# Supplementary Material for Multi-SpatialMLLM: Multi-Frame Spatial Understanding with Multi-Modal Large Language Models

Runsen Xu[1,2] Weiyao Wang[1] Hao Tang[1] Xingyu Chen[1] Xiaodong Wang[1]
Fu-Jen Chu[1] Matt Feiszli[1] Kevin J. Liang[1]
[1]FAIR, Meta [2]The Chinese University of Hong Kong

## Appendix



## A. MultiSPA Data Samples and Distributions

Our MultiSPA dataset has 26 subtasks in total. Each task with an example is shown from Fig. 2 to Fig. 10. We show the distribution of samples across different task types in Tab. 1. Note that our data engine is scalable, and we can generate more training samples for these tasks.

## B. MultiSPA Data Templates

Due to paper length limits, we only show part of the templates in Listing 4. Other templates are similar to those shown in this supplementary material.

## C. Details of Source Datasets

**ScanNet.** ScanNet [2] is an RGB-D dataset containing more than 1,500 indoor scans. Each scan provides reconstructed point clouds, 3D camera poses, camera intrinsics, depth maps, and 3D instance and semantic segmentation masks. Our data generation pipeline utilizes all these annotations, though segmentation masks are optional if object perception data is not required.

**PStudio.** The CMU Panoptic Studio dataset [5] comprises 65 sequences (5.5 hours total) of multiple people interacting with one another or with objects, captured within a light stage. It offers multi-view images, 3D body skeletons, and facial landmark annotations.

**ADT.** Aria Digital Twin [7] is an egocentric video dataset recorded with Aria glasses. It contains 200 sequences of real-world indoor activities, each with precise 6DoF camera poses, 3D human poses, 2D image segmentations, and depth maps, as well as a digital twin environment.

**TAPVid3D.** TAPVid3D [6] is a dataset for tracking 3D points in space. It provides temporal 3D point tracking, constructed from PStudio, ADT, and DriveTrack [1]. It leverages official annotations to produce temporally aligned 3D point sequences, along with camera pose sequences and intrinsics. We use these annotations for our data generation. Note that we exclude the DriveTrack split because its camera poses are insufficiently accurate.

## D. Image Pairs Sampling

To ensure a balanced selection of image pairs based on their overlap ratio, we adopt the procedure as follows. First, we separate pairs with zero overlap and randomly sample a predefined number of them. Next, we partition all nonzero-overlap pairs into bins according to their overlap ratio. We then distribute the sampling quota across bins in proportion to the number of bins, sorting them by bin size in ascending order to prevent smaller bins from being overshadowed by larger ones. Finally, we either sample or exhaust each bin, carrying over any unused quota to subsequent bins. This step balances pairs of different overlap levels, mitigating is-

Table 1. **Distribution of samples across different task types.**

| Depth Estimation | Visual Correspondence | Camera Orientation | Camera Translation | Object Movement | Object Perception |
|---|---|---|---|---|---|
| 1.6M | 1.5M | 4M | 9M | 13M | 1.5M |

Table 2. **Results of different fine-tuning strategies for InternVL2-8B.** Increasing the number of trainable parameters yields larger gains from MultiSPA. Tuning the vision encoder significantly improves performance on fine-grained spatial understanding tasks, such as predicting camera motion vectors.

| | LLM-LoRA | LLM-LoRA+MLP | LLM-LoRA+MLP+VE | LLM-Full+MLP+VE |
|---|---|---|---|---|
| Average | 56.11 | 57.46 | 65.41 | **66.72** |
| *Depth Estimation* | | | | |
| Comparison | 74.00 | 73.34 | 81.00 | 80.84 |
| Value | 75.33 | 75.84 | 78.50 | **79.67** |
| *Visual Correspondence* | | | | |
| Coordinate | 49.00 | 52.67 | 63.67 | **71.33** |
| MCQ | 90.00 | 90.33 | 94.67 | **95.67** |
| *Camera Orientation* | | | | |
| Direction | 90.83 | 91.50 | 94.00 | **94.17** |
| Degree | 45.50 | 46.84 | 59.67 | **60.50** |
| *Camera Translation* | | | | |
| Direction | 85.89 | 87.66 | 91.78 | **92.56** |
| Distance | 42.33 | 46.33 | 72.00 | **77.67** |
| Vector | 18.00 | 24.67 | 43.00 | **53.00** |
| *Object Movement* | | | | |
| Distance | 40.42 | 38.17 | 39.08 | **47.83** |
| Vector | 12.92 | 15.84 | 19.50 | **26.75** |
| *Object Perception* | | | | |
| Size | 49.11 | 46.33 | 48.00 | 47.22 |

sues caused by long-tail distributions. The main content of the full algorithm is shown in Listing 1.

