Autonomous user feeding by a Physically Assistive Robot

Maria Vila Abad

Institut de Robòtica i Informàtica industrial (IRI)
Supervisor: Gerard Canal
Co-supervisor: Guillem Alenyà
Automatic Control Department (ESAII)
Tutor: Cecilio Angulo

Bachelor Degree in Informatics Engineering
Computing Specialization
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya - BarcelonaTech
January 2019

Abstract (English)

Assistive technologies will be a key factor in the well-being of the older adults and the handicapped population in the close future. The usage of these technologies will empower this collective giving them more independence in the development of some daily living activities such as self-feed.

In this thesis, we propose a robot capable of feeding a person autonomously. The prototype is capable of detecting the world state through RGB-D cameras and a force sensor. Moreover, it is capable of detecting the user requirements through visual communication. With this information, the robot can autonomously move to the desired positions in order to feed the user. Some of the executed trajectories are computed with Atificial Intelligence (AI) techniques.

The main focus of this project is the robot behavior which has also been implemented using AI techniques. It takes into account the state of the world and the user requirements in order to decide which are the best set of actions that need to be performed in each situation.

The resulting behavior is deeply tested in order to decide if the chosen approach is appropriate for this application. Moreover, the movements obtained through AI are also tested to in order to determine its correct performance.

Abstract (Castellano)

En un futuro cercano, las tecnologías asistenciales serán un factor clave para el bienestar de las personas mayores y de los discapacitados. El uso de estas tecnologías apoderará este colectivo dándoles más independencia en el desarrollo de actividades cotidianes como por ejemple alimentarse.

En este proyecto proponemos un robot capaz de dar de comer a personas de manera autónoma. El prototipo es capaz de detectar el estado del mundo con cámaras RGB-D i un sensor de fuerza. Además, es capaz de detectar los requisitos del usuario a través del análisis visual. Con toda esta información, el robot se mueve autónomamente a la posición deseada con el objetivo de dar de comer al usuario. Algunas de las trayectorias ejecutadas se obtienen con técnicas de Inteligencia Artificial (IA).

Este proyecto se enfoca principalmente al comportamiento del robot que también ha sido implementado usando técnicas de IA. Tiene en cuenta el estado del mundo y los requisitos del usuario para decidir cuáles son las mejores acciones a ejecutar en cada situación.

El comportamiento resultante es estudiado profundamente para decidir si el enfoque escogido es apropiado para esta aplicación. Los movimientos obtenidos usando IA también son analizados con el objetivo de comprobar su correcta ejecución.

Abstract (Català)

En un futur porper, les tecnologies assistencials seran un factor clau pel benestar de les persones grans i dels discapacitats. L'ús d'aquestes tecnologies apoderarà aquest col·lectiu donant-los més independència en el desenvolupament d'activitats cotidianes com per ejemple alimentar-se.

En aquest projecte proposem un robot capaç de donar de menjar a persones autònomament. El prototip és capaç de detectar l'estat del món fent ús de càmeres RGB-D i un sensor de força. Adicionalment, és capaç de detectar els requisits de l'usuari a través d'anàlisi visual. Amb aquesta informació, el robot es mou autònomament a la posició desitjada amb l'objectiu de donar de menjar a l'usuari. Algunes de les trajectòries executades s'obtenen amb tècniques d'Intel·ligència Artificial (IA).

Aquest projecte s'enfoca principalment al comportament del robot que també ha estat implementat fent ús de tècniques d'IA. Té en compte l'estat del món i els requisits de l'usuari per a decidir quines són les millors accions a executar en cada situació.

El comportament resultant és estudiat profundament per tal de decidir si l'enfoc escollit és apropiat per a aquesta aplicació. Els moviments obtinguts usant IA també són analitzats amb l'objectiu de comprovar la seva correcta execució.

Contents

List of Figures

List of Tables

List of Acronyms

AI Atificial Intelligence 3

DOF Degrees Of Freedom 19

GMM Gaussian Mixture Model 46

GUI Graphical User Interface 39

HRI Human-Robot Interaction 15

IRI Institut de Robòtica i Informàtica Industrial 14

PDDL Planning Domain Definition Language 25

ROS Robot Operating System 23

1. Introduction

Life expectancy has been increasing worldwide over the years. For instance, according to the United Nations [1], on 2000 life expectancy worldwide at birth was of 65.7 years whereas it will be of 77 years in 2050. Another example is presented in Figure 1.1 which shows the population pyramid comparing the European Union population in 2017 and the 2080 projection made by the Eurostat [2]. With the increase of aging population, more gerontology assistants will be needed.

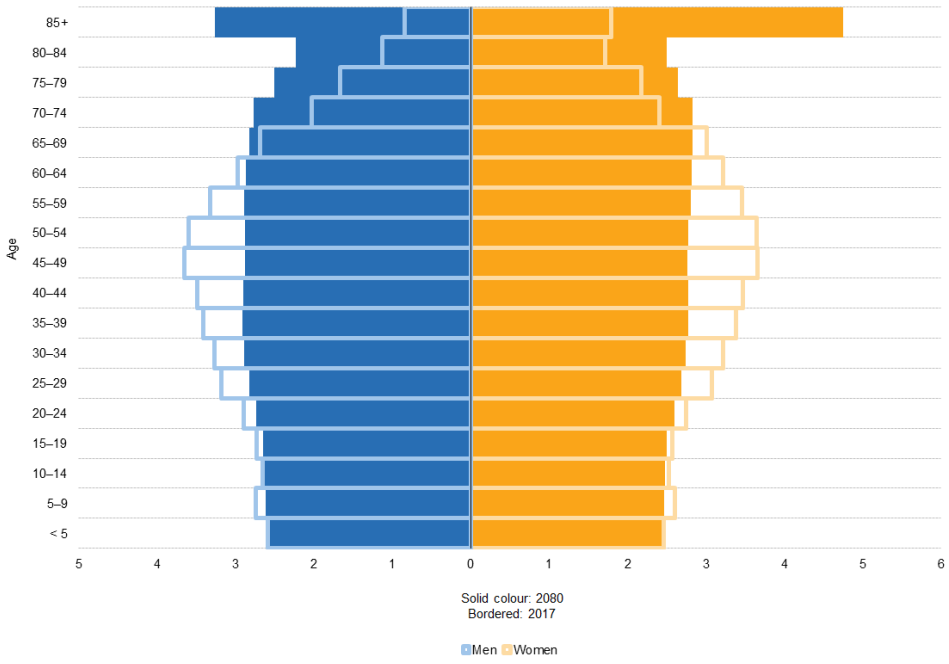


Figure 1.1: Population pyramid of EU in 2017 and expected in 2080
Source: Eurostat [2]

Moreover, the European Union expects a 34% increase in the number of stroke events [3] with also an increase in the number of stroke survivors who may potentially need assistive care. Currently, according to the World Health Organization [4] approximately 15% of the world's population, over a billion people, has some sort of disability.

This perspective clarifies the need for more assistive care, numbers with which

we may not be able to cope. Therefore, the need for assistive technologies will be a key factor in the well-being of the older adults and handicapped population in the close future.

One of the most important daily living activities that a person needs to be able to perform in order to feel independent is self-feeding. This task can be impossible to perform by a subset of the older adults and the handicapped population who undergo loss of upper limb functions. As a result, there is a need for assistive devices capable of helping a person self-feed. These types of devices are called Meal Assistance Robots. As will be explained in Chapter 3, currently there does not exist any robot capable of autonomously developing this task.

Consequently, this project consist on the development of a Meal Assistance Robot capable of naturally and autonomously feeding a user. In order to achieve it, the robot detects the world state through visual inputs and the user requests through visual stimulus. With this information and using AI techniques, the robot is capable of deciding which action to take at any given situation.

This project has been built upon a prototype [5] that had been developed at Institut de Robòtica i Informàtica Industrial (IRI) laboratories. The existing prototype focuses on the analysis of potential hazards in the feeding application. It studies how to avoid undesired contact between the user and the robot and how to act if undesired contact occurs in order to avoid further harm to the user. It was implemented with a State Machine and with predefined movements.

In this project, some improvements are integrated. The main improvement and focus of this project is the achievement of an autonomous robot behavior which has been implemented using AI techniques. The other main improvement is the movement of the robot as in this project it adapts to the user's mouth and dish position.

Moreover, this project aims to be built as a solid demonstration which could be easily executable, maintainable and extensible. Therefore, the parts of the existing prototype that can be reused for this project have been completely reimplemented in order to follow the same style on the whole project.

2. Objectives

The main goal of this project is to improve the abilities of a robot with the purpose of feeding a person with reduced mobility in an intelligent way. The feeding task on hands starts with the selection of the dish to serve. Then, the robot will grasp the food and feed the user. This action will be repeated until the user asks to stop the feeding process. The focus of this project will be on the following aspects:

- Achieve an autonomous behavior that adapts to the user's requirements
- Achieve a robot movement that adapts to the different elements of the system as the dish position or the user's mouth position
- Ease the code maintainability and extensibility

The user will be able to control the robot behavior through visual stimulus. Moreover, the robot will also have visual information of the world state. With this knowledge, the robot will be capable of autonomously and intelligently deciding which action to perform at any moment.

During this task, there exists a close Human-Robot Interaction (HRI). Therefore, it is of vital importance to perform all movements safely, especially the ones that require contact between the user and the robot.

3. State of the art

With aging societies and the increase of the handicapped population the demand for Meal Assistance Robots is increasing. As a result, research groups and companies are starting to develop this technology. Broadly, there exist three types of solutions which are stated below:

- **Forearms stabilizers:** these systems consist on an arm support where the user's arm is completely or partially fixed. This support helps to stabilize the user's arm when he is performing the feeding action and thus reduces the user's arm vibration. Some examples are the Table Mount Mobile Arm Support (MAS) [6] and the Jaeco arm support [7].

- **Manually operated eating systems:** these systems consist on a robotic arm which holds the used cutlery. The robotic arm does not move automatically and requires the user to move it. These systems stabilize the spoon movement and help the user move the used cutlery more accurately. However, they require the user to be able to move his arms in order to move the robotic arm. The most famous manually operated eating system is the Neater eater [8].

- **Electrically operated eating systems:** these systems consist on a robotic arm which holds the used cutlery. Unlike manually operated eating systems, electrically operated eating systems are capable of moving autonomously. There are several ways of controlling the robotic arm. For example, My Spoon [9] (which can be observed in Figure 3.1 (b)) uses a chin-operated joystick whereas Mealbuddy [10] (which can be observed in Figure 3.1 (c)) and Mealtime Partner [11] are controlled by buttons.

In order to be able to use forearms stabilizers or manually operated eating systems, the user needs to have at least partial mobility on the arms. However, this is not the case of a non-negligible subgroup of the older adults and handicapped collective. On the other hand, electrically operated eating systems can be used by anyone as they move autonomously. Therefore, this last systems will be further studied.

A key factor of the electrically operated eating systems is the control method used as it indicates which type of users the robot can have. Many prototypes opt for a mechanical actioning using buttons or a joystick. This solution is easy

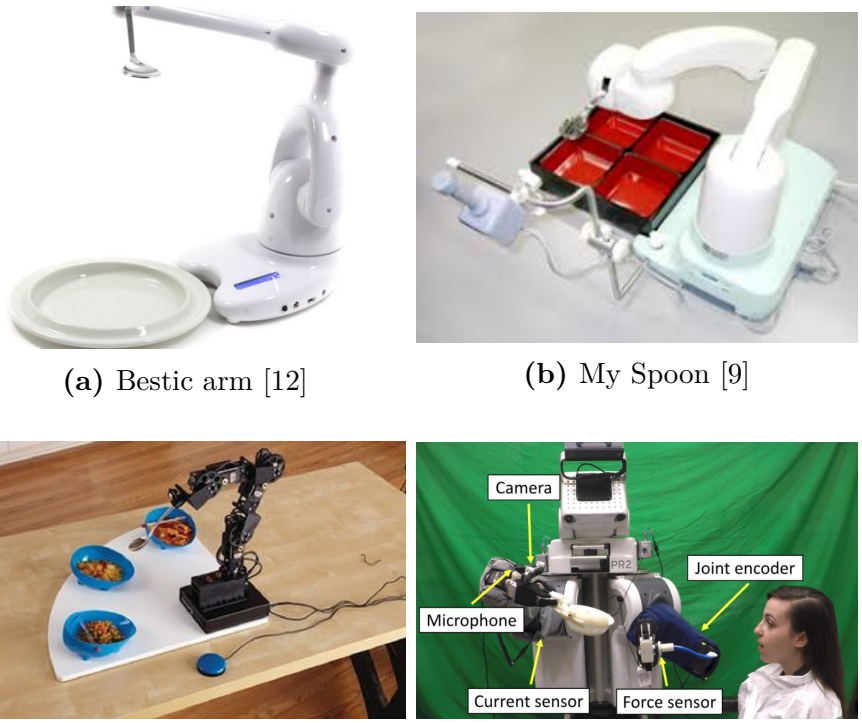
but in order to be able to use the robot the user needs to have mobility in the upper limbs. Some examples are Mealbuddy [10], Mealtime Partner [11], Bestic Arm [12] (which can be observed in Figure 3.1 (a)) or ASIBOT [13]. A special case of this group is My Spoon [9] as it uses a chin-operated joystick and thus it can only be used by people that can perfectly move the chin. However, there exist some prototypes with a more automatic approach which ensure the possibility of controlling the robot by the majority of the older adults and the handicapped population. One example is The voice bot [14] which enables the user to control the robot using its voice. Nevertheless, it may not be comfortable for the user to speak to the robot with the mouth full or the spoon approaching his mouth. Another example is the one presented by the Yamaguchi University [15] which enables the robot control through an eye interface. However, this approach may not be intuitive or fast enough for the user and it only gives the user few controlling options.

Most of these systems lack enough sensory information to provide full feeding assistance. For example, the vast majority of systems do not contain a camera and thus, the food and mouth detection cannot be performed. These systems work with predefined positions and operate as follows: the robot moves to a predefined dish position where it grasps the food, it then moves to a predefined mouth position and waits for the user to unload the food from the cutlery used. Therefore, they require a movement from the user towards the spoon and thus they can not be used by everyone. However, some authors like Park *et al.* [16] have started to work on a solution. Concretely, these authors propose a face detection and tracking using a Kinect camera and AR Tags. However, this system is not perfect as it does not detect the mouth openness and therefore it requires a manual selection through a GUI when the user wants to be fed. Another example of autonomous movement is presented in [17]. The presented movement is based on Probabilistic Movement Primitives and it allows a teaching and reteaching of the trajectories.

Another important factor is the behavior of the robot and the detection of the anomalies during the execution. The vast majority of the prototypes only have at their disposal one sensory information and thus the behavior is really simple and not human-like and few anomalies can be detected. The prototype presented by Georgia Tech in [18] (which can be observed in Figure 3.1 (d)) starts to work on this aspect. In this article, a prototype with 5 types of inputs is presented. These inputs are obtained through a camera, a microphone, a current sensor, a force sensor, and a joint encoder. This allows the robot to detect various types of anomalies. However, it still lacks further improvement as, for example,

this prototype does not detect the mouth openness. The prototype presented in [19] solves this problem as it includes mouth openness detection. This prototype also offers control through Electroencephalography in order to know the user's intentions. Nevertheless, it does not support any anomalies control.

The field of assistive robotics is rapidly increasing but there is still a lot of research to do. As explained before, there does not exist any prototype capable of feeding a person autonomously and safely as all approaches lack of sensory information, of mouth openness detection or of anomalies detection. Therefore, there exists a not covered need of a completely safe Meal Assistant Robot which does not require any type of mechanical actioning.



(a) Bestic arm [12] **(b)** My Spoon [9]

(c) Mealbuddy [10] **(d)** GeorgiaTech feeding robot [18]

Figure 3.1: Different state of the art self-feeding robots
Source: Own compilation

4. Resources

When trying to feed a person autonomously there are several resources that have to be used. The most important one is the robot itself as it is going to carry out the task. However, resources such as cameras and a force sensor allow the task to be safe and to adapt to the user's requirements. Moreover, in order to be able to execute the task continuously we also need some software resources. In this chapter, the most relevant resources needed for this task are going to be explained.

4.1 Barret WAM® robotic arm

The Barret WAM®, which can be observed in Figure 4.1, is a robotic arm with 7 Degrees Of Freedom (DOF). Therefore, it has redundant DOF which allow it to reach any point within its workspace with different joint positioning and any orientation of its end plate. Its workspace is an almost complete sphere with one meter of radius.

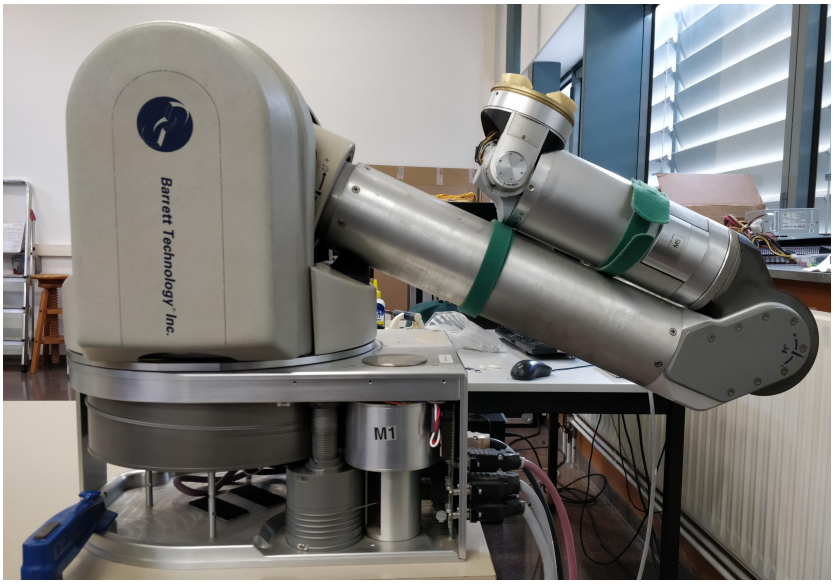


Figure 4.1: Barret WAM robotic arm
Source: Own compilation

The Barret WAM® is capable of lifting 3Kg of weight and reaching a maximum velocity of 3m/s which surpasses our requirements. This robotic arm uses back-drivable cables in order to move which allows it to have different levels of stiffness and to move with human-like grace and dexterity. The robot construction assures an excellent HRI, specifically when it involves direct contact between the user and the robot.

