POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Corso di Laurea Magistrale
in Mechatronic Engineering

Master's degree thesis

Interfacing Matlab with the collaborative robot UR3

Supervisors

prof. Mauro Stefano
prof. Pastorelli Stefano Paolo
prof. Antonelli Dario

Candidate

Gaidano Matteo

December 2018

Index:

1.Introduction..... 1
1.1. Collaborative robotics 1
2.The project..... 3
2.1. The complete project 3
2.2. Anti-collision algorithm 4
2.3. Anti-noise filter 4
2.4. Optimization algorithm 4
2.5. Study of single devices 4
2.6. Interconnection of the devices 4
2.7. Codes of support 4
2.8. The focus of the thesis 5
3.Network architecture 6
4.The UR3 robot 7
4.1. What it is 7
4.2. How to program the robot 7
4.3. Ethernet connection 8
4.4. Robot client interfaces 9
4.5. Computer as server 10
5.Robotiq gripper and FT sensor 11
5.1. FT 300 Sensor 11
5.2. 2F-85 Gripper 12
6.Codes 13
6.1. Matlab 13
6.2. Python 13
6.3. UR library 13
6.4. Robotiq library 14
6.5. Library in Matlab – UR 16
6.6. Library in Matlab – Robotiq 36
6.7. Library in Python – UR 39
6.8. Library in Python – Robotiq 63
6.9. URCap of support 66
6.10. Load a complete URScript 69

7.Test of libraries and how to program.....71
7.1. Introduction 71
7.2. Test 1 – Take position 71
7.3. Test 2 – Movecjl 72
7.4. Test 3 – Slider and speedl 91
7.5. Test 4 – Read FT sensor 98
8.Conclusions.....100
9.Bibliography.....101
10. Appendix

Summary

This thesis is part of a bigger project on cobots.

Cobots are collaborative robotic arms with sensors and control on motor currents to work with humans in safety. Though they are already collaborative, the project wants to implement a vision control system for changing in real-time the path behavior, therefore improving the safety.

The focus of this thesis is to study the interconnection protocols between robot and computer.

In particular how control a robot UR3, produced by Universal Robot, from Matlab.

To achieve the target also a study of Python has been done.

As result a series of libraries to control the robot from Matlab and Python have been written together a collection of examples to use them.

1-Introduction

Collaborative Robotics

From its birth around '40, the robotics rose quickly taking place everywhere, in industrial production, exploration, medical field, defense and also in our home.

Although the working fields are many and different in all of them the robotics change completely our life and our way of working.

For example in medical field robots are able to increase the precision during a surgery (fig1.1) while advance prosthesis and physiotherapy robots are able to restore in part humane injuries. In defense and exploration fields robots go in places harmful for humans (fig1.2, 1.3), as under the water, space, other planets or in radioactive or war area to do data acquisition, rescue or scientific analysis.

Fig 1.1 Da Vinci Robot

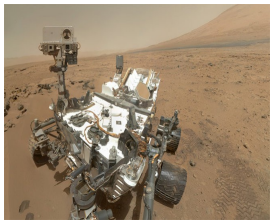


Fig 1.2 Curiosity

Fig 1.3 Wheelbarrow Mk8 plus

In industrial field robotics found its bigger contribution, moving heavy objects with speed and precision or doing boring and repetitive jobs (fig1.4, 1.5). This innovation, called third industrial revolution, changes completely our world and nowadays the Industry 4.0 is following the previous revolution with a wide use of internet connection and an improving in robotics.

Fig 1.4 Robotic arm Comau

Fig 1.5 Industrial plant

Roles, shapes and dimensions are different robot by robot. Some of them, the bigger and heavier, must work alone to avoid causing injuries to humans. Others smaller and lighter can work near humans.

Collaborative robotics works on robots that share the workspace with humans. Being the collaboration human robot the main features of collaborative robotics, as the name says, two are the more important points: the user interface and the safety.

Fig 1.6 Roomba

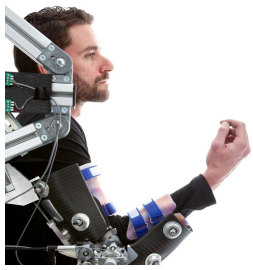


Fig 1.7 Project Ronda

Fig 1.8 Pepper

For example there are the domestic robots that clean the floor independently (fig 1.6), the recent little collaborative robot arm (cobot), the futuristic humanoid robots used by Japaneses in hotels and supermarkets to help the clients (fig1.8) or the previous mentioned medical robots (fig1.7).

In particular cobots were born in the industrial contest. The first company to produce a cobot was Kuka in 2004 with LBR3 model (fig 1.9). After another company called the Universal Robot, that produces only cobots, released its first robot, the UR5, in 2008 (fig1.10). Fanuc released its first collaborative robot, the CR-35iA ,in 2015 (fig1.11).

These are only three of many companies that have started to produce collaborative robots in the last years.

Fig 1.9 Kuka LBR3

Fig 1.10 Universal Robot UR5

Fig 1.11 Fanuc CR-35iA

The real interest for this robot arose from their possibility of integrating the complex role of a human with the precision and boring job of a robot all in a smaller working place. Without physical barriers the human operator can easily control the work in progress of the robot and if necessary correct errors or works with them in a shared space on the same product.

Seeing the growing presence on the market of these kind of robots, also the regulations EN ISO 10218 standard Parts 1 and Part 2 and the ISO/TS 15066 have been revised on the safety requirement for collaborative robotics.

In order to improve safety, companies use different techniques. Some are passive,such as the dimension, rounded shapes or soft external material; others active and more complex, such as the control of the current on the motor, external force sensor or the addiction of vision systems.

Cobots started a revolution that will continue in the next years and that will change completely our factories and our lives.

2-The project

The complete project

This thesis is a part of a bigger project on collaborative robotics.

All the cobots were born with the idea of reducing the risk caused by accidental contact with the human operator.

This project wants to improve this idea by implementing the safety of the robot and adding a 3D vision system that can make the robot change the trajectory in real-time.

Starting from the thesis of Giorgio Missiaggia called “Studio di algoritmi anticollisione in un ambiente virtuale” in which with a kinect v1 and a simulator he tried an anti-collision algorithm, we implement a real system.

The physical system consists of a UR3 cobot by Universal Robot, two kinect v2 by Microsoft, three computers and a router. (fig2.1, 2.2)

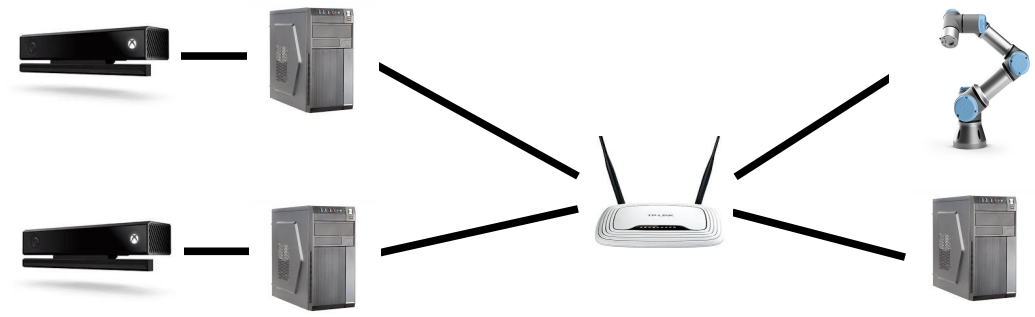


Fig 2.1 Scheme of the complete system

To be able of implement the complete system a lot of knowledge is necessary. Indeed the project presents many problems.

Knowing the algorithm is necessary to obtain the data to use that will come from the kinects and the robot. This is a first problem because both use their own protocol to study to use it in the proper way.

Then the data obtained from the kinects must be put together and from the composition it is necessary to obtain only the data of our interest. The operation requires a lot of computation power, indeed to do it is necessary to use an anti-noise filter and a optimization algorithm.

Another problem arises from the knowledge of robot. Any company develops its own software to control the robots and the Universal Robot too. A good knowledge of the software is necessary to control the cobot in real-time.

A team of master degree students and PhD students were composed in order to reach the goal.

The team worked on all the different problems mentioned before with the purpose to overcome them.

Fig 2.2 PhD Scimmi Leonardo with the implemented real system

The anti-collision algorithm

Over the years many kind of anti-collision algorithm have been proposed. The one that was chosen by Missiaggia in his thesis and that was studied in depth by the team is based on virtual fields.

This method works thanks to virtual fields constructed on the base of the acquisition data. The operator generates a repulsor field while the target generates an attractive field. The path of the robot will be a sum of the field. The end effector will try to reach the target but in the meanwhile will be repelled by the presence of the operator.

Anti-noise filter

The human movement is captured from two kinects that create a 3D image of the body. It is very common that the image is corrupted by noise data that must be deleted before computing the distance robot to human using filters.

Optimization algorithm

Although computing the distance between two points is not a complicated operation for a computer, it becomes a problem in the moment in which the distances to compute are infinite. This is the case of two real bodies. To compute the minimum distance it is not possible compute infinite distances and then chose the shorter.

In this case different methods of discretization and/or optimization algorithm are used.

Also for our case the team had to study the optimization algorithm to reduce the computation time and do the system works in real time.

Study of single devices

The knowledge of single devices is important to obtain the best result for our task and is also important for possible future applications.

Both Kinect than UR3 are devices with owns protocols that must be understood in the depth to obtain the best result.

Interconnection of the devices

To achieve our target more than one computer is necessary. The anti-collision algorithm needs a lot of computation power because in order to find the minimum distance, an optimization algorithm is necessary. Also the 3D video provided by the kinects needs a big amount of computation power.

So as a starting point, because the software and the hardware of the complete prototype system are not optimized, it was decided to use an architecture where every external device has its own computer.

The kinects are connected directly to their own computer with a USB 3.0, while the computers and the robot are connected together with an ethernet connection, for this purpose a router was added to the system.

Codes of support

The codes of support consist of a series of computer libraries written in different languages to make it easier to write, read and re-use the main code.

These libraries contain functions able to connect the devices and share data with each other or rename big and complex amount of data.

The focus of this thesis

In particular my task was to understand the standard and not standard protocol to connect the computer to the robot.

To reach the goal I studied new topics such as network connections, Matlab and Python programming and I did tests on the UR3 to better understand its behavior.

After some months of research I wrote several codes of support to easily interconnect the robot with the computer using three types of languages: Python, Matlab and URScript.

3 – Network architecture

The most used network architecture is the one proposed by Unix called Internet protocol Suite but known by everyone as TCP/IP protocol. Exactly as other network architectures, it has a layer composition.

Compared with the ISO/OSI network we can identify the most important specification of TCP/IP in the layer 4 starting from below (fig3.1).

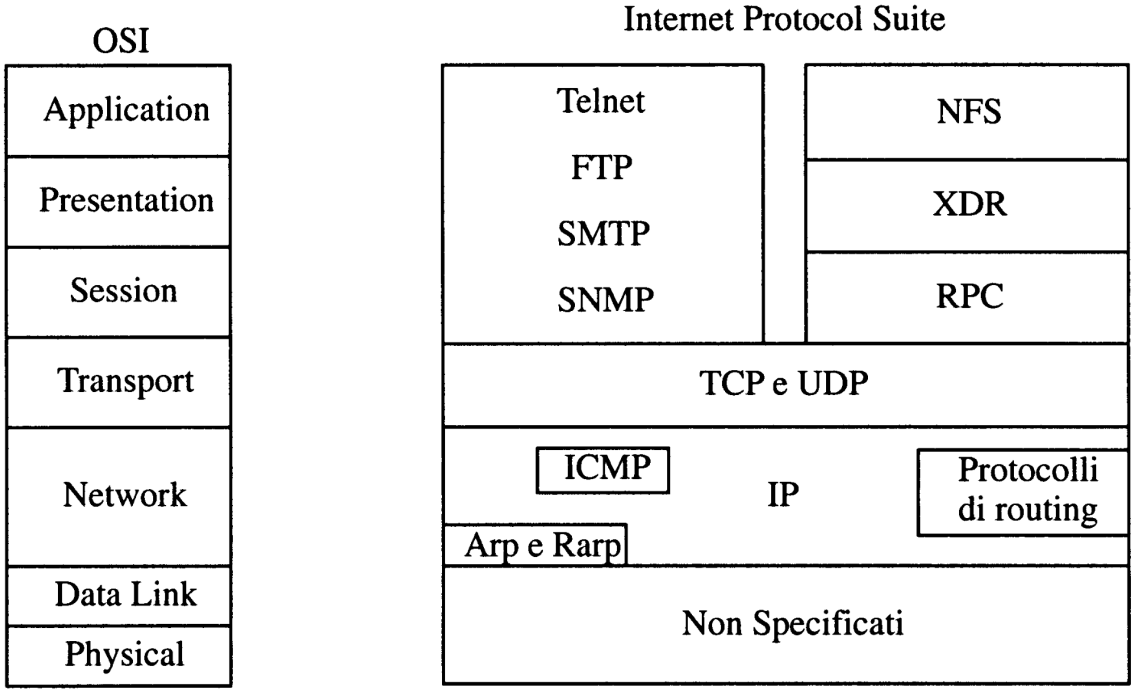


Fig 3.1 Network architecture

The layer four can use two different protocols: TCP, Transmission Control Protocol, and UDP, User Datagram Protocol.

The TCP implements an error control on the lost data while UDP does not control but is faster. Both the protocols are really fast in any case for our purpose, so a TCP protocol was chosen for more reliability.

TCP has the role to provide as interface at the higher level, a univocal connection between two hosts, while improving control on data transmitted by the lower level. Indeed in an Ethernet connection, very fast and stable, the Data Link layer uses a non-connected protocol, after that there is a control on the error of the transmitted data but not a response for ask again corrupted data or deleted cloned data.

The TCP protocol to connect two devices asks for an IP address and a port. The IP address can be IPv4 or IPv6 and is used by the Network layer to send the datagram to the correct destination while the port is used by the TCP layer to create a unique connection. Indeed any couple IP port creates a socket. Two sockets create an end-to-end connection.

Any device that uses TCP network architecture uses the Big Endian convention for the IP fields, instead this is not true for information that must be transmitted. Indeed it is a good idea to choose as convention the same, that usually is also the one chosen by default by some functions.

Big Endian convention states that the most significant bit must stay in the lower memory address. In alternative other conventions can be chosen, such as Little Endian.

4 – The UR3 Robot

What it is

Produced by Universal Robot, UR3 is the smallest robot of a family of three cobots (fig4.2). All three have the same operating system so the implemented codes here proposed can work also on the other two.

The robot is an arm with six degrees of freedom.

The control box uses a dedicated operating system called Polyscope, based on Linux and has a lot of I/Os including USB, Ethernet, modbus, digital and analog I/Os.

A teach pendant touch screen connected with the control box via cable allows to program the robot in an easy and intuitive way thanks to the dedicated GUI. (fig4.1, 4.3)

Fig 4.1 UR3 arm, cabinet and teach pendant

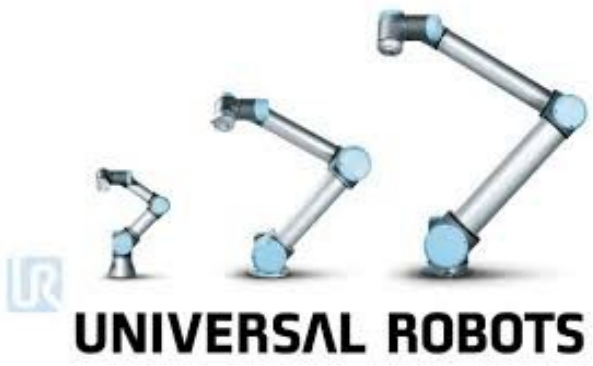


Fig 4.2 UR3, UR5 and UR10

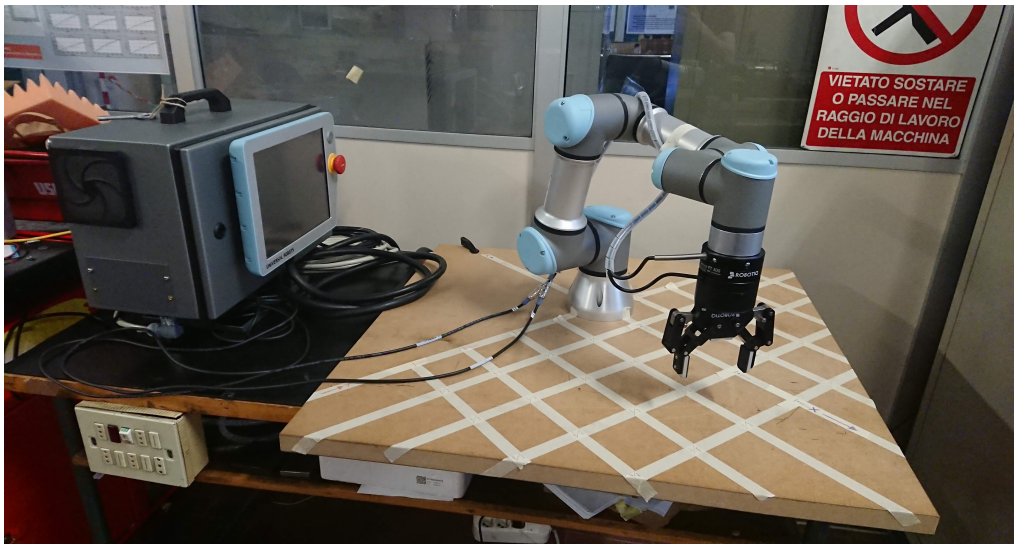


Fig 4.3 Our UR3

How to program the robot

The robot can be programmed in two ways.

The first, is the simplest and the first reason that makes this robot collaborative. The URCap is the language used on the teach pendant and thanks to an easy and intuitive GUI (fig4.4) allows everyone to program the robot using a few online interactive video lessons. The URCap can also be seen as a mask that conceals a more complicated program language, the URScript.

The second way is the URScript. The URScript is an ad hoc language that allows one to program the

robot with more degrees of freedom. Universal Robot provides a pdf file [8] where the available functions are descripted.

While the URCap needs its own GUI support, the URScript can be written in txt file and then imported in a second moment to the robot or sent via a network from the pc in real time, (chapter 6).

The first and the second way can be chained together to obtain a complete and powerful code. Usually a URCap can include URScript code using the Scrip command. This integration allows a more powerful URCap code without a complete and complicated URScript.

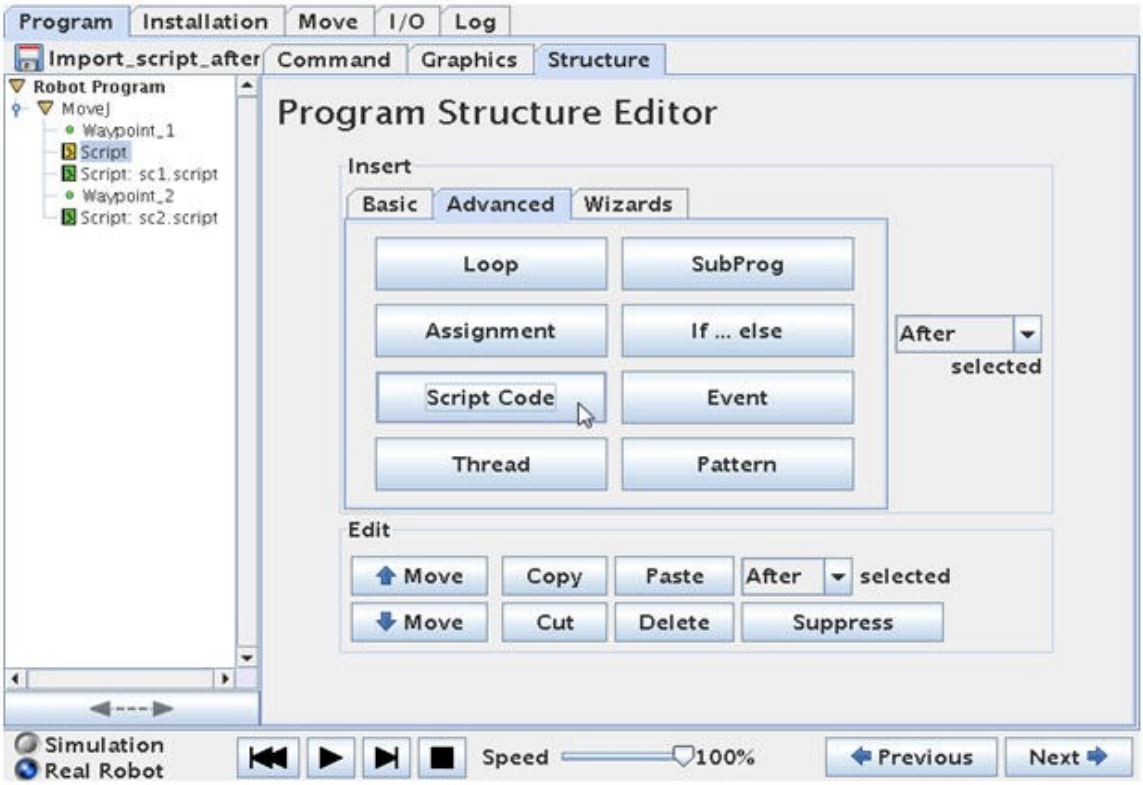


Fig 4.4 Polyscope interface

In this thesis will be shown different ways to control the robot form a PC. So both a URScript approach and a mixed approach is used.

On the PC side the code can be written in any programming language. In this case Python and Matlab have been chosen, having in this way a double library of both the languages.

Ethernet connection

The physical connection robot-computer is realized using an ethernet cable. The ethernet port is located under the control box case as it is possible to see in fig4.5, 4.6.

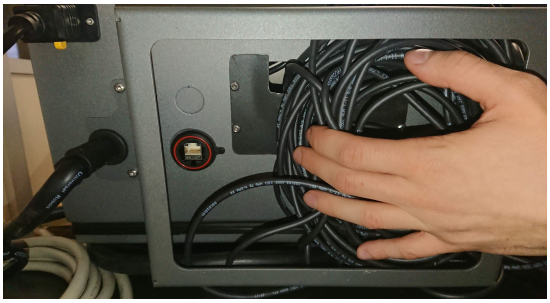


Fig 4.5 bottom side of the robot control box

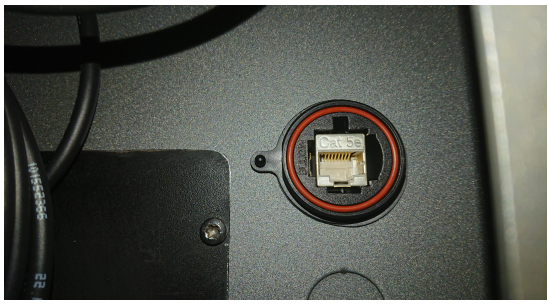


Fig 4.6 Detail of ethernet port

The cable can be connected directly to the computer or pass through a switch or router.

A socket using standard TCP/IP protocol is used to transfer command and data.

Robot client interfaces

The robot as server provides four client interfaces with four different ports and with different roles. All the ports are always open also if no robot programs is running.

The client interfaces are:

Name	port	input	output	frequency
Primary client interface (PCI)	30001	URScript	data	10Hz
Secondary client interface (SCI)	30002	URScript	data	10Hz
Real-Time client interface (RTCI)	30003	URScript	data	125Hz
Real-Time Data Exchange (RTDE)	30004	RTDE protocol	RTDE protocol	125Hz

The primary and secondary client interface working at 10Hz have not been studied in detail, but can receive URScript ad command and provide a sequence of data.

The real-time client provides information about the robot refreshed at 125Hz. The list and the order of the data are provided by Universal Robot in an excel file [10], the one used for writing the code is attached.

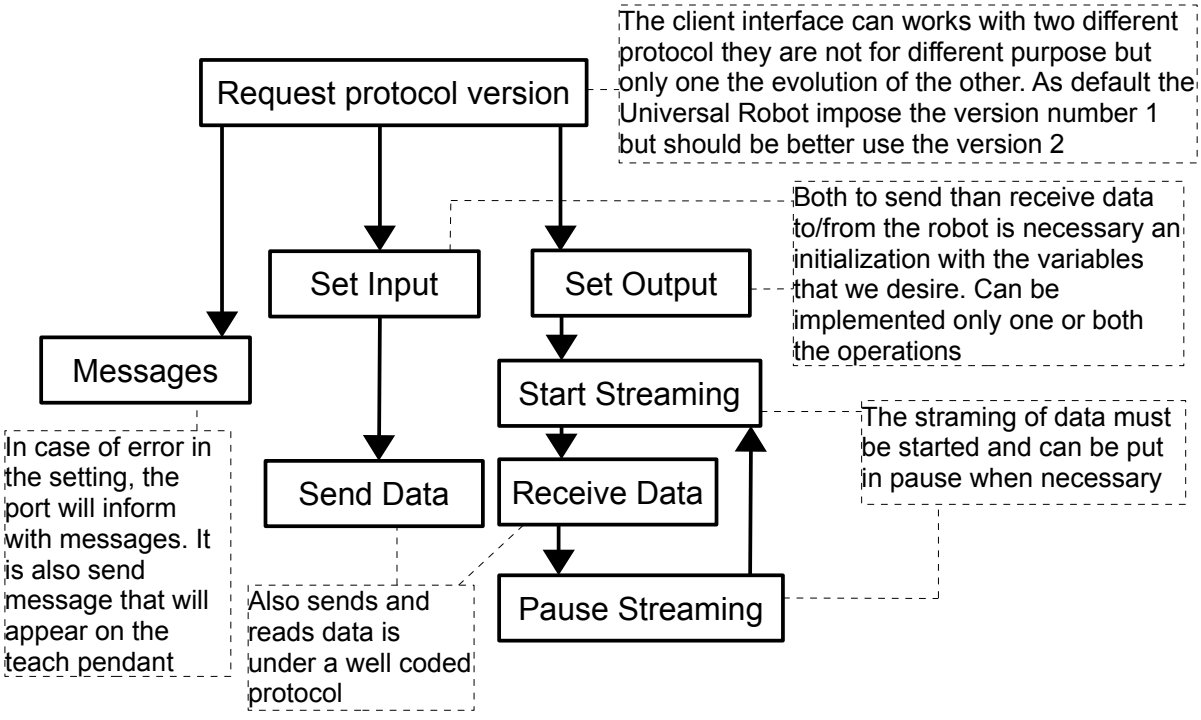
The RTCI can also receive URScript commands; any row must be delimited by the newline character "\n". More information on how use the URScript can be found in chapter 6.

The RTDE is a configurable interface for sending and receiving data at 125Hz. In particular it is possible to work with the internal register to exchange variables of the sensors.

This client interface is not ready to be used as the other three. It needs a sequence of initialization to decide which data provides and which receives.

Any operation of initialization and of data sharing is recognized by a code. Indeed any message sent or received on the RTDE port must start with the number of the byte of which is composed and the code of the operation.

After the connection the following operations are possible. More information in [9].



A part of the possible operations possible on RTDE port are also reported in the UR library.

Computer as server

It is also possible to use the computer as a server and open, thanks to the script code, a socket as client on the robot. In this case the connection will work at 125Hz but a script on the robot is mandatory.

A possible URCap of support is expose in chapter 6.

5 – ROBOTIQ Gripper and FT sensor

As a tool at the end of the UR robotic arm we have to add a FT 300 sensor (fig5.1) and a 2F-85 gripper (fig5.2) both produced by ROBOTIQ.

ROBOTIQ with the hardware provides also the software to install on Polyscope to control the devices from the teach pendant.

The nodes, name given by Universal Robot to any external software installed on Polyscope, allow to control easily sensor and gripper with URcap. It is more to use difficult a standalone URScript. Indeed any time that we run an URcap, after a node has been installed, the URScript produced add to the code all the information of the node itself. This operation could add a lot of lines of code. So also for very little programs that use only one external device that have a node all the corresponding code is necessary.

All information about the devices can be found in [12-13-14].

Fig 5.1 FT 300 Sensor

Fig 5.2 2F-85 Gripper

FT 300 Sensor

Positioned between the end of the arm and the gripper, measures forces and torques acting between the two.

After the installation of its own software on the Polyscope it is possible read the value of forces and torques measured by the sensor in the “output_double_register_X”, with X from 0 to 5.

The information was founded studying the code of the node in a URScript.

```
[...]
global TCP_FX_ACTUAL_OUTPUT_DOUBLE = 0
global TCP_FY_ACTUAL_OUTPUT_DOUBLE = 1
global TCP_FZ_ACTUAL_OUTPUT_DOUBLE = 2
global TCP_MX_ACTUAL_OUTPUT_DOUBLE = 3
global TCP_MY_ACTUAL_OUTPUT_DOUBLE = 4
global TCP_MZ_ACTUAL_OUTPUT_DOUBLE = 5
[...]
write_output_float_register(TCP_FX_ACTUAL_OUTPUT_DOUBLE,0)
write_output_float_register(TCP_FY_ACTUAL_OUTPUT_DOUBLE,0)
write_output_float_register(TCP_FZ_ACTUAL_OUTPUT_DOUBLE,0)
write_output_float_register(TCP_MX_ACTUAL_OUTPUT_DOUBLE,0)
write_output_float_register(TCP_MY_ACTUAL_OUTPUT_DOUBLE,0)
write_output_float_register(TCP_MZ_ACTUAL_OUTPUT_DOUBLE,0)
[...]
```

The first part defines six global variables corresponding to a number including between 0 and 5. Then according with the URScript language initializes the first six output float register at 0. In an other part of the script it is possible to find where the values in the register are updated after the initialization.

2F-85 Gripper

2F-85 Gripper is a two finger adaptive robot gripper with a maximum opening of 85mm. There is a bigger version called 2F-140 with a maximum opening of 140mm. The two grippers have the same software and base connection, it changes only the maximum opening of the gripper.

The software for control has not been studied. It will be for sure a point to study in detail.

6 – Codes

In order to achieve the goal of the thesis, two different program languages are used on the computer.

At first Matlab was chosen because it is easy to program and vastly used in Politecnico of Turin, but its nature of program for projects complicates works with bytes and interconnections. In a second moment I changed to Phyton because Universal Robot provides some codes written in this language. Being Python was born as a program language, it has a lot of its own libraries to read/write on sockets and manipulate bytes. With the knowledge learned using Python, it was easier to rewrite the same codes using Matlab.

Have the codes in Matlab is necessary because it is really useful in the designing and testing phase of our project.

Both the languages are interpreted but can be compiled in a second time.

On the robot side both URCap and URScript are used. After the presentation of the libraries, there is also a presentation of the URCap of support used and some rules on how to load and to use the URScript from the computer.

Matlab

Matlab is a software provided by MathWorks for engineer projects with a lot of scientific libraries. The Politecnico of Turin provides a free license to all its students.

Extra informations and download page can be found at [1]

For this thesis Matlab 2017a was used.

Python

Python is a open source program language with a huge community that writes libraries and gives information or solutions in dedicated forum.

Extra information and download page can be found at [2].

From the site it is possible download its own original IDE that is little more than a notepad program.

It is also possible download, always free, from external sites more complex IDE with a lot of useful toolboxes for the programming.

For my purpose I use an scientific IDE called Spyder that can be found inside Anaconda suite. All extra information and download page can be found at [3].

This IDE has the advantage of a lot of pre-installed scientific libraries, several tools for improving the debugging and a graphical interface similar to the Matlab one.

For this thesis Python 3.6 and Spyder 3.2.6 was used.

UR Library

This library was born to help the connection between the UR robot and the computer. Below are listed the functions in the library. Some of them are collected under a unique description for simplicity.

Group	Function	Description	Matlab	Python
Connection function to client interface:	connectURCI	Connects the computer to a generic UR client interface.	x	x
	connectPCI	Connects the computer to the PCI.	x	x
	connectSCI	Connects the computer to the SCI.	x	x
	connectRTCI	Connects the computer to the RTCI.	x	x
	connectRTDE	Connects the computer to the RTDE.	x	x
	connectAS2C	Using the robot as server connects the computer to the robot.	x	x

	streamRTCI	Read the data provided from the RTCI port.	x	x
	renameRTCI	Rename the data read with streamRTCI.	x	x
Command function:	movec	Call the movec URScript function on the robot.	x	x
	movej	Call the movej URScript function on the robot.	x	x
	movel	Call the movel URScript function on the robot.	x	x
	movep	Call the movep URScript function on the robot.	x	x
	speedj	Call the speedj URScript function on the robot.	x	x
	speedl	Call the speedl URScript function on the robot.	x	x
	stopj	Call the stopj URScript function on the robot.	x	x
	stopl	Call the stopl URScript function on the robot.	x	x
	halt	Call the halt URScript function on the robot.	x	x
	MMF	Create a .txt file.		x
	csvF	Create a .csv file.		x
RTDE functions:	RTDE_REQUEST_PROTOCOL_VERSION	Select the protocol version for the RTDE port.	x	x
	RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS	Set the desired output provided by the RTDE port.	x	x
	RTDE_CONTROL_PACKAGE_START	Start the straming of data from RTDE port.	x	x
	RTDE_CONTROL_PACKAGE_PAUSE	Pause the straming of data from RTDE port.	x	x
	RTDE_DATA_PACKAGE	Read the streaming of data provided by the RTDE port.	x	x
	info	Gives information on the library.		x

Robotiq Library

This short library was created to read and rename the data from force and torque sensor called FT 300 produced by Robotiq.

The sensor saves the measured data in the internal float register of the robot, so a connection to RTDE port using UR library is necessary.

The following function are implemented:

Function	Description	Matlab	Python
SET_RTDE_OUTPUT_FT_SE	Set the desired output provided by the RTDE	x	x

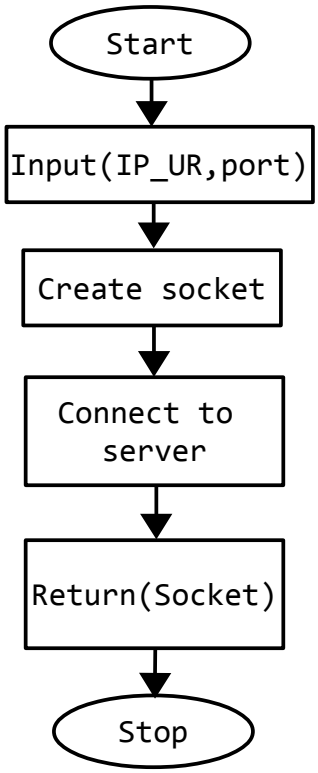
NSOR	port according to the data of the FT 300 sensor.		
rename_FORCE_FT_SENSOR	Rename the data read with SET_RTDE_OUTPUT_FT_SENSOR	x	x

Library in Matlab-UR
connectURCI

Description

ConnectURCI abbreviation of connection to UR Client Interface is an easy function that connects the computer as client to the ports provided from the robot UR that work as server with a socket based on TCP/IPv4 protocol .

Flow chart



Variables

Input		
Variables	Type	Description
IP_UR	String	The IPv4 of UR that identify its address in the local network
Port	Integer	The port provided by the robot for the required service

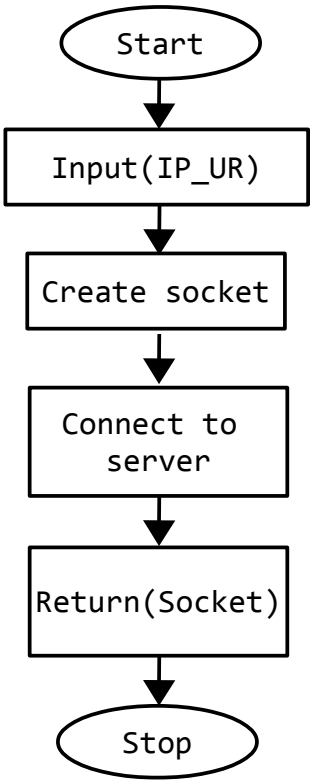
Output		
Variables	Type	Description
Socket (URCI)	Object	The socket of the server (robot).

connectPCI, connectSCI, connectRTCI, connectRTDE

Description

These four functions have the same structure. They open a socket with a defined port.
-connectPCI, connection to Primary Client Interface, port 30001
-connectSCI, connection to Secondary Client Interface, port 30002
-connectRTCI, connection to Real-Time Client Interface, port 30003
-connectRTDE connection to Real-Time Data exchange, port 30004

Flow chart



Variables

Input		
Variables	Type	Description
IP_UR	String	The IPv4 of UR that identify its address in the local network

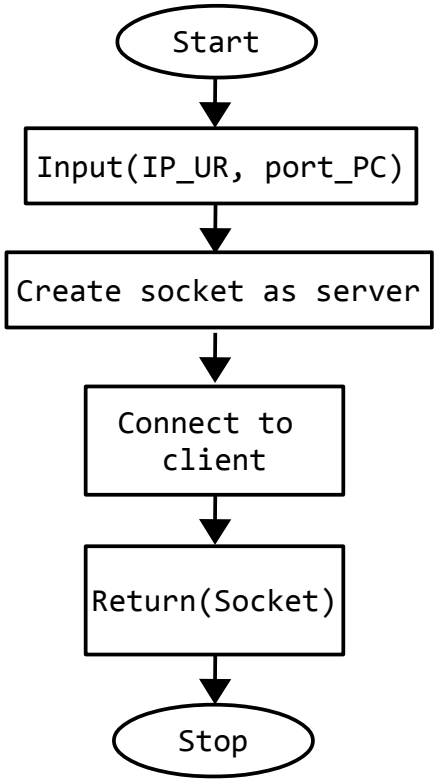
Output		
Variables	Type	Description
Socket	Object	The socket of the server (robot).