## E. Rotation Angles

Beyond translation, we estimate two orientation angles: yaw and pitch. We do not model roll, as it typically remains small in real-world use cases (*e.g.*, autonomous vehicles, robotics, wearable devices). Formally, let $\mathbf{E} \in \mathbb{R}^{4\times4}$ be the camera pose in world coordinates, which has the $z-$axis aligned with the gravity direction, and $\mathbf{R}$ its upper-left $3\times3$ submatrix:

$$\mathbf{R} = \mathbf{E}[0{:}3, 0{:}3]. \tag{1}$$

We then extract yaw and pitch by focusing on the camera's forward (i.e., $z$-) axis:

$$\mathbf{z}_{\text{fwd}} = \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2}$$

**Yaw** is defined as the angle of this rotated $z$-axis in the horizontal plane, measured around the gravity axis:

$$\text{yaw} = \arctan 2\big(\mathbf{z}_{\text{fwd}}[1],\, \mathbf{z}_{\text{fwd}}[0]\big) \times \frac{180}{\pi}. \tag{3}$$

**Pitch** is the angle of $\mathbf{z}_{\text{fwd}}$ relative to the ground plane:

$$\text{pitch} = \arcsin\Big(\frac{\mathbf{z}_{\text{fwd}}[2]}{\|\mathbf{z}_{\text{fwd}}\|}\Big) \times \frac{180}{\pi}. \tag{4}$$

With these two angles, we can determine whether the camera rotates left or right, and tilt up or tilt down.

## F. BFS-Based Minimum-Coverage-Set Search

To ensure that an object's full dimensions are captured across multiple images, we develop a breadth-first search (BFS) algorithm that identifies minimal sets of images whose combined coverage meets each dimension's size requirement. In particular, for each axis (height, width, length), we track which subsets of object points are visible per image. If the difference between the minimum and maximum coordinates of all selected points along that axis meets a target threshold (based on the object's 3D bounding box size), we consider it covered. Our BFS proceeds in two phases at each iteration:

1. *Phase A: Coverage check.* We examine the current sets and mark any that fully cover the object on the chosen axis. These sets are recorded as "minimal," and any set that is a superset of a previously found minimal set is pruned.

2. *Phase B: Expansion.* We expand the remaining (uncovered) sets to the next level by appending additional images, while pruning those that cannot possibly achieve coverage in deeper levels.

This process continues until either no further expansion is possible or the maximum number of images is reached. The final result is a collection of minimal sets that together span the object's relevant dimension. Although we include pruning steps, the search still becomes expensive when considering sets of three or more images. Hence, we only use two images for object size perception in our data. Listing 2 shows a simplified implementation.

## G. Clustering-Based Rigid Body Segmentation

In TAPVid3D [6], all points in a sequence often belong to the same object or scene region, but they can be unevenly distributed (*e.g.*, a human torso versus arms). To sample diverse motion patterns, we segment the point cloud into multiple rigid bodies, each undergoing a distinct motion. Our method accumulates inter-point distance changes over time and applies hierarchical clustering to identify coherent groups. We also filter groups with too less points to avoid noise. Listing 3 is a simplified code snippet.

## H. Different Fine-Tuning Strategies

For research efficiency, we adopt a resource-light setting: we fine-tune the LLM backbone with LoRA while freezing the vision encoder and the MLP projection layer. This limits the trainable components and thus the benefits from our proposed datasets. We also evaluate alternative fine-tuning strategies; results are shown in Tab. 2.

We observe that increasing the number of trainable parameters yields larger gains from MultiSPA. Notably, tuning the vision encoder significantly improves performance, especially on challenging fine-grained spatial perception tasks such as camera and object motion prediction. For example, enabling vision-encoder tuning increases accuracy on camera translation vector prediction from 24.67% to 43.00%, highlighting the importance of the vision encoder for learning spatially relevant visual representations. These results are consistent with the scalability trends reported in the main paper.

## I. Comparison with Expert Models

Multi-SpatialMLLM is a general-purpose VLM capable of broad vision–language tasks. Enabled by MultiSPA data and training, it acquires multi-frame spatial understanding, such as camera motion estimation, depth perception, and visual correspondence (image matching), traditionally achieved by specialized 3D vision models. We compare these capabilities to expert systems; results are summarized in Tab. 3.

For camera vector prediction, we compare to the SOTA VGGT [11] using our fully fine-tuned 26B variant trained on 2.5M camera-vector samples. Because VGGT does not output metric-scale vectors, we report results after scale normalization. For the depth comparison task, we evaluate against Depth-Anything [12]; for image matching, we evaluate against LoFTR [8], using our fully fine-tuned 8B variant trained on 3M mixed MultiSPA samples. We could not include a 26B fully fine-tuned model on the mixed MultiSPA set due to computational limits at submission.

Nevertheless, our models match VGGT and LoFTR, and our 8B model remains competitive with Depth-Anything. Based on the scaling trends reported in the main paper, we expect further gains with larger models and data. Our goal is not to surpass specialized systems, but to show that a general-purpose VLM can attain comparable spatial perception performance while retaining broad vision–language capabilities.

Table 3. **Comparison with expert models.** For the camera vector prediction task, we compare VGGT to our 26B model; for depth comparison and image matching, we compare to our 8B model.

| VGGT v.s. Ours | Depth Anything v.s. Ours | LoFTR v.s. Ours |
|---|---|---|
| 86 v.s. 82 | 86 v.s. 76 | 71 v.s. 71 |

## J. More on VQA Benchmarks

**More results on spatial benchmarks.** We extend our evaluation to held-out spatial benchmarks, including the popular CVBench-3D [10] and ERQA [9]. As shown in Tab. 4, our model outperforms the baseline on both benchmarks, with particularly strong gains on the ERQA multi-frame subset, confirming its effective generalization to unseen benchmarks.

