

Dynamic Scene Graph with Human Robot Interaction

by

Ualikhan Tukenov

Submitted to the Department of Robotics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

Apr 2026

© Nazarbayev University 2026. All rights reserved.

Author
Department of Robotics
Apr 17, 2026

Certified by
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by
Elizabeth Arkhangelsky
Dean, School of Engineering and Digital Sciences

Contents

1	Introduction	9
1.1	Importance of the Subject	10
1.2	Knowledge Gap	11
1.3	Potential Impact and Contributions	13
2	Literature Review	14
2.1	From Static Maps to Dynamic Scene Graphs	14
2.2	Open-Vocabulary Perception	15
2.3	Dynamic Scene Graphs for Mobile Manipulation	16
2.4	Temporal Reasoning and Change Detection	17
2.5	Scene Graphs for Task and Manipulation Planning	18
2.6	Multi-modal Understanding	19
2.7	Humans in the Scene Graph	20
2.8	LLM-Guided Robot Control	21
2.9	Summary and Gap	22
3	Methodology	24
3.1	System Architecture	25
3.1.1	Hardware Setup	25
3.1.2	Communication Architecture	26
3.1.3	Software Stack	26
3.2	Perception Pipeline	28
3.2.1	Open-Vocabulary Object Detection	28

3.2.2	Object Color Extraction	29
3.2.3	Object-Level 3D Reconstruction	30
3.2.4	Object Association	31
3.2.5	Tracking People by Their Movements	32
3.3	Interactive 3D World Map	34
3.3.1	Graph Structure	34
3.3.2	Scene description for language model	35
3.3.3	Relation Detection	36
3.3.4	Local Update Mechanism	39
3.3.5	Frustum-Aware Persistence	40
3.3.6	Workspace Scan Routine	41
3.4	LLM-Based Task Planning	42
3.4.1	Task Decomposition	42
3.4.2	Object Name Resolution	45
3.4.3	Scene-Change Replanning	46
3.4.4	Failure Replanning	47
3.4.5	Error Recovery	47
3.5	Manipulation and Execution	48
3.5.1	CRISP Compliant Controller	48
3.5.2	Grasp Planning	48
3.5.3	Handover Execution	50
3.5.4	Receive Execution	51
3.5.5	Human Proximity Safety	52
3.5.6	Speech Input	52
3.5.7	Visualization	53
3.5.8	Action Execution Flow	53
4	Experimental Setup	55
4.1	Hardware and Software	55
4.2	Test Environment	56

4.3	Experiments	56
4.4	Evaluation Metrics	58
4.5	Data Collection	58
5	Results	60
5.1	Implementation Status	60
5.2	Perception Pipeline	61
5.2.1	Object Association	62
5.2.2	Color Recognition	62
5.3	Scene Graph and Relation Detection	62
5.3.1	Spatial Relations	62
5.3.2	HOLDING Detection	63
5.3.3	Frustum-Aware Persistence	64
5.3.4	Scene Graph Accuracy	65
5.4	LLM Task Planning	65
5.4.1	Plan Generation	65
5.4.2	Scene-Change Replanning	67
5.5	Manipulation Execution	67
5.5.1	Simulation Testing	67
5.5.2	Real Robot Experiments	68
5.5.3	Extended Consecutive Task Sequences	69
5.6	Problems Found While Building	69
5.7	Summary	69
6	Discussion	72
6.1	Design Decisions	72
6.2	Limitations	75
6.3	Future Work	77
7	Conclusion	79
A	LLM System Prompt	82

List of Figures

2-1	DovSG pipeline.	17
3-1	System architecture.	25
3-2	ROS 2 node-topic graph.	27
3-3	Open-vocabulary perception pipeline.	29
3-4	Example scene graph.	35
3-5	Node state machine for frustum-aware persistence.	40
3-6	Top down view of the scan.	41
3-7	LLM planning flow.	45
3-8	Scene-change replanning timeline.	46
4-1	Robot with RGB-D camera mounted on a wrist.	57
5-1	Example of detected objects.	61
5-2	Human pose detector and tracker demonstration.	63
5-3	Example of human-object interaction.	64
5-4	Scene-graph detections description.	64
5-5	LLM task decomposition example.	66

List of Tables

2.1	Related work comparison.	23
3.1	Key software components of the DSG-HRI framework.	27
3.2	Grasp strategies by object category.	49
4.1	Hardware configuration.	55
4.2	Software components.	55
4.3	Test objects.	56
4.4	Task combinations for consecutive execution experiments.	58
4.5	Evaluation metrics.	59
5.1	Implementation of the framework at the moment.	60
5.2	Example LLM planning results from real-robot testing.	66
5.3	Simulation environments and results.	67
5.4	Individual action success rates (15 runs each).	68
5.5	Consecutive task combination success rates (15 runs each).	68
5.6	Issues identified and resolved during development.	70
6.1	Known limitations of the current system.	76
6.2	Planned system improvements.	77

Dynamic Scene Graph with Human Robot Interaction

by

Ualikhan Tukenov

Submitted to the Department of Robotics
on Apr 17, 2026, in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics

Abstract

This thesis presents DSG-HRI, a system which enables a fixed-base robotic arm to perform language-guided manipulation for extended periods with humans in the loop. Novel objects are recognized using state-of-the-art vision models such as RAM++, Grounding DINO, SAM2 and CLIP, while hands in the scene are tracked even under partial depth-occlusion. A dynamic scene graph is built, comprising all recognized objects, hands, and the relations between them, including spatial, semantic, and interaction-based relations. The scene graph is maintained locally by tracking objects, hands, and relations rather than being rebuilt from scratch. Recognized objects are kept in memory across wrist camera frames, even when the camera points away from them during a grasping motion. Language is parsed by a locally-run language model into a sequence of actions that the arm is then commanded to perform. The generated plan is executed by the arm and, as soon as execution begins, dynamically re-planned whenever the objects move more than 5 cm. A fully integrated system is presented for robots that perform a series of tasks involving physical human-robot interaction and compliant motion. The experiments were conducted in a real-world laboratory environment. A CRISP Cartesian impedance controller is used to achieve the compliant behavior needed for physical human-robot interaction, and a Franka Research 3 robot arm executes the tasks. I performed 125 real-world trials and obtained success rates of 87% for pick, 80% for place, 93% for handover, and 87% for receive. For consecutive task combinations, I obtained 60–87% success, and the longest continuous sequence reached 10 tasks. I also achieved an average scene graph accuracy of 83% against ground truth. In the real-robot experiments, I observed that GripperCommand and Franka Grasp are currently not compatible, and that the order of the workspace scan affects depth-based position accuracy.

Keywords - *Scene graph, franka emika, human-robot interaction (HRI), robot manipulator, large language model (LLM), dynamic open vocabulary.*

Thesis Supervisor: Almas Shintemirov

Title: Associate Professor

Acknowledgments

I would like to express my gratitude to my thesis supervisor Almas Shintemirov, and my lab-mates who contributed and participated in testing and setting up my project.

Chapter 1

Introduction

Machines are increasingly being placed in more cluttered and dynamic environments whether in a manufacturing setting or domestic living space, or in formal settings such as hospitals and work places. The biggest challenge to machine design is the unpredictable nature of the humans who will be using them. Humans and their behaviour are often unpredictable and can move objects around a setting at will. The layout of a corridor can change unexpectedly due to the random movement of other people. Jobs change over time and the way in which decisions are made varies. Instructions may need to be followed differently in various situations and new instructions may need to be issued as a result of a change in circumstances.

Current “old” robots are very simple minded. Their behavior is based on a pre-computed map of a task, which they then repeat over and over again. Such repetition is acceptable on an assembly line where all parts are always presented in exactly the same way. In a kitchen or shared workspace, this is not the case. Objects can be moved by a human right as the robot attempts to pick them up; as a result, there is very little predictability.

In order to engage in interaction with people, a robot for human-robot interaction must have some fundamental capabilities. First, it must be able to perceive its environment by acquiring a representation of its surrounding space, identifying static or moving objects, and understanding their relation to other objects. In addition, the robot must be able to understand spoken or written commands so that non-expert

users can interact with it naturally. Moreover, the robot must be able to detect changes in the situation while it is engaged in an action.

Internal data structures like scene graphs organize the surroundings of a scene in a structured layout. The objects are represented as nodes in the scene graph. The edges between nodes represent relationships between objects like parent/child relationships. The cup is *on* the table. The person is standing *near* the robot. The phone and the keyboard are *next to* each other. By organizing the surroundings of a scene in a scene graph, computers can understand what is around them. Kim [13] and colleagues previously explored how a robot might perceive in 3D to better understand the world around it. In another work, Chang [3] explains how to build 3D structure, and then how that 3D structure is used both in vision, and in robot tasks.

Modern robots can now recognize objects that were not part of their training set. As the environment changes slowly (e.g. a book falls off a shelf), the system can adjust for the change by updating a single part of its model. Recent improvements in scene understanding enable robots to better interpret the world around them. These advances are complementary to the other capabilities of collaborative robots, and combining them into a single functioning framework remains an open challenge.

1.1 Importance of the Subject

Collaborative robots (cobots) are becoming increasingly popular in manufacturing environments, where humans and machines share work surfaces. A robot might sit next to an employee at a table, for example, and the two might work together to assemble products.

The robot first observes the scene to detect the red cup from the other objects and also the location of the human. Once the red cup is detected, the robot plans a sequence of actions such as looking at the target, moving towards the target, picking the cup, lifting up the picked cup, moving closer to the recipient, and finally releasing the cup to the recipient. The robot changes its gripping motion when it moves towards the recipient and adapts to these changes in real time automatically. Updated

positioning triggers revised trajectory calculations.

In today’s fast changing world and surroundings, it is important to perform well by staying aware of what is going on around you. People and objects don’t stand still forever, and it is no longer sufficient to rely on data from past snapshots. New objects in the scene will have appeared that were not there previously, people will have changed position, and assumptions about what was going to happen may no longer be valid. Instead, all this information must be tracked on a living map of the surrounding space, updated by every signal received. The technology is continuously evolving and adjusting to the changing environment around us.

1.2 Knowledge Gap

All the technology to build a system that follows instructions given in natural language already exists. Instead of classifying images into fixed categories such as “dogs” or “cats”, open-vocabulary perception tools such as Grounding DINO or CLIP can be used to detect objects and name them using natural language. The instructions to a computer can then be encoded in machine-readable language using a large language model. Instead of processing commands phrase by phrase, the system first parses the intent of the command and then determines what steps need to be performed. The technology builds a scene graph of a space over time, tracking moving objects and their connections as objects move around in the space. At the same time it runs a human pose estimator that tracks where a person is sitting or standing and the 3D locations of their hands. It can even accurately track the body pose of a partially occluded person.

Currently, there is no framework that addresses the requirements of a fixed-base collaborative robot. While there are systems that support dynamic scene graphs for real-time rendering, humans in the workspace are not typically modeled. Human-robot collaboration typically treats the human as a static obstacle. Many mobile manipulators perform simple pick-and-place tasks by moving between rooms. Few systems attempt multi-step, long-term tasks, or involve human interaction while the

robot is performing its task.

There is a growing difference between what researchers study and what industry develops, which requires the automation of new tasks. The fixed-base robotic arm is one of the most common arm types currently in industrial use and research labs. These arms are usually installed in small areas, surrounded by people moving constantly around the workspace. Online adaptability to these changing workspaces is critical. In practice, a person-directed, speech-recognizable, autonomously adaptable arm fulfills an important industrial task, and advances robot-interactive research at the same time.

I present a flexible 3D scene graph system that allows a robotic arm to perform open-ended tasks using an open-ended vocabulary of nouns and actions while following natural language commands in dynamic environments with moving objects and people nearby.

The research question is:

How effective is the performance of a fixed-base arm in long-term tasks while a person moves about in the scene or manipulates objects, coupled with a dynamic open-vocabulary 3D scene graph representation and an LLM-based approach to task decomposition?

This thesis presents the full setup consisting of a Franka Research 3 arm built with open-vocabulary sensing (camera, range, grip), dynamic human position tracking, and a dynamic scene representation. The scene map is updated locally by the perception modules as the robot operates. Instead of following a pre-planned sequence of commands, the robot makes decisions based on language models that can re-plan online. The resulting motion follows soft physical rules and is implemented via a Cartesian impedance controller. All sensors and actuators of the robot are processed by software running on separate computers and using the ROS 2 middleware. The parts are connected in sequence to form the whole but they function independently as well.

1.3 Potential Impact and Contributions

Scientific Impact

The results from field tests where dynamic scene generation is evaluated in terms of accuracy, consistency, and performance on changing environments can lead to future directions of research, such as robot handling, scene understanding, and flexible behavior.

Technological Impact

This is one of the few projects built on ROS 2 Jazzy with all incompatibility issues between Jazzy and Humble resolved, and can contribute to future development in this ecosystem.

Economic Impact

After communicating and adapting to circumstances, also learning to talk and adapting to new situations, robots can be deployed in environments such as care-assistance, distribution, hospital-assistance and other production-line type tasks. As programming rules are relaxed and static settings become less relevant, robots can be introduced in real-world scenarios where they need to quickly adapt to new environments, circumstances and situations.

Societal Impact

Instead of just being placed in the environment, the robots interact with people, moving through the space while carefully tracking individuals. Thus, progress is made by having robots coexist with people. The system moves from having robots located near people to having robots share activity with people.

Chapter 2

Literature Review

A look back at earlier studies opens this section, showing paths that shaped the current project. Five lines of inquiry stand out - how 3D scene graphs shifted from fixed forms to adaptive ones, how open-vocabulary systems began linking with these structures, ways they support robots in planning actions, where human input fits into such setups, and how large language models started steering robotic decisions. Every strand adds one part to the framework built here.

2.1 From Static Maps to Dynamic Scene Graphs

Back then, robots mapped surroundings using grids or scattered points. Although these methods record shape, they ignore significance. Something blocks space - yet the machine cannot tell what it is, nor how it connects to nearby items.

Graph-based representations help overcome this constraint. Within such frameworks, the setting appears as interconnected points - each standing for items, areas, spaces, or beings - with links showing how they relate in space, meaning, or time sequence. By using these connections, machines interpret environments beyond mere shape data. Structure enables elevated understanding compared to basic spatial models.

Rosinol et al. [21] developed Kimera a construction method of metric-semantic maps. Levels (things, spaces, people, architectural components) are organised in

maps, organised in a metric-semantic hierarchy across levels. In the updated version of Kimera Rosinol et al. [22] instead of representing a mapping at startup time as a static one time scene creation Kimera can reconstruct and reorganise a mapping with respect to the scene and activities over time. In summary, a traditional concept of simple mapping started being used to represent a changing objects and environment.

In a bid to understand how robots can give meaning to their spatial environment through data, Mascaro and Chli [17] instead turned to reality and found that structured scene graphs can not only be used to support tasks such as meaning-aware motion and decision making, but also future planning. Rather than analysing unstructured sensor data, the robot can now interrogate the connections between objects in huge detail. For example, a robot may find and list objects on a table or other surfaces.

2.2 Open-Vocabulary Perception

Our current approaches only allow a single trained model to be effective at recognizing the items it has been shown in the past to recognize what it sees (e.g. a system that knows what objects are there can recognize them). However, the spaces in which humans live are filled with all sorts of unknown items (e.g. staplers, cable organizers, alarm clocks, etc.) and we would like to explore how systems can handle objects that do not fit within the rigid categories that have been used to train models.

Open-vocabulary learning makes robots more universal compared to rigidly labeled approaches. Instead of relying on image retrieval models such as CLIP, the robot can utilize any word that is part of the human language when learning from images with a vision-language model (VLM). Objects the robot has never seen before can still be identified and labeled by the robot using everyday language. No further training is required for the robot to assign meanings to the new objects.

Gu et al. [7] presented a method that combines multi-view 3D data with the CLIP model to obtain object-based, open-vocabulary 3D scene representations. Scenes can be queried with everyday language such as "find me a red cup in a room". The

results of scene finding and object location tasks outperform the old versions of similar systems. Building on these results, Linok and Naumov [14] extended 3D scene graphs with a hierarchical structure and achieved great results for indoor object finding and location tasks using images.