ConnectAS2C

Description

ConnectAS2C abbreviation of connection As Server to Client is an easy function that connects with a socket based on TCP/IPv4 protocol the computer as server to the robot UR that works as client. In this case a corresponding code in URCap on the robot is necessary.

Flow chart



Variables

Input		
Variables	Type	Description
IP_UR	String	The IPv4 the robot that will be connected to the computer (Server)
Port_PC	Integer	Port decided by programmers at which robot must be connected.

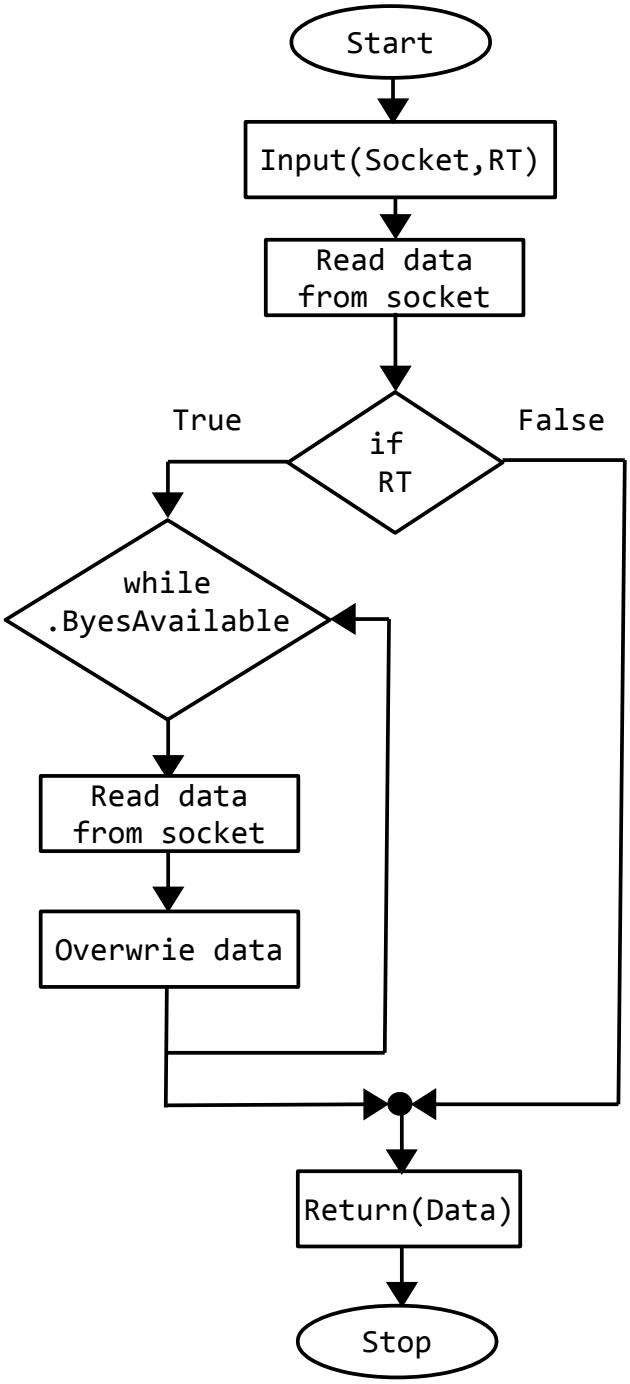
Output		
Variables	Type	Description
S_Client	Object	The socket of the client (robot).

streamRTCI

Description

StreamRTCI abbreviation of streaming Real-Time Client Interface read data provided by RTCIport.
As all received data from sockets, also these must be converted from byte to values. In this case after the first four byte the other are read in blocks of 8 as double with big endian convention.

Flow chart



Variables

Input		
Variables	Type	Description
Socket RTCI	Object	The function must know from which socket to read
RT	string	If the option input “RT” is added the function will control if newer data are available.

Output		
Variables	Type	Description
Data	132*double	A vector of 132 double is provided

Comments

The RT optional functionality is important because Matlab use a lot of computational power and sometimes the computer is not able to do a cicle in 8ms. In this case the RT option control if there are available newer data and overwrite the older. This is really important for avoid growing queue of data that will cause increasing delay on data acquisition.

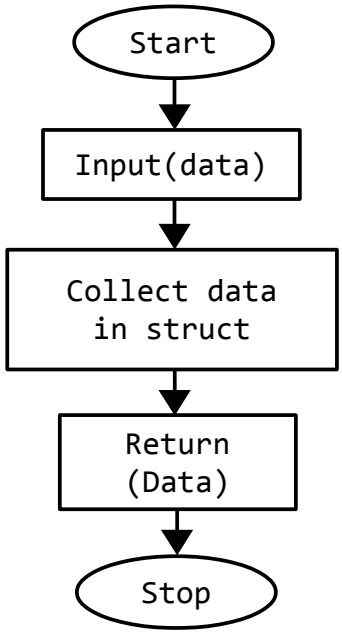
renameRTCI

Description

RenameRTCI abbreviation of rename Real-Time Client Interface is a function that divides the data provided from streamRTCI inside a dictionary (Python) or a struct (Matlab). A dictionary/struct is a collection of different types of data each bound with a descriptive name.

The assignment of the name at the variables is given using an exel file provided by Universal Robot.

Flow chart



Variables

Input		
Variables	Type	Description
Data	132xdouble	Must be the data provided by streamRTCI

Output		
Variables	Type	Description
Datar	Struct	Renamed data according to Universal Robot excel file

movec, movej, movel, movep, speedj, speedl, stopj, stopl, halt

Description

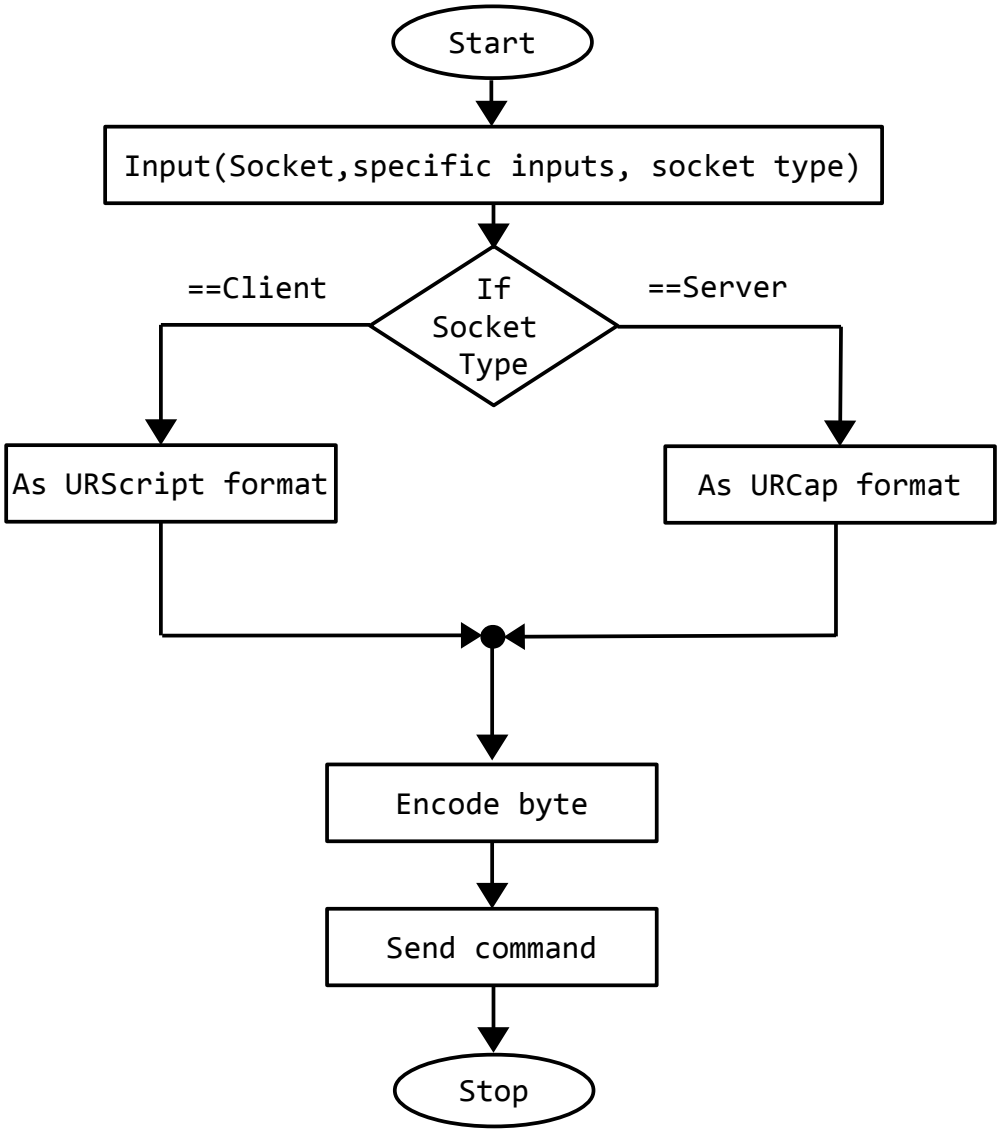
The functions movec, movej, movel, movep, speedj, speedl, stopj, stopl and halt are extensions of the corrisponding URScript provided by Universal Robot.

The URScript description of these functions and others can be read in the attached .pdf file.

The named functions have been created to provide the same functionalities of URScript from the computer. As explained in chapter 4 the PCI, SCI and RTCI are able to receive URScript command via socket. So the functions takes as input the same description in the UR pdf file and write and send the corresponding URScript code to the robot. In this case SocketType is already set as “CI”

In alternative, a different approach is possible. Using the support URCap provided, it is possible to command the robot with these functions using the robot as server (connectAS2C). In this case must be add as input “SocketType” and ”Server”.

Flow chart



Variables

movec

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
pose_via	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
pose_to	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
a	doble	[m/s^2]	As URScript manual
v	double	[m/s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

movej

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
q (angles)	6xdouble	[rad, rad, rad, rad, rad, rad]	As URScript manual
a	double	[rad/s^2]	As URScript manual
v	double	[rad/s]	As URScript manual
t (optional)	double	[s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

movel

Input			
Variables	Type	Physical dimension	Description

Socket	Object	none	The function must know in which socket writes
pose	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
a	double	[m/s^2]	As URScript manual
v	double	[m/s]	As URScript manual
t (optional)	double	[s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

movep

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
pose	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
a	double	[m/s^2]	As URScript manual
v	double	[m/s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

speedj

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
speed	6xdouble	[rad/s, rad/s, rad/s, rad/s, rad/s, rad/s]	As URScript manual
a	double	[rad/s^2]	As URScript manual
t (optional)	double	[s]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command

			to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

speedl

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
speed	6xdouble	[m/s, m/s, m/s, m/s, m/s, m/s]	As URScript manual
a	double	[m/s^2]	As URScript manual
t (optional)	double	[s]	As URScript manual
aRot (optional)	Double or char	[rad/s^2 or none]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

stopj

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
a	double	[rad/s^2]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

stopl

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
a	double	[m/s^2]	As URScript manual

aRot (optional)	Double/ or char	[rad/s^2 or none]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

halt

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

Comment

In these functions many inputs are optional. To change the default value it is necessary put as input the couple “name_value”, “value”.

As example if I want change the socket type of stopj I will must write:

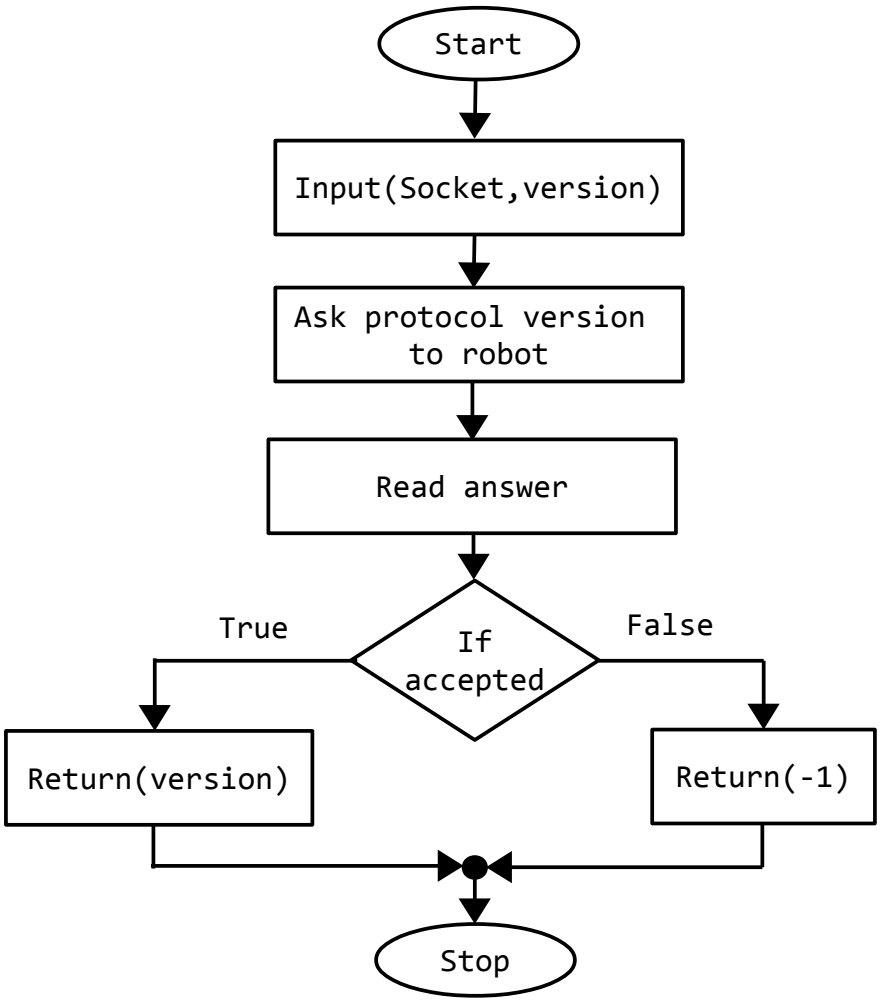
```
stopj(AS2C,1.4,”SocketType”,”Server”)
```

RTDE_REQUEST_PROTOCOL_VERSION

Description

The RTDE port can use two protocol versions. The two change in the input request and response provide for the same protocol.
This function asks to the robot to use the version that we want, 1 or 2.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
Version	Integer	The version to use, 1 or 2. 2 as default.

Output		
Variables	Type	Description

Version	Integer	If the request is accepted the function respond with the version asked.
Error	Integer	If error occurs the output will be -1.

Comments

It is not necessary to select the protocol version but the robot uses as default version 1 while version 2 gives more information. So as default this function impose version 2.

RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS

Description

This function is necessary to set the output of the RTDE port. Indeed without setup, the RTDE port will not provide any data.

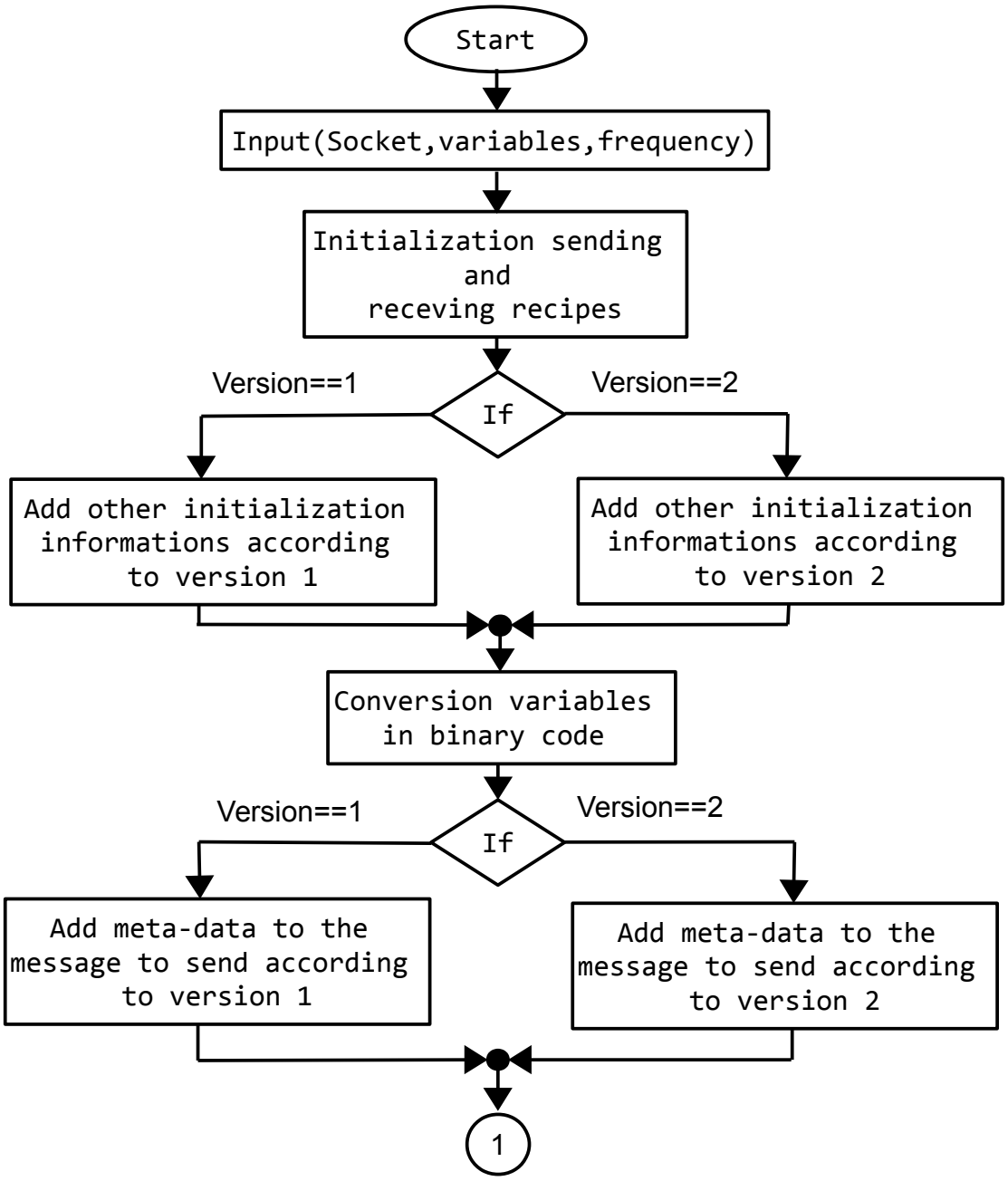
After the settings the output will provides the data requested in the order requested with a refresh of 125Hz for version 1 or required refresh for version 2.

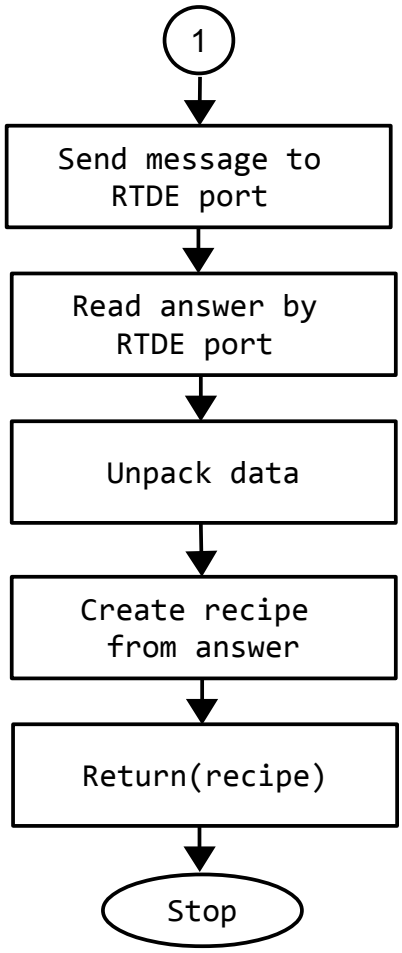
The information about allowed output can be founded in [9].

As default this function selects version 2 and refresh frequency at 125Hz.

As answer the robot will provide a recipe of how the data will have to be read.

Flow Chart





Variables

Input		
Variables	Type	Description
Socket(RTDE)	Object	The function must know where to send the request. Only RTDE port is valid.
Variables	List of strings	In according with the information provided by Universal Robot it is possible to send a list of data that we need to know in real-time by RTDE.
Version	Integer	The version to use, 1 or 2. 2 as default.
Frequency	Integer	If the version 2 has been selected, it is possible define the refresh frequency. 125Hz as default.

Output		
Variables	Type	Description
Output_recipe_id	Integer	Number corresponding to data output configuration.
Recipe	Cell 2xn	It is a cell matrix where the first row takes the name of the types and the second row takes the number of that kind of type.

Comments

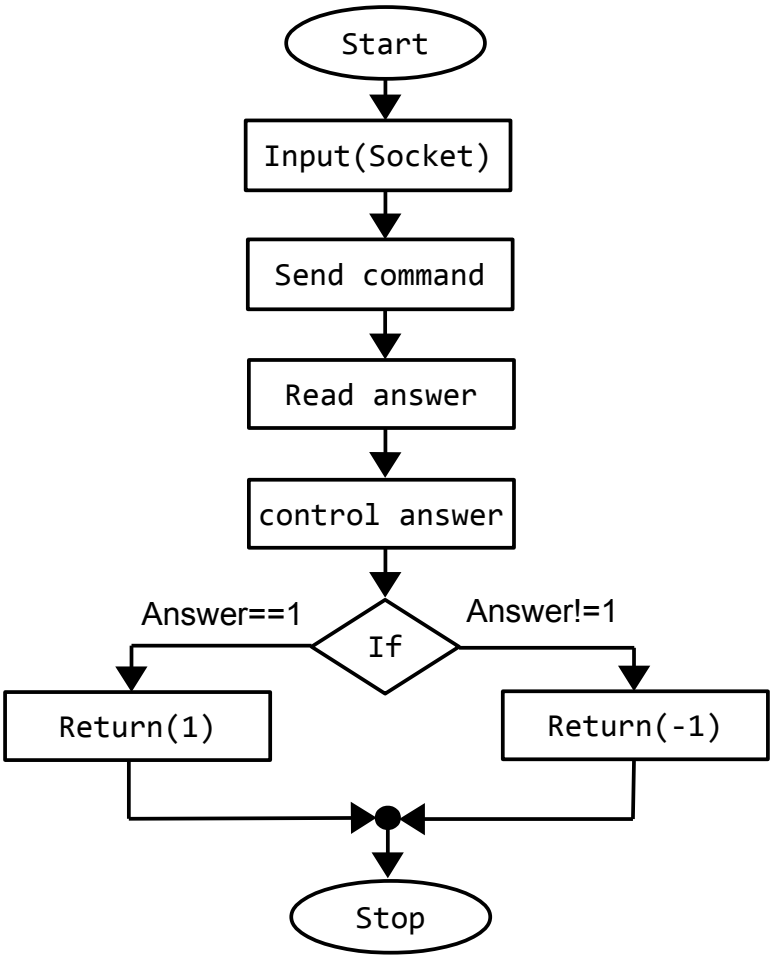
After the set of the output the streaming of data will not start till the RTDE_CONTROL_PACKAGE_START function calls.

RTDE_CONTROL_PACKAGE_START

Description

After the setting of the output, and if necessary of the version, the streaming of the data will not start untill this function is called.

Flow chart



Variables

Input		
Variables	Type	Description
Socket (RTDE)	Object	The function must know where to send the request. Only RTDE port is valid.

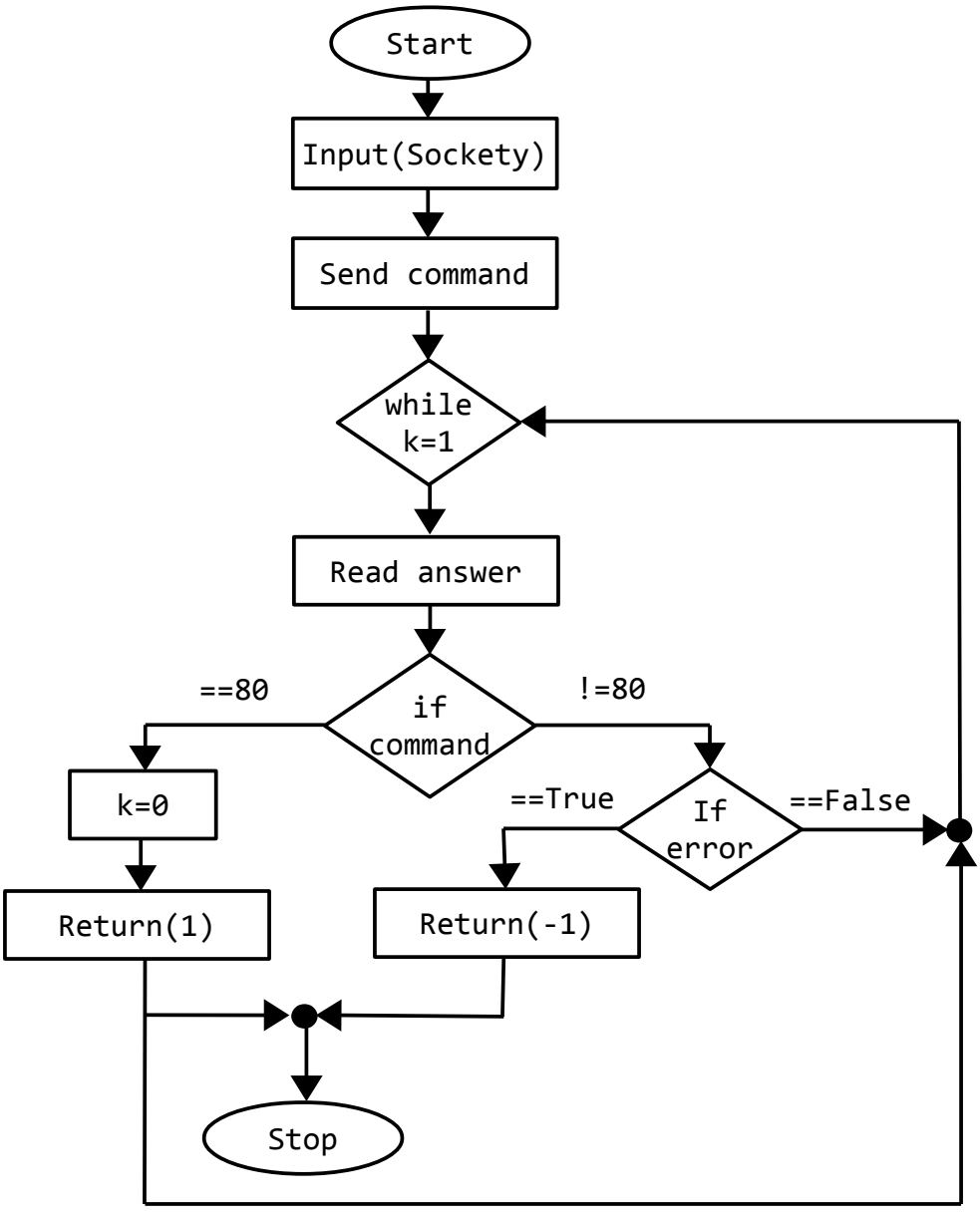
Output		
Variables	Type	Description
Start- return(1)	Integer	The RTDE port starts the streaming of data in the way asked in RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS.
Error- return(-1)	Integer	Some problem occur. No streaming of data.

RTDE_CONTROL_PACKAGE_PAUSE

Description

The streaming of the data can be put in pause in any moment using this function.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.

Output		
Variables	Type	Description
Start- return(1)	Integer	The RTDE port stops the streaming of data.
Error- return(-1)	Integer	Some problem occur. It is not possible put in pause the streaming of data.

Comment

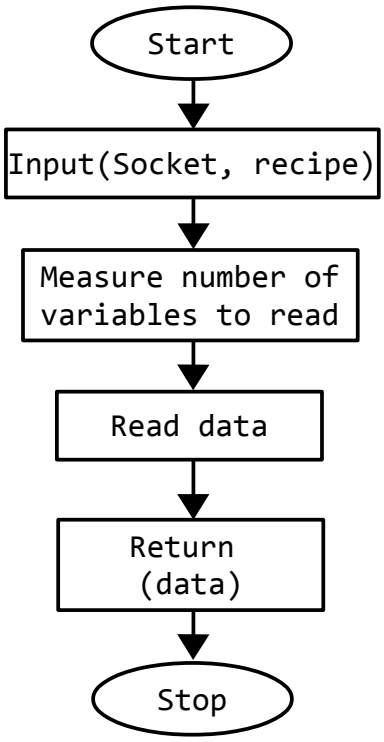
After the pause command is not necessary do again the initialization, the start command is enough. Cause the computational power used by Matlab a cycle to cancel extra data in the socket queue is necessary. This is done by the while command.

RTDE_DATA_PACKAGE

Description

Thanks to the RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS output reads and unpacks the byte in the correct values.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
recipe	Cell 2xn	For works correctly this function needs the output provided by RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS.

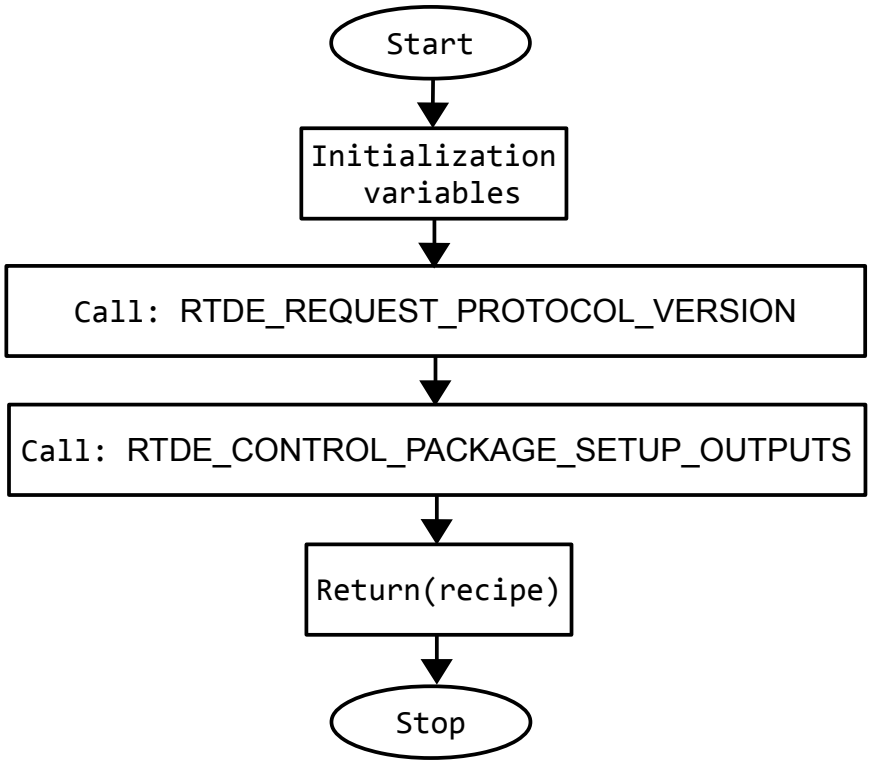
Output		
Variables	Type	Description
Data	Cell 1xm	The received data after the unpacking create a list of different kind of data corresponding to the information provided by Universal Robot.

Library in Matlab-ROBOTIQ
SET_RTDE_OUTPUT_FT_SENSOR

Description

This function uses the UR library to set the output of the RTDE port to read the 6 axis force-torque data saved in the “output_double_register_X”, with X from 0 to 5.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
frequency	integer	The refresh frequency. 125Hz as default.

Output		
Variables	Type	Description
Output_recipe_id	Integer	Number corresponding to data output configuration.
Recipe	Cell 2x6	Cell composed by elements that correspond to the format to unpack the streaming of data. In this case, six double.

Comment

For the correct operation, as explained in chapter 4 , is necessary:
-to open a connection with RTDE client using connectRTDE.

-use this function for set RTDE output.

-to start the data streaming with RTDE_CONTROL_PACKAGE_START

-to read with streamRTDE

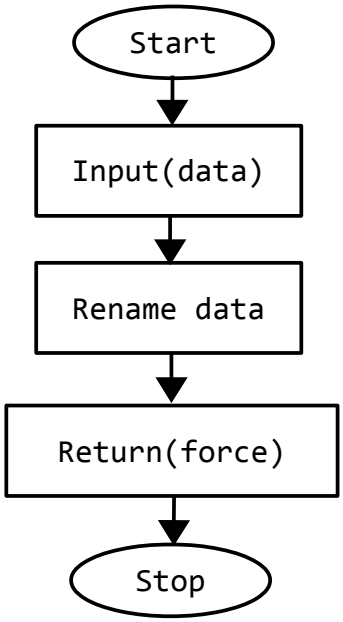
After this passages it is possible if desired to put the data in a struct with rename_FORCE_FT_SENSOR.

rename_FORCE_FT_SENSOR

Description

This function renames the data provided by streamRTDE after the correct setup with SET_RTDE_OUTPUT_FT_SENSOR.

Flow chart



Variables

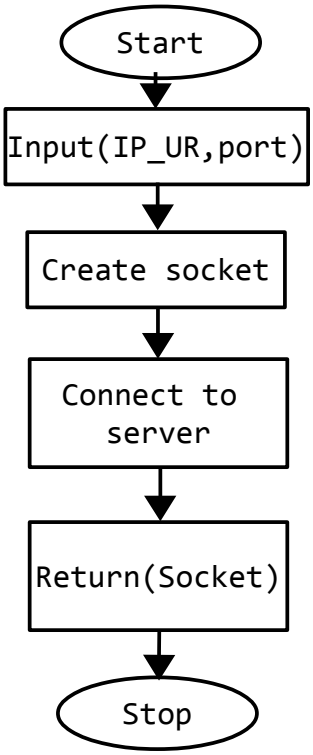
Input		
Variables	Type	Description
Data	Cell 1x6	Data from “output_double_register_X”, with X from 0 to 5

Output		
Variables	Type	Description
Force-torque	Struct	The sensor data are bounded with its own name FX, FY, FZ, MX, MY, MZ

Library in Python-UR
connectURCI

Description
ConnectURCI abbreviation of connection to UR Client Interface is an easy function that connects the computer as client to the ports provided from the robot UR that works as server with a socket based on TCP/IPv4 protocol .

Flow chart



Variables

Input		
Variables	Type	Description
IP_UR	String	The IPv4 of UR that identify its address in the local network
Port	Integer	The port provided by the robot for the required service

Output		
Variables	Type	Description
Socket	Object	The socket of the server (Robot).

connectPCI, connectSCI, connectRTCI, connectRTDE

Description

These four functions have the same structure and are all based on connectURCI. They call the function setting automatically the corresponding port.

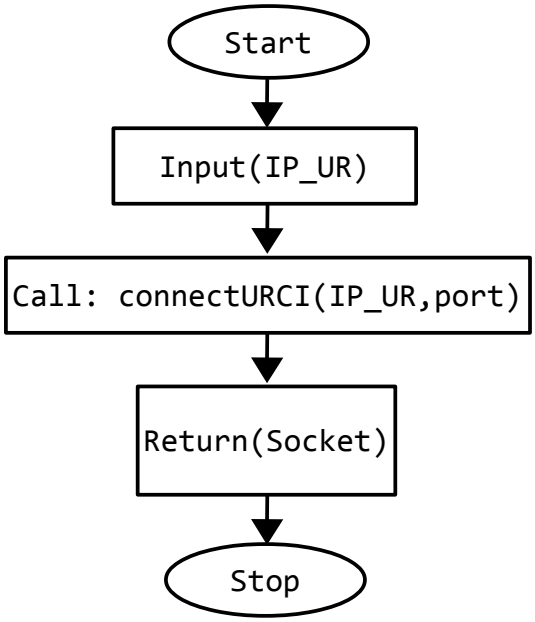
-connectPCI, connection to Primary Client Interface, port 30001

-connectSCI, connection to Secondary Client Interface, port 30002

-connectRTCI, connection to Real-Time Client Interface, port 30003

-connectRTDE connection to Real-Time Data exchance, port 30004

Flow chart



Variables

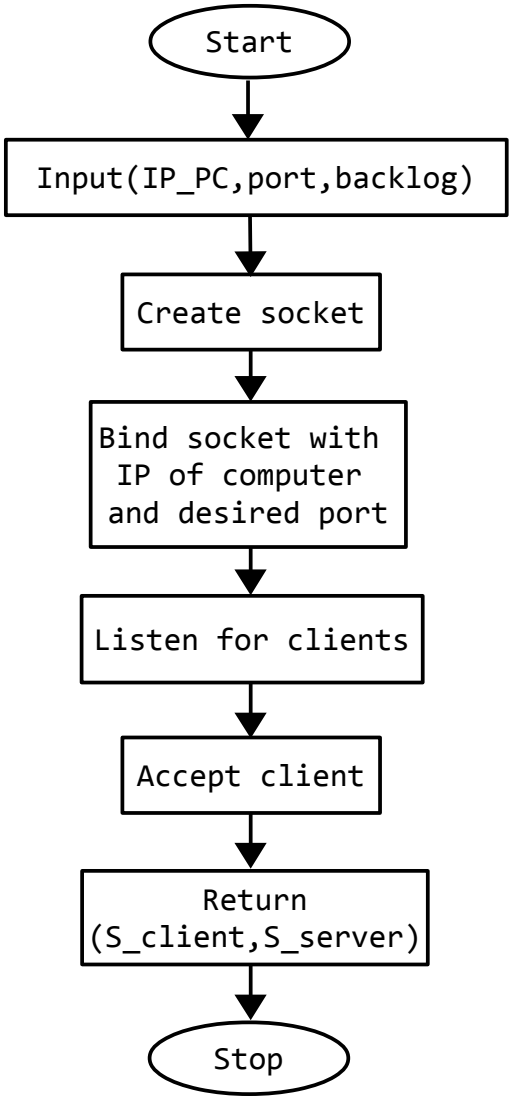
Input		
Variables	Type	Description
IP_UR	String	The IPv4 of UR that identify its address in the local network

Output		
Variables	Type	Description
Socket	Object	The socket of the sever (robot).

