2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Absolute Pose Estimation for a Millimeter-Scale Vision System

Derin Ozturk[1], Zilin Wang[1], E. Farrell Helbling[1]

***Abstract*— Vision is an important component of robotic perception systems due to the rich information provided by high resolution image sensors, but computer vision algorithms can be computationally expensive and ill-suited to resource-constrained robotic systems. Here, we present a mm-scale vision system capable of performing absolute pose estimation at 16.5 FPS. This novel vision system uses a commercial-off-the-shelf sensor and microcontroller unit, as well as planar light-based landmarks in the environment to simplify feature detection. We exploit the structure of the planar pose problem to reduce algorithmic complexity and improve latency and energy consumption through software-, processor-, and hardware-in-the-loop testing. The end-to-end system consumes 49 mA of current and computes absolute pose estimates within 15 mm over a number of reference trajectories.**

I. Introduction

Micro-aerial vehicles (MAVs) that can autonomously explore natural and man-made environments could have major impacts in agriculture, environmental monitoring, inspection, and reconnaissance operations [1]. Accurate pose estimation enables autonomous navigation, allowing a robot to follow reference trajectories and calculate relative position to objects in a scene. Camera-based pose estimation, such as visual SLAM and visual inertial odometry, has been a cornerstone in recent advances in autonomous robots [2], [3], [4], [5], due to the information density of high-resolution, large field of view sensors.

However, MAVs (length$<$ 10 cm, mass $\sim$ 10 g, and power$<$ 1 W) have limited payload capacity for onboard sensors and limited compute capability to process vision data due to the severe size, weight and power (SWaP) constraints at this scale. Commercial-off-the-shelf (COTS) microcontroller units (MCUs) that meet these SWaP constraints have lower quantities of available memory and slower clock frequencies than general purpose CPUs. Recent solutions have deployed customized hardware for camera-based pose estimation [6] and deep-learning solutions for navigation and collision avoidance [7]. At even smaller scales, sensing and estimation algorithms for high-resolution vision systems are primarily run on offboard computers [8], [9]. One solution to simplify the pose estimation problem is to use light-based landmarks to create sparse image data [10] (see Fig. 1 for a conceptual rendering).

In this work, we develop a mm-scale vision system that can perform absolute pose estimation at 16.5 FPS and demonstrate the feasibility of commercially-available components for vision-based pose estimation. Using planar, light-based landmarks simplifies both the feature detection and pose estimation algorithms, reducing latency and energy consumption of our compute system. In addition to making simplifications to our external environment, we leveraged hardware capabilities to process subsets of the images to increase feedback rate, and implemented optimized versions of the Direct Linear Transform (DLT) method and Singular Value Decomposition (SVD) solver for an ARM-Cortex M4. To validate our system, we conducted detailed latency and energy characterizations for both the baseline algorithm and optimized system, demonstrating a 4$\times$ improvement in latency. Using a custom-built evaluation board (see Fig. 1), our vision system can accurately estimate pose within 15 mm.

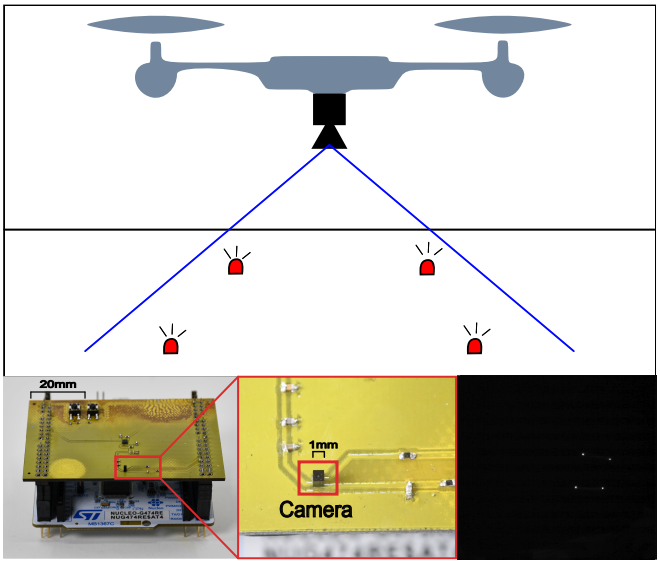

Fig. 1. Top: Conceptual illustration of a mm-scale monocular camera on an cm-scale MAV. The downward-facing camera detects LED landmarks in the environment and estimates absolute pose. Bottom left: the embedded system includes a custom-built sensor evaluation board mounted on an STM32G474 development board. Bottom center: enlarged view of the vision sensor on the board. Bottom right: still image from the vision sensor

Our contributions include: (1) an end-to-end demonstration of mm-scale vision system capable of absolute pose estimation on ARM-Cortex M4 class microcontrollers, (2) a novel absolute pose estimation system that exploits both the simplicity and resulting problem structure that is provided by planar, light-based landmarks and our hardware specifications, and (3) comprehensive latency and energy characterization of our embedded system. The paper is organized as follows: Section II details the hardware specifications; Section III discusses the system design and implementation; Section IV presents latency, energy consumption, and accuracy measurements; and Section V provides current limitations and areas for future work.

