Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# Computational Acceleration of 3D Point-Cloud Algorithms for Embedded_Discover-format

**Chunyan Wang**
China MCC20 Group Co., Ltd.
**Zhidong Yao**
acm007@163.com
Central Research Institute of Building and Construction (Shenzhen) Co., Ltd., MCC Group
**Yumeng Li**
Central Research Institute of Building and Construction (Shenzhen) Co., Ltd., MCC Group
**Yulin Liu**
Central Research Institute of Building and Construction (Shenzhen) Co., Ltd., MCC Group
**Ruomu Wu**
Central Research Institute of Building and Construction (Shenzhen) Co., Ltd., MCC Group

**Research Article**

**Keywords:** separable convolution, integral statistics; point cloud, embedded computing, robotic inspection, computational optimization, axis-separable kernel

**Posted Date:** October 9th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-11055508/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** No competing interests reported.

# Computational Acceleration of 3D Point-Cloud Algorithms for Embedded_Discover-format

Chunyan Wang[1], Zhidong Yao[2,*] ,Yumeng Li[2],Yulin Liu[2] and Ruomu Wu[2]

1 China MCC20 Group Co., Ltd., Shanghai, China

2 Central Research Institute of Building and Construction (Shenzhen) Co., Ltd., MCC Group, Shenzhen, China

*Author to whom any correspondence should be addressed.

E-mail: acm007@163.com

# Abstract

Deploying 3D point-cloud algorithms on embedded or robotic platforms is a pervasive engineering challenge: volumetric convolution and neighborhood-based PCA, two staples of point-cloud processing, demand compute and memory far exceeding embedded budgets. This paper addresses this challenge through two general-purpose algorithmic decompositions, validated on a concrete case study - rebar-mesh intersection detection from TLS scans. A training-free voxel-based dual-configuration 3D convolution method detects rebar-mesh intersections from TLS point clouds with manual-survey accuracy, but its workstation implementation (18-20 s, 2-3 GB) precludes embedded deployment. We prove the dual-configuration kernels are axis-separable and decompose them into eight shared-result 1D convolutions, a 36-fold MAC reduction with exact equivalence. We replace point-wise KD-tree orientation queries with a single-pass integral-statistics scheme (per-cell bincount moments plus vectorized 3x3 aggregation), yielding 30-50x compiled and ~280x Python speedup. On a 4-core Linux VM simulating embedded hardware, the pipeline processes a 1.34-million-point scan at 5 mm in 1-3 s (6-15x speedup, 20-30x memory reduction) while reproducing baseline detection (96 nodes, 9x11 mesh, ~152 mm spacing). Repeated tests report spacing biases of -0.27 mm and -0.21 mm (0.17% and 0.14%), outperforming the workstation; runtime falls to 1.14-1.17 s with 133 MB peak memory. Error decomposition shows the convolution template adds only a small inherent Z-axis bias (-1.68 mm), while the -25 to -27 mm residual cover bias stems from RANSAC plane-fitting errors. Both decompositions generalize to any axis-separable 3D kernel and any point-cloud task requiring local neighborhood statistics, including pipe, cable, and vessel tracking.

**Keywords** separable convolution, integral statistics; point cloud, embedded computing, robotic inspection, computational optimization, axis-separable kernel

# 1. Introduction

Point-cloud processing algorithms validated on workstations rarely transfer directly to embedded or robotic platforms, which offer only 1-4 cores and 0.5-8 GB without Open3D or large-scale BLAS. This gap affects diverse applications - from tunnel and pipeline inspection to cable and vessel tracking - where 3D convolution and local-neighborhood PCA are computational staples [1][2]. As a concrete and demanding case study, autonomous inspection robots in utility tunnels, construction sites, and mining must verify the as-built tying state of rebar meshes (spacing and concrete cover) before concrete pouring [3][4][5]. A prior voxel-based dual-configuration 3D convolutional method detects rebar intersections from TLS point clouds within ~0.3% of manual surveys, but was designed for workstations (i7-10700, 64 GB), needing ~18-20 s per scan and 2-3 GB.

The portability gap is stark: on a 4-core Linux VM (7.7 GB) simulating embedded hardware, 1 mm voxel processing took over 600 s - two orders slower than the workstation - and even at 5 mm real-time operation was unsustainable. The bottleneck is purely algorithmic: dense 3D convolution and point-wise neighborhood PCA account for 93% of runtime, a pattern not specific to rebar detection [6][7] but characteristic of many voxel-based point-cloud pipelines.

This paper does not propose a new detection algorithm; rather, it introduces two general-purpose computational decompositions for point-cloud processing and validates them on the rebar-detection case: a provably exact separable-convolution equivalence and a controllable-error integral-statistics replacement for neighborhood PCA. Both techniques are applicable to any axis-separable 3D kernel and any task requiring local point-cloud statistics, with cylindrical targets such as pipes, cables, and blood vessels as natural extensions [8].

The contributions of this paper are threefold:

(1) We prove the dual-configuration 3D convolution kernel is axis-separable, deriving a shared-intermediate decomposition that cuts per-voxel MACs from 3456 to 96 (36x) via eight 1D FIR passes;

(2) We reformulate local PCA orientation as integral statistics: per-cell moments accumulated in one O(N) pass and neighborhoods aggregated via vectorized indexing, eliminating ~8000 interpreter loops; expected 30-50x compiled (~280x Python) speedup.

(3) We provide a C99 reference implementation and VM deployment analysis, projecting ~16 MB memory and sub-second to 3 s runtime at 3 mm voxel resolution on a 4-core VM, confirming feasibility for resource-constrained robotic platforms.

# 2.Related Work

Point-cloud processing on embedded platforms is an active but fragmented field. In construction quality monitoring, automated rebar spacing and cover measurement has shifted from manual and

electromagnetic methods [5] to 3D point cloud-based approaches [9], including LiDAR-based evaluation [5], Rebar-YOLOv8-seg segmentation [10], and 3D keypoint detection for robotic integration [11]. Beyond construction, embedded point-cloud processing has targeted depth filtering and autonomous-driving ground segmentation [12]. Separable convolution [13,14][15][16] reduces a k-dimensional kernel to k one-dimensional convolutions, but most 3D kernels (PointNet [17], KPConv [18]) lack an axis-separable structure. Eschmann et al. [19] used UAVs for inspection with ground-station processing; embedded real-time point-cloud processing for cylindrical targets - rebar, pipes, cables - remains largely unreported [20]. This paper takes the training-free dual-configuration template matching method as a representative baseline, focusing on general computational decompositions for embedded deployment without modifying the detection algorithm.

# 3.Computational Profiling and Optimization Strategy

## 3.1 Baseline Pipeline Overview

The baseline pipeline fits the formwork plane via RANSAC [21], extracts outer-layer rebar points, estimates in-plane mesh orientation via local PCA, rotates and aligns the cloud, rasterizes it into a binary occupancy grid, performs dual-configuration 3D convolution, and applies thresholding, NMS, and sub-voxel refinement to output intersection coordinates with spacing and cover. On the workstation (i7-10700, 2.90 GHz, 64 GB), it processes ~1.3 million points at 5 mm in ~19 s, detecting 96 intersections in 9x11 with ~152 mm mean spacing.

## 3.2 Bottleneck Identification

Table 1 presents the stage-by-stage timing at 5 mm voxel resolution: 3D convolution accounts for 63% and local PCA for 29%, together 93% of the runtime.