In order to further assure the safety of the task development, this robotic arm also has a controller. This controller offers two types of emergency shutdown buttons. One of them immediately shuts down all the power to the robot and thus it falls down with its own weight. The other emergency exit is less radical as it gradually decrements the supply power to the robot resulting in a slow descension.

In order to be able to grasp the spoon, the WAM requires a gripper which is attached to the last joint. The gripper used for this application, which is shown in Figure 4.2 has been developed with a 3D printer at IRI laboratories. As it can be observed, it is designed to adapt perfectly to the spoon shape. Moreover, it also allows a camera and a force sensor to be mounted on them which is a requirement for this task.

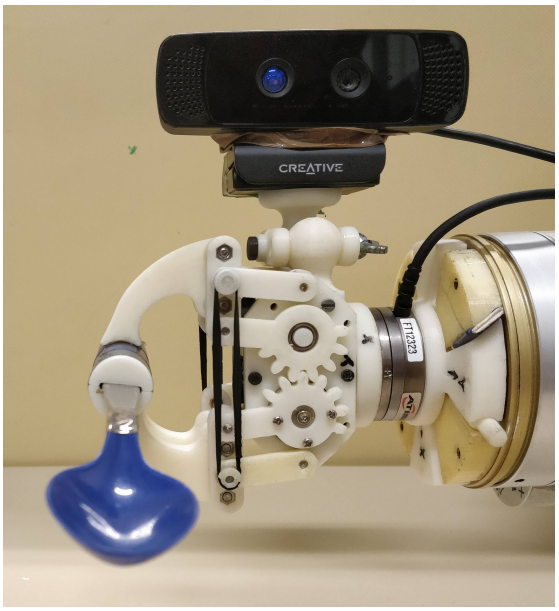


Figure 4.2: 3D printed gripper with camera and force sensor attached
Source: Own compilation

4.2 Creative Senz3D camera

In this task we need to have visual information about the state of the mouth of the user and, generally, the head of the user. However, depending on the positioning of the camera, the image may be occluded by the robotic arm. Regarding this, the best positioning for the camera is on the end plate of the robot as, in this position, the image does not have occlusions. Therefore, we need a small and close range camera that can be attached to the gripper of the robot. Moreover, the robot needs to know the position of the mouth of the user to develop the feeding action. Therefore, the camera has to be able to obtain RGB-D images.

The Creative Senz3D camera, shown in Figure 4.3, fulfills all the stated requirements as it is an small, close range RBG-D camera. In order to obtain Depth images it uses the time-of-flight technology. This technology consists on sending light signals which are reflected by the objects on the field of view and the camera lenses gather the reflected light signals. Knowing the speed of the light and the delay of the reflected light signals, the distance traveled is calculated. This technology allows the camera to have a better precision on closer objects.

Figure 4.3: Creative Senz3D camera
Source: Hardwarezone [20]

Therefore, the Creative Senz3D camera will be mounted on the gripper of the robot to obtain information about the user state, as seen in Figure 4.2.

4.3 Xtion camera

As the previous camera is mounted on the gripper of the robot it is not possible to obtain the dish positions with a natural movement of the robotic arm. Therefore, we need another camera to do this detection. This camera also has to obtain RGB-D images as we want to know the position of the dishes. The camera will be mounted above the table to be able to obtain information from all the workspace. Therefore, we need a camera with medium range depth detection.

For this usage, the Xtion camera shown in Figure 4.4 has been selected. It consists on an RGB-D camera which uses infrared sensors and the adaptive depth detection technique to obtain depth images. This results in a medium detection range depth camera which is perfect for our application.

Figure 4.4: Xtion camera
Source: Xtion webpage [21]

4.4 ATI mini 40 f/t force sensor

During the feeding task there exists direct contact between the robot and the user. In order to be able to assure a safe task performance, the forces and torques produced at the end plate of the robot need to be monitored. With these information, it is possible to decide if the robot needs to be stopped for safety reasons. For example, if a harmful force has been reached the robot has to be stopped to avoid further damage.

In order to obtain this information the ATI mini 40 f/t force sensor, which can be observed in Figure 4.5, has been used. This sensor provides information about the forces and torques produced in the three axis. Regarding the forces, it has a sensing range from -20N to 20N with a resolution of 1/200N which is ideal for our application as we want a precise sensing in small forces to be able to prevent

harmful contacts. Moreover, it is an small and light weight sensor that can be perfectly attached to the 3D built gripper.

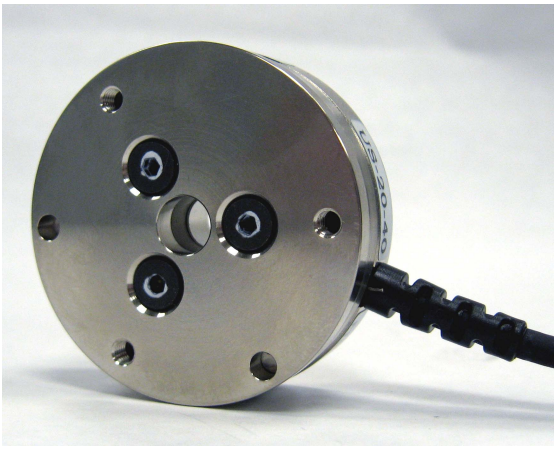


Figure 4.5: ATI mini 40 f/t force sensor
Source: ATI f/t sensor Mini40 webpage [22]

4.5 Robot Operating System

Robot Operating System (ROS) is a set of software libraries and tools developed to help build robot applications. Concretely, it consists on a message passing interface that provides inter-process communication. A ROS system is usually represented as a directed graph. Each node of the graph, called a ROS node, represents a ROS-based process. Each edge of the graph represents a communication between the two nodes it connects. Every edge can only transfer a unique type of messages which determines the type of the edge. All the nodes in the same graph need to be connected on the same network to receive and send the messages correctly. There exist three types of communications between nodes (therefore, three types of edges):

1. **Service:** services are a type of communication between two nodes. One node sends requests to the other node which is called the server. Once the server has processed the requests it sends back a response. Service calls are blocking and thus the caller remains blocked until it receives a response.

2. **Topic:** a topic is a type of communication that allows sending and receiving messages continuously. There is one node, called the publisher, which

publishes the messages to the topic. One or several nodes, called the subscribers, can be subscribed to the topic to receive all the messages that are published in it.

3. **Action:** actions are the most complex type of communication between nodes. It is composed by five topics: goal, cancel, status, feedback and result. In an action there are always two nodes: the action server which publishes on the status, feedback and result topics and the action subscriber which publishes on the goal and cancel topics. When the action client publishes a message to the goal topic, the action client starts to process the result. When the action client is processing, it continuously publishes on the feedback and status topics telling the feedback of the action being processed and its current status. Once the action has been finished, the action server publishes its result on the result topic. During all the action processing, the action client can cancel the action publishing on the cancel goal.

ROS also offers a wide variety of tools to support introspecting, debugging, plotting and the visualization of the state of the system being developed. One example is rviz, which allows a three dimensional visualization of the state of the system. It allows, for example, a visualization of the system, the pointclouds provided by the camera and/or a visualization of all the available transformation between the different elements of the scene. Another popular tool offered by ROS is rqt which allows for example the visualization of the system ROS graph or of the images captured.

4.6 Programming languages

ROS nodes working in the same network do not need to be implemented with the same programming language as the communication between them is done through services, topics or actions. Therefore, ROS nodes can be implemented in a variety of programming languages the most common of them being Python and C++.

In this project C++ is going to be used over Python as it is a compiled language. On the robotics sector, it is of vital importance to be able to execute the tasks on real time without any kind of delays; specially on the service robotics sector. This is due to the fact that there exist close contact between humans and robots and a delay in an operation may result in dangerous situations. Therefore, it is better to use a compiled language, such as C++, instead of an interpreted one, such as Python, as they are usually more efficient.

In order to determine the behavior of the robot a planner is going to be used. Concretely, in this application, the planner is going to determine the best set of actions that the robot needs to perform in order to achieve the desired goal given the current situation. The planner that is going to be used is Metric-FF as it is a widely used planner that allows minimization of the cost function. Finally, the domain and problem are going to be implemented with Planning Domain Definition Language (PDDL).

5. Regulation

This project is affected by two different type of regulations: software regulations which is detailed in Section 5.1 and service robots regulations which is detailed in Section 5.2.

5.1 Software regulations

All software is regulated by a specific license that determines the rights of the user over the software and under which conditions he can use the software. All the software used in this project is regulated by one of the following licenses.

5.1.1 Academic or non-profit organization noncommercial research use only

This license allows the software to be used only for non-commercial research projects. The proprietary of the software is the Licensor and person who uses the it is the Licensee. The Licensee cannot reproduce, distribute or publish the software, any portion of it or any modification of it. If derivations of the software are made, they are owned by the Licensor. Moreover, these modifications can only be used for non-commercial internal research purposes of the Licensee that has implemented them. If these modifications were to be published, they have to be sent to the Licensor within 30 days.

This license is used by the libraries OpenFace and OpenPose, which are used to detect the mouth openness, the mouth position, and the head orientation.

5.1.2 GNU Lesser General Public License

This free software license allows companies and developers to use and integrate the software to their own code freely. This license requires the software to be modifiable by the end user via source code availability. Any modification of the original software must maintain the same license and conditions. However, derivate software can use other licenses.

This license is used by the Institut de Robòtica i Informàtica Industrial(IRI) software.

5.1.3 BSD license

This free software license is one of the most permissive as it imposes minimal restrictions on the use and distribution of the software. It allows the Licensee to reproduce, distribute or publish the source code freely. The only condition that imposes this license is that if redistributions are made they need to have the same license and they need to acknowledge the Licensor.

This license is used by Robot Operating System (ROS) which is used to program the robot. This license is also used by Rosplan which is used to program the robot behavior.

5.2 Service robots regulations

Personal care robots are a new technology expected to increment the quality of our lives in the foreseeable future. However, unlike industrial robots, they require special safety measures as they are going to be in direct contact with their users. Therefore, institutions are increasingly focusing on the legal challenges proposed by the robotics sector.

The group ISO/TC 299 is in charge of standardization in the field of robotics, excluding toys and military applications. Currently, this group is focusing on developing the regulations for service robots. In 2014, they published the ISO 13482:2014 which is a general regulation of service robots. However, the technical regulation of service robotics is still under development. Nevertheless, the principles presented in ISO/TS 15066:2016 can be used as guidelines.

5.2.1 ISO 13482:2014

ISO 13482:2014 is an international regulation that proposes the security standards for robots and robotics devices in personal care. The aim of this regulation is to specify the contact conditions between human and robot. Particularly, it specifies the requirements and guidelines for the inherently safe design, protective measures, and information for the usage of personal care robots. This regulation considers three types of personal care robots: mobile servant robots, physical assistant robots, and personal carrier robots. The prototype developed in this project consists of a physical assistant robot as the robot physically helps the user to carry out a task.

5.2.2 ISO/TS 15066:2016

ISO/TS 15066:2016 is an international regulation that proposes the technical security standards for industrial robots systems. Concretely, is specifies the safety requirements for collaborative industrial robot systems and the work environment needed for these systems. Collaborative robots are those who are in direct contact with humans. This regulation also supplements the requirements and guidance on collaborative industrial robot operation.

6. Feeding assistance application development

Nowadays, the number of people not capable of performing some basic tasks such as self-feed is not negligible and it is also rapidly increasing. These people need external help in order to develop these basic tasks which is usually provided by nurses or relatives. However, as it has been explained before, we may not be able to cope with the needed numbers of nurses to attend this growing need in the close future.

If a robot was able to develop these tasks autonomously, it would solve the number of nurses problem. However, this robot would have to perform these tasks as a person would perform them. Ergo, developing the tasks autonomously while completely adapting to the user's preferences and commands. As it has been stated before, in this project we are going to focus on the feeding task. We will demonstrate that a robot is capable of feeding a person autonomously while adjusting to the user's needs and preferences and always assuring the safety of the process.

As we want the robot to be able to feed as much disabled people and older adults as possible, we can not assume that the user has mobility on the upper limbs. We will only assume that the user has mobility from the neck to the top of the head. This assumption is being made as we want the user to be able to communicate his requirements and desires to the robot. It is important that the user is able to communicate with the robot as feeding is an invasive task. Consequently, the user would only be comfortable when being fed if he feels in control of the robot. Therefore, there exists a trade-off between maximizing the potential users and offering a comfortable design. This is why we only assume the user to be able to move his neck, mouth, eyes and be able to speak.

With the mobility of the user that has been assumed it is not possible for him to control the robot movement with a comfortable and intuitive approach. Consequently, the robot will move autonomously and the user will only control the general behavior of the robot.

It is of vital importance that the development of the task is as natural and comfortable for the user as possible. In order to achieve it, the communication between the user and the robot needs to be intuitive and human-like. The most common way of communicating for a person is through speech and thus it will be

one way of communication between the user and the robot. However, there are some situations where speech is not the most intuitive way of communicating. For example, imagine that the robot is holding a spoon with food and it is approaching the mouth. In this situation, if the user does not want to be fed it is not comfortable for him to say so as the spoon is too close to the mouth. It is more intuitive for him to close his mouth or rotate his head. Therefore, there are two more interactions that have been taken into account: the openness of the mouth and the rotation of the head. In conclusion, the user will be able to communicate his desires and requirements to the robot through speech, mouth openness and head rotation.

As speech is the most comfortable way of communication it will be always used for the robot to communicate with the user: to explain his intentions or to inform about the user's options. Moreover, the speech will also be the way of communication for the user when the robot is not approaching the mouth. The user will be able for example to tell the robot which dish he desires, when he wants to start or to stop to eat. The rotation of the head and the openness of the mouth will only be used as a way of communication when the speech is not a comfortable option, i.e. when the robot is approaching the mouth. Therefore, for example, the robot will only try to enter the user's mouth when it is open or it will only approach the mouth when the user is paying attention to the robot, i.e. when he is looking towards it.

Another relevant matter is the safety of the process, as there exists direct contact between the user and the robot. However, undesired collisions between the user and the robot must be avoided as they can hurt the user. Therefore, strategies to avoid collisions are of vital importance. For example, if the robot is approaching the mouth and the user closes his mouth or moves his head the robot must stop the approaching and head back. However, there are some situations where the robot can not react on time and thus, the collisions can not be avoided. In these situations, the impacting force must be as low as possible in order to cause the least possible harm to the user.

As stated before, the objective of this thesis is to prove that an autonomous and intelligent yet intuitive and human-like feeder robot can be implemented. The following sections detail every part of the implementation of the feeder robot that has been described above.

6.1 ROS architecture

Before implementing any software it is of vital to determine its architecture. When deciding it, the requirements are being taken into account to organize the code and the communication between each part of the code. Therefore, the architecture lays the foundation of the software that is going to be implemented.

The ROS architecture of this project is shown in Figure 6.1. All the nodes marked with orange have been implemented for this project whereas all the nodes marked with green were already available. The node marked with blue was already available but due to technical problems it was not possible to use it. Each node has its own functionality and it communicates with the nodes which it is connected with through ROS services, topics, and/or actions. A detailed description of each ROS node is given below.

- **DepthsenseCamera:** this node is in charge of obtaining the RGB and the Depth images from the Depthsense camera. It publishes these images into the respective topics making them available to the other nodes.
- **OpenposeRos:** this node receives the RGB images from the Depthsense camera and process them using the OpenPose library. Therefore, it obtains the position within the image of each facial landmark[1] of the person being detected. It publishes the result obtained from this processing to the corresponding topic.
- **HeadDetection:** this node receives the RGB images from the Depthsense camera and process them using the OpenFace library. Therefore, it obtains the rotation of the head and decides with this information if the person is paying attention to the robot. It updates this information to the Rosplan node using the corresponding service.
- **MouthDetection:** this node receives the facial landmarks of the detected person and decides, with this information, the position of the mouth within the image and if it is open or not. It updates the state of the mouth to the Rosplan node using the corresponding service and publishes the mouth position within the image to the corresponding topic.
- **ObtainMouthPosition:** this node obtains the position of the mouth within the image and the depth image of the Depthsense camera. With this information and the transformation from the camera to the robot base, it obtains

[1]Facial landmarks are predefined points of a person's face

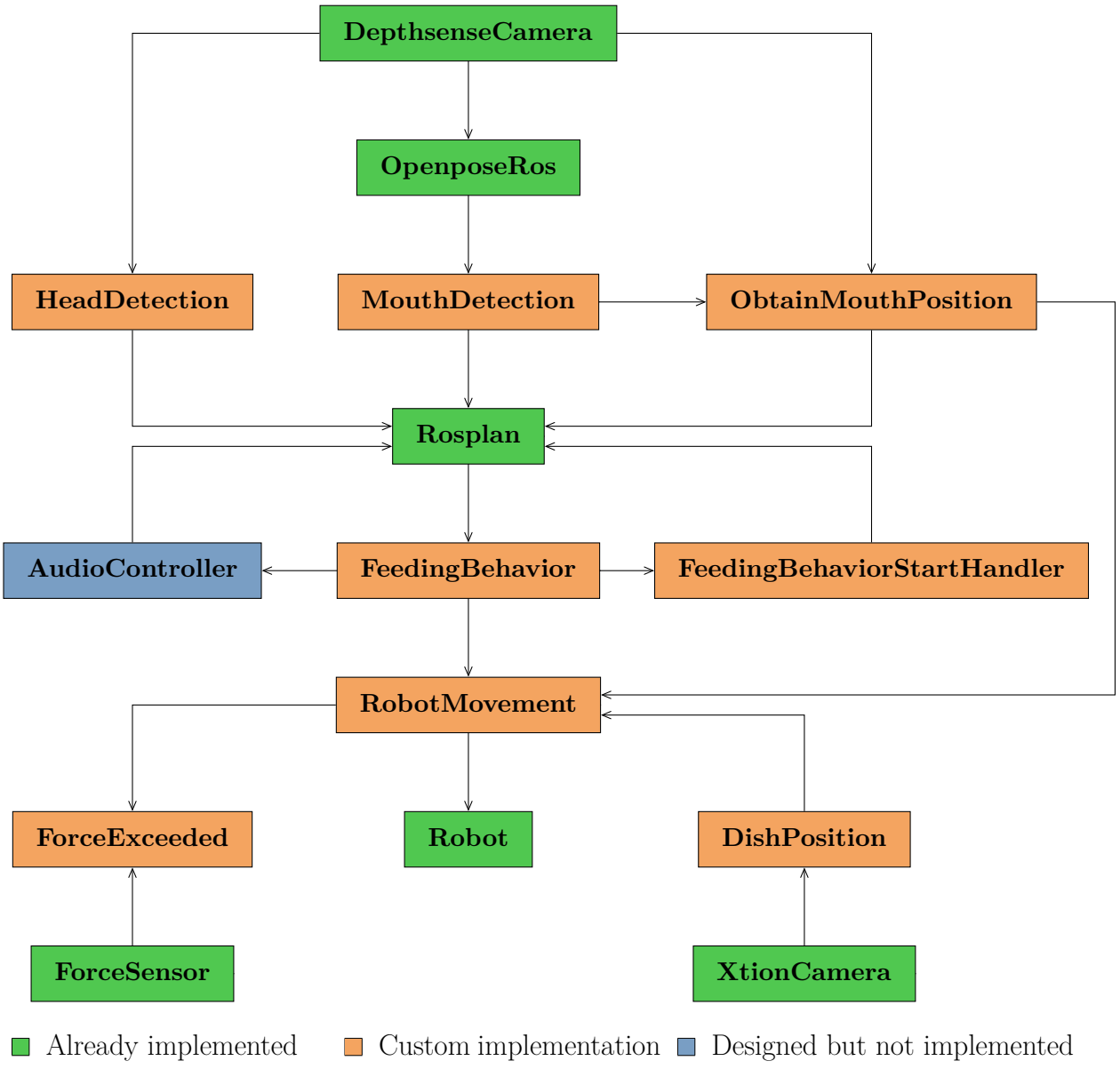


Figure 6.1: ROS Architecture of the feeding assistance implementation
Source: Own compilation

the position of the mouth relative to the robot base and publishes it to the corresponding topic. Moreover, it decides if the robot is too close to the face of the user to assure proper mouth detection. It updates this information to the Rosplan node using the corresponding topic.