Training an image recognizer used to require a fixed, pre-specified set of items to train against. With newer techniques, you can now tune and test against items you have never seen before. That leap matters most when robots move through homes, offices, or streets, places packed with surprises. Without such flexibility, performance crumbles in real life.

2.3 Dynamic Scene Graphs for Mobile Manipulation

Closest to this thesis stands the DovSG framework from Yan et al. [27]. Built around a single design, it combines open-vocabulary sensing, adaptive memory, task breakdown via large language models (LLMs), route planning, along with physical interaction. Movement through space becomes possible for a mobile robot handling complex sequences over extended periods, even as surroundings shift. Its standout feature? A selective memory refresh: instead of rebuilding the full scene layout, only portions tied to moved objects get revised. Tested in four actual rooms - each altered differently - it outperformed fixed approaches by reaching a 30% greater success rate on prolonged tasks. The whole pipeline of this work is showed in Figure 2-1.

Starting differently each time, Honerkamp [8] and team tested how spoken cues help robots build real-time maps while searching rooms. Because language informs decisions, their system found items faster than random scanning would allow. Though based on words, the method adjusted structure mid-search without fixed templates. While moving through spaces, the robot updated its understanding using only verbal hints. This shift cut down delays when tracking specific objects in cluttered areas. As actions unfolded, guidance from speech shaped where to look next. Instead of preloaded plans, responses emerged during interaction with surroundings. So, less time was wasted revisiting already-checked spots. By linking words to motion, search

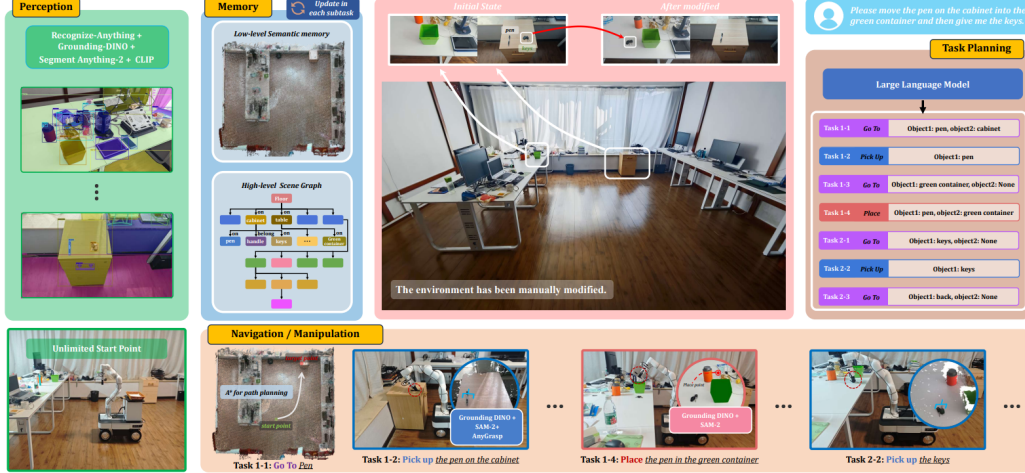


Figure 2-1: DovSG pipeline.

became more focused overall. Even so, success depended heavily on accurate initial descriptions provided by users.

Instead of moving around, one arm stays put while doing its work. Ideas once meant for roaming machines now fit an arm that cannot shift location, handling everything within reach from just one spot.

2.4 Temporal Reasoning and Change Detection

A shifting scene graph needs to capture what’s happening now, while also working out patterns across moments. Its role stretches beyond snapshots - predicting movement matters just as much. Time is a structure imposed on relations between objects. It is unavoidable when constructing any reality model and it changes how you see things. Something at rest turns out to be moving, and something changing can be described, step by step, as it changes from moment to moment. The system’s simulation generates a sequence of views of reality, each one corresponding to a moment in time. Basically to track things the system need to see past positions and shifts in them.

Instead of analyzing one image or video frame at a time, Feng and colleagues [6] learned a model that uses prior observations to make better predictions about object

relationships in a sequence of images over time. By looking back at what happened in prior images, the model can better understand what happened between them. For example, if a model sees a cup on a table and then the cup is not in image, the model prefers a removal explanation for the absence of the cup rather than an unexplained absence. The model uses the model’s memory of what happened before the removal to make better predictions about object relationships.

In another approach to the problem, in the paper Behrens et al. [2] addressed the problem using their “Lost and Found” framework to learn a solution that tracks objects over time from first-person views. Instead of static scenes, objects are moved between views. Such a method is useful in scenarios where the environment changes during everyday human activities and objects are frequently moved around.

While visual quality remains paramount, what ultimately matters most is how well a model can function as scene understanding is pushed to ever larger configurations at ever longer time scales. To this end, the authors in [23] introduce a compressor targeted specifically at scene graphs of point cloud data that can be used to efficiently store extended tracking information while preserving crucial information about the layout of the scene.

2.5 Scene Graphs for Task and Manipulation Planning

While it is common to regard scene graphs as facilitating 3D perception, they can actually have a considerable impact on the robot’s decisions, influencing it to perform specific tasks. The organization of scene elements within the scene graph structure allows the robot to utilize the graph to guide its actions.

In their paper, Jiao et al. [10] investigate the generation of manipulation actions by starting with a cup on a flat table, where objects and their interactions are represented as a contact structure (a graph) and analyzed with respect to their changes, which are then translated into physical robot movements to mimic common manipulation

tasks. The corresponding operations, however, are limited by the particular setup of objects and their interactions. Abstract planning is concretized.

In our previous work, we explored modeling beliefs in a POMDP framework by representing them using a structured representation, i.e., scene graphs (Amiri et al. [1]). By separately tracking known objects for which the robot has prior knowledge about their locations, and unknown objects that the robot encountered during run-time visual searches, we were able to significantly improve the robot’s performance in visual searches.

Starting from structured knowledge, Ni et al. [18] proposed the GRID framework, leveraging scene graphs to grasp entire scenes without relying on pixel data. Because it parses instructions through relational structures, tasks are broken down more naturally. With graphs as input, reasoning shifts toward how objects connect - bypassing low-level visual details altogether.

Starting with visual structures, Dai et al. [4] linked scene graphs to large language models for planning tasks. Instead of relying solely on raw text, the model takes a written summary of objects and relationships in a scene. Because it processes this structured information, the system produces step-by-step actions aimed at reaching an objective. Unlike many existing methods that are inspired by physical scenarios and utilize 3D computer vision, their system can be solved by simply using textual reasoning and transforming the input to a graph-like structure, which the language model can then process.

2.6 Multi-modal Understanding

A different strategy to fusing information from different input sources is to build input data into an existing scene graph representation. In existing work, a robot uses scene graphs for Visual Question Answering (VQA). RobotVQA [11] by Kenfach et al. uses a scene graph representation of an image to answer questions such as “what is next to the laptop?” and generates answers for such questions. We investigate how machines do not only see but also process the view they get.

In SGGNet2 [12], Kim et al. investigate the connection between speech comprehension and visual scene structure for voice-controlled steering of agents. For example, when commanded with “go to the red chair”, the network first builds a scene graph and searches for corresponding objects from linguistic words as opposed to processing words in isolation and neglecting word relationships. The network can accurately fulfill spoken commands for more realistic scenarios.

These papers show not just organization of visual data but how scene graphs are diverse in different ways of structure, building and etc.

2.7 Humans in the Scene Graph

Still, many scene graph models view surroundings as fixed items only. People show up now and then - just frames on a screen or dots on a map. To allow robots to move safely around humans, more depth matters. A dot won’t tell you if someone is moving, reaching, or about to step forward.

Close-range human pose and hand movement monitoring was tested successfully by Erdemir et al. [5] during direct hand-to-hand robotic interaction. Because individuals operate near machines, detailed body tracking becomes possible - this precision supports embedding human actions into a dynamic scene structure.

Starting from object positions, Yang et al. [28] introduced SRI-Graph, a method capturing how the robot’s end-effector connects with surrounding items. Because it maps these links clearly, detecting interactions in shared workspaces becomes more accurate. Awareness grows - not just of things nearby but also of humans - leading to smoother coordination during joint tasks. This feature affects the safety of the action for the user, allowing them to move differently depending on distance.

It’s often assumed that background props are the only things in the scene graph, but by also having people modeled with pose tracking and things like hand location, you can get some really safe and teamwork features for users who are also working in the scene.

2.8 LLM-Guided Robot Control

Recent results in robotics show that Large Language Models can be used for task planning of a robot. For this purpose, the models are trained on large amounts of text data to learn to recognize and reproduce human language. The task, given by a human operator as a spoken instruction, can then be directly transformed into an executable code for the robot’s controller. The generated code even follows the step-wise and transparent structure of the human intentions, following the logic of how a human would instruct a robot. With just a few words, the models are able to recognize the operator’s intention behind the spoken words and can generate a command for the robot that corresponds to the given instruction. These guiding principles are useful for a number of reasons. First, these principles are organized in a logical and practical way. Second, these principles are nuanced, flexible, and therefore more contextual than codified rules.

Tulbure et al. [24] studied how large language models can be used to support language-based task handover synchronization to support reasoning to generate responses to user speech. Models were trained to generate robot speech that corresponds to spoken user requests based on a general task decomposition plus specific actions. What stands out is how meaning gets preserved across communication layers. The method connects natural language input to physical output through structured planning steps. Instead of treating words and movements separately, it treats them as parts of one flow. This link becomes stronger when context guides both interpretation and behavior. A shift happens once instructions shape not only goals but also the way they are carried out.

Liu et al. [15] proposed a mixed approach: human operators show actions, guiding robotic skill acquisition. Following such input, the large language model integrates learned behaviors into future plans - gradually broadening what the machine attempts. While learning unfolds incrementally, each demonstrated example adds nuance to decision pathways. Over repeated interactions, capability grows without explicit reprogramming. As experience accumulates, previously unseen tasks become

manageable through adapted reasoning.

When robots gain more experience, their knowledge base expands. Hua et al. [9] demonstrated that such growing databases work alongside large language models to support flexible planning. With richer information available, the model adapts its choices over time. Instead of fixed rules, reasoning evolves as data accumulates.

Starting with spoken input, Wang et al. [25] explored how large language models create control plans, whereas Zahedifar [29] and team took a different path - teaching those same models to act like smart regulators. Their approach lets robots shift actions instantly when surroundings change, relying on live sensory data instead of fixed rules.

LLMs work better when they have as much organized details about their environment as possible. What makes a scene graph useful is its ability to map out real, specific facts: objects present, locations, and connections among them. Though not perfect, such frameworks anchor language models in tangible data instead of vague assumptions.

2.9 Summary and Gap

Over time, research shows a distinct shift. From fixed layouts based on shapes, scene graphs now adapt using flexible vocabularies to reflect meaning, position, while tracking changes across moments. Instead of relying solely on predefined logic flows, planning increasingly draws from large language model insights tied to organized environmental knowledge. Once limited to known categories, perception today manages unfamiliar items through vocabulary-aware detection frameworks.

A lingering shortfall still exists. Despite progress, no current approach brings together dynamic open-vocabulary 3D scene graphs, task planning driven by large language models, along with detailed human behavior representation - woven into one cohesive setup for a stationary collaborative robot working alongside humans. What came before is outlined in Table 2.1, setting this effort apart through integration where others fell short.

Table 2.1: Related work comparison.

System	Dyn.	OV	LLM	Human	Fixed
Kimera-DSG [22]	+				
ConceptGraphs [7]		+			
DovSG [27]	+	+	+		
Honerkamp et al. [8]	+	+			
GRID [18]			+		
Dai et al. [4]			+		
SRI-Graph [28]				+	+
Ours	+	+	+	+	+

A new approach brings together all five functions within one system, based on a Franka Research 3 robot arm. Structure emerges when each ability connects through unified design. Not separate parts, but linked elements shape the whole. Through integration, function follows form in measurable ways. The setup allows interaction without dependency overload. Each piece works alongside others, not in isolation. Framework behavior shifts with input patterns. Physical execution matches planning steps closely. Results show coherence across tasks.

Chapter 3

Methodology

This section presents experiences gained from constructing and evaluating the DSG-HRI framework. It explains the four layers and their interaction, particularly how the scene graph is used as shared memory among the layers. The scene graph is used as a dynamic physical scene model that is updated based on sensor input and contains relevant situational information for the planning component. At the same time, it receives information about physical constraints from the motor control component. Four levels make up its architecture: sensing surroundings, forming dynamic scene graphs, using large language models for planning actions, then carrying out physical tasks. At its core lies a scene graph - functioning as shared memory across components. Updates flow into it from sensory input; meanwhile, it supplies situational awareness to the reasoning module while signaling environmental conditions to the motor control stage.

The framework receives a verbal or text command, which is processed by the language model running on a GPU workstation alongside the perception system. The language model first interprets the command in the context of the visual scene (represented by the scene graph), and then generates a sequence of simple actions that need to be performed (translated into commands sent to the control interface on the real-time PC). As the scene changes during execution (captured by updates to the scene graph), the language model can generate a new sequence of actions.

3.1 System Architecture

3.1.1 Hardware Setup

All visual processing, depth estimation, object tracking, scene graph maintenance, and language-model-based decision making are performed on a single computer by the perception code, utilizing an NVIDIA RTX 2080 Ti GPU with 11 GB of video memory. The perception code communicates with a second computer over the LAN via ROS 2 Ethernet connections. The second computer uses these signals to control the motors of the arm with the timing precision required for safe physical interaction. The controller code on the second computer drives the Franka Research 3 arm using the Cartesian impedance algorithm through the libfranka library. Figure 3-1 illustrates how the two computers partition the tasks. The two machines communicate continuously, and the system runs in a fully distributed manner without a central coordinator.

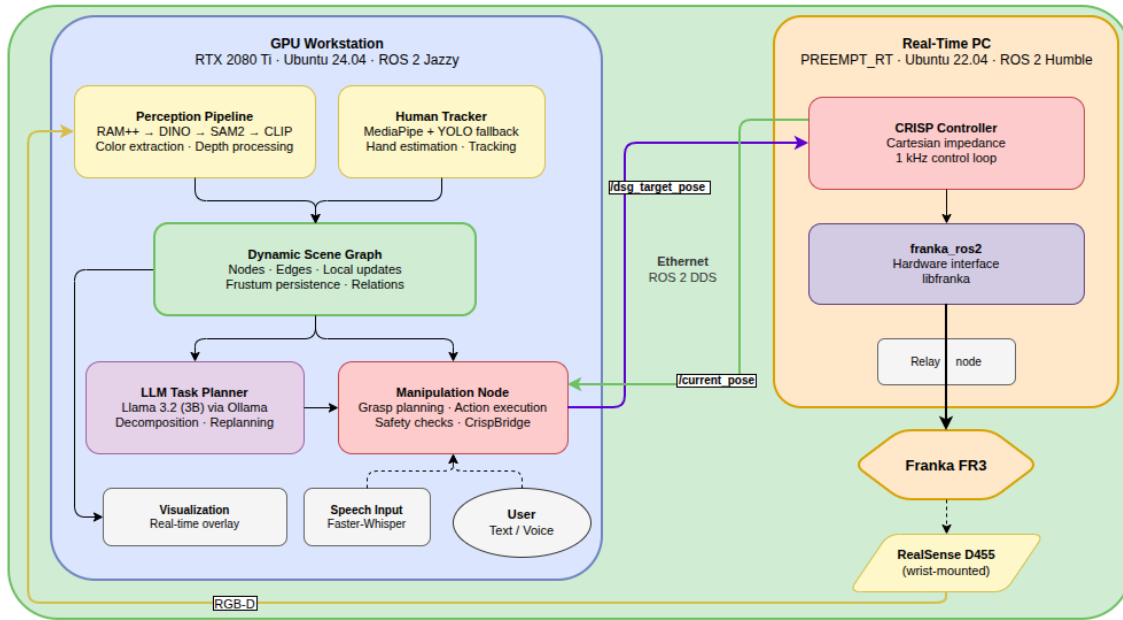


Figure 3-1: System architecture.

