Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Multi-Modal Interface for offline Robot Programming

Autor: Álvaro García Morcillo

Tutores: Ignacio Bravo Muñoz y José Luis Martínez Lastra

2018

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Multi-Modal Interface for offline Robot Programming

Autor: Álvaro García Morcillo
Trabajo realizado en: Tampere University of Technology
País: Finlandia
Tutor: Ignacio Bravo Muñoz
Cotutor: José Luis Martínez Lastra

TRIBUNAL:

Presidente: Felipe Espinosa Zapata

Vocal 1º: Iván García Daza

Vocal 2º: Ignacio Bravo Muñóz

FECHA: 12 de Septiembre de 2018

To all the friends I have made here in Tampere, even thou I have spent more time with this work than you guys. . .

A todos mis amigos de España, esos amores con esas almas tan grandes que a pesar de decirles que tendrían que haber *estudiao* les admiro un montón. . .

A mis padres y mi familia, que después de esto posiblemente tengamos que separarnos durante largos periodos. . .

Acknowledgements

This thesis has been made during an Erasmus grant at Tampere University of Technology, and will also serve as Final Master Work (TFM) for the Master Degree in Industrial Engineering at the University of Alcalá de Henares.

I hope this work will serve as a base for future students here at FAST-Lab so the many hours spent on it become fruitful.

I want to thank Professor Lastra for welcoming me to Fast-Lab, and the rest of the lab members for showing interest in this thesis and telling me how cool it looks.

Resumen

Este trabajo presenta una propuesta para mejorar al programación de robots fuera de línea usando métodos de entrada basados en habilidades humanas naturales. La propuesta se enfoca en enseñar operaciones básicas de ensamblaje y manipulación, utilizando un par de robots industriales en un entorno de simulación ya existente y se dispone para ser mejorado en trabajos futuros, los cuales también se proponen en este trabajo.

Con el fin de desarrollar esta propuesta y teniendo en cuenta los recursos disponibles, se ha desarrollado un *Add-In* para el programa de simulación y programación fuera de línea *Robot Studio*. Este *Add-In* combina pose humana, una interfaz gráfica de usuario y opcionalmente habla para enseñar al robot una secuencia de objetivos junto con el entorno de simulación para automáticamente generar instrucciones.

Dos diferentes tipos de sensores, *Kinect* y *Leap Motion Sensor* han sido evaluados en base a referencias para seleccionar el más adecuado para la implementación de este trabajo.

Las ejecuciones de las instrucciones programadas han sido evaluadas en simulación.

Palabras clave: Interfaz multimodal, Programación fuera de línea, Sensor *Leap Motion*, *Robot Studio*, Reconocimiento de voz.

Abstract

This thesis presents an approach for improving robot offline programming using input methods based on the human natural skills. The approach is focused to teach basic assembly and manipulation operations using a pair of industrial robots in an existing simulation environment and is meant to be improved in future works, that are also proposed in this thesis.

In order to develop this approach, and regarding the available resources, an Add-In for the simulation and offline programming software RobotStudio was developed. This Add-In combines human pose, a graphical user interface and optionally speech to teach the robot a sequence of targets, along with the simulation environment, to automatically generate instructions.

Two different kinds of sensors, Kinect and Leap Motion Sensor have been evaluated based on references in order to select the most suitable one for the implementation of this work.

The executions of the programmed instructions have been evaluated in simulation.

Keywords: Multi-Modal Interface, Offline programming, Leap Motion Sensor, Robot Studio, Speech Recognition.

Contents

List of Figures

List of Acronyms

API	Application Programming Interface.
AR	Augmented Reality.
DOF	Degrees of freedom.
GUI	Graphical User Interface.
HRI	Human-Robot Interaction.
I/O	Input/Output.
MMI	Multi-Modal Interaction.
RS	Robot Studio.
SDK	Software Development Kit.
VR	Virtual Reality.

Chapter 1

Introduction

1.1 Motivation

In this section we will give some reasons that motivate the work developed, answering the question:

"Why Multi-Modal Interface for Offline Robot Programming ?"

For years, we have seen many sci-fi movies where robots interact with people and following their commands in a very natural way, in some others we have seen how not autonomous robots are controlled imitating the user moves and, nowadays, industrial robots still lack artificial intelligence.

To program industrial robots, the programmer can use a virtual environment and teach them the tasks they need to do by showing them targets, making them follow a path with that targets and some parameters; and transforming that paths into code along with other instructions that read data from inputs and sets data to outputs; there is also the opportunity to teach the real robot in other ways.

Some implementations of controlling robots by imitation have been implemented these last years [1–3].

If a robot can imitate a person, a person would be able to easily teach a robot, however, robot movements have something we do not have, high precision.

Movements of humans are always done in a closed loop, we move our articulations, we see them through our eyes and we correct the position, robots otherwise can do many tasks with high precision not sensing anything, but needing a correct calibration; a lack of sensors will, of course, make a robot less flexible, an industrial robot with no sensors can only work in assembly lines where the pieces are positioned in a known, precise way.

Assuming that our robot does not have any sensor and it must be working most of the time, how can we program it to do certain tasks?

Using an offline environment, we can simulate pieces and robots, then we can sense the pieces and tell a virtual robot to handle it, anyway, to tell a robot to handle a piece requires a sensor to sense us, but in this case it will be connected to a computer, not to a robot.

Robot manipulators are also known as robot arms, and we humans have arms, two of them, then it is likely to teach two arms how to do an operation. Many research with artificial intelligence has been made with non-industrial robots to reach what it is written in the first paragraph of this motivation, but industrial robot manipulators are not receiving that treatment as much as mobile robots or assisting robots, in other words, industrial robot manipulators are not being treated in the human-robot interaction field as much as other types of robots.

Then, this thesis will focus on programming robot tasks using a multi-modal interface, making them imitate our movements and being able to command them with our voice somehow. In other words, we will apply human-virtual robot interaction in order to program them.

1.2 Justification

The motivation given for this thesis can be justified regarding both non-recent and more recent research, this way, the motivation can be proved to have potential interest in some research directions that are still of interest nowadays:

1. Making a task easier has always been beneficial in many disciplines. Automation requires a programming process that can be made easier and more intuitive. Even since 2007, the robot control development importance has been highlighted, specially in an industrial perspective, Brogårdh outlined the sensor-based human-robot interfaces for intuitive robot programming as a research direction [4].
2. The industry has evolved since its origin through three industrial revolutions, in 2013, the fourth one was announced. Nowadays, the so known Industry 4.0 has raised the complexity of the procedures inside the automated production systems. As highlighted by Wittenberg, this rising of complexity also raises the need for more intuitive interfaces. Moreover, the virtual commissioning along with the on-field commissioning of systems, takes the work from the field to the office, helping to save time [5].
3. Robots installed in factories usually require specialized human resources to take care of the maintenance, calibration, and programming. In small and medium enterprises, there are high parametrization and reconfiguration needs, requiring specialized staff to deal with robot systems [6]. To introduce robotics in small companies, achieving easier ways to deal with robots may ease their use and reduce the human resources costs and the availability of operators.
4. Interaction with robots in alternative or more natural ways has been repeatedly researched focusing more on assistive or mobility robots [7–11], however, there is also related research focusing robot manipulators [1–3, 12–16]. Some of these works focus on robot control [1, 2, 12] rather than in robot programming [3, 13–16]. None of these approaches integrate its work in a well-known off-line robot simulation and programming software, such as Robot Studio (RS). Within these references, only the most recent focus on collaborative manipulators [3, 15].
5. Integrating an interface which introduces new input modes to an already existing programming environment can make easier to combine the new modes with the already existing ones.
6. According to the 2016 article [13], literature about Multi-Modal Interaction is still scarce, then the thesis topic is suitable.

1.3 Problem Statement and Research Questions

Robotics in the industry is still challenging for the SME's decision on rather invest in it or not. The return on investment gets worse when a business has to face the costs of specialized staff to reprogram and reconfigure tasks in a robotic system. Therefore, new programming interfaces for making robot programming easier and more intuitive are required for saving time and avoiding the need for specialized staff, nowadays it is still being researched.

Given the previously exposed problem, an approach for programming robots in a multi-modal way can be interesting, as well as to ease future works on this complex task where the literature is still scarce. Regarding the interaction modes proposed, hand pose imitation and speech along with traditional Graphical User Interface (GUI) input/output methods, the following research questions are prompted:

- How can an input method such as the hands pose can be decided to be interpreted or not, avoiding additional inputs?
- How can the hands' poses and speech can be combined to achieve a better functionality?
- How can we translate our actions given these input modes into code or instructions?
- What sensors or devices can make the interaction with the robot more precise, flexible or intuitive?

1.4 Scope

The implementation of the proposed approach has been developed based on a real robotic cell installed at FAST-Lab in the Tampere University of Technology. This cell includes two ABB IRB140 robot manipulators. The material available has both inspired and limited the implementation developed.

The implementation can be used for ABB robots programming, using the simulation and offline programming environment RS, up to two robots can be programmed at the same time.

Human-motion sensing will be used to extract 3D information and discrete poses.

Regarding the speech recognition, the interface may need the en-US speech recognition package for Windows. Speech recognized commands allow the user to activate some buttons of the GUI for the robots in order to gain precision or to open and close the grippers.

The virtual station used has already modeled a robotic ABB cell, with a pair of IRB120 robot manipulators and it has been configured to hold two robot tasks in the same virtual controller, there is also the same RS *Smart Component* for both grippers (Two *Smart Components* of the same type), anyway, users may create their own *Smart Components* and use their own grippers in their simulations.

The pieces that have been assembled using the developed interface have their models imported to the used RS station, other models can be imported to new stations.

The interface is meant to program some assembly and manipulation basic operations.

The method used for this thesis implementation does not use artificial intelligence.

1.5 Limitations

Limitations of this thesis work are exposed here, not as a result of the development but as intended limitations.

- The proposal is only applicable to off-line robot programming environments which allow Add-Ins.
- The Add-In is only compatible with RS, and it is only compatible with Windows.
- The implementation is only compatible with the Leap motion sensor as human-motion input device.
- The implementation does not consider sensors installed in robots or collision detection.

- Targets will be manually generated in RS by the user by using the human-motion data.
- Automatic code generation is limited to automatic instructions generation in RS, using the targets defined by human-motion data.
- Virtual robots tools will only imitate the hand positions when these are closed.
- The points to be reached by the robots have to be taught one by one, there is no artificial intelligence generating targets automatically, the targets will be recorded when opening the hands instead.

1.6 Objectives

Now that the reasons for this thesis have been given, we define some objectives we want to achieve. It is important to highlight that the implementation of this work is also meant to be a base for possible future works on it.

- Sense some motion of the human body to be used as an input for the robot programming along with voice commands.
- Being able to represent that input in a virtual environment, at the same time that the user works with it.
- Get a flexible interface that allows the user to make some configuration and allows to teach two robots.
- Possibility to restrict some movements to do some precise movements a human could not easily achieve.
- Keep the code of the project as structured and readable as possible and document it to ease future works.
- Being able to program movement, Input/Output (I/O) and synchronization instructions.

1.7 Outline

Now a summary of the contents of this thesis is given.

Chapter 2, analyses related work in different topics related to this work as well as their relation to it. **Chapter 3** presents the theoric proposal of this thesis as well as the tests to be done for it. **Chapter 4** Makes a description of the implementation of the proposal used to make the tests. **chapter 5** makes a review of the behavior of the program as well as the executions of the programmed tasks.

Finally, **chapter 6** presents the conclusions of this work, summarizing the results and issues, as well as exposing future works.

Chapter 2

Literature Review

In this chapter, some related works will be summarized, classifying them into sub-topics and finally, some conclusions of the reviews will be presented. The sub-topics are prompted as follows:

1. Human-Robot Interaction.
   (a) Human-Robot Interaction for controlling.
   (b) Human-Robot Interaction for programming.
2. Multi-Modal Interfaces.

2.1 Human-Robot Interaction

The Human-Robot interaction is commonly referred as Human-Robot Interaction (HRI) by researchers. It is a multidisciplinary field with different contributions such as artificial intelligence, robotics, and social sciences.

2.1.1 Human-Robot Interaction for controlling

The HRI is typically known as interacting with a robot to make it do some task. Several methods for achieving a successful interaction have been researched for better and more natural ways to interact with them. Starting from 2005, we introduce the gesture-based interface in [17] for a robot competition, with this interface, the user can control a fighter robot by reading the user movements with a camera that simply process the skin color of the user and extract the fists and head features. The interface recognizes the gesture corresponding to the user's pose, generating a robot command to move it in a similar way. In this case, we have seen an interface which transforms pose into gesture and gesture into a command.

It is also important how the robot gives the user information, in 2007, [18] presents a Networked, Multi-sensored environment for controlling a mobile robot. A PDA is used as the interface device with the robot, being able to see the robot location through it and giving commands to it. In this case, the interface used does not turn natural movement to commands but enables the user to better understand the robot point of view.

Later, in 2009, a brain-robot interface is presented in [19] for controlling a robot arm. This kind of interfaces still have many limitations, and the related research is destined to people with severe disabilities. The interface allowed the user to control each degree of freedom of a robot at a time, still with an error

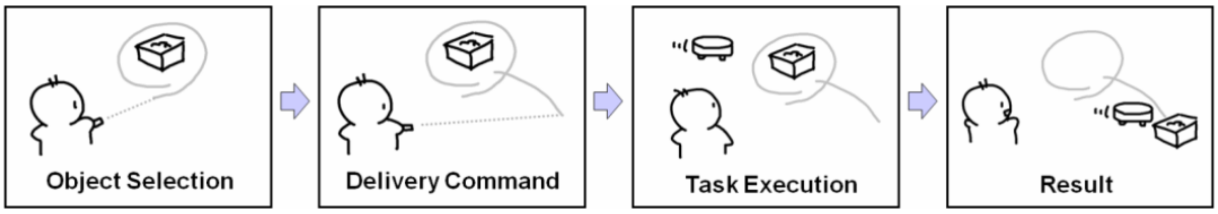


Figure 2.1: K. Ishii et al.' Robot Control using Laser Gestures

percentage to be reduced. These brain-robot interfaces require a specialized knowledge out of the scope of this thesis.

In the same year, another device for interfacing a robot such as a laser pointer has been used in [7]. The laser point is sensed by a camera and two additional cameras track the robot and objects in the environment using visual-based tags. The system recognizes laser stroke gestures such as lasso and stroke gestures for object selection and commands, an example is given in Figure 2.1 [7, Fig. 1], additionally, the laser pointer incorporates additional buttons used for canceling movement. For visual feedback, a projector located in the ceiling is used, it displays the trace of the laser point on the floor. This system allows to control a robot in a comfortable way but the system environment is complex and it is limited to a place where the environment is built.

Five years later, in 2014, [1] addresses the problem of mapping a human arm motion to a robot arm. A motion capture suit senses the movement of the entire arm and the end-effector trajectories are reconstructed from the human hand. The article denotes the importance of replicating the human body motion to transfer knowledge and experience to a robot. The system operates at real time. In this case, the control is made by imitation, being more natural than in previous approaches.

The next year, a manipulation system based on tablet PC has been designed in [12]. Generating a virtual ray from the tablet given its inclination and calculating the collision with a virtual environment, the interface is designed using Unity and generates visual feedback. In this approach a virtual environment can be found along with a real system.

In a similar way than [1], we can find another case of imitation in [2], this time imitating the hand and ignoring the rest of the arm, offering again a natural input of data, only controlling the position of the end-effector. It uses an Inertial Measurement Unit along with a Kinect camera and two Kalman filters for both position and orientation. The 3D camera locates the hand and the IMU measures its orientation. An over-damping method is finally used to soften the hand movement and avoid undesirable movements, disabling the movement if a violent move is done by the user. This approach will be similar to ours regarding the hand pose imitation, but this one is oriented to directly control a manipulator.

The last imitation case we will talk about [20] is done with a cheap human-motion sensor, this is the Leap Motion Sensor whose software allows the programmer to easily obtain parameters of the hand such as position, and orientation, of hands and its fingers. In this approach, they transform position and orientation of the hand into position and orientation of the end-effector of the robot.

2.1.2 Human-Robot Interaction for programming

Although human-robot interaction is more focused on robot controlling, many articles for programming them have been reviewed since it will be the main goal of our approach.

In 2014, the article [16] offers an additional review about human-robot interaction and proposes an Augmented Reality (AR)-based interface for HRI. It highlights that the industrial robots have a low level of autonomy, being designed for repetitious tasks in structured environments, later denoting how robots are increasingly found in SME's environments where a direct interaction between operators and robots is commonly needed. It also reviews new methods that have been reported by new research efforts in HRI such as Multi-Modal Interaction (MMI), programming by demonstration, AR and Virtual Reality (VR). The article proposal of AR-based interface combines a physical environment with virtual entities, along with the robot model, a cube marked with vision tags is used as interaction device. The interface is developed for path-planning. It is important to add that new AR devices models have been released more recently.

In the next year, [14] denotes that there is little experience on Teaching by Demonstration in industrial use cases despite of many research done in this field. This article proposes to manually specify what action to apply and teach by using gestures the relevant action parameter, removing the uncertainty of what action to perform. The concept of transformable robots is defined as those robots capable of solving a variety of tasks by using a certain level of cognitive capabilities. Many related works are mentioned, we denote that it makes references to some works focused on offline programming by using CAD models. The robot used in this approach is focused on industrial applications. Some predefined objects with QR codes are located on the experimental scenario and a GUI is used to specify the steps to follow when teaching an action to the robot, a set of five gestures is used, not only tasks can be taught, they can also be executed, even while teaching them. This approach contemplates the principal goal of making programming more intuitive and flexible, to be able to also control a robot to perform a wide set of different tasks while working in a natural way.

Recently, on 2017, an interface for off-line teaching collaborative robots is introduced in [3]. It mentions that the ways humans can interact with robots have still much room for improvement. This approach uses the ABB's collaborative robot IRB 14000 (YuMi), and it focuses in assembly tasks. The first teaching step is to manually specify the type of task to be performed, e.g. folding or insertion, models of the pieces have to be uploaded. The uploaded pieces are identified in the work environment and a RGBD camera records the demonstration by the user. The next step extracts key frames of the operation, with the possibility to manually modify or add information to them. After this phase, gripper fingertips on a 3D model are automatically generated depending on the pieces dimensions. Finally, in the training phase, the taught assembly can be simulated or executed. This approach achieves a natural demonstration with the robot by replicating the steps done by the user while assembling a set of pieces.

In the same year we find a proposal that also uses a dual-arm robot and extracts keyframes, additionally, it uses a set of geometric constraints [15], in this case, it is applied to manipulation tasks instead of assembly ones. A constraint used in this proposal is to compare the three rotational degrees of freedom of the end-effector to those of an object, an orientation constraint is created if a close difference is found, this is similar to the snapping feature found in CAD design programs. Then if this constraint is present in two consecutive keyframes, the move-in-line constraint is also applied. It uses a knowledge base for motions as seen in Figure 2.2 [15, Fig. 3]. This proposal uses a GUI for teaching, not other methods based on human motion or speech.

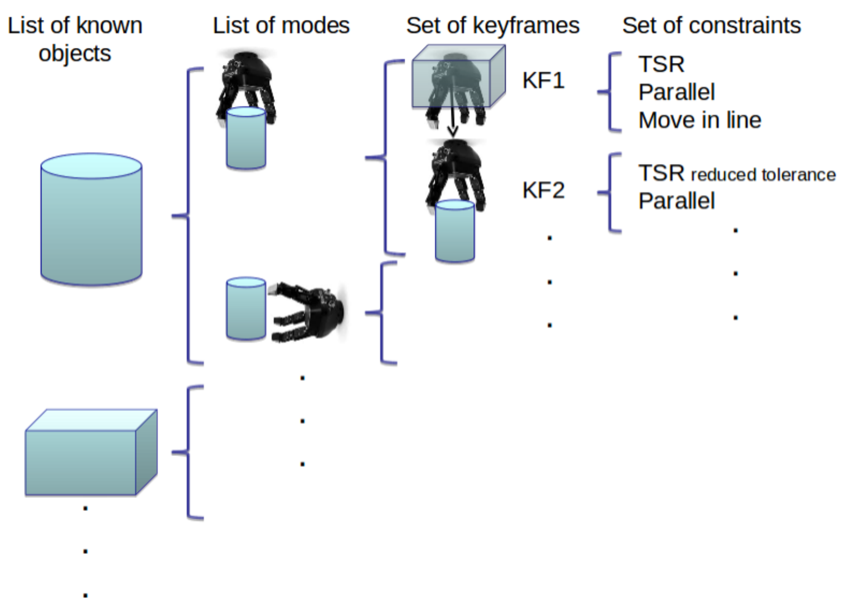


Figure 2.2: Pérez and Julie content of the objects knowledge base.

2.2 Multi-Modal Interfaces

When different kinds of communication channels are found in an interaction we can talk about Multi-modal Interaction.

The goal of effective interaction between user and robot assistant makes it essential to provide a number of broadly utilizable and potentially redundant communication channels.[21] Regarding the last citation, MMI will be necessary to achieve an effective HRI.

Most of the works found are focused on assistive robots rather than manipulators or robots meant for industrial applications, in 2012 [8] combines speech and gestures recognized through a Kinect camera to interact with a robo-receptionist. It also highlights that *priming (or long-term interaction) is an important consideration for robotic systems.*

More recently in 2016 [13] presents a framework for an industrial robot, introducing nowadays robots as difficult to program for end users as well as typically unaffordable for SME's. It also speaks about current off-line programming software problems for robots whose tasks need frequent changes, what is commonly found in SME's. They exemplify the effectiveness of MMI by combining hand gestures from a sensed glove to operate a gripper, text programming from a computer, and a GUI, illustrated in Figure 2.3 [13, Fig. 2], the work is under development.

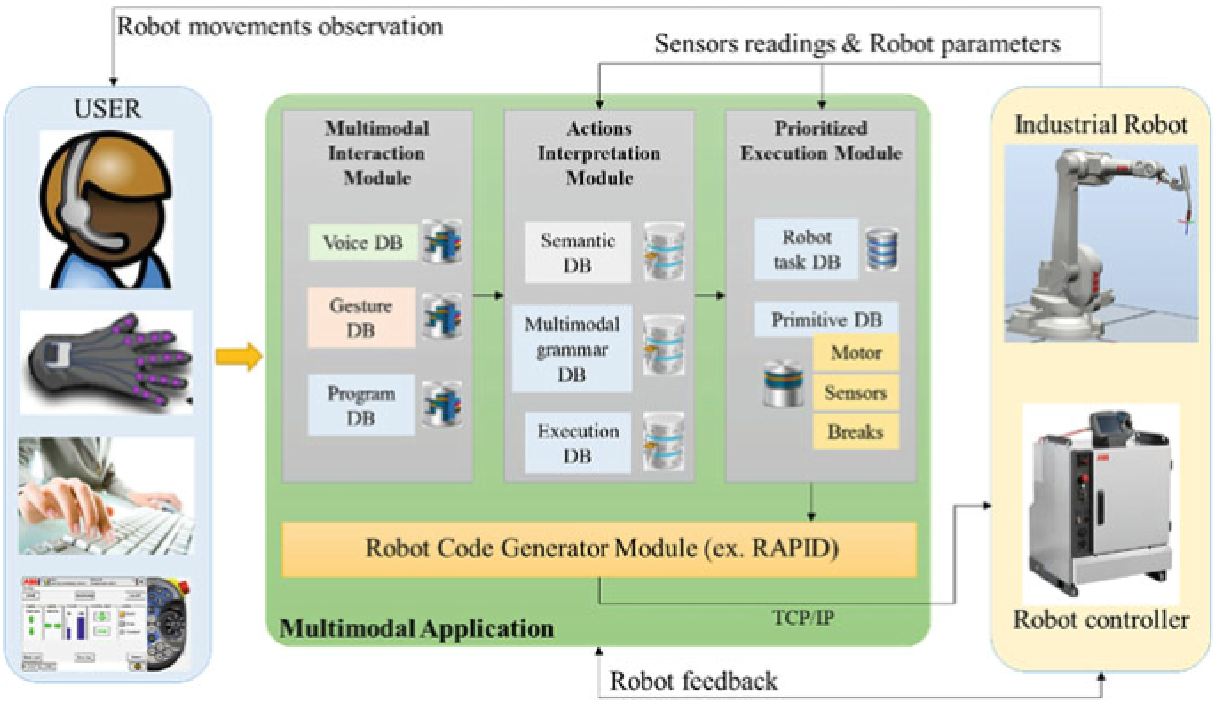


Figure 2.3: Mocan *et al.*' Multimodal interface framework.

