Extended Summaries: Optical Flow Algorithms and Hardware

Computer Architecture Project

Contents

1 Article 1: Efficient Hardware Implementation of the Horn-Schunck Algorithm

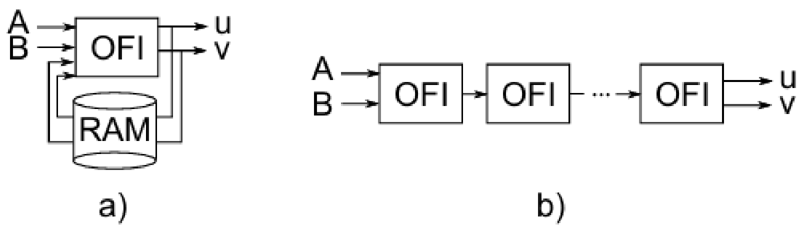
Source: Efficient Hardware Implementation of the Horn-Schunck Algorithm... (Doc AC IM)

This paper presents a high-performance hardware architecture for computing dense optical flow using the Horn-Schunck (HS) algorithm on an FPGA. The primary objective is to achieve real-time processing for Full HD video streams (1920×1080 at 60 fps). The authors propose a novel fully pipelined architecture that eliminates the bandwidth bottleneck caused by external memory access found in traditional iterative implementations.

1.1 Pipelined Dataflow Architecture

The "state-of-the-art" method for high-speed optical flow is to unroll the iterative loop into a pipeline. Data flows through a series of cascaded iteration blocks, removing the need for intermediate frame storage in external RAM. Consequently, the system only requires external memory to buffer the previous video frame needed for the temporal derivative (I_t).

Figure 1. Two ways of hardware optical flow computation: (**a**) iterative, (**b**) pipelined. OFI — single iteration of the HS algorithm, A i B — two consecutive frames from a video sequence.



1.2 Mathematical Formulation and Fixed-Point Logic

The implementation minimizes the global energy functional E. The hardware solves this iteratively using the standard update equations:

$$u_{n+1} = \bar{u}_n - \frac{I_x(I_x\bar{u}_n + I_y\bar{v}_n + I_t)}{\alpha^2 + I_x^2 + I_y^2} \tag{1}$$

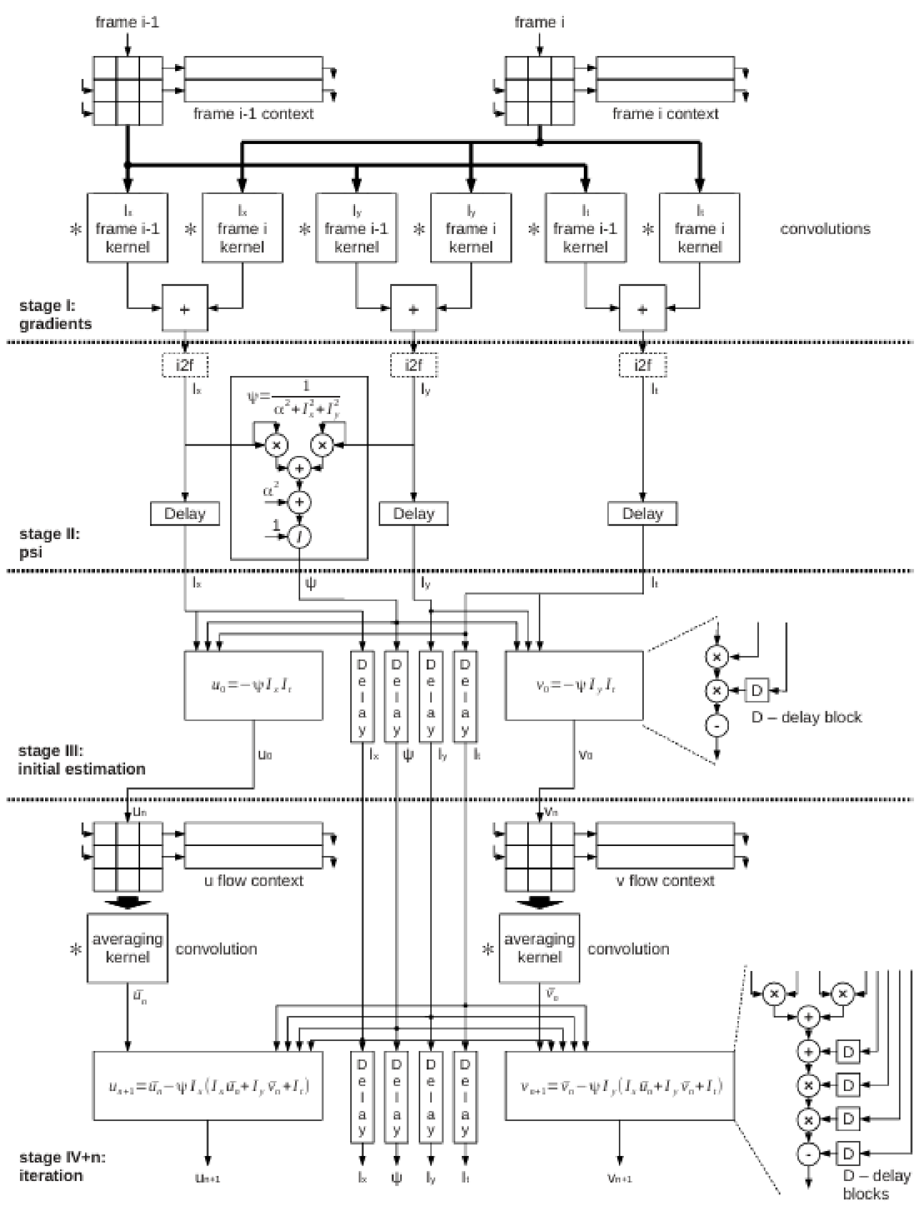
For the first stage of the pipeline, the previous flow values are assumed to be zero, simplifying the calculation to $u_0 = -\psi I_x I_t$, where ψ is a normalization factor.

To optimize FPGA resources, the design uses a specific 17-bit fixed-point representation (1 bit sign, 6 bits integer, 10 bits fractional). Research demonstrates that this offers accuracy comparable to floating-point implementations while using $4\times$ fewer FPGA resources. Reducing the fractional part below 8 bits leads to accumulating truncation errors, making the 10-bit choice critical for stability.