The Intel RealSense D455 RGB-D camera is mounted on the robot's wrist. In this configuration the robot can see objects close to the tool tip as well as view them from different angles as the arm moves through space. The camera captures synchronised colour and depth information at a resolution of 640×480 pixels. Because

the camera viewpoint shifts continuously as the arm moves, additional logic is needed within the scene graph to handle objects that temporarily leave the field of view (see Section 3.3.5). Mobility of the camera allows coverage from multiple viewpoints, but it also means that no single view captures the full workspace at once.

3.1.2 Communication Architecture

The machine-to-machine interaction is based on standard ROS 2 DDS topics published and subscribed via CycloneDDS. Only three streams of data are sent over the network.

- `/dsg_target_pose` (GPU→RT): Cartesian target
- `/current_pose` (RT→GPU): EE position feedback
- `/dsg/robot_command` (GPU→RT): Gripper open/close commands

The GPU workstation runs ROS 2 Jazzy, while the RT-PC runs ROS 2 Humble on a `PREEMPT_RT` kernel. Standard message types such as `PoseStamped` and `String` are transferable between the two versions, but some types require special handling. A minimal relay node on the RT-PC forwards the target pose topic from Jazzy into the Humble workspace.

Figure 3-2 illustrates the full ROS 2 node-topic structure, including connections within a single machine as well as those that span both machines. It shows all message flows between nodes, including topic-based publish/subscribe relationships and service-based direct calls.

3.1.3 Software Stack

The GPU workstation runs a conda-managed ROS 2 Jazzy environment provided by RoboStack, which allows the large language model (Ollama) and the perception models to run locally on the same machine. Most of the perception models are CUDA-accelerated and implemented in PyTorch. Table 3.1 lists the core software components of the framework.

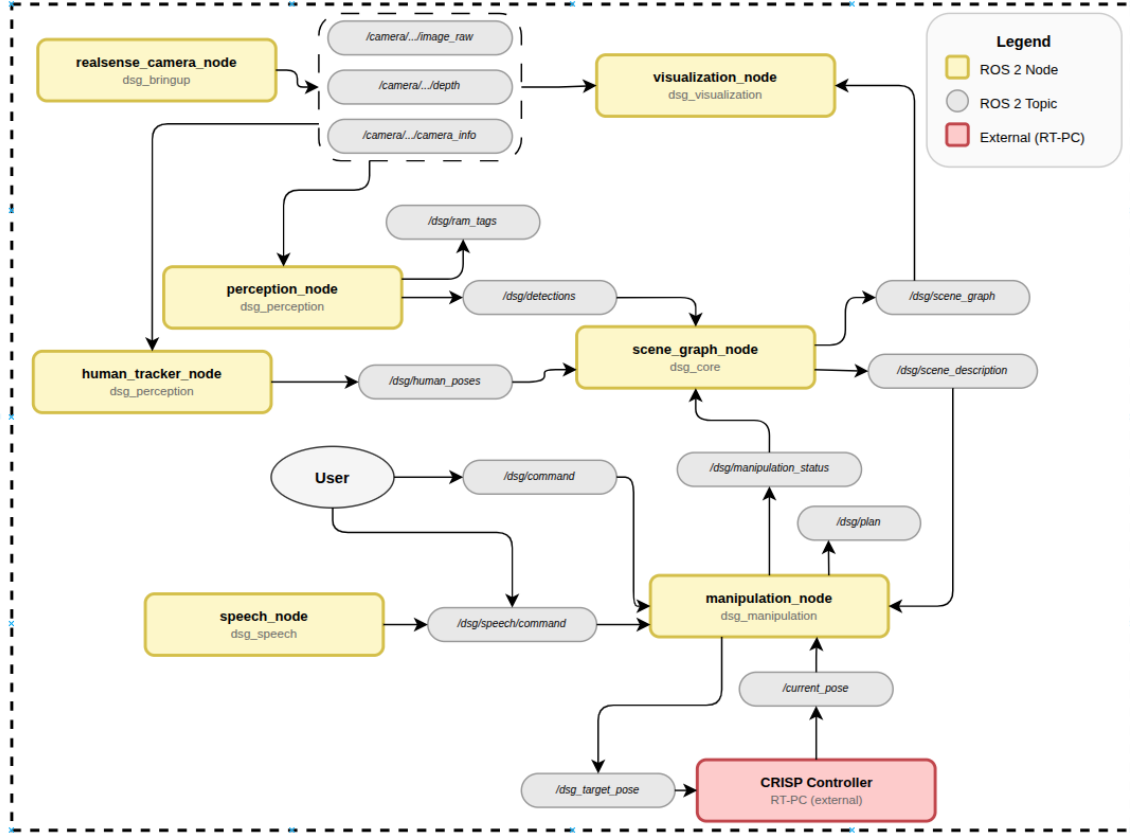


Figure 3-2: ROS 2 node-topic graph.

Table 3.1: Key software components of the DSG-HRI framework.

Component	Technology	Runs on
Object detection	RAM++ + Grounding DINO + SAM2 + CLIP	GPU PC
Human tracking	MediaPipe PoseLandmarker + YOLO-Pose	GPU PC
Scene graph	Custom Python (ROS 2 node)	GPU PC
Task planning	Llama 3.1 (8B) via Ollama	GPU PC
Robot control	CRISP impedance controller	RT-PC
Hardware interface	libfranka + franka_ros2	RT-PC
Communication	ROS 2 DDS (CycloneDDS)	Both

3.2 Perception Pipeline

A single image frame passes through several stages of analysis in the perception pipeline. Despite the noise introduced at each step, accurate object identifications are produced in the end. Because the pipeline is open-vocabulary, objects are recognised without being part of a predefined label set, and the system can handle categories that were not seen during training.

3.2.1 Open-Vocabulary Object Detection

A sequence of four steps handles detection, as shown in Figure 3-3.

1. **Image tagging.** Each RGB frame is analyzed through Recognize Anything Plus (RAM++) [30], producing a collection of potential object labels. Instead of relying on predefined categories, RAM++ matches visual data to semantic concepts learned during training. Because its vocabulary isn't locked, the model detects diverse objects based on patterns seen before.
2. **Grounded detection.** RAM++ outputs feed into Grounding DINO [16] as textual cues. Because these inputs guide the process, bounding boxes emerge aligned with described concepts. While one system recognizes terms widely, the other localizes them accurately in space.
3. **Segmentation.** Pixel-level masks emerge after refining bounding boxes through SAM2 [20]. Though compact, the small version holds up well under a strict 11 GB VRAM limit without sacrificing clarity.
4. **Feature extraction.** CLIP [19] generates semantic embedding vectors from separated image areas. For every object, two such embeddings emerge - one based on the bounded crop, another on the clean mask with background stripped away. Merging them happens through a weighted addition, borrowing the method used in HOV-SG [26].

Some practical adjustments were made:

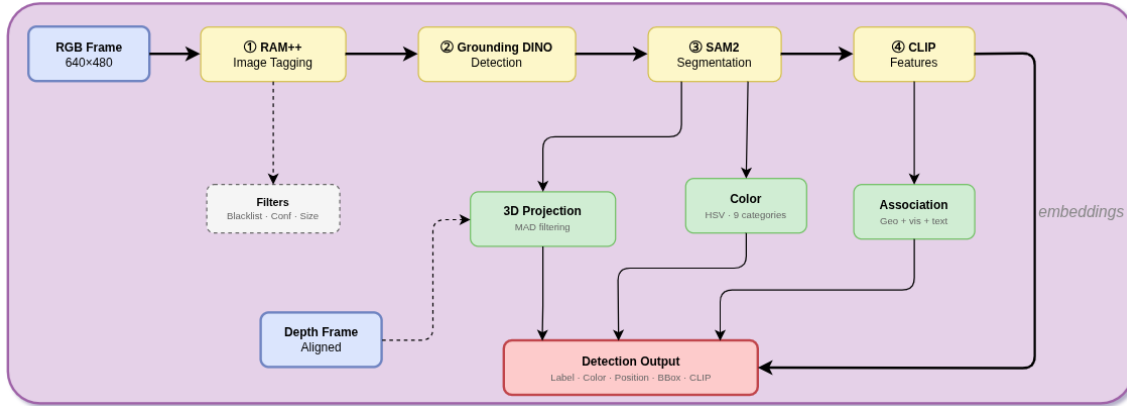


Figure 3-3: Open-vocabulary perception pipeline.

- **Label blacklist.** Some terms that were too vague or concepts that you can't grab like 'shadow', so they were filtered
- **Per-category confidence thresholds.** Items meant to be held need stronger evidence before counting them real, with grasp targets demanding clarity above 0.25. Background surfaces accept lower certainty.
- **Bounding box filtering.** Detections under twenty pixels across won't pass. Large boxes meet suspicion too - if a frame covers nearly the full view, it's likely wrong.
- **Frame subsampling.** Only every second image goes through analysis by default, which runs at about five detections each second using an RTX 2080 Ti. Though efficiency drops slightly, timing stays consistent across operations.

3.2.2 Object Color Extraction

The system first detects the objects in the image, then extracts the dominant colour of each detected object. The extracted colour is later compared against colour terms in user commands (e.g., “the red cup”) to identify the correct target. When a segmentation mask is available, the pixels inside the mask are used directly. When only a bounding box is available, the central 60% of the box is used to reduce contamination from the background. Because colour extraction depends on accurate object

localisation, high detection accuracy is important for obtaining a clean dominant hue.

The mean pixel value is converted from RGB to HSV colour space (using OpenCV’s ranges). Classification assigns one of nine colour labels based on the HSV values according to the rules shown in Equation 3.1. The same rules are applied to the colours of human clothing to support natural-language references such as “the person in the black jacket”.

$$\text{color} = \begin{cases} \text{black} & \text{if } S < 50 \text{ and } V < 80 \\ \text{white} & \text{if } S < 50 \text{ and } V > 200 \\ \text{gray} & \text{if } S < 50 \\ \text{red} & \text{if } H < 10 \text{ or } H > 170 \\ \text{orange} & \text{if } 10 \leq H < 25 \\ \text{yellow} & \text{if } 25 \leq H < 35 \\ \text{green} & \text{if } 35 \leq H < 85 \\ \text{blue} & \text{if } 85 \leq H < 130 \\ \text{purple} & \text{if } 130 \leq H < 170 \end{cases} \quad (3.1)$$

3.2.3 Object-Level 3D Reconstruction

Each detected 2D region is lifted into 3D space using a depth value and the camera intrinsic matrix K . The centre pixel of the region and its depth are transformed into world coordinates using Equation 3.2, which applies the pinhole camera model with focal lengths (f_x, f_y) and principal point (c_x, c_y) .

$$X = \frac{(u - c_x) \cdot d}{f_x}, \quad Y = \frac{(v - c_y) \cdot d}{f_y}, \quad Z = d \quad (3.2)$$

An outlier filter based on Median Absolute Deviation (MAD) is applied to the depth values inside the segmentation mask. Pixels whose depth falls outside two MADs of the median are discarded, and the median of the remaining pixels is reported as the object’s depth. This filter effectively removes errors caused by object

edges, specular highlights, and transparent surfaces. While achieving similar noise suppression it’s cheaper to compute than clustering filters.

$$\tilde{d} = \text{median}(d_1, \dots, d_n) \quad (3.3)$$

$$\text{MAD} = \text{median}(|d_i - \tilde{d}|) \quad (3.4)$$

The overall confidence score is computed from detector confidence at 0.7 weight and depth-reliability score at 0.3 weight. The depth-reliability score is the fraction of pixels inside the object mask that return valid depth values.

$$c_{\text{final}} = 0.7 \cdot c_{\text{detector}} + 0.3 \cdot c_{\text{depth}} \quad (3.5)$$

Each detected object stores a set of properties: a 3D axis-aligned bounding box with centroid position, learned CLIP visual features, a detection confidence score, a dominant colour label, and a size category (small, medium, or large) based on bounding box volume. Together, these properties define an object node in the scene graph.

3.2.4 Object Association

When a new object is detected, the system checks whether it has been seen before by computing a unified similarity value across several object features:

$$s(i, j) = w_g \cdot s_{\text{geo}}(i, j) + w_v \cdot s_{\text{vis}}(i, j) + w_t \cdot s_{\text{text}}(i, j) \quad (3.6)$$

where:

- $s_{\text{geo}}(i, j) = 1 - \frac{\|\mathbf{p}_i - \mathbf{p}_j\|}{d_{\text{max}}}$ is the geometric similarity, where values drop as separation grows beyond $d_{\text{max}} = 0.3 \text{ m}$.
- $s_{\text{vis}}(i, j) = \frac{1}{2} (\mathbf{f}_i^{\text{vis}} \cdot \mathbf{f}_j^{\text{vis}} + 1)$ is the normalized cosine similarity of CLIP visual features.

- $s_{\text{text}}(i, j) = \frac{1}{2} (\mathbf{f}_i^{\text{text}} \cdot \mathbf{f}_j^{\text{text}} + 1)$ is the normalized cosine similarity of CLIP text features.

If the highest similarity score exceeds 0.6, the new detection is associated with the existing object, and the object’s features are updated using a running average. If no existing object exceeds the threshold, a new object node is created in the scene graph.

3.2.5 Tracking People by Their Movements

Human tracking operates on standard RGB images using MediaPipe PoseLandmarker. Up to 33 landmarks are detected per person, including fingertips and wrists. When a person is occluded by an object, too far from the camera, or only partially visible, the system falls back to YOLO-Pose (YOLO11n-pose). YOLO-Pose handles multiple people well using 17 COCO keypoints, but provides less detail around the hands compared to MediaPipe.

Because a single-pixel shift in the 2D keypoint can produce very different 3D coordinates, each 2D point is averaged over a 5×5 depth window before being projected to 3D using Equation 3.2. The body centre is computed as the average of the 3D positions of the shoulders and hips.

Strong Hand Location Tracking

When a person is holding an object, the wrist is often obscured by the object itself from the camera’s point of view. To recover the hand position reliably, the system uses a multi-stage fallback strategy:

1. **Direct wrist measurement.** Pull depth straight from the wrist marker - if there and clear, nothing beats that precision.
2. **Fingertip averaging.** Should that spot be blocked, shift instead to tips of fingers seen, blend their 3D spots into one estimate.

3. **Arm extrapolation.** When even fingertips lack data, reach backward along the limb - elbow and shoulder guide a guess on arm direction to place the hand:

$$\mathbf{p}_{\text{hand}} = \mathbf{p}_{\text{elbow}} + 0.75 \cdot (\mathbf{p}_{\text{elbow}} - \mathbf{p}_{\text{shoulder}}) \quad (3.7)$$

All guesses go through a filter that removes any data past 3m depth, 2m shift, or 1.2m separation from mid-body. This is to prevent false HOLDING signals that would result from stretch errors if they had free reign to place hands wherever.

Persistent Tracking

We assign to each person a single tracking ID over time and depth and assume that people do not change over time unless obscured temporarily by other people. People are linked between images because they are spatially close, and a prior label is reused if the depth position change is less than half a meter (less than 150 pixels). New labels are created when there are no close matches. A track is removed from consideration after thirty frames or approximately three seconds at ten fps.

$$d_{\text{track}} = \begin{cases} \|\mathbf{p}_i^{3D} - \mathbf{p}_j^{3D}\| & \text{if 3D positions available} \\ \frac{\|\mathbf{c}_i^{2D} - \mathbf{c}_j^{2D}\|}{d_{\text{max}}^{2D}} \cdot d_{\text{max}}^{3D} & \text{otherwise (bbox center)} \end{cases} \quad (3.8)$$

Phantom Detection Filtering

Real-world testing revealed frequent phantom human detections — false positives triggered by objects, shadows, or camera artifacts. Multiple filtering stages address this:

- **Confidence filter:** Human detections below 0.5 confidence are rejected.
- **Distance filter:** Humans detected closer than 0.3 m or farther than 5.0 m from the robot base are rejected, as are those with $Z < -0.5$ m or $Z > 2.5$ m.
- **Consecutive frame confirmation:** A human must be detected in 8 consecutive frames before being promoted from TENTATIVE to CONFIRMED.

