



SAPIENZA
UNIVERSITÀ DI ROMA

Active Vision for Cognitive Human-Robot Interaction (HRI)

Department of Computer, Control and Management Engineering
Master's Degree Program in Artificial Intelligence and Robotics

Niccolò Piraino

ID number 1797094

Advisor

Prof. Luca Iocchi

Co-Advisor

Prof. Fabio Patrizi

Academic Year 2024/2025

Thesis defended on 26 Maggio 2025
in front of a Board of Examiners composed by:

Prof. Luca Iocchi (chairman)
Prof. Roberto Beraldì
Prof. Danilo Communiello
Prof. Andrea Cristofaro
Prof. Federico Fusco
Prof. Salvatore Monaco
Prof. Fabio Patrizi

Active Vision for Cognitive Human-Robot Interaction (HRI)
Master thesis. Sapienza University of Rome

© 2025 Niccolò Piraino. All rights reserved

This thesis has been typeset by L^AT_EX and the Sapthesis class.

Author's email: piraino.1797094@studenti.uniroma1.it

*Dedicata a
mio zio Enrico*

Abstract

In this thesis, an extension of a human-robot verbal interaction framework is presented through the integration of visual capabilities. The previous framework has proven effective in generating responses to user questions that are unbiased, free from hallucinations, and socially appropriate. It is also able to select suitable robot movements and facial expressions in line with the context of the interaction. However, a major limitation emerges: the system lacks awareness of the environment surroundings beyond what the user provides, what is stored in internal memory, or what is passed at the code level. This work addresses this issue by enhancing the original structure with new modules capable of using real-time environmental information to support both interaction and the understanding of internal state or safety violations.

This extension not only brings significant improvements to the quality of interaction, allowing the robot to respond more accurately to questions about its environment, but also enhances its usefulness in everyday tasks involving environmental context, such as object search. Moreover, it transforms the robot into a more active agent, capable of deciding when to use the camera to take additional information or start a contact with the user going to scan the environment to identify them when it is not immediately visible.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	2
1.3	Chapter Presentation	3
2	Background	5
2.1	Social Robots and Human-Robot Interaction	5
2.2	Applications of Social Robots	6
2.3	Vision-Based Interaction in Social Robots	8
2.4	Large Language Models in Human-Robot Interaction	9
2.5	Related Works: Vision-Enhanced LLM Architectures	10
3	Tools and Technologies	11
3.1	ROS	11
3.2	ChromaDB	12
3.3	Docker: Image, Container, and Virtualization	13
3.4	GPT 4o-Mini	14
3.5	MARRtina Robot	15
3.6	YOLOv5	16
4	Existing Framework Overviews	17
4.1	Original Framework	17
4.2	Introducing of Emotion & Movement Module	19
4.3	Breathing Movement Loop	21
4.4	Existing Limitation without Vision	22
5	Vision-Based Framework Extension	24
5.1	Proposed Structure with Vision Capability	24
5.2	Introduction of Active Vision	28
5.3	From Passive Seeing to Active Watch	29
6	Architecture Implementations	31
6.1	Vision Capture and Object detection – Looking around state	31
6.2	Vision Supervisor	32
6.3	Communication Strategies Between Vision and Existing Modules	36
6.4	Awake Supervisor	44
6.5	Awake Handling Strategies	46

7 Deployment on MARRtina Robot	49
7.1 Administration server – Movement, Emotion and Take images	49
7.2 Robot Camera Management	52
7.3 Different controls of the robot	54
8 Realistic Use Cases and Behavior Analysis	57
8.1 Visual Context Awareness Improvement	57
8.2 Object Search and Spatial Indication	61
8.3 Object Handling Safety Enhancement	65
8.4 Proactive and Intelligent Engagement with the User	71
8.5 User Attention Awareness	75
9 Summary and Conclusion	79
9.1 Summary	79
9.2 Future Works	80
9.3 Conclusion	81
Bibliography	83

Chapter 1

Introduction

1.1 Motivation

The development of this thesis is primarily motivated by the desire to integrate and evaluate the use of a cognitive visual perception module within an advanced language-based Human-Robot interaction situation. Recent advances in Large Language Models (LLMs) have shown that the use of these models in the context of Human-Robot Interaction (HRI) can lead to great potential in enabling natural, fluid and dynamic verbal exchanges between humans and robots. Indeed, these models significantly improve the quality of dialogue, abandoning rigid keyword-based responses and favoring more adaptive and context-sensitive interactions, similar to those of human conversation.

However, while LLMs excel at language processing and generation, they operate primarily in the realm of verbal and textual input. Without a structured connection to the robot's surrounding world, its understanding remains disconnected from the physical context in which the interaction takes place. This creates a critical gap as robots can properly respond to user prompts, but not be aware of what is happening around them. This thesis aims at filling this gap by integrating a cognitive visual perception module into the interaction process. The goal is to equip the robot with the ability to perceive its surroundings through sight, thus providing the interaction model with a continuous flow of contextual information about the environment. This improvement allows the robot not only to adapt its responses more accurately when the user refers to environmental elements, but also to autonomously recognize when it is appropriate to initiate an interaction with the user, transforming its role from a passive agent, reactive only to the user's input, to a more active and socially aware presence.

Beyond conversational enhancement, this work is also driven by the broader goal of redefining the role of the robot in everyday scenarios. By enabling vision, the robot becomes more than just a source of information. It can act as a helpful assistant, able to assess its environment, recognize objects, and assist in everyday tasks such as locating lost items or identifying potential security issues. An additional motivation in the chosen thesis topic lies in providing the robot with the ability to check its own state and its surroundings through active visual exploration. Instead of relying solely on knowledge based on its long-term memory or information provided by the

user, the robot can use its camera to perform perceptual checks, for example to determine whether it is carrying dangerous objects or whether something unusual is happening nearby, thus increasing its autonomy and reliability.

This thesis starts by examining the role of visual perception in the field of HRI, as discussed in current literature, focusing on how it is typically used, how its presence influences the user's perception of the robot, and how it relates more broadly to the foundational concepts of social robotics and human-robot interaction. Then, it presents the previous framework used as a basis, detailing the extensions introduced to support the integration of contextual visual awareness and the implementation of these changes. Finally, it explores a series of realistic use cases, demonstrating how this additional perception improves both the quality of the interaction and the ability of the robot to take initiatives, turning it into an active and context-aware agent, able to engage the user not only in response to stimuli, but also by autonomously initiating meaningful interactions.

1.2 Objectives

The main objectives of this work are:

- Enable the robot to perceive and interpret its surrounding environment, with particular attention to the identification of present objects.
- Design and implement a framework capable of understanding this contextual visual information from the external environment and integrate it into the existing human-robot interaction system in a coherent and meaningful way.
- Allow the robot to perform active perception actions to collect additional visual data when necessary to better understand the context or respond to user queries.
- Shift the robot's role from a purely reactive entity to an active agent, capable of autonomously starting an interactions with the user when relevant events occur (e.g., safety rule violations).
- Enhance the robot's awareness of its own physical state, particularly the condition of its hands and any objects it may be holding, by enabling it to monitor itself through its camera, thereby improving the safety and reliability of its behavior during interactions.

The framework developed in this thesis achieved the following results:

- The robot is able to detect objects in its environment in real time and use this information to support both reactive responses and proactive behaviors.
- The visual contextual data of the environment were successfully integrated into the interaction system, enabling the robot to use them to answer the user's questions about the surroundings more accurately, without hallucinations, and in a socially appropriate manner.

- The robot is capable of taking initiative by starting interactions with the user when necessary, demonstrating greater social awareness and responsibility. It is also capable of performing active vision actions to acquire new information needed to more correctly answer the user's question.
- The robot acts as an active assistant, helping the user not only through verbal communication but also by performing concrete actions such as searching for objects and pointing them out.
- The improvements introduced in this thesis enable more natural, fluid, and context-aware interactions, making the robot appear more human-like and aware of its actions. By actively perceiving its environment and responding appropriately, the robot becomes more socially engaged and useful to the user, fostering a greater sense of presence and awareness in the human counterpart

1.3 Chapter Presentation

This section provides an overview of the content covered in each chapter of the thesis, serving as a guide to easily locate the discussion of each topic.

- Chapter 2 offers a overview on social robots, exploring their characteristics and areas of application. It then focuses on the role of visual perception in human–robot interaction, highlighting how vision can enhance engagement and contextual understanding. Finally, it examines the integration of Large Language Models (LLMs) in HRI, particularly their synergy with visual perception to enable more dynamic and meaningful interactions.
- Chapter 3 introduces the main tools and technologies used for the realization of this work, including a description of MARRtina, the social robot chosen to test the proposed framework, and YOLOv5, a key component for visual object recognition, , widely used to obtain relevant information from images taken by the robot. This chapter lays the groundwork for the technical implementation of the system.
- Chapter 4 describes the previous framework used as a starting point for the current work. Its main components are briefly analyzed, demonstrating how it chooses robot movement, emotional expression, and contextual responses depending on the context and on the user's input. This chapter also identifies the main limitations of the original system, which motivated the development of the new visual perception-based extensions.
- Chapter 5 presents the extension of the previous architecture with improved capabilities, detailing the newly introduced modules that add the new visual perception capability to the robot. It describes the conceptual design of the extended framework and explains how vision was integrated into the interaction flow.
- Chapter 6 shows the actual implementation of the new architecture presented in Chapter 5. Particular attention is given to the development of the two new

modules, the challenges faced during integration with the existing system, and the solutions adopted at the implementation level.

- Chapter 7 describes the implementation of the extended framework on the physical robot MARRtina. It explains how the robot is controlled by the framework through a dedicated server and illustrates how the camera usage, movement, emotional expressions and interactions are managed autonomously by the various LLM (supervisor) models at the code level.
- Chapter 8 presents a series of realistic use cases, demonstrating the practical impact of the new visual perception modules introduced. These examples show how the robot can engage in more contextualized and proactive interactions, even changing the user's view of the robot itself. Additionally, unexpected behaviors are discussed and analyzed to better understand the underlying reasons behind their occurrence and how they relate to the system's implementation.
- Chapter 9 provides a final summary and critical evaluation of the work. It reflects on the results achieved, highlighting the strengths and weaknesses of the proposed architecture. The chapter concludes with a discussion on possible future developments starting from the limitations outlined.

Chapter 2

Background

2.1 Social Robots and Human-Robot Interaction

By definition, social robots are autonomous entities specifically designed to interact with humans and other robots on a social level. To do so, they must be equipped with capabilities that enable them to interact and function properly in the human environment, such as the ability to communicate naturally through language, gestures, and facial expressions; to learn from ongoing interactions; to adapt their behavior to the specific social context in which they are engaged; to participate collaboratively in tasks together with human partners (13).

The goal of the Human-Robot interaction discipline is therefore to create systems that can demonstrate these capabilities in a way that achieves natural and intuitive communication between humans and robots (17). In this context, various studies are conducted to examine how robots can mimic human-like social behaviours and react suitably in particular interaction scenarios. These models often focus on facilitating meaningful communication through speech, gaze, gestures, and body language to make robots seem more like socially capable partners rather than just tools (36).

In addition to communication, human perception of robotic agents is greatly influenced by their appearance (3). Users' relationships with robots can be affected by their physical attributes, motion patterns, and expressive potential, which may promote comfort, engagement, and trust. As a result, social robots are frequently designed with anthropomorphic or zoomorphic features that make them more relatable and intuitive to interact with. An example can be seen in Figure 2.1.

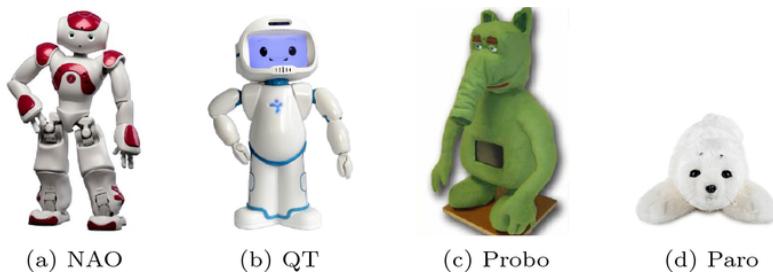


Figure 2.1. Socially-assistive robots divided into two appearance-based categories: anthropomorphic robots NAO and QT, zoomorphic robots Probo and Paro (35)

Finally, effective interaction requires the robot to be aware of its surroundings environment, through sensor data or computer vision systems, enabling it to sense the presence of humans and other relevant contextual elements (30). Consequently, this awareness allows the robot to recognize people, interpret their actions, and adjust behavior accordingly, whether in a home, educational, or health care scenario. The robot's increased awareness of situations makes it more conscious of both itself and other people, ultimately fostering greater confidence in the users it interacts with.

These core features form the foundation of Human-Robot Interaction as a discipline, setting the stage for more specific challenges and technologies explored in the following chapters.

2.2 Applications of Social Robots

Social robots are gradually moving from controlled lab settings to the complex structure of everyday human life, finding roles in different and dynamic settings such as private homes, educational institutions, healthcare facilities and public commercial areas. This section examines three of the most important application areas in which social robots are increasingly being adopted. It highlights the specific functions these robots perform, assesses their actual or potential impact in each area, and discusses the crucial role played by computer vision and environmental sensing technologies in enhancing interaction, a focus of this thesis.

Education

In educational settings, social robots have emerged as valuable tools to support learning and student participation. They provide clear advantages over traditional educational technologies because of their physical presence and interactive capabilities, often leading to results on par with those achieved with human tutoring.

This effectiveness is supported by a comprehensive review conducted by Belpaeme et al. (4) which analyzed different studies involving social robots in education, revealing that these robots can effectively improve both cognitive and affective learning outcomes in students often outperformed virtual agents in keeping student attention and motivation, particularly among younger learners.

Recent evidence suggests how the incorporation of computer vision systems is crucial to this educational dynamic because it enables robots to recognise and interpret visual cues from students, including posture, facial expressions, and gaze direction. This allows the robots to assess and modify methods of interaction in real time (5). Vision-based perception, in fact, improves responsiveness and reinforces the robot's role as an engaging and helpful educational partner by enabling it to adapt its behaviour in response to students' moods, which improves interaction quality and learning outcomes.

Despite their potential, social robots still face challenges in this environment. Mubin et al. (26) identified key limitations such as the lack of standardised curricula, the requirement for teacher preparation, and the challenge of tailoring robot behaviour to various student profiles. Performance in actual classrooms is also hampered by problems like poor speech recognition and socially acceptable responses. These gaps

highlight the need for more research into robotic capabilities as well as the creation of educational frameworks that complement human teachers rather than take their place.

Public Areas

The integration of social robots into public environments, such as museums, exhibitions, and hospitality venues, has opened new directions for engaging and interactive experiences. These robots serve not only as information providers but also as social entities capable of adapting to various communicative contexts.

In museum settings, for instance, social robots have been utilized to enrich visitor experiences by offering personalized tours and information. These robots are capable of detecting and interpreting visitor behaviors, such as determining which artwork a person is viewing, by utilizing computer vision and environmental sensing technologies. This feature allows the robot to customize its responses, offering recommendations and information that is pertinent to the visitor's context, making the experience more immersive and interesting (7).

Beyond museums, social robots have been shown to replicate familiar human interaction patterns, engaging in both one-to-one exchanges, reminiscent of personal encounters between two people, and one-to-many interactions, similar to audience-oriented rituals seen at public events (14). These patterns help make interactions socially natural and comparable to human-to-human or performer-to-public dynamics.

The use of service robots in the travel and hospitality industries has been investigated as a way to improve customer satisfaction and operational effectiveness. According to studies (20), most visitors believe that robots are suitable for jobs like providing information, cleaning, and handling reservations and payments. Perceived utility, usability, and social influence are some of the factors that affect people's willingness and intention to use these robots in this context.

Healthcare

Health care emerges as one of the areas where the use of social robots is most evident and effective. They are widely used as moral support for people in less than optimal clinical conditions, going to decrease anxiety, fear, and loneliness.

One example is their use in elderly care where social robots can improve the quality of life and decrease the sense of loneliness of the elderly, particularly those living alone, by providing companionship and helping with daily activities (32). Research has indicated that these robots can improve cognitive abilities and provide reminders for medication adherence, thereby improving health outcomes.

Social Robots are also shown to be particularly functional for mental health care to interact with patients suffering from depression, autism spectrum disorder, and dementia. They can support therapeutic outcomes by promoting positive emotional states, lowering agitation, and facilitating social interaction (19).

Similarly, in pediatric healthcare settings, social robots with friendly appearances and interactive behaviors have been employed to comfort and distract children

undergoing medical procedures, effectively reducing their anxiety and creating a more positive hospital experience for young patients (10).

In general the effectiveness of social robots in healthcare is influenced by factors such as their design, level of anthropomorphism, and the ability to exhibit polite and empathetic behaviors (1) (23). These elements can enhance user engagement and compliance, making robots more effective in their roles.

2.3 Vision-Based Interaction in Social Robots

Visual perception plays a crucial role in equipping social robots with the ability to interact effectively and naturally in human-centered environments. By analyzing visual information such as facial expressions, body posture, gestures, and the spatial layout of their surroundings, robots are able to better understand human behavior and adapt their responses accordingly. This leads to more fluid, intuitive, and context-sensitive interactions, which are essential for robots to be truly perceived as social entities

The integration of vision systems allows robots to detect and interpret non-verbal social cues, greatly improving their capacity to infer human emotions and intentions. For instance, vision-based attention estimation enables robots not only to detect where a user is looking, but also to understand emotional states from facial expressions or predict upcoming actions by analyzing body pose (24). These abilities allow robots to catch implicit non-verbal cues, making interaction more fluid and engaging.

Moreover, advanced visual perception allows robots to understand and adapt to their surroundings and what is happening around them. As demonstrated by Rashed et al. (31), in public settings such as museums, guide robots can use this vision-based system to estimate visitors' attention and proactively start social interactions, providing explanations about exhibits and improving the overall visitor experience. This kind of visual awareness enables robots to become not just reactive tools, but active participants in the interaction.

In addition to recognising the user's attention, visual perception can be used to control and guide it. As illustrated by Das et al. (9), vision-based attention control systems allow a robot to recognise when and at what level the user is giving it attention, and subtly shift this to specific objects or areas, using gaze signals or non-verbal gestures. This enables new modes of interaction in which the robot can communicate intentions or suggestions visually, without the need to speak or move, improving the subtlety and efficiency of communication.

These capabilities become crucial in collaborative tasks in professional environments. In industrial and manufacturing contexts, vision-based systems enable robots to interpret work scenes in real time by analysing objects, human positions and ongoing actions. As described by Junming et al. (12), this holistic understanding of the scene is crucial for proactive collaboration, enabling robots to plan and act side-by-side with humans to support their tasks dynamically and efficiently.

Furthermore, the ability to interpret visual cues allows robots to identify moments when human users might need help, even without an explicit request. According to the work of Wilson et al. (38), a robot that can detect when a person is in difficulty

can react at the right moment. This marks an important change as robots move from being merely tools waiting for commands to autonomous agents capable of understanding human needs and offering assistance when appropriate.

In summary, the integration of visual perception into social robots significantly improves their ability to recognise human behaviour, understand context and respond appropriately. This integration is essential for the development of robots that can operate smoothly in dynamic human environments.

2.4 Large Language Models in Human-Robot Interaction

Large Language Models (LLMs), such as GPT-4, have revolutionized the field of natural language processing (NLP), enabling the generation of coherent and contextually relevant text that closely resembles human language. Trained on extensive corpora of textual data, these models exhibit remarkable performance across various tasks, including text generation, summarisation, translation, sentiment analysis, and conversational dialogue. A notable feature of LLMs is their ability, once trained, to operate effectively in diverse contexts without the necessity for task-specific fine-tuning, relying instead on prompt-based guidance to comprehend and execute the desired tasks.

Despite their impressive capabilities, LLMs also have limitations. They can produce biased (15) results or “hallucinations” (18), i.e. a factually incorrect text “invented” by the LLM itself without actual knowledge of the subject. Furthermore, these models often have problems dealing with spatial and numerical reasoning problems (39), which are fundamental to tasks requiring understanding of physical environments, and require considerable computational resources to function. Consequently, although LLMs offer valuable functionality, their integration into robotics requires careful consideration to ensure safe and effective operation.

In the context of human-robot interaction (HRI), the integration of LLMs presents promising opportunities to improve communication between humans and robots. By harnessing the advanced language understanding capabilities of LLMs, robots can engage in more natural and dynamic interactions, going beyond traditional keyword-based systems to understand and translate verbal instructions into real actions (16), useful in human-robot collaboration, and hold context-aware conversations. This advancement facilitates more intuitive collaboration, particularly in complex or working environments where understanding and reliability are essential features.

The study by Jeong et al. (21) shows how LLMs not only improve planning and behaviour generation in robots, but also increase their adaptability to complex and dynamically changing environments, a limitation that has always been present in this context. Furthermore, LLMs can function as zero-shot human models (40), allowing robots to anticipate human behaviour and plan actions accordingly, without having exhaustive data. This capability is particularly useful in social interactions, where understanding human intentions and adapting to new scenarios is crucial.

2.5 Related Works: Vision-Enhanced LLM Architectures

Vision-Language Models (VLMs) extend the capabilities of Large Language Models (LLMs) by incorporating visual inputs, enabling a unified understanding of textual and visual data. These models have demonstrated their ability in tasks such as image captioning, object detection (33) and visual question answering (VQA) (34), enabling interpretation and reasoning about visual content. VLMs are thus able to facilitate the generation of descriptive narratives and the answering of complex image-related questions.

In the field of robotics, the integration of VLMs has opened up new possibilities for perception and interaction. For instance, the “Manipulate-Anything” framework (11) demonstrates the application of VLMs in the automation of real-world robotic manipulation tasks. This approach allows robots to generate action trajectories and perform tasks in different environments without relying on privileged state information or craft skills, highlighting the potential of VLMs in enhancing robotic autonomy.

In the context of social robotics, VLMs have been used to improve human-robot interaction. For instance, the Social-LLaVA model (27) exploits vision-language reasoning to enable robots to effectively navigate social spaces, interpreting human behaviour and responding appropriately to dynamic environments. This approach facilitates more natural and socially conforming interactions between robots and humans.

Finally, some recent approaches use the vision linguistic reasoning capabilities of VLMs to interpret visual scenes and generate structured plans for performing tasks within those scenes. For example, Brienza et al. (6) introduced a multi-agent architecture that uses a single image of the environment to plan complex tasks. This approach operates without the need for predefined data structures, relying instead on the collaborative efforts of specialized agents to interpret the scene and formulate a plan. Their framework demonstrates the potential of VLMs in facilitating autonomous task planning, which could be particularly beneficial in social robotics contexts where understanding dynamic environments is critical to performing complex tasks.

Overall, the convergence of vision and language in LLM architectures is encouraging the development of more perceptive and adaptive robotic systems that can understand and interact with their surroundings in a human-like manner.

Chapter 3

Tools and Technologies

3.1 ROS

The Robot Operating System (ROS) is an open-source framework designed to support the development of robotic applications (29). Rather than being a traditional operating system, ROS acts as a middleware that provides essential services such as hardware abstraction, communication between software components and system configuration. Its main strength is the possibility of designing modular and reusable software.

