2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

PicoVO: A Lightweight RGB-D Visual Odometry Targeting Resource-Constrained IoT Devices

Yuquan He[1,2], Ying Wang[1,2], Cheng Liu[1,2], Lei Zhang[1,2,3]

Abstract— **Ego-motion estimation with 3D perception using visual odometry (VO) is known to be robust and economical among the existing odometry techniques. However, existing VO solutions are typically both computation intensive and memory intensive, which dramatically inhibits their deployment in IoT platforms such as robotic vehicles and handheld devices mostly equipped with resource-constrained MCU-level processors. To enable real-time and high-quality VO on these scenarios with thrifty resource budgets, we investigate state-of-the-art edge-based VO (EBVO) and propose an optimization framework called PicoVO that can greatly reduce the amount of computation as well as the memory footprint from the perspectives of both algorithm and implementation. First of all, we revisit the key processing stages of EBVO and propose an EBVO-oriented lightweight edge detector in the pre-processing stage, a sparse-to-dense processing scheme in the tracking stage, and a lightweight key-frame management in the post-processing stage. In addition to the algorithmic optimization, we further develop a dedicated quantization scheme particularly for the 3D feature calculation and Levenberg-Marquardt (LM) solver that are critical to the memory footprint and computation requirements of PicoVO. Evaluation on realistic RGB-D benchmark datasets is conducted on NUCLEO-F767ZI equipped with a 216MHz Cortex-M7 MCU and 512KB RAM. It reveals that PicoVO achieves 33fps@320x240 with high tracking precision comparable to state-of-the-art VOs on PC.**

I. Introduction

The market of low-end ubiquitous-computing terminals based on microcontroller unit (MCU) is growing rapidly at an unprecedented rate [1], stimulating the development of low-power and low-cost applications in the fields of autonomous navigation, 3D reconstruction, and virtual/augmented reality. For such scenarios and applications as in robotic vehicles or hand-held devices, a fundamental technology required is to perform accurate ego-motion estimation with 3D environment perception, which falls in the category of Visual Odometry (VO). High-speed and high-quality VO is essentially required to match the throughput of a camera to initiate real-time interaction with human beings and the physical world. As the growing availability of inexpensive and accurate RGB-D sensors for somatosensory gaming and mobile devices, research in the field of RGB-D based VO has been rapidly evolving.

Researchers have been trying to improve the processing efficiency of either indirect (feature-based) [2] or direct (featureless) [3] VO approaches which usually entail computation-intensive operations, such as correspondence matching and dense image alignment. Recently a new class of edge-based VO (EBVO) [4] solutions emerge, and it combines indirect and direct approaches with a neat balance between computation complexity and tracking precision. However, prior solutions usually require a desktop PC or a 10W-level single-board-computer [5] with a gigahertz-speed multi-core processor and Gigabytes of RAM, being hopeless to fit into milliwatt-level platforms which are extensively utilized in battery-powered IoT devices. Such typical IoT devices may only include a weak processor and compact memory, e.g. a single-core 216MHz MCU with 512KB RAM [6], which can hardly upgrade to the desktop-level profile in a foreseeable future due to the dramatic slowdown of Moore's Law. Thereby, the key concern of achieving the low-power and low-cost VO capability on IoT devices is to alleviate its complexity from the aspects of computation and memory footprint, while still maintaining comparable tracking precision, which is also a general prerequisite of a high-quality robotic system.

To this end, we carefully study the critical procedures of the state-of-art EBVO and identify the performance bottlenecks in the pre-processing, tracking, and post-processing phases. On top of this analysis, we propose a lightweight EBVO framework called PicoVO to greatly reduce the computation and memory overhead from multiple angles. Specifically, this work makes the following contributions:

- We present a lightweight EBVO framework, PicoVO, to enable real-time and high-quality VO on milliwatts IoT devices with limited computing power and memory footprint.
- We investigate the algorithmic optimization for PicoVO in different processing stages. Specifically, we propose an EBVO-oriented lightweight edge detector to replace Canny detector in pre-processing, a sparse-to-dense scheme to replace the costly image pyramids in tracking, and a lightweight key-frame management to mitigate the computing overhead in post-processing.
- On top of the algorithmic optimizations, we further optimize the implementation for resource-constrained platforms with dedicated quantization particularly for the 3D feature calculation and the LM solver.
- We evaluate the PicoVO framework on a typical low-end IoT platform with a 216MHz MCU and 512KB RAM. The experiments reveal that PicoVO achieves 33fps@320x240 with high tracking precision comparable to the state-of-the-art VO solutions on PC.

II. Related Work

A. Visual Odometry

VO is used to determine the on-line position and orientation of the agent via image sensors. It is a key component in robotics and has been intensively explored in prior study. The

Y. He, Y. Wang, C. Liu and L. Zhang are with *Institute of Computing Technology, Chinese Academy of Sciences*[1], and *University of Chinese Academy of Sciences*[2]. L. Zhang is also with *Jeejio(Ningbo) Technology Co., Ltd*[3].

Email: {heyuquan20b, wangying2009, liucheng, zlei}@ict.ac.cn

various VO algorithms can be classified into three categories:

(1) Indirect (feature-based) approaches estimate the camera motion by extracting sparse features (e.g., SIFT, SUFT, ORB), matching their per-frame correspondence, and tracking them constantly. ORB-SLAM2 [2] is a functional Simultaneous Localization And Mapping (SLAM) system that can run in real-time on desktop-level computers. However, the processing stages of feature extraction and correspondence matching are the two well-known bottlenecks.