- **Object position filter:** Objects detected at physically impossible positions — $Z < -0.05$ m (below table), $Z > 2.0$ m, or $XY > 2.0$ m from base — are rejected.

Confirmed humans remain visible in the scene graph for 180 seconds after the last detection, and in the manipulation node’s spatial memory for 300 seconds. This prevents the human from being forgotten during a long pick sequence that precedes a handover action.

3.3 Interactive 3D World Map

3.3.1 Graph Structure

Entities in the scene graph are connected based on their roles. Four types of entities are represented as nodes:

- **Object nodes** store labels, hue, location, axis-aligned boundaries, CLIP-based features, certainty scores, approximate scale class, whether they shift when moved, and if part of a flat area.
- **Human nodes** store body points, coordinates for hands, and ID.
- **Robot node** stores the data from the robot, its movement and gripper position.
- **Region nodes** stores surface description objects.

All seven types of edges are grouped into three categories:

- **Spatial:** ON, NEAR, NEXT_TO — geometry shapes these relationships, measuring box positions.
- **Semantic:** INSIDE, BELONG — containment and overlap checks determine membership.
- **Interaction:** HOLDING, REACHING_FOR — hand proximity signals contact or intent.

Every edge carries a confidence value and, when applicable, additional metadata such as which hand is involved in a HOLDING relation. Figure 3-4 shows an example scene graph linking objects, a person, and the robot through spatial and interaction edges.

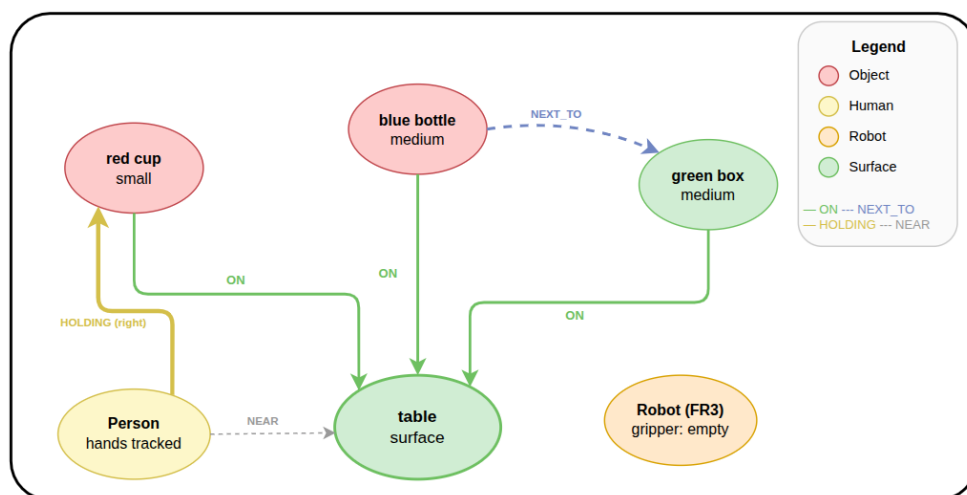


Figure 3-4: Example scene graph.

3.3.2 Scene description for language model

To disambiguate between similar-shape objects such as a red cup and a blue cup, colour information is embedded directly in the textual scene description. The description is generated on the fly from the scene graph and passed as a plain-text block to the LLM planner (Listing 3.1). Rather than separate fields, the block combines object positions, sizes, relations, and the current robot status. Because object sizes are included, the planner can determine whether an object fits within the gripper’s 8 cm aperture. Hand positions help the model infer which person is reaching for which item. Including the robot’s current gripper state prevents the planner from suggesting actions that are physically impossible, such as picking up another object while one is already held.

```

1      Scene Graph:
2
3      Objects:

```

```

4         - red cup (id: obj_0001, pos: (0.40, 0.10, 0.30),
5           size: 8x8x12cm, category: small, conf: 0.92)
6         - blue bottle (id: obj_0002, pos: (0.50, -0.15,
7           0.35),
8           size: 7x25x7cm, category: medium)
9
10        Humans:
11        - Person (id: human_0, clothing: black,
12          pos: (1.00, 0.00, 0.80),
13          left_hand: (0.75, 0.15, 0.45),
14          right_hand: (0.70, -0.10, 0.50))
15
16        Relations:
17        - cup(obj_0001) --on--> table(obj_0003)
18        - person(human_0) --reaching_for--> cup(obj_0001)
19
20        Robot (Franka FR3):
21        EE: (0.30, 0.00, 0.50)
22        Gripper: empty
23        Status: idle

```

Listing 3.1: Example scene description sent to the LLM planner.

3.3.3 Relation Detection

Box-shaped boundaries line up with coordinate axes to calculate spatial connections, borrowing techniques from DovSG’s point data approach [27]. The adjustable limits for each type of relationship appear in Listing 3.2.

```

1      class SpatialRelationDetector:
2          def __init__(self,
3              on_footprint_overlap_min = 0.15,    # 15%
4              on_height_threshold      = 0.25,    # 25 cm
5              belong_intersection_min  = 0.20,    # 20%
6              inside_margin            = 0.02,    # 2 cm
7              next_to_threshold        = 0.45,    # 45 cm

```

```

8         near_threshold           = 1.0,      # 1 m
9         holding_hand_threshold    = 0.35,     # 35 cm
10        holding_body_threshold    = 0.60,     # 60 cm
11        reaching_threshold         = 0.80,     # 80 cm
12    ): ...

```

Listing 3.2: Relation detection thresholds.

ON Relation

One object lies atop another under the ON relation. Meeting detection needs hinges on spatial alignment. Overlapping footprints appear when views flatten to ground level. More than 15 percent coverage fulfills initial placement rules. Vertical order matters just as much. Above means higher elevation, yet allows a gap near 25 centimeters. Size comparison steers which item sits on top. Position confirms support direction between pair members. Height together with how well regions match influences the score. If there are no bounding boxes, the method shifts to measuring spacing between centers, applying a limit of 40 cm.

$$A_{\text{overlap}} = \max(0, x_{\text{max}}^{\cap} - x_{\text{min}}^{\cap}) \cdot \max(0, z_{\text{max}}^{\cap} - z_{\text{min}}^{\cap}) \quad (3.9)$$

$$r_{\text{overlap}} = \frac{A_{\text{overlap}}}{\min(A_{\text{foot}}^A, A_{\text{foot}}^B)} \quad (3.10)$$

$$c_{\text{on}} = 0.4 \cdot \left(1 - \frac{|\Delta y|}{0.25}\right) + 0.6 \cdot \min\left(1, \frac{r_{\text{overlap}}}{0.5}\right) \quad (3.11)$$

INSIDE Relation

An object qualifies as INSIDE another if it lies fully enclosed within the boundaries of a second. Testing happens by checking whether the small item’s midpoint sits inside the axis-aligned bounding box of the larger, minus a 2 cm buffer along each edge. This check runs solely when the outer entity has a label matching known containers - terms like box, bin, basket, drawer, bowl, cup, or mug. Matching relies on full-word

patterns, so partial overlaps do not trigger results. Words such as “cupboard” or “keyboard” get left out even though they contain valid substrings, avoiding incorrect classifications. Only exact semantic fits undergo evaluation for containment status.

$$\text{INSIDE}(A, B) = \begin{cases} \text{true} & \text{if } \mathbf{p}_A \in [\mathbf{b}_{\min}^B - m, \mathbf{b}_{\max}^B + m] \\ \text{false} & \text{otherwise} \end{cases} \quad (3.12)$$

BELONG Relation

What ties an object to another often comes down to space it occupies inside something else. A handle fits into a drawer, not just near it - overlap matters. When one item sits within another, their 3D boxes might cross paths. Volume overlap becomes key - if more than fifth of the smaller piece lies within the bigger one, it counts as part of it. Items typically seen as components get first attention: knobs, lids, buttons, handles. Recognition leans on both shape fit and common function. Priority shifts toward those expected to be segments rather than separate things. Spatial inclusion defines membership here - not names, but position. Decisions rely on measurable space share, not assumptions about design.

$$V_{\text{int}} = \prod_{k \in \{x, y, z\}} \max(0, \min(a_{\max}^k, b_{\max}^k) - \max(a_{\min}^k, b_{\min}^k)) \quad (3.13)$$

HOLDING and REACHING_FOR Relations

A person linked to an item they grip defines the HOLDING connection. Distance from either hand to the object’s middle point decides the label. Less than 35 centimeters marks it as HOLDING. Between 35 and 80 centimeters counts as REACHING_FOR. Where hand data is missing, spacing from the body’s midpoint applies instead. This uses a cutoff at 60 centimeters. Which hand sits nearer - left or right - is noted in the data. As separation grows, certainty drops at a steady pace.

$$d_{\text{hand}} = \min(\|\mathbf{p}_{\text{left}} - \mathbf{p}_{\text{obj}}\|, \|\mathbf{p}_{\text{right}} - \mathbf{p}_{\text{obj}}\|) \quad (3.14)$$

$$\text{relation} = \begin{cases} \text{HOLDING} & \text{if } d_{\text{hand}} < 0.35 \text{ m} \\ \text{REACHING_FOR} & \text{if } 0.35 \text{ m} \leq d_{\text{hand}} < 0.80 \text{ m} \\ \text{none} & \text{otherwise} \end{cases} \quad (3.15)$$

$$c = 1 - \frac{d_{\text{hand}}}{\delta} \quad (3.16)$$

Proximity Relations

When items sit on one plane, their vertical positions guide clustering - matching Y-values within 8 centimeters trigger grouping via an algorithm designed for flat layouts. Horizontal separation alone defines adjacency among those sharing a surface; if less than 45 centimeters apart along the X-axis, they count as next to each other, sidestepping distortions caused by imprecise depth readings. Items resting at different heights rely instead on complete three-dimensional spacing to judge closeness - a meter or under makes them near. Distance here comes straight from standard spatial math, unaffected by alignment assumptions.

3.3.4 Local Update Mechanism

Rather than recomputing every edge on every update, the system performs local updates: only the edges connected to a modified node and its immediate neighbours are recomputed. This keeps the scene graph update cost bounded as the scene grows.

Also while updating it includes node types that connect the human and object node. If one of them is modified both will be recomputed to avoid these issues:

1. Human pose is updated and adds new human node but leaving updated object.
2. Relation detector doesn't see objects during scan.
3. HOLDING is undetected.

That is the reason why both types are computed together. The additional cost is

modest because most scenes contain fewer than twenty objects and only one or two humans.

3.3.5 Frustum-Aware Persistence

Because the camera is mounted on the wrist, the field of view changes as the arm moves. Objects may briefly disappear from the camera’s view during a motion, but this does not mean they have been removed from the scene.

From the camera intrinsics, image size, and depth range (0.15–4.0 m), the system computes a view frustum. When an object is not detected in a frame, its 3D position is tested against the frustum. If the object lies inside the frustum (i.e., it should have been visible), its tracking state advances through the transitions shown in Figure 3-5. If the object lies outside the frustum, no penalty is applied, and the object retains its current confidence level.

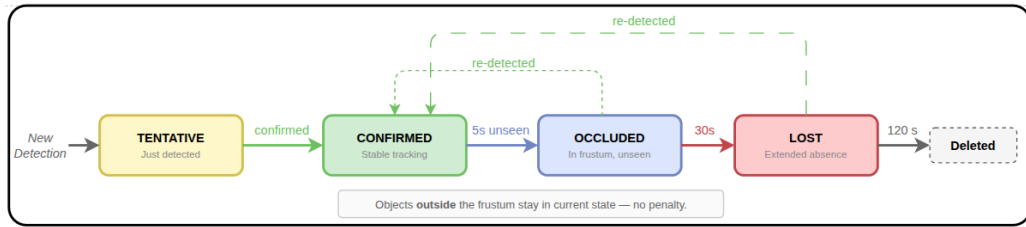


Figure 3-5: Node state machine for frustum-aware persistence.

A node must meet certain criteria before being promoted from TENTATIVE to CONFIRMED, and a separate set of timeouts governs the demotion back from CONFIRMED to LOST states (the robot node is always CONFIRMED). The time thresholds that control these transitions are shown in Listing 3.3.

```

1      class TrackingConfig:
2          confirmation_count = 8      # detections to
           confirm
3          occluded_timeout   = 5.0    # seconds ->
           OCCLUDED
4          lost_timeout       = 30.0   # seconds -> LOST
5          delete_timeout     = 180.0  # seconds -> deleted
           (humans)

```

Listing 3.3: Node tracking configuration.

3.3.6 Workspace Scan Routine

The system starts from scratch when it has no knowledge of the environment. Since the wrist-mounted camera has a restricted field of view, the system has to explore the scene to construct it. The scan follows a two-phase approach. Phase 1 scans for humans using three tilted positions (approximately 75 degrees from vertical) at elevated height (0.70 m), covering left, forward, and right directions. Phase 2 performs a semi-circle sweep for objects using five overhead positions at 0.55 m height, looking straight down. The overhead positions are executed last so that their more accurate depth measurements overwrite any corrupted object positions from the tilted Phase 1 views.

To keep up with changing environment the last detection dominates with ($\alpha = 0.9$). Making seeing order a crucial detail that can change perception. The route of scanning is seen in Figure 3-6.

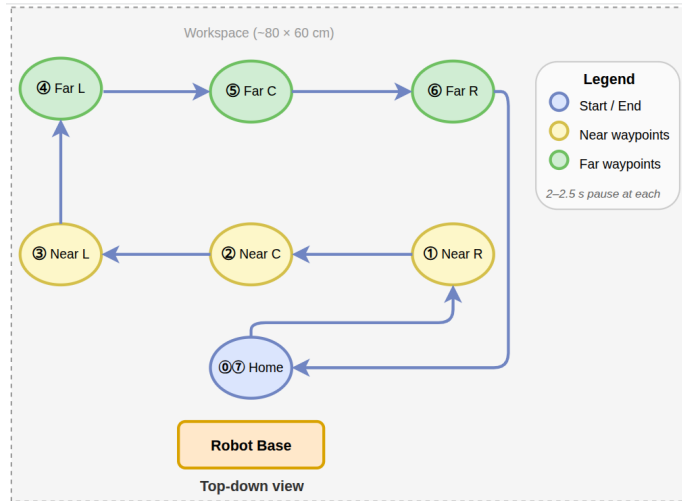


Figure 3-6: Top down view of the scan.

After the scan, the system reports the number of people and objects detected. The scan can also be re-triggered manually during operation if a major change in the

environment is noticed (e.g., someone rearranging the objects).

3.4 LLM-Based Task Planning

3.4.1 Task Decomposition

LLM processes two inputs together (command from the human and scene details). The system prompt describes the robot's capabilities and constraints:

```
1      You are a task planner for a Franka Emika Panda
2      robotic arm.
3
4      ROBOT SETUP:
5      - Fixed-base 7-DOF manipulator (cannot move its base
6        )
7      - Parallel-jaw gripper (max opening ~8 cm)
8      - RGB-D wrist camera, ~80 cm reach
9
10     AVAILABLE ACTIONS:
11     look_at, pick, place, handover, receive,
12     open_gripper, close_gripper, wait, speak
13
14     RULES:
15     1. Always look_at an object BEFORE picking it.
16     2. The robot can hold only ONE object at a time.
17     3. Use exact object labels or IDs from the scene.
18     4. If a human is HOLDING an object, cannot pick it.
19     5. If target not in scene, set "success": false.
20     6. Keep plans short.
21
22     RESPOND WITH ONLY A JSON OBJECT.
```

Listing 3.4: LLM system prompt (abbreviated).

