SynergAI: Perception Alignment for Human-Robot Collaboration

APPENDIX

I. Details of SynergAI

A. 3D Scene Reconstruction

a) Reconstruction and Segmentation: From a sequence of posed RGBD images, we can reconstruct the 3D scene with several solutions. For one, the depth frames can be fused into a Truncated Signed Distance Field (TSDF) volume using the camera trajectory, and the surface mesh is reconstructed using the marching cubes algorithm [54] following ScanNet [55]. With the recent development of Neural Radiance Fields (NeRFs), we can also optimize the neural implicit representation of the 3D scene via signed distance fields (SDF) by volume rendering, with MonoSDF [56] being a signature method. In the reconstructed scene, instance segmentation is necessary to obtain information about objects within the scene. We utilize the 3D-VisTA [58] method to segment and extract positional and size information of the 3D objects.

The 3D scene can also be represented by point clouds, where the points can be accumulated from the depth image like ConceptGraphs [59]. In this way, the semantic labels can be attained by merging the image-wise prediction from 2D foundation models from multi-views.

b) Data Collection: In real-world scenarios, comprehensive scene data is essential, including RGBD data, and camera intrinsic and extrinsic parameters, to achieve decent reconstruction results. To obtain this data, we employ RGBD cameras, like RealSense [71] or Kinect [72] from existing robot setups, or from captures from iPhone ARKit packages following Multiscan [73].

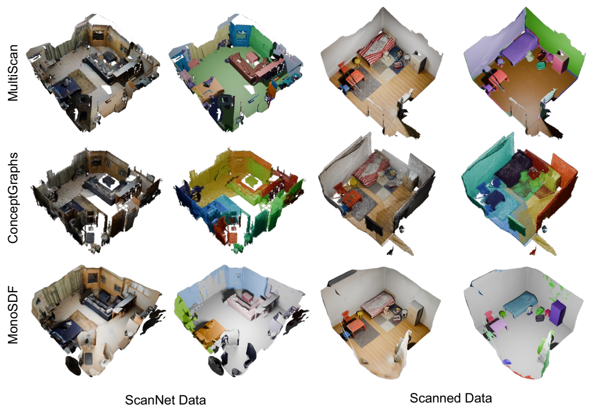


Fig. A.1: **Qualitative results of 3D reconstruction and segmentation.** MonoSDF [56] and MultiScan [73] reconstruct the 3D scenes from posed RGBD images and we apply 3D-VisTA [58] to obtain segmentation. ConceptGraphs [59] progressively reconstruct the 3D point clouds and assign semantic labels to them. The figure shows that different methods reveal limitations and failures in both reconstruction or segmentation.

B. 3D Scene Graph (3DSG)

Following prior work [64], our 3D scene graph is defined as a hierarchical graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the nodes $\mathcal{V}$ comprises $\mathcal{V}_1 \bigcup \mathcal{V}_2 \bigcup \cdots \bigcup \mathcal{V}_K$, with $\mathcal{V}_k$ representing the set of nodes at a particular hierarchical level. Each node v represents one distinct 3D object instance and the edges $\mathcal{E}$ represent spatial relationships between nodes. The relationships that our 3D scene graph captures are shown in Table A.1. The hierarchies are determined by the `support` relationship; for instance, objects supported by the floor constitute $\mathcal{V}_0$, while objects supported by the table will form $\mathcal{V}_1$, *etc.*. Note that edges originating from one node $v \in \mathcal{V}_k$ may only terminate in nearby hierarchies $\mathcal{V}_k \cup \mathcal{V}_{k+1} \cup \mathcal{V}_{k+1}$. In other words, edges in the scene graph exclusively connect nodes within the same hierarchical level, or one level higher or lower.

We instantiate the graph nodes with the instance segmentation from the point cloud and parameterize each node with object centroid $\mathrm{p_i} \in \mathbb{R}^3$ and size of the bounding box. Next, we traverse all the nodes to determine their spatial relationships. In addition, we utilize an automatic verification procedure to validate the scene graph, further improving the quality of the scene graph we constructed. One of the verification operations involves manually maintaining a mapping between objects and relationship descriptions based on common sense. For example, people usually use "mounted on" to describe the relation between TV and wall, rather than "hanging on."

To get detailed attributes of an object's visual and physical properties, we utilize the object captioning pipeline outlined as follows. Given the multi-view images, we use the point cloud of the object to get the visible points in the images through rendering. The image is then cropped with the rendered bounding box and processed through BLIP2 [40] to generate information about the object color, shape, material and affordance, *etc.* For the attributes from every image, we calculate its CLIP [74] similarity score between the text and the cropped image and select the top 10 with the highest CLIP score and minimal occlusion. The selected attributes are fed into a LLM to obtain a coherent summary for the object. In this process, we explicitly instruct the language model to identify and correct the potential errors.