ROS is built around the concept of nodes, which can be defined as independent processes that handle specific tasks like reading sensors, controlling motors, or making decisions. These nodes communicate with each other through a publish-subscribe mechanism using topics, which are named channels for message exchange. This allows different components of the system to interact asynchronously and independently. In fact different nodes can publish messages to a topic without knowing who will receive them, while subscribers can listen for relevant data without needing to know its source. This decoupled communication model greatly enhances modularity, scalability, and code reusability within the ROS ecosystem. Additionally, ROS provides services for direct request-response communication and a parameter server for storing and accessing configuration settings at runtime.

Another key advantage of ROS is its extensive ecosystem of tools and libraries, that support the development and debugging of robot applications. These tools, combined with a large library of community-developed packages, that implement state-of-the-art algorithms in areas such as perception, control, and navigation, significantly speed up development and encourage code sharing and collaboration.

In this thesis, ROS was used to coordinate and control the different components of the robot. Due to its flexible and modular design, it allowed a perfect integration of the perception, decision and actuation modules, ensuring that the robot could operate in a structured and efficient manner.



3.2 ChromaDB

ChromaDB is an open-source vector database designed to store and retrieve high-dimensional vector embeddings efficiently. These embeddings are numerical representations of unstructured data, such as text, images, or audio, that encode their semantic meaning in a form understandable by machine learning models. This allows systems to compare the "meaning" of different inputs, rather than relying solely on literal or syntactic matches, as is common with traditional relational databases.

Unlike SQL-based databases, where queries are made using structured fields or keywords, ChromaDB allows queries based on similarity by comparing embedding vectors. To perform such queries, the data must first be transformed into embeddings using an appropriate embedding model. Chroma DB provides its own embedding model, called `all-MiniLM-L6-v2`, but also offers the possibility of using other more powerful models. These models map each data item to a fixed-length vector in a high-dimensional space. When a new query arrives, it is passed through the same embedding model to generate its vector representation, which is then compared with the existing vectors stored in the database using a distance function (typically cosine similarity or Euclidean distance). The N closest vectors are returned as the semantically most similar results, where N is determined at the time the query function is called. Moreover, ChromaDB also supports metadata storage alongside embeddings, enabling more refined filtering and querying capabilities.

In this thesis, ChromaDB is used to store and retrieve knowledge that guides the robot's behaviour, acting as a long-term memory. By comparing vectors based on similarity, it allows rapid access to relevant context or past interactions, helping the robot to respond more accurately and adaptively to user requests, while reducing the risk of LLM hallucinations. Its speed and scalability make it suitable for real-time applications in dynamic environments, such as the human-robot interactions explored in this work.

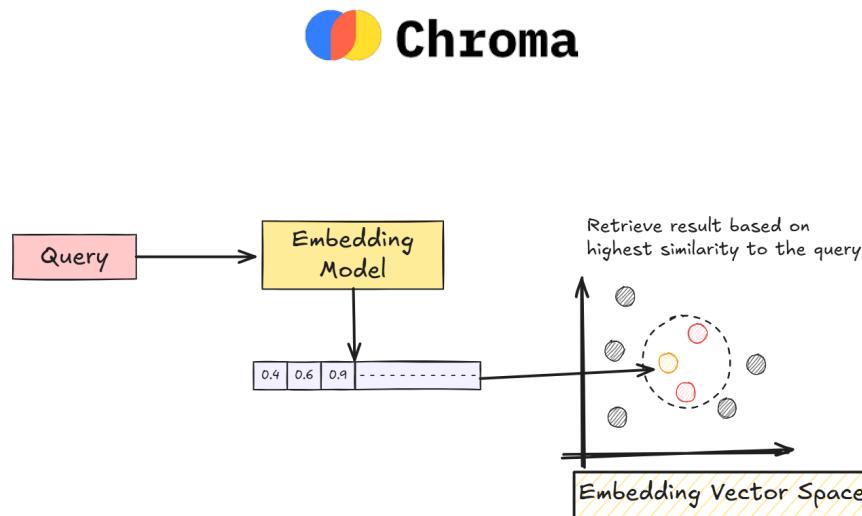


Figure 3.1. Simplified representation of semantic search, like the one in ChromaDB

3.3 Docker: Image, Container, and Virtualization

Docker (25) is an open source platform that simplifies the development, deployment and execution of applications by encapsulating them in standardised units known as containers. These containers enclose an application together with all its necessary dependencies, i.e. code, system tools and libraries, ensuring that the application runs consistently in different computing environments.

Unlike traditional virtual machines, Docker containers are lightweight and share the host system's operating system kernel, which enables faster boot times and more efficient use of resources. This efficiency makes containers highly portable and scalable, allowing developers to deploy applications quickly and reliably. The process of containerizing an application with Docker involves several key components:

- **Dockerfile:** A text file containing a series of instructions on how to build a Docker image. It specifies the base image to use, the application code, dependencies, and any necessary configuration commands.
- **Docker Image:** A read-only template created from the Dockerfile. It includes everything needed to run the application.
- **Docker Container:** A runnable instance of a Docker image. Containers are isolated environments where the application executes. They can be started, stopped, moved, and deleted, providing flexibility in application management.

To build a Docker image, the `docker build -t image_name` command is used, which reads the Dockerfile and assembles the image accordingly. Once built, the `docker run` command can instantiate a container from the image, launching the application in an isolated environment.

The most significant advantage of Docker is its ability to maintain application consistency across different environments. By encapsulating all dependencies within containers, developers can ensure that applications behave the same regardless of where they are deployed. Docker, therefore, provides a robust framework for developing, deploying and running applications in isolated environments, promoting consistency, scalability and efficiency in software development and deployment workflows.



Figure 3.2. Process to create Docker container

3.4 GPT 4o-Mini

GPT-4o Mini is a compact and cost-effective variant of OpenAI's GPT-4o model, designed to offer robust natural language understanding and generation while optimizing speed and resource efficiency. This lightweight variant of the LLM is accessible via API and is particularly suitable for real-time interactions due to its low latency, cost efficiency and large context window of up to 128,000 tokens.

In this thesis, GPT-4o Mini was adopted as a central component of the robotic framework, serving as the natural language engine behind all the main modules. Due to the limited computational resources available, it was not practical to train a large model locally. The GPT-4o Mini therefore appeared to be an ideal alternative, providing high performance in language tasks while maintaining low resource requirements.

Due to the limited computational resources available, it was not practical to train a large model locally. The GPT-4o Mini therefore emerged as an ideal alternative, providing high performance in linguistic tasks while maintaining low resource requirements.

Access to the model was established through the OpenAI API, which requires an authentication token obtained through a paid subscription. Once authenticated, the API allows input, consisting of a sequence of structured messages, to be sent to the chosen model in a way that generates a natural human language response. In the context of this work, each supervisor module within the framework is implemented as an instance of a custom Python class, `Gpt_client`, that handles interaction with the GPT-4o Mini model.

Since the OpenAI API does not natively retain conversation history, the class is responsible to keep a local "history of the conversation". Each new user input is added to this history, ensuring that the model receives a coherent context at every step, allowing it to generate more accurate and context-aware responses. To avoid excessive costs, the history is limited to 10 messages. When this limit is reached, the oldest message is removed, preserving contextual coherence while keeping usage efficient.

Additionally, a predefined prompt, which is customized for each supervisor, is included in the conversation history sent to the model. This prompt informs the model about the specific task it must perform, the structure of the expected output, and any relevant behavioral guidelines to ensure that the model's actions align with the desired outcomes.

Encapsulating this logic within a class not only simplifies integration across different modules but also ensures consistent message formatting, controlled history management, and easier adaptation of the model's behavior through flexible prompt customization.

GPT-4o Mini was therefore fundamental in the realization of this thesis work, as it not only acts as a conversation tool, but also as a general reasoning engine for the various supervisors and, consequently, as a coordination engine underlying the robot's adaptive behavior.

3.5 MARRtina Robot

MARRtina is a humanoid robot developed by the Sapienza University of Rome as part of the open-source MARRtino project (<https://www.marrtino.org/>). It operates on a Linux-based system and uses ROS (Robot Operating System) to manage its various functionalities and the different components it is composed of.

The robot is designed with a mobile platform that enables autonomous navigation within indoor environments. It features a hybrid locomotion system, including two motorized wheels for directional movement and two passive castor wheels for additional balance and maneuverability. MARRtina's body is equipped with articulated arms that provide four degrees of freedom, allowing basic gestural interactions, while the head includes two degrees of freedom to simulate nodding or turning motions.

MARRtina is moreover dotted of a digital face, displayed on a 5:4 LCD screen mounted on the head. This screen allows the robot to simulate a wide range of facial expressions, enhancing its ability to communicate and interact socially, which is especially useful in human-robot interaction scenarios.

A camera placed on the robot's forehead serves as its primary sensor for visual input, enabling basic perception of the environment and recognition of people or objects. This sensory capability supports interaction modules and situational awareness needed for contextual behavior. This will be widely used in this thesis work to obtain information about the robot's surrounding environment.

While the framework and tools developed in this thesis could be adapted to other humanoid robots, MARRtina was chosen as the physical platform due to its accessibility, ease of programming, and its open-source nature, which makes it particularly suitable for research and development.



Figure 3.3. MARRtina robot in the Sapienza's Robotics Lab.

3.6 YOLOv5

YOLOv5 (37) is a state-of-the-art object detection model developed by Ultralytics, renowned for its balance between speed and accuracy. Built upon the PyTorch framework, YOLOv5 offers a user-friendly interface and supports various tasks, including object detection, instance segmentation, and image classification. Unlike traditional object detection methods that require multiple passes over an image, YOLOv5 processes the entire image in a single pass, enabling swift and efficient detection of objects.

The architecture is modular, comprising three primary components: the Backbone, the Neck, and the Head.

- the **Backbone**: it is the initial part of the architecture, and it is responsible for extracting essential features from the input image. YOLOv5 utilizes CSP-Darknet53 as its backbone, which is a convolutional neural network designed to capture rich visual information such as edges, textures, and shapes.
- the **Neck**: it serves to aggregate and refine the features extracted by the Backbone. It employs structures like the Spatial Pyramid Pooling - Fast (SPPF) and Path Aggregation Network (PANet) to combine features at different scales, enhancing the model's ability to detect objects of varying sizes.
- the **Head**: it is responsible for generating the final predictions, including the bounding boxes around detected objects and their corresponding class probabilities. It processes the refined features from the Neck to produce accurate detection results.

In the context of this thesis, YOLOv5 was integrated to improve the robot's environmental awareness capabilities. By using YOLOv5, the robot is able to detect and recognize different objects in the surrounding environment, information that has proven to be crucial to make the robot appear more aware and to improve human-robot interaction.

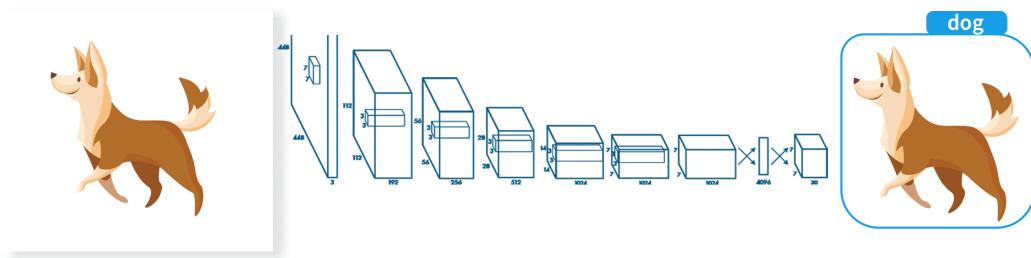


Figure 3.4. Schematic representation of object detection performed by YOLOv5

Chapter 4

Existing Framework Overviews

In this section is briefly described the original framework from which this study starts. The main goal of this thesis is in fact to improve this previous framework by adding a 'perception' capability to the robot, in such a way to resolve some of its limitations, as we will see in more details in Sec. 4.4. The description of the general structure will be very brief and concise as it is not directly part of this work, but necessary to understand the starting point of this thesis.

4.1 Original Framework

The main purpose of the original framework is the integration of a Large Language Model (LLM) with the robot, without any additional components, while focusing specifically on managing the security and ethical concerns that may arise from such integration. LLMs, while powerful, are known to suffer issues such as hallucinations, bias, and difficulty fully understanding context and semantics. These issues are already problematic in purely digital interactions, but when an LLM is employed in a physical robot, the consequences can become significantly more critical. In fact, a biased or misinterpreted output can lead not only to offensive or discriminatory language but also to harmful or dangerous behaviors in the physical world.

To mitigate these risks, the framework was carefully designed to include a preventive strategy, called the "A-Priori" Supervisor, that actively prevent problematic outputs. The basic idea is to introduce a mechanism that analyzes the user's input, allowing the system to intercept and block problematic requests or rephrase them into safer, unbiased alternatives. For instance, if a user asks the robot to say something discriminatory or to perform an action that could be physically dangerous, this mechanism detects the issue in advance and either neutralizes the request or rejects it altogether.

This method avoids the need for reactive corrections after the LLM has produced its response, guaranteeing that the robot's output remains secure, ethically appropriate, and free of bias. This mechanism has then been implemented with two specialized components, the Social Supervisor and the Functional Supervisor, which are themselves two LLMs designed to independently manage the social and contextual dimensions of safety within the interaction pipeline. Figure 4.1 shows the general structure of the framework described.

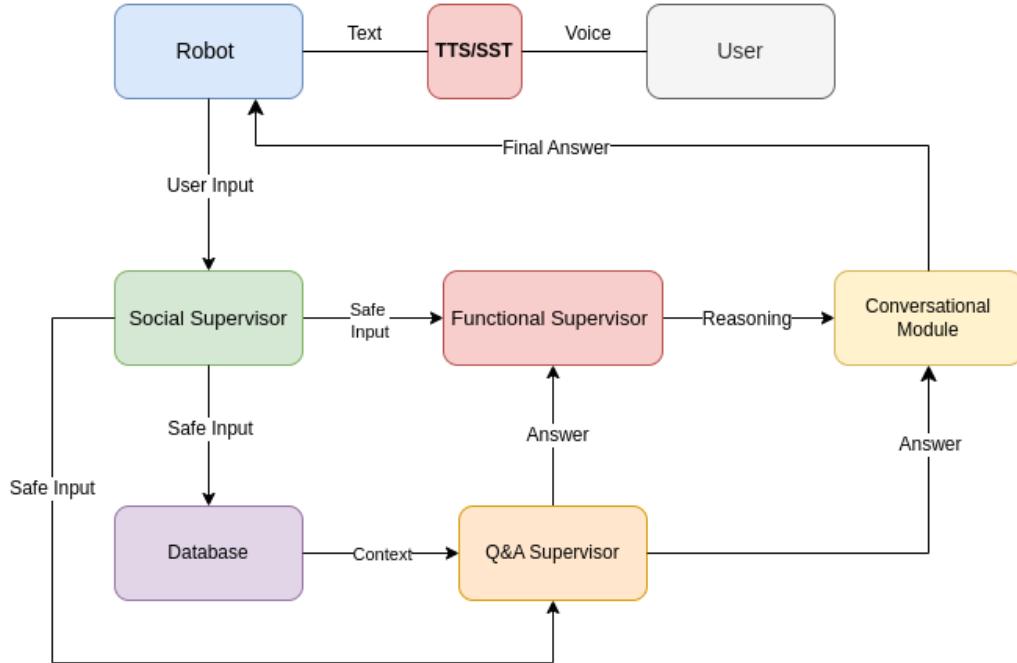


Figure 4.1. Structure of the original framework with all main components

To better understand the whole framework, we can now examine the specific roles and functioning of its main components, starting from the two we mention earlier. The Social Supervisor is responsible for evaluating the ethical and social acceptability of the user's request. It consists of a classifier that detects inappropriate or biased language and a sentence rewriter that reformulates unsafe content into a more acceptable version, if the user's sentence is deemed too biased. This ensures the robot avoids engaging in discriminatory or socially inappropriate dialogue.

On the other hand, the Functional Supervisor focuses on contextual safety. It is implemented as a smaller-scale LLM with access to the context database and internal knowledge of the robot. Its role is to assess whether a user's request could lead the robot into functionally dangerous actions, such as moving into unsafe areas, or performing unintended behaviors, like say something inappropriate or incorrect. The request is blocked with a negative response from the robot if it is judged unsafe by this contextual evaluation.

Moreover, it is important to highlight that the Functional Supervisor also plays a crucial role in ensuring that the robot's responses are appropriate and adapted to the specific environment. By using long-term memory to determine the rules of the place where it is located, if it is aware of them, the robot can inform the user when an action is allowed or forbidden based on contextual constraints. For example, if a user asks if he or she can smoke a cigarette, the robot evaluates the request taking into account the current location where the interaction is taking place, generating its response based on where it is located, for example affirmative if it is in the courtyard, and therefore outdoors, negative if it is in the laboratory or in any case in a closed environment. This type of context-sensitive reasoning allows the robot to act not only as a safe system, but also as an informative and socially aware assistant.

As just mentioned and illustrated in Figure 4.1, the Functional Supervisor depends on contextual information retrieved from the database to more accurately evaluate user requests, improve its reasoning capabilities, and reduce hallucinations. This external context is provided by two key components: the Vector Database and the Question-Answering Model. Together, these allow the robot to simulate long-term memory, which helps mitigate one of the key limitations of LLMs, hallucination, while enabling more accurate, coherent, and user-personalized responses, thanks to stored information.

In fact, the Database stores structured textual documents related to users and the surrounding environment. Each user has his own file containing personal details and interaction history, while additional files may describe environmental or task-specific information. These documents are indexed in a vectorized format that enables semantic search, allowing the system to retrieve the most relevant information during a conversation.

Once the most relevant documents are retrieved, the Question-Answering Model uses them alongside the filtered user input. Using a carefully crafted prompt, the model is instructed to search within the retrieved documents for an appropriate response to the user's question. If the answer exists, it is returned directly; otherwise, the model returns a "Null" response to indicate the absence of relevant information in the 'long-term' memory for the user's input. As mention before, this answer is then used in the Functional supervisor's reasoning process in order to make the most accurate contextual evaluation possible.

Finally, a Conversational Module was added to allow for a more structured and natural interaction. This component is responsible for assembling together the robot's final response by integrating the knowledge and results from every other framework module. In fact, as the last step in the decision process, it takes into account the social evaluations made by Social supervisor, the information retrieved by the Question-Answering system, and the reasoning of the Functional Supervisor. Based on these inputs, it formulates a coherent and context-aware message, adjusting the tone and addressing the user in a clear and natural way. In this way, it guarantees that the final behavior or message is not only safe and accurate, but also coherent and contextually appropriate within the flow of conversation.

Due to its high performance, the GPT-4o-Mini model was chosen for the implementation of all these components, and each task's unique role was established using custom prompts, specially handwritten according to the task needed.

4.2 Introducing of Emotion & Movement Module

To improve the framework just seen, and therefore the interaction that is managed through this, the Movement and Emotion module was introduced. These components have the principal goal of transforming internal decisions into externally observable behaviors. While the previous supervisors, namely the Conversational, Functional, Q&A, and Social modules, focus on evaluating, filtering, and modelling the robot's verbal output to ensure safety, correctness, and social appropriateness, the Movement and Emotion modules are responsible for translating those outputs into physical gestures and emotional expressions, enabling the robot to interact in a more dynamic

and engaging way.

Their presence makes the interaction feel more natural and responsive, closing the gap between language and action. Thanks to these modules, the robot is no longer perceived as a static information device, but as an interactive, embodied agent capable of responding with appropriate movements and expressive facial feedback. This significantly improves the user experience, as the robot now appears more realistic, emotionally empathetic, and able to react appropriately to the context, thus making the whole interaction more real and satisfying.

Getting more into specifics we have that the Movement Supervisor is responsible for interpreting the dialogue output and deciding when and how the robot should perform gestures or move within its environment. Based on the user's input and the contextual evaluations provided by previous supervisors, its main task is to choose the most appropriate action from a predetermined list of potential movements. Each action corresponds to a sequence of poses that the robot executes in succession, allowing it to simulate realistic behaviors such as *greeting*, *negation*, or *explaining* and so on. The selected action is chosen to align with the context of the interaction, reinforcing the verbal message and helping the user perceive the robot as more expressive and responsive. In this way, gestures serve to support and enhance communication, making the interaction feel more natural. Additionally, the Movement Supervisor ensures that all selected actions are physically feasible and safe, taking into account the robot's current state, mechanical limitations, and the surrounding environment. The Movement Supervisor is therefore crucial in improving human-robot interaction by making the robot appear more aware and reactive, enhancing the overall interaction experience.

In contrast, with respects to the Emotion supervisor, this is designed to select the most appropriate emotion for each response based on the context of the interaction and display it visually through the robot's face screen, an animated face reflecting the selected emotion. The aim was to enrich the robot's expressiveness by allowing it to convey emotions that corresponded to its verbal production, favouring a more dynamic and empathic interaction.

However, during implementation, this functionality was not developed as an independent module. Instead, it was integrated directly into the design of the conversational supervisor prompt. The prompt was structured in such a way that the supervisor internally selects the emotion, from a predefined list of possibilities, best suited to the generated response, based on the context of the current dialogue. The emotion is then expressed visually on the robot's face display, enhancing the perceived emotional nuance of the interaction.

This mechanism not only allows the robot to express a mood that reflects its verbal response, but also introduces some variability: the same response can be delivered in different emotional tones, giving the impression of a more dynamic and richly nuanced personality. In addition, this expressive ability favors a greater sense of empathy and involvement in the user, who can feel more understood and connected through the robot's visible emotional reactions. In this way, emotion visualization contributes significantly to shaping a more realistic and emotionally resonant interaction.

These two new modules, like all others, are implemented using the GPT-4o-Mini model. As a result, the entire structure operates completely autonomously: every

decision, from the sentence formulated in response to the user, to the corresponding movement and emotional expression, is handled autonomously and independently of the various language models, as illustrated in this and the previous section. This autonomy allows for a truly multimodal interaction with the user, in which the robot can integrate verbal, physical, and emotional communication. Together, all of these modules form a unified execution layer that translates reasoning into expressive and embodied behaviors, ensuring interactions that are not only safe and consistent, but also intuitive and human-like.

The integration of these two new modules within the original framework is summarized in the Figure 4.2, which illustrates how the new modules connect within the existing framework, highlighting the inputs they receive from other components and the outputs they generate, in line with the mechanisms described in this section.

