Flow Perception of Great Accuracy (FPoGA): An FPGA Optical Flow Sensor for Motion Tracking

Ryan Tomich
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge MA, USA
rjtomich@mit.edu

Andy Yu
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge MA, USA
andyy455@mit.edu

***Abstract*—Real-time motion estimation is difficult to achieve in embedded systems due to latency, drift, and nondeterminism in conventional processing pipelines. We presents FPoGA, a fully streaming FPGA implementation of a Lucas–Kanade optical flow pipeline that produces frame-level global motion estimates from a live camera stream. The design employs a three-level pyramidal formulation and a resolution-state-controlled datapath that reuses computational modules across scales, enabling coarse-to-fine motion estimation within a single streaming pipeline with bounded latency. Optical flow is computed sparsely at FAST-detected features using localized least-squares solving and aggregated into a global motion vector per frame. Experimental evaluation on a ground-plane motion setup demonstrates consistent discrimination of motion direction and temporally stable relative motion signals under controlled conditions. The reported magnitudes are not metrically calibrated velocities, reflecting the constraints imposed by fixed-point arithmetic, noise sensitivity, and illumination variability in streaming hardware implementations.**

***Index Terms*—Optical flow, Field-programmable gate arrays, Lucas–Kanade method, Embedded vision, Motion estimation, Streaming architectures**

I. Introduction

Relative motion estimation is a core requirement formany systems, including embedded sensing platforms, mobile robots, and autonomous vehicles. In these regimes, methods based on GPS, wheel odometry, or inertial sensing often degrade under drift, accumulated bias, latency, or environmental dependence, particularly at small spatial or temporal scales. Optical flow circumvents these limitations by estimating motion directly from spatiotemporal intensity variation in the image stream, providing a contact-free, scene-relative measurement.

This work presents *FPoGA*, an FPGA-based optical flow sensor that estimates global image motion using a hardware-oriented, multi-scale Lucas–Kanade formulation [1]. The design emphasizes low-latency operation for real-time motion estimation. Rather than computing dense optical flow, the system operates on a sparse set of trackable image features and refines motion estimates across multiple spatial resolutions.

The proposed architecture leverages the observation that a pyramidal Lucas–Kanade pipeline, when combined with streaming feature selection and localized least-squares solving, maps onto an image FPGA datapath. Computational units are reused across pyramid levels under finite-state control, enabling continuous processing directly from camera input with bounded latency and limited on-chip storage.

This paper describes the following aspects of the system:

- A fully streaming, deterministic FPGA implementation of the Lucas–Kanade optical flow pipeline that produces frame-level global motion estimates from a live camera stream.
- A three-level pyramidal formulation realized in hardware, enabling coarse-to-fine motion estimation within a continuous streaming pipeline.
- A resolution-state-controlled datapath organization that reuses processing modules across pyramid levels.

II. Background and Related Work

a) Background: The Lucas–Kanade optical flow method estimates apparent motion by assuming brightness constancy between consecutive frames, small inter-frame displacement, and locally uniform motion within a spatial window. Under these assumptions, optical flow can be computed from spatial and temporal image gradients using a local least-squares formulation. When these assumptions are violated, the local least-squares system may produce inaccurate estimates in the corresponding regions.

To relax the small-motion assumption, the pyramidal Lucas–Kanade approach evaluates optical flow hierarchically across multiple spatial resolutions. Motion estimates obtained at coarser scales are used to warp finer-resolution data, enabling the estimation of larger displacements while preserving the small displacement assumption of the least squares.

Lucas–Kanade is well suited for hardware implementation compared to global methods such as Horn–Schunck because it operates on local image neighborhoods and admits a closed-form least-squares solution. In contrast, global formulations typically require iterative, image-wide optimization. The locality of Lucas–Kanade allows computations to be performed incrementally as pixels stream through the pipeline, with partial results accumulated without multiple full-image passes, enabling more deterministic latency and efficient hardware utilization.

