POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

VOCAL CONTROL OVER AN INDUSTRIAL CARTESIAN ROBOT

Supervisor: Prof. Alessandro RIZZO

Candidate: Riccardo TANONI

Corporate Tutor: Ing. Tommaso ANTINORI

2021-2022

ABSTRACT

Nowadays, vocal control features are widely diffused in our everyday life.

The aim of this thesis is to implement a simple vocal control interface to pilot an industrial top-entry cartesian Robot throughout its simplest actions.

This specific treatise is divided in the following main parts:

1. Introduction, problem declaration and proposed solution
2. Brief notes on industrial Robots' typologies, with particular emphasis on cartesian ones (used for this project)
3. Modifications made on the Robot' main code and relative working test on real Robot simulator
4. Modbus protocol migration on Microcontroller (Arduino board) and vocal control adaptation for the chosen hardware
5. Real circuit building and final code assembling
6. Final working tests on both software and hardware
7. Conclusions and potential future applications

The final results showed that, with the right customization, it is possible to let an industrial Robot perform some basic actions through simple vocal commands: precise range movements, entering in specific modes, Stop and Reset actions performing.

Even if the subject topic was treated in an academic way, the possible future applications are many and the possibility to connect such machines to A.I. vocal assistants is very concrete.

INDEX

ACKNOWLEDGMENTS

Heartfelt thanks to:

- *“**Campetella Robotic Center S.R.L.**” for the opportunity to have seen and lived such a complex industrial environment, as well as all the material and instruments granted me to carry out this project*

- *My corporate tutor: **Tommaso Antinori**, for the big help and the precious cooperation*

- *My thesis supervisor: Prof. **Alessandro Rizzo**, for having accepted and helped me*

Chapter 1

INTRODUCTION

1.1 SMART INDUSTRY 4.0

The past few decades were characterized by the improvement of the so called "Smart Industry" (or Industry 4.0) that brought rapid changes to technological sectors and industrial environments, due to the large amount of inter-connectivity and smart features that characterized the 21st century. The most touched aspects were surely A.I. (artificial intelligence) and Robotics, through which A.I. can interact with the real world. The automation of traditional manufacture has undergone many changes, with the introduction of smart-technology, Machine-to-Machine (M2M) connections and Internet-of-Things (IoT), but one of the most diffused aspects was the implementation of vocal control (vocal assistants A.I.) over almost every feature of modern technology. The "Fourth Industrial Revolution" favored the development of the Smart Factory based on technical aspects, such as cyber-physical systems communicating within each-other using the IoT and many other services. [1]

With the increase of these modern technologies, the need of developing more up-to-date solutions accordingly grew and affected every aspect of our everyday life: correlated studying fields were integrated to create new solutions, with their utmost expression in A.I. and Robotics. In particular, modern Robots played a crucial role regarding our technological development, mostly for their ability to perform "unhuman" tasks in terms of safety, fast response and accuracy. [2]

For this reason, many modern industries' employees had to adapt their skills, not anymore to manufacture, but to tasks such as Robots programming, management and adaptation to the production process: from one side, the information flow is carried out within machines; from the other, such information support company management systems, in increasing the efficiency of production. It is therefore important to develop a flexible arrangement for the production system that will adapt to customer needs in continuous changing with the chance of the system future use anticipation. In a close future, specific systems which are already been used for years in industrial environments may lead to an industrial Robot programming developing, to simplify the cooperation within workstation operators and machines, with the possibility of creating flexible programs that allow the interaction between operator and Robots, using natural solutions like movements or voice: these tools are already used in different fields, not just like industrial applications, but also in transportations and home automation. [3]

1.2 PROBLEM DECLARATION AND PROPOSED SOLUTION

This thesis has the goal to investigate the developing, implementing and finally controlling of an industrial Top-Entry Cartesian Robot using an Android Vocal App and an Arduino microcontroller: very cheap and with a simple open-source programmability. Taking in consideration the Cartesian Robot characteristics, for an industrial application it is important to respect:

- Safety

  In an industrial environment, workers' safety is the most important aspect. In general, modern industrial Robots must respect very restrictive safety rules, however the latter are at-times not enough to prevent a possible dangerous situation. Vocal commands can therefore add a higher level of safety, avoiding human personnel to physically interact with those systems, using a more immediate method like the voice.

