Real-Time 6D Object Pose Estimation Using FoundationPose with Open-Vocabulary Segmentation

Bishal Shrestha

University of Miami, Department of Computer Science
Coral Gables, FL 33146, USA
CSC752: Autonomous Robotic Systems
bishal.shrestha@miami.edu

Abstract. This paper presents a complete system for real-time 6D object pose estimation on a Human Support Robot (HSR) in simulation. The system integrates FoundationPose, a state-of-the-art neural implicit representation-based pose estimation method, with open-vocabulary segmentation using Grounded SAM (Grounding DINO + Segment Anything Model). We address critical challenges including segmentation quality, timestamp synchronization for moving robots, and temporal filtering for pose stability. Through systematic optimization, we achieve an 8-11x speedup from an initial 17 seconds to 1.5-2.1 seconds per pose estimate, with median position error of 5cm and median orientation error of 10 degrees. The system demonstrates robustness across various objects and viewing angles, making it suitable for real-time robotic manipulation tasks.

Keywords: 6D pose estimation · robotic manipulation · FoundationPose · open-vocabulary segmentation · ROS

1 Introduction

Accurate 6D object pose estimation is fundamental for robotic manipulation, requiring estimates that are fast enough for closed-loop control and robust to novel objects and various viewing angles. Movable service robots introduce unique challenges: continuous camera motion creates temporal synchronization problems where RGB, depth, and segmentation masks arrive asynchronously through ROS callbacks, yet must be processed with the camera pose at the exact moment of image capture. A service robot moving at 0.3 m/s with a 30ms timestamp mismatch introduces approximately 9mm of position error per frame, accumulating to 30cm over several seconds—unacceptable for precise manipulation.

Traditional keypoint-based methods like DOPE [1] detect sparse geometric features and solve for pose using Perspective-n-Point (PnP) algorithms [2], but are brittle to occlusion. In our experiments, DOPE required all 9 keypoints (8 cuboid vertices plus centroid) to be detected simultaneously for pose estimation. Most frames detected only 1-3 keypoints, resulting in only 5% of frames producing valid pose estimates—insufficient for real-time manipulation.

We present a system integrating FoundationPose [3], a dense RGB-D refinement method, with open-vocabulary segmentation using Grounded SAM [4]. FoundationPose uses neural implicit representations and processes all pixels in a segmentation mask, providing robustness to partial visibility and enabling pose estimation for novel objects without fine-tuning. The dense approach fundamentally changes error characteristics: individual pixel errors average out rather than causing catastrophic failure, and the neural implicit representation interpolates pose information even when significant portions of the object are occluded.

1.1 Contributions

Our main contributions are: (1) a timestamp-preserving TF lookup and buffering strategy eliminating 30cm height drift on human service robots; (2) systematic error propagation analysis identifying three distinct failure chains with quantitative error accumulation models; (3) temporal consensus filtering using SE(3) clustering and quaternion averaging on SO(3); (4) open-vocabulary segmentation pipeline enabling novel objects without retraining; and (5) systematic performance optimization achieving 8-20x speedup (17s→1.5-2.1s) through parallel processing, consensus filtering, and configuration tuning.

2 Related Work

2.1 6D Object Pose Estimation

6D pose estimation methods can be categorized into keypoint-based, template-based, and dense refinement approaches. Keypoint-based methods like DOPE [1] detect sparse geometric features and solve for pose using PnP algorithms, but are brittle to occlusion. PoseCNN [5] directly regresses 3D translation and rotation (quaternion) from CNN features, avoiding PnP but still requiring object-specific training. Our experiments with DOPE showed only 5% success rate due to incomplete keypoint detection.

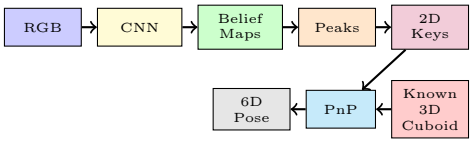


Fig. 1: DOPE architecture: CNN predicts belief maps for 9 keypoints (8 cuboid vertices plus centroid), peaks are extracted as 2D keypoints, then PnP solves for 6D pose using known 3D cuboid geometry. This sparse approach requires all keypoints to be visible, making it brittle to partial occlusion. In practice, most frames detected only 1-3 keypoints, resulting in 5% success rate.

Template-based methods match image features to 3D object models but struggle with textureless surfaces and require extensive template databases.