2.3 Summary of the review

The most recent approaches found in this review achieve alternative ways to program robots than those used in many industrial cases. SME's interest for implementing robotics but the flexibility and reprogramming issues are a main concern on most of cases.

Many of the human-robot interaction techniques extract gestures from the human pose, some others extract the tridimensional information such as positions or rotations, it is typical to use the Kinect camera as the main sensor for this task, the use of alternate or additional devices such as pointers can make the development of the interaction easier, making detection easier and more robust, not necessarily reducing intuitiveness since humans are used to this kind of tools.

The last articles reviewed for HRI for programming denoted the importance of using CAD models in off-line interaction with virtual robots, these models are supposed to be the same as the pieces to be manufactured, and therefore those models should be available for the enterprise manufacturing them.

It is also mentioned how much effort is academic research given to HRI while it is rarely found in industrial practices, therefore an interest for these methods should be awaken in robot manufacturers for adapting their off-line programming environments facilitating HRI and flexible programming using their programs.

Regarding MMI, many I/O methods have been used for HRI, combining them can improve robustness or reliability, as well as compensate the weak points of each other, using many different devices may also raise the need for an standard for MMI, many of the possible I/O methods that could be combined for a Multi-modal interaction are gathered in Figure 2.4.

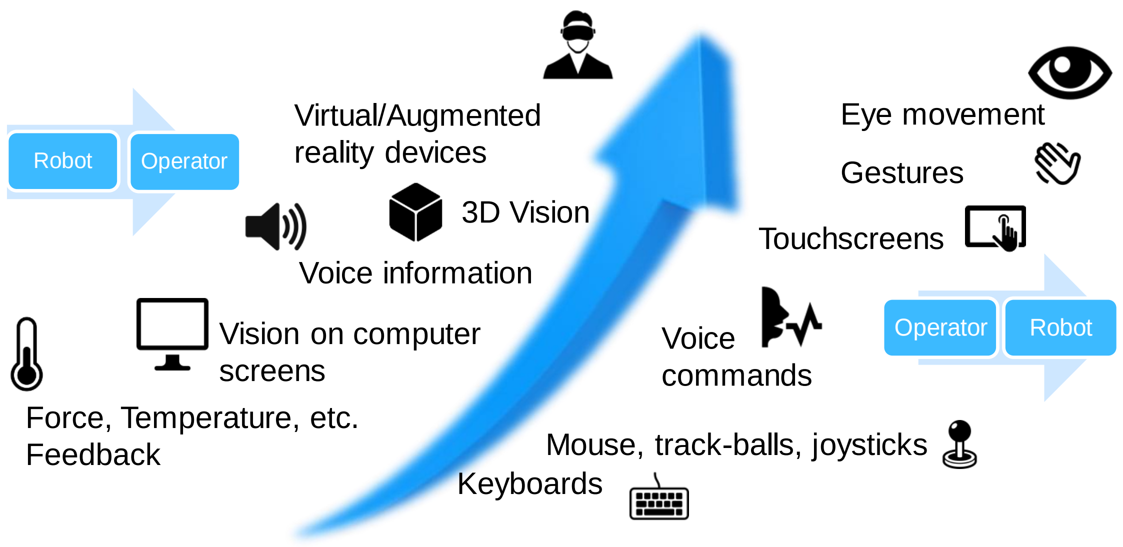

Robot
Operator
Virtual/Augmented reality devices
3D Vision
Voice information
Vision on computer screens
Force, Temperature, etc. Feedback
Eye movement
Gestures
Touchscreens
Voice commands
Operator
Robot
Mouse, track-balls, joysticks
Keyboards

Figure 2.4: I/O Modes. A mixture of modes, even potentially redundant, can improve intuitiveness of the interaction.

Chapter 3

Proposal for a Multi-Modal interface in an off-line programming environment

3.1 Description of the proposal

After the literature review, proposals based in implementing computer vision or automatic learning algorithms will be discarded in favor of achieving a programming method capable of imitating positions a human gives to a not specific robot, this way we ensure the scope of this thesis to do not mean a work overload.

Similarly as done in [13], a proposal for MMI interaction for robot programming will be given. In our case, focused on industrial robots, we propose **improving an existing off-line programming environment by adding other interaction modes**.

The MMI should implement a **base GUI**, to be added to the current off-line programming environment, allowing to select and/or configure which elements in the virtual environment will be commanded. In addition, some configuration features of the instructions and/or code generation can be added into the GUI. Additionally, the GUI can generate higher level commands than those allowed by the environment to ease its use.

Each additional interaction mode should implement a GUI ribbon added to the off-line programming environment GUI along with the GUIs of other modes. Each mode can include its own configuration in the GUI or other options as constraints for the data they offer.

The reason for implementing configuration in GUI is contradictory with the intuitiveness that a MMI is meant to achieve but as previously remarked, literature about MMI is still scarce [13], therefore further investigation should be done for erasing the need of configurations, anyway, the most appropriate configuration can be used by default.

Additionally, **each input mode should be able to interact** with the GUI, as well as with the elements incorporated in the virtual environment of the off-line programming environment, as pieces and robots.

As proposed in [15], **CAD constraints** will also be considered for this proposal as transforms in the virtual pieces of the environment. These transforms can be used to directly position a gripper to a piece, a held piece to another piece, to align a gripper to a piece, etc... The concept of keyframes is also used, in our proposal, keyframes will be created given a specific command, automating their generation to improve intuitiveness is not implemented in this thesis.

Regarding motion-based input modes, our proposal is not to use a gesture-based input but a pose based instead, meaning that the position and orientation of parts of the human body can be taken into account instead of only discrete gesture commands. This specific proposal is meant to achieve a robot-human imitation, directing further research to improve how robots can perform complex operations that humans can do on their own.

Additionally, to pose input modes, some **gestures** can be recognized with the same mode, they can either execute certain commands or to constrain the input data in function of the virtual environment (Pieces, Robots), like constraining the movement aligning the robot to an element or moving robots keeping their distance between end-effectors.

Regarding speech-based input modes, our proposal is to use speech for giving discrete commands, even being able to interact with other modes configuration or constraints on their inputs. This mode has a great potential and can be later improved with CAD information of the pieces to be manipulated or assembled, this way it is possible to give very high-level commands to robots. Anyway, our proposal is limited regarding speech, and only some simple commands will be implemented.

Given that the proposal aims for integrating modes into an existing off-line programming environment, there should exist an API for such environment. The API might use its own data formats for position, orientation, and 3D information of imported CAD models. For each input mode interacting with these elements, the data must be adapted to the same type as the one the environment uses.

Adding interaction modes to an existing off-line programming environment is possible by generating Add-Ins, if some input modes are meant to be fused they should be incorporated in the same Add-In.

A graphic summarizing the proposal can be found in Figure 3.1.

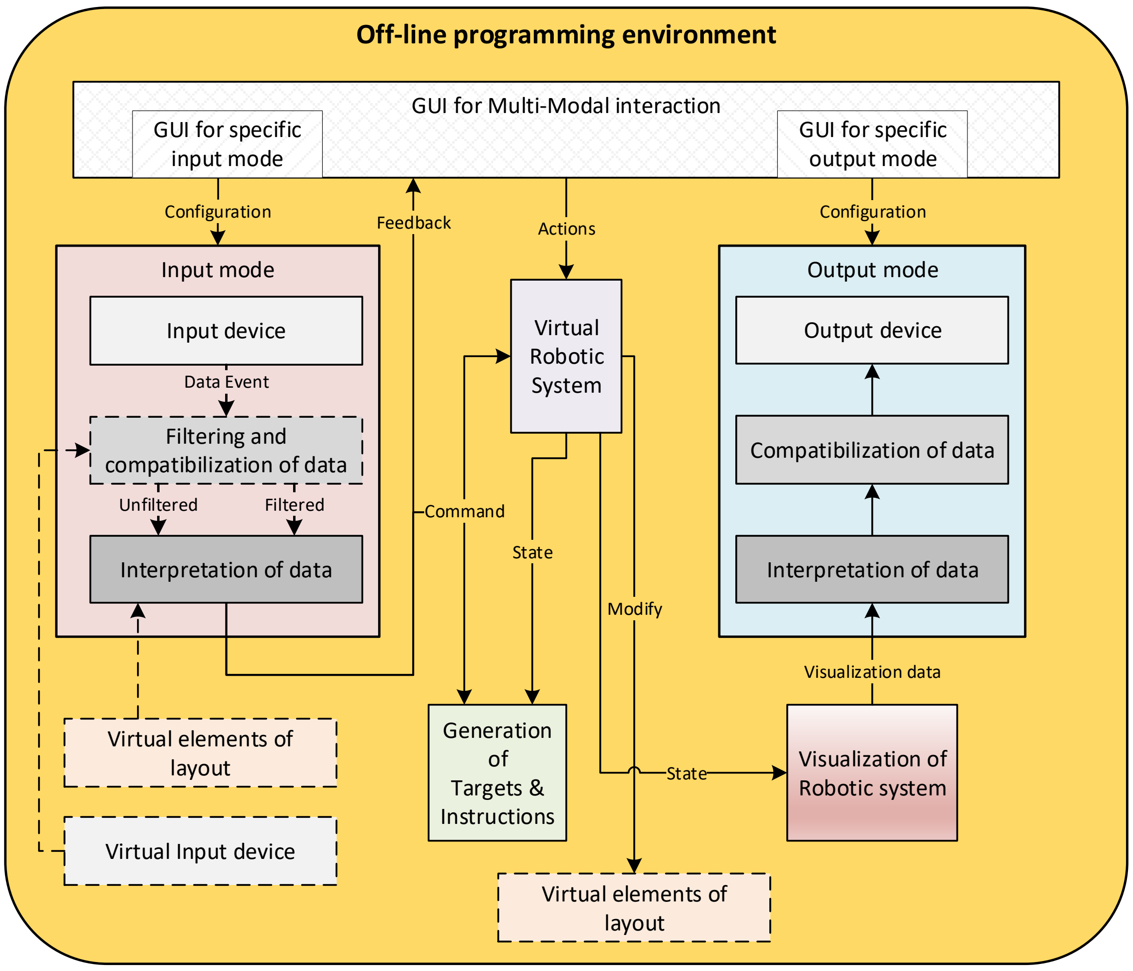

Off-line programming environment
GUI for Multi-Modal interaction
GUI for specific input mode
GUI for specific output mode
Configuration
Feedback
Actions
Configuration
Input mode
Input device
Data Event
Filtering and compatibilization of data
Unfiltered
Filtered
Interpretation of data
Virtual Robotic System
Command
State
Modify
Output mode
Output device
Compatibilization of data
Interpretation of data
Visualization data
Virtual elements of layout
Virtual Input device
Generation of Targets & Instructions
State
Visualization of Robotic system
Virtual elements of layout

Figure 3.1: Structure of the proposal.

3.2 Tests

The tests to be performed consist on programming procedures for a pair of robot manipulators, the execution of both procedures should be sucessful and therefore synchronized. The generated procedures will be RS instructions and they will be mainly generated by the additional input modes and the interface.

The next types of operations will be tested:

- Pick and place.
- Insertion using snap-fit (see Figure 3.2).
- Insertion using a bayonet mount (see Figure 3.3).
- Manipulation of an articulable piece such as a Rubik cube (see Figure 3.4).

Results of the tests are discussed at Chapter 6.

A virtual station has been prepared to test an implementation of the proposal.

The virtual station used for testing belongs to one of the robotic ABB cells of FAST-Lab at Tampere University of Technology that can be seen at Figure 3.5. The final look of the virtual station has been adapted for this work and it is the same as in Figure 3.6, this environment simulates a table with some

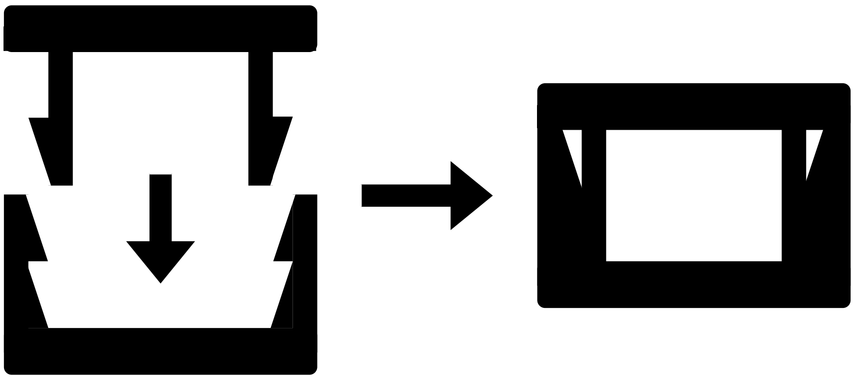


Figure 3.2: Snap-fit insertion.

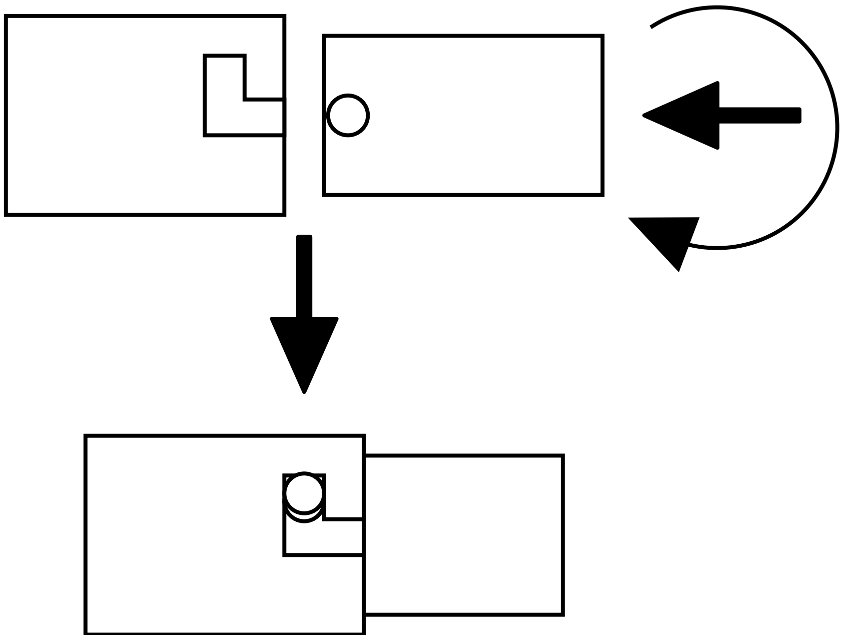


Figure 3.3: Bayonet insertion.

pieces located in designated places. In the center, it is located a thrash bin, which is supposed to cross the table.

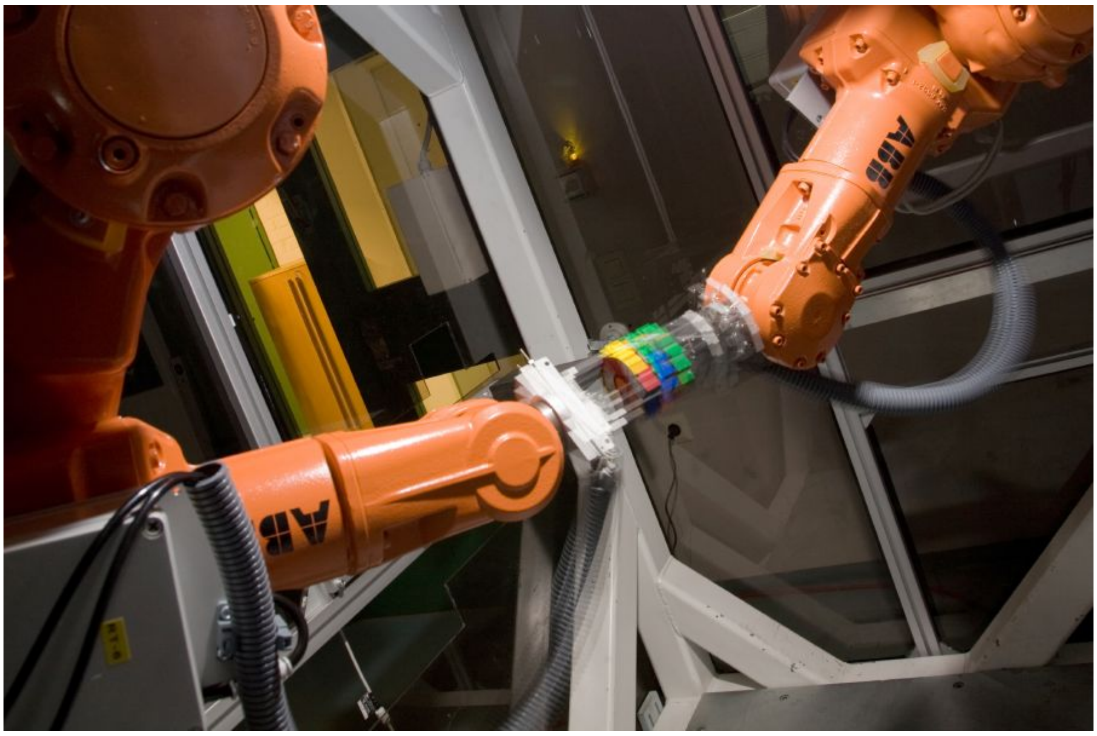


Figure 3.4: Rubik cube manipulation.

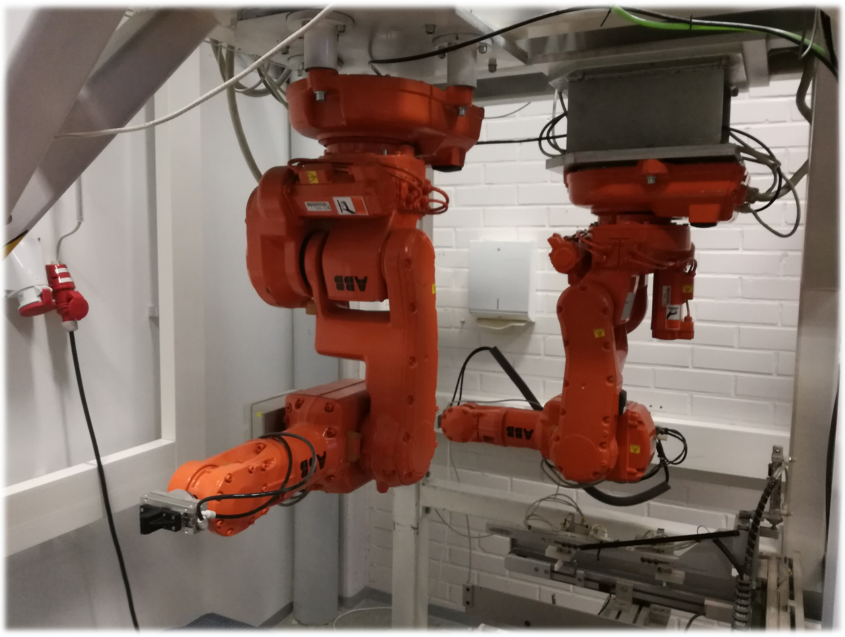


Figure 3.5: Robotic cell. The virtual model of this cell will be used for developing and testing the interface.

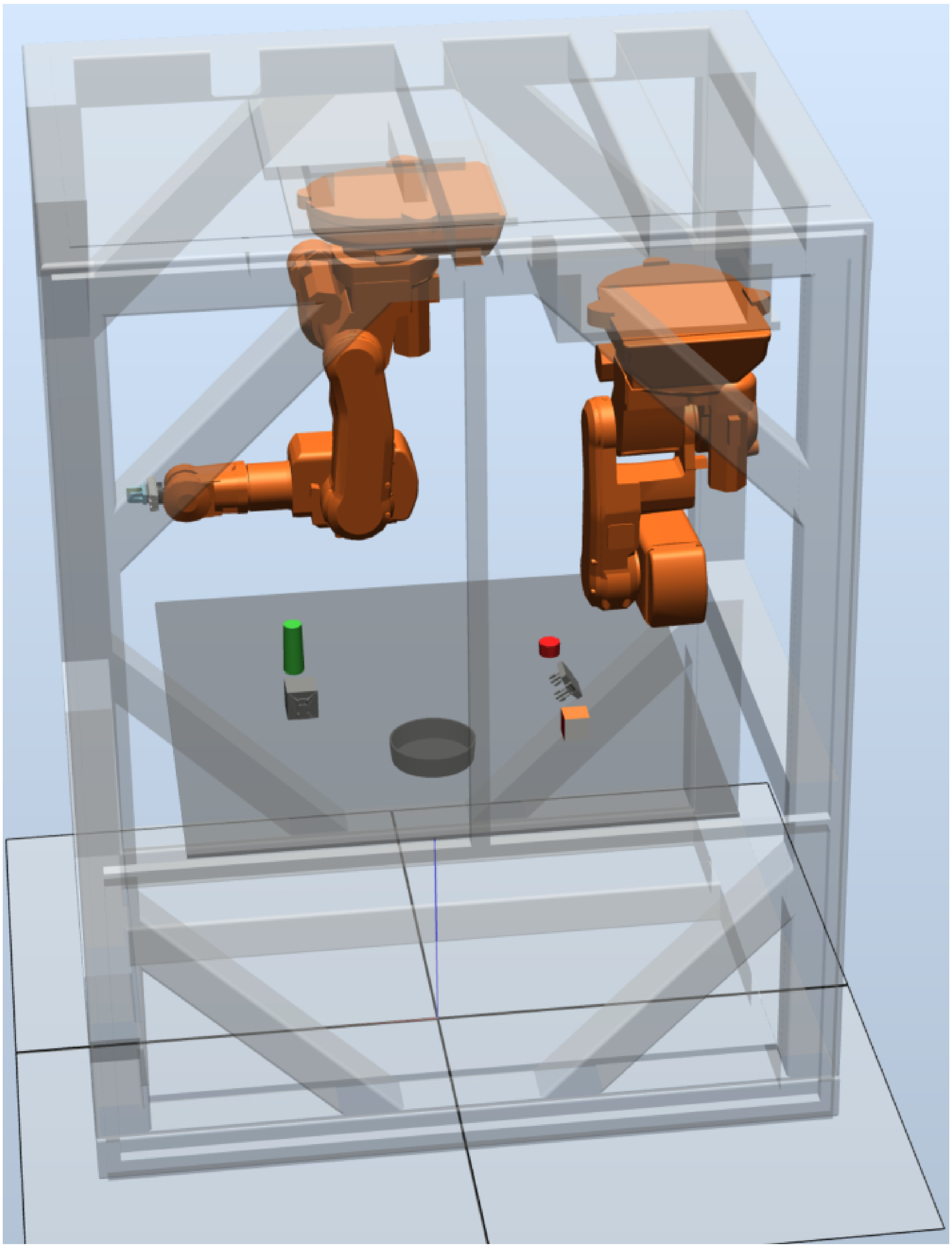


Figure 3.6: Testing environment used for making the programs using the interface.

Chapter 4

Implementation

In this chapter, the developed implementation of the proposal will be exposed in detail. The details of the environments can be found summarized in section 4.1.

Two input modes are incorporated to the interface, one of them is speech recognition, which will use the default audio input device of the computer, for this the Microsoft Speech Platform SDK 11. The other mode segments the hands of the user through a motion sensing device, the device chosen is the Leap Motion Sensor, its C# SDK is also used.

The proposed interface incorporates a GUI allowing to make some basic configuration as well as constraining the input from the motion sensing device to be used, enabling speech recognition or interact with the tools of the robots in the RS station, the GUI also provides additional output for the operator about the speech recognition and the hands pose segmentation, as well as their rotation and position in the equivalent virtual station.

The interface allows to jog up to two robots with the hands and specify targets in the station, as well as precisely align to objects in the station given their frame in order to make precise movements. The interface requires *Smart Components* in the station to provide all the output it can offer, the required configuration before using the interface will be detailed ahead.