ConnectAS2C

ConnectAS2C abbreviation of connection As Server to Client is an easy function that connects with a socket based on TCP/IPv4 protocol the computer as server to the robot UR that works as client. In this case a corresponding code in URCap on the robot is necessary.

Flow chart



Variables

Input		
Variables	Type	Description
IP_PC	String	The IPv4 of computer that identify its address in the local network
Port	Integer	Port decided by programmers at which robot must be connected.
Backlog	Integer	Number of clients accepted by computer. 1 as default.

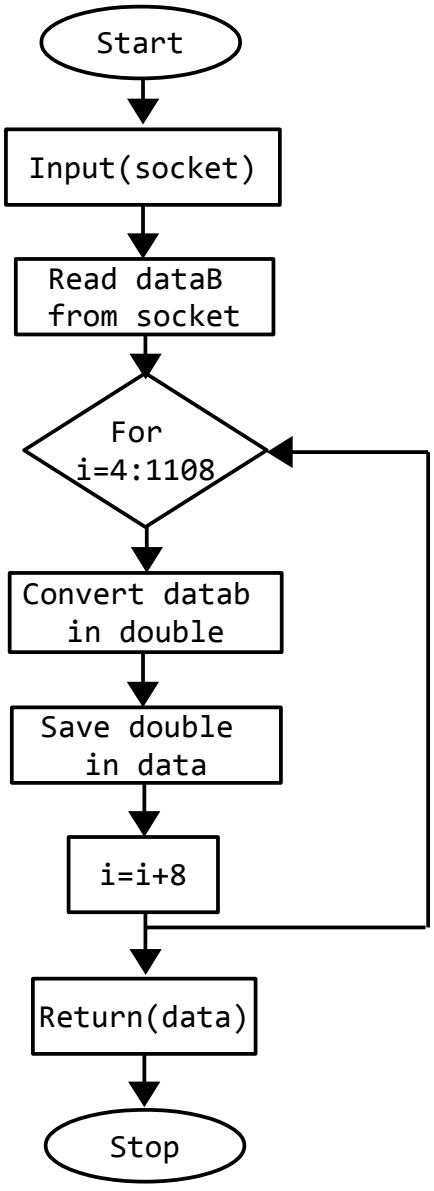
Output		
Variables	Type	Description
S_Client	Object	The socket of the client (robot).
S_Server	Object	The socket of the server (computer).

streamRTCI

Description

StreamRTCI abbreviation of streaming Real-Time Client Interface reads data provided by RTCI port.
As all received data from sockets, also these must be converted from byte to values. In this case after the first four byte the other are read in blocks of 8 as double with big endian convention.

Flow chart



Variables

Input		
Variables	Type	Description
Socket RTCI	Object	The function must know from which socket to read

Output		
Variables	Type	Description
Data	132*double	A vector of 132 double is provided

Comments

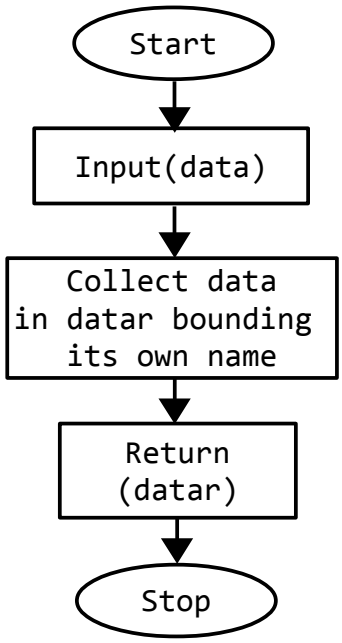
The for-cycle in the flow chart that is present in Python does not appear in Matlab because it is done by one dedicated Matlab function.

renameRTCI

Description

RenameRTCI abbreviation of rename Real-Time Client Interface is a function that divides the data provided from streamRTCI inside a dictionary. A dictionary is a collection of different types of data each bound with a descriptive name.
The correct assignment of the name at the variables was done using an exel file provided by Universal Robot. [9]

Flow chart



Variables

Input		
Variables	Type	Description
Data	132xdouble	Must be the data provided by streamRTCI

Output		
Variables	Type	Description
Datar	Dictionary/struct	Renamed data according to Universal Robot excel file

movec, movej, movel, movep, speedj, speedl, stopj, stopl, halt

Description

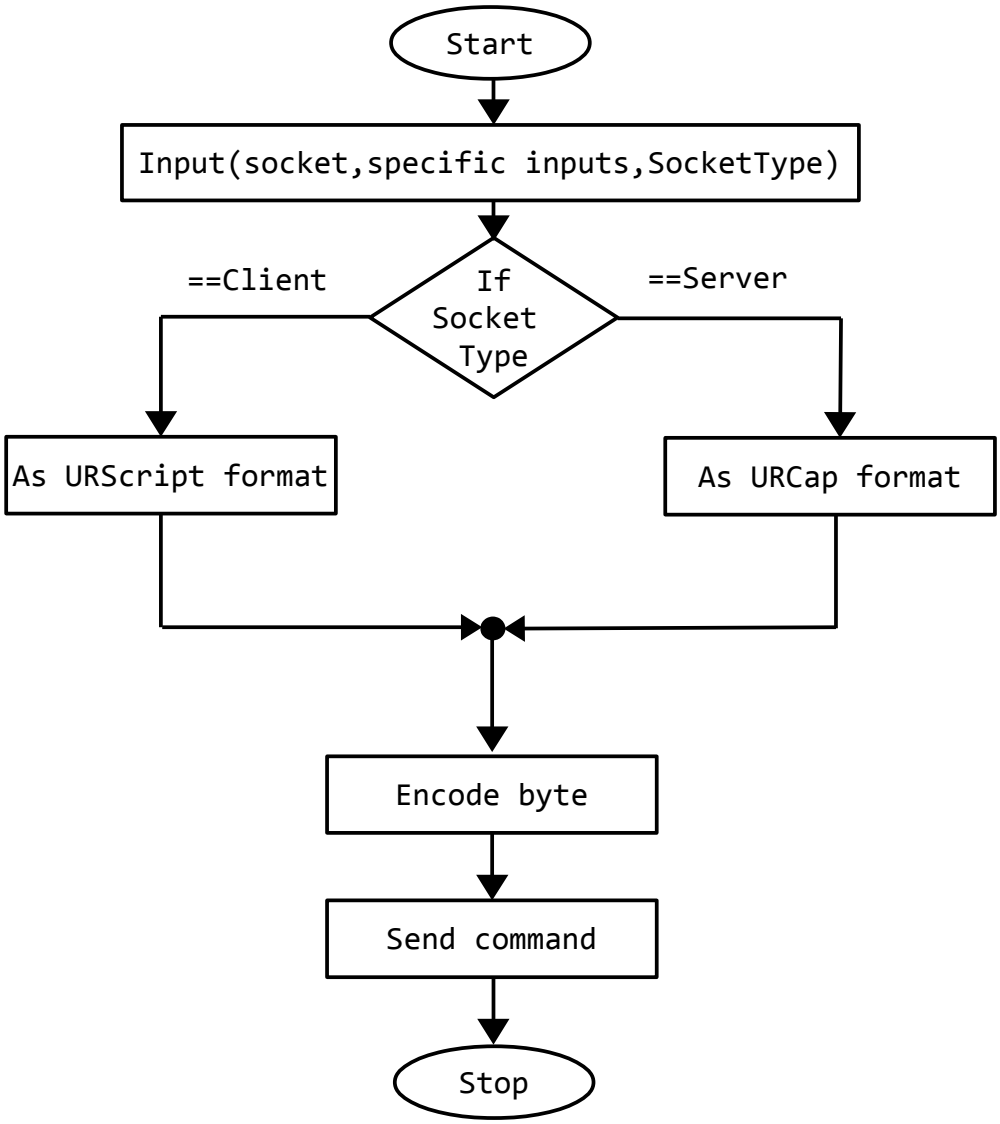
The functions movec, movej, movel, movep, speedj, speedl, stopj, stopl and halt are extensions of the corrisponding URScript provided by Universal Robot.

The URScript description of these functions and others can be read in [8]

The named functions have been created to provide the same functionalities of URScript from the computer. As explained in chapter 4 the PCI, SCI and RTCI are able to receive URScript command via socket. So the functions takes as input the same description in the UR pdf file and write and send the corresponding URScript code to the robot. In this case SocketType is already set as “CI”

In alternative, a different approach is possible. Using the support URCap provided, it is possible to command the robot with these functions using the robot as server (connectAS2C). In this case must be add as input ”Server”.

Flow chart



Variables

movec

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
pose_via	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
pose_to	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
a	doble	[m/s^2]	As URScript manual
v	double	[m/s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

movej

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
q (angles)	6xdouble	[rad, rad, rad, rad, rad, rad]	As URScript manual
a	double	[rad/s^2]	As URScript manual
v	double	[rad/s]	As URScript manual
t (optional)	double	[s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

movel

Input			
Variables	Type	Physical dimension	Description

Socket	Object	none	The function must know in which socket writes
pose	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
a	double	[m/s^2]	As URScript manual
v	double	[m/s]	As URScript manual
t (optional)	double	[s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

movep

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
pose	6xdouble	[m, m, m, rad, rad, rad]	As URScript manual
a	double	[m/s^2]	As URScript manual
v	double	[m/s]	As URScript manual
r (optional)	double	[m]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

speedj

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
speed	6xdouble	[rad/s, rad/s, rad/s, rad/s, rad/s, rad/s]	As URScript manual
a	double	[rad/s^2]	As URScript manual
t (optional)	double	[s]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command

			to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

speedl

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
speed	6xdouble	[m/s, m/s, m/s, m/s, m/s, m/s]	As URScript manual
a	double	[m/s^2]	As URScript manual
t (optional)	double	[s]	As URScript manual
aRot (optional)	Double or char	[rad/s^2 or none]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

stopj

Input			
Variables	Type	Physical dimension	Description
a	double	[rad/s^2]	As URScript manual
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

stopl

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
a	double	[m/s^2]	As URScript manual
aRot (optional)	Double/ or char	[rad/s^2 or none]	As URScript manual

Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

halt

Input			
Variables	Type	Physical dimension	Description
Socket	Object	none	The function must know in which socket writes
Socket_type (optional)	string	none	Knowing if the robot is the server or the client changes the format of the command to send. “CI” as default.

Output			
Variables	Type	Physical dimension	Description
none	none	none	none

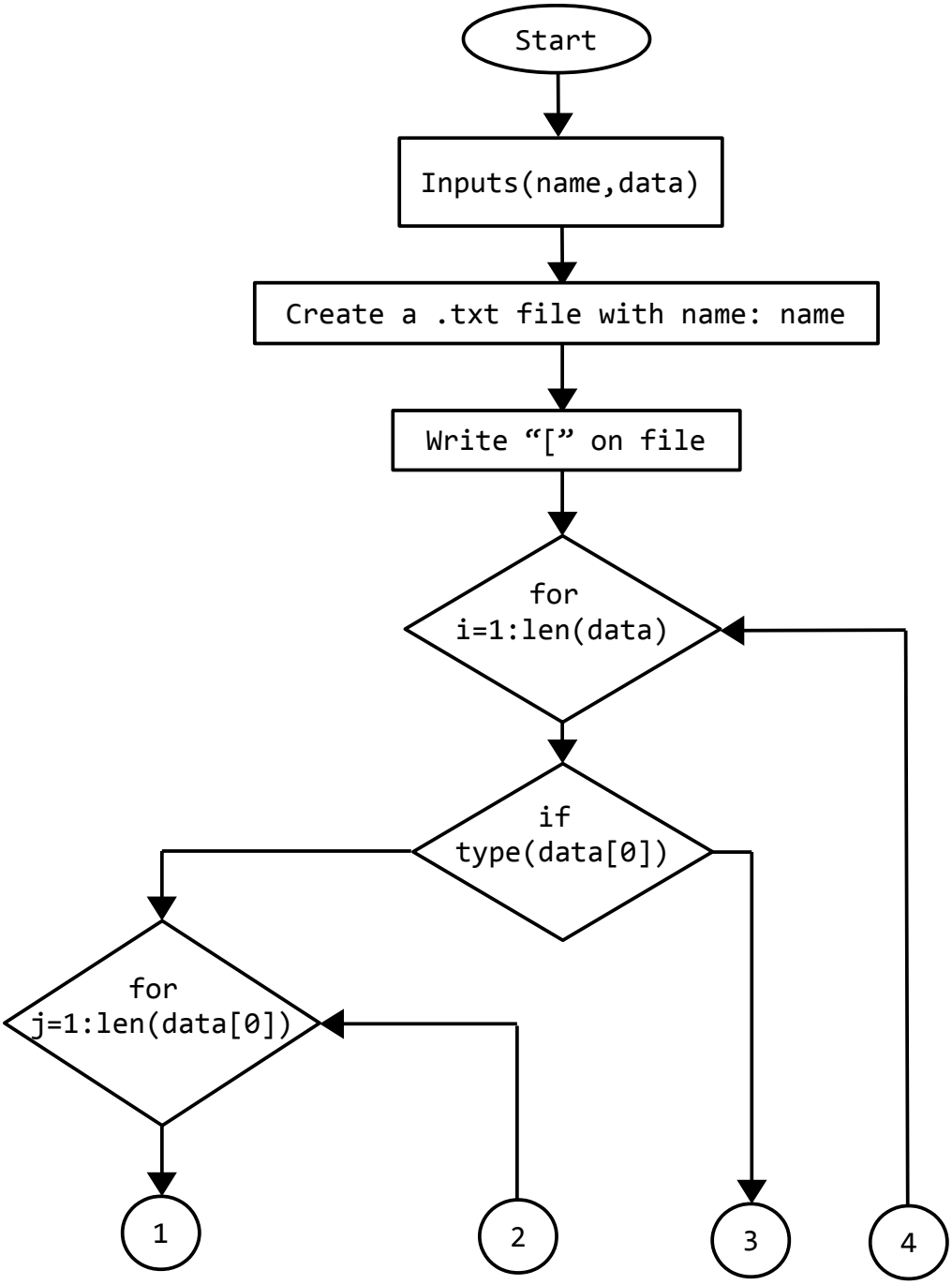
MMF

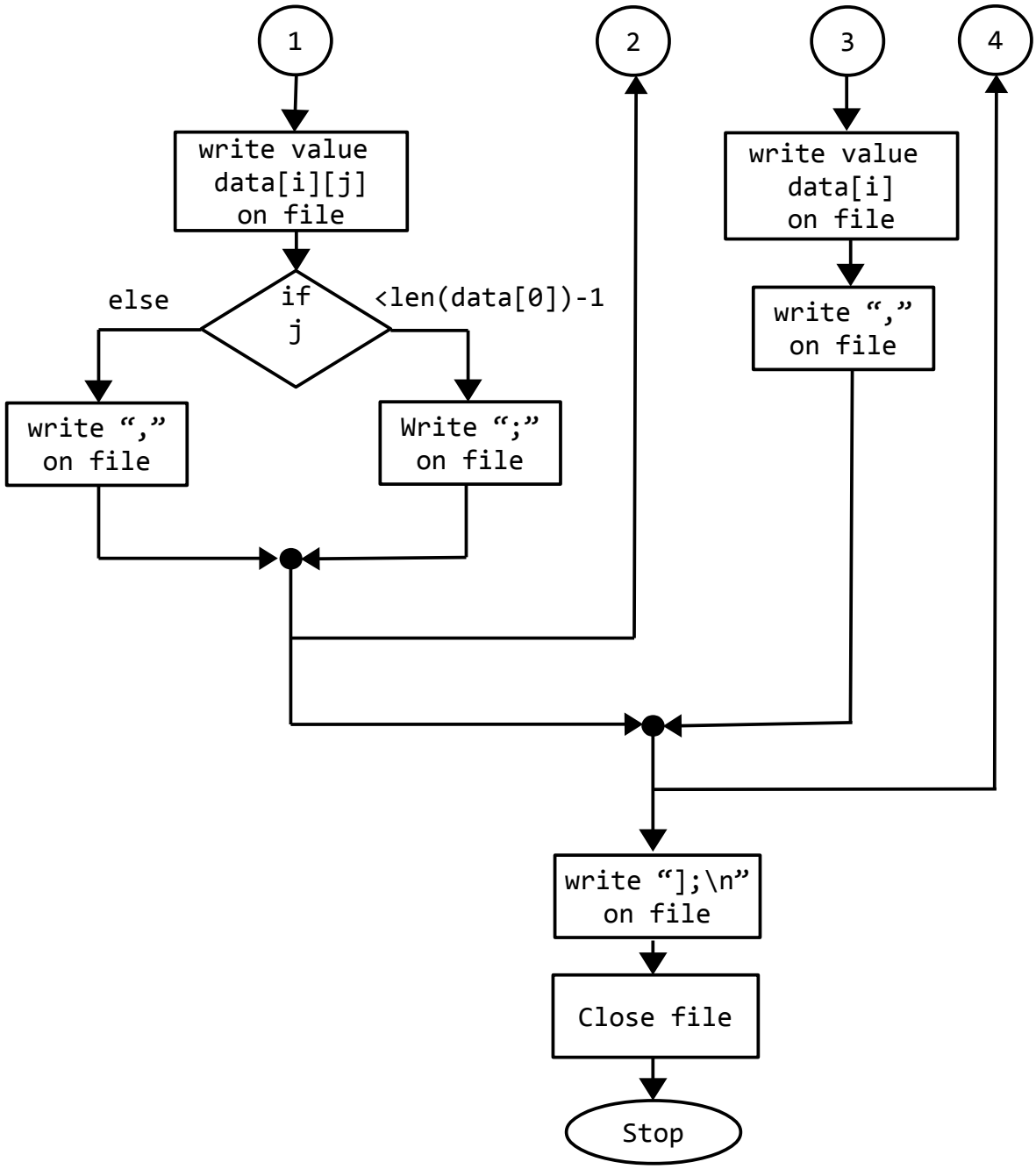
Definition

Matlab Matrix File is a function born to export the collected data in a .txt file. The data in the file are written in the same way in which are written in a matrix in matlab. Each row is divided from the next one with a ";" and each value in the row is divided with a ",". At the begin and at the end of the data two square brackets collect them "[","]".

The main idea was to export the value with this matrix to copy and paste in a second time the data from Python to Matlab. Indeed, although Python has plots functions, they are really poor respect to the Matlab one.

Flow chat





Variables

Input		
Variables	Type	Description
name	string	Name given at the .txt file
data	Double [n]x[m]	List of numeric value

Output		
Variables	Type	Description
none	none	none

Comments

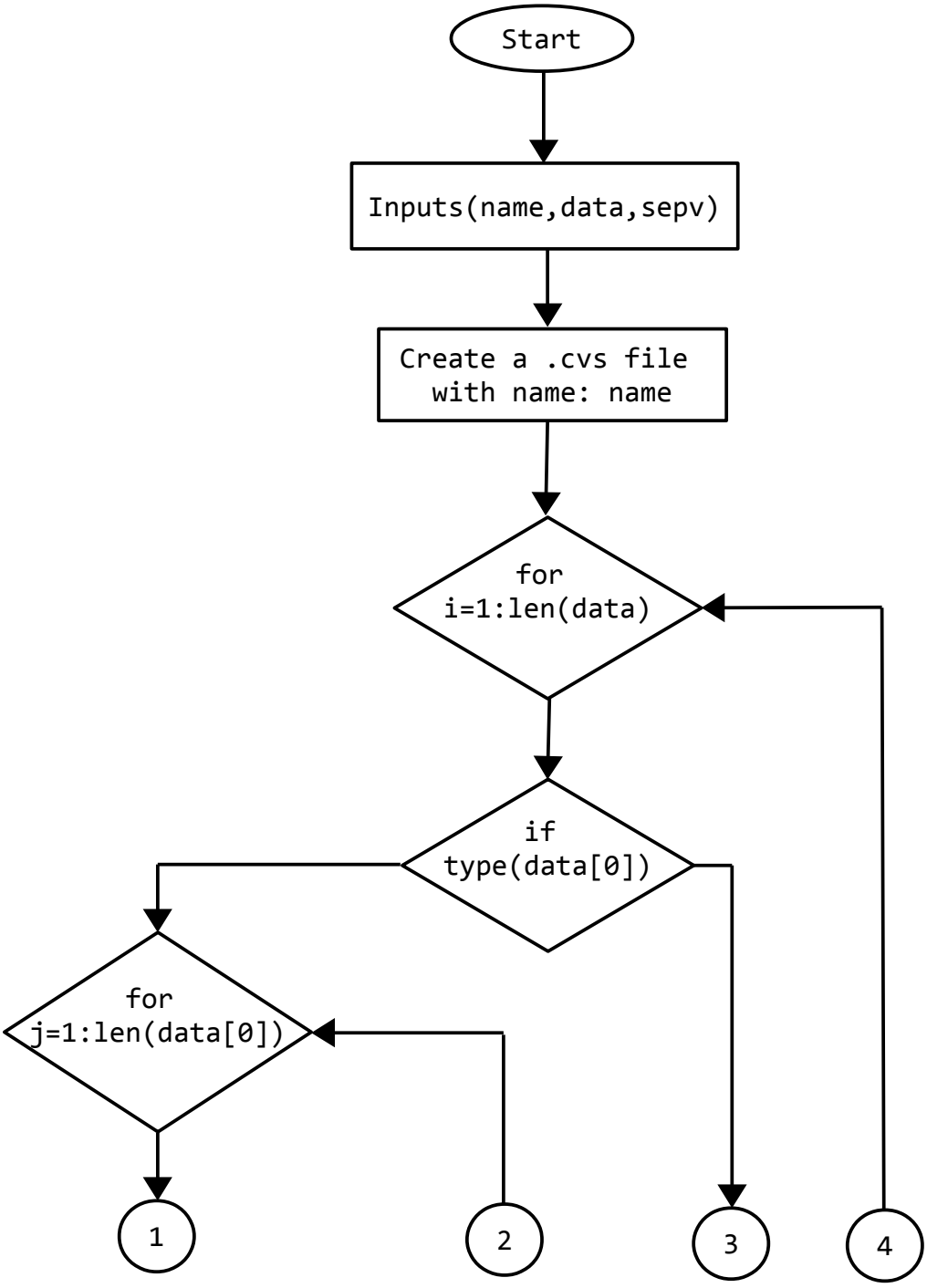
This function can be optimized in many ways but it is useless. It was born as approach to transfer data between Matlab and Python but a csv file is really more useful. So although proposed it is recommended to use csvF.

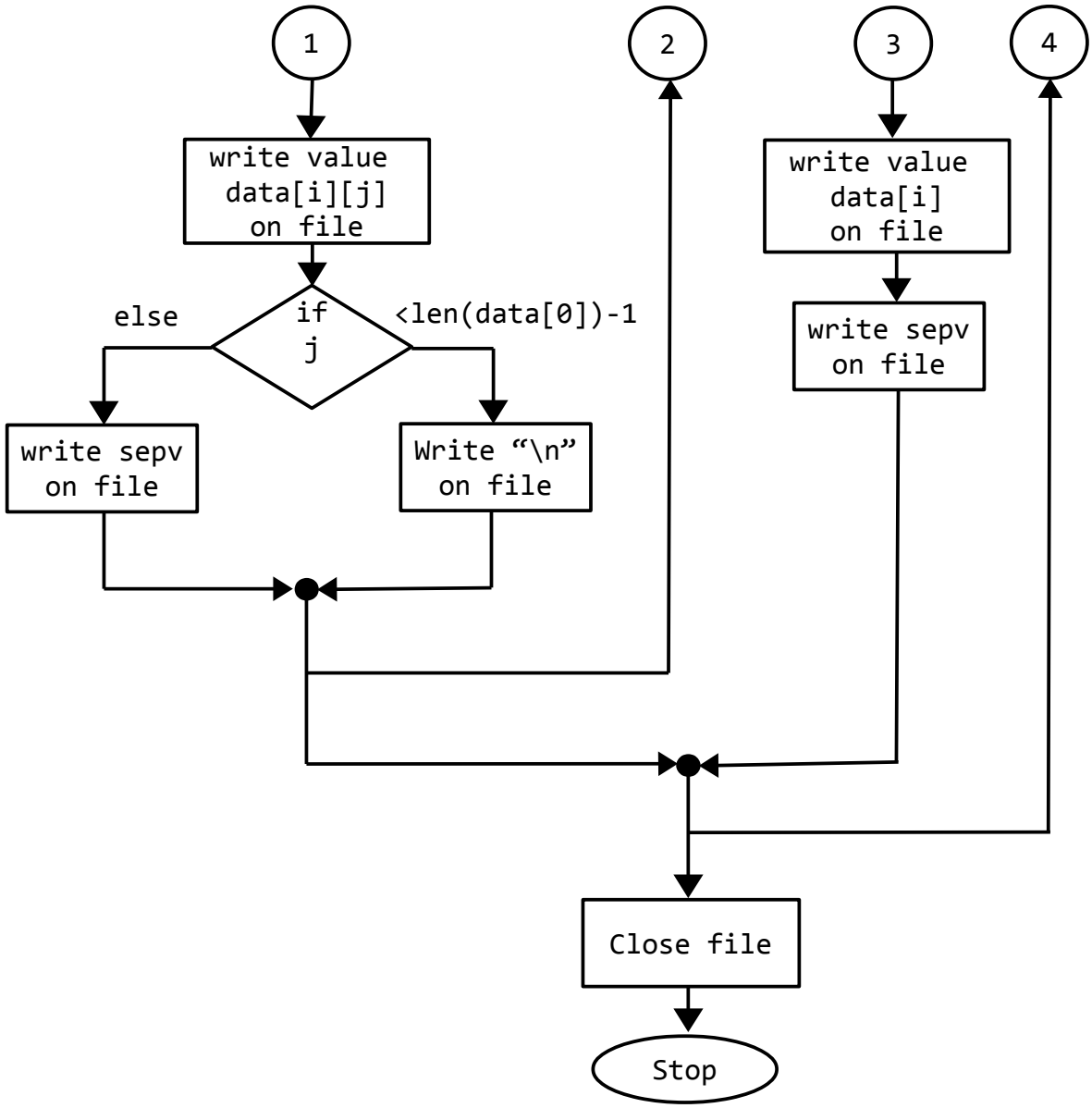
csvF

Description

Comma Separate Value File is a function that create from data a csv file. Although with a flow chart really similar to the MMF one, the two differ for the extension file that can be read both from calc program like Excell then Matlab. Saving the data in cvs is possible avoid the copy and paste process that cause many problems. Indeed the file can be open by Matlab with the command "`name=csvread("name.csv")`".

Flow chart





Variables

Input		
Variables	Type	Description
name	string	Name given at the .csv file.
data	Double [n]x[m]	List of numeric value.
sepv	char	Is the separator value. There is not a formal value. As default ",".

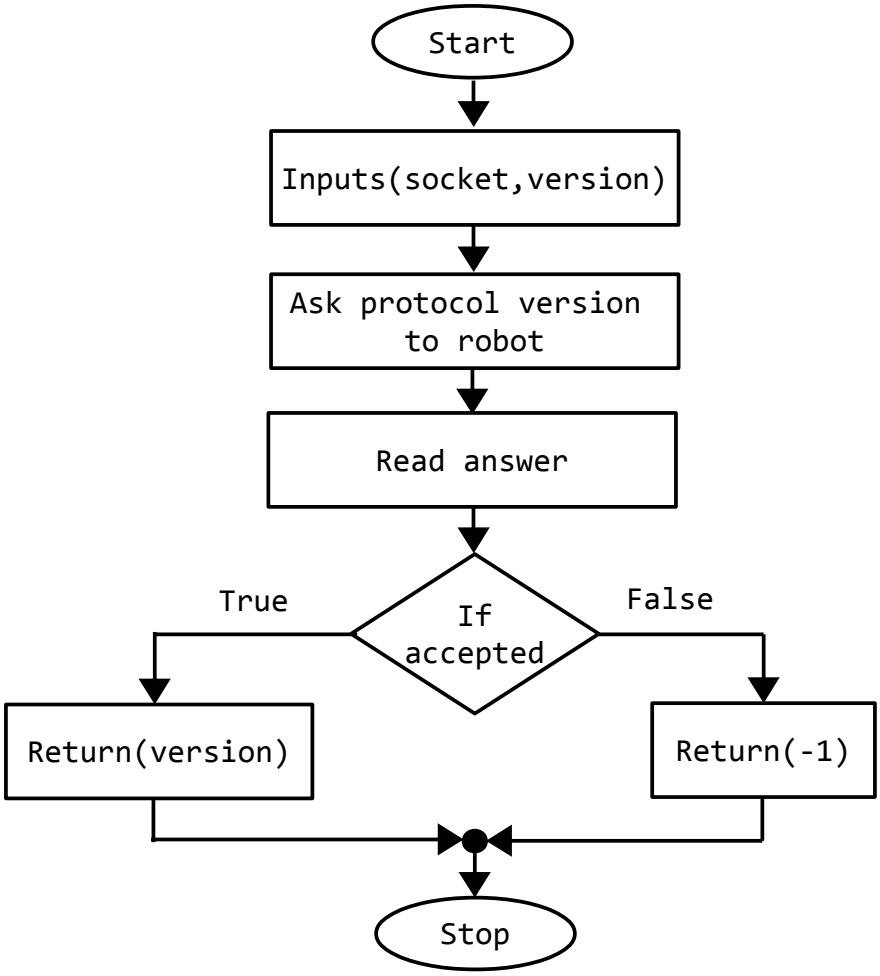
Output		
Variables	Type	Description
none	none	none

RTDE_REQUEST_PROTOCOL_VERSION

Description

The RTDE port can use two protocol versions. The two change in the input request and response provide for the same protocol.
This function asks to the robot to use the version that we want, 1 or 2.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
Version	Integer	The version to use, 1 or 2. 2 as default.

Output		
Variables	Type	Description

Version	Integer	If the request is accepted the function respond with the version asked.
Error	Integer	If error occurs the output will be -1.

Comments

It is not necessary to select the protocol version but the robot uses as default version 1 while version 2 gives more information. So as default this function impose version 2.

RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS

Description

This function is necessary to set the output of the RTDE port. Indeed without setup, the RTDE port will not provide any data.

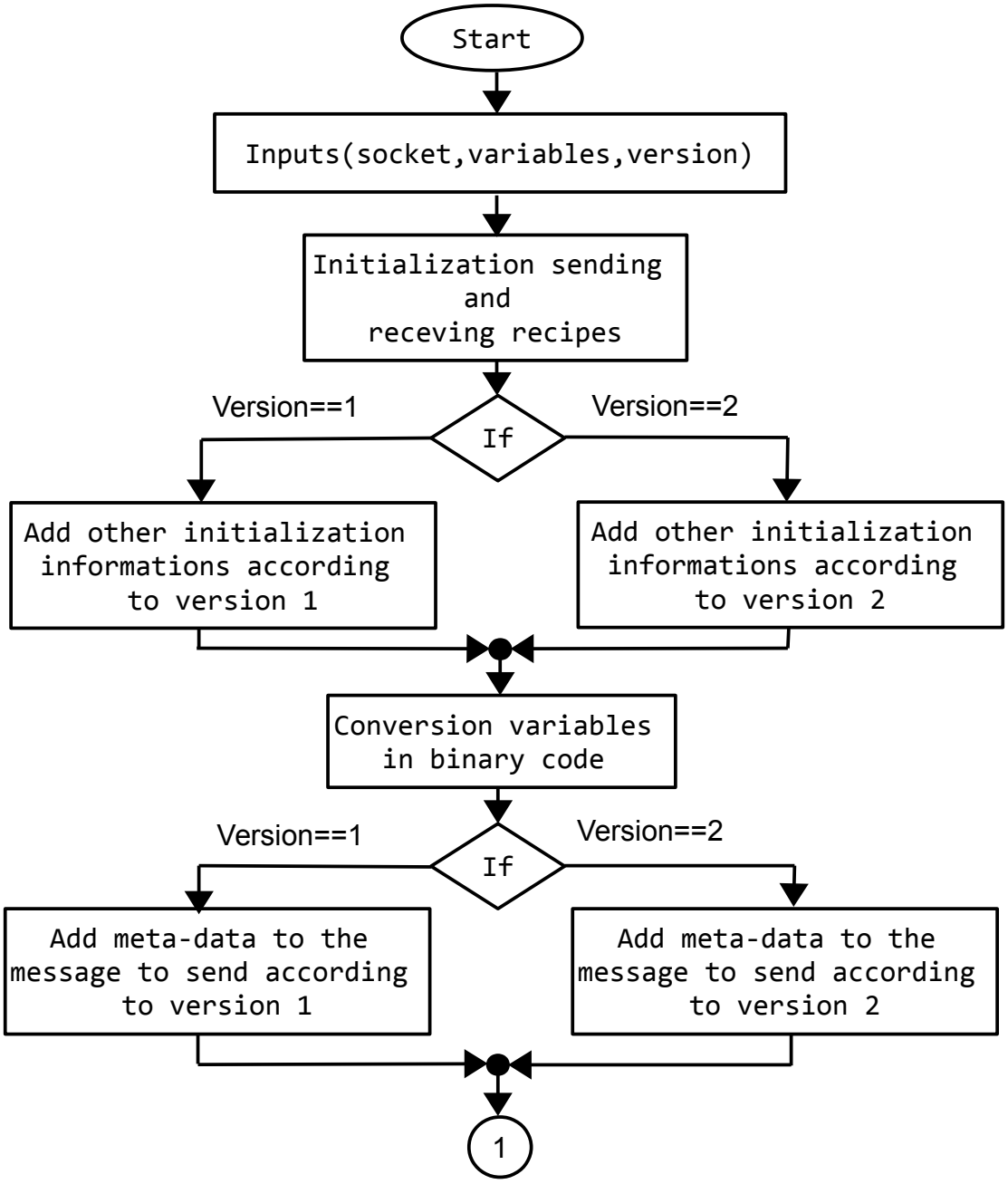
After the settings the output will provides the data requested in the order requested with a refresh of 125Hz for version 1 or required refresh for version 2.

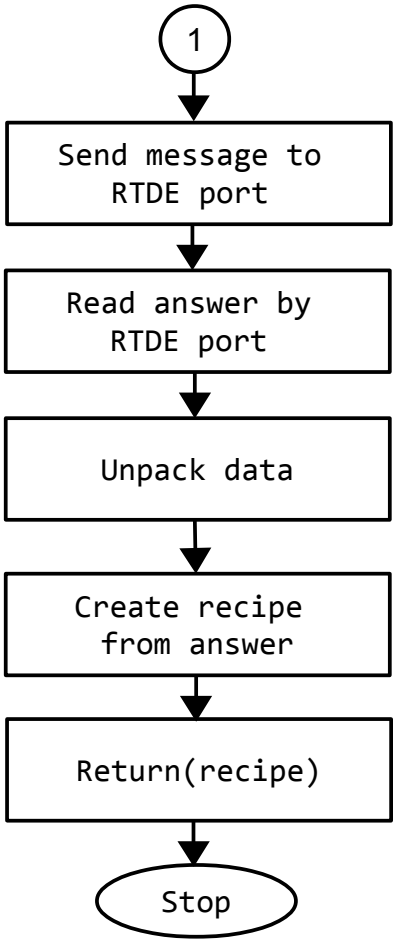
The information about allowed output can be founded in [9].

As default this function selects version 2 and refresh frequency at 125Hz.

As answer the robot will provide a recipe of how the data will have to be read.

Flow Chart





Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
Variables	List of strings	In according with the information provided by Universal Robot it is possible to send a list of data that we need to know in real-time by RTDE.
Version	Integer	The version to use, 1 or 2. 2 as default.
Frequency	Integer	If the version 2 has been selected, it is possible define the refresh frequency. 125Hz as default.

Output		
Variables	Type	Description
Output_recipe_id	Integer	Number corresponding to data output configuration.
Recipe	string	String composed by a sequence of letters that correspond to the format to unpack the streaming of data.

Comments

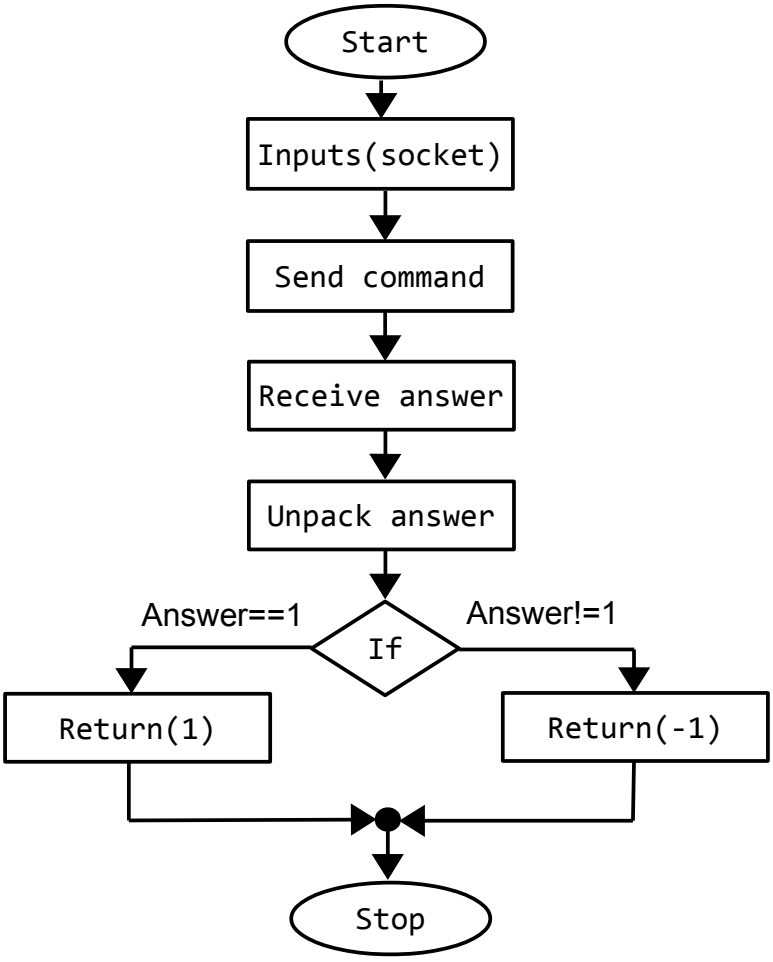
After the set of the output the streaming of data will not start till the RTDE_CONTROL_PACKAGE_START function will call.