C. System Design

Task Decomposition Our system decomposes the complex tasks into intermediate steps and allocates tools to complete them. We prompt the system for task decomposition, which are demonstrated from our prompt template in Fig. A.3a and Fig. A.3b. The first three lines in Fig. A.3a correspond to the prologue of this prompt that instructs the agent to rely on the tools for responding to a user's inputs. The prologue is followed by four in-context examples of plans. Note that in these plans we not only specify the tools to be used but also the reasons for selecting these tools.

TABLE A.1: **Relationships in 3D Scene Graph (3DSG).**

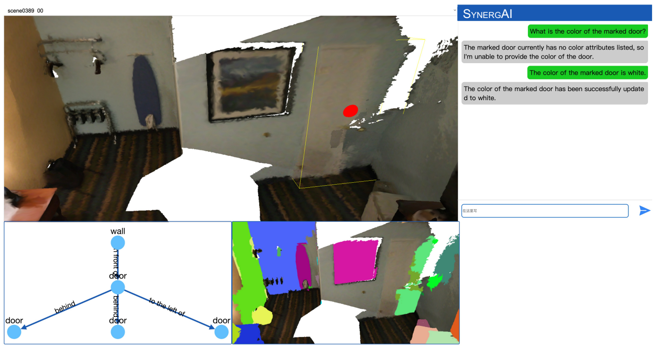
Category	Relation	
Vertical Proximity	support hanging on mounted on below higher than	embed inside affixed on above lower than
Horizontal Proximity	near besides	far next to
Allocentric	left behind	right is in front of

Fig. A.3b includes an answer format section that illustrates the syntax for calling tools and the sequential process of resolving a user input and several rules.

Observation For LLMs to comprehend information in 3DSGs, we render the information retrieved from actions using templates. Specifically, observations are organized on the basis of objects. Observation for each each object can be rendered from three templates. The first one is for position and size: "`The position of the {object.name} (id: {object.id}) is {object.position}.`". The second one is for attributes: "`The {object.name} (id: {object.id}) has attributes: {object.attributes}.`". The third one is for relations: "`The {object.name} (id: {object.id}) is {relation} {name_id_list}.`", where "`{name_id_list}`" is a list of strings in the format of "`{name} (id: {id})`" generated for objects that have the relation of interest to the object. To save up tokens, the third template is used only for tools that reason about relations, such as `query_for_relations`. When a tool retrieves multiple objects, we iterate through the objects, render strings, and then concatenate the strings to form the observation for the tool.

Human-Robot Interaction As mentioned in Section III-B, to facilitate human-robot alignment in the presence of perception errors, we design a GUI that allows users to freely interact with the scene by dragging, zooming-in/out, changing views, marking objects and asking free-form questions at will. Furthermore, it provides the potential to combine language-based interaction with object clicking for object referencing. Fig. A.2 shows an example, in which the user inspects and corrects the color of a door. In the GUI, the user selects the object by clicking on a point. After a point is clicked, we compute a 3D ray using the point (i.e. a point in the viewing plane) and the user's current viewing angle. We take the first object whose meshes intersect with this 3D ray as the marked object and display its bounding box in the GUI, which is the yellow box around the door in Fig. A.2.

Tools The tools are a set of Python functions designed for interacting with 3DSGs and the users. When generating the overall step-wise prompt, LangChain extracts the doc-strings of tools and injects them into the prompt, so the doc-strings play a vital role in implementing tool usage. We use a fixed format for the doc-strings and show an example in Fig. A.4. The doc-string starts with a brief description of the tool's functionality (lines 3 and 4). The "Hints" section specifies expected behaviors. For example, the 1th hint says that the ambiguity of objects should be resolved by the user. If one changes this hint to "......, call this tool for each of the candidates", then all of the objects related to the user input will be altered. The following three hints dictate constraints on the input argument, `new_attributes_list`. In this case, we expect the LLM to update one or more attributes of an object but keep the rest of the attributes unchanged. The "Argument" and "Return" sections are introduced to facilitate tool calling.



What is the color of the marked door?

The marked door currently has no color attributes listed, so I'm unable to provide the color of the door.

The color of the marked door is white.

The color of the marked door has been successfully update d to white.

