Master Final Thesis

Màster Universitari en Enginyeria Industrial (MUEI)

Strategies for remote control and teleoperation of a UR robot

MEMORY

June 14, 2020

Autor: Sergi Ponsà Cobas

Directors: Pedro Ponsa Asensio

Convocatòria: 02/2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Summary

In this project it's explored the different teleoperation options which Universal Robots (UR) offer. It's explained which options exist and compared the pros and cons of these methodologies.

Once they are compared we will select some of them to teleoperate multiple UR robots, taking into account these pros and cons of the methodologies.

This project wants to promote the use of the UR robots by creating a toolkit easy to understand which allow future students develop their projects with this toolkit as a base.
In order to test this toolkit and show the potential of these teleoperation methodologies, a user friendly PyQT (Python) application has been developed using this toolkit as a base.

Contents

List of Figures

List of Tables

1 Preface

1.1 Origin of the project

The university has recently acquired some UR3 robots from Universal Robots to use them at FIB, EEBE and ETSEIB robotics laboratories. These robots provide more options than the previous once they had, and this project wants to explore some of the possibles uses of these robots.

The student which it's realising this project has been doing an internship in Universal Robots, which allows to the student have some bases to develop this project.

1.2 Previous requirements

In order to use the elements developed in this project, it's highly recommended to have done previously the formations listed below to have a previous notion of how the UR robots works.

Recommended formations:

- Training program in UR site or the online Academy courses provided by Universal Robots.
- Realization of robotic courses in official University studies.

2 Introduction

2.1 Project objective

The objective of this project it's to determine which teleoperation protocols Universal Robots offer and which are more interesting to preform the different required actions to have a complete Teleoperation of the robot. Once these options are analyzed, a toolkit will be developed selecting the more interesting once to make them easy to use for future students.

2.2 Project requirements

In order to have a complete Teleoperation of a robot, it's required to fulfill some requirements:

- Be able to preform movements. [Tel.Req.1]
- Preform basic actions such as turn on the robot or change the digital outputs of the robots. [Tel.Req.2]
- Get data of the current state of the robot. [Tel.Req.3]
- Change the loaded programs and run programs. [Tel.Req.4]

These requirements are grouped inside:

- Remote Control. [Tel.Req.1] [Tel.Req.2]
- Remote Maintenance. [Tel.Req.3] [Tel.Req.4]

Additionally, it will be interesting to be able to access to the robot controller terminal.
This access to a device terminal is known as Remote Access.

These concepts mentioned are defined as:

Remote Control

- Refers to the ability to control some mechanism from a distance.

Remote Maintenance

- Refers to the ability to access a machine, for example to perform software maintenance, update documents and get data of the robot which allows the user to make a preventive maintenance. In order to fulfil the ISO 10218-1 & ISO 10218-2, UR has to provide this option.

Remote Access

- Refers to the ability to access a computer from a remote location.

2.3 Project scope

This project will develop a toolkit to be used in a computer and other device implementations as the teleoperation through PLCs will not be considered.

Neither will it be considered payment solutions which have been developed by private companies to teleoperate the robots, such as the Insights solution of Robotiq or other companies products.

The toolkit will be made to be comprehensive, using code where it can be seen everything that it's done. The use of external libraries has been avoided, because even they could have a greater performance, some parts of the process will be hidden .

2.4 State of the art

Currently in the academic field, the teleoperation of the robots have being done with ROS, using packages shared by the community. In the specific case of the Universal Robots, it already exists some ROS packages[1]. These packages are based in the protocols offered by Universal Robots, but they are hide to the student programming the robot in ROS. As a consequence of that, the student it's able to do teleopartion actions to the robot through ROS, but the student doesn't know how preform these actions without ROS.

Even if it's true that some companies are starting to use ROS, the vast majority of companies don't need the amount of control that ROS provide and prefer to use much more simpler implementations. For this reason, it's interesting that the student knows the basic protocols to control the robot too.

2.5 Project Structure

First, in the section 3 it will be introduced the different options, then each one of them will be briefly explained and their cons and pros will be analyzed. As a result of this, they will be classified according to their suitability to different tasks in the section 3.2.

Second, in the section 4 the protocols which will be used to teleoperate the robot will be selected and it will be evaluated if it's interesting to develop a toolkit to use them.

Third, in the section 5 an application using this toolkit will be developed and analyzed its utility.

Fourth, in the section 6 it will be estimated which will be the costs of this project and in the section 7 it will be determined the project environmental impact.

Finally, in the section 8 it will be considered the whole project to evaluate its utility comparing it with the current commercial solutions and elements which could be improved.

3 Teleoperation protocols offered by Universal Robots

Universal robots offer the following communication options:

- TCP/IP Sockets, Transmission Control Protocol/Internet Protocol Sockets[2]
- TCP/IP Dashboard, Transmission Control Protocol/Internet Protocol Dashboard [3][4]
- TCP/IP Primary client & Secondary client, Transmission Control Protocol/Internet Protocol Primary client & Secondary client [5]
- TCP/IP Real Time,Transmission Control Protocol/Internet Protocol Real Time [5]
- TCP/IP Real Time Data Exchanger, Transmission Control Protocol/Internet Protocol Real Time Data Exchanger [6]
- XML-RPC, Extensible Markup Language Remote Procedure Call[7]
- Modbus TCP, Modbus Transmission Control Protocol[8]
- EthernetIP[9]
- Profinet, Process Field Net[10]
- FTP, File Transfer Protocol [11][12]
- SSH, Secure Shell [13]

3.1 Description of the teleoperating protocols and evaluation

3.1.1 TCP/IP

All the teleoperating protocols shown are based on the TCP/IP stack protocol. In order to have a better understanding of them, this protcol it's briefly explained.

The TCP/IP stack protocol it's form by 5 layers:

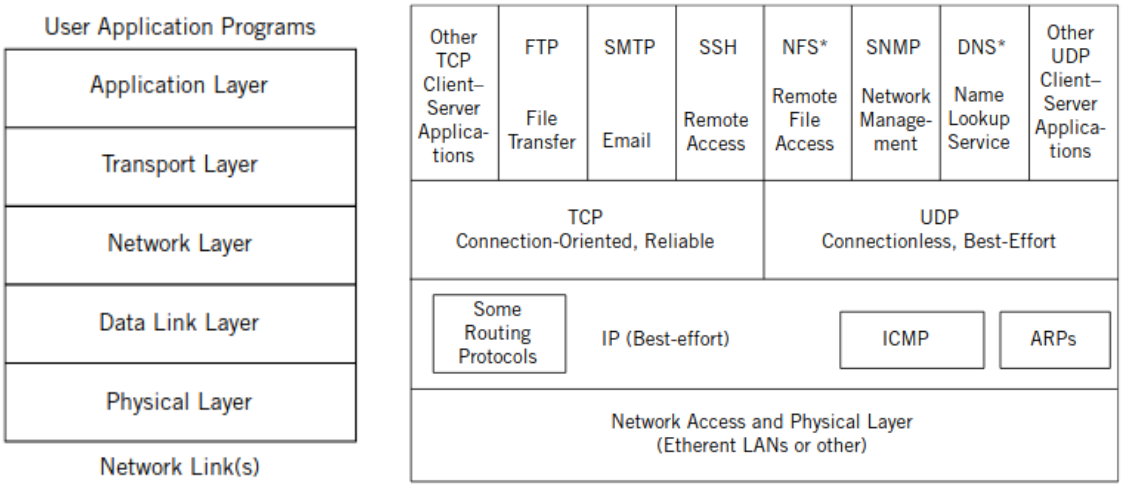


Figure 1: TCP/IP Layers figures from The Ilustrated Network [14]

- The Network Link is determined by how we are sending the data, by cable using the Ethernet protocol or wireless using the Wi-Fi protocol.
- The Physical Layer is the physical signal used to transmit bit-stream of data over a transmission medium.
- The Data Link Layer codifies the package of data wanted to send, identifying the start and end of the data too.
- The Network Layer uses the protocol Internet Protocol (IP), which identifies the devices and the net, using 4 bytes. The submask determine which bytes identify the device and which the net.
- The Transport Layer uses the Transmission Control Protocol, which guarantee that the data stream is delivered to the destination application error-free, with all data in sequence and complete, by using a sequence numbering the data send.
- The Application Layer determines the protocol for the use of the specific services of the chosen application, usually determined by the port (80 Http, 22 & 23 FTP, etc).

To get more information about the TCP/IP protocol, consult chapters 1 & 2 from The Ilustrated Network Book [14]

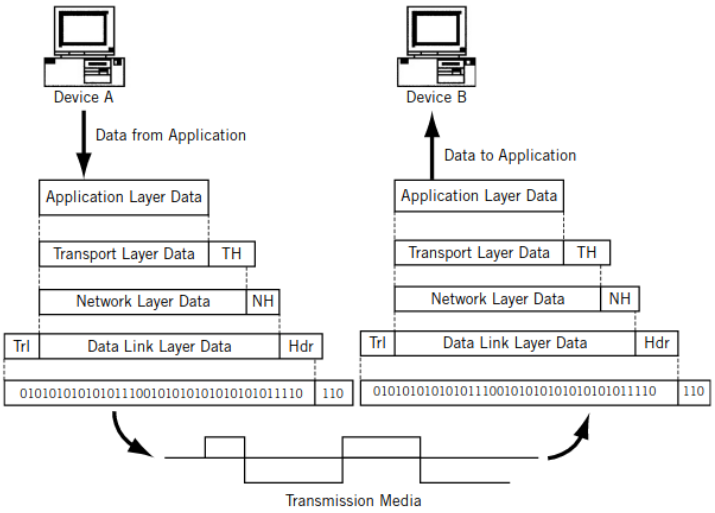


Figure 2: TCP/IP example of comunication, figure from The Ilustrated Network [14]

3.1.2 TCP/IP Socket

TCP/IP Socket description

A socket save a connection between 2 devices to transmit data using the stack protocol TCP/IP. It realize a first handshacking between the server and the client which consist of three transmissions:

1. The client ask for connection by sending a TCP segment with the synchronization control bit set identifying the client.

2. The server responds with its own synchronization segment that includes the information sent by the client in the initial synchronization segment.

3. The client acknowledges the server's synchronization.

This allows the client to ask for connection to the server, and the server can decide which port assign to attend the client. This imply that the server can create multiple connections to attend multiple clients, ensuring each receive what it's asking, or block the rest of clients to attend only that client.

Socket stream services implemented are the main TCP services: Open, Send, Receive,Status,Close & Abort. These services are few, but they are essentials ones to allow the data transmission, which have contributed to make the socket widely used by developers.

Exist other types of Socket, but we will not use them due to they work with other transportation protocols and are not supported by UR. Check the chapter 12 of The Illustrated Network to get more information [14].

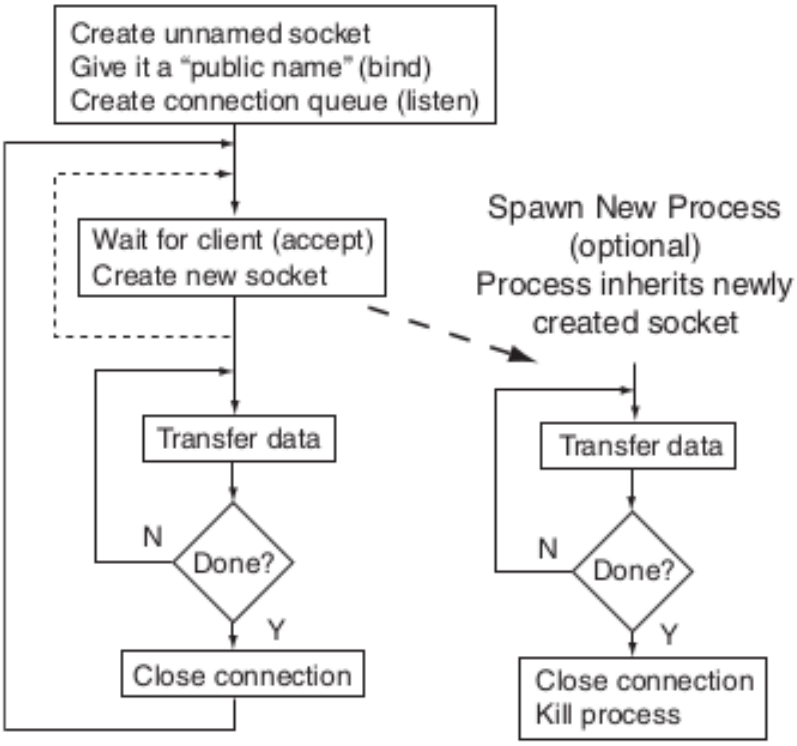


Figure 3: Soket server process, figure from Linux for Embedded and Real-Time Applications[15]

TCP/IP Socket evaluation

Pros:

- There are no limitations, you can have as many different commands as you want.
- The data type is selected by you.
- URScript has specific commands to use them.