- Fast Response

  Vocal control in industries environments needs a very fast response. For this implementation, the Google Assistant already included in Android OS devices is chosen and the time required from the App to acquire the pronounced word, interpretate it, convert it and send it to the Robot is unfortunately not instantaneous.

- Accuracy

  To substitute the manual work performed by a man, high accuracy is required: cartesian Robots are the most accurate Robots in industrial environments and the implemented solution will maintain this feature, acting directly on brushless motors (and therefore on drivers) that can perform very precise millimetric movements. To make this configuration adaptable to different industrial settlements, a common open-source microcontroller was selected to connect the voice-interpreting device to the main Robot Control Unit.

Tests results and Robot behaviors will be discussed at the end of this treatise.

To implement a proper hardware solution, the Open-Source Arduino Mega Board were selected, because it is very easy to find and widely diffused in every ambit.
The use of an Ethernet Shield and of a Bluetooth Transmitter/Receiver was necessary to connect the control board to the Robot Control Unit and to ensure the correct data exchange between Client (microcontroller) and Server (Robot).

For this specific solution, the Robot can be controlled through the following commands:

- ***Movements.*** The selected basic movement actions were chosen as follow:

  - "***Upwards***" and "***Downwards***" = movements along y-axis
  - "***Leftwards***" and "***Rightwards***" = movements along z-axis
  - "***Frontwards***" and "***Backwards***" = movements along x-axis

  these are the six possible moving actions that the Robot (via brushless motors) can perform along x, y, z coordinates.

- ***Modes***. The selected Modes, in which the Robot can enter, were chosen as follow:

  - "***STOP***" = the Robot performs a Stop action, ending any movements or previous commands
  - "***JOG***" = the Robot enters in Jog Mode, free to move in every direction

- "***TEACH***" = the Robot enters in Teach Mode, ready to memorize informations about the new aim-point to reach
- "***STEP***" = the Robot enters in Step Mode, performing one-by-one movements
- "***AUTO***" = the Robot enters in Automatic Mode, performing the entire cycle by itself

The chosen circuit configuration is the sequent:

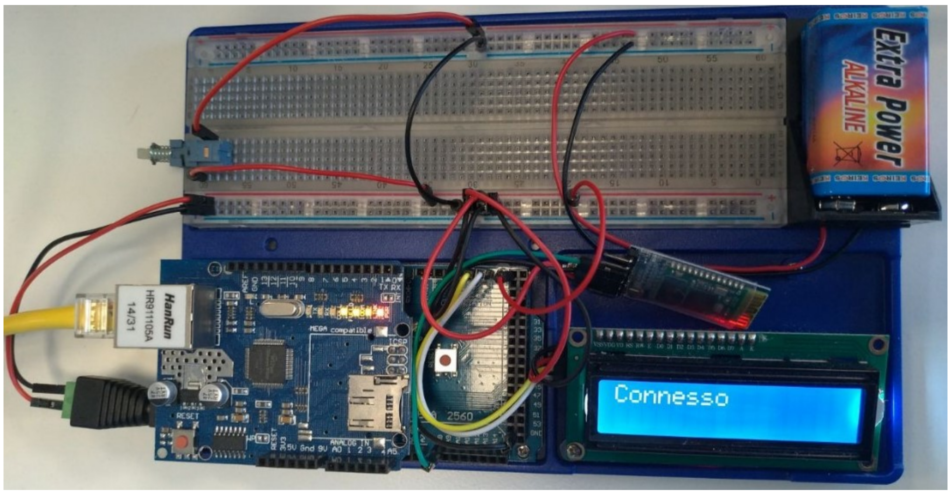


Figure 1.1: Final Circuit Configuration

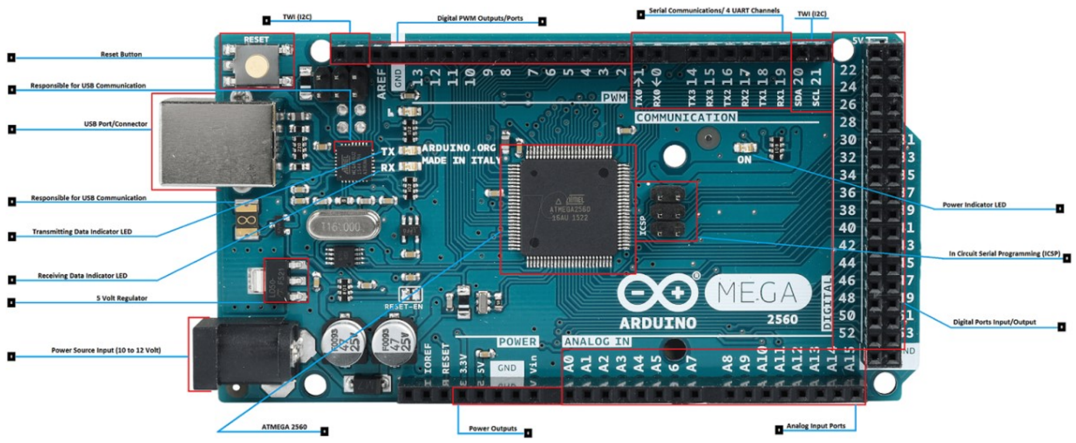

TWI (I2C)
Digital PWM Outputs/Ports
Serial Communications/ 4 UART Channels
TWI (I2C)
Reset Button
Responsible for USB Communication
USB Port/Connector
Responsible for USB Communication
Transmitting Data Indicator LED
Receiving Data Indicator LED
5 Volt Regulator
Power Source Input (10 to 12 Volt)
ATMEGA 2560
Power Outputs
Analog Input Ports
Power Indicator LED
In Circuit Serial Programming (ICSP)
Digital Ports Input/Output

Figure 1.2: Arduino Mega Schematic

Chapter 2

INDUSTRIAL ROBOTS

2.1 TYPOLOGIES OF INDUSTRIAL ROBOTS

Nowadays Robots are widely diffused in almost every technological field: the most common are industries, space investigation, military applications and medical assistance. They are used for different tasks such as: welding, painting, assembling/disassembling, pick-and-place (PCB), products inspection, and different testing. [4] In this thesis, only Industrial Robots will be treated, with particular emphasis on the typology used to develop the project: Cartesian Robots. Industrial Robots are developed, assembled and controlled by algorithms and programs installed in computers or controlling devices: they are classified depending on artificial intelligence complexity, structural characteristics, applications and operating skills. The rigid structures able to move are called links and the joints can be divided in two groups: revolute joints (or hinges) and prismatic joints (or sliding joints). [5]

It is possible to divide industrial Robots in six main typologies:

- **Cartesian Coordinate Robots**: better treated in the next paragraph.

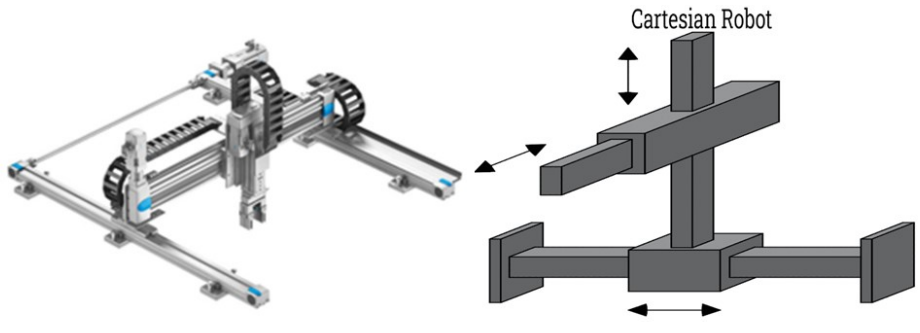


Figure 2.1: Cartesian Robot

- **Articulated (or Anthropomorphic) Robots**: they consist of three fixed axes linked to a revolving base. Configuring links and joints in specific ways, almost every point of the workspace can be reached: considering distances and angles, it is possible to characterize each of these reachable points.