A functionality to automatically generate path procedures, including synchronization points and output commands for tool action is also added.

It is important to highlight that more multi-modality is achieved when combining the input modes offered by this interface with the interaction modes already provided by RS.

The look of the RS software along with the proposed Add-In is shown in Figure 4.1.

About how the interface internally works, a general overview of the data flow can be seen in Figure 4.2, this one is just an orientative graph, a UML class diagram of the Add-In new classes can be found in Appendix A.

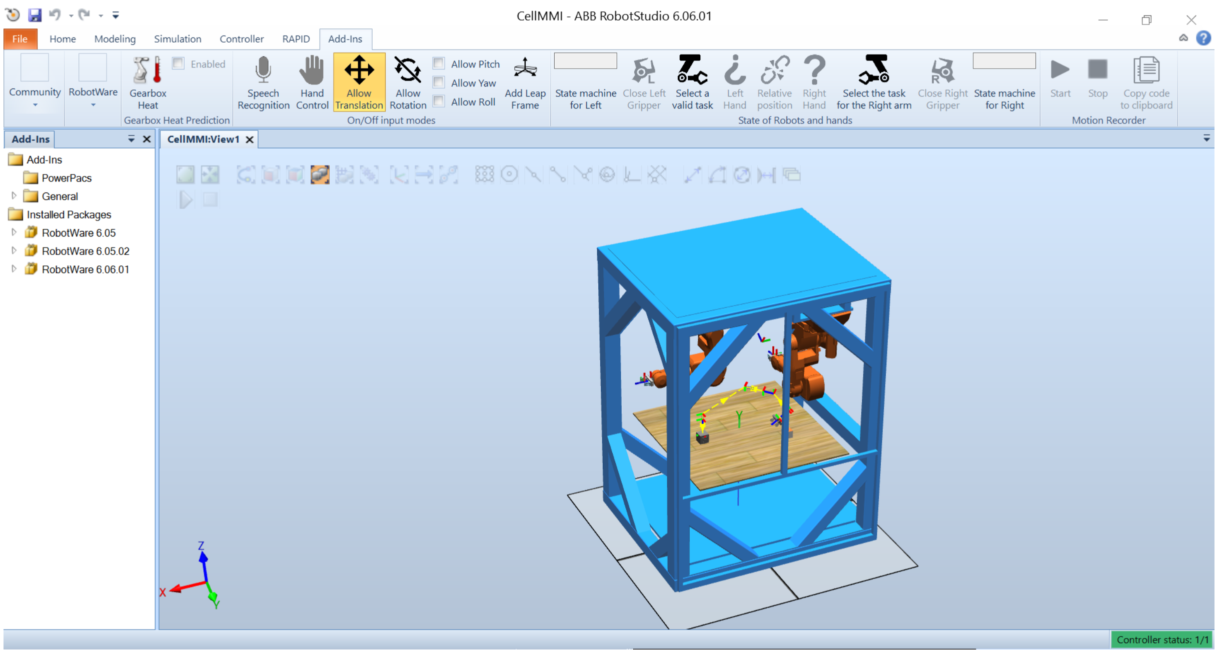


Figure 4.1: RS while in the Add-In tab. A preliminary look of the station can be seen

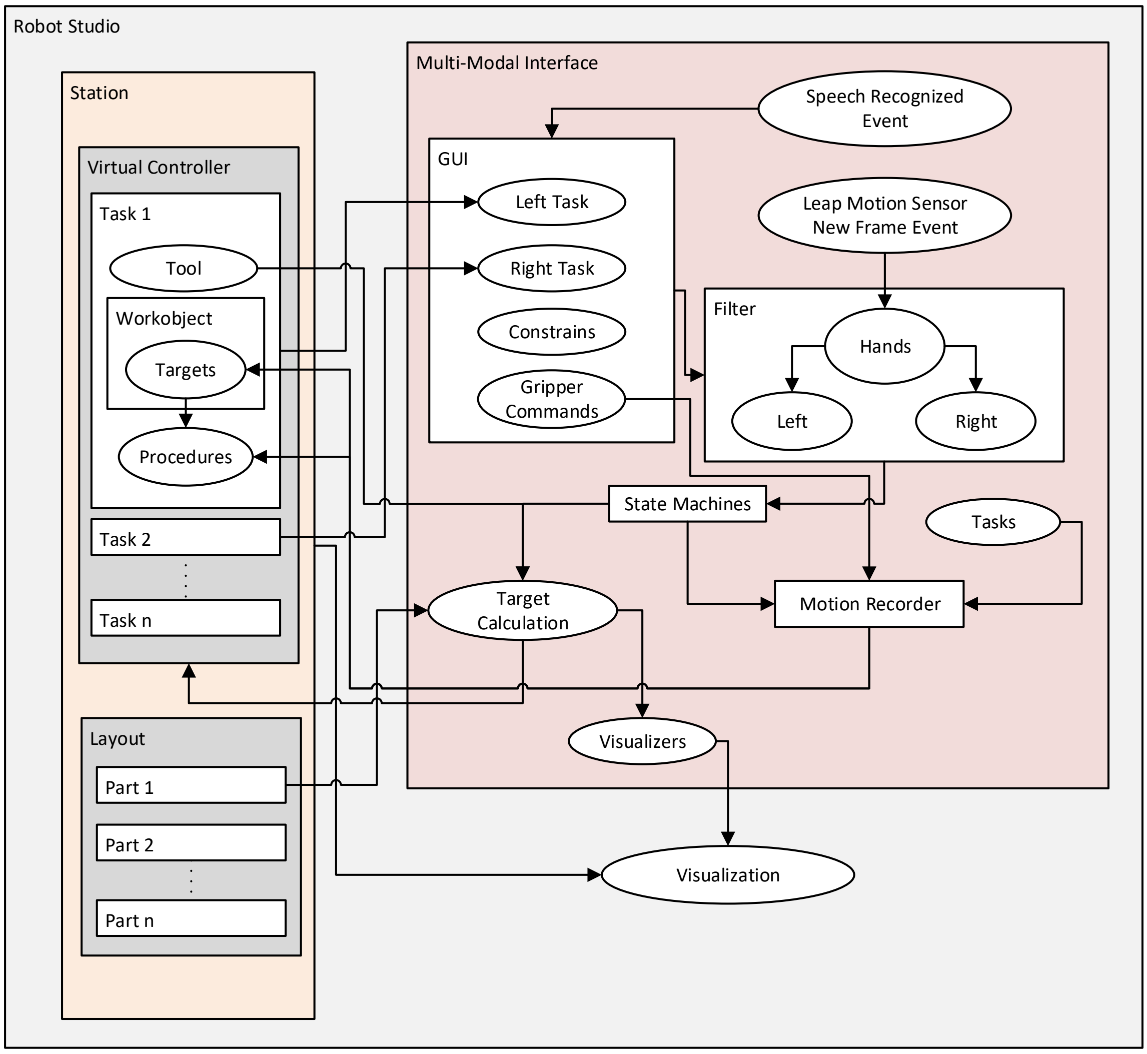


Figure 4.2: General overview of the interface internal operation. This is just an indicative graph.

4.1 Environment

In the next lines, a justification for the choice of the components to develop the implementation is given, as well as a summary of the environment.

In order to command specific positions in space with a non-classical input mode, the position of a hand in space could be useful for the robot and natural for a human, some kind of positioning sensor will be required, but some devices like the Kinect camera, the Leap Motion Sensor or the controllers used in VR devices can already give this detection.

Regarding the available material, in the case of human-motion sensors, we dispose of a Kinect camera and a Leap Motion Sensor. We can find a brief comparison between both of them below.

Whilst Leap Motion is more suitable for computer input due to its closer range, and it also has higher precision [22], we choose this device instead of the Kinect camera. Moreover, Leap Motion has several Application Programming Interfaces (APIs):

- JavaScript
- Unity
- C#
- C++
- Java
- Python
- Objective-C

Additionally, the Leap Motion documentation explains how to configure Microsoft Visual Studio to compile a program using their dynamic library.

In the case of off-line programming environments, we dispose of a Robot Studio license and the model of an existing station. RS is a specialized off-line programming environment for industrial robots and ABB also makes available an SDK for developers.

The Developer Kit offered for free by ABB contains the next SDK's:

- **RobotStudio SDK:** Allows the development of custom applications or Add-Ins to add new features to RobotStudio.
- **PC SDK:** Allows the creation of customized operator interfaces for an ABB robot controller over a network as independent applications.
- **Robot Web Services:** Exposes different APIs APIs facilitating platform independent and language communication with the robot controller.
- **FlexPendant SDK:** Allows the development of custom applications for the FlexPendant. Limited support.

On the one hand, choosing the PC SDK would require developing an interface from scratch, on the other hand, the RS SDK offers us a well-known interface and allows us to add modes to the already existing ones of the off-line environment.

As the RS SDK only supports C#. We should use the C# programming language and the respective Leap Motion API. This high-level language is easy to use and disposes of the .NET framework widely used in Windows applications. RS is only available for Windows and needs the Microsoft Visual Studio software to use its code templates. The documentation offers additional guidance in the configuration of the environment, as this is the most supported IDE that Microsoft supports, it is a good choice.

Finally, in the case of speech recognition, given that we are going to use the .NET framework, which also allows language interoperability, a good choice for speech recognition is the Microsoft Speech Platform SDK 11, which can be installed in addition to the .NET framework.

To summarize, the Add-In will be programmed using the RS SDK in the Visual Studio IDE along with the .NET framework, adding it the Microsoft Speech Platform SDK 11 and the en-US speech recognition package. The Leap Motion SDK for the Leap Motion Sensor is also required. The Add-In will be executed, debugged and tested executing it in RS. Everything will be programmed in C#. As devices, a laptop with an embedded microphone and the Leap Motion Sensor will be used.

4.2 Design and implementation of the GUI

The GUI that implements the interface is shown in Figure 4.3.

The buttons of the Add-In GUI, as well as their icons, can dynamically change, depending on speech input or the filtered data from the Leap Motion Sensor, these changes will be explained later along the remaining sections of this chapter.

Now we will explain the function of each button:

- **Speech Recognition:** This button enables and disables the speech recognition, it can be clicked anytime.
- **Hand Control:** This button enables when to consider the input given by the Leap Motion Sensor or not, it can be clicked anytime but depending on many factors it may not activate and instead of it, it will prompt a message asking for additional configuration actions.
- **Allow Translation**: This button constrains the position of the manipulator's tool in space, meaning that it can be translated depending on the position of the user's hands, it is useful when a precise turning with no translation must be performed.
- **Allow Rotation**: This button constrains the rotation of the manipulator's tool, it works in the same way that Allow Translation but with the rotations of the hands instead of their location. Clicking this button will activate pitch, yaw, and roll of the tool or disable all of them.
- **Allow Pitch (Roll, Yaw)**: Each of these three checkboxes can be independently activated, when all are unchecked the allow rotation button will also be unchecked if at least one of these is checked, the Allow Rotation button will also be checked.
- **Add Leap Frame**: This button aids configuration of the station, first, it will check if there is a frame in the station called LeapFrame, if not, it will create a new one, in any case, it will prompt the user to align the frame axes correctly so they correspond to the Leap Motion Sensor location, the frame will correspond to the frame of a virtual Leap Motion Sensor in the station, more information about this can be found in section 4.4.
- **Close Left (Right) Gripper**: By clicking it the gripper attached to the robot will open or close, it requires a smart component as a tool and to configure a station logic.
- **Select Left (Right) Task**: Each button displays a combobox to select among the tasks in the station to tell the interface which manipulator is going to be moved with the left or right hand. Moreover, they will indicate their own robot alignment state since the RS API does not implement indicators.
- **Left (Right) Hand**: These buttons serve just as indications, their image will change depending on the detected hand pose, more information can be found on section 4.4.

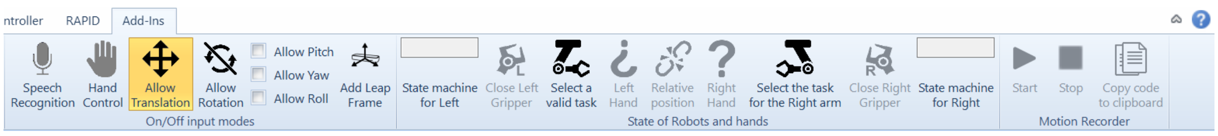


Figure 4.3: GUI of the Add-In. A checked button and disabled buttons are visualized

- **Relative Position**: This button is intended to constrain the relative position between the tools of both manipulators, but it is not implemented yet.
- **Start**: Starts the motion recorder, as well as it creates a new procedure in both tasks named by a timestamp, targets recorded with the interface will be chained in a path procedure, as well as in a vector with additional alignment information about this, check section 4.5.
- **Stop**: Stops the motion recorder, new targets recorded will not chain to the path procedure that was being programmed.
- **Copy code to clipboard**: Intended to copy the RAPID code for both tasks to the clipboard once the motion recorder is stopped, it is not implemented yet.

4.3 Configuration of the environment for the interaction

4.3.1 Station requirements for minimum use

In order to use the Add-In, it requires an RS station, containing a controller with two robots in it, it can be easily done by following these steps in the RS GUI:

1. In a new RS station, import two robots of your choice from ABB library, as in Figure 4.4.
2. Position them as wanted using the RS GUI.
3. Create a new Robot system from layout, as in Figure 4.5. Give it a name and select the *RobotWare* package of your choice. Finally, select the mechanisms to be included in the controller.

Additionally, to use hand input, a *LeapMotionFrame* must be included, this serves as a virtual Leap Motion Sensor in the virtual station. The Add-In GUI can be used for placing it, the corresponding button will generate the frame and give instructions to the user on how to position the frame. If the frame is not well positioned, the user will figure it out by moving the hand and checking that the hand movement does not correspond with the movement in the RS station.

Finally, it is required to select a task for each hand using the buttons for selecting tasks, both tasks must be selected, in case there is only one robot to be programmed, the same task can be assigned for both hands, then, it does not matter what hand to use.

This minimum configuration allows the user to use the hand inputs, record movement and use the speech for constraining hand inputs. It does not allow automatic generation of synchronization points or to simulate how pieces are attached during the recording process of the interface.

In case there is a robot with less than 6 Degrees of freedom (DOF), some of the GUI constraints need to be applied, so the hand rotation does not make it impossible for the robot to reach it.

4.3.2 Smart Component for tools and Station Logic

To use more features of the interface, specially commanding and recording tool actions, the tools attached to each robot must be Smart Components. Using a Smart Component as a tool allows it to, for example, attach pieces of the station to it.

In order to program assembly or manipulation operations, it is necessary for the Add-In to know when a piece is being attached to the robot or detached. It is also needed some way to leave a piece attached to the other robot gripper when it is detached by one robot and the other is still holding it, in other words, when both robots were holding a piece.

An example of *Smart Component* for a parallel mechanical friction gripper with two fingers with a unique signal for closing and opening is given, it senses if it also senses if it is opened or closed, Figure 4.6 gathers all the child components of the gripper and Figure 4.7 represent the logic these components follow.

The Add-In functionalities should work correctly as long as the user configures a *Smart Component* capable of attaching, detaching, and giving the attached piece of a robot to the other with some logic.

The input and output signals of the *Smart Component* also need to be configured to be matched to inputs and outputs of the Add-In. For this, the Add-In helps the user by automatically generating the inputs or outputs it requires in the RS station.

The Add-In generates inputs *CloseLeft*, *CloseRight* and *CheckGrippers*, and the outputs *IsClosedLeft* and *IsClosedRight* in the station logic, these I/O must be linked to the tools in a logic way, additional components can be added to the station if the user *Smart Components* I/O do not correspond to the I/O generated by the Add-In, for example, to invert a boolean value or set/reset a signal. An analogy for vacuum grippers can be also done from mechanical ones, the closure can be understood as suction.

The use of these I/O created by the Add-In is to be able to connect the GUI buttons for tool commanding to the tools in the station, therefore it serves for simulating the pieces movement when programming a task with the Multi-modal interface.

4.3.3 Defining pieces

In order to define a piece and to use the align feature that the developed interface implements, it is not enough to just import a CAD geometry in RS.

Unfortunately, RS does not allow to define some frames into a piece, only attaching frames of the station to them, this issue is not consistent with the RS SDK API, which gives access to a collection of frames in each graphic component, there is not, at least intuitive, way to add frames to a Part in RS.

There is a way to define a frame for the part, it is defining its local origin, that local origin orientation is going to be used for alignments, for this, the piece must be configured as follows:

1. Create an empty part as in figure 4.8a.
2. Link it your CAD geometry as in figure 4.8b, a file browser will appear.
3. Edit its local origin in order to do it, do as in figure 4.8c, making its rotation to be the same as the tool, as visualized in figure 4.8d.

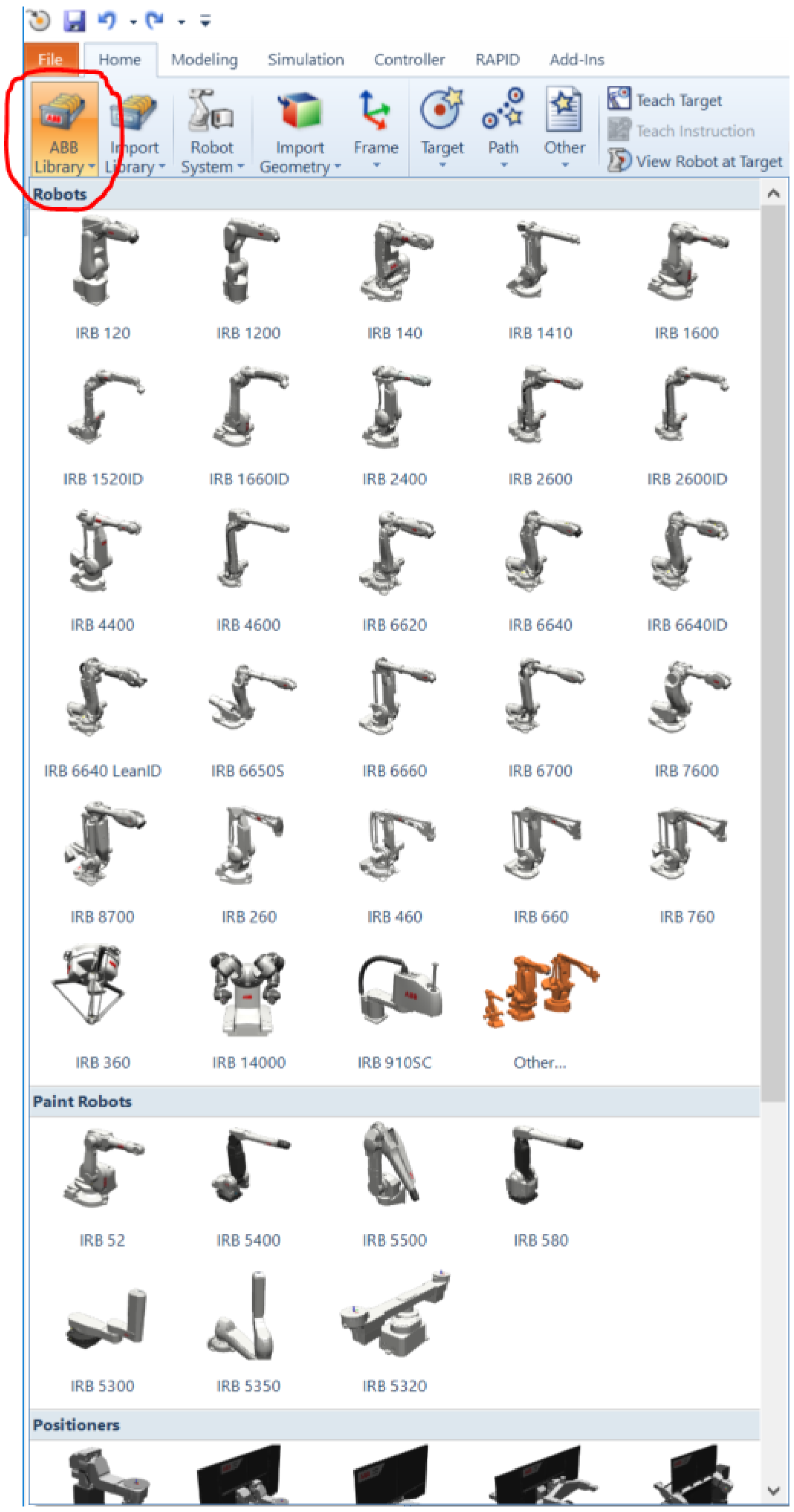


Figure 4.4: How to add robots to the station, using the RS GUI and their library.

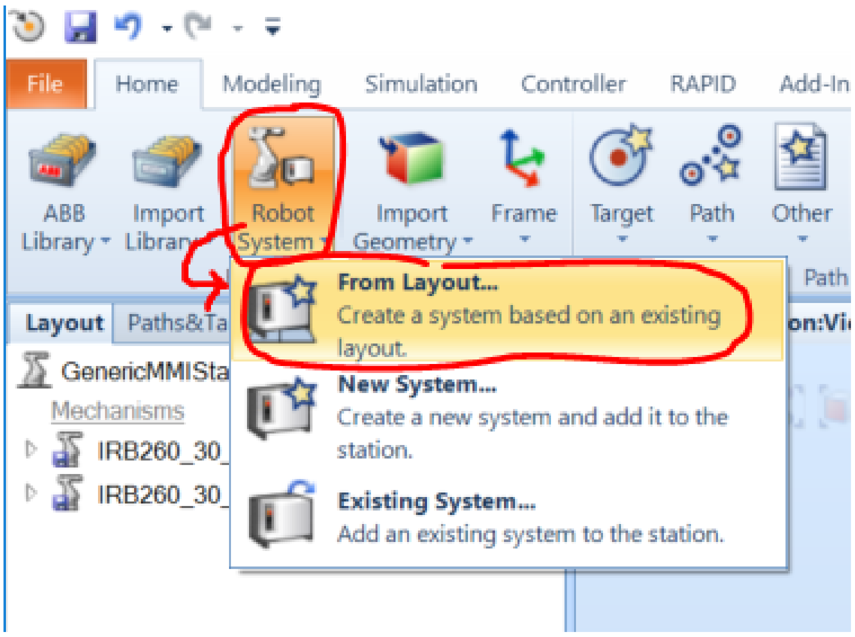


Figure 4.5: How to add a controller containing both robots, more tasks can be later included if needed.

