Real-Time Object Pose Estimation on HSR in IsaacSim

Bishal Shrestha

Fall 2025

CSC752 Final Project

Contents

Problem & Setup

- **Platform**: HSR robot + IsaacSim
- **ROS1** inside Docker
- **Goal**: Real-time 6D pose for grasping
- **Input**: RGB-D from robot head, object meshes

Challenge

Accurate, robust, fast pose estimation for manipulation

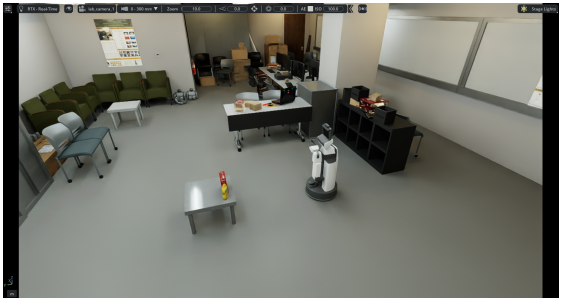


Figure 1: Simulation environment

6D Pose Estimation Geometry

Coordinate Frames

- **Object frame** (O_o, X_o, Y_o, Z_o): Object-centered
- **Camera frame** (O_c, X_c, Y_c, Z_c): Camera-centered
- **6D pose**: $[R|t]$ transforms object → camera frame

Pinhole Projection

- 3D point P_o in object frame
- **Step 1 - Transform**: $P_c = RP_o + t$ (object frame → camera frame)
- **Step 2 - Project**: $p = KP_c/Z_c$ (3D camera frame → 2D image pixels)
- $K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$: camera intrinsics matrix

RGB-D Advantage

- Depth Z_c directly measured
- Resolves scale ambiguity

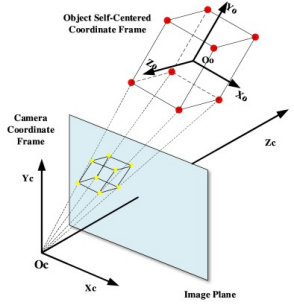


Figure 2: Real-time 6D pose estimation geometry from RGB-D image

Mask Leakage (Segmentation Errors)

Mask Leakage (Segmentation Errors)

- Bad mask → corrupted depth
- Depth noise → Z-axis instability
- Height error: 30cm drift

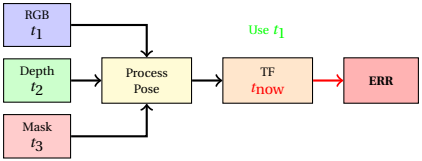
Key Insight

Segmentation quality → depth quality → pose stability

(a) RGB image (b) Object mask

Figure 3: Ideal scenario of RGB image and segmentation mask

Asynchronous Data (Timestamp Mismatches)



Asynchronous Data (Timestamp mismatches)

- RGB, depth, mask arrive separately
- Wrong TF lookup → incorrect $[R|t]$
- Result: 3D pose jumps, temporal drift

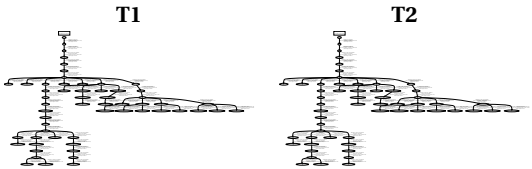
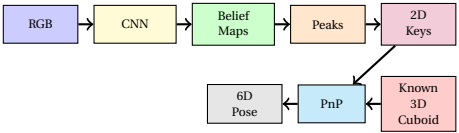


Figure 4: Timestamp mismatch causing pose drift

DOPE (Attempted, Failed)

DOPE Architecture



- CNN predicts belief maps for 9 keypoints
- **PnP** (Perspective-n-Point) solves for 6D pose
- Designed to use all 9 keypoints for robustness

Observed Results

- Partial keypoint detection: most frames had only 1-3 out of 9 keypoints
- Insufficient keypoints: needs at least 4 for PnP, designed for all 9
- **No poses published**: DOPE never detected all 9 keypoints simultaneously

Example DOPE Output (Partial Detection)

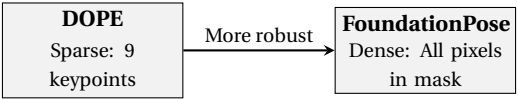
```
result from detection:
[None, None, None, None,
 None, None, None, None,
   (178.63, 194.47)]
```

× 8 corners: **NOT DETECTED**

✓ 1 center: **DETECTED** at (178, 194)

Result: NO POSE PUBLISHED
(Insufficient keypoints for PnP)

Motivation: FoundationPose



Why FoundationPose?