Fig. A.2: **A screenshot for our interface.** A user can select the scene to interact using the drop-down menu located at the upper left. The left part consists of the reconstructed view, the local 3DSG for the object of interest (bottom left), and object segmentation (bottom middle). The user can chat with our system using the input box located in the middle right. It can also select an object by clicking it in the reconstructed view or the 3DSG. In this example, the user marks the door.

II. Experiments

A. Setup

We utilize the reconstructed meshes provided by the dataset for our experiments on the ScanNet [55], which are reconstructed by depth fusion and marching cubes [54]. For real-world robot execution, we utilize MonoSDF [56] to obtain the 3D scene reconstruction. We utilize 3D-VisTA [58] to get the instance segmentation for all the experiments.

B. Alignment Tasks

We list all the alignment tasks from ScanNet in Table A.2. Based on the number of objects involved with perception misalignment, the tasks are divided into *EASY* (`misalignment=1`) to *HARD* (`misalignment>1`).

We engage the participation of 10 human subjects for our alignment experiment, each assigned tasks across 3 scenes. For the ablation study, the subjects are allowed to use the GUI but are instructed to refrain from using mouse clicks. We randomly select 3 participants from the 10 and assign each 2 scenes randomly from the task pool to report the quantitative evaluations.

~~~~
You are an assistant agent in a room and you have to respond to input from a user. When the user ask for information of items in the room, collect the required
information using your tools. When the user ask you to learn something, use your tools to do so. When the user ask you for advices or actionable plans, make up one with
your own knowledge and ground the plan to the room using your tools.

Make a rough plan first. Each step of this plan corresponds to a single action. State the plan before your first action.
[start of example plan]
User Input: Find the bed.
Plan:
1. Use query_related_objects to extract candidate items that could correspond to the bed.
2. If you can identify the bed, then include information of the bed in your response. Otherwise, in your response, ask the user for clarification.
3. Use post_process.
4. Use Final Answer to return your response to the user.

User Input: What is to the left of the marked item?
Plan:
1. Use find_marked_object to to locate the marked item.
2. Use query_for_relations to gather information of spatial relationship between the marked item and other items, then determine the all items that are to the left of them
marked item.
3. Use post_process.
4. Use Final Answer to return your response to the user.

User Input: What is in the middle of the bed and the desk?
Plan:
1. Extract the coordinates of the bed and the desk by using query_for_items_info.
2. Use calculate_mid_point to compute the mid point of the coordinates of the bed and the desk.
2. Use find_object_closest to find the item that is closest to the mid point of the bed and the desk.
3. Use post_process.
4. Use Final Answer to return your response to the user.

User Input: I want to fall asleep in a warm and dark environment. What should I do?
Plan:
1. Use query_related_objects to find all items that are related to [fall asleep warm dark]. Compose a imaginary plan as if you could operate the related items.
2. Use post_process.
3. Use Final Answer to return your imaginary plan to the user. Remember to include the [id] and [location] of related items in your plan.
[end of example plan]
~~~~

(a) The first part of our prompt template.

~~~~
At each step, select an action, analyze your observation, and determine your next action based on the observations you received so far, the user's request, and your
original plan. Follow this format.

[start of answer format]
Action: ```{{{{“action”: [selected action], "action_input": [action input],}}}}```
Observation:
Thought: [thought]
Action: ```{{{{“action”: [selected action], "action_input": [action input],}}}}```
Observation:
... (repeat Thought/Action/Observation N times)
Thought: [thought]
Action: ```{{{{“action”: post_process, "action_input": [action input],}}}}```
Observation:
Thought: [thought]
Action: ```{{{{“action”: Final Answer, "action_input": [final response],}}}}
```
[end of answer format]

Rules:
1. Be sure to include an Action in your response.
2. For [selected action], use one tool from [{tool_names}].
3. For [action input], refer to the descriptions of the tools.
4. When choosing Final Answer, do not format [final response] as a dict. Use a sentence in natural language for [final response].
5. Be sure to choose Final Answer at the last step.
6. For [thought], include your reasoning for choosing the next step.
7. State your plan before the first action.
"""
~~~~

(b) The second part of our prompt template.

Fig. A.3: **Our prompt template.** This template will be combined with doc-strings of tools, latest observations, and historical information.

```python
def update_object_attributes(
    new_attributes_list: List[str], object_id: str, **kwargs) -> dict:
    """Update the attributes of a specific object. Use this tool when the user
    ask you to change some of the attributes of an object.