Cons:

- Can be used to interact with running programs, but the robot has to be programmed previously to expect this interaction.
- There is not an automatic cyclic reception of data done, you have to program it in the robot.
- There are not any variable already created, you have to create all the variables you want to store the socket data previously in the program.

3.1.3 TCP/IP Dashboard

TCP/IP Dashboard description

The TCP/IP Dashboard is based in the usual TCP/IP communication and from there UR has created a set of services which are always available in the robot port 29999, waiting to be called. These service provide some functionalities which are not supported from any other telematic way.

The available commands can be seen in the appendix: A.1.

TCP/IP Dashboard evaluation

Pros:

- It allows some robots commands which can not be preform any other way, and are essential to teleoparete the robot, such as turn on the robot or release brakes.
- Their functions can be preform, without need to stop the current running program.

Cons:

- Only used by UR robots, it can not be used in other robots.
- The available functions are limited to the UR provided ones, new ones can not be created.

3.1.4 TCP/IP Primary client, Secondary client

TCP/IP Primary Client, Secondary Client description

The TCP/IP Primary & Secondary is based in the usual TCP/IP communication and from there UR has created a set of services which are always available in the robot ports 30001 & 30011 for the Primary Client and 30002 & 30012 for the Secondary Client. The difference between the primary and secondary is the data that the robot provides us:

- Primary Client transmits robot state data and additional messages.
- Secondary Client transmits robot state data only.

Additionally to provide data, if we connect to the ports 3001X, the server will write everything we send as the main script program of the robot and will run it. This means that the whole program has to be send as 1 command and has to follow the universal robots script structure (figure 4).

The UR script commands can be seen in the UR script manual from the download page of Universal Robots [16].

```
def myprogramname():
    #This is a commentary

    #To avoid to move through the teach pendant to the initial position
    #This is done by safety, but it doesn't allow to run programs by teleoparition
    movej(get_actual_joint_positions())

    #You can avoid the following while, but then the program
    # runs only once
    while(True):
        #Here you should write the script commands
        sleep(0.5)
    end
end
```

Figure 4: Example structure to send scripts

TCP/IP Primary client, Secondary client evaluation

Pros:

- It allows to program the robot, without need to do nothing previously on the robot.
- Constant flow of data coming from the robot.
- You can do everything , you can program on the robot.

Cons:

- Can not be used to interact with a running program.
- The coming data it's encoded by the specific UR protocol.
- There is not a programmatic way provided to decode the data, only an Excel with the coding is provided.
- Low frequency data refresh, 10Hz.

3.1.5 TCP/IP Real Time Data

TCP/IP Real Time Data description

The TCP/IP Real Time Data is based in the usual TCP/IP communication and from there UR has created a set of services which are always available in the robot ports 30003 & 30013.

As in the case of the Primary and Secondary clients, the port 3000x is only for data and the port 3001x is for data and send scripts to robot to be run. The main difference between the Real Time Data and the Primary and Secondary clients is that the data is provided much faster (10Hz Primary&Secondary, 120Hz CB series | 500Hz e-series Real Time Data), but with the cost to provide less data.

The difference of data provided can be seen in the universal robots web site [17] and in the appendix A.2.

TCP/IP Real Time Data evaluation

Pros:

- Constant flow of data coming from the robot.
- It allows to program the robot, without need to realize any previous task in the robot.
- High frequency data refresh, 125Hz CB series | 500Hz e-series.

Cons:

- Can not be used to interact with a running program.
- The coming data it's encoded by the specific UR protocol.
- There is not a programmatic way provided to decode the data, only an Excel with the coding is provided.
- The data flow provided is less than with other methods.

3.1.6 TCP/IP Real Time Data Exchanger

TCP/IP Real Time Data Exchanger description

The TCP/IP Real Time Data is based in the usual TCP/IP communication and from there UR has created a set of services which are always available in the robot port 30004.

This protocol allows us to send commands to determine the data that it's wanted to be received codified with a code defined by Universal Robots in an Excel. Some data provided is common with other tools, can be seen in the universal robots web site [17] and in the annex A.2.

TCP/IP Real Time Data Exchanger evaluation

Pros:

- Constant flow of data coming from the robot.
- High frequency data refresh, 125Hz CB series | 500Hz e-series.

Cons:

- The coming data is encoded by specific UR protocol.
- Can not be used to interact with a running program.
- There is not a programmatic way provided to decode the data, only an Excel with the coding is provided.

3.1.7 XML-RPC

XML-RPC description

The XML-RPC is used to call a function in another PC and receive as a response the result of that function. It's based on the TCP/IP stack protocol, and the application layer is form by the combination http protocol with the XML protocol. The http protocol is used to ask for the communication and the XML protocol is used to structure the data provided as parameters of the function we want to call or the result of that call.

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169
<?xml version="1.0"?>
<methodCall>
   <methodName>circleArea</methodName>
      <params>
         <param>
            <value><double>2.41</double></value>
         </param>
      </params>
</methodCall>
```

```
HTTP/1.1 200 OK
Date: Sat, 06 Oct 2001 23:20:04 GMT
Server: Apache.1.3.12 (Unix)
Connection: close
Content-Type: text/xml
Content-Length: 124

