Escola Tècnica Superior d'Enginyeria Electrònica i Informàtica La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Telecomunicació

AssIstent DE tasques Domèstiques (AIDED)

Selene Caro Via

[illegible]

Raquel Ros Espinoza

[illegible]

ACTA DE L'EXAMEN
DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Selene Caro Via

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

AssIstent DE tasques Domèstiques (AIDED)

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL　　　　　VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Abstract

Instrumental Activities of Daily Living (IADLs) are those activities related to allowing an individual to live independently in a community, and involve tasks like cooking, cleaning, transporting, doing laundry, and managing finances. In United States a 31.7% of the citizens have limitations in IADLs, affecting more to elders.

For this reason, in this project, we present our first steps towards a collaborative robot that supports Instrumental Activities of Daily Living. More precisely, we propose a collaborative table-setting scenario where a human and a robot work together to coordinate their actions to fulfil the task while considering their "abilities". Thus, given a set of objects on the table and after a user command, the robot sets the table in a different configurations, e.g. *table for one*, or *table for two*, either on its own, or requesting support to the user at specific points.

To this end, a system development and integration has been carried out. The main modules are: a *Task Planner*, in charge of planing the sequence of actions to achieve the task goal; a *Motion Planner*, to plan the object manipulation trajectories without colliding with the environment; a *Verbal Communication*, to interact with the user through voice; and a *Vision* module, to detect and localize objects in the environment.

Besides successfully fulfilling the task at hand in "ideal" scenarios, the system also adapts to unexpected situations such as having more objects than needed, missing objects, unfeasible robot movements, as well as handling vision or manipulation inaccuracies.

The system has been mainly tested in simulation due to the COVID-19 confinement, which has proved an additional challenge to the project.

Keywords— Assistive robotics, Human-Robot Interaction, table setting, ROS integration, planning, vision, verbal communication

Resumen

Instrumental Activities of Daily Living (IADLs) (o actividades instrumentales de la vida diaria) son aquellas actividades relacionadas con permitir que una persona viva de forma independiente en una comunidad e involucran tareas como cocinar, limpiar, mobilizarse, lavar la ropa y administrar las finanzas. En Estados Unidos un 31,7 % de los ciudadanos tiene limitaciones en IADLs, afectando más a la gente mayor.

Por ello, en este proyecto presentamos los primeros pasos hacia el desarrollo de un robot colaborativo que soporte las actividades instrumentales de la vida diaria. Más concretamente, proponemos un escenario colaborativo de “poner la mesa” donde un humano y un robot trabajan juntos coordinando sus acciones para cumplir con la tarea considerando las “habilidades” de cada uno. Así pues, el escenario consiste en dada una mesa con diversos de objetos, el usuario indica al robot qué configuración de la mesa desea, como por ejemplo, *mesa para uno*, y el robot planifica y ejecuta las acciones necesarias para llegar al objectivo, ya sea individualmente o con ayuda del usuario.

Para ello se ha llevado a cabo el desarrollo e integración del sistema, donde los módulos principales son: un *Planificador de Tareas*, encargado de planificar la secuencia de acciones para lograr el objetivo de la tarea; un textit Planificador de Movimientos, para planificar las trayectorias de manipulación de objetos sin colisionar con el entorno; un módulo de *Comunicación Verbal*, para interactuar con el usuario a través de la voz; y un módulo de *Visión*, para detectar y localizar objetos en el entorno.

Además de cumplir con éxito la tarea en cuestión en escenarios “ideales”, el sistema también es capaz de adaptarse a situaciones inesperadas como ser: tener más objetos de los necesarios, que falten objetos necesarios, gestionar movimientos del robot inviables, así como manejar imprecisiones de visión o manipulación.

El sistema se ha probado principalmente en simulación debido al confinamiento en respuesta al COVID-19 que nos ha impedido acceder al robot físico, generando un desafío adicional para el desarrollo proyecto.

Palabras clave— Robótica de Asistencia, interacción persona-robot, configuración de la mesa, integración con ROS, planificación, visión, comunicación verbal

Resum

Instrumental Activities of Daily Living (IADLs) (activitats instrumentals de la vida diaria) són aquelles activitats relacionades amb permetre a una persona viure independentment en una comunitat i que impliquen tasques com cuinar, netejar, movilitzar-se, rentar la roba i gestionar les finances. Als Estats Units, un 31,7% dels ciutadans tenen limitacions a IADLs, afectant més a la gent gran.

Per aquest motiu, en aquest projecte, presentem els nostres primers passos en el desenvolupament d'un robot col·laboratiu que doni suport a les activitats instrumentals de la vida quotidiana. Més exactament, proposem un escenari col·laboratiu que permeti parar taules on una persona i un robot treballen junts coordinant les seves accions per assolir la tasca tenint en compte les "capacitats" de cadascú. Així doncs, l'escenari consisteix en que donat un conjunt d'objectes a la taula, l'usuari indica al robot una configuració per parar la taula, com ara *taula per a un* o *taula per a dos*, i aquest planifica i executa les acciones necessàries per assolir la tasca, ja sigui individualment o amb l'ajuda de l'usuari.

Amb aquesta finalitat, s'ha dut a terme un desenvolupament i integració del sistema, on els mòduls principals són: un *Planificador de Tasques*, encarregat de planificar la seqüència d'accions per assolir l'objectiu de la tasca; un *Planificador de Moviments*, per planificar les trajectòries de manipulació d'objectes sense col.lisionar amb l'entorn; un mòdul de *Comunicació Verbal*, per interactuar amb l'usuari a través de la veu; i un mòdul de *Visió*, per detectar i localitzar objectes de l'entorn.

A més de complir amb èxit la tasca en escenaris "ideals", el sistema també s'adapta a situacions inesperades com ara: tenir més objectes dels necessaris, que faltin d'objectes necessaris, gestionar moviments inviables pel robot, així com manegar inexactituds per part dels mòduls de visió o manipulació.

El sistema s'ha provat principalment en simulació a causa del confinament en resposta a la COVID-19 on l'accés al robot físic ha sigut impossible, implicant un repte addicional pel desenvolupament del projecte.

Paraules clau— Robòtica Assistencial, interacció persona-robot, configuració de la taula, integració de ROS, planificació, visió, comunicació verbal

Contents

List of Figures

List of Tables

Acronyms

2D two-dimensional. 7

3D three-dimensional. 7, 12, 21, 45, 48

AAL Active and Assisted Living. 11, 15

ABSTRIPS Abstraction-Based Stanford Research Institute Problem Solver. 29

AD Alzheimer Disease. 15

ADLs Activities of Daily Living. 1, 13

AIDED *AssIstent DE tasques Domèstiques*. 19–21, 37, 61, 76

BFS Bread-First-Search. 30

CHOMP Covariant Hamiltonian Optimization for Motion Planning. 46

CNN Convolutional Neural Network. 53

COVID-19 Corona Virus Disease 2019. i–iii, 101

DFS Depth-First-Search. 30

e.g. for example. i, 2

GUI Graphical User Interface. 45

HOPE Smart Home for Elderly People. 13

HRI Human-Robot Interaction. i, 4–8, 16, 52, 102

HSP Heuristic Search Planner. 31

i.e. that is. 42, 92

IADLs Instrumental Activities of Daily Living. i–iii, vii, 1, 2, 11, 13

IP Internet Protocol. 48, 49

MD Mild Dementia. 15

MDP Markov decision processes. 23

MIT Massachusetts Institute of Technology. 17

NAT Network Address Translation. 48

OCC Ortony, Clore and Collins's. 7

OID Open Images Dataset. 53

OMPL Open Motion Planning Library. 46

PAD Pleasure, Arousal, and Dominance. 7

PDDL Planning Domain Definition Language. 16, 21, 30, 31, 37–40, 57, 58

PIR Passive Infrarred Sensor. 13

POMDP Partially observable Markov decision process. 23

POPF Partial Order Planning Forwards. 16, 39, 42

PUnS Planning with Uncertain Specifications. vii, 17

QR Quick Response. 17, 101

ROS Robot Operating System. i, 9, 20, 37, 40, 42, 45, 46, 58, 61, 101

RTDE Real-Time Data Exchange. 44

SBPL Search-Based Planning Library. 46

SKT Syndrom KurzTest. 15

SOCRATES Social Cognitive Robotics in The European Society. 15, 16, 101

SRDF Semantic Robot Description Format. 46

STOMP Stochastic Trajectory Optimization for Motion Planning. 46

STRIPS Stanford Research Institute Problem Solver. 23, 29, 30, 61

TFM *Treball Final de Màster*. 14, 53, 61, 99

TTL Time To Live. 49

TTS Text-To-Speech. 6, 52, 59

UR Universal Robots. vii, 42–46, 52, 88

URDF Unified Robot Description Format. 45

URL-MONITOR *Univeritat Ramon Llull - coMprensiÓ de l'eNtorn a través de vIsió de compuTadOR*. 19, 20, 47, 61, 72, 101

YOLO You Only Look Once. 53

1 Introduction

Instrumental Activities of Daily Living (IADLs)[19] are those activities related to allowing an individual to live independently in a community. It is important not to confuse ADLs with IADLs. ADLs are essential activities that allow unassisted survival such as: dressing (the ability to wear clothes in an acceptable fashion), eating (self-feeding so as to be in perfect health), ambulating (walking, sitting and then standing up, getting in and out of vehicles), toileting (ability to use the facilities and wipe and wash without help), and bathing to maintain proper hygiene. On the other hand, IADLs involve tasks like cooking, cleaning, transporting, doing laundry, and managing finances.

Early detection of an individual's inability to perform IADLs is important because it can indicate one or many potential issues going on, as well as a warning for worse problems. Some causes of this inability that tend to be the first step in the progression of IADLs loss include:

- Medication side effects, mixtures, or overuse can cause mental fog.
- Not getting enough sleep can cause or worsen the memory problems.
- Initial stages of dementia, caused by many factors, can involve memory loss.
- Medical issues or worsening pre-existing conditions.

Usually, when the individual is unable to perform IADLs or ADLs at home, needs to seek outside for help, such as family members, home caregiver, day centers, or a home care.

In Figure 1 we can see the percentage of persons having limitations in IADLs. People aged between 18 and 44 years represents a 1.7%, between 45 and 64 years represents a 4.0%, between 64 and 74 represents a 6.6%, and those of 75 years and over represents a 19.4% of the total people. We appreciate that a 31.7% of the citizens in United States presents limitations in IADLs.

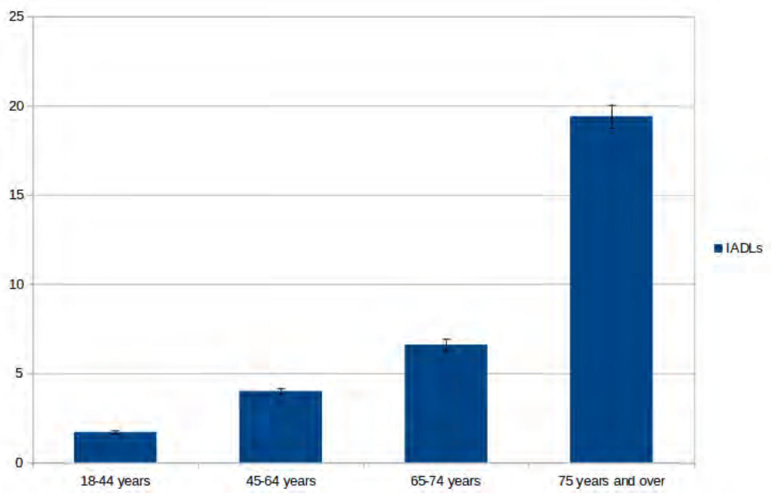


Figure 1: Age-adjusted percentages (with standard errors) of persons having limitation in IADLs among persons aged 18 and over, by age: United States, 2018 [27].

In this project we present our first steps towards a collaborative robot that supports Instrumental Activities of Daily Living. More precisely, we propose a collaborative table-setting scenario where a human and a robot work together to coordinate their actions to fulfil the task while considering their "abilities". As shown in Figure 2, the scenario consists in a set of object on the table and the user, in this case an elder, asks the robot to set the table in a different set of configurations (e.g. *table for one*, *table for two*, and so on). The robot, then, plans and starts moving the objects of the table to their positions. If the robot cannot perform a given task, then it asks the user to perform it. To achieve this goal, we integrate a *Vision* system to monitor the environment (state of the world), a *Task Planner*, a *Motion Planner*, a *Supervisor* system, and a *Verbal Communication* module based on commercial system.

Figure 2: Elder laying the table with our system.

Not all the robots are designed to do this type of collaboration and, for this reason, we are going to talk about a possible classification of them, shown in Figure 4. We can appreciate that there is a first differentiation between industrial and assistive robots. The industrial robots are those that do not consider working with people or having them around; they are focused on production processes. An example could be FANUC M-2000iA/2300 [26] (see Figure 3), which is the strongest robot in the market nowadays, with a load capacity of 2.3 t.

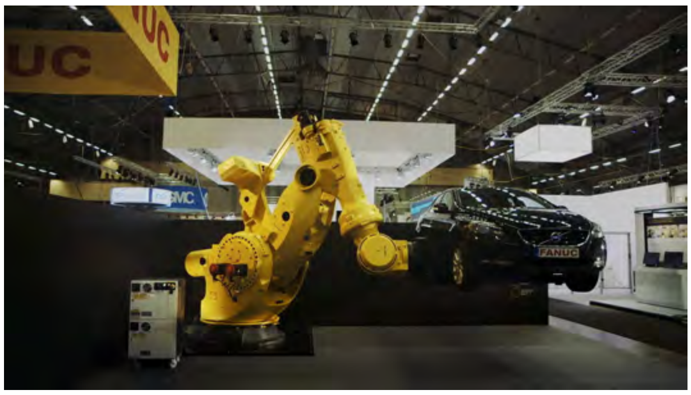


Figure 3: FANUC M-2000iA/2300.

Although these robots are not considered to work with humans around, there is a new trend in which this kind of robots collaborate with people. They are called "cobots". These can be used in many sectors, such as:

- Food and agriculture.
- Metal, machining, and metallurgy.
- Automotive and subcontractors.
- Electronics and technology.
- Plastics and polymers.
- Furniture and equipment.
- Pharmaceutical and chemical.
- Scientific and research.

Examples of companies that create cobots are Universal Robots, KUKA, and ABB.

The assistive robots, can be non social assistive or assistive social robots. The firsts ones, are those that help humans with a task, but do not interact; it is physical assistive technology. An example could be Roomba, or robots developed for rehabilitation and that are not in any way socially interactive, such as intelligent robotic wheelchairs, artificial limbs, and exoskeletons.

Finally, assistive social robots can be divided by service robots and companion robots. Service robots are used as functional devices and are not primarily designed for affective support. Examples could be TIAGo and PR2. Companion robots are affective. They provide pet-like companionship which is beneficial to the health and well-being of elderly users, but they do not provide functional assistance. They are also used with children with special needs. Examples of this type of robots are PLEO, Aibo and PARO.

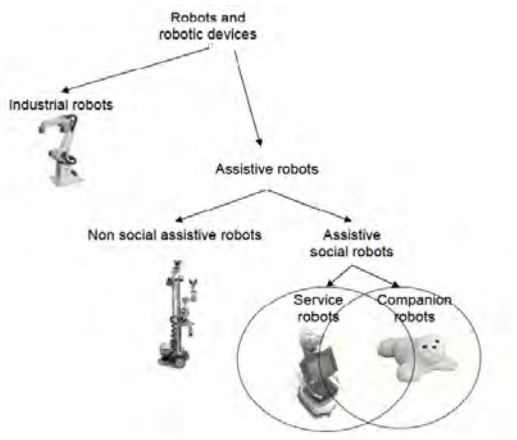


Figure 4: Robot's classification example by [30].

1.1 Human-Robot Interaction (HRI)

From the assistive social robots Human-Robot Interaction (HRI) [20] is born, concerned with the ways in which robots interact with people in the social world. These interactions usually include physical embodied robots; therefore, we have to understand how to design that embodiment, both in terms of software and hardware, as is common-place in robotics, and in terms of its effects on people and the kinds of interactions they can have with such a robot. For this reason, we are going to briefly describe multiple aspects to take into account in HRI: the design of the robot, the spatial interaction, the nonverbal interaction, the verbal interaction, and the emotion.

The design space of robots is relatively large and considers questions regarding from function, level of autonomy, interaction modalities, and how all these fit with particular users and contexts. The affordances and the design patterns are two important concepts in the robot design. The affordances of the robot need to be explicit, so that people can understand the robot capabilities and limitations appropriately and adapt their interactions accordingly. The design patterns "describe a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing the same way twice"[13].

When combining these two ideas of design affordances and patterns in the process of HRI design, HRI researchers have suggested some of the following principles to consider when developing the appropriate robot forms, patterns, and affordances:

- Matching the form and function of the design. This principle relates the design of the robot with the expectations it creates to humans. To illustrate, if a robot has eyes, people will expect it to see. People can also be prompted to associate specific social norms and cultural stereotypes with robots through design.

- Underpromise and overdeliver. When people's expectations are raised by the robot's appearance or by introducing the robot as intelligent or companion-like, and these expectations are not met by its functionality, people feel disappointed and negatively evaluate the robot. Hence, this principle is about decreasing people's expectations about robots.

- Interaction expands functions. This principle is related to take advantage of the people's blanks left open by the design. Particularly, for robots with limited capabilities, to design them in a somewhat open-ended way, as this will allow people to interpret the design in different ways.

- Do not mix metaphors. This principle remarks the importance of the coordination between the robot's capabilities, behaviors, affordances for interaction, and so on. This is related to the Uncanny Valley (Figure 5) because inappropriately matched abilities, behaviors, and appearance often lead to people having a negative impression of the robot and, therefore, they do not interact with it.

  Definition 1. Unncany Valley: In 1970, Japanese roboticist Masahiro Mori wrote an article entitled "The Uncanny Valley"[34], in which he argued that as human likenesses appear more realistic there is an increasing

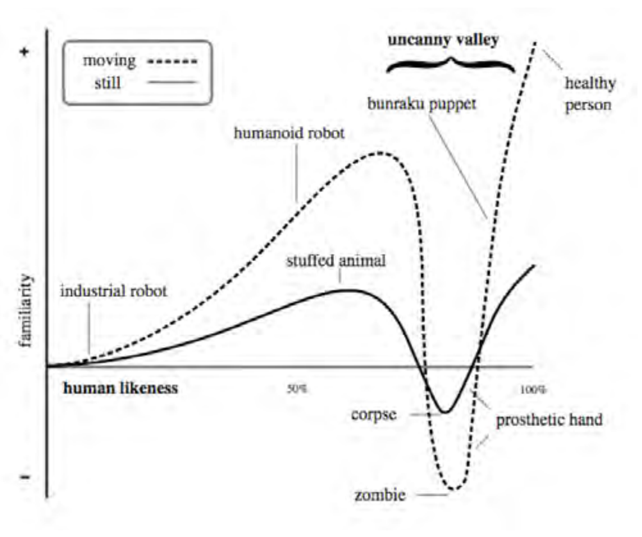


Figure 5: The Uncanny Valley.

level of comfort and familiarity up to a certain level. Yet at the point when a human facsimile is almost fully convincing it becomes repellent. This sudden dip in our emotional response is the Uncanny Valley. If we have a robot that it is in this valley, we would not be able to interact with people. Also, depending on the culture, our robot will be in one point or another of the graph.

The spatial interaction is related to the placement and movement of robots with respect to people. This factor also has to be considered when designing HRI, as if robots do not respect the personal space of the user, will evoke negative reactions or even rejection and withdrawal to the user. Robot designers can attempt to increase acceptance of the robot by having it keep an appropriate distance and adjusting its position to create a fitting interaction experience. To illustrate, imagine a robot arm and a person collaborating on a shared task. The robot must take the socially appropriate distance into account when computing a path for its end effector to reach its given goal. This fact may make the robot's movement inefficient from a purely functional standpoint, but it will lead to a more positive evaluation of the interaction by the user. Now, imagine the robot handing an object to the user. The user prefers a robot to behave with "legibility", a way that allows the user to understand the robot's goal and intention. Having this in mind, the robot could hand over an object to a person in many ways, but the most energy-efficient way may be incomprehensible to a user, who may think the robot is invading its personal area. For this reason, researchers have developed algorithms to control a robot arm to generate legible motions while reaching a given goal. To finish with this aspect in HRI, we also need to take into account the personal area of the user when the robot and the user talk. If the user speaks quietly or there is a lot of noise in the room, the robot may not understand what the user is saying. Therefore, a robot would need to conduct a careful computation to decide what distance it should keep during an interaction with the human.

As known, humans express more feelings with the nonverbal communication rather than verbal. Through nonverbal communication, people can signal mutual understanding, shared goals, and common ground. They can communicate thoughts, emotions, and attention. And they can do so in a more subtle, indirect manner than through verbal expression. Nonverbal cues produced by people when interacting with a robot can indicate whether a person is enjoying the interaction and whether the person likes the robot or not. In HRI, as in human-human interactions, verbal and nonverbal cues might be contradictory. To illustrate, when the robot produces gestures that do not match the rhythm or meaning of its speech or when it does not respond appropriately to user's nonverbal cues the interaction can appear awkward. On the other hand, when speaking to a robot, people would expect it to turn its head toward them and make eye contact with them, showing that it is attending to what they say. When the robot stares straight ahead and does not acknowledge users' presence or spoken requests, the interaction breaks down. Having seen these, there are more types of nonverbal interaction, such as gaze and eye movement, gesture, mimicry and imitation, touch, posture and movement, and interaction rhythm and timing.

The verbal interaction is a common mode of communication designed into robots. Producing robot speech is much simpler than understanding human speech, however. To illustrate, the speakers on which the models have been insufficiently trained, such as young speakers or elderly speakers, still provide a challenge. In addition, the local dialects of languages or nonnative speakers will often result severely reduced recognition performance. To add more complicity, the acoustic environment, such as noisy, reverberating, or crowded spaces, also is a determining factor. Nowadays, there are numerous speech-recognition engines available. The more common is to use a remote service that allow developers to send a recorded speech fragment over the internet, and the transcribed speech is returned soon after. It allows the robot to have a relatively low-cost computational core.

On the other hand, developers also need to transform Text-To-Speech (TTS). In HRI is important to consider which voice fits the robot and its application. For example, a small robot requires a voice that matches its appearance, rather than a commanding baritone. In other cases, though, a natural-sounding TTS engine might sit uneasily on an artificial agent. A curious fact is that the type of voice also affects the social perception of social robots. For instance, robots with a male voice are anthropomorphized and evaluated more favorably by men than by women, and vice versa.

Finally, another important aspect in HRI are the emotions. They can motivate and modulate behavior and are a necessary component of human cognition and behavior. Emotions are usually seen as being caused by an identifiable source, such as an event or seeing emotions in other people. They are often externalized and directed at a specific object or person. The successful communication of emotions promotes survival, enhances social bonds, and minimizes the chances of social rejection and interpersonal physical aggression. A robot that is not programmed to share, understand, or express emotions will thus run into problems when people interpret its behavior as disinterested, cold, or plain rude. There are various emotion models, such as:

- Ortony, Clore and Collins's (OCC) model. It specifies 22 emotions categories based on balanced reactions to situations. It also offers a structure for the variables, such as the likelihood of an event or the familiarity of an object, which determines the intensity of the emotion types. It contains a sufficient level of complexity and detail to cover most situations an emotional robot might have to deal with.
- Russel's two-dimensional (2D) space of arousal and valence captures a wide range of emotions on a 2D plane and is one of the simplest emotion models that still has sufficient expressive power for HRI. Although, places "angry" and "afraid" side by side, whereas most people would argue that these are vastly different emotions.
- The framework by Mehrabian and Russel (1974) adds a third axis. This framework captures emotions in a three-dimensional (3D), continuous space, with the dimensions consisting of Pleasure, Arousal, and Dominance (PAD). This space model has been used on many social robots to model the user's and the robot's emotional state.

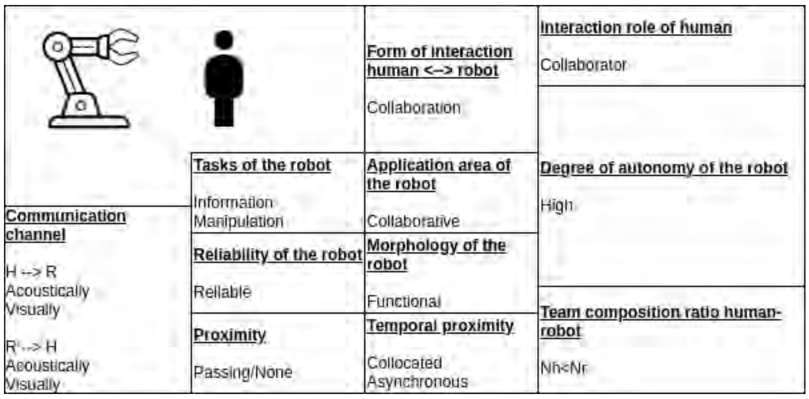
We must say that the addition of more information, such as the context of the interaction, animated rather than still expressions of emotion, and body language, allows to increase the recognition rate, both by people and by algorithms.

When we talk about HRI, we also must consider which role will have the user, which can be:

- Teammate: The human and the robot have a common goal and must solve it together. They can give commands to each other, interaction such as gestures and voice can be helpful. The human needs to understand the limitations of the robot. A company that creates robots to work in these conditions with humans is Boston Dynamics.
- Bystander: There is no direct interaction between this user and the robot. Although, the robot must co-exist in environment with the user. An example robot could be Roomba.
- Interacting user: This user interacts with the robot to get information, for instance. An example robot could be Pepper.

It is not the same to have two users working together than having a user working with a robot. This is because the cognition between one and another is different. The robot may not understand some verbal or gesture expressions. So, given one human and one robot working together, HRI awareness is the understanding that the human has of the location, activities, status, and surroundings of the robot; whereas the knowledge that the robot has of the human are the human's commands necessary to direct its activities and the constraints under which it must operate.

Having said all this, now we are going to talk about the taxonomy [49] of our robot (see Figure 6). The communication channel to interact the human to the robot are acoustically, so the user can talk to the robot, and visually, as the user can move objects and the robot knows that. On the other hand, the robot communicates to human through voice, as it indicates what it is going to do and, visually, as it moves the objects and the user can see it.



Form of interaction human <--> robot

Collaboration

Interaction role of human

Collaborator

Tasks of the robot

Information
Manipulation

Application area of the robot

Collaborative

Degree of autonomy of the robot

High

Communication channel

H --> R
Acoustically
Visually

R --> H
Acoustically
Visually

Reliability of the robot

Reliable

Morphology of the robot

Functional

Proximity

Passing/None

Temporal proximity

Collocated
Asynchronous

Team composition ratio human-robot

Nh<Nr

Figure 6: Our robot taxonomy [28].

The main task of the robot is the manipulation of the different objects to put them in their place, but also give feedback to the user.

The robot will always do its best to achieve the end goal, for this reason, it is reliable.

When talking about proximity, the robot can be defined by different proximity behaviors: avoiding, passing, following, approaching, touching, and/or none. In our case, the robot will not consider the user movements, so it is passing. Furthermore, in an ideal scenario (meaning there are all the needed objects, and the robot can reach them all and put them in their correct positions) the robot does not need the human at all, for this reason it can also be none.

The temporal proximity categorizes an HRI system based on whether human and robot share the same space (collocated, non-collocated), and whether they act at the same time or not (synchronous, asynchronous). In our case, the robot and the human are one in front of the other, so it is collocated. On the other hand, they can work synchronous, but to avoid the robot injuring the user, we have raised it as the robot works and if it needs the human help, it stops moving.

The robot morphology variable describes the robot appearance type, which could be anthropomorphic, zoomorphic, and functional. In our case it is an arm robot, so it is functional.

The main idea is that the robot plans the actions to do in order to achieve the goal and starts setting the table, if there is any problem, such as there are more objects, some objects are missing or the robot cannot do a movement, it will ask for help to the human. For this reason, it is collaborative, and the team composition ratio human-robot is higher for the robot.

Finally, the degree of autonomy of the robot is high, as the robot can see where all the objects are, the user gives a task and it creates the plan to achieve the goal, and deals with multiple problems that may arise.

1.2 Objectives

The main goal of the project is to develop a robotic assistant to assist a user performing a routine task in natural and efficient way. For this purpose, a robotic arm with vision, planning and verbal communication abilities is used. The chosen activity is to “set the table”. In the proposed scenario different elements are distributed around the table. Through a simple voice command from the user, the robot must first plan the actions needed to achieve the goal, execute them while interacting with the user when necessary, and solve any unexpected event that might impact on the original plan. To do this, it needs to know what the user’s preferences are, distinguish which objects are in the environment and know which ones should be manipulated.

The main objectives of this project are:

- To integrate cutting-edge technologies: vision, voice, planning and robot control.
- To customize and adapt the execution of the task: depending on the scenario and on the events that take place during the execution, the robot acts in one way or another.
- To provide a smooth interaction between the robot and the user that allows the task to be performed collaboratively.

1.3 Contributions

The contributions of this project are:

- Development of a first prototype of a collaborative robot to support instrumental daily life tasks.
- Study of multiple task planning approaches.
- Implementation of a basic way to communicate with the robot.
- Integration of different modules in the same system through ROS. For this, multiple messages, and services to send petitions between the different nodes have been created.
- Modularity of the system. Each module can be changed in an easy way, even the arm robot.
- Integration of an Echo device and development of an Alexa skill that can communicate with our system and with which the user can send commands.
- Robust system, that can adapt itself depending on the situation and what the user wants. It can adapt itself if:
  - in the scenario there are more objects than needed,
  - there are missing objects,
  - the object is not in its final position although having moved it,
  - the robot cannot move an object because the operation is out of its accessible area,
  - the scenario completely changes.

2 State-of-the-art

In this Section we will explain related work and evaluate them. First, we are going to talk about related systems dedicated to IADLs. More specifically, we will have a look at some projects of the Active and Assisted Living (AAL) program [1], which is a funding program that aims at creating better quality of life for older people and to strengthen industrial opportunities in the field of healthy ageing technology and innovation. Many of these projects are focused on the development of smart home technologies such as:

- SmartHeat [11] is a project which allows every single radiator to be controlled in a smart, easy, and safe way (see Figure 7). It is equipped with a set of technologies to understand the environment and the user needs and to wirelessly communicate with radiators or with smartphones and tablets. They did a first pilot study in Austria and Switzerland in 2017. They started with five households in Switzerland and three households in Austria, although, there were two households in Switzerland that were dropped on the installation day due to unexpected incompatibilities, and the three households in Austria dropped as a result of system instability. They also appreciate that the installation was complex and rather time-consuming. Although these drawbacks, the users expressed a strong interest of trying again with the optimized system.

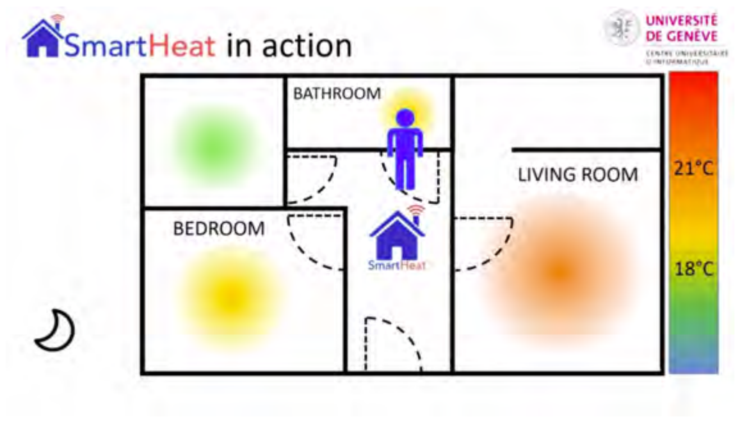


Figure 7: SmartHeat.

- CARU *cares* [2] is a digital flatmate, based on a voice assistant, that helps people in need of care to lead more independent, autonomous lives (see Figure 8). The main idea is that it allows communication between the nursing home, the care givers, the relatives, and so on. This project is in its beginnings, so they have not done yet a test. Although, they have done workshops and interviews with affected care and nursing staff, which make them add, in April 2020, new features:
  - CARU can establish a connection to the hotline of the care facility by voice command. If the center is outside service hours, CARU will automatically record the voice message and transmit it as soon as

the center is open again. It can also happen vice versa; the service center can send a voice message to CARU at any time.

- They add an arrival time notification based on the current personnel planning data and the current movement data of the nursing staff.
- It will also give assistance to the nursing staff, being able to "read out" task lists and leave a voice memo to the next caregiver.

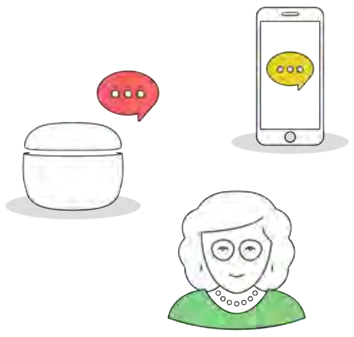


Figure 8: CARU cares.