RTDE_CONTROL_PACKAGE_START

Description

After the setting of the output, and if necessary of the version, the streaming of the data will not start till this function is called.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.

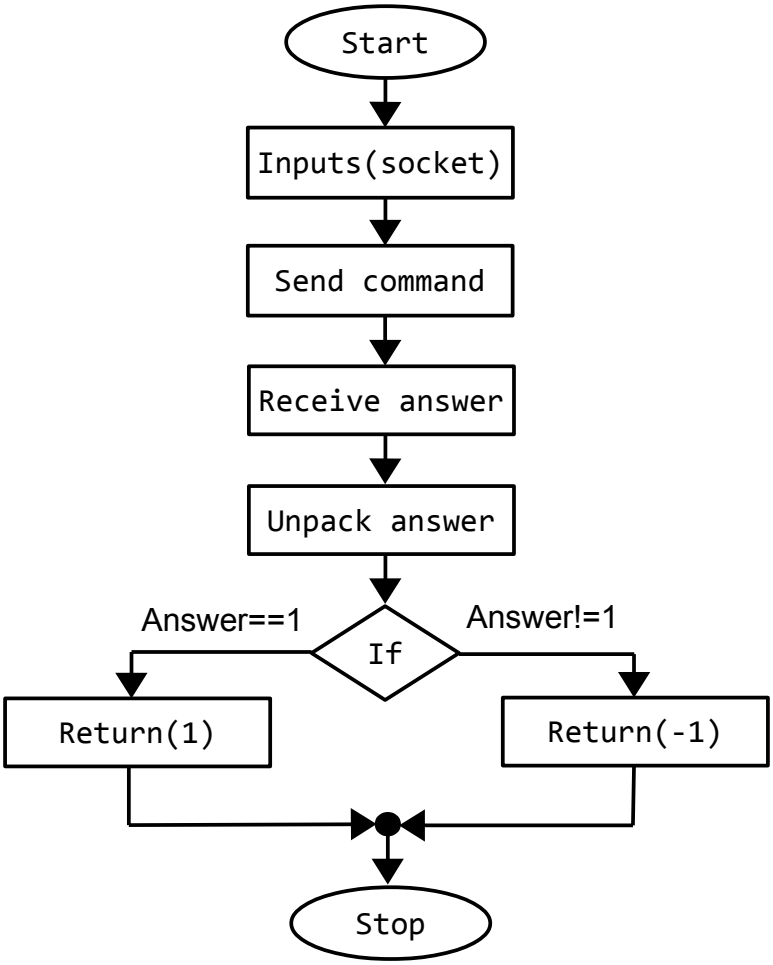
Output		
Variables	Type	Description
Start- return(1)	Integer	The RTDE port starts the streaming of data in the way asked in RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS.
Error- return(-1)	Integer	Some problem occur. No streaming of data.

RTDE_CONTROL_PACKAGE_PAUSE

Description

The streaming of the data can be put in pause in any moment using this function.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.

Output		
Variables	Type	Description
Start- return(1)	Integer	The RTDE port stops the streaming of data.
Error- return(-1)	Integer	Some problem occur. It is not possible put in pause the streaming of data.

Comment

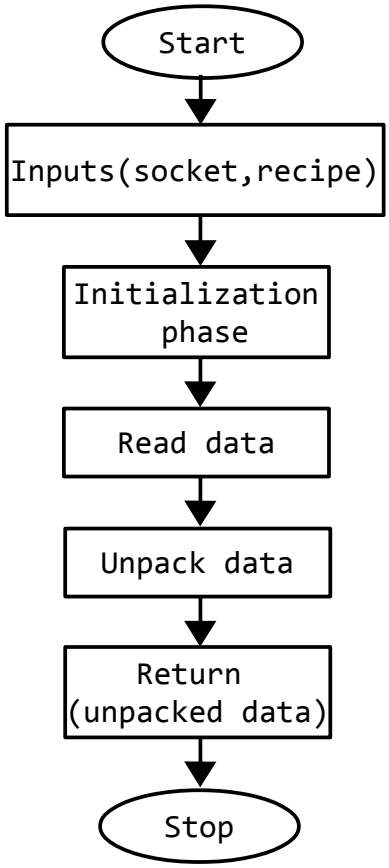
After the pause command is not necessary do again the initialization, the start command is enough.

RTDE_DATA_PACKAGE

Description

Thanks to the RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS output reads and unpacks the byte in the correct values.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
recipe	string	For works correctly this function needs the output provided by RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS.

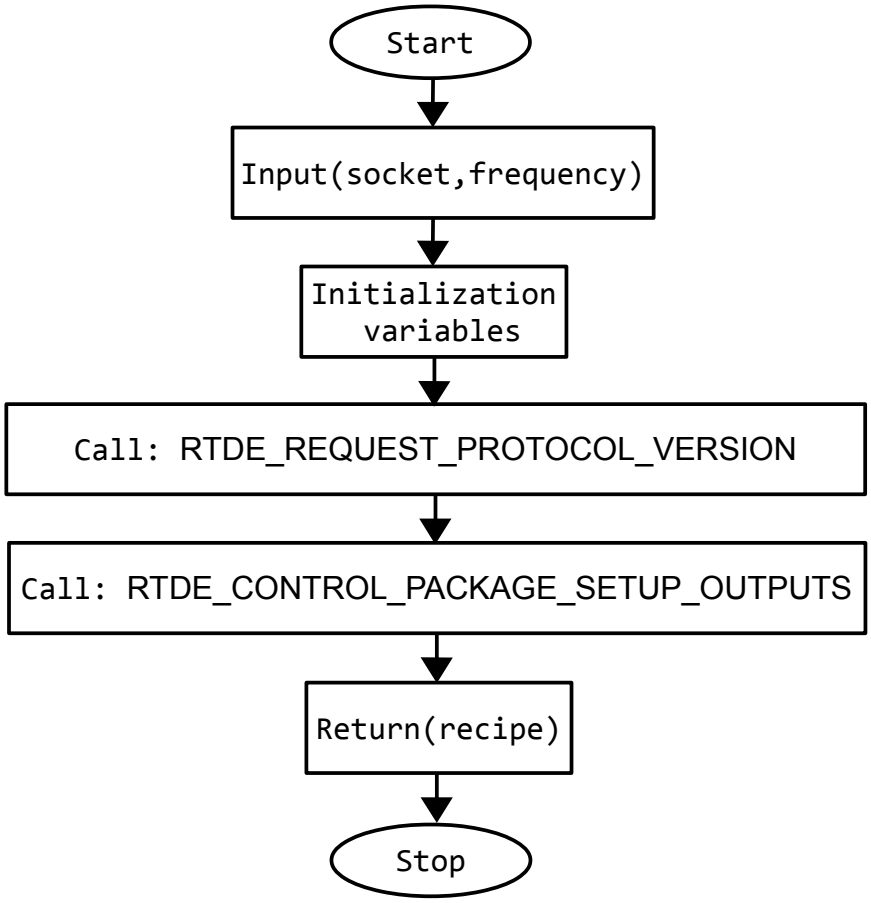
Output		
Variables	Type	Description
Data	List of different types	The received data after the unpacking create a list of different kind of data corresponding to the information provided by Universal Robot.

Library in Python-ROBOTIQ
SET_RTDE_OUTPUT_FT_SENSOR

Description

This function uses the UR library to set the output of the RTDE port to read the 6 axis force-torque data saved in the “output_double_register_X”, with X from 0 to 5.

Flow chart



Variables

Input		
Variables	Type	Description
Socket	Object	The function must know where to send the request. Only RTDE port is valid.
frequency	integer	The refresh frequency. 125Hz as default.

Output		
Variables	Type	Description
Output_recipe_id	Integer	Number corresponding to data output configuration.
Recipe	string	String composed by a sequence of letters that correspond to the format to unpack the streaming of data. In this case, six double.

Comment

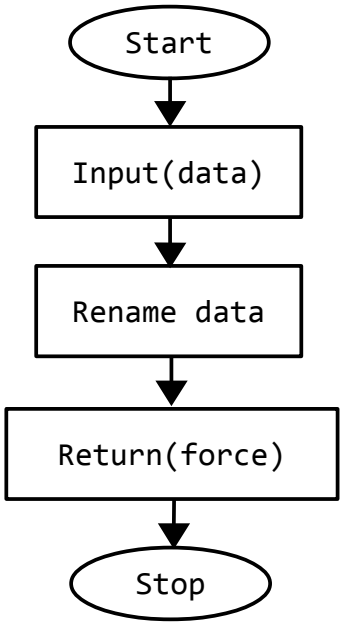
For the correct operation is necessary:
-to open a connection with RTDE client using connectRTDE.
-use this function for set RTDE output.
-to start the data streaming with RTDE_CONTROL_PACKAGE_START
-to read with streamRTDE
After this passages it is possible if desired to put the data in a dictionary with rename_FORCE_FT_SENSOR.

rename_FORCE_FT_SENSOR

Description

This function renames the data provided by streamRTDE after the correct setup with SET_RTDE_OUTPUT_FT_SENSOR.

Flow chart



Variables

Input		
Variables	Type	Description
Data	6xdouble	Data from “output_double_register_X”, with X from 0 to 5

Output		
Variables	Type	Description
Force	Dictionary	The sensor data are bounded with its own name FX, FY, FZ, MX, MY, MZ

URCap of support

An alternative to control the robot from the computer is to use a URCap of support. In this case the computer works as server and can be connected to the robot with “connectAS2C”.

This URCap has been designed for working to the command function. In this case SocketType must be set as “Server”.

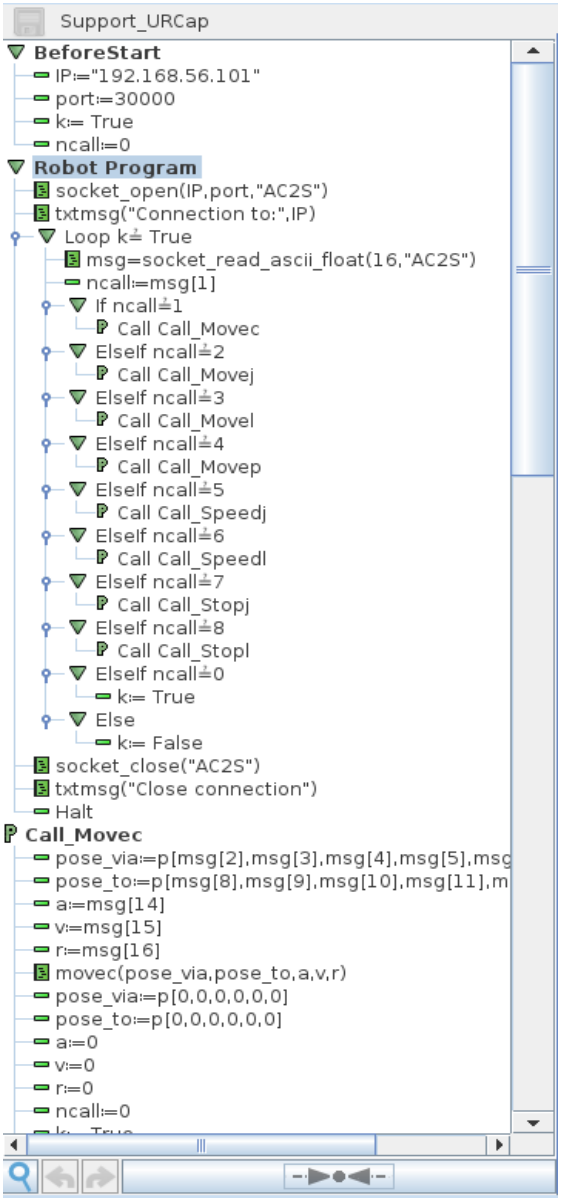
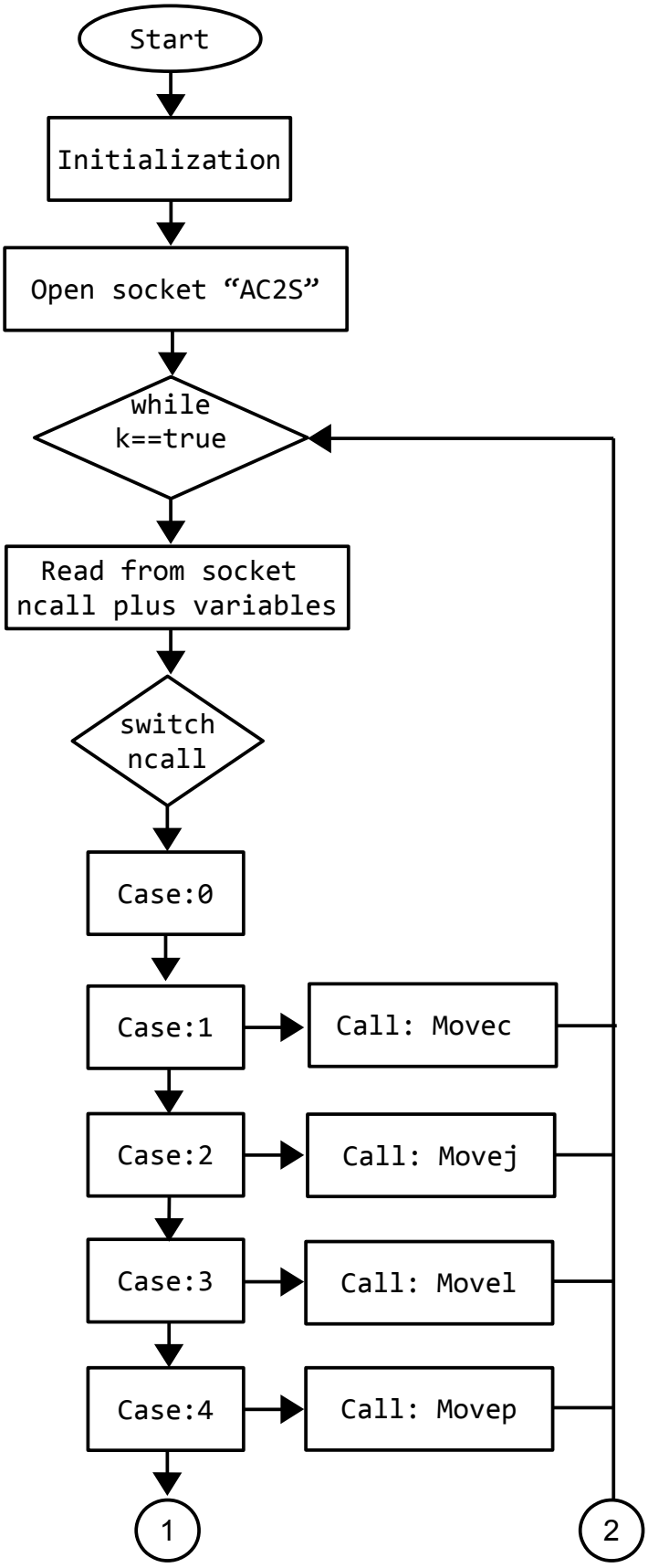


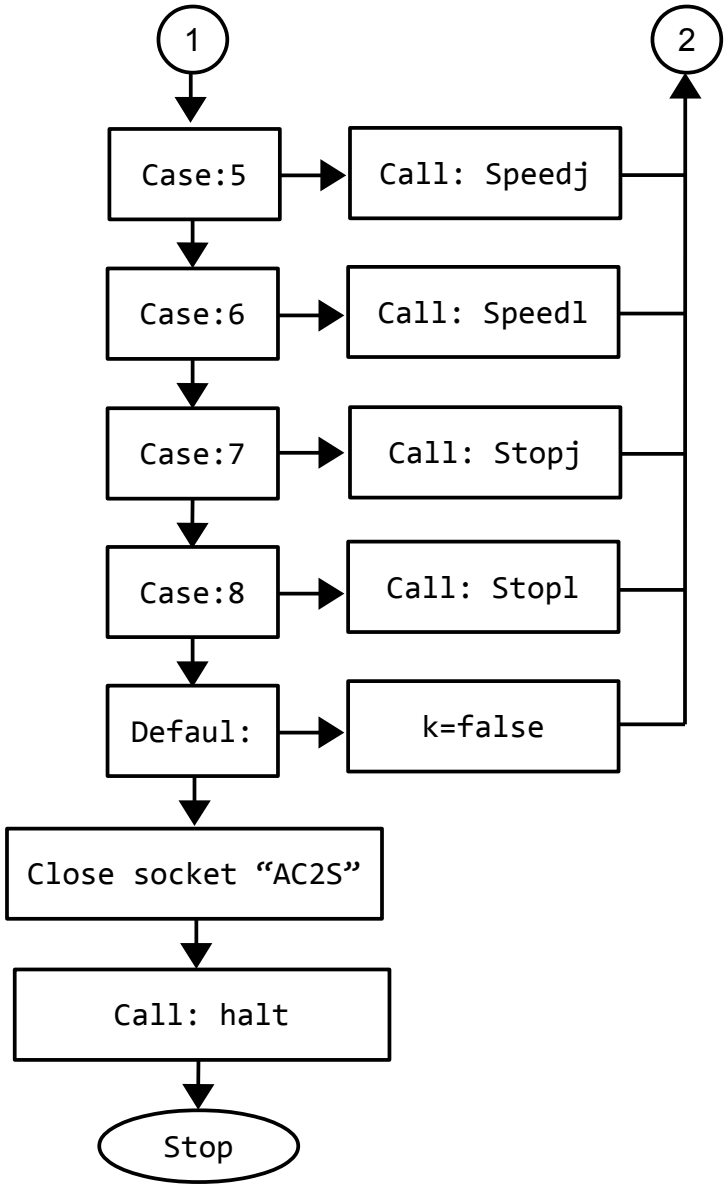
Fig 6.1 URCap of support on Polyscope

The program, after an initialization phase where opens a socket as client “AC2S”, starts a loop, that at any cycle, reads from socket and executes the sub-program corresponding to the correlate number.

When the “halt” command is called from computer the loop stops and the program too.

Flow chart


Start
Initialization
Open socket “AC2S”
while
k==true
Read from socket
ncall plus variables
switch
ncall
Case:0
Case:1
Call: Movec
Case:2
Call: Movej
Case:3
Call: Movel
Case:4
Call: Movep
1
2



Comment

The reason why the program at each choose call a sub-program and not the corresponding script directly is that I prefer a more robust approach in which after any script command the variables are set again to zero. This because if not specified in a URCap any variable is a global variable.

Load a complete URScript

As mentioned in chapter 4 a URCap hides a more complex URScript. Some of the rules, commands and functions used by URScript language can be found in [8]. I suggest to study the file generated by the Polyscope to better understand the URScript composition and other useful commands and functions.

For a very easy URCap program such this one:

```
Program
   Robot Program
     MoveJ
       Waypoint_1
       Waypoint_2
```

The robot generates the following URScript:

```
def MOVEJ():
  set_standard_analog_input_domain(0, 1)
  set_standard_analog_input_domain(1, 1)
  set_tool_analog_input_domain(0, 1)
  set_tool_analog_input_domain(1, 1)
  set_analog_outputdomain(0, 0)
  set_analog_outputdomain(1, 0)
  set_tool_voltage(0)
  set_input_actions_to_default()
  set_tcp(p[0.0,0.0,0.0,0.0,0.0,0.0])
  set_payload(0.0)
  set_gravity([0.0, 0.0, 9.82])
  while (True):
    $ 1 "Robot Program"
    $ 2 "MoveJ"
    $ 3 "Waypoint_1"
    movej([-1.60, -1.72, -2.20, -0.80, 1.59, -0.03], a=1.39, v=1.04)
    $ 4 "Waypoint_2"
    movej([-3.89, -1.72, -2.22, -0.76, 1.53, -2.32], a=1.39, v=1.04)
  end
end
```

If for any reason we have a URScript to load from computer it can be done using a PCI, SCI or RTCI connection.

To load the URScript is necessary sends any row terminated by the separator “\n” that correspond to the new line character.

For do it in Matlab I use the command:

```
fprintf(SOCKET,'%s\n',“URScriptLine”)
```

while in Python:

```
b=(“URScriptLine\n").encode("ascii")
SOCKET.send(b)
```

It is also possible to use shorter and easier URScript program, as follow:

```
def MOVEJ():
  while (True):
    movej([-1.60, -1.72, -2.20, -0.80, 1.59, -0.03], a=1.39, v=1.04)
```

```
    movej([-3.89, -1.72, -2.22, -0.76, 1.53, -2.32], a=1.39, v=1.04)
  end
end
```

This method is also used in chapter 7. The robot will provide to fill the starting initialization.

The most important rule is that a program must starts with: “def MyProg():\n” and stops with “end\n”.

When the last “end\n” will be sent the program will start soon.

Take in mind that if there is whatever error the script will be deleted by the robot without any information about it.

7 – Test of libraries and how to program

Introduction

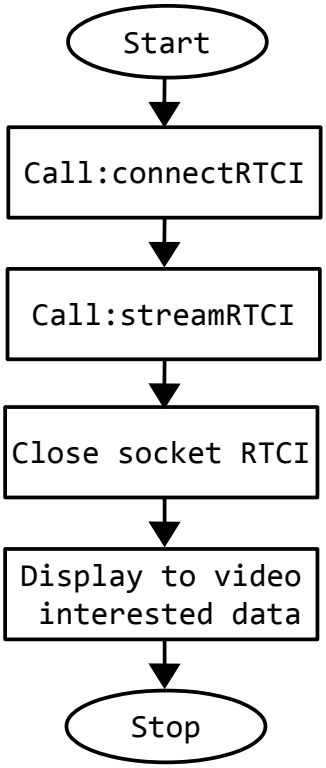
In this part will be reported some programs written with the previous libraries showing also some difference to using the RTCI compared to the URCap of support or Python compared to Matlab. All the complete programs are attached.

Test-1- Take position

Take_position – Program description

This first easy test is used for read joint position and TCP position from robot. The “Take_position.py” written in Python is taken in consideration.

Take_position – Flow-chart



Take_position – Comment

This function can be useful to take quickly the position from robot. The fig7.1 shows an example of output.

```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: runfile('D:/Università/Tesi Magistrale/Workspace_codici_python/Library Python UR and test/Take_position.py', wdir='D:/
Università/Tesi Magistrale/Workspace_codici_python/Library Python UR and test')
START...
Connection to UR3, IP: 192.168.56.103, 30003
q_actual:
 [-1.6007002035724085, -1.7271001974688929, -2.2029998938189905, -0.8079999128924769, 1.5951000452041626, -0.03099996248354131]
TCP_actual:
 [-0.11842706929373131, -0.2680453534302138, 0.15727817303169656, -0.0012209830284131572, 3.1162764812408956,
0.03889191616813326]
...END
```

Fig 7.1 Display output of "Take_position.py"

Test-2-movecjl

Test_movecjl_RTCI_1 – Program description

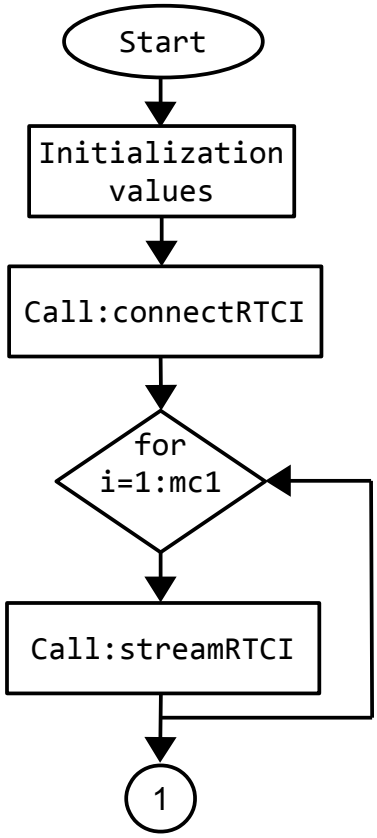
The second program test that I will discuss is a composition of movec, movej and movel. The program was written both in Matlab than Python and works both with RTCI than AS2C.

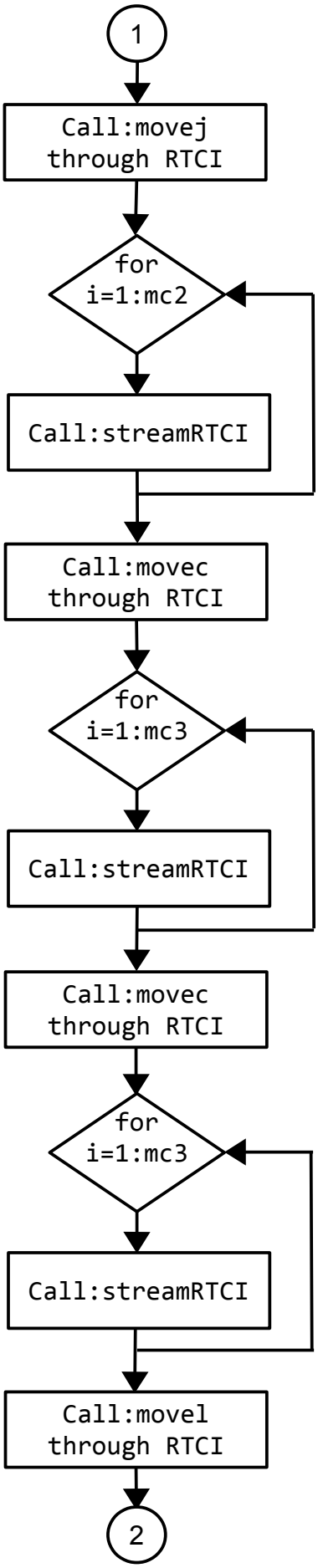
The focus of this program is of showing as a simple program can be composed and which problems take in mind for a good programming.

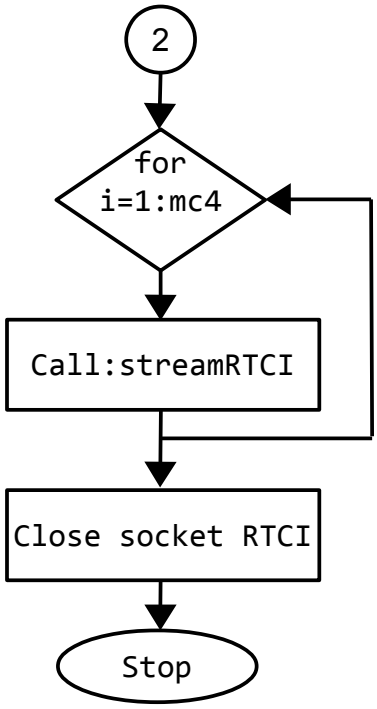
For explain different problems that can be found in the during the programming of the robot via computer I will start with the working program connected with RTCI and than I will move in the same program connected with AS2C. I will show what problems occur and how to solve them. The process will be repeat again from AS2C to RTCI.

Taking in consideration "Test_movecjl_RTCI_1.m" written in Matlab. The goal of the program is to execute four easy movement. At first the robot must reach the starting position using the movej command, then must do two movec command and finally does a linear movement with movel.

Test_movecjl_RTCI_1 - Flow-chart




1
Call:movej
through RTCI
for
i=1:mc2
Call:streamRTCI
Call:movec
through RTCI
for
i=1:mc3
Call:streamRTCI
Call:movec
through RTCI
for
i=1:mc3
Call:streamRTCI
Call:movel
through RTCI
2



Test_movecjl_RTCI_1 - Comment

Between any movement command there is a for cycle that read the data sent by RTCI every 8ms. The cycle is useful for two reasons. The first is acquire all the data in real time and the second is have a good synchronization with the robot. Indeed the robot sends data at 8ms and the read function inside streamRTCI must wait if does not find data to read. So increasing the iteration inside the loop it is possible increase the time between two consecutive movement commands.

The trajectory on the interested plane (X,Y) is reported in fig7.2

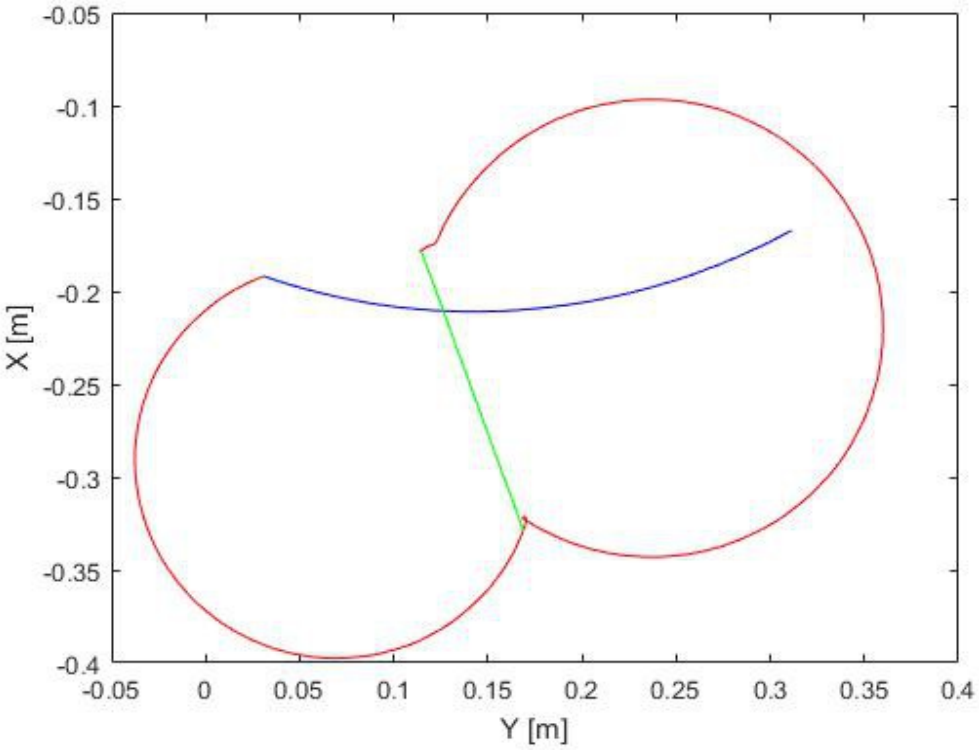


Fig 7.2 X,Y path for Test_movecjl_RTCI1.m

Movej is the starting movement and can be seen in the figure in blue, its path can change any time that the program is executed because is necessary to move the robot from a casual position to an initial position. Then it is possible to see the two movec in red and the movel in green, they will be the same any time that the program will run.

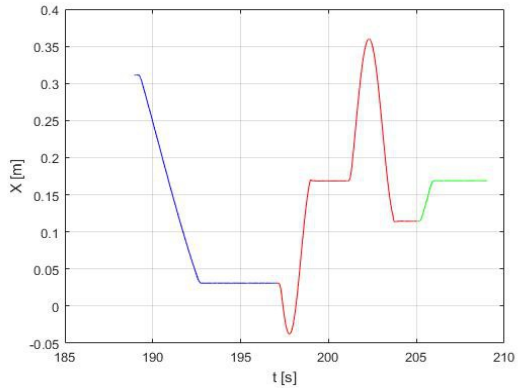


Fig 7.3 X(t) for Test_movecjl_RTCI1.m

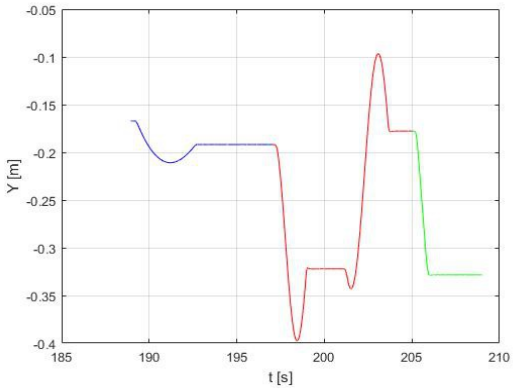


Fig 7.4 Y(t) for Test_movecjl_RTCI1.m

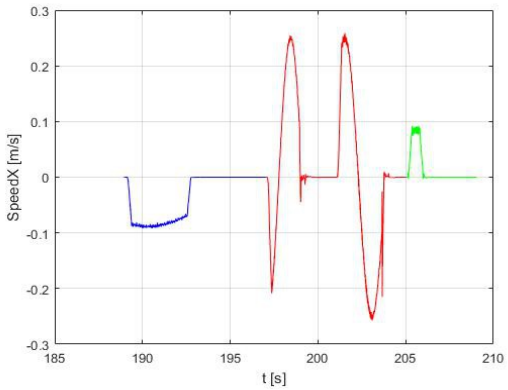


Fig 7.5 VX(t) for Test_movecjl_RTCI1.m

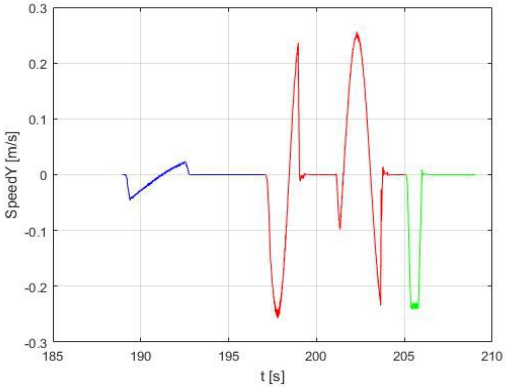


Fig 7.6 VY(t) for Test_movecjl_RTCI1.m

Above with the same colors for the different movements its possible to see the X and Y position and speed of the tool center point (TCP), (fig7.3, 7.4, 7.5, 7.6)

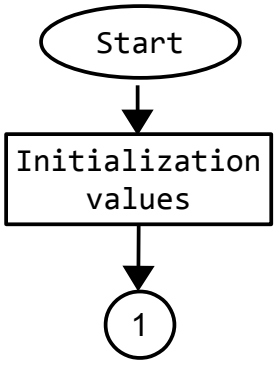
Test_movecjl_AS2C_1 - Program description

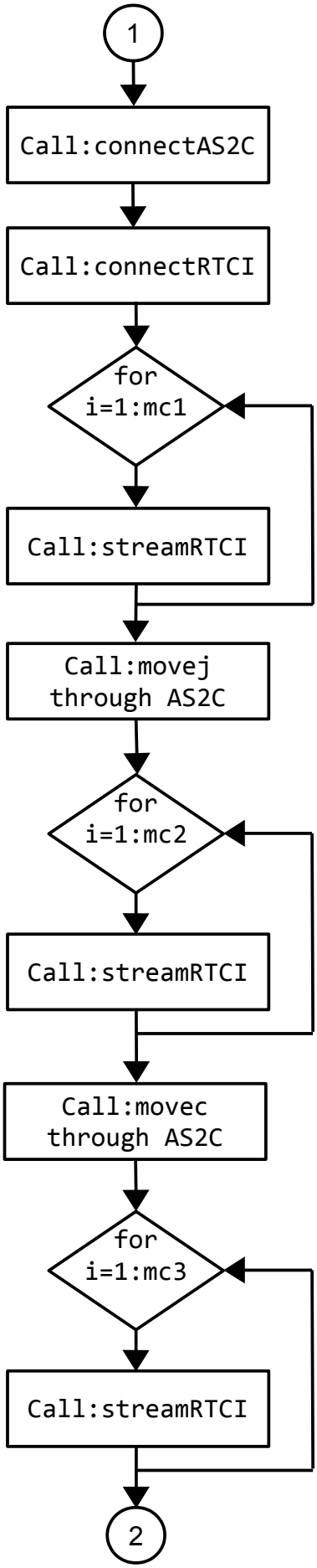
Now we change a little bit the program opening a connection to the robot in which the computer is the server to send command. See at the “Test_movecjl_AS2C_1.m” written in Matlab.

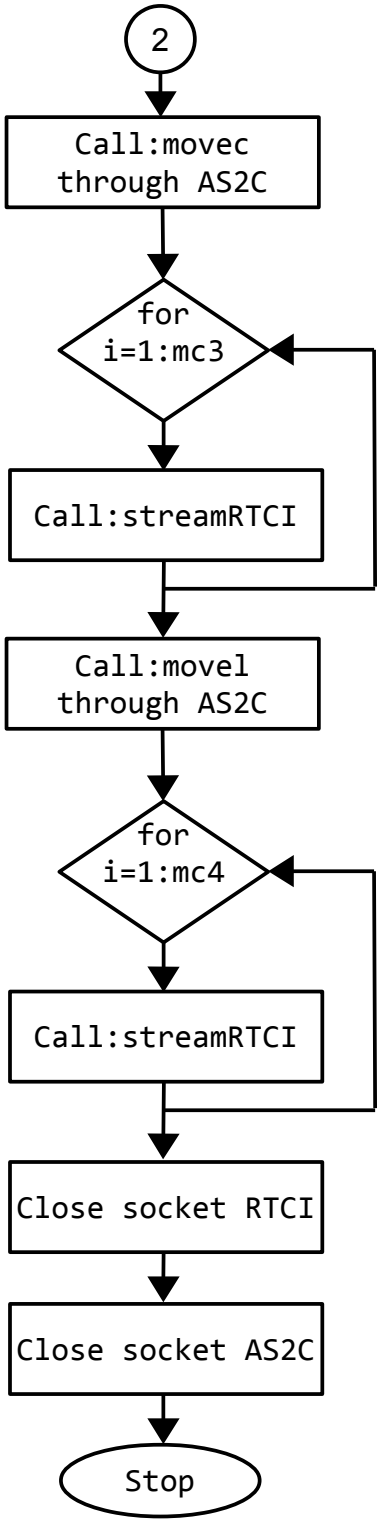
In this case its necessary to load “support_URCap.urp” on the robot.