Dense refinement methods like FoundationPose [3] and DenseFusion [6] use all pixels in a segmentation mask, providing robustness to partial visibility. FoundationPose uses neural implicit representations and contrastive learning, achieving state-of-the-art results on the BOP benchmark [7] while supporting novel objects without fine-tuning. However, existing dense methods assume static camera setups and do not address the temporal synchronization challenges inherent in mobile manipulation, nor do they provide systematic analysis of error propagation through the segmentation-to-pose pipeline.

2.2 Segmentation for Pose Estimation

High-quality segmentation is critical for pose estimation accuracy, as mask leakage corrupts depth measurements and causes pose instability. Traditional methods use fixed-vocabulary detectors like YOLO [8] or Mask R-CNN [9], limiting them to pre-trained object classes. Recent advances enable open-vocabulary detection via text prompts: Grounding DINO [10] combines vision-language models with object detection, while SAM [11] provides high-quality segmentation. Grounded SAM [4] integrates both, enabling segmentation of any describable object.

Dense methods like FoundationPose fuse RGB-D modalities for improved accuracy, but depth measurements are sensitive to segmentation quality—leaky masks include table pixels with different depths, corrupting 3D reconstruction. Temporal filtering reduces pose uncertainty by averaging multiple estimates. We therefore design a system that addresses these gaps through timestamp-preserving synchronization, systematic error propagation analysis, and temporal consensus filtering using a circular buffer with SE(3) clustering and quaternion averaging.

3 System Overview

3.1 Architecture

Our system integrates three components: open-vocabulary segmentation (Grounded SAM), 6D pose estimation (FoundationPose), and temporal filtering with validation. The pipeline architecture (Fig. 2) shows data flow from IsaacSim through segmentation (∼680ms), timestamp synchronization, pose estimation (∼1.5s), and consensus filtering to RViz. The thread pool enables parallel processing, improving throughput without reducing per-frame latency.

The system operates in ROS1 Noetic with three isolated conda environments to avoid dependency conflicts (e.g., libffi version mismatches with cv_bridge).

3.2 Design Goals

Our system design balances robustness, speed, accuracy, flexibility, and reliability. Robustness requires handling novel objects, partial occlusions, and various

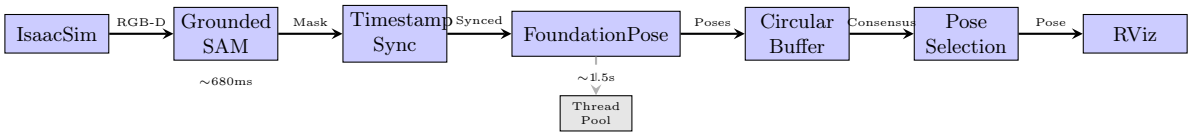


Fig. 2: System architecture showing the complete pipeline from RGB-D input to pose output. The simulation environment (IsaacSim) provides RGB-D data to Grounded SAM for segmentation (∼680ms), which feeds into FoundationPose for pose estimation (∼1.5s). Temporal filtering via circular buffer and consensus selection ensure stable output. The thread pool enables parallel processing for improved throughput.

viewing angles without object-specific training. Speed demands real-time performance with total latency under 2.5 seconds for responsive manipulation. Accuracy targets median position errors of 5cm, median orientation errors of 10°, and median height errors of 1.5cm for precise top-down grasping. Flexibility is achieved through open-vocabulary detection enabling new objects via natural language description. Reliability is ensured through multi-stage validation and temporal filtering that eliminate false positives and smooth pose trajectories.

4 Methods

4.1 Segmentation Options

Our segmentation approach evolved from manual input through fixed-vocabulary detection to open-vocabulary capabilities (Fig. 3). The initial YOLO+SAM approach used YOLOv8 for detection followed by SAM for mask refinement, achieving fast processing (∼370ms) but limited to COCO classes, requiring object mapping that led to detection failures.

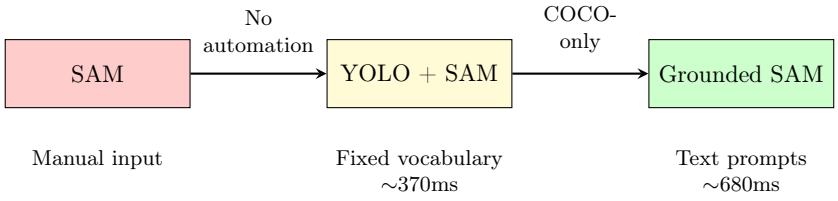


Fig. 3: Segmentation evolution from manual SAM input through YOLO+SAM (fixed vocabulary) to Grounded SAM (open vocabulary). The progression enables automatic detection of novel objects through natural language descriptions.