- Fixed-base 7-DOF manipulator with 80 cm reach
- 8 cm parallel-jaw gripper (anything broader stays out of reach)

- Wrist-mounted RGB-D camera, requiring it to point toward any item before pickup
- Can hold only one object at a time
- Cannot pick objects that a human is currently HOLDING

A single setup message holds three sample cases showing how the JSON result should look - one illustrates moving an item, another shows passing something between hands, while the third captures a situation where the model stops itself from selecting an object someone else is holding. Each case lines up with the structure seen in Listing 3.5, including the restraint shown in Listing 3.6. Though brief, the demonstration guides formatting through practical instances rather than abstract rules. The blocked choice appears alongside valid actions, making boundaries clear without extra explanation.

```

1      {
2          "success": true,
3          "reasoning": "Pick cup, place on box",
4          "actions": [
5              {"action": "look_at", "target_object": "cup"},
6              {"action": "pick",    "target_object": "cup"},
7              {"action": "look_at", "target_object": "box"},
8              {"action": "place",   "target_location": "box"}
9          ]
10     }
```

Listing 3.5: Expected JSON output format.

```

1      Command: "Pick up the bottle"
2      Scene: person (human_0) HOLDING bottle (obj_0002)
3
4      {
5          "success": false,
6          "error": "Cannot pick - human_0 is holding it.",
7          "actions": [
8              {"action": "speak",
```

```

9         "target_human": "human_0",
10        "parameters": {"message":
11            "Could you put the bottle down?"}}
12    ]
13 }
```

Listing 3.6: LLM response when target is held by a human.

A JSON structure emerges from the LLM, holding basic actions. Inside, every step defines its kind, points to an item or spot, then adds a clear explanation people can follow. These steps come in forms like `look_at`, `pick`, `place`, `handover`, `receive`, `open_gripper`, `wait`, or `speak`. Usually, it runs on Llama 3.1 - locally hosted via Ollama - with eight billion parameters shaping responses. When needed, OpenAI versions stand ready instead. From spoken instruction to chain of moves, the full path appears in Figure 3-7.

Several post-processing steps improve plan quality after LLM generation:

- **Command auto-expansion.** When the user issues a handover command such as “give the tool to the person” but the robot is not currently holding anything, the system automatically prepends “pick up and” to the command. This ensures the LLM generates a complete plan including the pick step. The expansion only triggers when the command contains handover-related words and does not already include “pick.”
- **Plan target validation.** After the LLM generates a plan, a validation step checks that pick targets match the user’s command. If the LLM selects the wrong object, the validator searches the scene for objects matching the user’s words and corrects the plan automatically.
- **Redundant action removal.** If the robot is already holding an object, any `look_at` and `pick` actions targeting that object are stripped from the plan automatically.
- **Human target handling.** When `look_at` targets a human (detected by keywords like “human,” “person”), the action is automatically skipped with a suc-

cess return. The subsequent `handover` or `receive` action locates the human independently through spatial memory recall.

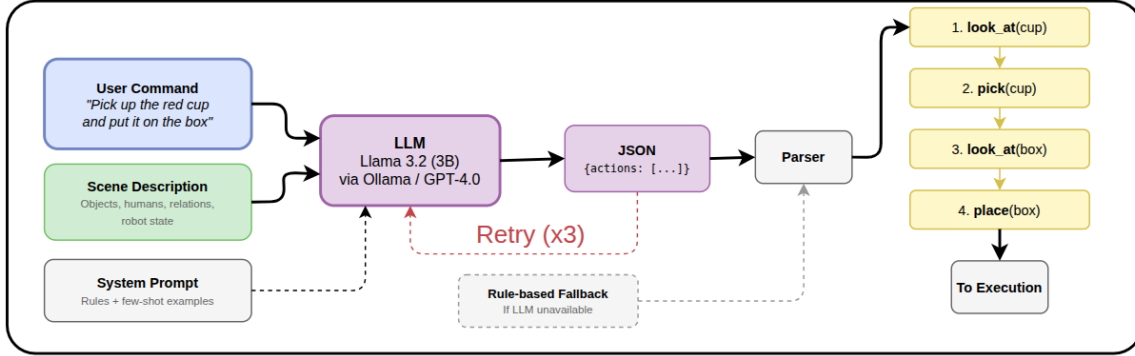


Figure 3-7: LLM planning flow.

3.4.2 Object Name Resolution

Should the LLM mention an object - like “cup,” “red cup,” or “obj_0001” - the system checks the scene graph through a layered approach:

1. **Exact ID match:** If the given name aligns precisely with a node’s identifier, that node is selected.
2. **Exact label match:** When the term fits a node’s label without variation, selection follows immediately.
3. **Substring match:** Where exact matches fail, similarity depends on how much of one string appears within another, measured by proportional length. The pairing showing highest density in overlap earns priority.

Frequent shifts in phrasing emerge naturally when the model describes items during repeated trials. Such adaptability allows alignment despite inconsistent naming patterns over multiple runs.

3.4.3 Scene-Change Replanning

Midway through complex tasks, surroundings can shift because people interact with them. Instead of relying on old data, the model checks real-time object locations next to an initial reference image. Whenever displacement exceeds five centimeters - or new items appear or vanish - a fresh plan begins automatically. Given the starting instruction, past steps performed, and the modified layout, the language model adjusts accordingly. A fresh plan emerges, shaped by the adjustments made. Shown along a sequence of moments, Figure 3-8 captures how it unfolds.

$$\Delta_i = \|\mathbf{p}_i^{\text{current}} - \mathbf{p}_i^{\text{snapshot}}\| \quad (3.17)$$

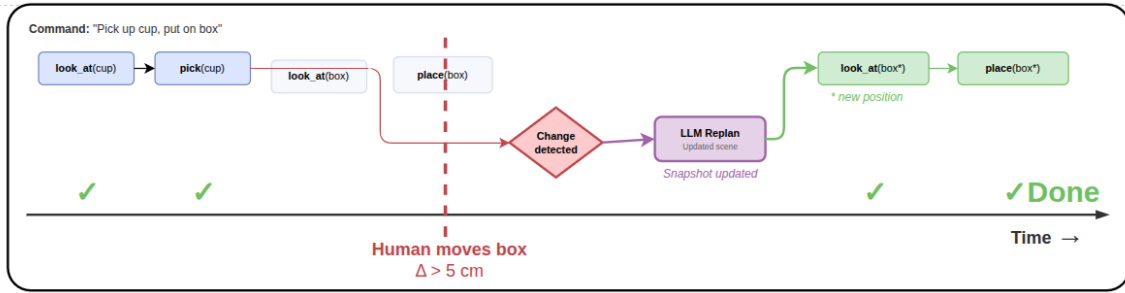


Figure 3-8: Scene-change replanning timeline.

Every time a replan occurs - no matter the outcome - the system adjusts the snapshot so identical changes do not initiate further replanning. That behavior once relied on tracking version numbers, but was phased out due to unnecessary replans caused by frequent updates to the scene graph, happening nearly five times per second. The code referenced in Listing 3.7 demonstrates how movement-driven differences are now detected.

```

1         def _scene_structurally_changed(self) -> bool:
2             current = {n['id']: n['position']}
3                 for n in self.current_scene.get('
4                     nodes', [])
5                     if n.get('type') == 'object'}
6             for obj_id, pos in current.items():
7                 if obj_id in self._snapshot:

```

```

7         old_pos = self._snapshot[obj_id]
8         disp = np.linalg.norm(pos - old_pos)
9         if disp > 0.05:  # 5 cm threshold
10             return True
11         if set(current.keys()) != set(self._snapshot.
12             keys()):
13             return True
14         return False

```

Listing 3.7: Position-based scene change detection.

3.4.4 Failure Replanning

If a step breaks while running - say, the target can’t be reached or the grip lies beyond reach - the setup feeds details to the LLM:

- The original command
- Which actions completed successfully
- Which action failed and why
- The current scene state

A different approach emerges when the LLM fails - another route takes shape. When access to the LLM drops, a backup shifts into place: rules step in. Commands like “pick up X” follow fixed logic. Placement actions, such as “put X on Y”, run through structured checks. Requests saying “give me X” move along preset paths instead.

3.4.5 Error Recovery

If Ollama drops the connection, the LLM client retries up to 3 times with exponential backoff (1 s base interval) before falling back to rule-based planning. Each call to a perception model is wrapped in exception handling with safe defaults so that a single failing frame does not stop the entire pipeline.

3.5 Manipulation and Execution

3.5.1 CRISP Compliant Controller

For the execution layer I used CRISP with Cartesian impedance controller, since it's better suit for this project to compute end-effector in cartesian space instead of calculating joint paths. When contact occurs with a human or obstacle, the arm yields smoothly rather than pushing through, giving inherent compliance without requiring external force sensing.

$$\boldsymbol{\tau} = \mathbf{J}^T [\mathbf{K}_p(\mathbf{x}_{\text{target}} - \mathbf{x}_{\text{current}}) - \mathbf{K}_d\dot{\mathbf{x}}] + \boldsymbol{\tau}_{\text{gravity}} \quad (3.18)$$

The manipulation node on the GPU workstation sends Cartesian target poses to the RT-PC. During execution, the RT-PC runs the CRISP control law at 1 kHz, producing joint-level torques for the robot. End-effector pose feedback is published back to the GPU machine to monitor execution progress.

End-effector feedback arrives at the GPU machine approximately every 50 ms and is used by the CrispBridge module to determine when the robot has converged on the commanded target. CrispBridge considers the motion complete once the positional error drops below 5 mm. If convergence takes longer than 30 s, the system logs a warning but continues execution.

$$\|\mathbf{p}_{\text{ee}} - \mathbf{p}_{\text{target}}\| < \epsilon, \quad \epsilon = 0.005 \text{ m} \quad (3.19)$$

3.5.2 Grasp Planning

Sorting items by labels, the grasp planner splits them into six groups - details appear in Table 3.2 alongside Listing 3.8. While matching occurs, category assignment follows strict labeling rules seen across both references.

```
1 GRASP_STRATEGIES = {  
2     'cylinder': {  
3         'objects': ['cup', 'mug', 'bottle', 'can'],
```

```

4         'primary_approach': TOP_DOWN,
5         'gripper_width_ratio': 1.2,
6         'grasp_height_ratio': 0.6,
7     },
8     'box': {
9         'objects': ['box', 'book', 'phone', 'remote'],
10        'primary_approach': TOP_DOWN,
11        'gripper_width_ratio': 1.1,
12    },
13    'tool': {
14        'objects': ['spoon', 'fork', 'knife'],
15        'primary_approach': TOP_DOWN,
16        'grasp_height_ratio': 0.3, # handle
17    },
18 }

```

Listing 3.8: Grasp strategy definitions.

Table 3.2: Grasp strategies by object category.

Category	Examples	Primary	Grasp height
Cylindrical	cup, bottle, can	top-down	60% of height
Box-shaped	box, phone, book	top-down	50% of height
Small	pen, key, coin	top-down	50% of height
Spherical	ball, apple, egg	top-down	50% of height
Flat	plate, paper, card	top-down	50% of height
Tool	spoon, fork, knife	top-down	30% (handle)

Although every type defines a grip opening proportion relative to the item’s width, it also sets a vertical contact point. Where the system lacks bounding box details, volume-based sizing - classified as small, medium, or large - influences standard measurements. All categories use a top-down approach, which proved more reliable than side grasps during real-robot testing.

Above the target, the gripper opens first. Following that, it shifts into position - stopping ten centimeters short of contact. Then comes downward motion until reaching the exact pickup spot. After positioning, the gripping mechanism activates by closing. Finally, an upward movement raises the object slowly at 2 cm/s through

a distance of ten centimeters.

Franka Grasp Action

The standard ROS 2 GripperCommand action caused the gripper to “bounce” after contact — briefly releasing and re-closing — which dropped objects. The solution was to use the Franka-native Grasp action (`/franka_gripper/grasp`), which maintains continuous force after contact without releasing. A custom minimal `franka_msgs` package provides the action definition on the GPU PC.

The Franka Grasp action accepts width, speed, force, and epsilon parameters.

Where it sends speed, force and width to close and keep the force at the same level. The reason why this action was implemented is because GripperCommand that controls the gripper doesn’t keep the same force that is applied and can override things to open the gripper. Because of this the handover implementation was changed to follow the new rules.

Grasp Verification

After gripper was closed it checks the current width, and if it exceeds threshold of 2mm then objects is considered as held. If it’s fully closed then grasp is empty.

3.5.3 Handover Execution

For the handover scenario, the held object is moved towards the position where the human would hold the object. For the handover sequence, when human hand positions (`left_hand`, `right_hand`) are provided in the scene graph attributes, the view controller will target the nearest hand. Otherwise, the object is placed at body center.

The handover position is calculated in the direction of the robot base and the target:

$$\mathbf{p}_{\text{handover}} = \hat{\mathbf{d}} \cdot \min(d_{\text{human}} - 0.20, 0.60) \quad (3.20)$$

The total length of the robot is 0.60 m. The minimum distance to a human is 0.20 m. The height at which objects are handed over to a human is 0.35 m. Additionally, the gripper is oriented towards a human at approximately 30 degrees. To achieve this orientation, the corresponding precomputed quaternions for the four main cardinal directions are used.

In our experiments, we kept the initial grasp force on Franka Grasp equal to the Franka Grasp set-point of 140 N. First tests to change to a softer grip via GripperCommand (5 N and then 30 N) did not succeed: the first GripperCommand made the gripper open completely and drop the object.

Unlike the approach in the previous section that follows scene graph HOLDING edges, we measure the absolute change in gripper width. Human touching or pulling the object causes its width to change by more than 2 mm above its initial value, triggering immediate gripper opening:

$$\text{release if } |w_{\text{current}} - w_{\text{initial}}| > 0.002 \text{ m} \quad (3.21)$$

After releasing the object, the robot moves up 10 cm before proceeding to the next action.

3.5.4 Receive Execution

Receive action for picking up an object is the counterpart of handover action, defined as how the robot receives an object from humans. During the receive action, the robot opens its gripper and moves forward to guide the human to put the object into the space between the robot’s fingers. As in handover, it is directed to the detected human hand if available.

For the receive movement of the gripper, the receive position is defined which is oriented towards the human similarly to the handover position. The flow proceeds as follows:

1. Open gripper fully (approximately 8 cm) and wait 1 s for the solenoid to settle.

2. Move to receiving position with gripper tilted toward the human.
3. Verify gripper is actually open (>50 mm); retry if not.
4. Record baseline width w_{baseline} .
5. Monitor for width drop >10 mm from baseline (object placed between fingers).
6. Close gripper using Franka Grasp (140 N) and wait 1.5 s for force to stabilise.
7. Verify firm grasp (width > 2 mm) and retract slowly upward.

The baseline verification step was added because the gripper occasionally reported a stale width reading (e.g., 0.4 mm) immediately after the open command, which would cause false detection. By monitoring a relative drop from the actual open width, the system avoids this failure mode.

3.5.5 Human Proximity Safety

During pick actions, the system scans for nearby hands using data pulled from the scene graph. Should a hand come closer than 10 centimeters to the current end-effector position, the pick operation pauses temporarily. A similar stop occurs when the body center enters a zone within 15 centimeters. Safety checks run only during pick actions — not during `look_at`, `place`, or `handover`, where human proximity is either irrelevant or intentional. Once space opens up again, the pick resumes.

3.5.6 Speech Input

Sometimes, a speech module runs alongside the system, turning spoken words into text instantly using microphones. When sound becomes visible language, specific terms catch attention - those tied to actions robots understand. Once spotted, these meaningful phrases flow into the `/dsg/command` channel just like typed messages do. This way, whether written or said, every instruction follows an identical path forward.

3.5.7 Visualization

Overlaid directly onto the live camera view, detection outcomes appear alongside pose data and scene relationships. Bounding regions marked with descriptive text and distinct hues highlight detected items. Human figures carry stick-figure outlines that move in sync with their actions. At times, a textual summary of the environment updates near the frame edge. Tags generated by RAM++ stay visible when active, shifting as needed. Whether an object is being moved or left still appears clearly indicated below. Pressing specific keys enables or disables each visual layer separately. Developers rely on this output while refining systems; others observe it to see capabilities firsthand.