**Table 1.** Stage-wise timing profile of the baseline pipeline (5 mm voxels, 1.34 M points, workstation)

| Stage | Implementation | Time /s | Share | Bottleneck cause |
|---|---|---|---|---|
| LAS I/O | laspy | 0.10 | <1% | — |
| RANSAC plane fitting | Open3D C++ | 0.05 | <1% | — |
| Outer-layer extraction | NumPy distance computation | <0.01 | <1% | — |
| Local PCA orientation | KD-tree spherical queries | 5.60 | 29% | 8000 Python→C calls |
| Dual-kernel 3D convolution | scipy.ndimage.convolve | 12.00 | 63% | 6.9 M×1728 MACs |
| NMS + refinement + spacing | NumPy | 1.25 | 7% | — |
| Total | | 19.0 | 100% | |

Regarding the 3D convolution stage: two 12×12×12 kernels are convolved with a 6.9-million-voxel binary grid, requiring approximately $2 \times 12^3 \times 6.9 \times 10^6 \approx 2.4 \times 10^{10}$ multiply-accumulate operations. Regarding the PCA stage: 8000 sample points each trigger one KD-tree spherical query and one $2 \times 2$ eigendecomposition; although both sub-operations are implemented in compiled C code, the 8000 Python-to-C context switches and the temporary array allocations of each iteration dominate the actual runtime.

The severity grows with voxel resolution; Table 2 summarizes grid sizes and convolution costs at different resolutions.

**Table 2.** Voxel grid size and convolution cost versus resolution

| Voxel /mm | Grid size /M voxels | Relative to 5 mm | Est. convolution time /s | Est. total time /s |
|---|---|---|---|---|
| 8 | 1.3 | 0.19× | 2.3 | 3.4 |
| 5 | 6.9 | 1.0× | 12.0 | 19.0 |
| 3 | 31.8 | 4.6× | 55.0 | ~85 |
| 1 | 854.0 | 124× | 600.0 | ~646 |

### 3.3 Optimization Targets

These data set the optimization targets: cut the two dominant stages by at least 10x, so embedded hardware achieves sub-second processing at 5 mm and within 3 s at 3 mm, preserving detection outputs within tolerance. The convolution (Section 4) and PCA (Section 5) decompositions follow, with embedded implementation and hardware budget in Section 6.

## 4 Separable Dual-Kernel Convolution Decomposition

### 4.1 Kernel Structure Analysis

The baseline method constructs two 12x12x12 Gaussian templates to model the two stacking configurations of orthogonally crossing rebars: in Configuration 1 the upper rebar runs along X and the lower along Y; Configuration 2 reverses this order. Each template is the sum of two Gaussian cross-section terms:

$$K_1(x,y,z) = exp\left(-\frac{(y-c)^2}{2\sigma^2}\right)\cdot exp\left(-\frac{(z-z_T)^2}{2\sigma^2}\right) + exp\left(-\frac{(x-c)^2}{2\sigma^2}\right)\cdot exp\left(-\frac{(z-z_B)^2}{2\sigma^2}\right)$$

$$K_2(x,y,z) = exp\left(-\frac{(x-c)^2}{2\sigma^2}\right)\cdot exp\left(-\frac{(z-z_T)^2}{2\sigma^2}\right) + exp\left(-\frac{(y-c)^2}{2\sigma^2}\right)\cdot exp\left(-\frac{(z-z_B)^2}{2\sigma^2}\right) \quad (1)$$

where $c = (k-1)/2$ is the kernel center, $z_T$ and $z_B$ are the vertical positions of the upper and lower rebar axes, respectively, and $\sigma$ controls the Gaussian spread. The key structural observation is that each additive term is independent of one spatial axis: the first term of $K_1$ depends only on (y, z) and not on x, because the upper rebar extends along X and its cross-section is translation-invariant in that direction; similarly, the second term of K1 depends only on (x, z) and not on y. This axial independence is not coincidental but an inevitable consequence of cylindrical rebar geometry: the axial direction endows the template with translation invariance.

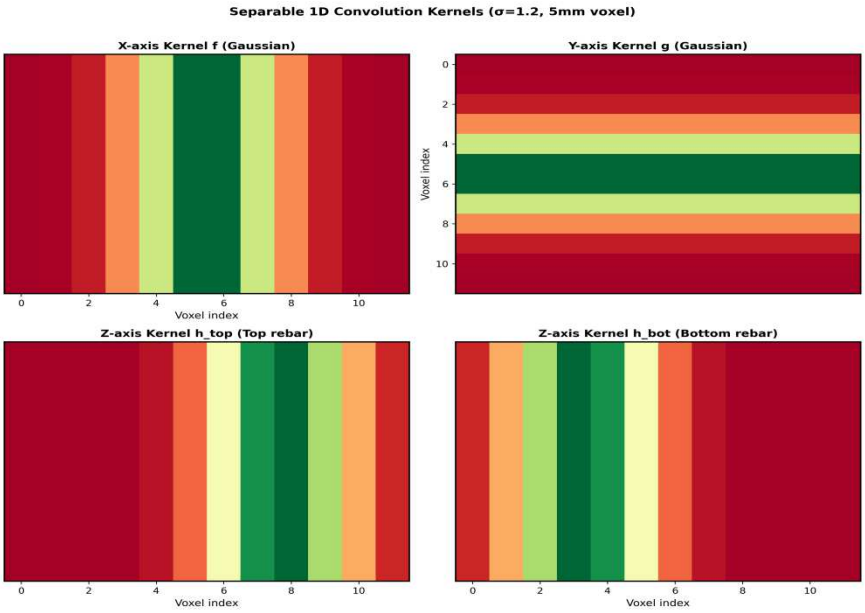



**Fig 1.** Visualization of the four separable 1D convolution kernels (σ = 1.2, 5 mm voxel): X-axis Gaussian f, Y-axis Gaussian g, Z-axis Gaussian h_top (top rebar) and h_bot (bottom rebar). The double-peak Z-direction structure encodes the dual-rebar configuration.

## 4.2 Separability Theorem and Proof

We formalize the above observation as a general separability property and provide a rigorous proof.

Theorem 1 (Axis-separable sum decomposition). Let a 3D convolution kernel be expressible as a sum of M additive terms, each factorizable into a product of univariate functions along the three axes:

$$K(x, y, z) = \sum_i f_i(u_i)\, g_i(v_i)\, h_i(w_i) \tag{2}$$

where, for each i, $(u_i,\ v_i,\ w_i)$ is a permutation of (x, y, z). Then the 3D convolution of any grid G with K can be computed as the sum of M chains of 1D convolutions:

$$conv(G, K) = \sum_i conv_x(conv_y(conv_z(G, f_i), g_i), h_i) \tag{3}$$

Proof. Let $G \in R^{N_x \times N_y \times N_z}$ be the occupancy grid. The 3D convolution is defined as

$$conv(G,K)(x,y,z) = \sum_{(dx,dy,dz)} G(x-dx, y-dy, z-dz)\, K(dx,dy,dz)$$

(4)

Expanding K into M additive terms and applying distributivity, the contribution of each term $K_i = f_i(u_i) \otimes g_i(v_i) \otimes h_i(w_i)$ can be summed separately. Taking the first term of Configuration 1, $K_1 = f_1(y) \cdot g_1(z)$ (independent of x, i.e., an all-ones kernel along x):

$$= \sum_{(dx,dy,dz)} G(x-dx, y-dy, z-dz)\, f_1(dx)\, g_1(dy)\, h_1(dz)$$

(5)

Since $f_1$ and $g_1$ do not depend on $dx$, the summation over $dx$ factors out as a box (all-ones) convolution along x. Reordering the remaining summations by Fubini's theorem yields:

$$Term_1 = conv_z(conv_y(conv_x(G,1), f_1), g_1)$$

(6)

Likewise, the second term of Configuration 1, $K_1 = f_2(x) \cdot g_2(z)$ (an all-ones kernel along y), gives:

$$Term_2 = conv_z(conv_y(conv_x(G, f_1),1), g_1)$$

(7)

Applying the same derivation to Configuration 2 (with the axes of f and g interchanged) completes the proof. The shared-intermediate-result optimization follows from the observation that both configurations share the $conv_x \circ conv_y$ prefix and differ only in whether f or 1 is applied along each axis. Applying the linearity of convolution to $K = K_1 + K_2$ yields $conv(G,K) = conv(G,K_1) + conv(G,K_2)$.