Table 4. **Evaluation on spatial benchmarks.**

| Model | CVBench-3D [10] | ERQA [9] | ERQA: M.V. [9] |
|---|---|---|---|
| InternVL-8B | 78.2 | 35.8 | 13.5 |
| Multi-SpatialMLLM | 81.7 | 36.2 | 21.6 |

**More results on general benchmarks.** In addition to evaluating our models on the general VQA benchmarks reported in the main paper, we further test them on three more widely-used VQA benchmarks to verify that our fine-tuning preserves most of the general VLM capabilities, as shown in Tab. 5.

Table 5. **Evaluation on standard VQA benchmarks.**

| Model | MMMU [14] | MME [3] | MMVet [13] |
|---|---|---|---|
| InternVL-8B | 47.7 | 85.8 | 62.0 |
| Multi-SpatialMLLM | 48.4 | 84.9 | 58.1 |

**Forgetting curve.** We also track the average performance on all 10 general VQA benchmarks throughout training, as

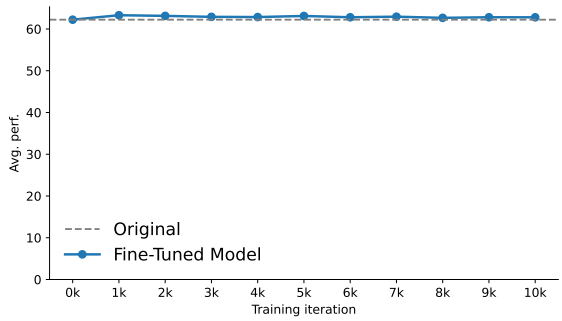



Figure 1. **Forgetting curves of Multi-SpatialMLLM training.**

shown in Fig. 1. The results demonstrate that our model maintains stable performance, alleviating concerns about forgetting. We hypothesize that the learned spatial skills are largely orthogonal to general capabilities, thereby minimizing interference with the model's pre-existing knowledge during fine-tuning.

## K. Multi-Task Synergy

**Further verification.** To further validate the benefits of multi-task training, we use the multiple-choice visual correspondence (V.C.) task, which provides a strong baseline (random guess: 25%). We report results under different cross-task training configurations in Tab. 6.

We find that including a small set of target-task samples, 1K V.C. examples, which alone are insufficient to improve accuracy, enables the model to leverage knowledge from other tasks (50K samples each). Experiments show that adding data from camera movement, object perception, and depth perception consistently boosts V.C. performance, further confirming the multi-task synergy highlighted in the main paper.

Table 6. **Multi-task synergy for visual correspondence (V.C.).** Incorporating training samples from other tasks improves V.C. performance. Abbreviations: Cam., camera movement; Obj., object perception; Depth., depth perception.

| Zero-Shot | V.C. | V.C. + Cam. | V.C. + Obj. | V.C. + Depth. |
|---|---|---|---|---|
| 33.3 | 33.1 | 40.6 | 41.0 | 57.3 |

**Benefits of different tasks.** We also observe that depth perception contributes the largest gains to V.C. The main paper shows that adding samples from other datasets helps, but it also remains unclear which task contributes most.

To study this, we adopt the same multi-task setting but include 400K training samples from a single task at a time, V.C., depth perception, or object perception. As shown in Tab. 7, V.C. and depth samples yield the largest improvements, reinforcing the benefits of multi-task training and suggesting that V.C. and depth perception may be more fundamental spatial tasks, consistent with the classic Structure-from-Motion pipeline [4].

Table 7. **Benefits of different tasks.** "Single" denotes training without samples from other tasks. Abbreviations: V.C., visual correspondence; Depth., depth perception; Obj., object perception.

| | Single | w. V.C. | w. Depth. | w. Obj. |
|---|---|---|---|---|
| Camera Movement | 9.30 | 15.00 | 15.00 | 12.00 |
| Object Movement | 17.50 | 21.88 | 22.59 | 18.54 |

## L. Generalization to More Images.

In the main paper, we primarily train on two-frame samples. To test generalization to more frames, we evaluate camera direction prediction by asking fully fine-tuned models to predict motion between the first and last frame while inserting 1–4 intermediate frames. As shown in Table 8, accuracy remains at 85%, demonstrating robustness and generalization to varying frame counts.

Table 8. **Performance of camera prediction when adding different numbers of frames.**

| # Frames | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Acc. | 85.0 | 85.3 | 85.3 | 85.3 |