(2) Direct (featureless) approaches track a frame by image alignment [7] directly based on photometric residual, which relieves the costly feature manipulations. However, the intrinsic photo-consistency assumption [3] restricts the estimators to small inter-frame motions (typically only a few pixels). And for the sake of tracking quality, they may also entail a dense/semi-dense map, a sophisticated photometric model [8], and an illumination-robust metric [9], to enhance the image alignment. These inevitably induce considerable computing and memory requirements that hamper their use on low-end embedded systems.

(3) Hybrid approaches combine indirect and direct frameworks to achieve the best of both worlds, allowing lightweight features and retaining robustness. For instance, SVO [10] shows a lightweight software implementation that tracks sparse features in real-time on mobile computers, dual-core Cortex-A9@1.6GHz, of a drone. Recently, a new class of edge-based VO (EBVO) [4][11][12] emerged. They use edges as illumination-robust features and track them through image alignment upon Distance Transform (DT) [13]. On top of the classic DT, [14] proposes Nearest Neighbor Field with equivalent performance. EBVO can construct a semi-dense map with high throughput, has a larger convergence basin than direct approaches, and also reveals promising results [11] compared to sparse algorithms. Tracking robustness can be further promoted by fusing photometric errors [15], or using sliding-window [16] with some computation overhead.

B. Optimization-based Estimator

State estimation in a VO is generally achieved by iteratively minimizing the residuals of the observation model. The first-order sub-gradient method [17] is lightweight in each iteration, but it leads to slow convergence rate and accuracy penalty. Second-order optimizers [18] are more preferable, such as the Levenberg-Marquardt (LM) solver, which can converge faster and ensure higher accuracy. In this work, we further attempt to leverage fixed-point quantization techniques to mitigate the overhead of Jacobian and Hessian calculation in LM.

C. Hardware Accelerations

Speciallized hardware accelerators using FPGA or ASIC are typically used to boost the performance of resource-constrained platforms. The work in [19] accelerates ORB extraction and matching on a ZYNQ 7045 FPGA, which enables a faster on-board performance than a desktop PC. Navion [20] is a 2mW ultra low-power ASIC designed for end-to-end real-time inference of visual-inertial odometry. The Intel Movidius VPU [21] and Microsoft HPU [22] are integrated in off-the-shelf sensors and AR headsets for real-time 3D perception. However, these domain-specific accelerators typically suffer from high costs and restricted functionalities compared to the general-purpose DSP/MCU processors. In this work, we investigate software-oriented optimizations of a VO application for MCU platforms, thereby allowing for more flexible use cases.

III. Preliminaries of Edge-based VO (EBVO)

In this section, we review the basic observation model during the tracking phase of an EBVO which is validated in this work. A key-frame (KF) based tracking strategy is adopted to estimate the pose of the current frame against the KF. We aim to formulate the observation model and will stick to these notations throughout the paper.

Above all, we define the input and output. We denote the KF as F_k, and the input frame at each time step t as F_t which includes a grayscale image I_t and a depth image D_t. The pose of each frame is expressed by a Lie Group element $G \in \boldsymbol{SE(3)}$ which comprises a 3×3 rotation matrix $R \in \boldsymbol{SO(3)}$ and a 3×1 translation vector $T \in \mathbb{R}^3$. For the optimization on Lie group manifold, we use the 6 degree-of-freedom (DOF) Lie Algebra $\xi \in \mathfrak{se}(3)$ as a minimal representation of G, transformed by the exponential map $G_\xi = \exp(\xi^\wedge)$ and its inverse map $\xi_G = \log(G^\vee)$. The estimator outputs the pose difference between F_t and F_k denoted as ξ_{tk}.

In the observation model of EBVO, a set of edge pixels p_m is first extracted from I_t and then warped to p'_m in F_k. The warp function is formulated via the rigid body transformation (RBT) of ξ_{tk}, and the standard pinhole camera model. The RBT of a 3D point $P = (X, Y, Z) \in \mathbb{R}^3$ is formulated as:

$$G_{\xi_{tk}}(P) = \exp({\xi_{tk}}^\wedge)(P) = R_{tk} \cdot P + T_{tk} \tag{1}$$

As for the camera model, we take f_x, f_y as the focal length and c_x, c_y as the optical center. The projection functions π and π^{-1} perform transformation between the 3D point P and the pixel $p = (u, v)$ with its depth $d = D_t(p)$:

$$p = \pi(P) = (f_x X/Z + c_x, f_y Y/Z + c_y) \tag{2}$$

$$P = \pi^{-1}(p, d) = (d(u - c_x)/f_x, d(v - c_y)/f_y, d) \tag{3}$$

By combining (1), (2) and (3), we formulate the warp function that takes the input of (p_m, d) in F_t and outputs p'_m in F_k:

$$p'_m = \pi\left(R_{tk} \cdot \left(\pi^{-1}(p_m, d)\right) + T_{tk}\right) \tag{4}$$

Based on (4), we define the residual $r_m = DT_k(p'_m)$ as the pixel value of Euclidean distance field DT_k in F_k. DT_k is obtained by Distance Transform (DT) [13] which exhaustively calculates the distance to the nearest edge for every pixel. Inspired by [12], although DT is relatively costly, we opt to warp from the current frame to the KF which only requires the DT on KF instead of all frames.