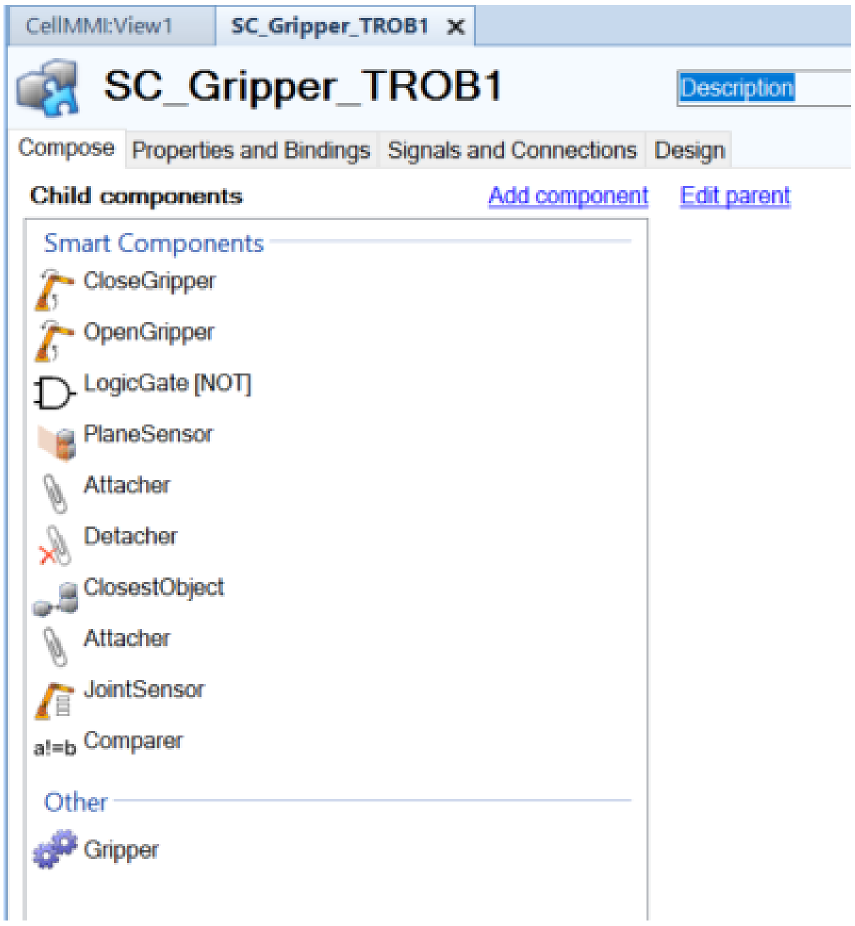
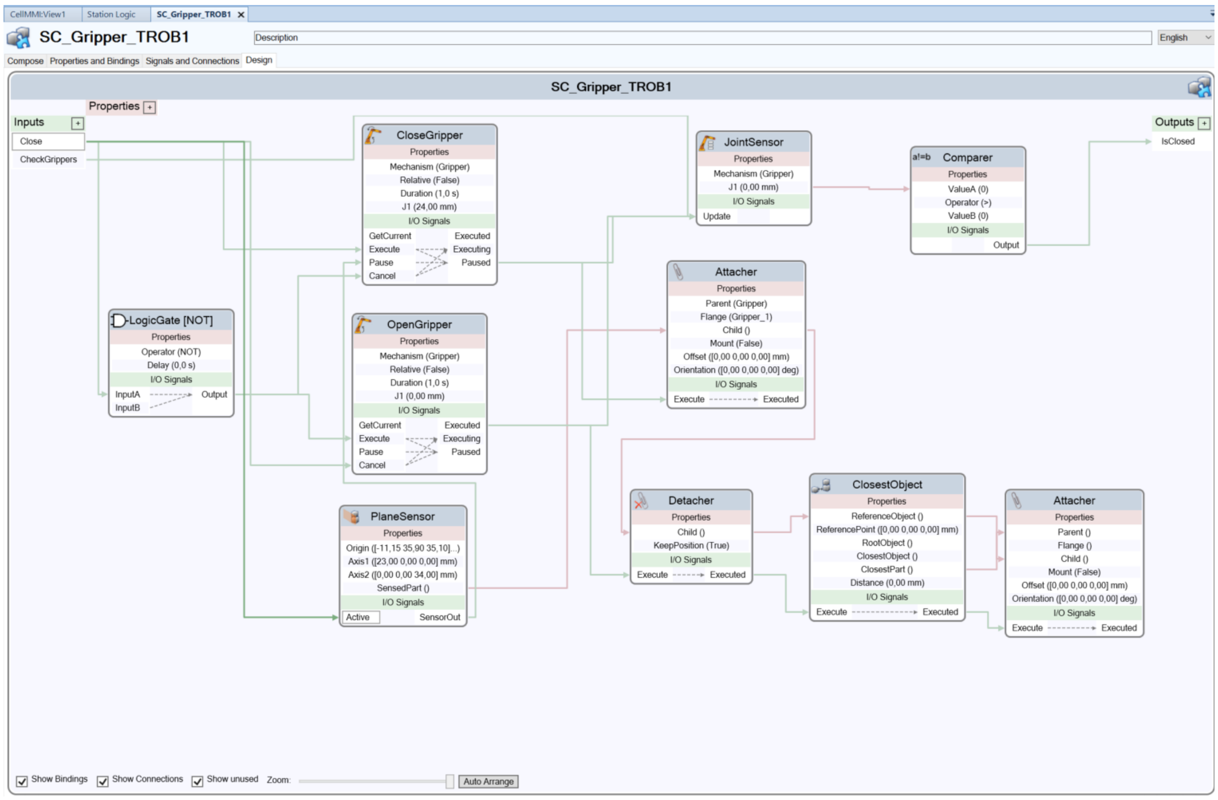


Figure 4.6: Gripper child components. Each of these components works as a function block in the Smart Component design, it also includes the gripper mechanism.



CloseGripper
Properties
Mechanism (Gripper)
Relative (False)
Duration (1,0 s)
J1 (24,00 mm)
I/O Signals
GetCurrent, Execute, Pause, Cancel → Executed, Executing, Paused

JointSensor
Properties
Mechanism (Gripper)
J1 (0,00 mm)
I/O Signals
Update

Comparer
Properties
ValueA (0)
Operator (>)
ValueB (0)
I/O Signals
Output

LogicGate [NOT]
Properties
Operator (NOT)
Delay (0,0 s)
I/O Signals
InputA, InputB → Output

OpenGripper
Properties
Mechanism (Gripper)
Relative (False)
Duration (1,0 s)
J1 (0,00 mm)
I/O Signals
GetCurrent, Execute, Pause, Cancel → Executed, Executing, Paused

Attacher
Properties
Parent (Gripper)
Flange (Gripper_1)
Child ()
Mount (False)
Offset ([0,00 0,00 0,00] mm)
Orientation ([0,00 0,00 0,00] deg)
I/O Signals
Execute → Executed

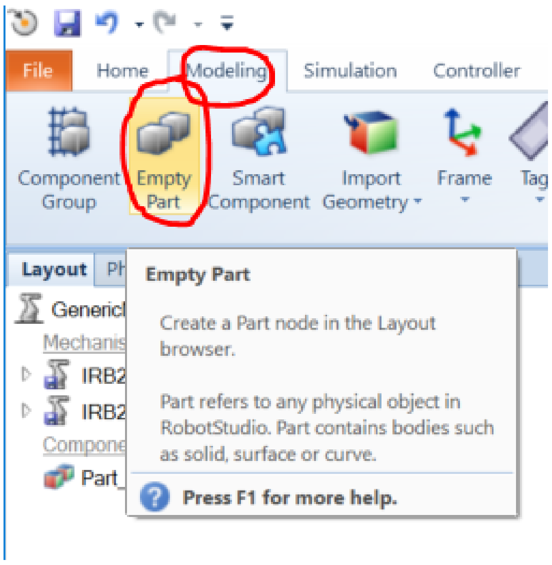
PlaneSensor
Properties
Origin ([-11,15 35,90 35,10]...)
Axis1 ([23,00 0,00 0,00] mm)
Axis2 ([0,00 0,00 34,00] mm)
SensedPart ()
I/O Signals
Active, SensorOut

Detacher
Properties
Child ()
KeepPosition (True)
I/O Signals
Execute → Executed

ClosestObject
Properties
ReferenceObject ()
ReferencePoint ([0,00 0,00 0,00] mm)
RootObject ()
ClosestObject ()
ClosestPart ()
Distance (0,00 mm)
I/O Signals
Execute → Executed

Attacher
Properties
Parent ()
Flange ()
Child ()
Mount (False)
Offset ([0,00 0,00 0,00] mm)
Orientation ([0,00 0,00 0,00] deg)
I/O Signals
Execute → Executed

Figure 4.7: Gripper smart components design. Connections between the components and their properties define the behaviour of the gripper with the rest of the station.



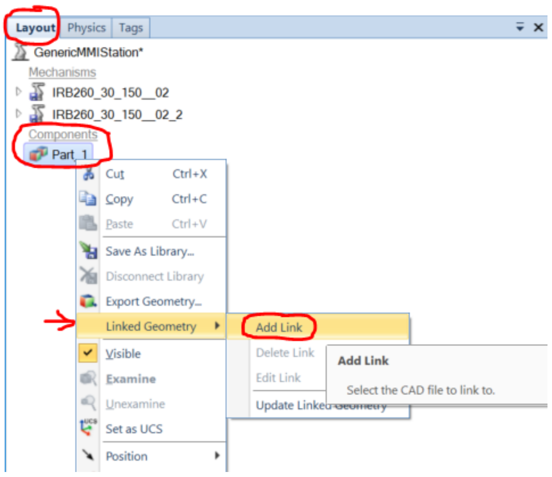
Empty Part

Create a Part node in the Layout browser.

Part refers to any physical object in RobotStudio. Part contains bodies such as solid, surface or curve.

Press F1 for more help.

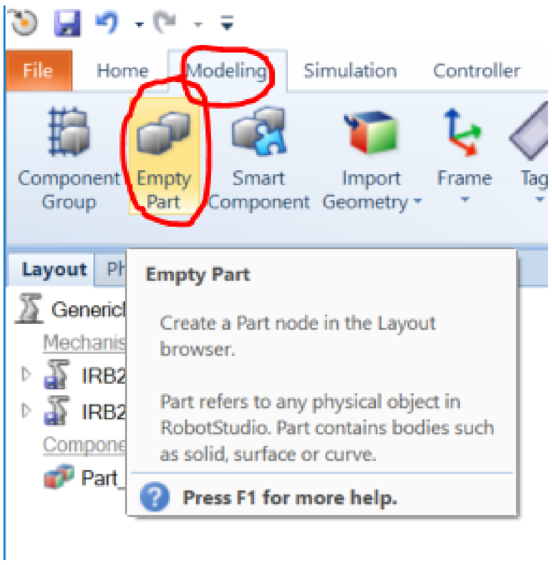
(a) Creating an empty part.



Add Link

Select the CAD file to link to.

(b) Linking the CAD geometry.



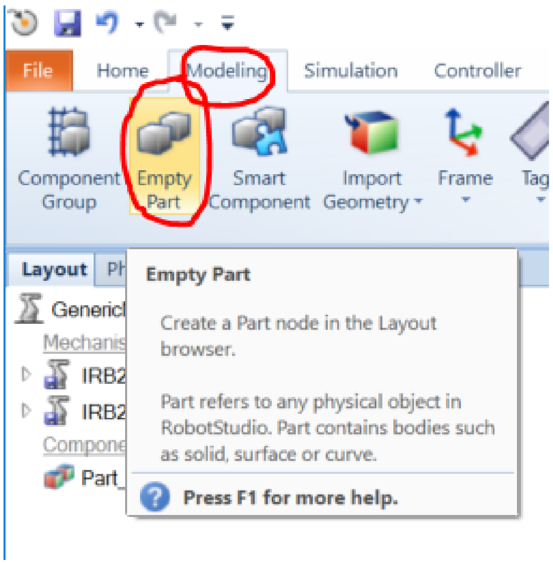
Empty Part

Create a Part node in the Layout browser.

Part refers to any physical object in RobotStudio. Part contains bodies such as solid, surface or curve.

Press F1 for more help.

(c) Editing the local origin.



Empty Part

Create a Part node in the Layout browser.

Part refers to any physical object in RobotStudio. Part contains bodies such as solid, surface or curve.

Press F1 for more help.

(d) Tool aligned to local origin of the piece.

Figure 4.8: Steps for configuring a piece.

4.4 Motion Sensing Input to RobotStudio Data

The Leap Motion Sensor is an inexpensive and small plug and play device. The manufacturer offers a developer SDK for it in their webpage. The sensor incorporates two cameras and three infrared LEDs and has a software embedded to make a segmentation of the hands.

Their API is simple and easy to use, it provides an event that will be triggered each time a new pair of camera frames has been processed, and this event is the one to be attended if it has been specified to attend it in the GUI, sample images taken from their program Leap Motion Visualizer are displayed on Figure 4.9.

The information to be extracted from the hand will be the next:

- Translation vector of the hand palm.
- Unitary vectors of the palm direction and the palm normal.
- Determine whether the hand is a left or a right one.
- Determine whether the hand is in one of these poses:
  - Opened.
  - Closed.
  - Making the thumb up sign.
  - None of the mentioned.

All the features mentioned above can be easily obtained using the API. The frame data has to be interpreted and it will also be filtered, the steps to follow are shown below:

1. Check the number of hands in the frame if zero or more than two ignore the frame.
2. Determine which fingers are extended to get the hand pose.
3. Discard the hand if there are fingers extended making a non-implemented gesture.
4. Filter the palm position, the palm direction, the palm normal and its pose with a low-pass filter along with $n - 1$ previous samples, regarding if the hand is the left or the right one.

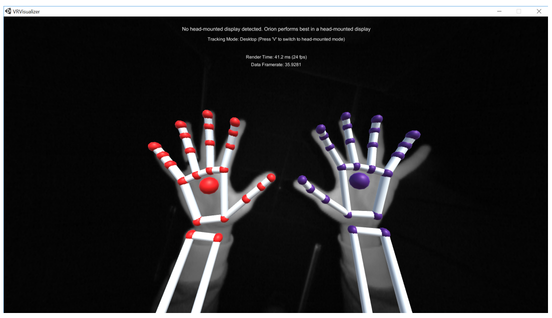
The low-pass filter helps to stabilize the value of these features, avoiding noise in the detection, it just consists on calculating the mean value of n samples, for the translation vector of the palm and its direction vectors, for these last ones, the value of the summatory of the vector will be normalized instead of divided by n.

In the case of the hand pose filtering, the last n samples have to be the same in order to change to a new pose, meaning that fast changes of pose or momentary misdetections will be ignored, instead of them the last valid pose will be considered, when the hand is not found in the frame there is an exception, the filtered pose will automatically be set as *Gone*.

In order to give feedback to the user, each time that the filtered position changes an event is raised to refresh an indicator in the GUI, the image varies depending on the filtered hand pose, the icons used for this visual feedback are shown in Figure 4.10, note that the thumb up pose will be really pointing horizontally, if not, the thumb will not be seen by the sensor.

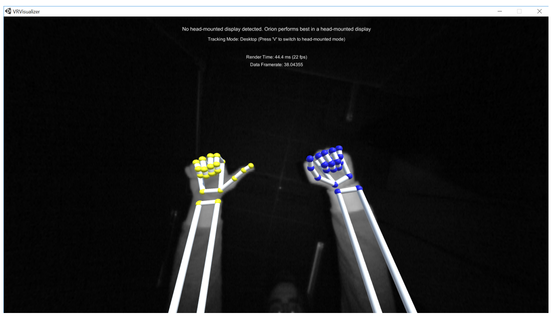
Coordinates of the hand will be obtained referred to the Leap Motion Sensor frame; since now, understand frame as XYZ axes, not as a camera frame; this frame is defined by default, as seen in Figure 4.11.

In order to refer the values given by the sensor to the RS station, we will add a frame to the station using the GUI, giving instructions to the user on how to direct its axes in the virtual robot cell.



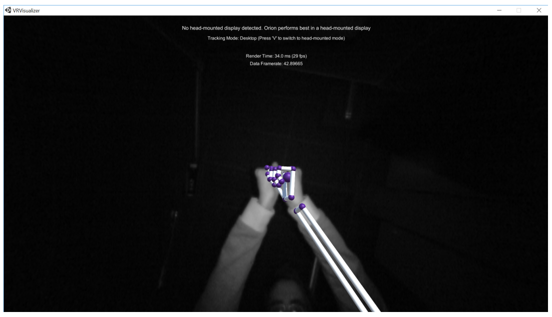
No head-mounted display detected. Orion performs best in a head-mounted display
Tracking Mode: Desktop (Press 'V' to switch to head-mounted mode)
Render Time: 41.2 ms (24 fps)
Data Framerate: 35.9281

(a) Opened hands.



No head-mounted display detected. Orion performs best in a head-mounted display
Tracking Mode: Desktop (Press 'V' to switch to head-mounted mode)
Render Time: 44.4 ms (22 fps)
Data Framerate: 38.04355

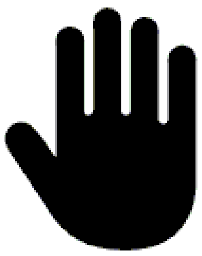
(b) Closed and "thumb up" hand.



No head-mounted display detected. Orion performs best in a head-mounted display
Tracking Mode: Desktop (Press 'V' to switch to head-mounted mode)
Render Time: 34.0 ms (29 fps)
Data Framerate: 42.89665

(c) Misdetection when hands are put together.

Figure 4.9: Captures of the sensor readings. Different situations are displayed.


(a) Icon of opened hand pose.

(b) Icon of closed hand pose.

(c) Icon of hand in the thumb "up" pose.

(d) Icon of unknown hand pose or not detected hand.

Figure 4.10: Icons of left hand poses. One icon for each hand will be displayed in the interface, they will be horizontally flipped for the right hand

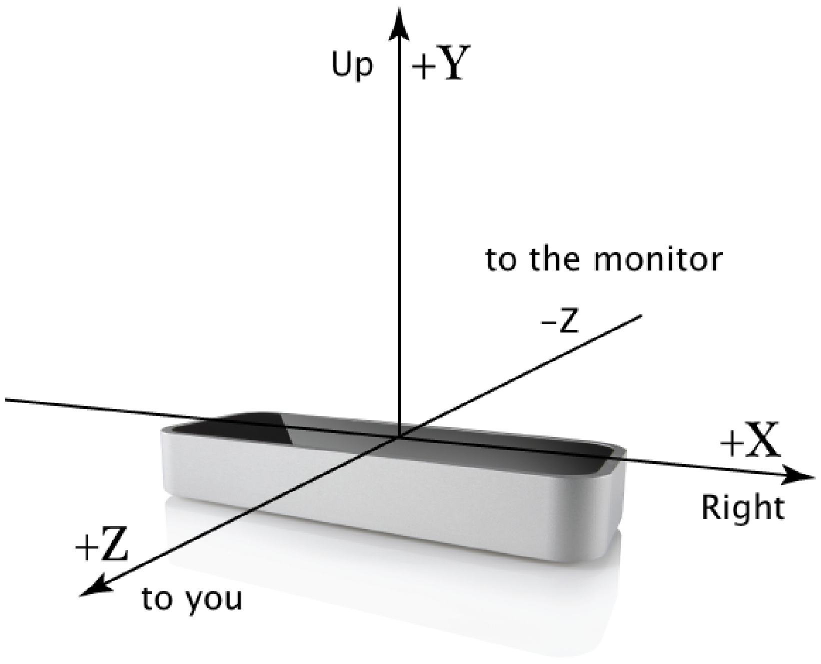


Figure 4.11: Leap Motion Sensor axes. Values given by the sensor are referred to these axes.

Currently, the interface does not implement a sensibility parameter for the hand pose reading, meaning that the same amount of movement of the real hands will be transmitted to the RS virtual station.

The operation to refer the vectors $\vec{V}$ to the world frame coordinates, given the transform L of the sensor frame is shown in equation 4.1.

$$\vec{V_O} = L^{-1}\vec{V_L} \tag{4.1}$$

Finally, when values are filtered, the output data of the filter is transformed to the data types used by RS, as well as the raw data used as input for the filter.

To summarize, the filtering process along with the consecutive process of commanding and visualization that will be explained in section 4.5 is depicted as a flowchart in Figure 4.12.

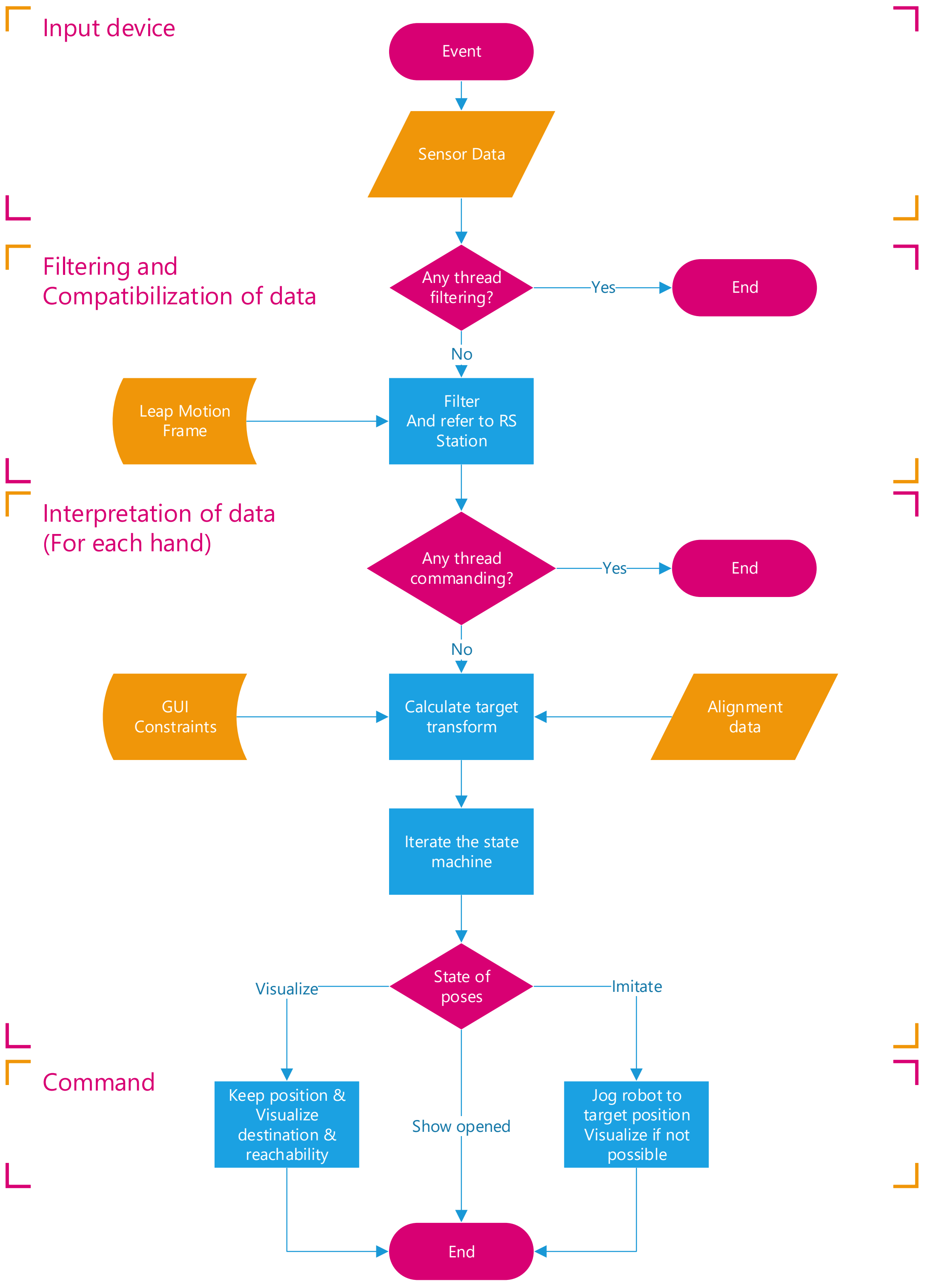

Input device
Event
Sensor Data
Filtering and Compatibilization of data
Any thread filtering?
Yes
End
No
Leap Motion Frame
Filter And refer to RS Station
Interpretation of data (For each hand)
Any thread commanding?
Yes
End
No
GUI Constraints
Calculate target transform
Alignment data
Iterate the state machine
State of poses
Visualize
Imitate
Command
Keep position & Visualize destination & reachability
Show opened
Jog robot to target position Visualize if not possible
End

Figure 4.12: Flowchart of the Leap Motion Sensor Input Mode. Interpretation stage is done for each hand, with different state machines. Feedback sent to the GUI is excluded in this representation.

4.5 Robots Movement

Now that we have positioned the user hands on the virtual station, some feedback has to be given to the user as well as determine whether to use that position to move the robots or not and how to record data for the desired program. The process is briefly summarized along with the filtering in Figure 4.12, excluding data generation.

4.5.1 State Machine

In order to make easier what to do with the extracted hands information a state machine has been implemented, not only the positions of the hands will be important but also how a pose is changed to another. The same state machine class will be used to create two objects for each hand, so they will run independently from each other. Transitions for the state machine are retrieved from the filtered pose and the unfiltered pose, also called instant pose of the hand.

The main states will be named as follows, their explanation is also given:

- Show opened hand: The hand was not found in the filtered pose that in this case equals the instant one.
- Opened: The hand was found as opened in the instant pose.
- Closed: The hand was found as closed in the filtered pose, meaning that a closed hand cannot be directly used.
- Align: This is a boolean flag rather than a state, each time a rising edge of thumb up pose is read from the filter output it will be toggled, more information is about this is given later.