## M. Limitations

Our generated data in the main paper uses two-frame scenarios. We show that models trained on two images generalize to more images, and our data-generation pipeline naturally scales to additional frames. However, future work is still needed to extend beyond pairs to exploit multi-view inputs for stronger spatial reasoning. Another limitation is that although we observe signs of the emergent phenomenon, further investigation is required to clarify what exact spatial abilities drive such emergence.

```python

def sample_dataframe(df, all_overlap_samples, non_overlap_samples,
                     overlap_min=0, overlap_max=100, interval=1):
    # 1) Sample pairs with overlap == 0
    non_overlap_df = df[df["overlap"] == 0].copy()
    sampled_non_overlap_df = (non_overlap_df if len(non_overlap_df) <= non_overlap_samples
                              else non_overlap_df.sample(n=non_overlap_samples))

    # 2) Partition the remaining pairs (overlap != 0) into bins
    remaining_df = df[df["overlap"] != 0].copy()
    bins = np.arange(overlap_min, overlap_max + interval, interval)
    remaining_df["overlap_group"] = pd.cut(remaining_df["overlap"], bins=bins, include_lowest=True)
    remaining_df.dropna(subset=["overlap_group"], inplace=True)

    bin_groups = []
    for ovlp_bin, group_df in remaining_df.groupby("overlap_group"):
        bin_groups.append((ovlp_bin, group_df))
    if not bin_groups:
        final_df = sampled_non_overlap_df.copy()
        final_df.drop(columns=["overlap_group"], errors="ignore", inplace=True)
        return final_df

    # 3) Distribute all_overlap_samples evenly across bins
    N = len(bin_groups)
    base_quota = all_overlap_samples // N
    remainder = all_overlap_samples % N
    bin_quotas = [base_quota] * N
    for i in range(remainder):
        bin_quotas[i] += 1

    # 4) Sort bins by size (ascending) and sample
    bin_data = []
    for i, (ovlp_bin, group_df) in enumerate(bin_groups):
        bin_data.append({
            "group_df": group_df,
            "quota": bin_quotas[i],
            "size": len(group_df)
        })
    bin_data.sort(key=lambda x: x["size"])

    sampled_df = pd.DataFrame()
    leftover = 0
    for info in bin_data:
        group, quota, size = info["group_df"], info["quota"], info["size"]
        current = quota + leftover
        if size <= current:
            sampled_df = pd.concat([sampled_df, group], ignore_index=True)
            leftover = current - size
        else:
            sampled_df = pd.concat([sampled_df, group.sample(n=current)], ignore_index=True)
            leftover = 0

    if leftover > 0:
        print(f"Warning: leftover {leftover} samples not used.")

    # 5) Combine sampled bins with zero-overlap samples
    final_df = pd.concat([sampled_df, sampled_non_overlap_df], ignore_index=True)
    final_df.drop(columns=["overlap_group"], errors="ignore", inplace=True)
    return final_df
```

Listing 1. The image pairs sampling algorithm for static scene data

```python
def compute_coverage(points, mask, axis):
    """Returns the min-to-max spread along 'axis' for points indicated by 'mask'."""
    if not mask.any():
        return 0.0
    coords = points[mask][:, axis]
    return coords.max() - coords.min()

def covers_dimension(coverage, target_dim, tol):
    """Checks if 'coverage' is within tolerance of the target dimension."""
    return abs(coverage - target_dim) <= tol * target_dim

def bfs_min_coverage(images, visibility, points, obj_mask, axis, target_dim, tol, max_k=2):
    """
    Finds minimal image sets up to size 'max_k' that meet coverage criteria along 'axis'.
    'images' is a list of candidate frames, 'visibility' maps frame->boolean mask,
    'obj_mask' indicates the object points in 'points'.
    """
    # Prepare BFS queue: each item is (set_of_images, combined_mask, last_idx)
    queue = []
    for i, img in enumerate(images):
        mask_i = visibility[img] & obj_mask
        queue.append(([img], mask_i, i))

    solutions = []
    k = 1
    while k <= max_k and queue:
        next_level = []
        for combo, comb_mask, last_idx in queue:
            cov = compute_coverage(points, comb_mask, axis)
            if covers_dimension(cov, target_dim, tol):
                solutions.append(combo)
            elif k < max_k:
                # Expand only if we have not reached max_k
                for j in range(last_idx + 1, len(images)):
                    mask_j = visibility[images[j]] & obj_mask
                    next_mask = comb_mask | mask_j
                    next_level.append((combo + [images[j]], next_mask, j))
        queue = next_level
        k += 1
    return solutions
```

Listing 2. Simplifed version of BFS-based minimum-coverage-set search with pruning.

```python
def smooth_distance_changes(dist_t, dist_prev, smooth_factor=0.01):
    """Zeroes out small distance changes to reduce noise."""
    diff = np.abs(dist_t - dist_prev)
    return np.where(diff > smooth_factor, diff, 0)

def rigid_body_segmentation(points, thr=0.1, smooth_factor=0.01):
    """
    points: Shape (T, N, 3), with T time steps & N points.
    thr: Threshold for clustering distance.
    smooth_factor: Ignored small changes.
    Returns: A list of groups, each group is a list of point indices.
    """
    T, N, _ = points.shape
    cum_loss = np.zeros((N, N))

    # Accumulate distance changes over time
    for t in range(1, T):
        dist_t = squareform(pdist(points[t]))
        dist_prev = squareform(pdist(points[t - 1]))
        cum_loss += smooth_distance_changes(dist_t, dist_prev, smooth_factor)

    # Hierarchical clustering
    Z = linkage(squareform(cum_loss), method="average")
    labels = fcluster(Z, thr, criterion="distance")

    # Group points by label
    groups = []
    for label_id in range(1, labels.max() + 1):
        group = np.where(labels == label_id)[0].tolist()
        groups.append(group)
    return groups
```