[1] School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14850. Contact e-mail farrell@cornell.edu

TABLE I

NANEYEC SPECIFICATIONS

Characteristic	Typical Value
Resolution	320×320
Diagonal FOV	$120°$
Shutter Type	Rolling Shutter
FPS	$0 - 58$ FPS
Focal Length	$367\ \mu m$
DOF	$4\ mm - \infty$
Focal Ratio	f/4
Pixel Packet Size	12 bits

TABLE II

ESTIMATED MASS AND POWER OF PROPOSED SYSTEM

Components	Size (mm^3)	Mass (mg)	Power (mW)
NanEyeC	$1 \times 1 \times 1.3$	7	9.7
STM32G474CB	$4.02 \times 4.27 \times 0.6$	2.4	115.5
LDO	$2 \times 2 \times 0.8$	21	–
Discretes	–	12	–
Total	–	42.4	125.2

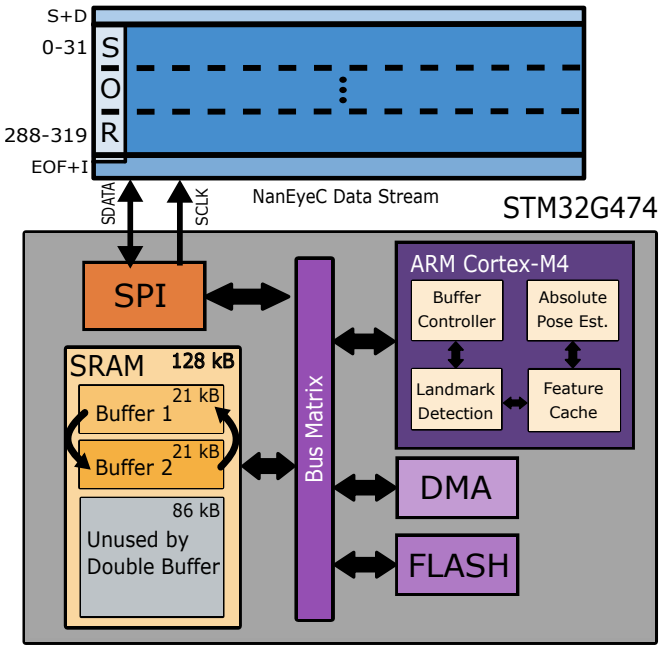

Fig. 2. The compute system architecture of our mm-scale vision system with the associated NanEyeC datastream for a single image. The datastream consists of 326 rows. Four rows for sync and delay (S+D), 320 rows for image readout, and two rows for end of frame and interface (EOF+I). Every row consists of 328 columns (320 pixels in the image and 8 Start of Row headers). The MCU communicates with the camera over SPI, the double buffer strategy is managed by the DMA, and the processor manages the images buffers, detects and caches landmarks, and computes absolute pose.

II. HARDWARE SELECTION

We have three hardware subsystems, a mm-scale high-resolution camera, an MCU to communicate with the sensor and compute pose estimates, and LED beacons to mark the environment. When selecting hardware we not only pay careful consideration to meeting the SWaP constraints of MAVs, but also prioritize ease of integration, memory capacity, and computation power for the algorithms discussion in Section III.

A. Camera

Because the vision system is intended for cm-scale MAVs, we need to select a camera that meets the SWaP constraints and can be easily integrated with a COTS MCU. The NanEyeC (ams OSRAM) is one of the smallest full camera modules, integrated with pre-installed optics in a $1.3\ mm^3$ package, suitable for insect-scale MAVs [8]. The camera supports single-ended interface mode, which requires a four-wire connection to supply power and communicate over Serial Peripheral Interface (SPI). While the camera utilizes a rolling shutter, we can mitigate motion blur by adjusting the exposure time and frame rate. These parameters can be defined using programmable registers on the NanEyeC or adjusted using the SPI clock rate on the MCU. See Tables I and II for a full list of specifications. Another camera that meets the SWaP constraints is the HM01B0 (HIMAX), as used in [9]. While this camera consumes less power ($2\ mW$) and can support higher frame rates, the 16-wire interface complicates integration with at-scale MCUs.

B. Microcontroller and supporting circuitry

For this system, we needed to select an MCU with SPI peripherals for camera communication, sufficient RAM to store reasonable partitions of the image (i.e., 32 rows), and direct memory access (DMA) controllers to store and manipulate image data without interrupting the processor. We chose the STM32G474 (STMicroelectronics), which has an ARM Cortex-M4 processor and a floating point unit (FPU) in a $4\ mm$ WLCSP. Our design may be easily ported to other MCUs with ARM Cortex-M4 processors, especially those from the STM32 family.

We use a linear voltage regulator (TPS71701, Texas Instruments) and supporting passives to supply a low-noise voltage signal to the camera. All components are listed in Table II with their associated size, mass and power parameters. We estimated the power consumption of the MCU using the current consumption provided in the datasheet for our specific operating frequency and multiplying by our operating voltage of $3.3\ V$. We can reduce our power consumption by including a level shifter on the camera data lines and operate the MCU at a lower voltage.

C. Light-based Landmarks

Similar to the system presented in [10], which uses IR LEDs, we place white LEDs in our experimental setup to simplify landmark detection and reduce computation cost. Because the exposure time of the camera is short, we are able to easily detect direct light sources in the environment. We arrange four LEDs on the ground plane within the camera's FOV (as seen in Fig. 1).