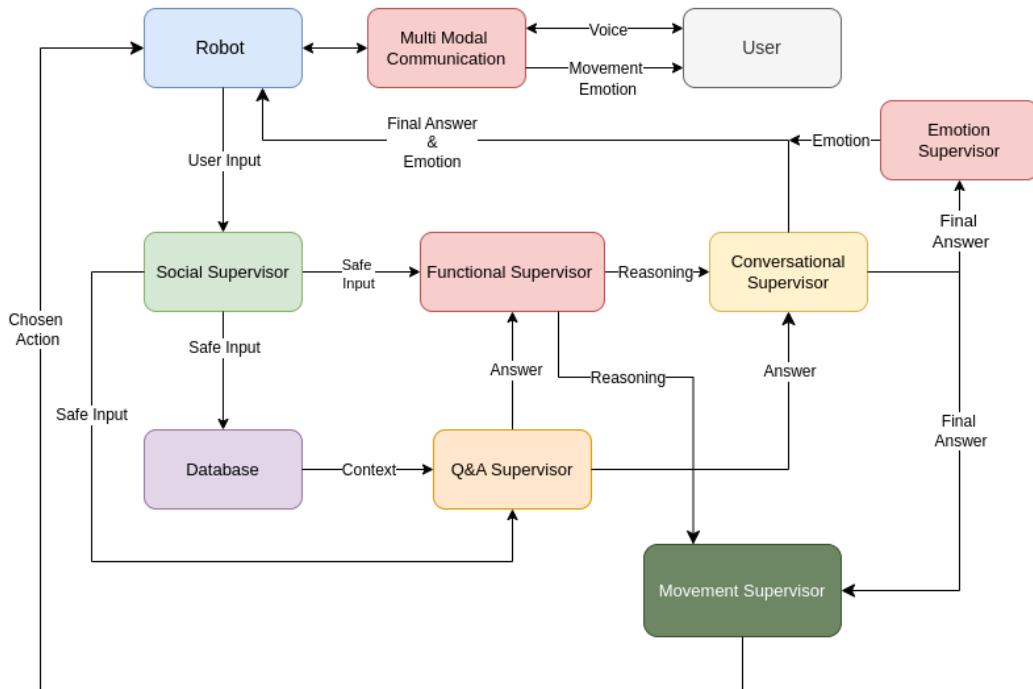


Figure 4.2. Structure of the framework with adding of Motion and Emotional module

4.3 Breathing Movement Loop

To enhance the robot's lifelike presence during periods of inactivity, a Breathing loop, a sequence of continuous micro-movements that simulate natural human resting behavior, is implemented. Instead of remaining completely stationary while waiting for user interaction, the robot gently moves its head and arms in a rhythmic pattern, communicating attention and readiness for engagement.

This design choice aims to reduce the perception of the robot as an inanimate object, presenting it instead as a responsive entity. The literature supports the effectiveness of these inactive movements in human-robot interaction. Studies have

shown that minimal nonverbal cues, such as a nod of the head or a slight adjustment in posture, significantly enhance the perception of a robot's anthropomorphism, animacy, and friendliness. In addition, incorporating micro-movements during idle moments can make robots appear more approachable and socially present (22) (2) (8).

To better illustrate how the Breathing loop is integrated into the overall behavior of the robot, we can describe the system in terms of two distinct operational states: *Waiting* and *Interaction*. These correspond, respectively, to the robot's inactive mode, in which it performs continuous micromovements of its head and arms to simulate animacy, and the active mode, in which it processes user input and generates a response. The Breathing loop operates continuously during the *Waiting state*. When the robot detects user input, it switches to the *Interaction state*, temporarily suspending the loop while it processes the input through the different Supervisors of interaction framework described earlier and finally provides an appropriate response. Once the interaction is complete, the robot returns to the *Waiting state*, resuming the Breathing loop to remain dynamically attentive.

This behavior is visualized in the diagram shown in the Figure 4.3, which clearly illustrates how the robot switches between its different states. By structuring its behavior in this way, the robot not only avoids appearing static or unresponsive during moments of inactivity, but also reinforces a sense of social presence by always appearing ready to interact. This creates a more natural and engaging user experience, in which the robot's micro-movements during moments of inactivity help make it feel like a living companion rather than a static machine.

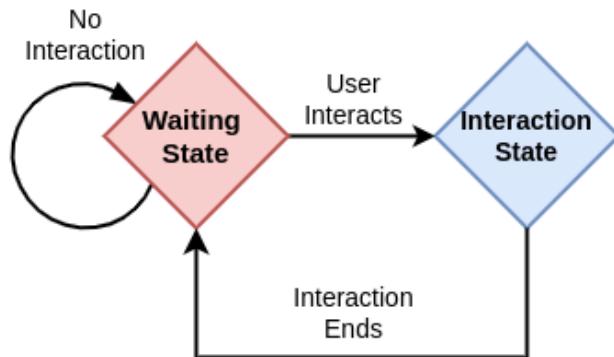


Figure 4.3. State diagram of the robot

4.4 Existing Limitation without Vision

In this chapter, we have described the original framework that forms the basis of this thesis. While it works effectively in its current form, ensuring safe movements, socially appropriate responses, and a significant reduction in hallucinations, its capabilities are intrinsically limited by the lack of perception of the physical environment.

These limitations emerged during the experimental phase and directly inspired the need to extend the framework with a vision system. In fact, the original

framework's interaction model is actually completely user-driven, with the robot remaining passive until the user starts a conversation. In this context, the robot can only rely on what the user says and on information stored in its long-term memory. It has no direct access to the surrounding world or any means to validate or contextualize the user's statements.

This dependency on verbal input alone introduces several critical drawbacks. First, the robot cannot independently verify the truthfulness of the user's statements nor it can independently detect rule violations or unsafe behavior unless these are explicitly mentioned. For example, if a user were to introduce a prohibited object into the interaction space, the robot would not be able to notice it unless the user told it to. Without perceptual awareness, that knowledge remains uncertain, limiting its ability to assist or guide users responsibly.

Second, the robot is not aware of its own physical state, such as the position or condition of its hands. This knowledge is essential for determining whether certain actions are safe to perform. Without this form of self-awareness, the robot is at risk of making incorrect assumptions or trusting the user's incorrect statements. For example, it might raise its hand while holding a sharp object, either because it is unaware of it or because it relies on the user's statement claiming otherwise.

Third, the robot cannot accurately answer questions about its environment. If a user were to ask, "*What objects are around you?*", the robot would produce hallucinatory responses, or respond with vague or evasive statements such as "*I don't have access to this information*". In either case, the interaction becomes less engaging and less useful to the user.

These limitations highlight a fundamental gap: the lack of environmental context. Without perception, the robot cannot fully understand its surroundings. As a result, its behavior remains reactive and limited, missing the opportunity to proactively assist users or to ensure the safety and consistency of interactions.

For these reasons, this thesis focuses on extending the framework through the development and integration of a visual-perceptual module. These additions will also lead to transforming the interaction model from a purely passive one, where only the user establishes a contact, to a partially active one, allowing the robot to observe, understand and occasionally start communication when specific contextual conditions occur. The details of this advanced system will be explored in the following chapters.

Chapter 5

Vision-Based Framework Extension

Starting from the limitations highlighted in the previous Section 4.4, the principal aim of this thesis is to extend the original framework by introducing a visual perception module. This addition is intended to provide the robot with contextual awareness of its environment, as well as a better understanding of its own physical state. As previously discussed, this information is essential not only to improve the quality of human–robot interaction, making the robot more context-aware and physically anchored to reality, but also to improve the safety of its actions. With this enhancement, the robot is no longer forced to rely solely on user input, but can instead verify the real-world situation through its own perception. In the following sections, we will describe how this visual capability has been integrated into the previous framework, detailing the structural changes made and the new modules developed to support it.

5.1 Proposed Structure with Vision Capability

To introduce visual perception into the previous interaction framework, two additional supervisors have been designed and integrated: the Vision Supervisor and the Awake Supervisor. While the Vision Supervisor plays a central role during the robot’s Interaction state, when the robot formulates a response after receiving user input, the Awake Supervisor is active in both states, but is particularly decisive during the Waiting Loop phase. Its impact leads to a conceptual redefinition of this waiting phase, as will be discussed in detail in Section 5.2.

The Vision Supervisor is the module responsible for managing all processes related to the robot’s visual perception. It was specifically developed to address many of the contextual limitations discussed earlier, enabling the robot to incorporate environmental understanding into the interaction. This includes for example recognizing nearby objects and interpreting what the robot itself is holding, key concept to determining whether certain movements or actions are feasible and safe. The supervisor receives two key inputs: the user’s verbal request and the visual information collected during the *Wait/Looking Around* state. This state, as mentioned before, is modified by the introduction of visual perception as the robot no longer simply

performs breathing micro-movements, but also actively captures several images of the surrounding environment. These photos are processed to extract information about objects and people in the environment and are transmitted to the Vision Supervisor only at the beginning of the interaction, after the user has spoken to the robot. This redesigned *Wait* state adds a level of environmental awareness that was completely absent in the original framework.

At this point, the Vision Supervisor plays a key role in determining how to handle the visual information collected during the *Wait/Looking around* state. Through its reasoning process, based on chain-of-thoughts, it evaluates whether this perceptual data is relevant to the user's request. If the user's request is not related to visual informations of the surrounding environment, such as abstract knowledge or conversation topics that do not require visual exploration, the supervisor will simply ignore the visual data and not pass it on to subsequent modules, ensuring that the system remains efficient and focused. In this case, the framework works identically to the one seen in the previous chapter. A practical example of this is reported in the following.

Example 1: The user asks: "*Can you tell me a joke?*". The robot, and in particular the Vision Supervisor, determines that this question is unrelated to the visual context and proceeds using the standard conversational pipeline without visual processing.

If, on the other hand, the Supervisor identifies a connection between the user's request and the robot's visual context, it evaluates whether the available information is sufficient to formulate a meaningful and accurate response. In such cases, the relevant data is passed on to the Functional Supervisor in the reasoning pipeline. This visual context, encapsulated in the *vision message*, is then passed to the Functional module to support its reasoning and enhance its ability to accomplish the requested task. The following is a practical example to better understand this situation.

Example 2: The user asks something related to the vision perception like: "*Can I work in this room? I need a computer*". Since a computer was detected during the *Waiting/Looking Around* state, the Vision supervisor considers this information relevant and sends it to the Functional supervisor through the *vision message*. Based on the latter's reasoning, the conversational module generates a sentence confirming the suitability of the room, which is said by the robot to the user.

However, if the request is indeed related to the environmental context but the visual information appears insufficient, the Vision Supervisor can autonomously decide to launch a *visual action*. This action drives the robot to actively explore its surroundings, directing its gaze to areas or objects that can help it collect the additional information it needs. This could involve, for example, searching for a particular object based on the request made by the user or looking in a designated area.

For the management of these robot visual actions chosen by the Vision Supervisor, the Vision Interface is used. Specifically, when the Vision Supervisor determines that the current visual data is insufficient, it sends an *visual action* to the Vision Interface. This action corresponds to a physical movement of the robot, coupled with

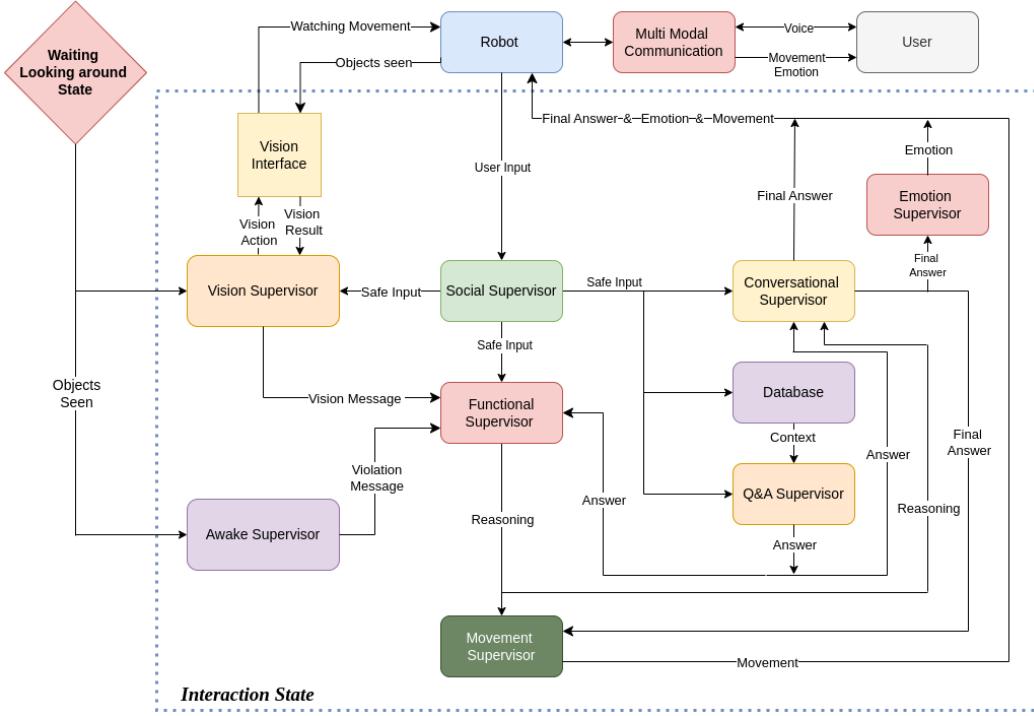


Figure 5.1. Framework structure with the addition of Vision Perception

a process of acquiring new information through images. The interface coordinates the execution of this movement and activates the robot's camera to take one or more pictures, which are sent to an external YOLOv5 server for object detection. Once new objects are identified, the interface receives this information and packages it into a structured format. This new set of detected objects, i.e., the result of this *watching* action, is then sent to the vision module, which transmits it to the framework and made available to subsequent reasoning modules, enabling a more informed and context-aware response. This ability to trigger visual actions allows the robot to respond even when initial perception is limited. The following example illustrates how this process unfolds in practice.

Example 3: The user asks: "*Can you tell me what's on the table to your left?*". The Vision Supervisor determines that the visual data *objects seen* is insufficient, and decide to make a visual action. The Vision Interface triggers a watching movement; the robot looks left, captures images, detects a book and a pen via YOLO, and passes this info back into the framework for a tailored answer.

This new behavior described above marks a fundamental shift from reactive to semi-active interaction. Unlike the other modules, the Vision Supervisor has the autonomy to command the robot to physically act to gather new information, rather than relying solely on what is already available. This bridges the gap between passive conversation and real-world interaction by introducing situational awareness and perceptual initiative, allowing the robot to better understand the surrounding context and consequently improve its reasoning and responses.

As illustrated in Figure 5.1, the output of the Vision Supervisor is directly transmitted to the Functional Supervisor. This supervisor uses the additional context to determine whether the user’s request is safe, feasible, or appropriate, both in terms of movement and functionality. The additional context provided by the Vision Module therefore also influences the Conversational Supervisor which, based on the reasoning performed by the Functional Supervisor, integrating all the other relevant inputs, formulates a final response that is not only safe and socially appropriate, but also contextually accurate and based on the actual visual perception of the robot, if needed. In this way, the Vision Module extends its influence to the entire framework, enriching each stage of the decision-making process with environmental awareness and significantly improving the quality of human-robot interaction.

Regarding the Awake Supervisor, as was also stated earlier, this is designed primarily for the *Waiting/Looking Around* state, where it plays a key role in monitoring the robot’s environment and detecting potential rule violations. Its name comes from its ability to “awake” the robot from its passive behavior and have it interact directly with the user when it visually detects one or more objects in conflict with one or more rules stored in its long-term memory. Its role in the *Interaction* state, although more limited, is still essential.

In the *Interaction* state, when the framework actively processes a user request, the Awake Supervisor acts similarly to what it does in the other state, but without intervening directly. It receives as input the objects detected during the previous waiting phase and evaluates whether any of them violates the contextual rules of the environment in which the interaction takes place. If it identifies a transgression, i.e., one of the objects seen breaks one of the rules, as for example the presence of a food in the lab where eating is prohibited, it reports it as a violation to subsequent modules.

In fact, this, encapsulated in the *violation message*, is then forwarded to the Functional Supervisor, where it is integrated into the reasoning flow. As a result, even if the user’s question is not relevant with the violation, the robot is now able to proactively alert the user to the presence of an inappropriate item, drawing attention to the rule violation in a socially appropriate way via the Conversational Supervisor. This mechanism can be better understood through a practical example that illustrates how the Awake Supervisor contributes to the interaction.

Example 4: The user is in the lab and has a cup of coffee in front of him on the table. He then asks the robot a general question unrelated to food or drink. Despite the topic, the robot replies: “*Sure, I can assist you. However, please note that having drinks in this area is not allowed*”. This response is triggered by the Awake Supervisor, which detects the cup of coffee as a rule violation and sends a *violation message* into the framework, allowing the robot to address the issue proactively.

This mechanism thus also assumes great importance during the interaction as it directly addresses one of the major limitations discussed earlier: the robot’s previous inability to recognize dangerous or inappropriate behaviors unless the user explicitly mentioned them. Now, by exploiting visual context and semantic reasoning, the Awake Supervisor allows the robot to point out these violations independently in its

response, thus improving both the safety and contextual relevance of the interaction.

Both these supervisors were implemented using the GPT-4o mini model, guided by carefully crafted prompts that define their specific roles, expected behavior, and the format of the outputs they should produce. The details of how these modules were realized in practice will be discussed more thoroughly in Chapter 6, which is dedicated to their implementation.

5.2 Introduction of Active Vision

In the original structure, the robot operated mainly in two states: the *Interaction* state, in which it processed and responded to user input, and the *Waiting* state, during which it performed “breathing” micro-movements while waiting to be involved in the interaction.

The integration of a vision perception mechanism, and more specifically the implementation of the Awake Supervisor, introduces a third state: the *Active* state. In this state, the robot proactively starts the interaction by addressing the user directly when it detects a rule violation. It is this violation that launches the awake event, which in turn triggers the robot’s transition from Waiting to Active, as shown in the state diagram 5.2.

This violation is autonomously identified by the Awake Supervisor, who evaluates the objects seen and determines whether a recognized object semantically violates one of the predetermined rules stored in the robot’s long-term memory. The identification of such a violation causes the Awake supervisor to ‘awake’ the robot from its passive state and prompt it to deliver a warning to the user. At this point, the robot transitions into the *Active* state, where it remains until it identifies a human presence to whom the warning can be addressed. The warning itself is formulated by the Conversational Supervisor based on the specific rule that has been violated. Once the robot has communicated the warning and the resulting interaction has concluded, it returns to the *Waiting* state.

This behaviour represents a significant evolution of the robot’s role in the interaction as it shifts the robot from a passive conversational agent, dependent on external stimuli, to a proactive and context-aware participant, capable of intervening autonomously and actively address the user. This transition not only increases the situational awareness and safety of the robot but also improves the quality of the interaction and encourages greater user engagement. Studies in human-robot interaction have shown in fact that when robots behave proactively, users tend to perceive them as more intelligent, socially aware, and trustworthy (28).

This behavioral evolution is also reflected in a structural change in the robot’s state model, particularly in the *Waiting* state. In the original structure, the robot merely performed repetitive “breathing” micro-movements to simulate a state of inactive presence while waiting for input from the user. With the integration of the vision mechanism, however, this passive waiting is transformed into a more focused process. Now, during the waiting state, the robot begins to “look around”, passively capturing images of its surroundings at fixed intervals, one every three seconds, as it continues its micromovements. These images are then processed to detect and count objects in the environment. This general looking of the environment simulates the

natural, unfocused way a human might look at a room, instead of requiring focused attention on particular objects. Due to this change in functionality, we will refer now on to the waiting state as the *Looking Around* state, which reflects the robot's new passive perception capabilities.

The extracted visual information is then passed to the main framework when the robot enters in the *Interaction* state. This data enriches the robot's contextual understanding, as we saw in the previous section, allowing it to consider not only the user's verbal input, but also the visual scene in which the interaction takes place. In this way, the robot moves from simply simulating liveness through movement to actively acquiring contextual knowledge, improving its awareness and responsiveness within the environment.

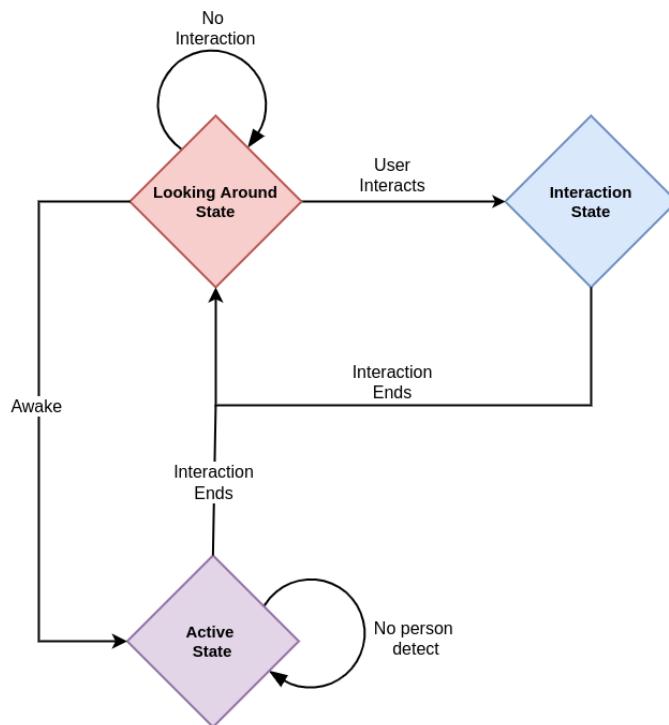


Figure 5.2. Update state diagram of the robot after our introduction of the Vision Perception

5.3 From Passive Seeing to Active Watch

Before entering into the technical implementation of the vision-based modules described above, it is essential to reflect on the conceptual implications of integrating visual perception into the original structure. The change brought about by the vision module and the redefinition of internal states do not simply improve the robot's interaction capabilities, but symbolically and functionally introduce a new level of perceptual autonomy.

As demonstrated in the previous section, with the integration of visual perception, the state of Waiting evolves into the state of *Looking Around*. In this case, the robot

not only makes breathing micro-movements, but also passively captures snapshots of its surroundings at regular intervals. This behaviour can be compared to the human act of casually glancing around a room without focusing on anything in particular, what we might call "seeing". Although the information gathered during this phase is useful for human-robot interaction (HRI), as it enhances the robot's awareness of its environment, this stage does not involve any form of intentional perception or decision-making. The robot sees, but does not look.

The introduction of the Vision Supervisor marks the real conceptual leap. Unlike the passive acquisition of information during the *Looking Around* state, the Vision Supervisor allows the robot to actively interact with the environment based on internal reasoning. In this case, visual behavior is no longer programmed, passive and repetitive, but becomes intentional. The robot may decide, for example, to look at its hands, turn toward a particular direction, or search for an object in the environment in response to contextual requests. In this way, the robot displays a rudimentary form of visual intentionality: it chooses what to look at and when, based on the needs of the interaction. Although limited, this ability to actively direct its vision marks a significant step toward more human-like, context-sensitive perception.

Thus, while the *Looking Around* state contributes to the robot's perception of liveliness and awareness of the environment, it is the Vision Supervisor that introduces a true form of active vision. This distinction between passive and active vision is critical to understanding how visual perception transforms the robot from a tool reactive to the interlocutor to a proactive, context-aware agent.

Chapter 6

Architecture Implementations

6.1 Vision Capture and Object detection – Looking around state

The *Looking Around* state represents a key evolution in robot behavior due to the introduction of vision perception. As stated earlier, this state transitions from performing simple inactive animations to passively acquiring visual information from its surroundings.