1.3 Kernel Simplification

The implementation supports simplified convolution masks (similar to OpenCV) rather than the original Horn-Schunck masks. The original masks sum to 12, requiring a hardware divider. The simplified masks sum to 4 (a power of two), allowing the division to be replaced by a simple bit-shift operation, significantly saving FPGA logic.

Figure 3. General block schematic of the hardware optical flow computation module.



2 Article 2: Real-Time FPGA Implementation of Multi-Scale LK and HS Algorithms for 4K Video

Source: Real-Time Efficient FPGA Implementation... (Blachut & Kryjak)

This paper presents a high-performance System-on-Chip (SoC) FPGA implementation of two classical optical flow algorithms: **Lucas-Kanade (LK)** and **Horn-Schunck (HS)**. The primary innovation is the capability to process **4K Ultra HD** (3840×2160) video streams in real-time at **60 fps**. The system uses a multi-scale pyramidal approach to detect large displacements and achieves high energy efficiency (< 6 W), making it suitable for autonomous vehicles.

2.1 Mathematical Foundations

The implementation is based on the fundamental gradient equation: $I_x u + I_y v + I_t = 0$.

Lucas-Kanade (LK): This local method solves the equation using the least-squares method in a small neighborhood. To avoid complex hardware matrix inversion, the authors implemented an explicit form based on the determinant:

$$\begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} A_{22} & -A_{12} \\ -A_{21} & A_{11} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \cdot \frac{1}{\det A} \tag{2}$$

Horn-Schunck (HS): This global method minimizes an energy functional involving a smoothness constraint. The hardware solves this iteratively using the update formulas:

$$u_{n+1} = \bar{u}_n - \frac{I_x(I_x\bar{u}_n + I_y\bar{v}_n + I_t)}{\alpha^2 + I_x^2 + I_y^2} \tag{3}$$

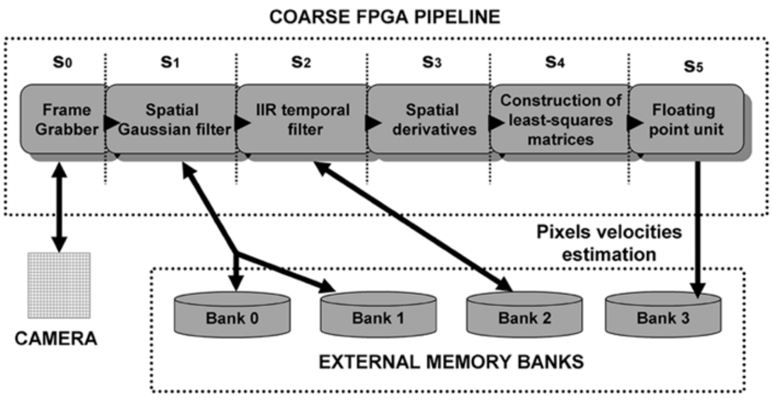


Fig. 1. Coarse pipeline processing architecture.

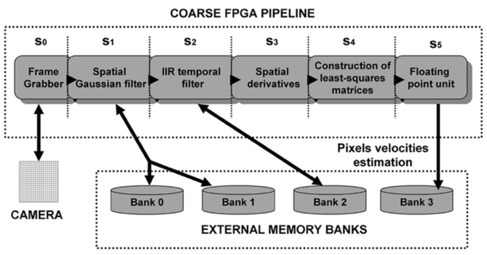


Fig. 1. Coarse pipeline processing architecture.

Figure 1: Enter Caption

Figure 2: The complete pipeline for the Lucas-Kanade implementation.

2.2 4K Hardware Architecture (Vector Format)

Processing 4K video at 60fps requires a pixel clock of nearly 600 MHz, which exceeds standard FPGA capabilities. The authors solved this by using a **vector format (4ppc)**, processing **4 pixels per clock cycle** in parallel.

- **Scale 0 (Input 4K):** Processed at 4ppc.
- **Lower Scales:** Processed at 2ppc or 1ppc to conserve resources.

This requires a complex sliding window context generation scheme to handle the parallel data stream.

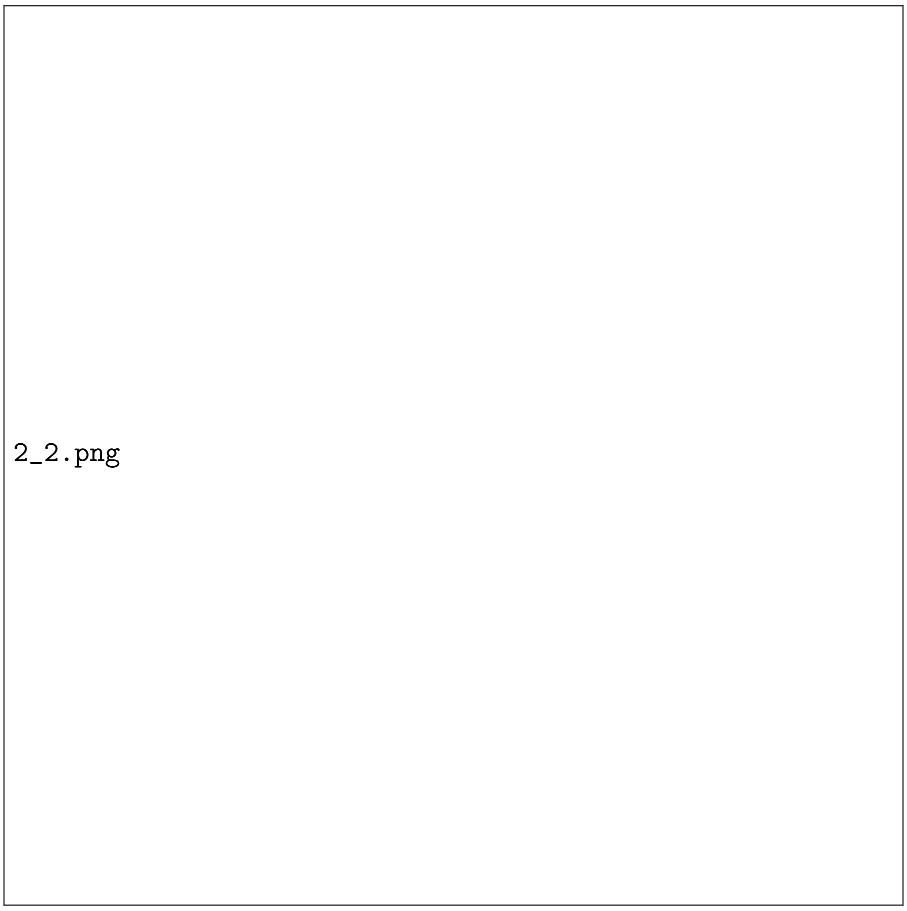


Figure 3: Detailed diagram of the multi-scale processing pipeline.