Grounded SAM addresses the vocabulary limitation by combining Grounding DINO (open-vocabulary detection) with SAM (segmentation). Given a natural

language text prompt, Grounding DINO detects objects by aligning text embeddings with visual features, outputting bounding boxes with confidence scores. Multi-stage validation applies confidence thresholds (box_threshold: 0.80, text_threshold: 0.80), phrase matching to reject false positives, and geometric validation. Non-Maximum Suppression (NMS) with IoU threshold 0.5 removes duplicate detections by eliminating overlapping boxes with lower confidence scores. SAM then generates precise binary masks from validated bounding boxes. The open-vocabulary capability enables detection of any describable object without retraining, at a computational cost: average processing time is ∼680ms (mean from 102 samples, range: 340-910ms), nearly double YOLO+SAM's ∼370ms, with CPU memory usage ∼5.3GB compared to ∼1.4GB. For novel objects not in standard datasets, this tradeoff is justified by eliminating dataset-specific training requirements.

4.2 Pose Estimation with FoundationPose

FoundationPose estimates 6D pose using dense RGB-D refinement: given a segmentation mask, the pipeline extracts masked RGB and depth pixels, performs neural implicit matching between observed data and the object mesh, refines pose using iterative optimization in registration mode, and outputs a 6D pose (rotation matrix plus translation vector) in the camera frame. We use registration mode for all poses (tracking not implemented), with parameters tuned for speed: `est_refine_iter`=1 (reduced from 3), `track_refine_iter`=1, and `debug`=0.

Average processing time is 1.53s (mean from 80 samples, range: 0.84-2.10s), with GPU memory usage ∼0.34GB. To improve throughput without reducing single-pose latency, we implement parallel processing using a thread pool with 8 workers (Fig. 4), improving throughput from ∼24 to 60-80 poses/min (2.5-3.3x) while maintaining single-pose latency, enabling backlog processing during high frame rates.

4.3 Timestamp Synchronization

Timestamp synchronization addresses the problem that camera pose changes continuously on moving robots, while ROS callbacks for RGB, depth, and mask messages fire asynchronously, arriving at slightly different times despite corresponding to the same scene capture. The TF tree stores transforms as functions of time, so using the wrong timestamp retrieves the camera pose at a different moment than image capture, causing pose drift. Even small temporal mismatches (20-30ms) can correspond to significant camera pose changes when the robot is moving.

Our solution preserves image timestamps end-to-end: we store `header.stamp` from the RGB image (actual capture time), pass it through the segmentation node so the mask inherits the same timestamp, buffer messages in timestamp-keyed dictionaries, and match RGB, depth, camera_info, and mask messages by exact timestamp. All TF lookups use the matched timestamp to retrieve the

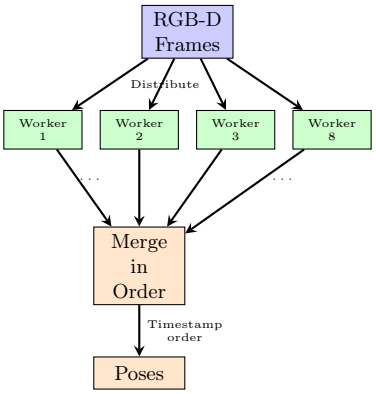


Fig. 4: Parallel pose estimation architecture using thread pool. Incoming RGB-D frames are distributed to 8 worker threads, enabling concurrent processing. Completed poses exit the pool and are merged in timestamp order, ensuring downstream consumers see poses in temporal order even though computations complete out of order. This architecture improves throughput from ∼24 to 60-80 poses/min without reducing single-pose latency.

camera pose at the exact moment of image capture, not at processing time. This eliminated the 30cm height drift observed without proper synchronization, reducing height error from ∼30cm to <2cm. The timestamp mismatch mechanism (Fig. 5) illustrates how using wrong timestamps (t_{now} instead of t_1) causes pose errors. Figure 6 shows the visual effect of this drift when the robot is moving.