III. SYSTEM DESIGN AND IMPLEMENTATION

The mm-scale vision system needs to collect data from the camera, detect landmarks, save relevant features, and

compute absolute pose estimates. This system-level design is informed by the hardware selected in the previous section. We developed custom drivers to read image data and leveraged the DMA controller to enable real-time operation. Furthermore, we implemented a linear pose estimation algorithm for planar landmarks and exploited the structure and sparsity of the resulting matrices to reduce latency.

A. Camera Driver

During operation, the NaneyeC sensor loops through three modes: sync and delay, where we perform sensor synchronization; readout, where we receive image data; and interface, where we can update configuration registers to change the frame rate and exposure time before the next frame. Each mode lasts a specific number of cycles, defined as row periods, and transmits 'rows' of data. During our operation, sync and delay occurs for four row periods, readout occurs for 320 row periods (corresponding to the number of rows in the image), and interface occurs for two row periods. Thus, the sensor sends 326 'rows' of data for every image. Because the MCU has insufficient memory to store a full images, we developed custom SPI drivers to set up DMA transactions and double buffer the data stream (see Fig. 2 for a diagram of the compute system). The DMA controller manages incoming data from the sensor and stores it in the first buffer (21 KB in SRAM). When the buffer fills, the DMA controller issues an interrupt signaling transfer completion. In this interrupt, we save the 'row' location of the datastream from NaneyeC in a buffer controller, and the CPU can begin performing landmark detection on the first rows of the image. In parallel, the DMA controller stores sensor data from the next rows of the image in the second buffer (21 KB in SRAM). This process continues until all image 'rows' are received. The buffer controller signals the end of frame and starts the pose estimation algorithm. Because landmark detection is performed in parallel to image data collection, we can reduce the time it takes to obtain a pose estimate (see Fig. 3).

B. Landmark Detection

Images captured by the NaneyeC are sparse, with bright peaks centered on the LEDs, as can be seen in Fig. 4. Our landmark detection algorithm is functionally similar to that of [10], but instead of processing on the entire image, we only process on the active data buffer. We first perform a threshold test, marking pixels as potential peaks if their intensity is greater than half of the maximum intensity. The algorithm compares the intensity of every potential peak and selects the first largest peak within a predefined radius. We store the pixel locations of the largest peaks (landmarks) in a cache data structure. At the end of frame, if the number of landmarks exceeds the number of known landmarks in the environment, we select the landmarks with the largest intensity.

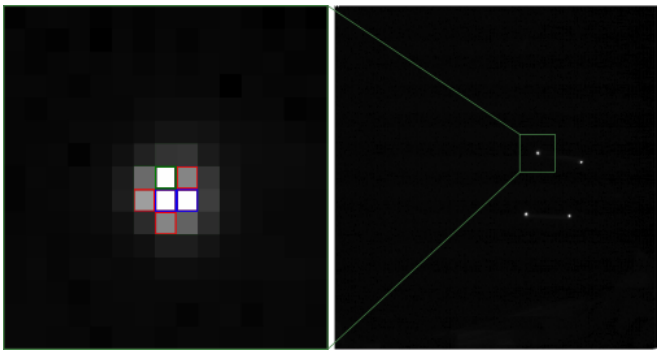

Fig. 4. One frame of the NanEyeC with an exposure time $\sim 2-6\text{ms}$ (right) and a detailed view of the top-left landmark highlighted in green (left). Our landmark detection algorithm selects the first largest peak it finds within some radius. Pixels with colored borders pass the threshold test. The pixel outlined in green is the detected landmark, blue pixels have matching intensity but do not replace the landmark, and red pixels have lower intensity than the maximum.

C. Pose Estimation

Because of the computational limitations of the onboard microcontroller, we chose a linear pose estimation algorithm. We selected the Direct Linear Transform (DLT) to estimate the homography of coplanar points projected onto a planar image, a well known solution to the Perspective-n-Points (PnP) problem. While recent work on P3P [11], [12], can provide solutions for the minimal three point case, we prefer a generic PnP solution that can be extended to any number of landmarks and applied to two-view relative pose [13]. To simplify the system, we eliminate 2D-3D point correspondences, so we need to assume the orientation of the landmarks in the image matches the orientation in 3d space. Furthermore, the linear system to be solved in the DLT

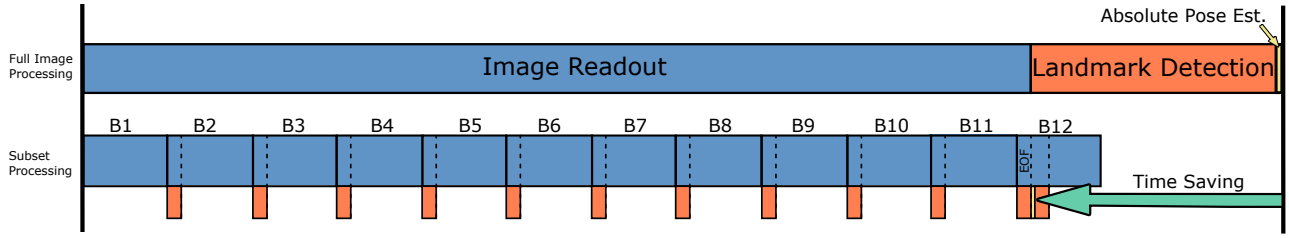

Fig. 3. Full Image Processing vs Subset Processing. The buffer in Fig. 2 stores 32 rows of image data. Because each image contains 326 rows, we store 11 buffers (B1-B12) during a single image read cycle. When the buffer is full, the CPU begins the landmark detection algorithm (orange). Because these processes (image read, landmark detection) are done in parallel in the subset processing, absolute pose estimates can be determined faster than the reading and processing the full image.

formulation exhibits a structure than can be exploited [14], and by carefully selecting the Singular Value Decomposition technique used to solve this system, we can further improve computational performance. Finally, the estimated homography can be used to develop a controller based on visual servoing in future work [15], [16].

