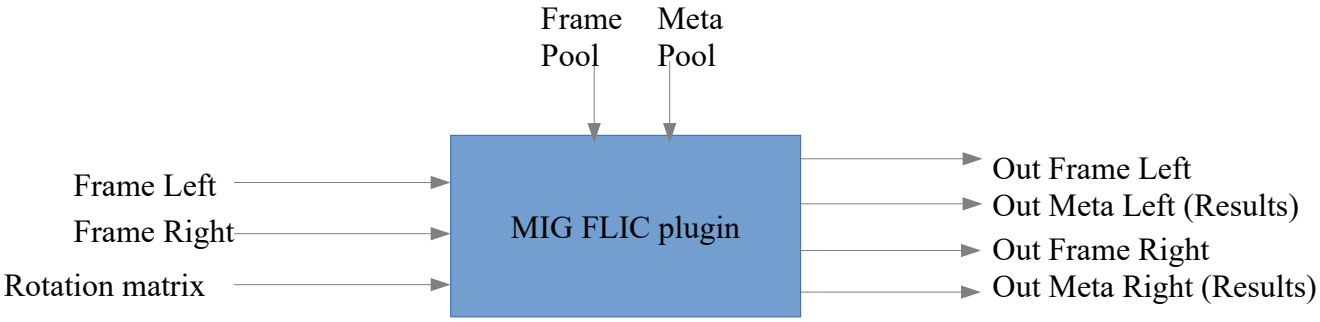
It is possible to use vTrack through a non-FLIC interface. This is described below. This is required to be used on PC as FLIC currently does not run on PC.

The public interface for vTrack is contained within the following header files:

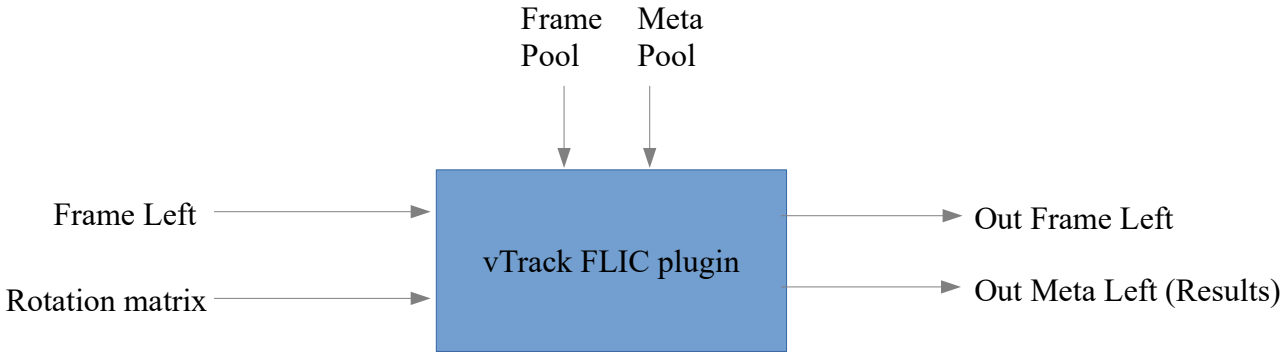
```
├── flicPlgVtrack.h
├── global_constants.h
├── vPipe.h
├── vPipePublicTypes.h
├── vTrack.h
└── vTrackOutput.h
```

4.5.1 FLIC Interface (Recommended)

The vTrack FLIC plugin implements a generic MIG plugin shown in the diagram below:



The vTrack flic plugin currently implements a single frame version of the above (only using frameLeft and metaLeft).



Using a vTrack plugin in a FLIC pipeline simply involves creating an instance of the vTrack FLIC plugin and connecting it's senders/receivers to the other plugins in the pipeline.

The configuration parameters/buffer pointers are provided to vTrack on creation (Interface here flicPlgVtrack.h).

If plgFlicVtrack is being constructed dynamically, after the config objects are created, the constructor can be called passing as arguments the pointers to the config objects.

```
plgFlicVTrack(
    t_vPipeResourceCfg* vResourceCfg,  // Contains (1)Shave
configuration, (2)Memory config - buffer ptrs
                                   //  (3)Cache config (partition
numbers),  (4) fifo config
    volatile t_vPipeMode vp_mode,         // vTrack run mode
    t_vPipeAlgoCfg* _algoCfg,                  // Algorithm
configuration params
    t_vPipeParams* _params,                   // Some config params +
buffer that stores harris thresholds
    customProcCtx _ctx,                              // frame rate
divider, cam count and drop condition
    uint8_t _outputFrameDisable,             // don't show output
    bool _gyroAssist                                      // use input
rotation matrix
)
```

If the vTrack object needs to be created before the config objects are created, the default constructor is called and Init then must be called before using the vTrack object.

In order to set default values for _ algoCfg and _params, the functions from vPipe.h can be used:

```
void vPipeInitDefaultAlgoCfg(t_vPipeAlgoCfg* algoCfg, frameSpec*
inputFrameSpec) ;
void vPipeInitDefaultAlgoShiTomasiCfg(t_vPipeAlgoCfg* algoCfg,
frameSpec* inputFrameSpec);
void vPipeInitDefaultAlgoMESTCfg(t_vPipeAlgoCfg* algoCfg, frameSpec*
inputFrameSpec);
void vPipeInitDefaultHarrisRuntimeParameters(t_vPipeParams* params);
The method to get the result configuration (including result buffer
size) is below. This should only be called once vPipe has been
initialised (vPipeInit):
t_vTrackResultConfig getResultConfig()
Sample FLIC connections to be used for vTrack are shown below as well:
            plgPoolMeta.out.Link(&vTrack->emptyMeta);
            plgPoolOut.out.Link(&vTrack->emptyFrame);
            inputStream->inputFrame.Link(&vTrack->frameLeft);
            vTrack->frameLeftOut.Link(&resultsStream->inFrame);
            vTrack->metaLeft.Link(&resultsStream->inMeta);
```

Once the above is configured, vTrack will run when the pipeline starts.