3 Article 3: Hardware Implementation of Multi-Scale Lucas-Kanade

Source: Hardware implementation of multi-scale Lucas-Kanade... (Doc 4)

This article details a full hardware pipeline implementation of the Multi-Scale Lucas-Kanade (LK) algorithm on a Xilinx Virtex-7 FPGA, targeting real-time processing of HD video. The multi-scale (pyramidal) approach is necessary because the standard LK algorithm relies on linearity assumptions that hold only for small pixel displacements.

3.1 Multi-Scale Algorithm Strategy

An image pyramid is constructed via successive subsampling. Optical flow calculation begins at the coarsest level (lowest resolution), where large real-world motions appear as small pixel displacements. This coarse flow is up-scaled and used to warp the image at the next level, allowing the hardware to iteratively refine the residual flow.

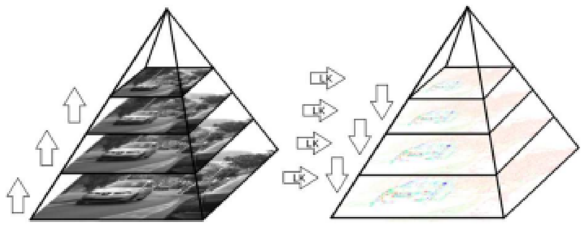


Fig. 1. A pyramid of images for one frame

3.2 Hardware Pipeline Implementation

The FPGA architecture implements a streaming pipeline. Stages include RGB-to-Grayscale conversion and Gaussian smoothing (using separated 1D masks [1 4 6 4 1]/16 for efficiency). The core computation solves the fundamental optical flow equation: $I_x u + I_y v + I_t = 0$. The system solves this via the Least Squares method over a local 5×5 window, resulting in the following matrix system solved per pixel:

$$\begin{bmatrix} \sum wI_x^2 & \sum wI_xI_y \\ \sum wI_xI_y & \sum wI_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum wI_xI_t \\ -\sum wI_yI_t \end{bmatrix} \tag{4}$$

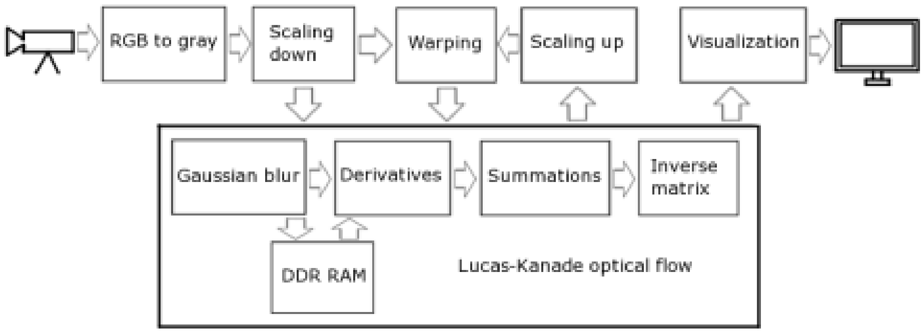


Fig. 2. Scheme of the LK computing system

The implementation successfully processes 1280x720 video at 50 fps, utilizing moderate FPGA resources (approx. 13.56% LUTs and 18% BRAM on the Virtex-7), validating the feasibility of embedded high-performance vision systems.

4 Article 4: Event-based Vision on FPGAs: A Survey

Source: Event-based vision on FPGAs: a survey (Doc 3)

This survey explores the intersection of Dynamic Vision Sensors (DVS) and FPGA processing. Unlike frame-based cameras, DVS pixels operate asynchronously, generating an event (x, y, t, p) only when intensity changes. This results in microsecond-level latency and high dynamic range, but requires novel processing architectures to handle sparse, non-matrix data.

4.1 Optical Flow for Event Data

The survey categorizes FPGA-based approaches for event-driven optical flow. A prominent method is **Block Matching on Pseudo-Frames**, where events are accumulated over short intervals to create binary activity maps, allowing the use of fast Hamming distance comparators. This approach has demonstrated latencies as low as 220 nanoseconds per event.

Alternative architectures include **Plane-Fitting** in the spatio-temporal domain and **Spiking Neural Networks (SNNs)**. SNNs are particularly well-suited for FPGAs as they process spikes natively and asynchronously, mirroring the sensor's output. The survey notes that while hardware performance is high, the field lacks standardized datasets for consistent benchmarking.

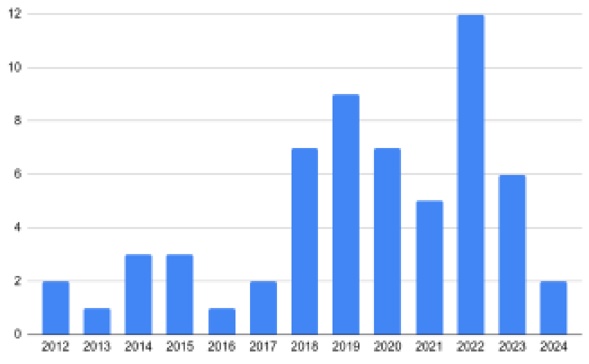


Fig. 1. Number of publications per year.

5 Article 5: Efficient Hardware Implementation of the Horn-Schunck Algorithm (Duplicate)

Note: This content seems identical to Article 1. To save space, please refer to Section 1.

6 Article 6: FPGA-based Real-Time Optical-Flow System (Lucas-Kanade)

This article describes a "virtual motion sensor" based on the Lucas-Kanade (LK) algorithm, implemented on a Xilinx Virtex FPGA. Unlike global methods, this local method is designed for embedded applications such as robot navigation. A standalone system embeds the frame grabber, optical flow processor, and output module on a single board.