- **Rosplan:** is a set of nodes that are in charge of deciding the robot behavior. Given the current state of the system and the objective state, it computes the best set of actions that need to be followed to achieve this objective state. using an AI planner, which is a solver that produces a plan -sequence of actions- based on an input domain defining the available actions and their constraints. This is done It also receives updates from the changes on the state variables through services. Moreover, it receives requests from the FeedingBehaviorStartHandler telling it when does it have to plan the solution, dispatch an action or other similar commands. Finally, it publishes the action that needs to be executed to the corresponding topic.

- **AudioController:** this node has been designed and simulated but not implemented. It is designed to inform the user aurally and to receive its requests also aurally. It receives the information to tell to the user from the FeedingBehavior node through the corresponding topic and it updates the user requests to Rosplan through the corresponding service.

- **FeedingBehavior:** this is the main node of the application. When it receives the start signal it calls the FeedingBehaviorStartHandler node through the appropriate action starting, in this way, the feeding process. Moreover, it receives the action that needs to be followed from the Rosplan node. With this information, it calls the RobotMovement and/or the AudioController node with the appropriate information.

- **FeedingBehaviorStartHandler:** this node receives the start signal from the FeedingBehavior node through an action. When this signal is received, it tells the Rosplan node through the corresponding services to start the planning of the robot behavior. This node is also in charge of telling the Rosplan node, through the corresponding services, when a re-planning of the current plan must be performed.

- **RobotMovement:** this node is in charge of moving the robot. It receives the mouth position and dish position through the corresponding topics. It also receives which is the trajectory that needs to be followed at every

moment from the FeedingBehavior node through an action. With this information, it calculates the appropriate trajectory that the robot has to follow. Moreover, when an undesirable collision can potentially occur it asks the ForceExceeded node, through the corresponding service, if it has occurred. If an undesirable collision has occurred it stops the robot movement and it moves back.

- **ForceExceeded:** this node receives the forces and torques of the end plate of the robot. With this information, it decides if an undesired impact has occurred.

- **Robot:** this node is in charge ob moving the robot itself. It receives the trajectory in joints from the RobotMovement node through a service or an action and moves the robot accordingly.

- **DishPosition:** this nodes receives the RGB and Depth images from the corresponding topics. With this information, it obtains the position of the dishes that are on the table and publishes these positions.

- **ForceSensor:** this node is in charge of obtaining the force and torques of the three axes of the end plate of the robot. It publishes this information to the corresponding topic making it available to the other nodes.

- **XtionCamera:** this node is in charge of obtaining the RGB and the Depth images from the Xtion camera. It publishes these images into the respective topics making them available to the other nodes.

With the architecture explained above we obtain different nodes with specific functionalities which are clearly different from the functionalities of the other nodes. The choice of the mean of communication between the nodes has been decided to ease the passing of information. As a general guideline, topics have been used when a continuous flow of information has to be passed. Services have been used when we want to check or update some specific information. Finally, actions have been used in the other cases. For example, when a node sends the information of a task development to another node and it needs to be able to be canceled.

An important aspect of this project is that it has been built in order to be easily maintainable and extensible. In order to achieve maintainability, all the nodes that have been developed for this project have been implemented following the guidelines and style that is used at IRI laboratories. With this approach, the

IRI staff can navigate effortlessly through the code and find or change any piece of code. The presented ROS architecture assures an easy extensible code. For example, if we were to change a node for another one, the new node would only have to use the same services, topics and/or actions than the node it was replacing. Moreover, if a new node was to be added it would only have to communicate with the other ones through the established services, topics and/or actions.

Some of the nodes developed in the existing prototype could have been reused in this project as the same functionality was required. However, these nodes were not developed following the guidelines and styles used at IRI and thus they had to be reimplemented for this project. The nodes that have been reimplemented because of this reason are the following ones: HeadDetection, MouthDetection, and ForceExceeded.

Another aspect that has been implemented to ease the debugging and testing task is the option of manually changing the user states and commands. With this option, it is possible to execute all the task without the need of a user interacting with the robot. For example, all the information which is extracted visually or aurally can be changed manually during the execution.

6.2 Existing prototype

This project is built upon an existing prototype developed at IRI laboratories. Therefore, part of its development has been used. However, the code has been reimplemented to comply with the IRI style and to adapt to the new ROS architecture. Concretely, the two parts of the existing prototype that have been reused are the safety of the prototype and the obtainment of the user requirements through visual sensing. A brief explanation of these parts is given below.

6.2.1 User requirements through visual sensing

It is thought that the most intuitive and comfortable way of communicating for a person is through speech. However, when the robot is approaching the user's face or is close to it, speech becomes an uncomfortable way of communication. In these situations, it is more intuitive to communicate through movements. For example, if the user wants to be fed, it is more intuitive for him to just open his mouth than to tell the robot to feed him.

Head orientation

In order to approach the user without scaring him, it is important to only move

when he is paying attention to the robot. The head orientation is obtained in the ROS node HeadDetection. This node receives the RGB images from the user from the DepthsenseCamera through the corresponding topic. It processes this images using the OpenFace [23] library obtaining the Euler angles of the user's head orientation. If this angles are inside the defined range, the user is considered to be paying attention at the robot. This system was tested in the existing prototype and it was determined that it was fast and accurate enough for this application.

Mouth openness
A key factor of the prototype is to only try to feed the user when his mouth is open. In order to obtain the mouth openness two ROS nodes are used: OpenposeRos and MouthDetection. This detection works as follows. The OpenposeRos node receives the RGB images obtained from the DepthsenseCamera. It processes these images using OpenPose [23] library and obtains the facial landmarks which sends to the MouthDetection node through the corresponding topic. Facial landmarks are predefined positions of a person's face. The MouthDetection node decides if the mouth is open or not comparing the height of the different facial landmarks that define the lips. This system was tested and validated in the existing prototype where it was determined that it is fast and precise enough for this application.

Inside mouth
It is important to know if the spoon is loaded with food or not to be able to decide which is the best action to take. We assume that the grasping never fails and thus, the spoon is always full after it. Moreover, if the distance between the camera and the user's mouth when the user is being fed is smaller than 20cm we also assume that the spoon has been unloaded of food. This information is obtained in the node ObtainMouthPosition. It receives the mouth position from the MouthDetection node and the depth images from the DepthsenseCamera node. With these information it calculates the distance between the camera and the user's face. If it is smaller than the set limit it considers that the spoon has been unloaded.

Reliable head and mouth detections
As the Crative Senz3D camera is mounted on the gripper, if it is too close to the user the images do not show the entire user's face. With only part of the face, the face detection libraries used (OpenPose and OpenFace) are not capable of detecting the facial landmarks or the head's Euler angles. Therefore, it is of vital importance to know when these detections can be relied on. This information can

be obtained knowing the distance between the camera and the user's face. The retrieval of this information works as follows. The node ObtainMouthPosition receives the mouth position from the MouthDetection node through the appropriate topic. It also receives the depth images obtained with the DepthsenseCamera node through the corresponding topic. With these information, it is possible to calculate the distance between the mouth of the user and the camera. If the distance is smaller than 40cm images the mouth openness and head orientation detection are considered to not be accurate enough. It is important to comment that the limiting distance can be changed during the execution in order to test the best value or to test other parts of the code.

6.2.2 Safety of the prototype

Eating assistance is an invasive task as cutlery is inserted inside the mouth of the user. Therefore, the user will only be comfortable with the robot if he feels safe during the whole process. The user feels safe if the robot does not move while he is not paying attention and if he feels in control of the robot behavior. These control methods are explained in Section 6.2.1 and in Section 6.3.1.

There exist some cases where the robot can not react on time or the user does not tell the robot a command on time. In some of these cases, undesired collisions may arise. As explained in [24], there exist two safety measures which can be followed in these situations.

The first one, called passive safety, is achieved implementing a compliant robot controller as explained in [25]. When a robot is compliant it moves with a reduced force and thus, in case of impact, low forces are applied. Nevertheless, there exists a trade-off between compliance and precision. When the robot compliance increases the robot precision decreases as the forces used to move the robot decrease. Another factor to take into account is that after an impact occurs the robot will repeatedly try to achieve the desired position and thus, it will repeatedly impact.

The second method, called active safety, is achieved by limiting the maximum force. A force sensor is mounted on the gripper of the robot to obtain force and torque readings. The maximum torques and forces that the robot can achieve are set and, when they are exceeded, the robot stops its movement. This method offers a precise movement. However, the impacting forces are greater than the ones achieved with passive safety and each robot movement has to be previously studied in order to set the precise limits.

Both methods were studied in the existing prototype. It was determined that

for this application, the best safety setup was to only use active safety as the maximum forces achieved with both methods were quite similar but passive safety drastically decreased the precision. Therefore, in this project only active safety has been used.

The safety of the prototype works as follows: a force sensor is mounted on the end plate of the robot as shown in Figure 4.2. It is constantly monitoring the maximum forces applied on the end plate of the robot. The forces are limited and, if they are exceeded, the robot stops its movement. The forces have only been limited when the robot is feeding the user as it is, potentially, the most dangerous trajectory. Concretely, the force has only been limited on the y-axis as it is the direction where the force would be applied to the user if an impact were to occur. The limiting force on this movement has been found empirically and can be observed in Table 6.1.

	Minimum value [N]	Maximum value [N]
Feed	-2	4.5

Table 6.1: Force limits of the y axis for the enter mouth and exit mouth movements
Source: Own compilation

When a limiting force is exceeded the robot stops its movement and enters the gravity compensation mode for one second. In this mode, the robot is completely safe as it stays in the same position only compensating the gravity force. Therefore, it lets itself be moved naturally without opposing the movement with a reaction force. After one second in this mode, the robot moves back. If during this movement a force is exceeded, the same safety procedure is applied. When the robot is at a safe distance of the person, it continues the feeding process.

6.3 Robot sensing

It has been determined that the most comfortable way to interact with the robot is through auditive communication. However, when the spoon is close to the user's face it is more comfortable and intuitive for the user to interact with the robot through movements. Therefore, when the robot is approaching the user, he will interact with movements as it is the most comfortable mean of communication in this situation. However, the auditive communication could not be developed due to technical problems so it has been simulated. Moreover, visual sensing has also been used to obtain the dish position and the user's mouth positions. Therefore, it is essential for this project.

The following subsections give a detailed explanation of the simulated audio sensing and the visual sensing that has been implemented for this project.

6.3.1 Audio sensing

It is thought that the most intuitive and comfortable way of communicating for a person is through speech. Therefore, speech is the perfect way of communication between the robot and the user as one of the main goals of the project is to provide a comfortable design for the user.

First of all, it is important to clarify that it was intended to capture the user's speech and inform the user with an Amazon Echo. However, due to technical problems, it has not been possible to use it. All these processing was intended to be implemented in the AudioController ROS node explained in Section 6.1. In order to be able to prove the behavior's performance without being able to use the Amazon Echo, the user requests have been simulated through the Graphical User Interface (GUI) offered by rqt[2].

Nowadays, the older adults are usually not very familiarized with how machines work. As they are an important percentage of the potential users of this project, it is of vital importance to design the prototype to be as simple to use as possible. Therefore, in order to ease the usage of the prototype, the user's requests will be limited. Concretely, the user will only be able to make the most essential requests for the functioning of the prototype. These request are the following ones:

- Inform the robot what dish the user wants to eat from
- Tell the robot to pause the feeding task for a while
- Tell the robot to resume the feeding task
- Tell the robot to end the feeding task

The robots needs to know from which dish he has to grasp the food and thus it is essential that the user tells him what dish he desires. Moreover, it is essential for the user to be able to partially control the robot behavior. Imagine for example that the user is talking with someone else when the feeding task is being performed. The user would want to pause the task for a short period of time and resume it later. Therefore, the user needs to be able to communicate to the robot when he wants to pause the task and to resume it. Another possible

[2]rqt is a Qt-based framework for GUI development in ROS

scenario is that the user does not want to continue eating because for example he/she is full. In this situation it is also essential that the user can communicate this request to the robot. Therefore, these are the four essential requests that have been taken into account.

In order for the user to be completely comfortable with the task performance, he needs to know the intentions of the robot at all times. It is then, essential that the robot can communicate them to the user. However, we do not want to over inform the user and thus the robot will only inform from the most important actions he is going to perform. The information he is going to give to the user is the following one:

- Inform the user that the feeding task is going to start and ask for the desired dish
- Inform the user that food has been grasped and that the robot is next going to feed the user
- Apologize to the user if some unwanted impact has occurred
- Inform the user that the feeding task has finished

6.3.2 Visual sensing

Information obtained through images is of vital importance. It allows the robot to be able to obtain information about the world state and the user state and movements or visual requests.

In order to be able to obtain all the relevant information of the system two cameras are used. To obtain information about the user with the least possible number of occlusions a Creative Senz3D camera is mounted on the gripper of the robot. The obtention of the images with this camera is done in the ROS node DepthsenseCamera explained in Section 6.1.

It allows to always see the user unless the camera is too close to his face. However, if this camera was to be used to obtain information about the dish state, the robot would have to perform extremely unnatural movements. This is why in order to obtain the information about the dish position another camera has been used. Concretely, an Xtion camera has been mounted over the dish to obtain its position. The obtaining of the images with this camera is done in the ROS node XtionCamera explained in Section 6.1. This camera has also been used to obtain the information about the position of the testing manikin. This manikin

has been set to simulate the user during the firsts tests. It has been set at the user's chair at proximately the same size of the user's head.

The obtainment of the user requirements through visual sensing has been detailed in Section 6.2.1 and the obtainment of the positions of the elements is explained below.

6.3.2.1 Positions of elements

The robot movements adapt to the positions of the important elements of the scene as has been explained in Section 6.5. For example, the food grasping movement adapts to the dish position, while the feed movement adapts to the user's mouth position. Therefore, these positions need to be known. The following subsections give a detailed explanation of how these positions are obtained.

Mouth position

The user mouth position is obtained in the ROS node ObtainMouthPosition. As has been previously explained, the node ObtainMouthPosition receives the mouth position from the MouthDetection node through the appropriate topic. It also receives the depth images obtained with the DepthsenseCamera node through the corresponding topic. With these information it is capable of computing the position of the mouth. However, this position consists on x and y of the image in pixels and the z of the image in cm, which is called camera coordinates.

In order for the robot to move to the mouth position, this position needs to be in point coordinates which are the location in 3D space. Therefore, we need to convert the position from pixel coordinates to point coordinates. This conversion can be done with the following equations that have been obtained from [26]:

$$x_{3D} = z_{image} \cdot (x_{aux} \cdot f + 2 \cdot K3 \cdot x_{aux} \cdot y_{aux} + P1 \cdot (r + 2 \cdot x_{aux}^2))$$

$$y_{3D} = z_{image} \cdot (y_{aux} \cdot f + 2 \cdot P1 \cdot x_{aux} \cdot y_{aux} + K3 \cdot (r + 2 \cdot y_{aux}^2))$$

$$z_{3D} = z_{image}$$

where,

$$r = x_{aux}^2 + y_{aux}^2$$

$$f = 1 + K1 \cdot r + K2 \cdot r^2 + P2 \cdot r^3$$

$$x_{aux} = (x_{image} - Cx)/Fx$$

$$y_{aux} = (y_{image} - Cy)/Fy$$

and Cx, Cy, Fx, Fy, K1, K2, K3, P1 and P2 are the are the internal construction parameters of the camera which are called the intrinsics of the camera. It is important to note that it is possible to do this conversion directly because the RGB images we are using are indeed depth registered images. Ergo, depth images that have the corresponding RGB image superposed. Therefore, the x and y coordinates of the depth registered image are the same ones as the depth image.

After applying this equations in the MouthDetection node, the mouth position in camera coordinates is known and can be published to the corresponding topic.

Dish and manikin's mouth positions

Before trying the prototype with real users it is important to test it without them. In order to achieve it a 3D carton head has been used. This head can be observed in Figure 6.2. As can be observed, it has a hole on his mouth to be able to reproduce the feeding movement and it also has a carton on his head where an AR tag is fixed. As can be observed in Figure 6.3, the dish also has an AR tag attached. These AR tags are used to obtain the manikin mouth and the dish positions.