The output of vTrack can be obtained from the metaLeft sender. The metaLeft is populated with a buffer of vTrack results. The size of the buffer is variable based on the configuration parameters. Specifically, the size of the output buffer is based on the maximum number of features allowed (maxNumFeatures).

A class is provided (class vTrackBulkResult) in vTrackOutput.h to parse the vTrack results buffer into the t_vTrackResultSF struct

```
vTrackBulkResult bulkResult;
bulkResult.setAddrAndPtrs((uint8_t*) meta->customMetadata);
t_vTrackResultSF* result;
t_vTrackResultConfig* config;
bulkResult.getConfigAndResults(&config, &result); // Args can be null
if you don't need the config or results
```

4.5.2 vPipe Interface

The vPipe interface is shown in vPipe.h and described by the public functions of Vpipe. In this case, the frame pool and meta pool are managed by the user (as opposed to the case of the *FLIC* interface).

In this case, *FLIC* does not manage the input image and metadata pools, so vTrack will use buffers provided to it by the application.

The input image is to be provided to the vPipeRun function as a pointer to a frameBuffer object.

The result buffer is to be provided to the vPipeRun function as a pointer to a vTrackBulkResult object. The vTrackBulkResult object can be populated using the helper methods (as shown below).

```
t_vTrackResultConfig resultCfg = vPipeGetResultConfig(); // only after
vPipeInit
uint32_t buffSize = bulkResult.setConfigAndPtrs(resultCfg, buffer);
// &bulkResult can be passed to vPipeRun
```

After running vPipeRun, the results are to be parsed as mentioned in the previous section (4.5.1).

4.6 Algorithm Validation

The validation is done using the moviVtrackEval application. This application will receive the algorithm output, together with the images, over USB from Myriad. Test scenario is with a device being moved side to side laterally, with a small portion of the scene persistently in view.

The PC application calculates two types of real-time graphs:

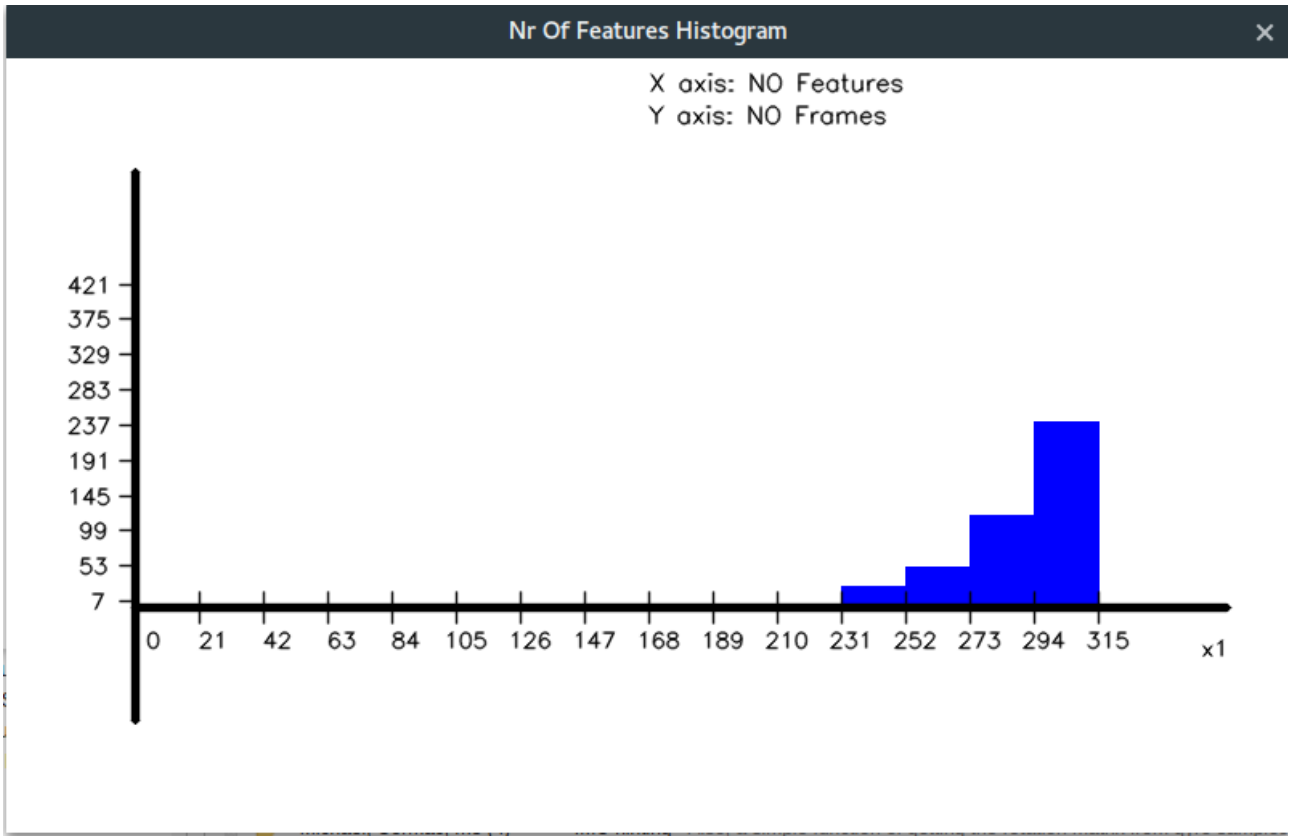
1. Histograms calculated using the last N (configurable) frames: The histogram will be built incrementally in each case. For the first N frames we add a new value to the histogram. After this, we remove the oldest value and add a new one at each frame. X axis will show the measured value (different in each case), while Y axis will show the frequency (number of occurrences) of the Ox value.
2. KPI values over time. Simple function over time. X axis showing the time in frames, Y axis showing the measured value.