6.1 Optimization: IIR Temporal Filters

A key innovation is replacing standard Finite Impulse Response (FIR) filters—which typically require storing 15 frames—with Infinite Impulse Response (IIR) filters. This optimization significantly reduces memory requirements to just 3 frames, making the system viable for embedded hardware. This allows for efficient calculation of the temporal derivative I_t without massive RAM usage.

6.2 Pipeline Architecture and Custom FPU

The architecture is divided into six specific stages ($S_0 - S_5$). Stage 4, the construction of Least-Squares Matrices, is the most expensive, consuming 79% of the device's slices.

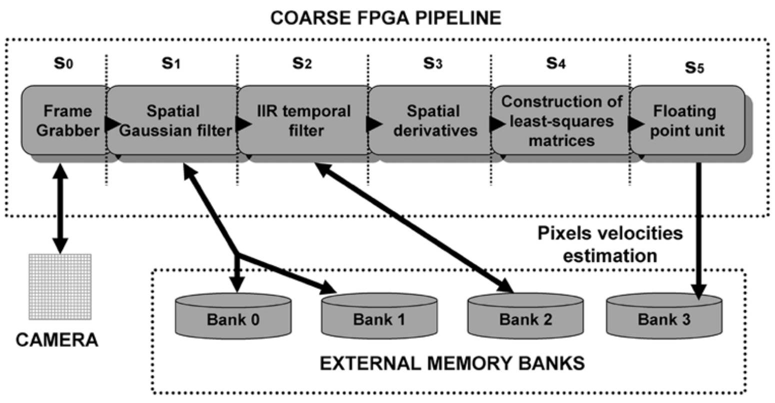


Fig. 1. Coarse pipeline processing architecture.

Unlike the Horn-Schunck implementation (Article 1) which utilized fixed-point arithmetic, this architecture employs a custom simplified Floating-Point Unit (FPU) for the final velocity estimation stage

(S_5). The authors argue that the final matrix inversion requires high dynamic range to avoid accuracy loss.

6.3 Mathematical Modifications

The algorithm solves the standard optical flow constraint using Weighted Least Squares. To handle the "aperture problem" where gradients are weak, the system adds a stabilization constant α to the matrix diagonal. This ensures the matrix remains invertible even in low-contrast regions.

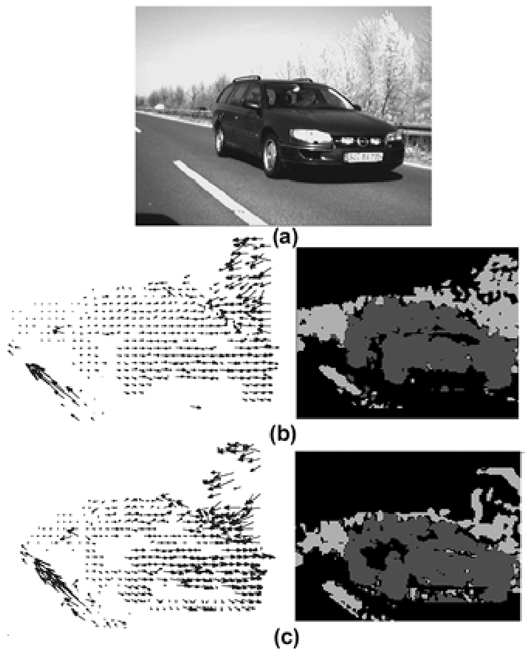


Fig. 2. Optical flow for the overtaking car. Software versus hardware estimations. (a) Original image extracted from the sequence. (b) Software result and (c) hardware result. The left-hand images use arrows to represent velocity vectors. In the right-hand images, for the sake of clarity, only leftwards (light colors) due to the landscape and rightwards (dark colors) due to the overtaking car are used to indicate the motion.

7 Article 7: FlowAcc: Real-Time DNN-based Optical Flow Accelerator in FPGA

Source: FlowAcc: Real-Time High-Accuracy DNN-based... (Doc 7)

"FlowAcc" presents a hardware accelerator architecture for FPGAs that bridges the gap between the speed of classical algorithms and the accuracy of Deep Neural Networks (DNNs). The design utilizes Binary Neural Networks (BNNs), where weights and activations are quantized to 1-bit, replacing expensive floating-point multiplications with bitwise XNOR operations.

7.1 Pyramidal Hardware Architecture

The system employs a pyramidal processing scheme to handle large displacements. Instead of instantiating separate hardware modules for each pyramid level (which consumes excessive area), the authors implement **temporal multiplexing**. A single, shared BNN module sequentially processes down-scaled versions of the image. The matching cost (C_l) at each level l is computed using the Hamming distance (H) between binary descriptors (f_1, f_2) extracted by the BNN:

$$C_l(p, \vec{mv}) = H(f_1^l(p), f_2^l(p + \vec{mv} + 2 \times \vec{mv}_f^{l-1}(p))) \tag{5}$$

7_1.png

7.2 LUT-6 Based Block Matching Unit

A critical hardware innovation is the Block Matching Unit, optimized for Altera Stratix V FPGAs. The authors map the Hamming distance calculation directly into the FPGA's 6-input Look-Up Tables (LUT-6). By integrating XOR operations and full adders into the LUTs, the architecture processes 64-bit feature vectors in parallel with minimal logic delay.

7_2.png

7.3 Regularization and Performance

To refine the flow field, a hardware-pipelined regularization module applies a local smoothness constraint. An energy function (E) combines the matching cost with a penalty factor (Θ) for abrupt variations. The final motion vector is up-scaled and refined iteratively:

$$\vec{mv}_f^{l}(p) = \vec{mv}_s^{l}(p) + 2 \times \vec{mv}_f^{l-1}(p) \tag{6}$$

FlowAcc achieves a throughput of 131.5 frames per second at 640x480 resolution, offering lower Average Endpoint Error (AEE) than comparable FPGA implementations of classical methods.

8 Article 8: ABMOF: A Novel Optical Flow Algorithm for Dynamic Vision Sensors

This paper presents Adaptive Block-Matching Optical Flow (ABMOF), an algorithm designed specifically for Dynamic Vision Sensors (DVS). Unlike standard cameras, DVS outputs a stream of asynchronous events. Standard algorithms like Horn-Schunck or Lucas-Kanade are not naturally matched to this sparse data. ABMOF accumulates events into "time slices" and uses a Block-Matching approach to calculate flow.