To solve ξ_{tk}, r_m is iteratively minimized based on the loss function with the Huber weight function of threshold θ_h:

$$\hat{\xi}_{tk} = \arg\min_{\xi_{tk}} \left(\sum_m \delta_m r_m^2\right),\ \delta_m = \begin{cases} 1, & |r_m| \le \theta_h \\ \frac{\theta_h}{|r_m|}, & others \end{cases} \tag{5}$$

We use an iteratively re-weighted LM solver [18] to obtain an incremental update of $\Delta\xi$ to the ξ_{tk}. Denoting $J_m = \frac{\partial r_m}{\partial \xi}$ as the Jacobian matrix derived from each pixel warp, and H as the Hessian matrix, $\Delta\xi$ is obtained by solving the 6 DOF

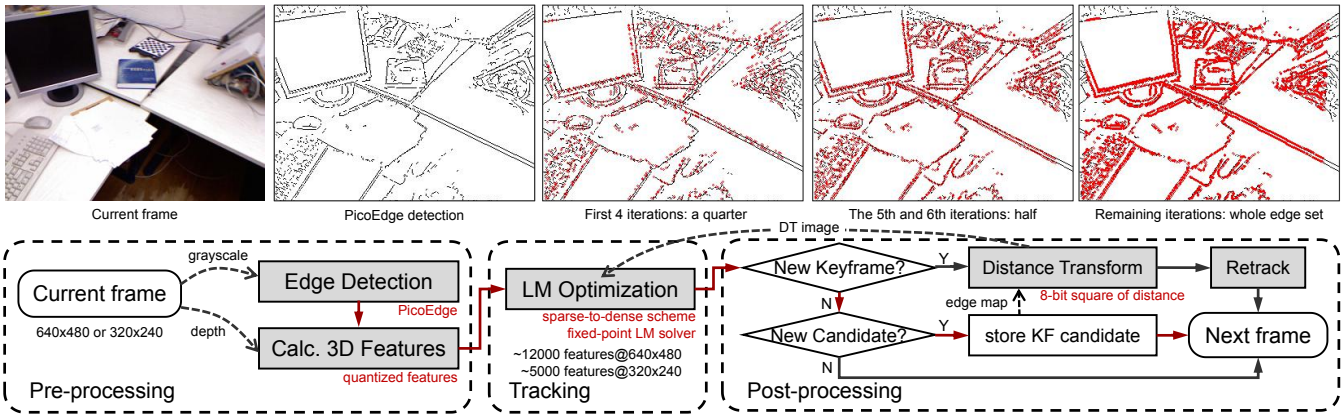

Fig. 1. The single-threaded framework of PicoVO. The upper part shows the warping results during the tracking procedure. Processing bottlenecks are emphasized in shaded blocks with dedicated optimization shown by red notes. Dashed arrows present the dataflow. Inspired by [12], instead of warping to the current frame, PicoVO inversely warps the current features to the KF, which bypasses the costly DT from the normal procedure shown by red arrows.

linear equations:

$$\Delta\xi = (H + \lambda I)^{-1} \sum_m J_m^T \delta_m r_m \ , \ H = \sum_m \delta_m J_m^T J_m \quad (6)$$

where λ is a damping factor that controls the stepping size. We refer the readers to [23] for the derivation of J_m. Hereby the LM solver is only related to one 6 DOF pose, and solving (6) is trivial. The bottleneck is caused by the calculation and summation of J_m and $J_m^T J_m$ matrices, which will be addressed in the following proposal.

IV. PicoVO Algorithmic Optimizations

To enable real-time VO on a low-end single-core MCU, we propose a single-threaded framework and attempt to mitigate the computation overhead without losing much accuracy. This framework is called PicoVO, which processes each frame in three steps: *pre-processing*, *tracking*, and *post-processing*, as shown in Fig. 1. First, the pre-processing step detects the edge pixels in the input frames and computes a set of 3D features. Second, these features are passed to the tracking step which estimates the current pose via the LM solver based on the EBVO observation model as previously discussed. Finally, the post-processing step assesses the tracking quality and maintains the KF and its candidate.

The processing bottlenecks of PicoVO, as emphasized in Fig. 1, requires dedicated optimization addressing both the algorithm and implementation. In this section, we focus on the algorithmic design in PicoVO, and will discuss the platform-oriented implementation in the next section.

A. Pre-processing: PicoEdge Detector

In the previous works, e.g. [16][14], Canny [24] is the de facto edge detector of EBVO. However, it can only achieve 8.8fps at QVGA (320x240) on a typical MCU like STM32F767 (tested on OpenMV [25]), which is far behind real-time processing. Unfortunately, the bottleneck is induced by the intrinsic computing requirements of 2D operations: image gradient calculation via Sobel convolution, followed by edge thinning via non-maxima suppression. Nevertheless, EBVO enjoys a good signal-to-noise ratio [11] thanks to the amount of data provided by the semi-dense distribution of edges. This suggests that we could use a less sophisticated edge detector without damaging the overall tracking quality. To rescue such poor performance, we downgrade the 2D operations to 1D to cut down a considerable amount of computation, while maintaining a satisfying detection result. We propose this lightweight edge detector specifically for our PicoVO, namely PicoEdge.

The basic idea is expressed in Alg.1, which computes a 1D image gradient, thresholds it by θ_1, and determines its maxima in a window of w pixels using a bitmask $hist$. Notice that generally all the varibles can be expressed in 8-bit, and this allows for the 32-bit SIMD acceleration [26] as long as the loop is unrolled four times. As such, we apply Alg.1 to each row and column (see Fig.2) as a preliminary edge detection, and the outputs are natually transformed into a bitmap for data compression and bitwise parallelism. Finally, to reduce some unpleasant salt-and-pepper noise, we apply pop-count [27] in each 3×3 regions to perform a bitwise convolution filter with a threshold θ_2. In practice, we generally fix $w = 3$ and $\theta_2 = 2$ for optimized implementation, while leaving θ_1 as a runtime parameter to control the edge density.