Field-programmable gate arrays (FPGAs) are particularly well suited for such workloads due to their support for fine-

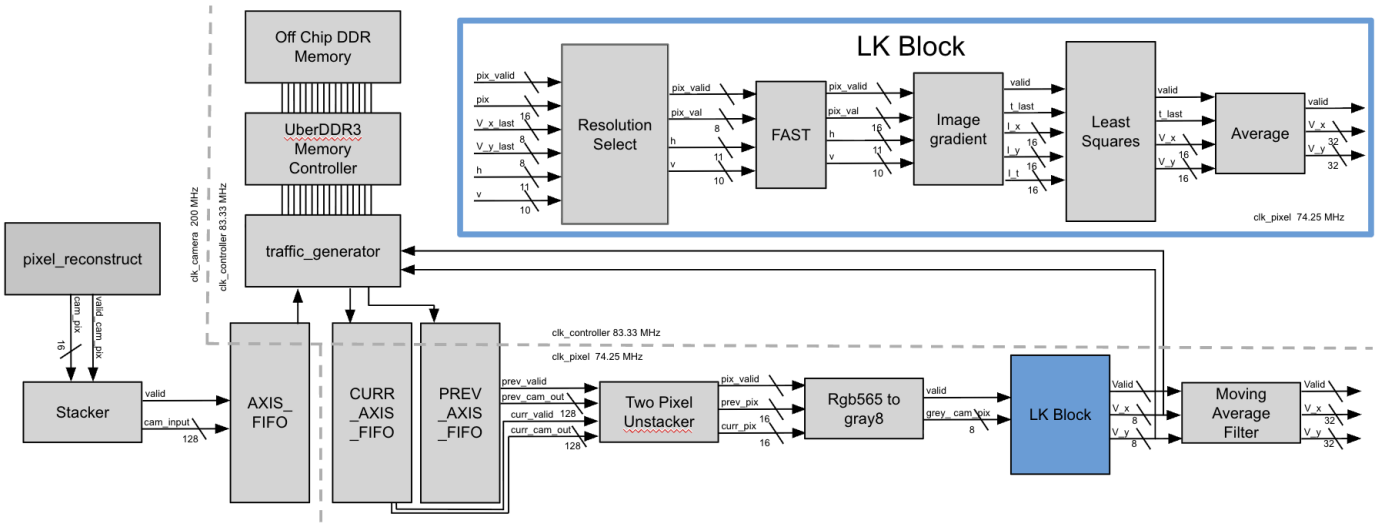

Fig. 1. Top-level block diagram of the FPoGA optical flow system. Camera frames stream through feature detection, gradient computation, and least-squares solving under resolution-state control to produce a single frame-level motion vector that can be used to warp subsequent read frames.

grained spatial parallelism. Local gradient computation and small fixed-size matrix operations can be executed concurrently across streaming data, allowing high-throughput optical flow estimation.

b) Related Work: Optical flow has been widely studied across software and hardware platforms, including implementations on CPUs, GPUs, ASICs [2], and FPGAs. Commercial motion sensors, such as optical computer mice and ground-speed ASICs, demonstrate real-time optical flow using fixed-function hardware, though their internal architectures and algorithmic flexibility are limited.

Software-based implementations on CPUs and GPUs provide high accuracy and configurability but often lack deterministic latency and are constrained by power and system-level scheduling effects. FPGA-based optical flow systems have been explored in both dense and sparse formulations, including Lucas–Kanade [3] and Horn–Schunck [4] variants, with trade-offs among throughput, memory usage, and architectural complexity.

III. System Overview

Figure 1 illustrates the overall FPoGA architecture and dataflow. The system implements a fully streaming optical flow pipeline on an FPGA, processing image data end-to-end from raw camera input to a per-frame global motion estimate.

Successive camera frames are stored in external memory to support temporal gradient computation. Image data from the current and previous frames are paired and streamed to a FAST corner detector, which identifies salient features suitable for tracking. For each detected feature, spatial and temporal image gradients are computed over a fixed window and supplied to a Lucas–Kanade least-squares solver that estimates the local flow vector.

The resulting per-feature motion estimates are averaged to produce a single optical flow vector per frame. This vector is then piped back to the next frame and resolution to warp the incoming frame. After all resolutions are calculated and accumulated, the final vector is temporally filtered. For visualization, the estimated motion vector is rendered as an on-screen overlay with its components showing on the display.