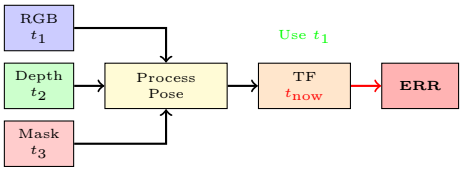


Fig. 5: Timestamp mismatch mechanism: RGB, depth, and mask arrive at different times (t_1, t_2, t_3). Using wrong timestamp (t_{now}) for TF lookup retrieves camera pose at different moment than image capture, causing pose errors. Solution: use RGB image timestamp (t_1) for all TF lookups.

4.4 Consensus Filtering

Consensus filtering addresses the problem that individual pose estimates exhibit inherent noise from segmentation boundary variations, depth uncertainty, and optimization convergence, occasionally producing outliers. We implement tem-

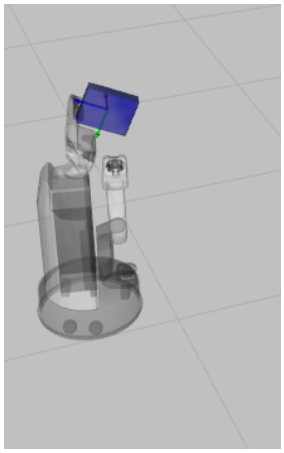


Fig. 6: Visual effect of pose drift without timestamp synchronization. When the robot moves and wrong timestamps are used for TF lookups, pose estimates gradually drift away from the true position over time, as shown by the misaligned coordinate frame markers.

poral filtering using a circular buffer maintaining the last 10 poses, providing a sliding window for consensus computation (Fig. 7).

The clustering stage groups similar poses using union-find: two poses are similar if position difference is within 0.1m, orientation difference is within the configured threshold (default 15°, configurable up to 30°), and timestamps differ by at most 0.5s. The temporal constraint ensures only poses from similar robot configurations are clustered, preventing averaging across large movements. The consensus requirement mandates the largest cluster contains at least 50% of the poses in the buffer.

Averaging poses in SE(3) requires careful treatment: positions are averaged using component-wise arithmetic mean in $\mathbb{R}^3$, while rotations are averaged on SO(3) using quaternion averaging—we convert rotation matrices to unit quaternions, compute their mean, then normalize. This geometric averaging preserves SE(3) group structure and produces more stable consensus poses than naive Euclidean averaging. If no cluster achieves the 50% threshold, the oldest pose is dropped and the new pose is added to the buffer.

4.5 Validation and Coordinate Frame Correction

Multiple validation checks prevent false positives: mask size (minimum 500 pixels or 0.5% of image area), depth range validation (0.3-2.0m), position bounds checking, and pose quality thresholds (position error <2.0m, orientation error <200°).

FoundationPose uses OpenCV coordinate convention, while ROS uses different coordinate convention. We apply conditional rotation, aligning with ROS

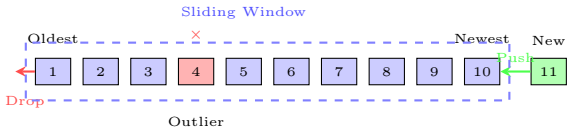
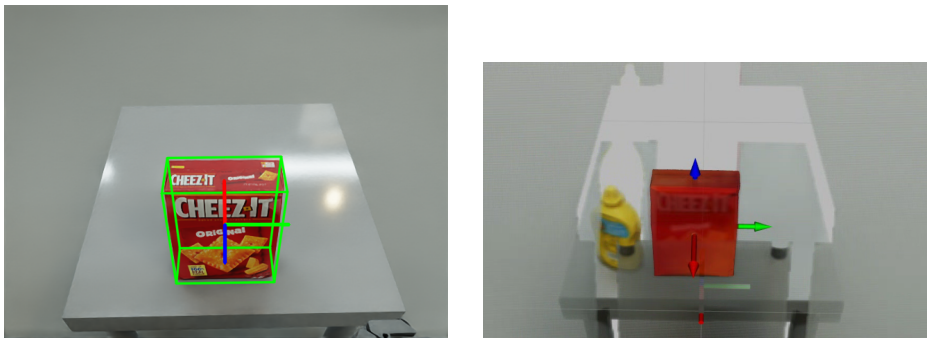


Fig. 7: Circular buffer mechanism for consensus filtering. Poses 1-10 represent the sliding window, with pose 4 marked as an outlier. When new pose 11 arrives, it pushes into the buffer and the oldest pose 1 is dropped. The dashed box indicates the temporal window used for clustering and consensus computation.

convention for RViz visualization while preserving position accuracy. Figure 8 illustrates the coordinate frame mismatch and correction.