The RTCI connection its always open to read the streaming of data from the robot but will not used to send command.

Test_movecjl_AS2C_1 – Flowchart




1
Call:connectAS2C
Call:connectRTCI
for
i=1:mc1
Call:streamRTCI
Call:movej
through AS2C
for
i=1:mc2
Call:streamRTCI
Call:movec
through AS2C
for
i=1:mc3
Call:streamRTCI
2



Test_movecjl_AS2C_1 - Comment

As before I will do some considerations on the graph from the collected data. Again the movement have the same colors so movel in blue, movec in red and movel in green.

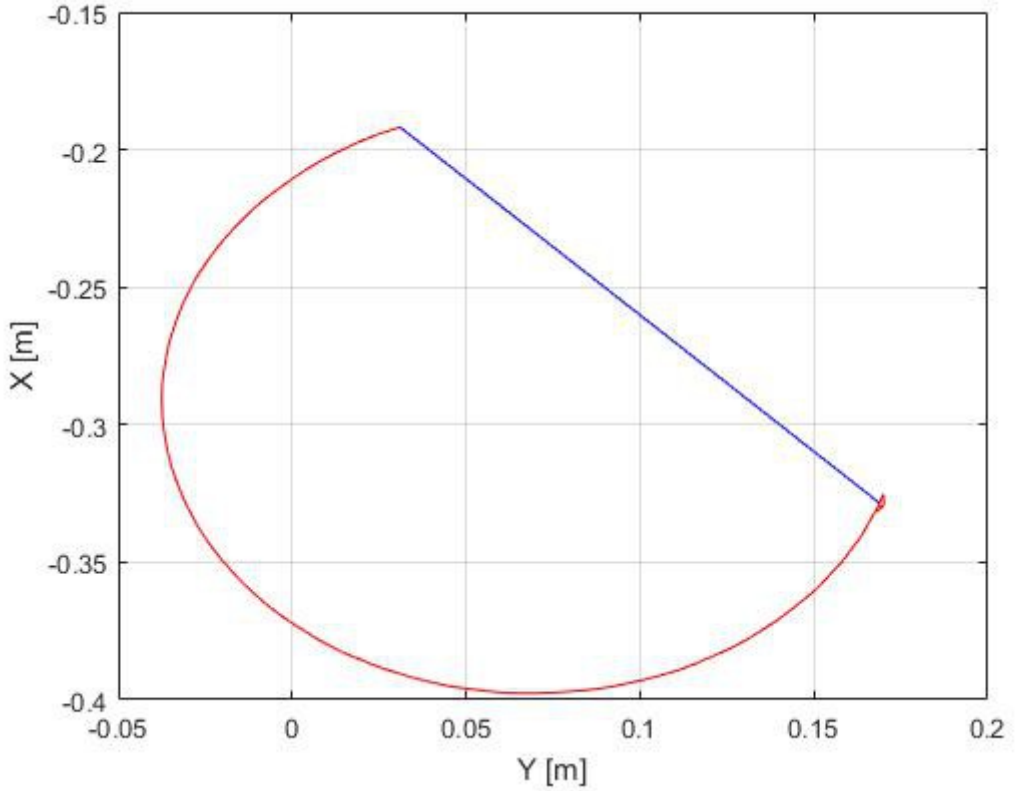


Fig 7.7 X,Y path for Test_movecjl_AS2C1.m

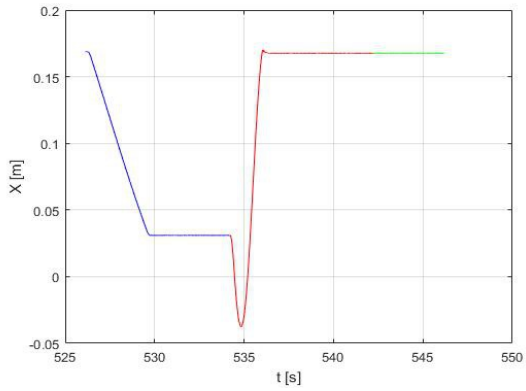


Fig 7.8 X(t) for Test_movecjl_AS2C1.m

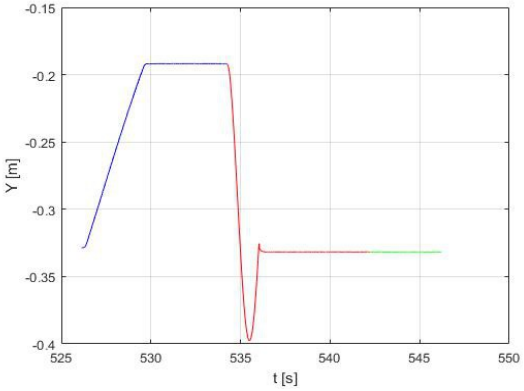


Fig 7.9 Y(t) for Test_movecjl_AS2C1.m

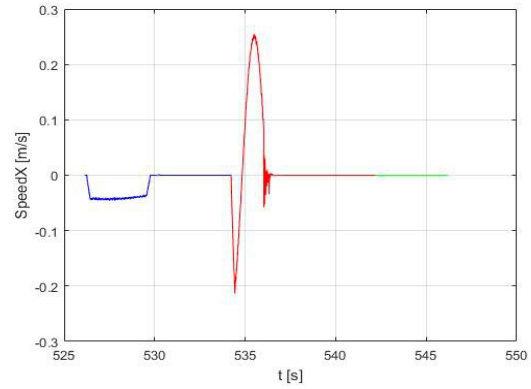


Fig 7.10 VX(t) for Test_movecjl_AS2C1.m

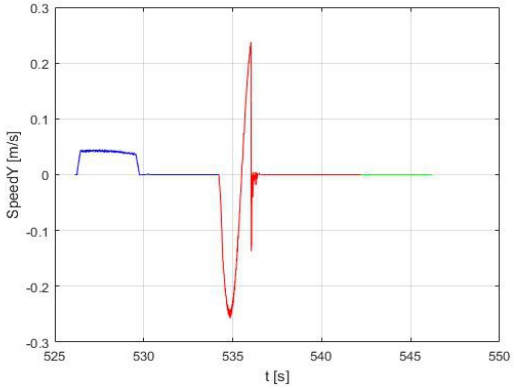


Fig 7.11 VY(t) for Test_movecjl_AS2C1.m

The graphs show a strange phenomenon. In fig7.7 where there is the position on the plain X,Y it is possible to see that disappear completely the second movec and movel and also in fig7.8, 7.9, 7.10, 7.11 after the first movec all the datas stay stable.

Watching the working robot it is also possible to see that after the first movec does not work more.

If we see at the teach pendant, it shows us the pop-up in figure 7.12.

Fig 7.12 Safety message on Polyscope

As explained by pop-up the program stops because the motion movec is not ramped correctly down.

Why does it works in the program before? And why does not appear the same message for movej?

At first the two motions are a little bit different. Movej, as movel, are point to point function, it means that both start with zero velocity and stop with zero velocity using a trapezoidal speed in their movements. Movec, as speedj or speedl, use instead a velocity ramp. They start from the actual speed, zero in this case, and stop with the desired final velocity.

So at the end of the movec movement the robot controller does not found any other command blocks the arm. Then because the speed is too high stops also the program and wait for a human correction.

Now, to answer at the first question, the movement in the previous program does not block because the robot sees any movement as a different program while in this case the robot runs a program with a loop that reads and then decide what movement uses. So also in the previous case the movement is not correctly ramped down but the program ends.

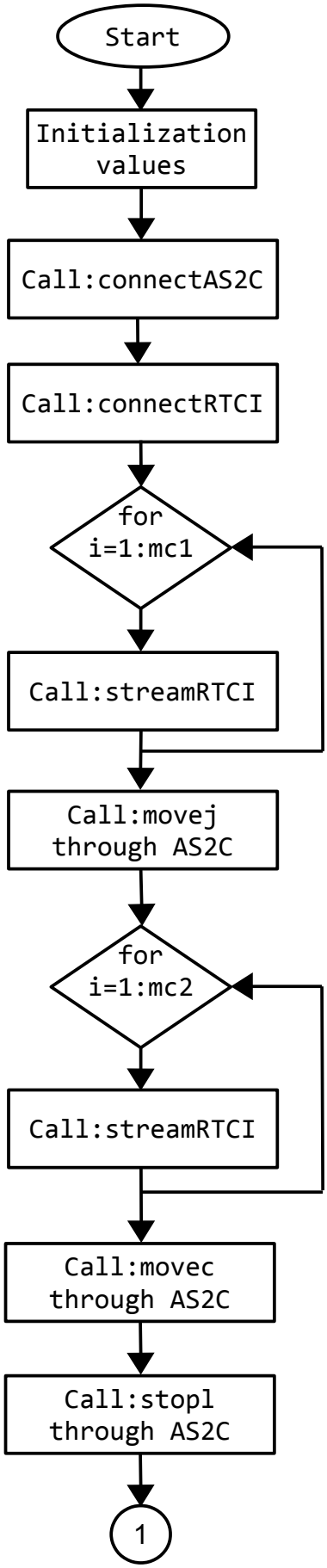
It is also noteworthy that the program on the pc is able to receive the data because the RTCI port provided by the robot, is not connected at the running program on the robot itself.

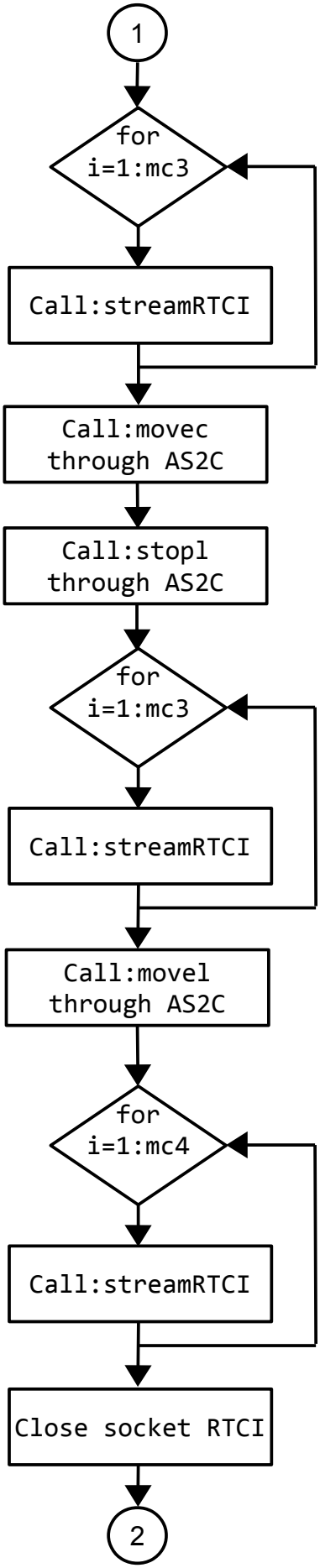
Test_movecjl_AS2C_2 - Program description

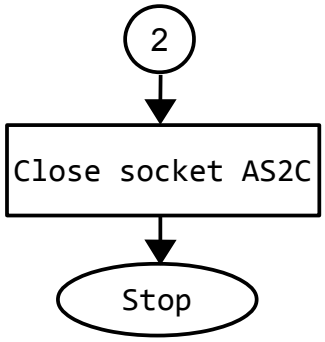
To avoid the error message appeared in the previous test and to avoid also sudden stop that can damage the cobot we can add as suggested by the pop-up a stopl to ramp correctly down the movement. See at the “Test_movecjl_AS2C_2.m” written in Matlab.

Also in this case the URCap of support on the robot is necessary.

Test_movecjl_AS2C_2 – Flow chart




1
for
i=1:mc3
Call:streamRTCI
Call:movec
through AS2C
Call:stopl
through AS2C
for
i=1:mc3
Call:streamRTCI
Call:movel
through AS2C
for
i=1:mc4
Call:streamRTCI
Close socket RTCI
2



Test_movecjl_AS2C_2 - Comment

As before I will do some considerations on the graph from the collected data. The movement have the follow colors: movel in blue, movec+stopl in red and movel in green.

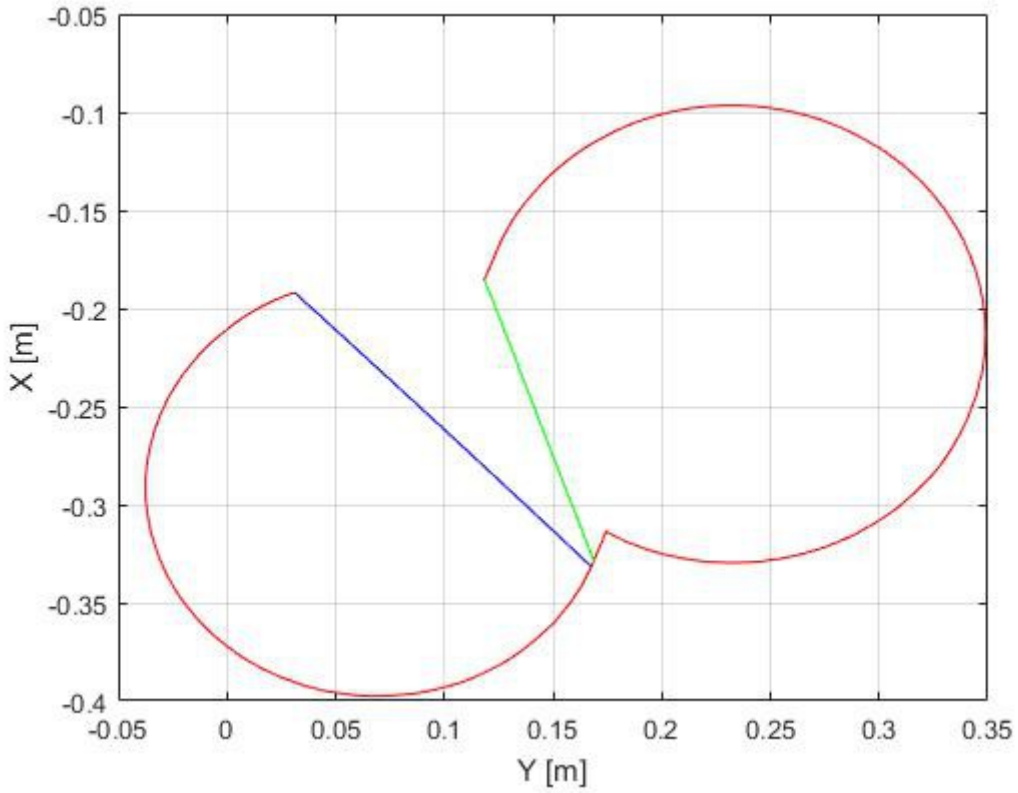


Fig 7.13 X,Y path for Test_movecjl_AS2C2.m

As its possible to see from fig7.13 now the program works properly.

It is important to take in mind that the movement in this case does not finish where we imposed as value, because reached that point the robot decelerate using stopl, so it will continue for a little bit forward on the path.

From the fig7.14, 7.15, 7.16, 7.17 is instead possible to see respect fig7.8, 7.9, 7.10, 7.11 that the end of the two movec thanks to stopl are more smooth with a reduced overshot.

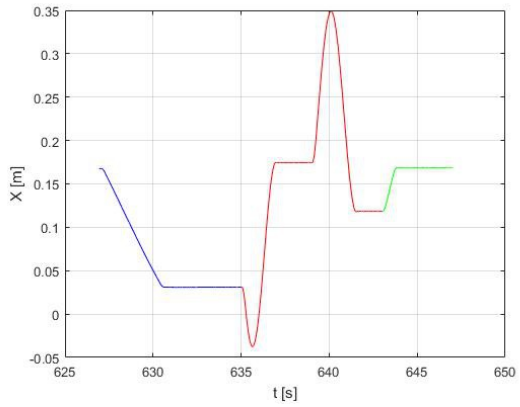


Fig 7.14 X(t) for Test_movecjl_AS2C2.m

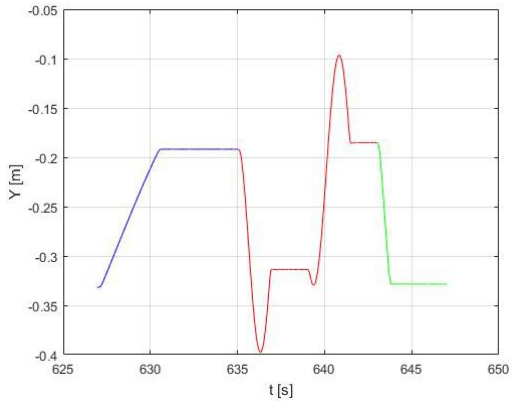


Fig 7.15 Y(t) for Test_movecjl_AS2C2.m

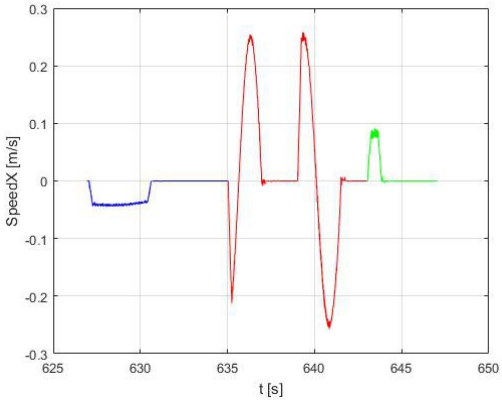


Fig 7.16 VX(t) for Test_movecjl_AS2C2.m

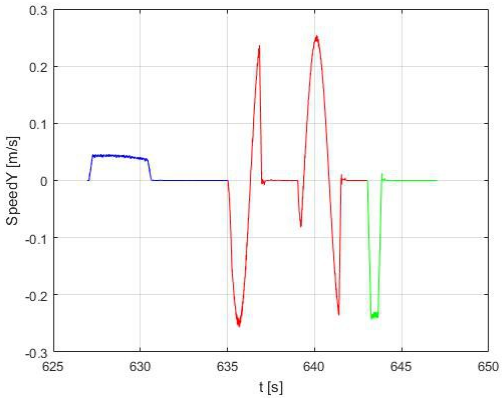
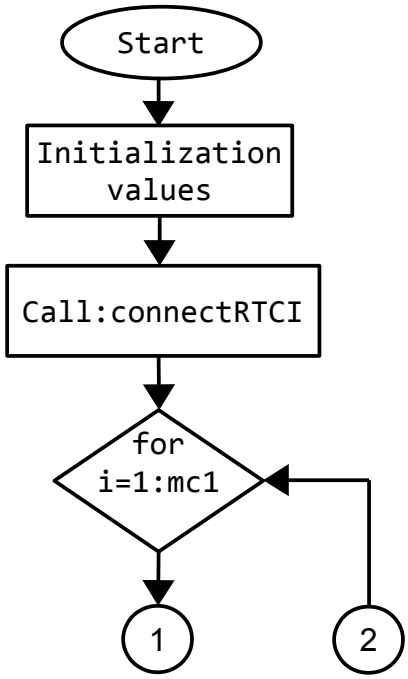


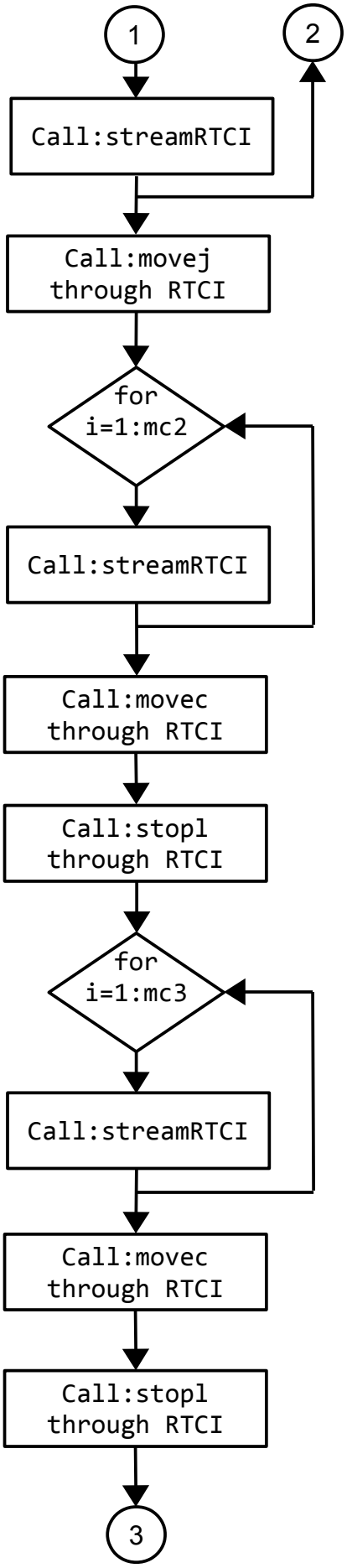
Fig 7.17 VY(t) for Test_movecjl_AS2C2.m

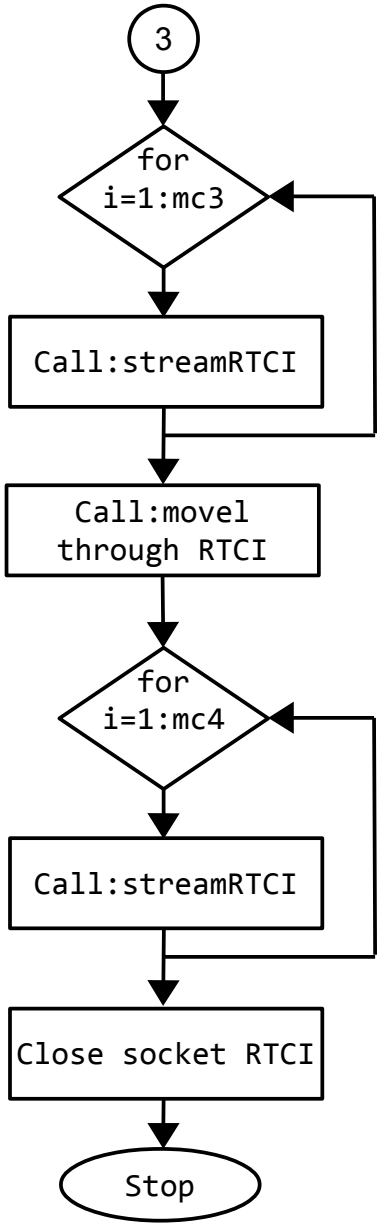
Test_movecjl_RTCI_2 - Program description

With the idea of avoiding sudden stop also in the first program we can add stopl also in the first program. See at the “Test_movecjl_RTCI_2.m” written in Matlab.

Test_movecjl_RTCI_2 - Flow chart




1
2
Call:streamRTCI
Call:movej
through RTCI
for
i=1:mc2
Call:streamRTCI
Call:movec
through RTCI
Call:stopl
through RTCI
for
i=1:mc3
Call:streamRTCI
Call:movec
through RTCI
Call:stopl
through RTCI
3



Test_movecjl_RTCI_2 - Comment

As before I will do some considerations on the graph from the collected data. The movement have the follow colors: movel in blue, movec+stopl in red and movel in green.

In this case, as we can see in fig7.18, 7.19, 7.20, 7.21, 7.22, using stopl after movec we have delete it.

Why?

As say in Test_movecjl_AS2C_1 - Comment the movement sent through RTCI are single program that starts and ends. Any time we send a new command to the robot, it overwrite the previous command.

In this case use stopl just after movec overwrite it, so the movement never starts.

The reason why in the URCap of support works is that in a unique program the next command waits the end of the previous.

To reach our goal it is so necessary write movec and stopl in an unique program.

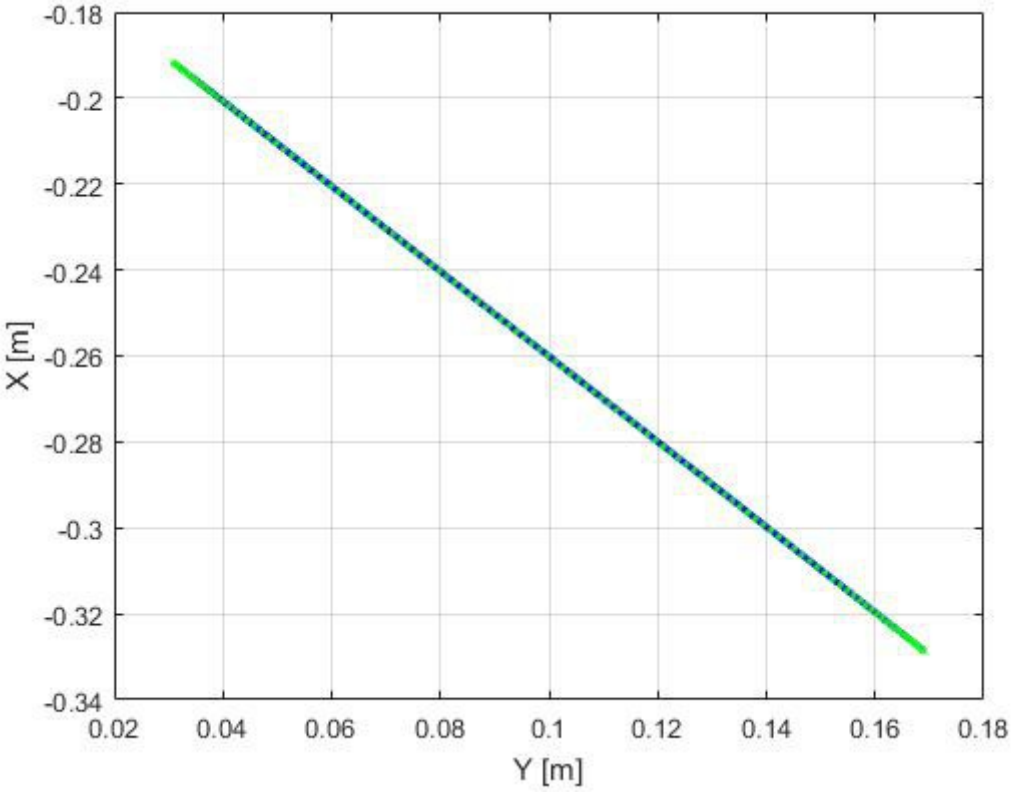


Fig 7.18 X,Y path for Test_movecjl_RTCl2.m

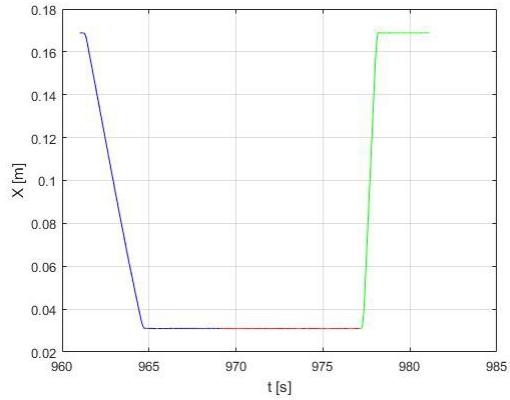


Fig 7.19 X(t) for Test_movecjl_RTCl2.m

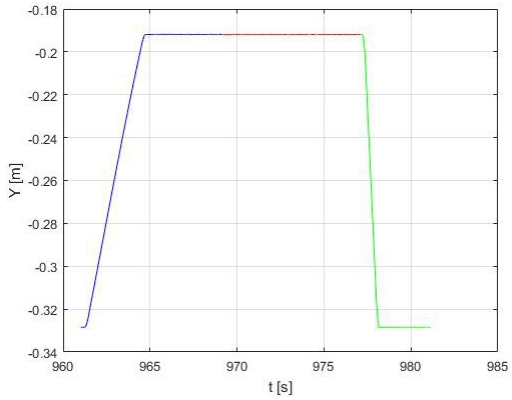


Fig 7.20 Y(t) for Test_movecjl_RTCl2.m

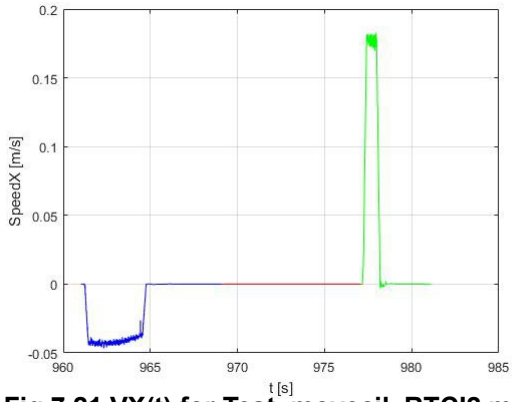


Fig 7.21 VX(t) for Test_movecjl_RTCl2.m

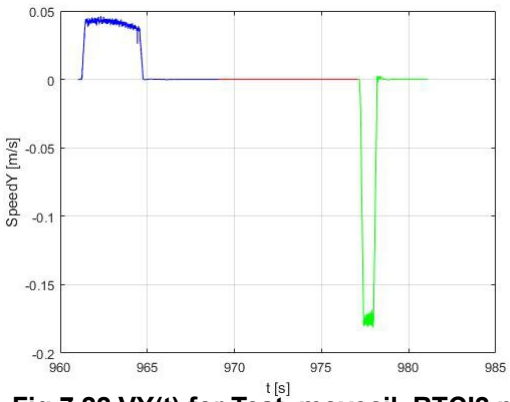
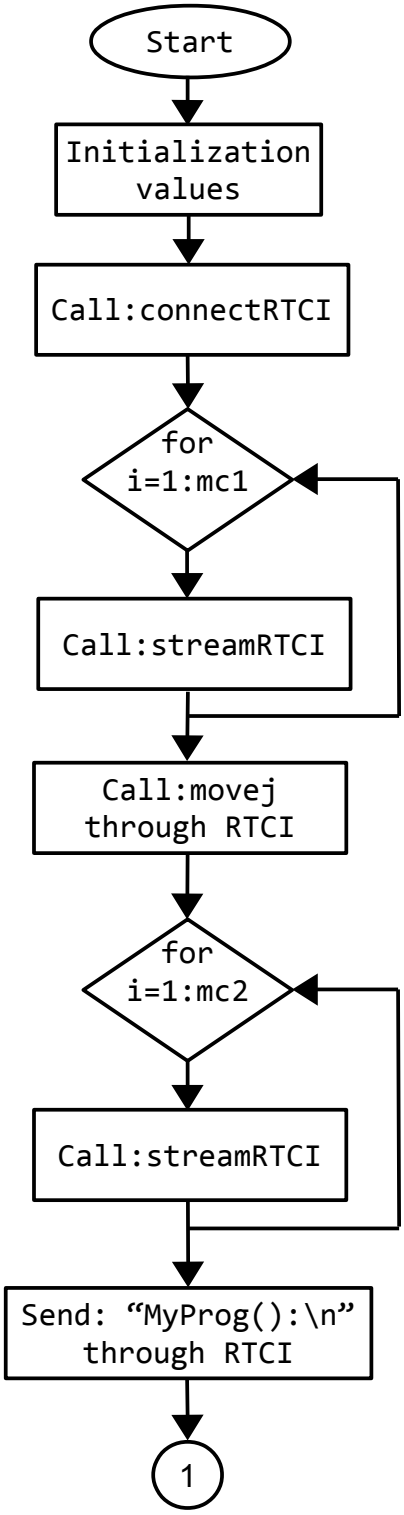


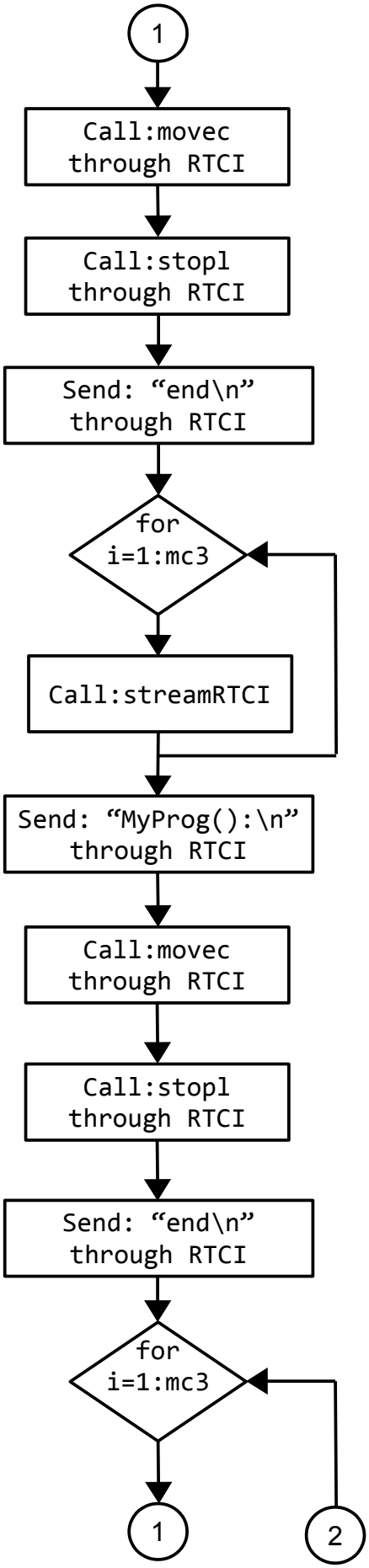
Fig 7.22 VY(t) for Test_movecjl_RTCl2.m

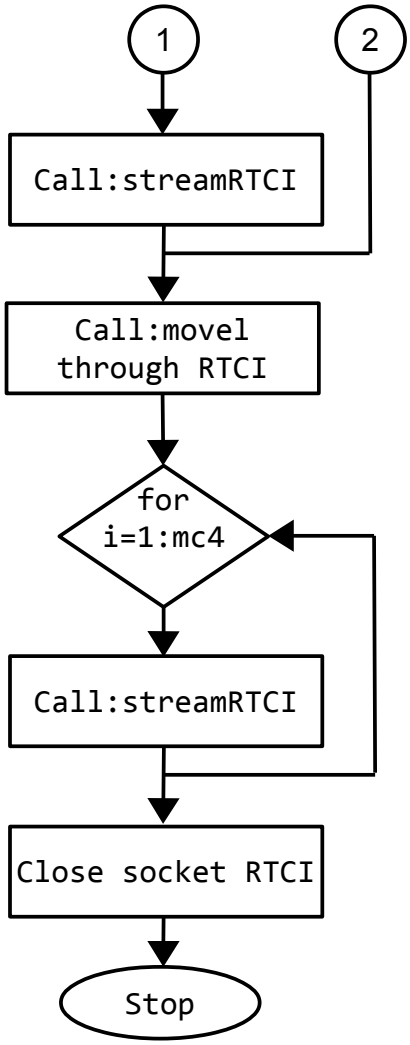
Test_movecjl_RTCI_3 - Program description

In this last program I introduce how write an URScript using the pc and sending it to the robot. The program is not so different from the before ones. See at the “Test_movecjl_RTCI_3.m” written in Matlab.

Test_movecjl_RTCI_3 - Flow chart




1
Call:movec
through RTCI
Call:stopl
through RTCI
Send: “end\n”
through RTCI
for
i=1:mc3
Call:streamRTCI
Send: “MyProg():\n”
through RTCI
Call:movec
through RTCI
Call:stopl
through RTCI
Send: “end\n”
through RTCI
for
i=1:mc3
1
2



Test_movecjl_RTCI_3 - Comment

As before I will do some considerations on the graph from the collected data. The movement have the follow colors: movel in blue, movec+stopl in red and movel in green.

The lines “MyProg():\n” and “end\n” start and end a URScript program. The complexity could be the one we want and we need. Also in this case if we will send a new command, it will kill the running program.

The chapter 6 can help in part how to write a URScript.

The new obtained graph in fig7.23, 7.24, 7.25, 7.26, 7.27 are, as for Test_movecjl_AS2C_2, smooth without high overshoots.