A. Memory Interface (Andy)

The memory pipeline expands on the lab 6 memory system. The DRAM is used as a 1280x720 frame buffer for incoming camera frames. We use 3 full-frame sized regions when accessing the DRAM; one to write the incoming frame, one to store and read the most recent frame written, and one to store and read the previous frame written. When a frame is completely written, the indices are rotated. The pixels from current and previous frame reads are fed into a separate read-axis FIFOs, which connects to a custom unstacker that outputs current and previous frame pixels of the same position.

B. Resolution State Machine (Ryan)

The Resolution State Machine orchestrates multi-resolution processing by controlling spatial downsampling, parameter selection, and result accumulation across pyramid levels. Rather than instantiating separate pipelines per resolution, the state machine sequences the system through three frame-level states corresponding to $1/16$, $1/4$, and full resolution, followed by two single-cycle states that clear and emit the accumulated result. Estimation goes from coarse-to-fine with each resolutions calculated vector being used to warp the current frame for the next resolution.

Downsampling is implemented implicitly by selectively asserting the valid signal while leaving the original pixel coordinates unchanged. Resolution-dependent parameters, including scaling shifts and accumulation enables, are configured per state. Flow vectors computed at each resolution are accumulated into a single motion estimate per pyramid. This approach enables coarse-to-fine estimation using a single

hardware datapath while maintaining synchronized control over resolution, arithmetic scaling, and aggregation.

C. Line Buffer (Ryan)

The line buffer converts the sparsely sampled pixel stream produced by resolution-dependent valid gating into a dense spatial representation suitable for downstream processing. While horizontal and vertical pixel counters remain unchanged, only selected samples are marked valid to achieve effective downsampling by factors of two or four in each dimension.

To preserve compatibility with later stages that expect contiguous image neighborhoods, the line buffer condenses sparse input rows and columns into a compact internal image by removing all-zero rows and columns. This allows later modules to operate on smaller effective images without awareness of the original sparse sampling pattern. By decoupling spatial indexing from sampling density, the line buffer enables a shared processing pipeline across resolutions.

D. FAST (Ryan)

The Features from Accelerated Segment Test (FAST) [5] is a computationally efficient corner detection algorithm well suited for hardware implementation. The Lucas–Kanade method estimates optical flow more reliably at locations with strong intensity gradients, such as corners. FAST identifies such locations by comparing the intensity of a candidate pixel to those on a Bresenham circle centered at that pixel. A pixel is classified as a corner if at least n contiguous pixels on the circle are either brighter than the center by a threshold t or darker by more than t. Optical flow is computed only at pixels identified as trackable by FAST in subsequent pipeline stages.

FPoGA uses a Bresenham circle of radius 3 with $n = 12$ contiguous pixels and a threshold $t = 100$, assuming 8-bit grayscale intensities in the range $[0, 255]$.

E. Image Gradient (Ryan)

The Lucas–Kanade method estimates optical flow using the spatial image gradients $I_x(x, y, t)$ and $I_y(x, y, t)$, together with the temporal gradient $I_t(x, y, t)$. For each tracked feature, these quantities are evaluated over a local window surrounding the pixel of interest.

FPoGA computes gradient information on a 5×5 window for each FAST-detected tracking pixel using a local image cache. The raw temporal gradient I_t is filterd with a deadband operation to suppress small-magnitude fluctuations. The spatial gradients are computed using Sobel operators, [1]

$$I_x(x, y, t) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \circledast I(x, y, t),$$

$$I_y(x, y, t) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \circledast I(x, y, t).$$

[1]The operator $K \circledast I(x, y, t)$ denotes convolution of image $I(x, y, t)$ with kernel K, centered at pixel (x, y).