8.1 Adaptive Slice Generation

The key innovation is the "Area Event Number" metric. Instead of accumulating events for a fixed time (which fails for fast motion) or fixed count (which fails for sparse scenes), the system rotates slices based on local event density. This ensures slices always have sufficient texture for matching.

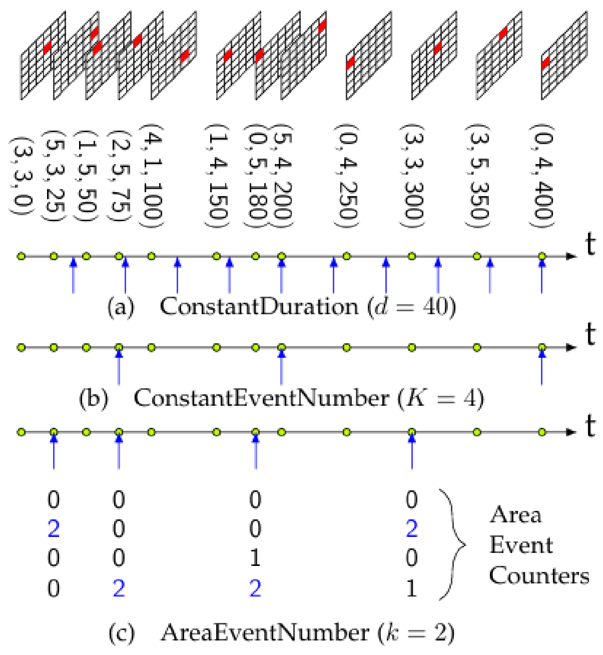


Figure 4: Comparison of (a) Constant time, (b) Global event count, and (c) The proposed "Area Event Number" method.

8.2 Block Matching Logic

Unlike gradient-based methods, ABMOF uses Sum of Absolute Differences (SAD) to track pixel blocks. SAD is computationally cheaper in hardware than solving linear systems. The logic minimizes the difference between a reference block in slice $t - d$ and a search region in slice $t - 2d$:

$$SAD(dx, dy) = \sum |I_{t-d}(x, y) - I_{t-2d}(x + dx, y + dy)| \tag{7}$$

To further optimize performance, the implementation uses Diamond Search rather than Full Search. This reduces the computational cost by 14× for a search radius of 12 pixels. A multi-scale strategy (pyramid) is also employed to handle very fast motion by subsampling the event addresses.

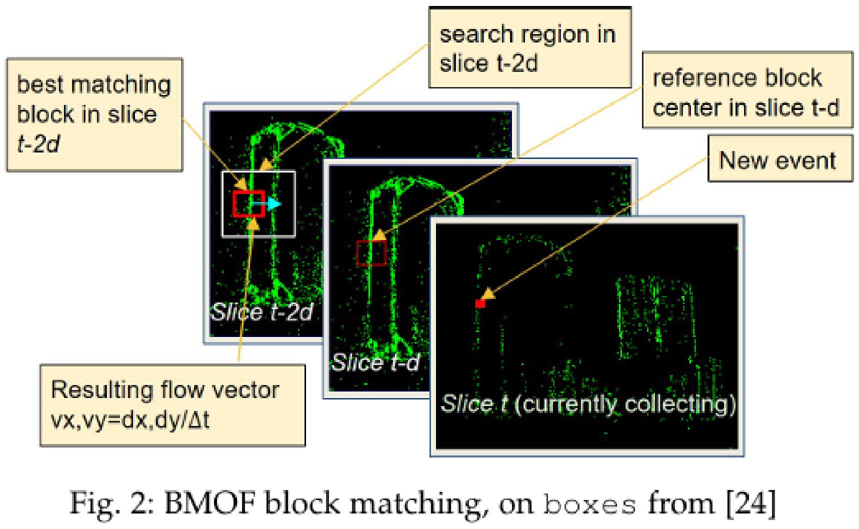

best matching block in slice t-2d
search region in slice t-2d
reference block center in slice t-d
New event
Slice t-2d
Slice t-d
Slice t (currently collecting)
Resulting flow vector vx,vy=dx,dy/Δt
Fig. 2: BMOF block matching, on boxes from [24]

Figure 5: Illustration of accumulating asynchronous events into time slices and performing block matching.

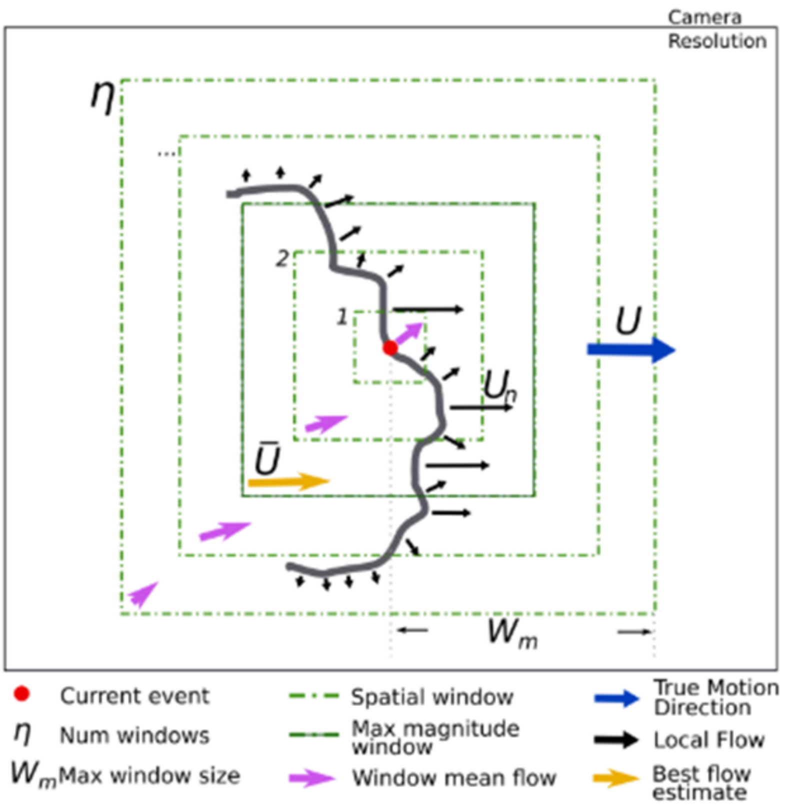
9 Article 9: hARMS: Hardware Architecture for Real-Time Event-Based Optical Flow

This paper presents **hARMS**, a hardware realization of the optimized **fARMS** (faster Aperture Robust Multi-Scale) optical flow algorithm. The primary goal is to calculate **true optical flow** (solving the aperture problem) in real-time for high-speed robotics. The solution targets a Xilinx Zynq-7000 SoC and achieves a throughput of up to **1.21 Mevt/s** with low power consumption ($<$ 1.3 W).