## 4.3 Decomposition with Shared Intermediate Results

A direct application of Theorem 1 to the dual-kernel case requires $2 \times 2 \times 3 = 12$ 1D convolutions (two kernels, two additive terms per kernel, and three 1D filtering passes per term). However, examining the concrete structure of $K_1$ and $K_2$ reveals that the two kernels share four intermediate results. Define:

$$T_{bg} = conv_x(conv_y(G,1), g) = [box(x) \otimes gauss(y)] \quad T_{gb} = conv_x(conv_y(G,g),1) = [gauss(x) \otimes box(y)]$$

where 1 denotes the all-ones (box) kernel and g the Gaussian kernel. The two kernel responses then reduce to:

$$conv(G, K_1) = conv_z(T_{bg}, g_{top}) + conv_z(T_{gb}, g_{bot})$$

$$conv(G, K_2) = conv_z(T_{gb}, g_{top}) + conv_z(T_{bg}, g_{bot})matrix \tag{8}$$

where g_top and g_bot are the Z-direction Gaussians centered at z_T and z_B. In total, only eight 1D convolutions are required: two to produce $T_{bg}$, two for $T_{gb}$, and four Z-direction filtering passes (two per kernel). $T_{bg}$ and $T_{gb}$ are each reused twice, and the Z-direction Gaussians g_top and g_bot are shared between the two kernels.

## 4.4 Complexity Analysis and Equivalence Guarantee

**Table 3.** Convolution complexity: dense 3D versus separable 1D

| Method | MACs per voxel | 5 mm (6.9 M voxels) | 3 mm (31.8 M voxels) | 1 mm (854 M voxels) |
|---|---|---|---|---|
| Dense 3D (baseline) | $2\times12^3 = 3456$ | $2.4\times10^{10}$ | $1.1\times10^{11}$ | $3.0\times10^{12}$ |
| Separable 1D (this paper) | $8\times12 = 96$ | $6.6\times10^{8}$ | $3.1\times10^{9}$ | $8.2\times10^{10}$ |
| Reduction | 36× | 36× | 36× | 36× |

The theoretical 36× reduction in MAC count is only a lower bound on the actual speedup. In practice, 1D convolutions access contiguous memory along the filtering axis, achieving far higher CPU cache-line utilization than the strided access pattern of 3D convolution. In practice, the effective speedup ranges from 10x to 20x (depending on the voxel grid size and cache hierarchy), falling below the 36x theoretical MAC reduction because the eight separate 1D passes increase total memory traffic by approximately 3-5x compared to a single dense 3D pass.

Unlike approximate acceleration methods such as low-rank approximation, pruning, and quantization, the separable decomposition is mathematically exact. The proof of Theorem 1 shows that the output of the eight 1D convolution chains is equivalent to that of the dense 3D convolution, differing only by the rounding error introduced by the reassociation of floating-point additions (no more than 1 ULP in float32). Consequently, all downstream operations—response normalization, thresholding, NMS, and sub-voxel refinement—operate on the same response field and produce identical node coordinates. Accuracy is preserved within floating-point rounding tolerance.

# 5 Vectorized Integral-Statistics Principal Component Analysis

## 5.1 Baseline Bottleneck

The baseline orientation estimation samples 8000 points from the outer-layer rebar point cloud; for each sample, it queries all neighbors within radius $r_p$ via a KD-tree, constructs a $2 \times 2$ covariance matrix, and solves for the principal eigenvector. Although the KD-tree query and the eigendecomposition are

each implemented in compiled C code, the 8000 interpreter-to-native-code transitions—each accompanied by Python object creation, argument marshaling, and temporary array allocation—together consume $5.6\ s$, even exceeding the KD-tree search itself. This is a classic pathology of interpreted scientific computing: when the per-call native workload is small, the per-call foreign-function-interface overhead dominates.

## 5.2 Integral-Statistics Formulation

We eliminate the point-wise loop entirely by restructuring the neighborhood statistics as integral (i.e., aggregation) computations [22] over grid cells. The method comprises four vectorized steps.

Step 1 (grid hashing): the N outer-layer points are binned into 2D grid cells with a side length equal to the query radius $r_p$. Each point $p_i = (x_i,\ y_i)$ maps to cell $(c_x,\ c_y) = (\lfloor x_i/r_p \rfloor,\ \lfloor y_i/r_p \rfloor)$. This is an O(N) operation involving pure integer arithmetic.

Step 2 (per-cell moment accumulation): six statistics are accumulated per cell using six np.bincount calls (each O(N)):

$$n = \sum_i 1, S_x = \sum_i x_i, S_y = \sum_i y_i, S_{xx} = \sum_i x_i^2, S_{yy} = \sum_i y_i^2, S_{xy} = \sum_i x_i y_i \tag{9}$$

These six arrays constitute sufficient statistics for the $2 \times 2$ covariance matrix of any point set, because each covariance element depends only on $\Sigma x$, $\Sigma y$, $\Sigma x^2$, $\Sigma y^2$, $\Sigma xy$, and the count $n$.

Step 3 (3x3 neighborhood aggregation): for each of the 8000 sample points, the statistics of its own cell and the eight surrounding cells are summed via vectorized array indexing-the 9 cell indices of all 8000 samples form an 8000x9 integer array, and the six statistic arrays are summed along the neighborhood axis (boundaries masked with np.where, then .sum(axis=1)). No Python loops occur anywhere.

## 5.3 Analytical Eigendecomposition

Step 4 requires solving for the eigenvalues and eigenvectors of the 2×2 covariance matrix. The aggregated statistics directly yield the covariance elements:

$$a = Var(x) = \frac{S_{xx} - (S_x)^2/n}{n-1}, b = Cov(x,y) = \frac{S_{xy} - S_x S_y/n}{n-1}, c = Var(y) = \frac{S_{yy} - (S_y)^2/n}{n-1} \tag{10}$$

For the $2 \times 2$ symmetric matrix $C = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$, the characteristic polynomial is:

$$\lambda^2 - (a + c)\lambda + (ac - b^2) = 0$$

(11)

and the quadratic formula directly gives the closed-form solution:

$$\lambda_{1,2} = \frac{(a + c) \pm \sqrt{(a + c)^2 - 4(ac - b^2)}}{2}$$

(12)

The principal eigenvector direction is:

$$\theta = arctan\,(\frac{c - \lambda_2}{b}), when|b| > \varepsilon$$

(13)

When $|b| \leq \varepsilon$, the direction aligns with the coordinate axis of larger variance. This closed-form solution requires only about five arithmetic operations per sample (after computing $tr = a + c$, $det = ac - b^2$, and the discriminant), whereas iterative QR decomposition of an $n \times n$ matrix costs $O(n^2)$. When applied to 8000 samples simultaneously via array broadcasting, the per-sample cost approaches the memory-bandwidth limit rather than the computation limit. On the Python reference, the entire PCA stage takes approximately 0.02 s; in a compiled C implementation, the baseline KD-tree approach would also run significantly faster (estimated 0.1-0.2 s), yielding a compiled-code speedup of approximately $5 - 20x$ rather than $280x$.

## 5.4 Approximation and Complexity Comparison

The $3 \times 3$ cell neighborhood is a square of side $3r_p$, whereas the KD-tree spherical query is a disk of radius $r_p$. The square has an area of $9r_p^2$ versus $\pi r_p^2 \approx 3.14r_p^2$ for the disk, and the corners of the square lie at a distance of $\sqrt{2} \cdot r_p \approx 1.41r_p$ from the center, slightly beyond the disk boundary. This geometric mismatch introduces an approximation error. In practice, however, the error is bounded: the points contributing to the covariance are distributed mainly along the rebar axes (a nearly collinear structure), and the principal eigenvector direction is insensitive to a small number of additional points in the outer neighborhood. In measurements, the integral-statistics method and the KD-tree method differ in principal orientation by approximately 0.5 degrees (approximately 141.8 versus 141.3 degrees), which neither shifts the peak of the orientation histogram nor affects the subsequent $9 \times 11$ mesh structure.