1) Direct Linear Transform: Landmarks, defined as $\mathbf{v}_i^W \in \mathbb{R}^3$ lie on a plane P, reducing the 3D world coordinates to a 2D subspace $\mathbf{v}_i \in \mathbb{R}^2$. For each landmark, there is an associated image coordinate, represented by a homogeneous, normalized camera pixel coordinate $\mathbf{v'}_i = (x'_i, y'_i, 1) \in \mathbb{R}^3$. Camera pixel coordinates are normalized into image coordinates using the camera calibration matrix and distortion coefficients, which are computed offline. Both the 2D and 3D points must also be isotropically normalized. We store the similarity transformations to rescale the estimated homography [13].

For each image-world point pair $v' \leftrightarrow v$, we are given the following two linearly-independent equations:

$$\mathbf{A}_i\mathbf{h} = \begin{bmatrix} \mathbf{0}_{1\times3} & \mathbf{v}_i^T & -y'_i\mathbf{v}_i^T \\ \mathbf{v}_i^T & \mathbf{0}_{1\times3} & -x'_i\mathbf{v}_i^T \end{bmatrix} \mathbf{h} = \mathbf{0}, \tag{1}$$

where the vector $\mathbf{h} \in \mathbb{R}^{9\times1}$ is the flattened homography matrix ($\mathbf{H}$) that represents the 2D transformation from an image coordinate v'_i, to a planar landmark coordinate v_i. For N point pairs, we have the linear system $\mathbf{A} \in \mathbb{R}^{2N\times9}$, which has rank 8 and can be solved using a minimum of 4 point pairs. We can solve for $\mathbf{h}$ by computing the right singular vector corresponding to the smallest singular value of $\mathbf{A}$.

2) Exploiting Structure and Sparsity: We can exploit the sparsity and structure of $\mathbf{A}$, as described in [14]. If we assume that the points are not all collinear through the origin, we can simplify Eq. 1 as a linear system that is only a function of $\mathbf{h}_{7-9}$:

$$\mathbf{A}_{7-9}^{\perp}\mathbf{h}_{7-9} = \mathbf{0} \tag{2}$$

where $\mathbf{A}_{7-9}^{\perp}$ is the projection of the matrix $\mathbf{A}_{7-9}$ onto the orthogonal complement of $\mathbf{A}_{1-6}$. This new system has shape $2N \times 3$, and the rest of the elements of the homography matrix, $\mathbf{h}_{1-6}$ can be found via backsubstitution relations [14].

3) Singular Value Decomposition: To estimate the homography $\mathbf{h}$, we need to compute the smallest right singular vector of $\mathbf{A}$. Bidagonalization and Jacobi methods are two dominant techniques to compute the singular value decomposition (SVD) of a matrix. While bidiagonalization methods are generally known to be faster and have lower FLOP counts than Jacobi methods [17], [18], recent work [19] demonstrated that Jacobi methods outperform bidiagonalization on ARM Cortex-M4 processors. We selected the One-Sided Jacobi method with row-cyclic ordering, which computes the SVD of a matrix $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$ for $\mathbf{A} \in \mathbb{R}^{m\times n}$ where $m \geq n$, $\mathbf{U} \in \mathbb{R}^{m\times n}$ and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$, and $\mathbf{V}^H \in \mathbb{R}^{n\times n}$ as follows:

$$\mathbf{A}_0 = \mathbf{A}, \quad \mathbf{A}_{k+1} = \mathbf{A}_k\mathbf{J}_k, \quad \mathbf{A}_k \to \mathbf{U}\mathbf{\Sigma} \tag{3}$$

$$\mathbf{V} = \mathbf{J}_0\mathbf{J}_1\cdots\mathbf{J}_k \quad \text{as } k \to \infty \tag{4}$$

where the Jacobi rotations ($\mathbf{J}$) are chosen by solving the 2×2 symmetric Schur decomposition [17]. Each Jacobi rotation $\mathbf{J}_k$ rotates two columns of $\mathbf{A}$ and is applied in a sweep across all possible column pairings $(1,2),(1,3),\ldots(1,n),(2,3),\ldots(n-1,n)$. With each sweep, $\mathbf{A}_k^T\mathbf{A}_k$ gradually becomes diagonal. Sweeps are performed iteratively until convergence, which is achieved when the off-diagonal elements of the Gram matrix are reduced to within a small predefined tolerance, ϵ, effectively making the Gram matrix approximately diagonal. Before performing a rotation, we check if the two columns are close to orthogonal within some tolerance, and the algorithm converges once we skip every rotation in a sweep. By simplifying the $\mathbf{A}$ from 9 columns in Eq. 1 to 3 columns in Eq. 2, we reduced the number of possible column pairs from 36 to 3. In our implementation, we further reduce computational complexity by unrolling dot product loops, limiting computation of column norms to only those that have been rotated in a previous iteration of the sweep, and updating the minimum singular value index during the iterative process, which avoids computing another set of column norms after convergence.