- FEARLESS [5] is a project created by cogvisAI in 2007 and designed to detect a wide range of risks with a single sensor unit, enhancing mobility and enabling elderly to take active part in the self-serve society by reducing their fears. They have a 3D smart-sensor, shown in Figure 9, that allows:
  - fall prevention,
  - fall detection,
  - absence detection,
  - aggression detection,
  - suicide prevention,
  - lighting control.

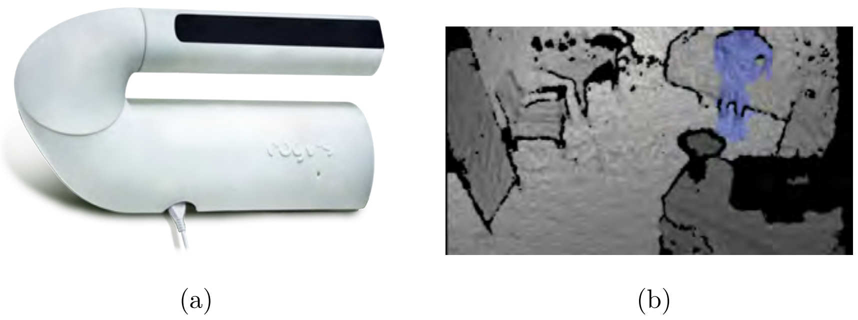
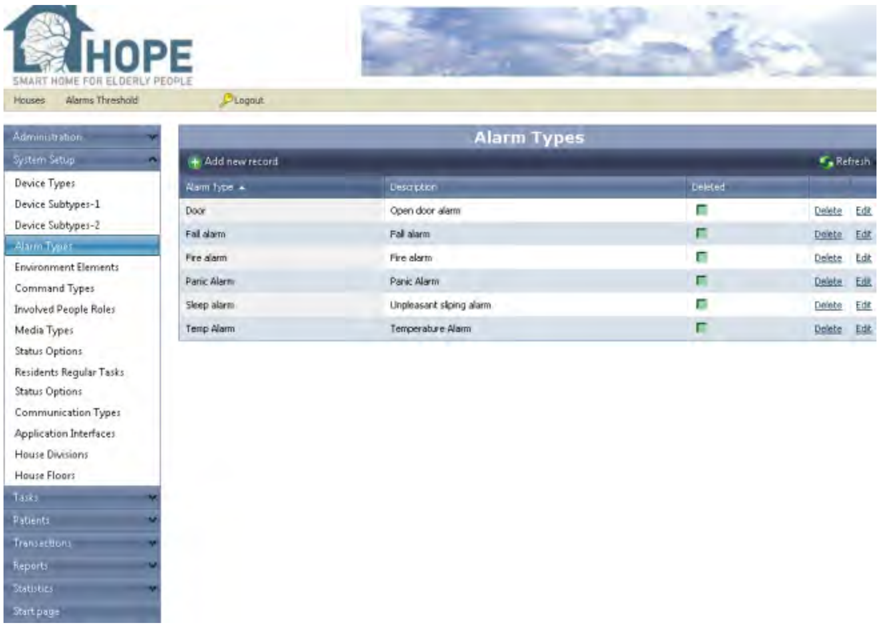


Figure 9: FEARLESS project. (a) cogvisAI 3D Smart-Sensor. (b) Depth image produced by a cogvisAI sensor.

- The HOPE project [7] is based on a smart home that enables the elderly people with Alzheimer's disease to use innovative technology for a more independent life, easy access to information, monitor their health, and serve as a source of inspiration for users as well as for people working with assistive devices. The idea is to set different sensors in the house of the elder and have control and feedback through a website, that the relatives and the doctors have access to it. In Figure 10 we can see that they have access to the alarms that will be created when a sensor overcome the threshold value.



Alarm Types

Alarm Type	Description	Deleted		
Door	Open door alarm	🟩	Delete	Edit
Fall alarm	Fall alarm	🟩	Delete	Edit
Fire alarm	Fire alarm	🟩	Delete	Edit
Panic Alarm	Panic Alarm	🟩	Delete	Edit
Sleep alarm	Unpleasant slping alarm	🟩	Delete	Edit
Temp Alarm	Temperature Alarm	🟩	Delete	Edit

Figure 10: HOPE private website, Alarm Types interface.

They did tests in Italy, Spain, and Greece in 2011, and used:

  – Temperature sensors.
  – PIR sensors to detect people movements,
  – Intrusion detectors to verify that the door is closed correctly.
  – Energy monitor and remote switch for wall plug.
  – Gas detectors.
  – Fall and pulse detector.

Thanks to it, they appreciate an improvement of a 3.5% on the ADLs and of a 2.8% on the IADLs, among other parameters, that also improve. The elders, relatives and doctors were really satisfied with the system.

There are other projects focused on doing exercises, such as CO-TRAIN [4]. Its main objective is to develop, test and validate a training and coaching system that addresses the specific deficiencies of older adults with dementia and frailty. The physiotherapist creates a training program with exercises that the user can visualize on their phones (see Figure 11) and do later at home. The physiotherapist can monitor the user and adjust the exercises depending on the user's feedback. Tests are being done with 60 participants from Austria, Switzerland, and the Netherlands (20 per country), aged between 65-90 years old, with a diagnosis of frailty and mild to moderate cognitive impairment.

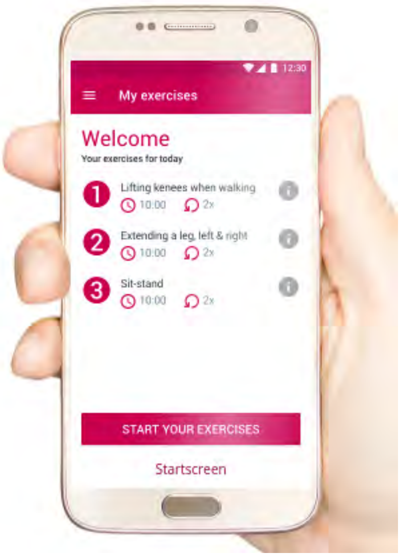


Figure 11: CO-TRAIN application, My exercises interface.

A somehow similar project, compared with the one presented in this TFM, is ChefMySelf [3]. It helps elder users with motor or cognitive impairments to prepare meals. ChefMySelf gives, through a tablet screen, step-by-step instructions for each recipe. The tablet also connects to a food processor to set it up and have the meal prepared. Throughout the project numerous usability tests were conducted in order to design the smoothest possible interaction with the system (see Figure 12). In April 2015 [37], they did a final test with 10 older adults from Italy and from the Netherlands. They used the complete ChefMyself system within the comfort of their own homes. These trials happened in two different rounds in both countries and they included end-users who mirror the characteristics of the target groups that were defined within the course of the project. In this first round there were women and men participants and people with varying cooking experience. They wanted to know the impact of the solution in different user profiles. The second round started later, again with 10 participants across Italy and the Netherlands.

During the trial period, participants had the opportunity to accept or edit weekly meal plan recommendations, use the automatic shopping list generator to help when going out for groceries shopping, or learning more about healthy eating choices. One major feature was the automatic setting of speed, temperature and time on the kitchen robot, rendering the human-machine interaction much simpler, as the user just had to follow the steps in the recipe and hit "go". After an effortless preparation of a healthy and tasty recipe, users could then share it on a social network purposefully designed to bring older adults together around the issues of healthy eating and, thus, prevent isolation.

After completion of the tests [31], the conclusion was that the system could still be improved in some aspects. These mainly concerned of the food processor itself, the recipes and increasing the possibility of getting truly personalized diet advice through the system. This concerned, for example, easier cleaning of the jug and even more adaptation of the recipes per country.

Figure 12: Tests for ChefMyself.

We appreciate that the mentioned AAL projects do not integrate robots to help people so they cannot directly provide support in tasks where moving objects is required. Hence, our project helps in an unaddressed sector: elder people with motion limitations doing basic tasks, such as setting the table.

Having seen that AAL projects do not use robots directly, we want to also mention projects that are dedicated to table-settings. To start with, we want to talk about SOCRATES [12]. It is created to develop the field of social robotics with an application focus on robotics in eldercare. More specifically, their goal is to provide personalized one-on-one cognitive training for Mild Dementia (MD) and Alzheimer Disease (AD) people. They propose a brain-training sessions of sorting tokens, an exercise inspired by the Syndrom KurzTest (SKT) neuropsychological test. Their system has two interaction loops [14]. First, the caregiver interacts with the robot to set up the initial desired behavior of the robot (for example, more helpful or more challenging). In the second loop, the robot interacts with the patient while s/he is playing a sorting tokens exercise. In [16] they focus in this second loop. The authors propose an adaptive module, that selects the best suitable action of engagement to support the patient at each step of the exercise. It has four levels of engagement:

1. Encouraging the user using verbal interactions.

2. Using verbal communication and gestures the robot shows an area of the board where the solution can be found.

3. Using verbal communication and gestures the robot shows the location of the token to move, hence the solution.

4. Picking up the correct token and offering it to the user so s/he only has to place it in its correct position.

Also, this loop has a safety module that monitors the user's safety and adapts the robot behavior accordingly. Dangerous behaviors of the user can be detected at two different times:

1. Before starting the engagement action, the Cognitive System tries to persuade the user, or alternatively, change the safety of its own actions.

2. While the robot action is being executed, it detects any safety concern. The robot warns the use about his/her behavior, moves to a safer position, changes its velocity and acceleration of the trajectory to be safer in case of collision, or stops.

In [15] they evaluate the adaptability of their system with real users (see Figure 13). To do so, they use their Cognitive System that relies on extended planning with adaptive capabilities and embed it in a TIAGo robot. They validate that the robot could improve user performance, the interaction modalities can make a difference in improving user's performance, people with no HRI and engineering backgrounds can have almost the same performance as people with those backgrounds, younger participants can perform faster moves but with worse results than older participants.

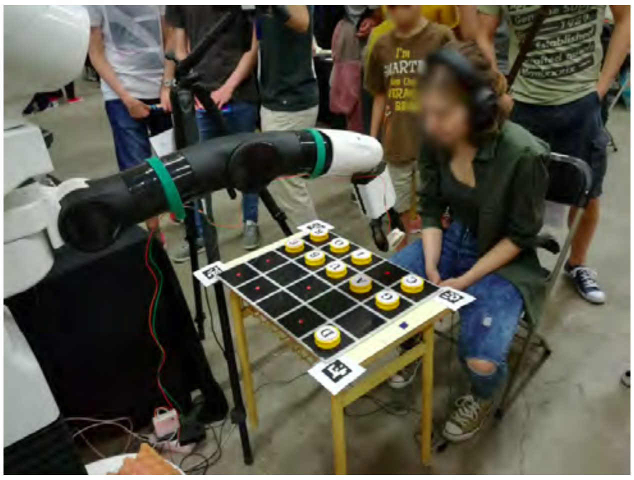


Figure 13: Example of participant interacting with TIAGo in the SOCRATES project.

If we compare their system with ours, they also use PDDL to model the problem and ROSPlan with POPF planner to plan the actions of the robot. A main difference between their project and ours is that they detect user's hands and track them when the robot is moving, so they can move the robot in a safe way. Furthermore, another difference is that the user is the one that moves the different objects, and the robot only helps by verbally communicating or using gestures; for instance, by indicating an area or the piece directly which the user must move to achieve the goal. In our case, the robot does the task and the user helps in case the robot asks for it. Finally, their objective is to provide cognitive training, whereas in our case we want to help users doing a specific task which is laying the table.

To finish this section, we want to talk about a MIT project [32][40] presented in March 2020 while we were also working on our project. They have designed a system that lets the robots learn complicated tasks such as setting the table, under certain conditions, called Planning with Uncertain Specifications (PUnS). In their work, they used eight objects (a mug, a glass, a spoon, a fork, a knife, a dinner plate, a small plate, and a bowl) that could be placed on a table in various configurations. The robot arm first observed a human demonstration of setting the table with those objects. Later, the robot weights many possible placement orderings, even when the items were removed, stacked, or hidden (see Figure 14). The robot made no mistakes over several real-world experiments, and only a handful of mistakes over tens of thousands of simulated tests runs.

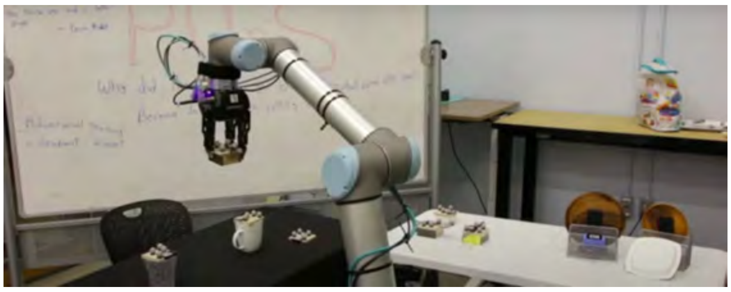


Figure 14: Planning with Uncertain Specifications (PUnS).

The main difference between their work and ours is that we do not include a learning process to teach the robot the final table configuration, but instead, use pre-fixed configurations that the robot shall reach. Another difference is that they user QR codes to recognize the objects, whereas we have a *Vision* module that recognizes them.

3 Design and technical implementation

This Section describes the design and technical implementation of AIDED, part of a bigger project named URL-MONITOR. As already described, the main goal of AIDED is to provide support during a collaborative task. Our project consists of a table setting scenario, where a robot has to set the table (being a dinner for one or for two) using the objects that are on it. As we can imagine, all the available objects will be related to the table setting, although the scenario may not be perfect: there can be objects that are not part of the setting (a mobile phone, for instance) or even missing. In the first situation, the robot will move them to a place of the table where they do not disturb and will ask the user to remove them. In the second situation, the robot can work with only the available objects or wait for the user to bring the missing ones.

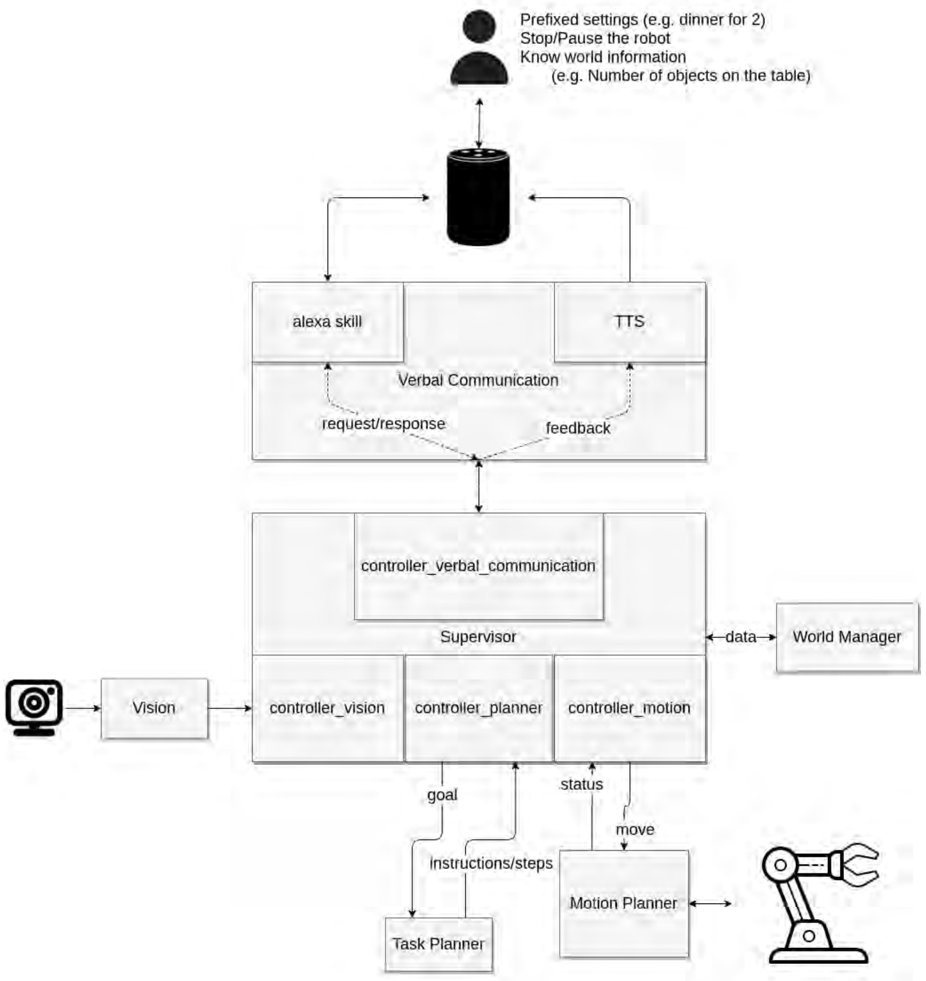


Figure 15: System overview sketch.

A description of the AIDED system can be observed in Figure 15. The system consists of 6 modules, each one with a certain purpose. The *Vision*[1] module identifies objects and their coordinates in the environment. The *Verbal Communication* module receives the petitions from the user and gives feedback back again regarding the state of the system or indicating actions that the user should perform. The *World Manager* manages the state of the world to provide information to the rest of the modules as required. The *Task Planner* module computes the steps or actions the robot and the user should perform to achieve the goal. The *Motion Planner* module controls the robot, making sure it does not collision with the known environment. Finally, as we can see, there is a central module, called *Supervisor*, which supervises and communicates all the modules in the system.

The AIDED system is built using the Robot Operating System (ROS) [9] framework. It is an open source set of software libraries and tools to build robot applications: from drivers to state-of-the-art algorithms, and with powerful developer tools. In Figure 16, we can see a high-level view of the ROS framework. The Master allows the nodes (ROS pieces of software) to find and talk to each other. Thanks to this, we can communicate two nodes whether they are on the same machine, or not.

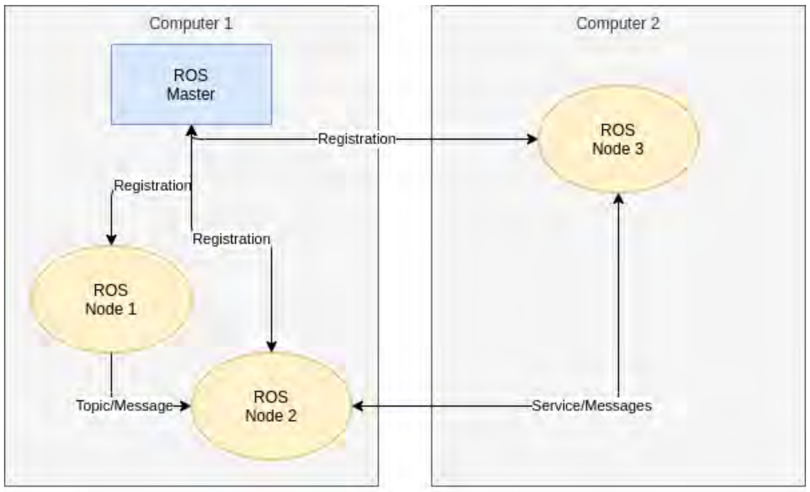


Figure 16: ROS system from a high-level view.

Topics are used to communicate the nodes with each other. For instance, in Figure 16, Node 1 publishes a message on one topic and Node 2 subscribes to the same topic to receive the message form Node 1. When a Node receives petitions that need to be answered, instead of using topics, we use services. Then the information sent is defined by a pair of messages: one for the request and one for the reply. We can see a service example between Node 2 and Node 3 in Figure 16.

Also, as we mainly work with a simulated environment, we use Gazebo and *rviz*, which work with ROS. Gazebo [6] offers the ability to simulate popula-

[1]The *Vision* system, developed in the URL-MONITOR project, has been integrated in the AIDED system to provide the visual input of the state of the world. Although, we are going to briefly describe the module and see how interacts with the rest of modules.

tions of robots accurately and efficiently in complex indoor and outdoor environments. It is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. *rviz* [10], on the other hand, is a 3D visualization tool for ROS that gives us access to the internal state of the system.

In the following sections we describe the different modules used in the AIDED system.

3.1 Supervisor

The *Supervisor* node is in charge of making the rest of the modules work together. We can see it as the brain of the system, as it knows what each node should do and at which moment in order to achieve the goal. For this reason, all the modules communicate with it.

To differentiate petitions, the *Supervisor* has different controllers (see Figure 15). We next describe each one of them:

- *controller_vision*: It makes the petition to the *World Manager* to store the objects identified and their locations. It also modifies the PDDL problem file to give a representation of the actual world.
- *controller_planner*: It requests the execution of the plan to the *Task Planner*, reads the plan, and changes its format to process it better.
- *controller_motion*: It translates the objects received by *Vision* to objects that the *Motion Planner* understands. It also translates the actions given by the *Task Planner* to coordinates of the arm, requesting the information to the *World Manager*. Finally, it makes petitions to the *Motion Planner* node to move the arm to execute the plan.
- *controller_verbal_communication*: It manages the *Verbal Communication* module petitions. To illustrate, if the user asks to set the table, it indicates so to the *Supervisor* module and it starts to do it.

To see it more clearly what the *Supervisor* module does we are going to describe all the actions and processing it must do to perform the task of setting the table.

To start with, we must talk about the *Vision* module. It is constantly sending messages to the *Supervisor* module. This one proceeds and sends these messages in the correct format to the *Motion Planner* and the *World Manager* modules, to plan the robot's movements according to the environment and to answer petitions, respectively. Now, the system and the robot know the state of the world, so they are ready to start setting the table.

The *Verbal Communication* node receives the user's petitions and sends it to the *Supervisor*. If the petition is to set the table, the *Supervisor* looks if all the necessary objects are on the table. If not, it informs the user, through the *Verbal Communication*, about the unnecessary or missing objects. If all the objects are present, the module sends to the *Task Planner* a petition to know the steps the robot must do to achieve the goal and, hence, the *Task Planner* replies indicating the sequence of actions to achieve the goal.

Once the *Supervisor* has the actions, it goes one by one, asking the necessary information to the *World Manager*, to know the coordinates of the object to move and where it must be placed. Once it has the information, it asks the *Motion Planner* module to execute the action. The *Vision* module does not perform any tracking task on the identified objects, thus the *Supervisor* does not listen to *Vision*'s messages, as the *Vision* and the *Motion Planner* modules perceptions of the world may not match. As soon as the *Motion Planner* indicates that the action is finished, the *Supervisor* module verifies the objects' positions, listening again to the *Vision* module. If the current state of the world does not match with the planned one, the *Supervisor* module requests the *Task Planner* to replan. On the contrary, if the objects are placed at the correct position the *Supervisor* proceeds with the next move. This cycle is repeated for all the actions, until the plan is either successfully executed or aborted.

3.2 Task Planning

Automatic planning [51] is a branch of artificial intelligence that consists of performing strategies or sequences of actions, often executed by agents. Unlike classical classification and control problems, solutions in automatic planning are complex and must be discovered and optimized in a space of more than one dimension.

Thus, a planner typically considers three entries:

- A description of the initial state of the world.
- A description of the goal to be achieved.
- A set of possible actions that can be performed. These actions detail some prerequisites for the action to be performed and some post-conditions, which indicate the effect that said action produces.

On the other hand, there are different types of planners. Before that, though, we need to keep in mind that each planner has some characteristics that make it better for some tasks than others. For this reason, we must ask ourselves what the problem we want to address is and, to do so, we must know the properties problems have:

- Deterministic or non-deterministic actions. It is understood as a deterministic that action in which only a possible future can be developed. Therefore, a non-deterministic action can develop more than one future.
- Discrete or continuous state variables. A discrete variable does not support any value. Instead, a continuous variable, does.
- Total or partial observability. The first one has a complete view of the world, while the second one only a portion.
- Number of initial, finite, or many arbitrarily states.
- Duration of the actions. The actions have a duration or not.
- Concurrency of actions. Several actions can be performed at once or just one at a time.
- Reaching the final goal or getting as close as possible.

- Number of agents available. Agents are cooperative or selfish. Agents make their plans separately or plans are built centrally for all agents.

From here, we have different planning models:

- Classic planning. It is determined by:
  - Unique and known initial state.
  - Deterministic actions.
  - Actions do not have a duration.
  - Only one action can be performed at a time.
  - There is only one agent.

  Given these characteristics, observability is irrelevant, as the state of the world can be accurately predicted after any action.

- Discrete-time Markov decision processes (MDP). Its characteristics are:
  - Actions without duration.
  - Non-deterministic actions with probabilities.
  - Total observability.
  - Maximizing a reward function.
  - A single agent.

- Partially observable Markov decision process (POMDP). The total observation of MDP is changed to the partial observation. Thus, a probability distribution must be maintained over the set of possible states, based on the set of observations and their probabilities.

- Multi-agent planning. Related to game theory. This type of planning involves coordinating the resources and actions of multiple agents.

Different implemented planners have been tested for this project. As we will see, we want a basic domain to work with, so the planner could find the solution in a fast way. For this reason, we tested different planners to see which one was the best in the scenario, beginning with a simple one, which is STRIPS, following with PDDL4J and finishing with ROSPlan. All these planners tested have the following properties:

- are deterministic,
- total observability,
- the actions do not have a duration, meaning that whatever action the planner chooses would not last more than another one. To illustrate, stacking an object has the same duration as moving it, so the planner does not have a preference of doing one action or another determined by the time factor,
- only one action can be performed at a time, although we will see that one of the planners detects if more than one action can be done at once,

- the planner must reach the final goal,
- there is only one agent: the robot. The human is not considered as such for the planner. He or she will help when robot cannot do some task because of limitations.

We can also classify the planners depending on their level of strategy for building a plan [29].

Definition 2. Plan: Sequence of operators that solve goal 1; the plan describes all the moves we need to make.

Definition 3. Operator: An operator o is a tuple $o =$ (name(o), precond(o), effect(o)) whose elements are:

- name(o), the *name* of the operator,
- precond(o), the *preconditions* of the operator, that is a set of literals that must be true to apply the operator o,
- effect$(o)^-$ is a set of literals that are false after the application of the operator o,
- effect$(o)^+$ is a set of literals that are true after the application of the operator o.

Definition 4. Action: An action is any ground instance of an operator. If a is an action and s is a state such that precond(a) are true in s, then a is *applicable* to s and the result of applying action a to state s is the state s':

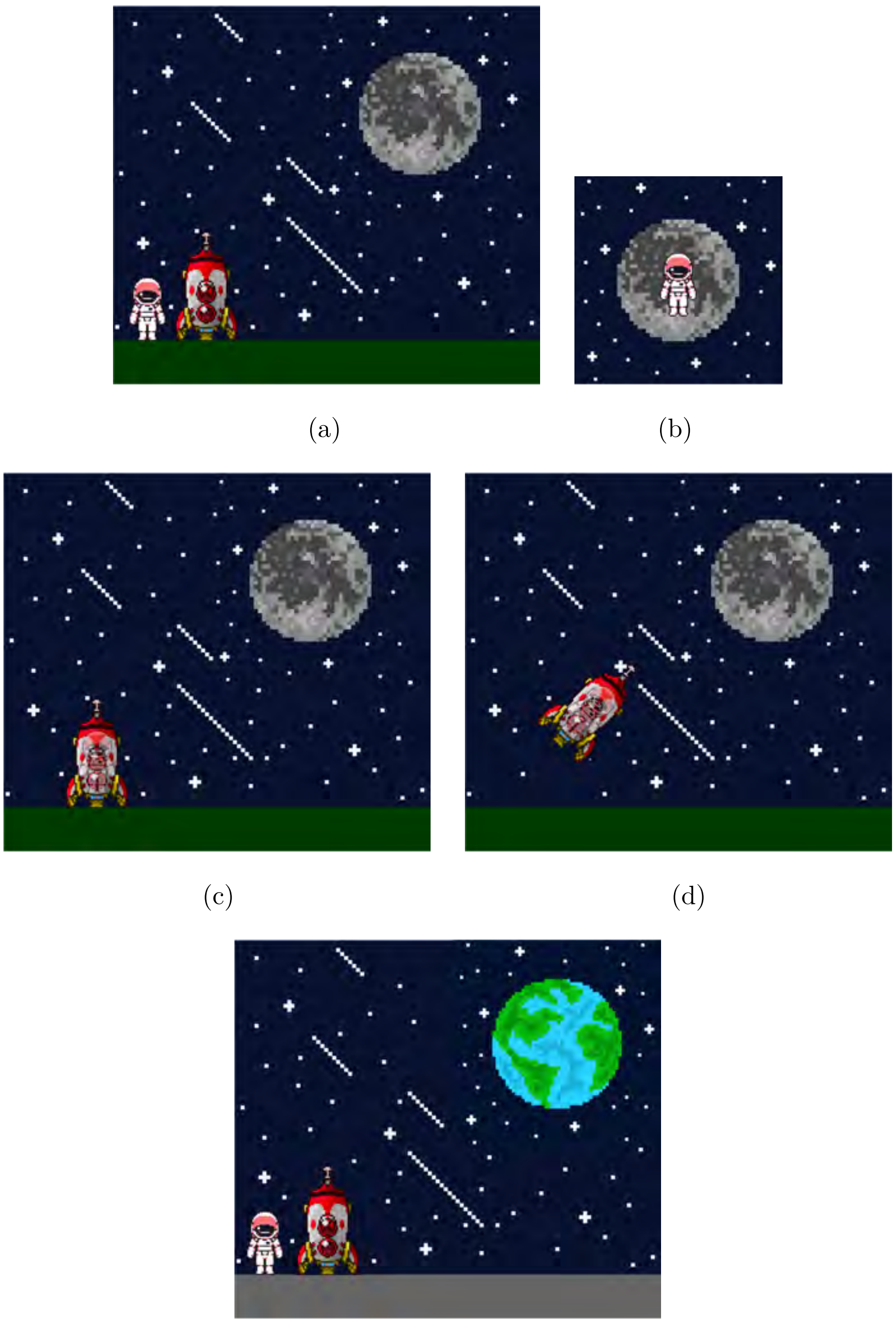
$$s' = \gamma(s, a) = (s - effect^-(a)) \cup effect^+(a)$$

The planners are:

- Linear planner
  - We need initial and final states.
  - We need a set of operators that allow us to change the state.
  - It gets the final state and splits it into different targets (it has a stack of targets):
    - It will take goal 1 and try to reach it.
    - It can be the fact that when achieving one goal, others appear.
    - It will have reached the end when the target stack is empty.

  Imagine the scenario illustrated in Figure 17 (a). In this figure we can see Joan, who wants to go to the moon with a rocket (see Figure 17 (b)). He is currently on earth, and the operators he must apply to achieve the goal are (see Figures 17 (c), (d), and (e)):

```
get_in_the_rocket(Joan)
rocket_from_earth_to_moon()
get_out_the_rocket(Joan)
```



(a) (b)

(c) (d)

(e)

Figure 17: Example of a linear planner application; Joan wants to go to the moon. (a) Initial state. (b) Goal 1: Joan on the moon. (c) Joan goes into the rocket. (d) The rocket goes from the earth to the moon. (e) Joan gets out of the rocket.

However, linear planners have the following limitation. As described before, it gets the first goal and tries to reach it, therefore, it does not consider the other goals. This is what is called the Sussman anomaly [50], first described by Gerald Sussman. We will see the problem with the same example as before but adding another goal; Maria also wants to go to the moon (see Figure 18).

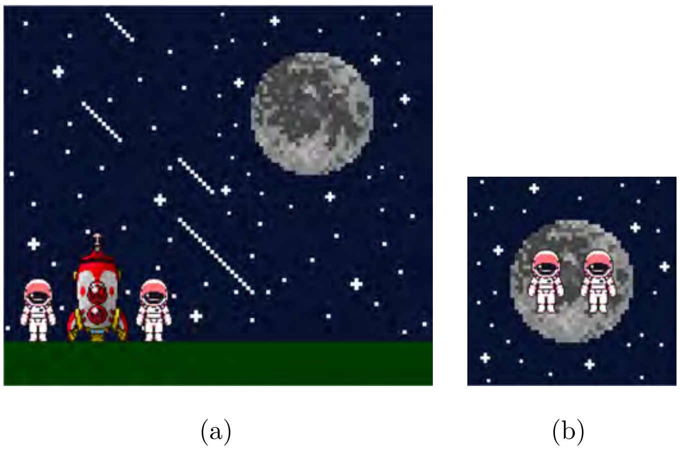


Figure 18: Problem example: Joan and Maria want to go to the moon. (a) Initial state. (b) Goal. Goal 1: Joan on the moon. Goal 2: Maria on the moon.

To do so, the planner will get the first goal: Joan on the moon. To achieve it, will do the same steps as before (see Figures 19 (a), (b), and (c)):

```
get_in_the_rocket(Joan)
rocket_from_earth_to_moon()
get_out_the_rocket(Joan)
```

Now, the planner gets the second goal: Maria on the moon. As we can see, there is a problem, since the rocket is not on earth; it is on the moon. For this reason, the second goal cannot be achieved (see Figure 19 (c), where Maria is on Earth).