Also depicted in Fig. 2, while Canny has a strong focus on rich textures, PicoEdge has a better balance of low-contrast areas, which improves the edge distribution. In addition, PicoEdge achieves 152fps@QVGA on STM32F767 (see Fig. 5), boosting 17x throughput over Canny.

B. Tracking: Sparse-to-Dense Scheme

During the LM process, most edge-based methods like [12][16][17] track features with a coarse-to-fine scheme from bottom to top of an image pyramid, which reduces the amount of computation and also alleviates the impact of local optimum on low pyramid level. However, the preparation of pyramid structures in EBVO involves costly edge detection and DT at multiple resolutions, which remains inefficient for an MCU in terms of both computation and memory. Hereby, we approximate this with a sparse-to-dense tracking scheme which only operates on the top pyramid level and saves the processing. In addition, this also avoids costly and imprecise interpolation at low pyramid levels, because the integral coordinate at a high-definition image is equivalent to the sub-pixel coordinate at a lower resolution.

To approximate a 3-level pyramid with a factor of 2, edges are first partitioned evenly into four batches and processed

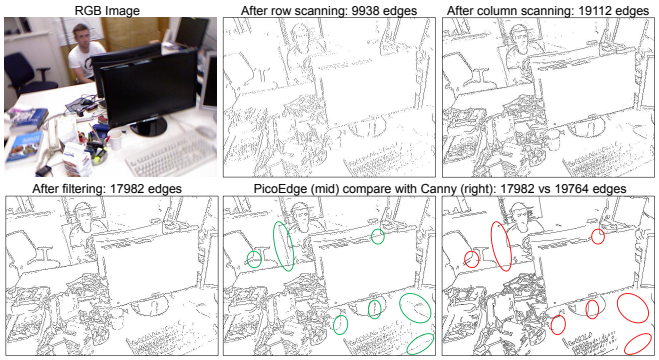

Fig. 2. PicoEdge detector ($\theta_1 = 7, \theta_2 = 2, w = 3$) compare to Canny (upper threshold 150, lower threshold 80, kernel size 3) on a 640x480 image. PicoEdge extracts around 10% less edge points while being able to detect more features in blurred areas, which reveals a better edge distribution.

Algorithm 1 detect-1D

Input: a 1D list I, length l, window size w, threshold θ_1
Output: a 1D edge pixel list E

$E = \{0\}$; $hist = 1 << (w-2)$; $lastgrad = 0$
for i from 2 to $l-1$ **do**
  $grad = \text{abs}(I[i-2] - I[i])$
  if $grad < lastgrad$ **then**
    if $hist == 0$ and $grad > \theta_1$ **then**
      set $E[i]$ as the edge
    $hist = (hist >> 1) + (1 << (w-2))$
  else
    $hist = (hist >> 1)$
  $lastgrad = grad$
return E

in the first four iterations respectively (see Fig. 1: *a quarter*). Then, the edges are divided into two batches and processed in the following two iterations respectively (see Fig. 1: *half*). The partitions can be done in-place by skipping 1 or 3 intervals when iterating over the edge set. Afterwards, the *whole edge set* is processed in the rest of the iterations. Typically, the LM solver converges within 10 iterations. In this case, the sparse-to-dense approach enables a 40% computation reduction compared to the naive processing over the entire edge set in all the iterations of LM solving.

C. Post-processing: Key-frame Management

One of the major challenges of VO is to insert a new KF when the incoming frame F_t can no longer match the latest KF. Prior works that mainly determine the KFs via runtime calculation or storing multiple KFs for culling later [16][2][12] remain too computation- and memory-intensive for the targeted low-end IoT devices. Instead, we adopt a threshold-based KF selection, which combines three critical metrics including: relative distance and angle, number of tracked edges, and average warp residual. These metrics can be calculated rapidly with small memory footprint.

In addition, we empirically choose KFs from frames that fulfill the metric $n_t > 0.7n'_t$, where n_t denotes the number of edges in F_t, and n'_t denotes the number of edges smoothed by a low-pass filter (7) to avoid abrupt motion blur. Note that α is utilized to control the smoothness and it is empirically set to 0.3. Last but not least, we will re-track the current frame on the update of new KF and restart the tracker on tracking failure to ensure the processing robustness.

$$n'_t = \alpha n_t + (1-\alpha)n'_{t-1} \tag{7}$$

V. PicoVO Implementation Optimizations

In this section, we propose quantization strategies to boost the performance of the LM solver with minor precision penalty. Quantized fixed-point data can be denoted as Qx.y, where x refers to the integer bits and y represents the fractional bits. Qx.y arithmetic can be efficiently conducted with standard fixed-point or SIMD DSP instructions [26].

A. Quantization of 3D Features

The first step of the LM solver is to warp 3D features to the KF with (4). Prior works like [12] adopt float-point pixel coordinates and use bilinear interpolation to sample sub-pixel gradient for the subsequent Jacobian calculation, which increases the amount of computation in tracking dramatically. However, we observe that interpolation has little influence on the tracker. Hereby, we opt to quantize the pixel coordinates and 3D features directly to alleviate the computation bottleneck. We unroll the warp function (4) which takes the input of (u, v, d) in the current frame and outputs the warp pixel coordinate (u', v') in KF:

$$\begin{aligned} u' &= f_x\frac{X'}{Z'} + c_x = f_x\frac{r_{00}x_0 + r_{01}y_0 + r_{02} + t_0/d}{r_{20}x_0 + r_{21}y_0 + r_{22} + t_2/d} + c_x \\ v' &= f_y\frac{Y'}{Z'} + c_y = f_y\frac{r_{10}x_0 + r_{11}y_0 + r_{12} + t_1/d}{r_{20}x_0 + r_{21}y_0 + r_{22} + t_2/d} + c_y \end{aligned} \tag{8}$$