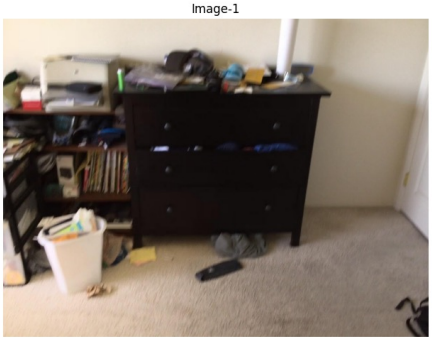
Listing 3. Rigid body segmentation with smoothing and hierarchical clustering.

```
# Depth Estimation-Dot
TASK_DESCRIPTION = [
    "<image>\nGiven an image with an annotated point, complete the question-answer task.",
]
TEMPLATES = {
    "questions": [
        "What is the depth of the annotated point in the image (in mm)?",
    ],
    "answers": [
        "The depth of the annotated point is `{depth}` mm.",
    ]
}

# Visual Correspondence Multiple-Choice
TASK_DESCRIPTION = [
    "Image-1: <image>\nImage-2: <image>\nGiven these two images, find the corresponding points between them.",
]

TEMPLATES = {
    "questions": [
        "Which point labeled A, B, C, or D in Image-2 corresponds to the circle point in Image-1? Please answer with the correct label from Image-2.",
    ],
    "answers": [
        "The correct point is labeled `{correct_label}`.",
    ]
}

# Object Perception
TASK_DESCRIPTION = [
    "Assume the scene remains unchanged. Your task is to determine the spatial properties based on the images. You need to integrate and analyze information from all provided images to get the answer.",
]

QUESTION_TEMPLATES = [
    "What is the {dimension} (in millimeters) of the {object_category} itself commonly visible in these images?",
]

ANSWER_TEMPLATES = [
    "The {dimension} is approximately `{value_mm}` millimeters.",
]

# Object Movement-Coordinate-Distance

TASK_DESCRIPTION = [
    "Image-1: <image>\nImage-2: <image>\nGiven two images, analyze the movements of objects in the images and the cameras that captured them. The movement should be relative to the first image. Note that the objects in the images and the camera may or may not have moved.",
]

QUESTION_TEMPLATES = [
    "How far did the point at [ {x1} , {y1} ] in Image-1 travel between the two shots? The coordinates [ x , y ] are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at the top-left corner. The x-axis represents the width, and the y-axis represents the height."
]

ANSWER_TEMPATES =  [
    "The point traveled a total of `{total_distance}` mm.",
]
```

Listing 4. Part of the templates used by MultiSPA dataset

Depth Comparison-Coordinate

Image-1

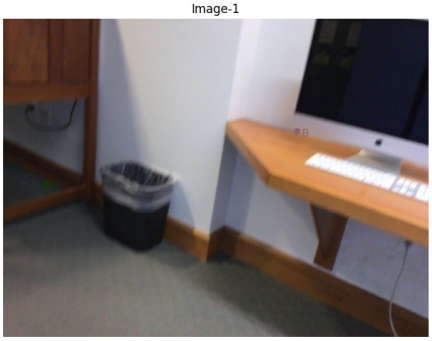


human: <image> Given an image with two points specified by their coordinates, determine which point is closer to or farther from the camera. The coordinates [ x , y ] are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at the top-left corner. The x-axis represents the width, and the y-axis represents the height. Which of these coordinates is more remote: [ 9 , 40 ] or [ 961 , 34 ]?

gpt: The point at `[ 961 , 34 ]` has the longer distance to the camera.

Depth Comparison-Dot

Image-1



human: <image> Examine the image with two annotated points and decide which point is positioned closer to the camera and which one is positioned farther from it. Can you identify which annotated point has the shorter distance to the camera?

gpt: The annotation `B` marks the closer point.

Depth Estimation-Coordinate

Image-1

human: <image> Given an image and a specific point's 2D coordinates, perform. The coordinates [ x , y ] are normalized to 0-1 and then scaled by 1000, with [ 0 , 0 ] starting at the top-left. The x-axis represents the width, and the y-axis represents the height. What is the depth at point [ 204 , 486 ] in the image (in mm)?

gpt: Measured depth of this point is `1941` mm

Depth Estimation-Dot

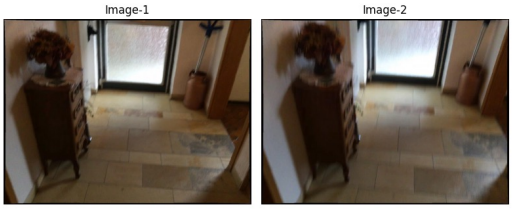
Image-1

human: <image> For the image with a noted point, ascertain the depth-related answers. Identify the depth value for the annotated point in millimeters.

gpt: The depth value of the marked point is `2249` mm.

Figure 2. **Data samples of depth perception.**

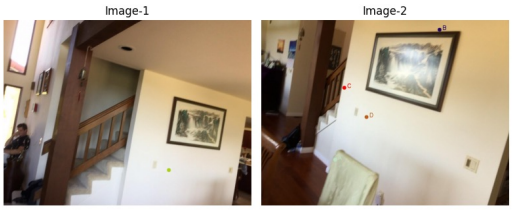
Visual Correspondence-Coordinate




human: Image-1: <image> Image-2: <image> Identify equivalent point positions between images. The [ x , y ] coordinates are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at top-left. Width follows x-axis, height follows y-axis. Find the matching position in Image-2 for point [ 314 , 111 ] from Image-1.

gpt: The point has moved to `[ 363 , 70 ]` in Image-2.