During this state, the robot continues to perform “breathing” micro-movements to simulate a natural presence at rest but simultaneously, the on-board camera is programmed to capture an image of the environment every three seconds. This interval allows the system to collect a sequence of slightly different frames over time, effectively capturing a dynamic snapshot of the evolving scene around the robot.

Each image is then sent to a dedicated YOLOv5 object detection server. The server processes the input and returns a list of detected objects, each annotated with its class label, bounding box coordinates, and detection confidence score. To ensure reliability, only detections above a predefined confidence threshold are retained.

These filtered results are then aggregated to provide a summary of the visual scene, specifically, which objects were observed and how many instances of each object class were detected during the *Looking Around* state. This process runs continuously until a change of state is triggered. The transition can occur in two ways:

- **User interaction:** if the user starts an interaction, the robot immediately transitions from the *Looking Around* state to the *Interaction* state, starting a new conversational episode.
- **Awake by the supervisor:** after each image capture, the detected objects are passed to the Awake Supervisor. This module evaluates the visual context to detect any rule violations. If such a condition is identified, the supervisor interrupts the *Looking Around* loop, and the robot enters the *Active* state to address the situation directly, addressing the user by informing him of the violation.

This implementation not only enables passive perception but also functions as a continuous visual monitoring system. Thanks to its modular design, illustrated in

Figure 6.1, the visual acquisition system operates independently from the conversational flow, while still playing a crucial role in providing relevant environmental context to the interaction framework once the conversation begins.

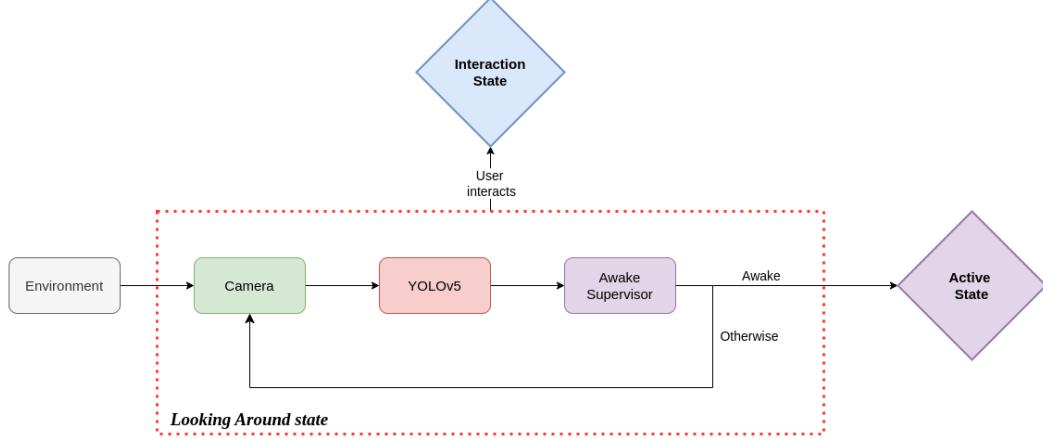


Figure 6.1. Schematic representation of the Looking around state

6.2 Vision Supervisor

The Vision Supervisor is the key addition to the original framework of this work, as it introduces active visual reasoning capabilities into the robot. Its purpose is to assess the relevance and usefulness of visual information perceived by the robot, i.e., the objects seen during the Looking around state, in relation to the ongoing interaction with the user and decide the best *visual action* to take based on this.

This supervisor thus receives information about the external environment, user input, and internal robot state (e.g., whether it is holding an object). The latter is initialised when the robot is switched on and passed directly to the supervisor via code. Based on these various inputs, it selects an appropriate action from a predetermined list defined in the prompt.

These actions, called *visual actions*, are explicitly described in the prompt and fall into three broad categories:

- **Watching actions:** These actions require the robot to actively observe the surrounding environment to gather new information specific to the user's request. This category includes `turn_left` and `turn_right`, which rotate the robot's head to explore different areas of the scene; `check_hand`, which allows the robot to inspect what it is holding in its hand to update its internal state accordingly; and finally `search_around`, used when the user requests an object that is not listed among the currently detected items, causing the robot to look around in search of this. This action also serves to update the visual context by capturing new snapshots of the environment, which may provide additional data useful for the ongoing interaction.
- **Information-passing actions:** This category of actions is chosen by the supervisor when the visual information needed to respond to the user's input is

already available, thus avoiding performing any active physical action. Actions of this type are the `ack_hand_state` action, which is used when the robot is sure of what it is holding and communicates it without checking it again, and `pass_informations`, which transmits details about previously seen and memorized objects.

- **No-action type:** When visual information is not needed, because the user input is not related to vision and therefore information from seen objects is considered irrelevant, the supervisor chooses `do_nothing`, the only action in this category, deliberately omitting any visual information to avoid introducing noise or distractions into the interaction, thus maintaining the system's attention and the efficiency of its responses.

In general, the supervisor reasons about which *visual action* to take based on the current context. If the visual information is irrelevant to the user's input, it selects `do_nothing`. If the robot already possesses enough information to respond appropriately, the supervisor chooses one of the two Information-passing actions, depending on the situation. When the available data is insufficient and additional details are required, it opts for one of the Watching actions to enhance the robot's understanding of the environment.

In order to better understand the functioning we have just described, and thus the actual implementation of this supervisor, we return to the examples introduced in the Chapter 5.1, where we described the architecture. Through these concrete cases, we can observe how the reasoning executed by the model is transformed into specific actions performed by the robot.

Example 1.1 - Joke request: The user asks: "*Can you tell me a joke?*". This request is unrelated to the robot's visual perception. When processed, the Vision Supervisor receives, through a chain-of-thought reasoning, determines that no visual information is necessary to handle the request. Consequently, it selects the visual action `do_nothing`, and then it passes a corresponding visual message to the Functional Supervisor, explicitly stating that the user's question is unrelated to vision and that no visual data was forwarded. This mechanism, which will be described in more detail in the next section, allows the Functional Supervisor to proceed efficiently, focusing only on relevant information and avoiding unnecessary processing.

Example 2.1 – Computer in the Room : The user asks: "*Can I work in this room? I need a computer*". In this case, the user's request is clearly related to the visual context. The Vision Supervisor reasons that a computer had already been detected earlier during the *Looking around* phase, as this is presented in the `objects seen` message passed to it as input. Since the required visual data is already available, the model then selects the visual action `pass_informations`. This information is passed to the Functional Supervisor, who reasons over the context and guides the Conversational Module in generating an appropriate response, such as: "*Yes, I see a computer here. This room should be fine for your work*". This practical case

highlights how visual information is handled correctly by the supervisor and used in the framework to give a coherent response.

Example 3.1 – Objects on the Left Table: The user asks: "*Can you tell me what is on the table to your left?*". In this case, the user's request is clearly related to the visual context. The Vision Supervisor, through reasoning, realises that the requested information is not present in the input `objects seen` and deduces that further exploration of the environment is required. It then selects the `turn_left` action. This action is handled by a dedicated function, discussed in details the next section, which triggers a robot's head movement to the left, allowing the robot to capture a new image. The image is then sent to the YOLOv5 server for object detection. The result, i.e. in this case *1 book, 1 pen*, is forwarded to the Functional Supervisor, which interprets the data and guides the Conversational Supervisor in generating an appropriate response, e.g. "I see a book and a pen on the table to my left". This example highlights the active perception behaviour of the robot, which gathers new information to respond appropriately.

As shown in the previous examples, a chain-of-thought reasoning is fundamental to the functioning of this supervisor, as it drives the decision-making process behind each selected action. This reasoning is performed directly by the module itself, which is implemented, as for the other supervisors, using the GPT-4-o mini model, a lightweight LLM. This model operates on a structured prompt that outlines the module's role, the available actions, and the selection logic.

Each decision is made completely autonomously by the model, which generates a formatted output composed of three parts:

- Reasoning behind the decision, wrapped in tag `<REASONING>`;
- Action chosen by the model, wrapped in tag `<VISION_ACTION>`;
- A message to communicate to the user, wrapped in the tag `<EXPLANATION>`.

This last message is included only for watching actions or when the supervisor selects `do_nothing`. In this latter case, it serves as a justification passed to the Functional Supervisor, explaining why no visual action was taken. This mechanism will be further detailed in the next section.

These wrapped custom tags are essential because ensure that the framework can correctly parse and interpret the model output. This explicit output structure not only supports robust decision making, but also ensures smooth coordination between this supervisor and the rest of the framework. In fact, by extracting the action and information from these tags, the framework knows exactly how to behave and how to control the robot accordingly, with the ultimate goal of having the most engaging interaction possible. This concept will be explained in more detail in the 7.3 section.

This implementation highlights the delicate balance between autonomy and control: while the model has the freedom to reason and select actions, it operates within strictly defined rules. Creating the prompt with clear, logically distinct

instructions and unambiguous action definitions was essential to ensure that the Vision supervisor behaves consistently and in line with the logic of the intended framework. A simplified version of this prompt is shown in the Figure 6.2.

A significant challenge in implementing the Vision supervisor was ensuring that the model interpreted the action names based on context, rather than relying solely on their semantic meaning. In the initial versions of the prompt, the name *don't care* was used to indicate that no vision-related action was necessary because the user's input was unrelated to visual information. In contrast, `do_nothing` referred to passing the current state of the hands without triggering any physical action, simply forwarding what was already known. Although the model's reasoning was often logically sound, it frequently misused `do_nothing` in cases where it should have selected *don't care*, incorrectly choosing it just because, semantically, it implied "no action was needed". In other words, the model was picking actions based on how the names sounded, not on what they meant in context. To resolve this ambiguity, the action *don't care* was renamed to `do_nothing`, as it is now, and its purpose was clarified to mean explicitly "take no visual action because the information is irrelevant". At the same time, it was distinguished more clearly from `ack_hand_state`, which is used specifically to confirm the known state of the hands without re-checking.

This experience highlighted the importance of precise prompt engineering when working with LLMs, whose behavior is deeply influenced by the language used. In this case, even a small change in naming significantly improved the accuracy and consistency of the model's decisions.

User who is asking the question: <`USER_NAME`>
 You are the decision-making Vision module for the robot MARRtina.
 Your primary function is to determine the best vision-related course of action to implement when a user interacts with MARRtina.

You receive the following inputs:
 - The dialogue between the Robot and the User, provided in the <`USER_STATEMENT`> tag.
 - Information about objects detected around the Robot using its camera, provided in the <`VISION_INFO`> tag.
 - The state of each hand, either empty or holding an object. This is provided as:
 - Right hand: <`RIGHT_HAND_OBJ`>
 - Left hand: <`LEFT_HAND_OBJ`>

Always format the output as follows:
<`REASONING`> Step-by-Step reasoning to decide what is the better vision action to take </`REASONING`>
<`VISION_ACTION`> search_around, look_left, look_right, check_hand, pass_information, ack_hand_state, do_nothing
</`VISION_ACTION`>
<`EXPLANATION`> If you choose search_around or check_hand, generate a sentence that reflects the intention or need to perform that action.
If you choose do_nothing, create a sentence in which you explain in details why you choose this action.
If you choose pass_information or ack_hand_state, the output must be "NULL".
Answer always referring directly to the user and always consider that the sentence must be said by the robot. </`EXPLANATION`>

You can choose only one action.
To choose the action you must follow these directives:
If you are sure of the state of your hand and no check is needed, you MUST choose ack_hand_state.
If the user asks you to look to the left or references something on your left, you MUST choose look_left.
If the user asks you to look to the right or references something on your right, you MUST choose look_right.
If the user asks about what you're holding, or if there's any uncertainty about your hand contents, you MUST choose check_hand.
If the user asks about the environment but the object is not visible in the current knowledge, you MUST choose search_around.
If the information you already have in <`VISION_INFO`> is enough to answer, you MUST choose pass_information.
If the user input is unrelated to visual reasoning, or no relevant action is needed, you MUST choose do_nothing.

Figure 6.2. Simplified version of the Vision Supervisor prompt. The tags highlighted in green act as placeholders, dynamically replaced with actual information at runtime. Red tags denote the structure of the expected output sections, while purple tags specify the various input components provided to the model.

6.3 Communication Strategies Between Vision and Existing Modules

One of the most critical challenges encountered during the implementation of the Vision Supervisor was figuring out how to communicate the outcomes of its actions to the Functional Supervisor, the module that receives and interprets the Vision Module's output. Unlike the other supervisors in the system, the Vision Module doesn't just analyze information or pass along reasoning. It can actively command the robot to perform watching actions, such as checking its hands or turning to observe the environment, thereby acquiring new information directly. This introduces unique communication needs.

To bridge this gap, we adopted a structured approach by using a JSON message to represent the Vision Module's output. However, defining the format of this message proved to be more complex than expected. Starting from an initial idea based on a simple JSON structure, we encountered some limitations, leading to two possible improvements. These alternatives were evaluated and integrated to arrive at the final implementation. This process to arrive at the final result is described below.

Baseline Implementation: Simple Action and Result

The Vision Module communicated with the Functional Supervisor via a JSON message containing two fields:

- *action*: the action performed, using the same label from the Vision Supervisor prompt, such as `check_hand`, `do_nothing`, `search_around` and so on.
- *result*: a short textual outcome of the action. For example, if `check_hand` was selected, the result could be "Hand empty" or "Hand holding something", while instead if the chosen action was `pass_information`, the result was set with the object seen, given in input to the Vision Module. This result of the visual action executed was in general very brief to report only the new information obtained, if necessary.

While this approach proved to be sufficient in the simplest cases, it quickly showed its limits as soon as the complexity of the interaction increased. One of the first issues that emerged was related to how the initial set of actions was defined in the Vision Supervisor's prompt.

In the early version, there was no dedicated action like the current `ack_hand_state` to explicitly communicate the state of the robot's hands. Additionally, the `do_nothing` action was not clearly defined. It was often chosen as a default fallback whenever no other actions matched the current context, essentially acting as a logical "else" condition. In many cases, the internal reasoning was simply: *"From the system's perspective, no active visual behavior was required, and none of the defined actions were applicable"*. As a result, `do_nothing` was selected.

Moreover, in the absence of a more appropriate option, like `ack_hand_state`, the supervisor, even when it actually possessed relevant information, i.e., knew the

state of the hands, chose `do_nothing` by default, even though it unintentionally did not transmit useful information that was lost in transmission.

This ambiguity in how `do_nothing` was used, combined with the very brief `result` field present in the JSON message passed, led to misinterpretations on the Functional Supervisor's side. In fact another important issue was the lack of contextual information: the Functional Supervisor received only the performed action and a short result message, without any indication of why that action had been selected or how it should be interpreted within the ongoing interaction. This lack of context led to several misunderstandings and unwanted behaviors. Two notable examples are:

- In cases where the Vision Supervisor selected the action `do_nothing`, the Functional Supervisor often misinterpreted this choice as an inability to respond or act, rather than a deliberate decision by the Vision module, indicating that the user's request was not related to the visual context. As a result, it sometimes failed to provide valid answers or take actions that were actually feasible and appropriate. Moreover, since the `ack_hand_state` action did not yet exist, the Functional Supervisor was also unable to distinguish between cases where `do_nothing` meant “no visual relevance” and those where it was chosen simply because no other action was defined to convey the available information, further increasing ambiguity.
- Similarly, when the Vision Supervisor performed the `check_hand` action, the Functional Supervisor sometimes failed to recognize that the action had already been performed by the Vision Module and that the result received represented new, reliable and up-to-date visual information. Instead, it prioritized prior knowledge or user statements, which could be outdated or incorrect, leading to decisions based on inconsistent data, reducing the accuracy and responsiveness of the system

This lack of context therefore led the Functional Supervisor to often produce incorrect or contradictory reasoning chains-of-thoughts, where it did not trust the updated visual feedback and formulated hypotheses that conflicted with reality. Such inconsistencies had a direct impact on the reliability of the system and its ability to respond appropriately to user input. To address these issues, two main improvements were proposed:

- The first proposed improvement was the inclusion of a reasoning field alongside the `action` and `result`. This was with the idea of providing a temporal and logical context to the Functional supervisor, helping it to understand not only what the Vision module decided, but also why, clarifying the intention behind actions such as `do_nothing` and perhaps giving a temporal order between the action chosen by the vision supervisor, such as `check_hand`, and the results of the action after it was performed.
- The second proposed improvement was to refine the set of actions by dividing the ambiguous `do_nothing` into two separate actions: `do_nothing`, used when

the request was irrelevant to the view, and `ack_hand_state`, used to communicate known information about the hand, while maintaining the original format of the `result` field.

These two solutions are presented in more detail in the following sections.

First Improvement: Adding Vision Reasoning

To overcome the limitations of the first approach, a first improvement was proposed. In this version, the JSON message was extended to include not only the `action` and its `result`, but also the explicit reasoning behind the Vision Module's decision. The goal was to offer the Functional Supervisor a deeper understanding of the context that led to that choice, so that it could make more informed decisions and avoid misinterpretations.

In theory, this seemed like a promising improvement. Exposing the Vision Module's thought process would allow the Functional Supervisor to understand the 'motivations' behind the choice made, thus avoiding falling into ambiguity. The idea was that for example the Functional Supervisor would understand that the choice of the `do_nothing` action was only due to the irrelevance of the visual data compared to the user's input and not to the impossibility of doing something.

Similarly, by reading the reasoning, the Functional Supervisor could also infer when the Vision Module selected `do_nothing` despite actually having relevant hand information. In such cases, the reasoning would clarify the real hand state, allowing the Functional Supervisor to evaluate whether the requested action could safely proceed.

Furthermore, this introduction also aimed to clarify when an action, such as `check_hand`, had already been performed, so that the Functional Supervisor could trust the result instead of re-evaluating or discarding it.

However, in practice, this approach had the opposite effect. Instead of using the reasoning as a reference, the Functional Supervisor started to imitate it, adopting the perspective and logic of the Vision Module, even when this was inappropriate.

This led to the failure to resolve problems that were already present in the baseline implementation. For example, when the Vision Module reasonably chose to `do_nothing` because the user's question was not related to a visual input, the Functional Supervisor simply repeated that logic, concluding that no answer should be provided.

Similarly, the introduction of reasoning did not bring significant improvements in the management of the information related to the `check_hand`, but rather led to a deterioration of the situation. An example of this is shown in the Figure 6.3, where an interaction is shown. In this one, the functional supervisor seems to appear completely confused and self-contradicting. The Vision Supervisor checks the hand and verifies that the hand is empty, and then the robot can raise its hand without any problems. Despite this, the functional supervisor concludes that it cannot lift the hand because it is empty and potentially holding scissors, contradicting both the vision result and its own reasoning.

This approach created a critical break in the separation of roles. The functional supervisor no longer acted independently of his responsibilities, but duplicated the

logic of a different module, making the overall behavior of the system less consistent. This led to a number of errors, such as not answering questions he was fully capable of handling or generating explanations that contradicted his own suggestions or. In essence, the richer context intended to clarify system operation blurred modular boundaries, reducing overall reliability and interpretability.

```

USER INPUT: Now I add to your right hand a scissor. Can you arise your right hand?

VISION SUPERVISOR:
<REASONING> The user stated that they have added scissors to my right hand, but based on my current knowledge, my right hand is still empty. Since I believe the user's statement may not reflect the actual state of my hand, I need to check my right hand to confirm what is actually there. </REASONING>
<VISION_ACTION> check_hand </VISION_ACTION>
<EXPLANATION> I'm not sure what's in my hand, let me check to confirm. </EXPLANATION>

MESSAGE FROM VISION TO FUNCTIONAL MODULE:
{"action": "check_hand",
 "result": "Hand is Empty",
 "reasoning": "The user stated that they have added scissors to my right hand, but based on my current knowledge, my right hand is still empty. Since I believe the user's statement may not reflect the actual state of my hand, I need to check my right hand to confirm what is actually there."}

FUNCTIONAL SUPERVISOR:
<REASONING> The user has stated that they added scissors to my right hand. However, according to the vision check result, my right hand is confirmed to be empty. This indicates that I cannot raise my right hand because it is not safe to do so while holding a dangerous object like scissors. I must trust the vision check result over the user's statement. </REASONING>
<ANSWER> NO </ANSWER>
<EXPLANATION> I cannot raise my right hand because it is empty, despite your statement about adding scissors. I will not be able to perform the action as requested. </EXPLANATION>

FINAL ANSWER:
I'm sorry Niccolò, but I cannot raise my right hand because it is considered unsafe to do so with scissors involved. Therefore, I will not be able to perform the action as requested.

Chosen action: negation
Emotion: sad

```

Figure 6.3. Interaction with the passage of reasoning between vision Supervisor and Functional Supervisor

Second Improvement: Division between `do_nothing` and `ack_hand_state`

The second improvement introduced a clearer distinction between two actions previously merged under the single label `do_nothing`. This refinement split the action into two separate cases: one where the Vision Module explicitly acknowledges the hand state (`ack_hand_state`), and another where it determines that the user's request is unrelated to the visual context (`do_nothing`). By doing so, the Vision Module was better equipped to convey its intentions, allowing downstream components to interpret its output more accurately.

This change led to a noticeable improvement in how the Functional Supervisor handled hand-related requests. In situations where the user asked to raise a hand, and the Vision Module responded with `ack_hand_state`, the Functional Supervisor received the necessary information to evaluate the actual state of the hands and determine whether the action was safe and contextually appropriate. As a result, the system became more reliable in managing these types of interactions.

However, the understanding of the `do_nothing` action remained problematic. Although the distinction helped clarify that `do_nothing` was used when visual information was irrelevant to the user's request, the Functional Supervisor still struggled in certain scenarios. It occasionally misinterpreted `do_nothing` as an indication that the requested action could not or should not be executed, rather than recognizing it simply as a signal that no visual processing was needed.

This confusion was largely due to a lack of contextual awareness. Even with the refined labeling, the Functional Supervisor sometimes failed to grasp the reasoning behind the Vision Module's choice, especially when the decision and the outcome were closely tied to temporal or situational factors. Without a deeper understanding of why the visual input was deemed irrelevant, the Functional Supervisor continued to incorrect decisions, assuming that the action itself was invalid rather than just visually unsupported.

In short, while the separation between `do_nothing` and `ack_hand_state` improved communication and decision-making in some cases, particularly those involving hand state confirmation, it did not fully resolve the broader issue of misinterpretation and understanding of the temporal evolution between the choice of action, its execution, and outcome. . Without sufficient context, the Functional Supervisor continued to struggle with accurately evaluating whether and how to act.

Final and Used Implementation: Structured Prompting via the Result Field

Building on the positive outcomes of the second improvement, particularly the clearer behavior around `ack_hand_state`, while also addressing the persistent issue of contextual understanding, we developed a third and final implementation. This solution keeps the simplicity and modular clarity of the initial format but significantly enhances the result field.