(a) OpenCV convention (b) ROS convention

Fig. 8: Coordinate frame conventions: OpenCV vs ROS. A conditional rotation correction converts from OpenCV to ROS convention, ensuring correct visualization in RViz while preserving position accuracy.

5 Implementation Details

The system runs on ROS1 Noetic with three isolated conda environments (`grounded_sam`, `foundationpose`, and system ROS) to avoid dependency conflicts. Conda's `libffi.so.7` conflicts with system ROS's `cv_bridge` (linked against `libffi.so.6`); we force conda's libffi to load first using `LD_PRELOAD` in wrapper scripts and launch files. Launch files organize node startup, with all parameters loaded from `config/foundationpose_config.yaml`, eliminating the need for launch arguments. Table 1 shows key configuration parameters and their values.

Table 1: Key system parameters from configuration file.

Parameter	Value	Description
FoundationPose		
`est_refine_iter`	1	Registration refinement iterations
`track_refine_iter`	1	Tracking refinement iterations (not used)
`debug`	0	Debug level (0=off, 1=basic, 2=detailed)
`pose_buffer_size`	10	Circular buffer size for consensus
`max_parallel_workers`	8	Thread pool size for parallel processing
Grounded SAM		
`sam_model_type`	`vit_b`	SAM model variant (vit_b=fastest)
`box_threshold`	0.80	Grounding DINO box confidence threshold
`text_threshold`	0.80	Grounding DINO text matching threshold
`iou_threshold`	0.5	NMS IoU threshold
Validation		
`publish_position_error_threshold`	2.0m	Maximum position error to publish
`publish_orientation_error_threshold`	200°	Maximum orientation error to publish

6 Experiments and Evaluation

6.1 Experimental Setup

The simulation environment uses IsaacSim with an HSR robot, RGB-D camera on the robot head, and YCB dataset objects (cracker_box, mustard_bottle) on a table. IsaacSim provides ground truth object poses through its physics engine, published via ROS topic `/{object_name}/ground_truth_pose` as `PoseStamped` messages. The pose estimation node subscribes to this topic and transforms ground truth poses to the camera frame using TF lookups at the image capture timestamp, enabling direct comparison. Error metrics computed include: position error (Euclidean distance), orientation error (geodesic distance on SO(3) from rotation matrix difference), and height error (Z-axis component difference). Metrics are automatically logged to JSON files and analyzed using `metrics/plot_metrics.py`.

6.2 Quantitative Results

Performance metrics across implementations are summarized in Table 2. The optimized system achieves median position error of 5cm (range: 5-15cm depending on viewing angle), median orientation error of 10° (range: 10-30°), and median height error of 1.5cm (range: 0.5-2cm), representing 10-15x improvement in position accuracy and 15-25x improvement in height accuracy compared to baseline configurations without proper segmentation or timestamp synchronization. These accuracy levels are suitable for precise manipulation tasks.

Table 2: Performance comparison across implementations.

Metric	FoundationPose	Grounded SAM	YOLO+SAM
Processing Time (ms)	1533 ± 155	681 ± 163	369 ± 77
GPU Memory (GB)	0.34 ± 0.06	1.00 (constant)	0.40 (constant)
CPU Memory (MB)	2145 ± 7	5320 (constant)	1380 (constant)
Throughput (poses/min)	∼24 (sequential) ∼60-80 (parallel)	N/A	N/A