where (X', Y', Z') is the 3D coordinate of the feature after rigid body transformation, $x_0 = \frac{u-c_x}{f_x}$, $y_0 = \frac{v-c_y}{f_y}$, and r_{ij} and t_i are the corresponding elements of the rotation matrix R_{tk} and translation vector T_{tk}. Since the output (u', v') is discrete, the precise float-point (X', Y', Z') input is not required. Based on the representation of (8), we use the inverse depth coordinate, i.e. $(x_0, y_0, \frac{1}{d})$, to replace the original 3D coordinate. Also note that the input d in the depth map is 16-bit, which implies that a 16-bit quantization is a modest and safe setup. As x_0 and y_0 roughly ranges within $(-0.5, 0.5)$, and $\frac{1}{d}$ ranges in $(0, 8)$ if $d > 0.125$, we have them quantized to be Q4.12 accordingly. According to our experiments, the quantized warping error is less than 1 pixel, which is sufficient to obtain pixel gradient without interpolation.

B. Quantization of the Jacobian Matrix

The second step of the LM solver is to obtain J_m and $J_m^T J_m$ matrices in (6). Specifically, we rearrange the calculation order of J_m and highlight the duplicate items for the sake of the computation reuse and reduction:

$$J_m = \begin{bmatrix} f_x I_u/Z' \\ f_y I_v/Z' \\ -(f_x I_u X' + f_y I_v Y')/Z'/Z' \\ -(f_x I_u X' + f_y I_v Y')/Z' \cdot Y'/Z' - f_y I_v \\ (f_x I_u X' + f_y I_v Y')/Z' \cdot X'/Z' + f_x I_u \\ f_y I_v \cdot X'/Z' - f_x I_u \cdot Y'/Z' \end{bmatrix} \tag{9}$$

where the image gradient of DT_k is denoted as I_u and I_v. Since X', Y', Z' are already obtained in (8), we focus on the calculation of I_u and I_v in the following.

Unlike prior works [12][15] that adopt float-point Euclidean DT in OpenCV, we use integers for the square of the distance as a lossless representation because DT operates on the discrete pixel coordinates. We also notice that the inter-

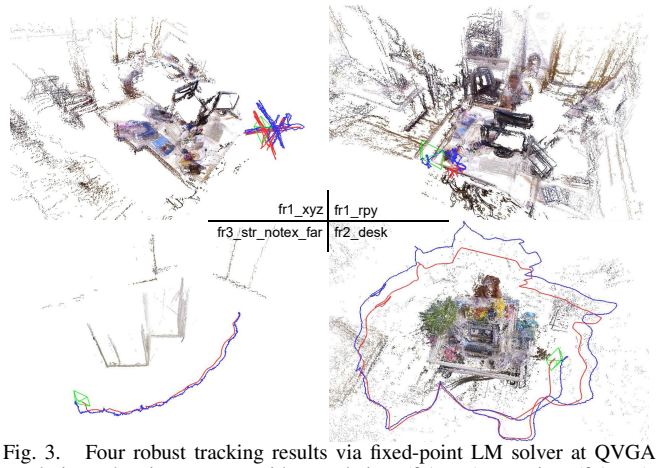

Fig. 3. Four robust tracking results via fixed-point LM solver at QVGA resolution, showing scenes with translation (fr1_xyz), rotation (fr1_rpy), low texture (fr3_notexture), and circular movement (fr2_desk). Estimated trajectories are colored in blue, and groundtruth in red.

frame motion is relatively small in EBVO, and we take 15 pixels as the outlier threshold so that `uint8_t` is sufficient to store this square-distance map. In turn, the image gradient can be obtained via a square root lookup table (LUT). We further provide two LUTs of $f_x I_u$ and $f_y I_v$ respectively along with the Huber weight function in (5), to efficiently obtain the terms $J_m^T \delta_m r_m$ and $\delta_m J_m^T J_m$ in (6). These LUTs utilize Q9.7 quantization format. In the summation of (6), we use Q14.2 for $J_m^T \delta_m r_m$, and Q29.3 for $\delta_m J_m^T J_m$ without computation overflow. The final step in each LM iteration is to solve the 6 DOF linear equations of (6) and update the pose via Lie algebra. Since these calculations are not the bottleneck of the solver, we in turn adopt 32-bit float-point representation to avoid numerical loss in the linear algebra and exponential map functions.

VI. Evaluation and Discussion

We evaluate on the TUM RGB-D benchmarking dataset [28] which contains a large variety of scenes and camera motions. As the feature-based VO baseline, we choose ORB-SLAM2 [2] and only enable its VO mode (by setting *mbOnlyTracking=True*). As the EBVO baseline, we choose REVO [12], which is the fastest open-source EBVO implementation to the best of our knowledge. We have them compared with the proposed PicoVO on both a desktop PC and an MCU platform. The PC comprises an Intel i7-6700@3.4GHz processor and 16GB RAM, while the MCU platform is NUCLEO-F767ZI [6] board which contains an STM32F767ZI@216MHz with 512KB RAM. Datasets are streamed from an SD card for the MCU. On the PC, we benchmark the systems with both VGA(640x480) and QVGA(320x240), while on the MCU, we can only benchmark with QVGA because of the limited memory.