Visual Correspondence-Dot

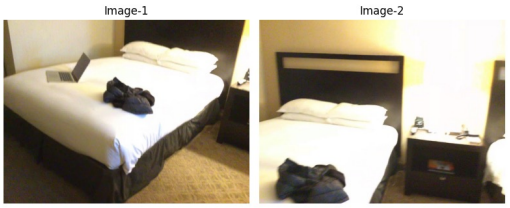



human: Image-1: <image> Image-2: <image> Determine the corresponding points between these images. Which point labeled A, B, C, or D in Image-2 corresponds to the circle point in Image-1? Please answer with the correct label from Image-2.

gpt: The correct point in Image-2 is `D`.

Figure 3. **Data samples of visual correspondence.**

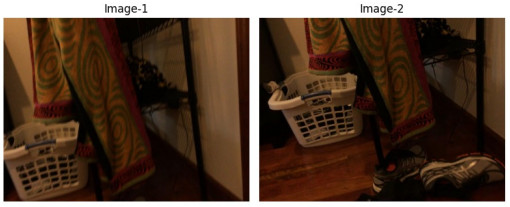
Camera Translation Direction-X axis




human: Image-1: <image> Image-2: <image> Observe and describe the camera's motion from one image to the other. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. Between these two frames, is the camera more to the left or to the right now?

gpt: It shows a noticeable move `left` along the horizontal axis.

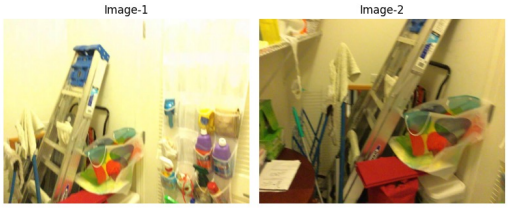
Camera Translation Direction-Y axis




human: Image-1: <image> Image-2: <image> Compare these images to infer the camera's motion. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. Is the camera vantage now elevated or depressed relative to its old vantage?

gpt: It exhibits a noticeable move `up` along the vertical axis.

Camera Translation Direction-Z axis




human: Image-1: <image> Image-2: <image> Observe and describe the camera's motion from one image to the other. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. If we consider the vantage, is it pushing in or pulling away from the center of the view?

gpt: The camera moved `forward` in depth.

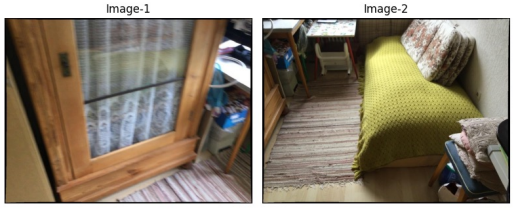
Figure 4. **Data samples of camera movement-translation direction.**

Camera Orientation Direction-Yaw


human: Image-1: <image> Image-2: <image> Observe and describe the camera's motion from one image to the other. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. Yaw is an angle describing the camera's rotation left or right around vertical. Please tell me if it pivoted left or pivoted right?

gpt: We confirm the camera has undergone a `left` yaw rotation.

Camera Orientation Direction-Pitch

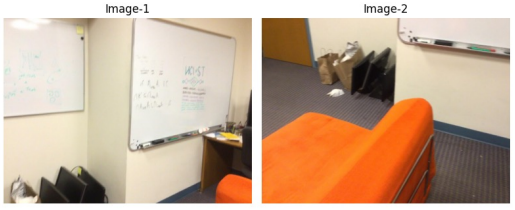



human: Image-1: <image> Image-2: <image> Examine how the camera shifted its position/orientation between these shots. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. By pitch angle, we refer to the camera's up-down rotation relative to the ground. Are we seeing an upward tilt or a downward tilt?

gpt: It shows a notable pitch rotation of `down`.

Figure 5. **Data samples of camera movement-orientation direction.**

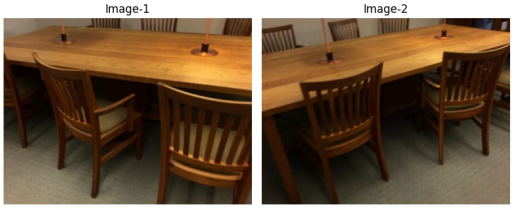
Camera Orientation Degree-Yaw




human: Image-1: <image> Image-2: <image> Determine the change in camera viewpoint across these two images. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. How many degrees does the camera revolve side-to-side from its original orientation?

gpt: The camera's left-right rotation measures about `61` degrees.

Camera Orientation Degree-Pitch

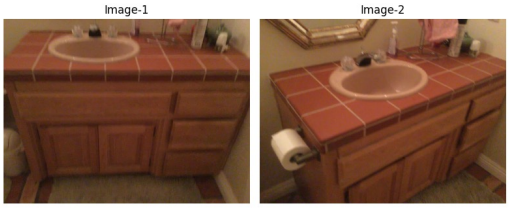



human: Image-1: <image> Image-2: <image> Examine how the camera shifted its position/orientation between these shots. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. Pitch is the angle capturing up-down spin. Could you share the camera's pitch rotation in degrees?

gpt: Observations indicate a pitch rotation of `1` degrees.