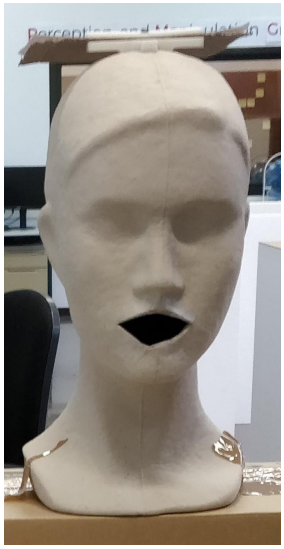


Figure 6.2: Carton face used as manikin
Source: Own compilation

These positions are computed at the DishPosition ROS node. This node receives the images from the XtionCamera node. With this information and using the ar_track_alvar package it calculates the position of the AR tags relative to

the camera in camera coordinates. Knowing this information it is effortless to calculate the position of the center of the dish and the position of the manikin's mouth as it is a simple and fixed transformation. Concretely, there are 13,5cm in the x axis and -2,5cm in the z axis from the dish's AR tag to the center of the dish and the manikin's mouth is at -15cm in the y axis and -17,5cm in z axis from the AR tag.

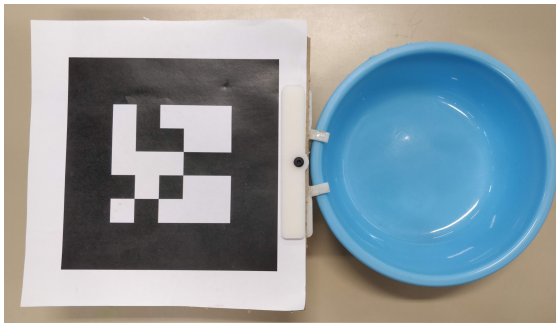


Figure 6.3: AR tag fixed on the dish
Source: Own compilation

Cameras and robot spatial link

It has been explained how to obtain the mouth and dish positions. However, these positions are relative to the camera that captures the images used to process the position. As the movements are relative to the robot, there exists the need to know the relation between the cameras and the robot. These relationship is called the extrinsics of the camera.

The camera needs to be calibrated to be able to obtain its extrinsics. At IRI laboratories this calibration is done through the Hand-Eye calibration technique. It is based on the simultaneous estimation of the unknown transformation from the robot coordinate system to a calibration pattern coordinate system as well as the transformation from the camera to the robot's hand coordinate system. With the cameras extrinsics the spatial link between the robot and the camera is obtained.

6.4 Autonomous behavior

The key factor of this project is the behavior of the robot. It has to be autonomous as we only assume the user to be able to move his neck, mouth, eyes and be able to speak (as has been stated before). However, the user has to be comfortable during the task development and thus the robot has to adapt to the user's requests.

Subsection 6.4.1 gives a detailed explanation of how the robot behavior is decided and Subsection 6.4.2 details how the this behavior is implemented inside the ROS architecture.

6.4.1 Robot behavior explanation

The robot behavior in this project has been implemented with an automatic planning and scheduling or simply called a planner. A planner is capable of finding the best strategy to achieve a desired goal given a set of actions which is exactly what we need for this application. Moreover, as explained in [27] the usage of a planner provides a robust behavior with less demonstrations.

In order to obtain the best strategy that the robot has to follow, the Metric-ff planner has been used. It is a simple planner and thus solutions are found faster. The computation of the solution needs to be as fast as possible as feeding is a real time application.

In order for the planner to be able to compute a solution, it needs to receive a domain and a problem. The solution is the sequence of ordered actions that the robot needs to follow in order to achieve the desired goal. The domain is the definition of the state variables and the actions which the system is provided with. Finally, the problem defines the current state of the system, the desired final state and what is the function to optimize. Appendix B gives a detailed explanation of the implemented domain and problem. Moreover, the implemented domain and one example of a problem can also be found in Appendix B.

This behavior is tested in Section 7.1. The experiments conducted test the applicability of the design in a real time application such as feeding and the results that it obtains.

6.4.2 Robot behavior implementation inside the ROS architecture

The previously explained behavior needs to be implemented inside the ROS architecture. In order to do this implementation, the Rosplan [28] framework has been

used. This framework provides a set of nodes which provide planning, problem generation, and plan execution.

First of all, Rosplan receives the desired domain and problem. It parses them and fills its internal Knowledge Base structure which is in charge of containing all the information about the state variables. Next, it passes the domain and problem to the desired planner. It obtains the planner results and stores the set of actions that need to be followed to achieve the desired goal. Once it knows all the set of actions, it dispatched the action that needs to be executed. The Knowledge Base can be updated during the execution of the tasks. Moreover, it is possible to tell to the system if an action has been achieved or not.

In the ROS architecture that is presented in Section 6.1, it is possible to observe all the nodes that send information to Rosplan. It is important to note that in the presented architecture Rosplan has been indicated as only one node. This has been done to be able to clarify the the interaction between the developed nodes.

As it can be observed, the HeadDetection node sends information to Rosplan about the head orientation of the user through the corresponding service. The MouthDetection node sends information about the mouth openness of the user. The ObtainMouthPosition node tells Rosplan if the camera is too close to the user's face to be able to obtain proper detections and it also tells Rosplan if the spoon is inside the user's mouth or not. The AudioController node tells Rosplan which is the dish the user desires and it also tells Rosplan if the user wants to end the feeding task of pause it for a while. The FeedingBehavior node actualizes all the other state variables which the Knowledge Base has.

The FeedingBehaviorStartHandler node is a special case as it does not update any variable in the Knowledge Base. It is in charge of sending the domain and the problem to Rosplan and to tell it when he has to parse the problem, generate it or dispatch the actions. When the current strategy is not valid because some state variable has a different value than expected, the plan is not achieved. In this situation the FeedingBehaviorStartHandler node tells Rosplan to re-plan the solution. Then, Rosplan gets the current state of the Knowledge Base and the desired goal and calculates the new actions to follow. Moreover, this node is also needed to do the dispatching of the actions. In order to tell Rosplan to dispatch the next action a service is used. As has been previously explained, service calls are blocking. Therefore, when the action is being performed, the FeedingBehaviorStartHandler remains blocked until the action is achieved or failed. Because of this factor, the action dispatch can not be done in the main node.

The FeedingBehavior node receives the actions that the Rosplan dispatches.

When it receives them, it calls the appropriate nodes which can be the Robot-Movement node or the AudioController node. Once the action has been achieved, the FeedingBehavior updates the changes occurred in the domain due to the action execution and tells Rosplan if the action has been completed or not.

Using this method, Rosplan is the only element of the Architecture that knows exactly the state of the system and the actions that need to be followed. However, it does not perform the actions. It is the FeedingBehavior node the one in charge of handling the action execution and updating its results. This results in an excellent encapsulation of the code.

6.5 Robot movement

There are several ways to move the Barrett WAM® robot and each of them has its advantages and disadvantages.

The most basic way to move the robot is through predefined trajectories. In this case, the robot receives the desired rotations of all the joints in each time interval. This type of movement allow the robot to move naturally and be completely predictable as the exact positions have been predefined. However, the problem with this type of movement is that it does not adapt to any position as the movement is always the same.

Using inverse kinematics the robot receives the position in cartesian of the end-plate of the robot. With this information it is capable of computing the joint rotations it needs to have in order to achieve this position. Therefore, inverse kinematics allow to convert a cartesian trajectory to a joint trajectory. However, the trajectory in cartesian must be known which is only feasible for simple trajectories where the only important points are the starting and ending point. Moreover, inverse kinematics presents another problem: as the robot used is redundant, there exist many possible joint rotations for the desired end-plate position. This can result in extremely unnatural movements or in abrupt movements. The abrupt movements are due to changes in the chosen solution between two points of the trajectory. Therefore, the movements adapt to desired positions but they can be unnatural.

These are the two classical solutions to the robot movement problem. However, there has been recently a lot of effort in finding a better solution to this problem. Most of the efforts have been set in applying machine learning techniques to solve this problem. In this project we are going to use the technique proposed in [29]. This solution uses Gaussian Mixture Model (GMM) to determine the trajectory that the robot has to follow.

In order to be able to execute the feeding task, the robot needs to know the position of the user's mouth and the position of the dish. Moreover, it also needs to know the location of the reference positions where it stops or waits. These positions are listed below and can be observed in Figure 6.4:

- **Home:** is the position where the robot rests when it is switched off.
- **Pre-grasping position:** is the position where the robot waits before grasping the food.
- **Pre-grasping position:** is the position where the robot waits for the user to open his mouth before feeding him

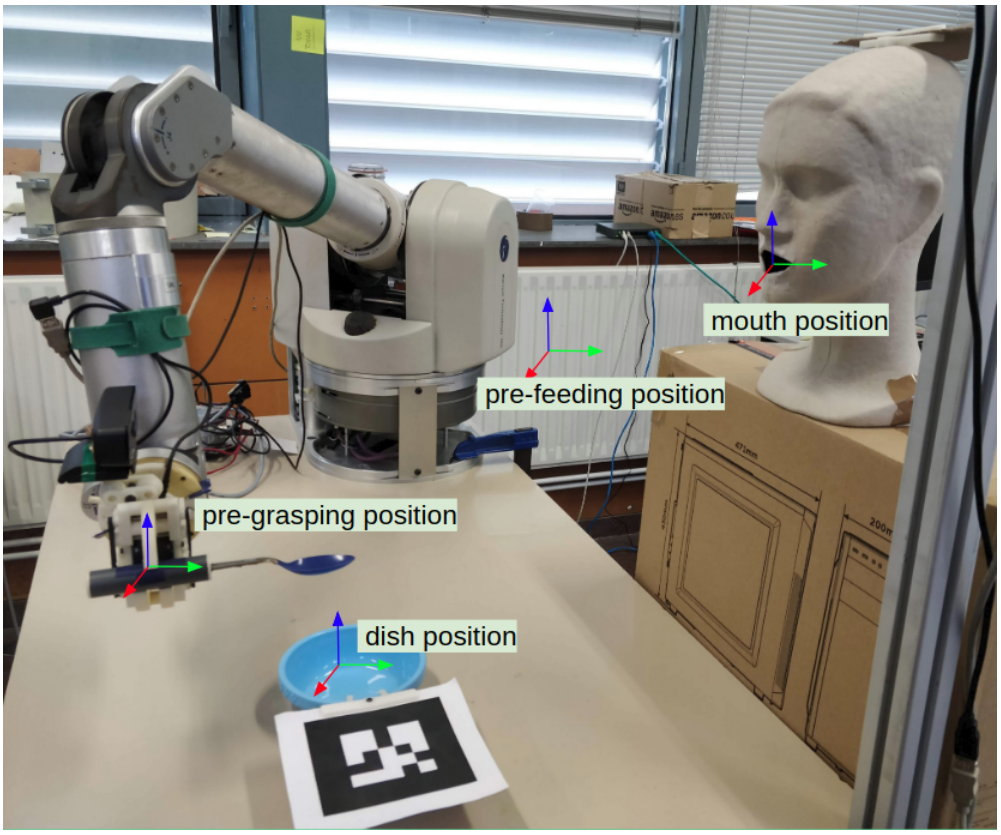


Figure 6.4: Relevant reference positions during the feeding process
Source: Own compilation

In this project, all the trajectories that have to be executed given the desired positions are computed in the RobotMovement ROS node. This node receives

the dish position from the DishPosition ROS node through the appropiate topic. It also receives the user's mouth position and the pre-feeding position from the ObtainMouthPosition ROS node. Once it obtains the trajectories it calls the ROS node Robot which is in charge of executing the trajectories.

6.5.1 Direct movements

As has been stated before, the most basic type of robot movement is through passing it the exact joint rotations that it needs to have each period of time.

At IRI laboratories trajectories in joints can be learned by demonstration. This technique allows the programmer to manually move the robot to teach it the desired trajectory. While this movement is being performed the joint rotations are saved at each period of time. Once this trajectory has been obtained, the robot is capable of exactly reproducing it.

This allows a natural and predictable movement as the programmer is the one to decide how it is exactly. However, it can not adapt to changing positions. Therefore, this technique can not be used for all the performed movements as some of them require adaptation to desired positions.

The different trajectories which are obtained with this technique or directly obtained with minor computation are listed below.

- **From home to pre-grasping position:** this is the trajectory the robot performs at the start of the feeding task. In this trajectory, the robot moves from home to the pre-grasping position which is predefined. It may seem that this trajectory needs to be able to adapt to the dish position. However, as the trajectory in charge of grasping the food can adapt to the dish position, it is not relevant that this ends just over the dish position. This trajectory has been obtained with the learning from demonstration technique.

- **From pre-feeding position to pre-grasping position:** this movement moves from the current pre-feeding position where the robot is waiting to the pre-grasping position. This trajectory is calculated before it is executed as the pre-feeding position depends on the mouth position of the user.

- **From inside the mouth to pre-feeding position:** when a force has been exceeded, the robot must go back to the pre-feeding position and wait there for the user to allow the robot to continue the feeding task. As the previous trajectory, this trajectory is calculated before it is executed as the

pre-feeding position depends on the mouth position of the user. As the camera is too close to the user's face to obtain images of the whole face, the pre-feeding position that uses this trajectory is the previously obtained one.

- **From the pre-grasping position to home:** this trajectory is in charge of moving the robot from the pre-grasping pposition to home. As both positions are predefined and do not change from execution to execution, this trajectory can be learned by demonstration. Therefore, this trajectory has been prerecorded and it only needs to be executed.

6.5.2 Inverse kinematics

Inverse kinematics allows the robot to adapt to desired positions. However, as it has been explained before, this robot is redundant and thus there exist many possible solutions on how the robot can achieve an end-plate position. If the trajectory is not simple, the robot can change from one solution to another during the movement. If the two solution have significantly different joint rotations, abrupt movements may occur. This abrupt movement can scare the user. Moreover, for safety reasons, the Barrett WAM® has its acceleration and velocities limited and if they are exceeded the current power is cut from the robot and thus it falls. Therefore, when the inverse kinematics changes from one solution to another one, the maximum velocities and/or accelerations may be exceeded resulting in a downfall of the robot which may harm the user. Moreover, the solution found by the inverse kinematics algorithm can not be predicted and can result in unnatural movements. This can lead to an uncomfortable task performance for the user.

Inverse kinematics is based on computing the joints rotations given an end-plate cartesian position. In order to obtain an accurate enough solution a reference position in joints is also provided. The inverse kinematics algorithm will try to find a solution as much similar to the reference position as possible. However, there does exist a trade-off between time spent to find the solution and similarity from the solution and the solution. As feeding is a real time application where positions constantly change, the solution must be found fast. Therefore, it may result in a not accurate enough resulting trajectory.

Therefore, this type of movement has only been used when it was indispensable. The movement that uses this method has to be simple and be able to adapt to a cartesian position. In this ways, solutions may not take long to be found and may be similar enough to the reference.

Concretely, this method has only been used in the trajectory where the robot goes from the pre-grasping position to the pre-feeding position. This trajectory is

a straight line and thus it is simple to find a solution for the inverse kinematics algorithm. Moreover, this movement needs to be able to adapt to the pre-feeding position. As it may be unnatural for the user if the pre-feeding position is not close to his mouth.

6.5.3 Learned movements

Recently, there have been many attempts to find a better solution to the robot movement case. In this project, the solution presented in [29] has been used. This article uses GMM [30] to obtain the trajectory that goes through the desired position. A GMM is a probabilistic model where all the data points are generated from a finite number of Gaussian distributions with unknown parameters.

In order for the GMM to be able to correctly learn the trajectories they have been given in cartesian coordinates. Because the trajectory in cartesian is very similar in each iteration. However, the trajectory in joints is completely different in each iteration as it extremely depends on the initial position and the desired position.

Before obtaining the GMM, we have to obtain the input information that it receives. As trajectories have been represented in cartesian space, there are 6 varibles per trajectory (x, y, z and the orientation of the end-effector of the robot). Each variable has been represented with 15 Gaussians as can be observed in Figure 6.5. In this figure the horizontal axis represents the time whereas the vertical axis represents the value of the variable. It is important to note that the total time of the trajectory has been normalized. This helps creating a general model of the desired trajectory.

Therefore, the position of the robot at the instance of time t in, for example, the x axis can be obtained with the following formula. It is important to note that this equation serves for the six variables.

$$x(t) = \sum_{N=1}^{15} \psi_N(t) * w_N$$

where,
$x(t)$ is the position of the end-plate of the robot in the x axis at the instance of time t
$\psi_N(t)$ is the value of the Gaussian N at the instance of time t
w_N is the weight of the Gaussian N

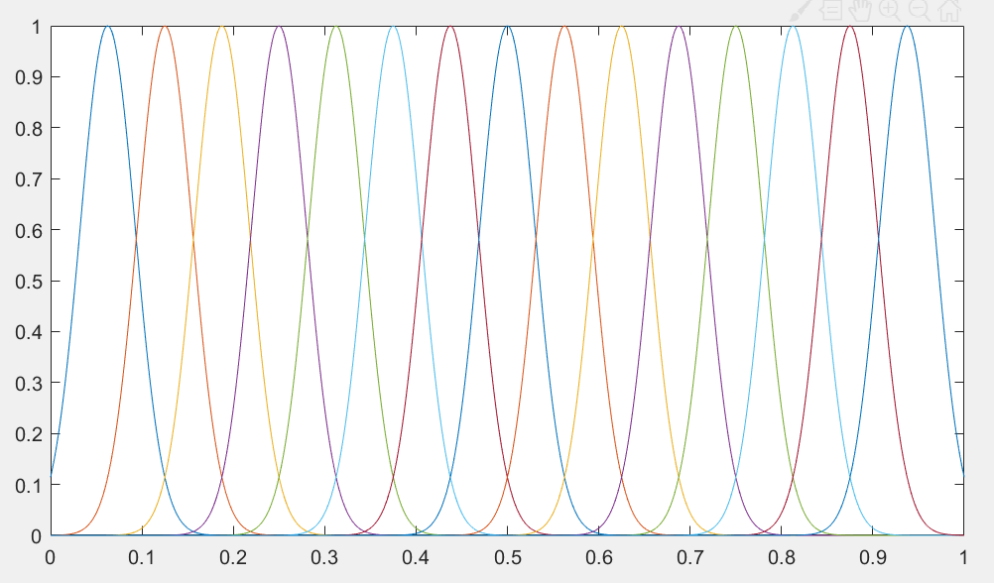


Figure 6.5: Example of the representation of a variable over time with 15 Gaussians
Source: Own compilation

These 15 Gaussians have a fixed median and variance. Therefore, it is possible to calculate the vector of weights given a trajectory. The input variables of the GMM are the vector of weights of each of the 6 variables (the three positions and the three rotations of the end-plate of the robot) and the reference position (which could be, for example, the position of the dish).

After the learning process has concluded, we can condition the learned trajectory to our reference position in order to obtain the vector of weights. This is achieved using Gaussian Mixture Regression. With the vectors of weights we can obtain the value of the variables at each instant of time with the formula shown above.