A. Accuracy

We take the RMSE of Relative Pose Error according to [28] as the accuracy metric of the local positional and angular drift. The experiment results are shown in Tab. I. It can be observed that ORB-SLAM2 has poor performance in its VO mode, and it encounters failure in the case of low texture (fr3_notexture_*, fr3_cab) and motion blur (fr1_floor). In contrast, REVO and PicoVO basically complete all these

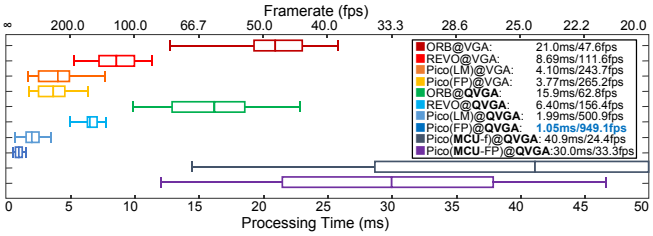

Fig. 4. Processing speed comparison on PC and MCU. All the PicoVO setups utilize PicoEdge. LM: basic LM; FP: fixed-point LM; MCU-f: 32-bit float-point LM on MCU; MCU-FP: FP on MCU. Among the PC evaluations, Pico(FP)@QVGA exhibits the peak framerate (949.1fps). The MCU varations both setup with PicoEdge, quantized 3D features and sparse-to-dense tracking, while MCU-FP achieves 33fps on average.

sequences thanks to their semi-dense feature points. In addition, we also evaluated the influence of the proposed approximation methods including PicoEdge (PE) and LM quantization (FP) on the tracking accuracy on the PC and MCU platforms. According to the experiments, it can be confirmed that PicoVO even shows slightly higher accuracy than REVO despite the proposed approximation optimizations that can greatly reduce the amount of computation and memory footprint. In addition, the evaluation also reveals that tracking with QVGA is close to that with VGA counterpart, as demonstrated in Fig. 3 which exhibits four satisfying tracking and mapping results using FP@QVGA.

B. Processing Speed

We only measure the processing time without considering the input/output overhead. First, we focus on VGA evaluations on PC as depicted in Fig. 4. Pico(LM) outperforms ORB-SLAM2 by 5.1x, because EBVO is intrinsically much more efficient than feature-based VO. It also outperforms REVO by 2.1x, because PicoVO does not require image pyramids or gradient interpolation. Pico(FP) approximates Pico(LM) via fixed-point quantization, thereby being even faster. Then, we focus on QVGA evaluations on PC. Compared with REVO@QVGA, Pico(LM) and Pico(FP) show 3.2x and 6x speedup, respectively. We achieve a higher speedup on QVGA thanks to the tiny memory footprint, which is mainly attributed to the quantization step as in Section V and fits the 128KB L1-D cache in the i7-6700 in the case of QVGA. It also reveals that the fixed-point processing achieves 1.9x speedup over the float-point counterpart.

As for the timing on MCU, we utilize the `arm-none-eabi-gcc 6.3.1` with `-Ofast` optimization. As shown in Fig. 4, the float-point Pico(MCU-f) achieves 24fps on average, while the fixed-point Pico(MCU-FP) achieves 33fps which is 36% faster. According to Fig. 5, we achieve around 1.5x speedup using fixed-point features and LM quantization compared to their float-point counterparts. This is because the target device, STM32F767 MCU, features an in-order dual-issue CPU [29] with two fixed-point ALUs but only one float-point FPU, which promises a higher instructions-per-clock (IPC) for the fixed-point processing.

Finally, we study how the proposed approximation strategies impact the convergence rate and the number of created KFs, which are critical to the amount of computing. As revealed in Tab. II, the influence of the sparse-to-dense (*s2d*) tracking scheme and quantized (*qnt*) 3D features on the convergence is trivial. The fixed-point (*int*) LM solver requires