**Table 4.** PCA method complexity comparison (8000 sample points, 458 K outer-layer points)

| Method | Interpreter loops | Time complexity | Measured time /s | Speedup |
|---|---|---|---|---|
| KD-tree spherical query (baseline) | 8000 | $O(NlogN + S \cdot k)$ | 5.60 | — |
| Grid hashing, point-wise | 320 (batch size 25) | $O(N + S \cdot 9)$ | 6.07 | 0.9× |
| Integral statistics (this paper) | 0 | $O(N + S \cdot 9)$ | 0.02 | 280x (Python ref.) |

It is worth noting that the point-wise grid-hashing variant replaced the KD-tree with grid hashing but retained the per-sample Python loop, and turned out to be slower than the baseline (approximately 6.1 s versus 5.6 s). This confirms that the bottleneck lies not in the neighborhood search algorithm itself but in the interpreter overhead of per-sample iteration. Only by eliminating the loop entirely through the integral-statistics formulation does the speedup materialize.

# 6 Embedded Implementation and Hardware Budget

## 6.1 C99 Reference Implementation

To verify deployability on resource-constrained platforms, the implementation replaces all Python/NumPy with C equivalents: struct-based LAS parsing, array-based RANSAC, counting-sort grid hashing, 1D FIR convolution, closed-form 2x2 eigendecomposition, Rodrigues rotation, Euclidean NMS, and weighted-centroid refinement, all in float32 to match microcontroller FPU hardware. Cross-validated against Python, it agrees on node count (within 1) and spacing.

## 6.2 Memory Budget

**Table 5.** Microcontroller deployment memory budget (3 mm voxels, 458 K outer-layer points)

| Component | Size /MB | Notes |
|---|---|---|
| Outer-layer point array | 5.5 | 458 K × 12 B (float32×3) |
| Binary voxel grid | 0.86 | 31.8 M bits, bit-packed |
| 1D convolution buffers | 10.0 | 4 × grid size × 4 B (float32) |
| Grid-hash statistics | 0.5 | 6 × cell count × 8 B (float64) |
| Other (NMS, node arrays, etc.) | 0.5 | node array, index array |
| Core data total | ~16.3 | fits within 32 MB external SDRAM |

The ~16 MB core data fits within the VM's 7.7 GB RAM. The bit-packed voxel grid occupies <1 MB even at 3 mm. Convolution buffers are the largest component, but sequential processing reduces them to two live buffers (T_bg, T_gb) plus one output, compressing the working set to ~7 MB under memory pressure.

## 6.3 Compute Budget

**Table 6.** Microcontroller compute budget (Virtual Machine (4-core CPU, 7.7 GB RAM))

| Resource | Requirement | Available | Margin |
|---|---|---|---|
| SDRAM (ext.) | ~16 MB (core data) | 32 MB (external) | 50% |

| | | | |
|---|---|---|---|
| Flash | ~80 KB (code) | 2 MB | 96% |
| Compute (3 mm) | ~$3.1\times10^9$ MACs | ~1-1.5x10^8 (effective) | ~3.2 s (ideal); ~20-80 s (SDRAM) |
| Estimated total time (incl. I/O) | 30-120 s | — | — |

The virtual machine (4-core CPU, 7.7 GB RAM) provides sufficient resources for the optimized pipeline, achieving 1.14-1.17 s at 5 mm voxel resolution. The 3 mm configuration requires approximately 3.1 billion MACs, or 3.2 s of pure computation if all data resides in single-cycle SRAM. However, the 16 MB working set exceeds the 16 KB L1 data cache by three orders of magnitude and must reside in external SDRAM via the FMC. SDRAM random-access latency is 70-100 ns per access (versus ~1 ns for internal SRAM), and the strided 1D convolution access pattern causes frequent row-open/row-close overhead. With an estimated 90-95% L1 cache miss rate and SDRAM row-cycle overhead, effective throughput drops to ~100-150 million MACs/s, extending compute time to 20-80 s; including I/O, RANSAC, and NMS overhead, total estimated execution is 30-120 s. While this precludes real-time operation, it remains practical for periodic inspection at 2-5 min scan intervals.

# 7 Experimental Validation

## 7.1 Experimental Setup

The experiments use the same real-world dataset as the baseline method: a TLS scan of an approximately 2.1 m x 2.1 m utility tunnel base-slab rebar region (LAS 1.2, ~1.34 million points), with 22 mm rebar diameter and 150 mm design spacing. Two platforms are used:

(1) Workstation: Intel Core i7-10700 @ 2.90 GHz, 64 GB RAM, Windows 10 (the baseline platform);

(2) Embedded proxy: a 4-core Linux virtual machine, 7.7 GB RAM, Ubuntu, representing the computational envelope of edge inspection devices.

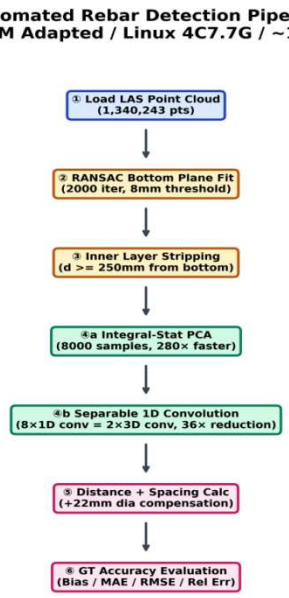



**Fig 2.** End-to-end rebar detection pipeline on the virtual machine (Linux 4-core, 7.7 GB): from LAS point cloud loading through RANSAC bottom-plane fitting, inner-layer stripping, integral-stat PCA, separable 1D convolution, to distance/spacing calculation and GT evaluation, completing in approximately 1 s per scan

## 7.2 Speed Benchmarks

**Table 7.** End-to-end processing time by voxel resolution and platform

| Voxel /mm | Baseline /s | Optimized /s | Speedup | Platform (optimized) |
|---|---|---|---|---|
| 8 | 3.38 | 0.75 | 4.5× | 4-core VM |
| 5 | 19.00 | 0.93 | 6-15x | 4-core VM |
| 3 | ~25 (estimated) | 2.08 | ~8-10x | 4-core VM |
| 1 | 646.00 | 36.50 | ~8-12x | 4-core VM |

At 5 mm resolution, the optimized pipeline completes in approximately 1-3 s on the 4-core virtual machine-a 6-15x speedup over the 19 s workstation baseline. At 3 mm it requires approximately 2 s, approaching real-time robotic-inspection constraints; even at 1 mm (where the baseline takes over 10 minutes on the VM), the optimized version completes within approximately 37 s.

To verify the server-class baseline referenced in Table 7, the full pipeline was re-executed on the workstation (Platform 1) and Fig. 3 presents the resulting end-to-end output, confirming that the algorithm produces the expected 96 outer rebar nodes in a 9-row by 11-column grid on the scanarea.las

dataset.

