3D-BBS: Global Localization for 3D Point Cloud Scan Matching Using Branch-and-Bound Algorithm

EECE5550 Mobile Robotics Project Report

By

Aryaman Shardul

***Abstract*—This project implements a CPU-based version of the 3D-BBS algorithm for Global Localization using LiDAR point clouds. By leveraging a branch-and-bound (BnB) search strategy over a voxelized map, the method efficiently aligns 3D scans with a prebuilt environment map. While the original paper uses hierarchical voxel resolutions for speed, my implementation maintains a fixed-resolution voxel map but preserves the hierarchical search behavior of the algorithm. I evaluated the system on the KITTI dataset and compared the performance with key results from the original paper. My implementation achieves comparable alignment quality, but requires higher computation time. Experimental results demonstrate successful localization even from perturbed or fake initializations, validating the robustness of the BnB-based approach. The implementation achieved sub-5m Absolute Trajectory Error (ATE) and sub-1° Angular Orientation Error (AOE) on KITTI sequence 00 from reasonable initialization. Limitations and future enhancements are discussed.**

***Index Terms*—Global Localization, 3D LiDAR Scans, Branch-and-Bound search, Point cloud compression, Memory Management**

I. Introduction

Global localization [3] is the process of estimating the pose of a robot within a known map without any prior information about its initial position. The 3D-BBS [1] algorithm formulates the task as a 4D optimization over position and yaw and solves it efficiently using a hierarchical branch-and-bound (BnB) strategy. It is a critical problem in autonomous navigation, especially in large, GPS-denied, or repetitive environments where accurate position estimation is essential for tasks such as path planning, mapping, and obstacle avoidance.

Traditional scan-matching methods such as Iterative Closest Point (ICP [4]) or feature-based matching work well when a good initial guess is available. However, they exhibit serious limitations when applied to Global Localization [3] scenarios:

- **Dependence on Initial Pose:** ICP [4] and many local optimization methods assume that the robot's pose is already close to the true solution. Without a good initialization, they often diverge or converge to incorrect solutions.
- **Vulnerability to Local Minima:** In environments with repetitive or symmetric structures (e.g. urban roads, corridors, or building facades), these methods can latch onto incorrect matches due to ambiguity in the scan, resulting in poor alignment.
- **Scalability Issues:** Searching over large maps is computationally expensive with naive brute-force approaches. Local methods cannot handle large search spaces effectively.

To address these challenges, I implement a Global Localization [3] algorithm inspired by 3D-BBS [1], which casts the scan-to-map alignment task as a 4D optimization problem over the robot's pose (x,y,z,yaw). Instead of relying on local gradients, it uses a Branch-and-Bound (BnB) [5], [6] strategy to systematically explore the search space and efficiently prune unpromising candidates. Candidate poses are evaluated based on the overlap between the scan and a voxelized 3D map, allowing robust localization even in large or ambiguous maps.

Importantly, the original paper achieves real-time performance by parallelizing BnB on GPU. In my implementation, I test this system in a CPU-only setup on real-world KITTI [2] data, aiming to study both the robustness of the algorithm and its computational scalability in resource-constrained environments.

This study not only highlights the strengths of global BnB search [5], [6] (robustness to poor initial guesses, map-scale invariance) but also reveals its practical limitations when high-performance hardware is unavailable.

II. Approach

This section describes the end-to-end implementation of the 3D-BBS algorithm using KITTI [2] LiDAR data. The system performs Global Localization [3] by aligning a scan to a prebuilt voxel map using a hierarchical branch-and-bound (BnB) [5], [6] search strategy. The overall workflow is shown in