Figure 6. **Data samples of camera movement-orientation degree.**

Camera Translation Distance




human: Image-1: <image> Image-2: <image> Identify the differences in camera pose when comparing these images. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. Does the camera have a large or small travel distance, and how much is it?

gpt: Comparing both images, the camera moved roughly `742` mm in distance.

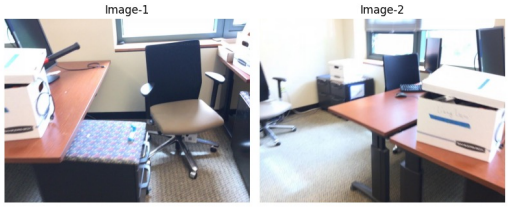
Camera Translation Vector


human: Image-1: <image> Image-2: <image> Examine how the camera shifted its position/orientation between these shots. The movement should be relative to the first image. Note that the objects in the images are assumed to be static. The first image orients X->right, Y->down, Z->forward. Please specify that vector showing exactly how the camera moved in all three dimensions.

gpt: Summarily, the camera's movement vector is `[ 18 , -148 , 287 ]` mm.

Figure 7. **Data samples of camera movement-translation distance and vector.**

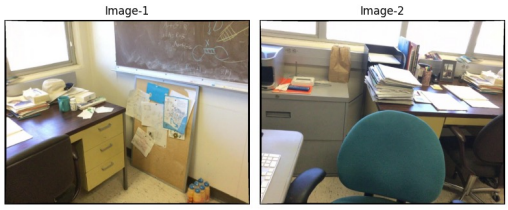
Object Perception-Height




human: Image-1: <image> Image-2: <image> Determine the spatial properties of the scene based on the images. Collate and interpret data from all provided images to conclude the answer. Find the height (in millimeters) of the monitor which is commonly visible in these images.

gpt: The height is understood to be `575` millimeters.

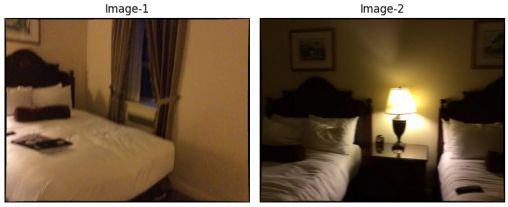
Object Perception-Length




human: Image-1: <image> Image-2: <image> Find the spatial properties of the scene using the images. You need to gather and interpret information from all provided images to obtain the answer. Calculate the length (in millimeters) of the desk that is commonly visible in these images.

gpt: The length is figured to be `1591` millimeters.

Object Perception-Width

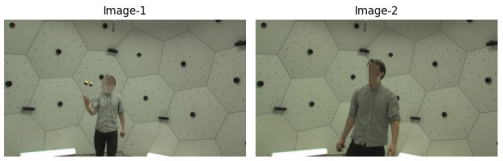



human: Image-1: <image> Image-2: <image> Using the images, identify the spatial properties of the scene. Synthesize and scrutinize information from all provided images to deduce the answer. What is the width (in millimeters) of the bed that is commonly visible across these images?

gpt: From my estimation, the width is `1595` millimeters.

Figure 8. **Data samples of object perception.**

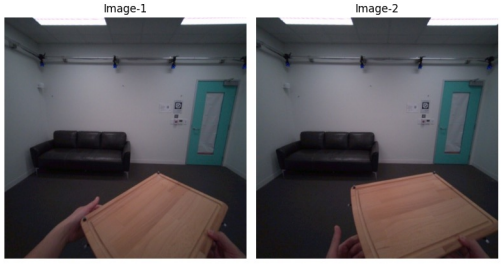
Object Movement-Coordinate-Distance (PStudio)




human: Image-1: <image> Image-2: <image> Evaluate these two images to determine how objects and camera positioning have changed, using the first image as baseline. Be aware that motion may affect either, both, or neither. Could you give me the magnitude of the point's displacement from [ 416 , 846 ] in Image-1? The coordinates [ x , y ] are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at the top-left corner. The x-axis represents the width, and the y-axis represents the height.

gpt: The total movement of the point measures `1147` mm.

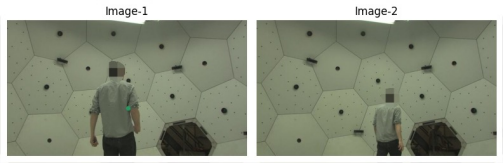
Object Movement-Coordinate-Distance (ADT)




human: Image-1: <image> Image-2: <image> Investigate the movement patterns in these two images, considering both object and camera motion relative to the first frame. Note that changes could affect either, both, or neither. Please specify the total movement distance of the point at [ 343 , 881 ] in Image-1. The coordinates [ x , y ] are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at the top-left corner. The x-axis represents the width, and the y-axis represents the height.

gpt: The total distance covered by the point is `114` mm.