4.6.1 Histograms using the last N frames

4.6.1.1 Number of Features histogram

Shows a histogram of the feature count for the last N frames:

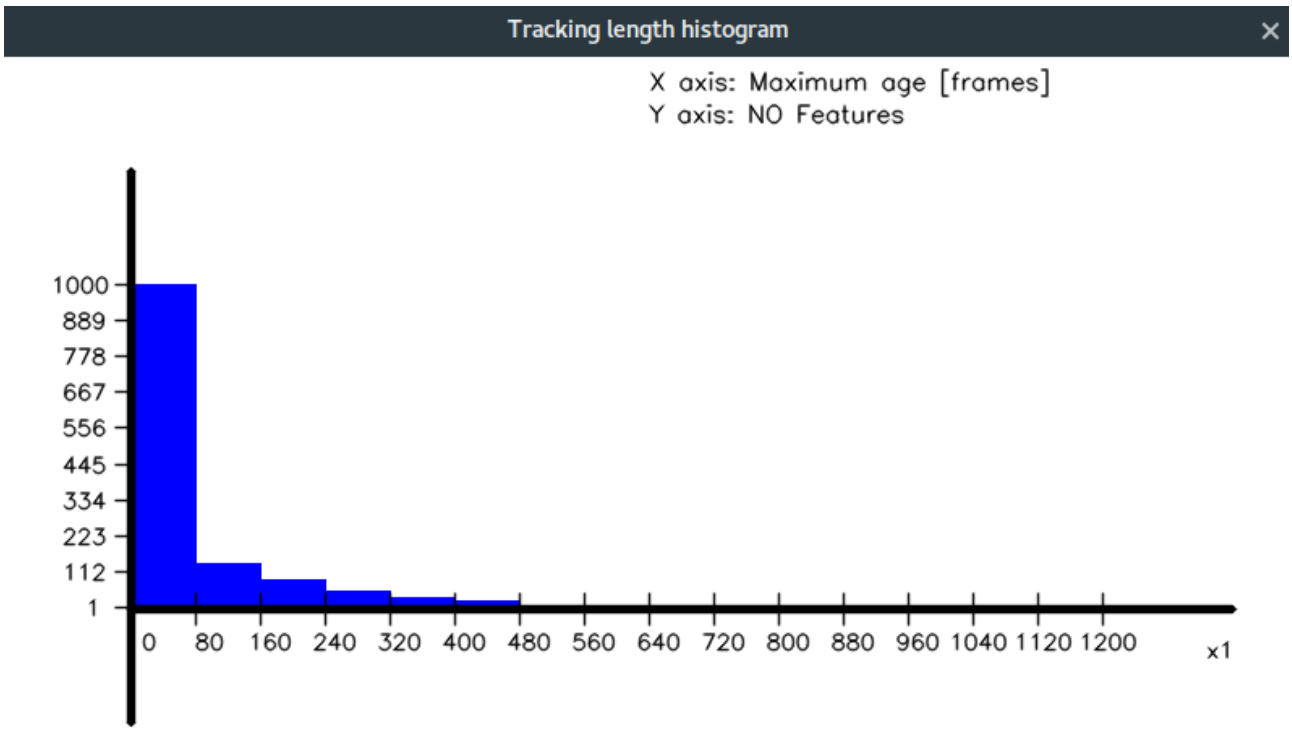
- X: feature count
- Y: frequency – number of frames



4.6.2 Feature Persistence

Shows the feature tracking length. The track length of a point is considered to be the value of feature age in the last frame where it was found. Note that this histogram will show pretty short track lengths with a still camera, because most of the features will not be lost, so they will not be shown on this histogram