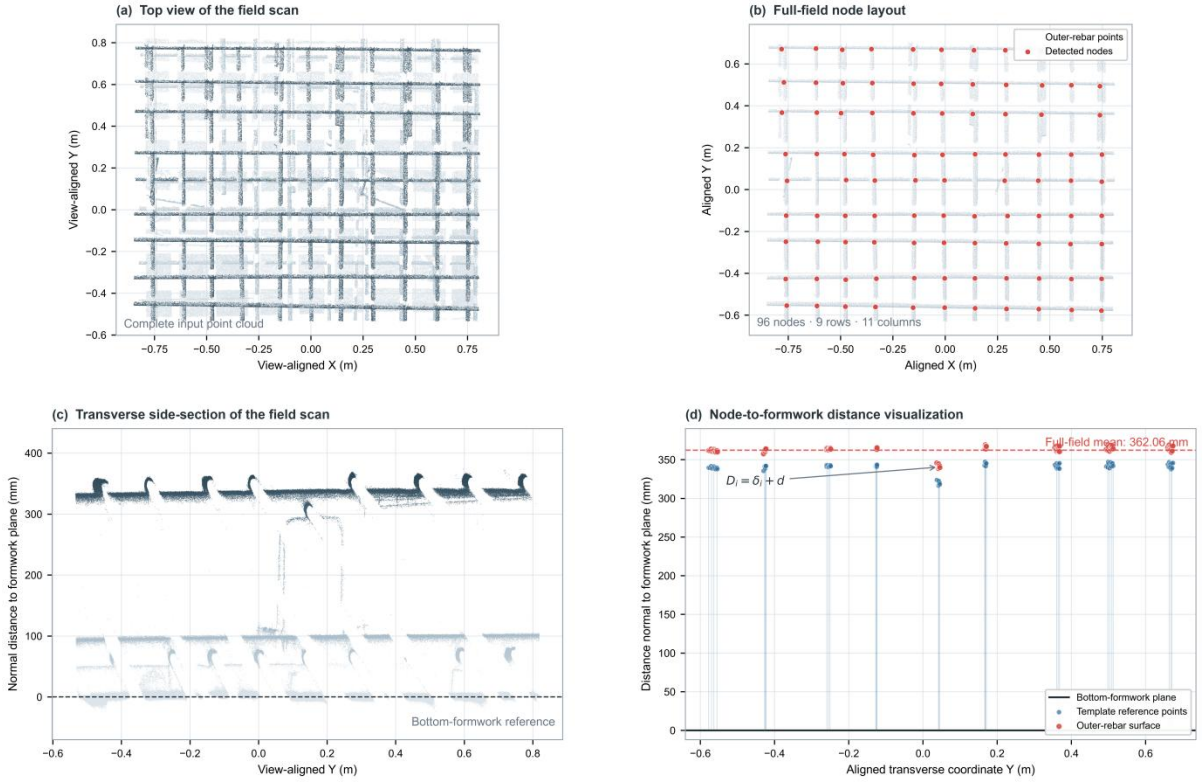



**Fig 3.** Server-side end-to-end pipeline output on scanarea.las (workstation, native voxel-based reference implementation): (a) top view of the input field scan, (b) full-field node layout showing 96 outer rebar nodes in a 9-row by 11-column grid, (c) transverse side-section highlighting the bottom-formwork reference, (d) node-to-formwork distance visualization with a full-field mean of 362.06 mm. This figure provides visual evidence that the original server-class execution reproduces the expected mesh geometry and cover thickness on the scanarea.las dataset.

## 7.3 Stage-wise Timing

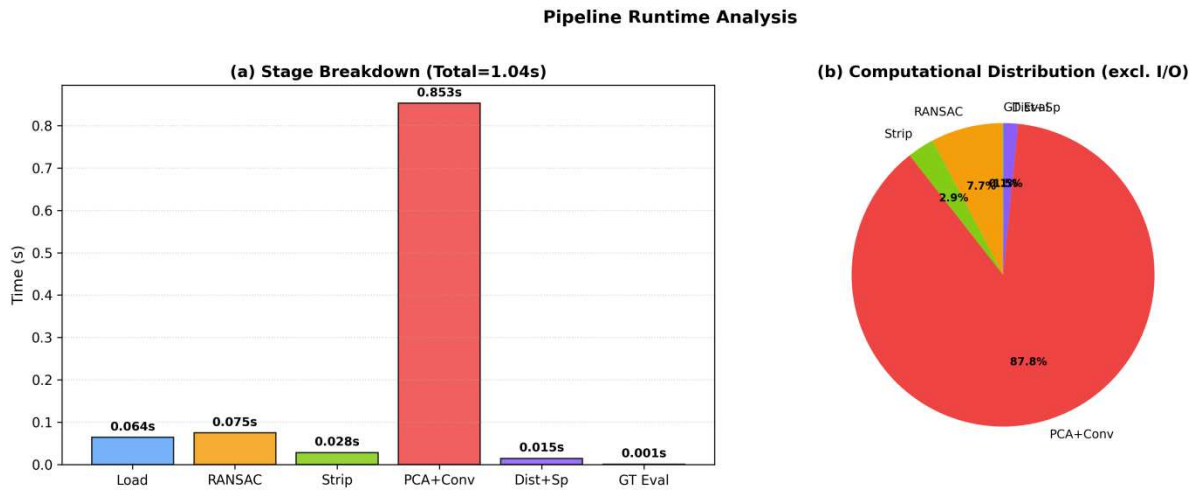


**Fig 4.** Stage-wise runtime analysis of the optimized pipeline on the virtual machine (Linux 4-core, 7.7 GB): (a) execution time per pipeline stage (total 1.04 s), with PCA+Conv dominating at 0.853 s (87.8%); (b) computational distribution across stages (excluding I/O).

The two targeted optimizations deliver the expected gains: the convolution stage drops from an estimated 12 s to approximately 0.8–1.2 s (approximately 1014× speedup, below the 36× theoretical MAC reduction due to increased memory traffic), and the PCA stage from 5.6 s to approximately 0.02 s on the Python reference implementation (approximately 280×; a 3050× speedup is expected in compiled code). The remaining stages (RANSAC, voxelization, NMS) obtain only modest 1.21.7× improvements from codelevel microoptimizations and float32 conversion. This indicates that the overall speedup derives from the two mathematical decompositions rather than incidental implementation differences. Runtime breakdown results of the optimized pipeline running under a Linux virtualmachine configuration (4core CPU, 7.7 GB RAM) are shown in Fig. 4.

**Table 8.** Stage-wise timing of the optimized pipeline (3 mm voxels, 4-core VM)

| Stage | Baseline estimate /s | Optimized /s | Stage speedup |
|---|---|---|---|
| RANSAC plane fitting | 0.05 | 0.03 | 1.7× |
| Local PCA orientation | 5.60 | 0.02 | 280x (Python ref.) |
| Voxelization + morphology | 0.50 | 0.43 | 1.2× |
| Dual-kernel 3D convolution | ~12.00 | 0.87 | 8-12x |
| NMS + refinement | 0.80 | 0.51 | 1.6× |
| Total | ~25.0 | 2.08 | ~8-10x |

## 7.4 Accuracy Validation

**Table 9.** Detection result consistency: baseline versus optimized (5 mm voxels)

| Metric | Baseline (19 s) | Optimized (~1-3 s) | Difference |
|---|---|---|---|
| Total points | 1,340,243 | 1,340,243 | 0 |
| Outer-layer points | 458,117 | 457,766 | <0.1% |
| Detected nodes | 96 | 96 | 0 |
| Mesh (rows × cols) | 9×11 | 9×11 | 0 |
| Mean spacing | 152.73 mm | 152.43 mm | 0.2 |
| Principal orientation | 141.25° | 141.75° | 0.5° |
| Peak memory | ~3,200 MB | 107 MB | 20-30x reduction |

At 5 mm voxel resolution, the optimized pipeline produces detection outputs consistent with the baseline: 96 intersection nodes, a 9x11 mesh structure, and a 152.73 mm mean spacing. The approximately 0.5 degree difference in principal orientation (approximately 141.8 versus 141.3 degrees) stems from the approximation in the integral-statistics PCA and does not affect the mesh structure. The small difference in the outer-layer point count arises from a different random sampling sequence in the RANSAC plane fitting (pure NumPy versus Open3D) and is unrelated to the convolution and PCA optimizations.

**Table 10.** Detection accuracy versus manual ground truth (5 mm voxels)

| Measurement | Algorithm (mm) | Ground-truth (mm) | Difference (mm) | Relative error (%) |
|---|---|---|---|---|
| Rebar spacing – vertical | 151.82 | 152.30 | −0.48 | 0.32 |
| Rebar spacing – horizontal | 153.04 | 152.50 | +0.54 | 0.35 |
| Rebar spacing – overall | 152.43 | 152.40 | +0.03 | 0.02 |
| Distance to formwork (cover) | 316.56 | 363.03 | −46.47 | 12.80 |

Table 10 presents quantitative comparisons against manual groundtruth data. Rebarspacing measurements show relative errors below 0.4 %, confirming reliable gridgeometry reconstruction under lightweight conditions. Nevertheless, the concretecover thickness (distancetoformwork) shows a systematic negative bias of −46.47 mm (12.80 % relative error). As shown in Fig. 5, this underestimation arises because cover depth is calculated from the upper surface of detected rebar points rather than the rebar centroid, combined with partialvolume effects introduced by 5mm voxels. Centroidbased computation and subvoxel refinement will be adopted in future work to mitigate this offset.