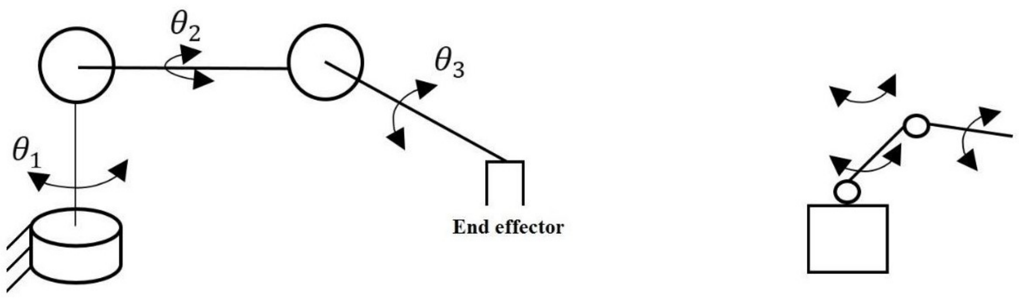


Figure 2.2: Articulated Robot Scheme

The most diffused example of articulated Robot is surly the Robotic Arm (Anthropomorphic) configuration: having more joints (normally six), its workspace dimension is wider but, due to different payloads and non-linear behaviors, the operating speed is quite limited. [6]

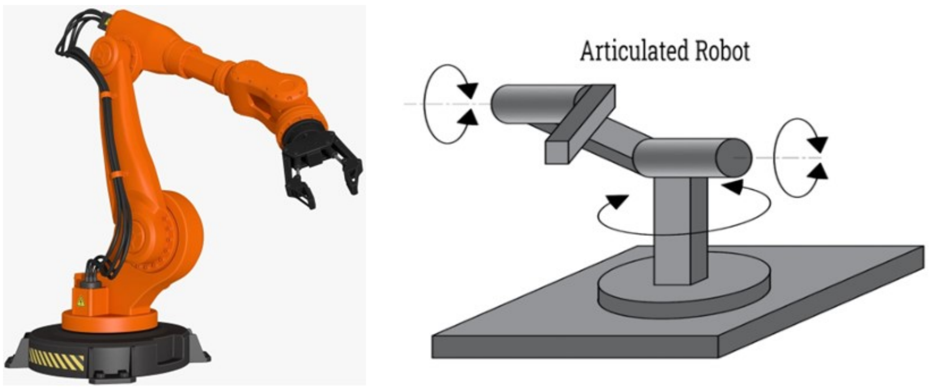


Figure 2.3: Articulated Robot

- **Cylindrical Coordinate Robots**: they consist of two prismatic joints and a rotatory one, forming a cylindrical workspace. A horizontal arm is mounted on a vertically moving cartage and both are connected to a revolving base: so, each reachable point can be represented by a cylindrical coordinates system. [7]

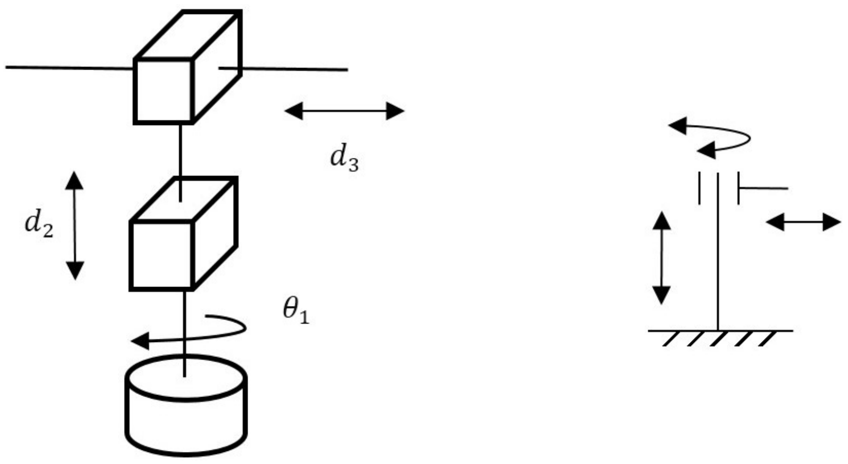


Figure 2.4: Cylindrical Robot Scheme

In this configuration, the Robot is able to move frontward and backwards along z axis, upwards and downwards along y axis and to rotate according to θ angle. [8]