6.3 Qualitative Results

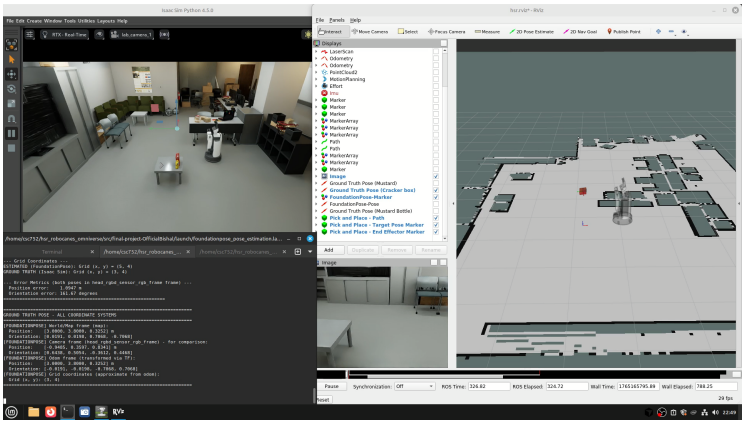
Figure 9 shows the visual evolution of the system across three stages. Without segmentation masks, FoundationPose uses depth-based heuristics that include table pixels, causing severe Z-axis drift with pose estimates jumping erratically (height error ∼30cm). With Grounded SAM segmentation but without timestamp synchronization, masks are clean but pose drifts over time due to incorrect TF lookups (height error ∼30cm). The final optimized system with clean segmentation and exact timestamp synchronization produces stable pose trajectories suitable for manipulation (height error <2cm).

6.4 Ablation Studies

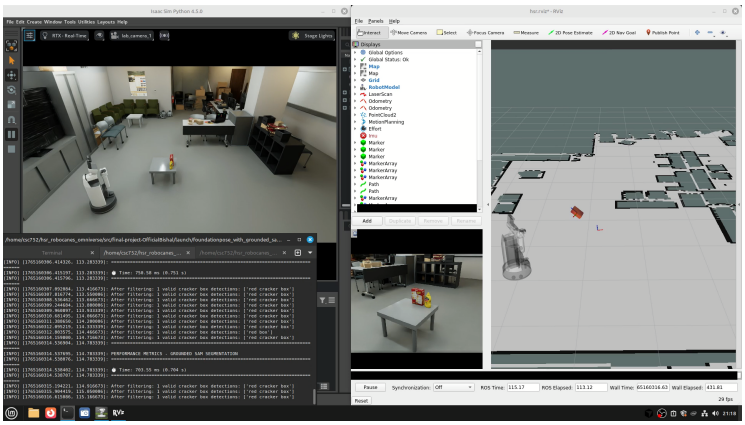
Three ablation studies compare system variants (visualized in Fig. 9): (1) Without segmentation mask (depth-based heuristic, depth 0.3-2.0m): severe Z-axis drift ∼30cm, position error ∼74cm, orientation error ∼154° due to mask leakage corrupting depth measurements. (2) With Grounded SAM but timestamp issues (using `rospy.Time.now()` instead of image timestamps): height drift ∼30cm persists despite clean masks. (3) Final optimized system (clean segmentation + exact timestamp sync + consensus filtering): stable pose trajectory with median height error 1.5cm, median position error 5cm, median orientation error 10°, suitable for manipulation.

6.5 Performance Distribution

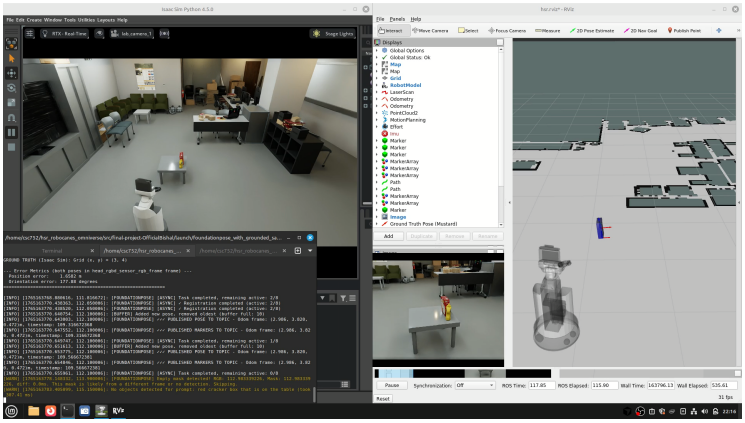
Processing time distributions (Fig. 10) show FoundationPose has higher variance (0.84-2.10s) due to varying scene complexity and is the bottleneck, while segmentation times are acceptable for real-time operation. Memory usage distributions (Figs. 11 and 12) show FoundationPose uses ∼2.1GB CPU memory consistently, Grounded SAM uses ∼5.3GB CPU memory, and GPU memory usage is reasonable (<1GB allocated), enabling deployment on mid-range GPUs.