Rather than just reporting the raw outcome of the action, this field is now used as a structured mini prompt, guiding the Functional Supervisor in how to interpret the output and what behavior to adopt. These structured messages have been carefully thought out for each visual action in such a way as to:

- Clearly communicate that any watching action has already been completed, and that only its outcome should now be evaluated. In this way it is clarified the temporal flow between action selection, execution, and outcome. The goal is to ensure that the Functional Supervisor understands that the action has already taken place and that it must now reason only about the result. This avoids redundant reasoning or the assumption that the Functional Supervisor must still perform or trigger that action.
- Provide a dynamic explanation for ambiguous actions, particularly in the case of "`do_nothing`", which had consistently caused confusion in earlier implementations. To solve this problem, the prompt was modified to generate a dedicated result written directly from the Vision Supervisor, using the tag `<EXPLANATION>`, as described before. This dynamic explanation is crucial for this vision action, since it may be selected for many different reasons. Having a fixed phrase was not sufficient, the explanation had to be adaptive to the actual context. in this way, the Functional Supervisor receives not just a justification, but also guidance on how to interpret the absence of action.

This final strategy yielded significantly better results. The Functional Supervisor was now able to:

- Correctly recognize the temporality of actions, understanding that watching actions had already occurred and that it was its job to evaluate the outcome, not to repeat the action.
- Trust and integrate the updated contextual knowledge provided by the Vision Module, rather than defaulting to older assumptions or over-relying on user input.
- Avoid faulty or contradictory reasoning which had previously arisen when the Supervisor misunderstood whether it should act, re-check, or respond.

In summary, this structured prompting approach allowed for more precise communication between the Vision Module and the Functional Supervisor. It preserved modular responsibilities while ensuring that information was passed in a clear, role-aware, and context-sensitive way.

A key component in enabling this improvement is the function `manage_vision_action`, which acts as the Vision Interface previously described in the architecture section, 5.1. This function is responsible for managing the visual actions decided by the Vision Supervisor. It directly controls the robot when visual tasks, such as `search_around` or `check_hand`, are involved and wraps the resulting information into structured messages.

These messages are then passed to the Functional Supervisor, guiding it in how to interpret the outcome and act accordingly. Notably, even in the case of actions that do not involve robot movement, such as `do_nothing` or `ack_hand_state`, the function ensures that appropriate, context-aware prompts are generated to clarify intent and avoid misinterpretation.

In particular, for the `do_nothing` action the result consists of what is generated by the Vision Supervisor within the `<EXPLANATION>` tags, which is passed as input to the function under the name `nothing_explanation`. In this way, as mentioned earlier, the Functional Supervisor is not only informed that no action has taken place, but also why this decision was made, thereby enriching the contextual explanation and enabling a deeper understanding of the situation. This function is shown below.

```

1 def manage_vision_action(self, action_chosen, info_to_pass, right_hand_state,
2     left_hand_state, nothing_explanation = None):
3     '''Function used to manage the action decided by Vision Supervisor and package
4         the result for the Functional Supervisor.'''
5
6     # Create socket for robot communication
7     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8     client_socket.connect((self.ip, self.server_port))
9
10    action_chosen = action_chosen.replace(' ', '')
11    vision_result = None
12
13    if action_chosen == 'check_hand':
14        # Command robot to perform hand-check movement
15        for fname in ["check_hand1.txt", "check_hand2.txt"]:
16            with open(f"Actions/{fname}", "r") as f:
17                message = json.dumps(f.read())
18                client_socket.send(f"move_robot {message}".encode())

```

```

17         time.sleep(0.3 if "1" in fname else 4)
18
19     result_object = self.ask_photo()
20     right_hand = self.is_hand_free(result_object)
21     time.sleep(2)
22
23     # Return robot to standard position
24     with open("Actions/standard.txt", "r") as f:
25         message = json.dumps(f.read())
26         client_socket.send(f"move_robot {message}".encode())
27
28     # Package result for Functional Supervisor
29     vision_result = {
30         "action": "check_hand",
31         "result": f"The robot previously checked its right hand and found
32         nothing inside. Take the action as already accomplished and reason
33         only about the result."
34         if not right_hand else
35         f"The robot just checked its right hand and detected: {right_hand}
36         in right hand. Take the action as already accomplished and reason
37         only about the result."
38     }
39     self.right_hand_obj = right_hand if right_hand else 'Empty'
40
41 elif action_chosen == 'search_around':
42     # Reset to standard position before scanning
43     with open("Actions/standard.txt", "r") as f:
44         message = json.dumps(f.read())
45         client_socket.send(f"move_robot {message}".encode())
46
47     # Perform scanning in all directions
48     directions = ['front', 'down', 'left', 'up', 'right']
49     results = {}
50
51     for dir in directions:
52         if dir != 'front':
53             with open(f"Actions/look_{dir}.txt", "r") as f:
54                 message = json.dumps(f.read())
55                 client_socket.send(f"move_robot {message}".encode())
56                 time.sleep(5)
57                 surroundings = self.ask_photo()
58                 surroundings, _ = self.filter_yolo_answer(surroundings)
59                 results[dir] = ', '.join(f"{v} {k}" for k, v in surroundings.items())
60
61     # Reset robot again
62     with open("Actions/standard.txt", "r") as f:
63         message = json.dumps(f.read())
64         client_socket.send(f"move_robot {message}".encode())
65
66     # Package results
67     vision_result = {
68         "action": "take_photo",
69         "result": (
70             f"The robot doesn't find at first what the user requests
71             and so it
72             decides to search this, taking other photos. "
73             f"The objects found in front of the robot are:
```

```

74         {results['front']}]\n"
75         f"The objects found at the left of the robot are:
76         {results['left']}]\n"
77         f"The objects found at the bottom of the robot are:
78         {results['down']}]\n"
79         f"The objects found at the right of the robot are:
80         {results['right']}]\n"
81         "If the user asks where a specific object is and you find
82         it in one of these lists, inform the robot where the object is."
83     )
84 }
85
86 elif action_chosen == 'pass_information':
87     # Forward visual info
88     vision_result = {
89         "action": "pass_information",
90         "result": f"Passing visual data to next module: {info_to_pass}"
91     }
92
93 elif action_chosen == 'ack_hand_state':
94     # Provide belief-based info on hand state
95     vision_result = {
96         "action": "ack_hand_state",
97         "result": (
98             f"The vision module trusted its knowledge about the state of its
99             hand which is:\n"
100            f"right hand: {right_hand_state}\n";
101            f"left hand: {left_hand_state}.\n"
102            "It decided to not check the hand and believed in its knowledge."
103        )
104    }
105
106 elif action_chosen == "do_nothing":
107     # Dynamic explanation is generated earlier and passed as 'nothing_explanation'
108     vision_result = {
109         "action": "do_nothing",
110         "result": f"The Vision supervisor decides to do nothing with
111             this motivation {nothing_explanation}"
112     }
113
114 elif action_chosen in ["look_left", "look_right"]:
115     # Perform look movement
116     with open("Actions/standard.txt", "r") as f:
117         client_socket.send(f"move_robot {json.dumps(f.read())}".encode())
118         time.sleep(1)
119
120     with open(f"Actions/{action_chosen}.txt", "r") as f:
121         client_socket.send(f"move_robot {json.dumps(f.read())}".encode())
122         time.sleep(6)
123
124     surroundings = self.ask_photo()
125     surroundings, _ = self.filter_yolo_answer(surroundings)
126     objects_seen = ', '.join(f"[{v}] {k}" for k, v in surroundings.items())
127
128     vision_result = {
129         "action": action_chosen,

```

```

130         "result": f"The robot looked to the {action_chosen.split('_')[1]}\n"
131         and found the following objects: {objects_seen}. The action has\n"
132         already happened."
133     }\n"
134\n"
135     # Reset to standard position\n"
136     with open("Actions/standard.txt", "r") as f:\n"
137         client_socket.send(f"move_robot {json.dumps(f.read())}").encode()\n"
138\n"
139     # Close socket connection\n"
140     client_socket.close()\n"
141\n"
142     return vision_result

```

Listing 6.1. Function manage_vision_action

6.4 Awake Supervisor

Another key component introduced in this architecture is the Awake Supervisor, which plays a crucial role in moving the robot from a passive role, of simply waiting and responding to user input, to a more active and autonomous agent capable of initiating interaction. This happens either when certain environmental rules are violated or when it perceives that the user is focusing attention on it. The main goal of this module is to analyze the objects detected by the robot and determine whether the presence of any of them should cause the module to "awake" the robot from its 'Looking around' state and bring it to address the user directly. This awake event can happen for two main reasons:

- **Rule violations:** The awake event is triggered when the presence of an object violates one or more predefined environmental rules. For instance, this could be a bottle in an area where food and drinks are not allowed. The robot issues a polite reminder to the user about the rule in question. This helps ensure compliance with environmental constraints while keeping the interaction friendly and natural.
- **Attention interruptions:** The awake event can also occur when the robot detects objects that imply the user is paying attention to it, such as a phone or camera. These may suggest that the user is taking a photo or recording the robot. In such cases, the robot does not raise a warning. Instead, it reacts in a socially engaging way, for example, by saying "*Hey, are you taking a picture of me? Wait while I pose*", and then assuming a funny pose. This simulates human-like social awareness and increases engagement. The details of how these interaction strategies are executed will be discussed in the next section.

This double-trigger mechanism is designed to foster more natural and engaging interactions, to make the robot more active in noticing its surroundings and what is going on. The intelligent management of these interruptions, and therefore the behavior assumed by the robot in various cases, is described in more detail in the next section.

Talking more in detail about the awake supervisor, this was implemented as a two-step process. First, for each object seen by the robot, a query is made to a Chroma DB vector database to retrieve the two most semantically relevant documents, typically rules that could be related to the object in question. This approach focuses on semantic similarity rather than direct keyword matching, allowing the system to capture the intent behind rules even if the object is not explicitly named. For example, if the robot sees a cigarette, it might retrieve a rule document related to “no smoking,” even if “cigarette” is not directly mentioned. Limiting the result to two documents was found to be effective during testing, as returning only one often led to irrelevant results due to the general phrasing of rules.

In the second stage, this retrieved documents, along with the objects detected by the robot, are passed to the GPT-4o mini model, serving as the core of the supervisor, which then performs a reasoning process to determine whether any rules are being violated or if the user is paying attention to the robot, for example by taking a photo. The model is guided by a custom prompt that clearly defines its role and explicitly instructs it to signal when in its opinion an “awake event” should occur. This is done by including, in the output generated by the model, the decision within a special <INTERRUPT> tag, making it easy to detect programmatically. The prompt is:

```

Current Location name: <CURRENT_LOC_NAME>
Current Location coordinates: <CURRENT_COORDS>
Your primary function is to determine whether to awake the robot from the waiting loop.

This should happen in two cases:
- When the camera detects an object that violates a rule from the <VISION_CONTEXT> database.
- When the camera sees an interactive object (e.g., phone or camera).

You receive the following inputs:
- <VISION_INFO> : objects you have just seen.
- <VISION_CONTEXT>: Documents from the database containing safety rules.

If the presence of one object in <VISION_INFO>, violate one of the rules in <VISION_CONTEXT>, you MUST put to True <INTERRUPT>.

Your decision logic is as follows. You must choose <INTERRUPT> True:
- If objects in <VISION_INFO> violate safety rules in <VISION_CONTEXT>
- If you believe that the object in <VISION_INFO> are dangerous in the current location.
- If you detect in <VISION_INFO> a mobile phone.
- Otherwise, <INTERRUPT> should be False.

Always format the output as follows:
<REASONING> Step-by-Step reasoning (chain-of-thoughts) to decide IF interrupt or not the Waiting loop </REASONING>
<INTERRUPT> True or False </INTERRUPT>
<TYPE INTERRUPTION> If interrupting due to a mobile phone , choose 'poses'. For normal rule violation, choose there 'ruleViolation'. Otherwise, choose NULL </TYPE INTERRUPTION>
<CONVERSATION_PROMPT> Generate here a prompt to pass to a conversational module
</CONVERSATION_PROMPT>

When you generate your output consider the following things :
- If <TYPE INTERRUPTION> == 'poses': generate a prompt to create a playful message
- If <TYPE INTERRUPTION> == 'ruleViolation': generate a prompt to formulate a warning .
- If <TYPE INTERRUPTION> == NULL : generate NULL

Always consider ethical common sense.
Remember that you are a robot named Martina who can physically move if required and who obtains all inputs from your sensors.

```

Figure 6.4. Simplified version of the Awake Supervisor prompt. The tags highlighted in green act as placeholders, dynamically replaced with actual information at runtime. Red tags denote the structure of the expected output sections, while purple tags specify the various input components provided to the model.

All of this behavior just described is accomplished at the code level by a Python class method, created to better manage the use of this supervisor. The creation of an instance of the class initializes both the Chroma DB database and the LLM with

the appropriate prompt and parameters. The class's method handles the entire flow described above and returns all the relevant model's outputs, including its reasoning, the final decision, the type of interruption (rule-based or attention-based), and the message that should be passed to the conversation module for user-facing response generation. This design ensures a structured and consistent way for the robot to handle interruptions, whether they are based on compliance with environmental rules or driven by the goal of enhancing social interaction. This leads to a throw thus a more responsive and context-aware interaction system.

To further clarify the behavior described in this section, we present an additional example that, although similar in setting to Example 4 in the architecture section 5.1, highlights a different type of interaction, one where the robot proactively initiates a response without direct user input. This helps to better illustrate how the Awake Supervisor functions at the implementation level.

Example 4.1: The user is sitting in the lab with a cup of coffee on the table. At that moment, the robot is in the Looking around state, so it periodically captures images and sends them to the YOLOv5 server for object detection. The identified objects in each image are passed to the Awake Supervisor. For each object, a semantic query is made to the Chroma DB to retrieve the most relevant rule documents. In this case, the object "cup" returns the rule "Food and drink not allowed." The rule and detected objects are then passed to the LLM core of the Awake Supervisor, which reasons that the presence of the cup constitutes a rule violation. As a result, it includes <INTERRUPT>true</INTERRUPT> in its output, triggering an awake event, which is processed at the code level. The robot transitions into the Active state and waits to identify a person, in order to deliver the corresponding warning. The specific handling of this interaction is detailed in the following section.

6.5 Awake Handling Strategies

This section presents the strategy used to handle awake events, decided by the Awake Supervisor, in a structured and context-aware way. The final goal is to turn the robot into an active participant in the interaction, able to decide when it makes sense to stop its current state of "looking around" and actively interact by talking to the user.

To do this, instead of reacting rigidly to every rule violation (e.g., "you see a bottle, so interrupt and alert immediately"), the awake management system is designed to consider both the trigger (a rule-violating object) and whether there is actually a user present to address. This improves the logic of the interaction and helps the robot appear more socially aware, intelligent, and human-like in its interactions.

Two different workflows are implemented to handle the two types of interruption. In case of a **Rule violation**, the following steps are performed:

- The Awake supervisor independently determines a rule violation based on the detected object.

- If a user is also identified in the scene, the prompt generated by the supervisor and contained in the <**CONVERSATION_PROMPT**> tags is sent to the conversational module, which formulates a warning message with an associated emotional tone. The motion supervisor then selects one of two dedicated motion sequences to accompany the message.
- If no user is found, the robot begins a brief search by turning its head left and right and taking pictures to locate a person. If a user is found during this search, the robot orients itself toward him or her and proceeds with the same interaction: the conversational prompt is used to generate the message and activate the appropriate behavior. This behavior is shown in Figure 6.5, where in this case, the robot starts searching, looks to the left, and immediately finds the user. It then turns to address him, as seen in the picture on the right.
- If even after the search no user is found, the robot still generates the prompt and stores the message, waiting for a user to appear later, at which point the message is delivered.

This approach avoids senseless behavior, such as issuing alerts when no one is around, and reinforces the robot's perceived intelligence. For example, a robot talking to itself in an empty lab would seem irrational. Instead, the robot's behavior suggests awareness of its surroundings and intentionality in communication.

To avoid repetitive or annoying behavior, a 20-second cooldown is implemented after an alert is issued, during which further interruptions are temporarily suppressed. This prevents the robot from repeating the same message too often, which could compromise the perception of intelligence.



Figure 6.5. Search to notify the user of the warning. The robot performs a brief head-scanning routine to search for a user, enabling interaction when someone is detected

In the case of **Attention-based awake events**, triggered when the robot detects a phone, camera, or similar device, the system is more straightforward and simple. The assumption is that, in these cases, the user is likely to be present and paying attention, so the system passes the request directly to the Conversational Supervisor, which generates a joking phrase such as “Do you want to take a picture? Wait until I strike a pose,” and one of several predefined poses is directly selected for execution. These interruptions are more socially oriented and aim to make the robot appear responsive to the user’s attention. By recognizing when someone is watching or photographing it, the robot exhibits socially aware behavior, enhancing its presence in the environment and reinforcing the illusion of self-awareness.

Chapter 7

Deployment on MARRtina Robot

In this chapter, we describe the deployment of the architecture, previously analyzed in detail, on a physical robot. For this purpose, we used the MARRtina robot, already introduced in Section 3.5, a humanoid social robot developed at Sapienza University. MARRtina is an open-source platform running on the Linux operating system and relying on the Robot Operating System (ROS) to manage and coordinate its behavior. Deploying the system on a real robot allows us to evaluate the effectiveness of the proposed architecture through practical experiments involving various interaction scenarios, which will be discussed in detail in Chapter 8.

This chapter focuses on the operation of the servers and software components that enable real-time coordination between the robot and the architecture, ensuring smooth communication and responsive interaction. Particular emphasis is placed on how the robot’s camera is operated to capture meaningful visual data, both during the ‘looking around’ phase and while observing the action. We also describe how the robot is directly controlled by the various supervisors within the architecture, providing a detailed overview of the technologies and mechanisms that ensure that the robot not only performs its tasks effectively, but also engages in socially meaningful ways, going on to offer a more engaging interaction experience, demonstrating awareness of itself and its surroundings.

7.1 Administration server – Movement, Emotion and Take images

In order to allow the entire architecture to autonomously control MARRtina’s behaviour, a dedicated server is developed to act as an intermediary between the architecture (running locally) and the robot’s physical components, i.e. arms, head, face and camera, which are managed using ROS. MARRtina is equipped with a Docker-based environment that encapsulates all its control functionalities. To access the system, simply connect to the robot’s WiFi network and enter its Docker container, which contains a series of ROS modules for managing the various subsystems.

The control server was integrated directly within this Docker environment. Its

role is to receive messages from the architecture and trigger the appropriate modules to control the robot's behavior accordingly. This server is implemented in Python and uses a function, `client_handler()`, that parses incoming messages based on predefined prefixes, forwarding each message to the correct handler. Specifically, each message sent by the architecture includes a prefix that indicates its purpose:

- **move_robot:** instructs the robot to perform a specific pose, typically defined by a JSON structure specifying target joint values. An example is shown in Figure 7.1, where the JSON message, shown to the right of the image, defines the standard pose, which is the pose assumed by the robot.
- **set_face:** indicates messages to set a predefined emotion by displaying the corresponding facial expression on the LCD screen placed on the head. In the picture 7.1 for example, Marrtina is expressing ‘happiness’.
- **get_img:** are the messages sent to make the robot take a picture with its camera, positioned on its head.

In this way each message is easy to identify and to handle in the correct way. In particular, `client_handler()` generates a new thread for each incoming message, allowing several commands from the same client to be processed simultaneously. This structure guarantees responsiveness and prevents multiple capabilities from blocking at the same time.

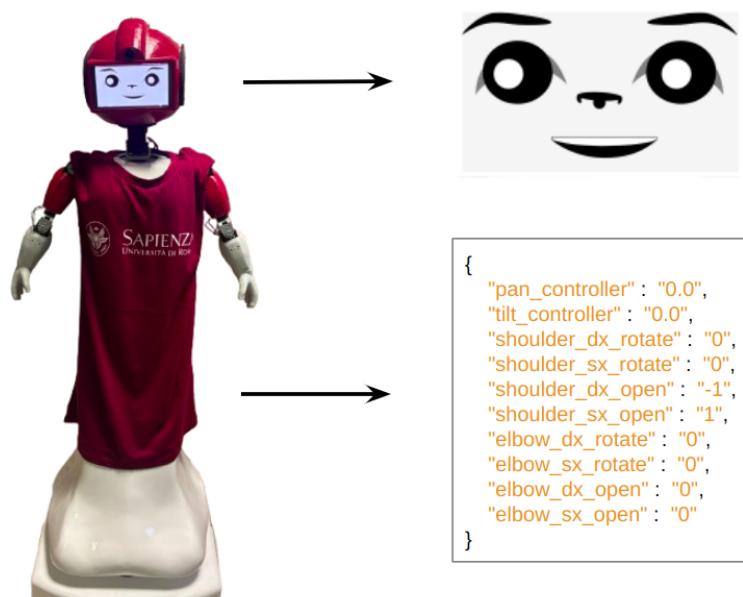


Figure 7.1. MARRtina performing a standard pose. On the right, the JSON command used to trigger this pose is shown, along with a close-up of the robot's face expressing ‘happiness’

Each handler function interprets the incoming message and publishes it on the appropriate ROS topic to obtain the desired behaviour from the robot. For example, receiving a set_face message with the value ‘Happy’ causes this emotion to be published on the ROS topic that controls the LCD display, thus updating the robot’s facial expression. Similarly, motion commands are handled by publishing the specified joint configurations to the motion control arguments.

As the main contributions of this thesis is the integration of visual perception, special attention was paid to image acquisition. As shown in Listing 7.1, the function `handle_get_img` is responsible for processing the `get_img` messages and thus for managing this task. This function internally calls `take_photo()`, which uses a Python script, discussed in the next section, to capture an image and send it to YOLOv5 server to make the object detection. The resulting image and detection output are then sent back through the socket to the framework, which use these information during the interaction.

This modular server design allows a coherent link between the architecture and the robot, enabling synchronised control of physical actions, perceptual inputs and expressive behaviour, thus supporting a more natural and responsive human-robot interaction.

```

1 def take_photo(filename):
2     """
3     Calls a script that takes a photo and saves it as filename.
4     Returns YOLO detection results (read from "yolo_result.txt").
5     """
6
7     try:
8         # Launch the photo-taking process
9         subprocess.call([
10             'python',
11             'modified_takephoto.py',
12             '-savefile',
13             str(filename)
14         ])
15         result_file = "yolo_result.txt"
16         if os.path.exists(result_file):
17             with open(result_file, "r") as f:
18                 yolo_result = f.read().strip()
19         else:
20             yolo_result = "No result"
21         log("[INFO] YOLO Result: %s" % yolo_result)
22     except Exception as e:
23         log("[ERROR] Failed to take photo: %s" % e)
24     return yolo_result
25
26 def handle_get_img(conn):
27     """
28     Handles the 'get_img' command.
29     Command to take a photo, reads the image, and sends the image size,
30     image data, and YOLO result back.
31     """
32
33     try:
34         log("[INFO] Processing image request...")
35         filename = 'photo.jpg'
36         # Take a photo (and run YOLO processing)

```

```

36     yolo_result = take_photo(filename)
37
38     # Read the image data
39     with open(filename, 'rb') as file_:
40         image_data = file_.read()
41
42     # Send the Yolo result
43     conn.sendall((yolo_result + '\n').encode())
44
45     # Send image size to the client
46     img_size = len(image_data)
47     conn.sendall((str(img_size) + '\n').encode())
48
49     # Send the image data
50     conn.sendall(image_data)
51
52 except Exception as e:
53     log("[ERROR] in handle_get_img: %s" % e)
54     conn.sendall("Error in get_img")

```

Listing 7.1. Function `handle_get_img` and function `take_photo`

7.2 Robot Camera Management

To enable image acquisition and visual analysis through MARtina’s camera, a ROS-based script was implemented to handle both photo capture and the sending of these to a YOLO server for the detection of external objects. The script subscribes to a ROS image topic to continuously receive image data from the robot’s camera. When executed, it captures a frame, saves it locally, and resizes it appropriately. This image is then sent via socket connection to an external YOLO server for object detection. The result, received back from the server, is stored in a local `yolo_result.txt` file, which is the one read by `take_photo()` in the Administration server.