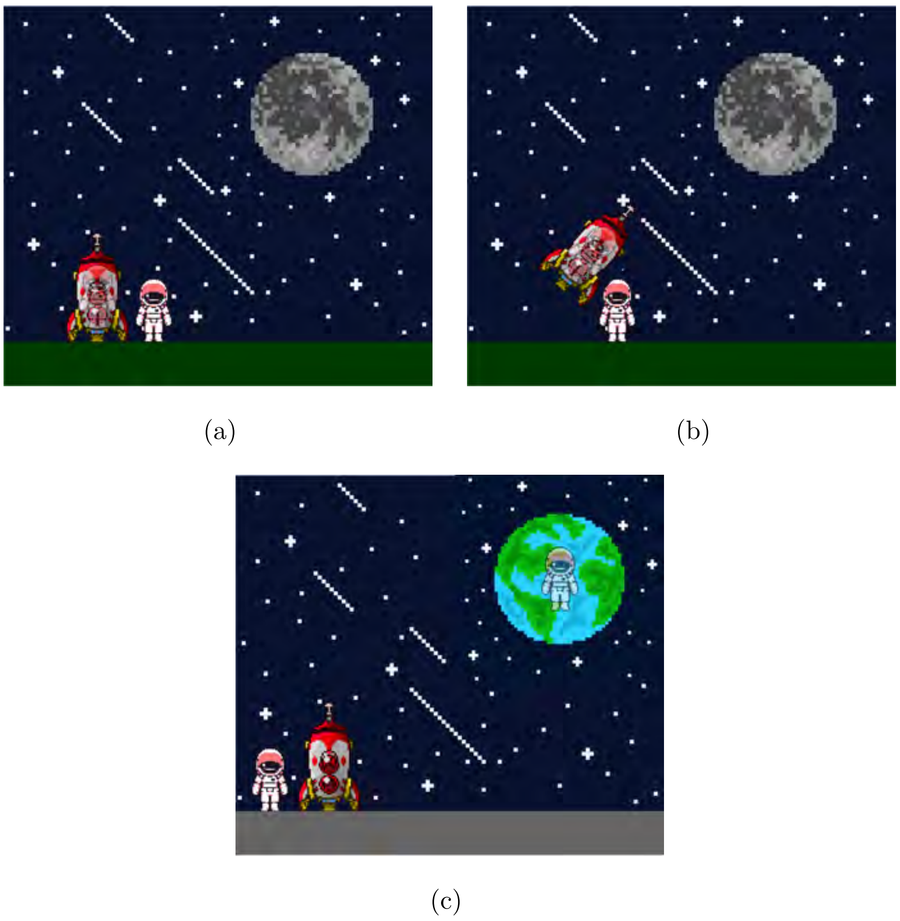


Figure 19: Example of Sussman anomaly in a linear planner; Joan and Maria want to go to the moon. (a) Joan gets in the rocket. (b) The rocket goes from the earth to the moon. (c) Joan gets out of the rocket, but Maria is on the earth and cannot travel to the moon because the rocket is on the moon.

- Non-linear planner
  - We need initial and final states.
  - We need a set of operators that allow us to change the state. These can interfere in a solution for an older problem.
  - A plan that simultaneously works with multiples subproblems is needed.
  - The plan is partially ordered and instantiated.

Let us go back to the example seen before; Joan and Maria want to go to the moon (see Figure 18). To achieve it, will do the following steps (see Figures 20 (a), (b), (c), (d), and (e)):

```
get_in_the_rocket(Joan)
get_in_the_rocket(Maria)
rocket_from_earth_to_moon()
```

```
get_out_the_rocket(Joan)
get_out_the_rocket(Maria)
```

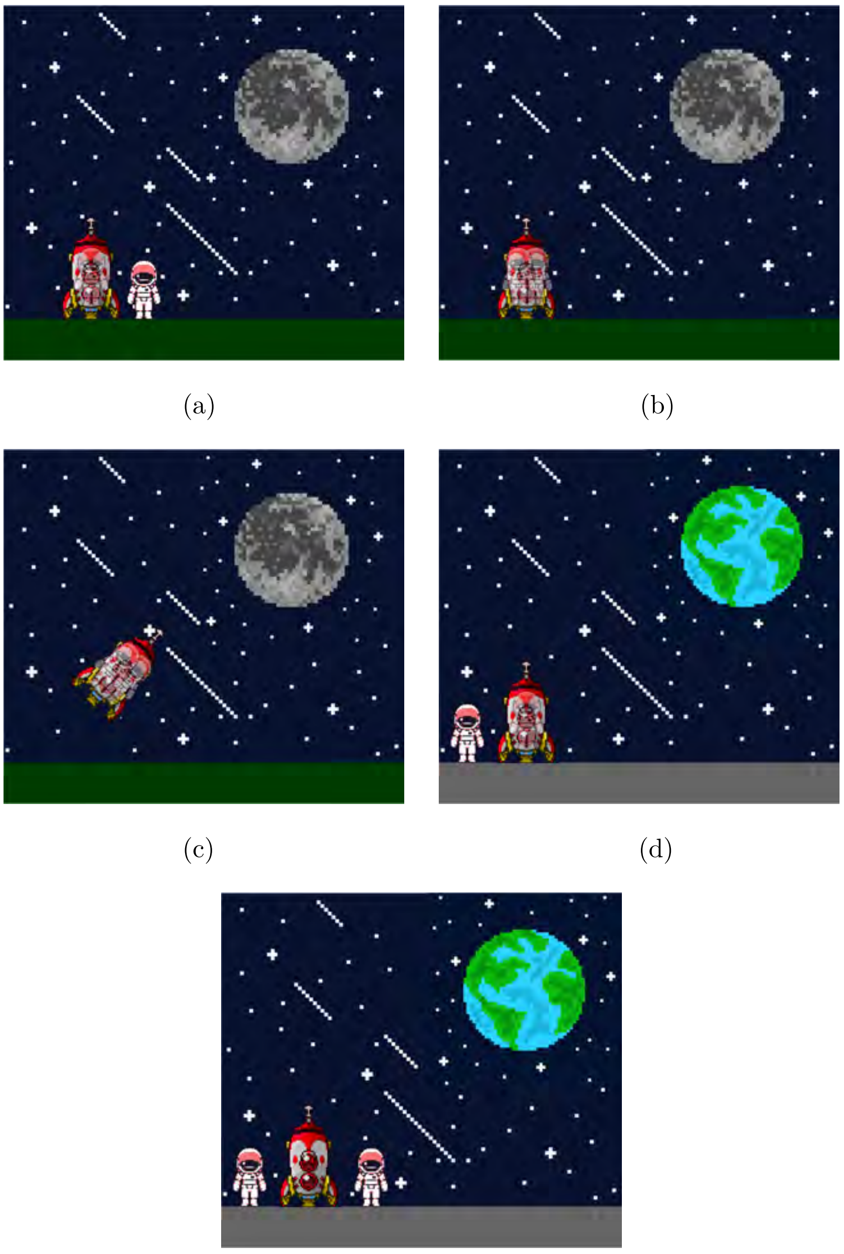

(a) (b) (c) (d) (e)

Figure 20: Example of a non-linear planner application; Joan and Maria want to go to the moon. (a) Joan goes into the rocket. (b) Maria goes into the rocket. (c) The rocket goes from the earth to the moon. (d) Joan gets out of the rocket. (e) Maria gets out of the rocket.

In this case, some operators can be done simultaneously, as represented in Figure 21.

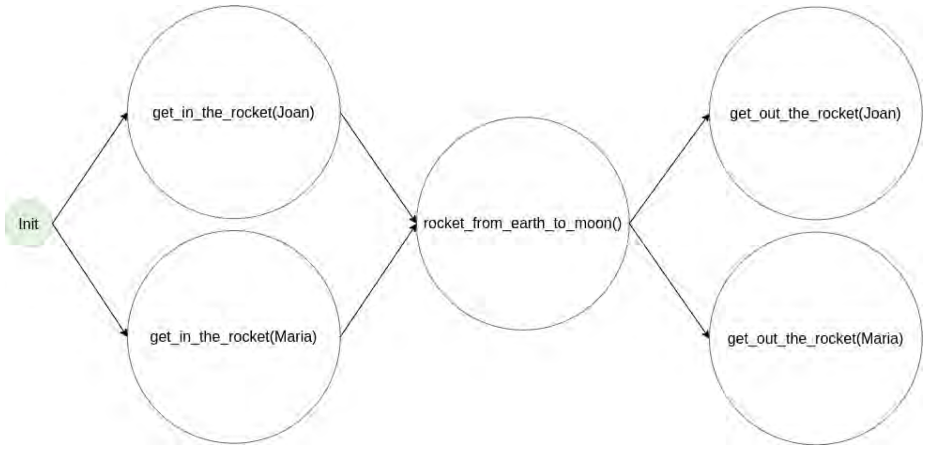


Figure 21: Sequence of actions in a non-linear planner with the example problem: Joan and Maria want to go to the moon.

- Hierarchical planning
  - We need initial and final states.
  - We need a set of operators that allow us to change the state.
  - It erases some details of the problem until the principal difficulties are not solved. There are two ways to do so:
    * Using macrooperators: From smaller operators, bigger ones are built.
    * Planning is done with a hierarchy of abstraction spaces. In each of which are ignored the preconditions of a lower level of abstraction. It is what purposes ABSTRIPS. To use this idea, it uses "first search in length":
      1. It solves the problem completely, considering only those preconditions which critical value is the highest as possible.
      2. It goes down level. In each of these levels uses the plans build in the previous levels as drafts of a complete plan.

  This type of planning has been used in a flexible job shop [35] and for the sugar cane harvest in Colombia [48].

STRIPS and PDDL4J are examples of linear planners, whereas ROSPlan is a non-linear planner. Although in our scenario we will not find the Sussman anomaly, as the plans do not interfere between them (in the final goal, the objects are not stack or placed in the same zone), we can use both planners. The main aspect, as we will see, is that testing a non-linear planner can lead to find other applications of our system. We will get more in deep in these planners in the following sections.

3.2.1 STRIPS

Stanford Research Institute Problem Solver (STRIPS) [21] is an automated planner developed by Richard Fikes and Nils Nilsson in 1971 at SRI International. In this planner, the first thing to do is describe the world, by providing objects, actions, preconditions, and effects. Once the world is described, a problem is set. A problem consists of an initial state and a goal condition. STRIPS can then search all possible states, starting from the initial one, executing various actions, until it reaches the goal. Nowadays, a common language for describing the domain and the problem is Planning Domain Definition Language (PDDL).

Once STRIPS has a domain and a problem, a graph can be constructed with all the available states and actions. It is called the planning graph. To solve it, a variety of algorithms can be used. For STRIPS, a common method is to use Bread-First-Search (BFS), Depth-First-Search (DFS) and A* search.

STRIPS is a linear planner, hence many problems can be solved with it, such as stacking blocks, Rubik's cube, cooking, and so on.

We started our evaluation with a STRIPS planner implemented in Python [46]. It uses DFS to solve the given problem. This implementation was discarded for our use because it is not able to solve the Hanoi tower. Therefore, we had to look for another one.

The second STRIPS planner we putted under test was implemented in JavaScript [22]. The user can choose to use BFS, DFS and A*. After some tests done, the best algorithm is the A*; with it the Hanoi tower can be optimally solved and in a fastest way (see Table 1).

Table 1: Solution comparison of the Hanoi Tower in different algorithms using [22].

Algorithm	Number of visited nodes	Steps to achieve the goal
BFS	134	7
DFS	22	15
A*	91	7

This last planner implementation was used to solve a simple implementation of the problem that would be addressed later. Thus, the initial goal was to stack a glass on top of a cell phone. Once everything together worked, the difficulty we encountered in dealing with the real problem was that this planner did not find the solution with many objects ($n > 7$). In our application domain the minimum number of objects is 13. Hence, we had to discard this planner as well.

3.2.2 PDDL4J

PDDL4J [36] is a library which purpose is facilitate the development of JAVA tools for Automated Planning based on PDDL language. PDDL was developed by Drew McDermott and the 1998 planning competition committee. It was created to encourage the empirical comparison of planning systems and the exchange of planning benchmarks within the community. This language has become a standard for describing planning domains.

PDDL4J library contains:

- A PDDL 3.1 parser and all the classes needed to manipulate the problem representation.

  The problem representation describes the initial state of the world, the objects under consideration and their properties, a goal to achieve and actions that can be performed on these objects to achieve the goal. Actions have triggering preconditions and resulting postconditions, which are the actions' effects. The preconditions define under which circumstances (states of the world) actions can be performed.

  The parser can be configured to accept only specified requirements of PDDL language.

- A set of useful pre-processing mechanisms to instantiate and simplify the operators into actions.

  To transform the operators of the PDDL language (in the domain file) into the actions used by the planning algorithm, an instantiation process is triggered.

- A set of already implemented classical heuristics.

- Several examples of planners using PDDL4J.

The planner used to test this library is based in A* search strategy and it is called Heuristic Search Planner (HSP). Also, we express the planning problem in two files:

1. the *domain* file describes the operators,

2. the *problem* file describes the initial state and the goal.

Our domain file, like the Hanoi Tower, is based in the BLOCKS world, which is similar to a set of wooden blocks of various shapes and colors sitting on a table. Only one block can be moved at a time, placing on a table, or putting it on top of another block. The goal is to build one or more vertical stacks of blocks. A domain example can be observerd in Section 8.1.1. Therefore, our objects are named as *blocks* and our zones of the table are named as *tables.* It has four operators:

1. *move (b, t1, t2)*: It moves a block *b*, that is on table *t1*, to table *t2* (see Figure 22).

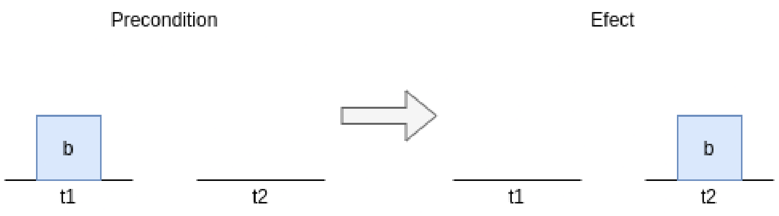


Figure 22: *move* operation in *domain* file representation.

2. *move2 (c, a, b)*: It moves a block *c*, that is on block *a*, on top of block *b* (see Figure 23). The blocks *a* and *b* can be on a table or on another block.

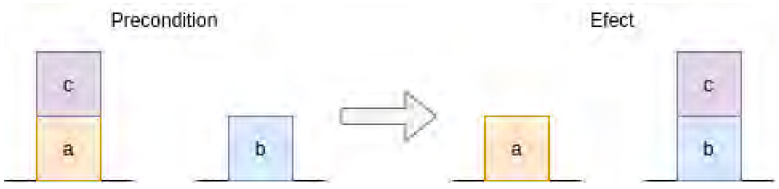


Figure 23: *move2* operation in *domain* file representation.

3. *stack (a, t1, b)*: It stacks a block *a*, that is on table *t1*, on block *b* (see Figure 24). The block *b* can be on a table or on another block.

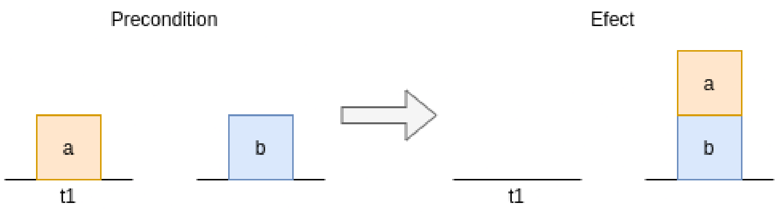


Figure 24: *stack* operation in *domain* file representation.

4. *unstack (a, b, t1)*: It unstacks a block *a*, that is on block *b*, to table *t1* (see Figure 25). The block *b* can be on a table or on another block.

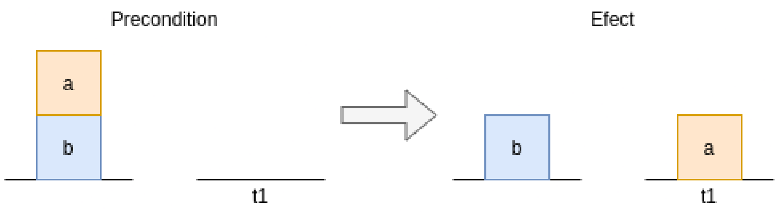
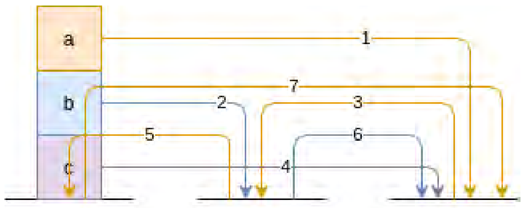
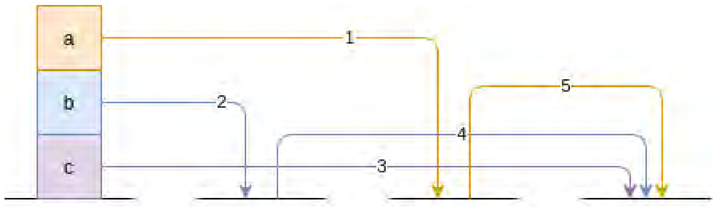


Figure 25: *unstack* operation in *domain* file representation.

Our *problem* file, also, includes these *blocks* and *tables*, that represents different kind of objects and zones, respectively. We have added some locations that are transitory; only used to temporally place objects. This extra space is to achieve the solution in less steps. To illustrate, the Hanoi Tower, with three tables, can be solved with 7 steps, seen in Table 1. If we add another table, it can be solved with 5 (see Figure 26). Thanks to this space, there are less *stacks* which, in our scenario, will also need an *unstack* to achieve the goal.



(a)



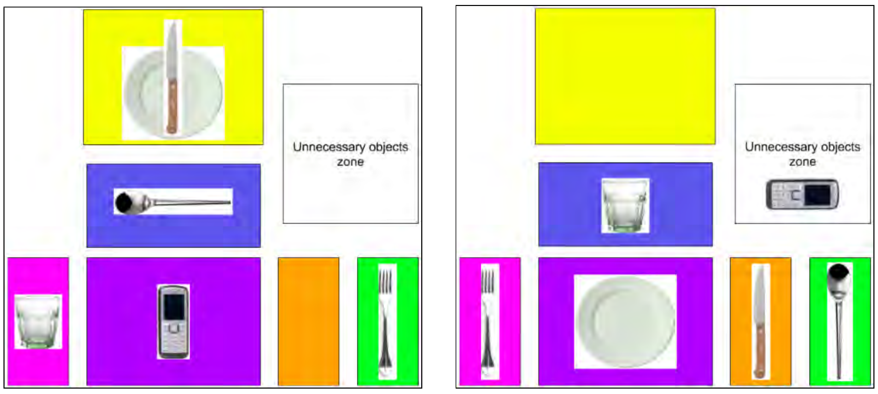
(b)

Figure 26: Solutions of the Hanoi Tower changing the number of tables. (a) Solution when there are three tables. (b) Solution when there are four tables

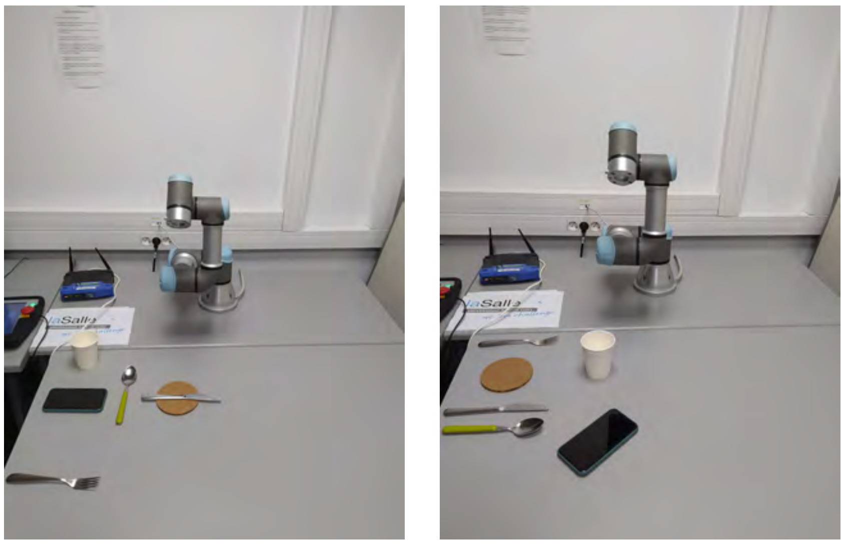
Also, there is a special one to place objects that do not belong to the setting, which is named *unnecessary objects* zone. We have implemented two scenarios: the *table for one* and the *table for two*.

We are now going to test the *table for one* (see file in Section 8.1.2), that has 13 objects (6 objects and 7 locations or zones). The initial state is the one shown in Figure 27 (a). As we can see, it has objects that are not needed for the task (the cell phone) and some objects are stacked (the knife is on top of the plate). The *yellow* zone is a place where no objects will be placed at the end, it is just transitory for the objects in case it is needed a zone with no objects to put another one. We can also see the *unnecessary objects* zone, explained before.

The final state of the table is the one shown in Figure 27 (b). We can appreciate that the *yellow* zone is completely empty and that the *unnecessary objects* zone has the cell phone, since this object is not part of the table setting.



(a) (b)



(a) (b)

Figure 27: Table for one. (a) Initial state. (b) Final state.

The files used can be found in Section 8. The computed plan is the following:

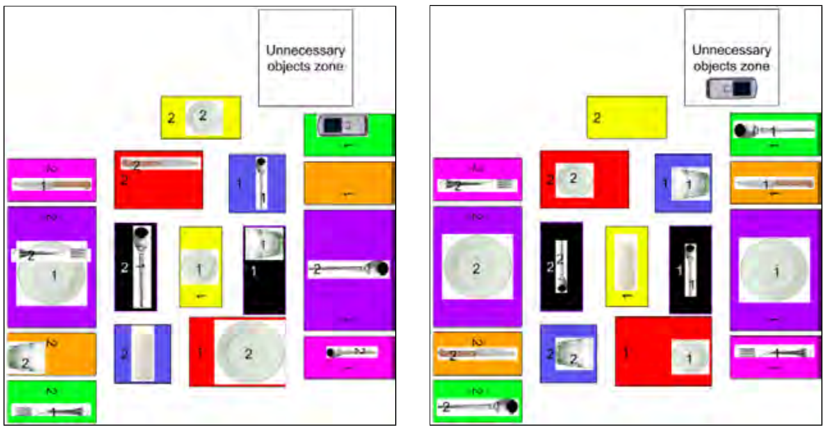
1. unstack *knife*1, which is on *plate*1, and put it on *orange* zone
2. move *cellphone*1, which is on *purple* zone, to *unnecessary_objects* zone
3. move *plate*1, which is on *yellow* zone, to *purple* zone
4. move *spoon*1, which is on *blue* zone, to *yellow* zone
5. move *glass*1, which is on *pink* zone, to *blue* zone
6. move *fork*1, which is on *green* zone, to *pink* zone

7. move *spoon*1, which is on *yellow* zone, to *green* zone

We can appreciate that the *yellow* zone is used once; to place the spoon to proceed.

The time spent to find the solution is of 0.08 seconds for parsing, 0.10 seconds for encoding and 0.13 seconds searching the solution. In conclusion, it founds the solution in 0.31 seconds.

Another scenario used is a romantic *table for two* (see file in Section 8.1.3), with 33 objects. The initial state is the one shown in Figure 28 (a). As we can see, we added a bread plate, a teaspoon, and a candle. Also, the numbers indicate the names of the zones and objects. For instance, since there are two *yellow* zones, one is *yellow1* and the other is *yellow2*, to differentiate them. The same happens with the objects.



(a) (b)

Figure 28: Table for two. (a) Initial state. (b) Final state.

The goal to achieve is the one shown in Figure 28 (b). As we can appreciate, all the objects numbered with a one, are on the right coinciding with the zones number, whereas the objects with the two are on the left, coinciding again with the zones number. The *yellow*2 zone is for transitory objects.

The solution given is the following, where with 19 movements the goal is achieved:

1. move *cellphone*1, which is on *green*1 zone, to *unnecessary_objects* zone
2. stack *plate*2, which is on *red*1 zone, to *cellphone*1
3. move *breadplate*1, which is on *yellow*1 zone, to *red*1 zone
4. move *candle*1, which is on *blue*2 zone, to *yellow*1 zone
5. move *knife*1, which is on *pink*2 zone, to *orange*1 zone
6. unstack *fork*2, which is on *plate*1, and put it on *pink*2 zone

7. move *glass*2, which is on *orange*2 zone, to *blue*2 zone
8. move *knife*2, which is on *red*2 zone, to *orange*2 zone
9. move *breadplate*2, which is on *yellow*2 zone, to *red*2 zone
10. stack *plate*1, which is on *purple*2 zone, to *knive*1
11. unstack *plate*2, which is on *cellphone*1, and put it on *purple*2 zone
12. move *spoon*1, which is on *black*2 zone, to *green*1 zone
13. move *minispoon*2, which is on *pink*1 zone, to *black*2 zone
14. move *fork*1, which is on *green*2 zone, to *pink*1 zone
15. move *spoon*2, which is on *purple*1 zone, to *green*2 zone
16. stack *minispoon*1, which is on *blue*1 zone, to *breadplate*1
17. move *glass*1, which is on *black*1 zone, to *blue*1 zone
18. unstack *plate*1, which is on *knive*1, and put it on *purple*1 zone
19. unstack *minispoon*1, which is on *breadplate*1, and put it on *black*1 zone

In this case, we can appreciate that the *yellow*2 zone is not used for transitory objects. It stacks some objects otherwise, such as the second action; stacks the *plate*2 on *cellphone*1.

The time spent to find the solution is of 0.08 seconds for parsing, 0.68 seconds for encoding and 26.18 seconds searching the solution. In conclusion, it founds the solution in 26.94 seconds. We can see that it took more time to find the solution compared to the scenario of the *table for one*. We must appreciate that in this scenario there are 20 more objects, which means that the decision tree is bigger. We can see a comparison of the results in Table 2.

Table 2: Comparison between scenarios with PDDL4J.

Scenario	Number of elements (u)	Parsing (s)	Encoding (s)	Searching the solution (s)	Total time (s)
Table for one	13	0.08	0.10	0.13	0.31
Table for two	33	0.08	0.68	26.18	26.94

3.2.3 ROSPlan

Although we have had good results with PDDL4J, we also want to put under test ROSPlan, as it is a very used planner in ROS. As said, PDDL4J is a linear planner, which can have some problems when trying to achieve some goals; therefore, we want to compare with ROSPlan, as it is a non-linear planner. Furthermore, the AIDED system is implemented in ROS, thus using ROSPlan is straightforward allowing the use of other planners in the future if needed.

ROSPlan [23] provides tools for AI Planning in a ROS system. It has nodes which encapsulate planning, problem generation, and plan execution (see Figure 29).

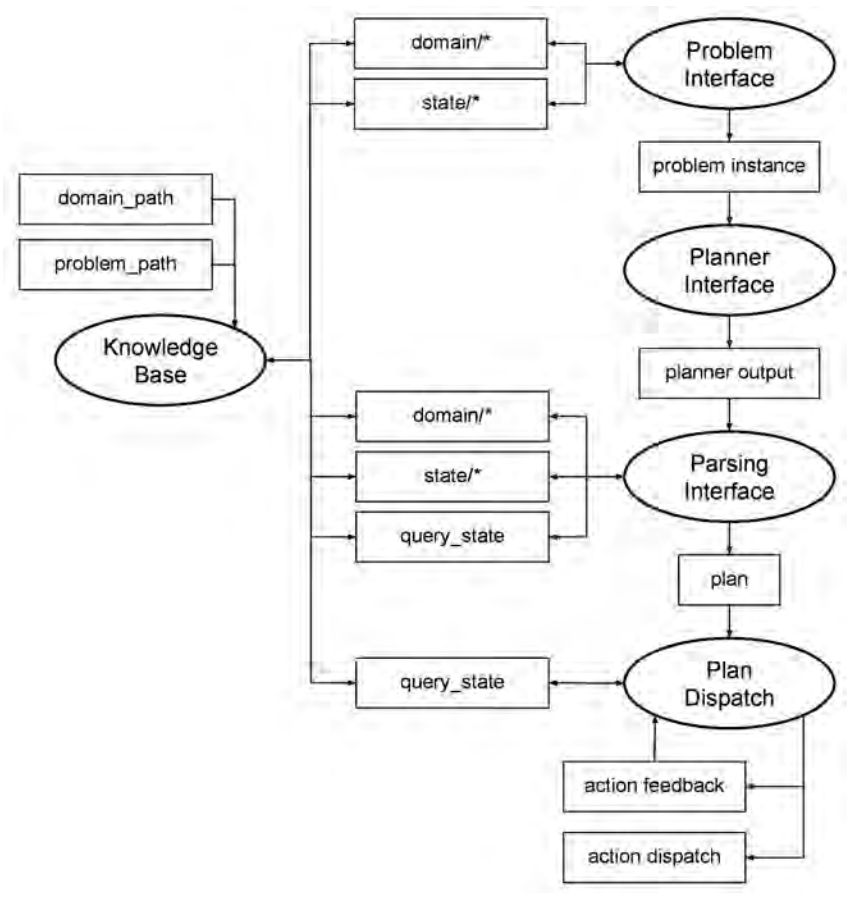


Figure 29: ROSPlan demo system.

- The Knowledge Base stores a PDDL model (see Figure 30). It loads a PDDL domain and, an optional problem file. It stores the state as a PDDL instance and updates it with ROS messages. Furthermore, it can be queried for information on the PDDL domain model, and problem instance.

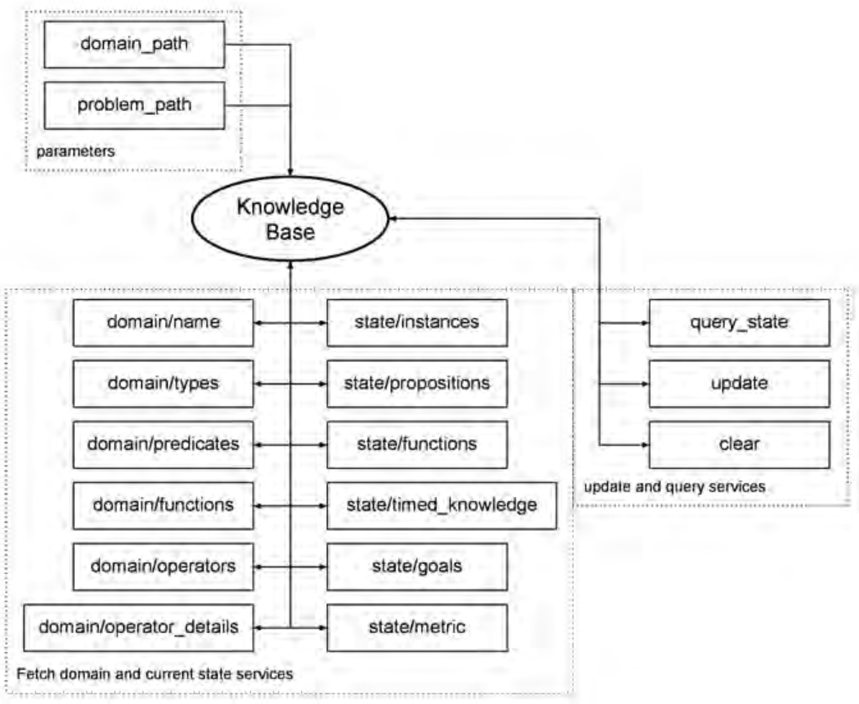


Figure 30: ROSPlan: Knowledge Base.

- The Problem Interface (see Figure 31) generates an instance of the problem. It obtains the details from the domain and the current state through doing petitions to Knowledge Base node and publishes an instance of the PDDL problem as a String (to either publish it through a topic or writing it to a file).

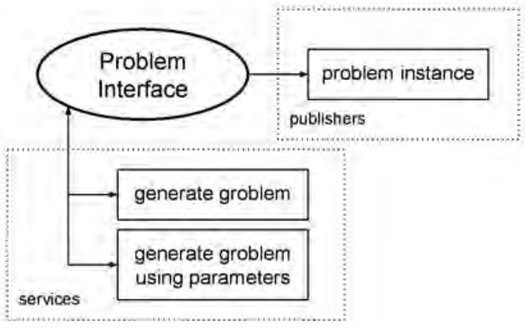


Figure 31: ROSPlan: Problem Interface.

- The Planner Interface (see Figure 32) calls a planner providing a domain file and an instance of the problem and publishes the plan (either through a topic or writing it on a file).

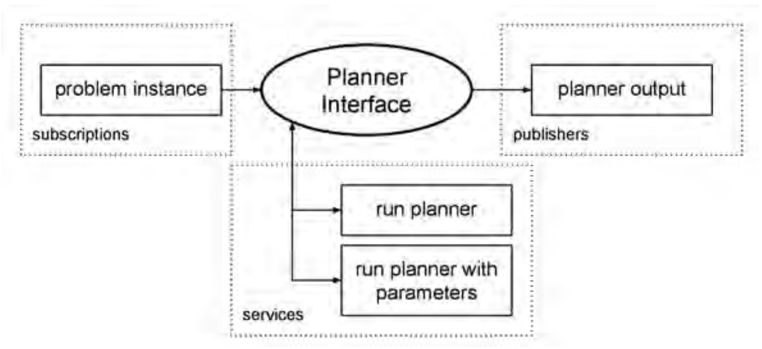


Figure 32: ROSPlan: Planner Interface.