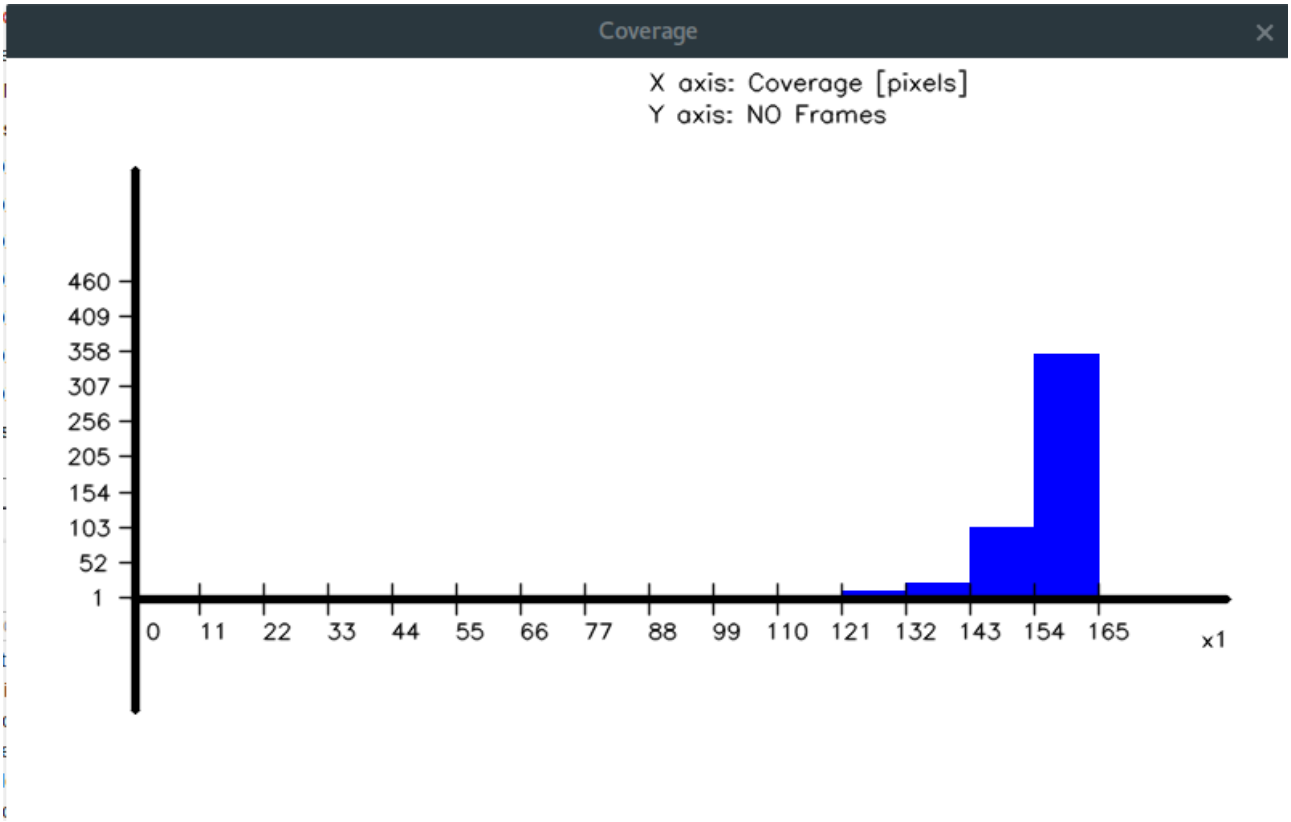
- X: tracking length
- Y: number of features



4.6.2.1 Feature Coverage histogram

Shows the feature coverage on the last N frames. The coverage calculation is described in this Section:

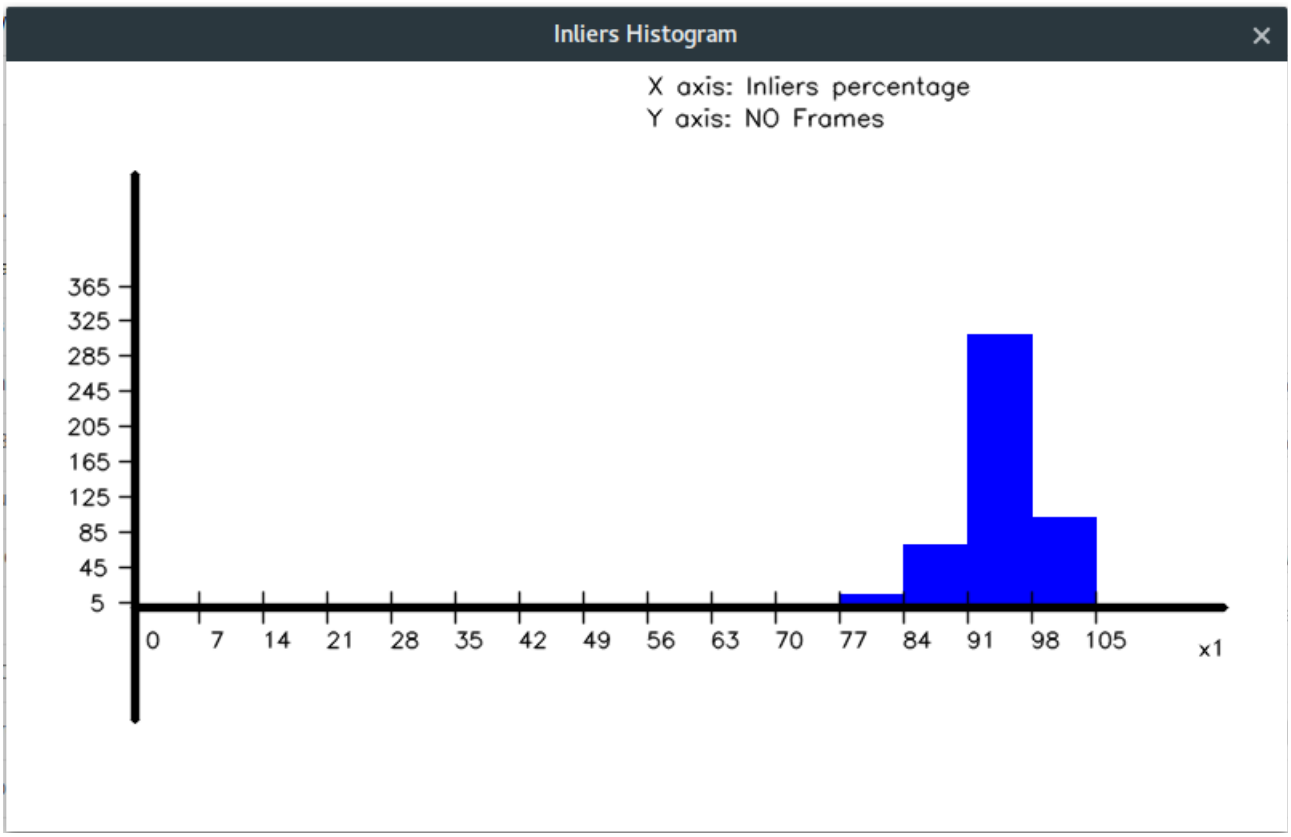
- X: coverage value in pixels
- Y: number of frames



4.6.2.2 Inlier Percentage

Histogram of inliers percentage. The number of features with movement consistent with global motion. Using OpenCV with Ransac, Epipolar distance and 0.5 pixel distance threshold for inliers.