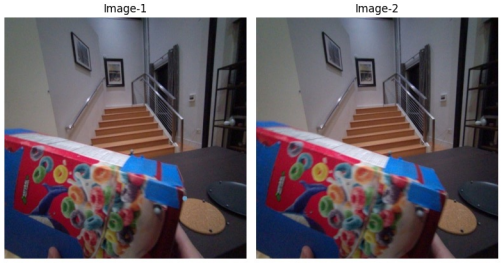
Object Movement-Dot-Distance (PStudio)




human: Image-1: <image> Image-2: <image> Map the positional changes between these two images, considering both object and camera movement from their initial state. Either, both, or neither may have shifted. Could you measure the total displacement for the annotated point in Image-1's movement?

gpt: The point's total displacement measures `1219` mm.

Object Movement-Dot-Distance (ADT)

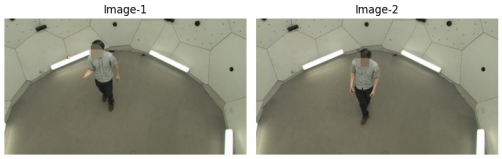



human: Image-1: <image> Image-2: <image> Identify any spatial changes between these two images, examining both object and camera motion from their initial positions. Remember that movement isn't required. I'm interested in the point-to-point distance of the annotated point in Image-1's shift.

gpt: The measured distance of the point's movement is `17` mm.

Figure 9. **Data samples of object movement-distance.**

Object Movement-Coordinate-Vector (PStudio)




human: Image-1: <image> Image-2: <image> Study these two images and identify any changes in position of objects or camera, using the first image as reference. Both elements may or may not show movement. Given X=right, Y=down, Z=forward, I'd like you to list the point at [ 408 , 435 ] in Image-1's total displacement vector `[ x , y , z ]` in mm. The coordinates [ x , y ] are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at the top-left corner. The x-axis represents the width, and the y-axis represents the height.

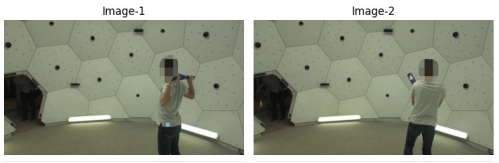
gpt: We have `[ 161 , 143 , -109 ]` mm as the shift.

Object Movement-Coordinate-Vector (ADT)


human: Image-1: <image> Image-2: <image> Compare these two images and describe how objects and the camera have moved relative to their positions in the first image. Keep in mind that either or both may have changed position. Remember X->right, Y->down, Z->forward. Please give me the exact displacement as `[ x , y , z ]` in mm for the point at [ 677 , 792 ] in Image-1. The coordinates [ x , y ] are normalized to 0-1 and scaled by 1000, with [ 0 , 0 ] at the top-left corner. The x-axis represents the width, and the y-axis represents the height.

gpt: The point's spatial transformation vector is `[ -372 , 66 , -149 ]` mm.

Object Movement-Dot-Vector (PStudio)




human: Image-1: <image> Image-2: <image> Analyze the positional shifts between these two frames, tracking both object and camera movement relative to the first image. Note that either or neither might have moved. Between these images, using X=right, Y=down, Z=forward, what's the vector that the annotated point in Image-1 traveled in mm?

gpt: The point's positional change vector is `[ -1 , -21 , 206 ]` mm.

Object Movement-Dot-Vector (ADT)


human: Image-1: <image> Image-2: <image> Study these two images and identify any changes in position of objects or camera, using the first image as reference. Both elements may or may not show movement. The first image is oriented such that positive X is right, Y is down, and Z is forward. Could you provide the annotated point in Image-1's movement vector in 3D coordinates?

gpt: The overall shift is `[ 167 , 320 , 564 ]` mm.

Figure 10. **Data samples of object movement-vector.**

## References


[1] Arjun Balasingam, Joseph Chandler, Chenning Li, Zhoutong Zhang, and Hari Balakrishnan. Drivetrack: A benchmark for long-range point tracking in real-world videos. In *CVPR*, 2024. 1

[2] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 1

[3] Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, et al. Mme: A comprehensive evaluation benchmark for multimodal large language models. *arXiv preprint arXiv:2306.13394*, 2023. 3

[4] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. 4

[5] Hanbyul Joo, Hao Liu, Lei Tan, Lin Gui, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic studio: A massively multiview system for social motion capture. In *Proceedings of the IEEE international conference on computer vision*, 2015. 1

[6] Skanda Koppula, Ignacio Rocco, Yi Yang, Joe Heyward, João Carreira, Andrew Zisserman, Gabriel Brostow, and Carl Doersch. Tapvid-3d: A benchmark for tracking any point in 3d. *arXiv preprint arXiv:2407.05921*, 2024. 1, 3

[7] Xiaqing Pan, Nicholas Charron, Yongqian Yang, Scott Peters, Thomas Whelan, Chen Kong, Omkar Parkhi, Richard Newcombe, and Yuheng Carl Ren. Aria digital twin: A new benchmark dataset for egocentric 3d machine perception. In *ICCV*, 2023. 1

[8] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. Loftr: Detector-free local feature matching with transformers. In *CVPR*, 2021. 3

[9] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025. 3

[10] Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *arXiv preprint arXiv:2406.16860*, 2024. 3

[11] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *CVPR*, 2025. 3

[12] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024. 3

[13] Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. Mm-vet: Evaluating large multimodal models for integrated capabilities. *arXiv preprint arXiv:2308.02490*, 2023. 3

[14] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *CVPR*, 2024. 3