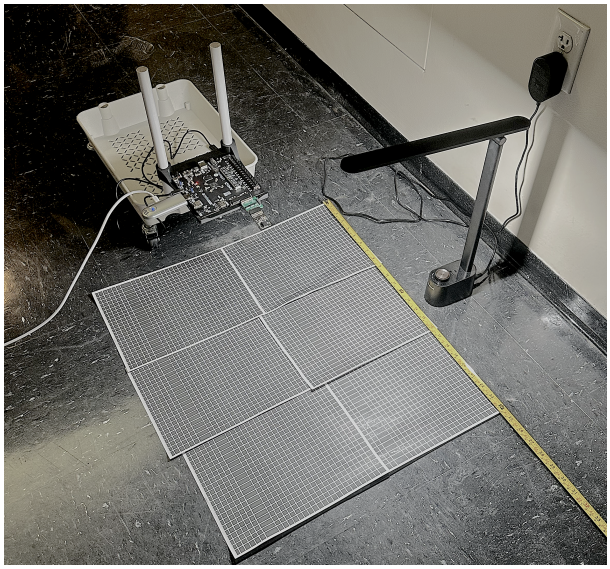

Fig. 2. Experimental evaluation setup with the FPGA and camera mounted on a wheeled cart and oriented normal to the ground.

$$I_t(x, y, t) = I(x, y, t + 1) - I(x, y, t)$$

F. Least Squares (Andy)

All pixels surrounding a target pixel in the 5×5 window are $\{q_1 \cdots q_{25}\}$. The Lucas–Kanade equations over the window can be written in matrix form $A\,v = b$ where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_{25}) & I_y(q_{25}) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_{25}) \end{bmatrix}.$$

The system is overdetermined, so the least-squares solution is obtained by $A^T A\,v = A^T b$, Explicitly, this yields:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{25} I_x(q_i)^2 & \sum_{i=1}^{25} I_x(q_i) I_y(q_i) \\ \sum_{i=1}^{25} I_y(q_i) I_x(q_i) & \sum_{i=1}^{25} I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{25} I_x(q_i) I_t(q_i) \\ -\sum_{i=1}^{25} I_y(q_i) I_t(q_i) \end{bmatrix}.$$

Upon receiving a 5×5 window of gradients, the module sequentially accumulates the required sums and evaluates the resulting system using non-blocking signed integer division. To balance numeric range, precision, and hardware complexity, all computation is performed in a signed-integer datapath with explicit shift-based scaling.

Dynamic range is bounded by applying a fixed `scale_shift` to accumulated sums and intermediate products prior to division, ensuring compatibility with a 32-bit signed divider while preserving relative magnitudes. Output precision is controlled independently by left-shifting the dividend via `dividend_shift`, introducing fractional resolution in the integer division without resorting to fixed- or floating-point arithmetic.

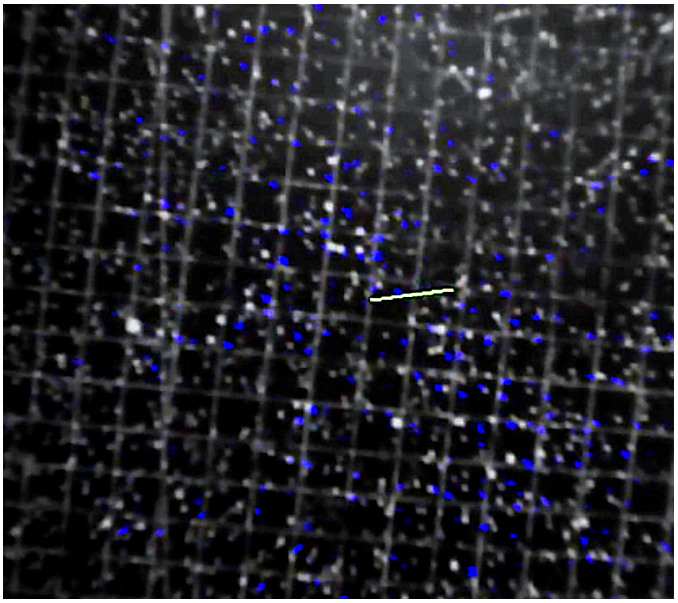

Fig. 3. Estimated global optical flow vector rendered as a directed green line segment over the input image during preliminary desktop testing with postcard.