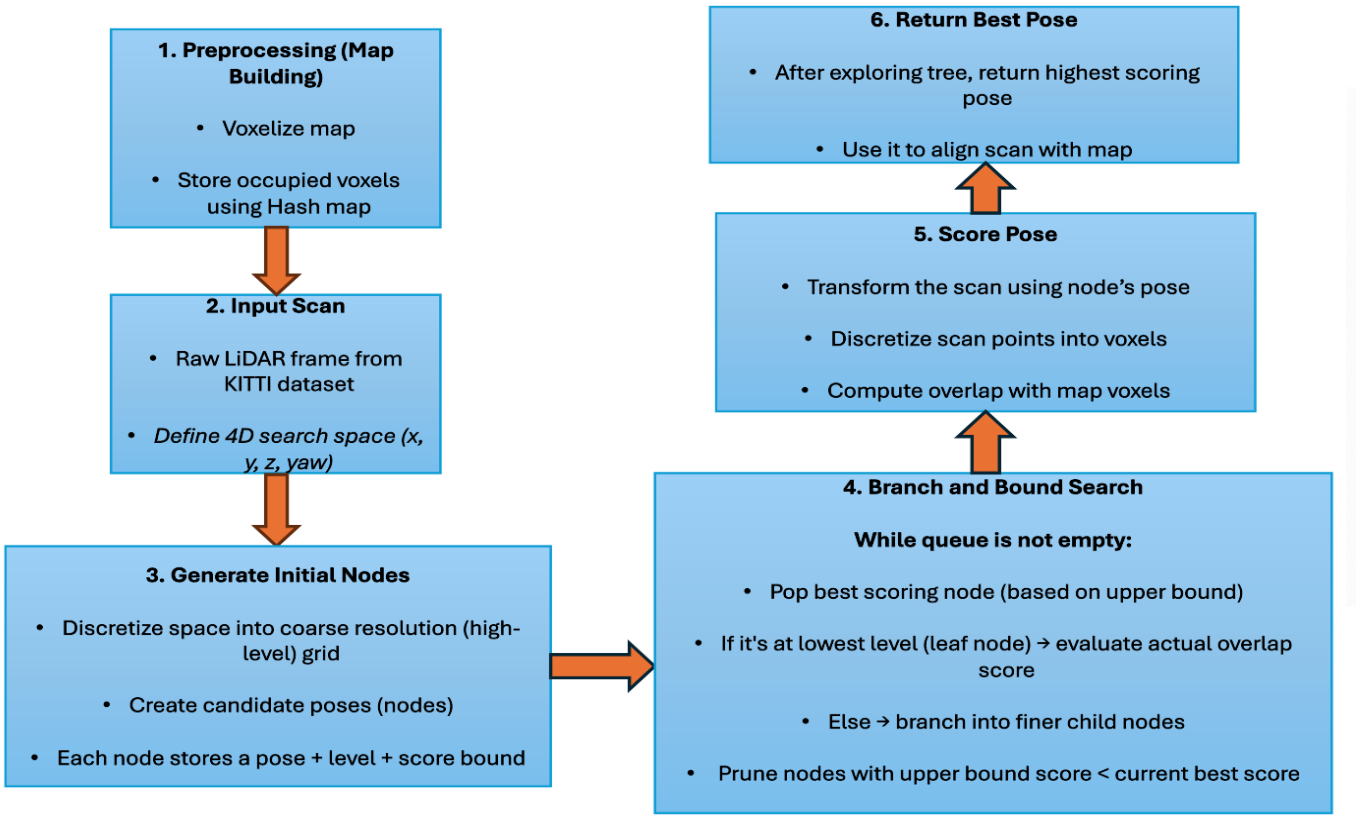


Fig. 1: Flowchart of the 3D-BBS scan matching pipeline implemented for Global Localization.

Fig. 1. Also, the block diagram in Fig. 2 illustrates the full processing pipeline implemented for Global Localization [3]. The system begins with raw LiDAR scans from the KITTI [2] dataset, which are aggregated into a voxel map. A hierarchical branch-and-bound (BnB) [5], [6] search then explores a 4D pose space (x, y, z, yaw) to align the incoming scan with the map. Each candidate pose is scored based on voxel overlap. The best-scoring pose is exported and visualized. Modules implemented in C++ are highlighted, with final evaluation and visualization handled in Python.

A. Problem Formulation

The goal of 3D Global Localization [3] is to estimate the 4-DoF pose of a LiDAR scan (x, y, z, yaw) with respect to a known 3D map. Given a voxelized map $M = \{m_n\}$ and an input scan $S = \{s_k\}$, we seek the optimal transformation T_x that maximizes the alignment:

$$x^* = \arg\max_{x \in X} \sum_{s \in \mathcal{S}} \mathbf{1}\left[\text{voxel}(T_x s) \in \mathcal{V}\right] \tag{1}$$

Where:

- $\mathcal{S} = \{s_1, s_2, \ldots, s_K\}$ is the scan (a set of 3D points)
- $\mathcal{V} \subset \mathbb{R}^3$ is the voxel map (set of occupied voxel centers)
- $T_x \in SE(3)$ is a rigid transformation defined by pose $x = (x, y, z, \text{yaw})$
- $T_x s$ applies the rigid transformation to scan point s
- $\text{voxel}(\cdot)$ maps a 3D point to its corresponding voxel index
- $\mathbf{1}[\cdot]$ is the indicator function (1 if true, 0 otherwise)

B. Map Representation with Voxel Hashing

To store the 3D map efficiently, I used a spatial hash table instead of dense 3D grids.
Each voxel coordinate v=[vx, vy, vz] is computed as:

$$v_i = \left\lfloor \frac{r_i}{p} \right\rfloor, \quad i \in \{x, y, z\} \tag{2}$$

Where $\lfloor \rfloor$ represents the floor function. These integer voxel indices are hashed using a hash function [7]:

$$\text{hash}(v) = (v_x \cdot p_1 \oplus v_y \cdot p_2 \oplus v_z \cdot p_3) \bmod T \tag{3}$$

where p1, p2, p3 are large prime constants and T is the hash table size. Collisions are handled by open addressing. For my code, I used p1 = 73856093, p2 = 19349663, and p3 = 83492791. I stored occupied voxels using the $std :: unordered\_set < uint64\_t >$ keyed by the hashed index.