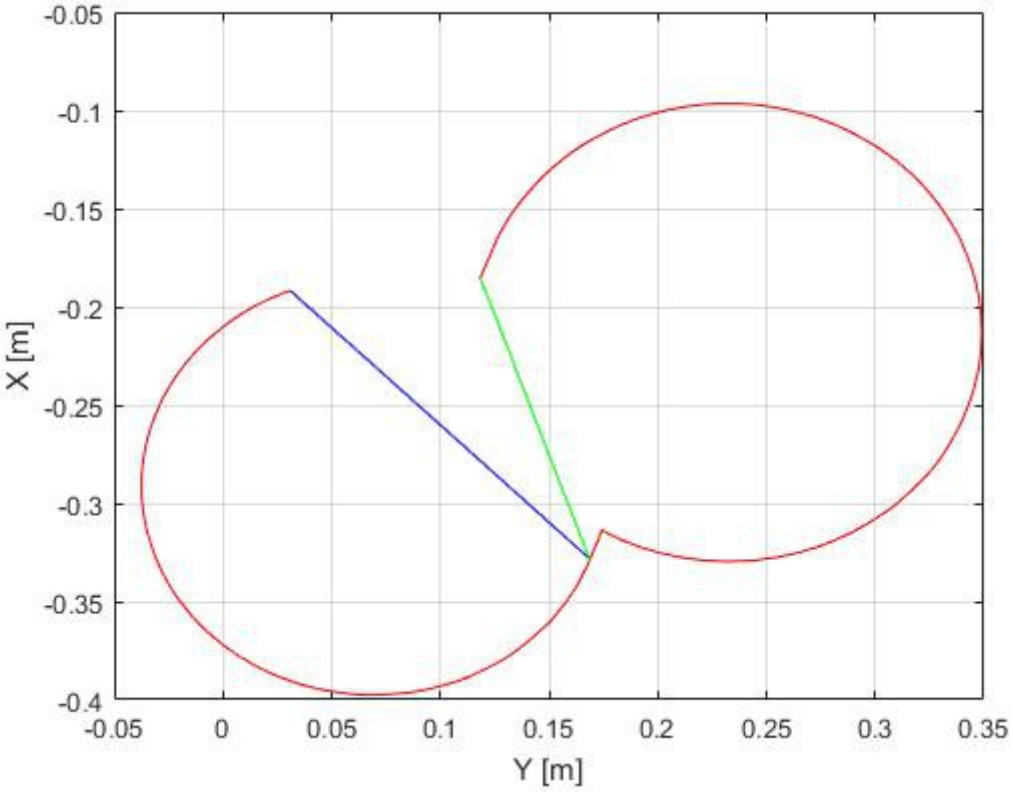


Fig 7.23 X,Y path for Test_movecjl_RTCI3.m

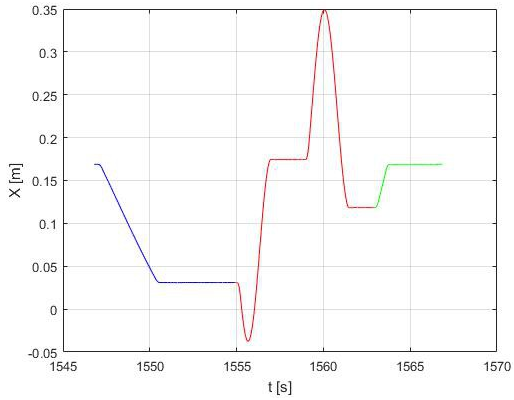


Fig 7.24 X(t) for Test_movecjl_RTCI3.m

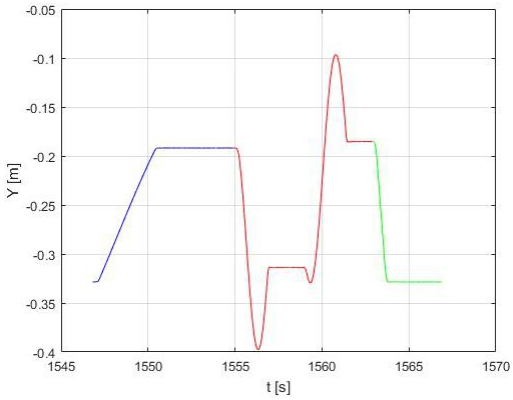


Fig 7.25 Y(t) for Test_movecjl_RTCI3.m

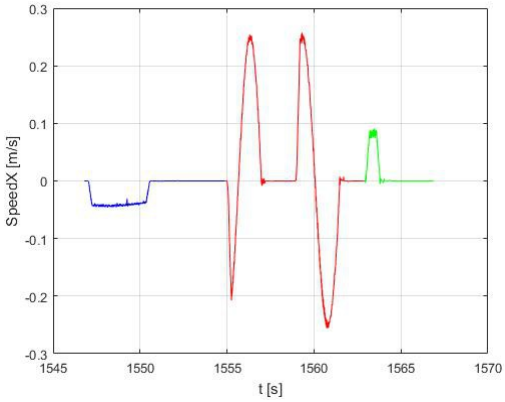


Fig 7.26 VX(t) for Test_movecjl_RTCI3.m

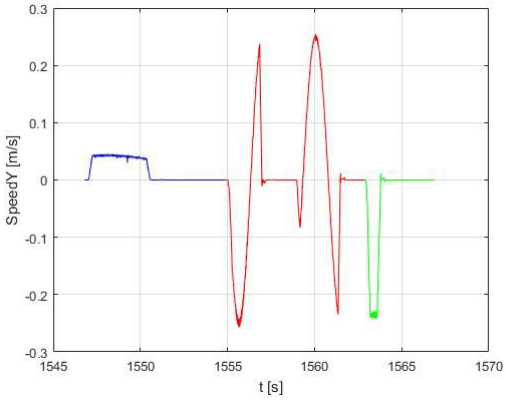


Fig 7.27 VY(t) for Test_movecjl_RTCI3.m

Test-3-Slider and Speedl

Test_SliderAndSpeedl – Program description

This third test try to control in real time the robot from the computer. Although it works, not works very well, but it is in any case interesting to study as starting point for future programs.

The program implement a user interface (UI) (fig7.28) that makes move the robot, on X,Y plan, in real time also thanks the URCap of support.

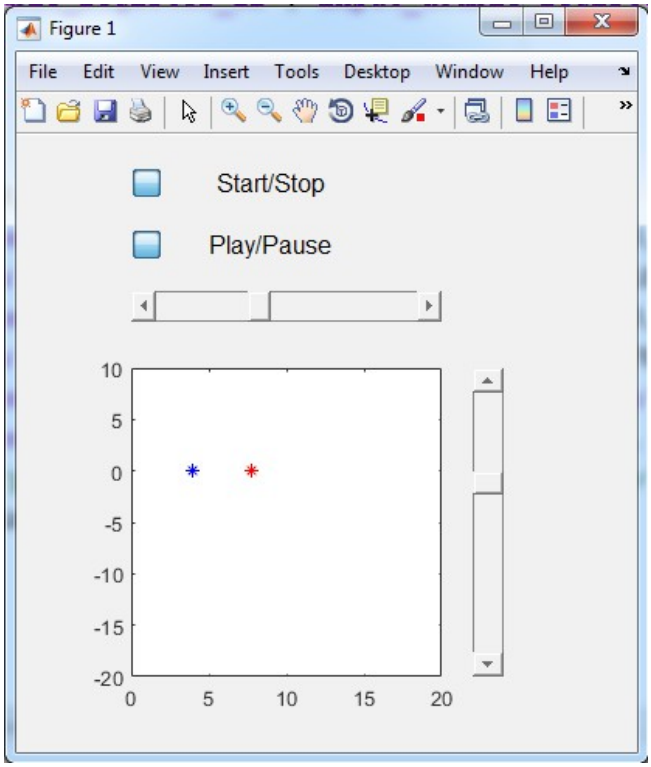


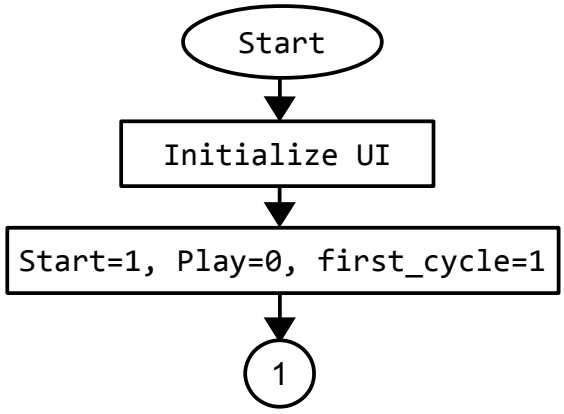
Fig 7.28 User Interface

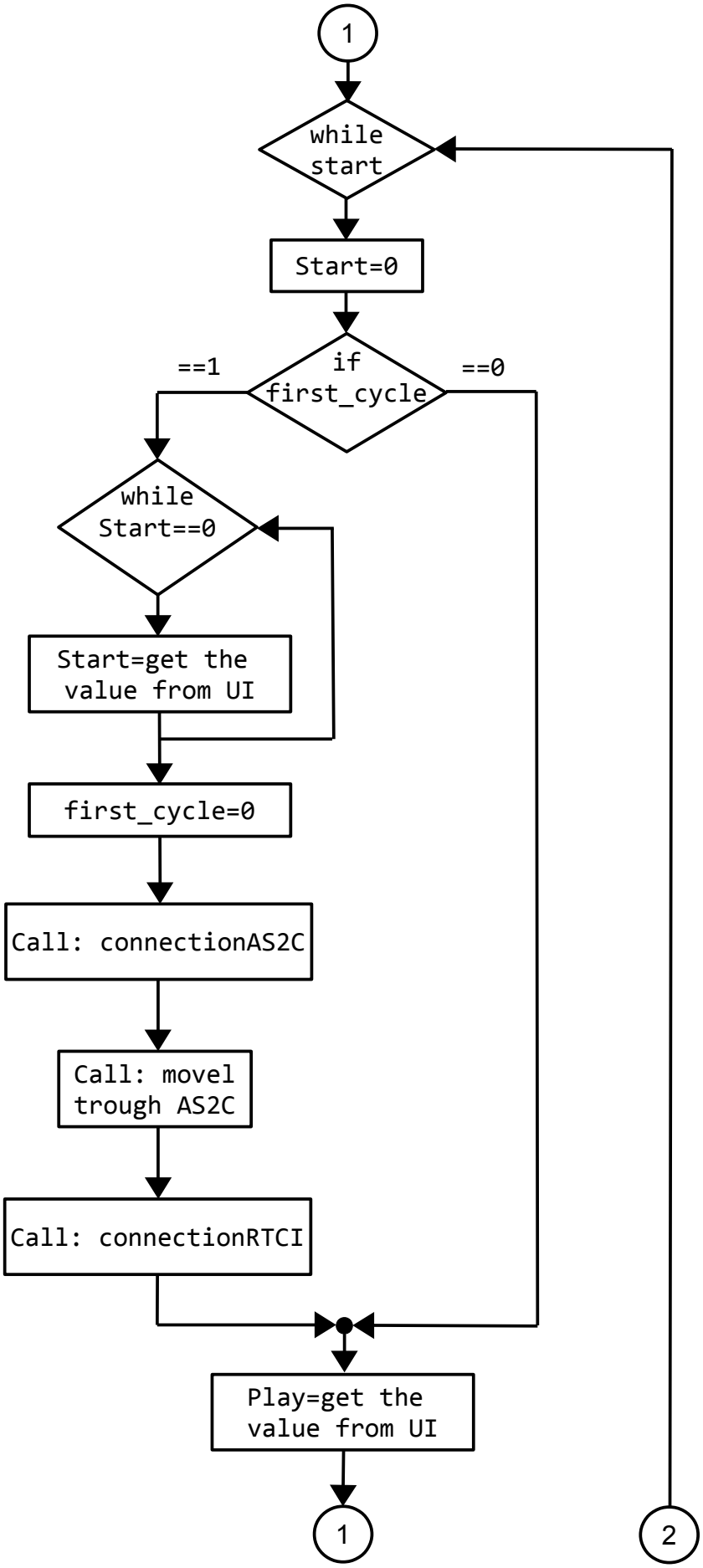
The UI has two push buttons, one for strarts and stops the program and one for put in play or pause the program. A screen where show the position of the target (red) and the robot tool center point (blue). And two sliders to move the target position on X,Y plan.

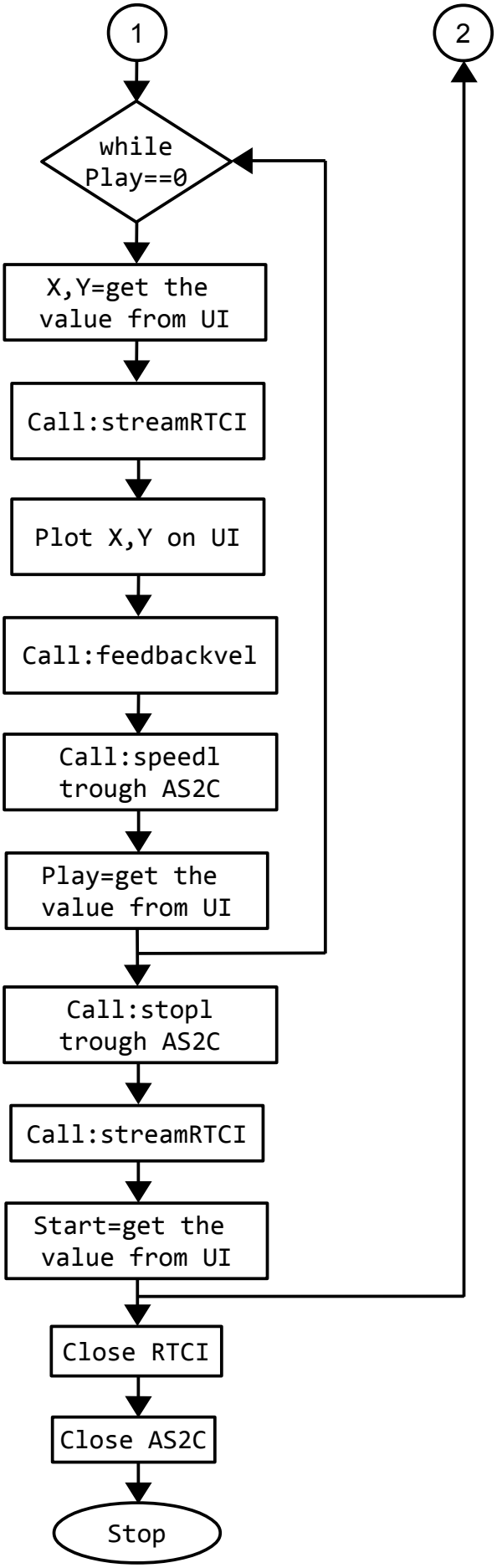
When starts the program initialize a series of operation including the opening of the sockets while when stops ends the program.

The play/pause push button is necessary to send the speedl command to the robot.

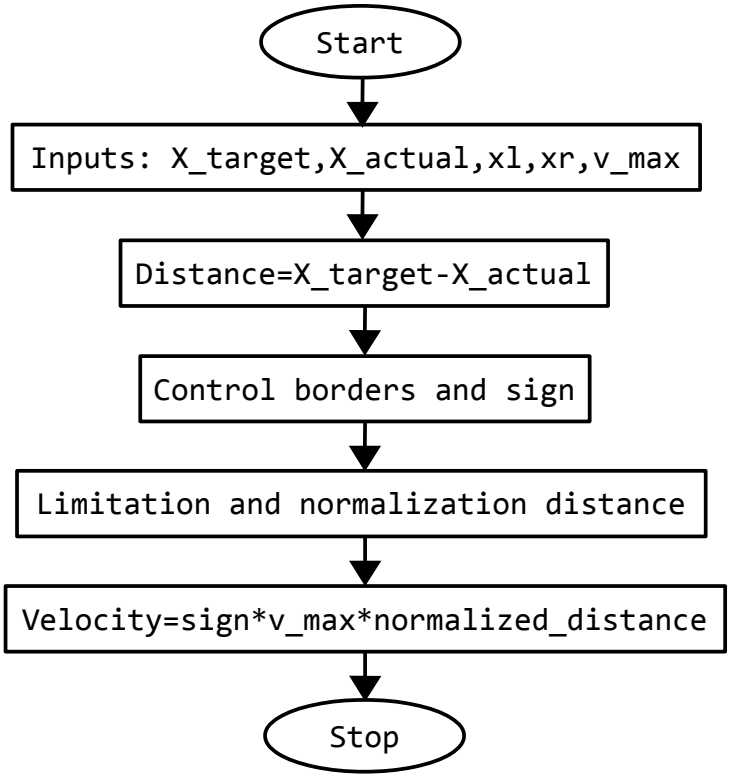
Test_SliderAndSpeedl – Flow-chart




1
while start
Start=0
==1
if first_cycle
==0
while Start==0
Start=get the value from UI
first_cycle=0
Call: connectionAS2C
Call: movel trough AS2C
Call: connectionRTCI
Play=get the value from UI
1
2


1
2
while
Play==0
X,Y=get the
value from UI
Call:streamRTCI
Plot X,Y on UI
Call:feedbackvel
Call:speedl
trough AS2C
Play=get the
value from UI
Call:stopl
trough AS2C
Call:streamRTCI
Start=get the
value from UI
Close RTCI
Close AS2C
Stop

feedbackvel function– Flow-chart



Test_SliderAndSpeedl – Comment

The “feedbackvel” is a function that compute the velocity proportional to the distance between the target point and the actual position of the robot.
The program was tried with three different values of “t” as input of “speedl” function in the URCap.

speedl(*xd, a, t, aRot=*'a')

Tool speed

Accelerate linearly in Cartesian space and continue with constant tool speed. The time t is optional; if provided the function will return after time t, regardless of the target speed has been reached. If the time t is not provided, the function will return when the target speed is reached.

Parameters

`xd:` tool speed (m/s) (spatial vector)

`a:` tool position acceleration (m/s^2)

`t:` time (s) before function returns (optional)

`aRot:` tool acceleration (rad/s^2) (optional), if not defined a, position acceleration, is used

Example command: `speedl([0.5,0.4,0.0,0.,1.57,0,0], 0.5, 0.5)`

- Example Parameters:
  - qd → Tool speeds of: x=500 mm/s, y=400 mm/s, rx=90 deg/s, ry and rz=0 mm/s
  - a = 0.5 rad/s^2 → acceleration of the leading axis (shoulder in this case)
  - t = 0.5 s → time before the function returns

As explained in [8] the variable “t” is the time in which the function returns.

The values are multiple of 8ms for simplicity, because 8ms is the best frame rate provided by the UR3. For the test t=0.008s, t=0.016 and t=0.024 was chosen.

t=0.008

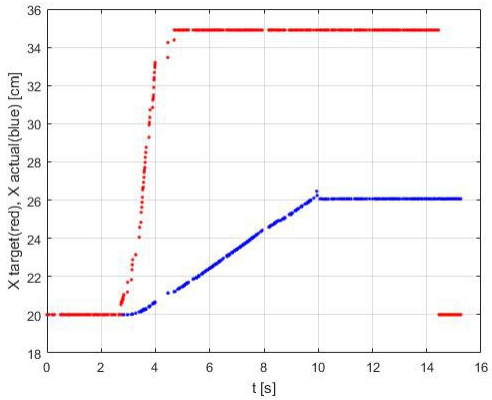


Fig 7.29 X(t) for Test_SliderAndSpeed.m

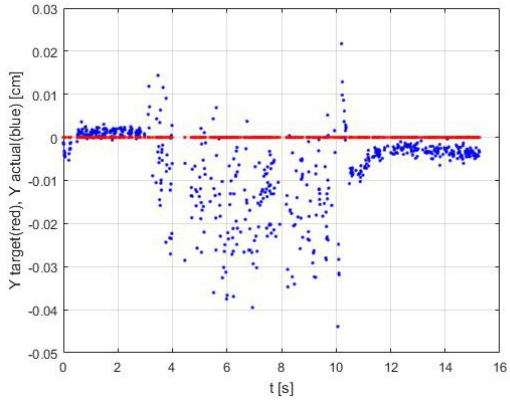


Fig 7.30 Y(t) for Test_SliderAndSpeed.m

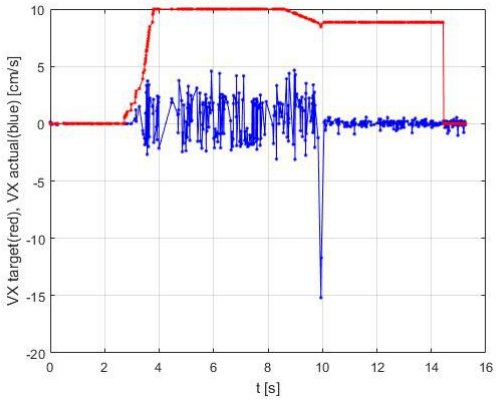


Fig 7.31 VX(t) for Test_SliderAndSpeed.m

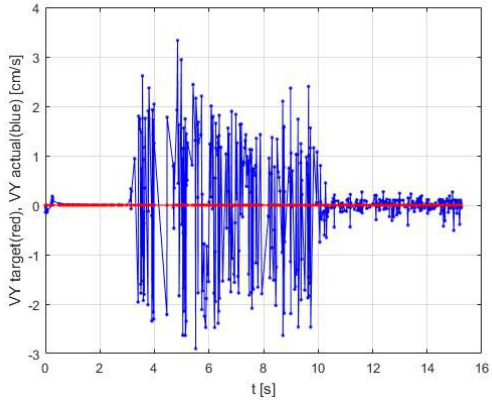
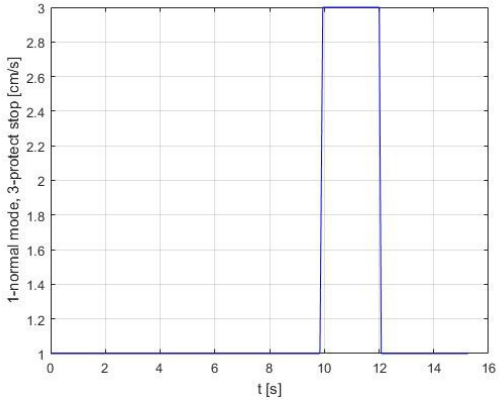


Fig 7.32 VY(t) for Test_SliderAndSpeed.m



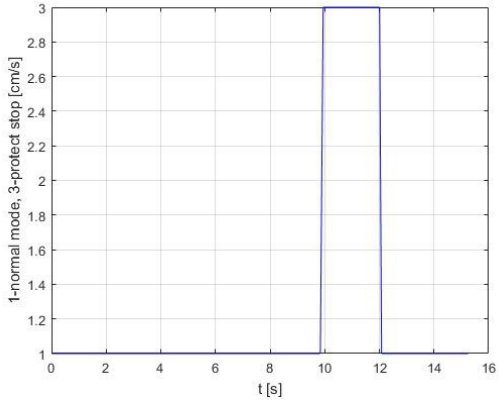


Fig 7.33 ERROR(t) for Test_SliderAndSpeed.m

With t=0.008s for a easy movement only on coordinate X the robot is not able to reach the position (fig7.29). It is possible to see a very nervous behavior on speed (fig7.31, 7.32) and around second 10 the URCap goes in protection stop (fig7.33).

The reason why it happens is not well known but a possible answer is that the refresh of command provided by the computer is too low respect to 8ms. Indeed measuring the refresh data of the computer we obtain:

$$\bar{t} = (t_{max} - t_{min})/n = 0.0256$$ where n=number of samples

We know the time thank to the collected data, indeed the first value provided by “streamRTCI” is the time of the robot. Although the RTCI provides data at 8ms (125Hz) the average time is not 8ms because using the “RT” as input in “streamRTCI” the function delete old data values to be sure to have always the last data.

t=0.016

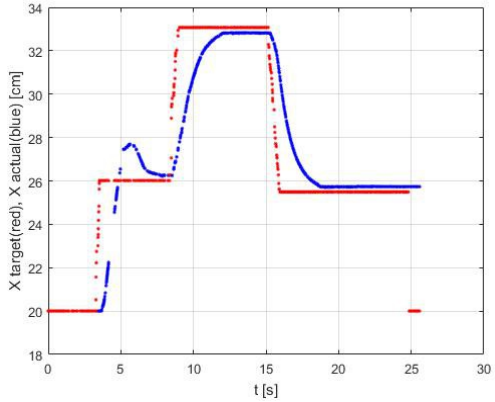


Fig 7.34 X(t) for Test_SliderAndSpeed.m

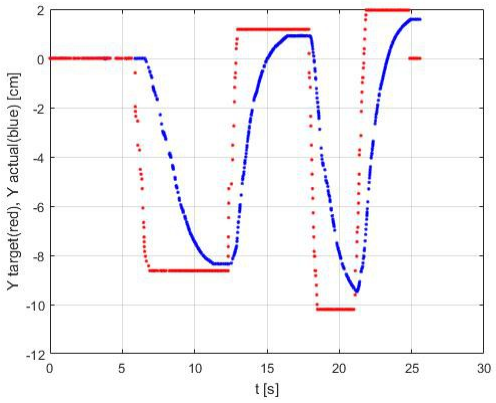


Fig 7.35 Y(t) for Test_SliderAndSpeed.m

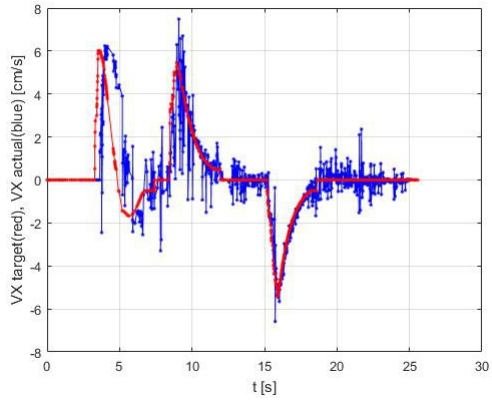


Fig 7.36 VX(t) for Test_SliderAndSpeed.m

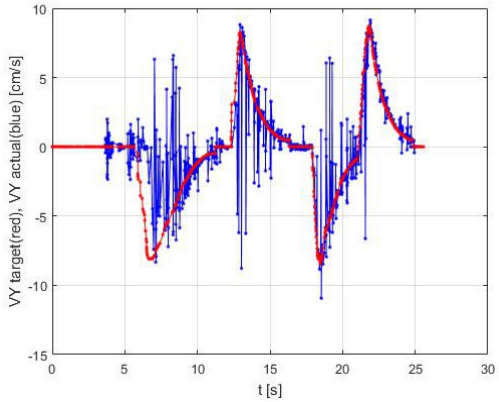


Fig 7.37 VY(t) for Test_SliderAndSpeed.m

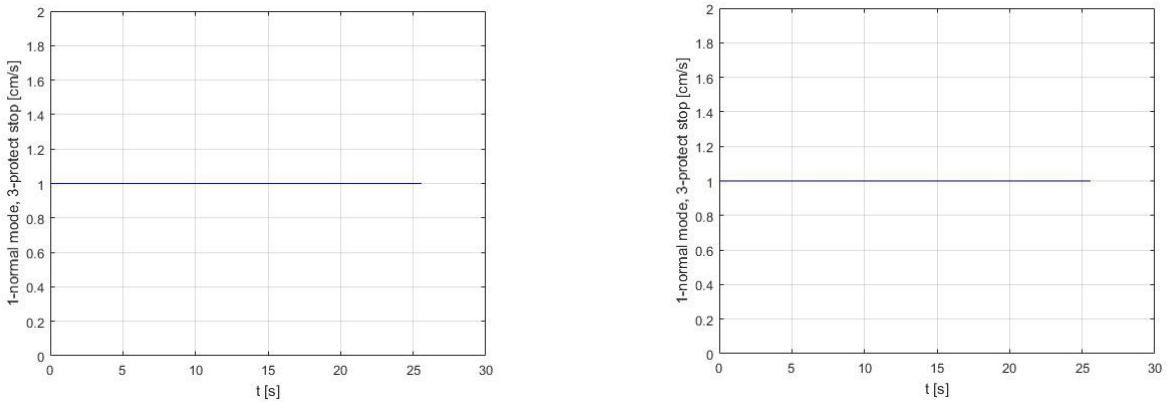


Fig 7.38 ERROR(t) for Test_SliderAndSpeed.m

With t=0.016 as speedl input the program works pretty well although sometimes can go in protection stop.

Also in this case the average time is around three times the nominal working period. Tm=0.0258.

It is reasonable that a better and more complex feedback function could improve a lot the stability and robustness of all the program.

t=0.024

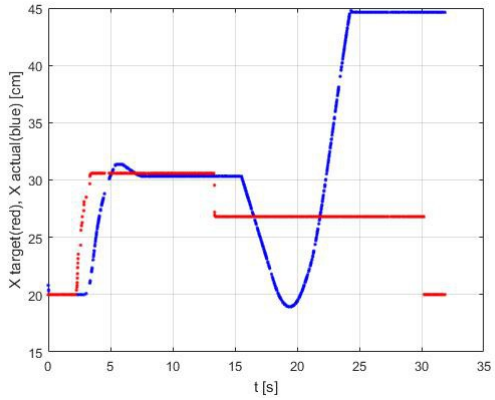


Fig 7.39 X(t) for Test_SliderAndSpeed.m

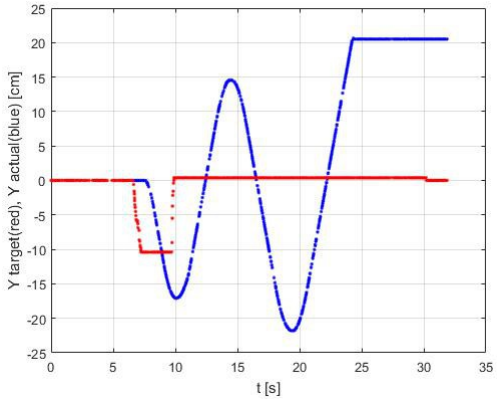


Fig 7.40 Y(t) for Test_SliderAndSpeed.m

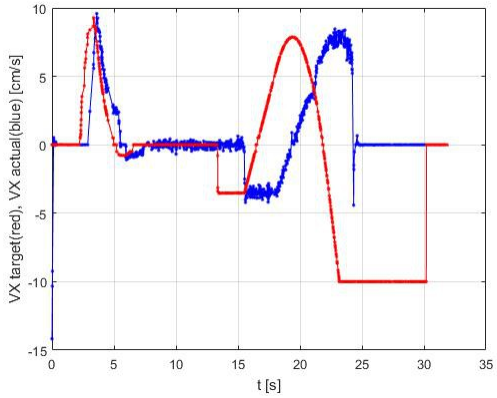


Fig 7.41 VX(t) for Test_SliderAndSpeed.m

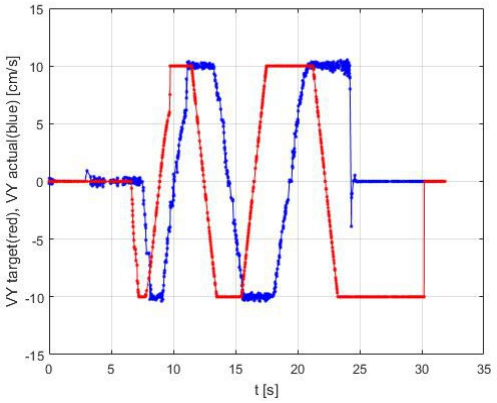
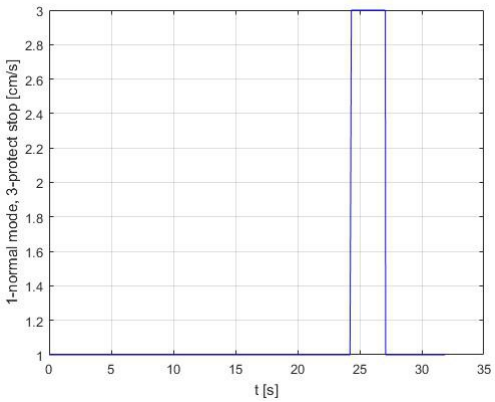


Fig 7.42 VY(t) for Test_SliderAndSpeed.m



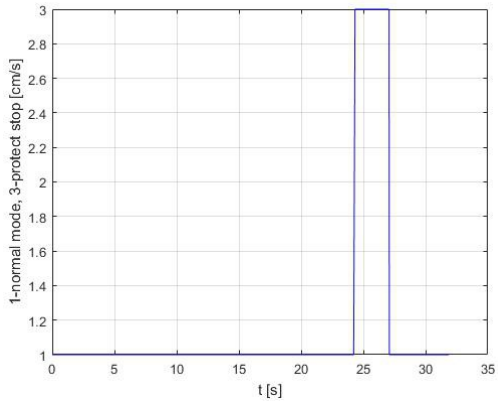


Fig 7.43 ERROR(t) for Test_SliderAndSpeed.m

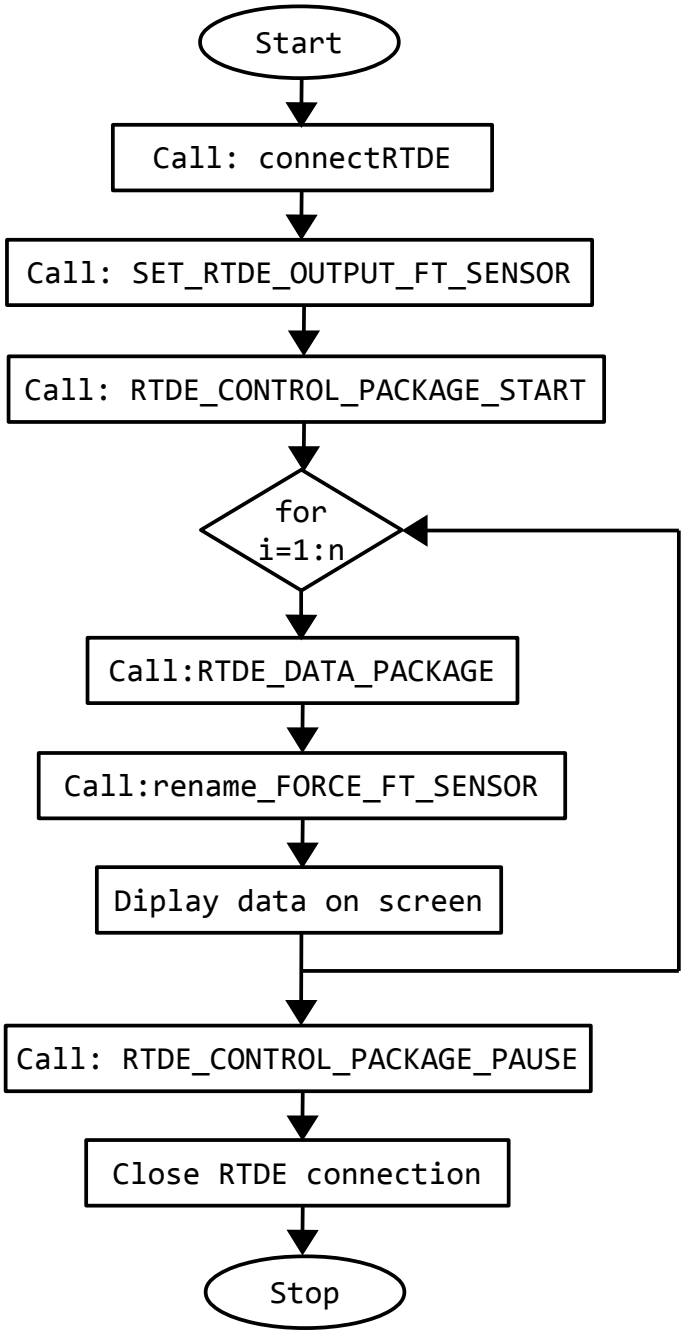
With t=0.024s the position becomes instable (fig7.39, 7.40) indeed even if the command speed follows the shape of the target speed increase the delay until the robot goes in protective stop fig(7.43) because reach a complete extended configuration. Also in this third test the average time is close to the previous with a tm=0.0257.

Test-4 – Read FT sensor

Test_read_sensor – Program description

This last test show how to use RTDE client interface to read the data of the force sensor FT 300. As described in chapter 6 the RTDE client interface use owns protocol to select inputs and outputs and start or close the streaming of data. Taking in consideration “Test_read_sensor.m” written in Matlab.

Test_read_sensor – Flow chart



Test_read_sensor – Comment

In figure 7.44 is showed an example of output data for n=1.

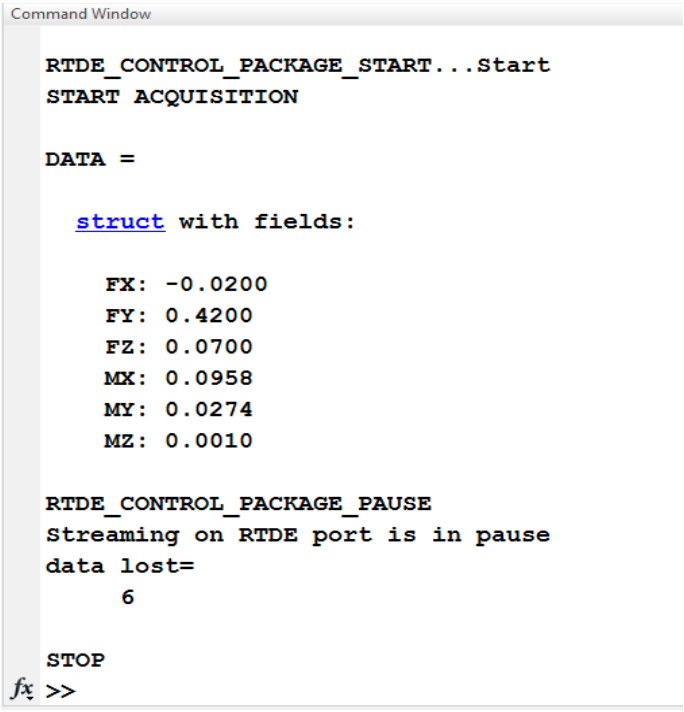


Fig 7.44 Display output of Test_read_sensor.m

8 – Conclusions

The work performed for this thesis was useful for many different facets.

First of all allows me to learn new skills and knowledge about computer science. I studied the UR3, the network connection protocols, I improved my knowledge in Matlab and problem solving and I learn to program in Python and URScript.

Even if my work is not yet integrated in the system of controlling of the robot, is ready to be used to it.

It is also important for the knowledge itself of the cobot that could be in a future used for other projects and educational demonstration.

To continue from this point could be interesting implement the libraries in the main project and study better the potentiality of the RTDE client interface that unfortunately I start to study and to know at the end of this journey.

Could be also interesting understand better the computation power used by Python respect to Matlab. Indeed even if I see better performances using Python I don't have extract any data that allow an objective comparison.