4) Extracting Pose from Homography: We rescale the estimated homography using the similarity transformations discussed in Section III-C.1. We then reshape the vector into a 3×3 matrix to extract the rotation matrix and translation vector.

IV. Results

We need to accurately profile the computational load and current consumption of the embedded perception system to estimate the impact on runtime for future applications. Existing low-order models that involve selective cycle counting [20], [21] can greatly undercount the number of cycles, and as a result current consumption, as they neglect load and store operations, and loading machine instructions from FLASH memory. To fully characterize our embedded system, we conduct software-in-the-loop (SIL), processor-in-the-loop (PIL), and hardware-in-the-loop (HIL) experiments using both simulated and real image data. These experiments functionally test the algorithms, quantify algorithm latency and current consumption on the target processor, and evaluate the full system with sensor peripherals to account for the effects of interrupts and bus activity on latency. Additionally, we test a prototype of our end-to-end system inside a motion capture arena (Vicon Vantage cameras, Tracker 3.6) to validate our pose estimates. All embedded code in these sections is compiled using the GCC compiler for our microcontroller with the O3 optimization flag. We utilize the ARM CMSIS-DSP header library as much as possible for vector and matrix operations.

A. Software in the Loop Experiments

We used the SIL framework to test the functionality of the landmark detection algorithm, the baseline DLT (Eq. 1), the optimized DLT-HO (Eq. 2), and the OSJ implementation (Eqs. 3) from Section III. We also used it to create

datasets for later PIL experiments, and estimate convergence statistics of the OSJ algorithm. The framework includes Monte Carlo simulations, with randomized camera poses and planar landmarks [22], alongside a high-level Python implementation of the vision pipeline. We randomly generate camera poses and a N-sided polygon's center with coplanar vertices. Bearing vectors from camera center to model points are parameterized by Gaussian noise. We ran 5000 iterations of the simulator and varied the number of landmarks ($N = \{4, 6, 8, 16, 32, 64, 128\}$), landmark pattern size (side length $\sim 10 - 40$ cm), and noise levels to assess the convergence properties (sweeps and rotations) of the OSJ method for the baseline DLT ($2N \times 9$) and the optimized DLT-HO ($2N \times 3$) methods.

For the baseline DLT, we measured 5 sweeps and 87.8 rotations for the minimal $N = 4$ case and 4 sweeps and 130.4 rotations for the cases where $N > 4$. The number of sweeps and rotations reduced when using the optimized DLT-HO to 4 sweeps and 6.6 rotations for the minimal $N = 4$ case and 3 sweeps and 4.9 rotations for the cases where $N > 4$. Generally, we expect a reduction in number of rotations when using the reduced three-column system because instead of 36 possible column pairs per sweep, there are only 3. However, we also observed a reduction in the number of sweeps required for convergence. When we compute the L1 norm of the diagonal and off-diagonal elements of the DLT-HO method, we notice that the the Gram matrix, $(\mathbf{A}_{7-9}^{\perp})^T \mathbf{A}_{7-9}^{\perp}$, is close to orthogonal, which can lead to faster convergence of the OSJ method [18]. In this sense, we have preconditioned the system for faster convergence.

B. Processor in the Loop Experiments

Our PIL testing serves two purposes, functionally testing our vision pipeline on the target MCU, and baseline testing the latency and current consumption of the pipeline with no sensor I/O in the loop. Tests are performed automatically with a host computer that transmits data to a device under test (DUT) over UART using a USB to UART serial cable (C232HD, FTDI). Our framework is modular, we can use data from our SIL experiments or captured images from the camera. To measure power consumption, we use a X-NUCLEO-LPM01A expansion board (ST Microelectronics), which can measure the dynamic current consumption of STM32 Nucleo development boards up to $50\,\mathrm{mA}$ and trigger acquisition with a GPIO pin. We measure computational latency with a logic analyzer (Saleae Logic 2) connected to a GPIO, which is triggered HIGH when entering a function under test and is driven LOW when exiting. The logic analyzer and the LPM01A are synchronized through the trigger signal. For all tests the microcontroller is running at its maximum clock speed of 170 MHz.

First-order approximations [20], [21] count the number of single-cycle instructions (i.e., multiplication, addition, and subraction) and multi-cycle instructions (i.e., division, sine, and cosine) in the algorithm to estimate overall latency and energy consumption. These methods underestimate the number of cycles, as they do not include load and store instructions or the time it takes to fetch machine instructions from FLASH memory. Using this method, we estimate the number of cycles for the OSJ SVD algorithm on two matrices of size 8×3 and 256×3 for the minimal case of four landmarks. Using the implemented OSJ SVD in C, we can count the number of cycles:

$$\begin{aligned} \text{cycles} = &(2M - 1) + ((N(N-1)) - 1) \\ &+ i\,(2M + s + 2) \\ &+ r\,(7 + 2s + 2d + 12M + 6N) \end{aligned} \tag{5}$$

where M is the number of rows of the matrix, N is the number of columns, i is the number of sweeps, r is the number of rotations, s is the multi-cycle cost of a floating-point square root (14 cycles), and d is the multi-cycle cost of a floating point division (14 cycles). The first two terms in the sum are initial pre-computations to find $\|\mathbf{A}_j\|^2$, for each column $\mathbf{A}_j$, as well as the dot product of each pair of columns for a full sweep, $\mathbf{A}_j \cdot \mathbf{A}_k$, as described in III-C.3. The third term computes a new dot product, $\mathbf{A}_j \cdot \mathbf{A}_k$, for the current column pair in the sweep, as well as executes a convergence criteria check. The final term encapsulates the Schur decomposition, the Jacobi rotation, and updates the column norms for $\mathbf{A}_j$ and $\mathbf{A}_k$. We estimate the number of cycles to be $1,504.2$ for the 8×3 matrix and $27,654.1$ for the 256×3 matrix. If we assume that we are operating at the maximum clock frequency and use the average current consumption provided by the datasheet, we can also estimate energy consumption. We calculate 0.61 µJ for the 8×3 matrix and 11 µJ for the 256×3 matrix.