9.1 Mathematical Foundations

The ARMS algorithm determines the true direction of motion by analyzing local flow across spatial windows of increasing sizes. The principle is based on the relationship between local flow (U_n, normal to the edge) and true flow (U): $U_n = |U|\cos(\theta)$. The algorithm identifies the correct "aperture" size (k) by maximizing the average magnitude of local flow vectors:

$$\arg\min_k(E) = \arg\max_k(\overline{|U|_k}) \tag{8}$$



9.2 Optimization: fARMS Complexity

The original ARMS relied on frame-based processing ($O(W_m^2\eta)$). The optimized **fARMS** introduces a **Recent Flow Buffer (RFB)** to store only the most recent N events, reducing complexity to $O(N\eta)$. For typical configurations, this reduces computational load by 99%.

9.3 Hardware System Architecture

The architecture is partitioned between the Processing System (PS - ARM CPU) and Programmable Logic (PL - FPGA):

- **Window Arbitration (PL):** Assigns "tags" to events based on spatial windows using a pipelined lookup table.

- **Stream Averaging (PL):** Calculates vector sums for all windows in parallel.
- **ARMS Compute Core (PL):** Filters events and selects the optimal window.

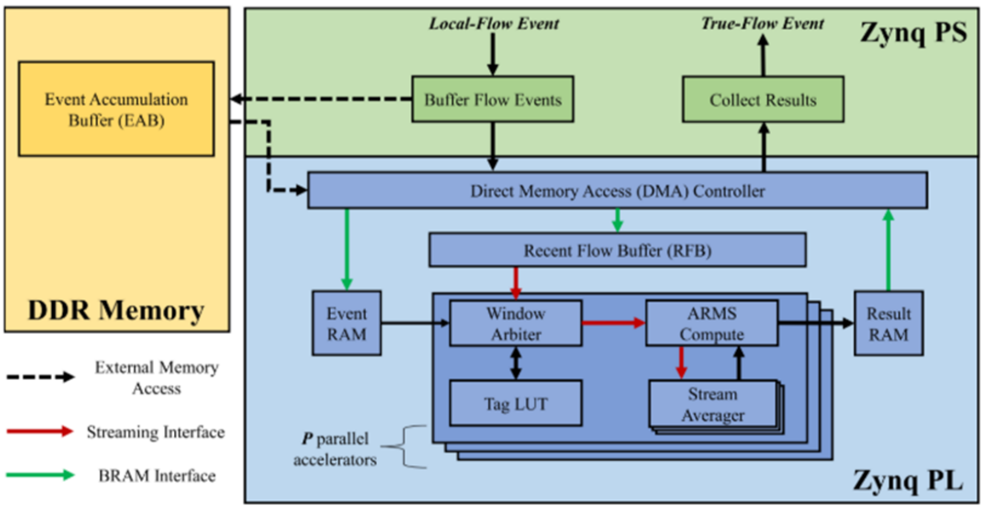


Figure 6: High-level block diagram of the Zynq SoC implementation.

10 Article 10: Structural Vibration Detection Using Optimized Optical Flow and UAVs

This study addresses the challenge of non-contact structural health monitoring (SHM) using Unmanned Aerial Vehicles (UAVs). The core problem is the contamination of vibration data by the UAV's ego-motion (hover drift and wind-induced shaking). The authors propose a two-stage solution: an optimized Lucas-Kanade (LK) algorithm for robust tracking and a motion cancellation mechanism based on background reference points.

10.1 Optimized Lucas-Kanade Algorithm

The traditional LK method assumes constant brightness and small displacements, leading to errors under varying outdoor illumination. To mitigate this, the authors introduce an angular constraint mechanism. The process begins with Harris corner detection. For each feature point, the optical flow vector is calculated, and its motion direction angle (A) is derived. The system computes the mean (M) and standard deviation (S) of these angles:

$$A = \frac{\operatorname{atan}(\frac{u_i}{v_i}) \cdot 180}{\pi} \qquad (9) \qquad\qquad M = \dots, \quad S = \sqrt{\frac{\sum_{i=1}^{n}(A_i - M)^2}{n}} \qquad (10)$$

Feature points deviating from the mean by more than a dynamic threshold (defined as a multiple of σ) are classified as outliers and discarded. This ensures only coherent motion vectors contribute to the displacement calculation.