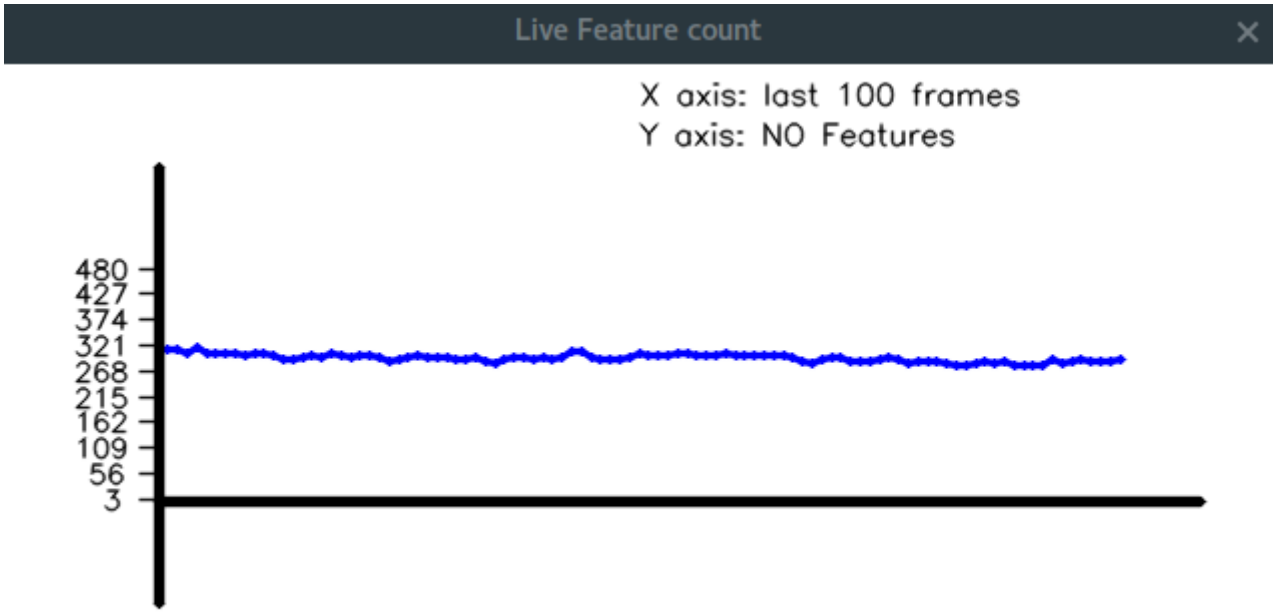
- X: inliers percentage
- Y: number of frames



4.6.3 KPI values over time

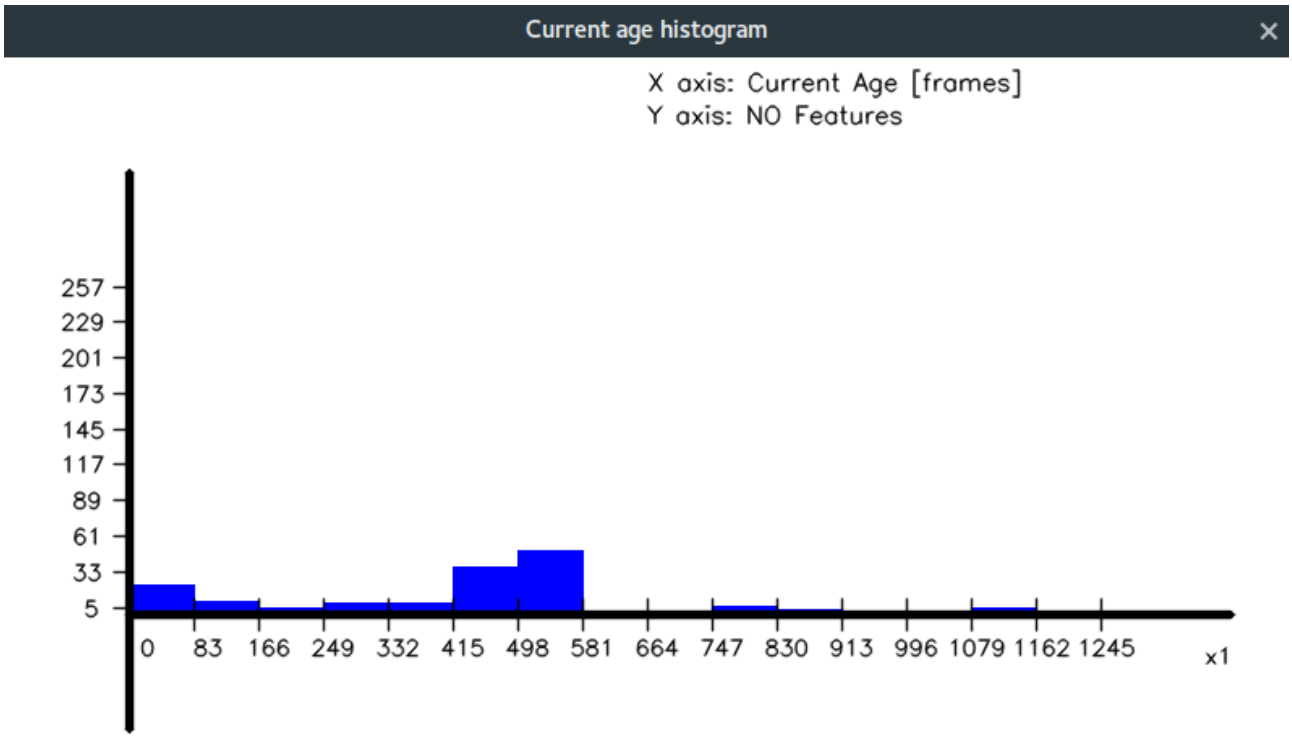
4.6.3.1 Feature number

Showing a function of feature counts over the last frames.