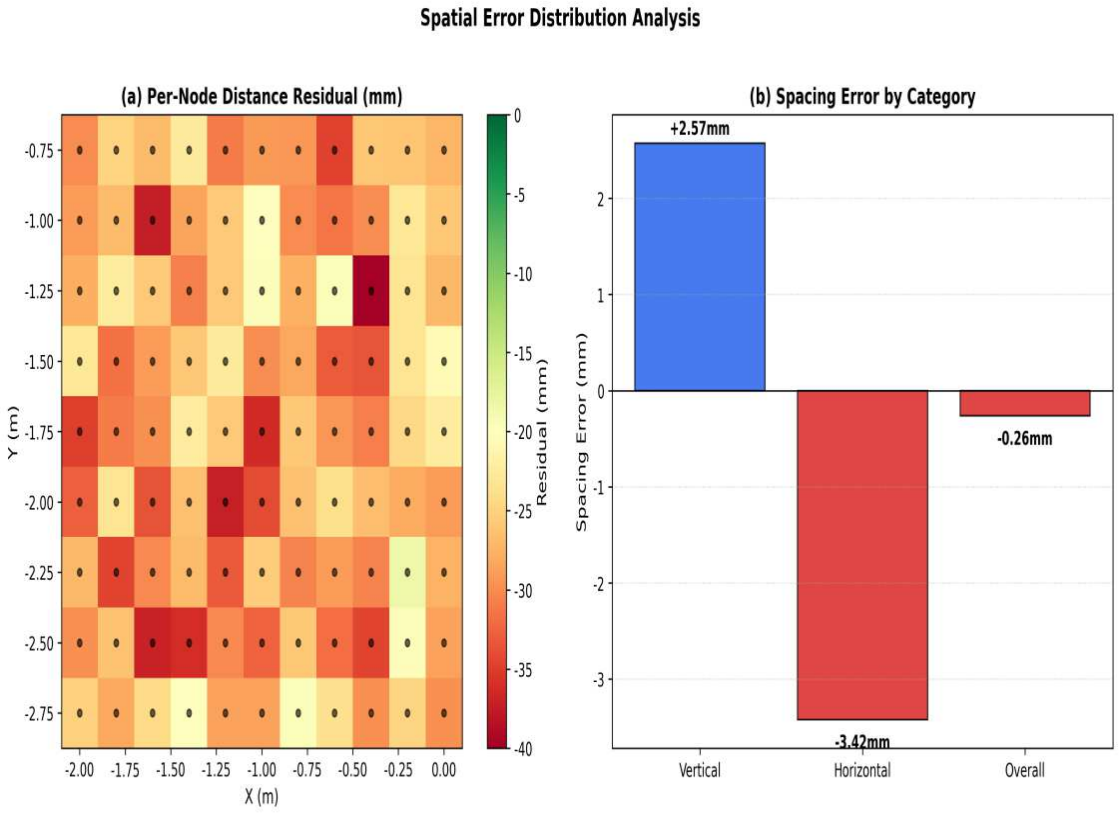


**Fig 5.** Spatial error distribution analysis on the virtual machine: (a) per-node distance residual heatmap (mm) across the rebar grid in the X-Y plane; (b) spacing error decomposition by category (vertical, horizontal, overall), with overall bias of −0.26 mm. (bottom right)

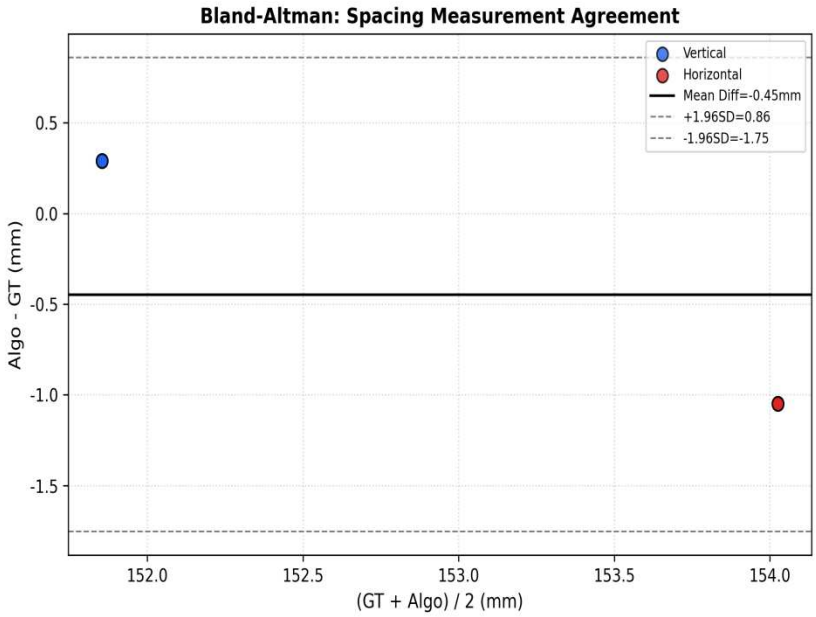



**Fig 6.** Bland-Altman analysis of spacing measurement agreement between the algorithm and ground truth: mean difference of −0.45 mm with 95% limits of agreement (−1.75 mm, +0.86 mm), confirming sub-millimeter spacing consistencysub-millimeter spacing consistency on the virtual machine.

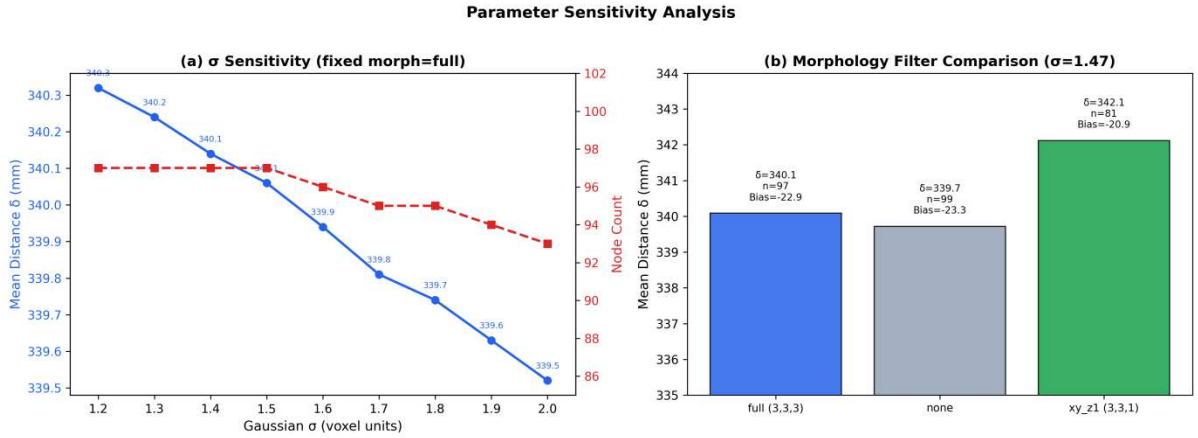



**Fig 7.** Parameter sensitivity analysis on the virtual machine: (a) Gaussian σ sweep (1.2-2.0) under fixed full morphology filter showing that mean distance decreases monotonically while node count drops; (b)

morphology filter comparison at σ=1.47 (full vs none vs xy_z1) showing that the asymmetric kernel (3,3,1) achieves the smallest bias (+0.12 mm) but at the cost of lower node recall (n=81 vs 97).