Bibliography

[1] Mathworks site: www.mathworks.com

[2] Phyton site: www.python.org

[3] Anaconda site: www.anaconda.com

[4] Micha Gorelick, Ian Ozsvald, “Python alla massima potenza”, Hoeply, 2015

[5] PyMike YouTube channel to learn Python: https://www.youtube.com/channel/UCLXlzAu0NhM5pnXLQdGnHJQ

[6] Universal Robot site: www.universal-robots.com

[7] Universal Robot, “User Manual UR3/CB3”, Version 3.4.5

[8] Universal Robot, “The URScript Programming Language”, Version 3.4.5, 2017

[9] Universal robot RTDE information: https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/real-time-data-exchange-rtde-guide-22229/

[10] Universal Robot file for client interface information, “Client_Interface_V3.5.xlsx”

[11] Zacobria site to learn UR3 in depth: http://www.zacobria.com/

[12] Robotiq site: https://robotiq.com/

[13] “Robotiq 2F-85 2F-140”, Original Notice, © 2018 Robotic Inc.

[14] “Robotiq FT 300 Force Torque Sensor”, © 2018 Robotic Inc.

Appendix-1 Matlab UR Library

```
function [ URCI ] = connectURCI( IP_UR, port )

URCI=tcpip(IP_UR,port);%connection with UR3 as server and
matlab as client
fopen(URCI);
end


function [PCI]=connectPCI(IP_UR)

port=30001; %real time port 10Hz
PCI=tcpip(IP_UR,port,'InputBufferSize',1108);%connection with
UR3 as server and matlab as client
fopen(PCI);
disp("Connection to PCI client interface")
end


function [SCI]=connectSCI(IP_UR)

port=30002; %real time port 10Hz
SCI=tcpip(IP_UR,port,'InputBufferSize',1108);%connection with
UR3 as server and matlab as client
fopen(SCI);
disp("Connection to SCI client interface")
end


function [RTCI]=connectRTCI(IP_UR)

port=30003; %real time port 125Hz
RTCI=tcpip(IP_UR,port,'InputBufferSize',1108);%connection with
UR3 as server and matlab as client
fopen(RTCI);
disp("Connection to RTCI client interface")
end


function [RTDE]=connectRTDE(IP_UR)

port=30004; %real time port 125Hz
RTDE=tcpip(IP_UR,port,'InputBufferSize',4096);%connection with
UR3 as server and matlab as client
fopen(RTDE);
disp("Connection to RTDE client interface")
end


function [AS2C]=connectAS2C(IP_UR,port_PC)

disp("Waiting for client...");
```

```
AS2C=tcpip(IP_UR, port_PC, 'NetworkRole', 'server');
fopen(AS2C);
disp("Connected");
end


function [data] = streamRTCI(RTCI,varargin)

%% Function for elaborate streaming data from UR3.

% Input control
if nargin>2
    disp('ERROR: too many inputs');disp('The only accempted
input is RT.');
    return
elseif nargin==2
    if strcmp(varargin{1},'RT')
        RT=true;
    else
        disp('ERROR: Input not valid.');disp('The only
accepted optional input is RT');
        return
    end
else
    RT=false;
end

%% First data lecture
s1=fread(RTCI,1,'int');%dimension of bytes receved
%% Bytes control
if s1~=1108
    disp('ERROR: Number of byte different to 1108. Verify if
the version of the Polyscope is v3.5');
    return
end
data=fread(RTCI,138,'double');%datas reved

%% RealTime mode implementation
if RT
    i=1;
    while RTCI.BytesAvailable>0
        s1=fread(RTCI,1,'int');%dimension of bytes receved
        data=fread(RTCI,138,'double');%datas reved
        i=i+1;
    end
   % i
end
end


function [DATA] = renameRTCI(data)
```

```
%% Function for rename streaming data from UR3.
s2=data;

%% Renamed datas
f1='Time'; v1=s2(1); %Time elapsed since the controller was
started
f2='q_target'; v2=s2(2:7); %target joint positions [rad]
f3='qd_target'; v3=s2(8:13); %target joint velocities in
[rad/s]
f4='qdd_target'; v4=s2(14:19); %target joint accelerations in
[rad/s^2]
f5='I_target'; v5=s2(20:25); %target joint currents
f6='M_target'; v6=s2(26:31); %target joint torques
f7='q_actual'; v7=s2(32:37); %actual joint positions [rad]
f8='qd_actual'; v8=s2(38:43); %actual joint velocities [rad/s]
f9='I_actual'; v9=s2(44:49); %actual joint currents
f10='I_control'; v10=s2(50:55); %joint control currents
f11='Tool_vector_actual'; v11=s2(56:61); %actual cartesian
coordinates of the tool: (x[m],y[m],z[m],rx,ry,rz), where rx,
ry and rz is a rotation vector representation of the tool
orientation
f12='TCP_speed_actual'; v12=s2(62:67); %actual speed of the
tool given in Cartesian coordinates
f13='TCP_force'; v13=s2(68:73); %Generalised forces in the TCP
f14='Tool_vector_target'; v14=s2(74:79); %Target Cartesian
coordinates of the tool: (x,y,z,rx,ry,rz), where rx, ry and rz
is a rotation vector representation of the tool orientation
f15='TCP_speed_target'; v15=s2(80:85); %Target speed of the
tool given in Cartesian coordinates
f16='Digital_input_bits'; v16=s2(86); %Current state of the
digital inputs. NOTE: these are bits encoded as int64_t, e.g.
a value of 5 corresponds to bit 0 and bit 2 set high
f17='Motor_temperatures'; v17=s2(87:92); %Temperature of each
joint in degrees celsius
f18='Controller_timer'; v18=s2(93); %Controller realtime
thread execution time
%Test_value=s2(94); A value used by Universal Robots software
only
Robot_mode=s2(95); %Robot mode
switch Robot_mode
    case 0
        Robot_mode_message='ROBOT_MODE_DISCONNECTED';
    case 1
        Robot_mode_message='ROBOT_MODE_CONFIRM_SAFETY';
    case 2
        Robot_mode_message='ROBOT_MODE_BOOTING';
    case 3
        Robot_mode_message='ROBOT_MODE_POWER_OFF';
    case 4
        Robot_mode_message='ROBOT_MODE_POWER_ON';
    case 5
```

```
        Robot_mode_message='ROBOT_MODE_IDLE';
    case 6
        Robot_mode_message='ROBOT_MODE_BACKDRIVE';
    case 7
        Robot_mode_message='ROBOT_MODE_RUNNING';
    case 8
        Robot_mode_message='ROBOT_MODE_UPDATING_FIRMWARE';
    otherwise
        Robot_mode_message='ROBOT_MODE_ERROR_VALUE';
end
f19='Robot_mode_message'; v19=Robot_mode_message;
Joint_modes=s2(96:101); %Joint control modes
for i=1:6
    switch Joint_modes(i)
        case 236

Joint_modes_message(i,:)='JOINT_SHUTTING_DOWN_MODE';
        case 237

Joint_modes_message(i,:)='JOINT_PART_D_CALIBRATION_MODE';
        case 238
            Joint_modes_message(i,:)='JOINT_BACKDRIVE_MODE';
        case 239
            Joint_modes_message(i,:)='JOINT_POWER_OFF_MODE';
        case 245

Joint_modes_message(i,:)='JOINT_NOT_RESPONDING_MODE';
        case 246

Joint_modes_message(i,:)='JOINT_MOTOR_INITIALISATION_MODE';
        case 247
            Joint_modes_message(i,:)='JOINT_BOOTING_MODE';
        case 248

Joint_modes_message(i,:)='JOINT_PART_D_CALIBRATION_ERROR_MODE'
;
        case 249
            Joint_modes_message(i,:)='JOINT_BOOTLOADER_MODE';
        case 250
            Joint_modes_message(i,:)='JOINT_CALIBRATION_MODE';
        case 252
            Joint_modes_message(i,:)='JOINT_FAULT_MODE';
        case 253
            Joint_modes_message(i,:)='JOINT_RUNNING_MODE';
        case 255
            Joint_modes_message(i,:)='JOINT_IDLE_MODE';
        otherwise
            Joint_modes_message(i,:)='JOINT_ERROR_VALUE';
    end
end
f20='Joint_modes_message'; v20=Joint_modes_message;
```

```
Safety_mode=s2(102); %Safety mode
switch Safety_mode
    case 1
        Safety_mode_message='SAFETY_MODE_NORMAL';
    case 2
        Safety_mode_message='SAFETY_MODE_REDUCED';
    case 3
        Safety_mode_message='SAFETY_MODE_PROTECT_STOP';
    case 4
        Safety_mode_message='SAFETY_MODE_RECOVERY';
    case 5
        Safety_mode_message='SAFETY_MODE_SAFEGUARD_STOP';
    case 6

Safety_mode_message='SAFETY_MODE_SYSTEM_EMERGENCY_STOP';
    case 7

Safety_mode_message='SAFETY_MODE_ROBOT_EMERGENCY_STOP';
    case 8
        Safety_mode_message='SAFETY_MODE_VIOLATION';
    case 9
        Safety_mode_message='SAFETY_MODE_FAULT';
    otherwise
        Safety_mode_message='SAFETY_MODE_ERRORE_VALUE';
end
f21='Safety_mode_message'; v21=Safety_mode_message;
%s2(103:108) Used by Universal Robots software only
f22='Tool_accelerometer_values'; v22=s2(109:111); %Tool x,y
and z accelerometer values
%s2(112:117) Used by Universal Robots software only
f23='Speed_scaling'; v23=s2(118); %Speed scaling of the
trajectory limiter
f24='Linear_momentum_norm'; v24=s2(119); %Norm of Cartesian
linear momentum
%s2(120) Used by Universal Robots software only
%s2(121) Used by Universal Robots software only
f25='V_main'; v25=s2(122); %Masterboard: Main voltage
f26='V_robot'; v26=s2(123); %Masterboard: Robot voltage (48V)
f27='I_robot'; v27=s2(124); %Masterboard: Robot current
f28='V_actual'; v28=s2(125:130); %Actual joint voltages
f29='Digital_outputs'; v29=s2(131); %Digital outputs
f30='Program_state'; v30=s2(132); %Program state
f31='Elbow_position'; v31=s2(133:135); %Elbow position
f32='Elbow_velocity'; v32=s2(136:138); %Elbow velocity

DATA=struct(f1,v1,f2,v2,f3,v3,f4,v4,f5,v5,f6,v6,f7,v7,f8,v8,f9
,v9,f10,v10,...

f11,v11,f12,v12,f13,v13,f14,v14,f15,v15,f16,v16,f17,v17,f18,v1
8,f19,v19,f20,v20,...
```

```
f21,v21,f22,v22,f23,v23,f24,v24,f25,v25,f26,v26,f27,v27,f28,v2
8,f29,v29,f30,v30,...
    f31,v31,f32,v32);
end


function movec(SOCKET,pose_via,pose_to,a,v,varargin)

%% Inizialization
Error=false;
r=0;
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
sp1=size(pose_via);
if sp1(1)~=1
    sp1=sp1(1);
elseif sp1(2)~=1
    sp1=sp1(2);
else
    sp1=0;
end
sp2=size(pose_via);
if sp2(1)~=1
    sp2=sp2(1);
elseif sp2(2)~=1
    sp2=sp2(2);
else
    sp2=0;
end
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif ~(isnumeric(v))
    disp("ERROR: velocity must be a number");
    Error=true;
elseif (sp1~=6||sp2~=6||(~isnumeric(pose_via))||
(~isnumeric(pose_via)))
    disp("ERROR: poses must be vectors of six elements");
    Error=true;
elseif dim>4
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif (dim==1||dim==3)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if ((i==1||i==3) && ~Error)
            if strcmp(varargin{i},"r")
                if isnumeric(varargin{i+1})
```

```
                    if varargin{i+1}>=0
                        r=varargin{i+1};
                    else
                        disp("ERROR: r cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of r must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
end
if Error
    disp("movec require:");
    disp("-SOCKET where write command");
    disp("    -pose_via that is a vector of six pose values");
    disp("    -pose_to that is a vector of six pose values");
    disp("    -a that is a number corresponding to
acceleration");
    disp("    -v that is a number corresponding to velocity");
    disp("    -optional arguments");

disp("-----------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'r',value (0 as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="movec(p[";
```

```
    middle="],p[";
    endmove="]";
else
    strstart="(1,";
    middle=",";
    endmove="";
end
command=strstart+num2str(pose_via(1))+','+num2str(pose_via(2))
+','+...
    num2str(pose_via(3))+','+num2str(pose_via(4))
+','+num2str(pose_via(5))+...
    ','+num2str(pose_via(6))+middle+num2str(pose_to(1))
+','+...
    num2str(pose_to(2))+','+num2str(pose_to(3))
+','+num2str(pose_to(4))+...
    ','+num2str(pose_to(5))+','+num2str(pose_to(6))
+endmove+','+...
    num2str(a)+','+num2str(v)+','+num2str(r)+')';
disp(command); %Only for testing purpose
fprintf(SOCKET,'%s\n',command);
end


function movej(SOCKET,q,a,v,varargin)

%% Inizialization
Error=false;
t=0;
r=0;
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
sq=size(q);
if sq(1)~=1
    sq=sq(1);
elseif sq(2)~=1
    sq=sq(2);
else
    sq=0;
end
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif ~(isnumeric(v))
    disp("ERROR: velocity must be a number");
    Error=true;
elseif (sq~=6||(~isnumeric(q)))
    disp("ERROR: angles must be a vector of six elements");
    Error=true;
elseif dim>6
    disp("ERROR: Too many optional arguments");
```

```
    Error=true;
elseif (dim==1||dim==3||dim==5)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if (i==1||i==3||i==5 && ~Error)
            if strcmp(varargin{i},"t")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        t=varargin{i+1};
                    else
                        disp("ERROR: t cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of t must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"r")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        r=varargin{i+1};
                    else
                        disp("ERROR: r cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of r must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
```

```
end
if Error
    disp("movej require:");
    disp("-SOCKET where write command");
    disp("    -q that is a vector of six angles");
    disp("    -a that is a number corresponding to
acceleration");
    disp("    -v that is a number corresponding to velocity");
    disp("    -optional arguments");

disp("-----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'t',value (0 as default)");
    disp("    -'r',value (0 as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="movej([";
    endmove="]";
else
    strstart="(2,";
    endmove="";
end
command=strstart+num2str(q(1))+','+num2str(q(2))
+','+num2str(q(3))+','+...
    num2str(q(4))+','+num2str(q(5))+','+num2str(q(6))
+endmove+','+...
    num2str(a)+','+num2str(v)+','+num2str(t)+','+num2str(r)
+')';
disp(command); %Only for testing purpose
fprintf(SOCKET,'%s\n',command);
end

function movel(SOCKET,pose,a,v,varargin)

%% Inizialization
Error=false;
t=0;
r=0;
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
sq=size(pose);
if sq(1)~=1
    sq=sq(1);
```

```
elseif sq(2)~=1
    sq=sq(2);
else
    sq=0;
end
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif ~(isnumeric(v))
    disp("ERROR: velocity must be a number");
    Error=true;
elseif (sq~=6||(~isnumeric(pose)))
    disp("ERROR: angles must be a vector of six elements");
    Error=true;
elseif dim>6
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif (dim==1||dim==3||dim==5)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if (i==1||i==3||i==5 && ~Error)
            if strcmp(varargin{i},"t")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        t=varargin{i+1};
                    else
                        disp("ERROR: t cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of t must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"r")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        r=varargin{i+1};
                    else
                        disp("ERROR: r cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of r must be a
number")
                    Error=true;
                end
```

```
            elseif strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
end
if Error
    disp("movel require:");
    disp("-SOCKET where write command");
    disp("    -pose that is a vector of six angles");
    disp("    -a that is a number corresponding to
acceleration");
    disp("    -v that is a number corresponding to velocity");
    disp("    -optional arguments");

disp("-----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'t',value (0 as default)");
    disp("    -'r',value (0 as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="movel(p[";
    endmove="]";
else
    strstart="(3,";
    endmove="";
end
command=strstart+num2str(pose(1))+','+num2str(pose(2))
+','+num2str(pose(3))+','+...
    num2str(pose(4))+','+num2str(pose(5))+','+num2str(pose(6))
+endmove+','+...
    num2str(a)+','+num2str(v)+','+num2str(t)+','+num2str(r)
+')';
disp(command); %Only for testing purpose
```

```
fprintf(SOCKET,'%s\n',command);
end


function movep(SOCKET,pose,a,v,varargin)
%% Inizialization
Error=false;
r=0;
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
sp=size(pose);
if sp(1)~=1
    sp=sp(1);
elseif sp(2)~=1
    sp=sp(2);
else
    sp=0;
end
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif ~(isnumeric(v))
    disp("ERROR: velocity must be a number");
    Error=true;
elseif (sp~=6||(~isnumeric(pose)))
    disp("ERROR: angles must be a vector of six elements");
    Error=true;
elseif dim>4
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif (dim==1||dim==3)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if ((i==1||i==3) && ~Error)
            if strcmp(varargin{i},"r")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        r=varargin{i+1};
                    else
                        disp("ERROR: r cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of r must be a
number")
```

```
                    Error=true;
                end
            elseif strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
end
if Error
    disp("movep require:");
    disp("-SOCKET where write command");
    disp("    -pose that is a vector of six angles");
    disp("    -a that is a number corresponding to
acceleration");
    disp("    -v that is a number corresponding to velocity");
    disp("    -optional arguments");

disp("-----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'r',value (0 as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="movep([";
    endmove="]";
else
    strstart="(4,";
    endmove="";
end
command=[strstart,num2str(pose(1)),',',num2str(pose(2)),',',nu
m2str(pose(3)),',',...

num2str(pose(4)),',',num2str(pose(5)),',',num2str(pose(6)),end
move,',',...
    num2str(a),',',num2str(v),',',num2str(r),')'];
```

```
%fprintf(SOCKET,'%s\n',command);
disp(command); %Only for testing purpose
end

function speedj(SOCKET,speed,a,varargin)
%% Inizialization
Error=false;
t=0;
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
sp=size(speed);
if sp(1)~=1
    sp=sp(1);
elseif sp(2)~=1
    sp=sp(2);
else
    sp=0;
end
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif (sp~=6||(~isnumeric(speed)))
    disp("ERROR: speed must be a vector of six elements");
    Error=true;
elseif dim>4
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif (dim==1||dim==3)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if ((i==1||i==3) && ~Error)
            if strcmp(varargin{i},"t")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        t=varargin{i+1};
                    else
                        disp("ERROR: t cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of t must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
```

```
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
end
if Error
    disp("speedj require:");
    disp("-SOCKET where write command");
    disp("    -speed that is a vector of six values");
    disp("    -v that is a number corresponding to velocity");
    disp("    -optional arguments");

disp("----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'t',value (0 as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="speedj([";
    endmove="]";
else
    strstart="(5,";
    endmove="";
end
command=strstart+num2str(speed(1))+','+num2str(speed(2))
+','+num2str(speed(3))+','+...
    num2str(speed(4))+','+num2str(speed(5))
+','+num2str(speed(6))+endmove+','+...
    num2str(a)+','+num2str(t)+')';
fprintf(SOCKET,'%s\n',command);
%disp(command); %Only for testing purpose
end

function speedl(SOCKET,speed,a,varargin)

%% Inizialization
```

```
Error=false;
t=0;
aRot="a";
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
sp=size(speed);
if sp(1)~=1
    sp=sp(1);
elseif sp(2)~=1
    sp=sp(2);
else
    sp=0;
end
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif (sp~=6||(~isnumeric(speed)))
    disp("ERROR: angles must be a vector of six elements");
    Error=true;
elseif dim>6
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif (dim==1||dim==3||dim==5)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if (i==1||i==3||i==5 && ~Error)
            if strcmp(varargin{i},"t")
                if isnumeric(varargin{i+1})
                    if varargin{i+1}>=0
                        t=varargin{i+1};
                    else
                        disp("ERROR: t cannot be less than
zero")
                        Error=true;
                    end
                else
                    disp("ERROR: the argument of t must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"aRot")
                if ~isalpha(varargin{i+1})
                    disp("ERROR: the argument of r must be a
number")
                    Error=true;
                end
            elseif strcmp(varargin{i},"SocketType")
```

```
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
end
if Error
    disp("speedl require:");
    disp("-SOCKET where write command");
    disp("    -speed that is a vector of six values");
    disp("    -v that is a number corresponding to velocity");
    disp("    -optional arguments");

disp("----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'t',value (0 as default)");
    disp("    -'aRot',value ('a' as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="speedl([";
    endmove="]";
else
    strstart="(6,";
    endmove="";
end
command=strstart+num2str(speed(1))+','+num2str(speed(2))
+','+num2str(speed(3))+','+...
    num2str(speed(4))+','+num2str(speed(5))
+','+num2str(speed(6))+endmove+','+...
    num2str(a)+','+num2str(t)+')';
fprintf(SOCKET,'%s\n',command);
disp(command); %Only for testing purpose
end

function stopj(SOCKET,a,varargin)
```

```
%% Inizialization
Error=false;
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif dim>2
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif dim==1
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if (i==1 && ~Error)
            if strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
                Error=true;
            end
        end
    end
end
if Error
    disp("stopj require:");
    disp("-SOCKET where write command");
    disp("    -a that is a number corresponding to
acceleration");
    disp("    -optional arguments");

disp("-----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
```

```
if strcmp(SocketType,"CI")
    strstart="stopj(";
else
    strstart="(7,";
end
command=strstart+num2str(a)+')';
fprintf(SOCKET,'%s\n',command);
%disp(command); %Only for testing purpose
end

function stopl(SOCKET,a,varargin)

%% Inizialization
Error=false;
aRot="a";
SocketType="CI";
%% Input control
dim=size(varargin);
dim=dim(2);
if ~(isnumeric(a))
    disp("ERROR: acceleration must be a number");
    Error=true;
elseif dim>4
    disp("ERROR: Too many optional arguments");
    Error=true;
elseif (dim==1||dim==3)
    disp("ERROR: no coupled arguments");
    Error=true;
else
    for i=1:dim
        if ((i==1||i==3) && ~Error)
            if strcmp(varargin{i},"aRot")
                if ~ischar(varargin{i+1})
                    disp("ERROR: the value of aRot must be a
letter")
                    Error=true;
                else
                    aRot="varargin{i+1}";
                end
            elseif strcmp(varargin{i},"SocketType")
                if strcmp(varargin{i+1},"CI")
                    SocketType="CI";
                elseif strcmp(varargin{i+1},"Server")
                    SocketType="Server";
                else
                    disp("ERROR: the argument of SocketType is
not correct")
                    Error=true;
                end
            else
                disp("ERROR: no correct argument")
```

```
                Error=true;
            end
        end
    end
end
if Error
    disp("stopl require:");
    disp("-SOCKET where write command");
    disp("    -a that is a number corresponding to
acceleration");
    disp("    -optional arguments");

disp("-----------------------------------------------------------
------");
    disp("Optional arguments must be couple of the following
type:");
    disp("    -'aRot',value ('a' as default)");
    disp("    -'SocketType','CI' or 'Server' ('CI' as
default)");
    return
end
%% Send message
if strcmp(SocketType,"CI")
    strstart="stopl(";
else
    strstart="(8,";
end
command=strstart+num2str(a)+')';
fprintf(SOCKET,'%s\n',command);
%disp(command); %Only for testing purpose
end

function [ version ] =
RTDE_REQUEST_PROTOCOL_VERSION( RTDE,varargin )
%Initialization
disp("RTDE_REQUEST_PROTOCOL_VERSION...Start")
dim=size(varargin);
dim=dim(2);
if dim==0 || dim==1 && varargin(1)==2
    version=2;
elseif dim==1 && varargin(1)==1
    version=1;
else
    disp("ERROR: The version can be 1 or 2 or nothing. If
version is not defined, version 2 is choosen as default.")
    version=-1;
    return
end
%Sending message
fwrite(RTDE,5,'uint16');
fwrite(RTDE,86,'uint8');
```

```
fwrite(RTDE,2,'uint16');
%Reading responce
nbyte=fread(RTDE,1,'uint16');
if nbyte~=4
    disp("ERROR: Number of bytes different from predicted.
Control that the socket used is RTDE")
    version=-1;
    return
end
command=fread(RTDE,1,'uint8');
if command~=86
    disp("ERROR: Command different from predicted. Control
that the socket used is RTDE")
    version=-1;
    return
end
check=fread(RTDE,1,'uint8');
if check~=1
    disp("ERROR: Check different from predicted. Control that
the socket used is RTDE")
    version=-1;
    return
end
disp("RTDE is using protocol version: ");
disp(version);
end

function [id_recipe, recipe] =
RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS(RTDE,variables,varargin)
%Initialization
size_msg=3;
size_payload=0;
payload="";
recipe_string="";
id_recipe=0;
disp("RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS...Start")
%Control on input
dim=size(varargin);
dim=dim(2);

if dim==0
    version=2;
    frequency=125;
elseif dim==1 || dim==2
    if ~(varargin{1}==1 || varargin{1}==2)
        disp("ERROR: The version can be 1 or 2 or nothing. If
version is not defined, version 2 is choosen as default.")
        return
    else
        version=varargin{1};
        frequency=125;
```

```
    end
elseif dim==2
    if varargin(2)<1 || varargin(2)>125
        disp("ERROR: The frequency(Hz) must be between 1 and
125. If frequency is not defined, frequency 125Hz is choosen
as default.")
        return
    else
        frequency=varargin{2};
    end
else
    disp("ERROR: The version can be 1 or 2 or nothing. If
version is not defined, version 2 is choosen as default.")
    return
end
%Dimension payload
nvar=size(variables);
nvar=nvar(2);
for i=1:nvar
    if i<nvar
        payload=payload+variables(i)+",";
        size_payload=size_payload+strlength(variables(i))+1;
    else
        payload=payload+variables(i);
        size_payload=size_payload+strlength(variables(i));
    end
end
size_msg=size_msg+size_payload;
%Setup Outputs
if version==2
    size_msg=size_msg+8;
    fwrite(RTDE,size_msg,'uint16');
    fwrite(RTDE,79,'uint8');
    fwrite(RTDE,frequency,'double');
else
    fwrite(RTDE,size_msg,'uint16');
    fwrite(RTDE,79,'uint8');
end
fprintf(RTDE,'%s',payload);
%Read answer
nbyte=fread(RTDE,1,'uint16');
command=fread(RTDE,1,'uint8');
minus_flag=3;
if version==2
    id_recipe=fread(RTDE,1,'uint8');
    minus_flag=4;
end
output=fscanf(RTDE,'%c',nbyte-minus_flag);
%Recipe rename
dim_recipe=size(output);
dim_recipe=dim_recipe(2);
```

```
flag_start=1;
ingredient=1;
for i=1:dim_recipe
    if strcmp(output(i),',')
        recipe_string(ingredient)=output(flag_start:i-1);
        flag_start=i+1;
        ingredient=ingredient+1;
    elseif i==dim_recipe
        recipe_string(ingredient)=output(flag_start:i);
    end
end
recipe=cell(2,ingredient);
for i=1:ingredient
    if strcmp(recipe_string(i),'INT32')
        recipe{1,i}='int32';
        recipe{2,i}=1;
    elseif strcmp(recipe_string(i),'UINT32')
        recipe{1,i}='uint32';
        recipe{2,i}=1;
    elseif strcmp(recipe_string(i),'VECTOR6D')
        recipe{1,i}='double';
        recipe{2,i}=6;
    elseif strcmp(recipe_string(i),'VECTOR3D')
        recipe{1,i}='double';
        recipe{2,i}=3;
    elseif strcmp(recipe_string(i),'VECTOR6INT32')
        recipe{1,i}='int32';
        recipe{2,i}=6;
    elseif strcmp(recipe_string(i),'VECTOR6UINT32')
        recipe{1,i}='uint32';
        recipe{2,i}=6;
    elseif strcmp(recipe_string(i),'DOUBLE')
        recipe{1,i}='double';
        recipe{2,i}=1;
    elseif strcmp(recipe_string(i),'UINT64')
        recipe{1,i}='uint64';
        recipe{2,i}=1;
    elseif strcmp(recipe_string(i),'UINT8')
        recipe{1,i}='uint8';
        recipe{2,i}=1;
    end
end
end

function [ check ] = RTDE_CONTROL_PACKAGE_START( RTDE )
disp("RTDE_CONTROL_PACKAGE_START")
%Sends start command
fwrite(RTDE,3,'uint16')
fwrite(RTDE,83,'uint8')
%Reading responce
nbyte=fread(RTDE,1,'uint16')
```

```
command=fread(RTDE,1,'uint8')
check=fread(RTDE,1,'uint8')
if nbyte~=4
    disp("ERROR: unexpected number of bytes");
    check=-1;
    return
elseif command~=83
    disp("ERROR: unexpected command return");
    check=-1;
    return
elseif check~=1
    disp("ERROR: unexpected check");
    check=-1;
    return
end
disp("RTDE_CONTROL_PACKAGE_START...Start");
end

function [ check ] = RTDE_CONTROL_PACKAGE_PAUSE( RTDE )
k=1;
lost_data=-1;
disp("RTDE_CONTROL_PACKAGE_PAUSE")
%Sends start command
fwrite(RTDE,3,'uint16')
fwrite(RTDE,80,'uint8')
%Reading responce
while k==1
    lost_data=lost_data+1;
    nbyte=fread(RTDE,1,'uint16');
    command=fread(RTDE,1,'uint8');
    check=fread(RTDE,1,'uint8');
    if nbyte<4
        disp("ERROR: Somethings goes wrong in the steaming of
data")
        check=-1;
        disp("data lost= ");
        disp(lost_data);
        return;
    elseif nbyte>4
        fread(RTDE,nbyte-4,'uint8');
    elseif nbyte==4 && command==80
        disp("Streaming on RTDE port is in pause")
        k=0;
    end
end
disp("data lost= ");
disp(lost_data);
end


function [ data ] = RTDE_DATA_PACKAGE( RTDE, recipe)
```

```
dim_recipe=size(recipe);
dim_recipe=dim_recipe(2);
nval=0;
for i=1:dim_recipe
    nval=nval+recipe{2,i};
end
data=cell(1,nval);
k=1;
nbyte=fread(RTDE,1,'uint16');
command=fread(RTDE,1,'uint8');
id_recipe=fread(RTDE,1,'uint8');
for i=1:dim_recipe
    for j=1:recipe{2,i}
        data{1,k}=fread(RTDE,1,recipe{1,i});
        k=k+1;
    end
end
end
```

Appendix-2 Matlab Robotiq Library

```
function [ id_recipe,recipe ] =
SET_RTDE_OUTPUT_FT_SENSOR( RTDE,varargin)
%Controll on input
dim=size(varargin);
dim=dim(2);
if dim==0
    frequency=125;
elseif dim==1
    frequency=varargin(1);
else
    disp("ERROR: Too many input arguments");
end
%Initilization
variables=["output_double_register_0","output_double_register_
1","output_double_register_2","output_double_register_3","outp
ut_double_register_4","output_double_register_5"];
%Body
version=RTDE_REQUEST_PROTOCOL_VERSION(RTDE);
[id_recipe,recipe]=RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS(RTDE,var
iables,version,frequency);
end

function [ force ] = rename_FORCE_FT_SENSOR( data )
dim=size(data);
dim=dim(2);
if dim~=6
    disp("ERROR: the force vector must be of 6 elements");
    force=-1;
    return
end
f1='FX'; v1=data{1,1};
f2='FY'; v2=data{1,2};
f3='FZ'; v3=data{1,3};
f4='MX'; v4=data{1,4};
f5='MY'; v5=data{1,5};
f6='MZ'; v6=data{1,6};
force=struct(f1,v1,f2,v2,f3,v3,f4,v4,f5,v5,f6,v6);
end
```

Appendix-3 Python UR Library

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jul  5 14:38:47 2018
Last Update 06/11/2018

This is a library for controlling some features of Universar Robot
UR3,5,10
from PC.
For more informations to tipe UR.info()

@author: Matteo Gaidano
"""

#Initialising
import socket
import sys
import struct
import time

#Generic connection
def connectURCI(IP_UR,port):
    try:
        URCI=socket.socket()
        URCI.connect((IP_UR,port))
        print(f"Connection to UR3, IP: {IP_UR}, {port}")
        return URCI
    except socket.error as errore:
        print(f"ERRORE URTCI: {errore}")
        sys.exit()
```

```
#Connection to PCI (10Hz)
def connectPCI(IP_UR):
    PCI=connectURCI(IP_UR,30001)
    return PCI

#Connection to SCI (10Hz)
def connectSCI(IP_UR):
    SCI=connectURCI(IP_UR,30002)
    return SCI

#Connection to RTCI (125Hz)
def connectRTCI(IP_UR):
    RTCI=connectURCI(IP_UR,30003)
    return RTCI

#Connection to RTDE (125Hz)
def connectRTDE(IP_UR):
    RTDE=connectURCI(IP_UR,30004)
    return RTDE

#Creats a socket between PC and robot using PC as server, can works
till 125Hz.
#A corresponding script on UR is mandatory.
def connectAS2C(IP_PC,port,backlog=1):
    try:
        S_SERVER=socket.socket()
        S_SERVER.bind((IP_PC,port))
        S_SERVER.listen(backlog)
        print(f"Server with IP:{IP_PC} and port: {port} waiting...")
    except socket.error as error:
```

```
        print(f"ERROR connectAS2C: {error}")
        sys.exit()
    AS2C,IP_client=S_SERVER.accept()
    print(f"Connection with client: {IP_client}")
    return AS2C,S_SERVER


#Reads the streming of data provided by RTCI. Updated for
Polyscope3.5
def streamRTCI(RTCI):
    tstart=time.clock()
    data=[]
    dataB=RTCI.recv(1108)
    #ndata=int.from_bytes(dataB[0:4],byteorder='big',signed=False)
    #print(f"Numeber of receved bytes: {ndata}")
    for i in range(4,1108,8):
        temp=struct.unpack("!d",dataB[i:i+8])
        data.append(temp[0])
        #print(f"range {i}-{i+7}")
        #print(list(dataB[i:i+8]))
        #print(f"number of bytes {len(dataB[i:i+8])}")
        #print(" ")
    return data, time.clock()-tstart


#Renames output data from streamRTCI into a dictionary where each
data has a
#own name value
def renameRTCI(data):
    tstart=time.clock()
    datar={}
    datar["Time"]=data[0]
    datar["q_target"]=data[1:7]
    datar["qd_target"]=data[7:13]
```

```
    datar["qdd_target"]=data[13:19]
    datar["I_target"]=data[19:25]
    datar["M_target"]=data[25:31]
    datar["q_actual"]=data[31:37]
    datar["qd_actual"]=data[37:43]
    datar["I_actual"]=data[43:49]
    datar["I_control"]=data[49:55]
    datar["Tool_vector_actual"]=data[55:61]
    datar["TCP_speed_actual"]=data[61:67]
    datar["TCP_force"]=data[67:73]
    datar["Tool_vector_target"]=data[73:79]
    datar["TCP_speed_target"]=data[79:85]
    datar["Digital_input_bits"]=data[85]
    datar["Motor_temperatures"]=data[86:92]
    datar["Controller_timer"]=data[92]
    #Test_value for UR software only data[93]
    Robot_mode_message={
            0:"ROBOT_MODE_DISCONNECTED",
            1:"ROBOT_MODE_CONFIRM_SAFETY",
            2:"ROBOT_MODE_BOOTING",
            3:"ROBOT_MODE_POWER_OFF",
            4:"ROBOT_MODE_POWER_ON",
            5:"ROBOT_MODE_IDLE",
            6:"ROBOT_MODE_BACKDRIVE",
            7:"ROBOT_MODE_RUNNING",
            8:"ROBOT_MODE_UPDATING_FIRMWARE"}

datar["Robot_mode_message"]=Robot_mode_message.get(data[94],"ROBOT_M
ODE_ERROR_VALUE")
    for i in range(6):
        Joint_modes_message={
                236:"JOINT_SHUTTING_DOWN_MODE",
```

```
                237:"JOINT_PART_D_CALIBRATION_MODE",
                238:"JOINT_BACKDRIVE_MODE",
                239:"JOINT_POWER_OFF_MODE",
                245:"JOINT_NOT_RESPONDING_MODE",
                246:"JOINT_MOTOR_INITIALISATION",
                247:"JOINT_BOOTING_MODE",
                248:"JOINT_PART_D_CALIBRATION_ERROR_MODE",
                249:"JOINT_BOOTLOADER_MODE",
                250:"JOINT_CALIBRATION_MODE",
                252:"JOINT_FAULT_MODE",
                253:"JOINT_RUNNING_MODE",
                255:"JOINT_IDLE_MODE"}

datar[f"Joint_modes_message{i+1}"]=Joint_modes_message.get(data[95+i
],"JOINT_ERROR_VALUE")
    Safety_mode_message={
            1:"SAVETY_MODE_NORMAL",
            2:"SAFETY_MODE_REDUCED",
            3:"SAFETY_MODE_PROTECT_STOP",
            4:"SAFETY_MODE_RECOVERY",
            5:"SAFETY_MODE_SAFEGUARD_STOP",
            6:"SAFETY_MODE_SYSTEM_EMERGENCY_STOP",
            7:"SAFETY_MODE_ROBOT_EMERGENCY_STOP",
            8:"SAFETY_MODE_VIOLATION",
            9:"SAFETY_MODE_FAULT"}

datar["Safety_mode_message"]=Safety_mode_message.get(101,"SAFETY_MOD
E_ERROR_VALUE")
    #Used by Universal Robot software only data[102:108]
    datar["Tool_accelerometer_values"]=data[108:111]
    #Used by Universal Robot software only data[111:117]
    datar["Speed_scaling"]=data[117]
```