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><double>18.24668429131</double></value>
      </param>
   </params>
</methodResponse>
```

Figure 5: XML-RPC Request and Response

Services are defined in the server code, registering a function as a service and specifying their parameters. These parameters can be single values types of the table shown bellow, arrays of these types or structures.

Type	Value	Examples
int or i4	32-bit integers between - 2,147,483,648 and 2,147,483,647.	<int>27</int> <i4>27</i4>
double	64-bit floating-point numbers	<double>27.31415</double> <double>-1.1465</double>
Boolean	true (1) or false (0)	<boolean>1</boolean> <boolean>0</boolean>
string	ASCII text, though many implementations support Unicode	<string>Hello</string> <string>bonkers! @</string>
dateTime.iso8601	Dates in ISO8601 format: CCYYMMDDTHH:MM:SS	<dateTime.iso8601> 20021125T02:20:04 </dateTime.iso8601> <dateTime.iso8601> 20020104T17:27:30 </dateTime.iso8601>
base64	Binary information encoded as Base 64, as defined in RFC 2045	<base64>SGVsbG8sIFdvcmxkIQ== </base64>

Figure 6: XML-RPC Types

And example of the code to create a XML-RPC server for Python or C++ can be find in the Universal Robots website and also how to call these functions in a robot program [7], a capture of the robot example code and Python code can be seen in the appendix A.3 too.

XML-RPC evaluation

Pros:

- Expand the capabilities of the robot.
- Can provide data and run the task at the same time.
- The server can be launch in the robot using SHH.
- Process can be externally done in different computer and then used in the robot with only 1 command in the robot.

Cons:

- It needs to be previously programmed in the robot program to use it.

3.1.8 ModbusTCP

ModbusTCP description

Modbus is a protocol to transmit data saved in registers of 16 bits (inputs|outputs) or coils of 1 bit (inputs|outputs). Modbus has built a set of services that allows the master to easily access the slave to read or modify its data.

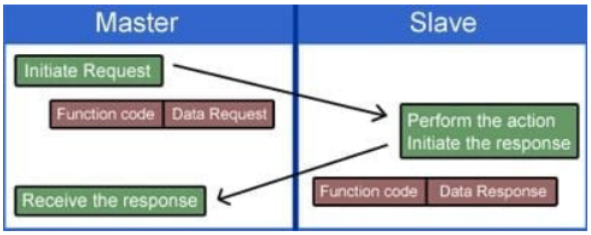


Figure 7: Modbus Master-Slave Communication

There are many implementations of Modbus but all of them have the same application protocol with few modifications.

The request and response are prefixed by six bytes as follows

byte 0:	transaction identifier – copied by server – usually 0
byte 1:	transaction identifier – copied by server – usually 0
byte 2:	protocol identifier = 0
byte 3:	protocol identifier = 0
byte 4:	length field (upper byte) = 0 (since all messages are smaller than 256)
byte 5:	length field (lower byte) = number of bytes following
byte 6:	unit identifier (previously 'slave address')
byte 7:	MODBUS function code
byte 8 on:	data as needed

Figure 8: Modbus TCP structure

- **Transaction Identifier (2 Bytes):** It's used to identify the transaction when several Modbus transactions are sent along the same connection.
- **Protocol Identifier (2 Bytes):** It's set to 0 for Modbus services, other numbers are reserved to extensions services.
- **Length (2 Bytes):** It's a byte count of the remaining fields and includes the destination identification (Register or Coil number) and data fields.

- **Unit Identifier (1 Byte):** It's used to identify the master, usually in TCP, but this function is already done by the IP, so it's usually ignored. It's only used when different elements of the same device ask different data to the slave.

- **Modbus Function Code (1 Byte):** It's used to tell the slave what service it's wanted.

CODE	FUNCTION
01 (01H)	Read Coil (Output) Status
03 (03H)	Read Holding Registers
04 (04H)	Read Input Registers
05 (05H)	Force Single Coil (Output)
06 (06H)	Preset Single Register
15 (0FH)	Force Multiple Coils (Outputs)
16 (10H)	Preset Multiple Registers
17 (11H)	Report Slave ID

Figure 9: Modbus TCP services

Some examples of the services requests and responses can be seen in the appendix A.4 and the document of Introduction to Modbus TCP [18].

ModbusTCP evaluation

Pros:

- Robot data frequently update.
- An standard protocol common in multiple devices, mainly PLCs.
- Server and Client in the same robot, with few actions.
- Parameters can be used to interact with running programs, but the robot has to be programmed previously to expect this interaction.
- Not linked to a specific brand, open-source.
- +100 registers already created to use them with the robot.

Cons:

- Less data is reachable than with other communication options.
- Does not exist modules to directly apply it to the device to get the robot data. It has to be done from the Excel provided.

3.1.9 Ethernet/IP

Ethernet/IP description

In the Ethernet/Ip protocol the devices are defined as collection of objects following the Common Industrial Protocol (CIP). This protocol provides a well defined data representation, connection and messaging protocol.

CIP data representation have required objects and application objects, all the devices which implement CIP have the required objects, while the application objects are specific of our device.

Objects are a collection of attributes where the data is saved, if there are more than one attribute of the same type they are differentiated by their instance value.

To access to this data exists 2 types of messages, explicit messages and implicit messages.

- **Explicit messages**: Communicates using the TCP/IP stack protocol, it's used for non time critical data.
- **Implicit messages**: Communicates using the UDP/IP stack protocol, it's similar to TCP but does not check that the data it's received. It's used for time critical data transfers.

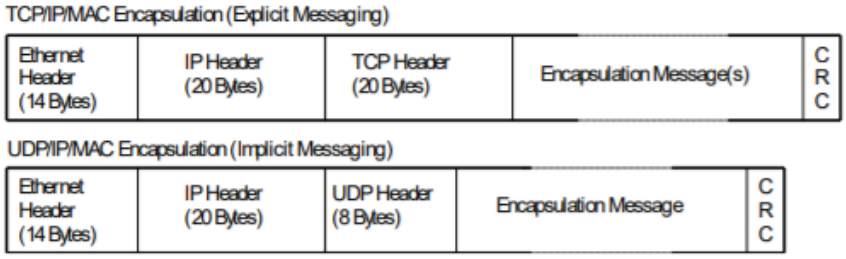


Figure 10: Ethernet/IP messages

In both cases the data is encapsulated the same way.

- **Command**: Identify what is wanted to be done.
- **Length**: Identify how much data is expected after the header.
- **Session Handle**: Identify the actual session between the client and the server. There are some commands which does not need a session handle.
- **Status**: Identify if the requested command has been accomplish or some error has happened.
- **Sender Context**: It's used to match a request with the response, both will have this value equal.
- **Option Flags**: It's used to allow a modification in the meaning of the encapsulation, if we do not want to do any modification, set it to 0.
- **Data**: Data provided to realize the command.

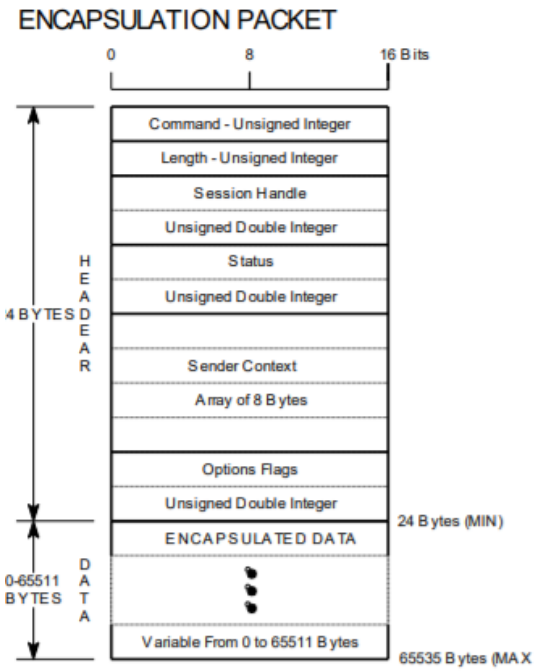


Figure 11: Ethernet/IP messages encapsulation

The available commands and status can be seen in the appendix A.5 or the website [19]. In order to use this protocol with the UR robots, universal robots provides a module to program the PLC and the functions to use them in the robot program. [9]

Ethernet/IP evaluation

Pros:

- Robot data frequently update.
- Based on standard protocols TCP/IP and UDP/IP.
- Server and Client in the same robot, with few actions.
- More data is reachable than with Modbus.
- Universal Robots have created modules to directly apply it to the devices, to get the robot data.
- A clear structure of the data.
- Can be used to interact with robot programs if they are previously programmed to do it.
- There is a company behind giving support to this communication.

Cons:

- Mainly linked to Rockwell products.
- The protocol is more complex than Modbus.

3.1.10 Profinet

Profinet description

Profinet is based in different protocols of communication [20]:

- TCP/IP or UDP/IP communications for certain non-time critical tasks, such as configuration, parameterization, and diagnostics.
- PROFINET RT handles time-critical data exchange. The frame skips the TCP/IP layers and avoids the variable time it takes to be processed.
- PROFINET IRT is a step beyond PROFINET RT. Unavoidably, under high network traffic, some time-critical messages can gain jitter. Profinet IRT, creates a special set of rules to avoid it.

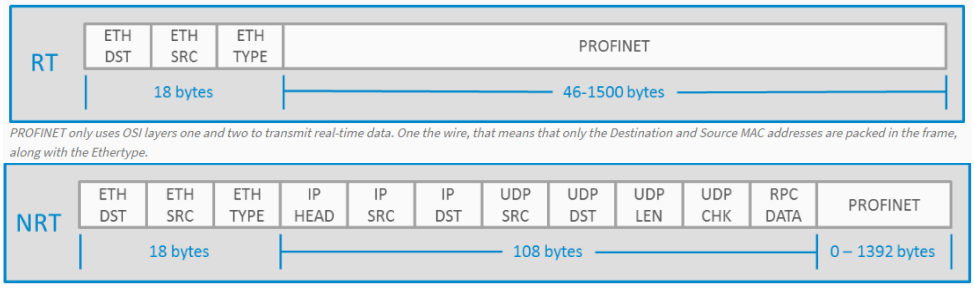


Figure 12: Real time and not Realtime codification [21]

In Profinet the data is saved in channels which can be discrete (0 or 1) or analog. Channels are grouped in Subslots, and these are grouped on Slots.[22]

Modules are defined components to plug into Slots to get data. Universal Robots provide 10 modules, 7 modules which contain data from the robot and 3 modules to write data on the Robot.

The content of these modules can be abstracted from the GSDML file provided by UR.

The GDSML file it's an XML decription of the Profinet IO device.

The GDSML file and how to use the profinet data in a program can be found in their website.[10]

The codification of the profinet data is quiet complex but it can be found in the book Automating with PROFINET: Industrial Communication Based on Industrial Ethernet.

Profinet evaluation

Pros:

- Robot data frequently update.
- Server and Client in the same robot, with few actions.
- More data is reachable than with Modbus.
- Universal Robots has created modules to directly apply it to the devices to get the robot data.
- Can be used to interact with robot programs if they are previously programmed to do it.
- There is a company behind giving support to this communication.

Cons:

- Mainly linked to Siemens products
- Use of not standard protocols to increase the velocity, which increase their complexity

3.1.11 FTP

FTP description

The FTP protocol allows to transfer files between 2 different devices.

The FTP protocol is based in the TCP/IP stack protocol, connecting to 2 ports the 20 and the 21 of the server. The connection to the port 20 is used to send commands only and the connection to the port 21 is used to transfer the files data. All the commands available are form by 4 ASCII characters and parameters if needed, they can be consulted in the appendix A.6 and the book [23].

This protocol has the problem that the data is not encrypted and opens 2 ports in firewall.
For this reason the Security Shell File Transfer Protocol was created to solve this security issues.

This protocol encrypts the information as it's done in the SSH protocol and differently to the FTP protocol, both the commands and the data are send through the port 22.

Nowadays almost all the users which use this protocol doesn't know the commands behind this protocol, due to they use open source software available to transfer files between devices, such as Filezilla, which use these protocol commands but are hidden to the user.