Fig. 4. 7 segment display showing signed hexadecimal x- and y-components of the estimated global motion vector.

G. Frame Average (Ryan)

The Frame Average module aggregates the per-feature optical flow estimates produced within a frame into a single representative motion vector. Upon completion of a frame, the accumulated sums are normalized by the number of valid features to produce a frame-level average motion vector. Only vectors corresponding to valid FAST features are included in the accumulation. To maintain hardware efficiency, the number of vectors accumulated per frame is limited to the largest power of two not exceeding the number of valid features.

H. Moving Average Filter (Ryan)

The moving average filter is a parameterized module that maintains a moving average of a parameterized, N, number of items. Currently, $N = 8$ to balance responsiveness and noise. Upon receiving a new valid input, the value is stored and the oldest value is removed, with the average recalculated as necessary. This is done for both the V_x and V_y values, with the average values always available combinationally.

I. Vector Draw (Ryan)

The Vector Draw module displays the temporally averaged flow vector once per frame as a directed line segment centered at the image midpoint. For each raster coordinate (x, y) relative to the center, an implicit line test based on the vector components is evaluated, and pixels are marked as part of the line when $|V_y x - V_x y|$ falls below a magnitude-scaled tolerance. Additional sign and magnitude constraints limit rendering to a finite segment in the direction of the vector, ensuring that the displayed line is proportional to the underlying pixel displacement. [6]

IV. Experimental Setup

The proposed design was implemented on an Urbana development board featuring a Xilinx Spartan-7 FPGA and synthesized using Vivado 2025.1. The system interfaces with an Adafruit OV5640 camera module equipped with a 72° field-of-view lens. The design operates with a 100 MHz global system clock and an 83.333 MHz pixel clock derived from the onboard clocking resources.

For experimental evaluation, the FPGA board was mounted on a wheeled cart with the camera oriented normal to the ground plane. The camera was positioned at a fixed height of 18 cm above the ground. Experiments were conducted on a well-lit surface with substantial visual texture to ensure reliable feature detection. The physical setup is shown in Fig. 2. Preliminary functional validation was also performed in a desktop setting using a postcard, as illustrated in Fig. 3.

Motion experiments consisted of manually translating the cart back and forth along a straight path, approximating simple harmonic motion. Under ideal conditions, this trajectory produces a velocity profile of the form

$$v(t) = A\omega \cos(\omega t + \phi),$$

where A is the displacement amplitude, ω the angular frequency, and ϕ a phase offset. During each trial, motion estimates were read directly from the onboard seven-segment display at 0.25 s intervals. A total of 333 samples were recorded across all trials. Ground-truth direction was determined by the known direction of cart motion during each measurement interval.

V. Results

A. Resource Usage

Post-implementation results indicate that the design occupies 9,346 lookup tables (28.7% of the device), 54 DSP blocks (45%), and 16.5 block RAM tiles (22%). The design meets timing with 0.325 ns buffer. With more conservative fixed-point bit widths, LUT and BRAM utilization could likely be reduced further.

The end-to-end processing pipeline introduces a latency lower bound of (3 pyramid levels x 1650 rows x 750 columns) clock cycles or (≈ 0.05 ms) in the pixel-clock domain. This is the minimum number of clock cycles for the average over the entire frame to be calculated. By similar logic, the maximum

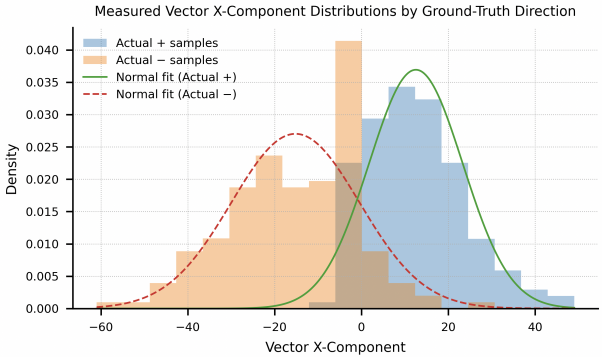

Fig. 5. Empirical distributions of the measured x-component conditioned on known positive and negative motion directions.