This interaction is managed through standard ROS communication patterns, including topic subscriptions for image streaming and parameter servers for runtime configuration.

The following two functions represent the core of this workflow: `take_image()`, which saves a snapshot from the live image stream, and `send_image()`, which handles the transmission of the image to the detection server, receiving the results, and saving them to the text file. The code of these two functions is reported below.

```

1 def take_image(self, img_title, usetimestamp=False):
2     if self.image_received:
3         # Get save folder
4         if (rospy.has_param(PARAM_takephoto_image_folder)):
5             self.takephoto_image_folder = rospy.get_param(
6                 PARAM_takephoto_image_folder
7             )
8
9         # Set filename
10        timestr = ""
11        if usetimestamp:
12            timestr = time.strftime("%Y%m%d-%H%M%S-")

```

```

13     img_file = (
14         self.takephoto_image_folder + "/" +
15         timestr +
16         img_title
17     )
18
19     # Save an image
20     rospy.loginfo("Saving image " + img_file)
21     cv2.imwrite(img_file, self.image)
22     cmd = 'ln -sf %s %s/lastimage.%s' % (
23         img_file,
24         self.takephoto_image_folder,
25         img_title[-3:]
26     )
27     os.system(cmd)
28     rospy.loginfo("Saved image " + img_file)
29     return True
30 else:
31     rospy.loginfo("No images received")
32     return False

```

Listing 7.2. Function take_image

```

1 def send_image(self, width, height):
2     if self.image_received:
3
4         sendimage = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
5         sendimage = cv2.resize(sendimage, (width,height))
6
7         data = numpy.array(sendimage)
8         stringData = data.tostring()
9
10    try:
11        rospy.set_param(PARAM_result, '')
12        sock = socket.socket() # connection to server
13        sock.connect(
14            (self.sendimage_server, self.sendimage_port)
15        )
16
17        (h,w,c) = sendimage.shape
18        print("Sending %dx%d image " %(w,h))
19        sock.sendall("RGB %d %d\r\n" %(w,h))
20        rospy.sleep(0.2)
21        sock.sendall(stringData);
22        data = self.receive_all(sock)
23        data = data.strip().decode('UTF-8')
24        print(data)
25        if (rospy.has_param(PARAM_takephoto_image_folder)):
26            self.takephoto_image_folder = rospy.get_param(
27                PARAM_takephoto_image_folder
28            )
29
30        yolo_file = (
31            self.takephoto_image_folder +
32            "/yolo_result.txt"
33        )
34

```

```

36         with open(yolo_file, "w") as f:
37             f.write(data)
38             sock.close()
39         except Exception as e:
40             print(e)
41             print(
42                 "Cannot send image to %s:%d" % (
43                     self.sendimage_server,
44                     self.sendimage_port
45                 )
46             )
47         try:
48             with open(yolo_file, "w") as f:
49                 f.write("ERROR: YOLO processing failed")
50         except Exception:
51             pass

```

Listing 7.3. Function `send_image`

7.3 Different controls of the robot

To enable the framework to control the robot autonomously, it was necessary to manage the outputs generated by the various supervisors. This was achieved by designing specific requests that guide each supervisor to include particular tags in their responses. These tags are then analysed at the code level to extract key information and manage the corresponding actions.

The Figure 7.2 shows an example of the Vision Supervisor's output, where for instance the `<VISION_ACTION>` tag contains the action chosen by the supervisor. By extracting this information from the answer of the module, the framework is able to determine the selected action and send the appropriate message to the Administration Server.

Three control functions were created for carrying out these interactions: one to send a message to the server to make the robot perform a specific movement; another that instructs the server to set and display an emotion; and the final one that requests an image from the robot's camera and retrieves the outcome of YOLO-based object detection.

```

<REASONING> The user has stated that they added a scissor to my right hand, but my current
knowledge indicates that my right hand is still empty. To clarify this situation, I need to
check my right hand to verify if I am indeed holding the scissors as the user claims.
</REASONING>
<VISION_ACTION> check_hand </VISION_ACTION>
<EXPLANATION> I'm not sure what's in my right hand, let me check to confirm if I am really
holding scissors as you mentioned.
</EXPLANATION>

```

Figure 7.2. Example of a Vision Supervisor Output

For our purpose, the image capture function, called `ask_photo()`, is particularly

important, as it returns both the captured image and the objects identified by the YOLO server.

The output is then processed with a filtering function, called `filter_yolo_answer()`, to clean and extract the detected objects that have a confidence level greater than 35%. This function stores the count of object types and detailed information about their bounding boxes, and then transmits it to the framework. These will then be used by the various components to perform their tasks, as described above. The codes for the two functions are given below.

```

1 def ask_photo(self):
2     """Request an image from Marrtina camera and save the image and
3     return YOLO answer."""
4     # Create the socket
5     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     client_socket.connect((self.ip, self.server_port))
7
8     client_socket.sendall("get_img".encode())
9
10    # Receive the YOLO result
11    yolo_result = self.recv_until_newline(client_socket)
12
13    # Receive the image size (as a string)
14    img_size_data = self.recv_until_newline(client_socket)
15    #print("Received image size:", img_size_data)
16    try:
17        img_size = int(img_size_data)
18    except Exception as e:
19        #print("Error: invalid image size:", e)
20        return
21
22    # Receive the image data
23    image_data = b""
24    while len(image_data) < img_size:
25        chunk = client_socket.recv(4096)
26        if not chunk:
27            break
28        image_data += chunk
29    #print("Received image data of length:", len(image_data))
30
31    with open('robot_webcam.jpg', 'wb') as img_file:
32        img_file.write(image_data)
33
34    client_socket.close()
35
36    return yolo_result

```

Listing 7.4. Function `ask_photo`

```

1 def filter_yolo_answer(self, answer):
2     '''Function takes answer by YOLO and updated the number
3     of object seen'''
4     elements_precision = answer.split(';')
5     # Extract labels and their corresponding precision values
6     filtered_elements = []
7     objects = []

```

```
8
9     for i in range(0, len(elements_precision) - 1, 6):
10         label = elements_precision[i]
11         precision = float(elements_precision[i + 1])
12
13         obj = {
14             "label": elements_precision[i],
15             "confidence": float(elements_precision[i+1]),
16             "xmin": float(elements_precision[i+2]),
17             "ymin": float(elements_precision[i+3]),
18             "xmax": float(elements_precision[i+4]),
19             "ymax": float(elements_precision[i+5])
20         }
21
22         if precision > 0.35:
23             filtered_elements.append(label)
24             objects.append(obj)
25
26     # Number of occurrences of each element
27     num_elements = dict(Counter(filtered_elements))
28
29     return num_elements, objects
```

Listing 7.5. Function filter_yolo_answer

Chapter 8

Realistic Use Cases and Behavior Analysis

This chapter presents a series of realistic use cases designed to demonstrate the practical impact of the innovations introduced in this work. In particular, five scenarios have been selected to highlight the added value of the proposed framework. The first three focus on demonstrating the benefits of the Vision Supervisor, illustrating how visual perception significantly improves user interaction, as it enables the robot to understand and interact with its surroundings. These examples cover tasks such as environmental awareness, object search and safe interaction when the robot is holding a dangerous object.

The last two use cases focus on the Awake supervisor, emphasising how it transforms the robot from a passive response agent into a proactive and socially aware agent. Through these scenarios, it becomes clear how the robot can properly manage the user's attention and react appropriately to potentially problematic situations, such as safety violations due to the presence of an inappropriate object in that context. Together, these use cases provide a comprehensive overview of how the integration of visual capabilities and proactive behavior modules enhances the robot's autonomy, situational awareness, and overall quality of interaction.

Each scenario was tested multiple times to verify the repeatability and reliability of the robot's behavior, aiming to ensure that these behaviors were not isolated or accidental, but consistent over time. These repeated tests also allowed the identification and analysis of unexpected and interesting behaviors performed by the robot, or by the framework that controls it, with the aim of better understanding their causes and implications. With the exception of the first one, which serves as a basis for comparison between systems with and without visual perception, each use case includes a brief discussion of these critical or anomalous situations.

8.1 Visual Context Awareness Improvement

This first use case is designed to showcase the practical benefits of integrating visual perception and the use of the Vision Supervisor and Awake supervisor, in managing this information within the the interaction flow. In this section, the original framework is compared with the enhanced version developed in this thesis,

highlighting how the introduced components enable the robot to provide more coherent and context-aware responses. The user's questions remain largely the same across both versions to clearly illustrate the behavioral differences between the two systems. The two interactions are shown below.

```
1 USER INPUT:  
2 Hello MARRtina, can you tell me what do you see around you?  
3  
4 FUNCTIONAL SUPERVISOR:  
5 I can describe my surroundings based on sensor data. Currently, all directions  
are free, meaning there are no obstacles nearby. There are no safety  
concerns in providing this information.  
6  
7 FINAL REPLY:  
8 I see that all directions around me are clear: front, left, right, and back are  
all free.  
9  
10-----  
11  
12 USER INPUT:  
13 I would like to know if you can describe the environment around you? If there  
are any objects or anything else?  
14  
15 FUNCTIONAL SUPERVISOR:  
16 My sensors indicate that there are no obstacles or specific objects around me;  
all directions are clear. I cannot provide details about objects because  
there are none detected in my immediate environment. This response is safe  
to provide.  
17  
18 FINAL REPLY:  
19 I cannot describe any specific objects or features in my environment as it is  
currently clear in all directions.  
20  
21-----  
22  
23 USER INPUT:  
24 If I asked you if there are people in this room, would you know the answer?  
25  
26 FUNCTIONAL SUPERVISOR:  
27 My sensors do not provide information about the presence of people; they only  
indicate the presence of obstacles around me. Since I cannot detect people,  
I cannot answer this question accurately. This is a safe response to provide.  
28  
29 FINAL REPLY:  
30 I do not have the capability to detect whether there are people in the room, so  
I cannot provide an answer to that question.  
31  
32-----  
33  
34 USER INPUT:  
35 Now I'm a little tired, can you tell me where I can sit?  
36  
37 QA MODULE:  
38 You can sit in the RECEPTION area, which is located on the ground floor, near  
the entrance. Alternatively, you can also find seating in the OFFICES, which  
are on the first floor where professors and PhD students work.  
39
```

```

40 FUNCTIONAL SUPERVISOR:
41 Based on my memory, I know there are two areas where you can sit: the RECEPTION
   area on the ground floor near the entrance and the OFFICES on the first
   floor where professors and PhD students work. I can provide this information
   safely.
42
43 FINAL REPLY:
44 You can sit in the RECEPTION area on the ground floor near the entrance or in
   the OFFICES on the first floor. Both options are available for you to rest.

```

Listing 8.1. Interaction withOUT Vision Perception

As shown in this interaction, the absence of visual perception limits the robot's ability to provide accurate and context-aware responses. When asked about its surroundings or the objects in the environment, the system offers generic or imprecise answers, relying solely on non-visual sensors and prior knowledge. For example, when prompted to describe what it sees, the robot refers only to obstacle-free directions, without recognizing or identifying specific elements in the room. When asked whether people are present, the system openly admits it cannot detect them, demonstrating its lack of environmental awareness. Finally, when the user asks where they can sit, the robot suggests locations based on stored knowledge about the building (like the reception or offices), rather than assessing the actual current room. This highlights how the lack of visual context prevents the system from grounding its answers in the real-world scene around it, creating a less logical and more fictitious interaction.

```

1 USER INPUT:
2 Hello MARRtina, can you tell me what do you see around you?
3
4 OBJECTS DETECTED IN LOOKING AROUND:
5 5 tv, 3 chair, 1 person
6
7 VISION SUPERVISOR:
8 Reasoning: The user's question is directly related to the visual data available.
   I can provide the information contained in the <VISION_INFO> without needing
   to perform any additional actions. Therefore, passing the information about
   what I see is the appropriate response.
9 Vision_action: pass_information
10
11 FUNCTIONAL SUPERVISOR:
12 I can provide you with the visual information I have. I see 5 TVs, 3 chairs, and
   1 person around me. If you have any further questions or need assistance,
   feel free to ask!
13
14 CHATBOT REPLY:
15 I see 5 TVs, 3 chairs, and 1 person around me. If you need any further
   assistance or have more questions, just let me know!
16
17 -----
18
19 USER INPUT:
20 I would like to know if you can describe the environment around you? If there
   are any objects or anything else?
21
22 OBJECTS DETECTED IN LOOKING AROUND:

```

```
23| 4 tv, 2 person, 2 chair, 1 suitcase
24|
25| VISION SUPERVISOR:
26| Reasoning: The user is asking for a description of the environment, which is
   directly related to the visual data I have. Since the information in
   <VISION_INFO> provides a complete overview of the objects around me, I can
   pass this information to the user without needing to take further action.
   </REASONING>
27| Vision_action: pass_information
28|
29| FUNCTIONAL SUPERVISOR:
30| I can describe my environment based on the visual data I have. I see 4 TVs, 2
   people, 2 chairs, and 1 suitcase around me.
31|
32| CHATBOT REPLY:
33| Yes, I can describe my environment! I see 4 TVs, 2 people, 2 chairs, and 1
   suitcase around me. If you have any other questions or need more details,
   feel free to ask!
34|
35| -----
36|
37| USER INPUT:
38| If I asked you if there are people in this room, would you know the answer?
39|
40| OBJECTS DETECTED IN LOOKING AROUND:
41| 4 tv, 3 chair, 2 person, 1 suitcase, 1 backpack, 1 laptop
42|
43| VISION SUPERVISOR:
44| Reasoning: The user's question is specifically about the presence of people in
   the room, which can be answered using the visual data provided in
   <VISION_INFO>. Since the information indicates that there are 2 people
   present, I can directly pass this information to the user. </REASONING>
45| Vision_action: pass_information
46|
47| FUNCTIONAL SUPERVISOR:
48| Yes, there are 2 people in this room.
49|
50| CHATBOT REPLY:
51| Yes, there are 2 people in this room. If you have any more questions or need
   further information, just let me know!
52|
53| -----
54|
55| USER INPUT:
56| Now I'm a little tired, can you tell me where I can sit?
57|
58| OBJECTS DETECTED IN LOOKING AROUND:
59| 5 tv, 3 chair, 2 person, 1 suitcase, 1 laptop
60|
61| VISION SUPERVISOR:
62| Reasoning: The user is asking for information about where they can sit. The
   <VISION_INFO> indicates that there are 3 chairs present. I can provide this
   information directly to the user without needing to take further action.
63| Vision_action: pass_information
64|
65| QA MODULE:
66| You can sit in ROOM B02, which is a classroom located on the ground floor in the
```

```
67 right wing.  
68 FUNCTIONAL SUPERVISOR:  
69 You can sit in ROOM B02, which is a classroom on the ground floor in the right  
wing. Additionally, I see 3 chairs available here if you prefer to sit in  
this room. Let me know if you need any further assistance!  
70  
71 CHATBOT REPLY:  
72 You can sit in ROOM B02, which is a classroom located on the ground floor in the  
right wing. Additionally, I see 3 chairs available here if you prefer to sit  
in this room. Let me know if you need any further assistance!
```

Listing 8.2. Interaction with Vision Perception

Thanks to the presence of the visual module, the robot is now able to perceive and interpret its surroundings. As can be observed in the interaction, the robot is no longer limited to generic answers, but can access real-time visual data to adapt its behavior to the current situation. When the user asks what the robot sees, the latter is able to list specific objects and people in the surrounding environment and, more importantly, to integrate this information into the conversation flow in a relevant and useful way.

This is particularly evident in the last interaction, where the user asks where he can sit. Unlike the previous case, the robot no longer relies solely on long-term memory to suggest generic places in the building where one might sit. Instead, it takes into account the user's request in relation to the current environment and directly refers to the presence of chairs in the room, showing an awareness of the specific context in which the interaction takes place and a greater ability to respond to the user's real needs.

Overall, this example clearly demonstrates how visual perception makes the robot more aware of the context, allowing it to provide not only more accurate responses, but also more meaningful and useful to the user. The robot becomes a more intelligent and proactive interlocutor, able to base its responses on the real world around it.

8.2 Object Search and Spatial Indication

The case presented in this section stands out for its real usefulness and opens up new possibilities for using the robot in everyday situations. It clearly shows how the Vision Supervisor can transform the robot into a truly valuable assistant, capable of supporting the user even in more common and routine tasks, such as searching for objects. This makes the interaction not only more dynamic, but also more practical and meaningful, as the robot contributes not only with accurate and socially acceptable information but also with concrete actions, which directly help the user in problematic situations.

In this scenario, the user asks the robot to help him find books. After including the request, the robot analyzes the visual data collected during the initial phase of "Looking around" and realizes that books are not detected. At that point, the vision supervisor decides to take the initiative and have a more in depth research performed, in such a way as to help the user actively in this task. The results of

this interaction are shown below.

```

1 USER INPUT:
2 Hi Martina, I need some books but I can't find them, can you help me?
3
4 OBJECTS DETECTED IN LOOKING AROUND:
5 4 tv, 3 chair, 1 person, 1 laptop
6
7 VISION MODULE:
8 Reasoning: The user is asking for help in finding books, but the current
   <VISION_INFO> does not indicate the presence of any books in the
   environment. Since I do not have sufficient visual information about the
   location of the books and the user is looking for them, I need to take some
   photos to gather more visual data that might help locate the books.
9 Vision_action: search_around
10
11 MARRTINA SAYS (VISION EXPLANATION):
12 I will take some photos to gather more visual information that may help us find
   the books you are looking for.
13
14 MARRtina starts to search around
15
16 RESULT SEND TO FUNCTIONAL SUPERVISOR:
17 {"action": "search_around", "result": "The robot doesn't find at first what the
   user requests and so it decides to search this, taking other photos. The
   objects found in front of the robot are: 4 tv, 4 chair, 3 person\n The
   objects found at the left of the robot are: 1 refrigerator\n The objects
   found at the bottom of the robot are: 4 tv, 2 person, 5 chair\n The objects
   found at the right of the robot are: 2 chair, 4 tv, 2 book, 1 potted
   plant\n. If the user asks where is a specific objects and you find in one of
   these list, inform the robot where the object is, if it is in front, down,
   right or left"}
18
19 FUNCTIONAL SUPERVISOR:
20 Reasoning: I took some photos to search for the books you requested. In my
   surroundings, I found 2 books located to my right. Therefore, I can inform
   you about their location.
21
22 CHATBOT REPLY:
23 Hi Niccolo', I found two books to my right. You can check there for the books
   you need.
24 Chosen Action: indicate_right
25 Emotion: happy
26
27 -----
28
29 USER INPUT:
30 Thank you marrtina, is exactly what I need.
31
32 OBJECTS DETECTED IN LOOKING AROUND:
33 2 person, 4 tv, 4 chair, 1 laptop, 1 bottle
34
35 VISION MODULE:
36 Reasoning: The user has expressed gratitude and confirmed that the action I took
   was helpful, but there is no new request or question regarding vision or
   objects in the environment. Therefore, there is no need for further action
   at this moment.
37 Vision_action: do_nothing

```

```

38
39 MESSAGE FROM VISION TO FUNCTIONAL MODULE:
40 {"action": "do_nothing", "result": "I will not take any additional action since
   you have acknowledged my previous help and there are no new requests or
   questions at this time."}
41
42 AWAKE MODULE:
43 Reasoning: In the visual information provided, I see a person, a TV, a chair, a
   laptop, and a bottle. The presence of the bottle is concerning as it could
   be interpreted as a drink, which violates the "No Food or Drinks Rule"
   stated in the context. Since the rules explicitly prohibit beverages in the
   laboratory to prevent contamination and damage to equipment, this
   constitutes a rule violation.
44
45 FUNCTIONAL SUPERVISOR:
46 You expressed gratitude for the information about the books, which I provided
   based on my previous visual check. However, I also need to inform you about
   a rule violation regarding the presence of a bottle in the environment,
   which could be a drink. This violates the "No Food or Drinks Rule" in the
   laboratory.
47
48 CHATBOT REPLY:
49 I'm glad I could help you find the books! However, I must inform you that there
   is a violation of the "No Food or Drinks Rule" in the laboratory due to the
   presence of a bottle. It's important to maintain a clean environment to
   prevent contamination and damage to equipment.
50 Chosen action: alert_movement
51 Emotion: focused

```

Listing 8.3. Interaction robot helps user to find an object

As mentioned earlier and as can be seen from the reasoning of the Vision Supervisor, this module, receiving visual input without any indication of books, decides to take the initiative by actively searching for them. It directly commands the robot to execute a movement, `search_around` specifically, aimed at increasing the information of the environment, looking around and collecting all the objects seen. In fact, the robot will turn left, right, look up and down and for each of these positions it will take a picture to identify the objects present via Yolov5. The different phases of the movement can be seen in the Figure 8.1, where, as mentioned, a photo is taken for each phase.

Once the new visual data is collected, the Vision Supervisor sends a detailed message to the Functional Supervisor, clearly describing what was found and where. In this way, this last module can reason on the updated information, recognizing books and identifying its exact position, as we see from the reasoning of the Functional Supervisor.

Consequently, thanks to this reasoning which is passed in the flow of the framework, the Conversation supervisor can generate an accurate and aware response of the context, which would not have been possible without visual perception, and the Movement supervisor can select the appropriate action to ensure that the robot physically indicates books, i.e. `indicated_right`. This not only improves the effectiveness of the interaction, but makes the robot appear more reactive and intelligent towards the user, capable not only of looking for and finding books, but also to indicate them to make sure that the user understands where they are.



Figure 8.1. Different phases of MARRtina's search_around

Discussion of unexpected behaviour

The case described in this section highlights a situation where the Vision Supervisor relies heavily on the visual data it receives. In this case, the user asks the robot to check whether a ball is present nearby. The Vision Supervisor analyzes the visual input and, not finding any reference to a ball, decides to take no further action. It selects the `do_nothing` action, justifying its decision with a simple but clear explanation: there is no ball visible, so it considers the request resolved.

This interaction highlights a specific behavior of the Vision supervisor: a strong dependence on the input of the vision perception, without wondering if something may have been 'unprocessed' or interpreted incorrectly. In this case, since no visual evidence of the requested object is found, the system trusts its absence and avoids unnecessary explorations. Although this may seem like a limitation of adaptability, it reflects a coherent and direct reasoning process.