In addition, there are implementations of the Planner Interface node used with the different planners summarized in Table 3. In our case we are using the popf_planner_interface, as it is a temporal planner: during search, when applying an action to a state, it seeks to introduce only the ordering constraints needed to resolve threats, rather than insisting the new action occurs after all of those already in the plan. Thanks to this, we can know if to actions can be done at the same time. It also uses PDDL to define the domain and the problem.

Table 3: Implementations of the Planner Interface node.

Node Type	Compatible Planners	Website
popf_planner_interface	POPF	nms.kcl.ac.uk/planning/software/popf
	OPTIC	nms.kcl.ac.uk/planning/software/optic
ff_planner_interface	FF, Metric-FF, Contingent-FF	fai.cs.uni-saarland.de/hoffmann/ff
lpg_planner_interface	LPG	lpg.unibs.it/lpg
tfd_planner_interface	TFD	gki.informatik.uni-freiburg.de/tools/tfd
smt_planner_interface	SMTPlan	kcl-planning.github.io/SMTPlan

- The Parsing Interface (see Figure 33) converts a PDDL plan into ROS messages, ready to be executed.

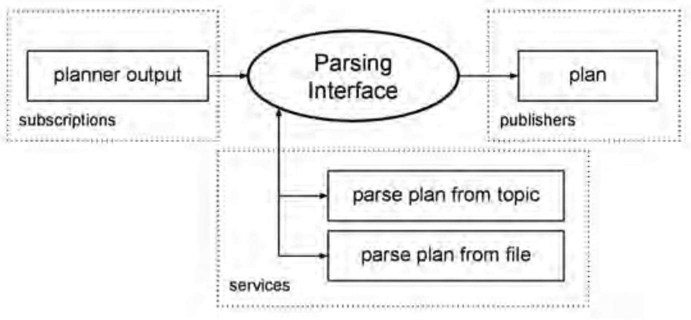


Figure 33: ROSPlan: Parsing Interface.

- The Plan Dispatch (see Figure 34) encapsulates plan execution. It includes plan execution, and the process of connecting single actions to the processes which are responsible for their execution.

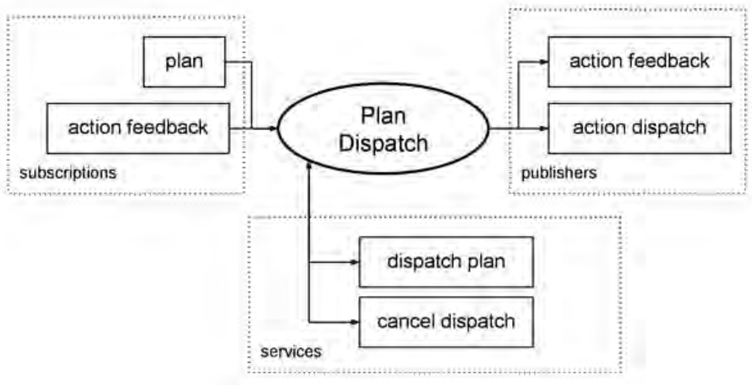


Figure 34: ROSPlan: Plan Dispatch.

In this project we have used the Knowledge Base, the Problem Interface, and the Planner Interface, which output is the sequence of actions that the agents (robot and human) will have to perform. Every time the user asks to set the table, we launch a file, that executes these three nodes, to obtain the solution.

To compare the solution given with the PDDL4J (see Section 3.2.2), we used the same scenarios shown in Figure 27, for the *table for one* and Figure 28, for the *table for two*. We used the popf_planner_interface in both situations.

In the *table for one* scenario, we get the following result, where the solution is found in 9 steps:

1. stack *spoon*1, which is on *blue* zone, with *cellphone*1
2. unstack *knife*1, which is on *plate*1, and put it on *orange* zone
3. stack *fork*1, which is on *green* zone, with *knife*1

4. move *glass*1, which is on *pink* zone, to *blue* zone
5. unstack *spoon*1, which is on *cellphone*1, and put it on *green* zone
6. unstack *fork*1, which is on *knife*1, and put it on *pink* zone
7. stack *plate*1, which is on *yellow* zone, and put it on *fork*1
8. move *cellphone*1, which is on *purple* zone, to *unnecessary_objects* zone
9. unstack *plate*1, which is on *fork*1, and put it on *purple* zone

Whereas in the *table for two*, we get the solution with 25 steps:

1. move *cellphone*1, which is on *green*1 zone, to *unnecessary_objects* zone
2. move *knife*1, which is on *pink*2 zone, to *orange*1 zone
3. stack *glass*1, which is on *black*1 zone, to *minispoon*2
4. stack *candle*1, which is on *blue*2 zone, to *breadplate*1
5. stack *minispoon*1, which is on *blue*1 zone, to *cellphone*1
6. unstack *fork*2, which is on *plate*1, and put it on *pink*2 zone
7. move *glass*2, which is on *orange*2 zone, to *blue*2 zone
8. move2 *candle*1, which is on *breadplate*1, to *breadplate*2
9. unstack *glass*1, which is on *minispoon*2, and put it on *blue*1 zone
10. move *spoon*1, which is on *black*2 zone, to *green*1 zone
11. move *knife*2, which is on *red*2 zone, to *orange*2 zone
12. stack *breadplate*1, which is on *yellow*1 zone, to *minispoon*1
13. move *plate*1, which is on *purple*2 zone, to *black*1 zone
14. stack *spoon*2, which is on *purple*1 zone, to *glass*1
15. move *minispoon*2, which is on *pink*1 zone, to *black*2 zone
16. unstack *candle*1, which is on *breadplate*2, and put it on *yellow*1 zone
17. stack *fork*1, which is on *green*2 zone, to *knife*2
18. move *breadplate*2, which is on *yellow*2 zone, to *red*2 zone
19. move2 *breadplate*1, which is on *minispoon*1, to *candle*1
20. unstack *spoon*2, which is on *glass*1, and put it on *green*2 zone
21. unstack *fork*1, which is on *knife*1, and put it on *pink*1 zone

22. move *plate*1, which is on *black*1 zone, to *purple*1 zone
23. move *plate*2, which is on *red*1 zone, to *purple*2 zone
24. unstack *minispoon*1, which is on *cellphone*1, and put it on *black*1 zone
25. unstack *breadplate*1, which is on *candle*1, and put it on *red*1 zone

If we compare these solutions with the ones given by PDDL4J, in Section 3.2.2, we can see that in the *table for one* scenario PDDL4J finds the solution with 7 movements, and ROSPlan with 9, i.e. ROSPlan requires two additional actions to achieve the goal.

On the other hand, comparing the solutions of the *table for two* scenario, PDDL4J finds the solution with 19 steps, whereas ROSPlan with 25. Again, ROSPlan needs more moves, 6 to be concrete.

Appreciate that PDDL4J uses more the *move* operator, whereas ROSPlan uses *stack*. When using *stack*, the planner will need to do also an *unstack*, as we do not have stacked objects in the end goal. Hence, ROSPlan has more moves as needs two actions (*stack* and *unstack*) to achieve the end position of one object.

Despite of ROSPlan being less effective finding the solution than PDDL4J, we choose ROSPlan because it indicates the time at which each action is taken and, thanks to it, we can know if various actions can be done at the same time. This fact will be useful when more than one agent collaborates in the same scenario, being robots or users. Furthermore, it is easy to change the planner used (as said, we are currently using POPF) and it uses ROS, the framework we are already using. These are advantages if we want to use our system in other domains in the future, different from the setting the table one used in this work.

3.3 Motion Planning

The *Motion Planner* module is in charge of controlling the robotic arm used in this project.

We have used two robots at different stages of the project, each with two different controllers, as we describe later. The first robot is a UR3 (Figure 35 (a)) is the smallest Universal Robots' arm [38]. It can lift up to 3Kg, and weights 11Kg. This robot arm is the one used for the first tests done. We use it physically and in simulation.

The second robot used is the PANDA Robotic Arm (Figure 35 (b)), a collaborative robot arm developed by FRANKA EMIKA [25]. It can also lift up to 3Kg, and weights 18Kg. This robot arm is used in later tests of the project. We use it in simulation.

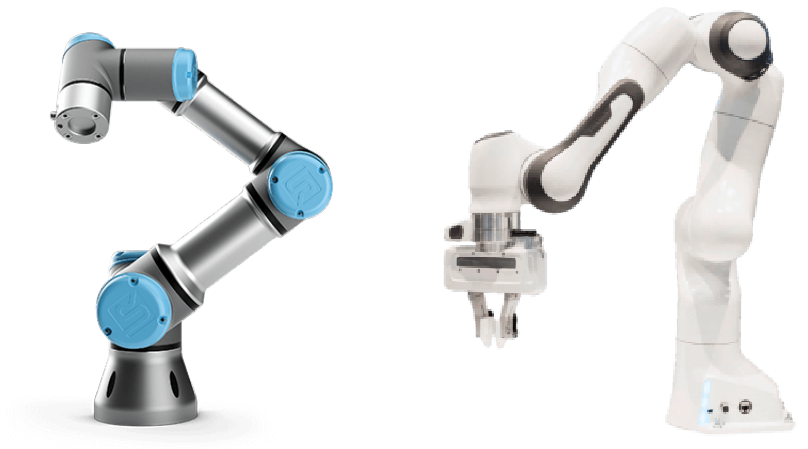

(a) (b)

Figure 35: Arm Robots. (a) UR3 arm. (b) PANDA Robotic Arm.

3.3.1 UR remote control

To control the UR3 robot, we used the UR remote control [42], provided by Universal Robots. 1the communication with the UR3 arm is done via socket, as it has multiple open ports to send commands. In a first place, there is the 29999 which enables to control the interface [43]. We use it, mainly, to control the execution of pre-existing programs, such as loading a program, playing it, pausing it, and stopping it. Secondly, we use the port 30002 [44], which is dedicated to send URScript commands. With this port we send functions to the robot and it executes them. We created the following functions:

- movej_pose: We ask the UR3 to move to a target pose, with a given joint acceleration of leading axis, joint speed of leading axis, time, and blend radius. A pose consists of a x, y, z position and rx, ry, rz rotation.
- movej_rad: We ask the UR3 to move giving the same parameters as in the movej_pose command, but changing the target pose with the radians angle of each joint.
- freedrive_mode: We indicate the UR3 that we want to move it manually.
- end_freedrive_mode: We indicate the UR3 that we have finished moving it manually.
- open_gripper: It opens the gripper attached to UR3.
- close_gripper: It closes the gripper attached to UR3.
- send_program_from_doc: Given a URScript in a file, this function sends it to the UR3.

Finally, we use the Real-Time Data Exchange (RTDE) protocol [45] to know internal information of the UR3 arm. To get this information, we have enabled one port, the 30100, which does not have any specific purpose. We basically use it to know if the robot is moving, to send to it the next pose.

The disadvantage of using this type of control with the robot arm is that when we specify a final position to the robot, it moves using the shortest path and avoiding internal collisions with its own body, but without taking into account potential collisions in the world, such as objects in the environment. That means that it can accidentally move other objects that are within its path. In Figure 36 we illustrate two possible scenarios where the robot has initial and final poses, and an object is between them. There are two possible solutions for this problem; the first one is the one illustrated in Figure 36 (a), where the arm passes above the object. The second solution is illustrated in Figure 36 (b), where the arm passes next to the object.

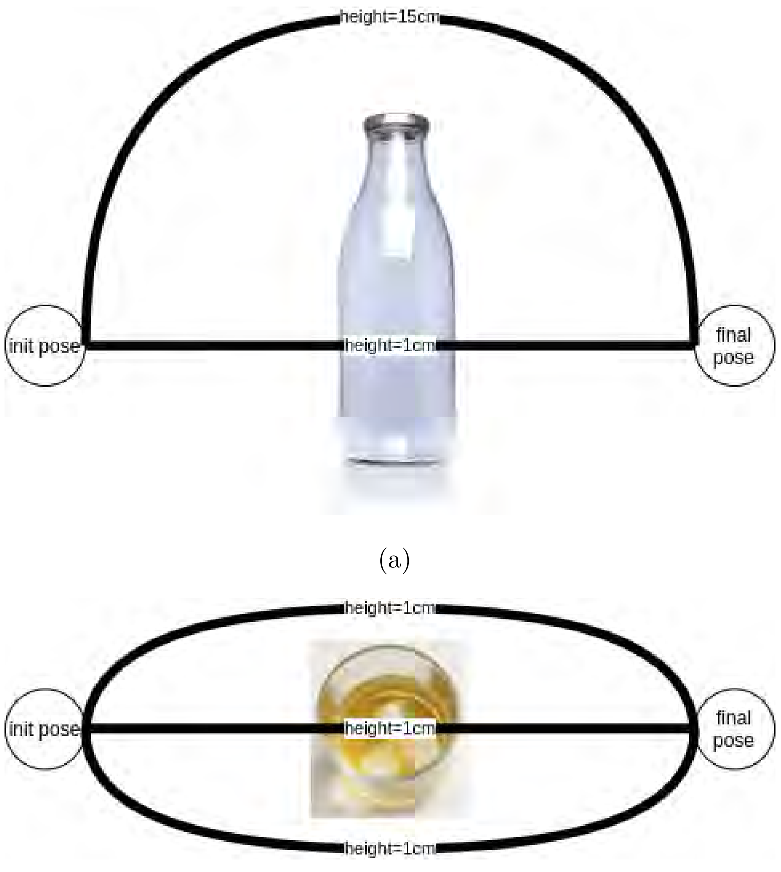


Figure 36: Examples of possible collisions and solutions. (a) Changing the z coordinate. (b) Changing the x coordinate.

Whatever the option we choose, we must appreciate that the we need to know the exact position of the entire robot and the objects to make sure not only that the end effector does not touch the middle objects, but also the rest of the arm (see Figure 37). We thus need of a motion planner tool to plan the robot's movements taking into account not only internal collisions of the robot, but also with the environment.

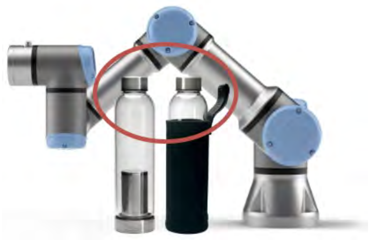


Figure 37: Example of collision between the UR3 arm and some bottles.

3.3.2 MoveIt (ROS)

MoveIt [41] is an open source robotics manipulation platform that works with ROS. It provides tools for motion planning, manipulation, inverse kinematics, control, 3D perception and collision checking. Its main node is *move_group* (see Figure 38), that serves as an integrator. It pulls all the individual components together to provide a set of ROS actions and services for users to use through *C++*, *Python*, or a GUI.

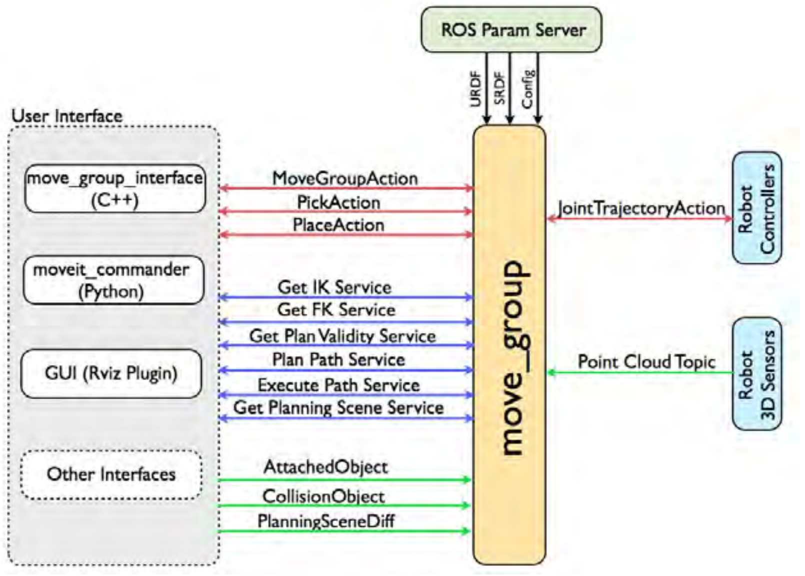


Figure 38: High-level system architecture for the primary node provided by MoveIt called move_group.

move_group uses the ROS param server to get the following information:

- URDF, which is an XML format to represent a robot model.

- SRDF, which is a representation of semantic information about robots.

- MoveIt configuration, which is a specific configuration of MoveIt that includes joint limits, kinematics, motion planning, perception, among others.

Finally, *move_group* talks to the robot through ROS topics and actions. The robot gives to this node its current state information (such as the positions of the joints, the operation mode, and so on), point clouds and other sensor data from the robot's sensors. The *move_group* also talks to the controllers on the robot.

MoveIt integrates a Motion Planning Plugin which allows to communicate with and use different motion planners from multiple libraries. Below is a list of planners that have been used with MoveIt, in descending order of popularity/support within MoveIt:

- Open Motion Planning Library (OMPL) is the default planner of MoveIt. It primarily implements randomized motion planners, that are abstract. For instance, OMPL has no concept of a robot. Instead, MoveIt configures OMPL and provides the back-end for it to work with problems in robotics. It is fully supported by MoveIt. For this reason, is the motion planner that we use in our system.

- Stochastic Trajectory Optimization for Motion Planning (STOMP) can plan smooth trajectories for a robot arm, avoiding obstacles, and optimizing constraints. The algorithm does not require gradients; hence it can optimize arbitrary terms in the cost function like motor efforts. It is partially supported by MoveIt.

- Search-Based Planning Library (SBPL) is a generic set of motion planners that use search-based planning that discretize the space. It is currently being integrated in the latest MoveIt version.

- Covariant Hamiltonian Optimization for Motion Planning (CHOMP) capitalizes on covariant gradient and functional gradient approaches to the optimization stage to design a motion planning algorithm based entirely on trajectory optimization. It makes many motion planning problems simple and trainable. It is currently being integrated in the latest MoveIt version.

By default, MoveIt works with the PANDA Robotic Arm (see Figure 35 (b)). To use a UR3 robot arm (see Figure 35 (a)), we need an extra software to communicate it with our system through ROS [18]. We also need to install an URCap [17] in the robot, a piece of software to communicate the robot with MoveIt. Thanks to MoveIt, our system can work with both robotic arms, as it is easy to change from one to another by only changing the robot name.

Moreover, the main reason for using MoveIt is that the problems seen in Figures 36 (a), (b) and 37 are resolved. The OMPL plans a trajectory free of collisions. In addition, the robot does not move directly to a specified pose. Instead, it stops near the object (in our case, the minimum distance is 9.5cm over the object), whenever a "grasp" or "place" action takes place, making the robot movements safer. The drawback though is that sometimes it cannot find a safe motion, or it cannot reach an object and, therefore, the robot does not move. In these situations, the robot, as described in previous sections, will ask for the user's help. Therefore, before starting the execution of any action, we always evaluate if the robot can grasp or place an object.

3.4 Verbal Communication

Communication between the user and the robot takes place via verbal communication. For this reason, in this Section we are going to explain different dialogue managers but, before, we want to briefly describe what we were looking for when choosing one.

In the first place, we are looking for an event-based manager, that could allow us flexibility for most language-based commercial interactions. Second, we considered the response time, as it is critical in natural interactions: a delayed response is seen as disturbing, whereas a very quick response is often seen as insincere. Third, we want the robot to ask for additional information. For example, the user may ask to set the table, but the robot does not know for how many diners, so we want it to ask for this information. Finally, we also want to be able to generate speech fast, since the robot must give feedback of what it is doing.

In the following sections, we will talk about (1) the different assistants that we have considered for this project given the requisites explained above, (2) which one has been chosen, and (3) the verbal requests that the robot understands.

3.4.1 Siri

Siri, the intelligent assistant developed by Apple, was the first assistant that came out to the market. However, it is only compatible with approximately 250 devices. This is because all devices that want to connect to Apple's smart home software (HomeKit) must meet the encryption requirements of Apple. However, the issue of privacy is a positive aspect. The company does not associate the data with a personal profile.

This voice assistant has not been selected as it is only compatible with Apple devices.

3.4.2 Google Home

Goggle Assistant, developed by Google, was one of the first virtual assistants. It is integrated into all mobile phones that have Android as their operating system. In addition, it has an application available for both Android and IOS. With this assistant, more than 10,000 devices can be controlled.

In terms of its level of comprehension, complex questions that require some abstraction can be asked. It also identifies the different members of the family. Despite these advantages, it was decided not to use this assistant because its voice is too robotic. Moreover, few recognition tests were informally performed by other members of the URL-MONITOR project, and its speech recognition accuracy was not as high as with Alexa, which we describe next.

3.4.3 Alexa

Amazon's Alexa, launched five years ago, has over 28,000 compatible devices. Similar to Google Home, it has an application for Android and IOS. Contrary to the Google Home Assistant, Alexa cannot process complex utterances which require additional abstraction capabilities. On the other hand, its voice sounds more natural than the Google Home assistant. For this reason, Alexa has been chosen as an assistant in this project.

Amazon has multiple devices integrated with Alexa. They also have WiFi connectivity and Bluetooth. We are now going to talk about them, and which one we are using in this project:

- Echo Flex. This device is a plug-in speaker. It integrates a 15mm mini speaker, therefore the quality of the audio is poor.
- Echo Dot. It has a fabric-finished design and a 40mm speaker. It can also integrate a clock. The quality of the audio is better than the Echo Flex.
- Echo Plus. It has high-quality surround sound (76mm neodymium woofer and 20mm tweeter), integrated digital home controller (Zigbee) and temperature sensor.
- Echo Studio. It is defined as a high-fidelity smart speaker with 3D audio (3x50mm midrange, 25mm tweeter and 133mm woofer). It integrates Dolby Atmos and Zigbee.
- Echo Auto. This device is designed for the car. It connects to the car speakers through Bluetooth.
- Echo Show 5. It integrates a 5.5" screen and 1MP camera that allows video calls. The speaker is of 4W.
- Echo Show 8. Similar to the Echo Show 5, with an 8" screen and integrates two speakers of 10W.
- Echo Show. It has a screen of 10,1", two speakers of 10 W and a passive bass radiator with Dolby technology for immersive sound. The camera is of 5MP.

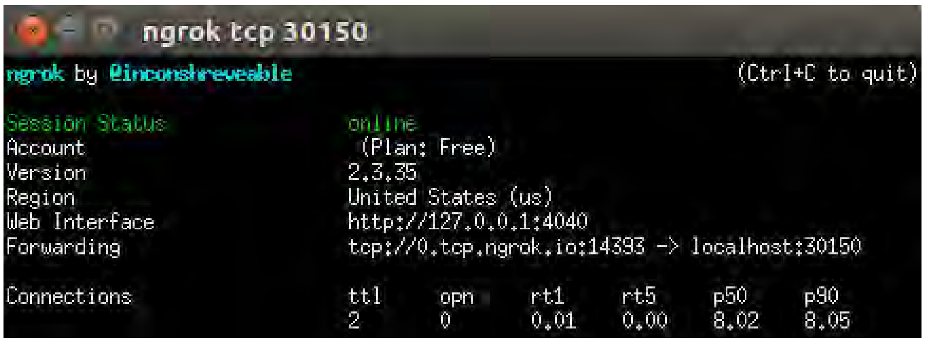
In this project we have opted for the Echo Plus device, since it has a good sound quality and can connect with Zigbee devices, which could be useful if our system is integrated in smart homes in the future.

To interact with our system, we have developed a skill in the Alexa Development platform. This skill is in Python and communicates with the *Supervisor* via socket using ngrok [8], a tunneling tool. Alexa skills are run in the cloud and therefore, we need to translate private IPs to public IPs and vice versa (NAT). ngrok provides us this service. An example of a ngrok window, where we can see the public IP and port, and the private IP and port is shown in Figure 39. In this case:

- public IP: 0.tcp.ngrok.io
- public port: 14393

- private IP: localhost
- private port: 30150

We can also observe the number of petitions done, which in this case is 2 (looking at TTL parameter).



```
ngrok tcp 30150
ngrok by @inconshreveable                                          (Ctrl+C to quit)

Session Status                online
Account                        (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    tcp://0.tcp.ngrok.io:14393 -> localhost:30150

Connections                   ttl     opn     rt1     rt5     p50     p90
                              2       0       0.01    0.00    8.02    8.05
```

Figure 39: Example of ngrok window.

Our Alexa skill is called "aided skill", so the user has some sentences, with its name, to activate it, such as:

- Open aided skill
- Start aided skill
- Aided skill
- Ask aided skill to ...

Alexa skills are based on intents. An intent represents an action that fulfills a user's spoken request. Intents can optionally have arguments called slots. We have created a slot, called "objects", that include:

- cell phone, phone, mobile phone, mobile
- scissors
- pencil
- pen
- cup
- candle
- wine
- fork
- water bottle, water
- wine glass
- spoon

- knife
- pocket knife
- dining table, table
- glass
- bread plate
- plate
- coffee cup
- teaspoon

The intents created are:

- AskHowManyObjects: To request the number of objects in the scenario. To activate this intent, the user can say:
  - How many objects are on the table
  - How many objects do you see
  - How many objects are there
- AskWhichObjectsAre: To request which objects are in the scenario. The system also replies indicating the number of objects of each type. The possible questions are:
  - What do you see
  - Which objects can you see
  - Which objects are on the table
- AskDoYouSeeObject: To ask if the robot sees a specific object. This intent has a slot named "object", which is of type "objects", seen before. To activate the intent, the user should say something similar to:
  - Do you see a {object}
  - Can you see the {object}
  - Do you see the {object}
  - Do you see {object}
  - Can you see {object}
  - May you tell me if you can see the {object}
- Replan: To request the system to replan. The robot will evaluate the scenario once again and will create a new plan to achieve the goal. The intent can be activated by saying:
  - Launch rosplan
  - Plan again
  - Replan

- Wait: To ask to wait if the robot needs a specific object and it is not in the scenario, so he or she can bring it. The user can say:
  - Wait a second
  - Wait for a while
  - One moment
  - One second
  - Wait a moment
  - Wait
- UserDoesIt: In some situations, the robot can ask the user if the user wants to do a specific task. If he or she wants to do so, s/he just needs to say:
  - I can move it
  - I can reach it
  - I will do it
- TakeObjectOut: To request the robot to remove an object. The robot will get it and put it or them apart, in the *unnecessary objects* zone. It is only done when there are extra objects in the scenario. This intent has a slot named “object”, which is of type “objects”, seen before. To activate the intent, the user can say:
  - Put them all apart
  - Take out the objects
  - Take out all the objects
  - Put the objects apart
  - Put the {object} apart
  - Put the object apart
  - Take out {object}
  - Take out the object
  - Take out the {object}
- SetTheTable: The user can ask to start setting the table and must indicate for how many people. This intent has a slot named “number” which is of type “AMAZON.NUMBER”. This slot is compulsory so if the user does not say it, Alexa will ask for it. The user can say the following sentences:
  - Lay the table
  - Let’s set the table for {number}
  - Let’s set the table
  - We can start setting the table for {number}
  - We can start setting the table
  - Start setting the table
  - Set the table

  - Set the table for {number}

- StopRobot: It stops the robot, allowing the robot to continue the task if the ContinueRobot petition is launched. The intent is launch by saying:
  - Cease moving
  - Look before you leap
  - Watch robot's step
  - Watch your step
  - Pause the robot
  - Pause your movement
  - Stop UR
  - Stop the robot
  - Danger
  - Stop the movement
- ContinueRobot: To resume the task in case it was stopped with the previous intent. The user can say:
  - Get into work again
  - Continue
  - Continue working
  - Robot continue what you were doing
  - Robot go on
- UserIsDone: To indicate that the user has finished his/her task. For instance, if s/he had to bring a missing object. To begin the intent, the user can say:
  - I have finished
  - I am done
  - I'm done
- AMAZON.CancelIntent: To cancel what the robot is doing.
- AMAZON.HelpIntent: To ask for help if the user does not know what the skill covers.
- AMAZON.StopIntent: To stop the skill.
- AMAZON.NavigateHomeIntent: To exit the skill and return the user to the home screen. It is only active on screen devices.

Although having all these intents created, Alexa is not able to start a conversation. It can only reply to a conversation initiated by the user. This is a disadvantage because in some situations, we need to indicate the user what the robot is doing, as part of an HRI design principle on feedback. For this reason, we use the alexa-remote-control [47] module, which is the block in Figure 15 named TTS. We can thus directly connect to the Echo Plus and send a String with the utterance we want the system to say. Unfortunately, the Echo Plus will not reproduce the utterance if the skill is open. Therefore, every time the user actives an intent, the skill is automatically closed right after. There are only three intents that leave the skill open, as they are thought to use when the robot is not doing a task and are only used to know the state of the world seen by the robot's eyes perceived by the *Vision* system. These intents are AskHowManyObjects, AskWhichObjectsAre and AskDoYouSeeObject.

3.5 Vision system

This module has not been implemented in this TFM, so we are going to do an overview in this section. For further details see [24].

The module uses You Only Look Once (YOLO) which is a Convolutional Neural Network (CNN), a type of neural network, which are very good at detecting patterns (and by extension objects and the like) in images.

The dataset used is from Open Images Dataset (OID) V4. It is provided by Google and is the largest existing dataset with object location annotations with approximately 9M images for 600 object classes that have been annotated with image-level labels and object bounding boxes. In our case, we use seven classes:

- Bottle of any kind: from a liquor bottle to a ketchup one.
- Fork.
- Kitchen knife. We differentiate it from the pocket knives and the butcher knives.
- Mobile phone.
- Plate: from empty plates to plates full of meal.
- Spoon.
- Wine glass.

Table 4 shows the number of images that have been used as dataset in different situations. We must say that we have used all the images provided by OID V4, since the more images we have, the better is the recognition. The training set consists of all the images used to train the neuronal network. The validation set is used every once in 50 iterations, in our case, to check whether the training process is complete and to avoid overfitting. Thus, we ensure that there are images that the network has never seen before.

Finally, once the training has finished, the test set are new images to validate the response of the neuronal network: to illustrate, if we have an image of a bottle, the neuronal network should identify it as so. On the other hand, we can appreciate that there are more labels than images. This is because some images have more than one object of the class.

Table 4: Number of images used as dataset for every class.

Class	Training set	Test set	Validation set
Bottle	11456 images 40155 labels	533 images 979 labels	177 images 340 labels

Table 4: Number of images used as dataset for every class.

Class	Training set	Test set	Validation set
Fork	1034 images 1687 labels	103 images 143 labels	32 images 45 labels
Kitchen knife	274 images 350 labels	61 images 83 labels	21 images 28 labels
Mobile phone	4312 images 6365 labels	354 images 454 labels	98 images 137 labels
Plate	2023 images 5416 labels	153 images 214 labels	47 images 67 labels
Spoon	1044 images 1709 labels	119 images 162 labels	40 images 50 labels
Wine glass	4894 images 12934 labels	239 images 356 labels	82 images 119 labels

Although, in order to increase more the dataset, data augmentation is performed.

Definition 5. Data augmentation: Process in which a dataset size is increased by obtaining more training samples from every image applying transformations. These transformations usually involve [33]:

- Random Image Warping Transformations: creates a randomized 2-D affine transformation from a combination of rotation, translation, scale (resizing), reflection, and shear.

- Cropping Transformations: crops the image. It is important to select a cropping window that includes the desired content in the image.

- Color Transformations: randomly adjusts the hue, saturation, brightness, and contrast of a color image.

- Synthetic Noise: adds noise in the picture from a noise model, such as Gaussian, Poisson, salt and pepper, and multiplicative noise. The strength of the noise is also adjustable.

- Synthetic Blur: applies randomized Gaussian blur to an image. The amount of smoothing can be specified.

In the project scope, flipping, resizing and random color transformation, as shown in Figure 40, is applied in means to augment the dataset.

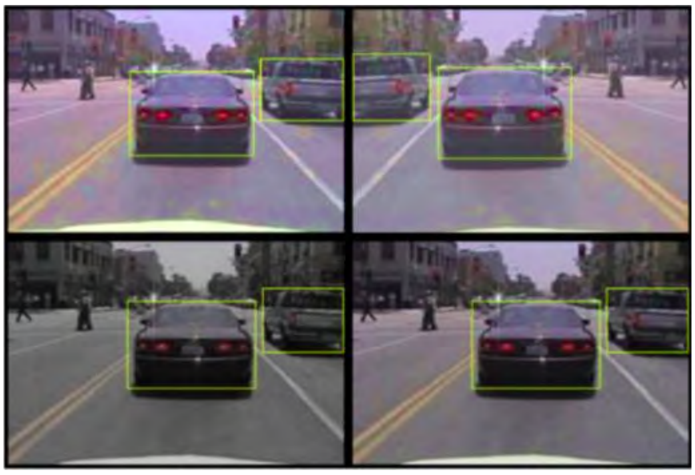


Figure 40: Transformations applied to the dataset images.