TABLE I. RMSE COMPARISON OF RELATIVE POSE ERROR

Seq.	ORB V.[1] t.(m/s)	ORB V.[1] r.(°/s)	ORB QV.[2] t.(m/s)	ORB QV.[2] r.(°/s)	REVO V.[1] t.(m/s)	REVO V.[1] r.(°/s)	REVO QV.[2] t.(m/s)	REVO QV.[2] r.(°/s)	P.Canny V.[3] t.(m/s)	P.Canny V.[3] r.(°/s)	P.PE V.[4] t.(m/s)	P.PE V.[4] r.(°/s)	P.PE QV.[5] t.(m/s)	P.PE QV.[5] r.(°/s)	P.FP V.[6] t.(m/s)	P.FP V.[6] r.(°/s)	P.MCU QV.[7] t.(m/s)	P.MCU QV.[7] r.(°/s)
fr1_desk	0.053	2.51	0.046	2.51	**0.033**	**2.04**	0.044	2.40	0.052	4.14	0.097	5.42	0.085	3.69	0.062	3.08	0.069	4.70
fr1_desk2	0.064	**3.10**	**0.059**	3.32	0.063	3.93	0.068	4.08	0.088	5.44	0.186	11.5	0.077	4.14	0.116	7.35	0.138	7.11
fr1_360	**0.066**	**2.58**	0.073	2.81	0.122	4.54	0.126	4.65	0.097	5.68	0.131	8.24	0.141	4.89	0.182	4.94	0.145	5.16
fr1_floor	0.071[9]	2.90[9]	0.197[9]	5.34[9]	0.102	3.76	0.137	3.83	0.086	4.31	**0.068**	2.62	0.073	**2.59**	0.074	3.15	0.071	3.27
fr1_plant	0.041	1.75	0.050	2.20	0.032	1.41	0.028	1.33	0.044	1.82	**0.026**	1.49	0.029	1.52	0.046	1.27	0.028	**1.21**
fr1_room	0.067	**2.53**	0.073	2.72	0.057	2.62	0.055	2.65	**0.054**	2.89	0.058	4.65	0.076	3.88	0.088	5.94	0.080	2.99
fr1_rpy	**0.032**	**2.24**	0.036	2.47	0.037	2.77	0.037	2.89	0.054	5.02	0.067	6.12	0.047	3.93	0.069	5.02	0.052	2.91
fr1_xyz	0.027	1.36	**0.019**	**1.09**	0.024	1.39	0.024	1.46	0.029	1.56	0.025	1.27	0.023	1.35	0.026	1.45	0.030	1.82
fr2_coke	0.148	5.79	0.158	5.10	0.172	5.17	0.179	4.65	0.113	3.15	**0.043**	**1.16**	0.068	1.69	0.048	1.39	0.069	2.17
fr2_dishes	0.031	1.42	0.029	1.25	**0.015**	0.66	**0.015**	**0.64**	0.022	1.09	0.021	0.95	0.026	1.26	0.029	1.07	0.035	1.23
fr2_rpy	0.006	0.36	0.013	0.61	0.005	0.38	0.006	0.41	**0.004**	0.39	**0.004**	0.39	0.006	0.46	0.010	0.41	0.008	0.50
fr2_desk	0.027	1.13	0.032	1.36	0.019	0.64	0.021	0.72	**0.015**	0.54	0.016	**0.51**	0.016	0.62	0.027	0.53	0.020	0.69
fr2_xyz	0.006	0.34	0.009	0.41	**0.005**	0.37	0.007	0.42	**0.005**	**0.33**	**0.005**	0.34	0.007	0.39	0.011	0.34	0.009	0.47
fr3_cabinet	0.073[8]	3.04[8]	-[9]	-[9]	0.094	3.74	0.097	4.11	0.064	2.75	0.063	2.56	0.083	3.35	**0.056**	**2.35**	0.102	3.81
fr3_large_cabinet	0.314	6.81	0.127	1.73	0.259	5.58	0.394	7.63	0.075	0.97	**0.062**	**0.88**	0.085	1.11	0.110	1.04	0.133	1.45
fr3_long_office	0.021	0.92	0.022	0.93	0.012	0.52	0.016	0.71	**0.011**	0.49	**0.011**	**0.48**	0.012	0.58	0.025	0.55	0.018	0.73
fr3_nst_tex_near	0.017	0.89	0.022	1.03	0.019	0.89	0.017	0.88	0.016	0.87	**0.014**	**0.80**	0.023	1.04	0.025	0.93	0.041	1.64
fr3_nst_tex_far	0.124	2.41	0.086	2.29	0.116	2.20	0.228	6.44	**0.040**	**0.90**	0.050	1.03	0.045	1.01	0.044	0.85	0.173	3.31
fr3_st_tex_near	0.019	0.99	0.015	0.80	0.014	0.78	0.015	0.79	**0.011**	**0.62**	0.013	0.78	0.013	0.73	0.015	0.65	0.015	0.81
fr3_st_tex_far	0.017	0.60	**0.014**	0.52	0.018	0.55	0.019	0.61	**0.014**	0.50	**0.014**	**0.49**	0.015	0.56	0.024	0.53	0.019	0.66
fr3_st_ntex_near	0.065[8]	3.34[8]	-[9]	-[9]	0.554	32.5	-[9]	-[9]	0.175	5.69	0.034	1.61	0.052	2.28	**0.023**	**1.10**	0.039	2.13
fr3_st_ntex_far	0.018[8]	0.62[8]	-[9]	-[9]	0.042	1.05	0.060	1.30	0.032	0.79	**0.015**	**0.52**	0.023	0.69	0.025	0.63	0.028	0.77
Average[10]	0.046	1.907	0.051	1.926	0.055	2.148	0.056	2.217	0.045	2.103	**0.039**	1.853	0.044	1.770	0.045	**1.697**	0.055	2.016

[1] VGA baseline on PC [2] QVGA baseline on PC [3] VGA using Canny on PC [4] VGA using PicoEdge on PC [5] QVGA using PicoEdge on PC
[6] VGA fixed-point LM on PC [7] QVGA fixed-point LM on MCU [8] Initialization failed [9] Tracking lost/failed [10] Trimmed 2 max and 2 min

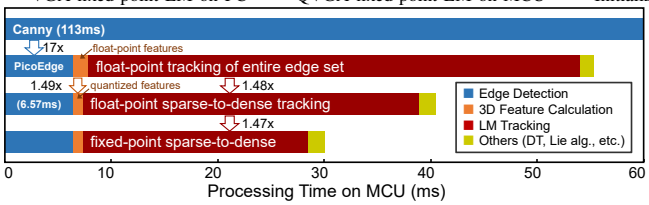

Fig. 5. Average timing results on MCU (PicoVO@QVGA). The Canny implementation [25] is too slow to process in real-time, while the proposed PicoEdge enjoys a 17x speedup over Canny. The sparse-to-dense scheme is 1.48x faster than the naive tracking using every edge in all iterations. The calculation of quantized inverse depth coordinates is 1.49x faster than that of the original float-point 3D coordinates. Finally, the fixed-point LM further accelerates the original float-point tracking process by 1.47x.

an additional LM iteration due to the numerical loss in the quantized Jacobian and Hessian matrices. Nevertheless, this does not override the benefits brought by the fixed-point computing. All the approximation strategies can induce a higher KF occurence (*pct*), which slightly increases the post-processing time as shown in Fig. 5.