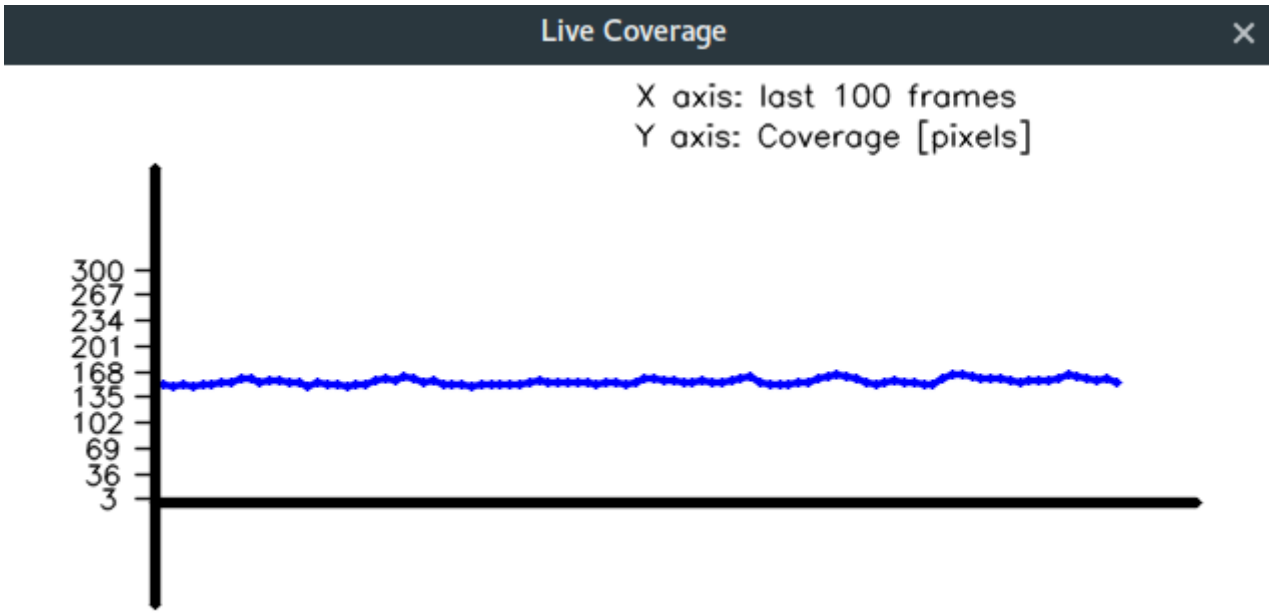
4.6.3.2 Feature age histogram

Showing a histogram of the feature ages for the current image. The feature age is the number of frames where the feature was found and tracked.



4.6.3.3 Coverage on current image

Showing the coverage value for the last images. This coverage value is used to construct the histogram described in this section. The value is in pixels (distance). The method is described in reference [3].



4.6.3.4 Algorithm Latency

Latency is a very important factor in tracking applications. This graph shows two functions: Myriad overall latency and vTrack processing time.

The overall latency number includes the time spent from end of exposure to the moment when the Myriad chip starts to send the data to PC. This includes:

- Readout time from the camera.
- Image conversion (from 10bpp to 8 bpp).
- vTrack processing time.
- Any application overhead.
- And the time while Myriad waits for the PC to trigger the read.

This number does not include the USB readout which is dependent on the PC and the USB version used.

In normal case, the latency is: Readout time (~10ms) + Conversion time (1.7 ms) + vTrack time (5-12 ms, depending on the number of features). This means that the latency in the evaluation release is ~20ms.

➢ **Note on latency reduction**

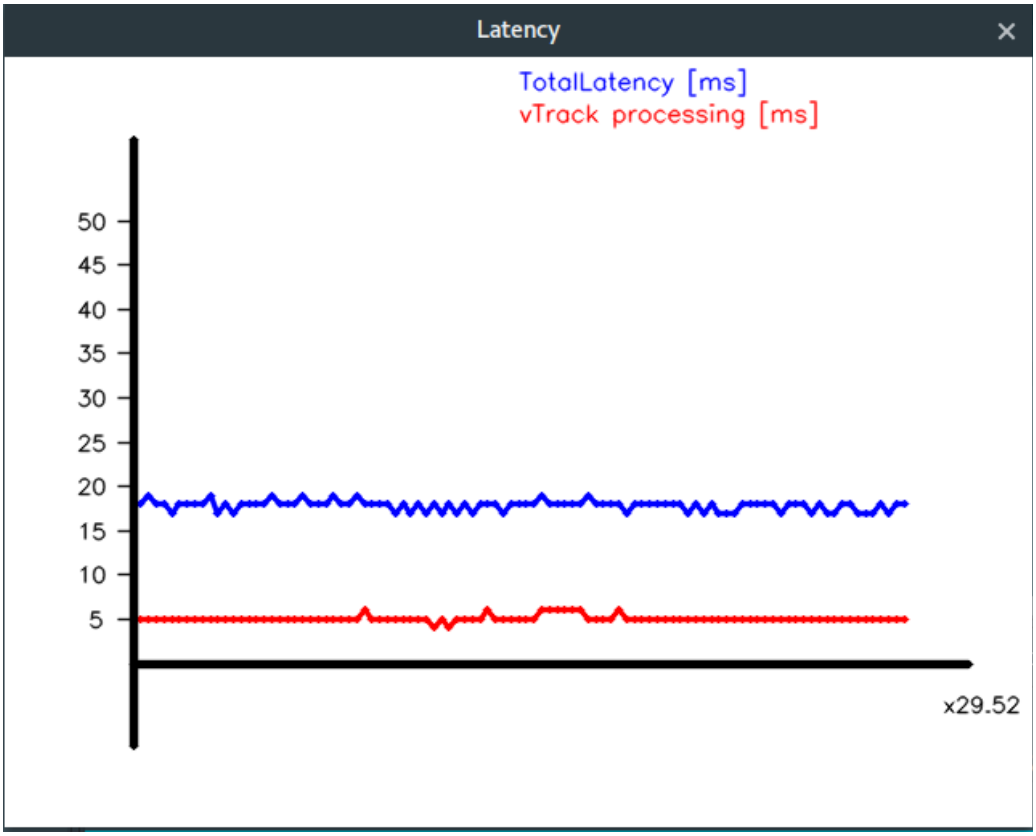
The evaluation release does not contain the absolutely lowest latency possible in an integrated and well-tuned system. The ~20ms latency of the evaluation release can be optimized as follows:

- The camera readout time can be optimized.

- The conversion time (~1.7ms) can be eliminated. Certain cameras can be configured to output u8 data. There is also a possibility to do the conversions with the Rx filters when receiving the frame on Myriad.
- First module of vTrack (pixelPipe) can already operate when the first line of the frame arrived. This means that we can completely overlap the pixelPipe processing with the readout time (~3ms).
- The pyramidal optical flow is running on 2 shaves only. As the algorithm operates on points, this can be split over any number of shaves. The speedup is linear, power numbers remain the same, the only penalty being blocking more shaves for the time-slice of OF.
- The latency numbers above are not including the USB readout. The USB readout time can be minimized by using USB3. Also, usually the latency of the features matter, not the latency of the actual image data. Sending the features before the frame can reduce the transmission time.

➢ **Note on latency jitter**

If the PC has high load, and is not reading the images in time, the Myriad chip will drop frames. However, it always needs to keep the latest frame until a new one arrives. This means, that depending on PC load, the latency number can have a variation of (1/Frequency) ms (i.e. 16ms @ 60FPS).



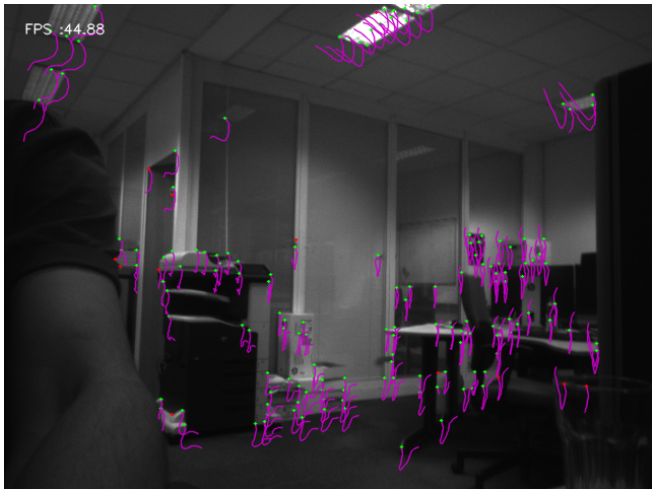
4.6.3.5 Algorithm visualization

The PC visualization tool displays the features along with drawing of the history of that feature. The images below show the visualization in two cases: moving object in the image and moving camera. The Features

are drawn as green dots if they're tracked features and red dots if they're first time features

The tracking length is configurable in the drawing code that is included with the release. It is configurable via the MAX_FEATURE_TRACK variable in `featuresDrawer.cpp`.

Moving object



Moving Camera

4.7 Functional Test cases

The following software tests cases exist to validate the elements of the component:

Testcase	Description
T1_pixelPipe	gaussian image and corners generated by the pixelPipe component.
T2_vPipe_PP_FM	pixel Pipe and Feature maintenance.
T3_dummyVpipe	vTrack in dummy mode – checks if the features are coming out correctly.
T4_vPipe_all	Run vTrack multiple times on a shifted image.
T5_vPipe_TwoImage	Run vtrack on images from slightly different perspectives.
T6_vPipe_allPixels	Run vTrack on sevaral points from each part of the image.
T7_vPipe_all_HD	HD version of T4.
T9_vPipe_Threaded	Multi-threadded version of T4.
T10_vPipe_mest	Run vTrack in Mest mode and check consistency of the tracking.

5 Performance assessment

5.1 Runtime Performance

This diagram below shows the runtime of each vTrack component. The measurements show runtime for tracking an average number of 310 features. These measurements show average runtime over 500 runs. With heavy movement, OF processing time can increase by 1-2 ms.