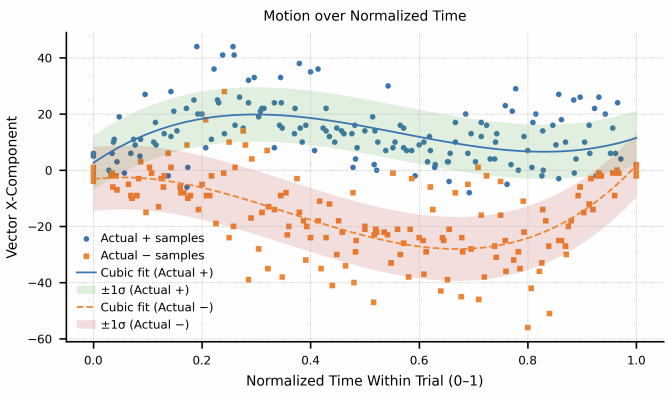

Fig. 6. Time-normalized trajectories of the measured x-component during repeated rest-to-rest harmonic motion trials.

throughput of the design is 20Hz. Additional latency variability is introduced by external memory accesses. This nondeterminism arises from clock-domain crossings implemented via asynchronous FIFOs, shared DDR memory arbitration on the AXI bus, and burst-alignment effects during DRAM transactions. These effects do not alter functional correctness but result in bounded variation in the time at which output vectors become available.

B. Empirical Data

Fig. 3 shows an example of an estimated optical-flow vector overlaid on the input image during preliminary desktop testing. Fig. 4 shows the seven-segment display used to report the signed hexadecimal magnitude of the estimated motion vector components.

a) Direction: To assess directional discrimination, the empirical distribution of the measured x-component was evaluated conditional on the known direction of cart motion. Fig. 5 shows the conditional distributions $f_{X|D}(x|d)$ for positive ($d = +$) and negative ($d = -$) motion. For positive motion, the measured x-component is well approximated by a normal distribution with mean $\mu = 12.52$ and standard deviation $\sigma = 10.80$. For negative motion, the distribution has mean $\mu = -15.21$ and standard deviation $\sigma = 14.76$.

b) Magnitude: The reported magnitude does not correspond to true physical velocity, but is proportional to the motion. To evaluate temporal behavior, six trials of positive and negative motion were time-normalized. The resulting trajectories, shown in Fig. 6, exhibit the expected sinusoidal structure consistent with rest-to-rest harmonic motion. Quantitative agreement with the expected motion profile was evaluated using root-mean-square (RMS) error, shape correlation, and peak timing error. For positive-direction trials, the RMS error was 0.4600, the shape correlation was 0.1124, and the peak timing error was 0.2831. For negative-direction trials, the RMS error was 0.4644, the shape correlation was 0.1624, and the peak timing error was 0.4697.

C. Checklist

A set of implementation commitments were defined at project inception. The design required storage of two consecutive video frames in off-chip memory. In practice, a rolling buffer of three frames was implemented in DRAM to support multi-resolution processing. All core components of the Lucas–Kanade pipeline were successfully implemented. These include FAST feature detection using convolution-based pixel selection, hardware computation of image gradients, and a closed-form least-squares solver for per-feature motion estimation. Frame-level averaging and a moving-average temporal filter were incorporated to produce stable global motion estimates.

The primary objective of implementing a three-resolution pyramidal Lucas–Kanade pipeline was achieved using a resolution-controlled finite state machine coordinating downsampling, warping, and cross-resolution accumulation. The system produces consistent motion estimates, as demonstrated by the experimental results, and displays the resulting flow vector in real time overlaid on the video stream.

Two stretch goals were not met: quantitative comparison against an external reference sensor and streaming motion estimates to a host machine via a Python interface. Motion values were instead observed through the onboard seven-segment display.

VI. Discussion

The conditional distributions of the x-component demonstrate clear separation between positive and negative motion directions. The maximum a posteriori estimates of the means are positive for forward motion ($\mu = 12.52$) and negative for reverse motion ($\mu = -15.21$), indicating reliable sign discrimination under the tested conditions. The relatively large standard deviations ($\sigma = 10.80$ for positive motion and $\sigma = 14.76$ for negative motion) reflect variability arising from manual actuation, uneven ground texture, and the fixed-point precision of the hardware pipeline. Despite this spread, the sign of the motion remains consistently identifiable.