Initially, it was intended to also condition this movement to the current robot position so the first position of the trajectory was the current robot position. However, it was evaluated that the results obtained were not satisfactory. Therefore, this contextualization has not been done on the prototype and thus, before executing these trajectories, the robot will be sent to their starting point.

It is important to note that these trajectories are obtained in cartesian space as has been explained before. Therefore, in order to obtain the trajectory in joint rotations, inverse kinematics is used. The references passed to the inverse kinematics algorithm are the medians of the joint rotations of all the trajectories

that have been used to train the GMM.

This method has been used in the two complex trajectories that these application requires. These trajectories are:

- **Get food:** this trajectory starts in the pre-grasping position, grasps the food and ends in the pre-grasping position. The variables that are given to the GMM are the x and y of the dish as they are the most representative ones.
- **Feed:** this trajectory starts in the pre-feeding position, feeds the user and ends in the pre-feeding position. The variables that are given to the GMM are the x, y and z of the user's mouth as they are the most representative ones.

These trajectories are tested in Section 7.2. The experiments conducted test the correctness of the results of this technique for the trajectories get food and feed.

7. Experiments

After implementing a system it is of vital importance to validate it. This chapter details the experiments developed to validate the proposed prototype. In Section 7.1 the validation of the presented behavior is performed. In Section 7.2 the validation of the learned movements is performed.

7.1 Experiments of the robot behavior

The key factor of this work is the robot behavior that has been implemented. In order to be able to assure that the chosen methodology is appropriate for this application several experiments have been performed. First of all, the execution time is tested in order to define if the methodology used can be used in a real time application such as feeding. Then, the correctness of the resulting behavior and its cost is evaluated.

7.1.1 Execution time

This experiment is in charge of testing the execution time of the implemented behavior. In robotics it is of vital importance that the execution of the tasks are done at real time. It is specially in applications like the one in hands where direct contact exists between the user and the robot. Therefore, if it takes too long to plan the trajectory, the used methodology will also be discarded.

In order obtain the needed information, 10.000 executions of the problem have been done. This executions have different feasible starting situations. However, the goal state is always the same which is the goal in the all the real executions. With this situations, we have simulated different possible situations where the robot has to plan or re-plan its behavior.

The time spent in all the executions has been calculated and can be observed with the Boxplot shown in Figure 7.1. As can be observed, the median execution time is extremely low as it is only of 0.0032 seconds and the planning does not usually take more than 0,0035 seconds. Therefore, this experiment shows that the planning time of the behavior implemented is so fast that can be used in a real time application such as the one in hands.

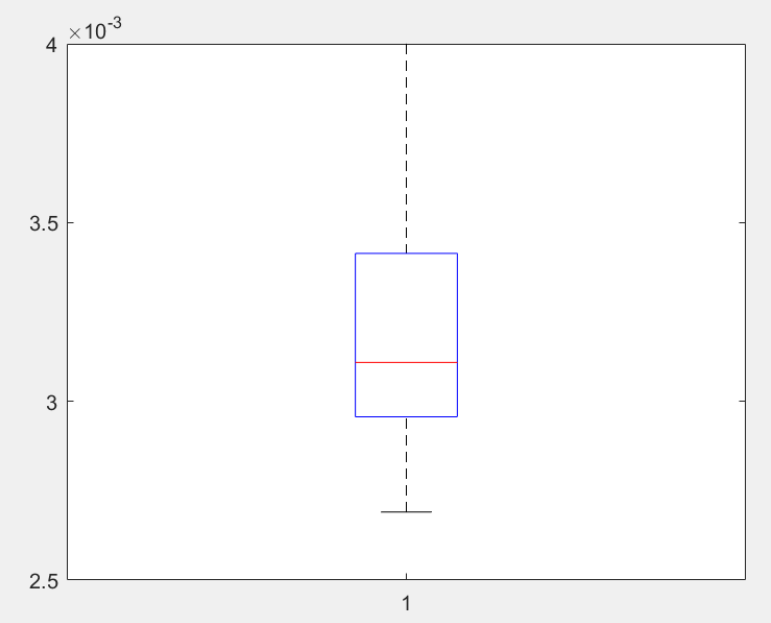


Figure 7.1: Boxplot of the execution time of the behavior experiment
Source: Own compilation

7.1.2 Behavior planning and costs

The purpose of this experiment is to demonstrate the correct development of the behavior that has been implemented.

First of all we are going to study the decisions that the planner takes depending on the current situation which is detailed in Table 7.1. More information about the meaning of the predicates included in this table can be found in Appendix B. Figure 7.2 shows a graphic that informs about the set of actions that have been followed in different situations. This graph only shows the part of the food grasping and feeding of the task.

In this figure, situation 1 represents the most basic set of actions which are followed when the user is looking at the robot and has his mouth open. Situation 2 represents the set of actions the planner has returned when the user is not looking but has his mouth open. On the other hand, the situation 3 represents the set of actions returned when the user is looking at the robot but does not have his mouth open. Finally, situation 4 represents the set of actions the robot follows when, after every movement the robot performs, the user asks it to wait for a while.

Situation	True predicates
Situation 1	(mouth open) (head forward)
Situation 2	(mouth open)
Situation 3	(head forward)
Situation 4	(mouth open) (head forward) (wait)

Table 7.1: True predicates at the start of the four example situations of the decision graphic

Source: Own compilation

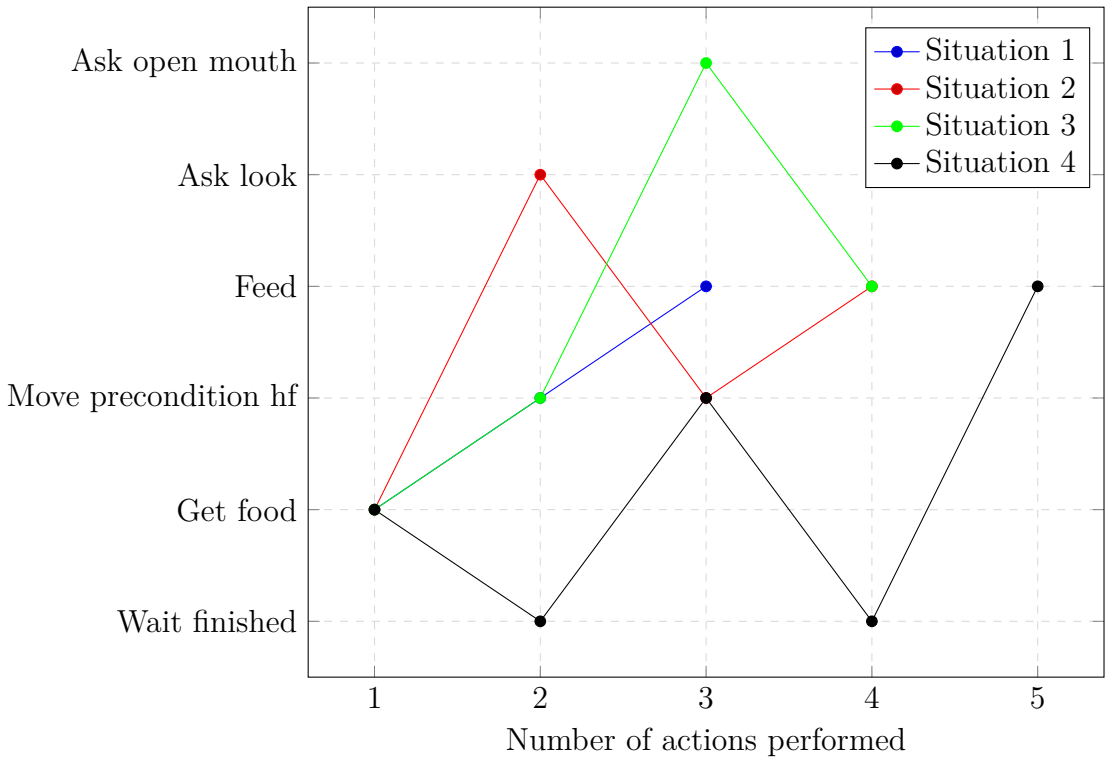


Figure 7.2: Part of the possible behavior decisions
Source: Own compilation

We can observe in these situations that the planner always chooses the same sequence of actions that a person would perform. We can say then, that the plans returned simulate the human behavior.

Only some of the possible ramifications of the behavior have been shown in this graphic as there are endless possible situations. For example, if the user asks to stop the feeding process the robot should go to home, throwing the food first if the spoon is full. Another example could be that the feeding action is not successful and thus, when it finishes, the spoon is still full. It would be an enormous job for a programmer to take into account all the possible states and ramifications in order to encode them in a a Machine State. Moreover, this is still a pretty simple application but if more improvements were applied, it would become an even harder job. Therefore, using a planner enormously eases the programmer job as its implementation is easier and it returns the expected results.

We are now going to study the effect of the starting situation of the planner to the cost of the returned plan. A summary of the true predicates at the start of each situation can be observed in Table 7.2. More information about the meaning of the predicates included in this table can be found in Appendix B. It is obvious that the more actions the planner has to perform, the more cost the planner will have. However, the cost of the actions vary from cost 1 to cost 10 which causes different incrementation of the cost. A lower cost of an action reflects that we want the robot to prioritize it. For example, the basic action to move to a waypoint has a cost of 10. However, the action that goes to a waypoint only if the user has its head forward has a cost of 5 and the action to ask the user to look at him has a cost of 1. Therefore, we can observe that it has a lower cost to first ask the user to pay attention and then move than to simply move which is the behavior that we expect from the robot.

Situation	**True predicates**
Situation 1	-
Situation 2	(mouth open) (head forward)
Situation 3	(wait)
Situation 4	(dish wanted is known)

Table 7.2: True predicates at the start of the four example situations of the cost graphic

Source: Own compilation

In Figure 7.3 it can be observed the cost of the planner in four different situations. All this situations start with the robot at home and with only one food grasping in the dish. Moreover, all of this situations end at home having fed the user one time. What changes over this situations is the state of the user and the requests he makes.

The first situation (marked with blue) consists on the most normal task development. In this situation, the user is not looking at the robot, his mouth is not open and he has not decided which dish he wants to eat from yet. Therefore, the behavior decides that the robot has to perform the following set of actions:
1: INFORM START AND ASK DISH_WANTED
2: GO TO WAYPOINT FROM HOME TO PRE-GRASP
3: GET FOOD FROM D2
4: ASK LOOK
5: GO TO WAYPOINT PRECONDITION HF FROM PRE-GRASP TO PRE-FEED
6: ASK OPEN MOUTH
7: FEED
8: GO TO WAYPOINT FROM PRE-FEED TO PRE-GRASP

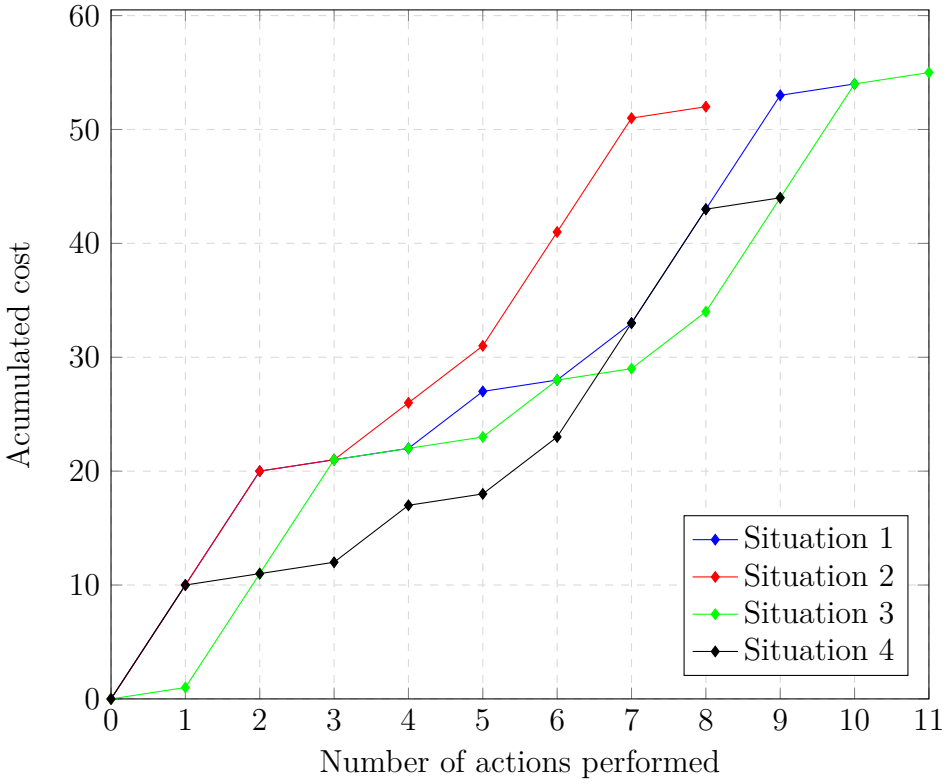


Figure 7.3: Planner cost in different situations
Source: Own compilation

9: GO TO WAYPOINT FROM PRE-GRASP TO HOME
10: AUXILIAR FINISH NO FOOD
More information about these actions can be found in Appendix B. As it can be seen this set of actions make perfect sense and are the actions and the order a person would choose in this situation.

The second situation (marked with red) has two differences from the first one. In this case, the user is looking towards the robot and he has his mouth open. As can be observed in Figure 7.3, the second situation has two less actions that the first one. This is because in this case the robot does not have to ask the user to look at him or to open his mouth. The rest of the planning is exactly the same as the first situation as it logically has to be. We can observe that the cost of the plan is lower than in the first situation but not extremely lower. This is due to the fact that the two actions that are not performed in this plan (ask look and ask open mouth) have a low cost. Their cost is low because we want the robot to prioritize these actions in order for the user to feel comfortable during the task development.

The third situation (marked with green) starts with the same situation than the first one but with one change: in this case the user asks the robot to wait for a while. In response, the first action that the robot has planned is to ask for the user to end the waiting period. After this request, the robot executes the exact same actions than in the first situation. This es exactly the desired behavior that we expect from the robot as it does not have to move when the user has asked him to wait. Therefore, it is normal that the robot, before executing any movement, asks the user if it can resume the feeding task. It is important to note that, in the planner, this action assumes that the user will always accept to resume the feeding task. If this were not the case, the action will not be successful and thus a re-planning would be done. This plan is the most expensive due to the robot having to ask the user if it can resume the feeding task. This additional action has a low cost as we want the robot to prioritize it. Therefore, the cost of the plan has a smaller increment compared with the cost of the plan of the first situation.

The last situation that has been studied (marked with black) is like the first one but, in this case, the user has already told the robot which dish he desires. In this case, the robot follows the same plan than in the first situation but omitting its first action. Ergo, it does not execute the inform start and ask dish wanted action as the robot already knows what dish the user wants. Therefore, the resulting behavior is the one which could be expected. In this case the cost is also lower than in the first case as the robot performs one action less. Moreover, this action has the lowest cost as the inform start and ask dish wanted action has the highest

cost.

Therefore, it can be concluded that the planner helps build strong designs which always choose a suitable strategy to follow.

7.2 Experiment of the learned trajectories

The only trajectories that need to be deeply studied are the ones that are obtained through learning. It is clear that direct trajectories do not present any challenges as they are predefined by the user. Moreover, the trajectories obtained only with inverse kinematics in this prototype are very simple. Therefore, they do not require a deep experimentation. The only trajectories which require this testing are the ones that have been obtained through learning.

The testing of the trajectories has been developed with a manikin instead of a user as their safety could not be assured yet. Therefore, its testing has been developed with a manikin instead of real users. The setup in this experiment can be observed in Figure 7.4. It can be observed that the manikin used has an AR tag on the top of his head which is used to obtain its mouth position.

Figure 7.4: Setup of the experiment of learned trajectories
Source: Own compilation

With this setup, the get food and feed trajectories have been tested. Before testing the trajectories with the robot, it has been checked that they are correct. The following subsections show the result of the previous checking and detail the results of the testing of the trajectories get food and feed.

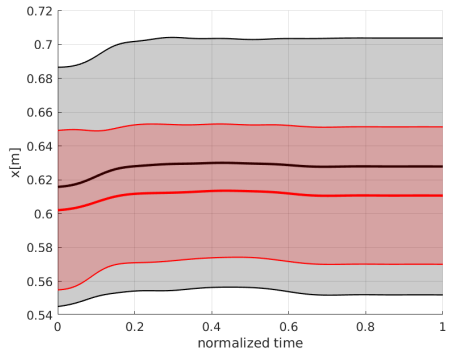
7.2.1 Get food trajectory

As has been stated before, it is important to first check the correctness of the trajectory before trying it with the robot. In Figure 7.5 it can be observed the learned trajectory and the learned trajectory after being conditioning to the dish position. Concretely, each sub-figure shows, for every instance of time, the mean and the deviation of the learned trajectory in black and of the conditioned trajectory in red. Therefore Sub-figure 7.5 (a) shows the mean and the deviation of the position of the x axis of the end-plate of the robot in every instance of time for the learned and conditioned get food trajectories.

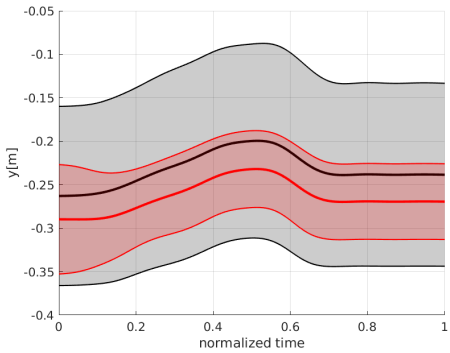
As can be observed, the conditioned trajectory is extremely similar to the learned trajectory. Therefore, we can assure that the trajectory is correct. It can be observed that the means of the three rotations and the z axis of the learned and conditioned trajectories are extremely similar. This is due to the fact that, in this trajectory, the movement performed is always the same in these four variables. It only varies the position of x and y of the robot as it directly depends on the position of the dish. However, the x and y positions of the dish are not very correlated to the rotations and the z positions of the end-plate of the robot. We can observe then that the mean and deviation of the conditioned trajectory are a little bit different than the ones of the learned trajectory as there exists a major correlation between the context variables and the x and y of the end-plate of the robot. As the mean of the conditioned trajectory is inside the deviation of the learned trajectory, we can assure that the prediction is correct.

Therefore, it is safe to try the trajectory with the robot. However, before testing it with real users it has been tested without them in case any anomalies occur.