Universal Robots has a default user and password to enable the file transfer with the robot which can be seen in the website.[12]

FTP evaluation

Pros:

- The server is always running in the robot.
- No interaction with the robot is needed to use it.
- Exist free software easy to use available.
- A safety backup of the programs and calibration can be done.

Cons:

- If you don't know what you are doing, you can damage the robot software

3.1.12 SSH

SSH description

The SSH protocol is usually used to access to another device terminal. The SSH protocol covers the authentication, encryption and integrity of the data.

- **Authentication**: In order to start the connection, this protocol ask for a proof of your identity. If you pass the control, it enables you to connect.
- **Encryption**: All the data transferred during the communication is encrypted for that session, if you connect another session the encryption will be different. Only the server and the client know the encryption.
- **Integrity**: Ensures that the data transmitted has not been modified. If a third party modifies the data, it's available to detect it.

In order to do that it uses different types of encryption:

- Asymmetric encryption (the key for encrypt is different from the one to decrypt).
- Symmetric encryption (The key for encrypt and decrypt is the same).
- HASH, converts a package content to a fix longitude code using a mathematical function, it's used to check that the package has not been modified.

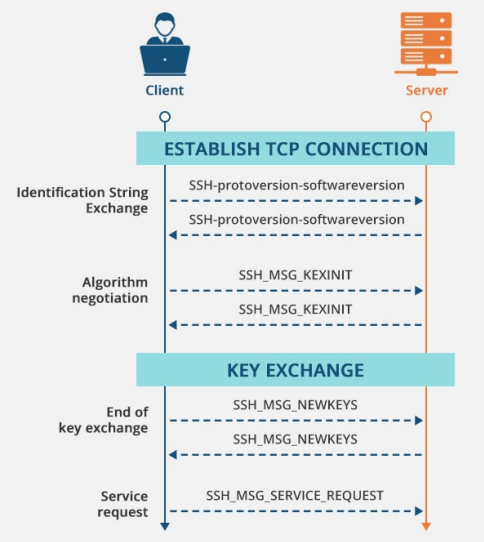


Figure 13: SSH handshake

1. Initially it preforms the typical TCP handshake, which it's already explained in the TCP/IP Sockets section.

2. Then they share which version of SSH they are using.

3. They share which options of symmetrical encryption have.

4. They share they public key, the encryption each one use in their asymmetric encryption.

5. The server check if the provided key it's already a key of known clients, otherwise ask for a user and a password to allow the connection, and the client receive the public key of the server to ensure it's the server wanted to communicate with.

6. Once the server and the client accept the connection, they select a encryption that both know and it's send encrypted in the each other public key and decrypted with their own private key that only each one know.

7. They start the transmission of data using this new encryption, this can be done using the port 22 or another port, which is known as SSH tunneling.

For more information consult the book [24]

SSH evaluation

Pros:

- You can preform terminal commands, that other way are not possible, such as run a Python code.

Cons:

- If you don't know what you are doing, you can damage the robot software.

3.2 Selected teleoperating protocols depending on the desired use

After seen the different teleoperating protocols available in the UR robots and their pros and cons, the following table of protocols for desired use has been done.
It's has been done a classification considering the best option in the majority of cases, but there are some cases where another protocol is better. Read the justification to ensure that it's the best option for your case.

Data acquisition			
1. Modbus TCP	2. Profinet/Ethernet IP	3. TCP/IP Real Time Exchanger	
Remote control			
1. TCP/IP Real Time	2. TCP/IP Socket	3. Modbus TCP	4. Profinet/Ethernet IP
Dynamic programs			
1. XML-RPC	2. TCP/IP Socket	3. Modbus TCP	4. Profinet/EthernetIP
Remote Maintenance			
1. TCP/IP Dashboard & FTP (SFTP)		2. TCP/IP Dashboard & SSH (Terminal)	
Remote Access			
SSH			

Table 1: Teleoperation protocol depending on the use

Justification: Data acquisition

It's highly recommended to consult the Data Matrix Table provided by UR, to check that the chosen protocol provide the data you are looking for. Consult the website [17] or the appendix A.2.

If there are more than one protocol providing this data, it's suggested the protocols shown in the previous table for the reasons shown below.

The Modbus TCP [8] protocol has been chosen as the first option due to:

- Can be used in multiple devices.
- Robot data is always accessible.
- Essential information is available.
- It's data codification is easy to understand and implement.

The Profinet[10]/EthernetIP[9] protocol has been chosen as the second option due to:

- It provides more data than Modbus, but their use is linked to a brand (Siemens by the case of Profinet & Rockwell by the case of EthernetIP).
- They have companies behind giving support to them.
- Universal Robots facilitates its use providing modules to easy connect them to the PLCs.

The TCP/IP Real Time Exchanger[6] protocol has been chosen as the third option due to:

- Its protocol is specific of the Universal Robots, and a specific code has to be develop to extract data using this protocol.
- Almost all the information that Universal Robots provide can be obtained using this protocol.
- Accepts commands to modify the data provided by this protocol and get only the data of interest.

Justification: Remote Control

The TCP/IP Realt Time protocol has been chosen as the first option due to:

- It allows to control the robot without need of any previous process in the robot.
- It goes fast, 125hz in the CB-series and 500hz in the e-series
- It's easy to use with the URScript commands, and can do everything you can do in a robot program.

The TCP/IP Socket protocol has been chosen as the second option due to:

- It's the easier protocol to interact with a running program.

The Modbus TCP has been chosen as the third option due to:

- The protocol is more complex than the TCP/IP Socket
- It's a great option for an specific type of remote control, the Slave - Master remote control, were you control 1 robot and have another one following it. All the position data of the Master robot is provided and you only have to tell the slave robot to copy this data.
- Universal Robots provide a variable creation and reading for Modbus registers in the installation tab, that makes this approach easier.

The Profinet/Ethernet IP has been chosen as the fourth option due to:

- The protocol is more complex than Modbus TCP.
- The amount of devices which can be used are less.
- It's great option for an specific type of remote control, the Slave - Master remote control, were you control 1 robot and have another one following it. All the position data of the Master robot is provided and you only have to tell the slave robot to copy this data.
- There is not an installation option to create variables, they have to be created in the robot program, increasing the complexity.

Justification: Dynamic Programs

The XML-RPC protocol has been chosen as the first option due to:

- It makes much more easier to interact with provided data, you only have to create the functions with the expected parameters, the process to be done and register the function as an available function.
- Can be written in Python and run in the same robot using terminal commands.

The TCP/IP Socket protocol has been chosen as the second option due to:

- It can easily programmed in a computer to interact with the robot
- Can be written in python and run in the same robot using terminal commands.
- You have to create and structure to manage the received data and respond consequently, which makes it more complex than XML-RPC.

The Modbus TCP has been chosen as the third option due to:

- The protocol is more complex than the TCP/IP Socket
- Can be used in computers and PLCs
- It's a great option if we are working with huge amount of data which it's constantly changing, for example to avoid collisions with another robot, when they are working in the same area.
- Universal Robots provide a variable creation and reading for Modbus registers in the installation tab, that makes this approach easier.

The Profinet/Ethernet IP has been chosen as the fourth option due to:

- The protocol is more complex than Modbus TCP.
- The amount of devices which can be used are less.
- It's a great option if we are working with huge amount of data which it's constantly changing, for example to avoid collisions with another robot, when they are working in the same area.
- There is not a installation option to create variables, they have to be created in the robot program, increasing the complexity.

Justification: Remote Maintenance

The TCP/IP Dashboard has been always selected because it's the only want that allows to do maintenance operations in the robot, but there are some commands that doesn't exist which with the help of other protocols can be acquired.

The TCP/IP Dashboard & FTP (SFTP) has been chosen as the first option due to:

- The FTP allows us to know which programs and installation exist in order to ask the robot to load them using TCP/IP Dashboard
- FTP also allows us to transfer new programs and installation to the robot, modify them or do a safety copy.
- The SFTP allows use to do all that encrypting the information to avoid that anyone who it's in the network, steals our data.
- Exist free software such as Filezilla, that allows the user do these actions using only mouse comands.

The TCP/IP Dashboard & SSH (Terminal) has been chosen as the second option due to:

- It's more complex than the FTP, the user has to know the terminal commands in order to use it.
- It allows us to do more things than FTP, such as launch a python code to interact with the robot. But also, if we don't know what we are doing we can damage the robot software much easily.

Justification: Remote Access

It has been chosen the SSH protocol because it's the only one provided by UR. It's true that exist more user friendly remote access tools provided by private companies, but we are not evaluating these options.

4 Toolkit created

From the different teleoperating operations selected in the previous section:

Data acquisition → Modbus

Remote control → TCP/IP Real Time

Dynamics → XML-RPC

Remote Maintenance → TCP/IP Dashboard & FTP (SFTP)

Remote Access → SSH