The time-normalized magnitude trajectories exhibit the expected structure of harmonic motion, including reduced sample density near zero velocity where motion direction reverses. This manifests as a central sparsity in Fig. 6, consistent with

the sinusoidal velocity model. Across both directions, RMS error values are comparable, indicating symmetric overall accuracy. The higher peak timing error observed for negative motion suggests asymmetry in either the motion execution or the image-plane response, which is also consistent with the observed negative bias in Fig. 5 and the larger peak magnitudes seen in Fig. 6.

The observed performance also suggests that the current design is well suited for coarse motion detection tasks, such as object presence, direction-of-travel detection, or event counting, rather than precise velocity estimation. These use cases require reliable sign and relative magnitude discrimination, which the system already provides without modification. With minimal changes limited to downstream interpretation or thresholding of the output vector, the same pipeline could support applications such as motion-triggered sensing, flow-based gating, or low-latency directional feedback in embedded systems.

VII. Retrospective

This work exposed practical limitations that are consistent with known challenges in hardware optical flow systems. [7]. In particular, sensitivity to illumination conditions, noise sources, and fixed-point precision dominated both architectural decisions and evaluation outcomes.

Although the Lucas–Kanade formulation and the implemented pipeline treat horizontal and vertical motion symmetrically, the system exhibited substantially higher sensitivity and stability along the x-axis than the y-axis. For this reason, quantitative analysis was restricted to the horizontal component, despite full 2D vector support in the design. A persistent bias toward negative-valued motion estimates was also observed, visible in the asymmetries of Fig. 5 and Fig. 6. While the exact cause was not conclusively identified, this bias is suspected to arise from fixed-point bit-width choices that introduced asymmetric clipping behavior.

The end-to-end pipeline operates largely in fractional numerical regimes, requiring extensive use of fixed-point arithmetic, particularly within the least-squares solver. Preventing overflow and precision loss necessitated conservative scaling and saturation strategies. These choices significantly complicated debugging, as invalid or clipped intermediate values could silently suppress downstream valid signals.

Noise handling proved to be a primary design driver. The system was sensitive to scene flicker, environmental interference, lighting spectrum, and frame-to-frame jitter. These effects motivated multiple mitigation strategies, including FAST feature thresholding, Gaussian spatial filtering, a pyramidal formulation, frame-level and rolling temporal averaging, small-determinant gating in the least-squares stage, and deadbanding of the temporal image gradient. While these mechanisms improved stability, they also reduced sensitivity to small, high-frequency motion.

Ultimately, while the system reliably produced directionally meaningful motion vectors, it did not achieve the original goal of calibrated velocity estimates in pixel-per-frame units. The combined challenges of noise, fixed-point arithmetic, and scaling limited the interpretability of the outputs as true velocities. The resulting vectors are better characterized as relative motion indicators rather than metrically accurate velocity estimates. Future work will focus on validating motion estimates against ground-truth trajectories.

VIII. Conclusion

This work presented *FPoGA*, a fully streaming FPGA implementation of a pyramidal Lucas–Kanade optical flow pipeline operating on a live camera stream. The system demonstrates that sparse, feature-based optical flow can be realized in hardware with bounded latency and modest on-chip storage by carefully structuring computation around streaming dataflow and localized least-squares solving.

The three core contributions of this work are as follows. First, we demonstrated a fully streaming FPGA implementation of the Lucas–Kanade optical flow algorithm producing global motion estimates directly from camera input. Second, we implemented a three-level pyramidal formulation in hardware that supports coarse-to-fine motion estimation in a streaming setting. Third, we introduced a resolution-state-controlled datapath organization that reuses computational modules across pyramid levels, enabling resource-efficient real-time operation.