Using the logic analyzer, we measured that the OSD SVD algorithm completes in 5743.5 cycles for the 8×3 matrix and $74,984.14$ cycles for the 256×3 matrix. Thus, for every estimated single-cycle operation, we fail to account for 4 cycles for the 8×3 matrix, and approximately $1.5 cycles$ for the 256×3 matrix. Additionally, we measured the total energy consumption of the algorithm and found 4.21 µJ for the 8×3 matrix and 59.5 µJ for the 256×3 matrix. The first-order estimates had a relative error of 79.84% and 80.87%, respectively. This error does not match the error we see in latency, and we predict that the increased energy is due to the FPU. Because the first-order estimates have significant error compared to measured values, we characterize the latency and energy for our pose estimation system running on the ARM-Cortex M4 processor.

We measure the average latency of the landmark detection algorithm, the OSJ SVD, and the DLT and DLT-HO pose estimation algorithms for $N = \{4, 6, 8, 16, 32, 64, 128\}$ landmarks (results listed in Table III. We captured 100 images with the NanEyeC for $N = \{4, 6, 8\}$ landmarks and used this data as input to the landmark detection algorithm. Landmark detection has the longest latency (1.27 ms per buffer), and the number of landmarks has little effect on the overall latency of landmark detection. Therefore, this latency is primarily do to iterating over the sensor data and a result of the buffer

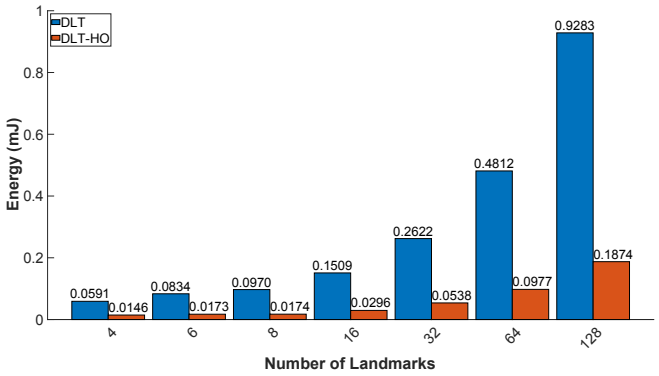

Fig. 5. Average energy consumption for both the DLT and DLT-HO method as described in III-C using the One-Sided Jacobi SVD algorithm as described in III-C.3.

size.

We used simulated data to measure latency for the OSJ SVD algorithms as well as the pose estimates for all N. When we compare the latency for the SVD of the DLT system and the DLT-HO system, we see a $12.02\times$ improvement in latency with the OSJ method for the minimal case of four landmarks and greater than $15\times$ improvement for the case with more landmarks. These values are greater than the reported $11\times$ improvement for the minimal case and $5\times$ improvement for arbitrarily large N, as previously reported [14]. We attribute this performance increase to our careful implementation of the OSJ algorithm, as described in Section III-C.3, and the reduction in the number of sweeps and rotations due to preconditioning, described in the SIL results.

TABLE III
PIL LATENCY (μS) COMPARISON WITH VARYING N

Workload	**4**	**6**	**8**	**16**	**32**	**64**	**128**
LD (per buffer)	127.5	125.5	125.5	–	–	–	–
SVD (2Nx9)	406.4	567.5	662.6	1132.7	1983.1	3640.8	6866.4
SVD (2Nx3)	33.8	31.6	37.3	70.2	120.5	226.2	441
Speedup	**12.02x**	**17.96x**	**17.76x**	**16.13x**	**16.46x**	**16.10x**	**15.57x**
DLT	475.4	661.6	763.2	1171.4	2019.9	3702	7090.4
DLT-HO	115	132.6	152	239.1	412.5	758.3	1473.4
Speedup	**4.13x**	**4.99x**	**5.02x**	**4.9x**	**4.9x**	**4.88x**	**4.81x**

The baseline DLT latency includes the time to undistort points, normalize points isotropically [13], prepare the linear system for the SVD, perform the SVD computation, and extract the pose from the homography. The DLT-HO method contains the additional calculation of $\mathbf{A}_{7-9}^{\perp}$ in addition to the aforementioned steps. The DLT-HO system shows significant improvement compared to the full DLT, showing a minimum $4.13\times$ speedup with four landmarks. Performance is better with more landmarks, suggesting that the preconditioning of the system is more beneficial with larger matrices. Using these results, we measure a 12.7 ms time savings by processing on subsets of the image (processing scheme shown in Fig. 3).