Once we have our neuronal network working, we need to define the area of detection and the setup of the camera, since these are important aspects when recognizing the objects and localizing them.

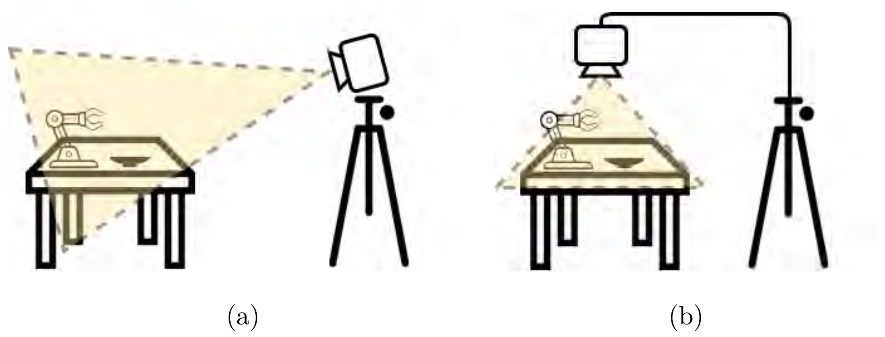


Figure 41: Setups of the camera used. (a) The camera is higher and next to the table. (b) The camera is above the table.

In a first instance, the setup was the one shown in Figure 41 (a). As we can see, the camera is next to the table, a little bit higher. The problems we encountered in this setup were:

1. It was hard to detect the objects that were far from the camera (see Figure 42 (a)).
2. The objects are in perspective.
3. Due to the previous problem, achieving an accurate positioning of the objects was harder.

To solve points 2 and 3 we opted to: (1) change the area of detection to only recognize the objects on the table, (2) use the corners of the table as a reference point, and (3) distorted the image according to it. Knowing the real dimensions of the table and the pixels it occupies on the image we can do a simple operation to find the positions of the objects and their dimensions. However, another inconvenience then arises:

4. The objects are distorted (see Figure 42 (b)).

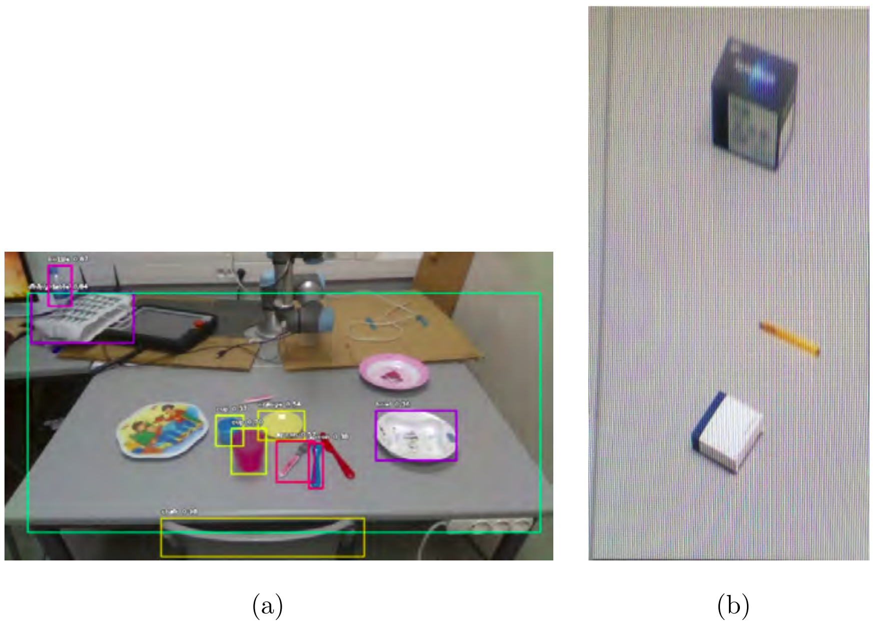


Figure 42: (a) Image captured by the camera when placed in front of the table with the corresponding perspective view. (b) Distorted objects after applying the transformations to remove perspective.

Because of these drawbacks, we opted for the setup shown in Figure 41 (b). The area of detection is the limited zone created by the four red points shown in Figure 43. With this top view, we can see that the objects are not distorted, and the localization is simpler. Although, it presents a problem, which is that some objects, such as a bottle, are not recognized, because the neuronal network used to recognize objects is mainly trained on images from the side view. This is not the case for other objects, such as spoon, knife, and so on, as they are trained with pictures seen from the plan view.

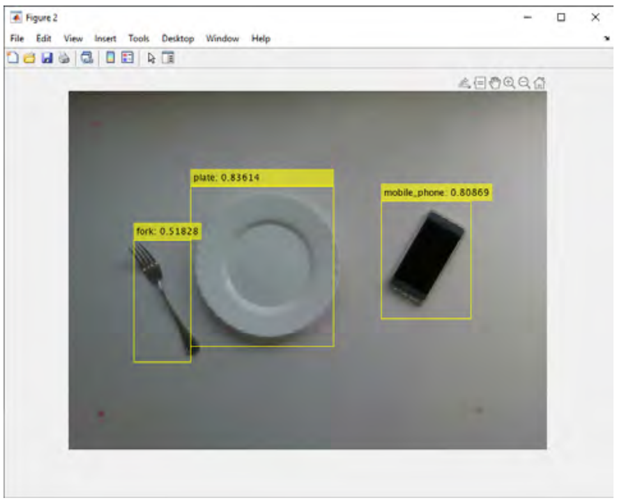


Figure 43: Image taken from the top view of the table.

3.6 World Manager

This module is in charge of processing the state of the world send by the *Supervisor*, which in turn, receives it from the *Vision* module (see Figure 15). To do so, it receives a list of the objects that the *Vision* module recognizes, along with the following additional information:

- *id*: it is the id of the object. It is a number.
- *name*: it is the name of the object. It is formed by the type and a number. For instance, *plate*1.
- *type*: it is the type of the object. In the example above, it would be *plate*.
- *pddl type*: it is the PDDL type, which can be a *block* or a *table*. In the previous example, it would be a *block*.
- *color*: is the color of the object. This information is not currently used in this project.
- *location*: it is the position of the item with respect to the robot. It contains x, y and z values.
- *dimension*: it is the dimension of the item. It contains x, y and z values
- *is on*: it indicates if the item is on top of another one. This field is empty if the item is a zone.
- *is bellow*: it indicates if the item is under another object.

On the other hand, this module receives petitions from other modules and, therefore, has to send the information required. The petitions that it can receive are the following:

- get the vision objects,

- get object with a specific name,
- get number of objects,
- get number of objects, but without tables (for PDDL type),
- get instances of, given a type,
- get location of an object given an id,
- get location of an object given a name,
- get the name of the object given the position of the list,
- get type of the object given the position of the list,
- get the PDDL type given the position of the list,
- get the table names for the PDDL file,
- get the block names for the PDDL file,
- get if the object is under another one given the position of the list,
- get if the object is on another one given the position of the list.

3.7 Communication

To communicate the different nodes of the project, different ROS topics and services have been created (as described in Figure 44. We will next go through them.

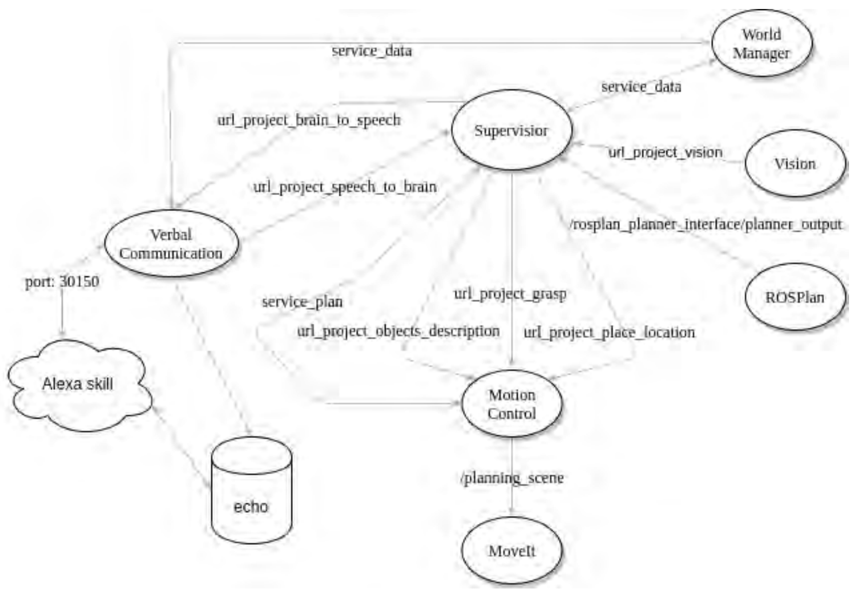


Figure 44: Topics and Services to communicate the different Nodes.

3.7.1 Topics

Many different topics were created in order to communicate the different nodes. We next describe their purposes, the information they send, and which node is the publisher and which ones are subscribed to it.

- url_project_brain_to_speech:
  - Message type: std_msgs/String (see Section 8.2.1)
  - Publisher: *Supervisor*
  - Subscriber: *Verbal Communication*
  - Function: This topic sends Strings to the *Verbal Communication* to give feedback to the user. Hence, is the channel the *Supervisor* uses to convert Text-To-Speech.
- url_project_speech_to_brain:
  - Message type: url_project/speech_to_brain_msg (see Section 8.2.2)
  - Publisher: *Verbal Communication*
  - Subscriber: *Supervisor*
  - Function: Sends the petitions received from the user to the *Supervisor* to process them and act according to the user's needs.
- url_project_grasp:
  - Message type: url_project/grasp_description (see Section 8.2.3)
  - Publisher: *Supervisor*
  - Subscriber: *Motion Planner*
  - Function: Sends the required information to grasp the object.
- url_project_objects_description:
  - Message type: url_project/objects_description (see Section 8.2.4)
  - Publisher: *Supervisor*
  - Subscriber: *Motion Planner*
  - Function: Sends the information about the objects.
- url_project_place_location:
  - Message type: url_project/place_location_description (see Section 8.2.5)
  - Publisher: *Supervisor*
  - Subscriber: *Motion Planner*
  - Function: Sends the required information to place an object on a surface.
- url_project_vision:
  - Message type: url_project/vision_description (see Section 8.2.6)
  - Publisher: *Vision*
  - Subscriber: *Supervisor*
  - Function: The *Vision* node sends the objects information through this topic to the *Supervisor*.

3.7.2 Services

There are two services in the system:

- Data
  - Messages send are shown in Section 8.2.7.
  - Node that offers the service: *World Manager*
  - Nodes that make petitions: *Supervisor* and *Verbal Communication*
  - Function: Gives the information of the world state required.
- Plan
  - Messages send are shown in Section 8.2.8.
  - Node that offers the service: *Motion Planner*
  - Node that make petitions: *Supervisor*
  - Function: It evaluates if the specified movement can be performed. It returns a Boolean indicating if it is feasible or not.

4 Use Case evaluation

In this Section we are going to describe the different tests done to our system. We will start with a first proof of concept of a simplified version of the system. Next, we evaluate the skill we have developed. We continue evaluating the integration with the *Vision* system. Thirdly, we analyze different situations the system faces and how it responds showcasing the different features designed. Finally, we show a full scenario setting.

4.1 Initial proof of concept evaluation

As a first step, done to test the viability of the URL-MONITOR project, we designed a simple scenario where the goal was to detect two objects on the table, a cup and a cell phone, and to ask the robot to move the cup on top of the phone. The aim was thus to test the integration of the three main modules of the system: *Vision*, *Motion Planner*, and *Task Planner*. This first system, which is far from resembling the one presented in this project in terms of design and complexity, had the following characteristics:

- It was not running with ROS, but used a simple Python controller and all the communications between the different modules were done through sockets.
- As already mentioned before, the system used STRIPS [22] to plan the actions the robot has to do.
- It also had some basic Alexa intents to indicate the robot that it had to start doing the specified task.
- Regarding to the movements of the robot, all the communication was done through the ports specified in Section 3.3.1, where the problems described there were not significant, as only two objects were on the table.
- The *Vision* module's setup was the one shown in Figure 41 (a) and it was able to detect the objects and compute their coordinates to grasp and place the cup properly. It also had the height of the cup hard-coded, as the module could not detect it from the perspective from where the camera was placed.

The sequence of actions is shown in Figure 45, where we can see that this first implementation was functional. Encouraged by the success of this initial prototype, we continued developing the AIDED system described in this TFM.

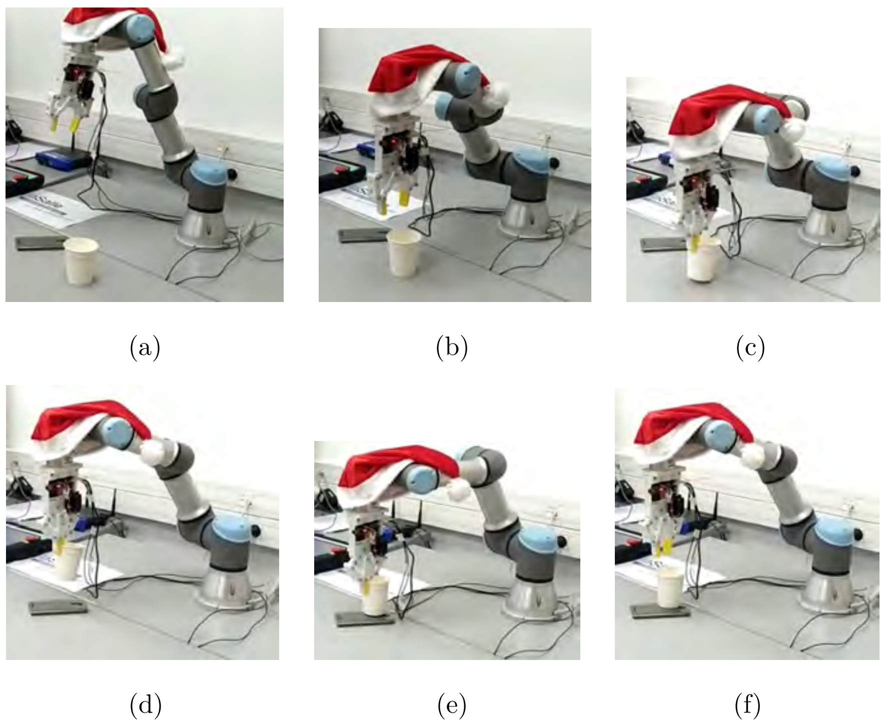


Figure 45: First evaluation with a basic system. (a) Initial state. The cup and the cell phone are on the table. (b) The robot goes closer to the cup. (c) The robot grasps the cup. (d) The robot moves the cup. (e) The robot places the cup on top of the mobile phone. (f) Final state. The cup is on the cell phone.

We next describe the evaluations done of the different modules, as well as a final performance evaluation of the whole system.

4.2 Verbal Communication

In this Section we want to test if the Alexa skill can recognize verbal petitions from users. Therefore, we want to know if the intents created fulfill their function or, otherwise, some basic sentences have to be added. Furthermore, we want to evaluate the usability aspect of Alexa's skill activation process.

4.2.1 Intents evaluation

In this Section we evaluate the different intents created. To do so, 10 users (see Table 5), from different backgrounds have been picked. There are six users from Spain and four from California. They also have ages from 13 to 48 years old.

Table 5: Users' information.

User	Important information	User	Important information
1	Gender: F Age:18 From: Spain	6	Gender: F Age: 23 From: Spain
2	Gender: M Age: 48 From: Spain	7	Gender: F Age: 45 From: California
3	Gender: F Age: 23 From: Spain	8	Gender: F Age: 17 From: California
4	Gender: F Age: 23 From: Spain	9	Gender: F Age: 13 From: California
5	Gender: F Age: 23 From: Spain	10	Gender: M Age: 45 From: California

We described a situation to them, and they came up with sentences that would say to the robot. In the following list, we indicate the intent of the Alexa skill, the situation described, and the commands that the skill did not recognize that the users came up:

- AskHowManyObjects: You want to know the quantity of objects on the table.
  - Count the number of objects on the table, please
- AskWhichObjectsAre: You want to know what is on the table.
  - What is there
  - Can you see anything
  - Can you tell me everything you see on the table
  - Can you explain to me what you see on the table
  - Describe everything on the table
- AskDoYouSeeObject: You want to know if the robot sees an object in concrete (glass, fork, for instance).
  - Is the glass on the table
  - Is the glass there
  - Is there any fork on the table
  - I cannot find my glass. do you see it
  - Is the fork visible
- Replan: Some objects are missing to complete the task and you want the robot to work without them.

- Reschedule
- Do it anyway
- Do it again without the objects
- Work with that
- Work with them
- Set the table anyway
- Lay the table anyway
- Just ignore the objects that are missing
- Set the table without these objects, please
- Proceed setting the table regardless
- Complete the task
- Do not mind what is missing
- Finish the task without those objects
- Proceed to set the table with only the objects available

• Wait: Some objects are missing to complete the task and you will bring them to the robot.

- Do not worry I will bring them
- Stop. Here you have the objects. You can continue
- Please, wait. I will bring them to you.
- Let me find them. One second.
- Wait, I will bring the missing objects
- I will provide the rest of the objects necessary

• UserDoesIt: The robot asks for help to move an object, as it cannot do it. You want to help it, what would you say to it, to indicate so?

- I do
- Do not worry, I will help you
- Stop, let me help you
- I am going to help you
- May I help you
- I will move the object
- Okay, I will pass to you
- Wait a minute I am my way to help
- I am coming

• TakeObjectOut: There are more objects than needed on the table. The robot asks you what you want it to do with them and you do not want it on the table.

- Remove the dish

  - I do not want them
  - Move them away
  - Put it somewhere else
  - Place it outside the table, please.
  - I don't want it on the table
  - Remove the objects from the table
  - Please, remove the objects from the table
  - Is it possible to move the objects from the table
  - Please remove all
  - Please take it all
  - Go ahead and remove all
  - Remove the unidentified objects
  - Continue removing all objects

- SetTheTable: You want the robot to set the table.
  - All the sentences that the users came up with were recognized by the skill

- StopRobot: The robot is moving towards you, with risk of touching you and hurting you.
  - Do not move
  - You are too close
  - Take care
  - Stop
  - Be careful, I am close
  - Stop and wait to be reset
  - Do not come close to me
  - You are going to hurt me
  - Stop now

- ContinueRobot: The robot is not moving, as you have stopped it. What would you say to it for make it do what it was previously doing?
  - Start the robot
  - Resume activity
  - Move
  - Start
  - Start working
  - Start moving
  - Do your job
  - Play

  - Why aren't you moving
  - Reset
  - Proceed with the task
  - Keep it going
  - Please finish your job
  - Go back to your job please
  - May you go back to work
  - You are good to go

- UserIsDone: You helped the robot and it is waiting you to indicate somehow that you have done it.
  - I finished what I was doing
  - I finished the work I was doing
  - I finished the task
  - I did for you the task
  - I made for you that
  - I did the task
  - The task is finished
  - The task is done
  - I am finished
  - I just finished the task
  - Here are the objects that were missing
  - You may put these objects that were missing
  - Proceed to put these missing objects on the table
  - Here you have what you need to finish/complete
  - The situation has been resolved
  - I've already helped you

We must say that some of the commands such as the ones in the StopRobot ("Stop") or in the UserIsDone ("I am finished") intents, close the skill, instead of activating the intents. Hence, the system does not receive the petition. Furthermore, we appreciate that the ContinueRobot and the UserIsDone intents are quite similar, as in both cases the robot has to start working again. Finally, we would like to remark that some commands such as "do not worry, I will help you" did not work to activate the intent, although it works saying "I will help you".

We can see that the only intent that has worked with all the purposed sentences is SetTheTable.

All these commands purposed would be added in a future work, as they have to be evaluated in order to make sure that the skill recognize them as the correct intent. To illustrate, in the Wait intent, one user came out with "Stop. Here you have the objects. You can continue", which could be confused with the StopRobot intent.

In order to narrow the sentences said by users, the system proposes sentences to the user while it asks to wait, for instance. It says *" Okay, waiting... Let me know when you are done saying: 'Alexa, ask aided skill I am done' "*. You will see more in sections 4.4.2 and 4.4.3.

4.2.2 Speech recognition

In this Section we evaluate the accuracy of the Alexa skill when it comes to recognizing a given sentence. The main aspect we want to evaluate is the ease of understanding the name of the Alexa skill, which in our case is "aided skill", because we want to know if Alexa can recognize it properly or if by changing it we could improve its recognition. To do so, 11 users (the same ones specified in Table 5) where selected to say the command shown bellow, 10 times.

ask aided skill to set the table for four

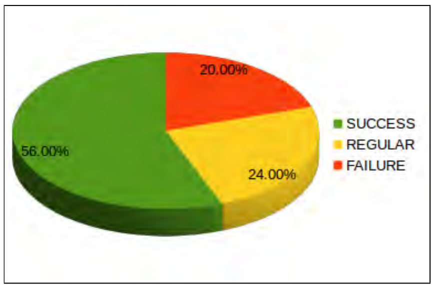
We show in Table 6 the results of each user, where a "Y" means that the sentence was understood by Alexa, "R" means that Alexa partially understood the command (it does not understand the number of people to set the table for), and a "N" means that Alexa did not recognize the command at all.

Table 6: Results of different users repeating out loud the same sentence 10 times.

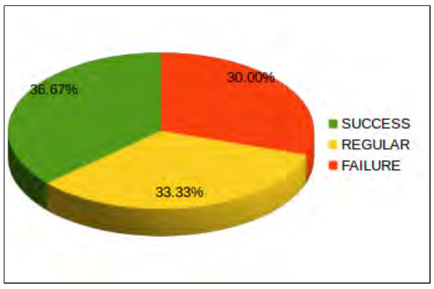
											Total		
User	Results										Y	R	N
1	Y	Y	N	Y	N	N	R	N	N	N	3	1	6
2	Y	N	Y	Y	R	N	N	Y	N	R	4	2	4
3	Y	N	Y	N	Y	N	N	R	Y	Y	5	1	4
4	R	Y	R	N	R	R	N	R	R	R	1	7	2
5	R	Y	Y	Y	N	N	R	Y	R	Y	5	3	2
6	R	Y	Y	Y	Y	R	R	R	R	R	4	6	0
7	Y	N	R	Y	Y	N	Y	Y	Y	Y	7	1	2
8	R	R	Y	R	Y	Y	Y	Y	Y	Y	7	3	0
9	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10	0	0
10	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	10	0	0

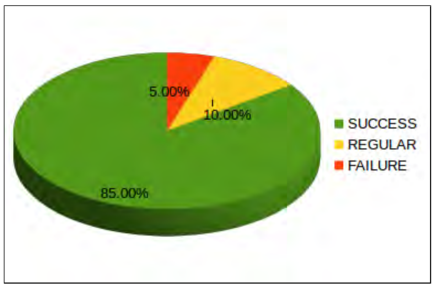
In Figure 46 (a) we can see that Alexa has a 56% of probability to understand the sentence, a 24% to partially understand it, and a 20% to fail if we take into account all the results. However, we also wanted to analyze the impact on performance of native speakers against non-native speakers. So if we take a look at Figure 46 (b), we can see that having only Spanish users as testers, the Alexa probability of understanding the command is reduced compared to Figure 46 (a), whereas the probability of partially understanding the sentence or to not understanding it is increased. On the other hand, considering the results of the Californian users (Figure 46 (c)), we can see that is easier for Alexa to understand the full sentence (85%).

In conclusion, we can confirm that Alexa has a high probability to understand which skill the user wants to use (looking at SUCCESS and REGULAR results). Therefore, it seems that the Alexa skill's name is properly understood, having better results with native speakers.



(a)





(b) (c)

Figure 46: Results of different users saying the same sentence 10 times. (a) Percentages taking into account all the user. (b) Percentages taking into account the Spanish users. (c) Percentages taking into account the Californian users.

4.3 Communication with *Vision* module

In this Section we will explain the different test we have done integrating the *Vision* module. We first evaluate if the information send by the *Vision* module is received by the rest of the modules. Next, we analyse the reliability of the *Vision* module information (in terms that the objects dimensions and positions are consistent). Finally, a setting the table scenario is tested.

4.3.1 Knowing the state of the world

In this Section we showcase the flow of information provided by the *Vision* module throughout the system. Based on this information, the system builds its internal world model, which is used by the different components to achieve the different goals set by the user.

Vision is constantly sending information to the *Supervisor* module which is sent to the *World Manager* to store it, and to the *Motion Planner* to represent the virtual world of the robot (used by the motion planner). When the user asks a question about the environment, the *Verbal Communication* module starts a connection with the *World Manager*, which replies with the required information. Once the *Verbal Communication* module has the response, it verbally transmits it to the user. The information flow is represented in Figure 47.

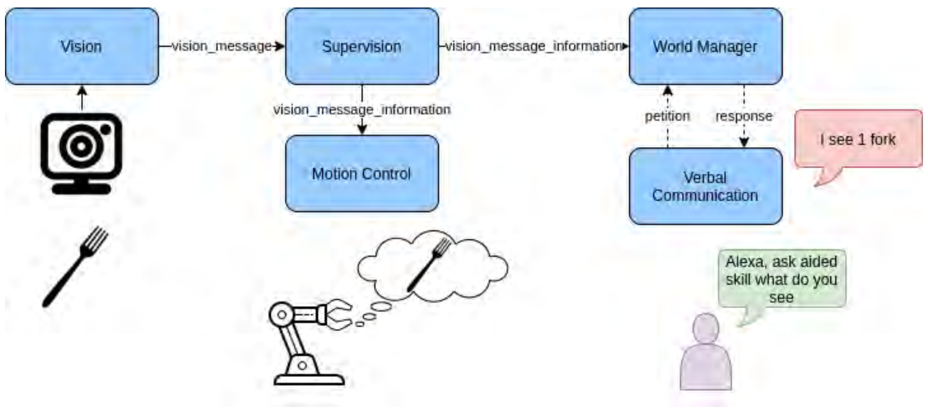
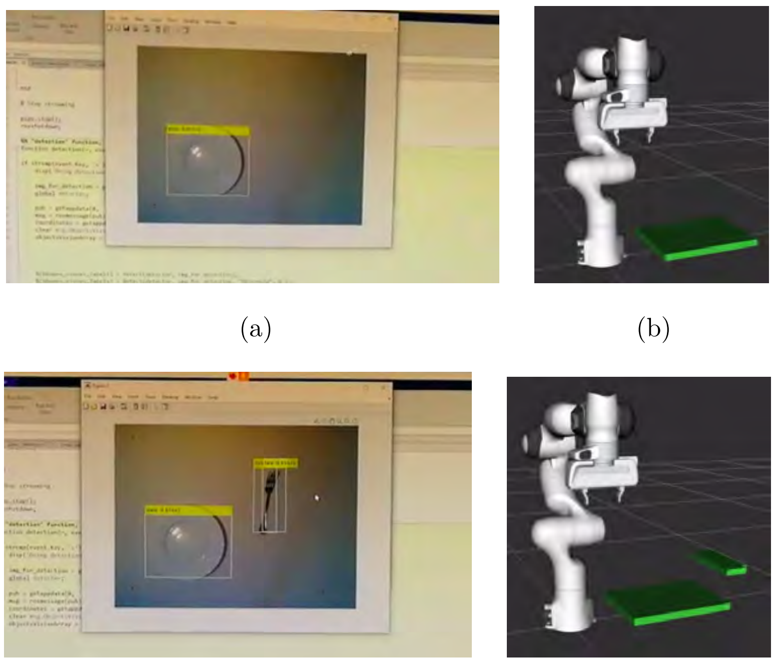


Figure 47: Diagram block of the communication between the *Vision*, *Supervisor*, *Motion Planner*, *World Manager* and *Verbal Communication* modules to get information of the environment.

To verify the flow of the information within the system, we place different objects on the table and see if they appear in the world representation. First, we start with a plate, and we later add a fork. In Figure 48 we can see that the robot represents what the *Vision* module is sending perfectly, in both situations.

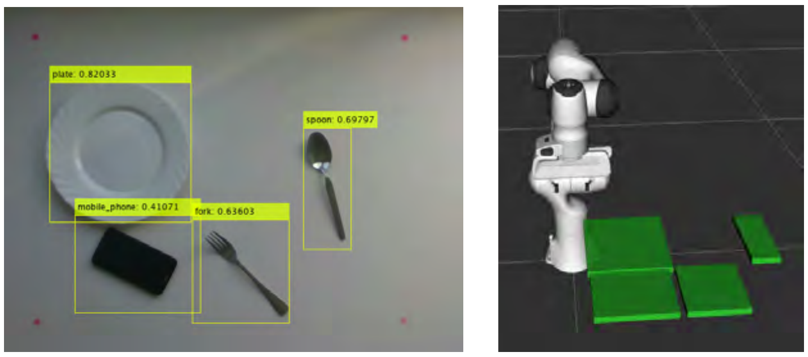


(a) (b)

(c) (d)

Figure 48: Two scenarios and the representation of the world from the robot's point of view. (a) Vision detection of one plate on the table. (b) Robot's world representation. The green rectangle represents the plate detection. (c) Vision detection of one plate and one fork on the table. (d) Robot's world representation. The green rectangle on the left represents the plate detection and the one on the right the fork.

At this point, we can verify if the *World Manager* module sends the information properly to the *Verbal Communication* module. To do so, we ask the system three different questions about the state of the world (subset from Section 3.4.3 which are applicable in this case) shown in Figure 49.

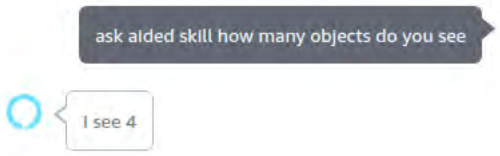


(a) (b)

Figure 49: Scenario to test the Alexa skill feedback. (a) Frame from the *Vision* module to test the Alexa skill. (b) Representation of the state of the world.

The questions are the following:

- AskHowManyObjects. We can appreciate in Figure 50 that the system correctly responds that there are four objects on the table.

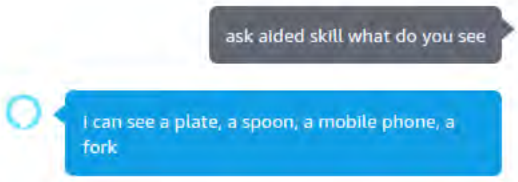


ask aided skill how many objects do you see

I see 4

Figure 50: Alexa response for the AskHowManyObjects intent.

- AskWhichObjectsAre. In Figure 51 Alexa answers the objects that are on the table, which are a plate, a spoon, a mobile phone, and a fork.

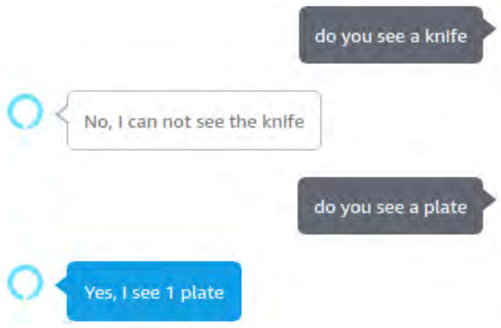


ask aided skill what do you see

I can see a plate, a spoon, a mobile phone, a fork

Figure 51: Alexa response for the AskWhichObjectsAre intent.

- AskDoYouSeeObject. Figure 52 shows the reaction of the skill in two situations: when the user asks for an object that is not in the scenario, and when s/he asks for another that is on the table.


do you see a knife

No, I can not see the knife

do you see a plate

Yes, I see 1 plate

Figure 52: Alexa response for the AskDoYouSeeObject intent.

We can see that Alexa is able to answer the petitions of the user and even indicating the number of objects of each type that are on the table, accordingly to the detections of the *Vision* module.

4.3.2 Reliability on the *Vision* module

In this Section we want to evaluate the reliability of *Vision* in terms of identifying objects and their locations.

To do so, as shown in Figure 53 the modules involved in this test are *Vision*, *Supervisor*, and *Motion Planner*. The *Vision* module constantly sends messages to the *Supervisor*, which re-sends them to the *Motion Planner*.