Experimental results show that the system reliably discriminates motion direction and produces temporally consistent motion signals under controlled conditions. While the measured magnitudes are not metrically calibrated velocities, the outputs capture meaningful relative motion consistent with the underlying image dynamics. The retrospective highlights the dominant role of noise, fixed-point precision, and illumination sensitivity in constraining accuracy and interpretability, underscoring the gap between algorithmic formulation and hardware reality.

Overall, FPoGA serves both as a functional optical flow sensor and as a concrete case study in mapping multi-scale vision algorithms onto streaming FPGA architectures. The design decisions, evaluation results, and implementation lessons reported here provide practical guidance for future hardware optical flow systems that seek low latency, predictable timing, and efficient resource utilization.

IX. Code Availability

The hardware and software implementations used in this work correspond to commit `1d36521264e98cd102b7bb78d3c7bb4dd4251d13` on the `3_res` branch of the project repository: https://github.mit.edu/6205F25/fa25-6205-team26.

X. Attribution

The work described in this paper was carried out jointly by the authors. To improve transparency, primary responsibility for specific components is summarized below; however, these assignments are approximate. Both authors reviewed, edited, and discussed nearly all modules, and many components

were developed through pair programming and iterative joint refinement.

Andy Yu was primarily responsible for the development of the following hardware modules:

- `least_squares.sv`
- `high_definition_frame_buffer.sv`
- `traffic_generator.sv`
- `unstacker.sv`
- `seven_segment_vector.sv`

In addition, Andy led the CAD, sourcing, and construction of the experimental test rig, as well as video capture and editing for demonstration and evaluation.

Ryan Tomich was primarily responsible for the development of the following hardware modules:

- `pixel_downsampler.sv`
- `gaussian_blur_column.sv`
- `fast.sv`
- `image_gradient.sv`
- `frame_average.sv`
- `rolling_average.sv`
- `vector_draw.sv`

Ryan also developed the Python-based software model used for functional validation and produced the system block diagrams used throughout the paper.

All remaining work, including research, project scoping, system integration, debugging, architectural tradeoff considerations, testing, data analysis, and manuscript preparation (unless notated otherwise), was performed jointly by the authors.

XI. Acknowledgments

The authors thank Joe Steinmeyer, Monica Chan, and Kiran Vuksanaj, as well as the rest of MIT 6.2050 staff, for their guidance, technical discussions, and debugging assistance. The authors also gratefully acknowledge the Massachusetts Institute of Technology for providing the laboratory facilities and hardware resources, including FPGA development boards, camera modules, and HDMI transmitter.

References

[1] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. Imaging Understanding Workshop*, 1981, [Online]. Available: https://cseweb.ucsd.edu/classes/sp02/cse252/lucaskanade81.pdf.

[2] J. Kühne, M. Magno, and L. Benini, "A fast and accurate optical flow camera for resource-constrained edge applications," in *Proc. 9th Int. Workshop on Advances in Sensors and Interfaces (IWASI)*, Jun. 2023, pp. 143–148.

[3] R. Allaoui, H. H. Mouane, Z. Asrih, S. Mars, I. El Hajjouji, and A. El Mourabit, "Fpga-based implementation of optical flow algorithm," in *Proc. 2017 Int. Conf. on Electrical and Information Technologies (ICEIT)*, Nov. 2017, pp. 1–5.

[4] K. Blachut and T. Kryjak, "Real-time efficient fpga implementation of the multi-scale lucas–kanade and horn–schunck optical flow algorithms for a 4k video stream," *Sensors*, vol. 22, no. 13, p. 5017, Jan. 2022.

[5] Wikipedia contributors. (2025, Nov.) Features from accelerated segment test. Accessed: Nov. 26, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Features_from_accelerated_segment_test&oldid=1322149841

[6] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, 1965, accessed: Nov. 26, 2025. [Online]. Available: https://web.archive.org/web/20080528040104/http://www.research.ibm.com/journal/sj/041/ibmsjIVRIC.pdf

[7] A. Alfarano, L. Maiano, L. Papa, and I. Amerini, "Estimating optical flow: A comprehensive review of the state of the art," *Computer Vision and Image Understanding*, vol. 249, p. 104160, Dec. 2024.