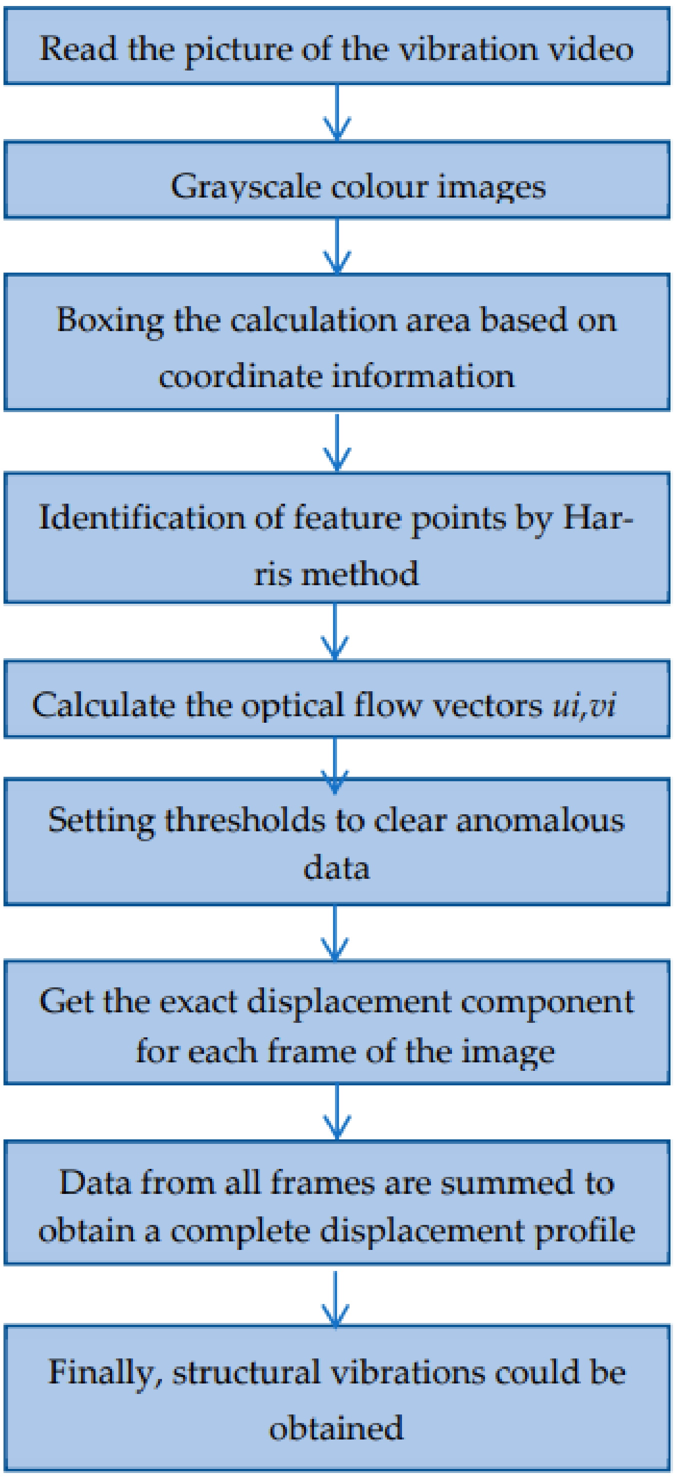

Read the picture of the vibration video
Grayscale colour images
Boxing the calculation area based on coordinate information
Identification of feature points by Harris method
Calculate the optical flow vectors *ui,vi*
Setting thresholds to clear anomalous data
Get the exact displacement component for each frame of the image
Data from all frames are summed to obtain a complete displacement profile
Finally, structural vibrations could be obtained

Figure 7: Logic flow of the optimized angle-constrained algorithm.

10.2 UAV Motion Elimination

To decouple drone movement from structural vibration, the system tracks stationary background reference points. First, a planar homography transformation rectifies geometric distortions caused by camera rotation (pitch/yaw). Once corrected, the translational component (x_d) is calculated from static points. True structural displacement (x_r) is derived by subtracting camera motion from total measured motion (x_0):

$$x_r = x_0 - x_d \tag{11}$$

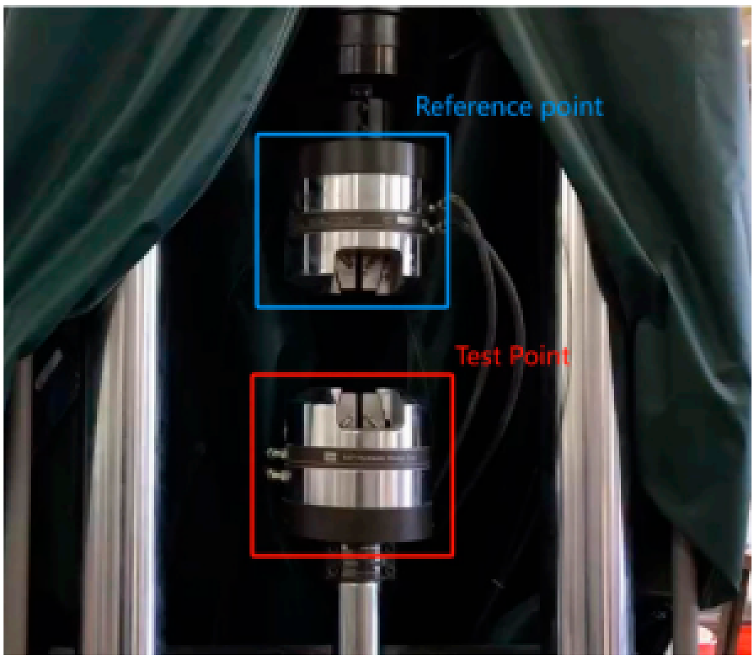


Figure 8: Mechanism for background subtraction to eliminate ego-motion.

10.3 Experimental Results

Experimental validation using a high-precision MTS hydraulic system demonstrated that the optimized method achieves approximately 97% accuracy in laboratory settings. Field tests with a hovering UAV showed that the algorithm effectively reconstructs the true vibration waveform from highly noisy data, achieving over 90% accuracy even under significant wind interference.

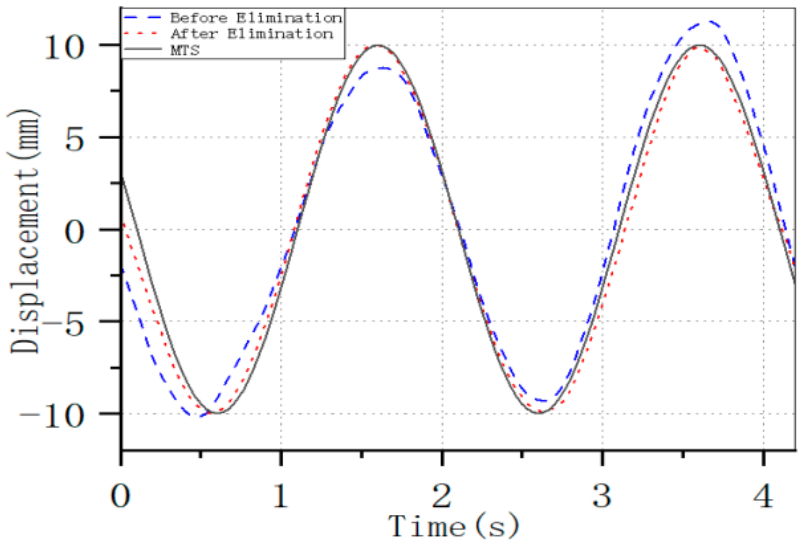


Figure 9: Signal correction results during UAV field tests (0.5 Hz).