As it can be seen, the machine jumps to the opened state with an instant pose, this is done because when preceded by a filtered closed state an event to record the current position will be launched and while opening/closing the hand, the rotation and position read by the sensor tends to vary more than usual.

As well as each one of the state machines is associated with their respective hand, the reference of the corresponding task (robot) selected with the GUI will be associated with the same state machine object.

Different actions will be done with opened or closed states, an opened state will visualize if a position is reachable by the robot while a closed state will perform movement to the target position of the hand in the station, as long as it is possible to make it, more information about visualization and movement is given in section4.5.

The state machine also does special actions when some transitions are given:

- From Closed to Opened: An event is raised, this event is used to record the target corresponding to the current position of the robot if the motion recorder is active, see section 4.7.
- From Opened to Closed: Raises an event, but no handler has been implemented for it.
- Point pulse: A rising edge of a filtered pose of thumb up, reads the robot current position to find an object in the virtual station to align with.

The state machine graph for each of the hands is represented in Figure 4.13.

4.5.2 Visualization and movement process

After filtering and iterating the state machine, depending on the machine state and the readings from the sensor, actions may be performed in the elements of the virtual RS station. The end of using a state machine allows the user to have some control avoiding involuntary behaviors. The movement provoked by the user's hand on the robots is called imitation, and it will work as follows:

Opened state: A big RS frame will be displayed on the station to give the user feedback of its position. In addition, the frame will be linked with an arrow to the robot's tool end-effector, the color of the line will be green if the position is reachable with any configuration and red otherwise. The colors of the frame axes will follow the color code used in RS, RGB for XYZ, red for X, green for Y and blue for Z. The coordinate system is dextro-rotatory.

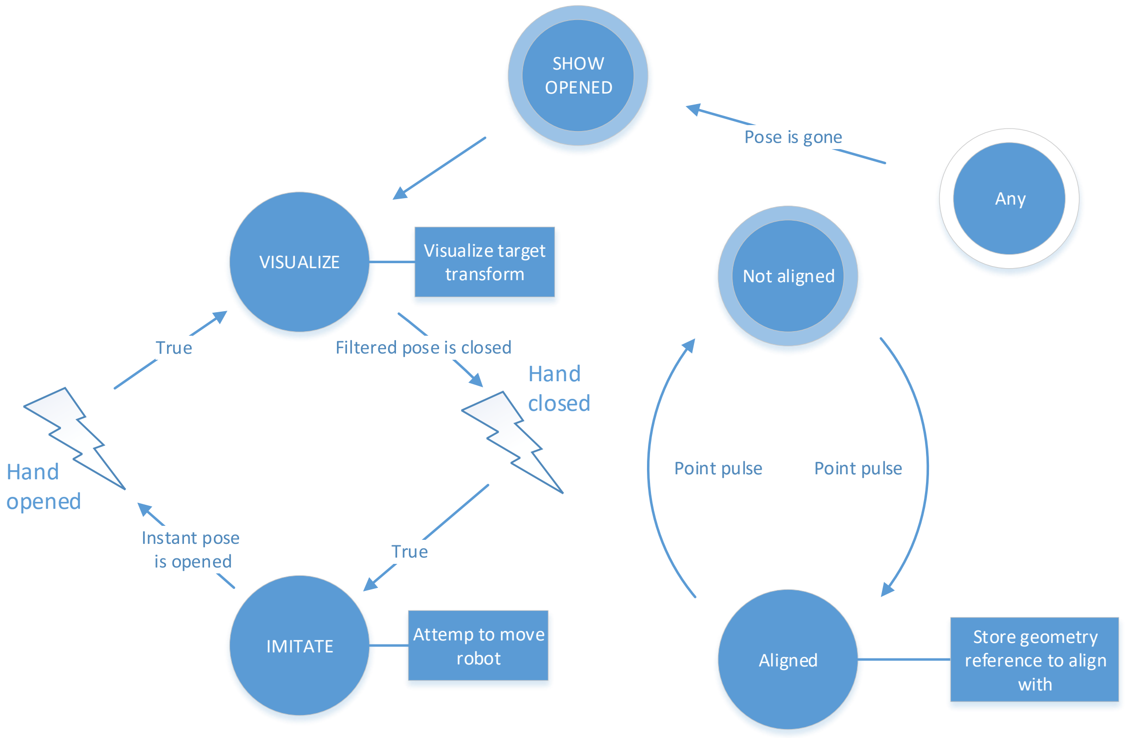


Figure 4.13: State machine for each hand. Lightning symbols represent triggered events.

Closed state: In case we show the sensor an opened hand and given the visual feedback we close it, we will drag the corresponding manipulator to the target position, a different approach where opening and closing the hands would have opened and closed the grippers instead of controlling when to move or not the robots could have been done, but this action has been relieved to the GUI or the speech because the precision needed for successful manipulation is not easily achieved by the sensor. An example of this display and the moved robot is given in Figure 4.14.

Alignment state: A parallel state that is combined with the previous one, this state makes the robot or the target frame visualization to be constrained to the Z-axis of the frame of a piece in the station. Making a thumb up pose with the hand will enable or disable this mode, it provides more precision to engage a piece with a gripper and also to align both manipulators when each of them is holding a piece. When this state is enabled the corresponding task button of the GUI will incorporate a chain. Additionally, the visualization will show a dark cyan line between the target position and the aligned piece. The difference of icons and the visualization of the alignment can be seen in Figure 4.15.

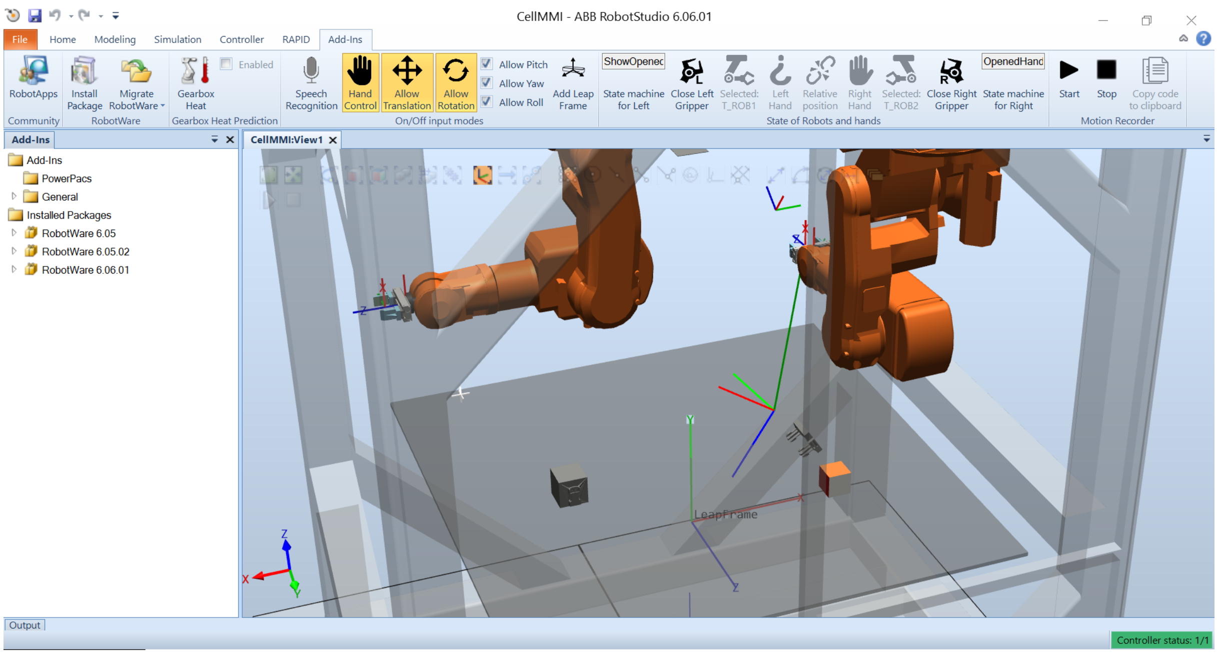
Now that the visualization feedback regarding the movement of the robots has been explained, how this movement is calculated and executed is going to be explained. First, the states of the state machine are read, the difference between an opened state and a closed state is just the robot movement as explained before.

For making a movement we need to know the active work object of the RS task and the active tool data. If any options are available they can be changed manually using the RS GUI, the Add-In does not implement other ways to do it.

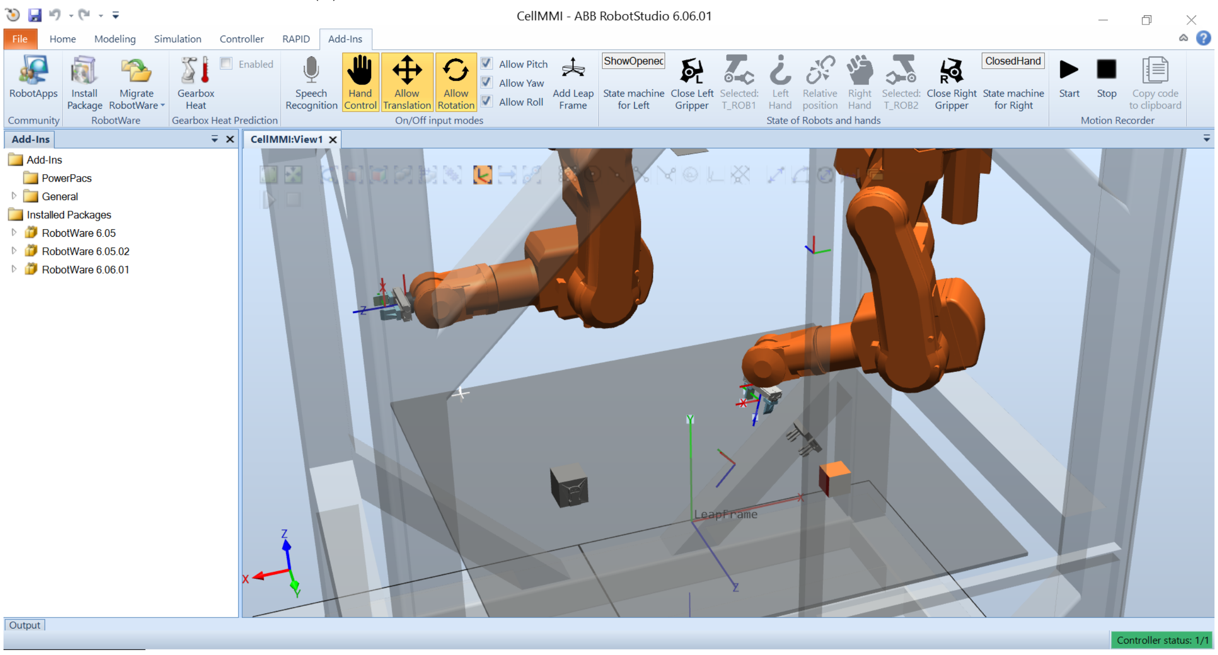
In the first place, it is necessary to calculate the target destination. Along with the filtered sensor data, we add some constraints specified in the GUI, these ones can constraint the translation and the rotation of the manipulator tool, we can detail the calculations of each type of movements.

Rotation matrix of the hand is extracted given its palm normal and its palm direction. There is an analogy between the hand and the gripper. When closing a hand with all its fingers pointing to the same direction a piece can be taken from parallel faces, palm direction will be the translation axis of the gripper fingers. Then, the palm normal will be the direction which a gripper approaches a piece. As specified in the gripper axes requisites in section 4.3, in this case, Z-axis is the palm normal $\vec{N}$, Y-axis is the palm direction $\vec{D}$ and the remaining X-axis is the common perpendicular, $\vec{D} \times \vec{N}$ fulfilling the dextro-rotatory system criteria.

- In aligned state:
  - **Translation is allowed:** In case the translation is allowed, we calculate the target frame translation as the nearest point to the hand of the Z-axis of the piece which the robot is aligned with. So it becomes a "closest distance point to line" problem, in this case, we calculate the point of the line which makes the distance lower. Being $\vec{V}$ the translation vector of the hand, $\vec{P}$ the translation vector of the piece to align with and $\vec{R_z}$ its Z-axis direction, we calculate the closest point to the line translation $\vec{T}$ as in equation 4.2.
  - **Translation is not allowed:** In this case, it does not mean that the robot won't move, actually, the robot will be positioned aligned to the piece but the hand translation will be ignored. Equation 4.2 can also be applied but changing the hand translation for the end-effector translation.
  - **Rotation is allowed:** In case we allow rotation we will use the hand rotation instead of the one of the target piece. Roll, yaw, and pitch can be independently constrained or not. For the calculation, we take the Euler XYZ angles of the hand rotation as well as those from the piece. Roll corresponds to the Z-axis of the tool, yaw with Y and pitch with X. In case the rotation is constrained, the target frame corresponding Euler angle will be fixed to the piece's one, if not, it will imitate the hand's one. These Euler XYZ angles are directly extracted using the RS API.
  - **Rotation is not allowed:** If all rotations are constrained we will directly copy the piece rotation to the target frame, this would be the best option to pick or place a piece.
- In not aligned state:
  - **Translation is allowed:** Translation of the target frame equals the translation of the hand.
  - **Translation is not allowed:** Translation of the target frame equals the current translation of the end-effector.

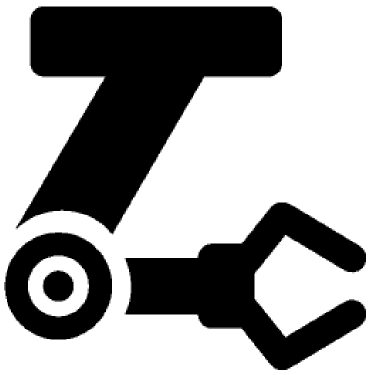


(a) Visualization of a destination with an opened hand.

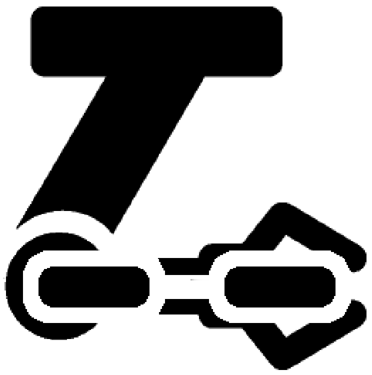


(b) Robot moved to a destination with closed hand, the robot moves as long as the hand is closed.

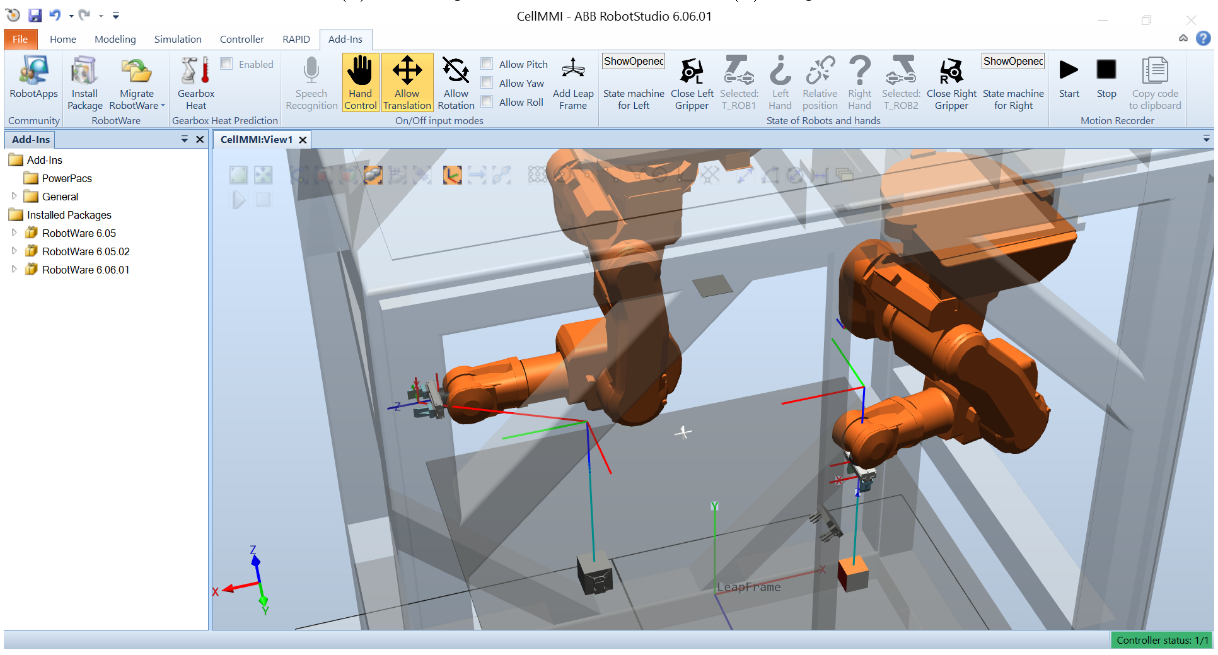
Figure 4.14: Visualization example.



(a) Not aligned.



(b) Aligned.



(c) Visualization of alignment.

Figure 4.15: Alignment state feedback. The icon will change whenever the target frames are aligned to a piece. A dark cyan line is added to the aligned piece.

  - **Rotation is allowed:** As explained in the same case for the aligned state, if unconstrained, Euler angles of hand rotation will be applied to the target frame.
  - **Rotation is not allowed:** Otherwise, the current angle of the end-effector will remain in the target frame.

Finally the target frame T, which is defined in world coordinates, needs to be referred to the active workobject of the RS task, being its frame W_{obj}, the conversion is done as in equation 4.3.

$$\vec{T} = (\vec{V} - \vec{P}) - ((\vec{V} - \vec{P}) \cdot \vec{R_z}) \circ \vec{R_z}. \tag{4.2}$$

$$T_{W obj} = {W_{obj}}^{-1} T. \tag{4.3}$$

Now that the target frame has been calculated, the reachability of the position needs to be checked. Because of this, the RS SDK API provides methods for calculating it on an asynchronous and a fast way. In order for these methods to work, an RS target needs to be built and added to the task. This RS target requires the creation of an RS RobTarget, containing information of the transform. Later, it is required to add the RS RobTarget the task data declarations. Finally, an RS Target is created combining the RobTarget and the active WorkObject of the RS task, this target needs to be added to the task.

When a target is added to a task in RS it will appear in the "Path&Targets" tab and in the RS GUI, with the API we can specify the target to be invisible and do not interfere with other visualizations.

The before-mentioned target procedure is necessary to do not make the reachability checking and movement functions of the RS API to fail internally. This issue is not specified in the ABB official API documentation.

Another procedure consisting on calculating the inverse kinematics of the task mechanism has been tested before solving the before-mentioned problem, however, this method did not take in consideration all the possible configurations that can be used to reach a point, highly limiting the reachability and resulting in a cumbersome procedure to move the arms.

To check reachability and to command the movement, which is instantly performed as a "jump to pose" operation, the API uses asynchronous methods which allow the GUI to be refreshed in parallel.

After the movement is performed, the previously added target is erased from the task and the task data declarations, how targets are recorded will be explained later.

The actions to be done with this target are the next ones:

- Target position is not reachable: A red line between the end-effector and the target frame is displayed.
- Target position is reachable:
  - **State is opened hand:** A green line between the end-effector and the target frame is displayed.
  - **State is closed hand:** Robot will move its end-effector to the target frame.

Additionally, all the performed movements can be undone and redone using the RS GUI, it must be specified in the code of the Add-In what changes in the station are being considered in an undo step.

4.6 Speech Recognition

Speech Recognition has been added to the Add-In as another input mode but this is not the most focused of the modes. Although speech recognition is a powerful tool that allows implementing very high-level commands, in our case it is mainly used to allow the user to virtually press the buttons in the interface or execute an alignment to a piece.

The Microsoft Speech Platform SDK 11 has been used to implement the recognition in this thesis. The en-US speech recognizer must be installed in Windows in order to be able to use this feature, it can be easily downloaded from the Windows configuration.

All the speech commands that can be given are shown below:

- Allow translation.
- Allow rotation.
- Allow roll.
- Allow pitch.
- Allow yaw.
- Open left gripper.
- Open right gripper.
- Close left gripper.
- Close right gripper.
- Align left to *name of piece*.
- Align right to *name of piece*.
- Misalign.

From the speech commands mentioned above, most of them just make the same action as pressing an interface button, for the ones which press the gripper-related buttons, the corresponding action (Open/Close) desired must be mentioned.

Finally, the alignment-related commands make the alignment as if it would be done by the corresponding hand gesture but choosing which piece to align with instead of the closest piece. If the piece name is somehow readable, it can be recognized.

In order to build speech commands, some grammars must be built for the recognizer, appending words or choices of words. An example of choice is all the different commands that have the "Allow" word prepended. The structure of the speech recognizer is depicted in Figure 4.16.

In this implementation case, a unique speech recognizer with a set of grammars is initialized at once, the speech recognition can also be a process where a recognizer with certain grammars activate others after recognizing certain commands, this way, a more than one step recognition could have been implemented.

To conclude about speech recognition it is important to highlight that the used language is not the same as the author's native one, it has been checked that the pronunciation of the words drastically affects the success of a recognition. Moreover, the speech recognizer can be trained using the Windows speech recognition voice training.

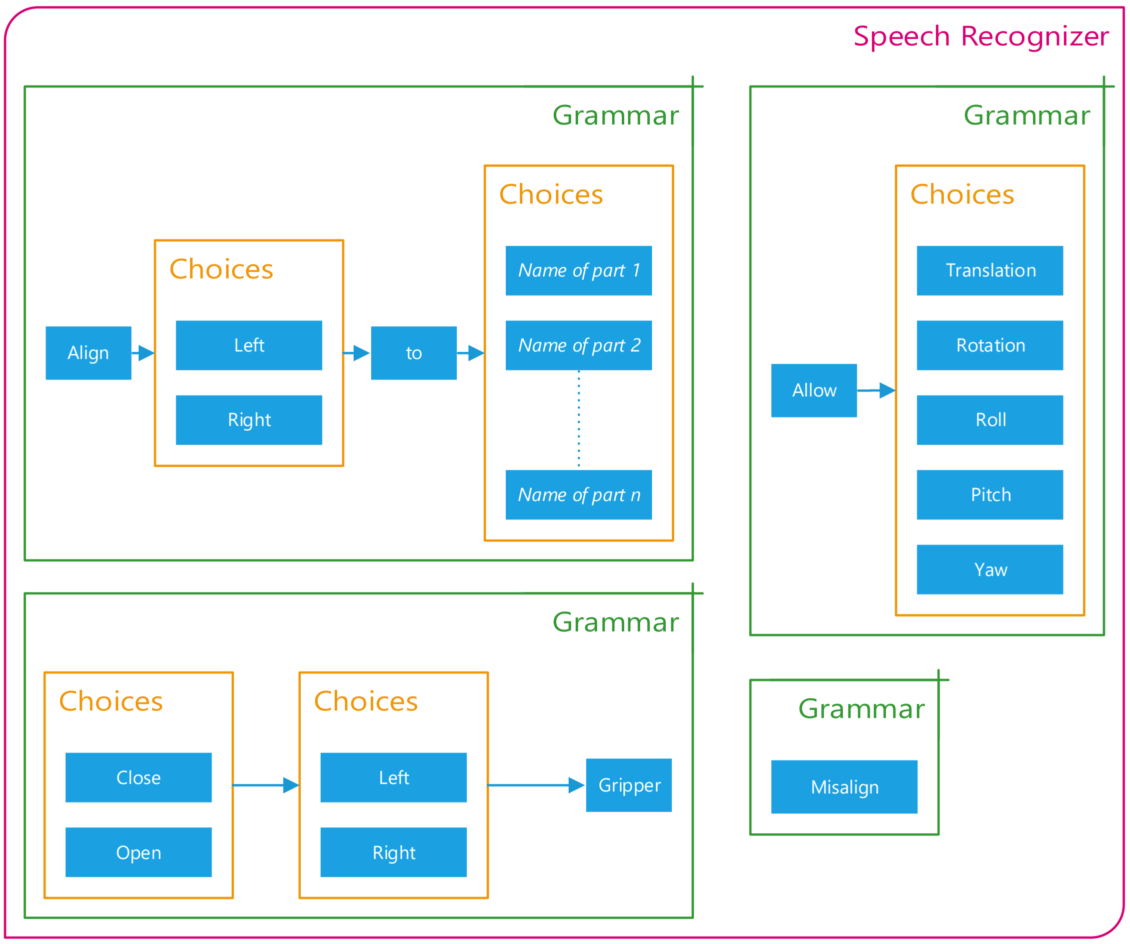


Figure 4.16: Speech Recognizer structure.