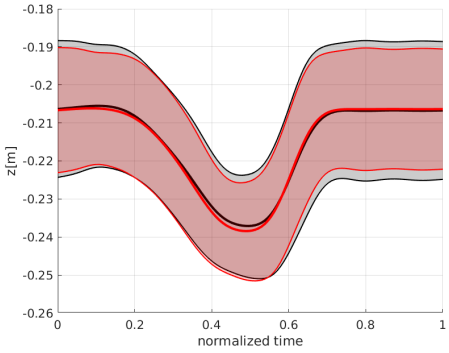
After several trials of the trajectories, it has been observed that it correctly developed the moving action on all the trials. Therefore, it has been concluded that the proposed method to obtain trajectories given a reference position is reliable for this trajectory.



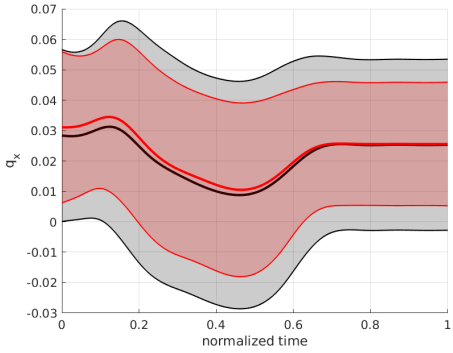
(a) Position relative to the x axis



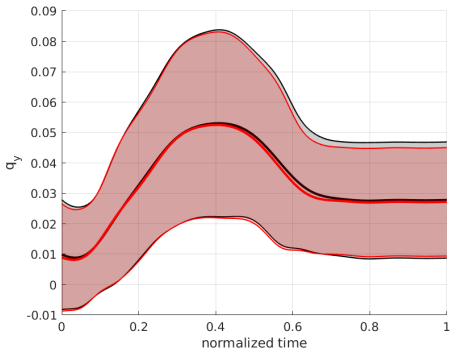
(b) Position relative to the y axis



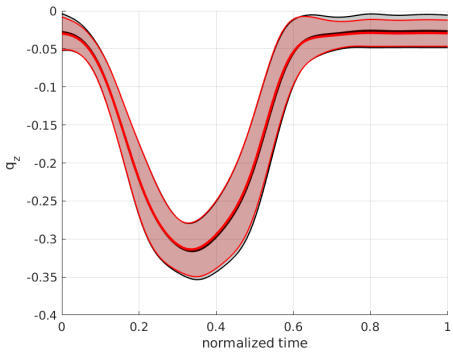
(c) Position relative to the z axis



(d) Position relative to the q_x rotation



(e) Position relative to the q_y rotation



(f) Position relative to the q_z rotation

Figure 7.5: Learned (black) and conditioned (red) get food trajectory
Source: Own compilation

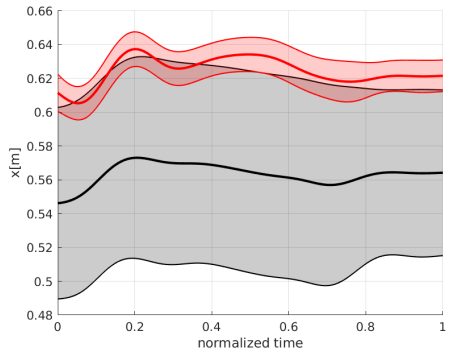
7.2.2 Feed trajectory

The experiment performed in for the feed trajectory is exacly the same as the one that has been performed for the get food trajectory. The get food experiment is explained in Subsection 7.2.1. As a reminder, before trying the the trajectory with the robot it has been checked its correctness.

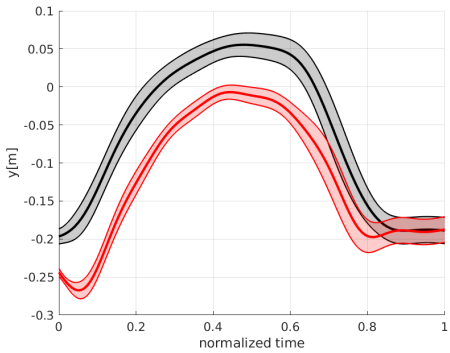
It can be observed in Figure 7.6 the learned trajectory and the conditioned trajectory to the mouth position. Concretely, the mean and the deviation of the learned trajectory are marked with black whereas the mean and the deviation of the conditioned trajectory are marked with blue.

As it can be observed, all the variables of the learnt trajectory significantly change when the trajectory is conditioned to the mouth position. This is due to the fact that all the variables are correlated with the x, y and z of the mouth position. In some variables the mean of the conditioned trajectory is partly outside the deviation of the learned trajectory. However, the shape of the both trajectories are more or less similar for all the variables except q_y. However, the the absolute change of its value is extremely slow and thus its change on shape is not very relevant. Therefore, it can be concluded that the trajectory seems to be correct.

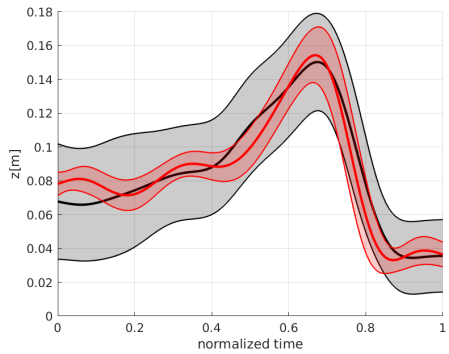
This trajectory was then tested on a robot with the setup explained previously. After several repetitions, it was observed that all the trajectories correctly developed the feeding action. Therefore, it we can conclude that the proposed method to obtain the feed trajectory given the mouth position as a reference is reliable.



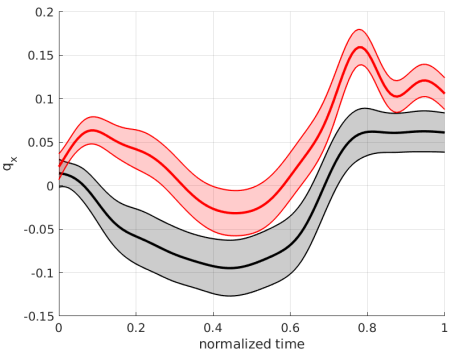
(a) Position relative to the x axis



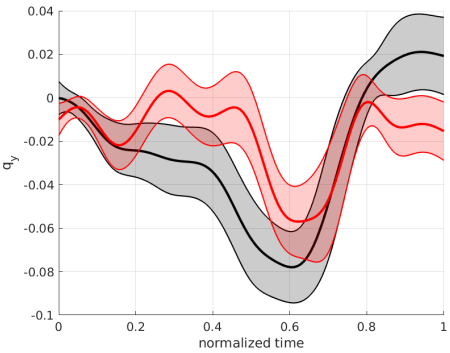
(b) Position relative to the y axis



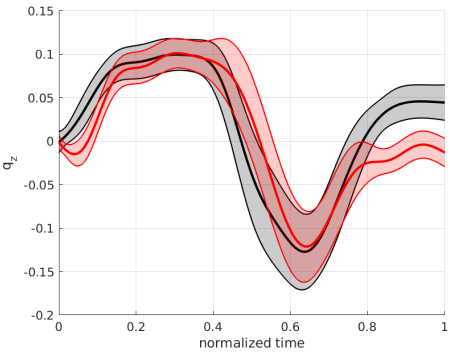
(c) Position relative to the z axis



(d) Position relative to the q_x rotation



(e) Position relative to the q_y rotation



(f) Position relative to the q_z rotation

Figure 7.6: Learned (black) and conditioned (red) feed trajectory
Source: Own compilation

8. Project planning

This chapters gathers the information related to temporal planning which is explained through a Gantt chart. Concretely, the project is divided into tasks. Each task is described in terms of what has to be done in each activity, its length, and the resources needed to complete it.

It is important to comment that the original planning has been revised and updated as a result of the evolution of the project. The new planning can be observed in Section 8.5. Moreover, the duration of the project is also subject to change due to the usage of the methodology described before which implies a constant update on the objectives. Therefore, alternatives solution to these changes on the planning will be explained.

The starting date of this project was the 10th of September, while its deadline is the 24th of January. Therefore, the estimated length of the project is about 4 months with a workload of 4 hours every working day.

8.1 Task description

The basic human resource for this project is the developer with the supervision of the director and co-director. The basic material resources are a computer and a robot capable of developing the task. In order to ease the planning of the project, it will be divided into the following tasks.

8.1.1 Preamble

This task is in charge of defining the project. The first thing that needs to be done is a literature overview. Once we understand the existing robots and applications capable of helping a person to self-feed, it is possible to concrete the objectives of this project. Finally, in order to decide the exact scope of the project and lay its foundations, the software architecture will be established.

8.1.2 Robot behavior

This task is in charge of deciding how the robot is going to behave and how to implement it. Therefore, the first two things to do will be determining the robot behavior itself and deciding the Artificial Intelligence(AI) technique that is going

to be used to implement it. The next step will be to implement the robot behavior module and to test it. Finally, the documentation of this task will be done.

8.1.3 Communication between modules

This task is in charge of communicating all the modules. Concretely, it needs to communicate with the robot behavior module, supplying it with the requested information, and call the required action. Therefore, the communication module needs to be implemented, tested and documented.

8.1.4 Detect the user state visually

This task is in charge of detecting information about the user state visually, such as the mouth state or head orientation. The first thing that has to be considered is the decision of the libraries that are going to be used. Afterwards, the visual user detection will be implemented, tested and documented.

8.1.5 Audio module

This task is in charge of talking to the user to determine his requests. In order to talk with the user, an additional material resource, such as an Amazon Echo device, is required. This tasks starts with the decision of this additional resource followed by the implementation, testing and documentation of the audio module.

8.1.6 Robot movement

This task is in charge of moving the robot. It will receive the requests from the communication module and will call the robot movements accordingly. Therefore, this task consists on the implementation, testing and documentation of the robot movement module.

8.1.7 Safety

This task is in charge of guaranteeing the safety of the robot performance. In order to achieve this objective, additional resources, such as a force sensor, are needed. Therefore, this tasks consists on the decision of the safety measure that is going to be applied and the tools that it needs. It is also includes the implementation, testing and documentation of the safety module.

8.1.8 Experiments

This task consists on the definition of the experiments, its performance and documentation. Some of the experiments will be performed with real users which conform additional human resources.

8.1.9 Final stage

This is the final task of the project and consists on the preparation of the delivery. Moreover, a final check of the code and the delivery will also be performed.

8.2 Estimated time

In Table 8.1 we can observe an estimation of the number of hours dedicated to each task. The total amount of days available for the development of the project is 95, which adds up to 380 hours. However, in order to be able to cope with unpredictable changes in the planning, the last two weeks will be left as margin to overcome possible deviations from the current plan due to unexpected issues or to explore possible extensions or work on flaws provided that everything works out as expected.

Task	**Estimated duration (h)**
Preamble	40
Robot behavior	60
Communication between modules	20
Detect user state visually	40
Audio module	40
Robot movement	40
Safety	30
Experiments	50
Final stage	20
Total	340

Table 8.1: Estimated time needed for the project
Source: Own compilation

8.3 Gantt chart

On this Figure 8.1 we can observe the Gantt chart of the project. It describes the temporal planning of the tasks and its order of procedure.

8.4 Alternatives and action plan

This section is in charge of describing the methods that are going to be followed in order to assure that the project will be finished within the given time. The main problem that we can encounter with the current planning is a lack of time. We have three methods to solve it:

- The first one is the two margin weeks at the end of the project.
- As we described previously, the selected methodology for this project allows a constant revision and adaptation of the original planning. This can help to solve a lack of time problem as weekly objectives can be redefined according to the state of the project.
- The last method will be only used if the first two methods are not enough to finish the project within the given time. The project has been divided into tasks assuring a modular approach. Therefore, this method consists of discarding one (or more) modules. Modules have been planned in order of importance which assures that if one module is omitted it will be the less relevant. If a module is removed, the resources that it uses will not be necessary.

Finishing the project before the given date gives us the opportunity to improve a module. Alternatively, given the modularity of the project, we can add one (or more) modules to the project. One example of a new module we could add is a module capable of detecting the food state. This could help in stopping the feeding process once the food has been finished or in applying different food grasping approaches.

The most relevant issues that can be encountered have been specified in the section Obstacles. The encountering of the following problems will result in a lack of time issue:

- **Bugs:** for every implementation task this problem can delay the planning from one day to four days.

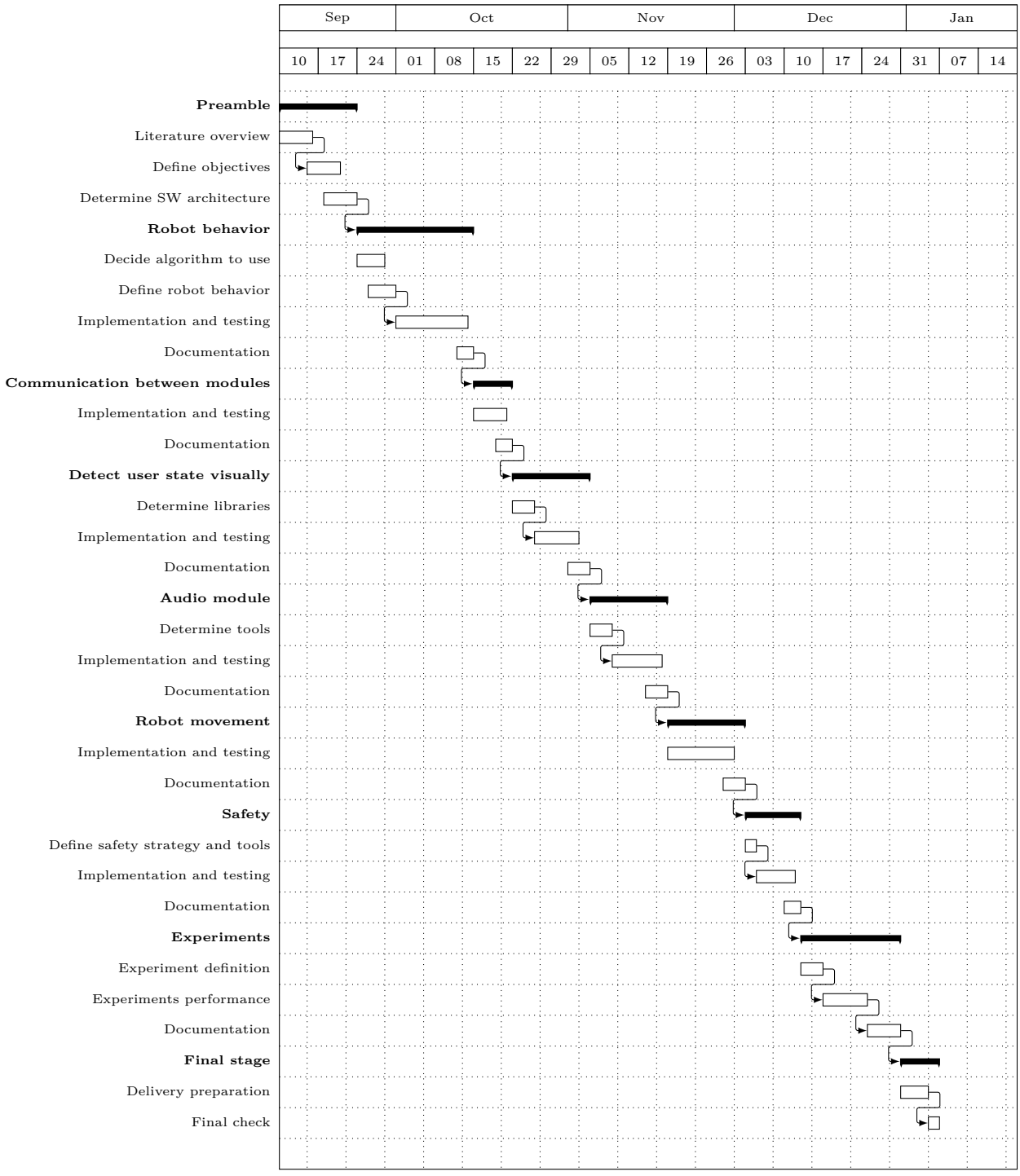


Figure 8.1: Gantt chart of the project
Source: Own compilation

- **Compatibilities problems:** this problem can delay the planning from two days to one week.
- **Lack of computational power:** this problem can delay the planning from two days to three days.

Moreover, in order to adapt to the availability of the robot, we may have to change the planning, especially the Experiments task, delaying up to one week the planning. It is important to comment that, although the learning of software tools is taken into account, it may cause a lengthening of some tasks. Concretely, it can delay the planning up to three days per tool.

8.5 Review of the planning

The planning that was propose at the start of the project has not been entirely followed. The main reason of this disarrangement is amount of time needed to solve some bugs. However, every task has its own reasons for the time disarrangement. The tasks that have taken longer than expected are explained below:

- **Robot behavior:** this is the task that has taken longer than expected to finish. Concretely, it has taken 20hours more to finish. This is mainly due to the learning process of the Rosplan framework. Before starting to use it, I thought it would be easier to understand and use its architecture. However, as it is a newly developed prototype, there is not a lot of documentation or examples to learn from. One factor that delay its implementation was the differences between the PDDL parser of Rosplan and the PDDL parser of the Metric-ff planner.
- **Detect the user state visually:** this task has taken 10h longer that it was expected. This is partially due to the amount of time taken to calibrate the camera extrinsics which was not taken into account when the initial planning was developed. Moreover, there were some problems with the Creative Senz3D camer driver that also delayed task finalization.
- **Robot movement:** this task has taken 10h longer that it was expected. Concretely, what took longer than expected was the time needed to implement the learned trajectories inside the ROS architecture.

Therefore, the project has taken 40h more than what was predicted at the start of the project. Thanks to the two weeks that were left as margin, the project has been finished before the given date.

9. Budget

This section is devoted to explaining the budget of the project. It is important to remark that it is an estimation which is reviewed in Section 9.4.

9.1 Direct costs

Direct costs are the costs that can be directly linked to a specific project, product or facility. In this project, there are three types of direct costs: hardware, software and human resources.