C. Scan Transformation and Scoring

The scan is transformed using standard rigid body transformations (translation and yaw rotation). For each transformed point, the voxel occupancy is checked using the hash table.

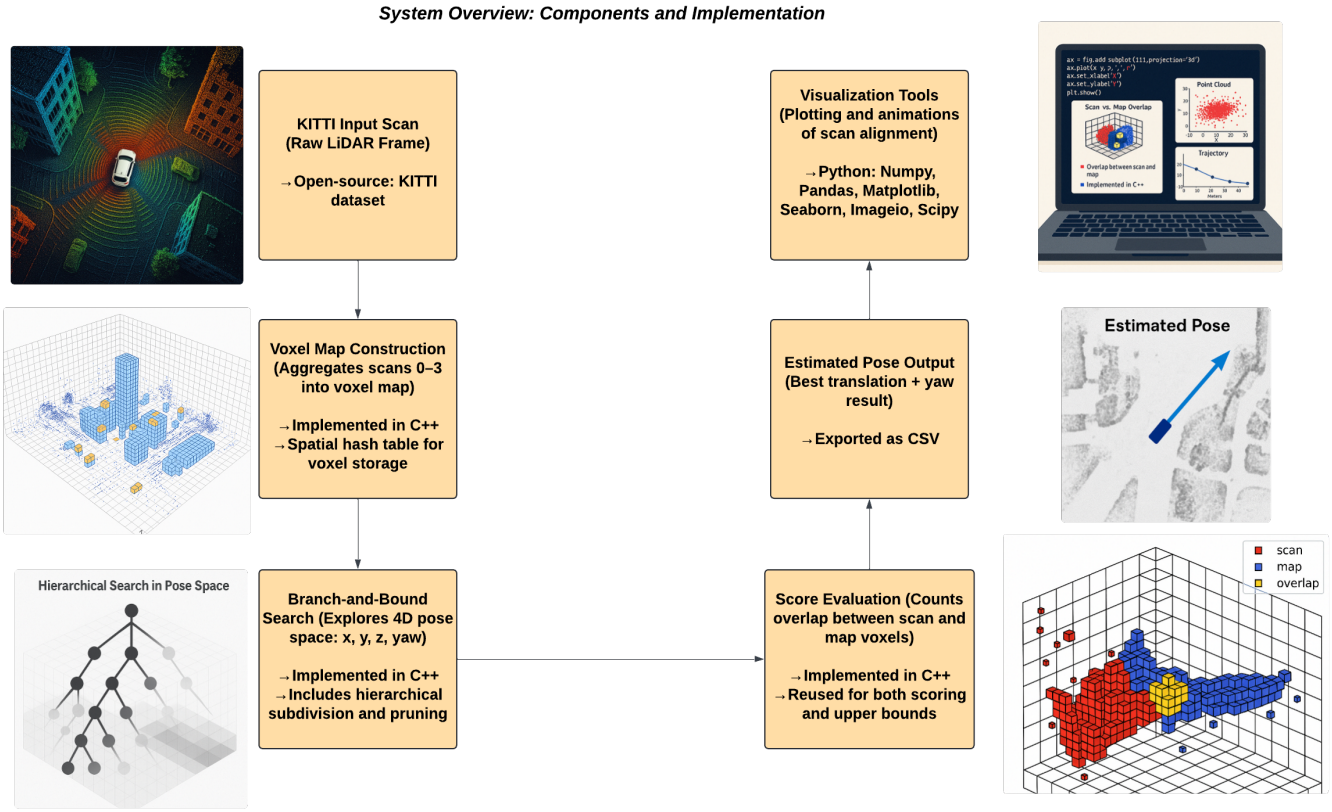


Fig. 2: Block Diagram depicting an overview of the system - Component-wise Breakdown of the 3D-BBS Localization Pipeline.

The score is the number of transformed points that land in occupied voxels:

$$\text{score}(x) = \sum_{k=1}^{K} \mathbf{1}\left[\text{voxel}(T_x s_k) \in \mathcal{V}\right] \tag{4}$$

This score is used both for evaluating candidate poses and as an upper bound during BnB.

D. Search Space Definition

To perform Global Localization [3], I define a discrete 4D search space over the robot's possible poses. Each candidate pose is parameterized by translation (x,y,z) and yaw angle θ. Pitch and roll are not considered in this formulation, under the assumption that the robot operates on relatively flat terrain (as is typical in autonomous driving datasets such as KITTI [2]).