4.7 Generation of instructions

At this point it has been explained how the robots are moved using the Leap Motion Sensor and the RS tasks defined in the station, in order for these actions to aid programming, events will be activated to request recording orders from the state machine and the speech recognizer as explained before.

The motion recorder object developed for generating RS instructions will not only record *Move* **instructions** given their **targets**, it will use the alignment information when a target is requested to be recorded to find out if the path to be done needs to follow a **joint or linear** trajectory. It also will record instructions for activating **outputs** for controlling both grippers of the robots and figure out when the movements of the manipulators need **synchronization** points.

The motion recorder will store the reference to both left and right robot tasks in order to be able to create synchronization instructions, a boolean flag determines if it is recording or not.

Starting the recorder:

When the motion recorder is started using the GUI, it adds two path procedures to each RS tasks, these are named with a character for each task, L for left, and R for right, and a time-stamp giving the day of the year (from 1 to 366), the local hour, minute and second. For example, if start recording is pressed on 3rd of February at 13:22:08, the path procedure for the left task will be named like *PathL_34_132208*.

Additionally, the *CheckGrippers* signals will be pulsed to refresh the actual state of the grippers, just in case if some changes have been done in simulation.

Recording actions:

The recorder provides methods for **adding targets and gripper actions**. In the case of gripper actions, we consider grippers that can be commanded to be opened or closed but it is not possible to know when these actions are already performed, therefore we consider waiting one second after each action is done.

As before-mentioned, when a closed hand moving a robot is opened it will launch an **event**, this event is used to record a target. To avoid recording a movement, the user can either press undo or take the hand out of sight of the sensor.

When the recorder is told to **add a gripper order** (close or open), it generates three RS instructions, first a *SetDO* followed by a *SetDO* and another *WaitTime* instruction. The first one is used to wait until the robot has completed its previous movement, otherwise, the signal will be executed when the robot is not yet positioned. The second one changes the corresponding output signal to command the gripper, in the implementation it uses the name of our signals, these cannot be configured before-hand. Finally, the third one gives time for the grippers to open and close since no real sensor is used to tell us when they finish to open or close. More configuration options for specifying the name of the Virtual controller signal to be changed could be added in minor changes of future works. The generated instructions are added to their corresponding path procedure.

Gripper orders are recorded clicking in the Add-In GUI to open/close them, or via speech, the reasons why it was **not implemented by making a gesture** with the hand is that the Leap Motion Sensor lacks confidence on this kind of detections, mainly because of it only tracks the hands from a single point of view.

When the recorder is told to **add a target order**, the alignment parameter will also be needed, this one tells if the target was ordered to be recorded while the manipulator was aligned to a piece or not. Alignment information is added as an attribute to the target, an attribute consists of key-string and value.

Knowing that this Add-In is oriented to assembly and manipulation, some rules can be abstracted to automatically determine when to synchronize the movement of the robots or not, as well as to follow a linear trajectory or not.

The **steps for generating a new Move instruction** given the target to be recorded are detailed as follows, additionally, Figure 4.17 complements the explanation with other details:

1. **Add the target of the current task pose to the task**, the same naming procedure followed for paths will be applied for targets, but adding an extra number in case more than one is recorded in a second.

2. **Determine whether the *Move* instruction needs to be done as joint or linear:**

   The Add-In does not use any path planner or collision avoidance algorithm, instead, when programming a task, the user must considerate what **keyframes** will be needed. In the case of engaging a piece with a gripper, it must first be set over the piece, then approached and closed to take the piece.

   The **alignment** feature helps the user to locate the end-effectors in suitable positions, then a target recorded with alignment which its previous one was also recorded with alignment will need to be reached making a linear trajectory. In addition to the previous condition, when two consecutive aligned targets have different piece targets, the linear trajectory will not be needed, therefore a joint trajectory will be performed in that case.

3. **Determine when a *Move* instruction needs to be preceded by a synchronization point:**

   As it has been explained until now, the Add-In is meant to program **collaborative manipulators**, up to two of them can be programmed, but their movements would need some kind of synchronization. Both arms will need to be synchronized when the pieces they are manipulating are going to be assembled or they both manipulate the same piece.

   In order to establish when this **synchronization** is needed, one of the conditions needed is that the movement to be done is linear, linear movements are needed to approach a piece, as explained before, or taking it from where it was deposited. When the manipulator is aligned to a piece held by the other manipulator, before that movement, a synchronization point has to be inserted in both tasks procedures to ensure that the recorded movements of the other manipulator are already done.

   The first synchronization to start manipulation of a piece, or the first assembly operation between two pieces is met. But more synchronization points are needed during manipulation or finishing it.

   After the first synchronization point, each time a different manipulator starts performing an action a synchronization point must be added. Once none of the manipulators are aligned to the others piece, the manipulation is finished and no more synchronization points are added.

4. **Create** the synchronization **instructions** for both procedures if necessary and the move instruction.

If there are errors during recording, **the last instructions can be undone**, this undoes the instructions in the task and therefore in the RS GUI, thanks to embedding the alignment information into the target, this information is also handled. The recorder does not store extra information that cannot be located somehow in the RS GUI.

The motion recorder is **not implemented to allow redo** operations while recording, besides, using both hands at a time is not recommended because although seeming more intuitive, precision alignments make it more cumbersome to try to command both manipulators at the same time.

Stopping the motion recorder:

Once the recording finishes, the recorder can be stopped using the GUI, each instruction generated is automatically stored in the corresponding task procedure and can be seen with the GUI. If no instructions have been added to a path procedure it is deleted.

Additionally, data declarations of variables and persistent data needed for the synchronization are added to both tasks. For this, a task list with both tasks is added to each task. Iterating through all the synchronization instructions added creates synchronization variables. After synchronizing, these data declarations are automatically added to the RAPID code as follows:

```
PERS tasks task_list{2}:=[["T_ROB1"],["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
```

After recording a program:

The programmed procedure can be tested using the RS GUI and synchronized to RAPID as classically done in RS. If there are instructions not generated as expected, or targets that would require modification, they can be easily tuned using the RS GUI until the procedure behaves as expected. Moreover, the configurations of the target can be optimized using the auto-configuration wizard.

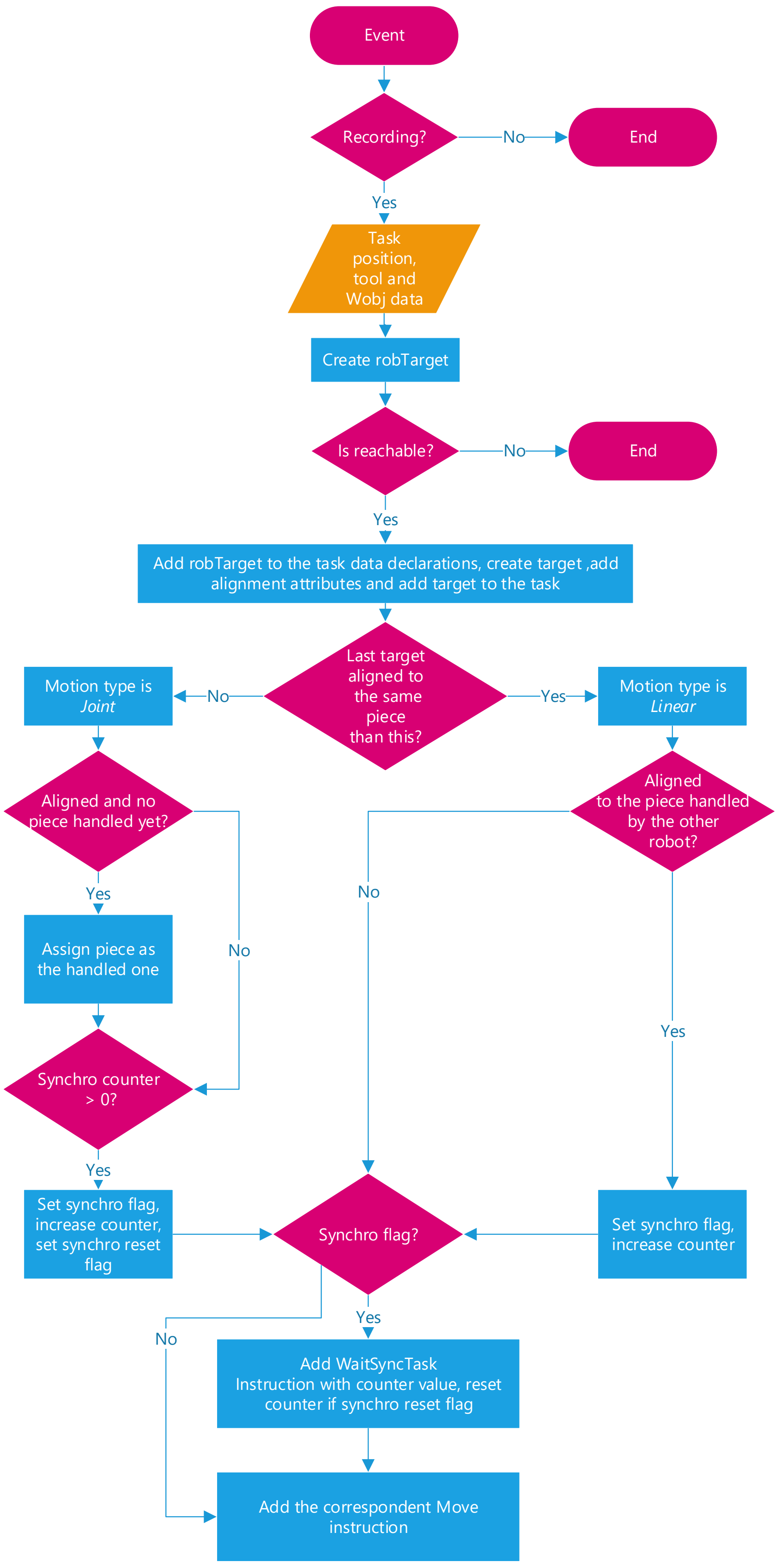

Event
Recording?
No
End
Yes
Task position, tool and Wobj data
Create robTarget
Is reachable?
No
End
Yes
Add robTarget to the task data declarations, create target ,add alignment attributes and add target to the task
Last target aligned to the same piece than this?
No
Motion type is *Joint*
Yes
Motion type is *Linear*
Aligned and no piece handled yet?
Yes
No
Assign piece as the handled one
Synchro counter > 0?
Yes
Set synchro flag, increase counter, set synchro reset flag
Aligned to the piece handled by the other robot?
No
Yes
Set synchro flag, increase counter
Synchro flag?
Yes
No
Add WaitSyncTask Instruction with counter value, reset counter if synchro reset flag
Add the correspondent Move instruction

Figure 4.17: Flowchart of the target recording. Flags are reset on every order.

Chapter 5

Results and Analysis

In order to be able to program the procedures exposed in section 1.6, the trial and error methodology has been followed until the programming of all the task could have been achieved. Some videos have been recorded to demonstrate how the Add-In works.

The results obtained with this implementation are qualitatively described since the result is the implementation itself and how it works, no survey has been done to test the interface, given that the complexity of this work it is still difficult to use.

Regarding its utility for jogging robots, the interface works as it could be expected, the visualization fluently follows the position given by the user unless the CPU is overloaded with other processes such as screen video recording.

In the case of the snap-fit insertion task, no singular problem can be found.

In the case of the bayonet mount, making an alignment to the piece holded by the other robot can make a "rotation problem" after assembling, meaning that if the manipulator is aligned to the piece which is attached to the piece holded by the robot, it will change its position each time the robot tries to align to it. This singularity happens because the manipulator is moving the piece it wants to align with in every movement, anyway it does not mean a big problem while programming.

In the case of the Rubik cube manipulation, since it is simulated as a single piece, the programming processed resulted significantly non-intuitive, and the visual feedback confusing. It would be necessary to model the Rubik cube as a mechanism to be able to program its manipulation in a good way, this last step has not been performed and the implementation does not regard handling pieces behaving as mechanisms.

It has been checked that it is better to continue programming if a mistake has been done in the process instead of undoing the last actions, the undo command is implemented in a way that the synchronization correctness can be lost.

5.1 Review of limitations

In this section, the limitations found during the implementation development are explained, limitations have affected both easiness of the implementation development as well as the quality of the implementation.

Regarding the use of the Leap Motion Sensor, its limitations mainly affect the implementation quality:

- Occasionally, a left hand may be confused with a right hand.
- The rotation of the palm becomes less precise as the palm normal is not pointing to the camera of the sensor.
- When both hands are being used at the same time, keeping them close to each other may cause the detection to fail completely.
- When the palm looks upwards, the sensor is more likely to understand that it is a palm facing downwards instead.
- Discerning rotation becomes more difficult while the hand is closed.
- Gesture recognition is not available in the last release of the Software Development Kit (SDK)

Limitations found in RS and RS SDK made more difficult the development of the interface:

- Examples for using the RS SDK are limited to simple tasks.
- While using the Add-In tab, the *Path&Targets* menu is not visible.
- Bad use of the SDK may trigger internal errors on the API whose error text messages do not give a hint of the real problem.
- Assistance found in ABB forums is limited, and is not as wide as, for example, .NET one.
- The way that the attachment information is stored on RS (in the station instead of in each object) makes difficult to discern what object is holding each robot, making the programming flexibility of the Add-In more limited.

Final limitations of the interface created:

- The interface only allows to align the robots to a single frame per piece, in other words, it is not possible to align the position of the robot to different places of a big piece.
- The points to be reached by the robots have to be teached one by one, there is no artificial intelligence generating targets automatically, the targets will be recorded when opening the hands instead.
- This last limitation makes the programming much less intuitive.
- Configuration and use of an appropriate smart component are necessary to use all the functionalities.

Chapter 6

Conclusions

The proposal of this thesis has been proved to be both limitated and aided by the capabilities of the off-line programming environment it is added in, in this case, on the one hand, RobotStudio aids the implementation by generating instructions instead of directly code, errors done during the programming process can be easily edited and visualized. On the other hand, the modeling capabilities of RobotStudio make difficult to make an implementation based on the information that can be extracted from a virtual piece.

The developed implementation can either be used for jogging or automatic code generation. Even if the automatic generation of commands does not result as expected it is easier to edit instructions than generating them from scratch.

6.1 Open issues

The complexity of this work has meant to finish with some open issues not solved yet.

In order to program an assembly operation, the programming procedure needs to be done at once, if not, the synchronization instructions won't be automatically generated.

The speech recognition has been implemented in English, instead of the author native language Spanish, it has been checked that a correct pronunciation while commanding is essential to obtain the expected behavior. There is a significant presence of false positives, false negatives and mistakes between commands.

The speech recognition can require opening the program as administrator depending on the permissions used when installing it.

Detection of the handled piece for synchronization is made based on the first target with alignment recorded instead of the actual piece attached to the robot, meaning that no mistaken targets can be recorded before aligning to the piece that is going to be manipulated.

After a while using the hand control, the Add-In starts becoming slower.

Data declarations regarding synchronization cannot be visualized in the RS tasks.

6.2 Future Works

The implementation of this Add-In can serve as a base for future works.

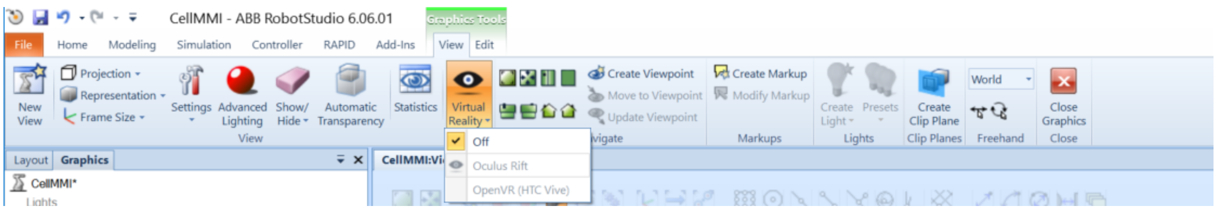


Figure 6.1: RobotStudio compatibility with VR headsets.

Improvement of the filter:

The implemented low-pass filter is quite simple, some other methods like an extended Kalman filter, or other ways to discern the hand pose instead of relying in the Leap Motion API high-level functions can be implemented to make the rest of the Add-In more precise and powerful. Moreover, some kind of logic filtering like a delay of the input data regarding position but no delay on pose could maintain a more stable position when opening the hand for recording.

Improvement by adding a complex algorithm or artificial intelligence:

The target of Multi-modal interfaces is making interaction more intuitive, in this case, programming. Our implementation has been limited to teaching key-frames or key-targets and generating instructions with them automatically.

A goal to accomplish in the future would be to automatically generate this targets along with the instructions by showing the software a set of positions, commands, movements of the hands or any other input mode and use all the possible information of the virtual station to generate the teached program in an optimal way, and using the aperture of the hands as a closing/opening input command, not needing any constraint.

For this work, the motion recorder should be re-implemented and the state machines would not be necessary.

Giving a "programming aid" focus:

The development of this interface was mainly focused on automating instructions generation but multi-modes can also offer assistance to make programming easier without directly programming. More functions could be aided by using gestures as inputs to execute commands easing traditional programming in RS.

Implementation of more I/O modes:

Of the many possible ways of interaction, one of the most popular could be the virtual or augmented reality. These devices commonly include controllers which can be located in the 3D space with high precision. This controllers have many buttons that can be used to implement a set of commands, like the alignment used in this thesis implementation and activation or deactivation of tools. These devices also can visualize a virtual environment surrounding the user. This mode could be implemented in RS, it has compatibility with VR devices as the Oculus Rift and HTC Vive headsets as can be seen in Figure 6.1.

Bibliography

[1] F. Ficuciello, A. Romano, V. Lippiello, L. Villani, and B. Siciliano, *Human Motion Mapping to a Robot Arm with Redundancy Resolution.* Cham: Springer International Publishing, 2014, pp. 193–201. [Online]. Available: https://doi.org/10.1007/978-3-319-06698-1_21

[2] G. Du, Y. Lei, H. Shao, Z. Xie, and P. Zhang, "A human–robot interface using particle filter, kalman filter, and over-damping method," *Intelligent Service Robotics*, vol. 9, no. 4, pp. 323–332, Oct 2016. [Online]. Available: https://doi.org/10.1007/s11370-016-0202-9

[3] C. Papadopoulos, I. Mariolis, A. Topalidou-Kyniazopoulou, G. Piperagkas, D. Ioannidis, and D. Tzovaras, *An Advanced Human-Robot Interaction Interface for Teaching Collaborative Robots New Assembly Tasks.* Cham: Springer International Publishing, 2017, pp. 180–190. [Online]. Available: https://doi.org/10.1007/978-3-319-66471-2_20

[4] T. Brogårdh, "Present and future robot control development?an industrial perspective," *Annual Reviews in Control*, vol. 31, no. 1, pp. 69 – 79, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1367578807000077

[5] C. Wittenberg, "Cause the trend industry 4.0 in the automated industry to new requirements on user interfaces?" in *Human-Computer Interaction: Users and Contexts*, M. Kurosu, Ed. Cham: Springer International Publishing, 2015, pp. 238–245.

[6] *Interface Devices and Systems.* Boston, MA: Springer US, 2007, pp. 173–223. [Online]. Available: https://doi.org/10.1007/978-0-387-23326-0_4

[7] K. Ishii, S. Zhao, M. Inami, T. Igarashi, and M. Imai, *Designing Laser Gesture Interface for Robot Control.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 479–492. [Online]. Available: https://doi.org/10.1007/978-3-642-03658-3_52

[8] T. Kollar, A. Vedantham, C. Sobel, C. Chang, V. Perera, and M. Veloso, *A Multi-modal Approach for Natural Human-Robot Interaction.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 458–467. [Online]. Available: https://doi.org/10.1007/978-3-642-34103-8_46

[9] J. G. Victores, F. R. Cañadillas, S. Morante, A. Jardón, and C. Balaguer, *Assistive Robot Multi-modal Interaction with Augmented 3D Vision and Dialogue.* Cham: Springer International Publishing, 2014, pp. 209–217. [Online]. Available: https://doi.org/10.1007/978-3-319-03413-3_15

[10] N. Rudigkeit, M. Gebhard, and A. Gräser, *A Novel Interface for Intuitive Control of Assistive Robots Based on Inertial Measurement Units.* Cham: Springer International Publishing, 2016, pp. 137–146. [Online]. Available: https://doi.org/10.1007/978-3-319-26345-8_12

[11] E. Tamura, Y. Yamashita, Y. Ho, E. Sato-Shimokawara, and T. Yamaguchi, *Robot Control Interface System Using Glasses-Type Wearable Devices.* Cham: Springer International Publishing, 2016, pp. 247–256. [Online]. Available: https://doi.org/10.1007/978-3-319-43518-3_24

[12] Y.-H. Su, C.-C. Hsiao, and K.-Y. Young, *Manipulation System Design for Industrial Robot Manipulators Based on Tablet PC*. Cham: Springer International Publishing, 2015, pp. 27–36. [Online]. Available: https://doi.org/10.1007/978-3-319-22876-1_3

[13] B. Mocan, M. Fulea, and S. Brad, *Designing a Multimodal Human-Robot Interaction Interface for an Industrial Robot*. Cham: Springer International Publishing, 2016, pp. 255–263. [Online]. Available: https://doi.org/10.1007/978-3-319-21290-6_26

[14] M. R. Pedersen and V. Krüger, "Gesture-based extraction of robot skill parameters for intuitive robot programming," *Journal of Intelligent & Robotic Systems*, vol. 80, no. 1, pp. 149–163, Dec 2015. [Online]. Available: https://doi.org/10.1007/s10846-015-0219-x

[15] C. Pérez-D'Arpino and J. A. Shah, "C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 4058–4065.