9.1.1 Hardware resources

The costs of the hardware resources used can be observed in Table 9.1. In order to obtain the cost of usage per hour of every resource the following equation has been used:

$$Price\_per\_hour = Unit\_price/(Amortitzation\_period * 250 * 8)$$

In this formula, the 250 is the number of working days in a year and the 8 belongs to the working hours of a day. In order to obtain the real cost of the hardware

Resource	**€/unit**	**Amortization period [years]**	**Cost per hour[€/h]**
WAM robot	97.500	10	4,875
Creative Senz3D	149	4	0,018625
Force sensor	6.000	3	1
Amazon Echo	100	4	0,0125
PC	1.500	4	0,1875

Table 9.1: Cost of each hardware resource
Source: Own compilation

resources of this project we have estimated the number of hours of usage of every resource for every task. We can observe this estimation and the cost of the hardware resources in Table 9.2. Therefore the total cost of the hardware resources is 503,18 €.

Task	WAM	C. Senz3D	Force sensor	Amazon Echo	PC
Preamble	0	0	0	0	30
Robot behavior	0	0	0	0	55
Communication	0	0	0	0	20
Detect user state visually	0	30	0	0	40
Audio module	0	0	0	35	40
Robot movement	30	5	0	5	40
Safety	20	5	20	5	30
Experiments	30	30	30	30	50
Final stage	0	0	0	0	20
Total hours	80	70	50	75	325
Cost [€]	390	1,30	50	0,94	60,94

Table 9.2: Hours per task and hardware resource
Source: Own compilation

9.1.2 Software resources

All software resources used for this project have a free non-commercial license. Therefore, the software cost of this project is 0 €. However, if this project or its outcome were to be used for commercial purposes this price would change. Concretely, all resources are also free for commercial purposes unless the OpenFace library (which will be used to obtain head and mouth detections). A license of OpenFace costs 12.887,69€ a year.

9.1.3 Human resources

This project is mainly built by the developer but there are other roles that take part in it. Concretely, the project managers (in this project the director and co-director) and the testers. In Table 9.3 we can observe the salary per hour of the three roles that take part in the project.

Role	€/hour
Project Manager	40
Developer	20
Tester	15

Table 9.3: Salary per hour of each role
Source: Own compilation

In Table 9.4 we can observe an estimation of the number of hours that each

role is going to spend in each task. We can also observe the total amount of hours that each role spends in the project and their final salary. Therefore the cost of human resources is 9.535 €.

Task	**Developer**	**Project managers**	**Testers**
Preamble	40	10	0
Robot behavior	60	10	0
Communication	20	2	0
Detect user state visually	40	4	0
Audio module	40	4	0
Robot movement	40	4	0
Safety	30	5	0
Experiments	50	10	25
Final stage	20	10	0
Total hours	340	59	25
Cost [€]	6.800	2.360	375

Table 9.4: Hours of each role per task and total cost per role
Source: Own compilation

9.2 Indirect costs

We can observe in Table 9.5 the indirects costs associated to this project. In particular, they consist on the electricity needed, the Internet connection used during the period and other office supplies.

Resource	**Cost**	**Units**	**Cost [€]**
Electricity	0,123 €/KWh	700 KWh	86,1
Internet connection	40 €/month	4	160
Office supplies	100 €	-	100
Total cost	346,1 €		

Table 9.5: Indirect costs
Source: Own compilation

9.3 Total budget and control management

In Table 9.6 we can observe the total budget of the project. This budget is the result of the addition of the costs specified before. It is important to comment that a 5% of contingency has been added over the cumulative total in order to cope with unexpected costs that can occur during the development of the project.

Concept	**Cost [€]**
Hardware resources	503,18
Software resources	0
Human resources	9.535
Indirect costs	346,1
Subtotal	10.384,28
Contingency (5%)	519,21
Total	10.903,49

Table 9.6: Total budget
Source: Own compilation

This budget is only an estimation thus deviations may occur. It is improbable that deviations on software resources occur as nowadays there exist many alternatives with a free non-commercial license that we could use instead. For the case of hardware resources and indirect costs, deviations can occur as for example a better camera or more electricity may be needed. However, most of the budget deviations will happen in the human resources as more hours than the estimated may be needed.

After finishing each task we will determine the current deviation from the estimated hardware and human resources budget using the following formulas:

$$Deviation_{human} = \sum_{i}^{role} (estimated\_hours_i - real\_hours_i) * cost_i$$

$$Deviation_{hardware} = \sum_{i}^{resource} (estimated\_hours_i - real\_hours_i) * cost_i$$

This constant updating and checking are going to help determine when a deviation is produced and try to solve it when it has occurred. Software resources will be also monitored at the end of every task. However, the indirect cost will

only be monitored at the end of the project as they can not be easily divided into parts.

It is important to comment that a certain margin deviation from the budget is not relevant as a contingency percentage has been applied to the budget estimation.

9.4 Budget review

As it is explained in Section 8.5, the project has taken 40hours more than what it was expected to complete. Therefore, there have been 40 more hours of work of the developer with adds a total cost of 800€. However, the performed tests were not which real users which substracts s total of 375€. Therefore, the final human resources are of 9.960€.

As the developer has worked 40 hours more, there are 40 hours more of usage of one computer. This adds a total cost of 7,5€. Moreover, the Amazon Echo has not been used which subtracts a total cost of 0.94€. Finally, apart from the Creative Senz3D camera an XtionCamera has been used. This camera has also been used during 70h. It has a cost of 200€and an amortization period of 3 years. Therefore, the usage of this camera adds a total of 2,33€. After applying these changes, the hardware resources have a total cost of 512,07€.

The cost of the software resources has not changed. Moreover, it can be considered that the indirect costs have not changed. Figure 9.7 details the total cost of the budget after applying the explained changes.

The final budget of the project has increased in comparison with the estimated budget. However, the increase does not exceed the contingency that had been set in the estimated budget. Therefore, these changes are not a problem as the final project cost does not exceeded the estimated total.

Concept	**Cost [€]**
Hardware resources	512,07
Software resources	0
Human resources	9.960
Indirect costs	346,1
Total	10.818,17

Table 9.7: Review of the total budget
Source: Own compilation

10. Sustainability

The project's sustainability has been analyzed regarding three major factors: environmental, social and economical. The analysis is based on the application of the sustainability matrix to the project as shown in Table 10.1.

	PPP	**Useful life**	**Risks**
Environmental	7/10	18/20	-1/-20
Economical	9/10	14/20	-5/-20
Social	9/10	20/20	-20/-20
Sustainability range	25/30	52/60	-26/-60
	51/90		

Table 10.1: Sustainability matrix of the project
Source: Own compilation

10.1 Environmental dimension

Although this prototype needs many hardware resources that have a great environmental impact, they can all be used for various tasks and therefore they can be reused. In contrast, the majority of the state of the art devices for the feeding task have a specific equipment that can only be used for this task. Therefore, this solution will improve the environment with respect to other existing solutions. The electricity needed to supply all the hardware resources also has an environmental impact. This impact is not very relevant, although it is not avoidable.

10.2 Economical dimension

A detailed quantification of all the costs involved in the project has been explained in Chapter 9. The estimated budget allows a completion of the problem if minimal or none deviations occur.

Nowadays, there exist two solutions to the problem of feeding the elderly and disabled. The first one is nurses who are expensive. The second one is some state of the art devices which are not able to feed some disabled or elderly as they are

not completely autonomous. This prototype will be cheaper than a nurse but more expensive than the current state of the art devices as it has more receptors and the robotic arm is more flexible. However, it will be capable of feeding completely autonomously and, thus, it has a larger market.

10.3 Social dimension

Nowadays, most of the elderly or disabled people need assistance in order to eat. This fact detriments their feeling of independence. In order to solve this issue, some feeding devices have been proposed but none of them are autonomous enough to be able to feed a person without requiring any movement of the person. Therefore, there is a need for a completely autonomous device capable of feeding a person. With the prototype presented in this project, this collective will be able to feed by themselves and therefore feel more independent. This will result in an empowerment of this collective. However, there exists a lot of risks as the robot will be in close contact with the user's mouth. Therefore, any not well-treated anomaly can cause a major injury in the user.

One may think that the development of this project will take the jobs of nurses. However, the reality is that there is currently a worldwide shortage of nurses. Moreover, replacing nurses is not the objective of this prototype as it is thought that human contact and human care is also essential. It is intended to be a tool to empower the user and ease the task of the nurse instead of acting as a substitute for the human professional.

From a personal perspective, this project will help me learn robotics with a hands-on approach. In order to succeed in the development of this project, I will have to learn a variety of skills and programs. It will also allow me to discover how is the work of a researcher and help me decide my future career path.

11. Conclusions and future work

This thesis improves the abilities of a robot so it is capable of intelligently feeding a person. The user of the robot is thought to only have mobility from the neck to the top of his head. Therefore, the robot needs to be completely autonomous. The aspects of the feeding task that have been taken into account are the food graping and the feeding of the user.

The presented prototype has been implemented on the WAM robot. Moreover, two RGB-D cameras and a force sensor are used in this prototype. The RGB-D cameras provide information about the state of the user and the positions of the dishes and the mouth of the user. The force sensor has been used to obtain the force produced on the end-effector of the robot.

The behavior of the robot has been implemented with a planner, concretely, with the Metric-ff. This technique offers a less ad-hoc solution than other approaches easing the project's extendability. It allows the robot to autonomously find the best strategy to feed the user given the current situation. Moreover, the behavior also adapts to the user's requirements which are obtained through visual communication. With the defined visual communication the robot knows if the user is paying attention to it or has his mouth open. Moreover, an auditive communication has also been designed and simulated to prove the behavior's adaptability to different user requirements.

Another aspect that has been studied in this project is the movement of the robot. In this prototype three types of movements have been implemented: direct movements, movements using inverse kinematics and learned movements. Each trajectory has been studied to determine which movement type suits it best.

The implemented behavior has been deeply studied in order to test its robustness and utility. It has been concluded that the chosen approach is fast, robust, and eases the programmer's job. Therefore, a planner can be perfectly used to determine the behavior of the robot in this kind of applications. The learned trajectories have also been tested and, after simulation and trial with the robot, it has been concluded its correct performance. Therefore, the technique that has been used is appropriate for these type of trajectories.

On a more general matter, it can be concluded that a prototype like the one presented could greatly improve the quality of life of the older adults and the handicapped population. However, there are a lot of improvements that could be performed in order to obtain a better task performance.

The first improvement that could be included in the current system is the implementation of the audio module. The fact that the user and the robot can communicate through speech is desired in order to achieve a more comfortable task performance for the user.

Another improvement that could be added is the autonomously grasping of the cutlery to use. The big problem that this extension would encounter is the grasping of the cutlery in an horizontal inclination. In order to achieve this, the spoon handle would probably have to be changed to have a more rectangular, pentagonal or hexagonal shape. If the spoon handle shape changed the gripper itself would also have to be changed. Moreover, the gripper would have to be able to autonomously open and close and this movement would have to be able to be controlled through a ROS node.

A further improvement would be to implement a dish detection system. With this, the dished would not need to have an AR tag attached. This would increment the commerciality and robustness of the prototype. It would help the prototype be perceived more as a commercial product than a research project.

Apart from detecting the dish itself, it would also be extremely interesting to also detect the food state. This information could be used to determine when there is no food left. Moreover, in a more advanced point, this information could also be used to execute different grasping movements depending on the current food state. Therefore, the robot would completely adapt to the state of the food inside the dish and make the appropriate grasping movement.

In order to make the system completely autonomous, another improvement could be implemented. This would include the usage of another robot capable of navigating through the room and grasping objects. This robot would be used to get the different dishes and bring them to the table. With this extension, the user could be able to chose the desired dish from a wider variety.

Bibliography

[1] *World Population prospects - Population Division - United Nations*. http://population.un.org. [Online: last accessed September 2018].

[2] *Eurostat: Population structure and ageing*. https://ec.europa.eu/eurostat. [Online: last accessed September 2018].

[3] Eleanor Grace Victoria Stevens et al. *The Burden of Stroke in Europe*. Stroke Alliance for Europe, May 2017.

[4] *World Health Organization: Disability and health*. https://www.who.int/en/news-room/fact-sheets/detail/disability-and-health. [Online: last accessed January 2019].

[5] M. Vila. "Safe feeding strategies for a Physically Assistive Robot". In: *Bachelor thesis of Industrial Technology Engineering of ETSEIB-UPC*. 2018.

[6] *Mobile Arm Support MAS Table Mount*. https://www.rehabmart.com/. [Online: last accessed September 2018].

[7] *JAECO MultiLink Arm with Elevation Assist*. http://jaecoorthopedic.com/. [Online: last accessed September 2018].

[8] *Neater Eater*. http://www.neater.co.uk/. [Online: last accessed September 2018].

[9] I. Sumio. "Meal support robot "My spoon"". In: *The Robotics Society of Japan Journal* 21.4 (2003), pp. 378–381.

[10] *Meal buddy System*. https://www.performancehealth.com/. [Online: last accessed September 2018].

[11] *Mealtime Partner Dining System*. http://www.mealtimepartners.com/. [Online: last accessed September 2018].

[12] J. Jiménez Villarreal and S. Ljungblad. "Experience Centred Design for a Robotic Eating Aid". In: *Proceedings of the 6th International Conference on Human-robot Interaction*. HRI '11. Lausanne, Switzerland: ACM, 2011, pp. 155–156.

[13] A. Jardón, C. A. Monje, and C. Balaguer. "Functional Evaluation of Asibot: A New Approach on Portable Robotic System for Disabled People". In: *Applied Bionics and Biomechanics* 9.1 (2012), pp. 85–97.

[14] B. House, J. Malkin, and J. Bilmes. "The VoiceBot: A Voice Controlled Robot Arm". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 183–192.

[15] K. Tanaka, S. Mu, and S. Nakashima. "Meal-Assistance Robot Using Ultrasonic Motor with Eye Interface". In: *Int. J. Automation Technol.* 8.2 (2014), pp. 186–192.

[16] D. Park et al. "Towards assistive feeding with a general-purpose mobile manipulator". In: *IEEE International Conference on Robotics and Automation - workshop on Human-Robot Interfaces for Enhanced Physical Interactions*. 2016.

[17] G. Canal, G. Alenyà, and C. Torras. "Personalization Framework for Adaptive Robotic Feeding Assistance". In: *International Conference on Social Robotics (ICSR)*. 2016, pp. 22–31.

[18] D. Park et al. "Multimodal execution monitoring for robot-assisted feeding". In: *Intelligent Robots and Systems (IROS)*. 2017.

[19] C. J. Perera, T. D. Lalitharatne, and K. Kiguchi. "EEG-controlled meal assistance robot with camera-based automatic mouth position tracking and mouth open detection". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1760–1765.

[20] *Creative Senz3D image reference*. `https://www.hardwarezone.com.sg/review-creative-senz3d-interactive-gesture-camera-kinect-pc`. [Online: last accessed January 2019].

[21] *Xtion camera*. `https://www.asus.com/es/3D-Sensor/Xtion_PRO/`. [Online: last accessed January 2019].

[22] *ATI F/T sensor Mini40 webpage*. `https://www.ati-ia.com/products/ft/ft_models.aspx?id=Mini40`. [Online: last accessed January 2019].

[23] A. Zadeh, T. Baltrusaitis, and L. P. Morency. "Convolutional experts constrained local model for facial landmark detection". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017.

[24] M. Vila, G. Canal, and G. Alenyà. "Towards safety in Physically Assistive Robots: eating assistance". In: *IEEE International Conference on Intelligent Robots and Systems (IROS) - workshop on Robots for Assisted Living*. 2018.

[25] A. Colomé, A. Planells, and C. Torras. "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 5649–5654.

[26] *IntelRealSense library*. `https://github.com/IntelRealSense/librealsense/blob/master/include/librealsense2/rsutil.h`. [Online: last accessed December 2018].

[27] G. Canal et al. "Joining high-level symbolic planning with low-level motion primitives in adaptive HRI: application to dressing assistance". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3273–3278.

[28] M. Cashmore et al. "Rosplan: Planning in the robot operating system". In: *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*. 2015, pp. 333–341.

[29] A. Colomé and C. Torras. "Dimensionality reduction in learning Gaussian mixture models of movement primitives for contextualized action selection and adaptation." In: *IEEE Robotics and Automation Letters, 3(4): 3922-3929*. 2018.

[30] H. Sung. "Gaussian mixture regression and classification". In: *Phd thesis, Rice University, Houston, Texas*. 2014.

[31] *Amazon Echo*. `https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/`. [Online: last accessed September 2018].

[32] *Robot Operating System*. `http://www.ros.org/`. [Online: last accessed September 2018].

[33] *Git*. `https://git-scm.com/`. [Online: last accessed September 2018].

Appendix A

Stakeholders

The stakeholders of a project are individuals or sets of people that have a specific role in a project. In this project there exist the following stakeholders:

Developer

The developer is the person in charge of the project. He/she is responsible for conducting all needed research in the field of study. Moreover, he/she is responsible for implementing the prototype and making the appropriate decisions. Finally, he/she is responsible for writing the documentation and the report. The fulfillment of these tasks must be done while following the schedule and regarding the director's and co-directors suggestions and advice. The person taking this role in the project is the author of this document.

Director and co-director

The director and co-director are the responsible of guiding and assessing the developer. They follow the progress closely, supply advice and warn of possible errors about either the project proposal or its execution. In this project, the director is Gerard Canal and the co-director is Guillem Alenyà both from the Institut de Robòtica i Informàtica Industrial (IRI).

Beneficiaries

This project has two type of beneficiaries:

- **Direct beneficiaries:** the direct beneficiaries of this project are the potential users of the prototype. As stated in the introduction, the target audience

of the prototype is the older adults and the handicapped population, especially the ones who undergo a loss of upper limb functions. This collective is the one that can take more advantage of the proposed prototype as it allows them to perform a task that they can not do independently. With this gain of independence comes an empowerment of this group of people which can lead to a significant social impact.

- **Indirect beneficiaries:** Nowadays, older adults and the handicapped population who can not feed by themselves are usually helped by nurses. It then may seem that the prototype developed in this project would take their job. However, it is important to remark that replacing nurses is not the objective of this prototype as human contact is thought to be essential. Therefore, the robot would ease the nurse's job doing the most repetitive and tiring part the feeding task. The patient's families are also indirectly affected by the prototype. The user would be more independent and thus would not have to depend on their families.

Scope

As stated in the Chapter 2 the main goal of this project is to improve the abilities of a robot to be able to feed a person autonomously. It is a general goal that needs to be divided into different modules. These modules are the result of the development of a ROS architecture, which has been obtained after the analysis of the requisites and available software.

The first module to develop is the one in charge of controlling the robot behavior. This module will receive the current state and some aural and visual stimulus and will decide the appropriate action to take. This decision will be made using Artificial Intelligence techniques (AI).