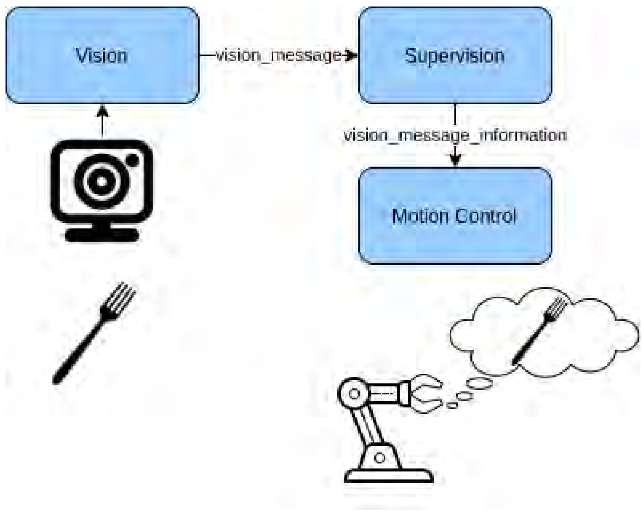
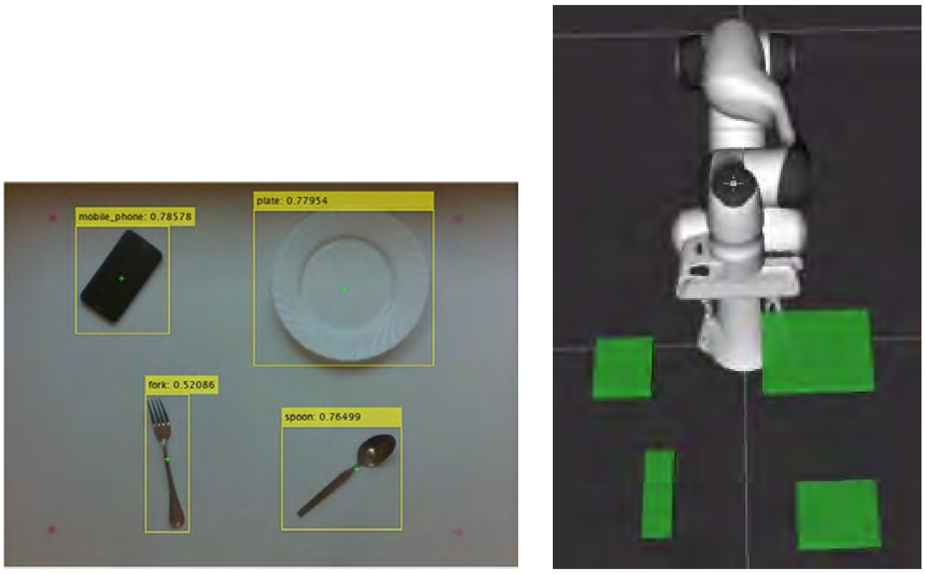


Figure 53: Diagram block of the communication between the *Vision*, *Supervisor*, and *Motion Planner* modules to get information of the environment.

The main idea is that the *Vision* module constantly sends messages and we see the objects in the world representation. With this, we can observe the accuracy of the objects' locations reported by the *Vision* module. Moreover, we can see if it detects all the objects or sometimes some disappear because *Vision* does not recognize them. To see the changes in different detections, we are showing multiple frames in Figures 54, 55, 56, and 57. We also want to show the different values of the dimensions and locations of the objects in Tables 7, 8, 9, 10. As we can see, the detection is almost the same in the four frames and, therefore, we cannot appreciate a big difference in the world representation.

We can thus conclude that the *Vision* module is reliable enough for the purpose of the URL-MONITOR project, as it detects the different objects on the table, and their locations and sizes are consistent as we can appreciate based on their representation in the virtual world.

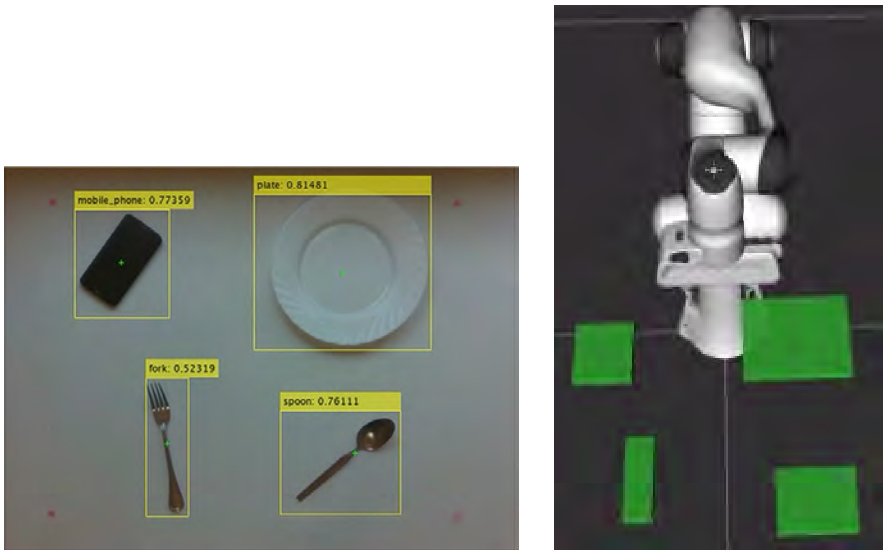


(a) (b)

Figure 54: Frame 1 to evaluate the reliability of the *Vision* module. (a) Object detection by *Vision*. (b) Virtual world representation.

Table 7: Dimensions and locations of the recognized objects in frame 1.

Object	Dimensions	Location
Mobile phone	x: 0.120389103889 y: 0.104455254972 z: 0.019999999553	x: 0.212372452021 y: -0.216245129704 z: 0.209999993443
Plate	x: 0.173501938581 y: 0.200058370829 z: 0.019999999553	x: 0.187525197864 y: 0.135311290622 z: 0.209999993443
Fork	x: 0.154027238488 y: 0.0495719835162 z: 0.019999999553	x: 0.478027820587 y: -0.144163429737 z: 0.209999993443
Spoon	x: 0.115963034332 y: 0.133667320013 z: 0.019999999553	x: 0.5289888978 y: 0.154280155897 z: 0.209999993443



(a) (b)

Figure 55: Frame 2 to evaluate the reliability of the *Vision* module. (a) Object detection by *Vision*. (b) Virtual world representation.

Table 8: Dimensions and locations of the recognized objects in frame 2.

Object	Dimensions	Location
Mobile phone	x: 0.121274322271 y: 0.106225684285 z: 0.019999999553	x: 0.211107864976 y: -0.216245129704 z: 0.209999993443
Plate	x: 0.173501938581 y: 0.197402730584 z: 0.019999999553	x: 0.187523603439 y: 0.134046688676 z: 0.209999993443
Fork	x: 0.15491245687 y: 0.0486867688596 z: 0.019999999553	x: 0.476763218641 y: -0.144163429737 z: 0.209999993443
Spoon	x: 0.115963034332 y: 0.134552523494 z: 0.019999999553	x: 0.528987288475 y: 0.153015568852 z: 0.209999993443

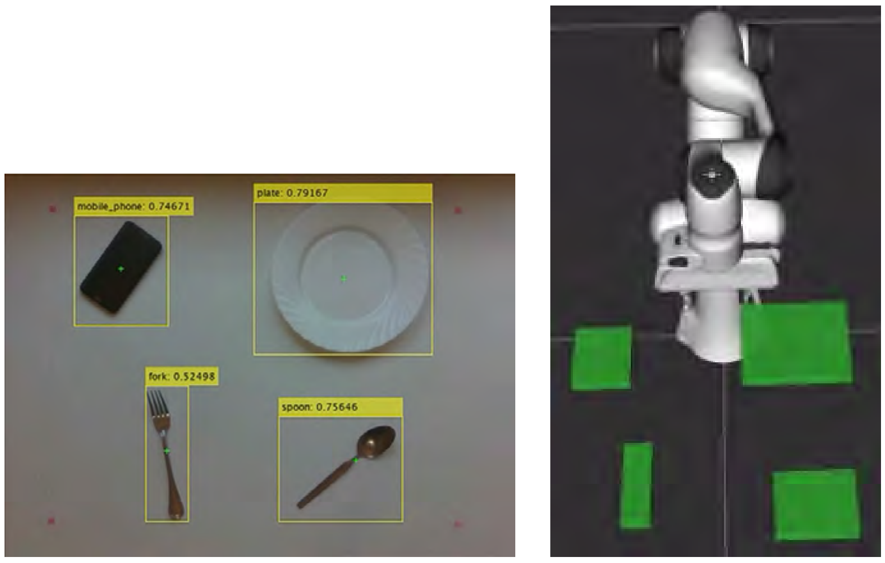
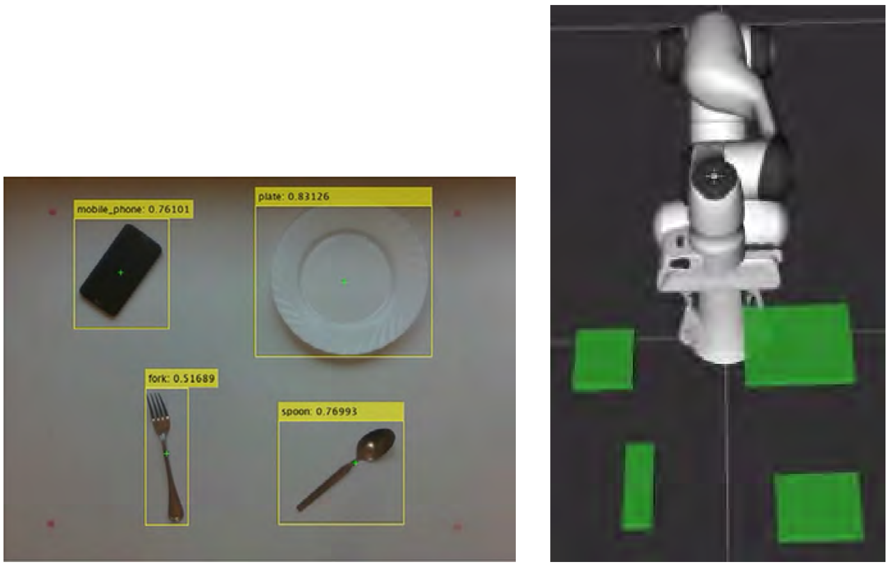


Figure 56: Frame 3 to evaluate the reliability of the *Vision* module. (a) Object detection by *Vision*. (b) Virtual world representation.

Table 9: Dimensions and locations of the recognized objects in frame 3.

Object	Dimensions	Location
Mobile phone	x: 0.123044744134 y: 0.105340465903 z: 0.019999999553	x: 0.20984326303 y: -0.216245129704 z: 0.209999993443
Plate	x: 0.169961094856 y: 0.198287934065 z: 0.019999999553	x: 0.188789784908 y: 0.135311290622 z: 0.209999993443
Fork	x: 0.151371598244 y: 0.0486867688596 z: 0.019999999553	x: 0.478027820587 y: -0.144163429737 z: 0.209999993443
Spoon	x: 0.117733463645 y: 0.13809338212 z: 0.019999999553	x: 0.526459693909 y: 0.154280155897 z: 0.209999993443



(a) (b)

Figure 57: Frame 4 to evaluate the reliability of the *Vision* module. (a) Object detection by *Vision*. (b) Virtual world representation.

Table 10: Dimensions and locations of the recognized objects in frame 4.

Object	Dimensions	Location
Mobile phone	x: 0.123044744134 y: 0.106225684285 z: 0.019999999553	x: 0.20984326303 y: -0.216245129704 z: 0.209999993443
Plate	x: 0.168190658092 y: 0.196517512202 z: 0.019999999553	x: 0.190055981278 y: 0.136575877666 z: 0.209999993443
Fork	x: 0.153142020106 y: 0.0486867688596 z: 0.019999999553	x: 0.476763218641 y: -0.144163429737 z: 0.209999993443
Spoon	x: 0.115963034332 y: 0.139863818884 z: 0.019999999553	x: 0.5289888978 y: 0.154280155897 z: 0.209999993443

4.3.3 Vision integration evaluation

This section shows the *Vision* integration with the AIDED system to set a table.

In a first place, we want to test the *table for one* scenario. The dimensions and localizations of the different zones are shown in Figure 58. The first problem we find is that the zones specified are too small for the objects used. This is mainly because if the spoon, for instance, is placed in diagonal (see Figure 54) it has a bigger area than the zone, so the system detects that it is not correctly placed or that the object that has to go next to it cannot be placed because of collision with the spoon.

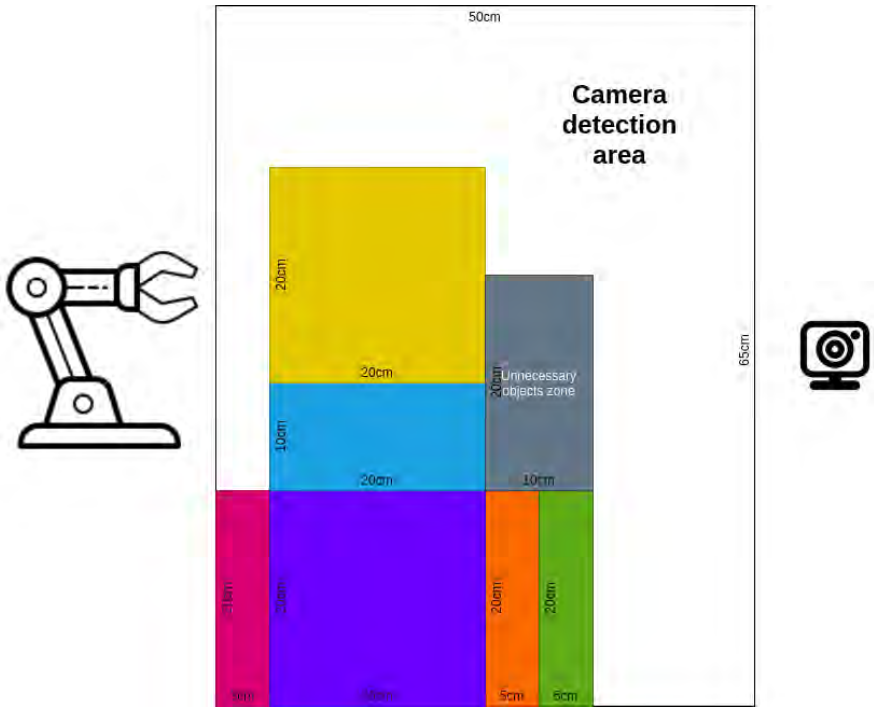


Figure 58: Dimensions of the set up of the *table for one* scenario.

For this reason, we design a completely new scenario, where the zones are 20cm x 20cm and exists a separation between them. In Figure 59 we can see the dimensions of the table used and the positions and dimensions of the different zones. The objects we use in this setting are a spoon, a fork, a plate, and a cell phone.

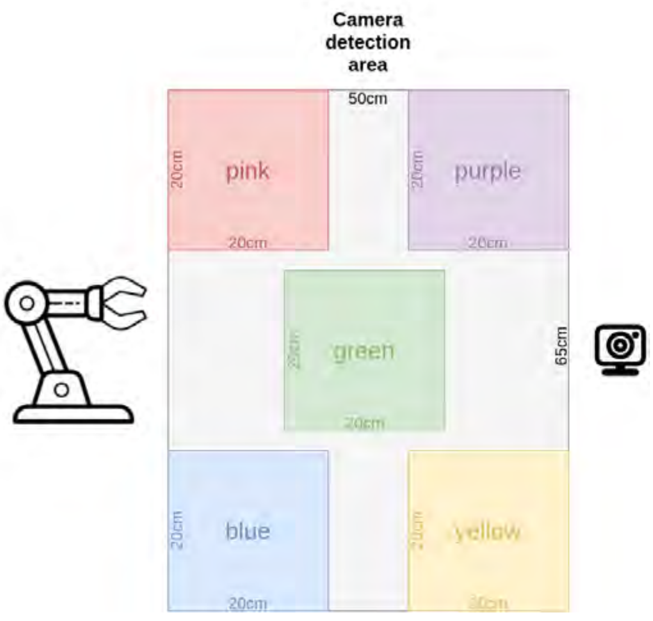


Figure 59: Dimensions of the set up of the scenario.

The initial and final state of the world are shown in Figure 60.

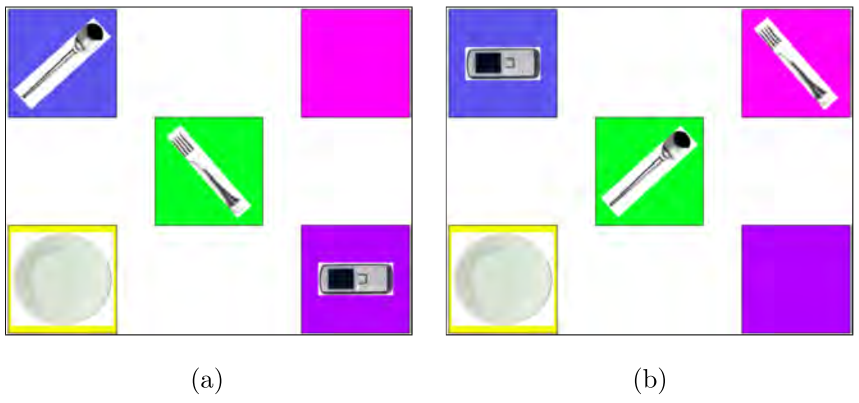
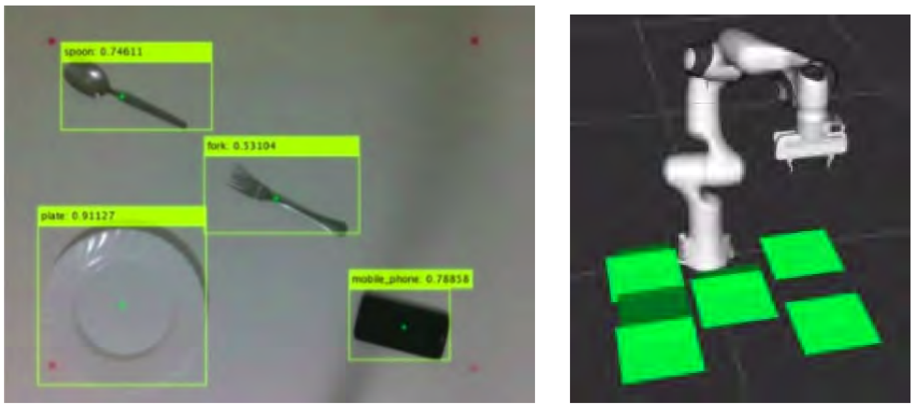


Figure 60: Initial and final conditions to evaluate the complete system with *Vision*. (a) Initial scenario. (b) Final scenario.

To do it, we have our system simulated and we moved the objects manually while the robot is moving them in the representation of the world (see Figure 61). We do not check if there are missing objects in the scenario and if they are placed properly in order to avoid managing that the *Vision* module does not recognize on which zone is the object in that new position and if we placed it correctly.



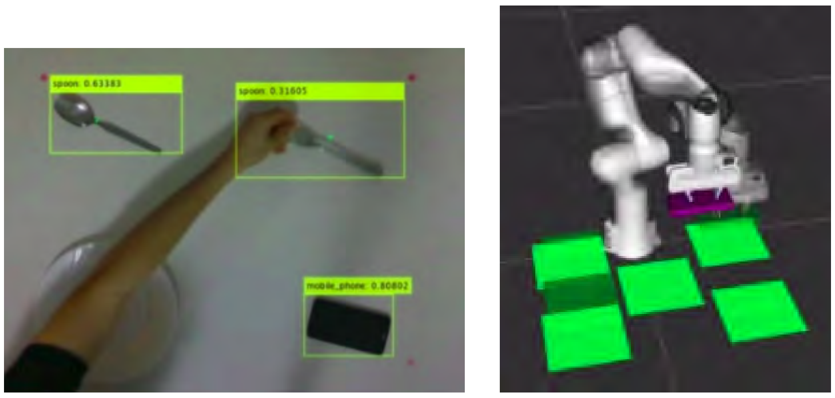
(a) (b)

Figure 61: Initial conditions to evaluate the complete system with *Vision*. (a) Real state of the world. (b) Representation of the world.

The first step given by ROSPlan is:

> 1. Move $fork1$, which is on *green* zone, to *pink* zone

As we can see in Figure 62 we place the fork in that position at the same time the robot is doing it in the representation of the world.



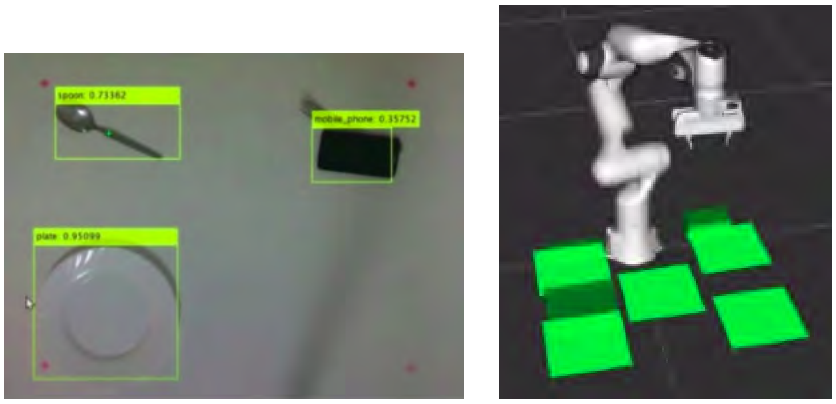
(a) (b)

Figure 62: Moving the fork on the *pink* zone. (a) Real state of the world. (b) Representation of the world.

The second action is:

> 2. Stack *mobile_phone*1, which is on *purple* zone, to $fork1$

We appreciate in Figure 63 (a) that the fork is not recognized by the *Vision* module. For this reason, the mobile phone is on the *pink* zone in Figure 63 (b).



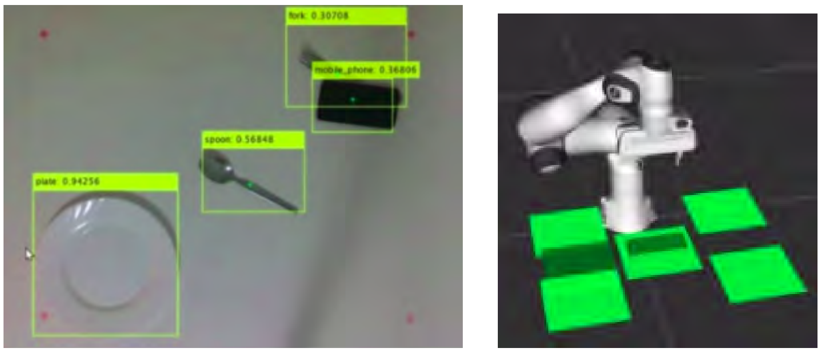
(a) (b)

Figure 63: Mobile phone on top of the fork. (a) Real state of the world. (b) Representation of the world.

The third action to perform is:

3. Move *spoon*1, which is on *blue* zone, on *green* zone

We can see that in Figure 64 the spoon is on the *green* zone and that it recognizes the fork, which is under the cell phone.



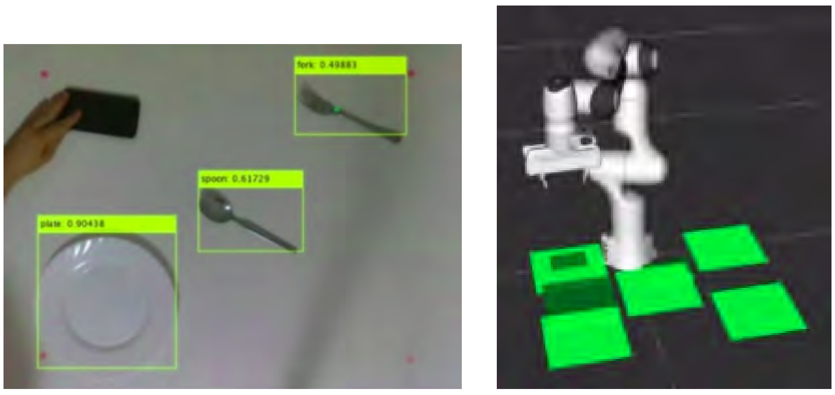
(a) (b)

Figure 64: Spoon on the *green* zone. (a) Real state of the world. (b) Representation of the world.

The last action is:

4. Unstack *mobile_phone*1, which is on *fork*1, and put it on *blue* zone

In Figure 65 we can see that we are moving the mobile phone on the *blue* zone and that the *Vision* module does not recognize it while we are moving it. In this case, when the system reads a *Vision* message to know if the object is placed in its position, the mobile phone is not recognized and, therefore, the system stops.



(a) (b)

Figure 65: Moving the cell phone on the *blue* zone. (a) Real state of the world. (b) Representation of the world.

To conclude with this section, we want to make emphasis in that we need further work in order to perform a full task with all the functionalities of the system. To illustrate, we need:

- the *Vision* module should recognize in each step the zone on which the object is placed,
- the *Vision* module should know the angle of the object in order to put it vertically or horizontally on its place,
- if we know the angle of the object, the *Motion Planner* should indicate it to the gripper for better grasping the object and placing it.
- the *Vision* module should know that if an object is placed on top of another one, the object bellow is still there, although the recognition does not say so. This could be achieved doing object tracking,
- the *Supervisor* module should be able to wait for more *Vision* messages to make sure that the object that has disappeared (in this case, the mobile phone) is not really there,
- the *Task Planner* should not consider as a solution the fact of stacking objects when the object or zone bellow is smaller than the object that is placed on it,

4.4 Scenarios

In this Section we want to test the overall system response to different situations. We will analyze the following situations:

- if there are objects in the scenario that are not needed
- if there are missing objects
- if the robot cannot move an object because of physical limitations or collisions with other objects
- if the moved object is not where it is supposed to be

4.4.1 There are more objects than needed

To test the response of the system in this situation, we work with a scenario with a spoon, a fork, a cell phone, and a candle. The required objects are the spoon and the fork, whereas the extra objects are a cell phone and a candle. We have all four objects on the table, as shown in Figure 66. As the image illustrates, the candle is on the *unnecessary objects* zone, the spoon is on the *blue* zone, the fork on the *green* zone and the cell phone on the *purple* zone (see Figure 59, where the different zones are illustrated).

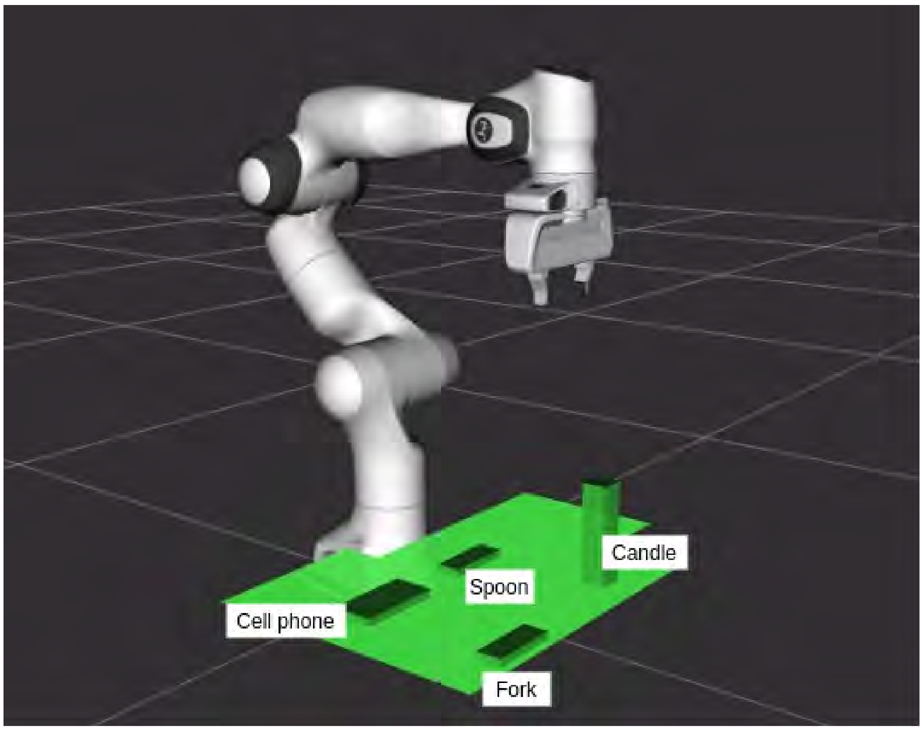


Figure 66: Initial position of the objects with more objects than needed.

As previously described, the *unnecessary objects* zone is used to store the extra objects, so all those objects that are not needed will go there. If there is only one unnecessary object, the robot grasps it and places it in that zone. However, in this scenario there are two unnecessary objects, so we must consider how the robot must perform. We have three options:

1. Stack the unnecessary objects, one on top of another within the *unnecessary objects* zone.

2. Add a fixed number of *unnecessary objects* zones to put each extra object in one of them.

3. Ask the human for help to remove the extra objects.

We discard the first one because, depending on the number of objects to stack and their material, some objects could be broken. The second option is also discarded, as we can reach a situation where there are more extra objects than *unnecessary objects* zones. Therefore, we opted for option 3, which is asking the user for help. In Figure 66 we can see that the candle is already on the *unnecessary objects* zone, so the robot only needs to move the cell phone to this zone. But because, as the *unnecessary objects* zone is full (the candle is already there), the robot will ask to user for help.

At this point, the user has three options:

1. The user does not help the robot. In this situation, the robot cannot proceed, so the task is finished without success.

2. The user removes the candle. Now that the *unnecessary objects* zone is free, the robot can move (if it reaches to grasp and place the object) the cell phone there.

3. The user removes the cell phone from the table. In this case, the robot does not need to do anything else, since the extra objects are removed or on the *unnecessary objects* zone.

In the second and third options, the robot will proceed moving the needed objects to their positions. A flow diagram of all the options explained is shown in Figure 67.

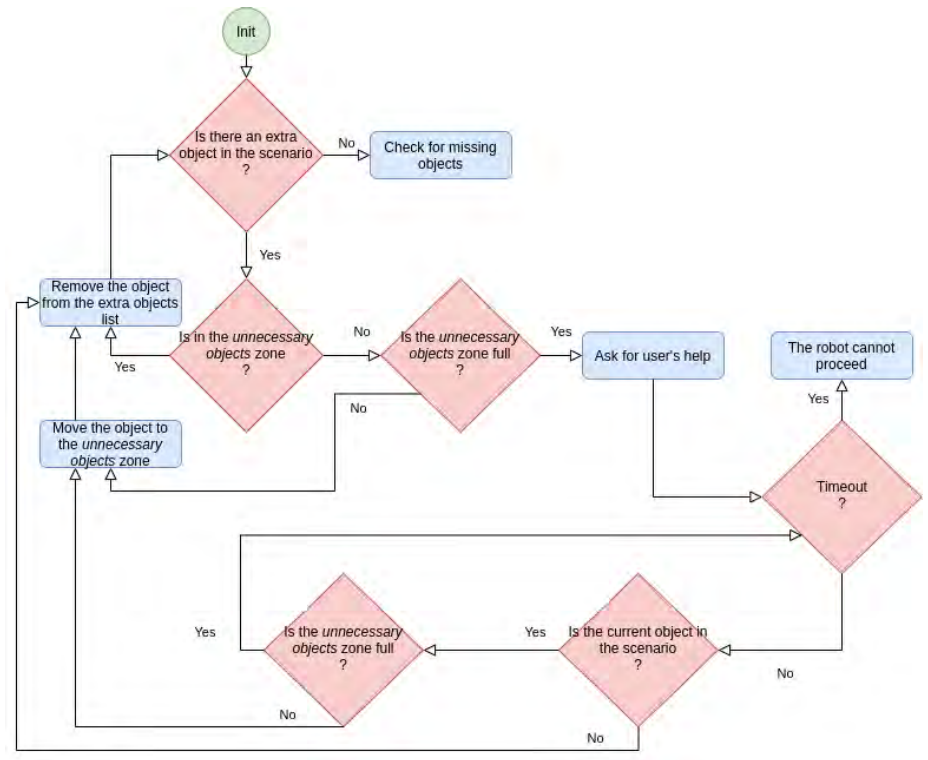

Init
Is there an extra object in the scenario ?
No
Check for missing objects
Yes
Remove the object from the extra objects list
Is in the unnecessary objects zone ?
No
Is the unnecessary objects zone full ?
Yes
Ask for user's help
The robot cannot proceed
Yes
No
Move the object to the unnecessary objects zone
Timeout ?
Yes
Is the unnecessary objects zone full ?
Yes
Is the current object in the scenario ?
No
No
No

Figure 67: Diagram flow when there are extra objects in the scenario.

Having seen how the robot acts depending on the situation, we tested the different flows based on the scenario described in Figure 66. The first thing the robot does when noticing that it cannot move the cell phone because the *unnecessary objects* zone is full is to advise the user by saying *“It seems that the unnecessary objects zone has an object, may you take it away, please? Or if you prefer to take out the cell phone”*. Now, we are listing the three scenarios and their results:

1. The user does not help the robot. We have fixed two attempts for the user to help. The first timeout, the robot reminds the user by saying *“Remember to clear the unnecessary objects zone or to take out the cell phone to let me proceed”*. In case that there is a second timeout, the robot says *“It seems you cannot help me, and I cannot proceed. I am aborting...”*. In this case, the final state reminds as the initial one, as shown in Figure 66 since the robot could not find any alternative plan to reach the final goal.

2. The user takes out the candle, and the state of the world is updated as shown in Figure 68 (a). Once the robot detects it, it proceeds computing if it can move the cell phone to the *unnecessary objects* zone. As it can, it warns to the user by saying *“I am moving the cell phone”* and it grasps the cell phone (Figure 68 (b)) and poses it in the *unnecessary objects* zone (Figure 68 (c)). When the task is finished, the scenario looks as shown in Figure 68 (d), the final state. The task has been successfully achieved.