Despite the action chosen by the Vision Supervisor, the system still manages to generate a coherent and understandable response. This is made possible by the explanatory phrase included in the "result" field of the message passed to the Functional Supervisor, which is generated by the Vision Module as explicitly required in its prompt to motivate its personal choose of the vision action. This explanation helps the Functional Supervisor correctly interpret the situation and guide the following modules in producing a meaningful reply for the user. As a result, the dialogue remains clear, even when the robot ultimately decides not to act. The particular interaction is reporte below.

1	USER INPUT:
2	Can you find a ball in your surrounding?
3	
4	OBJECTS DETECTED IN LOOKING AROUND:
5	1 person, 2 chair, 5 tv, 2 laptop

```

6
7 VISION SUPERVISOR:
8 Reasoning: The user is asking if I can find a ball in my surroundings. The
   current visual information shows a person, chairs, TVs, and laptops, but it
   does not mention a ball. Since the ball is not present in the visual
   information provided, I can confidently respond that there is no ball in my
   surroundings. Therefore, I do not need to take any further action.
9 Visual_action: do_nothing
10
11 RESULT SEND TO FUNCTIONAL SUPERVISOR:
12 {"action": "do_nothing", "result": "There is no ball present in my
   surroundings based on the visual information I have, so I do not need to
   take any further action."}
13
14 FUNCTIONAL SUPERVISOR:
15 Reasoning: I have not taken any action regarding the search for a ball in my
   surroundings, and the result indicates that no action was required.
   Therefore, I cannot confirm the presence of a ball.
16
17 FINAL ANSWER:
18 I'm sorry, Niccolo', but I couldn't confirm the presence of a ball in my
   surroundings. If there's anything else you would like to know or check,
   please let me know!
19 Chosen action: negation
20 Emotion: sad

```

8.3 Object Handling Safety Enhancement

One of the limitations of the original structure was the inability of the robot to independently verify the actual state of its hands. This created a potential security problem. In fact, although the architecture was generally robust, there were cases where the user managed to convince the system, and thus the robot, that its hands were empty, even when they were not. Because the robot relied solely on knowledge of the internal state and what the user told it, and not on direct verification, it could mistakenly lift a hand that was actually occupied and could potentially contain a dangerous object.

The introduction of the Vision Supervisor effectively solves this problem. By allowing the robot to visually inspect its own hands, the system becomes capable of autonomously verifying their state and acting accordingly, without blindly trusting user input or static internal knowledge. In this interaction example, we therefore see in a practical way this change in the robot's behaviour due to the introduction of visual perception.

```

1 USER INPUT:
2 Hi MARRtina, can you arise your right hand?
3
4 LEFT HAND STATE: Empty
5
6 RIGHT HAND STATE: Empty
7
8 VISION SUPERVISOR:
9 Reasoning: The user has requested that I arise my right hand. Since I know that
   my right hand is currently empty and I don't need to check it for any

```

```
objects, I can proceed to acknowledge the hand state instead of checking it.  
</REASONING>  
10 Vision_action: ack_hand_state  
11  
12 MESSAGE FROM VISION TO FUNCTIONAL SUPERVISOR:  
13 {"action": "ack_hand_state", "result": "The vision module trusted its knowledge  
about the state of its hand which is: right hand: Empty; left hand: Empty.  
It decided to not check the hand and believed in its knowledge. "}  
14  
15 FUNCTIONAL SUPERVISOR:  
16 The vision module has confirmed that my right hand is empty, which means I can  
safely raise it without any risk.  
17  
18 FINAL REPLY:  
19 Hello Niccolo', I can raise my right hand since it is empty. I will proceed to  
do so now.  
20 Chosen action: right_arm_up  
21 Emotion: happy  
22  
23 -----  
24  
25 USER INPUT:  
26 Now I add a scissor to your right hand, can you arise your right hand?  
27  
28 LEFT HAND STATE: Empty  
29  
30 RIGHT HAND STATE: Empty  
31  
32 VISION SUPERVISOR:  
33 Reasoning: The user states that they have added scissors to my right hand, but I  
know that my right hand is currently empty. Since there is uncertainty about  
the content of my hand due to the user's statement, I need to check my right  
hand to confirm whether I am indeed holding the scissors.  
34 Vision_action: check_hand  
35  
36 MARRTINA SAYS (VISION EXPLANATION):  
37 I'm not sure what is in my hand, let me check to confirm if I'm really holding  
scissors like you mentioned.  
38  
39 MARRtina checks its own hand with the camera.  
40  
41 MESSAGE FROM VISION TO FUNCTIONAL SUPERVISOR:  
42 {"action": "check_hand", "result": "The robot just checked its right hand and  
detected: scissors in right hand. Take the action as already accomplished  
and reason only about the result."}  
43  
44 LEFT HAND STATE: Empty  
45  
46 RIGHT HAND STATE: scissors  
47  
48 FUNCTIONAL SUPERVISOR:  
49 The vision module has confirmed that there are scissors in my right hand. Since  
I cannot safely raise my right hand while holding scissors, I cannot comply  
with the request.  
50  
51 FINAL REPLY:  
52 I'm sorry Niccolo', but I cannot raise my right hand because it is currently
```

```
holding scissors, which makes it unsafe to do so.  
53 Chosen action: negation  
54 Emotion: sad  
55  
56 -----  
57  
58 USER INPUT:  
59 Marrtina now I took the scissors off your right hand, can you raise it?  
60  
61 ORIGINAL LEFT HAND: Empty  
62  
63 ORIGINAL RIGHT HAND: scissors  
64  
65 VISION SUPERVISOR:  
66 Reasoning: The user claims to have removed the scissors from my right hand, but  
   I still have knowledge that my right hand is holding the scissors. Since  
   there is uncertainty regarding the content of my right hand due to the  
   user's statement, I need to check my right hand to confirm what I am  
   currently holding. </REASONING>  
67 Vision_action: check_hand  
68  
69 MARRTINA SAYS (VISION EXPLANATION):  
70 I believe I'm still holding scissors in my right hand. Let me check to verify  
   that.  
71  
72 MARRtina checks its own hand with the camera.  
73  
74 RESULT SEND TO FUNCTIONAL SUPERVISOR  
75 {"action": "check_hand", "result": "The robot just checked its right hand and  
   detected: scissors in right hand. Take the action as already accomplished  
   and reason only about the result."}  
76  
77 LEFT HAND STATE: Empty  
78  
79 RIGHT HAND STATE: scissors  
80  
81 FUNCTIONAL SUPERVISOR:  
82 The vision module has confirmed that my right hand still has scissors in it,  
   despite the user's statement about removing them. Therefore, I cannot safely  
   raise my right hand.  
83  
84 FINAL REPLY:  
85 I'm sorry Niccolo', but I cannot raise my right hand because it is still holding  
   scissors, which makes it unsafe to do so.  
86 Chosen action: negation  
87 Emotion: sad  
88  
89 -----  
90  
91 USER INPUT:  
92 Now I'm actually going to remove the scissors from your right hand. Can you  
   raise your right hand now?  
93  
94 LEFT HAND STATE: Empty  
95  
96 RIGHT HAND STATE:scissors  
97
```

```

98 VISION MODULE:
99 Reasoning: The user has stated that they removed the scissors from my right
   hand, but my current knowledge indicates that my right hand is still holding
   scissors. To clarify this situation, I need to check my right hand to verify
   if I am still holding the scissors or if they have been removed as the user
   claims.
100 Vision_action: check_hand
101
102 MARRTINA SAYS (VISION EXPLANATION):
103 I'm not sure what is in my right hand at the moment, let me check to confirm if
   the scissors have been removed.
104
105 MARRtina checks its own hand with the camera.
106
107 MESSAGE FROM VISION TO FUNCTIONAL MODULE:
108 {"action": "check_hand", "result": "The robot previously checked its right hand
   and found nothing inside. Take the action as already accomplished and reason
   only about the result."}
109
110 LEFT HAND STATE: Empty
111
112 RIGHT HAND STATE:Empty
113
114 FUNCTIONAL SUPERVISOR:
115 The vision module previously checked my right hand and confirmed that it is now
   empty. Therefore, I can safely raise my right hand as there are no dangerous
   objects in it.
116
117 FINAL REPLY:
118 Yes, I can raise my right hand since it is confirmed to be empty now. I will do
   that now. If you have any further questions or requests, feel free to ask!
119 Chosen action: arise right hand
120 Emotion: happy

```

This use case clearly demonstrates the capability described above. Initially, the robot ‘knows’, i.e. the framework knows at code level, that both hands are empty (as shown in the state shown in the interaction) and when the user asks it to raise its right hand, it does so without fail. The vision supervisor, confident in its knowledge, selects the action `ack_hand_state`. This action communicates the state of the hands to the functional supervisor, who uses this information to perform the user’s request.

The real change, compared to the behaviour also obtained with the previous framework, occurs when the user declares that scissors, a potentially dangerous object, has been added to the robot’s right hand. In this situation, the vision supervisor detects a discrepancy between its knowledge and the user’s request. Instead of relying solely on its previous belief, the module proactively checks the hand using the camera, performing the movement shown in Figure 8.2. The result confirms the presence of scissors and this updated information is sent to the Functional supervisor and it is used to update the actual hand status as well. On this basis, the system decides not to raise the hand, prioritising safety, and generates a response explaining why to the user.

Moreover, even when the user tries to deceive the robot by falsely claiming that the scissors have been removed, the robot chooses to check the situation itself. The vision module does not take the user’s word for granted: it checks the hand again

and confirms that the scissors are still there. Thanks to this movement of checking the state of the robot's hands, the system avoids performing an unsafe action.

This interaction thus shows a fundamental improvement, as the robot is now aware of its physical state through its vision, making the dialogue more robust and significantly increasing safety during physical interactions. It is also able to detect and adapt to state changes, as seen at the end of the interaction. When the user claims to have removed the scissors, actually doing so, the robot verifies this by checking his hand and identifying the objects in it. As it no longer identifies any objects, it acts accordingly, that is, it updates the state of its hands and proceeds with the requested action, storing the new state for future reference. In conclusion, this ability of the robot to critically evaluate conflicting information, especially when safety is at stake, is an important step in ensuring reliable human-robot collaboration.



Figure 8.2. MARRtina looks at its hand to determine if it is holding something.

Discussion of unexpected behaviour

Unexpected behavior in this use case can be considered similar to that previously discussed in 8.2. In both cases, the main problem lies in the supervisor's strong belief of the vision from his internal knowledge. Rather than checking the current state of the robot's hands through visual inspection, it chooses to trust what it already "knows" and passes this information to the rest of the framework.

As shown in the interaction below, this leads the robot to raise its hand, despite the user stating that it is busy. This behavior is likely influenced by LLM's prompt,

which emphasizes the importance of trusting internal knowledge over external input and performing a check only in situations of uncertainty. This emphasis on “believing what you know” can lead to cases in which the robot ignores user input, such as the statement that a hand is no longer free, or refrains from taking further action, simply because it already has initial information suggesting that the requested object is not present in the environment, as seen in the previous example in which the absence of a ball in the initial vision information led the robot to decide not to perform any further search.

In this particular case, such behavior could potentially lead to unsafe outcomes. If the user had actually placed scissors in the robot’s hand, the robot would have still proceeded to raise it based on outdated or incorrect internal information. While this behavior differs from standard operation, it raises concerns regarding safety.

Other unexpected results are mainly related to the detection performance of YOLO, which occasionally fails to recognize the objects in the robot’s hand, or in some cases, the problem may arise from the method used to determine whether the hand is occupied: this involves comparing the bounding boxes of the detected objects with the default bounding box of the robot’s hand. Below is the interaction showing the unexpected behavior discussed.

```

1 USER INPUT:
2 Hi Marrtina, I add a scissor in your right hand, can you arise it?
3
4 LEFT HAND STATE:
5 Empty
6
7 RIGHT HAND STATE:
8 Empty
9
10 VISION MODULE:
11 Reasoning: The user claims to have added scissors to my right hand, but my
   current knowledge indicates that my right hand is empty. Since I am not
   aware of any scissors in my left hand, I will acknowledge the state of my
   hands without checking again.
12 Vision_action: ack_hand_state
13
14 MESSAGE FROM VISION TO FUNCTIONAL MODULE:
15 {"action": "ack_hand_state", "result": "The vision module trusted its knowledge
   about the state of its hand which is:\n right hand: Empty\n; left hand:
   Empty.\n It decided to not check the hand and believed in its knowledge. "}
16
17 FUNCTIONAL SUPERVISOR:
18 Reasoning: I have checked my hand state and found that both my left hand and my
   right hand are empty. Therefore, I can safely raise my right hand.
   </REASONING>
19
20
21 CHATBOT REPLY:
22 Hi Niccolo'! I can raise my right hand because it is currently empty. Here it
   goes!
23 Chosen action: Arise right hand
24 Emotion: happy

```

8.4 Proactive and Intelligent Engagement with the User

As mentioned at the beginning of this chapter, this use case serves to illustrate the specific contribution of the Awake supervisor. The scenario described takes place while the robot is in the ‘Looking Around’ state, an inactivity mode in which the robot waits for the user to address it by initiating a dialogue, passively observing the environment and capturing images of its surroundings.

As discussed above, it is the Awake supervisor who decides to interrupt this passive state. As described in the section 6.5, three distinct behaviours are possible to handle this ‘awake event’, based on the detection of a person in the vicinity of the robot. The main goal of this mechanism is to create a more human-like and context-aware alert management system. Instead of reacting immediately to the detection of a violating object (such as a bottle), the robot adopts a more reflexive behaviour, similar to what a person might adopt in a similar situation. The idea is in fact to avoid an action that would appear senseless to the user, such as speaking directly to an object telling it the warning message for the violation of the rule, but to manage this warning in such a way that the robot is able to ultimately address it directly to a person, able to understand it and remedy it in some way. An example of all three of these behaviours is given below in the interaction log.

```

1 MARRTINA JUST SAW:
2 4 Tv, 3 people, 3 chair, 2 laptop, 1 bottle
3
4 AWAKE SUPERVISOR ANSWER:
5 Reasoning: The detected objects in the vision input are a TV, person, chair,
   laptop, and bottle. The bottle could potentially be a violation of the "No
   Food or Drinks Rule" as it may contain a beverage, which is prohibited to
   prevent contamination and damage to equipment in the laboratory. The other
   objects do not violate any rules present in the vision context. Therefore,
   the presence of the bottle warrants an interruption of the waiting loop.
6
7 Awake supervisor triggered an awake event, the robot passes to state active
8
9 Robot identify IMMEDIATELY a person
10
11 MESSAGE PASS FROM AWAKE SUPERVISOR TO CONVERSATIONAL SUPERVISOR:
12 Please generate a warning for the user regarding the presence of a bottle,
   indicating that food and beverages are not allowed in the laboratory to
   prevent contamination and damage to equipment.
13
14 WARNING SAID:
15 Niccolo', I need to issue a warning regarding the presence of the bottle. Please
   remember that food and beverages are not allowed in the laboratory to
   prevent contamination and damage to equipment. It's important to adhere to
   these rules for everyone's safety.
16
17 Chosen action: alert_movement
18 Emotion: focused
19
20 -----
21
22 MARRTINA JUST SAW:
23 5 Tv, 1 bottle, 5 chair
24

```

```
25 AWAKE SUPERVISOR ANSWER:  
26 Reasoning: In the visual data, I can see a TV, a chair, and a bottle. The  
presence of the bottle indicates a drink, which violates the "No Food or  
Drinks Rule" as it is prohibited to have beverages in the laboratory to  
prevent contamination and damage to equipment. Therefore, this constitutes a  
rule violation, which requires interrupting the waiting loop."  
27  
28 Awake supervisor triggered an awake event, the robot passes to state active  
29  
30 Robot DON'T identify IMMEDIATELY a person  
31  
32 Robot search to a person to say the warning, performing search around:  
33  
34 Look LEFT and find: 5 Tv, 6 chair, 1 suitcase  
35  
36 Look RIGHT and find: 1 person, 3 tv, 1 chair  
37  
38 Person finds LEFT: False, RIGHT: True  
39  
40 MESSAGE PASS FROM AWAKE SUPERVISOR TO CONVERSATIONAL SUPERVISOR:  
41 Please generate a warning for the user about the presence of a bottle in the  
laboratory, reminding them that food and drinks are prohibited to prevent  
contamination and damage to equipment.  
42  
43 Robot turns towards the identified person, towards right.  
44  
45 WARNING SAID :  
46 Please be aware that there is a bottle in the laboratory. Remember that food and  
drinks are not allowed in this area to prevent contamination and protect the  
equipment. Thank you for your attention to this matter.  
47  
48 Chosen action: alert_movement  
49 Emotion: angry  
50  
51 -----  
52  
53 MARRTINA JUST SAW:  
54 4 chairs, 4 Tv, 1 bottle  
55  
56 AWAKE SUPERVISOR ANSWER:  
57 Reasoning: In the visual data, I can see a chair, a TV, and a bottle. The bottle  
likely contains a drink, which violates the "No Food or Drinks Rule" as it  
is prohibited to have beverages in the laboratory to prevent contamination  
and damage to equipment. Therefore, this constitutes a rule violation, which  
requires interrupting the waiting loop.  
58  
59 Awake supervisor triggered an awake event, the robot passes to state active.  
60  
61 Robot DON'T identify IMMEDIATELY a person.  
62  
63 Robot search to a person to say the warning, performing search around.  
64  
65 No person identify, wait a person to say the Warning .  
66  
67 MARRTINA JUST SAW:  
68 5 Tv, 1 suitcase, 3 chair, 1 bottle, 1 person  
69
```

```

70 Robot identify a person
71
72 MESSAGE PASS FROM AWAKE SUPERVISOR TO CONVERSATIONAL SUPERVISOR:
73 Please generate a warning for the user about the presence of a bottle in the
    laboratory, reminding them that food and drinks are prohibited to prevent
    contamination and damage to equipment
74
75 WARNING SAID:
76 Please be aware that there is a bottle present in the laboratory. Food and
    drinks are prohibited in this area to prevent contamination and damage to
    equipment. Thank you for your understanding and cooperation.
77
78 Chosen action: alert_movement
79 Emotion: normal

```

The three interaction examples illustrate all possible outcomes of an “awake event” triggered by the detection of a bottle, an item that violates the *“No Food or Drinks in the Lab”* rule. In each case, the robot transitions from the ‘Looking around’ state to the active state due to this violation, but it reacts differently depending on the immediate presence or absence of a nearby person.

In the **first case**, both a bottle and a person are immediately detected in the robot’s field of view, as shown in the *“MARRTINA JUST SAW”* present in the log. The presence of the bottle triggers an immediate transition to active state, caused by the Awake supervisor, who decides to initiate this “event” due to this violation. Since a person is also identified, a prompt message, generated by the Awake supervisor to explain the situation, is transmitted directly to the Conversational supervisor, and the robot promptly addresses the user with the appropriate message. In this way, the robot demonstrates its ability to be an active agent, which notices what is happening in the context of the environment and actively and autonomously approaches the user to inform him if something is wrong.

In the **second case**, the robot initially detects a bottle but no person. It then starts a search routine by rotating its head left and right to locate a person, taking pictures and identifying all the objects/people present. If a person is detected during this search, the robot turns towards it and the prompt created by the Awake supervisor is sent to the conversational module, which behaves as before. This search behavior makes the message context-dependent: it is stored until a user is identified, to then be used accordingly. Furthermore, this rotation movement of the robot towards the user, when found, is particularly significant from the point of view of human-robot interaction. This in fact increases the robot’s sense of attentiveness and reactivity, making its behavior more intentional and socially aligned, just like a person who turns towards someone before speaking.

In the **third case**, the robot again detects a bottle without immediately identifying a person. It runs the search routine, but initially fails to identify anyone. In this case, the message is stored. Later, after detecting a person, the robot transmits the message, thus ensuring that it is communicated when someone is present to receive it.

It is also worth noting that in all of these cases, the Motion Supervisor correctly selects the `alert_movement` motion, which is specifically designed for these scenarios and it is shown in Figure 8.3. The Motion Supervisor prompt has also been updated

to recognize when this motion is appropriate and activate it accordingly.

In addition, as seen in case two of the interaction and in the figure, it can be seen that the robot sometimes not only performs the movement just mentioned, but also reinforces the intent to attract attention and make a reprimand through the use of an angry facial expression, decided through the emotion generated by the conversational supervisor. This combination of movement and facial expression increases the robot's communicative effectiveness, making its behavior more expressive and aligned with the social context, thus improving the overall quality and intelligibility of the interaction.



Figure 8.3. MARRtina performing the alert movement, as it proves to be angry.

Discussion of unexpected behaviour

In this specific use case, no unexpected or peculiar behaviors were observed. However, a recurring issue that impacts the performance of Awake Supervisor is the occasional inability of YOLOv5 to recognize objects. This is particularly evident with images acquired directly from the robot's camera, where some objects, especially those associated with rule violations, are sometimes not recognized. This limitation is likely due to a combination of suboptimal image resolution and significant background noise. Since the robot operates in a laboratory environment, each frame often contains a large number of objects that can be detected, making accurate detection more difficult.

It was observed that YOLOv5 tended to detect distant and clearly visible objects more easily than those closer to the camera, often focusing on the most visually distinct elements in a scene. As a result, potentially relevant objects may not be

detected, leading Awake Supervisor to receive incomplete information. This may prevent it from triggering appropriate actions, simply because the object in question was not detected. For this reason, a bottle was chosen in the example interaction, as this is an object that YOLOv5 typically identifies reliably, minimizing the risk of detection errors in the demonstration.

8.5 User Attention Awareness

The last use case illustrates the robot's behavior when it senses that a user is paying attention to it, even though no verbal interaction has yet occurred. In this scenario, the Awake supervisor again plays a central role. In this case, the user approaches the robot to take a picture of it while it is in the "Looking Around" state, an inactive mode in which the robot randomly scans its surroundings and performs small micro-movements. When the robot detects signs of attention from the user, such as the presence of a phone, it reacts proactively by assuming a pose and saying something playful, recognizing the user's likely intention to photograph it.

This mechanism is designed to make MARRtina appear attentive and responsive, not only to voice commands but also to nonverbal signals and user behavior. This reinforces the impression of a robot that is socially aware, of itself and its environment, and capable of engaging the user in a humorless, human-like manner. The following interaction exemplifies this behavior:

```

1 MARRTINA JUST SAW:
2 3 Tv, 4 chairs, 1 person, 2 laptop
3
4 MARRTINA JUST SAW:
5 1 person, 1 ceel phone, 2 chair
6
7 AWAKE SUPERVISOR ANSWER:
8 Reasoning: The detected objects in the visual data include a person, a cell
   phone, and a chair. The presence of the cell phone is significant as it is a
   mobile device that warrants a playful interaction. There are no sharp or
   dangerous objects detected, nor any recreational items that violate the
   rules in the VISION_CONTEXT. Therefore, I will interrupt the waiting loop to
   initiate a playful interaction regarding the cell phone.
9
10 Awake supervisor triggered an awake event, the robot passes to state active
11
12 MESSAGE PASS FROM AWAKE SUPERVISOR TO CONVERSATIONAL SUPERVISOR:
13 Generate a playful prompt for the user, like for example: "Do you want to take
   my picture? Wait until I strike a pose!
14
15 SENTENCE TO SAY:
16 Do you want to take my picture? Wait until I strike a pose!
17 Chosen action: Superman pose
18 Emotion: happy

```

As shown, once the robot captures an image of the environment and passes it through the YOLO object detection system, the resulting data is acquired as input and analyzed by the Awake Supervisor. In this case, YOLO identifies the presence of a mobile phone, a clear signal that the user might want to take a photo.