From these options the most interesting ones in order to apply teleportation are: the Data acquisition, Remote Control and Remote Maintenance. In the case of the FTP, it's interesting mainly to know which programs the robot has and load files to the robot, for these purpose exist already software of free use easy to understand such as Filezilla, and for this reason a toolkit to do these actions have not been done. For the rest of protocols we don't have a program that allows us to do the desired actions and it's interesting to have toolkit with the essentials functions to use them.

It has been decided to create a class with all the functions of each protocol, to be able to communicate with multiple robots easily. The user can decide to create an object of the class *UR_Communication* for each robot, that way you have for each robot their Modbus,TCP/IP Dashboard & TCP/IP Scripts functions. If they are interested only in the functions of one of the teleoperanting protocols, as a modular approach has been chosen creating a class for each teleoperating protocol, they can create an object of the teleoperating protocol of interest.

All the functions created can be consulted in the appendix B, additionally it can be seen in the application some of the possibilities of the Toolbox.

The toolbox can be dowload from `https://github.com/SergiPonsa/UR-Communication-SergiPonsa.git`

5 Application based on the toolkit

The first we have to start the communication, by providing the Robot IP.

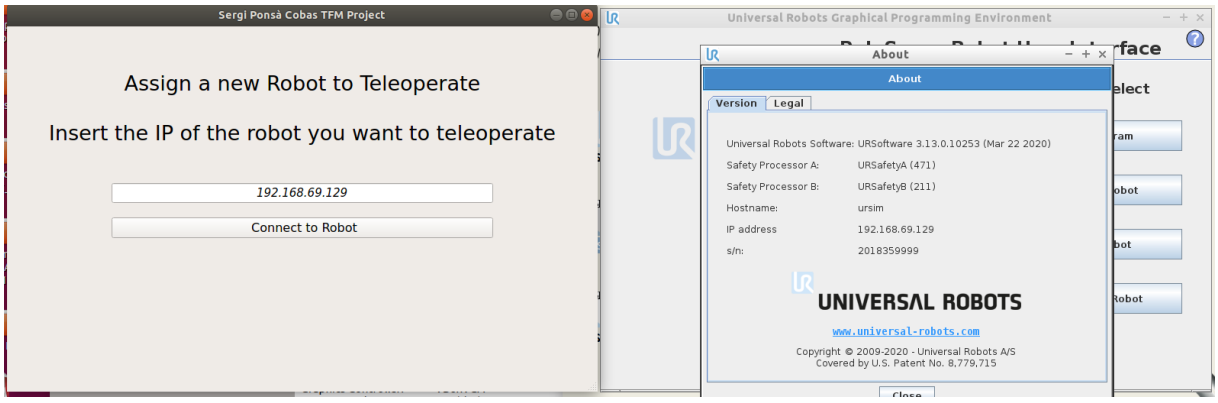


Figure 14: App connection screen

Once we have the communication open we have to use DashBoard functions to turn on and release the breaks.

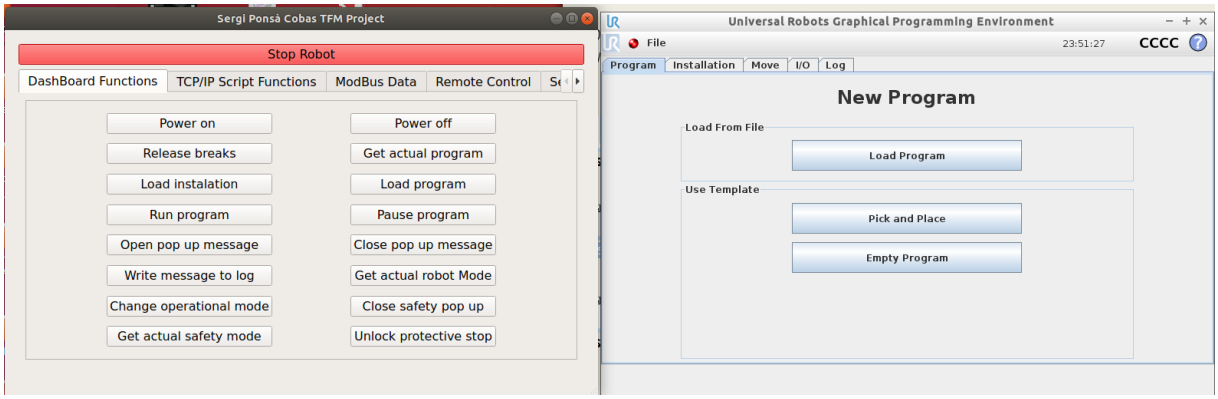


Figure 15: App Dashboard screen

Then we can preform all the actions of the Toolbox.
We can send more Dashboard commands, send Scripts to be executed or consult the robot Modbus data.

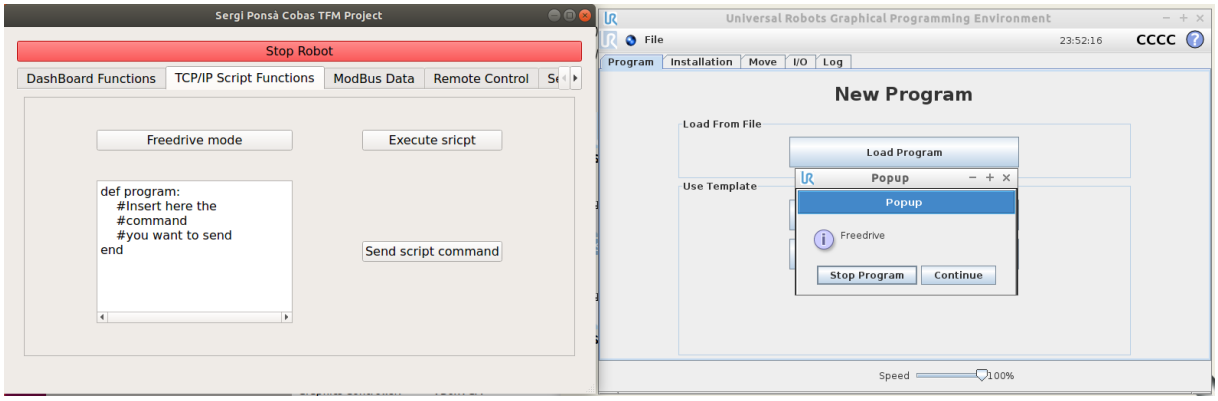


Figure 16: App UR Script screen

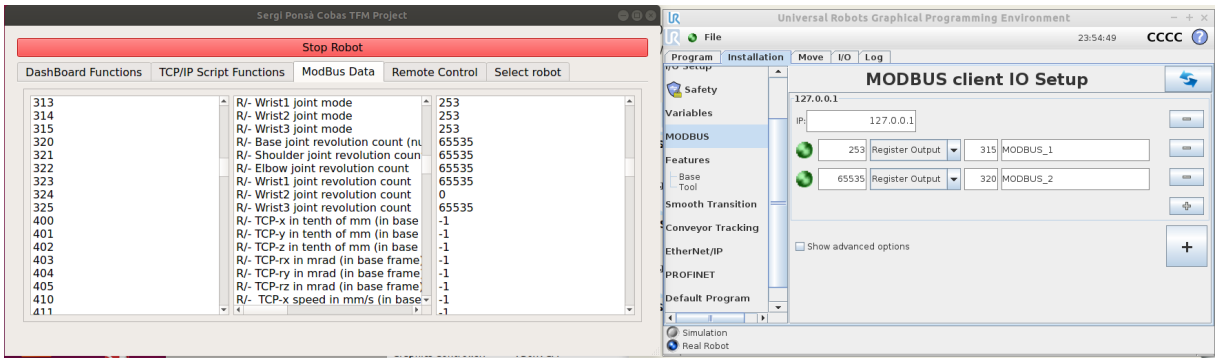


Figure 17: App Modbus screen

If we use the Modbus data and the URScipts functions to remote control the robot.

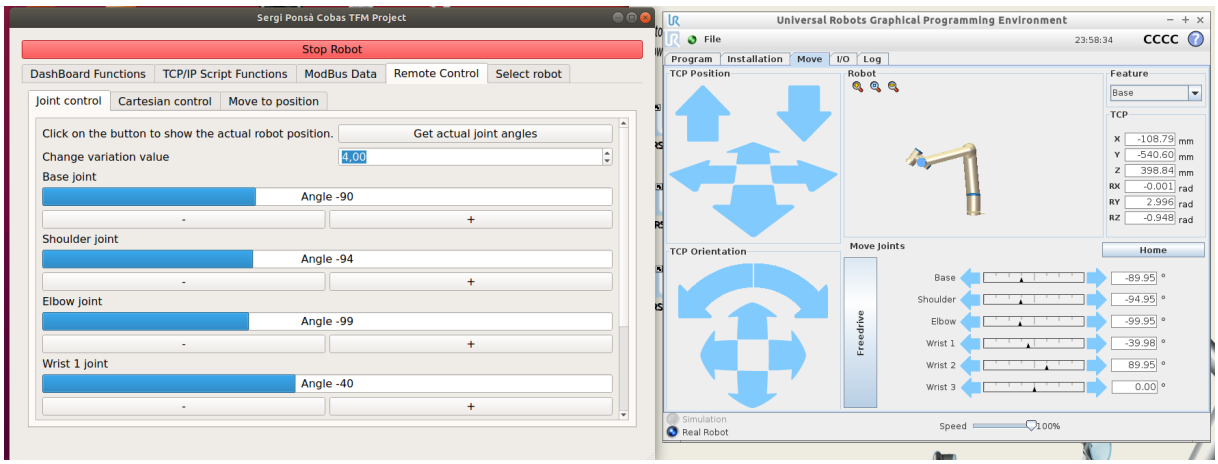


Figure 18: App Remote Control screen

As the toolbox is based on classes, we can create new object with the communication of another robot, to control different robots.

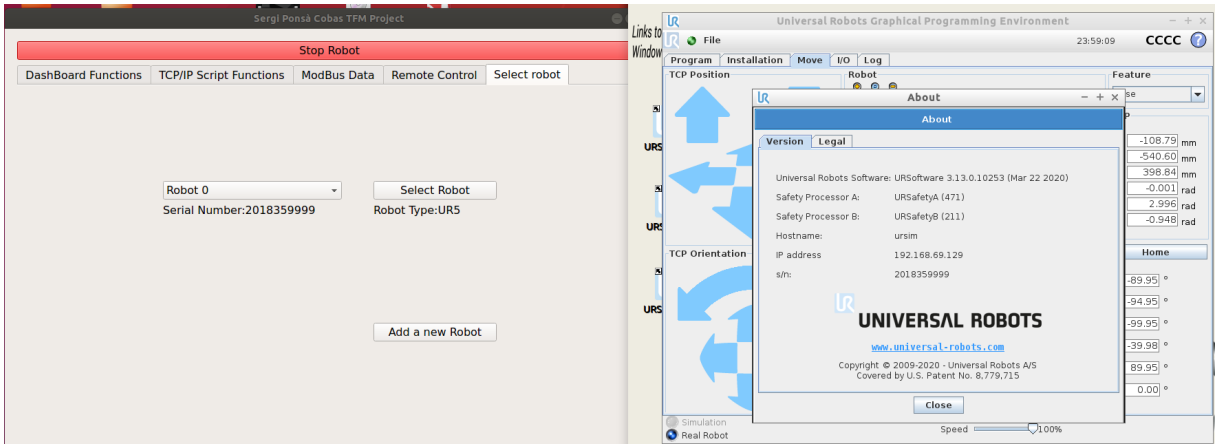


Figure 19: App Select Robot screen

A video showing the application working with the simulator can be watch on the following link.
https://www.youtube.com/watch?v=POw3n69_7Nk&t=1251s
Due to the Covid situation , it has not been possible to test it with a real robot.

6 Budget

The current project is a research project so it only includes the software tool and the hours dedicated to develop the project. As the master final thesis has 12 credits, the total dedicated hours by the student are 360 hours.

6.1 Personal costs

This personal budget has been based on the last approved Spanish BOE [25]. This BOE establishes a salary of 23.618,28 annual gross euros, considering 230 working days (considering 23 holidays), resulting on 12,836 €/h.

Concept	**Hours**	**Cost in €**
Formation Core	16	205.376 €
Formation Advance	16	205.376 €
Formation Interfaces	8	102.688 €
Formation Industrial Communication	8	102.688 €
Research	112	1 437.632 €
ToolBox Development	100	1 283.6 €
App Development	30	385.08 €
Documentation	70	898.52 €
Total	360	4 620.96 €

6.2 Material costs

To develop this project a computer it's required & a UR robot or the Universal Robots free virtual machine.

Option A		**Option B**	
Material	Cost	Material	Cost
Computer	600 €	Computer	600 €
UR3	23 000 €	Virtual Machine	0 €

It has been decided to develop the product using the Option B and only use the robot to do the final test.

6.3 License costs

All the software use it's free by personal use, but some of the software, if it's used by commercial purposes it has a cost.

Software	**Personal use cost**	**Commercial use cost**
Python	0	0
Oracle VM	0	0
PyQT	0	550 €

6.4 Energy costs

Option A		Option B	
Device	Cosumption	Device	Consumption
Computer	50 W	Computer	50W
UR3	100 W	Virtual Machine	0W
Total consumption	150 W	Virtual Machine	50W
Total Cost 0.1199*10^{-3} €/Wh	0.017€/h	Virtual Machine	0.005€/h
Total Cost 312h	5.61€	Virtual Machine	1.87 €/h

6.5 Formation costs

Course	**Cost**
Formation Core	1000€
Formation Advance	1200€
Formation Interfaces	600€
Formation Industrial Communication	600€
Total	**3 400€**

6.6 Total cost

Personal costs	4 620.96€
Material costs	600€
License costs	0€
Energy costs	1.87€
Formation costs	3 400€
Total	**8 622.83€**

7 Environmental impact

Even this project is based in a software that by itself will not have an environmental impact, we can't omit that the use of it implies the need of a computer to execute it and can imply the use of a robot if it's not used with the simulator by educational purpose.

For this reason, the use of this software will have an environmental effect caused by the computer and the robot electrical consumptions.

It estimated that the emissions of 2019 was 241 g CO_2/kWh, so with 150 Wh the use of the application will cause 36,15 g CO_2/h

8 Conclusions

Taking into account the costs and it's environmental impact it's considered that the resulting application using the developed Toolkit could be useful and compete with the current commercial options.

The current commercial options are based mainly on VNC, which it's a protocol that allows you to interact with a device as if you where there by transmitting its display and transmitting your controls to the device. This allows them to have exactly the same functions which can be done in the Robot Teachpendat, due to they are doing the actions in the TeachPendant but the actions has to be preformed by a user, they can not be programmed.

In this project we haven't considered the VNC option due to it's not available by default, a software that enable and create a VNC server has to be installed in the robot.

This project approach works with every UR robot, without need to install anything previously and it controls the robot without interacting with the TeachPendant. This has the inconvenient that the graphical user interface to preform the actions is not provided, but it has the advantage that we can automatize much easily the actions using code.

(i.e. With our approach we could have a schedule program which by itself changed the loaded program to the expected production or make some robot actions in function of the data received, while the current commercial solutions have to send an email or sms to someone to connect with the robot and preform the action).

The commercial solutions allow the connection with external cameras to be able to see the robot while they are controlling it and they can see the robot movements from the TeachPendant animation. Currently our approcah doesn't allow this, but it could be added the connection with external cameras and an animation of the current robot configuration could be done using the joints data, which it's already available.

Acknowledgements

I would like to thank to the whole Universal Robots Technical Support team for their help during the process and their explanations.

Additionally, I would like to thank my TFM director Pere Ponsa for all his support and guidance during the project.

Finally, I will like to thank my family and to Carmen Aznar to help me during the whole process.

A Teleoperation Protocols Appendix

A.1 Dasboard Commnads

Command	Return value	Description	Supported from	Only remote control
Load <program.urp>	**On Success:** • “Loading program: <program.urp>” **On Failure:** • “File not found: <program.urp>” • “Error while loading program: <program.urp>	Returns when both program and associated installation has loaded (or failed). The load command fails if the associated installation requires confirmation of safety. The return value in this case will be 'Error while loading program'.	SW3.4.0 & 5.0.0	X
Play	**On success:** • "Starting program" **On failure:** • "Failed to execute: play"	Returns failure if the program fails to start. In previous versions this did not happen in all cases.	SW3.4.0 & 5.0.0	X
Stop	**On success:** • "Stopped" **On failure:** • "Failed to execute: stop"	Returns failure if the program fails to stop. In previous versions this did not happen in all cases.	SW3.4.0 & 5.0.0	
Pause	**On success:** • "Pausing program" **On failure:** • "Failed to execute: pause"	Returns failure if the program fails to pause. In previous versions this did not happen in all cases.	SW3.4.0 & 5.0.0	
Quit	"Disconnected"	Closes connection	SW1.4 & 5.0.0	
Shutdown	“Shutting down”	Shuts down and turns off robot and controller	SW1.4 & 5.0.0	
Running	"Program running: true" OR "Program running: false"	Execution state enquiry	SW1.6 & 5.0.0	
Robotmode	text is returned "Robotmode: <mode>", where <mode> is • NO_CONTROLLER • DISCONNECTED • CONFIRM_SAFETY • BOOTING • POWER_OFF • POWER_ON • IDLE • BACKDRIVE • RUNNING	Robot mode enquiry	SW1.6 & 5.0.0	
get loaded program	"Loaded program: <path to loaded program file>" OR "No program loaded"	Which program is loaded	SW1.6 & 5.0.0	
popup <popup-text>	“showing popup”	The popup-text will be translated to the selected language, if the text exists in the language file	SW1.6 & 5.0.0	
close popup	"closing popup"	Closes the popup	SW1.6 & 5.0.0	
addToLog <log-message>	"Added log message" Or "No log message to add"	Adds log-message to the Log history	SW1.8.1 & 5.0.0	
isProgramSaved	"true <program.name>" OR "false <program.name>"	Returns the save state of the active program and path to loaded program file.	SW1.8.1 & 5.0.0	
programState	"STOPPED" if no program is running "PLAYING" if program is running "PAUSED" if program is paused	Returns the state of the active program and path to loaded program file, or STOPPED in no programs is loaded.	SW1.8.1 & 5.0.0	
PolyscopeVersion	“version number, like "3.0.15547"	Returns the version of the Polyscope software	SW1.8.1 & 5.0.0	

set operational mode <mode>, where <mode> is • manual • automatic	"Setting operational mode: <mode>" OR "Failed setting operational mode: <mode>" • manual = Loading and editing programs is allowed • automatic = Loading and editing programs and installations is not allowed, only playing programs If this function is called the operational mode cannot be changed from PolyScope, and the user password is disabled.	Controls the operational mode. See User manual for details. Warning: This functionality is intended for using e.g. Ethernet based Key Card Readers to switch operational modes. The device for switching operational mode should be placed in vicinity to the robot.	5.0.0	