In addtion to quantifying latency for the various workloads, we also measured dynamic current consumption during landmark detection and pose estimation. We measure an average current consumption of 35 mA while performing landmark detection, and 37 mA to perform the DLT-HO computation with $N = 4$ landmarks and 39 mA for $N = 128$ landmarks. We can attribute this observed increase in current consumption to the FPU, which is used to compute the pose, but not to detect landmarks. Additionally, we observed larger spikes of current consumption at larger N, up to 46.7 mA for 128 landmarks, suggesting that increased usage of the FPU can result in increased power consumption. Figure 5 shows the measured average energy consumption for the DLT and DLT-HO methods for all N. The energy savings between the DLT and DLT-HO methods and the relative performance matches the latency improvement reported in Table III.

C. Hardware in the Loop Experiments

We utilize the HIL setup to quantify the effects of bus congestion on latency, determine the maximum frame rate we can support to achieve real-time operation, and measure current consumption when SPI and DMA are continuously servicing the incoming pixel stream.In the HIL setup, we mock the NanEyeC with a second microcontroller (STM32H7A3), which has sufficient RAM (1.2 MB to store an image for testing and can be easily integrated with the STM32G474. We reformatted experimental images taken with the NaneyeC to match the data stream shown in Fig. 2 and stored these rows in the STM32H MCU.

As expected, bus traffic has no significant effect on latency, we measured a 3% increase in time to compute the pose estimate. The maximum frequency we can run the vision system is 16.5 FPS. At faster speeds, the DMA receives the next frame before the CPU calculates the previous pose estimate. This speed is primarily limited by the computational latency of the landmark detection algorithm. In future, we will track regions of interest centered on the landmarks to decrease this latency. While these frame rates are significantly slower than proprioceptive sensor refresh rates used in controlled flight of cm-scale MAVs [23], the low mass of the camera enables use in a future multi-sensor suite.

At 16.5 FPS, the current consumption reported in the previous section increases by approximately 9 mA when the SPI and DMA peripherals are continuously active, resulting in a total average current consumption of approximately 45 mA. For cm-scale MAVs, we will need to investigate further strategies to decrease current consumption. The STM32U575, which is built on the new ARM Cortex-M33 architecture and similar in size and mass, supports DMA and SPI operation while in deep-sleep and reports one-third of the current consumption of the STM32G474 [24]. At 16.5 FPS, it takes 6 ms for the buffer to fill with image data, thus the processor can go into deep sleep mode for 4.7 ms after the landmark detection algorithm finishes running after 1.2 ms.

D. Demonstrating the End-to-End System

We manufactured a custom printed circuit board (PCB) to mount the camera module and associated peripherals

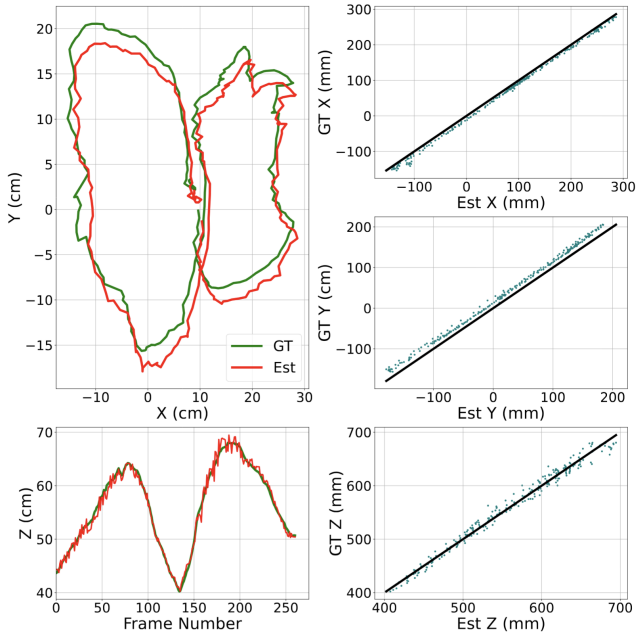

Fig. 6. Trajectory Demonstration. Left: position data of a single figure eight trajectory with ground truth (green) and experimental measurements (red). Right: Single axis data from the same trajectory (blue dots), the black line has a slope of one, dots that fall on this line indicate that the estimated position matches the ground truth measurement.

to the STM32 development board (see Fig. 1). The PCB was patterned on copper-clad FR4 using the Protolaser U4 (LPKF). We assembled a housing for the electronics and added motion capture reflective markers to track the object in the arena. We placed four LED markers in a square configuration with a side length of 20 cm on the ground plane. A GPIO pin sends pulses at the beginning of every frame to the motion tracking computer to synchronize the estimated pose from our system with the motion capture data. A second 'host' computer received the landmark and pose data from the MCU over UART at the end of the acquisition.

We moved the camera by hand through the tracking volume (side length $\sim$ 75 cm) and recorded trajectories for six experiments. During testing, we restrict the camera's motion around its optical ($Z-$) axis, as we eliminate 2D-3D point correspondences to simplify our algorithm (discussed in Section III-C). Our average error across all six trajectories is 7.06 ± 6.55 mm, 12.85 ± 4.83 mm, and 5.39 ± 6.40 mm in translation (x, y, z, respectively) and $1.58 \pm 1.36°$, $0.74 \pm 0.61°$, $1.48 \pm 1.53°$ in rotation (θ_x, θ_y, θ_z). Figure 6 shows the pose estimates for one trial where we moved the camera in a figure eight trajectory and varied the height from the ground plane. The estimates for rotation and height are in alignment with the motion capture data; however the x position is consistently underestimated, and the y position is consistently overestimated. This can be due to the misalignment of the defined object in the motion capture software.