3.5.8 Action Execution Flow

Starting with a directive like “pick up the red cup and put it on the green box,” the process unfolds step by step:

1. Input arrives through text, spoken words, or a ROS 2 message channel.
2. The manipulation module pulls an updated layout of objects from the scene graph.
3. A large language model breaks down the instruction into discrete steps: `look_at(cup)`, `pick(cup)`, then `look_at(box)`, `place(box)`.
4. Before acting, position data is recorded so shifts in state can later be identified.
5. Each time an action begins, a check runs for shifts in the environment - triggering replanning when detected - or nearby people, which causes a temporary stop.
6. Every step becomes a set of spatial coordinates sent out through CRISP.
7. CrispBridge waits until alignment is achieved prior to advancing further.
8. Once everything finishes, operation resets back to standby mode automatically.

When the box shifts during the robot’s cup pickup, the structural change detector activates between steps 2 and 3. Because of this shift, the LLM adjusts the plan using the new box location. Then, the system proceeds with the revised goal (refer to Figure 3-8).

Chapter 4

Experimental Setup

This chapter describes the hardware, software, test environment, and experimental procedures used to evaluate the DSG-HRI framework. Development began with simulation-based testing on a PC, followed by experiments on the real Franka Research 3 robot. Results are presented in Chapter 5.

4.1 Hardware and Software

Table 4.1: Hardware configuration.

Component	Specification	Role
Robot	Franka Research 3 (7-DOF)	Manipulation
Camera	Intel RealSense D455 (wrist)	RGB-D perception
GPU PC	RTX 2080 Ti, 64 GB RAM	Perception + planning
RT-PC	PREEMPT_RT kernel	CRISP controller

Table 4.2: Software components.

Component	Model / Version
Object detection	RAM++ + Grounding DINO + SAM2 + CLIP
Human tracking	MediaPipe PoseLandmarker / YOLO-Pose
Task planning	Llama 3.1 (8B) via Ollama
Robot control	CRISP Cartesian impedance
Middleware	ROS 2 Jazzy (GPU PC), Humble (RT-PC)

The two computers communicate over Ethernet using CycloneDDS as the messaging layer. The GPU workstation runs perception, scene graph maintenance, language

model reasoning, and manipulation coordination. The RT-PC handles the time-critical CRISP impedance control loop and interfaces directly with the Franka robot hardware.

4.2 Test Environment

The FR3 robot is positioned at one edge of a table, operating within an 80 cm reach envelope. A human operator stands across the table from the robot. Both work within their respective zones, and the distance between the robot base and the interaction area remains fixed across all trials.

Table 4.3: Test objects.

Object	Grasp type	Purpose
Ball	Top-down	Spherical object handling
Cube	Top-down	Box-shaped object handling
Screwdriver	Top-down	Tool-shaped object handling
Container	– (place target)	Place target surface

Three graspable objects and one place target were used across all experiments. The container served as the target surface for all place actions. This limited object set is acknowledged as a limitation — future work should expand testing to a wider variety of shapes, sizes, and materials.

The object set was intentionally kept small to focus on validating the core pipeline — perception, planning, grasping, and human interaction — before expanding to a broader range of objects.

4.3 Experiments

Three groups of experiments were conducted: individual action tests, consecutive task combinations, and extended multi-task sequences. All experiments were performed on the real Franka Research 3 robot with a human operator present.



Figure 4-1: Robot with RGB-D camera mounted on a wrist.

Experiment 1: Individual Action Success

Each of the four core actions — pick, place, handover, and receive — was tested independently with 15 runs each, for a total of 60 trials. Each trial began with a workspace scan, followed by a single natural language command (e.g., “pick up the tool” or “give me the tool”). Success was defined as the action completing without failure or human intervention.

Experiment 2: Consecutive Task Combinations

Four two-action combinations were tested with 15 runs each, totaling 60 trials. These test whether the system can chain actions reliably, since errors in the first action propagate to the second.

Table 4.4: Task combinations for consecutive execution experiments.

#	Task combination
1	Pick → Place (e.g., “pick up the tool and put it on the box”)
2	Pick → Handover (e.g., “pick up the tool and give it to the person”)
3	Handover → Receive (give object, then receive it back)
4	Receive → Place (receive object from human, then place it)

Experiment 3: Extended Consecutive Sequences

Five runs tested how many consecutive tasks the robot could complete before failure. In each run, task combinations were chained in no particular fixed order until the robot failed an action. A test of endurance. This move has pretty good individual attempts (80-93%), but with so many moves in a row, the errors add up in the end.

Experiment 4: Scene Graph Accuracy

For evaluation, the generated scene graphs are compared to ground truth graphs of test scenes and to system-generated scene graphs of previous scenes. For each test scene, the operator places objects on a table and performs a workspace scan. Then for each node and edge in the system-generated scene graph for the test scene, the operator compares it to corresponding nodes and edges in the ground truth scene graph or the system-generated scene graph for a previous scene. Corresponding nodes must have the same node label, with synonyms acceptable. Corresponding nodes must also be within a 10 cm distance of each other.

4.4 Evaluation Metrics

4.5 Data Collection

Each trial followed a standardized procedure:

1. Place all objects on the table and verify the camera views and CRISP connection function correctly.

Table 4.5: Evaluation metrics.

Category	Metric	Definition
Scene graph	SGA	Fraction of nodes and edges matching ground truth
Execution	Individual success	Single action completed correctly
	Combination success	All actions in sequence completed
HRI	Handover success	Human receives object via width detection
	Receive success	Robot grasps object placed by human

2. Build the scene graph from a workspace scan.
3. Send the task command via the `/dsg/command` topic.
4. Record the outcome of the experiment (pass or fail). Complete the relevant failure mode if test failed.
5. Run object recognition match accuracy on an scene understanding accuracy metric by comparing the accuracy to a gold standard (ground truth).

During the evaluation, a total of 125 trials were conducted, comprising 60 single action trials, 60 consecutive combination trials and 5 sequences until failure.

Chapter 5

Results

This chapter presents results from tests and simulations, and then the experiments that were carried out on the real Franka Research 3 robot.

5.1 Implementation Status

Here are the system part status comments for each core part, shown in Table 5.1. Every core part has a working version and has been tested on real hardware.

Table 5.1: Implementation of the framework at the moment.

Component	Status	Tested on
RAM++ + DINO + SAM2 + CLIP	Complete	Real camera
Object color extraction	Complete	Real camera
Human tracking (MediaPipe)	Complete	Real camera
YOLO-Pose fallback	Complete	Simulation
Dynamic scene graph	Complete	Real camera + simulation
Relation detection (7 types)	Complete	Real camera + simulation
Frustum-aware persistence	Complete	Real camera
LLM task decomposition	Complete	Real camera + real robot
Scene-change replanning	Complete	Simulation + real camera
Grasp planning (heuristic)	Complete	Real robot
CRISP bridge + convergence	Complete	Real robot
Workspace scan routine	Complete	Real robot
Handover with width detection	Complete	Real robot
Receive with width detection	Complete	Real robot
Human proximity safety pause	Complete	Real robot
Speech input (Whisper)	Implemented	Not tested
Evaluation toolkit	Complete	Real camera

The software code for this demonstration experiment is provided in 9 ROS 2 packages (msg, services, launch, etc) and contains 48 Python source files in total. All packages compiled on my development laptop (running ROS 2 Jazzy) as well as on my GPU workstation with a RoboStack conda environment.

5.2 Perception Pipeline

We evaluate real-time performance using a single Intel RealSense D455 camera. For the second experiment, we capture a scene with several items on a table, including two bottles, three phones, three boxes, two cups and two keyboards. Most objects are detected by DINO, which is grounded by RAM++ tags. The total detection time is less than 33ms. SAM2 produced accurate pixel-level masks by following object edges closely. The CLIP feature vectors stored with each object also enable later language-based querying of the scene graph.

The pipeline processes frames at approximately 2–5 Hz on the RTX 2080 Ti using the small SAM2 variant. Individual stages can take hundreds of milliseconds, but this throughput is sufficient for the slow-changing scenes in which the robot operates.

Figure 5-1 shows an example of detected objects.

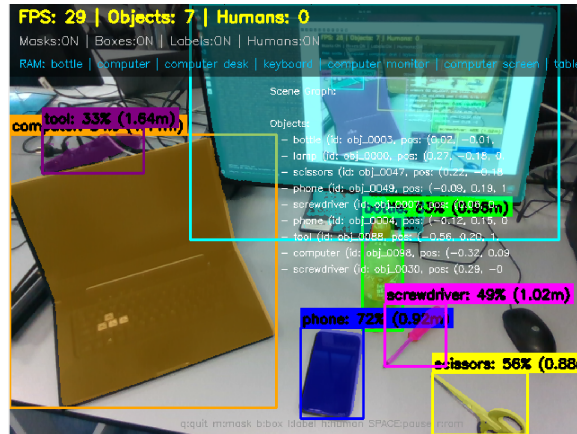


Figure 5-1: Example of detected objects.

5.2.1 Object Association

Initial versions of the association logic produced incorrect detections in the form of duplicate nodes: for a scene with fewer than 10 objects, the older code produced thousands of duplicate nodes when the same object appeared at slightly different positions across frames. This issue was resolved by merging duplicate detections using the composite similarity score shown in Equation 3.6. Updated to always track the same number of objects at all times and now prevents extra nodes from being generated for small positional variations between frames.

5.2.2 Color Recognition

The colour of objects that are mainly a single colour can be classified using HSV. This system was tested with a range of objects of different colours (a red cup, a blue bottle, a green container, a black phone) under normal room lighting and accurately classified the red cup, blue bottle, green container and black phone as being the colours red, blue, green and black respectively. This information is incorporated into the scene classification and provided in the language model's output, meaning that commands such as "pick up the red cup" can be generated even when there are multiple cups on the table.

I want to note that in some cases incorrect classification may occur when an object is of a mixed colour, such as a white box with a coloured label. In addition, strongly directional lighting can result in large colour errors, and as in highly lit scenes the direction of light can be from the front this can result in significant misclassification.

Figure 5-2 shows the human pose detector and tracker in action.

5.3 Scene Graph and Relation Detection

5.3.1 Spatial Relations

ON relationships are detected quickly, typically within the first one or two frames, using footprint overlap to determine which object is supported by which. The full

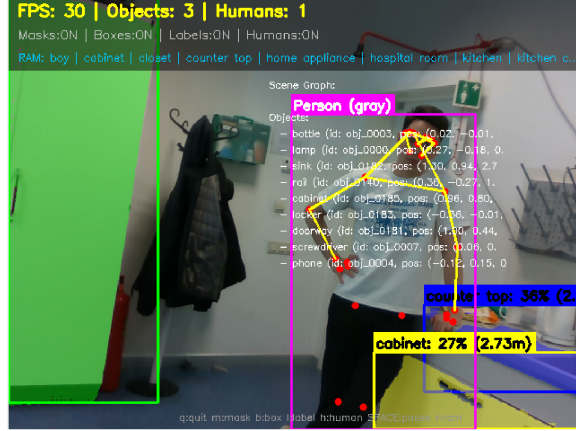


Figure 5-2: Human pose detector and tracker demonstration.

scene graph for a table typically stabilises within about 3 seconds. In live testing, the system handles up to 5 objects simultaneously without noticeable slowdown.

INSIDE relations are detected once the outer object is recognised as a known container type. NEXT_TO and NEAR relations capture the proximity of objects that share a surface or are within a short 3D distance. INSIDE detection depends on the outer object being labelled with a word matching the known-container list.

5.3.2 HOLDING Detection

HOLDING is recognised when a hand and an object are within approximately 35 cm of each other. An early issue with the local update mechanism was that pose updates only included human nodes, without the surrounding object nodes, so the HOLDING relation could never be detected during human movement.

After fixing the update group to include both human and object nodes together, HOLDING is detected consistently, and the visualisation correctly indicates which hand (left or right) is holding which object.

The multi-stage hand tracking proved essential during real-world testing. When a person holds an object, the object itself frequently occludes the wrist from the camera, so depth data at the wrist marker becomes unreliable. Falling back to fingertip averaging (and finally to arm-direction extrapolation) allows the hand position to be recovered even when the wrist is not directly visible.

Figure 5-3 shows an example of human-object interaction.

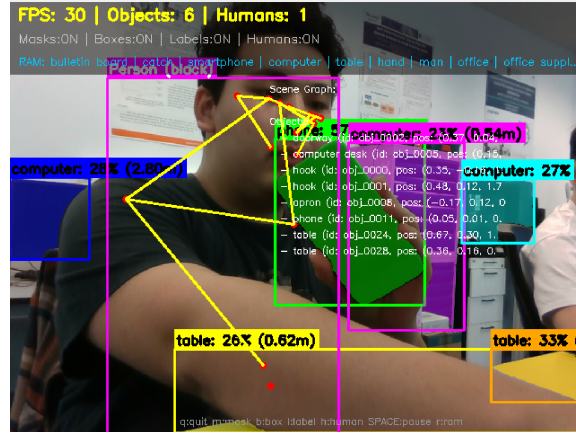


Figure 5-3: Example of human-object interaction.

Figure 5-4 shows the scene graph detections with relation labels and object descriptions.

```
data: "Scene Graph:\n\nObjects:\n - table (id: obj_0006, pos: (-0.42, 0.23, 0.87), category: small, conf: 0.22, is_surface=true)\n - table (id: obj_0001, pos: (0.26, 0.17, 0.60), category: small, conf: 0.33, is_surface=true)\n - phone (id: obj_0000, pos: (0.08, -0.07, 0.56), category: small, conf: 0.61)\n\nHumans:\n - Person (id: human_0, clothing: black, pos: (-0.10, 0.01, 0.79), left_hand: (0.15, -0.05, 0.57), right_hand: (0.24, 0.12, 0.69))\n\nRelations:\n - phone(obj_0000) --on--> table(obj_0001) [0.31]\n - person(human_0) --holding--> phone(obj_0000) [0.80] (with left hand)\n - person(human_0) --reaching_for--> table(obj_0006) [0.20]\n - person(human_0) --holding--> table(obj_0001) [0.69] (with right hand)\n\nRobot (Franka FR3):\n Gripper: empty\n Status: idle"
```

Figure 5-4: Scene-graph detections description.

5.3.3 Frustum-Aware Persistence

Without persistence, objects disappear from the scene graph as soon as the camera looks away from them. This issue is particularly severe on a wrist-mounted setup, where every grasp attempt re-orientates the camera and temporarily hides previously detected objects. The frustum-aware persistence described in Section 3.3.5 keeps objects in the scene graph across these viewpoint changes, removing them only after a configurable timeout (e.g., 2 minutes). This prevents false dropouts during interaction, while still allowing genuinely removed objects to be cleaned up eventually.

5.3.4 Scene Graph Accuracy

The accuracy of the generated scene graphs was evaluated against manually annotated ground truth across several test scenes. On average, the system achieved 83% accuracy. Three factors had the largest effect on this score, raising it from a raw 60% to the final 83%:

- **Synonym labeling.** Objects and their outlines were accurately detected and drawn on the image, but were labeled with inaccurate synonyms (e.g. box labeled as container, screwdriver labeled as tool). Accuracy improved after incorporating synonym matching.
- **Occasional duplicates.** Some objects are currently represented as multiple nodes during the viewing-angle scan, but then correctly merged by the association mechanism.
- **Self-referential objects.** The system is unable to recognize these objects. Objects such as a table surface that the object sits upon are not recognized due to the downward field of view of the camera mounted on the wrist at home position.

5.4 LLM Task Planning

5.4.1 Plan Generation

All of the conversations with the Llama 3.1 model with 8B parameters were conducted locally using Ollama. Examples of input and generated sequences are presented in Table 5.2.

Adding Franka-specific context to the system prompt noticeably improved the baseline performance. Because the robot has a fixed base and cannot move towards people, the prompt explicitly instructs the planner that humans must approach the robot for object transfer. A `look_at` step is also added before every pick to orient the

Table 5.2: Example LLM planning results from real-robot testing.

Command	Generated plan
“Pick up the tool”	look_at(tool) → pick(tool)
“Put the tool on the box”	look_at(box) → place(box)
“Pick up the tool and put it on the box”	look_at(tool) → pick(tool) → look_at(box) → place(box)
“Give tool to human”	look_at(tool) → pick(tool) → han- dover(person)

wrist camera at the target. The planner is explicitly prevented from selecting objects that are currently held by a person.