5.1.1 Myriad 2 (VGA)

Stage	Interval / ms	Timeline profile (0.5 ms slots, 'X' is active)	Comments
Pixel Pipe	3.484	XXXXXXX----------	3 Shaves + Harris SIPP
• Master	0.771	XXXXXXX----------	Scheduler + DMA (Shave 0)
• Gauss task	2.495	XXXXXXX----------	Gauss Pyramid Generation (Shave 1)
• Corner task	2.812	XXXXXXX----------	Generate Features (Shave 2 + Harris SIPP)
Optical Flow	5.535	-------XXXXXXXXXX--	2 Shaves
• OpticalFlow 1	5.535	-------XXXXXXXXXX--	Optical Flow (Shave 0)
• OpticalFlow 2	5.535	-------XXXXXXXXXX--	Optical Flow (Shave 1)
Feature Maintenance	0.844	----------------X-	Feature Maintenance (Shave 0)
Misc	0.153	----------------X	Miscellaneous (Leon)
Total	10.016		3 Shaves + Harris

5.1.2 Myriad 2 (720p)

Stage	Interval / ms	Timeline profile (1 ms slots, 'X' is active)	Comments
Pixel Pipe	7.232	XXXXXXX---------	3 Shaves + Harris SIPP
• Master	1.259	XXXXXXX---------	Scheduler + DMA (Shave 0)
• Gauss task	4.726	XXXXXXX---------	Gauss Pyramid Generation (Shave 1)
• Corner task	6.143	XXXXXXX---------	Generate Features (Shave 2 + Harris SIPP)
Optical Flow	7.872	-------XXXXXXXX-	2 Shaves
• OpticalFlow 1	7.872	-------XXXXXXXX-	Optical Flow (Shave 0)

Stage	Interval / ms	Timeline profile (1 ms slots, 'X' is active)	Comments
• OpticalFlow 2	7.872	-------XXXXXXXX-	Optical Flow (Shave 1)
Feature Maintenance	0.884	---------------X	Feature Maintenance (Shave 0)
Misc	0.157		Miscellaneous (Leon)
Total	16.145		3 Shaves + Harris

5.1.3 Myriad X (VGA)

Stage	Interval / ms	Timeline profile (0.5 ms slots, 'X' is active)	Comments
Pixel Pipe	4.023	XXXXXXXX----------------	2 Shaves + Harris SIPP + Minmax SIPP
• Master	1.345	XXXXXXXX----------------	Scheduler + DMA (Shave 0)
• Gauss task	3.532	XXXXXXXX----------------	Gauss Pyramid Generation (Shave 1)
• Corner task	2.538	XXXXXXXX----------------	Generate Features (Harris SIPP + Minmax SIPP)
Optical Flow	6.637	--------XXXXXXXXXXXXX---	2 Shaves
• OpticalFlow 1	6.637	--------XXXXXXXXXXXXX---	Optical Flow (Shave 0)
• OpticalFlow 2	6.637	--------XXXXXXXXXXXXX---	Optical Flow (Shave 1)
Feature Maintenance	0.996	----------------------XX	Feature Maintenance (Shave 0)
Misc	0.674		Miscellaneous (Leon)
Total	12.330		2 Shaves + Harris & Minmax

5.1.4 Myriad X (720p)

Stage	Interval / ms	Timeline profile (1 ms slots, 'X' is active)	Comments
Pixel Pipe	7.414	XXXXXXX----------	2 Shaves + Harris SIPP + Minmax SIPP
• Master	1.832	XXXXXXX----------	Scheduler + DMA (Shave 0)

Stage	Interval / ms	Timeline profile (1 ms slots, 'X' is active)	Comments
• Gauss task	6.155	XXXXXXX----------	Gauss Pyramid Generation (Shave 1)
• Corner task	5.243	XXXXXXX----------	Generate Features (Harris SIPP + Minmax SIPP)
Optical Flow	9.318	-------XXXXXXXXX-	2 Shaves
• OpticalFlow 1	9.318	-------XXXXXXXXX-	Optical Flow (Shave 0)
• OpticalFlow 2	9.318	-------XXXXXXXXX-	Optical Flow (Shave 1)
Feature Maintenance	1.058	----------------X	Feature Maintenance (Shave 0)
Misc	0.609		Miscellaneous (Leon)
Total	18.399		2 Shaves + Harris & Minmax

5.1.5 Myriad X - Mest (VGA) [1024 features]

Stage	Interval / ms	Timeline profile (0.5 ms slots, 'X' is active)	Comments
Pixel Pipe	2.386	XXXXX-----	1 Shave + Harris SIPP + Minmax SIPP
• Master	1.345	XXXXX-----	Scheduler + DMA (Shave 0)
• Corner task	2.386	XXXXX-----	Generate Features (Harris SIPP + Minmax SIPP)
MEST	1.921	-----XXXX-	MotEst SIPP
Feature Maintenance	0.658	---------X	Feature Maintenance (Shave 0)
Misc	0.660		Miscellaneous (Leon)
Total	5.625		1 Shave + Harris, MinMax & MotEst

5.1.6 Myriad X – Mest (720p) [1024 features]

Stage	Interval / ms	Timeline profile (0.5 ms slots, 'X' is active)	Comments
Pixel Pipe	4.760	XXXXXXXXX---------	1 Shave + Harris SIPP +

Stage	Interval / ms	Timeline profile (0.5 ms slots, 'X' is active)	Comments
			Minmax SIPP
• Master	1.832	XXXXXXXXX---------	Scheduler + DMA (Shave 0)
• Corner task	4.760	XXXXXXXXX---------	Generate Features (Harris SIPP + Minmax SIPP)
MEST	3.315	---------XXXXXXX--	MotEst SIPP
Feature Maintenance	0.674	----------------XX	Feature Maintenance (Shave 0)
Misc	0.600		Miscellaneous (Leon)
Total	9.350		1 Shave + Harris, MinMax & MotEst

5.2 Power Measurement

The power measurement below was taken with the configuration described in section 5.1.1.

Power numbers are aggregate numbers for all rails required by the MA2150 package, including stacked DDR.

Idle power with VGA camera streaming at @30Hz IMU Sampling Enabled:	tbc
Additional power for vTrack processing enabled at 30Hz:	45-55 mW (static - moving)