- **Dense approach**: Uses all pixels, not just 9 keypoints
- **RGB-D fusion**: Combines color and depth for accuracy
- **Novel objects**: Works without retraining
- **Robust to occlusion**: Handles partial visibility

State-of-the-Art Performance

BOP Challenge Leaderboard (2024): Outperformed specialized methods on multiple datasets. CVPR 2024 Highlight paper.

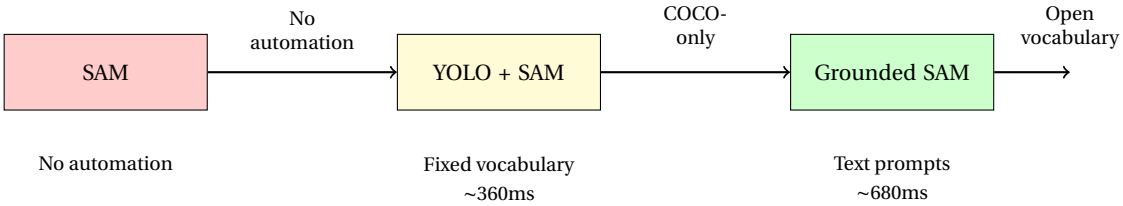
Performance Optimization

- **Initial**: 17s per pose (unacceptable)
- **Parallel processing**: 8 workers for throughput
- **Consensus filtering**: Circular buffer averaging
- **Timestamp sync**: Exact TF lookups
- **Final**: 1.5-2.1s per pose

Key Insight

Dense pixel-level processing provides more information and better robustness than sparse keypoint methods, especially under occlusion and novel objects.

Segmentation Evolution



- **SAM**: Requires manual input (points/boxes)
- **Limited Object Detection + Segmentation**: COCO-only, fixed vocabulary

- **Open-set Object Detector + Segmentation**: Text-conditioned detection
- Processing: 680ms (detection + segmentation)

Object Detection and Segmentation

Two-Stage Pipeline

- **Detection (Grounding DINO)**: Open-set object detector. Generates text-to-box proposals.
- **Segmentation (SAM)**: High-quality mask refinement
- Open-vocabulary: any describable object
- Processing: 680ms

Why Open-Vocabulary

- Text prompts: "bottle", "cup", etc.
- No fixed vocabulary limitation
- Enables novel objects via description

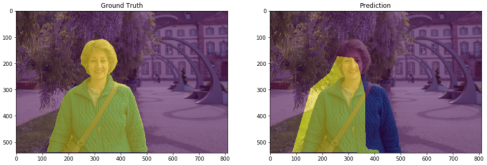


Figure 5: Segmentation quality

Final System Architecture

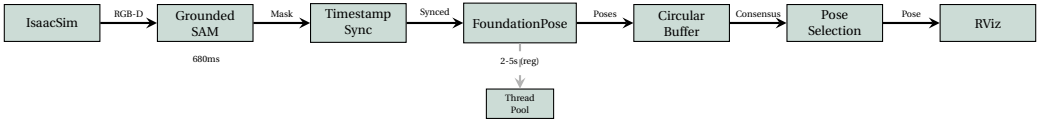


Figure 6: Architecture of Real-Time Object Pose Estimation on HSR in IsaacSim

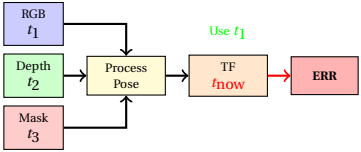
Components

- Grounded SAM: Detection + segmentation
- FoundationPose: 6D pose estimation
- Thread pool: 8 parallel workers
- Circular buffer: Consensus voting

Features

- Exact timestamp matching
- Parallel processing
- Pose consensus filtering
- Quality validation

Timestamp Synchronization



The Problem

- Robot motion during capture
- Asynchronous callbacks (RGB, depth, mask)
- Wrong TF lookup → incorrect $[R|t]$
- Even small time mismatches cause drift