clear operational mode	"operational mode is no longer controlled by Dashboard Server" If this function is called the operational mode can again be changed from PolyScope, and the user password is enabled.		5.0.0	
power on	"Powering on"	Powers on the robot arm	SW3.0.0 & 5.0.0	X
power off	"Powering off"	Powers off the robot arm	SW3.0.0 & 5.0.0	
brake release	"Brake releasing"	Releases the brakes	SW3.0.0 & 5.0.0	X

safetymode (deprecated from software 3.11 – use "safetystatus" instead)	"Safetymode: <mode>", where <mode> Is • NORMAL • REDUCED • PROTECTIVE_STOP • RECOVERY • SAFEGUARD_STOP • SYSTEM_EMERGENCY_STOP • ROBOT_EMERGENCY_STOP • VIOLATION • FAULT	Safety mode inquiry. A Safeguard Stop resulting from any type of safeguard I/O or a configurable I/O three position enabling device result in SAFEGUARD_STOP. *This function is deprecated.* Instead, use 'safetystatus'.	SW3.0.0 & 5.0.0	
safetystatus	"Safetystatus: <status>", where <status> is • NORMAL • REDUCED • PROTECTIVE_STOP • RECOVERY • SAFEGUARD_STOP • SYSTEM_EMERGENCY_STOP • ROBOT_EMERGENCY_STOP • VIOLATION • FAULT • AUTOMATIC_MODE_SAFEGUARD_STOP • SYSTEM_THREE_POSITION_ENABLING_STOP	Safety status inquiry. This differs from 'safetymode' by specifying if a given Safeguard Stop was caused by the permanent safeguard I/O stop, a configurable I/O automatic mode safeguard stop or a configurable I/O three position enabling device stop. Thus, this is strictly more detailed than 'safetymode'.	SW3.11.0 & 5.4.0	

unlock protective stop **IMPORTANT:** It is the responsibility of the integrator to ensure the cause of the protective stop is resolved before using the "unlock protective stop" command	**On succes:** • "Protective stop releasing" **On failure:** • "Cannot unlock protective stop until 5s after occurrence. Always inspect cause of protective stop before unlocking"	Closes the current popup and unlocks protective stop. The unlock protective stop command fails if less than 5 seconds has passed since the protective stop occurred.	SW3.1.0 & 5.0.0	
close safety popup	"closing safety popup"	Closes a safety popup	SW3.1.0 & 5.0.0	
load installation <default.installation>	**On success:** • "Loading installation: <default.installation>" **On failure:** • "File not found: <default.installation>" • "Failed to load installation: <default.installation>"	Loads the specified installation file but does not return until the load has completed (or failed). The load command fails if the associated installation requires confirmation of safety. The return value will be 'Failed to load installation'.	SW3.4.0 & 5.0.0	X
restart safety IMPORTANT: You should always ensure it is okay to restart the system. It is highly recommended to check the error log before using this command (either via PolyScope or e.g. ssh connection).	Restarting safety	Used when robot gets a safety fault or violation to restart the safety. After safety has been rebooted the robot will be in Power Off.	SW3.7.0 & 5.1.0	X

get operational mode	MANUAL, AUTOMATIC or NONE	Returns the operational mode as MANUAL or AUTOMATIC if the password has been set for Mode in Settings. Returns NONE if the password has not been set.	5.6.0	
is in remote control	"true" or "false"	Returns the remote-control status of the robot. If the robot is in remote control it returns false and if remote control is disabled or robot is in local control it returns false.	5.6.0	
get serial number	Serial number like "20175599999"	Returns serial number of the robot.	SW3.12.0 & 5.6.0	
get robot model	UR3, UR5, UR10, UR16	Returns the robot model	SW3.12.0 & 5.6.0	
generate flight report <report type> where possible report types are: • controller • software • system Default Type is 'system' if no option is specified	On success report id is printed. Error Message on a failure. Command can take few minutes to complete.	Triggers a Flight Report of the following type: • Controller - report with information specific for diagnosing controller errors. For example, in case of protective stops, faults or violations. • Software - report with information specific for polyscope software failures. • System - report with information about robot configuration, programs, installations etc. It is required to wait at least 30 seconds between triggering software or controller reports.	SW3.13.0 & 5.8.0	

generate support file <Directory path> where <Directory path> represents path to an already existing directory location inside the programs directory. In particular path can point to special usbdisk subfolders inside programs folder.	On success "Completed successfully: <result file name>" is printed otherwise an error message with possible cause of the error is shown. Command can take up to *10 minutes* to complete.	Generates a flight report of the type "System" and creates a compressed collection of all the existing flight reports on the robot along with the generated flight report. Result file *ur_[robot serial number]_YYYY-MM-DD_HH-MM-SS.zip* is saved inside <Directory path>	SW3.13.0 & 5.8.0	

A.2 Data Matrix

Name	Primary Client	Secondary Client	Real Time Client	RTDE	Modbus	EIP & PN
Robot Data Source Summary						
Timestamp	x	x	x	x		
Controller Timer			x			
RTMachine Time					x	x
Actual Execution Time				x		
Physical Robot Connected	x	x				
Real Robot Enabled	x	x				
Robot Power On	x	x			x	x
is EmergencyStopped	x	x			x	x
is robot emergency stopped						
is system emergency stopped						x
is stopped due to safety						x
Is Fault						x
Is violation						x
Protective Stopped	x	x			x	x
Program Running	x	x				x
Program Paused	x	x				
Program State			x	x		
Robot Mode	x	x	x	x	x	x
Robot Status				x		
IsNormalMode						x
IsReducedMode						x
IsRecoveryMode						x
Robot State				x	x	
ControlMode	x	x				
Joint Mode			x	x		
Safety Mode			x	x		x
SafetySignalSuchThatWeShouldStop					x	
Target Speed Fraction	x	x				
Speed Scaling	x	x		x		
Speed Slider Mask				x		
Speed Slider Fraction				x	x	x
Linear momentum norm			x	x		
TargetSpeedFractionLimit	x	x	x	x		
Actual Joint Positions (array of six angles)	x	x	x	x	x	x
Target Joint Positions (array of six angles)	x	x	x	x		
Actual Velocity (array of six angular velocities)	x	x	x	x	x	x
qd target (velocity)			x	x		
qdd target (acceleration)			x	x		
Tool Acceleration	x		x			
Base joint revolution count (number of full turns, typically 0 or 1)					x	
Shoulder joint revolution count					x	
Elbow joint revolution count					x	
Wrist1 joint revolution count					x	
Wrist2 joint revolution count					x	
Wrist3 joint revolution count					x	
Actual Joint Current	x	x	x	x	x	x
I target (control)			x	x		
M target			x	x		
Actual Joint Voltage	x	x	x	x		
V mail			x			
V robot			x			
I robot			x	x		
Elbow position			x	x		
Elbow velocity			x	x		
Actual Joint Temperature	x	x	x	x	x	x
Joint Mode	x	x			x	x
Tool Analog Input 1 Range (voltage or current)	x	x		x	x	x
Tool Analog Input 2 Range (voltage or current)	x	x		x	x	x
Tool Analog Input 1	x	x		x	x	x
Tool Analog Input2	x	x		x	x	x
Tool Voltage 48V	x	x				
Tool Output Voltage (0,12,24)	x	x		x	x	x
Tool Current	x	x		x	x	x
Tool Temperature	x	x		x		
Tool Mode	x	x		x		

Tool States					x	
Tool State					x	
Tool Temp					x	
DigitalInputBits?	x	x	x		x	x
DigitalOutputBits?	x	x	x	x	x	x
ToolDigitalInputBits				x	x	
ToolDigitalOutputBits				x	x	x
Tool Digital Output Mask (TDOM)						x
Tool Digital Inputs (TDI)						x
Tool Digital Outputs (TDO)						x
Tool Analog Input Types (TAIT)				x		x
SetOutputBitsMask				x	x	x
ClearOutputBitsMask					x	
Configurable inputs, bits [BBBBBBBBxxxxxxxx] x=undef, T=tool, B=box					x	x
Configurable outputs, bits [BBBBBBBBxxxxxxxx] x=undef, T=tool, B=box				x	x	x
Bit mask configurable outputs, bits [BBBBBBBBxxxxxxxx] x=undef, T=tool, B=box				x	x	x
Clear configurable outputs, bits [BBBBBBBBxxxxxxxx] x=undef, T=tool, B=box					x	
Modbus General purpose 16 bit registers					x	
Internal Bit input registers				x		x
Internal Bit output registers				x		x
Internal Integer Input registers				x		x
Internal Integer Output registers				x		x
Internal Input Double registers				x		
Internal Output Double registers				x		
Internal Float input registers						x
Internal Float output registers						x
Analog Input 0 Range	x	x		x	x	
Analog Input 1 Range	x	x		x	x	
AOM						x
AOT						x
Analog Input0	x	x		x	x	x
Analog Input1	x	x		x	x	x
Analog Output 0 Range	x	x		x	x	x
Analog Output 1 Range	x	x		x	x	x

Analog Output 0	x	x		x	x	x
Analog Output	x	x		x	x	x
Safety Control Board Temperature	x	x				
Robot Voltage 48V	x	x		x		
Main Voltage				x		
Joint Voltage				x		
RobotCurrent?	x	x		x	x	x
MasterIOCurrent?	x	x		x	x	x
Safetymode	x	x				
InReducedMode?	x	x				
Euromap67Installed?	x	x				
EuromapInputBits?	x	x		x	x	x
EuromapOutputBits?	x	x		x	x	x
EuromapVoltage?	x	x		x	x	x
EuromapCurrent?	x	x		x	x	x
OperationalModeSelectorInput	x	x				
ThreePositionEnablingDeviceInput	x	x				
Tool Position X	x	x		x	x	x
Tool Position Y	x	x		x	x	x
Tool Position Z	x	x		x	x	x
Tool Position Rx	x	x		x	x	x
Tool Position Ry	x	x		x	x	x
Tool Position Rz	x	x		x	x	x
Tool vector actual			x			
Tool vector target			x			
TCP Target				x		
TCP speed actual			x	x	x	x
TCP speed target			x	x		
TCP X	x	x			x	
TCP Y	x	x			x	
TCP Z	x	x			x	
TCP RX	x	x			x	
TCP RY	x	x			x	
TCP RZ	x	x			x	

Joint Min Rotation Limit	x	x				
Joint Max RotationLimit	x	x				
Joint Max Speed	x	x				
Joint Max Acceleration	x	x				
VJointDefault?	x	x				
AJointDefault?	x	x				
VToolDefault?	x	x				
AToolDefault?	x	x				
EqRadius?	x	x				
DHa	x	x				
DHd	x	x				
DHalpha	x	x				
DHtheta	x	x				
Masterboard Version	x	x				
Controller Box Type	x	x				
Controller version high nuber					x	x
Controller version low number					x	x
Robot Type	x	x				
Robot Sub Type	x	x				
TCP FX	x	x		x		x
TCP FY	x	x		x		x
TCP FZ	x	x		x		x
TCP MRx	x	x		x		x
TCP MRy	x	x		x		x
TCP MRz	x	x		x		x
TCP Force Scalar				x		
TCP for e						
RobotDexterity?	x	x				
FreedriveButtonPressed	x	x			x	
FreedriveButtonEnabled	x	x				
PowerButtonPressed					x	x
IOEnabledFreedrive	x	x				
project Name	x	x				
major Version	x	x				
minor Version	x	x				
svn Revision	x	x				
build Date	x	x				

A.3 XML-RPC

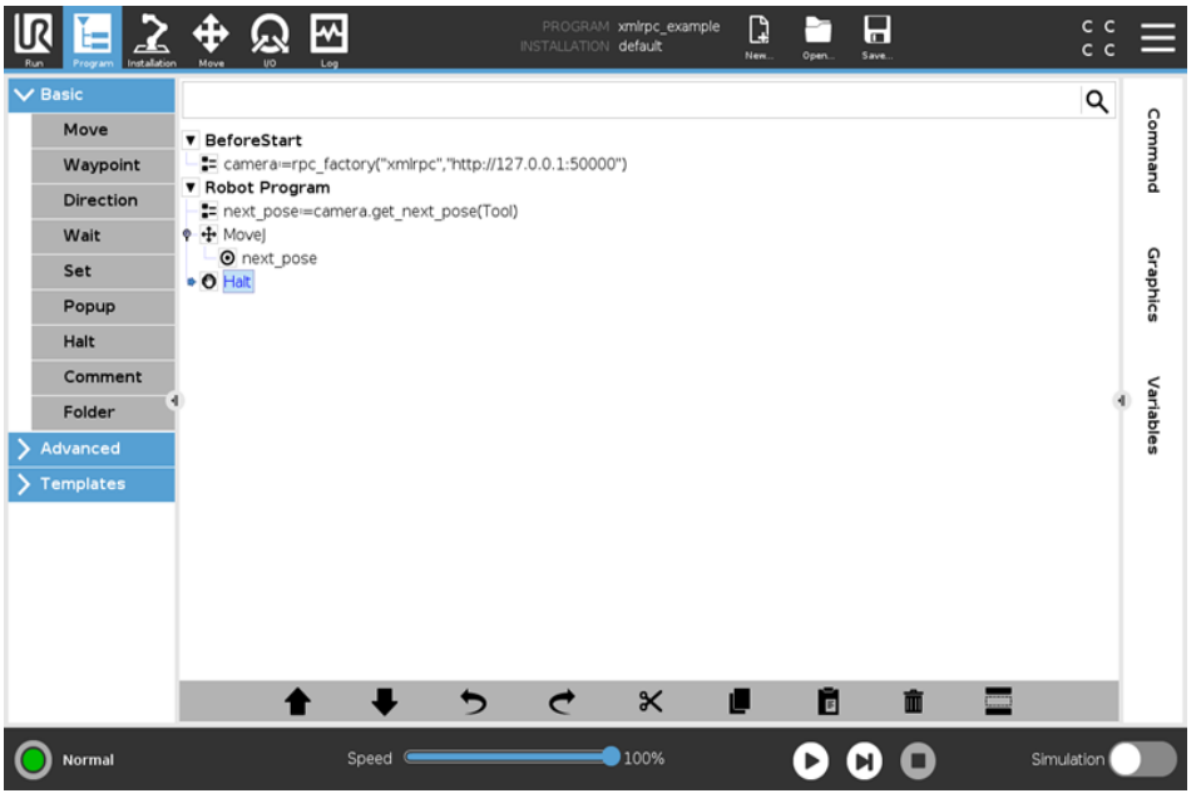
```
import sys
import urlib
is_py2 = sys.version[0] == '2'
if is_py2:
  from SimpleXMLRPCServer import SimpleXMLRPCServer
else:
  from xmlrpc.server import SimpleXMLRPCServer