The second module to develop is the one in charge of communicating all the modules. It demands information to some modules and passes it to other modules that need this information. Therefore, it acts as a link between modules.

The third module to develop is in charge of detecting the user sate visually. Images of the user will be processed with Face detection libraries. These libraries will give the location of the facial landmarks and the head orientation. Using this information and with simple algorithms, the user's readiness to eat and the openness of the mouth will be decided.

The next module that will be addressed is in charge of detecting the user requests through audio. An Amazon Echo device [31] which uses Amazon's as-

sistant "Alexa" to process the user input, will be used to interact with the user. The user's orders will be processed in order to obtain the user's responses and requests.

The following module to develop will be in charge of moving the robot. The communication with the robot will be made through Robot Operating System (ROS) [32] messages. As this prototype is thought to be used in different robots, the communication with the robot needs to be modular and easily extensible.

The last module will be in charge of deciding if the robot is hurting the user. In order to do so, it will obtain information about the force and torque applied to the end-tip of the robotic arm. With this information, it will decide if the robot has reached a harmful force.

Every time a module is finished it will be tested alone in order to verify its functionality. Once it has been confirmed, the new module will be connected to the already developed modules.

When all the modules are working the resulting prototype will be deeply tested with simulation. Once it has been checked that the robot behaves as expected and there is no harm for the potential user, the prototype will be tested with real users. The feedback of this users will be studied and presented.

Scope review

There are some changes that have been performed on the scope of the project. The first change is in the module in charge of detecting the user state visually. In the final prototype, this module is also in charge of obtaining the position of the dish and the position of the mouth of the user relative to the robot. The other change that has been produced is in the Audio module. Due to technical reasons, this module could not be developed and thus it has only been simulated. Finally, the prototype has not been tested with real users as the testing information they provided was not entirely relevant. The prototype has then been tested with a manikin.

Possible obstacles and solutions

Scheduling

A schedule is needed in order to determine the workload and to distribute it within the available time. If the schedule is not precisely followed the available time to develop this project can be insufficient. The weekly meetings with the director

and co-director will ensure that the project follows the schedule or will serve to apply the appropriate changes in case something unexpected happens.

Bugs

As it is common when developing any kind of software, bugs will be appearing during the process. All bugs will be corrected as they are identified. Moreover, when a module has finished a set of test will be run in order to assure that the module is completely working.

Compatibilities

Many libraries will be used in this project which will create a lot of dependencies. This may cause a problem regarding the compatibility of all the software. If it is possible the software should be made compatible with all the dependencies. However, if this is not the case some dependencies may have to be dropped and other solutions may have to be found.

Computational power

Face detection libraries tend to need substantial computational power. Moreover, face detection needs to be performed at a good frame-rate as it is essential during the performance of the task. Therefore, the available computer may not be able to cope with this process. Shall it be the case, another computer connected by Ethernet or via Wifi will be in charge of processing the images and passing the results to the main computer.

Methodology

The objective of the project is clear but there are many things that need to be decided during its development. Considering this and the limited amount of time, an iterative methodology similar to an agile methodology, will be used to develop the project. This approach with short cycles facilitates to be conscious about the current state of the project and to keep it on schedule.

This methodology will be conducted as follows: every week will start with a defined and realistic objective. At the end of the week, a reunion will be held between the director, co-director and developer. During this meeting, the progress towards the week goal will be reviewed. If the objective is achieved a new objective

will be set for the next week. On the other hand, if it is not achieved the same objective will be set for the following week but it will be more adequate to the existing capabilities.

This project will use Git [33] as a version-control system for tracking changes. This tool enforces a short cycle approach as every change needs to be committed. Moreover, it allows the developer to comment on all changes. The monitoring of the process will be also done through the weekly reunion where the current objective will be reviewed.

This reunion will also serve as a validation method as it will serve to verify the week's objective. Regarding the code itself, the validation will be performed through unit tests. Another validation method will consist on the users using the prototype. This is a crucial validation method as the user's feedback is essential to improve a project. The last validation method will consist on determining the amount of requirements that the prototype has achieved and the success of the prototype itself.

Methodology review

As explained before, an iterative methodology similar to an agile methodology has been used in this project. This methodology perfectly suits this project because it has a strict deadline and there are many things that need to be decided during its development. This approach with short cycles facilitates to be conscious about the current state of the project and to keep it on schedule.

This methodology has been completely followed except one thing: some weeks the co-advisor and/or the advisor were not able to hold the reunion. If one of them had not availability, the reunion was held only between the developer and the advisor or co-advisor that had availability. After the reunion, the advisor or co-advisor that was not able to attend was informed about the current objectives. When none of them were available, the developer reviewed the current objectives and set the objectives for the next week. Once this had been done, the developer informed the advisor and co-advisor about his decisions. With this procedure, the objectives were reviewed and set every week and the developer, advisor, and co-advisor were always informed about the current state. With this almost complete adaptation of the methodology, it has become clear that it has been extremely useful during the development of the project.

Appendix B

This chapter explains the domain and the problem which implement the robot behavior. Moreover, the code of the implemented domain and a problem example are also given.

Domain explanation

The domain includes an explanation of all the state variables that define the system and all the actions that the robot can follow in order to change the state variables.

Before explaining the state variables and the actions of this prototype, the types of the design must be defined. It is important for this application to have types as it eases the implementation, the debugging and the execution of the tasks. The defined types of this application are the following ones:

- **Robot:** defines a robot. Currently there is only one robot in this prototype so this type is not useful. However, it has been included in order to ease the extensibility of the prototype.
- **Waypoint:** defines all the positions the robot can go to or stop in. Some examples are: home, pre-grasping or pre-feeding.
- **Dish:** defines a dish. It is needed as the user is able to choose what dish he wants to eat from.
- **Food:** this defines the quantity of food that a spoon can carry. It is used to define the number of food grasps that can be done on a dish. Therefore, if there was only one food in the desired dish, the robot would only be able to grasp the food one time.

After these definitions, it is now possible to define the predicates. These predicates define the world itself, the robot state, the user state, the user requirements

and some auxiliary variables. The predicates of this application are described below:

- **P1: Next to:** this variable defines if there exist a defined movement between two waypoints.
- **P2: Movement check hf[1]:** this variable defines that there exists a movement between two waypoints. It gives the same information than the previous variable but it is used for the movements where, in order to execute them, the user has to be looking to the robot.
- **P3: Movement check hf mo[2]:** this variable defines that there exists a movement between two waypoints. It gives the same information than the previous variables. However, it is used for the movements where, in order to execute them, the user has to be looking to the robot and to have its mouth open.
- **P4: Can ask look:** this variable defines the waypoints where it is acceptable that the robot asks for the user to look at him.
- **P5: Dish at:** this variable defines the waypoint where a certain dish is.
- **P6: Home at:** this variable defines what is the home or resting position of the robot.
- **P7: Robot at:** this variable defines the waypoint where a certain robot is currently located; or where is located its end-plate.
- **P8: Spoon full:** as his name indicates, this variable defines that the spoon is full.
- **P9: Too close to detect mouth:** this variable indicates that the camera is too close to the user's face to be able to obtain images from an important percentage of the user's face.
- **P10: Inside mouth:** this variable indicates that the spoon is inside the mouth of the user.
- **P11: Head forward:** this variable indicates that the head of the user is towards the robot and thus that the user is paying attention to the robot.

[1]hf stands for head forward
[2]mo stands for mouth open

- **P12: Mouth open:** this variable indicates that the user's mouth is open.
- **P13: Stop feeding:** this variable indicates that the user has requested the feeding process to end.
- **P14: Wait:** this variable indicates that the user has requested the feeding process to stop for a while. When this variable is set, the robot does not move until he is told otherwise.
- **P15: Dish wanted:** this variable defines the dish that the user wants to currently eat.
- **P16: Dish wanted is known:** this is an auxiliary variable which indicates that the user has decided which dish he wants to eat from.
- **P17: Finish feeding:** this is an auxiliary variable that indicates the end of the feeding process. This ending can be due to the fact that there is no food left or that the user has requested it.
- **P18: Food left:** this variable indicates that there a food which is passed as parameter has not been grasped yet.

With these state variables the system is completely defined. As can be observed, there can be several dishes to choose from. Moreover, all the information extracted from visual and auditive communication (explained in Section 6.3) is defined in the system. The state of the robot and the amount of food is also completely defined in this system.

Finally, the actions that the robot can execute also need to be defined. All these actions modify the state variables defined previously. These actions have a prerequisite which is defined by some state variables and and effect which is the change that is produced in the state variables after executing the action. In order to be able to find the best solution there is a cost function which is incremented after all the actions. This cost function has a minor penalization on the actions that the robot should prioritize. The different actions that the robot can execute are listed below:

- **A1: Go to waypoint:** when this action is executed the robot moves from a defined waypoint to another one.
- **A2: Get food:** when this action is executed the robot grasps the food from the selected dish.

- **A3: Throw food:** when the robot executes this actions it throws the food that he has on the spoon onto the dish.
- **A4: Go to waypoint precondition hf:** this actions moves the robot from a waypoint to another one. However, in order to execute this action, the user needs to be paying attention to the robot movement.
- **A5: Feed:** this actions is in charge of feeding the user. In order to execute this action, the user needs to be paying attention to the robot movement and to have his mouth open.
- **A6: Inform start and ask dish wanted:** this action is in charge of informing the user of the start of the feeding task and to ask him what dish he desires.
- **A7: Ask open mouth:** this action is in charge of asking the user to open his mouth.
- **A8: Ask look:** this action is in charge of asking the user to pay attention at the robot.
- **A9: Wait finished:** when this action is executed the user tells the robot that the waiting period has been finished.
- **A10: Auxiliar finish stop:** as its name tells us, this action is an auxiliary action is charge of correctly ending the feeding process when the user has asked for it.
- **A11: Auxiliar finish no food:** as its name tells us, this action is an auxiliary action is charge of correctly ending the feeding process when there is no food left on the dish.

It may seem incorrect that there is no action capable of setting the wait, stop, head forward, etc. variables. However, due to the functioning of the implementation (which is explained in Subsection 6.4.2), the setting of these variables is manually done. What is indispensable is to have actions that allow to handle these variables when they are set. Therefore, actions that the robot can perform in order to "solve" the current situation. For example, if the robot was about to feed the user but his mouth was not open, the robot would have to perform an action to get the user to open his mouth. This particular action is called ask open mouth. Without it, the robot would not have a way to continue feeding the user and thus, the problem would be unsolvable.

Problem explanation

With the definition of the system that has been previously explained, the definition of the problem is certainly simple. Every possible problem needs to the find the world state. Therefore, the adjacencies between nodes, the location of the dishes, the location of the home position, etc. Moreover, it needs to define the state of the robot and the user. From these two groups of variables, the only variable that has to be always included is the current position of th robot. The other ones depend on the actual state of the system. The last set of variables that need to be included are the amount of food that has each dish.

Another factor that the problem has to include is the function that has to be optimized. In this case, it is the cost function defined previously and it needs to be minimized as all the actions increment its value.

Finally, the problem has to include the desired ending state. In this application it would consist on the robot being at the home position without food on the spoon when there is no food left or the user has asked to stop the feeding task. For simplification, the variable finish feeding, described previously, indicates if this situation is currently true.

Implemented domain

```
(define
        (domain feeding_behavior)
        (:requirements :strips :typing :fluents
           :negative-preconditions :adl)
        (:types
                robot
                waypoint
                dish
                food
        )

        (:predicates
                ;World definition
                (next_to ?w1 - waypoint ?w2 - waypoint)
                (movement_check_hf ?w1 - waypoint ?w2 - waypoint)
                (movement_check_hf_mo ?w1 - waypoint)
                (can_ask_look ?w1 - waypoint)
                (dish_at ?d - dish ?w - waypoint)
                (home_at ?w - waypoint)
                (food_left ?f - food)
```

```
20
21                ;Robot state
22                (robot_at ?r - robot ?wp - waypoint)
23                (spoon_full)
24                (too_close_to_detect_mouth)
25                (inside_mouth)
26
27                ;User state
28                (head_forward)
29                (mouth_open)
30
31                ;User requirements
32                (stop_feeding)
33                (wait)
34                (dish_wanted ?d -  dish)
35
36                ;Auxiliar predicates
37                (dish_wanted_is_known)
38                (finish_feeding)
39        )
40
41        (:functions (penalization))
42
43
44        (:action GO_TO_WAYPOINT
45                :parameters(
46                        ?r - robot
47                        ?w_act -waypoint
48                        ?w_fin - waypoint
49                )
50                :precondition(and
51                        (robot_at ?r ?w_act)
52                        (next_to ?w_act ?w_fin)
53                        (not (wait))
54                )
55                :effect(and
56                        (not(robot_at ?r ?w_act))
57                        (robot_at ?r ?w_fin)
58                        (increase (penalization) 10)
59                )
60        )
61
62        (:action GET_FOOD
63                :parameters(
64                        ?r -robot
65                        ?w - waypoint
```

```
66                      ?d - dish
67                      ?f - food
68              )
69              :precondition(and
70                      (dish_wanted ?d)
71                      (dish_at ?d ?w)
72                      (robot_at ?r ?w)
73                      (not(spoon_full))
74                      (food_left ?f)
75                      (not (wait))
76                      (not(stop_feeding))
77              )
78              :effect(and
79                      (spoon_full)
80                      (increase (penalization) 1)
81                      (not(food_left ?f))
82              )
83      )
84
85      (:action THROW_FOOD
86              :parameters(
87                      ?r - robot
88                      ?w - waypoint
89                      ?d - dish
90              )
91              :precondition(and
92                      (dish_wanted ?d)
93                      (dish_at ?d ?w)
94                      (robot_at ?r ?w)
95                      (spoon_full)
96                      (not(wait))
97                      (stop_feeding)
98              )
99              :effect(and
100                     (not(spoon_full))
101                     (increase (penalization) 1)
102             )
103     )
104
105     (:action GO_TO_WAYPOINT_PRECONDITION_HF
106             :parameters(
107                     ?r - robot
108                     ?w_act -waypoint
109                     ?w_fin - waypoint
110             )
111             :precondition(and
```

```
112                     (spoon_full)
113                     (head_forward)
114                     (robot_at ?r ?w_act)
115                     (movement_check_hf ?w_act ?w_fin)
116                     (not (wait))
117                     (not(stop_feeding))
118             )
119             :effect(and
120                     (not(robot_at ?r ?w_act))
121                     (robot_at ?r ?w_fin)
122                     (increase (penalization) 5)
123             )
124     )
125
126     (:action GO_TO_WAYPOINT_PRECONDITION_HF_MO
127             :parameters(
128                     ?r - robot
129                     ?w_act - waypoint
130             )
131             :precondition(and
132                     (spoon_full)
133                     (head_forward)
134                     (mouth_open)
135                     (robot_at ?r ?w_act)
136                     (movement_check_hf_mo ?w_act)
137                     (not (wait))
138                     (not(stop_feeding))
139             )
140             :effect(and
141                     (not(spoon_full))
142                     (increase (penalization) 5)
143             )
144     )
145
146     (:action INFORM_START_AND_ASK_DISH_WANTED
147             :parameters(
148                     ?d - dish
149             )
150             :precondition(and
151                     (not (dish_wanted_is_known))
152                     (not (wait))
153                     (not(stop_feeding))
154             )
155             :effect (and
156                     (increase (penalization) 10)
157                     (dish_wanted_is_known)
```

```
158                         (dish_wanted ?d)
159                 )
160         )
161
162         (:action ASK_OPEN_MOUTH
163                 :parameters(
164                         ?r - robot
165                         ?w_act - waypoint
166                 )
167                 :precondition(and
168                         (spoon_full)
169                         (not(mouth_open))
170                         (robot_at ?r ?w_act)
171                         (movement_check_hf_mo ?w_act)
172                         (not (wait))
173                         (not(stop_feeding))
174                 )
175                 :effect(and
176                         (mouth_open)
177                         (increase (penalization) 1)
178                 )
179         )
180
181         (:action ASK_LOOK
182                 :parameters(
183                         ?r - robot
184                         ?w_act - waypoint
185                         ?w_fin - waypoint
186                 )
187                 :precondition(and
188                         (spoon_full)
189                         (not(head_forward))
190                         (robot_at ?r ?w_act)
191                         (can_ask_look ?w_act)
192                         (not (wait))
193                         (not(stop_feeding))
194                 )
195                 :effect(and
196                         (head_forward)
197                         (increase (penalization) 1)
198                 )
199         )
200
201         (:action WAIT_FINISHED
202                 :parameters()
203                 :precondition(
```

```
204                     (wait)
205             )
206             :effect(and
207                     (not(wait))
208                     (increase (penalization) 1)
209             )
210     )
211
212     (:action AUXILIAR_FINISH_STOP
213             :parameters(
214                     ?r - robot
215                     ?home - waypoint
216             )
217             :precondition(and
218                     (home_at ?home)
219                     (robot_at ?r ?home)
220                     (not(spoon_full))
221                     (stop_feeding)
222             )
223             :effect(and
224                     (finish_feeding)
225                     (increase (penalization) 1)
226             )
227     )
228
229     (:action AUXILIAR_FINISH_NO_FOOD
230             :parameters(
231                     ?r - robot
232                     ?home - waypoint
233                     ?f - food
234             )
235             :precondition(and
236                     (home_at ?home)
237                     (robot_at ?r ?home)
238                     (not(spoon_full))
239                     (not(food_left ?f))
240             )
241             :effect(and
242                     (finish_feeding)
243                     (increase (penalization) 1)
244             )
245     )
246 )
```

Problem example

```
(define (problem feeding_task)
        (:domain feeding_behavior)
        (:objects
                wam - robot
                home wd1 wd2 pos2 mouth - waypoint
                d1 d2 - dish
                f0 - food
        )
        (:init
                (next_to home wd1)
                (next_to wd1 home)
                (next_to home wd2)
                (next_to wd2 home)
                (movement_check_hf wd1 pos2)
                (next_to pos2 wd1)
                (movement_check_hf wd2 pos2)
                (next_to pos2 wd2)
                (movement_check_hf_mo pos2)
                (can_ask_look pos2)
                (can_ask_look wd1)
                (can_ask_look wd2)
                (dish_at d1 wd1)
                (dish_at d2 wd2)
                (home_at home)
                (= (penalization) 0)
                (robot_at wam home)
                (food_left f0)
        )
        (:goal (finish_feeding))
        (:metric minimize (penalization))
)
```