(a) Without mask



(b) Grounded SAM (no sync)



(c) Grounded SAM (optimized)

Fig. 9: System evolution showing pose stability progression. Top: without segmentation mask, pose jumps erratically due to depth corruption. Middle: with Grounded SAM but timestamp mismatch causes gradual drift. Bottom: optimized system with clean segmentation and timestamp synchronization produces stable trajectories.

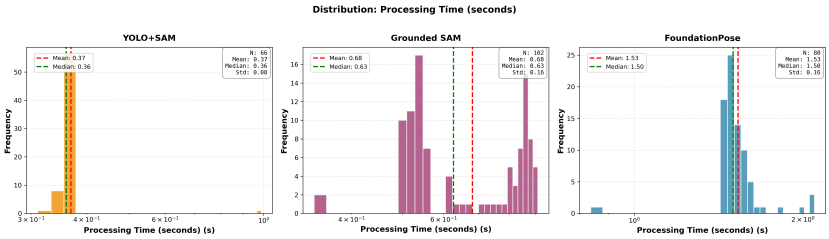


Fig. 10: Distribution of processing times. FoundationPose: higher variance (0.84-2.10s); Grounded SAM: moderate variance (0.34-0.91s); YOLO+SAM: most consistent (0.31-0.99s, clustered around 0.36s).

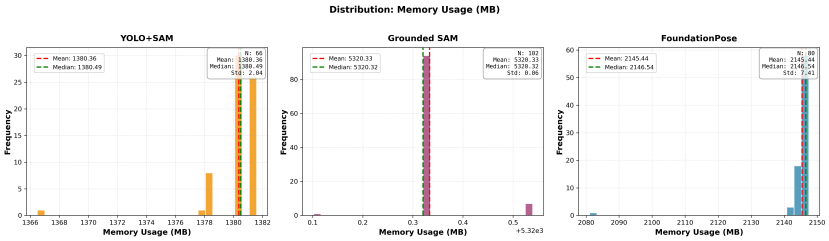


Fig. 11: Distribution of CPU memory usage. FoundationPose: ∼2.1GB; Grounded SAM: ∼5.3GB; YOLO+SAM: ∼1.4GB.

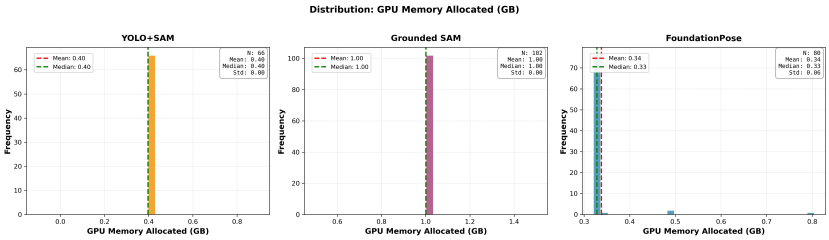


Fig. 12: Distribution of GPU memory allocation. FoundationPose uses ∼0.34GB (range: 0.32-0.80GB), Grounded SAM uses ∼1.00GB consistently, YOLO+SAM uses ∼0.40GB consistently.

7 Discussion

We identified three primary failure modes that propagate through the system (Fig. 13), each representing a distinct error propagation pathway that must be interrupted at the appropriate stage.

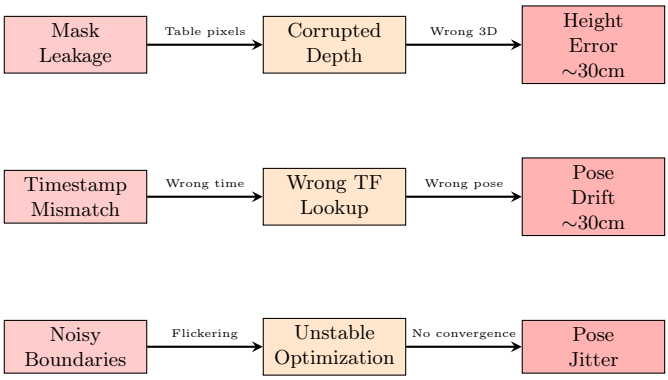


Fig. 13: Three primary failure modes and their error propagation chains. Mask leakage corrupts depth measurements, leading to height errors. Timestamp mismatches cause incorrect TF lookups, resulting in pose drift. Noisy boundaries prevent stable optimization convergence, causing pose jitter.