[16] H. C. Fang, S. K. Ong, and A. Y. C. Nee, "A novel augmented reality-based interface for robot path planning," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 8, no. 1, pp. 33–42, Feb 2014. [Online]. Available: https://doi.org/10.1007/s12008-013-0191-2

[17] H. S. Park, E. Y. Kim, and H. J. Kim, *Robot Competition Using Gesture Based Interface*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 131–133. [Online]. Available: https://doi.org/10.1007/11504894_20

[18] H.-G. Lee, Y.-G. Kim, H.-D. Lee, J.-H. Kim, and G.-T. Park, *Human Interface for the Robot Control in Networked and Multi-sensored Environment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 928–935. [Online]. Available: https://doi.org/10.1007/978-3-540-73281-5_101

[19] E. Iáñez, M. C. Furió, J. M. Azorín, J. A. Huizzi, and E. Fernández, *Brain-Robot Interface for Controlling a Remote Robot Arm*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 353–361. [Online]. Available: https://doi.org/10.1007/978-3-642-02267-8_38

[20] P. González, A. Brumovsky, and M. Anigstein. (2015, April) Implementación de un control gestual para robots. [Online]. Available: http://www.secyt.frba.utn.edu.ar/gia/trabajosviiijar/jar8_submission_12.pdf

[21] P. D. E. Prassler, D. A. Stopp, M. Hägele, I. Iossifidis, D. G. Lawitzky, D. G. Grunwald, and P. D.-I. R. Dillmann, *1 Multi-modal Robot Interfaces*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 5–7. [Online]. Available: https://doi.org/10.1007/978-3-540-31509-4_1

[22] D. Walter. (2014, April) Product comparison ? kinect and leap motion. [Online]. Available: http://ashlandtech.org/2014/04/02/product-comparison-kinect-and-leap-motion/

Appendix A

UML Class Diagram

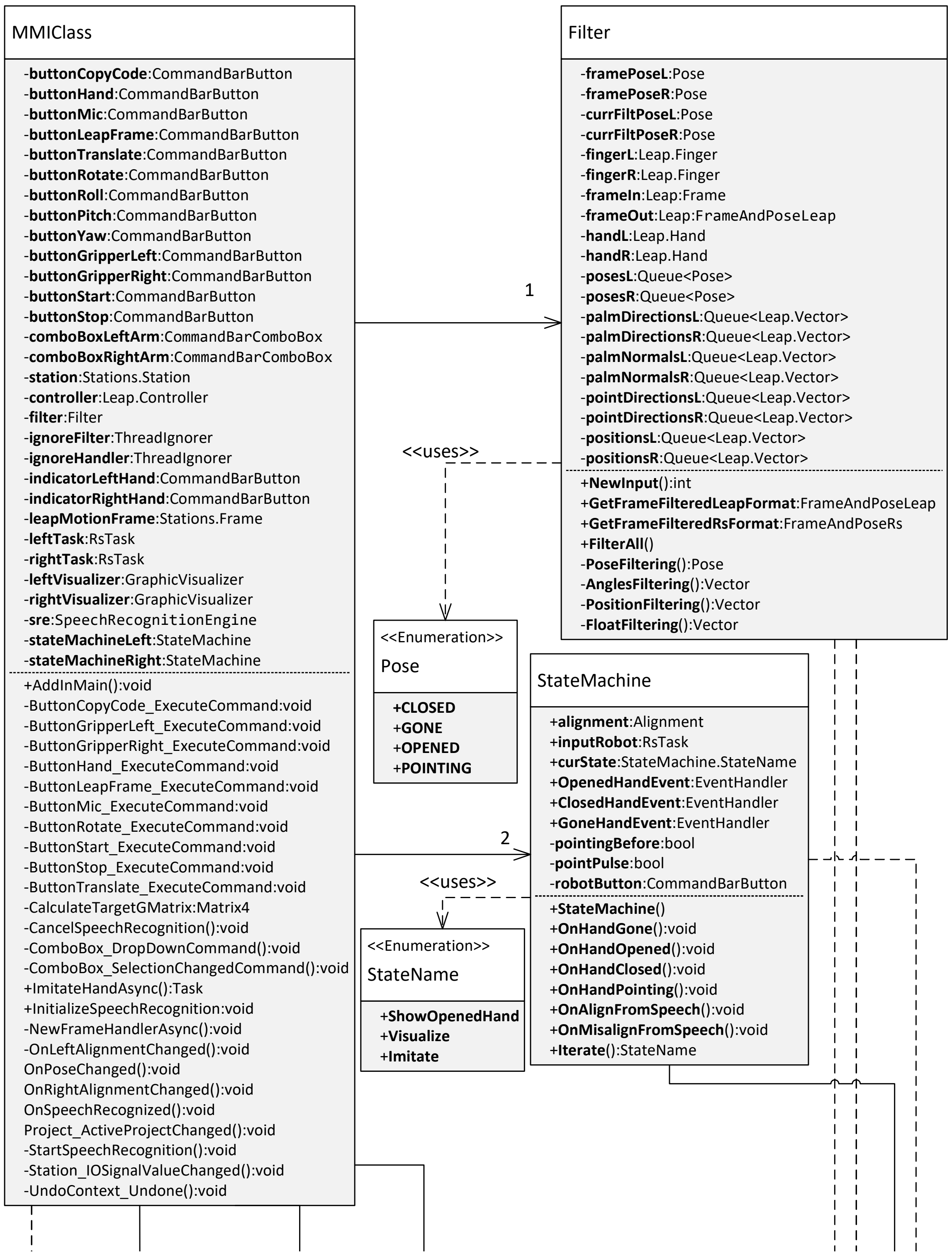

MMIClass
-buttonCopyCode:CommandBarButton
-buttonHand:CommandBarButton
-buttonMic:CommandBarButton
-buttonLeapFrame:CommandBarButton
-buttonTranslate:CommandBarButton
-buttonRotate:CommandBarButton
-buttonRoll:CommandBarButton
-buttonPitch:CommandBarButton
-buttonYaw:CommandBarButton
-buttonGripperLeft:CommandBarButton
-buttonGripperRight:CommandBarButton
-buttonStart:CommandBarButton
-buttonStop:CommandBarButton
-comboBoxLeftArm:CommandBarComboBox
-comboBoxRightArm:CommandBarComboBox
-station:Stations.Station
-controller:Leap.Controller
-filter:Filter
-ignoreFilter:ThreadIgnorer
-ignoreHandler:ThreadIgnorer
-indicatorLeftHand:CommandBarButton
-indicatorRightHand:CommandBarButton
-leapMotionFrame:Stations.Frame
-leftTask:RsTask
-rightTask:RsTask
-leftVisualizer:GraphicVisualizer
-rightVisualizer:GraphicVisualizer
-sre:SpeechRecognitionEngine
-stateMachineLeft:StateMachine
-stateMachineRight:StateMachine
+AddInMain():void
-ButtonCopyCode_ExecuteCommand:void
-ButtonGripperLeft_ExecuteCommand:void
-ButtonGripperRight_ExecuteCommand:void
-ButtonHand_ExecuteCommand:void
-ButtonLeapFrame_ExecuteCommand:void
-ButtonMic_ExecuteCommand:void
-ButtonRotate_ExecuteCommand:void
-ButtonStart_ExecuteCommand:void
-ButtonStop_ExecuteCommand:void
-ButtonTranslate_ExecuteCommand:void
-CalculateTargetGMatrix:Matrix4
-CancelSpeechRecognition():void
-ComboBox_DropDownCommand():void
-ComboBox_SelectionChangedCommand():void
+ImitateHandAsync():Task
+InitializeSpeechRecognition:void
-NewFrameHandlerAsync():void
-OnLeftAlignmentChanged():void
OnPoseChanged():void
OnRightAlignmentChanged():void
OnSpeechRecognized():void
Project_ActiveProjectChanged():void
-StartSpeechRecognition():void
-Station_IOSignalValueChanged():void
-UndoContext_Undone():void
1
Filter
-framePoseL:Pose
-framePoseR:Pose
-currFiltPoseL:Pose
-currFiltPoseR:Pose
-fingerL:Leap.Finger
-fingerR:Leap.Finger
-frameIn:Leap:Frame
-frameOut:Leap:FrameAndPoseLeap
-handL:Leap.Hand
-handR:Leap.Hand
-posesL:Queue<Pose>
-posesR:Queue<Pose>
-palmDirectionsL:Queue<Leap.Vector>
-palmDirectionsR:Queue<Leap.Vector>
-palmNormalsL:Queue<Leap.Vector>
-palmNormalsR:Queue<Leap.Vector>
-pointDirectionsL:Queue<Leap.Vector>
-pointDirectionsR:Queue<Leap.Vector>
-positionsL:Queue<Leap.Vector>
-positionsR:Queue<Leap.Vector>
+NewInput():int
+GetFrameFilteredLeapFormat:FrameAndPoseLeap
+GetFrameFilteredRsFormat:FrameAndPoseRs
+FilterAll()
-PoseFiltering():Pose
-AnglesFiltering():Vector
-PositionFiltering():Vector
-FloatFiltering():Vector
<<uses>>
<<Enumeration>>
Pose
+CLOSED
+GONE
+OPENED
+POINTING
2
<<uses>>
StateMachine
+alignment:Alignment
+inputRobot:RsTask
+curState:StateMachine.StateName
+OpenedHandEvent:EventHandler
+ClosedHandEvent:EventHandler
+GoneHandEvent:EventHandler
-pointingBefore:bool
-pointPulse:bool
-robotButton:CommandBarButton
+StateMachine()
+OnHandGone():void
+OnHandOpened():void
+OnHandClosed():void
+OnHandPointing():void
+OnAlignFromSpeech():void
+OnMisalignFromSpeech():void
+Iterate():StateName
<<Enumeration>>
StateName
+ShowOpenedHand
+Visualize
+Imitate

Figure A.1: UML class diagram. Classes of the RS, Leap Motion and .NET APIs have been omitted, as well as some unused methods or fields.

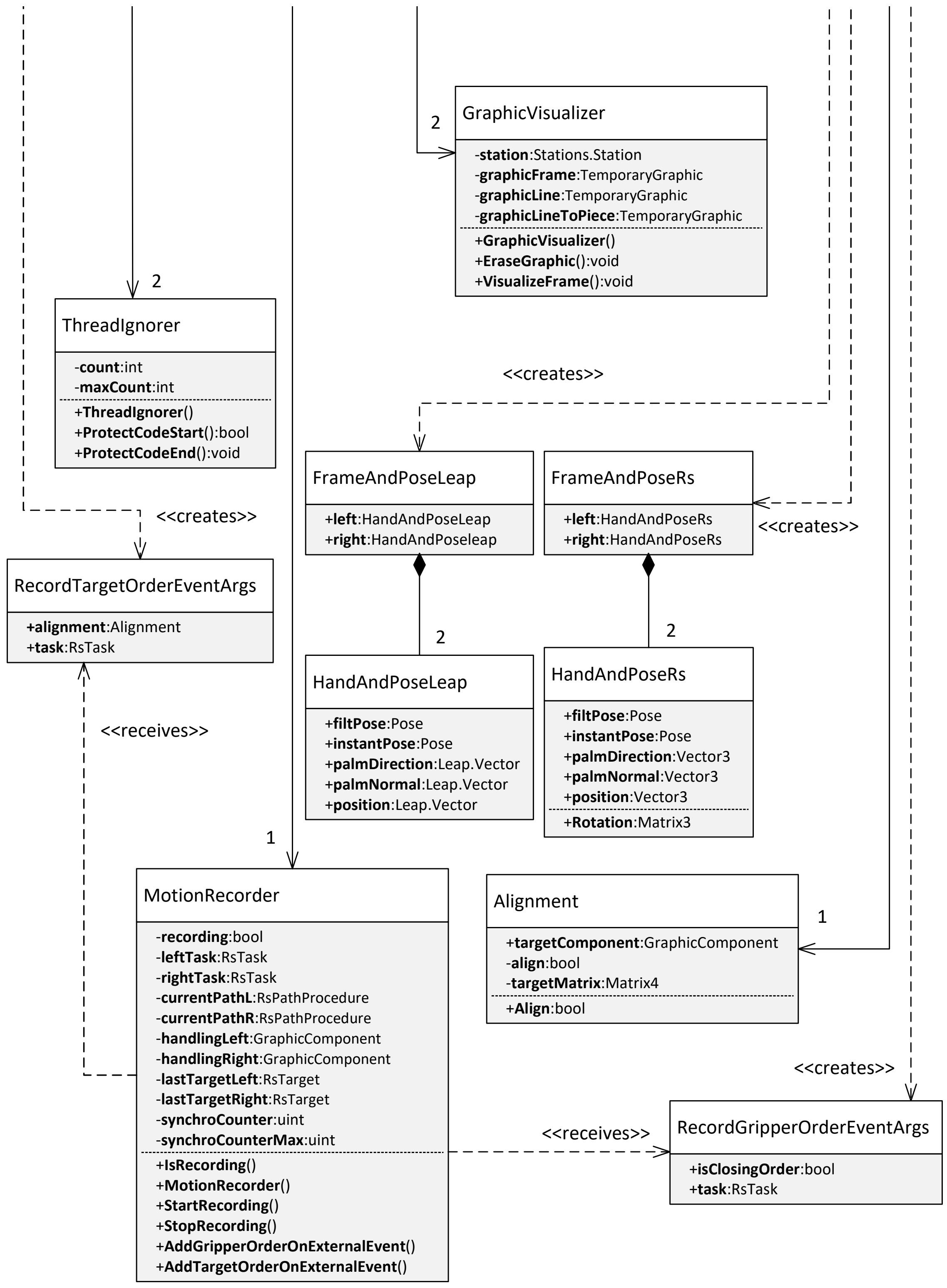


Figure A.2: UML class diagram, lower part. Classes of the RS, Leap Motion and .NET APIs have been omitted, as well as some unused methods or fields.

Appendix B

XML Code Documentation

This code XML documentation can be visualized in XML visualizers, it includes all the code that was left in the final implementation including unused fields or methods.