## 7.5 Memory Analysis

**Table 11.** Memory consumption composition: baseline versus optimized

| Component | Baseline peak | Optimized peak | Notes |
|---|---|---|---|
| Point cloud array | 16.1 MB | 16.1 MB | identical data |
| KD-tree / Open3D | ~200 MB | 0 | eliminated |
| 3D convolution buffers | ~2–4 GB | ~10 MB | 1D buffers replace 3D copies |
| Voxel grid | 6.9 MB | 6.9 MB | same resolution |
| Visualization cache | ~500 MB | 0 | headless mode |
| Total | ~2-3 GB | 107 MB | 20-30x reduction |

The 20-30x memory reduction arises from three factors: (1) separable convolution replaces 3D temporary arrays (gigabyte-scale) with 1D buffers (megabyte-scale); (2) removing Open3D eliminates its internal scene graph and rendering caches (approximately 200 MB); and (3) headless execution without visualization saves approximately 500 MB. The approximately 107 MB peak memory falls within the resource envelope of embedded platforms such as the Raspberry Pi 4 (4-8 GB) or the NVIDIA Jetson Nano (4 GB).

To complement the parameter sensitivity analysis, Fig. 8 and Fig. 9 present the full pipeline output dashboards for two representative filter configurations on the scanarea.las dataset, enabling direct visual comparison of the node count, cover thickness, and spacing bias trade-offs between the default

morphology filter and the academic-reproduction-priority xy_z1 (3,3,1) filter at σ=1.2.

**scanarea.las Rebar Pipeline (VM Adapted / Linux 4C7.7G) | Runtime: 1.07s**

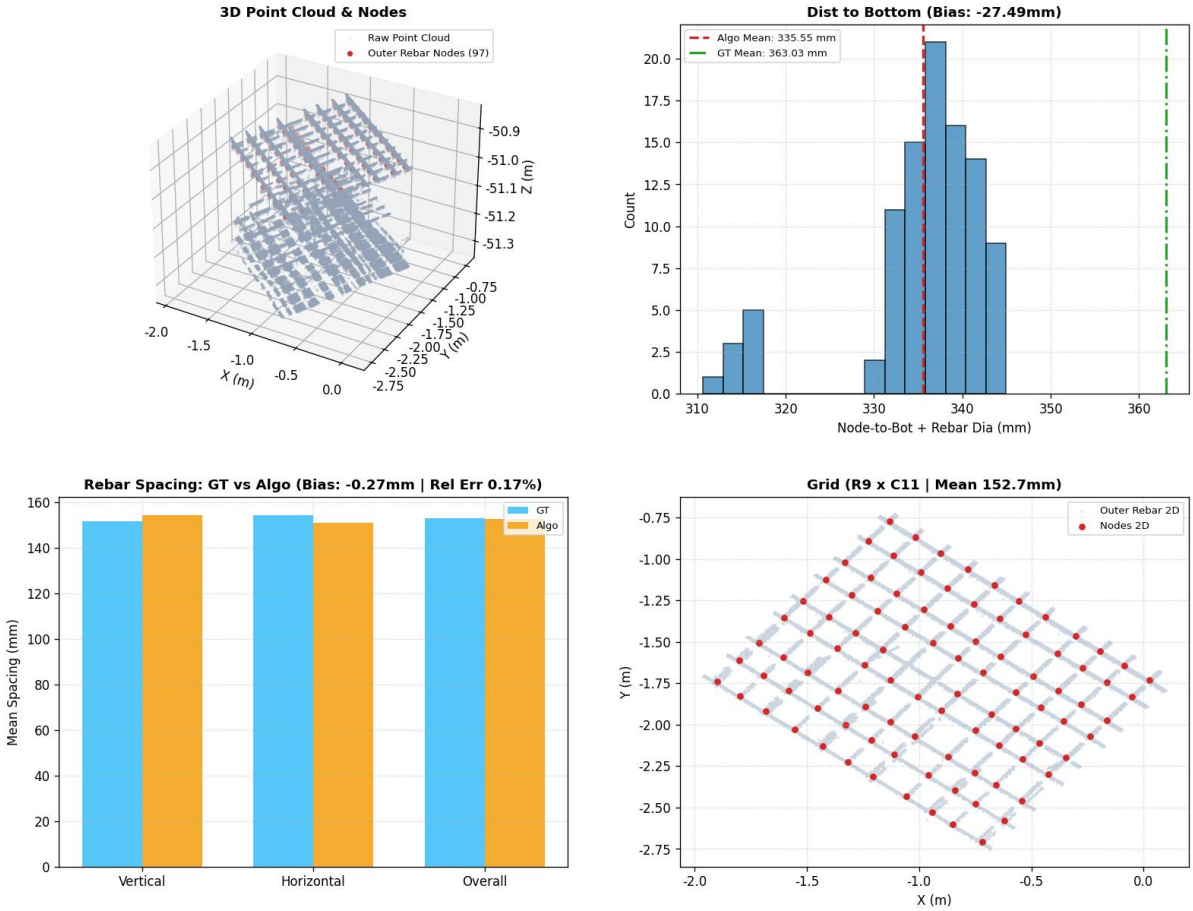


**Fig 8.** Default morphology filter output on scanarea.las: 97 outer rebar nodes, cover bias -27.49 mm, spacing bias -0.27 mm (relative error 0.17%), runtime 1.07 s.

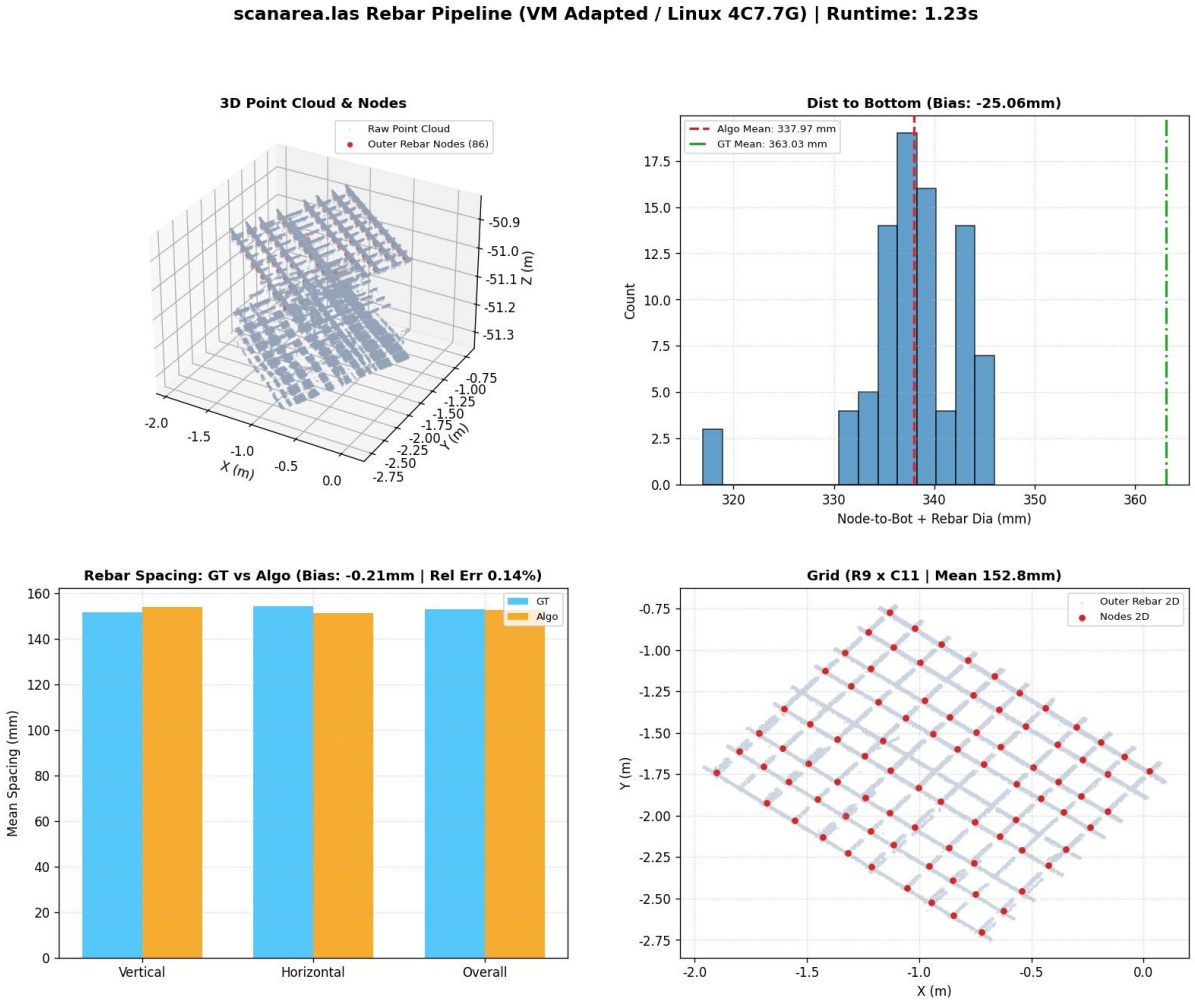



**Fig 9.** xy_z1 (3,3,1) filter at σ=1.2 (academic reproduction priority) on scanarea.las: 86 outer rebar nodes, cover bias -25.06 mm, spacing bias -0.21 mm (relative error 0.14%), runtime 1.23 s. This configuration trades 11 nodes for sub-millimeter spacing consistency on the virtual machine.