def get_next_pose(p):
   assert type(p) is dict
   pose = urlib.poseToList(p)
   print("Received pose: " + str(pose))
   pose = [-0.18, -0.61, 0.23, 0, 3.12, 0.04];
   return urlib.listToPose(pose);

server = SimpleXMLRPCServer(("", 50000), allow_none=True)
server.RequestHandlerClass.protocol_version = "HTTP/1.1"
print("Listening on port 50000...")

server.register_function(get_next_pose, "get_next_pose")

server.serve_forever()
```



A.4 Modbus

Modbus Request ADU Example Header

MBAP Header Fields	Example Decimal (Hexadecimal)
Transaction ID High Order	0 (00) *Client sets, unique value.*
Transaction ID Low Order	1 (01) *Client sets, unique value.*
Protocol Identifier High Order	0 (00) *Specifies Modbus service.*
Protocol Identifier Low Order	0 (00) *Specifies Modbus service.*
Length High Order	0 (00) *Client calculates.*
Length Low Order	6 (06) *Client calculates.*
Unit Identifier	255 (FF) or 0 (00) *Do not bridge.*

Modbus Request ADU Example - Read Coil Status Query

Field Name	Example Decimal (Hexadecimal)
Function Code	1 (01)
Starting Address High Order	0 (00)
Starting Address Low Order	0 (00)
Number Of Points High Order	0 (00)
Number Of Points Low Order	4 (04)

Modbus Response ADU Example Header

MBAP Header Fields	Example Decimal (Hexadecimal)
Transaction ID High Order	0 (00) *Echoed back, no change*
Transaction ID Low Order	1 (01) *Echoed back, no change*
Protocol Identifier High Order	0 (00) *Echoed back, no change*
Protocol Identifier Low Order	0 (00) *Echoed back, no change*
Length High Order	0 (00) *Server calculates*
Length Low Order	4 (04) *Server calculates.*
Unit Identifier	255 (FF) or 0 (00) *No change*

Modbus Response ADU Example - Read Coil Status Response

Field Name	Example Decimal (Hexadecimal)
Function Code	1 (01)
Byte Count	1 (01)
Data (Coils 3-0)	10 (0A)

Modbus PDU Example - Read Holding Register Query

Field Name	Example Decimal (Hexadecimal)
Function Code	3 (03)
Starting Address High Order	0 (00)
Starting Address Low Order	5 (05)
Number Of Points High Order	0 (00)
Number Of Points Low Order	3 (03)

Modbus PDU Example - Read Holding Register Response

Field Name	Example Decimal (Hexadecimal)
Function Code	3 (03)
Byte Count	6 (06)
Data High (Register 40006)	(3A)
Data Low (Register 40006)	75%=15000 (98)
Data High (Register 40007)	(13)
Data Low (Register 40007)	25%=5000 (88)
Data High (Register 40008)	(00)
Data Low (Register 40008)	1%=200 (C8)

Modbus PDU Example - Read Input Registers Query

Field Name	Example Decimal (Hexadecimal)
Function Code	4 (04)
Starting Address High Order	0 (00)
Starting Address Low Order	2 (02)
Number Of Points High Order	0 (00)
Number Of Points Low Order	2 (02)

Modbus PDU Example - Read Input Registers Response

Field Name	Example Decimal (Hexadecimal)
Function Code	4 (04)
Byte Count	4 (04)
Data High (Register 30003)	(3E)
Data Low (Register 30003)	80%=16000 (80)
Data High (Register 30004)	(00)
Data Low (Register 30004)	136 (88)

Modbus PDU Example - Force Single Coil Query and Response

Field Name	Example Decimal (Hexadecimal)
Function Code	5 (05)
Coil Address High Order	0 (00)
Coil Address Low Order	3 (03)
Force Data High Order	255 (FF)
Force Data Low Order	0 (00)

The Force Single Coil response message is simply an echo (copy) of the query as shown above, but returned after executing the force coil command. No response is returned to broadcast queries from a master device (serial Modbus only).

Modbus PDU Example - Preset Holding Register Query and Response

Field Name	Example Decimal (Hexadecimal)
Function Code	6 (06)
Register Address High Order	0 (00)
Register Address Low Order	1 (01)
Preset Data High Order	0 (00)
Preset Data Low Order	2 (02)

The response message is simply an echo (copy) of the query as shown above, but returned after the register contents have been preset. No response is returned to broadcast queries from a master (serial Modbus only).

Modbus PDU Example - Force Multiple Coils Query

Field Name	Example Decimal (Hexadecimal)
Function Code	15 (0F)
Coil Address High Order	0 (00)
Coil Address Low Order	0 (00)
Number Of Coils High Order	0 (00)
Number Of Coils Low Order	4 (04)
Byte Count	01
Force Data High (First Byte)	5 (05)

A.5 Ethernet/IP

Ethernet/IP Commands

CODE (HEX)	DESCRIPTION
0000H	NOP (No operation) – Sent only via TCP.
0001..0003H	*Reserved for legacy*
0004H	List_Services - May be sent via TCP or UDP
0005H	*Reserved for legacy*
0006..0062H	*Reserved for future expansion – compliant products may not use command codes in this range.*
0063H	List_Identity – May be sent via TCP or UDP.
0064H	List_Interfaces (optional) - May be sent via TCP or UDP.
0065H	Register_Session – Sent only via TCP.
0066H	UnRegister_Session – Sent only via TCP.
0067..006EH	*Reserved for legacy*
006FH	SendRRData – Sent only via TCP.
0070H	SendUnitData – Sent only via TCP.
0071H	*Reserved for legacy*
0072H	Indicate_Status (optional) – Sent only via TCP.
0073H	Cancel (optional) – Sent only via TCP.
0074..00C7H	*Reserved for legacy*
00C8..FFFFH	*Reserved for future expansion – compliant products may not use command codes in this range.*

Ethernet/IP Status

CODE (HEX)	DESCRIPTION
0000H	Indicates Success
0001H	Invalid or unsupported command.
0002H	Insufficient memory resources for processing command.
0003H	Poorly formed or incorrect data in data portion of message.
0004..0063H	Reserved for legacy.
0064H	Invalid session handle.
0065H	Invalid length message.
0066..0068H	Reserved for legacy.
0069H	Unsupported encapsulation protocol revision.
006A..FFFFH	Reserved for future expansion. Compliant products may not use error codes in this range.

A.6 FTP

FTP Commands

Table 8-6: Most useful FTP commands

Command	Argument	Description
USER	username	User name allocated by ISP
PASS	password	Password (allocated by ISP)
ACCT	account info	User account
CWD	pathname	Change working directory
CDUP	none	Change to parent directory
SMNT	pathname	Structure mount
REIN	none	Terminate and re-initialize
QUIT	none	Logout FTP
RETR	pathname	Retrieve file from server
STOR	pathname	Store data to server
RNFR	pathname	Rename 'from'
RNTO	pathname	Rename 'to'
DELE	pathname	Delete file
RMD	pathname	Remove directory
MKD	pathname	Make directory
LIST	pathname	List files or text
STAT	pathname	Status
HELP	subject	Print help on
PORT	host-port	Specify port for transfer (non-default)
TYPE	type code	Type of transfer (ASCII, image, etc.)
MODE	mode code	Transmission mode (stream, block, etc.)

FTP Responds

Table 8-7: Format for replies in manual interactive Telnet

Reply	Description
1yz	Action initiated. Expect another reply before sending a new command
2yz	Action completed. Can send a new command
3yz	Command accepted, but on hold due to lack of information
4yz	Command not accepted or completed. Temporary error condition exists. Command can be re-issued
5yz	Command not accepted or completed. Do not re-issue – reissuing the command will result in the same error.

B Toolbox Appendix

B.1 TCP/IP Dashboard

B.1.1 TCP/IP Dashboard Parameters

DashBoard_Connected	It's a boolean variable that check if the last communication with the robot was successful or not, if it was not successful it's set to false. It's used in all the communication functions, to create again the socket of communication if for some reason the communication has been lost and the socket has been closed.
DashBoard_sock	It's a socked created to do all the communication of DashBoard to the port 29999
RobotIPDashBoard	It's a string variable, that saves the Robot IP we are doing the communication with.

B.1.2 TCP/IP Dashboard Functions

DashBoard_Connect	**Parameters:** None **Returns:** Nothing It's used to try open a socket of communication with the Robot port 29999 in 3 intents, at the first intent that it's able to connect sets the parameter Dashboard_Connected to True and it doesn't do more intents. If the 3 intents have pass, and still it's not able to connect, the parameter Dashboard_Connected is set to false and a message of error it's printed
DashBoard_SendCommand	**Parameters:** cmd (string) **Returns:** Nothing It sends the command wanted to be preformed to the Robot Dashboard server

DashBoard_GetData	**Parameters:** None **Returns:** Data (string) It reads the buffer of data send by the robot Dashboard server, and clears the buffer. If we are interested in a specific server response from a function it's recommended to call first this function to clear the buffer, call the function off interest and then call again this function to get that specific response.
DashBoard_LoadProgram	**Parameters:** ProgramName (string) **Returns:** Nothing It loads the desired program from the robot programs root directory. If the program it's in a folder inside the programs root directory, the directory has to be provided too with the name, in the usual file exploration format. The Robot extension .urp, it's already concatenated to the provided name.
DashBoard_RunProgram	**Parameters:** None **Returns:** Nothing It runs the current program loaded in the robot. Take into consideration that UR by safety asks to move to the robot to the first position of the program before start it by pressing a button in the Teachpendant. To avoid that, the first position has to be relative to the current robot position. It can be just mode to the current robot position, but if the first movement it's not that and the robot it's not actually in the first position of the program, the program will not start.

DashBoard_StopProgram	**Parameters:** None **Returns:** Nothing If there is a program running stop the robot in the current position and if we run again the program, it will start again from the first line of the program. If there isn't a program running it doesn't do anything.
DashBoard_PauseProgram	**Parameters:** None **Returns:** Nothing If there is a program running stop the robot in the current position and if we run again the program, it will start again from the line of the program it was before it was stopped it. If there isn't a program running it doesn't do anything.
DashBoard_ShutDownRobot	**Parameters:** None **Returns:** Nothing It shutdowns the Robot Control box and the robots. It disconnects the power from the robot and sets the breaks before the shutdown. It can only be turned on by pressing the power on button in the Teachpendant or by an electrical signal to the remote power on input of the robot.
DashBoard_RobotState	**Parameters:** None **Returns:** Nothing It informs about the current state of the robot. i.e: Power off, IDLE (The robot has power, but the breaks are set), Power On If the robot doesn't do anything, can be due to it's not powered or the breaks are set, it allows, this command allows use to check if this it's the reason remotely. It's recommended to check the safety status too,it can be that the robot it's stopped because a sa If we want to get the server response to this function we have to call the function Dashboard_GetData

DashBoard_ProgramState	**Parameters:** None **Returns:** Nothing It allows us to check if there is a program ruining in the robot or not. If we want to get the server response to this function we have to call the function Dashboard_GetData
DashBoard_ProgramLoaded	**Parameters:** None **Returns:** Nothing It allows us to know which program it's currently load in the robot If we want to get the server response to this function we have to call the function Dashboard_GetData
DashBoard_PopUp	**Parameters:** None **Returns:** Nothing It allows us to send an emergent window message in the Robot Teachpendant. It will remain there until a close popup message it's send or someone close it from the Teachpendant. It's recommended to send popup when we are teleoperating the robot, to inform to the rest of people around, that the robot it's currently controlled externally.
DashBoard_ClosePopUp	**Parameters:** None **Returns:** Nothing It allows us to close an emergent window message in the Robot Teachpendant. If the popup it's a safety message will remain.

DashBoard_WriteToLog	**Parameters:** None **Returns:** Nothing It allows us to write and informative message to the robot log file, with the time and date when it was wrote. The log file messages can't not be erased. The log file can be extracted from the robot and read with, a software provided by UR.
DashBoard_CheckSavedProgram	**Parameters:** None **Returns:** Nothing It allows us to know if the program currently load, if it has been modified , if the changes have been saved or not. If we want to get the server response to this function we have to call the function Dashboard_GetData
DashBoard_RobotVersion	**Parameters:** None **Returns:** Nothing It allows us to know the robot polyscope version. If we want to get the server response to this function we have to call the function Dashboard_GetData
DashBoard_SerialNumber	**Parameters:** None **Returns:** Nothing It allows us to know the robot serial number, it can be useful to check that we are teleoperating it's the one we wanted to teleoperate . If we want to get the server response to this function we have to call the function Dashboard_GetData

DashBoard_Model	**Parameters:** None **Returns:** Nothing It allows us to know the robot model, if it's a UR3, UR5, UR10 or UR16 . If we want to get the server response to this function we have to call the function Dashboard_GetData
DashBoard_SetUserRole	**Parameters:** option (integer (0 to 4)) **Returns:** Nothing It allows us to block some of the Teachpendant options depending on the user it's going to use the robot **Operator (option=0)** Only it's able to run program, it's not able to create or modify them. The configuration options are locked. **None (option=1)** Everything it's unlocked. **Programmer (option=2)** Some configurations options are locked **Locked (option=3)** The user can't do anything in the Teachpendant
DashBoard_PowerOn	**Parameters:** None **Returns:** Nothing It allows us power on the robot arm, remember to release the breaks before moving the robot
DashBoard_PowerOff	**Parameters:** None **Returns:** Nothing It allows us power off the robot arm, it will set the breaks too.

DashBoard_BrakeRelease	**Parameters:** None **Returns:** Nothing It allows us to release the breaks when the robot arm has power.
DashBoard_CheckSafetyMode	**Parameters:** None **Returns:** Nothing It allows us to know the robot safety state If we want to get the server response to this function we have to call the function Dashboard_GetData
DashBoard_UnlockProtectiveStop	**Parameters:** None **Returns:** Nothing It allows us unlock the robot, if for some reason the robot has done a protective stop Before unlocking the robot we should check the cause of that protective stop by safety.
DashBoard_CloseSafetyPopUp	**Parameters:** None **Returns:** Nothing It allows us close any safety popup resulting of a the robot safety stop, if for some reason the robot has done a protective stop

DashBoard_LoadInstallation	**Parameters:** InstallationFile(string) **Returns:** Nothing It loads the desired installation from the robot programs root directory. If the installation it's in a folder inside the programs root directory, the directory has to be provided too with the name, in the usual file exploration format. The Robot extension .installation, it's already concatenated to the provided name.
DashBoard_RestartSafety	**Parameters:** None **Returns:** Nothing It loads the desired installation from the robot programs root directory. If the installation it's in a folder inside the programs root directory, the directory has to be provided too with the name, in the usual file exploration format.

B.2 TCP/IP Script

B.2.1 TCP/IP Script Parameters

URScript_Connected	It's a boolean variable that check if the last communication with the robot was successful or not, if it was not successful it's set to false. It's used in all the communication functions, to create again the socket of communication if for some reason the communication has been lost and the socket has been closed.
URScript_sock	It's a socked created to transmit UR scripts to be executed by the robot, using the port 30002
RobotIPURScript	It's a string variable, that saves the Robot IP we are doing the communication with.

B.2.2 TCP/IP Script Functions

URScript_Connect	**Parameters:** None **Returns:** Nothing It's used to try open a socket of communication with the Robot port 30002 in 3 intents, at the first intent that it's able to connect sets the parameter URScript_Connected to True and it doesn't do more intents. If the 3 intents have pass, and still it's not able to connect, the parameter Dashboard_Connected is set to false and a message of error it's printed
URScript_SendCommand	**Parameters:** cmd (string) **Returns:** Nothing It sends the command wanted to be preformed by the Secondary server