The 8B Llama 3.1 model reliably produces valid JSON output. Occasional parsing errors occur but are resolved by the 2–3 automatic retries. End-to-end latency from command to plan is 2–5 seconds, which is acceptable for this application. The rule-based fallback planner has not needed to activate in practice, because the LLM planner remains available throughout testing.

Figure 5-5 shows the LLM task decomposition example.

```

ros2 run dsg_manipulation manipulation_node --ros-args \
  -p use_ollama:=true \
  -p ollama_model:=llama3.2:3b \
  -p execution_timeout:=5.0
[INFO] [1773922257.062016213] [manipulation_node]: [CrispBridge] CrispBridge initialized
[INFO] [1773922257.062384937] [manipulation_node]: [CrispBridge] Target topic: /crisp/target_pose
[INFO] [1773922257.062654711] [manipulation_node]: [CrispBridge] Feedback topic: /crisp/current_pose
[INFO] [1773922257.065840980] [manipulation_node]: Manipulation Node initialized!
[INFO] [1773922257.066276173] [manipulation_node]: LLM: Ollama (llama3.2:3b)
[INFO] [1773922287.889677681] [manipulation_node]: Command: pick up the red cup
[INFO] [1773922287.891233960] [manipulation_node]: Scene: 1367 chars
[INFO] [1773922287.891640924] [manipulation_node]: Planning task...
[INFO] [1773922297.677053724] [manipulation_node]: Plan: 2 actions
[INFO] [1773922297.677508536] [manipulation_node]: 1. [look_at] Look at the red cup
[INFO] [1773922297.677891087] [manipulation_node]: 2. [pick] Pick up the red cup
[INFO] [1773922297.678240485] [manipulation_node]: Executing: [look_at] Look at the red cup
[INFO] [1773922297.679323782] [manipulation_node]: [CrispBridge] No EE feedback – using time-based wait
[INFO] [1773922299.688362461] [manipulation_node]: Executing: [pick] Pick up the red cup
[INFO] [1773922300.292574593] [manipulation_node]: [CrispBridge] No EE feedback – using time-based wait
[INFO] [1773922302.299693280] [manipulation_node]: [CrispBridge] No EE feedback – using time-based wait
[INFO] [1773922304.907925636] [manipulation_node]: [CrispBridge] No EE feedback – using time-based wait
[INFO] [1773922306.914318263] [manipulation_node]: Picked: red_cup
[INFO] [1773922306.915113450] [manipulation_node]: Plan complete!

```

Figure 5-5: LLM task decomposition example.

5.4.2 Scene-Change Replanning

Replanning was tested and validated in simulation. The test scenario ran a sequence of pick and place actions: first picking up a cup and placing it on the table, and then picking up the cup and placing it on a box. Partway through the sequence, the box was moved to a new position unexpectedly. The sensor feeds registered the movement while the robot was grasping the cup. After detecting a displacement greater than 5 cm, the system re-queried the LLM with the updated scene, which produced a revised plan that matched the box’s new position.

An earlier version of the trigger relied on incrementing a version number on the scene graph after every update. This approach did not work well in practice: the version number increased approximately 5 times per second due to continuous edge recomputation, triggering many unnecessary replans. The current version instead compares object positions directly across snapshots.

5.5 Manipulation Execution

5.5.1 Simulation Testing

Tests for the manipulation procedure were performed in two separate test series. The results, together with the test parameters, are presented in Table 5.3.

Table 5.3: Simulation environments and results.		
Environment	Setup	Result
Numerical	Mock scene + sim node	Full pipeline verified: plan, execute, complete
Gazebo	FR3 + ForwardCommandController	Robot moves to commanded positions

The full pipeline was first validated in the numerical simulator using artificial detections fed into the scene graph. This setup allowed the relational structure of the objects to be verified without running perception, and exposed how different commands map to different physical action sequences. The LLM decomposed each command into a series of low-level actions, and all core physical actions — picking,

moving, and releasing — were exercised within the simulation environment before being run on the real robot.

5.5.2 Real Robot Experiments

The complete pipeline was run on the real Franka Research 3 robot. Four individual actions — pick, place, handover, and receive — were tested for 15 runs each, and four two-action combinations were tested for 15 runs each. Five additional runs tested longer sequences of consecutive tasks. Results are summarised in Table 5.4 and Table 5.5.

Table 5.4: Individual action success rates (15 runs each).

Action	Successes	Rate	Common failure
Pick	13/15	87%	Grasp misalignment
Place	12/15	80%	Position drift from close-up
Handover	14/15	93%	Human timeout
Receive	13/15	87%	Width detection sensitivity

Table 5.5: Consecutive task combination success rates (15 runs each).

Task combination	Successes	Rate
Pick → Place	10/15	67%
Pick → Handover	13/15	87%
Handover → Receive	13/15	87%
Receive → Place	9/15	60%

Handover achieved the highest individual success rate at 93%, because the width-change detection mechanism triggers reliably on human interaction. The combined pick-and-handover sequence also reached 87%, because handover does not require precise positioning — it only needs to extend the gripper in the direction of the detected hand.

Combinations involving a place action achieved success rates of only 60–67%. The main cause was position drift: during `look_at`, the close-up view of the target surface shifts the apparent centroid, leading the robot to place the object off-centre. Plan-time position freezing reduced but did not fully eliminate this failure mode.

5.5.3 Extended Consecutive Task Sequences

Five runs tested extended sequences where task combinations were chained together until failure, without any particular fixed order.

- Best run: 10 consecutive tasks completed
- Also completed 6, 5, 5 and 3 other clears.

Failures in extended sequences were accounted for by the accumulated failure rates for individual actions (approximately 80–93% reliability per action, see Table 5.4), which in practice meant that the failure rate for longer sequences was substantially higher. Most failures in reaching resulted from a marginal initial grasp that failed during transport to the place location.

5.6 Problems Found While Building

As shown in Table 5.6, major problems that arose are listed alongside how they were addressed. Several issues were only discovered during real-robot testing and could not have been predicted from simulation alone.

Several of these issues could not have been anticipated from the design alone; they became visible only during real-robot testing. During this tests two issues popped up that we found particularly intriguing. Firstly, an incompatibility between GripperCommand and Franka Grasp actions required us to redesign our handover logic. Secondly, we observed a constant bias in the depth-camera centroid that could only be cured by a fixed offset compensation and not by a simple parameter adjustment.

5.7 Summary

We have tested the complete system on the real Franka Research 3 arm (Franka) over 125 trials where 60 were individual action trials, 60 trials involved two actions being performed together, and 5 trials involved a sequence of length 5. These trials

Table 5.6: Issues identified and resolved during development.

Issue	Cause	Solution
Duplicate objects	Association threshold too strict	Combined similarity score with three components
HOLDING never detected	Local update excluded objects from human callback	Cross-type inclusion in update set
Continuous false replanning	Version number increments on edge recomputation	Position-based structural comparison
LLM cannot find “red cup”	Scene context used labels without colors	Scene description with color extraction
ROS 2 Jazzy→Humble type mismatch	Type hash incompatibility	Relay node on RT-PC
Wrist depth occlusion during hold	Held object blocks depth sensor	Multi-stage hand fallback (fingertip, extrapolation)
Gripper bounces after contact	GripperCommand releases before re-closing	Use Franka Grasp action (maintains force)
Object drops during handover	GripperCommand overrides Franka Grasp	Keep 140 N, detect width change to release
Phantom human detections	Camera artifacts, shadows	8-frame confirmation + confidence/distance filters
Human forgotten before handover	60 s scene timeout too short	Increased to 180 s (scene) / 300 s (memory)
Place rotates while holding	Memory recall moves camera with tilt	Use memory position directly, no camera movement
look_at human always fails	Scene search only queries objects	Skip look_at for human targets
Place lands on near edge	Depth camera centroid bias	6 cm outward nudge compensation

included reach, move to location, rotate to face, handover and place actions. Success rate for individual actions was between 80% and 93%. Handover was the most successful action at 93% and place was the least successful at 80%. Two-action combinations achieved success rates around 60% to 87%. On average the system was accurate for 83% of the scene graphs it generated when compared to ground truth. In most cases this was due to the use of synonym matching. Importantly we have gained a number of insights in to the workings of the system that only came to light through the real-robot experiments, for example GripperCommand and Franka-Grasp are incompatible for handover, and the order of camera views during the workspace scan for place action is critical to obtain accurate object positions. And although the results are great, it worth to mention that they can't represent overall system's reliability since experiments conducted only on a few object and success was counted even in non perfect attempt as long as task was completed. If more object were to be added and more experiments conducted the results could differ.

Chapter 6

Discussion

In this chapter I explain some of my choices regarding the framework and how they affected the results. Some limitations were found during implementation, and alternative design paths exist that could have been explored. These decisions form the basis of the system presented in this thesis.

6.1 Design Decisions

Wrist-Mounted Camera

The wrist-mounted camera is not locked to one position above the workspace. As the arm moves, it provides a range of vantage points along the arm’s path, including close-up views where the gripper makes contact with an object. The workspace scan captures information from seven different poses, giving a broader understanding of the scene than a single fixed viewpoint could provide.

The main drawback of a wrist-mounted camera is that objects fall out of view as the arm moves. Items on one side of the workspace can be missed while the arm is reaching for an object on the other side, so some objects will be outside the camera’s view at any given moment. The frustum-aware persistence described in Section 3.3.5 distinguishes between an object that has genuinely disappeared and one that is temporarily out of the field of view. A complementary fixed overhead camera

would give continuous coverage of the whole workspace and would be a useful addition for future work.

CRISP vs. MoveIt

Instead of relying on MoveIt’s built-in trajectory planning, the controller directly follows a straight line through space via CRISP Cartesian impedance control. MoveIt plans a path in configuration space, avoiding other objects and objects in the workspace. CRISP plans a path in task space, following the desired point through space via impedance control resulting in a roughly linear path through the task space.

When a human is working near a robot, it is desirable to penalize inaccuracies in compliance instead of in trajectory. When contact occurs during a task, an impedance controlled system will behave naturally and gently. In contrast, a position controlled robot will push strongly against a human, and this is preferable to unsafe compliance for appropriate workspaces within the joint space. In this sense, safety is improved by choosing to override compliance for this task.

For collision checking, the robot’s movement is broken down into straight lines to individual waypoints along its path, with no acceleration between these. For picking an object from a table for example, the gripper first moves horizontally towards the table, then straight down, then up once the object has been picked. Once freed from the object the gripper moves straight up. This simple method of clearing obstacles allows for very simple scene modeling. It is very effective for actions such as moving an object up and over a surface, down onto a surface, etc. as described on the pages for flat surfaces covered, but for movements such as seamed on or very close to walls (i.e. looking for very little space), or even reciprocally moving alongside a barrier (i.e. sideways past an edge of table), coupled with MoveIt can help achieve a better result.

LLM Planning vs. Classical Planning

The framework does not rely on a formal task specification. The LLM processes natural-language input directly and produces a sequence of executable actions. Unlike

traditional planners that solve PDDL problems, no hand-coded action preconditions or effects are required, and the LLM can handle new command phrasings without modification to the planner.

Reliability drops as plan length grows. Three design choices help reduce incorrect or inefficient plans. We enforce robot realities by constraining both the system prompt (e.g. one object at a time, look before pick, max gripper width 8 cm) and the LLM’s output to a JSON schema. In addition to this, we replan not only on failures of a generated plan, but also whenever the scene changes by more than 5 cm while executing a plan.

It works well for simple commands and for commands of moderate scope. It fails, however, to generate complete and accurate plans for multi-part commands that span multiple statements, and not all such commands appear to work.

Local LLM vs. Cloud API

We also played around with running the LLM offline using Ollama, which meant we didn’t have to do any web requests, external services or pay per token for the LLM. The real cost was in memory on our local machines to run the LLMs — the 8B model was approximately 5 GB on the GPU and responded in 2–5 s. We could have swapped in a machine with more memory to run the larger Llama 3.1 70B model, or even just used cloud APIs like GPT-4o for trickier prompts. The LLM client can switch between these modes with a flag.

Gripper Control Strategy

One of our design choices was to not make GripperCommand and Franka Grasp (the “Franka Grasp” action) be compatible. The reason for this is that Franka Grasp applies force to the object after contact to get a firm grip on it. However, any GripperCommand (to open the gripper to the same width) will override the force controller. This had implications for the handover design, where we chose not to transition to a soft grip for human retrieval. Instead, the system continues to apply

the full Franka Grasp force, and detects that the human has taken the object by the gripper width having changed. This interaction was discovered through real-robot testing, not through any of the simulators that we tested.

Structural Change Detection

Instead of tracking a version number of the scene graph, the system tracks version number based on the position change. Simulation results showed that the edge-based version counter increased almost 5 times per second on typical dynamic scene graph updates, leading to a lot of unnecessary replanning. The current value of 5 cm displacement is large enough to avoid replanning although changes in objects' positions affect the execution stream of the replanning task.

For the simulation we chose a rather moderate detection change of 5 cm to avoid false triggering (e.g. a flickering fluorescent light) but at the same time not to ignore a person moving a mug from one side to another (e.g. from left to right across a table top). We tried several values and 5 cm seemed to be a good compromise. Too small and there would be too many false alarms, too large and there would not be enough detected changes.

6.2 Limitations

This section discusses some of the known limitations of this work as well as some speculative commentary regarding the implications of these limitations.

Table 6.1 summarizes the known limitations and their potential impact.

We currently have a grasp planner based on heuristics that works fine for simple shapes like cups, bottles, boxes etc. but fails for objects with an unusual or complicated shape. Instead of adding more rules to the heuristic approach, it might be more promising in the long run to learn a grasp predictor like AnyGrasp that generalises to objects not in the training data.

For longer actions, viewing all angles with a single wrist mounted camera is not sufficient. A single fixed camera positioned on the side or top of the monitor would

Table 6.1: Known limitations of the current system.

Limitation	Impact	Potential solution
Single wrist camera	Objects lost during manipulation	Add fixed overhead camera
No collision-aware paths	Cannot reach around obstacles	Integrate MoveIt for planning
Heuristic grasp planner	Fails on novel geometries	Replace with AnyGrasp or learned model
RAM++ label noise	False detections (“shadow”, “indoor”)	Expand blacklist, fine-tune tagger
No articulated objects	Cannot open drawers or doors	Extend scene graph with joint states
LLM fails on complex commands	Multi-step compound tasks may produce wrong plans	Use larger model or structured prompting
Color under lighting variation	Misclassification under strong directional light	Histogram-based color with white balance
Franka Grasp incompatibility	Cannot switch grip modes while holding	Design around single-mode gripper control
Camera-to-base calibration	Manual static transform, not auto-calibrated	Implement hand-eye calibration routine
Place position drift	Close-up re-detection shifts target center	Improve plan-time position freezing
Only a few object were tested	The results might differ with various objects	Test with more object

ideally provide a continuous view of the object as the arm moves around, capturing a continuous view of the object’s position. Although the system does perform periodic re-scans of the workspace, a top-down continuous view would be the ideal solution.

6.3 Future Work

Near-Term

In the near term, the most valuable next step is to broaden testing. Adding more objects to the test set — particularly objects with complex shapes or reflective materials — would stress-test the perception and grasping pipelines. Testing under varied lighting conditions would expose weaknesses in the colour extraction and object detection stages. Running more than 5 extended-sequence runs would also give stronger statistical confidence in the long-term reliability figures reported here.

System Improvements

A few more adjustments and it would work better.

Table 6.2: Planned system improvements.

Improvement	Expected benefit
Fixed overhead camera	Continuous workspace observation independent of arm motion
AnyGrasp integration	Learned grasp prediction for novel objects
Hand-eye calibration	Automatic camera-to-base transform, replacing manual measurement
MoveIt as backup planner	Collision-aware paths for cluttered environments
Articulated object modeling	Support for drawers, doors, and other jointed objects
Multi-person tracking	Handle multiple humans simultaneously in the workspace

Research Directions

Future perception systems may follow one of two paths: a modular pipeline with separately optimised components for recognition, segmentation, and tracking, or a fully integrated vision-language-action model that performs all tasks end-to-end. The end-to-end approach may achieve higher overall performance but is harder to interpret and query about specific objects. Scene graphs provide an efficient intermediate representation that pairs well with both approaches: the graph can support high-level scene understanding while the policy network focuses on low-level action selection.