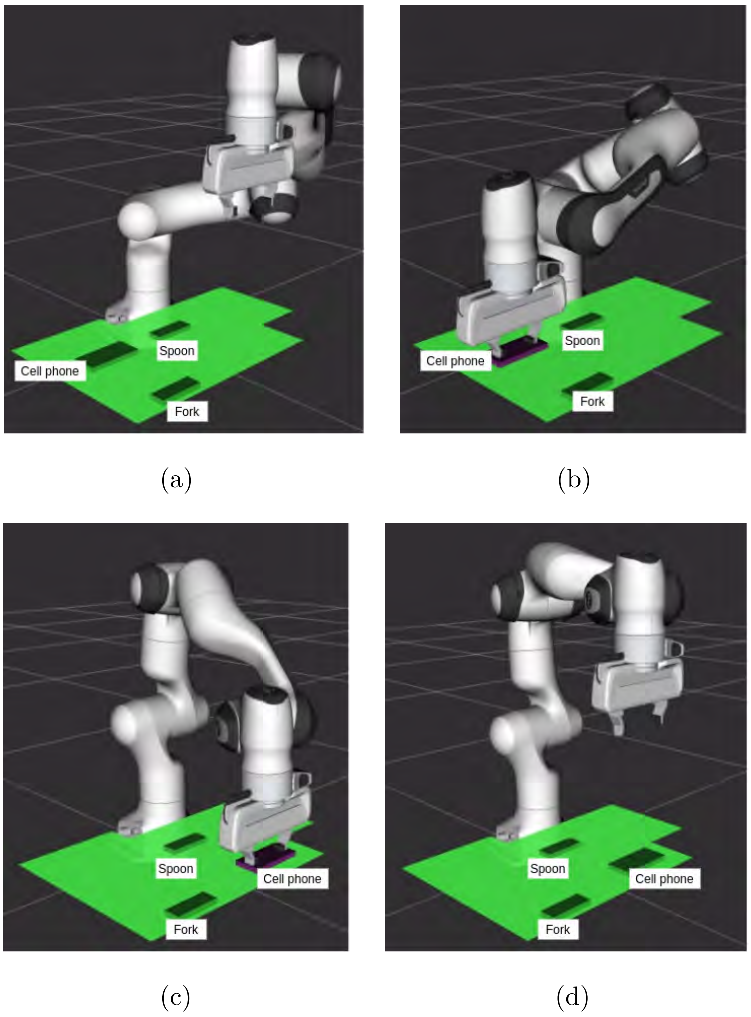


Figure 68: Performance of the robot when the user takes out the candle. (a) The user has just removed the candle and therefore, is no longer represented in the virtual world. (b) The robot grasps the cell phone. (c) The robot places the cell phone on the *unnecessary objects* zone. (d) Final state of the scenario.

3. The user takes out the cell phone, as shown in Figure 69. In this case, the robot does not need to do anything since the candle is already in the *unnecessary objects* zone, and the cell phone is removed from the working area.

As explained before, in the second and third situation the robot proceeds replanning what to do to achieve the goal and moves the objects to their places.

In conclusion, the robot is able to find a solution when there are objects in the table that are not needed and the user should help to remove the objects from the table, in case there are more than one.

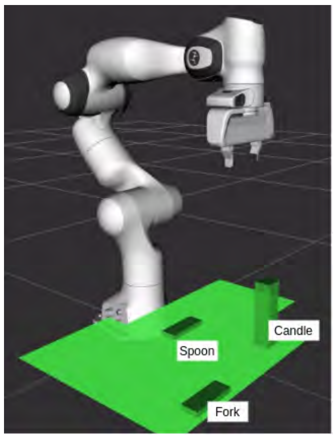


Figure 69: Scenario when the user removes the cell phone from the table, and therefore is no longer visible in the virtual world.

4.4.2 Some objects are missing

In this Section we will evaluate the robot behavior when some needed objects are not on the table. We are using the same scenario as shown in Figure 66, but in this case, we will need the plate, the knife, and the glass, which are not in the scenario. Therefore, there are the missing objects.

First of all, as seen in Section 4.4.1, the robot will start by putting away the extra objects (cell phone and candle). In this case, the user takes out the cell phone (see Figure 69).

Secondly, it evaluates if the necessary objects are present. As already described, the plate, the knife and the glass are missing, therefore the robot says *" I need a glass, a plate, a knife to proceed. Say 'Alexa, ask aided skill to wait', if you are going to bring me one, or, 'Alexa, ask aided skill to replan', if I have to work without them' "*. The user, again, has three options:

1. The user does not say anything. In this case, the robot cannot do anything else, so it finishes the task. Similarly to the previous case, the user has two opportunities to help out. If the first attempt fails, the robot reminds the user that it is waiting for the user's decision: *"I'm waiting for you to tell me if you want me to wait for you to bring me a glass, a plate, a knife or if you want me to work without them"*. If the second attempt fails again, the robot finishes the execution and tells to the user: *"I do not know what you want me to do. I am aborting..."*.
2. The user says "Wait". In this case the robot waits, and indicates to user that it receives the petition by saying *" Okay, waiting... Let me know when you are done saying: 'Alexa, ask aided skill I am done' "*. The user, then, has two options:
   (a) The user does not do anything. The robot provides a remember after some time to indicate that it is waiting. It says, *"I am waiting a glass, a plate, a knife to proceed"*. After a second timeout, the robot cannot proceed so it ends the task and indicates it by saying *"It seems you cannot help me, and I cannot proceed. I am aborting..."*.

(b) The user brings the missing objects and says, *"I am done"*. The robot indicates that it has received the message saying *"Okay, received, thank you"* and proceeds replanning and executing the task.

3. The user says *"Replan"*. In this case the robot plans with only the current objects and starts again. It indicates to user that it is thinking saying *"Let me see..."*.

A diagram flow of all the options explained above is shown in Figure 70.

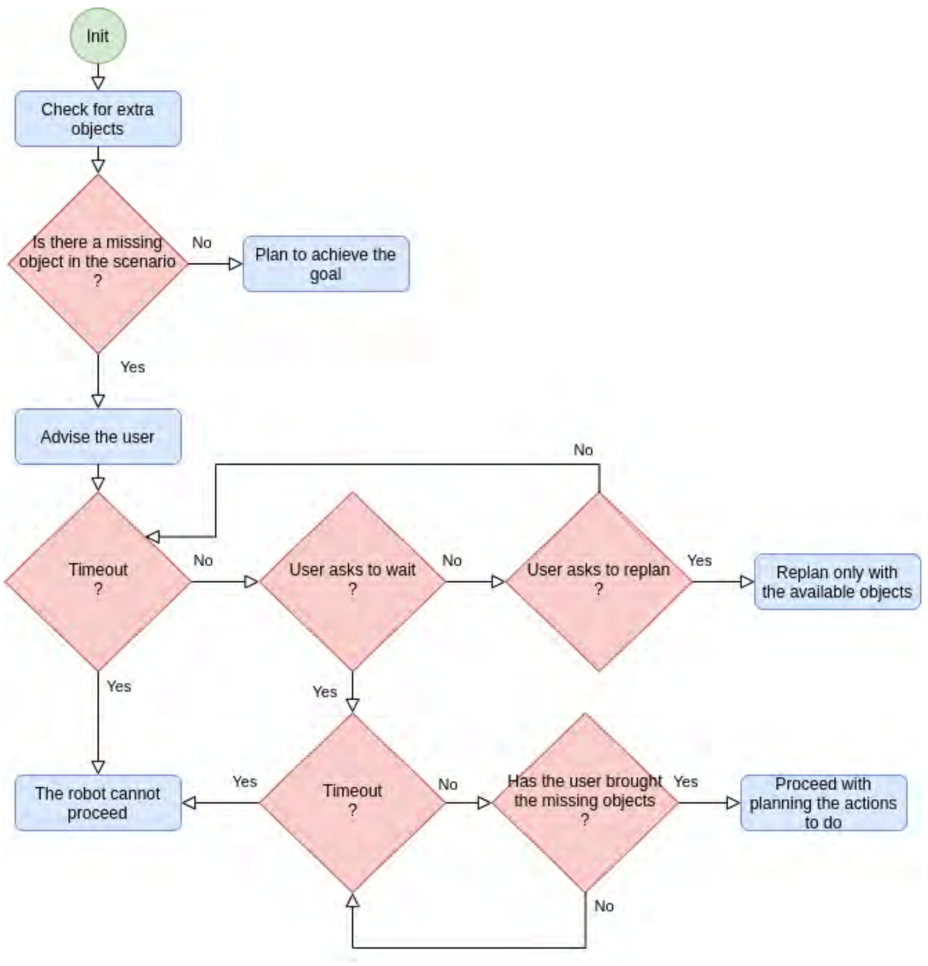
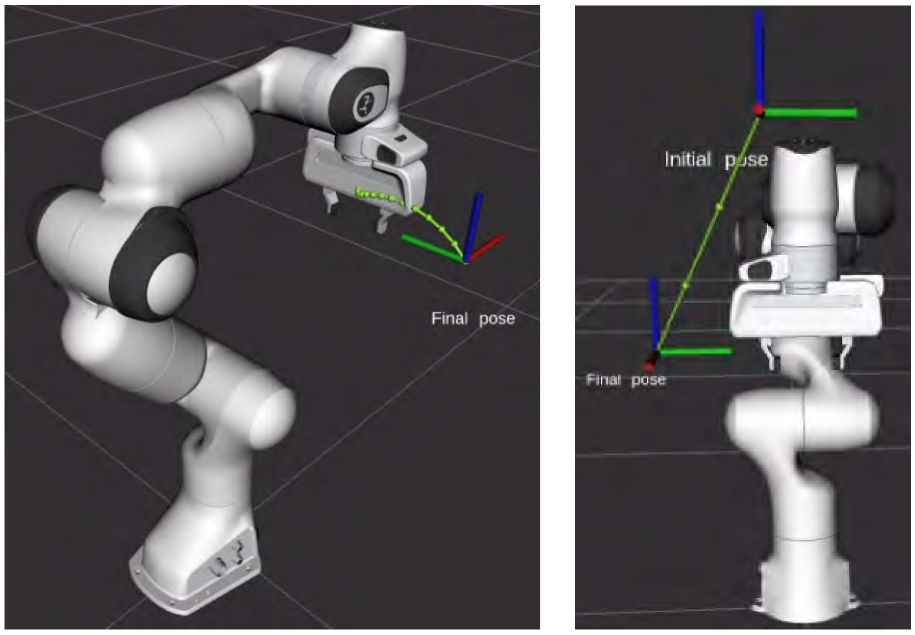


Figure 70: Diagram flow when there are missing objects in the scenario.

4.4.3 The robot cannot move an object

Every time the robot must perform a movement, it evaluates if it can do it or not. It evaluates both *grasp* and *place* actions. To do so, the robot must know which objects are on the table, the initial position it must go to, and the final pose to reach. We are going to separate the grasp and the place because, as we will see, they do not work in the same way.

When talking about grasping, the initial pose is the current one, whereas the final pose is the place where the object is (see Figure 71 (a) to see an example). MoveIt evaluates if the action can be successfully performed without colliding with other objects.



(a) (b)

Figure 71: Robot planned trajectory. (a) Only providing the final pose, and therefore, the initial pose is the current one. (b) Providing both, the initial pose and the final pose.

Whereas when we talk about placing the object, the initial pose is the position of the object, and the final pose is the surface where it must be on (see Figure 71 (b) to see an example). To evaluate if a *place* action is feasible, we attach a virtual object (see Figure 72), with the same characteristics as the object we want to move, to the end effector of the robot arm. We do this because it is not the same to evaluate the movement without the object than with the object. Imagine that the object to move is a bottle. The robot will grasp it by the neck, so now, the distance between the end of the arm and the area of collision is increased; the motion planner must consider not only the dimensions of the end effector, but also the bottle. All these dimensions will be, from now on, part of the collision area of the robot.

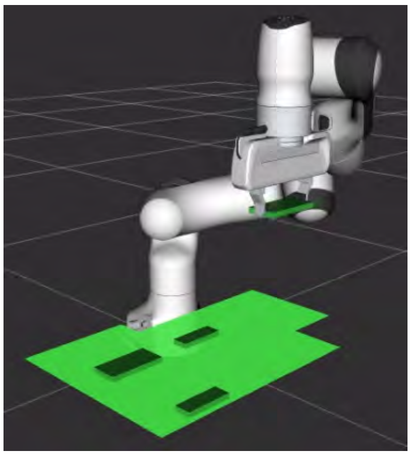


Figure 72: Arm with a copy of the cell phone in its end effector.

As part of the safety measures of the robot, once the robot has evaluated if it can do the movement, it processes, once again, the movement before actually execution the action for real. In Figure 73 (a) we can see this procedure, where the real robot and a watermarked copy of the robot are shown. The watermarked copy represents the motion of the robot. This watermarked copy appears with all the robots, although in Figure 73 the UR3 arm is shown.

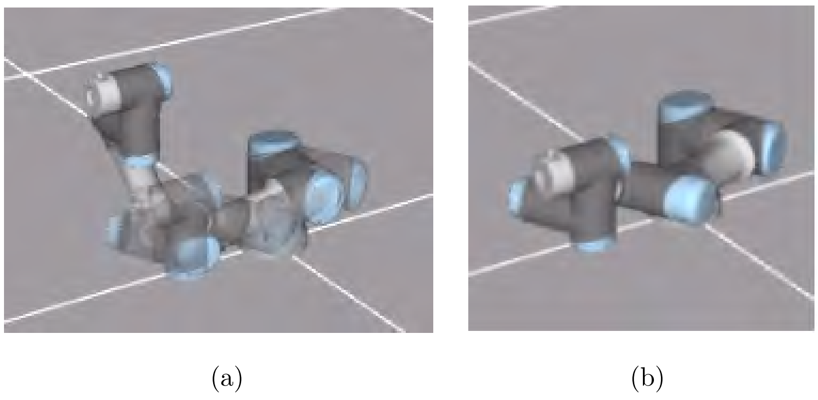


Figure 73: Robot planning and moving. (a) The robot is planning if it can move to the desired pose. (b) The robot has moved to the desired pose as there are no problems.

The first evaluation we have done is to see if the robot is able to dodge objects that are between the initial and end poses. Therefore, we add an object and run the planner. The result is shown in Figure 74, where the robot avoids it perfectly well. The green line represents the trajectory of the end effector, and it is one of the many trajectories that the robot could do to achieve the end pose.

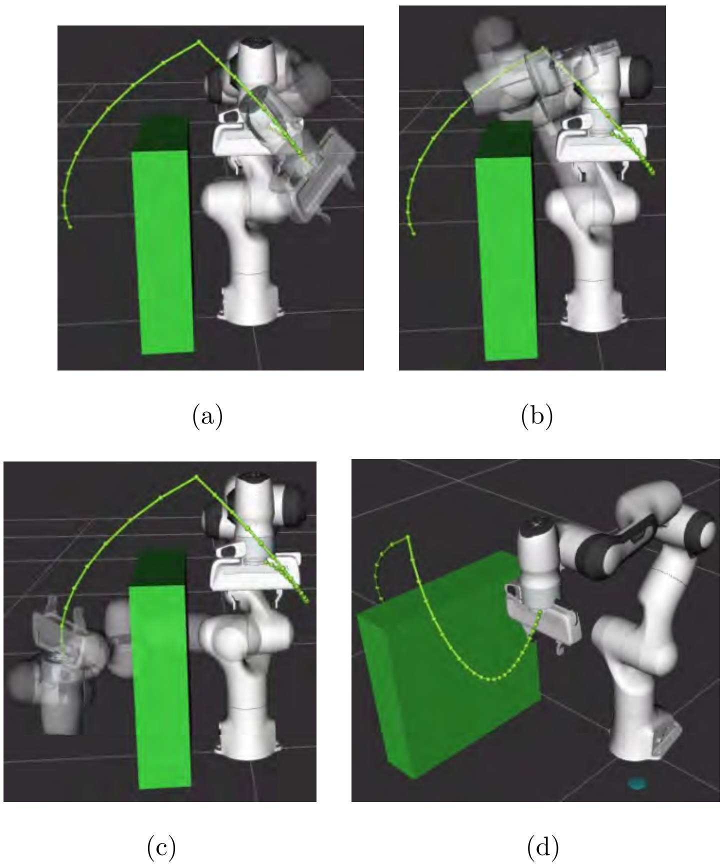


Figure 74: Arm avoiding a wall which is between the initial and end poses. (a) The robot starts the planning. (b) The robot is in the middle of the planning. (c) The robot reaches its final position while planning. (d) The trajectory that would have to do the robot to achieve the final pose.

Now we know what are the conditions to evaluate if the robot can grasp and place the object, and that the robot is able to avoid the objects, we can enumerate the multiple causes of why the robot could not perform the movement:

1. There are objects surrounding the one that the robot wants to grasp that collision with the end effector. It can also happen when placing the object.

2. When trying to make a move, the robot finds multiple objects that do not allow it to move.

3. The robot does not reach one or both poses specified (initial and/or final).

In these situations, the robot asks for the user's help. An example would be *" I cannot move the glass on the blue zone. Can you do it, please? When you have finished say, 'Alexa, ask aided skill I am done' "*. At this point, the user has two options:

1. The user does not do anything. In this case, the robot cannot do anything else, so it finishes the task. The user has two opportunities to help out: the

first one as a remember that it is waiting, and the second one to finish the program as the user has not helped. When the first timeout finishes, the robot says, *"Remember to move the glass on the blue zone"*. On the second timeout, it says *"It seems you cannot help me, and I cannot proceed. I am aborting..."* and finishes the program.

2. The user helps the robot moving the object to its position and says, "I am done". In this case the robot will proceed its execution.

We could add another option that would be that the user says that s/he cannot help and the robot, then, aborts. This option has not been implemented as we assume that s/he can do everything.

A diagram flow of all the options explained above is shown in Figure 75.

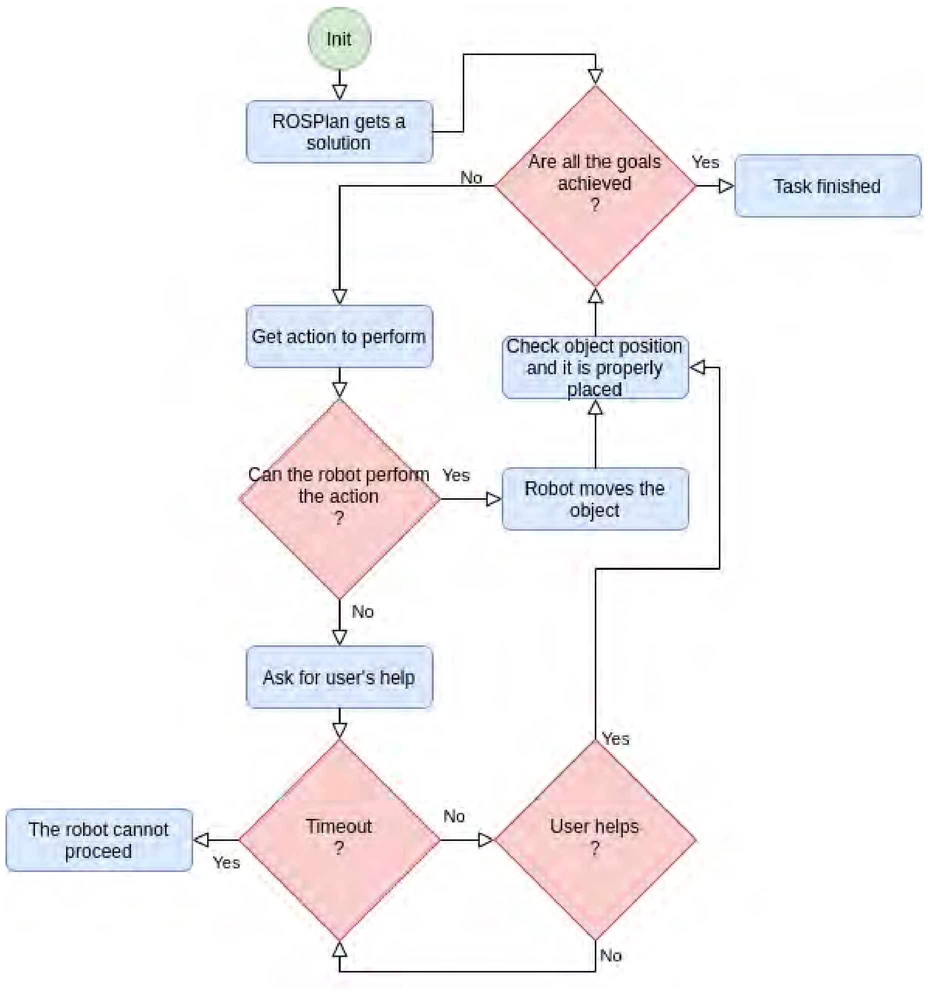


Figure 75: Diagram flow when the robot cannot move an object.

4.4.4 The object does not reach its final position

In this Section we evaluate what happens if the object grasped by the robot is not in the desired position. It can be for many causes, such as the user has moved it, the object fell from the gripper, or simply, because the robot did not actually grasp the object. Before talking about what does the robot do in this situation, we need to talk about what we refer to when saying that an object is not at its place.

Each object and zone has two specifications: dimension and location, which are shown in Figure 76. The locations belongs to the midpoint of the object or zone, and has an x, y, and z values, referenced from the base of the robot, whereas the dimensions represent the size of the object or zone, and also have an x, y, and z values. We are only working on a surface, so the z values of location and dimension are not used to know if the object is on the zone.

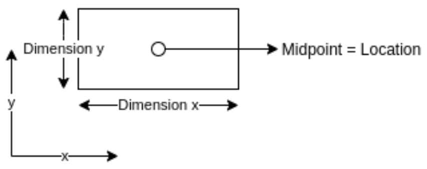


Figure 76: Objects and zones specifications.

Once we know the specifications, we can now show the computation done to know if the object is inside a zone or not. In Figure 77 we can see an example where the object is inside the zone. Also, the formulas applied for each zone limitation to compare with the limits of the object are represented. In case that the object is on top of another object, the formulas are still valid. As we can see, the object border cannot coincide with the limits of the zone. With this we want to make sure that the object does not hinder another item that will be placed in the side zone. If so, it will not be considered a valid position.

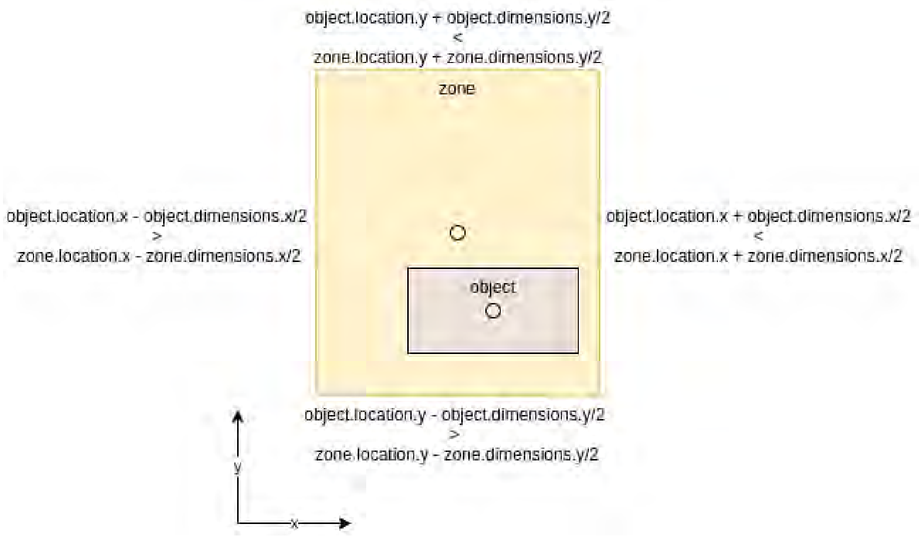

object.location.y + object.dimensions.y/2
<
zone.location.y + zone.dimensions.y/2
zone
object.location.x - object.dimensions.x/2
>
zone.location.x - zone.dimensions.x/2
object.location.x + object.dimensions.x/2
<
zone.location.x + zone.dimensions.x/2
object
object.location.y - object.dimensions.y/2
>
zone.location.y - zone.dimensions.y/2
y
x

Figure 77: Formulas applied to know if the object is inside the zone limits.

We thus say that an object has not reached its final position if the object is not within a given zone. Let us go back to analyzing what happens if during the execution of a plan, the object does not reach the expected position, i.e. it is not correctly placed at its final position.

As described in Section 3.5, the *Vision* module does not perform object tracking. Therefore, while the robot executes an action, the system stops listening the *Vision* messages regarding the positions of the objects because we cannot rely on the information provided. Hence, the internal state of the world and the real world, might temporally mismatch.

The system, so, only relies on its internal world during the execution of an action. Once the robot finishes, the system verifies with the *Vision* system the current state of the world. It is only then, that it can realize if the objects are at their expected locations or not. Figure 78 illustrates a situation where the robot believes it has properly grasped an object (the spoon). It is only at the end of the execution that the system realizes that it has not. At this point, it informs the user by saying *"It seems that the object is not in its place, I will do it again"*. Then, it finishes the current program, starts a new one, and plans again.

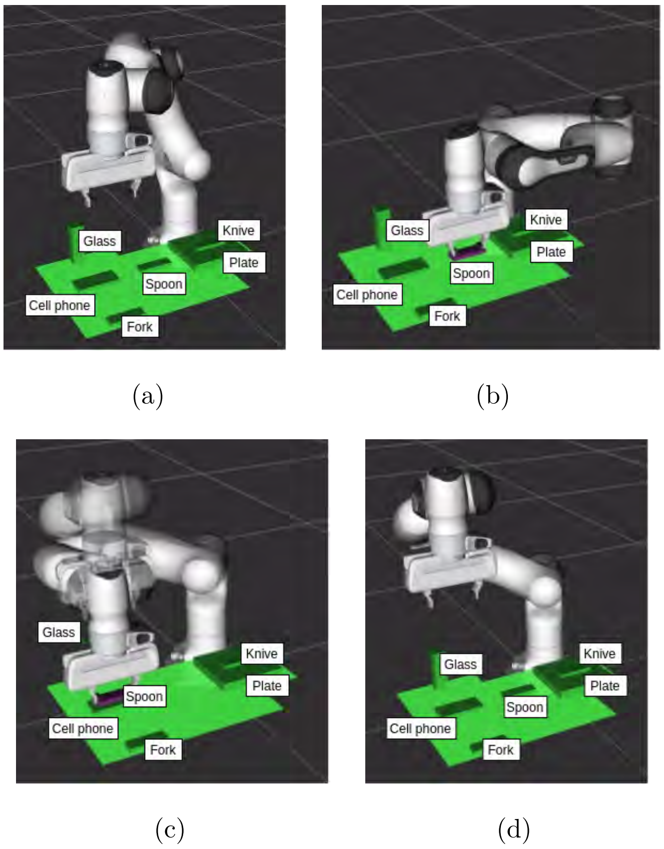
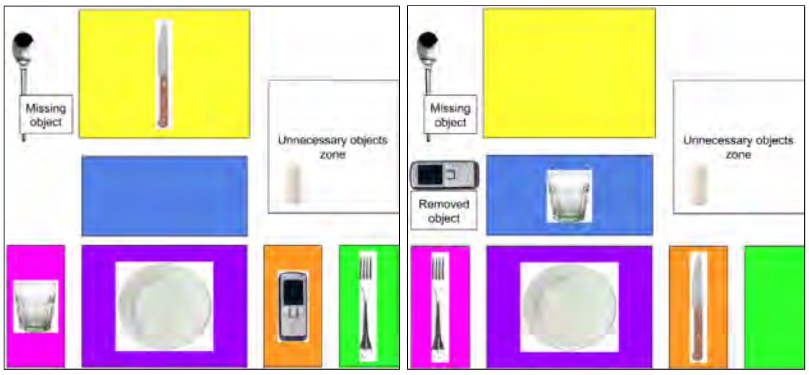


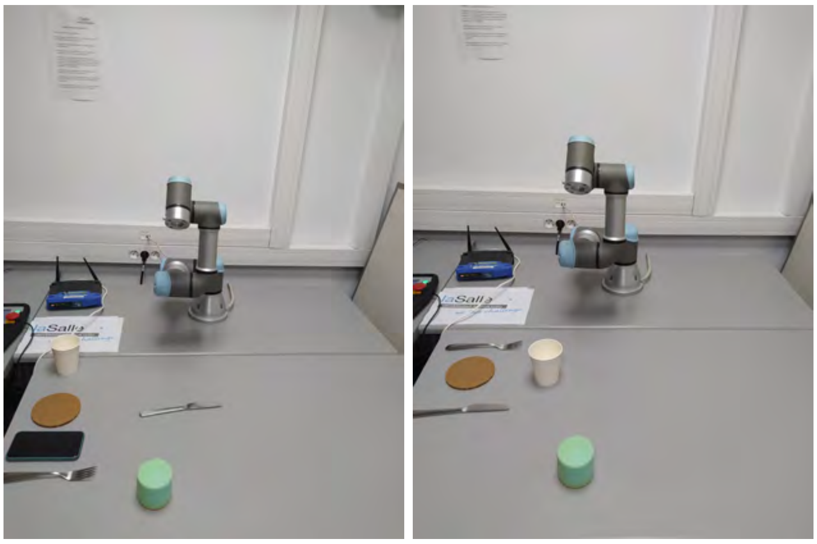
Figure 78: Robot beliefs when moving an object. (a) Initial scenario. (b) The robot believes it is grasping the spoon. (c) The robot believes it is placing the spoon on the cell phone. (d) The real state of the world after the performance of the robot.

4.5 Setting the table - a full task

In this section we show the full robot's task when setting the table for one. The scenario is the one shown in Figure 79 (a), where there are two extra objects (the candle and the cell phone) and a missing one (the spoon). The final goal we will achieve is the one shown in Figure 79 (b), where we will remove the cell phone from the scenario and we will not bring the spoon.



(a) (b)



(a) (b)

Figure 79: (a) Initial and (b) Final states of the world to perform a full setting the table task with the robot.

It is important to mention that the *Vision* module has not been used as the focus of this Section is to test the system combining different situations described in Section 4.4. Hence, the different scenarios that the *Supervisor* receives are hard-coded and are only send once. In Figure 80 we can see the initial scenario, from the robot believes.

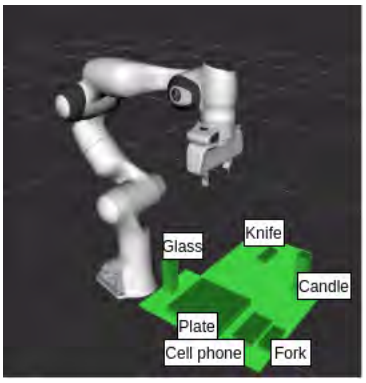


Figure 80: Initial state of the system to test the full system.

The first thing the system does is to evaluate if there are extra objects, which in this case there are two: the candle and the cell phone. For this reason, the robot says:

"The cellphone, the candle are not needed, taking them out."

Although, when trying to move the cell phone to the *unnecessary objects* zone, the robot detects that there is a candle there, and asks for user's help saying:

"It seems that the unnecessary objects zone has an object, may you take it away, please? Or if you prefer to take out the cell phone."

In this case the user takes out the cell phone, as shows Figure 81. The robot indicates to the human that it can proceed saying:

"All the extra objects are out."

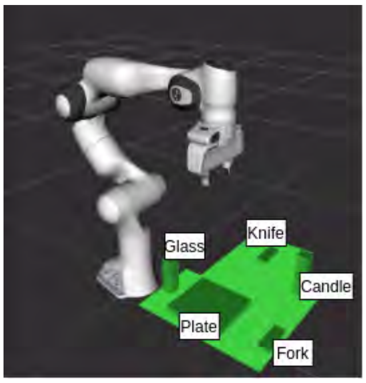


Figure 81: The cell phone is removed from the scenario.

Once the extra objects are out or on the *unnecessary objects* zone, the system computes if any object is missing, which in this case the spoon is not on the table. Therefore, the robot asks the human what s/he wants it to do by saying:

"I need a spoon to proceed. Say 'Alexa, ask aided skill to wait', if you are going to bring me one, or, 'Alexa, ask aided skill to replan', if I have to work without it."

In this case the user asks to replan:

"Alexa, ask aided skill to replan."

Then, the robot evaluates the initial state, which is the goal, and builds a plan. To indicate this to the user, says:

"Let me see..."

And when starts planning says:

"Replan started."

In this case, the first step of the plan is:

1. Move *glass1*, which is on *pink* zone, to *blue* zone

The *Motion Planner* computes if the goals are reachable, but in this case, they are not, so the robot asks for the user's help saying:

"I cannot move the glass on the blue zone. Can you do it, please? When you have finished say, 'Alexa, ask aided skill I am done'."

The user moves the glass to that zone and indicates it to the system.

"Alexa, ask aided skill I am done."

"Okay, thank you."

The result is shown in Figure 82.