1) Translation Sampling: The translation domain is bounded within a cube centered around the origin (or an initial guess, in the case of constrained search). The uniform sampling steps along each axis are defined as:
$x, y, z \in [-a, a]$ in steps of $\Delta x = \Delta y = \Delta z = 0.5$ meters.
Where a represents the maximum extent of the search along each direction. For example, with a = 1.0, this results in a 3×3×3=27 grid of possible translations at each level of the Branch and Bound (BnB) tree.

2) Yaw Sampling: The yaw orientation is discretized uniformly over the full 360° range:
$\theta \in [0, 2\pi)$ in steps of $\Delta\theta = \frac{\pi}{6}$ radians (30°)
This results in 12 discrete yaw values at each level of the Branch and Bound (BnB) tree.

3) Search Nodes: Each node in the BnB tree represents a unique 4D cell in the search space:
Node n $= (x_i, y_j, z_k, \theta_l)$
Here $(x_k i, y_j, z_k)$ are the discrete translation coordinates and θ_l is the corresponding yaw bin. At the coarsest level, the node covers a larger region of the pose space, and as branching occurs, this region is subdivided further, refining the search.

4) Hierarchical Subdivision: At each level, the BnB algorithm recursively subdivides the nodes into finer children, halving the translation step size and adding local angular offsets (e.g., 5°) to increase the resolution around promising candidates. This hierarchical search enables the algorithm to prioritize high-score regions while pruning low-scoring nodes early.

Algorithm 1 Branch-and-Bound Scan Matching

Initialize priority queue $\mathcal{C}$ with all nodes at level $l_{\max}$
best_score $\leftarrow -1$
best_node $\leftarrow$ None
while $\mathcal{C}$ is not empty **do**
  Pop node c with highest score from $\mathcal{C}$
  if c.level $== 0$ **then**
    score $\leftarrow$ ComputeScore(c)
    if score $>$ best_score **then**
      best_score $\leftarrow$ score
      best_node $\leftarrow c$
    end if
  else
    children $\leftarrow$ Branch(c)
    for all child in children **do**
      score $\leftarrow$ ComputeUpperBound(child)
      if score $>$ best_score **then**
        Push child into $\mathcal{C}$
      end if
    end for
  end if
end while
return best_node and best_score

This algorithm version modifies the original BBS algorithm from [1] by using a max-priority queue ordered by upper bound score and a hierarchical voxel map with pruning at each level.

E. Branch and Bound Search

The core of the 3D-BBS algorithm lies in using Branch-and-Bound (BnB) [5], [6] to efficiently search the 4D pose space comprising translation (x,y,z) and yaw (θ). Each node in the BnB tree represents a discrete region within this space.

1) Node Representation and Branching: At each level l of the tree, a node represents a subregion of the total search space. The resolution of each node is determined by the level:

$$\Delta x_l = \Delta y_l = \Delta z_l = 2^l \Delta_0, \quad \Delta\theta_l = 2^l \Delta\theta_0 \tag{5}$$

where Δ_0 is the base resolution (e.g., 0.5 m) and $\Delta\theta_0$ is the initial yaw step size (e.g., $\frac{\pi}{6}$).

Formally, let a node at level l have integer indices $(c_x, c_y, c_z, c_\theta)$. The children at level $l-1$ are defined as:

$$\begin{aligned} \mathcal{C}_c^l = \big\{ & (2c_x + j_x,\ 2c_y + j_y, \\ & 2c_z + j_z,\ c_\theta + j_\theta,\ l-1) \\ & \mid j_x, j_y, j_z, j_\theta \in \{0,1\} \big\} \end{aligned} \tag{6}$$

This yields $2^4 = 16$ children per node. Each child represents a finer subregion in the pose space.

Each parent node is subdivided into 16 children by:

- Splitting the 3D translation into 8 octants (2 subdivisions along each axis).
- Splitting yaw into 2 angular subdivisions.

This results in: Children per node = $2^3 \times 2 = 16$

Rotational Subdivision: The yaw range $[W_{\min}, W_{\max}]$ is subdivided at each level. The number of angular bins $a^{(l)}$ is given by:

$$a^{(l)} = \left\lceil \frac{W_{\max} - W_{\min}}{\Delta\theta_l} \right\rceil \tag{7}$$

where $\Delta\theta_l$ is the yaw resolution at level l. The step size is:

$$\delta^{(l)} = \frac{W_{\max} - W_{\min}}{a^{(l)}} \tag{8}$$

Each rotational child is indexed with:

$$c_\theta^{(child)} = a^{(l)} \cdot c_\theta + j_\theta, \quad j_\theta \in \{0, 1, ..., a^{(l)} - 1\} \tag{9}$$

2) Scoring and Upper Bound Computation: Each node maintains an upper bound score representing the maximum possible alignment score within that region. The scoring function is based on the number of transformed scan points that fall into occupied voxels in the map:

$$\text{score}(T) = \sum_{k=1}^{K} \mathbf{1}\left[\text{voxel}(T s_k) \in \mathcal{V}\right] \tag{10}$$

where:

- s_k is the k^{th} scan point,
- T is the candidate transformation,
- $\mathcal{V}$ is the set of occupied voxels in the map,
- $\mathbf{1}[\cdot]$ is the indicator function.