Mask leakage occurs when segmentation masks include table pixels alongside object pixels. Background pixels have depth values corresponding to the table surface (∼0.6m) rather than the object (∼0.8m). FoundationPose uses these masked depth pixels to reconstruct 3D points for pose optimization, so corrupted depth values produce incorrect 3D point clouds, leading to systematic pose errors. The Z-axis (height) is particularly sensitive because table and object differ primarily in vertical position, causing height errors of 30cm or more when mask leakage is severe.

Timestamp mismatches arise from asynchronous ROS message callbacks combined with continuous robot motion. Using the wrong timestamp for TF lookups retrieves the camera pose at a different moment than image capture, introducing systematic error that accumulates over time. A mobile base moving at 0.3 m/s with a 30ms timestamp mismatch creates approximately 9mm of position error per frame, accumulating to 30cm over several seconds.

Noisy boundaries occur when mask edges flicker between frames due to segmentation uncertainty or depth noise, causing FoundationPose's iterative optimization objective function to change between iterations, preventing stable convergence and resulting in pose jitter.

Our validation and filtering stages interrupt error propagation at each stage: mask validation prevents corrupted depth from entering pose estimation, times-

tamp synchronization prevents temporal drift, and consensus filtering smooths individual pose noise before publication.

7.1 Tradeoffs

Our design decisions involve explicit trade-offs: speed versus accuracy (reducing refinement iterations from 3 to 1 improves processing time by 2.3x with accuracy loss of 0.5cm position error and 2° orientation error—acceptable for real-time operation); flexibility versus speed (Grounded SAM requires ∼680ms compared to YOLO+SAM's ∼370ms, an 84% increase in processing time, but enables novel objects without retraining—a worthwhile tradeoff for open-vocabulary capability).

7.2 Limitations

Several limitations constrain applicability: tracking mode is not implemented (all poses use slower registration mode 1.5-2.1s rather than faster tracking mode ∼100-200ms); the system handles only a single object at a time; evaluation has been conducted only in IsaacSim simulation, which may not capture all real-world challenges such as lighting variations and sensor noise; and pick-and-place integration was attempted but not successfully completed.

8 Conclusion and Future Work

We presented a complete system for real-time 6D object pose estimation integrating FoundationPose with open-vocabulary segmentation. Through systematic optimization, we achieved 8-20x speedup (17s→1.5-2.1s) with median accuracy suitable for manipulation (position 5cm, orientation 10°, height 1.5cm). Key technical contributions include exact timestamp synchronization eliminating 30cm drift, consensus filtering for temporal stability, open-vocabulary segmentation enabling novel objects, and comprehensive validation preventing false positives.

Future work includes implementing tracking mode for faster pose updates (∼100-200ms) after stable initialization, extending to multi-object scenarios with parallel pipelines and occlusion handling, real-world validation on physical HSR robots, completing pick-and-place integration with MoveIt, investigating learning-based temporal filtering, and extending error propagation analysis to model failure mode interactions.

References

1. Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., Birchfield, S.: Deep object pose estimation for semantic robotic grasping of household objects. arXiv preprint arXiv:1809.10790 (2018)

2. Lu, X.X.: A review of solutions for perspective-n-point problem in camera pose estimation. In: Journal of Physics: Conference Series. vol. 1087, p. 052009. IOP Publishing (2018)
3. Wen, B., Yang, W., Kautz, J., Birchfield, S.: Foundationpose: Unified 6d pose estimation and tracking of novel objects. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 17868–17879 (2024)
4. Sun, P., Chen, S., Luo, P.: Grounded segment anything: From objects to parts. `https://github.com/Cheems-Seminar/grounded-segment-any-parts` (2023)
5. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. arXiv preprint arXiv:1711.00199 (2017)
6. Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., Savarese, S.: Densefusion: 6d object pose estimation by iterative dense fusion. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 3343–3352 (2019)
7. Hodan, T., Michel, F., Brachmann, E., Kehl, W., GlentBuch, A., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., et al.: Bop: Benchmark for 6d object pose estimation. In: Proceedings of the European conference on computer vision (ECCV). pp. 19–34 (2018)
8. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
9. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 2961–2969 (2017)
10. Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Jiang, Q., Li, C., Yang, J., Su, H., et al.: Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In: European conference on computer vision. pp. 38–55. Springer (2024)
11. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W.Y., et al.: Segment anything. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 4015–4026 (2023)