C. Memory Footprint and Power Consumption

We measure the memory footprint at the third frame to avoid the overhead of the global map, and results are shown in Tab. III. It can be seen that both the PC baselines, ORB-SLAM2 and REVO, have massive memory consumption beyond the capacity of the MCU. Nevertheless, thanks to our bare-metal implementation, PicoVO packs everything into the 512KB on-chip RAM, which reduces the memory footprint by nearly two orders of magnitude compared to the PC baselines. The PicoVO core on MCU only consumes 355KB RAM, including:

- **Input**: 8-bit grayscale and 16-bit depth (230KB);
- **Current**: an edge map (10KB) and 5k features (30KB);
- **Key-frame candidate**: an edge map (10KB);
- **Key-frame**: distance field (75KB)

Also shown in Tab. III, PicoVO on MCU reduces two orders of magnitude power consumption than the PC baselines. We further evaluate the energy efficiency with a more comprehensive metric i.e. Joule per frame. The experiment reveals that PicoVO on MCU reduces 63x energy over ORB-SLAM2 on PC, and it also reduces 31x energy than REVO.

VII. CONCLUSIONS

In this work, we propose PicoVO, a high-quality and super lightweight VO solution for low-end IoT devices like MCU. PicoVO is a single-threaded EBVO for RGB-D sensors with multi-layer optimization from both algorithm and hardware mapping perspectives. First, we propose the EBVO-oriented and lightweight PicoEdge to replace the costly Canny detector. Second, we design a sparse-to-dense tracking scheme to replace the costly image pyramids. Third, we exploit quantized data structures and fixed-point LM solver for further acceleration. With these approaches, PicoVO successfully fits the computing resource of an MCU and saves a great amount of memory and power resource compared to the PC solutions. It can process 33fps@QVGA on NUCLEO-F767ZI with high tracking precision, which can compete with many state-of-art PC solutions. In this sense, we believe PicoVO will help propel the development of applications involving 3D environment perception on the MCU-based IoT systems.

PicoVO is still a preliminary work. In the future we will seek for optimizations on VIO and full system of SLAM, as well as the design of ultra-low power hardware accelerators.

TABLE II. NUMBER OF ITERATIONS IN LM

	float/xyz/full	float/xyz/s2d	float/qnt/s2d	int/qnt/s2d
Iteration	7.263	7.368	7.368	8.105
KF pct.	6.3%	7.5%	7.6%	7.7%

TABLE III. SYSTEM OVERHEAD COMPARISON

	PC		MCU
	ORB-SLAM2	REVO	PicoVO
Memory	31.4MB	41.0MB	**0.35MB**
Power	60W	69.4W	**0.31W**
Energy per Frame	2.812J/frame	1.291J/frame	**0.041J/frame**

ACKNOWLEDGMENT

This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC05030201. The corresponding authors are Ying Wang and Cheng Liu.

REFERENCES

[1] "Microcontroller market size to reach $42.19 billion by 2027: at 11.5% cagr." [Online]. Available: https://www.globenewswire.com/news-release/2020/07/20/2064465/0/en/Microcontroller-Market-Size-to-Reach-42-19-Billion-by-2027-at-11-5-CAGR.html

[2] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[3] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[4] J. Jose Tarrio and S. Pedre, "Realtime edge-based visual odometry for a monocular camera," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 702–710.

[5] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2502–2509.

[6] STMicroelectronics, "Nucleo-f767zi," 2020. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f767zi.html

[7] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[8] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[9] X. Wu and C. Pradalier, "Illumination robust monocular direct visual odometry for outdoor environment mapping," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2392–2398.

[10] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[11] L. Kneip, Z. Yi, and H. Li, "Sdicp: Semi-dense tracking based on iterative closest points." in *Bmvc*, 2015, pp. 100–1.

[12] F. Schenk and F. Fraundorfer, "Robust edge-based visual odometry using machine-learned edges," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1297–1304.

[13] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory of computing*, vol. 8, no. 1, pp. 415–428, 2012.

[14] Y. Zhou, H. Li, and L. Kneip, "Canny-vo: Visual odometry with rgb-d cameras based on geometric 3-d–2-d edge alignment," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 184–199, 2018.

[15] X. Wang, W. Dong, M. Zhou, R. Li, and H. Zha, "Edge enhanced direct visual odometry." in *BMVC*, 2016.

[16] F. Schenk and F. Fraundorfer, "Reslam: A real-time robust edge-based slam system," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 154–160.

[17] M. Kuse and S. Shen, "Robust camera motion estimation using direct edge alignment and sub-gradient method," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 573–579.

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[19] R. Liu, J. Yang, Y. Chen, and W. Zhao, "eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[20] 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, 2019.

[21] D. K. Mandal, S. Jandhyala, O. J. Omer, G. S. Kalsi, B. George, G. Neela, S. K. Rethinagiri, S. Subramoney, L. Hacking, J. Radford *et al.*, "Visual inertial odometry at the edge: A hardware-software co-design approach for ultra-low latency and power," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 960–963.

[22] E. Terry, "Silicon at the heart of hololens 2," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 2019, pp. 1–26.

[23] C. Kerl, "Odometry from rgb-d cameras for autonomous quadrocopters," *Master's Thesis, Technical University*, 2012.

[24] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[25] I. Abdelkader, Y. El-Sonbaty, and M. El-Habrouk, "Openmv: A python powered, extensible machine vision camera," *arXiv preprint arXiv:1711.10464*, 2017.

[26] *ARM v7-M Architecture Reference Manual*, Arm Limited, 2018.

[27] S. E. Anderson, "Counting bits set, in parallel." [Online]. Available: https://graphics.stanford.edu/˜seander/bithacks.html

[28] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573–580.

[29] *ARM Cortex-M7 Processor Technical Reference Manual*, Arm Limited, 2014.