This function is reused for both evaluating nodes and estimating their upper bounds.

3) Pruning Strategy: To avoid evaluating unpromising regions, the BnB algorithm prunes any node whose upper bound score U is lower than the best known score S^*:

$$\text{If } U < S^*, \text{ prune the node.}$$

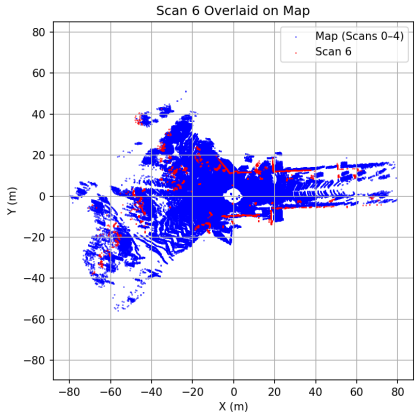
This ensures that only regions with potential to improve the solution are explored, dramatically reducing computational load.

4) Queuing and Expansion Order: The BnB [5], [6] search uses a max-priority queue sorted by upper bound scores:

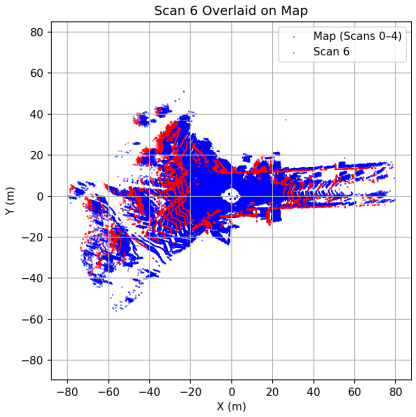
- Nodes with higher potential scores are expanded first.
- Ties are broken arbitrarily or by lowest level (finer resolution preferred late).

The use of Best-First Search (BFS) behavior in early iterations allows faster convergence to promising regions before deeper exploration.

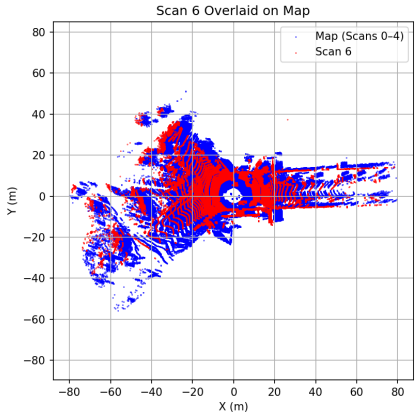
This queuing mechanism allows the algorithm to operate in a hybrid mode: coarse-to-fine hierarchical refinement with best-first prioritization.



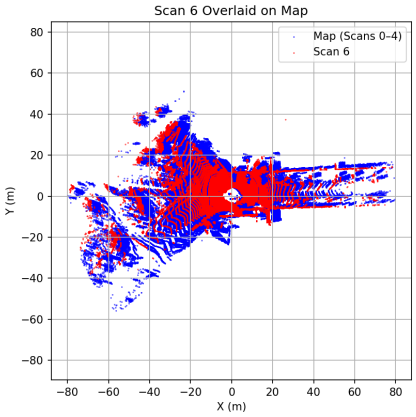
(a) Timestep 1



(b) Timestep 10



(c) Timestep 20



(d) Timestep 30

Fig. 3: Visualization of Scan 6 being globally aligned to a map built from Scans 0–4. Red points represent the transformed scan under different pose guesses. As the Branch-and-Bound search progresses, the scan shifts from misaligned outskirts to a tightly overlapping position, maximizing alignment with the blue voxel map.

5) Termination: The algorithm terminates when:

- The queue is empty, or
- A maximum number of nodes have been expanded.

The node with the highest confirmed score is selected as the final estimated pose.

III. Results

This section describes the various experiments performed to test the robustness of the algorithm and the results I got.

A. Dataset and Experimental Setup

The KITTI [2] Vision Benchmark Suite was selected as the testing environment due to its challenging real-world urban driving scenarios and publicly available ground truth. Specifically, Sequence 00 from the KITTI [2] Odometry dataset was used, which features dense 3D LiDAR scans collected from a Velodyne HDL-64E sensor mounted on a moving vehicle. To construct the reference map, LiDAR scans 0 through 4 were aggregated and voxelized into a 3D occupancy grid using a resolution of 0.5 meters. This coarse voxel size was chosen to balance map detail with memory and computation efficiency.