The Solution

- Preserve image timestamp end-to-end
- Use robot pose at exact capture time
- Match RGB-D-mask by timestamp

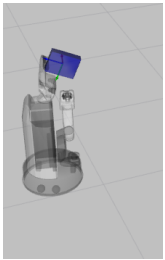


Figure 7: Visual effect of pose drift without timestamp sync

Circular Buffer Consensus Filtering

Circular Buffer Mechanism

- Why circular buffer: Filters outliers, reduces jitter, critical for stability during robot movement
- Configurable buffer size (default: 10 poses)
- Push new pose, drop oldest when full
- Perform clustering with thresholds:
  - Position difference ≤ 0.1m (10cm)
  - Orientation difference ≤ 15°
- Majority consensus requirement (50%+)
- If no consensus, latest pose from buffer is used as fallback

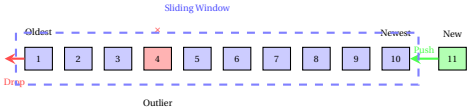


Figure 8: Circular buffer over recent poses

Parallel Processing Strategy

Parallel Architecture

- 8 workers process multiple frames simultaneously
- Improves throughput (poses/min), not single-pose latency
- Results merged in timestamp order (even if workers finish out of order)
- Worker count tuned to avoid GPU memory overflow

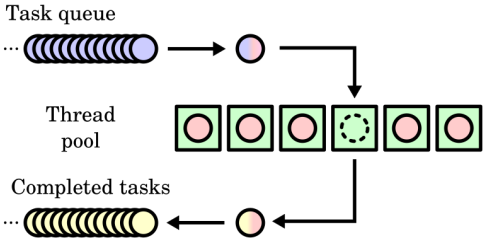


Figure 9: Parallel pose estimation with thread pool

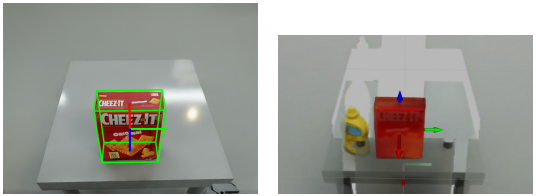
Validation and Coordinate Frame Correction

Multi-Level Validation

- Mask size: minimum 500 pixels (0.5% of image)
- Depth validation: object between 0.1m and 5.0m from camera
- Position validation: within 2m of camera center (X/Y)
- Temporal consistency: no jumps > 0.5m from last pose

Coordinate Frame Correction

- Convention mismatch: OpenCV vs ROS
- Conditional rotation applied to align with ROS convention
- Only affects orientation, position unchanged



(a) OpenCV convention (b) ROS convention

Figure 10: Coordinate frame correction

Quantitative Results

Metric	Before	After
Time per pose	17s	1.5-2.1s
Throughput (sequential)	12 poses/min	24 poses/min
Throughput (parallel)	N/A	**60-80 poses/min**
Position error	50-75cm	**< 5cm**
Orientation error	100-150°	**< 10°**
Height error	30-50cm	**< 2cm**
Segmentation (Grounded SAM)	680ms	680ms
Segmentation (YOLO+SAM)	370ms	370ms

Table 1: Performance metrics comparison: Before (initial implementation) vs After (optimized system)