URScript_SetFreeDrive	**Parameters:** None **Returns:** Nothing It show a popup in the Teachpendant which says "Freedrive" and if the continue button it's clicked, sets the robot in freedrive. When the robot it's in freedrive, allows the user to move it wherever they want. **Be careful, check the payload before activate freedrive.** If the payload it's less than the actual weight it's carrying the robot will fall, but if it's higher I will rise with the difference of force and could hit you.
URScript_StopFreeDrive	**Parameters:** None **Returns:** Nothing It stops the freedrive mode and show a popup in the Teachpendant which says "End Freedrive".

URScript_StopJoints	**Parameters:** a(optional,double) it's the acceleriton with will brake the joints, by default 2 **Returns:** Nothing It stops the current robot program and slowdown the movement, until the robot movement it's 0. **Be careful it doesn't substitute the emergency stop**
URScript_MoveJoints	**Parameters:** q (list of 6 doubles): It's the joint position we want to move, we can also give the pose p[] and the robot will do the inverse kinematics. Joint pos. [base,shoulder,elbow,w1,w2,w3] rad Robot pose p[x,y,z,ax,ay,az] (ax,ay,az, is the rotation vector, it's not intuitive use a function to transform RollPitchYaw to rotation vector), check function MoveToPoseJointSpaceBaseFrame a (optional,double): joints acceleration [rad/s^2], by default 1.4 v (optional,double): joint speed [rad/s] t (optional,double): It's used to specify the time [s] to reach the point, if it's different from 0, then it doesn't use the parameter a & v r (optional,double): to specify the blend radius[m] **Returns:** Nothing Ask the robot to do a movej to the specified position, movej will do the easier path for the joints, we can't predict the trajectory & ensure that can't collide If we want that the robot follows a linear trajectory we have to use a cartesian movement (a movel or movep)

URScript_MoveToPoseJointSpaceBaseFrame	**Parameters:** pose (list of 6 doubles): It's the TCP pose [x,y,z,rx,ry,rz], [x,y,z] in m [rx,ry,rz] orientation in RollPitchYaw in rad. This function converts the rx,ry,rz to rotational vector a (optional,double): joints acceleration [rad/s^2], by default 1.4 v (optional,double): joint speed [rad/s] t (optional,double): It's used to specify the time [s] to reach the point, if it's different from 0, then it doesn't use the parameter a & v r (optional,double): to specify the blend radius[m] **Returns:** Nothing Ask the robot to do a movej to the specified TCP pose in reference to the robot base. Movej will do the easier path for the joints, we can't predict the trajectory & ensure that can't collide If we want that the robot follows a linear trajectory we have to use a cartesian movement (a movel or movep)

URScript_MoveToPoseJointSpaceToolFrame	**Parameters:** pose (list of 6 doubles): It's the TCP pose [x,y,z,rx,ry,rz], in reference to the past TCP pose, [x,y,z] in m [rx,ry,rz] orientation in RollPitchYaw in rad. This function converts the rx,ry,rz to rotational vector and then transform it to the base reference a (optional,double): joints acceleration [rad/s^2], by default 1.4 v (optional,double): joint speed [rad/s] t (optional,double): It's used to specify the time [s] to reach the point, if it's different from 0, then it doesn't use the parameter a & v r (optional,double): to specify the blend radius[m] **Returns:** Nothing Ask the robot to do a movej to the specified TCP pose in reference to the robot base. Movej will do the easier path for the joints, we can't predict the trajectory & ensure that can't collide If we want that the robot follows a linear trajectory we have to use a cartesian movement (a movel or movep)
URScript_RemoteJointControl	**Parameters:** q (list of 6 doubles): It's the joint position we want to mo a (optional,double): joints acceleration [rad/s^2], by default 1.4 v (optional,double): joint speed [rad/s] t (optional,double): It's used to specify the time [s] to reach the point, if it's different from 0, then it doesn't use the parameter a & v r (optional,double): to specify the blend radius[m] **Returns:** Nothing It's behaviour it's similar to the MoveJoints function, but it's though to work with low displacements that change really fast, using the function servoj for movements lower than 5 degrees and the movej with larger movements.

URScript_RemoteCartesianControl	**Parameters:** pose (list of 6 doubles): It's the TCP pose [x,y,z,rx,ry,rz], a (optional,double): joints acceleration [rad/s^2], by default 1.4 v (optional,double): joint speed [rad/s] t (optional,double): It's used to specify the time [s] to reach the point, if it's different from 0, then it doesn't use the parameter a & v r (optional,double): to specify the blend radius[m] **Returns:** Nothing It's behaviour it's similar to the MoveToPoseJointSpaceBaseFrame function, but it's though to work with low displacements that change really fast, using the function servoj for movements lower than 5 degrees and the movej with larger movements
URScript_RemoteCartesianControl_offset	pose (list of 6 doubles): It's the TCP pose [x,y,z,rx,ry,rz], a (optional,double): joints acceleration [rad/s^2], by default 1.4 v (optional,double): joint speed [rad/s] t (optional,double): It's used to specify the time [s] to reach the point, if it's different from 0, then it doesn't use the parameter a & v r (optional,double): to specify the blend radius[m] **Returns:** Nothing It's behaviour it's similar to the MoveToPoseJointSpaceToolFrame function, but it's though to work with low displacements that change really fast, using the function servoj for movements lower than 5 degrees and the movej with larger movements

URScript_ScriptFile	**Parameters:** FilePath script file wanted to send **Returns:** Nothing It allows us to send the script program we want the robot execute

B.3 Modbus

B.3.1 Modbus Parameters

Modbus_Connected	It's a boolean variable that check if the last communication with the robot was successful or not, if it was not successful it's set to false. It's used in all the communication functions, to create again the socket of communication if for some reason the communication has been lost and the socket has been closed.
Modbus_sock	It's a socked created to transmit UR scripts to be executed by the robot, using the port 502
RobotIPModbus	It's a string variable, that saves the Robot IP we are doing the communication with.
Modbus_df	It's a pandas dataframe, with all the data from the Excel provided by UR. It's obtained from executing the Modbus_clean_data function
Modbus_df_reg	It's a pandas dataframe based on the dataframe from the Universal Robots Excel, from by the columns: Address, R, W, Description & Data. We will use this dataframe to save and consult data.
path_data	It's a string with the path of the Excel we are using to create the dataframes.
Modbus_registers_all_ind	It's an list of integers form by the indices of all the registers in the Modbus_df_reg

Modbus_registers_read_ind	It's an list of integers form by the indices of all the registers that can be read in the Modbus_df_reg
Modbus_registers_write_ind	It's an list of integers form by the indices of all the registers that can be write in the Modbus_df_reg
Modbus_registers_all	It's an list of integers form by the addresses of all the registers in the Modbus_df_reg
Modbus_registers_read	It's an list of integers form by the addresses of all the registers that can be read in the Modbus_df_reg
Modbus_registers_write	It's an list of integers form by the addresses of all the registers that can be write in the Modbus_df_reg
Modbus_registers_all_description	It's an list of string form by the descriptions of all the registers in the Modbus_df_reg
Modbus_registers_all_data	It's an list of integers form by the data of all the registers in the Modbus_df_reg

B.3.2 Modbus Functions

Modbus_Connect	**Parameters:** None **Returns:** Nothing It's used to try open a socket of communication with the Robot port 502 in 3 intents, at the first intent that it's able to connect sets the parameter URScript_Connected to True and it doesn't do more intents. If the 3 intents have pass, and still it's not able to connect, the parameter Dashboard_Connected is set to false and a message of error it's printed

Modbus_SendCommand	**Parameters:** cmd (string) **Returns:** Nothing It sends the command wanted to be preformed to the Modbus server
Modbus_GetData	**Parameters:** None **Returns:** Data (string) It reads the buffer of data send by the robot Dashboard server, and clears the buffer. If we are interested in a specific server response from a function it's recommended to call first this function to clear the buffer, call the function off interest and then call again this function to get that specific response.
Modbus_Request_to_write	**Parameters:** Addres_dec (integer): Address to write in decimal Value_to_write_dec (integer): Value to write in the seleceted register printb (boolean): If it's set to true will show the command send, by default set to False. **Returns:** Nothing It sends the robot modbus server a request to write a value in the desired register and reads to respond
Modbus_Request_to_read	**Parameters:** Addres_dec (integer): Address to read in decimal printb (boolean): If it's set to true will show the command send, by default set to False. **Returns:** value (integer) It sends the robot modbus server a request to read the desired register and receive the server respond

Modbus_dec_to_hex	**Parameters:** dec_value (integer): the value in Integer that we want to convert to hexadecimal **Returns:** [hex_hight_modbus , hex_low_modbus]: It's an list with the decimal value converted to be send It used to convert the decimal values to hexadecimal values, with the right format to send them.
Modbus_dataframe_update_read_registers	**Parameters:** printb (boolean): If it's set to true will show the command send, by default set to False. **Returns:** Nothing Sends request to read to all the read registers and update the Modbus_df_reg data column
Modbus_dataframe_update_read_registers	**Parameters:** printb (boolean): If it's set to true will show the command send, by default set to False. **Returns:** Nothing Sends request to read to all the read registers and update the Modbus_df_reg data column
Modbus_get_joint_angles	**Parameters:** degrees (boolean,optional): If it's set to True will return the angles in degrees otherwise in radians, by default set to True. **Returns:** joint_angle_value (list of 6 doubles) Read the joints mrad angle and revolutions register, in order to compute the joints angles

Modbus_get_tcp_pos	**Parameters:** rotvec (boolean): If it's set to True will show the TCP pose it's returned with the orientation in rotational vector, otherwise the orientation it's returnen in Roll Pitch degrees (boolean,optional): If it's set to True will return the angles in degrees otherwise in radians, by default set to True. **Returns:** Nothing Read the TCP position registers and computes the TCP position.
Modbus_rpy2rv	**Parameters:** roll(double): roll value to convert pitch(double): pitch value to convert yaw(double): yaw value to convert degrees (boolean,optional): If it's set to True will return the angles in degrees otherwise in radians, by default set to True. **Returns:** rotational_vector (list of 3 doubles) Converts the roll, pitch and yaw value to rotational vector
Modbus_rv2rpy	**Parameters:** rx(double): rx value of the rotational vector to convert ry(double): ry value of the rotational vector to convert rz(double): rz value of the rotational vector to convert degrees (boolean,optional): If it's set to True will return the angles in degrees, otherwise in radians, by default set to True. **Returns:** Nothing Converts the rotational vector to roll, pitch and yaw values

Bibliografy

[1] ROS, *UR ROS Package*,
Consulted (21-05-2020),
http://wiki.ros.org/action/show/universal_robots?action=show&redirect=universal_robot

[2] Universal Robots, *The URScript Programming Language (pag 95-104)*,
Consulted (14-04-2020),
https://s3-eu-west-1.amazonaws.com/ur-support-site/69544/scriptManual.pdf

[3] Universal Robots, *DASHBOARD SERVER CB-SERIES, PORT 29999*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/dashboard-server-cb-series-port-29999/

[4] Universal Robots, *DASHBOARD SERVER E-SERIES, PORT 29999*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/dashboard-server-e-series-port-29999/

[5] Universal Robots, *Remote Control Via TCP-IP*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/remote-control-via-tcpip/

[6] Universal Robots, *REAL-TIME DATA EXCHANGE (RTDE) GUIDE*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/real-time-data-exchange-rtde-guide/

[7] Universal Robots, *XML-RPC COMMUNICATION*,
Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/xml-rpc-communication/

[8] Universal Robots, *MODBUS SERVER*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/modbus-server/

[9] Universal Robots, *ETHERNET IP GUIDE*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/ethernet-ip-guide/

[10] Universal Robots, *PROFINET GUIDE*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/profinet-guide/

[11] RoboDK, *Transfer a Program (FTP)*,
Consulted (14-04-2020),

https://robodk.com/doc/en/Robots-Universal-Robots.html#UR-FTP

[12] Robotiq, *Is ftp server always started*,
Publicació(12/04/2018), Consulted (14-04-2020),
https://dof.robotiq.com/discussion/1097/is-ftp-server-on-the-robot-always-started

[13] Robotiq, *Remote Control of Universal Robots Pag2*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://dof.robotiq.com/discussion/553/remote-control-of-universal-robots-user-interfa
p2

[14] Walter Goralski, *The Illustrated Network (Second Edition), 2017*,
Consulted (21-05-2020),
https://www.sciencedirect.com/science/article/pii/B9780128110270000011

[15] Doug Abbott, *Linux for Embedded and Real-Time Applications, 2003*,
Consulted (21-05-2020)

[16] Universal Robots, *Download website UR*,
Consulted (16-04-2020),
https://www.universal-robots.com/download/?option=69543#section69281

[17] Universal Robots, *ROBOT DATA SOURCE SUMMARY*,
Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/
data-source-matrix-for-remote-access/

[18] Acromag, *Introduction to Modbus TCP*,
Consulted (21-05-2020)
https://acromag.com/wp-content/uploads/2019/08/White-Paper-Introduction-to-ModbusTCP_
765B.pdf

[19] RTAutomation, *Overview of EtherntIP*,
Consulted (21-05-2020)
https://www.rtautomation.com/technologies/ethernetip/

[20] Profinet, *Profinet-WhitePaper*,
Consulted (21-05-2020)
https://us.profinet.com/wp-content/uploads/2014/04/
PROFINET-for-Network-Geeks2019.pdf

[21] Profinet University, *Profinet Blog*,
Consulted (21-05-2020)
https://profinetuniversity.com/profinet-basics/profinet-communication-channels/

[22] RTAutomation, *Overview of Profinet*,
Consulted (21-05-2020)
https://www.rtautomation.com/technologies/profinet-io/

[23] Edward Insam, *IP EMBEDDED INTERNET APPLICATIONS,chapter 8, FTP*,

Consulted (21-05-2020)
https://epdf.pub/ip-embedded-internet-applications.html

[24] Daniel J. Barrett and Richard E. Silverman, *SSH, The Secure Shell: The Definitive Guide*, Consulted (21-05-2020)
https://docstore.mik.ua/orelly/networking_2ndEd/ssh/index.htm

[25] BOE, *XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos.*, Consulted (11-06-2020)
https://www.boe.es/buscar/doc.php?id=BOE-A-2019-14977

[26] RTAutomation, *Overview of Modbus TCP*, Consulted (21-05-2020)
https://www.rtautomation.com/technologies/modbus-tcpip/

[27] Acromag, *Introduction to Ethernet/IP*, Consulted (21-05-2020)
https://acromag.com/wp-content/uploads/2019/06/Acromag_Intro_EthernetIP_747C.pdf

[28] Universal Robots, *OVERVIEW OF CLIENT INTERFACES*, Publicació(12/04/2020), Consulted (14-04-2020),
https://www.universal-robots.com/articles/ur-articles/overview-of-client-interfaces/

[29] Universal Robots, *Academy website UR*, Consulted (08-04-2020),
https://www.universal-robots.com/es/academy/

[30] Rabbit, *Socket Process*, Consulted (21-05-2020),
https://www.digi.com/resources/documentation/digidocs/PDFs/0190074_j.pdf

[31] ScienceDirect, *StackProtocols*, Consulted (21-05-2020),
https://www.sciencedirect.com/topics/computer-science/protocol-stack

[32] RTAutomation, *Overview of Modbus*, Consulted (21-05-2020) https://www.rtautomation.com/techno\logies/modbus/

[33] TutorialsPoint, *XML-RPC Tutorial*, Consulted (21-05-2020)
https://www.tutorialspoint.com/xml-rpc/xml_rpc_quick_guide.htm

[34] SSH, *Commands of SFTP*, Consulted (21-05-2020)
https://www.ssh.com/ssh/sftp/#sftp-protocol