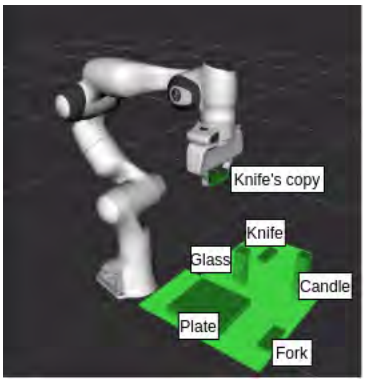


Figure 82: Glass on the *blue* zone.

The second step to achieve the goal is the following:

2. Move *knife1*, which is on *yellow* zone, to *orange* zone

This time the robot is capable to perform the action, and indicates it, while moving the knife, by saying:

"I am moving the knife."

The result is shown in Figure 83.

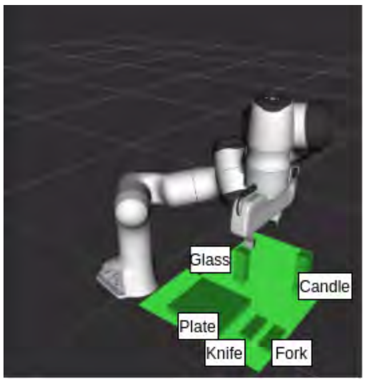


Figure 83: Knife on the *orange* zone.

The third and final step to achieve the goal is:

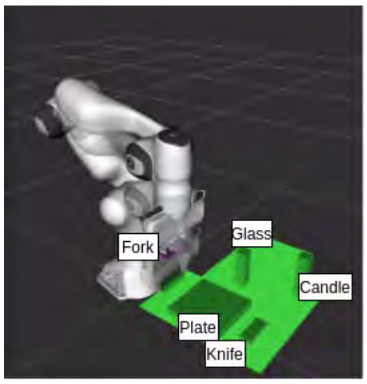
3. Move *fork1*, which is on *green* zone, to *pink* zone

The robot says it can do it by saying:

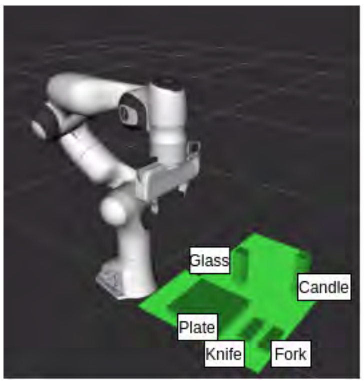
"I am moving the fork."

In this case, however, it believes that it has grasped the fork, as shows Figure 84 (a). Although, once the action is performed, it realizes that it did not grasp the object (shown in Figure 84 (b)) and, therefore, indicates to the user that it is going to do it again by saying:

"It seems that the object is not in its place, I will do it again."



(a)



(b)

Figure 84: Robot's beliefs and the real state of the world. (a) Moving the fork on the *pink* zone. (b) Received a *Vision* message and the fork has not moved to the *pink* zone.

It performs again the movements, grasping the fork this time, and saying again:

"I am moving the fork."

The result is shown in Figure 85.

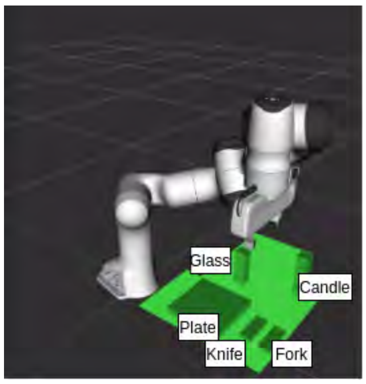


Figure 85: After another performance of the robot, the fork is on the *pink* zone, and the program is finished.

The robot realizes that there are no more actions to perform and, hence, it indicates the human so by saying:

"I have finished."

5 Time costs

In this Section we indicate the time costs of this TFM.

- **State-of-the-art study**: 20 hours.
- **Follow-up meetings**: 40 hours.
- **Planning techniques**: 250 hours.
- ***Motion Planner* module integration**: 230 hours.
- ***Verbal communication* module integration**: 150 hours.
- ***Vision* module integration**: 100 hours.
- **System tests and bug fixes**: 100 hours.
- **Document**: 220 hours.

Hence, the total of hours invested in this project is of 1110 hours. In Figure 86 a graphic with the percentages of time invested in each task with respect to the total is shown.

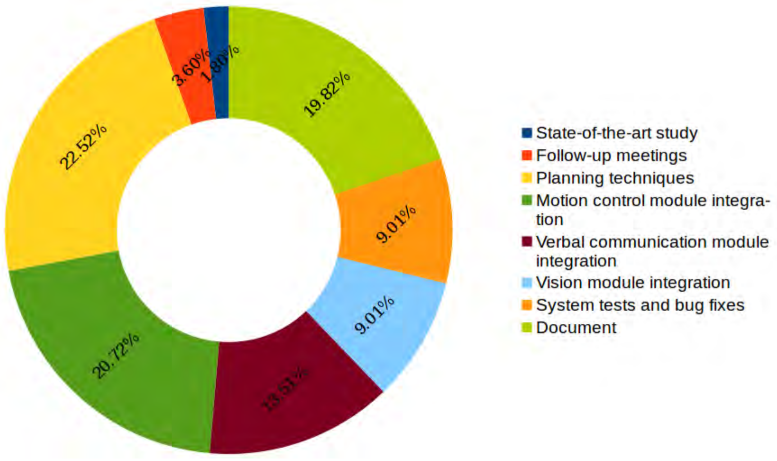


Figure 86: Time costs.

6 Conclusion and future lines

We have presented our work towards a collaborative robot that supports instrumental daily life tasks. Although the Corona Virus Disease 2019 (COVID-19) has limited our evaluation of the system, we are proud of the results achieved, as we have a robust and functional system that includes:

- The *Task Planner* that finds an optimal solution for different situations.
- The *Motion Planner* that plans a collision-free trajectories.
- The *Vision* module capable of detecting objects instead of QR codes. Furthermore, we have evaluated the reliability of the detections and its consistency.
- The *Verbal Communication*, capable of recognizing user petitions thanks to the different sentences that the testers came up with. It also provides feedback of what the robot is doing.
- The *Supervisor* module capable of orchestrating the different modules of the system and adapting its behavior to different situations:
  - when there are more objects than needed on the table,
  - when objects are missing,
  - when the *Motion Planner* cannot find a collision-free trajectory and, therefore, the robot cannot move the object,
  - when the object is not correctly placed.

Furthermore, the URL-MONITOR project can still be improved in an easy and fast way, as it is implemented with ROS.

As for further directions, a number of improvements and new features can be added to the current project. As already said in Section 2, in the SOCRATES project [12], they detect any safety concern. It would be interesting to add this in our project too as the robot has to work with a human and it should know what s/he is doing to change its behavior accordingly.

Right now, the system treats two objects of the same type (spoon, to illustrate) as if they were two items completely different and each one has to be placed in a specific zone. Imagine that we have $spoon1$ on the zone which it has to be placed $spoon2$. The robot would take out $spoon1$ to place $spoon2$, when the result is exactly the same, as the goal is that it must be a spoon there. The main disadvantage of this is that the system executes unnecessary actions, hence it takes more time to achieve the goal. To solve this, we should get more in deep with ROSPlan.

Following with the previous example, another important aspect of having two equal spoons is that we cannot assure that, in different frames of the *Vision* module recognition, the $spoon1$ of the first frame is the same $spoon1$ of the second frame, as the module does not do tracking of the objects. Adding object tracking we could also know if an object is below another one if a stack operation is performed and the *Vision* module does not detect it. To illustrate, the mobile can be placed on the top of a fork and then it does not recognize the fork. With the tracking, the system would know that the fork is under the mobile phone.

We could also work with the depth sensors of the camera to know the height of the objects, and color detection to detect the zones and to know on which zone is every object. All this information is hard-coded in the current system.

Moreover, if we want more interaction between the robot and the user, we should be more specifics with the objects position, and it is more important if there are objects of the same type. Going back to the previous examples of the spoons, imagine that the robot wants the user to pick up the spoon of his/her left. If the robot only says to the user to move the spoon, there will not be enough information since the user has two possibilities. We could differentiate them if the spoons had different colors; hence the robot could ask the user to pick up the blue one. Work related in this field can be found in [39]. We have also talked about the importance of the non-verbal communication. We could also add gesture recognition to facilitate the HRI. If the user is now the one who wants an object and the spoons are equals, having only the verbal communication s/he would have to indicate what objects are next to it or if it is on top of an specific zone, whereas with gestures the robot could infer which object the user is referring to without ambiguity.

In Section 4.4.3 we have talk about a scenario where the user cannot help the robot moving an object. We could also add this command for those users that have physical limitations. In this case, neither the robot nor the user could fully perform a given action. One potential solution could be to split the action in actions that are affordable for both agents. For instance, one agent could reach the object and hand it to the other agent, who would then place it at the final location. By doing so, we would achieve a collaborative action.

The gripper used could also be improved. In our case we use the one shown in Figure 87. Such gripper type has difficulties when it comes to picking up objects like a plate or a cell phone. For this reason, we are developing a new one based on electromagnetic fields to easily manipulate different types of objects with no clear grasping points.

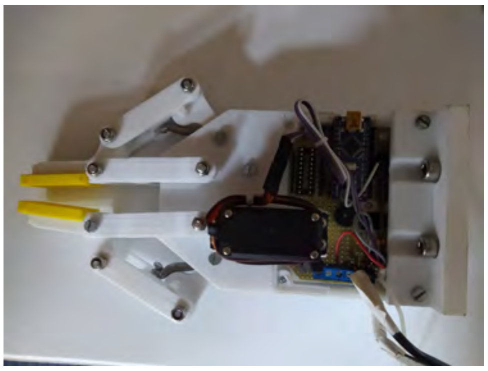


Figure 87: Gripper.

To conclude, as it uses Alexa, it can easily connect to smart devices in a digital home with WiFi, Bluetooth and Zigbee. Thanks to it, the user could control the temperature, the lights, among others, of the house. Other skills could be created such as CARU *cares* [2], or CO-TRAIN [4], already mentioned in Section 2.

7 References

[1] *AAL official page*. URL http://www.aal-europe.eu/.

[2] *CARU cares*. URL http://carucares.com/en/home-en/.

[3] *ChefMySelf*. URL http://www.aal-europe.eu/projects/chefmyself/.

[4] *CO-TRAIN*. URL https://www.cotrain.eu/.

[5] *FEARLESS*. URL https://cogvis.ai/.

[6] *Gazebo*. URL http://gazebosim.org/.

[7] *HOPE*. URL http://www.hope-project.eu/.

[8] *ngrok*. URL https://ngrok.com/.

[9] *ROS*. URL https://www.ros.org/.

[10] *rviz*. URL http://wiki.ros.org/rviz.

[11] *SmartHeat*. URL http://www.smartheat-aal.eu/.

[12] *SOCRATES*. URL http://www.socrates-project.eu/.

[13] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. 1977. ISBN 0-19-501919-9.

[14] A. Andriella, J. Lobo-Prat, C. Torras, and G. Alenyà. *Robot interaction adaptation for healthcare assistance*, . URL http://www.iri.upc.edu/files/scidoc/2196-Robot-interaction-adaptation-for-healthcare-assistance.pdf.

[15] A. Andriella, C. Torras, and G. Alenyà. *Short-Term Human-Robot Interaction Adaptability in Real-world Environments*, . URL http://www.iri.upc.edu/files/scidoc/2243-Short-term-human-robot-interaction-adaptability-in-real-world-environments.pdf.

[16] A. Andriella, C. Torras, and G. Alenyà. *Cognitive System Framework for Brain-Training Exercise based on Human-Robot Interaction*, . URL http://www.iri.upc.edu/files/scidoc/2318-Cognitive-System-Framework-for-Brain-Training-Exercise-Based-on-Human-Robot-Interaction.pdf.

[17] U. R. A/S. *Universal Robots ExternalControl URCap*, 2020. URL https://github.com/UniversalRobots/Universal_Robots_ExternalControl_URCap.

[18] U. R. A/S. *Universal Robots ROS Driver*, 2020. URL https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.

[19] AssistedLivingToday. *What Are IADLs or the Instrumental Activities of Daily Living?*, 2019. URL https://assistedlivingtoday.com/blog/what-are-iadls/.

[20] C. Bartneck, T. Belpaeme, F. Eyssel, T. Kanda, M. Keijsers, and S. Sabanovic. *Human-Robot Interaction: An Introduction*. 02 2020. ISBN 9781108735407. doi: 10.1017/9781108676649.

[21] K. Becker. *Artificial Intelligence Planning with STRIPS, A Gentle Introduction*, 2015. URL http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/.

[22] K. Becker. *strips*, 2018. URL https://github.com/primaryobjects/strips.

[23] M. Cashmore, D. Magazzeni, S. Krivic, G. Canal, D. Buksz, A. Collins, B. Krarup, I. Moraru, P. Zehtabi, and O. Lima. *ROSPlan*. URL https://kcl-planning.github.io/ROSPlan/.

[24] R. Cecilia. *Detecció d'objectes amb xarxes neuronals per a la comprensió de l'entorn en robòtica col·laborativa*, 2020.

[25] F. EMIKA. *Introducing the Franka Emika Robot*. URL https://www.franka.de/.

[26] FANUC. *M-2000iA/2300*. URL https://www.fanuc.eu/es/es/robots/p%c3%a1gina-filtro-robots/serie-m-2000/m-2000ia-2300.

[27] N. C. for Health Statistics (https://www.cdc.gov/nchs/nhis/shs/tables.htm). *Table P-3. Persons having limitation in activities of daily living and instrumental activities of daily living among persons aged 18 and over, by selected characteristics: United States, 2018*, 2018. URL https://ftp.cdc.gov/pub/Health_Statistics/NCHS/NHIS/SHS/2018_SHS_Table_P-3.pdf.

[28] A. Garrell. *Experimentation on HRI*. Oral presentation at IRI's summer school ROBOTICS & AI, 29-30 June and 1 July 2020. URL https://www.iri.upc.edu/workshops/RoboticsAISummerSchool2020/.

[29] E. Golobardes and A. Orriol. *Intel·ligència Artificial*, 2008.

[30] M. Heerink. *Assessing acceptance of assistive social robots by aging adults*. PhD thesis, 11 2010. URL https://www.researchgate.net/publication/254852003_Assessing_acceptance_of_assistive_social_robots_by_aging_adults.

[31] U. KBO. *ChefMyself Conclusies*. URL https://www.uniekbo.nl/aal/chefmyself/?page=conclusies.

[32] R. Matheson. *Showing robots how to do your chores*, 03 2020. URL https://news.mit.edu/2020/showing-robots-learn-chores-0306.

[33] MathWorks. *Augment Images for Deep Learning Workflows Using Image Processing Toolbox*. URL https://es.mathworks.com/help/deeplearning/ug/image-augmentation-using-image-processing-toolbox.html?s_tid=srchtitle#AugmentImagesForDeepLearningWorkflowsExample-1.

[34] M. Mori. *The Uncanny Valley*, 1970. URL https://web.ics.purdue.edu/~drkelly/MoriTheUncannyValley1970.pdf.

[35] J. Osorio, G. Tulio, and G. Motoa. Planificación jerárquica de la producción en un job shop flexible. hierarchical production planning in the flexible job shop. *Rev. Fac. Ing. Univ. Antioquia*, 44:158–171, 07 2008. URL https://www.researchgate.net/publication/313407409_Planificacion_jerarquica_de_la_produccion_en_un_job_shop_flexible_Hierarchical_production_planning_in_the_flexible_job_shop.

[36] D. Pellier and H. Fiorino. Pddl4j: a planning domain description library for java, journal of experimental theoretical artificial intelligence. *Journal of Experimental Theoretical Artificial Intelligence, DOI: 10.1080/0952813X.2017.1409278*, 30:1:143–176, 2017. URL https://doi.org/10.1080/0952813X.2017.1409278. GitHub: https://github.com/pellierd/pddl4j.

[37] F. Portugal. *ChefMyself – Beginning the trials in two countries*. URL https://www.aicos.fraunhofer.pt/en/news_and_events_aicos/news_archive/older_archive/chefmyself-_-beginning-the-trials-in-two-countries.html.

[38] U. Robots. *Cobots from Universal Robots*. URL https://www.universal-robots.com/.

[39] D. Roy. Learning visually-grounded words and syntax for a scene description task. *Computer Speech Language*, 16:353–385, 07 2002. doi: 10.1016/S0885-2308(02)00024-4.

[40] A. Shah, S. Li, and J. Shah. *Planning With Uncertain Specifications (PUnS)*, 06 2019. URL https://www.researchgate.net/publication/333671681_Planning_With_Uncertain_Specifications_PUnS.

[41] I. A. Sucan and S. Chitta. *MoveIt*. URL https://moveit.ros.org/.

[42] U. R. Support. Overview of client interfaces. 2020. URL https://www.universal-robots.com/articles/ur-articles/overview-of-client-interfaces/.

[43] U. R. Support. Dashboard server cb-series, port 29999. 2020. URL https://www.universal-robots.com/articles/ur-articles/dashboard-server-cb-series-port-29999/.

[44] U. R. Support. Remote control via tcp/ip. 2020. URL https://www.universal-robots.com/articles/ur-articles/remote-control-via-tcpip/.

[45] U. R. Support. Real-time data exchange (rtde) guide. 2020. URL https://www.universal-robots.com/articles/ur-articles/real-time-data-exchange-rtde-guide/.

[46] W. Tansey. *strips*, 2011. URL https://github.com/tansey/strips.

[47] thorsten gehrig. *alexa-remote-control*. URL https://github.com/thorsten-gehrig/alexa-remote-control.

[48] I. Tischer and A. Carrión García. La planificación jerárquica y su aplicación a la cosecha de la caña de azúcar en colombia. *Ciencia y Tecnología*, 4:42–52, 01 2003. doi: 10.25100/iyc.v4i2.2312. URL https://www.researchgate.net/publication/266878034_La_Planificacion_Jerarquica_y_su_Aplicacion_a_la_Cosecha_de_la_Cana_de_Azucar_en_Colombia.

[49] K. Tsiakas, V. Karkaletsis, and F. Makedon. A taxonomy in robot-assisted training: Current trends, needs and challenges. pages 208–213, 06 2018. doi: 10.1145/3197768.3197787. URL https://www.researchgate.net/publication/326026076_A_Taxonomy_in_Robot-Assisted_Training_Current_Trends_Needs_and_Challenges.

[50] Wikipedia. *Sussman anomaly*, 2019. URL https://en.wikipedia.org/wiki/Sussman_anomaly.

[51] Wikipedia. *Automated planning and scheduling*, 2020. URL https://en.wikipedia.org/wiki/Automated_planning_and_scheduling.

8 Appendix

8.1 PDDL files used for test

In this Section we show the files used to test the different planners.

8.1.1 domain.pddl

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block table - obj)
  (:predicates (on ?x - block ?y - obj)
               (clear ?x - obj)
               )
  (:action move
     :parameters (?b - block ?t1 - table ?t2 - table)
     :precondition (and (on ?b ?t1) (clear ?t2) (clear ?b))
     :effect (and (on ?b ?t2) (not (on ?b ?t1)) (clear ?t1) (not
     (clear ?t2)))
  )
  (:action move2
     :parameters (?b - block ?a - block ?c - block)
     :precondition (and (on ?c ?a) (clear ?b) (clear ?c))
     :effect (and (on ?c ?b) (not (on ?c ?a)) (clear ?a) (clear ?c)
     (not (clear ?b)))
  )
  (:action stack
     :parameters (?a - block ?b - block ?t1 - table)
     :precondition (and (clear ?a) (clear ?b) (on ?a ?t1))
     :effect (and (on ?a ?b) (not (on ?a ?t1)) (clear ?t1) (not
     (clear ?b)))
  )
  (:action unstack
     :parameters (?a - block ?b - block ?t1 - table)
     :precondition (and (clear ?a) (on ?a ?b) (clear ?t1))
     :effect (and (on ?a ?t1) (not (on ?a ?b)) (clear ?b) (not
     (clear ?t1)))
  )
)
```

8.1.2 table_for_one_problem.pddl

```
(define (problem table_for_one)
  (:domain BLOCKS)
  (:objects blue pink purple yellow green orange
            unnecessary_objects - table
            glass1 plate1 spoon1 knive1 fork1 cellphone1 - block)
  (:INIT
      (on glass1 pink)
      (on cellphone1 purple)
      (on spoon1 blue)
      (on plate1 yellow)
      (on knive1 plate1)
      (on fork1 green)
      (clear glass1)
      (clear cellphone1)
      (clear orange)
      (clear fork1)
      (clear spoon1)
      (clear knive1)
      (clear unnecessary_objects)

  )
  (:goal (and (on fork1 pink) (on plate1 purple) (on knive1 orange)
  (on spoon1 green) (on glass1 blue) (clear yellow)
  (on cellphone1 unnecessary_objects)))
)
```

8.1.3 romantic_dinner_problem.pddl

```
(define (problem dinner_problem)
  (:domain BLOCKS)
  (:objects blue1 blue2 red1 red2 green1 green2 orange1 orange2
            pink1 pink2 purple1 purple2 black1 black2 yellow1
            yellow2 unnecessary_objects - table
            glass1 glass2 plate1 plate2 breadplate1 breadplate2
            spoon1 spoon2 knive1 knive2 fork1 fork2 minispoon1
            minispoon2 candle1 cellphone1 - block)
  (:INIT
      (on fork1 green2)
      (on glass2 orange2)
      (on plate1 purple2)
      (on fork2 plate1)
      (on knive1 pink2)
      (on spoon1 black2)
      (on candle1 blue2)
      (on knive2 red2)
      (on breadplate1 yellow1)
      (on breadplate2 yellow2)
      (on plate2 red1)
      (on minispoon1 blue1)
      (on glass1 black1)
      (on minispoon2 pink1)
      (on spoon2 purple1)
      (on cellphone1 green1)
      (clear unnecessary_objects)
      (clear cellphone1)
      (clear orange1)
      (clear spoon2)
      (clear minispoon2)
      (clear glass1)
      (clear minispoon1)
      (clear plate2)
      (clear breadplate2)
      (clear breadplate1)
      (clear candle1)
      (clear knive2)
      (clear spoon1)
      (clear fork1)
      (clear glass2)
      (clear fork2)
      (clear knive1)


  )
  (:goal (and (on glass1 blue1) (on plate1 purple1)
         (on knive1 orange1) (on fork1 pink1) (on spoon1 green1)
         (on minispoon1 black1) (on breadplate1 red1)
         (on glass2 blue2) (on plate2 purple2) (on knive2 orange2)
         (on fork2 pink2) (on spoon2 green2) (on minispoon2 black2)
         (on breadplate2 red2) (on candle1 yellow1)
         (on cellphone1 unnecessary_objects)
         )
  )
)
```

8.2 Communication

In this Section we show the messages created to communicate the different nodes of the project.

8.2.1 std_msgs/String

It is the message send by the url_project_brain_to_speech topic. It contains:

```
string data
```

8.2.2 url_project/speech_to_brain_msg

It is the message send by the url_project_speech_to_brain topic. It contains:

```
uint8 request
url_project/speech_new_goal_description new_goal_rosplan
  std_msgs/String object_1
    string data
  std_msgs/String object_2
    string data
  std_msgs/String action
    string data
std_msgs/String set_table_file
  string data
```

8.2.3 url_project/grasp_description

It is the message send by the url_project_grasp topic. It contains:

```
std_msgs/String support_surface_name
  string data
std_msgs/String object_name
  string data
moveit_msgs/Grasp[] grasp_object
  string id
  trajectory_msgs/JointTrajectory pre_grasp_posture
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
      float64[] positions
      float64[] velocities
      float64[] accelerations
      float64[] effort
      duration time_from_start
  trajectory_msgs/JointTrajectory grasp_posture
    std_msgs/Header header
```

```
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
      float64[] positions
      float64[] velocities
      float64[] accelerations
      float64[] effort
      duration time_from_start
  geometry_msgs/PoseStamped grasp_pose
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    geometry_msgs/Pose pose
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
  float64 grasp_quality
  moveit_msgs/GripperTranslation pre_grasp_approach
    geometry_msgs/Vector3Stamped direction
      std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
      geometry_msgs/Vector3 vector
        float64 x
        float64 y
        float64 z
    float32 desired_distance
    float32 min_distance
  moveit_msgs/GripperTranslation post_grasp_retreat
    geometry_msgs/Vector3Stamped direction
      std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
      geometry_msgs/Vector3 vector
        float64 x
        float64 y
        float64 z
    float32 desired_distance
    float32 min_distance
  moveit_msgs/GripperTranslation post_place_retreat
    geometry_msgs/Vector3Stamped direction
      std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
      geometry_msgs/Vector3 vector
        float64 x
        float64 y
        float64 z
    float32 desired_distance
    float32 min_distance
  float32 max_contact_force
  string[] allowed_touch_objects
```

8.2.4 url_project/objects_description

It is the message send by the url_project_objects_description topic. It contains:

```
moveit_msgs/AttachedCollisionObject[] static_objects
  string link_name
  moveit_msgs/CollisionObject object
    byte ADD=0
    byte REMOVE=1
    byte APPEND=2
    byte MOVE=3
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string id
    object_recognition_msgs/ObjectType type
      string key
      string db
    shape_msgs/SolidPrimitive[] primitives
      uint8 BOX=1
      uint8 SPHERE=2
      uint8 CYLINDER=3
      uint8 CONE=4
      uint8 BOX_X=0
      uint8 BOX_Y=1
      uint8 BOX_Z=2
      uint8 SPHERE_RADIUS=0
      uint8 CYLINDER_HEIGHT=0
      uint8 CYLINDER_RADIUS=1
      uint8 CONE_HEIGHT=0
      uint8 CONE_RADIUS=1
      uint8 type
      float64[] dimensions
    geometry_msgs/Pose[] primitive_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    shape_msgs/Mesh[] meshes
      shape_msgs/MeshTriangle[] triangles
        uint32[3] vertex_indices
      geometry_msgs/Point[] vertices
        float64 x
        float64 y
        float64 z
    geometry_msgs/Pose[] mesh_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    shape_msgs/Plane[] planes
      float64[4] coef
    geometry_msgs/Pose[] plane_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    string[] subframe_names
    geometry_msgs/Pose[] subframe_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    byte operation
  string[] touch_links
  trajectory_msgs/JointTrajectory detach_posture
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
      float64[] positions
      float64[] velocities
      float64[] accelerations
      float64[] effort
      duration time_from_start
  float64 weight
moveit_msgs/AttachedCollisionObject[] movable_objects
  string link_name
  moveit_msgs/CollisionObject object
    byte ADD=0
    byte REMOVE=1
    byte APPEND=2
    byte MOVE=3
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string id
    object_recognition_msgs/ObjectType type
      string key
      string db
    shape_msgs/SolidPrimitive[] primitives
      uint8 BOX=1
      uint8 SPHERE=2
      uint8 CYLINDER=3
      uint8 CONE=4
      uint8 BOX_X=0
      uint8 BOX_Y=1
      uint8 BOX_Z=2
      uint8 SPHERE_RADIUS=0
      uint8 CYLINDER_HEIGHT=0
      uint8 CYLINDER_RADIUS=1
      uint8 CONE_HEIGHT=0
      uint8 CONE_RADIUS=1
      uint8 type
      float64[] dimensions
    geometry_msgs/Pose[] primitive_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    shape_msgs/Mesh[] meshes
      shape_msgs/MeshTriangle[] triangles
        uint32[3] vertex_indices
      geometry_msgs/Point[] vertices
        float64 x
        float64 y
        float64 z
    geometry_msgs/Pose[] mesh_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    shape_msgs/Plane[] planes
      float64[4] coef
    geometry_msgs/Pose[] plane_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    string[] subframe_names
    geometry_msgs/Pose[] subframe_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    byte operation
  string[] touch_links
  trajectory_msgs/JointTrajectory detach_posture
```

```
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
      float64[] positions
      float64[] velocities
      float64[] accelerations
      float64[] effort
      duration time_from_start
  float64 weight
```

8.2.5 url_project/place_location_description

It is the message send by the url_project_place_location topic. It contains:

```
std_msgs/String support_surface_name
  string data
std_msgs/String object_name
  string data
moveit_msgs/PlaceLocation[] place_location_object
  string id
  trajectory_msgs/JointTrajectory post_place_posture
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
      float64[] positions
      float64[] velocities
      float64[] accelerations
      float64[] effort
      duration time_from_start
  geometry_msgs/PoseStamped place_pose
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    geometry_msgs/Pose pose
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
  moveit_msgs/GripperTranslation pre_place_approach
    geometry_msgs/Vector3Stamped direction
      std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
      geometry_msgs/Vector3 vector
        float64 x
        float64 y
        float64 z
    float32 desired_distance
    float32 min_distance
  moveit_msgs/GripperTranslation post_place_retreat
    geometry_msgs/Vector3Stamped direction
      std_msgs/Header header
        uint32 seq
        time stamp
        string frame_id
      geometry_msgs/Vector3 vector
        float64 x
        float64 y
        float64 z
    float32 desired_distance
    float32 min_distance
  string[] allowed_touch_objects
```

8.2.6 url_project/vision_description

It is the message send by the url_project_vision topic. It contains:

```
url_project/object_vision[] objects_vision
  std_msgs/String id
    string data
  std_msgs/String name
    string data
  std_msgs/String type
    string data
  std_msgs/String type_pddl
    string data
  std_msgs/String color
    string data
  url_project/object_dimensions dimensions
    float32 x
    float32 y
    float32 z
  url_project/object_location location
    float32 x
    float32 y
    float32 z
  std_msgs/String[] is_on
    string data
  std_msgs/String[] is_bellow
    string data
```

8.2.7 Data

They are the messages received and send by the Data service, respectively (separated by $---$):

```
uint8 request
std_msgs/String string_information
  string data
uint16 integer_information
url_project/vision_description objects
  url_project/object_vision[] objects_vision
    std_msgs/String id
      string data
    std_msgs/String name
      string data
    std_msgs/String type
      string data
    std_msgs/String type_pddl
      string data
    std_msgs/String color
      string data
    url_project/object_dimensions dimensions
      float32 x
      float32 y
      float32 z
    url_project/object_location location
      float32 x
      float32 y
      float32 z
    std_msgs/String[] is_on
      string data
    std_msgs/String[] is_bellow
      string data
---
bool error
url_project/object_vision object
  std_msgs/String id
    string data
  std_msgs/String name
    string data
  std_msgs/String type
    string data
  std_msgs/String type_pddl
    string data
  std_msgs/String color
    string data
  url_project/object_dimensions dimensions
    float32 x
    float32 y
    float32 z
  url_project/object_location location
    float32 x
    float32 y
    float32 z
  std_msgs/String[] is_on
    string data
  std_msgs/String[] is_bellow
    string data
std_msgs/String string_information
  string data
uint16 integer_information
float32[] array_location
url_project/vision_description objects
  url_project/object_vision[] objects_vision
    std_msgs/String id
      string data
    std_msgs/String name
      string data
    std_msgs/String type
      string data
    std_msgs/String type_pddl
      string data
    std_msgs/String color
      string data
    url_project/object_dimensions dimensions
      float32 x
      float32 y
      float32 z
    url_project/object_location location
```

```
      float32 x
      float32 y
      float32 z
    std_msgs/String[] is_on
      string data
    std_msgs/String[] is_bellow
      string data
```

8.2.8 Plan

They are the messages received and send by the Plan service, respectively (separated by $---$):

```
bool is_place
geometry_msgs/Pose target_pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
geometry_msgs/Pose start_state
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
moveit_msgs/AttachedCollisionObject attached_object
  string link_name
  moveit_msgs/CollisionObject object
    byte ADD=0
    byte REMOVE=1
    byte APPEND=2
    byte MOVE=3
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string id
    object_recognition_msgs/ObjectType type
      string key
      string db
    shape_msgs/SolidPrimitive[] primitives
      uint8 BOX=1
      uint8 SPHERE=2
      uint8 CYLINDER=3
      uint8 CONE=4
      uint8 BOX_X=0
      uint8 BOX_Y=1
      uint8 BOX_Z=2
      uint8 SPHERE_RADIUS=0
      uint8 CYLINDER_HEIGHT=0
      uint8 CYLINDER_RADIUS=1
      uint8 CONE_HEIGHT=0
      uint8 CONE_RADIUS=1
      uint8 type
      float64[] dimensions
    geometry_msgs/Pose[] primitive_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    shape_msgs/Mesh[] meshes
      shape_msgs/MeshTriangle[] triangles
        uint32[3] vertex_indices
      geometry_msgs/Point[] vertices
        float64 x
        float64 y
        float64 z
    geometry_msgs/Pose[] mesh_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    shape_msgs/Plane[] planes
      float64[4] coef
    geometry_msgs/Pose[] plane_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    string[] subframe_names
    geometry_msgs/Pose[] subframe_poses
      geometry_msgs/Point position
        float64 x
        float64 y
        float64 z
      geometry_msgs/Quaternion orientation
        float64 x
        float64 y
        float64 z
        float64 w
    byte operation
  string[] touch_links
  trajectory_msgs/JointTrajectory detach_posture
    std_msgs/Header header
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
```

```
      float64[] positions
      float64[] velocities
      float64[] accelerations
      float64[] effort
      duration time_from_start
  float64 weight
---
bool error
```