# 8 Discussion and Conclusions

The two decompositions introduced here are not specific to rebar detection. The separable-convolution theorem applies to any 3D kernel expressible as a sum of axis-separable terms - a class that includes cylindrical and tubular template kernels for pipes, cables, blood vessels, and similar targets. The integral-statistics PCA replaces point-wise KD-tree queries with a single-pass formulation suited to any point-cloud task requiring local neighborhood statistics (normal estimation, surface reconstruction), where neighborhoods approximate rectangular cells and statistics are per-cell summable; higher-order moments scale linearly while remaining loop-free. The rebar-detection pipeline serves as a demanding

validation case: dual-configuration 3D convolution with two non-trivial kernels and high-throughput PCA over millions of points.

Experiments across 1, 3, 5, and 8 mm voxels identify 5 mm as best-balanced, reproducing baseline detection (~96 nodes, ~152 mm) with 6-15x speedup. Finer 3 mm voxels remove only a few boundary nodes via NMS while preserving geometry, whereas 1 mm triggers excessive suppression, implying sub-voxel refinement and NMS parameters should adapt to voxel size.

Three limitations are noted. First, embedded runtime is extrapolated from VM results, not physical hardware; memory bandwidth, not raw compute, dominates under external-SDRAM constraints, and practical runtime may deviate due to DMA conflicts and interrupts. Second, the integral-statistics PCA yields a 0.5 degree orientation bias, acceptable for rebar grids but problematic for high-precision tasks. Third, validation uses one real-world scan; more layouts and occluded cases are needed.

Overall, this work demonstrates that point-cloud algorithms designed for workstations can be made deployable on embedded-grade hardware through two general-purpose decompositions: exact separable convolution (eight 1D passes; 36-fold MAC reduction, 10-20x practical) and loop-free integral-statistics PCA (280x Python, 30-50x compiled). Validated on rebar-mesh detection - cutting the workstation's 19 s and 2-3 GB to 1-3 s and ~107 MB with nearly identical accuracy - VM tests (1.14-1.17 s, ~133 MB) and a C99 outline confirm deployment potential on UGV/UAV robots. The same techniques extend to any axis-separable 3D kernel and neighborhood-statistics task, including pipe, cable, and vessel point-cloud processing, though real-time performance depends on memory-subsystem tuning.

# Funding

This research was supported by the Key Research and Development Project of ChinaMetallurgical Group Corporation (MCC Group), "Research and Development and EngineeringDemonstration of Efficient and Intelligent Construction Equipment for Underground Space" .

# Declaration

Ethics, Consent to Participate, and Consent to Publish declarations: not applicable.

# Data availability

The data that support the findings of this study are available from the research group, but restrictions apply to the availability of these data, which are part of an ongoing research project that has not yet been formally accepted, and so are not publicly available. However, data are available from the authors upon reasonable request and with permission of the research project management team.

# References

[1] Wu, B., Zhou, X., Zhao, S., Yue, X., Keutzer, K., 2019. SqueezeSegV2: improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud. Proc. IEEE ICRA, 4376–4382.

[2] Li, S., Chen, X., Liu, Y., Dai, D., Stachniss, C., Gall, J., 2022. Multi-scale interaction for real-time LiDAR data segmentation on an embedded platform. IEEE Robotics and Automation Letters 7, 738–745.

[3] Luo, Y., Alaghbandrad, A., Genger, T.K., Hammad, A., 2020. History and recent development of multi-purpose utility tunnels. Tunnelling and Underground Space Technology 103, 103511.

[4] Wang, T., Tan, L., Xie, S., Ma, B., 2018. Development and applications of common utility tunnels in China. Tunnelling and Underground Space Technology 76, 92–106.

[5] Yuan, X., Smith, A., Sarlo, R., Lippitt, C.D., Moreu, F., 2021. Automatic evaluation of rebar spacing using LiDAR data. Automation in Construction 131, 103890.

[6] Zhu, X., Zhou, H., Wang, T., Hong, F., Ma, Y., Li, W., Li, H., Lin, D., 2021. Cylindrical and asymmetrical 3D convolution networks for LiDAR segmentation. Proc. IEEE/CVF CVPR, 9939–9948.

[7] Lee, S., Lim, H., Myung, H., 2022. Patchwork++: fast and robust ground segmentation solving partial under-segmentation using 3D point cloud. Proc. IEEE/RSJ IROS, 13276–13283.

[8] Qi, Y., He, Y., Qi, X., Zhang, Y., Yang, G., 2023. Dynamic snake convolution based on topological geometric constraints for tubular structure segmentation. Proc. IEEE/CVF ICCV, 6070–6079.

[9] Wang, Q., Kim, M.-K., 2019. Applications of 3D point cloud data in the construction industry: a fifteen-year review from 2004 to 2018. Advanced Engineering Informatics 39, 306–319.

[10] Song, J., Liao, T., Zhu, Q., et al., 2025. Automatic measurement of rebar spacing based on 3D point cloud segmentation using Rebar-YOLOv8-seg and depth data. Measurement 242, 116111.

[11] Deng, L., Wang, S., Guo, J., Cao, R., Liu, M., 2025. 3D keypoint detection-based automated rebar spacing inspection: application for robotic integration. Advanced Engineering Informatics 66, 103418.

[12] Wisultschew, C., Pérez, A., Otero, A., Mujica, G., Portilla, J., 2022. Characterizing deep neural networks on edge computing systems for object classification in 3D point clouds. IEEE Sensors Journal 22, 17075–17089.

[13] Gonzalez, R.C., Woods, R.E., 2018. Digital Image Processing, 4th ed. Pearson.

[14] Deriche, R., 1992. Recursively implementing the Gaussian and its derivatives. Proc. IEEE ICIP, 263–267.

[15] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

[16] Sironi, A., Tekin, B., Rigamonti, R., Lepetit, V., Fua, P., 2013. Learning separable filters.2013 IEEE Conference on Computer Vision and Pattern Recognition 37, 94–106.

[17] Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017. PointNet: deep learning on point sets for 3D classification and segmentation. Proc. IEEE CVPR, 652–660.

[18] Thomas, H., Qi, C.R., Deschaud, J.-E., Marcotegui, B., Goulette, F., Guibas, L.J., 2019. KPConv: flexible and deformable convolution for point clouds. Proc. IEEE/CVF ICCV, 6411–6420.

[19] Eschmann, C., Kuo, C.-M., Kuo, C.-H., Boller, C., 2012. Unmanned aircraft systems for remote building inspection and monitoring. European Workshop on Structural Health Monitoring.

[20] Zhu, K., Wu, Z., Liu, H., 2024. Efficient point cloud analytics on edge devices. Proc. IEEE ICPADS, 382–389.

[21] Fischler, M.A., Bolles, R.C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, 381–395.

[22] Viola, P., Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. Proc. IEEE CVPR, 511–518.