A second direction is to connect a live-updating knowledge base to the LLM planner. As the robot accumulates experience (the usual locations of objects, the relative frequency of different commands, the most effective grasp strategies for particular items), this knowledge can inform future planning decisions. This would let the system improve over time without retraining the underlying language model.

A third direction could be the adding the support of the multiple robots. A shared scene graph that is able to support coordination between the robots, divide tasks between them, move objects in a shared space while also cooperation with each other.

The LLM planner could reason over this multi-robot scene graph to assign sub-tasks to specific robots based on their current positions and held objects.

Chapter 7

Conclusion

This thesis addressed the problem of enabling a fixed-base robotic arm to perform long-term tasks using natural language instructions in a workspace shared with a human operator. The resulting framework, DSG-HRI, combines four main components: an open-vocabulary visual recognition system for unfamiliar objects, a dynamic scene graph that acts as a real-time map of the workspace, a high-level task planner based on a large language model, and a compliant motion controller for safe physical interaction.

Several open-vocabulary perception techniques were integrated into a single pipeline: image tagging via RAM++, open-vocabulary object detection via Grounding DINO, pixel-level segmentation via SAM2, and semantic feature extraction via CLIP. Body and hand poses are estimated using MediaPipe PoseLandmarker, with YOLO-Pose as a fallback when MediaPipe fails due to occlusion or distance. We compute dominant colours for each object in the scene to support colour-based disambiguation (e.g. “pick up the red cup”). Recovering partially-occluded hands is a very challenging problem; we tackle this through a multi-stage fallback process beginning from wrist location and then moving through fingertip averaging and arm direction extrapolation where necessary. We describe techniques to suppress detections of phantom humans and describe in more detail how we use detection confidence, distance and position bounds and track confirmation duration (a detected human must be confirmed real over an 8-frame window).

The scene graph encodes a dynamic memory of the current scene, capturing relationships of position, semantics, and interaction (e.g. picking, grasping) between objects, humans and the robot itself. These relationships are stored in the scene graph as nodes and edges. The graph is updated locally rather than rebuilt from scratch: only the edges around modified nodes are recomputed, keeping the update cost bounded. A frustum-aware persistence mechanism keeps objects in the graph even when the wrist camera is pointed away from them, so that objects are not forgotten during a grasp motion. Human-object interactions such as `HOLDING` and `REACHING_FOR` are detected reliably once both the human and the nearby objects are included in the same update group.

The LLM planner (Llama 3.1 8B, run locally via Ollama) takes the natural-language command together with a textual summary of the scene graph and produces a sequence of executable actions in a JSON format. Each object’s colour, size, position, relations, and the robot’s current gripper state are included in the scene description, allowing the planner to reason about which object the user means and whether the robot can act on it. When objects move by more than 5 cm during execution, the planner is re-queried with the updated scene, and the remaining actions are replaced by a revised plan. A set of post-processing steps validate the plan, remove redundant actions, and handle edge cases such as `look_at` commands targeting a human.

Physical execution is handled by a CRISP Cartesian impedance controller running at 1 kHz on a real-time PC. This provides inherent compliance: when contact occurs with a human or obstacle, the arm yields smoothly rather than pushing through. Handover and receive actions both use the human’s detected hand position as the target when available. For handover, the Franka Grasp action maintains constant force on the object, and a change in gripper width greater than 2 mm is used to detect the human taking the object. For receive, the gripper is opened and a drop in width signals that the human has placed an object between the fingers, triggering a Franka Grasp. These width-based interaction patterns work around an important constraint discovered during testing: any GripperCommand sent after a Franka Grasp

overrides the force controller and drops the object.

The capabilities of the system were evaluated through physical experiments on the Franka Research 3 robot, comprising 60 individual action trials, 60 consecutive two-action combination trials, and 5 extended-sequence runs. The best extended-sequence run completed 10 consecutive actions before failure. Handover achieved the highest individual action success rate at 93%. Scene graphs generated by the system matched manually annotated ground truth at 83% accuracy on average. Real-robot testing also revealed findings that were not visible in simulation: that GripperCommand and the Franka Grasp action are mutually incompatible; that the order of viewpoints during the workspace scan affects depth-based position accuracy; and that the depth camera centroid has a systematic bias that must be compensated in the place logic.

The contribution of this work lies not in any single component but in their integration. The underlying techniques — open-vocabulary perception, dynamic scene graphs, LLM-based planning, human pose tracking, and compliant control — are all drawn from prior work and have been used separately in earlier systems. DSG-HRI combines them into a single cohesive framework targeted at a fixed-base collaborative robot, a configuration that had not previously been addressed by a single integrated system.

Appendix A

LLM System Prompt

The following is the complete system prompt used by the LLM task planner. This prompt is sent with every planning request alongside the current scene description and user command. The abbreviated version appears in Listing 3.4.

```
1 You are a task planner for a Franka Emika Panda robotic arm.
2
3 ROBOT SETUP:
4 - Fixed-base 7-DOF manipulator (it CANNOT move its base -- no navigation)
5 - Parallel-jaw gripper (max opening ~8 cm)
6 - RGB-D wrist camera
7 - Workspace: ~80 cm reach from base
8
9 AVAILABLE ACTIONS:
10   look_at  -- Point camera at an object (always do this before pick)
11   pick     -- Grasp and lift an object (requires target_object)
12   place    -- Put the held object onto a surface (requires target_location)
13   handover -- Extend the held object toward a person (requires target_human)
14   receive  -- Open gripper and wait for human to place object (requires target_human)
15   open_gripper / close_gripper -- Direct gripper control
16   wait     -- Pause (optional parameters.seconds)
17   speak   -- Say something (requires parameters.message)
18
19 SCENE INFORMATION:
20 You will receive a list of objects (with positions, sizes, IDs), humans
21 (with hand positions), and relations such as:
22   ON        -- object is resting on a surface
23   HOLDING   -- a human is grasping the object
24   NEAR / NEXT_TO -- objects close together
25   INSIDE    -- object is inside a container
```

```

26
27 RULES:
28 1. Always look_at an object BEFORE picking it.
29 2. The robot can hold only ONE object at a time.
30 3. Use the exact object labels or IDs from the scene.
31 4. If a human is HOLDING an object you cannot pick it -- ask them
32    to put it down or wait.
33 5. If the target is not in the scene, set "success": false and
34    explain in "error".
35 6. Keep plans short. Do not add unnecessary steps.
36
37 RESPOND WITH ONLY A JSON OBJECT -- no commentary before or after:
38 {
39   "success": true,
40   "reasoning": "short explanation",
41   "error": "",
42   "actions": [
43     {"action": "look_at",
44       "target_object": "cup",
45       "target_location": null,
46       "target_human": null,
47       "description": "Look at the cup"},
48     {"action": "pick",
49       "target_object": "cup",
50       "target_location": null,
51       "target_human": null,
52       "description": "Pick up the cup"}
53   ]
54 }
55
56 EXAMPLES:
57
58 User: "Pick up the cup and put it on the box"
59 Scene: cup (obj_0003) ON table, box (obj_0005) ON table
60 {"success":true,
61   "reasoning":"Pick cup, place on box",
62   "error": "",
63   "actions":[
64     {"action":"look_at","target_object":"cup",
65       "target_location":null,"target_human":null,
66       "description":"Look at the cup"},
67     {"action":"pick","target_object":"cup",
68       "target_location":null,"target_human":null,
69       "description":"Pick up the cup"},
70     {"action":"look_at","target_object":"box",

```

```

71     "target_location":null,"target_human":null,
72     "description":"Look at the box"},
73     {"action":"place","target_object":null,
74       "target_location":"box","target_human":null,
75       "description":"Place cup on box"}}}
76
77 User: "Give me the pen"
78 Scene: pen (obj_0001) ON table, person (human_0)
79 {"success":true,
80   "reasoning":"Pick pen and hand to person",
81   "error":"",
82   "actions":[
83     {"action":"look_at","target_object":"pen",
84       "target_location":null,"target_human":null,
85       "description":"Look at pen"},
86     {"action":"pick","target_object":"pen",
87       "target_location":null,"target_human":null,
88       "description":"Pick up pen"},
89     {"action":"handover","target_object":null,
90       "target_location":null,"target_human":"human_0",
91       "description":"Hand pen to person"}}}
92
93 User: "Pick up the bottle"
94 Scene: person (human_0) HOLDING bottle (obj_0002)
95 {"success":false,
96   "reasoning":"Bottle held by human",
97   "error":"Cannot pick bottle -- human_0 is holding it.",
98   "actions":[
99     {"action":"speak","target_object":null,
100      "target_location":null,"target_human":"human_0",
101      "parameters":{"message":"Could you put the bottle down?"},
102      "description":"Ask human to release bottle"}}}

```

Listing A.1: Complete LLM system prompt for task decomposition.

Bibliography

- [1] Saeid Amiri, Kishan Chandan, and Shiqi Zhang. Reasoning with scene graphs for robot planning under partial observability. *IEEE Robotics and Automation Letters*, 7:5560–5567, 4 2022.
- [2] Tjark Behrens, René Zurbügg, Marc Pollefeys, Zuria Bauer, and Hermann Blum. Lost & found: Tracking changes from egocentric observations in 3d dynamic scene graphs. *IEEE Robotics and Automation Letters*, 10:3739–3746, 2025.
- [3] Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. A comprehensive survey of scene graphs: Generation and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45:1–26, 1 2023.
- [4] Zhirui Dai, Arash Asgharivaskasi, Thai Duong, Shusen Lin, Maria Elizabeth Tzes, George Pappas, and Nikolay Atanasov. Optimal scene graph planning with large language model guidance. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 14062–14069. Institute of Electrical and Electronics Engineers Inc., 2024.
- [5] Gokhan Erdemir, Michael Liles, and Erkan Kaplanoglu. Body pose estimation and hand position detection for hand-by-hand human-robot interaction. In *International Conference on Systems, Signals, and Image Processing*. IEEE Computer Society, 2025.
- [6] Mingtao Feng, Chenbo Yan, Zijie Wu, Weisheng Dong, Yaonan Wang, and Ajmal Mian. History-enhanced 3d scene graph reasoning from rgb-d sequences. *IEEE Transactions on Circuits and Systems for Video Technology*, 35:7667–7682, 2025.
- [7] Qiao Gu, Ali Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, Chuang Gan, Celso Miguel De Melo, Joshua B. Tenenbaum, Antonio Torralba, Florian Shkurti, and Liam Paull. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5021–5028. Institute of Electrical and Electronics Engineers Inc., 2024.
- [8] Daniel Honerkamp, Martin Buchner, Fabien Despinoy, Tim Welschhold, and Abhinav Valada. Language-grounded dynamic scene graphs for interactive ob-

- ject search with mobile manipulation. *IEEE Robotics and Automation Letters*, 9:8298–8305, 2024.
- [9] Yiwei Hua, Kerun Li, Ru Wang, Yingjie Li, Guoxin Wang, and Yan Yan. Integration of dynamic knowledge and llm for adaptive human-robot collaborative assembly solution generation. *Advanced Engineering Informatics*, 68, 11 2025.
 - [10] Ziyuan Jiao, Yida Niu, Zeyu Zhang, Song Chun Zhu, Yixin Zhu, and Hangxin Liu. Sequential manipulation planning on scene graph. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2022-October, pages 8203–8210. Institute of Electrical and Electronics Engineers Inc., 2022.
 - [11] Franklin Kenghagho Kenfack, Feroz Ahmed Siddiky, Ferenc Balint-Benczedi, and Michael Beetz. Robotvqa - a scene-graph- and deep-learning-based visual question answering system for robot manipulation. In *IEEE International Conference on Intelligent Robots and Systems*, pages 9667–9674. Institute of Electrical and Electronics Engineers Inc., 10 2020.
 - [12] Dohyun Kim, Yeseung Kim, Jaehwi Jang, Minjae Song, Woojin Choi, and Dae-hyung Park. Sggnet2: Speech-scene graph grounding network for speech-guided navigation. In *IEEE International Workshop on Robot and Human Communication, RO-MAN*, pages 1648–1654. IEEE Computer Society, 2023.
 - [13] Ue Hwan Kim, Jin Man Park, Taek Jin Song, and Jong Hwan Kim. 3-d scene graph: A sparse and semantic representation of physical environments for intelligent agents. *IEEE Transactions on Cybernetics*, 50:4921–4933, 12 2020.
 - [14] S. Linok and G. Naumov. Open-vocabulary indoor object grounding with 3d hierarchical scene graph. *Optical Memory and Neural Networks (Information Optics)*, 34:323–333, 9 2025.
 - [15] Haokun Liu, Yaonan Zhu, Kenji Kato, Atsushi Tsukahara, Izumi Kondo, Tadayoshi Aoyama, and Yasuhisa Hasegawa. Enhancing the llm-based robot manipulation through human-robot collaboration. *IEEE Robotics and Automation Letters*, 9:6904–6911, 8 2024.
 - [16] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2024.
 - [17] Ruben Mascaro and Margarita Chli. Scene representations for robotic spatial perception. *Robotics, and Autonomous Systems Rev. Control Robot. Auton. Syst.* 2025, 17:15, 2025.
 - [18] Zhe Ni, Xiaoxin Deng, Cong Tai, Xinyue Zhu, Qinghongbing Xie, Weihang Huang, Xiang Wu, and Long Zeng. Grid: Scene-graph-based instruction-driven robotic task planning. In *IEEE International Conference on Intelligent Robots*

and Systems, pages 13765–13772. Institute of Electrical and Electronics Engineers Inc., 2024.

- [19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.
- [20] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024.
- [21] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping*, pages 1689–1696, 05 2020.
- [22] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *International Journal of Robotics Research*, 40:1510–1546, 12 2021.
- [23] Nikolaos Stathoulopoulos, Christoforos Kanellakis, and George Nikolakopoulos. Have we scene it all? scene graph-aware deep point cloud compression. *IEEE Robotics and Automation Letters*, 2025.
- [24] Andreea Tulbure, René Zurbrügg, Timm Grigat, and Marco Hutter. Llm-handover: Exploiting llms for task-oriented robot-human handovers. *IEEE Robotics and Automation Letters*, pages 1–8, 9 2025.
- [25] Xiaopeng Wang, Guoliang Zhou, Hongpeng Hua, Zhengyang Tang, and Yutao Cao. Generating robotic control strategies with llms: Via human–robot voice interaction. *Concurrency and Computation: Practice and Experience*, 37, 11 2025.
- [26] Abdelrhman Werby, Chenguang Huang, Martin Büchner, Abhinav Valada, and Wolfram Burgard. Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation. In *Robotics: Science and Systems XX*. Robotics: Science and Systems Foundation, July 2024.
- [27] Zhijie Yan, Shufei Li, Zuoxu Wang, Lixiu Wu, Han Wang, Jun Zhu, Lijiang Chen, and Jihong Liu. Dynamic open-vocabulary 3d scene graphs for long-term language-guided mobile manipulation. *IEEE Robotics and Automation Letters*, 10:4252–4259, 2025.

- [28] Dong Yang, Xiao Xu, Mengchen Xiong, Edwin Babaian, and Eckehard Steinbach. Sri-graph: A novel scene-robot interaction graph for robust scene understanding. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2023-May, pages 8171–8178. Institute of Electrical and Electronics Engineers Inc., 2023.
- [29] Rasoul Zahedifar, Mahdieh Soleymani Baghshah, and Alireza Taheri. Llm-controller: Dynamic robot control adaptation using large language models. *Robotics and Autonomous Systems*, 186, 4 2025.
- [30] Youcai Zhang, Xinyu Huang, Jinyu Ma, Zhaoyang Li, Zhaochuan Luo, Yanchun Xie, Yuzhuo Qin, Tong Luo, Yaqian Li, Shilong Liu, Yandong Guo, and Lei Zhang. Recognize anything: A strong image tagging model. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1724–1732, 2024.