Listing B.1: Code XML documentation

```xml
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>MultiModalInterface</name>
    </assembly>
    <members>
        <member name="T:MMI.MMIClass">
            <summary>
            Main class of the Add-In
            </summary>
        </member>
        <member name="F:MMI.MMIClass.sre">
            <summary>
            Speech recognizer for the Add-In speech commands
            </summary>
        </member>
        <member name="F:MMI.MMIClass.sreYesNo">
            <summary>
            Speech recognizer for Yes/No acknowledge commands
            </summary>
        </member>
        <member name="M:MMI.MMIClass.InitializeSpeechRecognition">
            <summary>
            Initializes the speech recognition, configuring the recognizers and the
                grammars for commands. But does not start the recognition
            </summary>
        </member>
        <member name="M:MMI.MMIClass.StartSpeechRecognition">
            <summary>
            Starts the recognizer so the user can start speaking to the computer
            </summary>
        </member>
        <member name="M:MMI.MMIClass.OnSpeechRecognized(System.Object,System.Speech.
            Recognition.SpeechRecognizedEventArgs)">
            <summary>
            The event handler for general recognition
            </summary>
            <param name="sender"> Not used </param>
            <param name="e"> Contains information about the recognized speech </param
                >
        </member>
        <member name="M:MMI.MMIClass.OnSpeechHypothesized(System.Object,System.Speech
            .Recognition.SpeechHypothesizedEventArgs)">
            <summary>
            The event handler for debugging recognized speech hypotheses
            </summary>
            <param name="sender"> Not used </param>
            <param name="e"> Contains information about the hypothesized speech </
                param>
        </member>
        <member name="M:MMI.MMIClass.OnYesNoRecognized(System.Object,System.Speech.
            Recognition.SpeechRecognizedEventArgs)">
            <summary>
            The event handler for yes/no recognition
            </summary>
            <param name="sender"> Not used </param>
            <param name="e"> Contains information about the recognized speech </param
                >
        </member>
        <member name="F:MMI.MMIClass.leftVisualizer">
            <summary>
            Object for visualizing the left robot destination target and reachability
                .
            </summary>
        </member>
        <member name="F:MMI.MMIClass.rightVisualizer">
            <summary>
            Object for visualizing the right robot destination target and
                reachability.
            </summary>
        </member>
        <member name="M:MMI.MMIClass.ButtonMic_ExecuteCommand(System.Object,ABB.
            Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the microphone button, starts or cancels the speech
                recognition.
            </summary>
            <param name="sender"> Not used. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonHand_ExecuteCommand(System.Object,ABB.
            Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the Hand control button, initializes everything
                needed for reading hands or prompts if some configuration is still
                missing.
            </summary>
            <param name="sender"> Not used. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonTranslate_ExecuteCommand(System.Object,ABB
            .Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the Allow translation button, allows translation or
                not.
            </summary>
            <param name="sender"> Not used. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonRotate_ExecuteCommand(System.Object,ABB.Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the Allow rotation/pitch/yaw/roll buttons, the rotation is disabled if every subrotation is disabled,
            and it is enabled if any subrotation is enabled.
            </summary>
            <param name="sender"> Used to check what button has been clicked. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonLeapFrame_ExecuteCommand(System.Object,ABB.Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the Add Leap Frame button, if the Leap frame does not exist it creates a new one and prompts how to position it, if it does exists it prompts how to position it.
            </summary>
            <param name="sender"> Not used. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonGripperLeft_ExecuteCommand(System.Object,ABB.Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for the Left gripper button, toggles the CloseLeft gripper signal value.
            </summary>
            <param name="sender"> Just takes the refernce of the button. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonGripperRight_ExecuteCommand(System.Object,ABB.Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for the Right gripper button, toggles the CloseRight gripper station signal value.
            </summary>
            <param name="sender"> Just takes the reference of the button. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonStart_ExecuteCommand(System.Object,ABB.Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Starts the motionRecorder so a new path procedure is created and actions performed with hands or gripper commands will be recorded.
            </summary>
            <param name="sender"> Just takes the reference of the button. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonStop_ExecuteCommand(System.Object,ABB.Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the button stop. Stops the motionRecorder, so stops recording actions and stops using the same path procedure.
            </summary>
            <param name="sender"> Just takes the reference of the button. </param>
            <param name="e"> Not used. </param>
        </member>
        <member name="M:MMI.MMIClass.ButtonCopyCode_ExecuteCommand(System.Object,ABB.
            Robotics.RobotStudio.Environment.ExecuteCommandEventArgs)">
            <summary>
            Handler for clicking the copy code to clipboard button. Not implemented
                yet.
            </summary>
            <param name="sender"></param>
            <param name="e"></param>
        </member>
        <member name="M:MMI.MMIClass.ComboBox_DropDownCommand(System.Object,System.
            EventArgs)">
            <summary>
            Handler for clicking the Select left/right task buttons. Searchs for
                tasks in the station and add them as items of the dropbox.
            </summary>
            <param name="sender"> Just takes the reference of the button.</param>
            <param name="e"> Not used.</param>
        </member>
        <member name="M:MMI.MMIClass.ComboBox_SelectionChangedCommand(System.Object,
            System.EventArgs)">
            <summary>
            Handler for clicking a selection of the combobox for selecting Left/Right
                 task. Changes the caption of the comboBox with the selected task.
            </summary>
            <param name="sender">Takes the reference of the comboBox. Allows to know
                which combobox are we using.</param>
            <param name="e"> Not used.</param>
        </member>
        <member name="F:MMI.MMIClass.station">
            <summary>
            Reference to the active station of RS, that contains everything the user
                can see in the RS GUI (mechanisms, parts, tasks).
            </summary>
        </member>
        <member name="F:MMI.MMIClass.leftTask">
            <summary>
            Reference to the left task for the MM control, to be assigned when
                selected with the Add-In GUI.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.rightTask">
            <summary>
            Reference to the right task for the MM control, to be assigned when
                selected with the Add-In GUI.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.leapMotionFrame">
            <summary>
            Frame of the virtual Leap Motion Sensor to be ubicated in the virtual
                station.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.filter">
            <summary>
            Filter for the leap Motion Sensor.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.stateMachineLeft">
```

```
            <summary>
            State Machine for the left arm.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.stateMachineRight">
            <summary>
            State Machine for the right arm.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.motionRecorder">
            <summary>
            Recorder of commands given by the hand control or some speech commands,
            records gripper actions and targets given the hands positions,
            automatically generating path procedures.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.controller">
            <summary>
            Controller of the Leap Motion Sensor.
            </summary>
        </member>
        <member name="M:MMI.MMIClass.AddinMain">
            <summary>
            Entry point of the MultiModal interface Add-In. The GUI is created as
                well as event handlers are assigned.
            </summary>
        </member>
        <member name="M:MMI.MMIClass.Project_ActiveProjectChanged(System.Object,
            System.EventArgs)">
            <summary>
            Event handler that assigns the station, creates the Add-In I/O and
                assigns some event handlers depending on the current project.
            </summary>
            <param name="sender"> Object sending the event, not used </param>
            <param name="e"> Arguments sent by the object sending the event, not used
                 </param>
        </member>
        <member name="M:MMI.MMIClass.UndoContext_Undone(System.Object,System.
            EventArgs)">
            <summary>
            This delegate is meant to be executed whenever we do or undo, checking
                how the state of the buttons should be corrected
            </summary>
            <param name="sender"> Not used </param>
            <param name="e"> Not used </param>
        </member>
        <member name="M:MMI.MMIClass.Station_IOSignalValueChanged(System.Object,ABB.
            Robotics.RobotStudio.Stations.IOSignalChangedEventArgs)">
            <summary>
            This delegate is meant to be executed whenever a station I/O signal has
                changed its value. Refreshes the state of the gripper buttons
                whenever CheckGrippersSignal changes.
            </summary>
            <param name="sender"></param>
            <param name="e"></param>
        </member>
        <member name="F:MMI.MMIClass.ignoreFilter">
            <summary>
            Object for not executing a piece of code twice and leaving the the method
```

```
                instead of waiting.
            </summary>
        </member>
        <member name="F:MMI.MMIClass.ignoreHandler">
            <summary>
            Object for not executing a piece of code twice and leaving the the method
                instead of waiting.
            </summary>
        </member>
        <member name="M:MMI.MMIClass.OnPoseChanged(System.Object,System.
            ComponentModel.PropertyChangedEventArgs)">
            <summary>
            Handler for property change of the filter, executed whenever a filtered
                hand pose changes
            </summary>
        </member>
        <member name="M:MMI.MMIClass.OnLeftAlignmentChanged(System.Object,System.
            ComponentModel.PropertyChangedEventArgs)">
            <summary>
            Handler for changing the alignment state of left robot. Changes the image
                of the left robot.
            </summary>
            <param name="sender"> A boolean indicating the alignment state.</param>
            <param name="e"> Not used.</param>
        </member>
        <member name="M:MMI.MMIClass.OnRightAlignmentChanged(System.Object,System.
            ComponentModel.PropertyChangedEventArgs)">
            <summary>
            Handler for changing the alignment state of right robot. Changes the
                image of the right robot.
            </summary>
            <param name="sender"> A boolean indicating the alignment state.</param>
            <param name="e"> Not used.</param>
        </member>
        <member name="M:MMI.MMIClass.NewFrameHandlerAsync(System.Object,Leap.
            FrameEventArgs)">
            <summary>
            Handler for actions to be done with a new Leap Motion Frame.
            </summary>
            <param name="sender"></param>
            <param name="e"></param>
        </member>
        <member name="M:MMI.MMIClass.ImitateHandAsync(ABB.Robotics.Math.Matrix4,ABB.
            Robotics.RobotStudio.Stations.RsTask,MMI.GraphicVisualizer,MMI.Alignment,
            System.Boolean)">
            <summary>
            ImitateArmAsync imitates the position of your hands in the simulator,
                allowing movement without grabbing gesture, this method
            cannot be safe for direct control of the robot and requires orienting and
                positioning the leap motion frame.
            Choose either this function or MoveArmAsync instead
            </summary>
            <param name="targetGMatrix"> The destination global matrix. </param>
            <param name="task"> The task of the corresponding robot </param>
            <param name="graphVis"> The corresponding visualizer </param>
            <param name="alignment"> The alignment state and data </param>
            <param name="move"> True if the arm needs to be moved</param>
            <returns></returns>
        </member>
        <member name="M:MMI.MMIClass.CalculateTargetGMatrix(MMI.HandAndPoseRs,ABB.
            Robotics.RobotStudio.Stations.RsTask,MMI.Alignment)">
            <summary>
            CalculateTargetGMatrix returns the matrix that represents the hand
                position seen by the Leap Motion sensor in the world coordinates of
                the RsStation.
            </summary>
            <param name="hand"> The hand to be calculated. </param>
            <param name="task"> The corresponding task. </param>
            <param name="alignment"> The alignment data. </param>
            <returns>A global matrix with the target transform. </returns>
        </member>
        <member name="T:MMI.MotionRecorder">
            <summary>
            This class allows to record the consecutive targets created by the
                MultiModal Interface Add-In
            </summary>
        </member>
        <member name="M:MMI.MotionRecorder.#ctor(ABB.Robotics.RobotStudio.Stations.
            RsTask,ABB.Robotics.RobotStudio.Stations.RsTask)">
            <summary>
            Constructor needs the robots specified to allow to record both of them
                positions
            </summary>
            <param name="leftRobot"></param>
            <param name="rightRobot"></param>
        </member>
        <member name="P:MMI.MotionRecorder.IsRecording">
            <summary>
            Getter to obtain the recording field, meaning it is read-only
            </summary>
        </member>
        <member name="M:MMI.MotionRecorder.StartRecording">
            <summary>
            Initializes the motion recorder, pulses the CheckGrippers signal to
                refresh the state of grippers in the Add-In GUI and creates the paths
                 to be recorded for both tasks
            </summary>
        </member>
        <member name="M:MMI.MotionRecorder.StopRecording">
            <summary>
            Stops the motion recorder, finishes the recorded paths, erases a path if
                empty, adds the data declarations related to synchronization if
                necessary
            </summary>
        </member>
        <member name="M:MMI.MotionRecorder.AddGripperOrderOnExternalEvent(ABB.
            Robotics.RobotStudio.Environment.CommandBarButton,MMI.
            RecordGripperOrderEventArgs)">
            <summary>
            Adds gripper instructions to the environment if the motion recorder is
                recording
            </summary>
            <param name="sender"> Not used </param>
            <param name="e"> Event args containing the task to add the instructions
                to and whether it is a closing order or not </param>
        </member>
        <member name="M:MMI.MotionRecorder.AddTargetOrderOnExternalEvent(System.
            Object,System.EventArgs)">
            <summary>

            </summary>
            <param name="sender"> Object sending the event which triggers this method
                 </param>
            <param name="e"> Event args containing the information neccesary, of the
                type RecordTargetOrderEventArgs </param>
        </member>
        <member name="T:MMI.ThreadIgnorer">
            <summary>
            Class for leaving methods if another thread is already in a given piece
                of code.
            </summary>
        </member>
        <member name="F:MMI.ThreadIgnorer.count">
            <summary>
            Number of threads which started protecting code using this object.
            </summary>
        </member>
        <member name="F:MMI.ThreadIgnorer.maxCount">
            <summary>
            Maximum number of threads which can enter a protected zone instead of
                leaving the current method.
            </summary>
        </member>
        <member name="M:MMI.ThreadIgnorer.#ctor(System.Int32,System.Int32)">
            <summary>
            Constructor of the thread ignorer.
            </summary>
            <param name="initialInside"> Initial number of threads inside considered.
                </param>
            <param name="maxInside"> Maximum number of threads allowed in a
                protection.</param>
        </member>
        <member name="M:MMI.ThreadIgnorer.ProtectCodeEnd">
            <summary>
            Specify the end of the piece of code to protect
            </summary>
        </member>
        <member name="M:MMI.ThreadIgnorer.ProtectCodeStart">
            <summary>
            Specify the beggining of the piece of code to protect
            </summary>
        </member>
        <member name="T:MMI.GraphicVisualizer">
            <summary>
            Class for visualizing specific temporary graphics in the RS Visualization
                 of the station.
            </summary>
        </member>
        <member name="F:MMI.GraphicVisualizer.graphicLine">
            <summary>
            Graphics for line and frame and line to piece in case of alignment.
            </summary>
        </member>
        <member name="F:MMI.GraphicVisualizer.graphicFrame">
            <summary>
            Graphics for line and frame and line to piece in case of alignment.
            </summary>
        </member>
        <member name="F:MMI.GraphicVisualizer.graphicLineToPiece">
            <summary>
            Graphics for line and frame and line to piece in case of alignment.
            </summary>
        </member>
        <member name="F:MMI.GraphicVisualizer.station">
            <summary>
            Station reference.
            </summary>
        </member>
        <member name="M:MMI.GraphicVisualizer.#ctor(ABB.Robotics.RobotStudio.Stations
            .Station)">
            <summary>
            Constructor, needs the station reference.
            </summary>
            <param name="station">Station reference.</param>
        </member>
        <member name="M:MMI.GraphicVisualizer.EraseGraphic">
            <summary>
            Deletes the graphics if they already exist
            </summary>
        </member>
        <member name="M:MMI.GraphicVisualizer.EraseGraphic(System.Object,System.
            EventArgs)">
            <summary>
            Deletes the graphics if they already exist, can handle events
            </summary>
            <param name="sender">Object sending the event, not used</param>
            <param name="e">Event arguments, not used</param>
        </member>
        <member name="M:MMI.GraphicVisualizer.VisualizeFrame(ABB.Robotics.Math.
            Vector3,ABB.Robotics.Math.Matrix4,System.Double,System.Double,System.
            Double,System.Drawing.Color,MMI.Alignment)">
            <summary>
            Draws a new pair of line-frame graphics.
            </summary>
            <param name="from">Where the line starts.</param>
            <param name="to">Where the line ends and the frame is located.</param>
            <param name="size">Size of the frame.</param>
            <param name="lineWidth">Width of the line.</param>
            <param name="frameWidth">Width of the frame lines.</param>
            <param name="color">Color of the line.</param>
            <param name="alignment">Alignment state and data</param>
        </member>
        <member name="T:MMI.RecordGripperOrderEventArgs">
            <summary>
            EventArgs that will be sent from gripper buttons, containing whether it
                is commanded to close or open them.
            </summary>
        </member>
        <member name="F:MMI.RecordGripperOrderEventArgs.task">
            <summary>
            Task corresponding to the gripper order.
            </summary>
        </member>
        <member name="F:MMI.RecordGripperOrderEventArgs.isClosingOrder">
            <summary>
            Specifies if isClosingOrder with 1, 0 otherwise.
```

```xml
            </summary>
        </member>
        <member name="M:MMI.RecordGripperOrderEventArgs.#ctor(ABB.Robotics.
            RobotStudio.Stations.RsTask,System.Boolean)">
            <summary>
            Constructor for RecordGripperOrderEventArgs.
            </summary>
            <param name="task"> Task corresponding to the gripper order. </param>
            <param name="isClosingOrder"> Specifies if isClosingOrder with 1, 0
                otherwise. </param>
        </member>
        <member name="T:MMI.RecordTargetOrderEventArgs">
            <summary>
            Event args sent by the state machine for target recording,
            includes task sending the event and alignment data
            </summary>
        </member>
        <member name="F:MMI.RecordTargetOrderEventArgs.task">
            <summary>
            Task sending the event
            </summary>
        </member>
        <member name="F:MMI.RecordTargetOrderEventArgs.alignment">
            <summary>
            Alignment data
            </summary>
        </member>
        <member name="M:MMI.RecordTargetOrderEventArgs.#ctor(ABB.Robotics.RobotStudio
            .Stations.RsTask,MMI.Alignment)">
            <summary>
            Constructor of the event args
            </summary>
            <param name="task"> Task sending the event </param>
            <param name="alignment"> Alignment data </param>
        </member>
        <member name="T:MMI.Alignment">
            <summary>
            Class containing alignment data
            </summary>
        </member>
        <member name="E:MMI.Alignment.PropertyChanged">
            <summary>
            Event handler for sending events each time the alignment changes
            </summary>
        </member>
        <member name="F:MMI.Alignment.targetComponent">
            <summary>
            Target part of the alignment
            </summary>
        </member>
        <member name="P:MMI.Alignment.TargetOrientation">
            <summary>
            Property to get the object orientation, the same as the part orientation
            </summary>
        </member>
        <member name="P:MMI.Alignment.ObjectTranslation">
            <summary>
            Property to get the object translation
            </summary>
        </member>
        <member name="P:MMI.Alignment.Align">
            <summary>
            Property to get if the state is aligned, and set it invoking the align
                change when necessary
            </summary>
        </member>
        <member name="M:MMI.Alignment.#ctor(System.Boolean,ABB.Robotics.Math.Matrix4)
            ">
            <summary>
            Constructor of the Alignment class
            </summary>
            <param name="doAlign"> Align or not </param>
            <param name="targetMatrix"> TargetMatrix of where to align </param>
        </member>
        <member name="T:MMI.StateMachine">
            <summary>
            This state machine class allows to monitor the states driven by the pose
                of the hand and when allignments commands are given by pointing with
                a hand
            </summary>
        </member>
        <member name="F:MMI.StateMachine.OpenedHandEvent">
            <summary>
            Event handlers for when the hand opens, closes or disappears
            </summary>
        </member>
        <member name="F:MMI.StateMachine.ClosedHandEvent">
            <summary>
            Event handlers for when the hand opens, closes or disappears
            </summary>
        </member>
        <member name="F:MMI.StateMachine.GoneHandEvent">
            <summary>
            Event handlers for when the hand opens, closes or disappears
            </summary>
        </member>
        <member name="F:MMI.StateMachine.inputRobot">
            <summary>
            Task of the robot that this state machine commands
            </summary>
        </member>
        <member name="F:MMI.StateMachine.alignment">
            <summary>
            Alignment state and data of the state machine
            </summary>
        </member>
        <member name="F:MMI.StateMachine.curState">
            <summary>
            Current state of the state machine
            </summary>
        </member>
        <member name="T:MMI.StateMachine.StateName">
            <summary>
            Enumerated states names
            </summary>
        </member>
        <member name="F:MMI.StateMachine.StateName.ShowOpenedHand">
            <summary>
            When hand is not located, it must be shown opened before movement
            </summary>
        </member>
        <member name="F:MMI.StateMachine.StateName.Visualize">
            <summary>
            When shown opened, only visualization
            </summary>
        </member>
        <member name="F:MMI.StateMachine.StateName.Imitate">
            <summary>
            When finally closed, there is movement
            </summary>
        </member>
        <member name="M:MMI.StateMachine.#ctor(ABB.Robotics.RobotStudio.Stations.
            RsTask,ABB.Robotics.RobotStudio.Environment.CommandBarComboBox)">
            <summary>
            Constructor of the state machine
            </summary>
            <param name="inputRobot"> Task of the robot to command </param>
            <param name="buttonToChangeImg"> Button of the interface for debugging
                state </param>
        </member>
        <member name="M:MMI.StateMachine.OnHandGone">
            <summary>
            Method to be executed when hand disappears, invoking an event
            </summary>
        </member>
        <member name="M:MMI.StateMachine.OnHandOpened">
            <summary>
            Method to be executed when hand opens, invoking an event
            </summary>
        </member>
        <member name="M:MMI.StateMachine.OnHandClosed(MMI.HandAndPoseRs,ABB.Robotics.
            RobotStudio.Stations.RsTask)">
            <summary>
            Method to be executed when hand closes, invoking an event
            </summary>
            <param name="hand"></param>
            <param name="robot"></param>
        </member>
        <member name="M:MMI.StateMachine.OnHandPointing(ABB.Robotics.RobotStudio.
            Stations.RsTask,ABB.Robotics.Math.Matrix4)">
            <summary>
            This method searches along geometries in the station around the end
                effector and tries to align it to the nearest geometry frame.
            Therefore the robot enters a state where it should be aligning to the
                axis of that object.
            Pointing will switch between align or dealign the robot to the object.
            </summary>
        </member>
        <member name="M:MMI.StateMachine.OnAlignFromSpeech(System.String)">
            <summary>
            Method to be executed when an alignment to an specific piece is requested
                , instead of finding the nearest one.
            </summary>
            <param name="pieceName"> strin of the part as it appears in the RS GUI.</
                param>
        </member>
        <member name="M:MMI.StateMachine.OnMisalignFromSpeech">
```

```
            <summary>
            Method for unaligning the arm from speech, directly sets alignment.Align
                to false, meaning that there is no alignment
            </summary>
        </member>
        <member name="M:MMI.StateMachine.Iterate(MMI.HandAndPoseRs,ABB.Robotics.Math.
            Matrix4)">
            <summary>
            Iterates according to hand pose, jumping through states
            </summary>
            <param name="inputHand"> HandAndPose in RS format </param>
            <param name="targetGMatrix"> Global matrix of the hand position in RS </
                param>
            <returns></returns>
        </member>
        <member name="T:MMI.Pose">
            <summary>
            Enumerated type for hand poses
            </summary>
        </member>
        <member name="F:MMI.Pose.GONE">
            <summary>
            Tag meaning that the hand is not in the frame or no condition of the
                other tags is found
            </summary>
        </member>
        <member name="F:MMI.Pose.CLOSED">
            <summary>
            Tag meaning that the hand has no finger extended
            </summary>
        </member>
        <member name="F:MMI.Pose.POINTING">
            <summary>
            Tag meaning that the thumb is extended
            </summary>
        </member>
        <member name="F:MMI.Pose.OPENED">
            <summary>
            Tag meaning that the hand has all the fingers extended
            </summary>
        </member>
        <member name="T:MMI.HandAndPoseLeap">
            <summary>
            Class for structuring the data of the Leap motion hand.
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseLeap.position">
            <summary>
            Position of the hand in the 3D space (X,Y,Z) taking as reference the Leap
                 Motion Sensor and its axes
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseLeap.palmNormal">
            <summary>
            Direction of the unitary normal vector from the hand palm  taking as
                reference the Leap Motion Sensor and its axes
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseLeap.palmDirection">
```

```xml
            <summary>
            Direction of the unitary vector parallel to the palm direction , pointing
                to the center part of the fingers , taking as reference the Leap
                Motion Sensor and its axes
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseLeap.pointDir">
            <summary>
            Direction of the extended finger if the hand is pointing , as unitary
                vector , taking as reference the Leap Motion Sensor and its axes
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseLeap.filtPose">
            <summary>
            Filtered pose
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseLeap.instantPose">
            <summary>
            Raw pose found in this frame
            </summary>
        </member>
        <member name="M:MMI.HandAndPoseLeap.#ctor">
            <summary>
            Constructor of the class initializing empty vectors and GONE poses
            </summary>
        </member>
        <member name="M:MMI.HandAndPoseLeap.#ctor(Leap.Vector,Leap.Vector,Leap.Vector
            ,Leap.Vector,MMI.Pose,MMI.Pose)">
            <summary>
            Constructor of the class , initializing the internal values with given
                values
            </summary>
            <param name="position"> Position of the hand in the 3D space (X,Y,Z)
                taking as reference the Leap Motion Sensor and its axes </param>
            <param name="palmNormal"> Direction of the unitary normal vector from the
                 hand palm, taking as reference the Leap Motion Sensor and its axes <
                /param>
            <param name="palmDirection">Direction of the unitary vector parallel to
                the palm direction , pointing to the center part of the fingers ,
                taking as reference the Leap Motion Sensor and its axes </param>
            <param name="pointDir"> Direction of the extended finger if the hand is
                pointing , as unitary vector , taking as reference the Leap Motion
                Sensor and its axes </param>
            <param name="filtPose"> Filtered pose </param>
            <param name="instantPose"> Raw pose found in this frame </param>
        </member>
        <member name="T:MMI.HandAndPoseRs">
            <summary>
            Class for structuring the data of the Leap Motion hand, with the data
                types supported by the RS methods.
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseRs.position">
            <summary>
            Position of the hand in the 3D space (X,Y,Z) with coordinates transformed
                 to the RS world according to the LeapMotion frame located in the
                station
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseRs.palmNormal">
            <summary>
            Direction of the unitary normal vector from the hand palm, transformed to
                 the RS world according to the LeapMotion frame located in the
                station
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseRs.palmDirection">
            <summary>
            Direction of the unitary vector parallel to the palm direction, pointing
                to the center part of the fingers, transformed to the RS world
                according to the LeapMotion frame located in the station
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseRs.pointDir">
            <summary>
            Direction of the extended finger if the hand is pointing, as unitary
                vector, transformed to the RS world according to the LeapMotion frame
                 located in the station
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseRs.filtPose">
            <summary>
            Filtered pose
            </summary>
        </member>
        <member name="F:MMI.HandAndPoseRs.instantPose">
            <summary>
            Raw pose found in this frame
            </summary>
        </member>
        <member name="M:MMI.HandAndPoseRs.#ctor">
            <summary>
            Constructor of the class initializing empty vectors and GONE poses
            </summary>
        </member>
        <member name="M:MMI.HandAndPoseRs.#ctor(ABB.Robotics.Math.Vector3,ABB.
            Robotics.Math.Vector3,ABB.Robotics.Math.Vector3,ABB.Robotics.Math.Vector3
            ,MMI.Pose,MMI.Pose)">
            <summary>
            Constructor of the class, initializing the internal values with given
                values
            </summary>
            <param name="position"> Position of the hand in the 3D space (X,Y,Z)
                transformed to the RS world according to the LeapMotion frame located
                 in the station </param>
            <param name="palmNormal"> Direction of the unitary normal vector from the
                 hand palm, transformed to the RS world according to the LeapMotion
                frame located in the station </param>
            <param name="palmDirection">Direction of the unitary vector parallel to
                the palm direction, pointing to the center part of the fingers,
                transformed to the RS world according to the LeapMotion frame located
                 in the station </param>
            <param name="pointDir"> Direction of the extended finger if the hand is
                pointing, as unitary vector, transformed to the RS world according to
                 the LeapMotion frame located in the station </param>
            <param name="filtPose"> Filtered pose </param>
            <param name="instantPose"> Raw pose found in this frame </param>
```

```xml
        </member>
        <member name="P:MMI.HandAndPoseRs.Rotation">
            <summary>
            HandRotation: Gets the rotation matrix of the hand based on its fields
                data
            </summary>
        </member>
        <member name="T:MMI.FrameAndPoseLeap">
            <summary>
            A class containing the pair of hands
            </summary>
        </member>
        <member name="F:MMI.FrameAndPoseLeap.left">
            <summary>
            Left hand data
            </summary>
        </member>
        <member name="F:MMI.FrameAndPoseLeap.right">
            <summary>
            Right hand data
            </summary>
        </member>
        <member name="T:MMI.FrameAndPoseRs">
            <summary>
            A class containing the pair of hands in RS format.
            </summary>
        </member>
        <member name="F:MMI.FrameAndPoseRs.left">
            <summary>
            Left hand data
            </summary>
        </member>
        <member name="F:MMI.FrameAndPoseRs.right">
            <summary>
            Right hand data
            </summary>
        </member>
        <member name="F:MMI.FrameAndPoseRs.bothPointing">
            <summary>
            Boolean reserved for future use.
            </summary>
        </member>
        <member name="T:MMI.Filter">
            <summary>
            Filter class, implements the INotifyPropertyChanged for notifying changes
                in pose
            </summary>
        </member>
        <member name="E:MMI.Filter.PropertyChanged">
            <summary>
            Event for change of a property.
            </summary>
        </member>
        <member name="T:MMI.Filter.FloatToFilter">
            <summary>
            Enum for specifying what value to filter
            </summary>
        </member>
        <member name="F:MMI.Filter.POSE_DEQUE_SIZE">
```

```
            <summary>
            Number of frames for pose filtering.
            </summary>
        </member>
        <member name="F:MMI.Filter.POSITION_DEQUE_SIZE">
            <summary>
            Number of frames for position filtering.
            </summary>
        </member>
        <member name="F:MMI.Filter.PALMNORMAL_DEQUE_SIZE">
            <summary>
            Number of frames for palm normal filtering.
            </summary>
        </member>
        <member name="F:MMI.Filter.PALMDIRECTION_DEQUE_SIZE">
            <summary>
            Number of frames for palm direction filtering.
            </summary>
        </member>
        <member name="F:MMI.Filter.POINTDIRECTION_DEQUE_SIZE">
            <summary>
            Number of frames for direction of the pointing finger filtering, not used
                 in the implementation.
            </summary>
        </member>
        <member name="P:MMI.Filter.CurrFiltPoseL">
            <summary>
            Property for the current filtered pose of the left hand
            </summary>
        </member>
        <member name="P:MMI.Filter.CurrFiltPoseR">
            <summary>
            Property for the current filtered pose of the right hand
            </summary>
        </member>
        <member name="M:MMI.Filter.NewInput(Leap.Frame)">
            <summary>
            Sets the frame to filter, if there is no frame in the filter, filtering
                methods will return invalid data, use this method with each new frame
                , if not the last added frame
            will have more weight.
            </summary>
            <param name="frame"> Leap Motion Frame from the sensor. </param>
            <returns> 0 if error, 1 if everything is Ok. </returns>
        </member>
        <member name="M:MMI.Filter.GetFrameFilteredLeapFormat">
            <summary>
            Returns the filtered frame data with the data format of Leap Motion
            </summary>
        </member>
        <member name="M:MMI.Filter.GetFrameFilteredRsFormat(ABB.Robotics.RobotStudio.
            Stations.Transform)">
            <summary>
            Returns the filtered frame data with the data format of Robot Studio.
                Needs the Leap Motion frame added to the RS station
            </summary>
            <param name="leapFrameInRs"> The transform of the leap motion frame
                required in the RS station </param>
        </member>
        <member name="M:MMI.Filter.FilterAll">
            <summary>
            Executes all the other filter methods
            note that executing a filtering method more than once will add more
                weight to the
            current input frame
            </summary>
            <returns> 0 if error, 1 if everything is Ok </returns>
        </member>
        <member name="M:MMI.Filter.FilterLeft">
            <summary>
            Filters left hand.
            </summary>
            <returns> 0 if error, 1 if everything is Ok </returns>
        </member>
        <member name="M:MMI.Filter.FilterRight">
            <summary>
            Filters right hand.
            </summary>
            <returns> 0 if error, 1 if everything is Ok </returns>
        </member>
        <member name="M:MMI.Filter.FilterPoses">
            <summary>
            Filter poses of both hands.
            </summary>
            <returns> 0 if error, 1 if everything is Ok </returns>
        </member>
        <member name="M:MMI.Filter.FilterPointDirections">
            <summary>
            Filters the point direction of both hands
            </summary>
            <returns> 0 if error, 1 if everything is Ok </returns>
        </member>
        <member name="M:MMI.Filter.FilterPalmPositions">
            <summary>
            Filters the spatial position of both hands(X, Y, Z)
            (Left, Right)
            </summary>
            <returns></returns>
        </member>
        <member name="M:MMI.Filter.FilterAngles">
            <summary>
            Filters the palm normal and direction of the hands
            </summary>
            <returns></returns>
        </member>
        <member name="M:MMI.Filter.FilterPoseLeft">
            <summary>
            Filters the left pose
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterPoseRight">
            <summary>
            Filters the right pose
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterPalmPositionLeft">
```

```xml
            <summary>
            Filters the left palm position
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterPalmPositionRight">
            <summary>
            Filters the right palm position
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterPointDirectionLeft">
            <summary>
            Filters the left pointing direction of the finger
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterPointDirectionRight">
            <summary>
            Filters the right pointing direction of the finger
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterAnglesLeft">
            <summary>
            Filters the palmNormal and palmDirection of the left hand
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="M:MMI.Filter.FilterAnglesRight">
            <summary>
            Filters the palmNormal and palmDirection of the right hand
            </summary>
            <returns> Returns 1 </returns>
        </member>
        <member name="T:MultiModalInterface.Properties.Resources">
            <summary>
              A strongly-typed resource class, for looking up localized strings, etc.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.ResourceManager">
            <summary>
              Returns the cached ResourceManager instance used by this class.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.Culture">
            <summary>
              Overrides the current thread's CurrentUICulture property for all
              resource lookups using this strongly typed resource class.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.CheckGrippersSignal
            ">
            <summary>
              Looks up a localized string similar to CheckGrippers.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            CloseLeftGripperSignal">
            <summary>
              Looks up a localized string similar to CloseLeft.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            CloseRightGripperSignal">
            <summary>
              Looks up a localized string similar to CloseRight.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageCopy">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageGoneLeft">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageGoneRight">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageGripperLeft">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageGripperRight">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageHandClose">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageHandCloseFlip">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageHandOff">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageHandOpen">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageHandOpenFlip">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageLeapFrame">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageLeft">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageLeftAligned">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageLinked">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageMicOff">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageMicOn">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageNoLinked">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageNoRotate">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageNoTranslate">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imagePoint">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imagePointFlip">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageRight">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
```

```xml
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageRightAligned">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageRotate">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageStart">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageStop">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.imageTranslate">
            <summary>
              Looks up a localized resource of type System.Drawing.Bitmap.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            IsClosedLeftGripperSignal">
            <summary>
              Looks up a localized string similar to IsClosedLeft.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            IsClosedRightGripperSignal">
            <summary>
              Looks up a localized string similar to IsClosedRight.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.LeapFrameName">
            <summary>
              Looks up a localized string similar to LeapFrame.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            UndoStepCloseGripperLeft">
            <summary>
              Looks up a localized string similar to Close Left Gripper.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            UndoStepCloseGripperRight">
            <summary>
              Looks up a localized string similar to Close Right Gripper.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.UndoStepLeapFrame">
            <summary>
              Looks up a localized string similar to Leap Frame.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.UndoStepMovement">
            <summary>
              Looks up a localized string similar to Movement.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            UndoStepOpenGripperLeft">
            <summary>
              Looks up a localized string similar to Open Left Gripper.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            UndoStepOpenGripperRight">
            <summary>
              Looks up a localized string similar to Open Right Gripper.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            UndoStepRecordMovement">
            <summary>
              Looks up a localized string similar to Record Movement.
            </summary>
        </member>
        <member name="P:MultiModalInterface.Properties.Resources.
            UndoStepStartRecording">
            <summary>
              Looks up a localized string similar to Start Recording.
            </summary>
        </member>
    </members>
</doc>
```