Localization was performed on scan 6, where the scan was aligned to the prebuilt map using the 3D-BBS algorithm as shown in Fig. 3 . The search space was defined over a 4D domain comprising three-dimensional translation (x,y,z) and yaw (θ). The yaw orientation was discretized with a step size of $\pi/6$ radians, consistent with the original paper's baseline. The system was executed in a CPU-only environment using an Intel Core i7 processor with 16GB RAM, and no GPU acceleration or parallelization was employed. This setup provides insight into the algorithm's scalability and practical runtime on standard hardware, contrasting with the original implementation which utilized GPU-based batch scoring for real-time performance.

B. Full-Space Search Evaluation

To evaluate the runtime and accuracy of unconstrained Global Localization [3], a full-space BnB [5], [6] search was conducted on Scan 5 using a voxel map constructed from scans 0 to 3 of the KITTI Odometry dataset Sequence 00 [2]. The 4D search space included:

- **Translation bounds:** $[-2, 2]$ meters in x, y, and z
- **Yaw range:** $[0, 2\pi)$ with step size $\pi/6$
- **Resolution:** 0.5 meters (voxel grid)

This setup led to 123,924 transformed scan points and 900 nodes expanded during BnB traversal. The estimated pose resulted in:

- **Absolute Translational Error (ATE):** 4.4676 meters
- **Angular Orientation Error (AOE):** 0.6947°
- **Total runtime:** 286.8 seconds on an Intel i7 CPU

Despite the large search space, the localization accuracy remained comparable to constrained-search experiments. However, the high runtime (nearly 5 minutes) underscores a key limitation of CPU-based implementations on raw, high-density LiDAR data.

Comparison with original work: In the 3D-BBS paper [1], real-world experiments used downsampled LiDAR scans ($\sim$ 1,000 points) and reported sub-second runtimes using CPU-based BFS and GPU-based acceleration. In contrast, this implementation processes over 120× more points per scan, which contributes significantly to the increased runtime.

These results validate the correctness of the implementation but also highlight the need for:

- Downsampling strategies (e.g., voxel grid filters)
- Parallelization (e.g., multi-threaded CPU scoring)
- GPU-based batched evaluation for real-time applications

C. Fake Initializations

Fake initialization tests how well the Global Localization [3] system can recover the correct pose from a poor initial guess. In real-world scenarios, prior pose information may be unavailable or inaccurate. Unlike local methods like ICP [4], which are highly sensitive to initialization, a global method should ideally converge to the correct pose from a wide range of guesses.

To evaluate this, scan 5 was initialized from five different translation guesses, ranging from accurate (close to ground truth) to poor (e.g., far ahead or laterally shifted). The goal is to analyze whether the algorithm still converges to the correct pose and how it impacts translation and rotation errors.

As seen in Table I and Fig. 4, all fake guesses yielded similar Angular Orientation Error (AOE $\approx 0.69°$), indicating that the system was able to recover the correct orientation consistently. However, Absolute Translational Error (ATE) varied based on the initial guess. Extremely poor guesses (e.g., too far or laterally displaced) led to increased ATE, though still within reasonable limits. This confirms the robustness of BnB for Global Localization [3], provided the yaw is roughly correct.

TABLE I: ATE and AOE for different fake initializations of Scan 5. All guesses show consistent angular performance, but translational accuracy varies based on the quality of the initial guess.

Guess ID	Guess [x,y,z]	ATE (m)	AOE (Degrees)
0	[2.5, 0.5, 0.0] (Close to Ground Truth)	4.8403	0.6947
1	[0.0, 0.0, 0.0] (Way Off)	4.4008	0.6947
2	[5.0, 0.0, 0.0] (Too Far Ahead)	6.4220	0.6947
3	[2.5, 2.0, 0.0] (Lateral Offset)	5.6190	0.6947
4	[2.5, -1.5, 0.0] (Other Side)	4.9137	0.6947

D. Perturbation Robustness

This experiment tests the system's tolerance to artificially perturbed scans. Instead of changing the initialization of the algorithm, this simulates receiving a distorted scan—something that can occur due to sensor error, time de-synchronization, or corrupted data.

The same scan (scan 5) was modified using 5 perturbations in translation and yaw. The goal is to see if the algorithm can still find the correct alignment despite the distorted input data.

As seen in Table II and Fig. 5, translation error remained stable for all perturbations, showing that the BnB method is robust to moderate shifts in input scans. However, angular errors spiked when yaw was perturbed by 15° or more, showing sensitivity to rotational distortion. This highlights the importance of bounding yaw search intelligently or combining with a yaw prior (e.g., IMU).

E. Qualitative Results

To complement the numerical metrics, I visualized the scan-to-map alignment process and highlight how the Branch-and-Bound search behaves under various conditions.