Performance Distribution: Processing Time

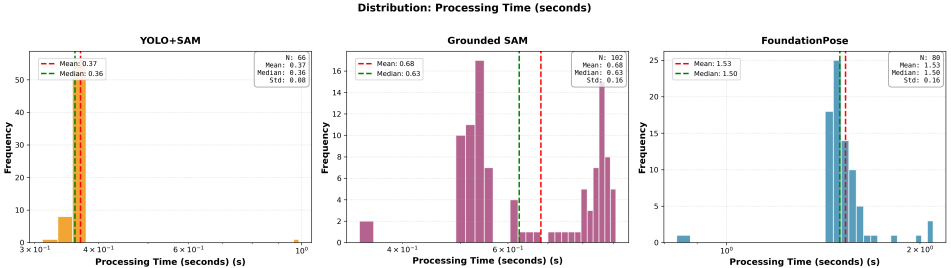


Figure 11: Distribution of Processing Time (seconds) across all implementations

Performance Distribution: Memory Usage

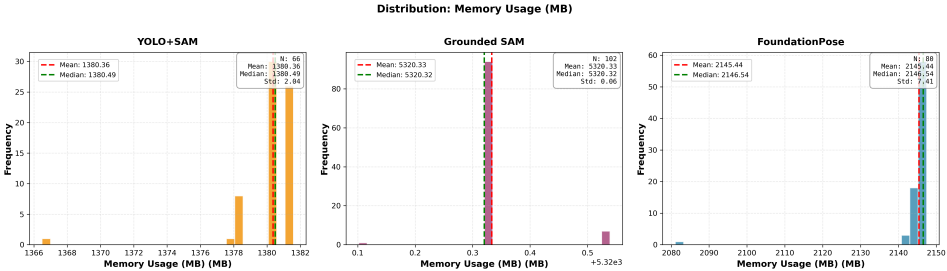


Figure 12: Distribution of Memory Usage (MB) across all implementations

Performance Distribution: GPU Memory

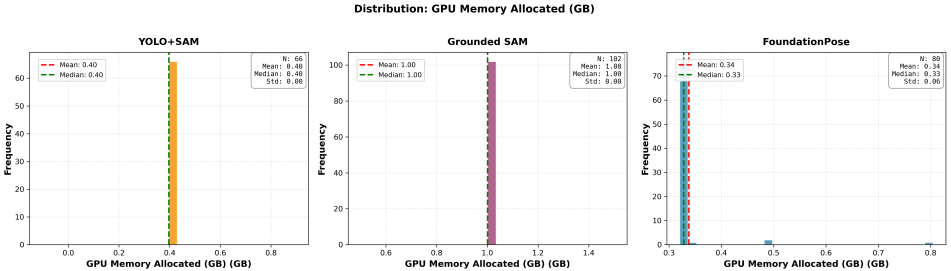
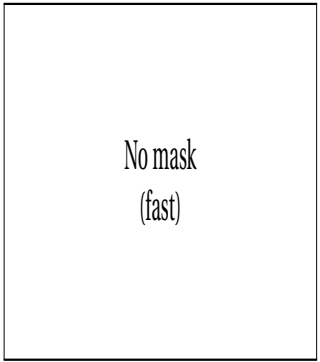


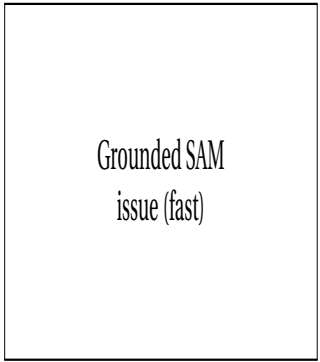
Figure 13: Distribution of GPU Memory Allocated (GB) across all implementations

System Evolution: Visual Comparison

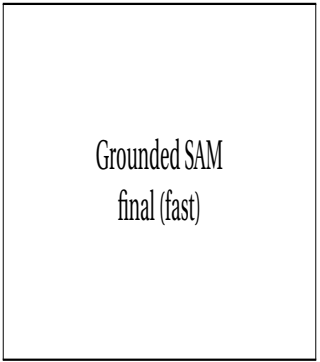
No SAM



Grounded SAM



Grounded SAM + Optimized



Video playback requires Adobe Acrobat. All three clips auto-play when this slide opens.

Lessons Learned

Technical Insights

- Validate early, fail fast
- Preserve timestamps end-to-end
- Design for parallelism
- Monitor performance continuously

Scientific Lessons

- Segmentation quality → pose stability
- Timestamp sync essential for robotics
- Temporal filtering reduces uncertainty

Methodological Insights

- Incremental improvement works
- Measure before optimizing
- Trade-offs are inevitable
- Real-world testing is essential
- Robustness over speed

Key Takeaway

10-30x speedup achieved through systematic, incremental optimizations

Conclusion

Final System Characteristics

- **Robust**: Handles novel objects, various viewing angles
- **Fast**: Real-time capable (2.2s per frame)
- **Reliable**: Stricter validation eliminates false positives
- **Flexible**: Open-vocabulary detection supports any describable object
- **Observable**: Comprehensive performance monitoring

Pose estimation system ready for robotic manipulation!