V. Conclusion

In this work, we present a mm-scale COTS vision system capable of estimating absolute pose at 16.5 FPS. Through careful hardware selection, the system meets the mass requirements of cm-scale MAVs. To reduce computational load, we simplify feature detection with light-based landmarks in the environment. This allows us to exploit the structure and sparsity of the DLT algorithm and use the OSJ SVD method to further reduce latency and energy consumption. The system presented here provides a strong case for utilizing vision on cm-scale MAVs, and other resource-constrained systems. The vision system can be used as an exteroceptive sensor for task-specific deployments or implemented in a Kalman Filter to correct drift of faster, noisier sensors to stabilize pose estimates for longer flights.

In addition to implementing the system on the STM32U575 and analyze the energy savings in deep sleep mode, we will miniaturize the components to build an at-scale hardware platform. We plan on further testing our vision system in larger environments, as we are currently limited by the size of our motion capture arena. We expect the error to increase as the distance from the landmarks increases, especially as we do not find landmarks with subpixel accuracy. We can also extend the work to incorporate feature matching, in order to make 2D-3D point correspondences. We would also like to extend our system to perform two-frame, relative pose estimation using a similar DLT approach presented in [13].

Acknowledgment

The authors would like to thank Hang Gao, Cameron Urban, and Julie Villamil for assistance with PCB fabrication.

References

[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *nature*, vol. 521, no. 7553, pp. 460–466, 2015.

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[3] J. Jeon, S. Jung, E. Lee, D. Choi, and H. Myung, "Run Your Visual-Inertial Odometry on NVIDIA Jetson: Benchmark Tests on a Micro Aerial Vehicle," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5332–5339, Jul. 2021.

[4] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny, "Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking," *Journal of Sensors*, vol. 2021, no. 1, p. 2054828, Jan. 2021.

[5] H. Oleynikova, C. Lanegger, Z. Taylor, M. Pantic, A. Millane, R. Siegwart, and J. Nieto, "An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments," *Journal of Field Robotics*, vol. 37, no. 4, pp. 642–666, Jun. 2020.

[6] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, Apr. 2019.

[7] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, Oct. 2019.

[8] S. Mange, E. F. Helbling, N. Gravish, and R. J. Wood, "An actuated gaze stabilization platform for a flapping-wing microrobot," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore: IEEE, May 2017, pp. 5409–5414.

[9] V. Iyer, A. Najafi, J. James, S. Fuller, and S. Gollakota, "Wireless steerable vision for live insects and insect-scale robots," *Science Robotics*, vol. 5, no. 44, p. eabb0839, Jul. 2020.

[10] M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza, "A monocular pose estimation system based on infrared LEDs," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, pp. 907–913.

[11] Y. Ding, J. Yang, V. Larsson, C. Olsson, and K. Åström, "Revisiting the P3P Problem," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vancouver, BC, Canada: IEEE, Jun. 2023, pp. 4872–4880.

[12] M. Persson and K. Nordberg, "Lambda Twist: An Accurate Fast Robust Perspective Three Point (P3P) Solver," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, vol. 11208, pp. 334–349.

[13] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[14] M. J. Harker and P. L. O'Leary, "Computation of Homographies," in *Procedings of the British Machine Vision Conference 2005*. Oxford: British Machine Vision Association, 2005, pp. 30.1–30.10.

[15] E. Malis and M. Vargas, "Deeper understanding of the homogrpahy decomposition for vision-based control," *Bulletin of Sociological Methodology/Bulletin de Méthodologie Sociologique*, vol. 37, no. 1, pp. 55–57, Sep. 2007.

[16] S. Benhimane and E. Malis, "Homography-based 2D Visual Tracking and Servoing," *The International Journal of Robotics Research*, vol. 26, no. 7, pp. 661–676, Jul. 2007.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, fourth edition ed., ser. Johns Hopkins Studies in the Mathematical Sciences. Baltimore: The Johns Hopkins University Press, 2013.

[18] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale," *SIAM Review*, vol. 60, no. 4, pp. 808–865, Jan. 2018.

[19] M. Alessandrini, G. Biagetti, P. Crippa, L. Falaschetti, L. Manoni, and C. Turchetti, "Singular Value Decomposition in Embedded Systems Based on ARM Cortex-M Architecture," *Electronics*, vol. 10, no. 1, p. 34, Dec. 2020.

[20] Y. P. Talwekar, A. Adie, V. Iyer, and S. B. Fuller, "Towards Sensor Autonomy in Sub-Gram Flying Insect Robots: A Lightweight and Power-Efficient Avionics System," in *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, PA, USA: IEEE, May 2022, pp. 9675–9681.

[21] S. Fuller, Z. Yu, and Y. P. Talwekar, "A gyroscope-free visual-inertial flight control and wind sensing system for 10-mg robots," *Science Robotics*, vol. 7, no. 72, p. eabq8184, Nov. 2022.

[22] L. Kneip and P. Furgale, "OpenGV: A unified and generalized approach to real-time calibrated geometric vision," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, pp. 1–8.

[23] E. Farrell Helbling and R. J. Wood, "A Review of Propulsion, Power, and Control Architectures for Insect-Scale Flapping-Wing Vehicles," *Applied Mechanics Reviews*, vol. 70, no. 1, p. 010801, Jan. 2018.

[24] "STM32U575xx," ST Microelectronics, Datasheet, 2024.