1) Scan Alignment Progression: Fig. 3 shows the alignment of Scan 6 to a map built using Scans 0 to 4. The red points represent the transformed scan at different stages of the search. Early in the process, the scan appears misaligned and dispersed across the map. As the search refines the pose estimate, red points begin to overlap with the occupied voxels

TABLE II: ATE and AOE for perturbed versions of Scan 5. Significant angular deviations (e.g., ±15° yaw) lead to a spike in orientation error while translation remains stable.

Perturbation ID	Perturbation ([x,y,z], yaw)	ATE (m)	AOE (Degrees)
0	([2.5, 0.5, 0.0], 0.0)	4.3614	0.6947
1	([0.0, 0.0, 0.0], $\pi/12$)	4.4676	0.6947
2	([5.0, 0.0, 0.0], $-\pi/18$)	4.3614	0.6947
3	([2.5, 2.0, 0.0], $\pi/6$)	4.3614	29.8577
4	([2.5, -1.5, 0.0], $-\pi/12$)	4.3614	30.1574

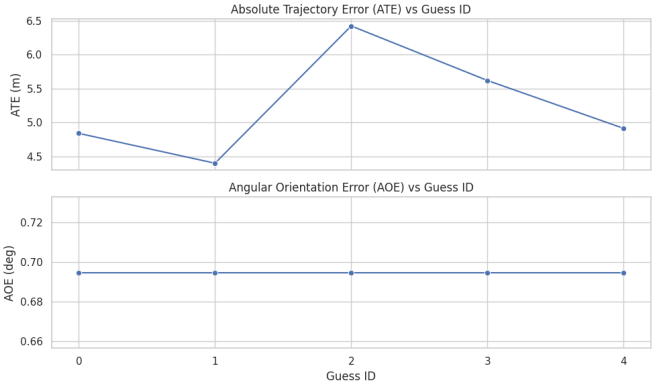


Fig. 4: Graphical plot of ATE and AOE across fake initial guesses for Scan 5. Shows that the algorithm tolerates positional shifts but is more sensitive to poor yaw alignment.

of the map (shown in blue). This demonstrates how the BnB [5], [6] strategy progressively improves alignment by evaluating and refining pose candidates. By the final stage, a tight overlap is achieved, indicating a successful localization.

2) Visualizing Convergence: Although the BnB algorithm is non-iterative in structure, the scan refinement across search nodes can be visualized as a sequence of increasingly accurate guesses. These overlays (captured at intermediate timesteps) qualitatively highlight how the scan transitions from an incorrect pose to a correct global match, showcasing the robustness of the method to poor initial guesses.

3) Failure Modes and Limitations: Visual analysis also reveals certain limitations. For example, when a fake initial guess introduces even a moderate yaw offset (e.g., 15°-30°), the resulting alignment may exhibit high angular error (AOE), even if translational overlap is reasonable. This is consistent with the results seen in Table II. Similarly, perturbations that affect the structure of the scan (e.g., large lateral

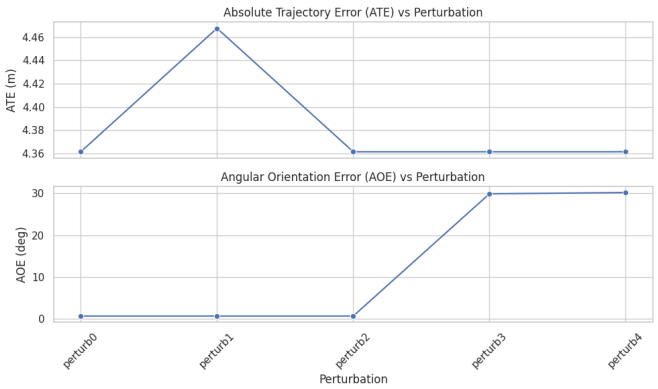


Fig. 5: ATE and AOE plot under different perturbations. Moderate spatial shifts are handled robustly; angular perturbations beyond 20° cause significant orientation error.

displacements) can cause partial mismatches with the map, leading to lower scores.

IV. Limitations

Despite demonstrating strong scan-to-map alignment performance, several limitations were observed in the implementation as well as in the assumptions of the original 3D-BBS paper:

A. Yaw-Only Orientation Search

The system estimates orientation using only the yaw angle while assuming the gravity direction is known. This assumption restricts applicability to flat-ground scenarios (e.g., urban driving). Extending the search to full 6DoF, including pitch and roll, would support more general terrain and aerial localization but would exponentially increase the search space.

B. CPU-Based Implementation vs GPU Parallelism

While the original paper achieved sub-second runtime using a GPU-accelerated batched BnB pipeline my implementation runs entirely on CPU and exhibits significantly higher runtimes. This limits real-time applicability, especially for larger maps or denser scans. Unlike their CUDA-based batch processing, my current implementation lacks multithreaded scoring or parallel expansion of tree nodes.