    Hints:
        1. If there are multiple candidates for object_id, do not call this tool
        and ask the user for clarification.
        2. The attributes of an object is a list of strings that specify the color,
        the texture or other information of an object. Before using this tool, use
        query_for_objects to get the current attribute list of the object under
        consideration.
        3. To prepare the input argument new_attributes_list, start with the
        current attributes list. Replace the corresponding elements in the current
        attributes list with new values specified by the user. For example, if the
        current attributes list is ["blue", "wooden", "rectangular"] and the user
        asks you to change the color to red and the shape to triangular, then the
        new attributes list should be ["red", "wooden", "triangular"].
        4. Do not alter the values of the attributes not mentioned by the user.
    Arguments:
        new_attributes_list: list, the new list of attributes to be assigned to the
        object, eg. ["blue", "wooden"]
        object_id: str, the id of the object to be updated, eg. "1".

    Return:
        observation: str, information about the object under consideration.
        results: list, a list that contains the object under consideration.
    """
```

Fig. A.4: **An example for the doc-strings of tools.**

TABLE A.2: **Evaluation tasks designed for the ScanNet dataset.** In our experiments, we also provide participants with images for objects involved in each task, so that they can refer to the objects by clicking.

Scene	**Task ID**	**Category**	**Difficulty**	**Task**	**Expected Answer**
scene0011_00	t1	Attribute	Easy	What is the tv made from?	Plastic.
	t2	Spatial	Hard	What is the item that is above the stove?	Stove hood.
	t3	Numeric	Hard	How many tables are there in the room?	Two.
scene0050_00	t1	Attribute	Easy	Is the blue box triangular in shape?	No.
	t2	Spatial	Easy	What is the item above the blue box?	Toolbox.
	t3	Spatial	Hard	What is the item's name for sitting and to the right of desk and behind the door?	Sofa.
	t4	Spatial	Hard	What is the item above the desk and beside the laptop?	Printer.
	t5	Spatial	Hard	What is the item in front of the piano?	Piano bench.
	t6	Numeric	Easy	How many chairs are there in the room?	One.
scene0169_00	t1	Attribute	Easy	What is the color of the plastic trash can?	Gray.
	t2	Attribute	Hard	What is the partition made from?	Glass.
	t3	Attribute	Hard	Is the red backpack oblong in shape?	No.
scene0342_00	t1	Attribute	Easy	What is the color of the screen?	Black.
	t2	Attribute	Easy	What is the color of the desk that is lower than the screen?	Brown.
	t3	Spatial	Hard	Is the red backpack placed on the black table?	Yes.
	t4	Spatial	Easy	Is the red backpack hung on the wall?	No.
scene0355_00	t1	Attribute	Easy	What is the microwave oven made from?	Stainless steel.
	t2	Attribute	Hard	Is the armchair round in shape?	No.
	t3	Attribute	Easy	Is the armchair made from wood?	No.
	t4	Numeric	Hard	How many tables are there in the room?	Two.
	t5	Numeric	Hard	How many chairs are there in the room?	Eight.
scene0356_00	t1	Attribute	Easy	What is the color of the dresser?	Shallow yellow.
	t2	Attribute	Easy	Is there any white recycling bin in the room?	Yes.
	t3	Attribute	Easy	Is the black chair rectangular in shape?	No.
	t4	Attribute	Easy	Are the shelf and the desk of the same color?	Yes.
	t5	Numeric	Hard	How many doors are there in the room?	Two.
scene0389_00	t1	Attribute	Easy	What is the color of the door?	White.
	t2	Spatial	Hard	Is the hanger located to the left of the ironing board?	Yes.
	t3	Spatial	Hard	What is to the left of the cabinet on which a TV is resting?	Include a refrigerator.
	t4	Attribute	Easy	Are the two doors in the same color?	Yes.
scene0406_00	t1	Attribute	Easy	What is the shape of the door?	Rectangular / oblong.
	t2	Attribute	Easy	What is the white sink made from?	Ceramic.
	t3	Attribute	Easy	Are the sink and the bathtub made from the same materials?	Yes.
scene0427_00	t1	Spatial	Hard	What is behind the trash bin?	A glass partition and a door frame.
	t2	Numeric	Easy	How many tables are there in the room?	One.
	t3	Numeric	Hard	How many chairs are there in the room?	Four.
	t4	Numeric	Hard	How many doors are there in the room?	One.
scene0144_00	t1	Attribute	Easy	What is the white rectangular dresser made from?	Wood or metal.
	t2	Attribute	Easy	What is the monitor made from?	Plastic.
	t3	Spatial	Easy	What is item on top of nightstand?	Printer.
	t4	Spatial	Easy	What is item supporting the monitor?	Small desk.
	t5	Numeric	Easy	How many dressers are there?	Two.