11 Article 11: FPGA Based High Performance Optical Flow Computation

[cite$_s$tart]*Source: FPGA Based High Performance Optical Flow Computation Using Parallel Architecture [cite: 1*

[cite$_s tart]Thispaperpresentsahighlyparallelhardwarearchitectureforhigh-performanceopticalflowcomputatio$
$5].[cite_s tart]ThesystemutilizestheLucas-Kanade(L\&K)algorithmextendedwithamulti-scale(pyramidal)appro$
$6].[cite_s tart]Theprimaryobjectiveistoachieveathroughputofonepixelperclockcycleacrosstheentireprocessingsche$
$pipelinearchitecture[cite: 7]$.

11.1 Mathematical Foundations

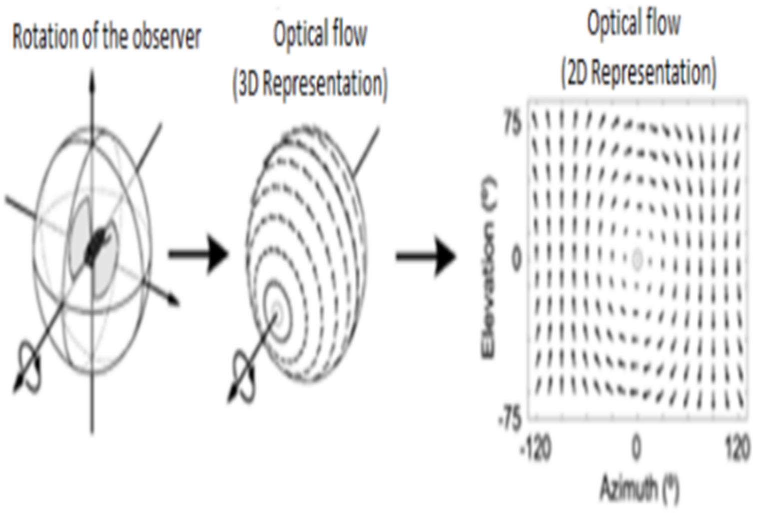
[cite$_s tart]Opticalflowestimationisbasedonthebrightnessconstancyassumption,leadingtotheopticalflowconstrai$
$10,11]: I_x V_x + I_y V_y + I_t = 0(12)$ [cite$_s tart]Orinvectorform: \nabla I \cdot V = -I_t$[cite: 12].

[cite$_s tart]Sincethisequationhastwounknowns(\mathrm{V}_x, V_y)$, the Lucas-Kanade algorithm assumes constant flow within a local window, creating an over-determined system solved via Weighted Least Squares (matrix W emphasizes central pixels)[cite: 15, 16, 18]:

$$v = (A^T W A)^{-1} A^T W b \tag{13}$$

11.2 System Architecture and Methodology

[cite$_s tart]Theproposedsystemem ploysahybridsimulationapproach(Matlab+Modelsim)[cite: 22].[cite_s tart]Thep$
$24]$.



[cite$_s tart$]

Figure 10: Processing flow: Video to Optical Flow Estimation[cite: 25].

[cite$_s tart]Thehardwarecomputationcoreisdividedinto5pipelinestagestomaximizespeed[cite: 27]$:

[cite$_s tart$]

Gaussian Smoothing: Noise reduction[cite: 28]. [cite$_s tart$]

FIR Temporal Filters: Calculation of derivatives over time[cite: 29]. [cite$_s tart$]

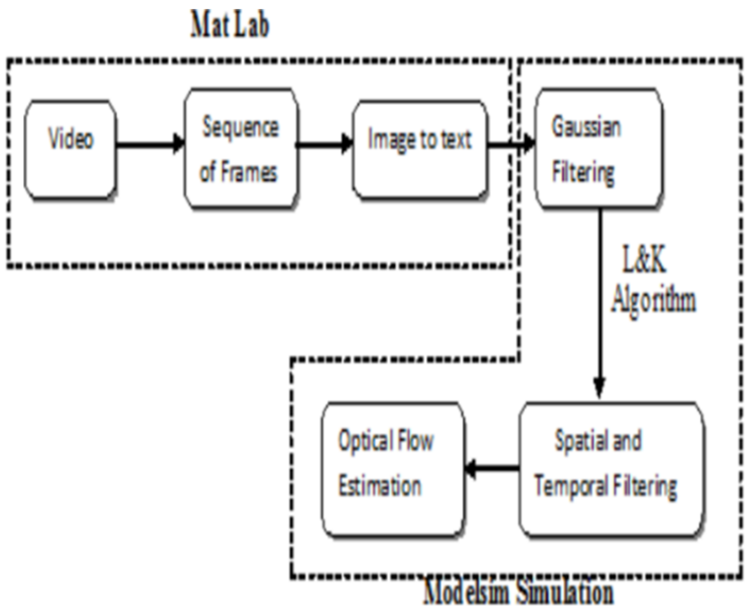
FIR Spatial Derivatives: Calculation of gradients[cite: 30]. [cite$_s tart$]

Construction of Least-Squares Matrices: Building matrices for the system equation[cite: 31]. [cite$_s tart$]

Fixpoint Unit: Final resolution of the velocity vector[cite: 32].

11.3 Multi-Scale Estimation

[cite$_s tart]Todetectlargemovements,thesystemimplementsaGaussianpyramid(Coarse-to-Fineapproach)$
$34,35].[cite_s tart]Thisprocessincludesmotionestimationatlowresolution,upscaling,imagewarping,andrefi$
$36]$.



[cite$_{s}tart]$

Figure 11: Multi-scale process diagram: Motion Estimation, Scaling, and Warping[cite: 37].

11.4 Results

[cite$_{s}tart]Simulationsindicatethearchitectureachievesagoodbalancebetweenaccuracyandefficiency[cite:$
$45].[cite_{s}tart]Thefine-pipelinearchitectureenableshighthroughput, makingitsuitableforphysicalimplementatio$
$70FPGAtoleveragemassiveparallelism[cite:45,48,50]$.