```
    datar["Linear_momentum_norm"]=data[118]
    #Used by Universal Robot software only data[119]
    #Used by Universal Robot software only data[120]
    datar["V_main"]=data[121]
    datar["V_robot"]=data[122]
    datar["I_robot"]=data[123]
    datar["V_actual"]=data[124:130]
    datar["Digital_outputs"]=data[130]
    datar["Program_state"]=data[131]
    datar["Elbow_position"]=data[132:135]
    datar["Elbow_velocity"]=data[135:138]
    #for i in range(len(datar)):
        #print(f"{list(datar.keys())[i]}:\n
{list(datar.values())[i]}\n")
    return datar, time.clock()-tstart


#This function comunicate through a socket for sends movec command
to UR3
def movec(SOCKET,pose_via,pose_to,a,v,r=0,SocketType="CI"):
    tstart=time.clock()
    print("movec start")
    if SocketType=="CI":
        strstart="movec(p["
        middle="],p["
        endmove="]"
    elif SocketType=="Server":
        strstart="(1,"
        middle=","
        endmove=""
    else:
        print("ERROR movec: Invalid input value. SocketType must be
CI or Server")
```

```
        return
    if r==0:
        r=""
    elif r>0:
        r=","+str(r)
    else:
        print("ERROR movec: Invalid input value. r cannot be
negative")


    b=(strstart+str(pose_via[0])+","+str(pose_via[1])
+","+str(pose_via[2])+","+str(pose_via[3])+","+str(pose_via[4])
+","+str(pose_via[5])+middle+str(pose_to[0])+","+str(pose_to[1])
+","+str(pose_to[2])+","+str(pose_to[3])+","+str(pose_to[4])
+","+str(pose_to[5])+endmove+","+str(a)+","+str(v)
+r+")\n").encode("ascii")
    print(b)
    SOCKET.send(b)
    return time.clock()-tstart


#This function comunicate through a socket for sends movej command
to UR3
def movej(SOCKET,q,a,v,t_in=0,r_in=0,SocketType="CI"):
    tstart=time.clock()
    if SocketType=="CI":
        strstart="movej(["
        endmove="]"
    elif SocketType=="Server":
        strstart="(2,"
        endmove=""
    else:
        print("ERROR movej: Invalid input value. SocketType must be
CI or Server")
        return
    if t_in==0:
```

```
        t=""
    elif t_in>0:
        t=","+str(t)
    else:
        print("ERROR movej: Invalid input value. t cannot be
negative")
    if r_in==0:
        r=""
    elif r_in>0:
        r=","+str(r)
    else:
        print("ERROR movej: Invalid input value. r cannot be
negative")


    b=(strstart+str(q[0])+","+str(q[1])+","+str(q[2])+","+str(q[3])
+","+str(q[4])+","+str(q[5])+endmove+","+str(a)+","+str(v)
+t+r+")\n").encode("ascii")
    SOCKET.send(b)
    print(b)
    return time.clock()-tstart


#This function comunicate through a socket for sends movel command
to UR3
def movel(SOCKET,pose,a,v,t=0,r=0,SocketType="CI"):
    tstart=time.clock()
    if SocketType=="CI":
        strstart="movel(p["
        endmove="]"
    elif SocketType=="Server":
        strstart="(3,"
        endmove=""
    else:
        print("ERROR movel: Invalid input value. SocketType must be
```

```
CI or Server")
        return
    if t==0:
        t=""
    elif t>0:
        t=","+str(t)
    else:
        print("ERROR movel: Invalid input value. t cannot be
negative")
    if r==0:
        r=""
    elif r>0:
        r=","+str(r)
    else:
        print("ERROR movel: Invalid input value. r cannot be
negative")


    b=(strstart+str(pose[0])+","+str(pose[1])+","+str(pose[2])
+","+str(pose[3])+","+str(pose[4])+","+str(pose[5])
+endmove+","+str(a)+","+str(v)+t+r+")\n").encode("ascii")
    print(b)
    SOCKET.send(b)
    return time.clock()-tstart


#This function comunicate through a socket for sends movep command
to UR3
def movep(SOCKET,pose,a,v,r=0,SocketType="CI"):
    tstart=time.clock()
    if SocketType=="CI":
        strstart="movep(p["
        endmove="]"
    elif SocketType=="Server":
        strstart="(4,"
```

```
        endmove=""
    else:
        print("ERROR movep: Invalid input value. SocketType must be
CI or Server")
        return
    if r==0:
        r=""
    elif r>0:
        r=","+str(r)
    else:
        print("ERROR movep: Invalid input value. r cannot be
negative")


    b=(strstart+str(pose[0])+","+str(pose[1])+","+str(pose[2])
+","+str(pose[3])+","+str(pose[4])+","+str(pose[5])
+endmove+","+str(a)+str(v)+r+")\n").encode("ascii")
    SOCKET.send(b)
    return time.clock()-tstart


#This function comunicate through a socket for sends speedj command
to UR3
def speedj(SOCKET,speed,a,t=0,SocketType="CI"):
    tstart=time.clock()
    if SocketType=="CI":
        strstart="speedj(["
        endspeed="]"
    elif SocketType=="Server":
        strstart="(5,"
        endspeed=""
    else:
        print("ERROR speedj: Invalid input value. SocketType must be
CI or Server")
        return
```

```
    if t==0:
        t=""
    elif t>0:
        t=","+str(t)
    else:
        print("ERROR speedj: Invalid input value. t cannot be
negative")
    b=(strstart+str(speed[0])+","+str(speed[1])+","+str(speed[2])
+","+str(speed[3])+","+str(speed[4])+","+str(speed[5])
+endspeed+","+str(a)+t+")\n").encode("ascii")
    SOCKET.send(b)
    print(b)
    return time.clock()-tstart


#This function comunicate through a socket for sends speedl command
to UR3
def speedl(SOCKET,speed,a,t=0,aRot="a",SocketType="CI"):
    tstart=time.clock()
    if SocketType=="CI":
        strstart="speedl(["
        endspeed="]"
    elif SocketType=="Server":
        strstart="(6,"
        endspeed=""
    else:
        print("ERROR speedl: Invalid input value. SocketType must be
CI or Server")
        return
    if t==0:
        t=""
    elif t>0:
        t=","+str(t)
    else:
```

```
        print("ERROR speedl: Invalid input value. t cannot be
negative")

    b=(strstart+str(speed[0])+","+str(speed[1])+","+str(speed[2])
+","+str(speed[3])+","+str(speed[4])+","+str(speed[5])
+endspeed+","+str(a)+t+")\n").encode("ascii")

    SOCKET.send(b)

    print(b)

    return time.clock()-tstart


#This function comunicate through a socket for sends stopj command
to UR3

def stopj(SOCKET,a,SocketType="CI"):

    tstart=time.clock()

    if SocketType=="CI":

        strstart="stopj("

    elif SocketType=="Server":

        strstart="(7,"

    else:

        print("ERROR stopj: Invalid input value. SocketType must be
CI or Server")

        return

    b=(strstart+str(a)+")\n").encode("ascii")

    SOCKET.send(b)

    print(b)

    return time.clock()-tstart


#This function comunicate through a socket to send stopl command to
UR3

def stopl(SOCKET,a,SocketType="CI"):

    tstart=time.clock()

    if SocketType=="CI":

        strstart="stopl("

    elif SocketType=="Server":
```

```
        strstart="(8,"
    else:
        print("ERROR stopl: Invalid input value. SocketType must be
CI or Server")
        return
    b=(strstart+str(a)+")\n").encode("ascii")
    SOCKET.send(b)
    print(b)
    return time.clock()-tstart


#This function comunicate through a socket to send halt command to
UR3
def halt(SOCKET,SocketType="CI"):
    if SocketType=="CI":
        b=("halt\n").encode("ascii")
    elif SocketType=="Server":
        b=("(-1)\n").encode("ascii")
    else:
        print("ERROR halt: Invalid input value. SocketType must be
CI or Server")
        return
    SOCKET.send(b)
    print(b)
    return


#Matlab Matrix File: rewrite in a txt file a list in a matlab matrix
form. This
#function born for transfer with copy and paste the data on matlab.
Better use
#csvF.py
def MMF(name,data):
    name=str(name)
    Data_file=open(name,"w")
```

```
    Data_file.close()
    Data_file=open(name,"a")
    Data_file.write("% Data from python code\n")
    Data_file.write(name+"=[")
    for i in range(len(data)):
        if type(data[0])==list:
            for j in range(len(data[0])):
                Data_file.write(str(data[i][j]))
                if j<len(data[0])-1:
                    Data_file.write(",")
                else:
                    Data_file.write(";")
        else:
            Data_file.write(str(data[i]))
            Data_file.write(",")
    Data_file.write("];\n    ")
    Data_file.close()


#csv File: rename in a generic comma separeted value. Readed by
exel, calc and
#matlab
def csvF(name,data,sepv=","):
    name=str(name)+".csv"
    Data_file=open(name,"w")
    Data_file.close()
    Data_file=open(name,"a")
    for i in range(len(data)):
        if type(data[0])==list:
            for j in range(len(data[0])):
                Data_file.write(str(data[i][j]))
                if j<len(data[0])-1:
```

```
                    Data_file.write(sepv)
                else:
                    Data_file.write("\n")
        else:
            Data_file.write(str(data[i]))
            Data_file.write(sepv)
    Data_file.close()


#Request protocol version at RTDE
def RTDE_REQUEST_PROTOCOL_VERSION(RTDE,version=2):
    if not(version==1 or version==2):
        print("There are only two versions: 1 or 2")
        #UR supports two version of streaming on 30004 RTDE port. As
default on
        #UR the first version is applied but for our purpous version
2 works
        #better so the default protocol version for this function is
the second
        return (-1)
    print("Request protocol version")
    #formattation and sending
    size=struct.calcsize("!HBH")
    mex=struct.pack("!HBH",size,86,version)
    RTDE.send(mex)
    #verification of UR responce
    respB=RTDE.recv(4096)
    resp1=struct.unpack("!HBB",respB[0:4])
    if (resp1[0]!=4 or resp1[1]!=86):
        print("ERROR: Error in the responce. Verify that the socket
is RTDE")
        return (-1)
    if resp1[2]!=1:
```

```
        print("ERROR: Version request not accepted, verify UR
version")
    return (version)


#Initializzation Outputs on RTDE
def
RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS(RTDE,variables,version=2,frequenc
y=125):
    #Control on version
    if not(version==1 or version==2):
        print("There are only two versions: 1 or 2")
        return (-1)
    #initializzation variables
    recipe=""
    frmt="!HB"
    if version==2:
        frmtr=frmt+"B"
        frmt=frmt+"d"
        flag=4
    else:
        frmtr=frmt
        flag=3
    #formattation and sending
    print("Output setting on RTDE")
    payload=(",".join(variables)).encode("ascii")
    size=struct.calcsize(frmt)+len(payload)
    if version ==1:
        mex=struct.pack(frmt,size,79)+payload
        #In version 1 a paylod composed by the variable of interest
is required
    else :
        mex=struct.pack(frmt,size,79,frequency)+payload
```

```
        #In version 2 is also possible set the streaming frequency,
in this
        #function 125Hz is setted as default
    RTDE.send(mex)
    time.sleep(0.1)
    #verification of UR responce
    respB=RTDE.recv(4096)
    resp1=struct.unpack(frmtr,respB[0:flag])
    if resp1[0]!=len(respB):
        print("ERROR: dimension mistmach. Verify that the socket is
RTDE")
    elif resp1[1]!=79:
        print("ERROR: Error in the responce. Verify that the socket
is RTDE")
    resp2=respB[flag:].split(b',')
    #creation of recipe formula for reading and convert streaming
data
    for i in resp2:
        if i==b'INT32':
            recipe += 'i'
        elif i==b'UINT32':
            recipe += 'I'
        elif i==b'VECTOR6D':
            recipe += 'd'*6
        elif i==b'VECTOR3D':
            recipe += 'd'*3
        elif i==b'VECTOR6INT32':
            recipe += 'i'*6
        elif i==b'VECTOR6UINT32':
            recipe += 'I'*6
        elif i==b'DOUBLE':
            recipe += 'd'
```

```
        elif i==b'UINT64':
            recipe += 'Q'
        elif i==b'UINT8':
            recipe += 'B'
    print("Output setted, the formatted output corresponds to:
",resp2)
    print("Output formatted: ",recipe)
    if version==1:
        return(recipe)
        #In version 1 only recipe is provided
    else:
        return(resp1[2],recipe)
        #In version 2 is also provided an id output setting


#Start streaming
def RTDE_CONTROL_PACKAGE_START(RTDE):
    #Starts the streaming of data from RTDE port
    size=struct.calcsize("!HB")
    mex=struct.pack("!HB",size,83)
    RTDE.send(mex)
    #verification of UR responce
    respB=RTDE.recv(4096)
    resp1=struct.unpack("!HBB",respB)
    if (resp1[0]!=4 or resp1[1]!=83):
        print("ERROR: Error in the responce. Verify that the socket
is RTDE")
        print(resp1[0],resp1[1])
        return (-1)
    if resp1[2]:
        print("Streaming on RTDE port starts")
        return(1)
    else:
```

```
        print("ERROR: Server reject command")
        return(-1)


#Pause streaming
def RTDE_CONTROL_PACKAGE_PAUSE(RTDE):
    #Pauses the streaming of data from RTDE port
    size=struct.calcsize("!HB")
    mex=struct.pack("!HB",size,80)
    RTDE.send(mex)
    #verification of UR responce
    respB=RTDE.recv(4096)
    resp1=struct.unpack("!HBB",respB)
    if (resp1[0]!=4 or resp1[1]!=80):
        print("ERROR: Error in the responce. Verify that the socket
is RTDE")
        return (-1)
    if resp1[2]:
        print("Streaming on RTDE port is in pause")
        return(1)
    else:
        print("ERROR: Server reject command")
        return(-1)


#Unpacks the bytes received from RTDE in the correct way
def RTDE_DATA_PACKAGE(RTDE,recipe):
    recipe="!"+recipe
    respB=RTDE.recv(4096)
    (nbytes,command,id_recipe)=struct.unpack("!HBB",respB[0:4])
    resp1=struct.unpack(recipe,respB[4:])
    return(resp1)
```

```
#Description of the library
def info():
    print("""
Library name:
    UR v3.0


Library description:
    This library contains some useful commands to control a
UR3/UR5/UR10
    Universal Robot. All functions was written respect to 3.5 UR
Polyscope.


Library functions:
    connectURCI
    connectPCI
    connectSCI
    connectRTCI
    connectRTDE
    connectAS2C
    streamRTCI
    renameDATA
    movec
    movej
    movel
    movep
    speedj
    speedl
    stopj
    stopl
    MMF
    csvF
    RTDE_REQUEST_PROTOCOL_VERSION
```

RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS

RTDE_CONTROL_PACKAGE_START

RTDE_CONTROL_PACKAGE_PAUSE

RTDE_DATA_PACKAGE

info

Functions description:

URCI=connectURCI(IP_UR,port)

This function asks in input the IP of the robot and the port to connect at

the wanted client inteface. Provides as output the socket over reads info

from the robot.

The Robot works as server so it is not necessary writes a corresponding

URScript

PCI=connectPCI(IP_UR)

This function asks in input the IP of the robot to connect at Primary

Client Interface.

The Robot works as server and it is not necessary writes a corresponding

URScript

SCI=connectSCI(IP_UR)

This function asks in input the IP of the robot to connect at Secondary

Client Interface.

The Robot works as server and it is not necessary writes a corresponding

URScript

RTCI=connectRTCI(IP_UR)

This function asks in input the IP of the robot to connect at Real Time

Client Interface.

The Robot works as server and it is not necessary writes a corresponding

URScript

RTDE=connectRTDE(IP_UR)

This function asks in input the IP of the robot to connect at Real Time

Data Exchange Client Interface.

The Robot works as server and it is not necessary writes a corresponding

URScript

[SCRIPT,S_SERVER]=connectAS2C(IP_PC,port)

This function asks in input the IP and the port of the PC where robot will

connects and provides as output the socket over write/receve to/from the

robot and the server socket(not used).

In this case the PC is the server and the robot needs a URScript for

connecting to this socket.

[data,t]=streamRTCI(RTCI)

This function asks in input the RTCI socket created with UR.connectRTCI(IP)

and provides in output the data streamed by the robot at 125Hz and

the execution time of the function.

[DATA,t]=renameDATA(data)

This function asks in input the data read with streamRTCI and put them in

a dictionary format where every value is connected with it's own name value.

It provides also the execution time.

t=movec(SOCKET,pose_via,pose_to,a,v,r=0,SocketType="CI")

t=movej(SOCKET,q,a,v,t=0,r=0,SocketType="CI")

t=movel(SOCKET,pose,a,v,t=0,r=0,SocketType="CI")

t=movep(SOCKET,pose,a,v,r=0,SocketType="CI")

t=speedj(SOCKET,speed,a,t=0,SocketType="CI")

This function ask in input the socket where to write, the speed and the

acceleration to send to the robot for execute the speedj URScript command.

Can be also specify the ending execution time (see URScript manual for

more informations).

As default t=0, so no specifications occures to the robot on t.

SocketType is set as default with "CONTROL".

If SocketType="CONTROL" the function expect to write at the CONTROL socket,

while if SocketType="SCRIPT" the function expect to write at the SCRIPT

socket. IT IS IMPORTANT UNDERSTAND THAT THE ROBOT USING DIFFERENT WAY TO

READ THE COMMAND HAS NECESSITY OF THE CORRECT SocketType.

It provides the execution time.

t=speedl(SOCKET,speed,a,t=0,aRot="a",SocketType="CI")

t=stopj(SOCKET,a,SocketType="CI")

This function ask in input the socket where write and the acceleration to

send to the robot for execute the stopj URScript command.

SocketType is set as default with "CONTROL".

If SocketType="CONTROL" the function expect to write at the CONTROL socket,

while if SocketType="SCRIPT" the function expect to write at the SCRIPT

socket. IT IS IMPORTANT UNDERSTAND THAT THE ROBOT USING DIFFERENT WAY TO

READ THE COMMAND HAS NECESSITY OF THE CORRECT SocketType.

It provides the execution time.

t=stopl(SOCKET,a,aRot="a",SocketType="CI")

MMF(name,data)

This function ask in input the name of the wanted output .txt file and the

datas provided by the "stream" function.

A file "name.txt" will be created with write inside a matrix in Matlab

format called "name".

IMPORTANT.1: data can be one output or a sequence outputs. For a unique

output a vector will be provide while for a sequence a matrix will be

provide.

IMPORTANT.2: The function is born thinking about the stream function for

move quickly the data on Matlab and have better graphics. But this function

can also works with any list or list of lists of the same dimensions.

IMPORTANT.3: Not yet controls on inputs! Maybe will be add in a second time.

csvF(name,data,sepv=",")

This function create a .csv file. Ask as input the name to call the file,

the values to save in and the separetor value that as default is ",".

version=RTDE_REQUEST_PROTOCOL_VERSION(RTDE,version=2)

This function asks to the RTDE Client Interface which version to use. Ask

in input the RTDE socket and which version to use, 1 or 2, as default 2.

recipe_id,recipe=RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS(RTDE,variables,version=2,frequency=125)

This function sets the outputs on RTDE client interface.

Ask as input the socket where to write, the variables (output) to set, the

version to use to write the, responce as defaut 2, and if the version

chosen is 2 the frequency at which the server will provide the data.

Returns, if the version is 2, an id corresponding to the recipe and a

recipe, that is the way in which the streaming must be unpacked.

check=RTDE_CONTROL_PACKAGE_START(RTDE)

This function asks to the RTDE client interface to start the streaming of

data.

check=RTDE_CONTROL_PACKAGE_PAUSE(RTDE)

This function asks to the RTDE client interface to pause the

```
streaming of
    data.


data=RTDE_DATA_PACKAGE(RTDE,recipe)
    This function unpacks the bytes received by RTDE client
interface in in the
    correct way thanks to the recipe provided by
    RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS.


info()
    This function print to screen this message.
    """)


#Description of the library
if __name__=="__main__":
    info()
```

Appendix-4 Python Robotiq Library

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 17 17:10:02 2018


@author: Matteo Gaidano
"""


import UR


# set the RTDE protocol of UR with the parameter needed for read the force
# values of ROBOTIQ FT sensor
def SET_RTDE_OUTPUT_FT_SENSOR(RTDE,frequency=125):


variables=["output_double_register_0","output_double_register_1","output_double_register_2","output_double_register_3","output_double_register_4","output_double_register_5"]
    version=UR.RTDE_REQUEST_PROTOCOL_VERSION(RTDE)


(id_recipe,recipe)=UR.RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS(RTDE,variables,version,frequency)
    return (id_recipe,recipe)


# rename the receved vector as request by ROBOTIQ
def rename_FORCE_FT_SENSOR(data):
    if len(data)!=6:
        print("ERROR: the force vector must be of 6 elements")
        return(-1)
    force={}
    force["FX"]=data[0]
    force["FY"]=data[1]
```

```
    force["FZ"]=data[2]
    force["MX"]=data[3]
    force["MY"]=data[4]
    force["MZ"]=data[5]
    return(force)

#Description of the library
def info():
    print("""
Library name:
    ROBOTIQ v1.0

Library description:
    This library contains some useful commands to read the value measured by FT
    sensor by Robotiq.

Library functions:
    SET_RTDE_OUTPUT_FT_SENSOR
    rename_FORCE_FT_SENSOR

Functions description:

[id_recipe,recipe]=SET_RTDE_OUTPUT_FT_SENSOR(RTDE,frequency=125)
    Using the RTDE client interface sets the outputs to read the values of
    force and torque measured by the FT Sensor.
    Asks as input the RTDE socket and the streaming frequency.
    Provides the id of the recipe and the recipe to unpack the data.

force=rename_FORCE_FT_SENSOR(data)
    This function rename the data provided by UR.streamRTDE when
```

```
setted with

    SET_RTDE_OUTPUT_FT_SENSOR.

    The output is a dictionary that bound every value with its own
name.


info()

    This function print to screen this message.

    """)


#Description of the library

if __name__=="__main__":

    info()
```

Appendix-5 Test

Test1: Take_position.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 21 15:51:25 2018

@author: Matteo Gaidano
"""

#Initialization
import UR
IP_UR="192.168.56.103"
print("START...")
#open, read, close
RTCI=UR.connectRTCI(IP_UR)
data=UR.streamRTCI(RTCI)
RTCI.close()
#print position
q_actual=data[0][31:37]
Tool_vector_actual=data[0][55:61]
print("q_actual: \n",q_actual,"\nTCP_actual: \n",Tool_vector_actual)
print("...END")
```

Test2: Test_movecjl_RTCI_1.m

```
clear all
close all
clc

%% Test funzionamento movej, movec e movel
% This is a test to show how to struct a a program that works
with the UR
% library and RTCI data transmission.
```

```
%START
disp("START...")
%Initialization
IP_UR="192.168.56.103";

q_start=[-0.785443131123678, -0.9384468237506312,
-1.9489944616900843, -1.7903927008258265, 1.570642113685608,
0.0];

pose_to_1=[0.1688709249454708, -0.3284885700478256, 0.115,
-1.2082754108078722, 2.89807531154162, -0.027717054051475451;
pose_via_1=[-0.019334054551769755, -0.35006490384635497,
0.115, -1.2082754108078722, 2.89807531154162,
-0.027717054051475451;

pose_to_2=[0.1240804133476665, -0.17081157185403162, 0.115,
-1.2044397488219285, 2.895416415018972, -0.051847793820446421;
pose_via_2=[0.23961678609703793, -0.09685563830352593, 0.115,
-1.2082754108078722, 2.89807531154162, -0.027717054051475451;

a=1.2;
v=0.25;
k=0;
mc1=10;
mc2=1000;
mc3=500;
mc4=500;
data=zeros(mc1+mc2+mc3+mc4,138);
RTCI=connectRTCI(IP_UR);

%First measurations
for i=1:mc1
    data(i,:)=streamRTCI(RTCI);
end
k=k+i;
%movej
movej(RTCI,q_start,a,v);
for i=1:mc2
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
disp("STARTING POSITION REACHED")
%movec
movec(RTCI,pose_via_1,pose_to_1,a,v);
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
movec(RTCI,pose_via_2,pose_to_2,a,v);
for i=1:mc3
```

```
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
%movel
movel(RTCI,pose_to_1,a,v);
for i=1:mc4
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
fclose(RTCI);
%END
disp("...END")

figure(1);
plot(data(1:1010,56),data(1:1010,57),'blue',data(1011:2010,56)
,data(1011:2010,57),'red',data(2011:2510,56),data(2011:2510,57
),'green');
xlabel("Y [m]")
ylabel("X [m]")
```

Test2: Test_movecjl_AS2C_1.m

```
clear all

close all
clc

%% Test funzionamento movej, movec e movel
%START
disp("START...")
%Initialization
IP_UR="192.168.56.103";

q_start=[-0.785443131123678, -0.9384468237506312,
-1.9489944616900843, -1.7903927008258265, 1.570642113685608,
0.0];

pose_to_1=[0.1688709249454708, -0.3284885700478256, 0.115,
-1.2082754108078722, 2.89807531154162, -0.02771705405147545];
pose_via_1=[-0.019334054551769755, -0.35006490384635497,
0.115, -1.2082754108078722, 2.89807531154162,
-0.02771705405147545];

pose_to_2=[0.1240804133476665, -0.17081157185403162, 0.115,
-1.2044397488219285, 2.895416415018972, -0.05184779382044642];
pose_via_2=[0.23961678609703793, -0.09685563830352593, 0.115,
-1.2082754108078722, 2.89807531154162, -0.02771705405147545];

a=1.2;
v=0.25;
k=0;
```

```
mc1=10;
mc2=1000;
mc3=500;
mc4=500;
data=zeros(mc1+mc2+mc3+mc4,138);

AS2C=connectAS2C(IP_UR,30000);
RTCI=connectRTCI(IP_UR);

%First measurations
for i=1:mc1
    data(i,:)=streamRTCI(RTCI);
end
k=k+i;
%Starting position
movej(AS2C,q_start,a,v,"SocketType","Server");
for i=1:mc2
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
disp("STARTING POSITION REACHED")
%Circle
movec(AS2C,pose_via_1,pose_to_1,a,v,"SocketType","Server");
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
movec(AS2C,pose_via_2,pose_to_2,a,v,"SocketType","Server");
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
%Line
movel(AS2C,pose_to_1,a,v,"SocketType","Server");
for i=1:mc4
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
halt(AS2C,"SocketType","Server");
fclose(RTCI);
fclose(AS2C);
%END
disp("...END")
```

Test2: Test_movecjl_AS2C_2.m

```
clear all

close all
clc

%% Test funzionamento movej, movec e movel
```

```
%START
disp("START...")
%Initialization
IP_UR="192.168.56.103";

q_start=[-0.785443131123678, -0.9384468237506312,
-1.9489944616900843, -1.7903927008258265, 1.570642113685608,
0.0];

pose_to_1=[0.1688709249454708, -0.3284885700478256, 0.115,
-1.2082754108078722, 2.89807531154162, -0.02771705405147545];
pose_via_1=[-0.019334054551769755, -0.35006490384635497,
0.115, -1.2082754108078722, 2.89807531154162,
-0.02771705405147545];

pose_to_2=[0.1240804133476665, -0.17081157185403162, 0.115,
-1.2044397488219285, 2.895416415018972, -0.05184779382044642];
pose_via_2=[0.23961678609703793, -0.09685563830352593, 0.115,
-1.2082754108078722, 2.89807531154162, -0.02771705405147545];

a=1.2;
v=0.25;
k=0;
mc1=10;
mc2=1000;
mc3=500;
mc4=500;
data=zeros(mc1+mc2+mc3+mc4,138);

AS2C=connectAS2C(IP_UR,30000);
RTCI=connectRTCI(IP_UR);

%First measurations
for i=1:mc1
    data(i,:)=streamRTCI(RTCI);
end
k=k+i;
%Starting position
movej(AS2C,q_start,a,v,"SocketType","Server");
for i=1:mc2
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
disp("STARTING POSITION REACHED")
%Circle
movec(AS2C,pose_via_1,pose_to_1,a,v,"SocketType","Server");
stopl(AS2C,2,"SocketType","Server")
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
```

```
movec(AS2C,pose_via_2,pose_to_2,a,v,"SocketType","Server");
stopl(AS2C,2,"SocketType","Server")
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
%Line
movel(AS2C,pose_to_1,a,v,"SocketType","Server");
for i=1:mc4
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
halt(AS2C,"SocketType","Server");
fclose(RTCI);
fclose(AS2C);
%END
disp("...END")
```

Test2: Test_movecjl_RTCI_2.m

```
clear all

close all
clc

%% Test funzionamento movej, movec e movel
% This is a test to show how to struct a a program that works
with the UR
% library and RTCI data transmission.

%START
disp("START...")
%Initialization
IP_UR="192.168.56.103";

q_start=[-0.785443131123678, -0.9384468237506312,
-1.9489944616900843, -1.7903927008258265, 1.570642113685608,
0.0];

pose_to_1=[0.1688709249454708, -0.3284885700478256, 0.115,
-1.2082754108078722, 2.89807531154162, -0.027717054051475451;
pose_via_1=[-0.019334054551769755, -0.35006490384635497,
0.115, -1.2082754108078722, 2.89807531154162,
-0.027717054051475451;

pose_to_2=[0.1240804133476665, -0.17081157185403162, 0.115,
-1.2044397488219285, 2.895416415018972, -0.05184779382044642];
pose_via_2=[0.23961678609703793, -0.09685563830352593, 0.115,
-1.2082754108078722, 2.89807531154162, -0.027717054051475451];

a=1.2;
v=0.25;
```

```
k=0;
mc1=10;
mc2=1000;
mc3=500;
mc4=500;
data=zeros(mc1+mc2+mc3+mc4,138);
RTCI=connectRTCI(IP_UR);

%First measurations
for i=1:mc1
    data(i,:)=streamRTCI(RTCI);
end
k=k+i;
%movej
movej(RTCI,q_start,a,v);
for i=1:mc2
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
disp("STARTING POSITION REACHED")
%movec

movec(RTCI,pose_via_1,pose_to_1,a,v);
stopl(RTCI,2)
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
movec(RTCI,pose_via_2,pose_to_2,a,v);
stopl(RTCI,2)
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
%movel
movel(RTCI,pose_to_1,a,v);
for i=1:mc4
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
fclose(RTCI);
%END
disp("...END")

figure(1);
plot(data(1:1010,56),data(1:1010,57),'blue',data(1011:2010,56)
,data(1011:2010,57),'red',data(2011:2510,56),data(2011:2510,57
),'green');
xlabel("Y [m]")
ylabel("X [m]")
```

Test2: Test_movecjl_RTCI_3.m

```
clear all

close all
clc

%% Test funzionamento movej, movec e movel
% This is a test to show how to struct a a program that works
with the UR
% library and RTCI data transmission.

%START
disp("START...")
%Initialization
IP_UR="192.168.56.103";

q_start=[-0.785443131123678, -0.9384468237506312,
-1.9489944616900843, -1.7903927008258265, 1.570642113685608,
0.0];

pose_to_1=[0.1688709249454708, -0.3284885700478256, 0.115,
-1.2082754108078722, 2.89807531154162, -0.02771705405147545];
pose_via_1=[-0.019334054551769755, -0.35006490384635497,
0.115, -1.2082754108078722, 2.89807531154162,
-0.02771705405147545];

pose_to_2=[0.1240804133476665, -0.17081157185403162, 0.115,
-1.2044397488219285, 2.895416415018972, -0.05184779382044642];
pose_via_2=[0.23961678609703793, -0.09685563830352593, 0.115,
-1.2082754108078722, 2.89807531154162, -0.02771705405147545];

a=1.2;
v=0.25;
k=0;
mc1=10;
mc2=1000;
mc3=500;
mc4=500;
data=zeros(mc1+mc2+mc3+mc4,138);
RTCI=connectRTCI(IP_UR);

%First measurations
for i=1:mc1
    data(i,:)=streamRTCI(RTCI);
end
k=k+i;
%movej
movej(RTCI,q_start,a,v);
for i=1:mc2
    data(i+k,:)=streamRTCI(RTCI);
end
```

```
k=k+i;
disp("STARTING POSITION REACHED")
%movec
command='def MyProg():';
fprintf(RTCI,'%s\n',command);
movec(RTCI,pose_via_1,pose_to_1,a,v);
stopl(RTCI,2)
command='end';
fprintf(RTCI,'%s\n',command);
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
command='def MyProg():';
fprintf(RTCI,'%s\n',command);
movec(RTCI,pose_via_2,pose_to_2,a,v);
stopl(RTCI,2)
command='end';
fprintf(RTCI,'%s\n',command);
for i=1:mc3
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
%movel
movel(RTCI,pose_to_1,a,v);
for i=1:mc4
    data(i+k,:)=streamRTCI(RTCI);
end
k=k+i;
fclose(RTCI);
%END
disp("...END")

figure(1);
plot(data(1:1010,56),data(1:1010,57),'blue',data(1011:2010,56)
,data(1011:2010,57),'red',data(2011:2510,56),data(2011:2510,57
),'green');
xlabel("Y [m]")
ylabel("X [m]")
```

Test3: SliderESpeedl_AS2C.m

```
%% Test funzionamento grafica con slidere e speedl

clear all
close all
clc

%% Initialization
IP_UR="192.168.56.103";
xl=0;xr=20;yl=-20;yr=10;%cm
```

```
v_max=10;%cm/s
a=2;%m/s^2
variables=["input_int_register_10","input_double_register_12",
"input_double_register_13","input_double_register_14","input_d
ouble_register_15","input_double_register_16","input_double_re
gister_17","input_double_register_18"];

%Figure with User Interface
offset_y_line=75;
offset_x_line=50;
f=figure('visible','off','position',[900, 200, 400, 400]);
push_button1=uicontrol(f,'style','togglebutton','position',
[offset_y_line, 360, 20, 20]);
Start_Stop=uicontrol('style','text','position',
[offset_y_line+40, offset_x_line+310, 100,
20],'String','Start/Stop','FontSize',12);
push_button2=uicontrol(f,'style','togglebutton','position',
[offset_y_line, 320, 20, 20]);
Play_Pause=uicontrol('style','text','position',
[offset_y_line+40, offset_x_line+270, 100,
20],'String','Play/Pause','FontSize',12);
sliderX=uicontrol(f,'style','slider','position',
[offset_y_line, 280, 200, 20],'min',xl,'max',xr);
sliderY=uicontrol(f,'style','slider','position',
[offset_y_line+220, offset_x_line, 20,
200],'min',yl,'max',yr);
%target_position=uicontrol(f,'style','text','position',[170,
340, 40, 15],'visible','off')
axes('units','pixels','position',[offset_y_line,
offset_x_line, 200, 200]);
set(f,'visible','on');
pause(1);

%%Start
start=1;
play=0;
first_cicle=1;
k=0;
while start
    start=0;
    if first_cicle
        while start==0
            start=get(push_button1,'value');
            pause(0.1);
        end
        disp("START");
        first_cicle=0;
        AS2C=connectAS2C(IP_UR,30000);
        movel(AS2C,[0.2,0.0,0.06,2.18,-
2.18,0.0],1.2,a,"SocketType","Server");
        RTCI=connectRTCI(IP_UR);
```

```
    end
    play=get(push_button2,'value');
    while play==1
        k=k+1;
        X=get(sliderX,'value');
        Y=get(sliderY,'value');
        data=streamRTCI(RTCI,'RT');
        DATA(k,1:138)=data;
        Xr=data(56)*100-20;
        Yr=data(57)*100;
        plot(X,Y,'*r',Xr,Yr,'*b')
        V_X=feedbackvel(X,Xr,xl,xr,v_max)/100;
        V_Y=feedbackvel(Y,Yr,yl,yr,v_max)/100;
        speed=[V_X,V_Y,0,0,0,0] %m/s
        speedl(AS2C,speed,a,"SocketType","Server")
        DATA(k,139:144)=speed;
        DATA(k,145:146)=[X,Y];
        ax=gca;
        ax.XLim=[xl,xr];
        ax.YLim=[yl,yr];
        play=get(push_button2,'value');
        pause(0.01)
    end
    stopl(AS2C,a,"SocketType","Server")
    k=k+1;
    data=streamRTCI(RTCI,'RT');
    DATA(k,1:138)=data;
    DATA(k,139:144)=[0,0,0,0,0,0];
    DATA(k,145:146)=[0,0];
    start=get(push_button1,'value');
    pause(0.01)
end
fclose(RTCI);
fclose(AS2C);
disp("END");

function velocity=feedbackvel(X_target,X_actual,xl,xr,v_max)
deltaX=X_target-X_actual
velocity=0;
if abs(deltaX)<=0.3
    return
elseif X_target>X_actual
    sign=1;
    if X_actual>=xr
        return
    end
else
    sign=-1;
    if X_actual<=xl
        return
    end
```

```
end
if abs(deltaX)>10
    m=1;
else
    m=abs(deltaX)/10;
end
velocity=sign*v_max*m;
if abs(velocity)<0.5
    velocity=sign*0.5;%cm/s
end
end
```

Test4: Test_read_sensor.m

```
clear all

close all
clc

%% Test acquisition from sensor and rename of data

IP_UR="192.168.56.103";
disp("START")
RTDE=connectRTDE(IP_UR);
[id_recipe,recipe]=SET_RTDE_OUTPUT_FT_SENSOR(RTDE);
RTDE_CONTROL_PACKAGE_START(RTDE);
disp("START ACQUISITION")
for i=1:1
    data=RTDE_DATA_PACKAGE(RTDE,recipe);
    DATA=rename_FORCE_FT_SENSOR(data)
end
RTDE_CONTROL_PACKAGE_PAUSE(RTDE);
fclose(RTDE);
disp("STOP")
```