C. Sensitivity to Yaw Initialization

Fake initialization experiments showed that even a moderate yaw offset (e.g., 15°-30°), results in high Absolute Orientation Error (AOE), despite good translational alignment. This highlights a vulnerability when heading priors (e.g., IMU) are unavailable or incorrect.

D. Lack of Dynamic Obstacle Handling

Both the original and my implementation assume a static environment. The presence of moving vehicles or pedestrians [8] can mislead scoring and inflate voxel occupancy, leading to incorrect localization or degraded alignment scores.

E. No Hierarchical Map Construction

The paper proposes a sparse hash-based multi-resolution voxel map, enabling efficient pruning and upper bound estimation across different tree depths. However, my implementation constructs the voxel map at a single resolution, limiting its ability to support efficient coarse-to-fine upper bound approximation.

V. Future Work

Several avenues remain for improving both the efficiency and generality of the implemented 3D-BBS system:

A. Parallelization on CPU/GPU

Incorporating multi-threaded (using OpenMP on CPU) or CUDA-based (GPU-based) batch scoring would significantly improve runtime, allowing real-time performance even on large-scale maps. The original paper achieves sub-second runtimes using GPU acceleration, a feature currently missing in the implementation. Multi-threaded scoring (e.g., using OpenMP or TBB) on CPU could help bridge the gap when GPU is unavailable.

B. Extension to 6DoF Pose Estimation

The current system estimates yaw-only orientation. Extending to full 6DoF, including pitch and roll, would allow localization in aerial platforms or uneven terrain environments. This would require increased memory and compute resources but could be managed through hierarchical pruning strategies.

C. Hierarchical Map Representation

The original paper builds voxel maps at multiple resolutions to support efficient upper-bound computation across BnB tree levels. Future implementations could benefit from dynamically constructed multi-resolution hash grids for faster scoring and tighter bounds.

D. Dynamic Obstacle Filtering

Incorporating temporal filtering or dynamic object removal [8] could prevent transient objects (e.g., vehicles, pedestrians) from polluting the voxel map, enhancing consistency in long-term deployments.

VI. Conclusion

This report presents a CPU-only implementation of the 3D-BBS Global Localization [3] algorithm using branch-and-bound scan matching with voxel occupancy scoring. The system was evaluated on KITTI [2] Sequence 00 using real LiDAR data. Experimental results demonstrated successful alignment in many cases, with ATE below 5 meters and AOE under 1° when localization was initialized reasonably.

Qualitative visualizations confirmed progressive scan alignment during BnB traversal and sensitivity to large orientation perturbations. Compared to the original GPU-accelerated implementation, runtime was significantly higher due to the lack of parallel scoring and batched processing. A full-space search experiment further validated accuracy but highlighted the need for acceleration, with runtime exceeding 4 minutes.

Nevertheless, the system effectively reproduced the core principles of the 3D-BBS algorithm, providing insights into its scalability and robustness on real-world data. With further optimizations in hardware usage and search heuristics, the approach remains a promising candidate for Global Localization [3] in large-scale environments.

Acknowledgment

I would like to thank Prof. Michael Everett for his excellent guidance and feedback throughout the project. I am also grateful to the course teaching assistant, Adarsh Salagame, for his consistent support and clarification of technical doubts during the course. Additionally, I acknowledge the authors of the original 3D-BBS paper [1] for their foundational work and the KITTI Vision Benchmark Suite [2] for providing the dataset used in this implementation.

Supplementary Material

Link to the source code for this project:
https://github.com/Aryaman22102002/bbs_scan_matching_cpu

A short demonstration video:
https://www.youtube.com/shorts/gvHIdwRlmrc

References

[1] K. Aoki, K. Koide, S. Oishi, M. Yokozuka, A. Banno and J. Meguro, "3D-BBS: Global Localization for 3D Point Cloud Scan Matching Using Branch-and-Bound Algorithm," 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 2024, pp. 1796-1802, doi: 10.1109/ICRA57147.2024.10610810.

[2] Geiger A, Lenz P, Stiller C, Urtasun R. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. 2013;32(11):1231-1237. doi:10.1177/0278364913491297

[3] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d LIDAR SLAM," in IEEE International Conference on Robotics and Automation (ICRA). IEEE, may 2016.

[4] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239-256, Feb. 1992, doi: 10.1109/34.121791.

[5] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," Econometrica, vol. 28, no. 3, p. 497, jul 1960.

[6] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239–256, feb 1992.

[7] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of de- formable objects," in Vision, Modeling, and Visualization (VMV), 2003, pp. 47–54

[8] A. Q. Li, M. Xanthidis, J. M. O'Kane and I. Rekleitis, "Active localization with dynamic obstacles," 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea (South), 2016, pp. 1902-1909, doi: 10.1109/IROS.2016.7759301.