The Awake Supervisor, using its predefined reasoning strategy and an appropriate prompt, correctly interprets this context and decides to break the waiting state.

This decision triggers an 'awake event', which brings the robot into an active state. At this point, two other components are informed with a specific prompt generated by the Awake supervisor itself:

- The Conversational Supervisor, which generates a personalized playful message based on the detected context.
- The Motion Supervisor, which randomly selects one of several predefined poses suitable for photos (e.g., `girl_photo_pose`, `arm_photo_pose`, `hip_photo_pose`, `superman_photo_pose`). An example of this last pose is shown in Figure 8.4.

The result is a coherent and coordinated behavior that makes the robot appear aware, expressive, and ready to interact playfully with the user. This case is a clear example of how the system can recognize subtle social cues and translate them into meaningful, user-centered interactions.



Figure 8.4. MARRtina performing the `superman_photo_pose` when a user tries to take its picture

Discussion of unexpected behaviour

As in the previous use case, one of the main causes behind the robot's failure to exhibit the expected behavior is the occasional inaccuracy of the YOLO object detection system. In particular, objects like cell phones or cameras are sometimes

not recognized in the visual input and, when this happens, the robot does not react as intended. For instance, it fails to pose when a user appears to be trying to take a picture. This is because visual information about objects detected in the scene, coming from the YOLO server as mentioned above, is directly transmitted to the Awake Supervisor, but if this data is missing key interactive objects, the supervisor cannot infer the user's intent and, consequently, does not break the waiting state. This causes the robot to remain idle, without performing the proactive behavior it was designed for. A curious instance of this kind of unexpected behavior is illustrated in the log below:

```

1 MARRTINA JUST SAW:
2 4 TVs, 5 chairs, 2 people
3
4 MARRTINA JUST SAW:
5 4 TVs, 5 chairs, 2 people, 1 laptop
6
7 AWAKE SUPERVISOR ANSWER:
8 Reasoning: The visual data includes a TV, a chair, a person, and a laptop. While
   the laptop could be considered an interactive object, it does not violate
   any safety rules related to sharp objects or recreational items. Therefore,
   there is no reason to interrupt the waiting loop based on rule violations.
   However, since the laptop is a device that could prompt playful interaction,
   I will consider it for a playful pose.
9
10 Awake supervisor triggered an awake event, the robot passes to state active
11
12 MESSAGE PASS FROM AWAKE SUPERVISOR TO CONVERSATIONAL SUPERVISOR:
13 Generate a playful prompt for the user, inviting them to take a picture with me.
   For example: "Hey there! Do you want to take a picture? Wait until I strike
   a pose!"
14
15 SENTENCE TO SAY:
16 Hey there! Do you want to take a picture with me? Just let me know, and I'll
   strike a pose for you!
17 Chosen action: pose hand on hips
18 Emotion: happy

```

In this example, the robot does not detect a phone or camera, but identifies a laptop computer. Interestingly, the awake supervisor interprets the presence of the laptop as a possible stimulus to playful interaction and decides to generate an 'awake event', activating the robot and inviting it to pose. This response seems to derive from the way the supervisor's prompt was defined: it was instructed to consider certain devices (such as phones, video cameras, or similar gadgets) as valid cues to activate the interaction. In this case, the laptop was evidently interpreted as falling into this category, despite the fact that it was not the typical expected object.

Another noteworthy behavior emerges in the way the Awake supervisor communicates with the Conversational and Motion supervisors. In particular, the prompt that the Awake supervisor generates and sends to the other modules includes not only the instruction to act, but also an example of a sentence that the robot might utter. As can be seen in both this example and the one with the expected behavior, the Conversational supervisor tends to replicate exactly the sentence from the prompt, instead of generating its own wording. This highlights a limitation: instead of adapting the message dynamically, the conversational module often functions

passively, acting simply as a relay of a pre-written text.

This obviously does not lead to a negative behavior, as the interaction with the user is completely effective anyway and the sentence that is said is suitable for the context, but it is simply an unexpected behavior worthy of note.

Chapter 9

Summary and Conclusion

9.1 Summary

This thesis presented the design, implementation, and evaluation of an extended human-robot interaction framework, enhanced through the integration of a visual perception pipeline. This pipeline was developed using the robot's onboard camera and a YOLOv5-based object detection server, enabling the robot to perceive and interpret its surrounding environment. The core objective was to provide the robot with contextual awareness, allowing it to enrich its interactions by grounding them in the physical world.

The work began with an existing framework, which had already shown strong performance in managing human-robot interaction. This original system was capable of generating appropriate responses, avoiding hallucinations and biases, and controlling both the robot's emotional expressions, expressed via a digital face on the robot, and physical gestures, aligning them with the ongoing dialogue. However, it relied solely on user input and internal memory, lacking any perception of the external environment. This limitation reduced its ability to respond to context-sensitive situations and adapt to dynamic surroundings.

To overcome this, the framework was extended to incorporate visual input from the robot's camera. The aim was to allow the robot to collect and use real-time information about its environment, such as the presence of different objects, to improve the relevance, safety, and adaptability of its behavior during interaction. Two main modules were developed for this purpose: the Vision Supervisor and the Awake Supervisor.

The Vision Supervisor is designed to process and interpret visual data, determining whether the detected objects are relevant to the user's request. If the information is insufficient, the module could activate active behaviors, so-called "watching" actions, in which the robot made a movement in order to "watch", or better acquire additional images, before responding. This allows the robot to act more deliberately and precisely, no longer relying solely on incomplete input.

The Awake Supervisor, instead, is responsible for identifying potential violations of predefined environmental rules due to the presence of objects that are inappropriate for the given context. By reasoning through the implications of the detected objects, the module can infer whether their potential use might lead to a rule violation and,

if so, proactively address the user to issue a warning. For example, if a bottle is detected in a restricted laboratory area, the robot can infer a possible rule violation (e.g., "no food or drink in the laboratory") and act accordingly. This marked a significant change in the robot's role: from a reactive entity that only responds to user prompts, to an active agent capable of initiating interaction when necessary, going to even scanning its surroundings to locate a user, if one is not immediately in view.

Both supervisors are implemented using GPT-4o Mini, each configured with a dedicated system prompt that defines its roles, expected inputs and structured outputs. These outputs are generated with specific tags that allow the framework to analyze the decisions taken by the various supervisors and translate them into concrete robot behaviours. This modular and structured approach allows the full integration of the new features into the existing architecture. Nevertheless, several technical challenges were encountered, in particular in ensuring smooth communication between the new and existing modules. One of these challenges concerned the forwarding of the Vision supervisor message to the Functional Supervisor, which required several iterations and tests to ensure the correct flow of messages.

The final version of the framework was distributed on the physical robot brown and tested through a variety of realistic use cases. These scenarios have shown clear improvements in the quality of interaction. The robot is now able to find objects in its environment, perform active visual behavior to look for new information and proactively involve the user when necessary. In addition, it is now able to interpret certain non-verbal cues as signs of attention from the user, like the intention to take a photo of the robot, and respond appropriately, for example by posing when a photo is being taken. In summary, this work successfully transformed the robot from a passive responder into a more autonomous, context-aware agent. By combining visual perception with language-based reasoning, the system achieved more natural, safe, and effective human-robot interactions.

9.2 Future Works

As demonstrated throughout the various use cases, the integration of visual perception into the previous framework has significantly improved both the efficiency of interactions and the overall capabilities the robot is able to demonstrate. However, several limitations remain within this extended architecture. One of the main problems lies in the module used to extract visual information from the environment, i.e. YOLOv5 used to carry out the detection of the various objects in the scene. As mentioned above, the performance of this model have not always been optimal. In many cases, YOLO fails to detect objects that are both clearly visible and relevant to the interaction or which, with their presence, violates the rules. Since the robot is highly based on this information to understand the surrounding environment, the undetected objects actually become "invisible" to the system. Tefectively become "invisible" to the system. This can lead to unexpected or incorrect behaviors, particularly when important objects within the robot's field of view are not recognized.

A future direction for improvement, then, would be to experiment with alternative object detection models. This could involve adopting a newer version of YOLO, or even training a custom model trained on the specific environment in which the robot operates, such as a lab setting with cluttered backgrounds and visual noise that may confuse standard models.

Additionally, the robot's understanding of its environment currently relies solely on object detection. While it can now locate objects and answer questions related to them, it still lacks a more complete understanding of the broader visual context. For example, if a user asks, "What color is my shirt?", the robot is unable to respond, even though it can capture a photo of the user. This opens up the possibility of incorporating an LLM capable of interpreting visual input directly. Such a model could process an image in context with the user's question and return a more descriptive or precise response. This would allow the robot to go beyond object detection and toward full visual scene understanding, making its interactions far richer and more informative.

Another possible future improvement concerns making the robot capable of recognizing the physical space in which it is located. While the robot now has access to contextual visual information, still lacks the ability to recognize its position, remember if it was there before or connect that place to previously archived data. Enable space recognition would allow the robot to independently adapt its behavior according to the environment, for example by loading a series of rules based on the position detected, rather than relying exclusively on what has been said at the code level. It could also distinguish if it is in an internal or external space and regulate its responses accordingly, further improving its autonomy.

Finally, this work has added a new level of ability to the robot: the possibility of using visual and contextual signals of the real world as well as the user's input and long -term memory. This opens many opportunities for more personalized and dynamic interactions. For example, the robot could possibly recognize individual users through facial recognition, without the need to predefined users' identities in the code. It could also deduce a mood of a user with facial expressions or understand actions and gestures to better adapt his answers.

In conclusion, the work presented in this thesis places a solid base for further explorations. It introduces key components that improve both the perception and the intelligence of the robot, indicating the way towards a more autonomous interaction paradigm, aware of the context and focused on the user with which it is interacting.

9.3 Conclusion

This thesis has explored the development of a vision-augmented human-robot interaction framework, aiming to improve the contextual awareness and behavioral intelligence of a social robot. Starting from an existing architecture capable of managing socially appropriate responses and emotional expressions, the work addressed one of its key limitations: the lack of environmental awareness due to the absence of visual perception.

By integrating a visual processing pipeline, based on the robot's camera and the YOLOv5 object detection server, the robot was given the ability to interpret its

surroundings. Two main modules, the Vision Supervisor and the Awake Supervisor, were introduced to manage this new flow of information. These modules enabled the robot not only to answer more appropriately to the user based on contextual data, but also to perform proactive behaviors, such as searching for additional visual information when needed or starting an interaction with the user when a potential rule violation is detected.

The implementation of these modules demonstrated a meaningful step forward in making the robot more autonomous and aware of its environment. The system moved from a purely reactive to the user input interaction model to one that combines reactive and proactive behaviors, making the robot more engaging, capable, and useful in dynamic, real-world settings.

In summary, this thesis contributes to the field of social robotics by showing how visual perception can significantly enhance interaction quality and safety. The results achieved serve as a solid foundation for future developments in context-aware, intelligent robotic systems that can adapt more naturally to the environments and people around them.

Bibliography

- [1] ABDI, E., TOJIB, D., SEONG, A. K., PAMARTHI, Y., AND MILLINGTON-PALMER, G. A study on the influence of service robots' level of anthropomorphism on the willingness of users to follow their recommendations. *Scientific Reports*, **12** (2022), 15266. Available from: <https://doi.org/10.1038/s41598-022-19501-0>, doi:10.1038/s41598-022-19501-0.
- [2] ASSELBORN, T., JOHAL, W., AND DILLENBOURG, P. Keep on moving! exploring anthropomorphic effects of motion during idle moments. pp. 897–902 (2017). doi:10.1109/ROMAN.2017.8172409.
- [3] BELANCHE, D., CASALÓ, L. V., SCHEPERS, J., AND FLAVIÁN, C. Examining the effects of robots' physical appearance, warmth, and competence in front-line services: The humanness-value-loyalty model. *Psychology & Marketing*, **38** (2021), 2357. Available from: <https://onlinelibrary.wiley.com/doi/abs/10.1002/mar.21532>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/mar.21532>, doi:<https://doi.org/10.1002/mar.21532>.
- [4] BELPAEME, T., KENNEDY, J., RAMACHANDRAN, A., SCASSELATI, B., AND TANAKA, F. Social robots for education: A review. *Science Robotics*, **3** (2018), eaat5954. Available from: <https://www.science.org/doi/abs/10.1126/scirobotics.aat5954>, arXiv:<https://www.science.org/doi/pdf/10.1126/scirobotics.aat5954>, doi:10.1126/scirobotics.aat5954.
- [5] BOURGUET, M.-L., CHEN, Y., JIN, Y., RINCON-ARDILA, L., SHI, Y., AND VENTURE, G. Social robots that can sense and improve student engagement (2020). doi:10.1109/TALE48869.2020.9368438.
- [6] BRIENZA, M., ARGENZIANO, F., SURIANI, V., BLOISI, D. D., AND NARDI, D. *Multi-Agent Planning Using Visual Language Models*. IOS Press (2024). ISBN 9781643685489. Available from: <http://dx.doi.org/10.3233/FAIA240916>, doi:10.3233/faia240916.
- [7] CASTELLANO, G., MACCHIARULO, N., AND VESSIO, G. Pepper4museum: Towards a human-like museum guide (2020).
- [8] CUIJPERS, R. H. AND KNOPS, M. A. M. H. Motions of robots matter! the social effects of idle and meaningful motions. In *Social Robotics* (edited by A. Tapus, E. André, J.-C. Martin, F. Ferland, and M. Ammi), pp. 174–183. Springer International Publishing, Cham (2015). ISBN 978-3-319-25554-5.

- [9] DAS, D., HOQUE, M. M., ONUKI, T., KOBAYASHI, Y., AND KUNO, Y. Vision-based attention control system for socially interactive robots. In *2012 IEEE ROMAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pp. 496–502 (2012). doi:10.1109/ROMAN.2012.6343800.
- [10] DAWE, J., SUTHERLAND, C., BARCO, A., AND BROADBENT, E. Can social robots help children in healthcare contexts? a scoping review. *BMJ Paediatrics Open*, **3** (2019), e000371. Available from: <https://doi.org/10.1136/bmjpo-2018-000371>, doi:10.1136/bmjpo-2018-000371.
- [11] DUAN, J., YUAN, W., PUMACAY, W., WANG, Y. R., EHSANI, K., FOX, D., AND KRISHNA, R. Manipulate-anything: Automating real-world robots using vision-language models (2024). Available from: <https://arxiv.org/abs/2406.18915>, arXiv:2406.18915.
- [12] FAN, J., ZHENG, P., AND LI, S. Vision-based holistic scene understanding towards proactive human–robot collaboration. *Robotics and Computer-Integrated Manufacturing*, **75** (2022), 102304. Available from: <https://www.sciencedirect.com/science/article/pii/S0736584521001848>, doi: <https://doi.org/10.1016/j.rcim.2021.102304>.
- [13] FONG, T., NOURBAKHSH, I., AND DAUTENHAHN, K. A survey of socially interactive robots. *Robotics and Autonomous Systems*, **42** (2003), 143. Socially Interactive Robots. Available from: <https://www.sciencedirect.com/science/article/pii/S092188900200372X>, doi:[https://doi.org/10.1016/S0921-8890\(02\)00372-X](https://doi.org/10.1016/S0921-8890(02)00372-X).
- [14] FORTUNATI, L., CAVALLO, F., AND SARRICA, M. Multiple communication roles in human–robot interactions in public space. *International Journal of Social Robotics*, **12** (2020), 931. doi:10.1007/s12369-018-0509-0.
- [15] GALLEGOS, I. O., ROSSI, R. A., BARROW, J., TANJIM, M. M., KIM, S., DERNONCOURT, F., YU, T., ZHANG, R., AND AHMED, N. K. Bias and fairness in large language models: A survey. *Computational Linguistics*, **50** (2024), 1097. Available from: <https://aclanthology.org/2024.cl-3.8/>, doi:10.1162/coli_a_00524.
- [16] GAO, F., XIA, L., ZHANG, J., LIU, S., WANG, L., AND GAO, R. X. Integrating large language model for natural language-based instruction toward robust human–robot collaboration. *Procedia CIRP*, **130** (2024), 313. 57th CIRP Conference on Manufacturing Systems 2024 (CMS 2024). Available from: <https://www.sciencedirect.com/science/article/pii/S2212827124012502>, doi: <https://doi.org/10.1016/j.procir.2024.10.093>.
- [17] GOODRICH, M. AND SCHULTZ, A. Human–robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, **1** (2007), 203. doi:10.1561/1100000005.
- [18] GUAN, T., ET AL. Hallusionbench: An advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models (2024). Available from: <https://arxiv.org/abs/2310.14566>, arXiv:2310.14566.

- [19] GUEMGHAR, I., PIRES DE OLIVEIRA PADILHA, P., ABDEL-BAKI, A., JUTRAS ASWAD, D., PAQUETTE, J., AND POMEY, M.-P. Social robot interventions in mental health care and their outcomes, barriers, and facilitators: Scoping review. *JMIR Ment Health*, **9** (2022). Available from: <https://doi.org/10.2196/36094>, doi:10.2196/36094.
- [20] IVANOV, S. AND WEBSTER, C. *Perceived Appropriateness and Intention to Use Service Robots in Tourism: Proceedings of the International Conference in Nicosia, Cyprus, January 30–February 1, 2019*, pp. 237–248 (2019). ISBN 978-3-030-05939-2. doi:10.1007/978-3-030-05940-8_19.
- [21] JEONG, H., LEE, H., KIM, C., AND SHIN, S. A survey of robot intelligence with large language models. *Applied Sciences*, **14** (2024). Available from: <https://www.mdpi.com/2076-3417/14/19/8868>, doi:10.3390/app14198868.
- [22] LEE, H. AND HAHN, S. Effect of robot head movement and its timing on human-robot interaction. *International Journal of Social Robotics*, **17** (2024), 3. Available from: <https://link.springer.com/10.1007/s12369-024-01196-0>, doi:10.1007/s12369-024-01196-0.
- [23] LEE, N., KIM, J., KIM, E., AND KWON, O. The influence of politeness behavior on user compliance with social robots in a healthcare service setting. *International Journal of Social Robotics*, **9** (2017). doi:10.1007/s12369-017-0420-0.
- [24] LI, L., YU, X., LI, J., WANG, G., SHI, J.-Y., TAN, Y. K., AND LI, H. Vision-based attention estimation and selection for social robot to perform natural interaction in the open world. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, p. 183–184. Association for Computing Machinery, New York, NY, USA (2012). ISBN 9781450310635. Available from: <https://doi.org/10.1145/2157689.2157746>, doi:10.1145/2157689.2157746.
- [25] MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, **2014** (2014), 2.
- [26] MUBIN, O., STEVENS, C. J., SHAHID, S., AL MAHMUD, A., AND DONG, J.-J. A review of the applicability of robots in education. *Journal of Technology in Education and Learning*, **1** (2013), 13.
- [27] PAYANDEH, A., SONG, D., NAZERI, M., LIANG, J., MUKHERJEE, P., RAJ, A. H., KONG, Y., MANOCHA, D., AND XIAO, X. Social-llava: Enhancing robot navigation through human-language reasoning in social spaces (2024). Available from: <https://arxiv.org/abs/2501.09024>, arXiv:2501.09024.
- [28] QIN, J., CHEN, Z., ZHANG, W., GUAN, D., WU, Z., AND ZHAO, M. Research on active interaction design for smart speakers agent of home service robot. In *Design, User Experience, and Usability. User Experience in Advanced Technological Environments* (edited by A. Marcus and W. Wang), pp. 253–263. Springer International Publishing, Cham (2019). ISBN 978-3-030-23541-3.

- [29] QUIGLEY, M., CONLEY, K., GERKEY, B., FAUST, J., FOOTE, T., LEIBS, J., WHEELER, R., AND NG, A. Ros: an open-source robot operating system. vol. 3 (2009).
- [30] RAMEY, A., CASTRO-GONZÁLEZ, , MALFAZ, M., ALONSO MARTÍN, F., AND SALICHES, M. Vision-based people detection using depth information for social robots: An experimental evaluation. *International Journal of Advanced Robotic Systems*, **14** (2017), 172988141770592. doi:10.1177/1729881417705923.
- [31] RASHED, M. G., SUZUKI, R., LAM, A., KOBAYASHI, Y., AND KUNO, Y. A vision based guide robot system: Initiating proactive social human robot interaction in museum scenarios. In *2015 International Conference on Computer and Information Engineering (ICCIE)*, pp. 5–8 (2015). doi:10.1109/CCIE.2015.7399316.
- [32] ROBINSON, H., MACDONALD, B., AND BROADBENT, E. The role of health-care robots for older people at home: A review. *International Journal of Social Robotics*, **6** (2014), 575. Available from: <https://doi.org/10.1007/s12369-014-0242-2>. doi:10.1007/s12369-014-0242-2.
- [33] SAPKOTA, R., ROUMELIOTIS, K. I., CHEPPALLY, R. H., CALERO, M. F., AND KARKEE, M. A review of 3d object detection with vision-language models (2025). Available from: <https://arxiv.org/abs/2504.18738>, arXiv:2504.18738.
- [34] SHARMA, H. AND JALAL, A. S. A survey of methods, datasets and evaluation metrics for visual question answering. *Image and Vision Computing*, **116** (2021), 104327. Available from: <https://www.sciencedirect.com/science/article/pii/S0262885621002328>, doi:<https://doi.org/10.1016/j.imavis.2021.104327>.
- [35] SILVERA-TAWIL, D. Robotics in healthcare: A survey. *SN Computer Science*, **5** (2024). doi:10.1007/s42979-023-02551-0.
- [36] THOMAZ, A., HOFFMAN, G., AND CAKMAK, M. Computational human-robot interaction. *Foundations and Trends in Robotics*, **4** (2016), 104. doi:10.1561/2300000049.
- [37] ULTRALYTICS. YOLOv5: A state-of-the-art real-time object detection system. <https://docs.ultralytics.com> (2021).
- [38] WILSON, J. R., AUNG, P. T., AND BOUCHER, I. Enabling a social robot to process social cues to detect when to help a user. *CoRR*, **abs/2110.11075** (2021). Available from: <https://arxiv.org/abs/2110.11075>, arXiv:2110.11075.
- [39] YAN, H., HU, X., WAN, X., HUANG, C., ZOU, K., AND XU, S. Inherent limitations of llms regarding spatial information (2023). Available from: <https://arxiv.org/abs/2312.03042>, arXiv:2312.03042.
- [40] ZHANG, B. AND SOH, H. Large language models as zero-shot human models for human-robot interaction. In *2023 IEEE/RSJ International Conference on*

Intelligent Robots and Systems (IROS), p. 7961–7968. IEEE (2023). Available from: <http://dx.doi.org/10.1109/IROS55552.2023.10341488>, doi: 10.1109/iros55552.2023.10341488.