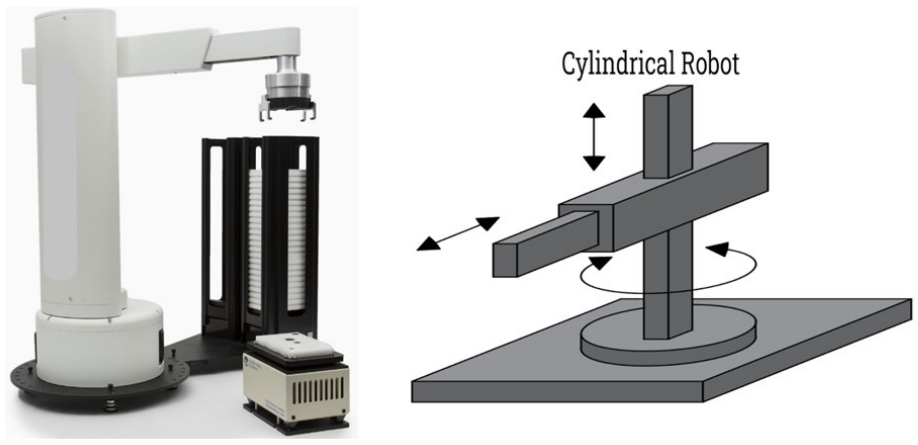


Figure 2.5: Cylindrical Robot

- **Spherical (or Polar) Coordinate Robots**: they consist of two moving joints (at least) and a fixed one: their size is very big and they present a telescopic arm, to perform movements such as base rotation, arm angular inclination and end-effector slide. [9]

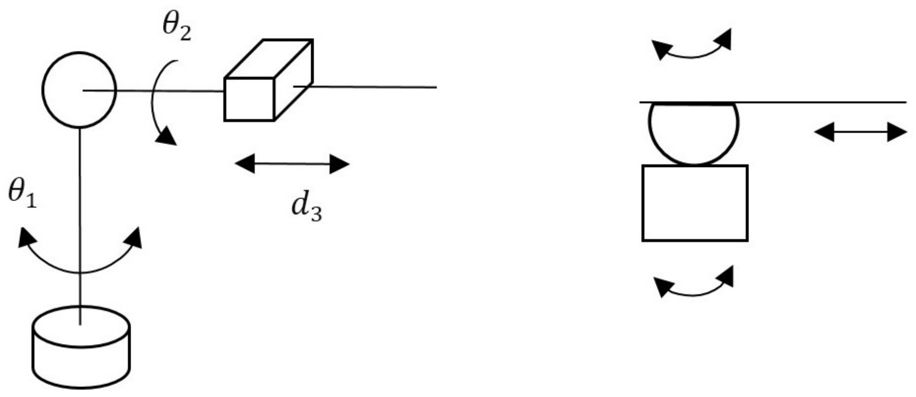


Figure 2.6: Polar Robot Scheme

They are commonly used for welding, die-casting, plastic injection and extrusion due to their specific orientation in the spherical sector workspace, including two concentric hemispheres: an inner one, reachable when the arm is completely retracted and an outer one, reachable when the arm is completely extended. [10]

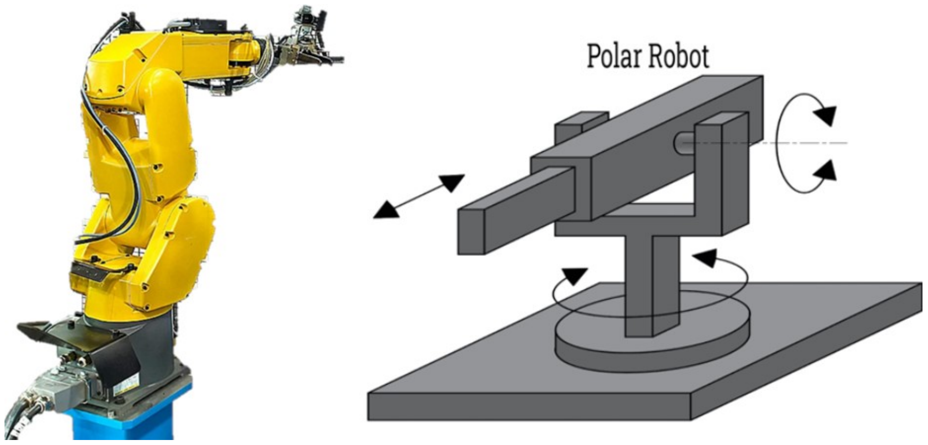


Figure 2.7: Polar Robot

- **SCARA Robots**: acronym of “Selective Compliance Assembly Robot Arm”, they consist of two parallel rotatory joints and a prismatic one: the first ones performing movements along the horizontal plane and the second one along the vertical plane. [7]

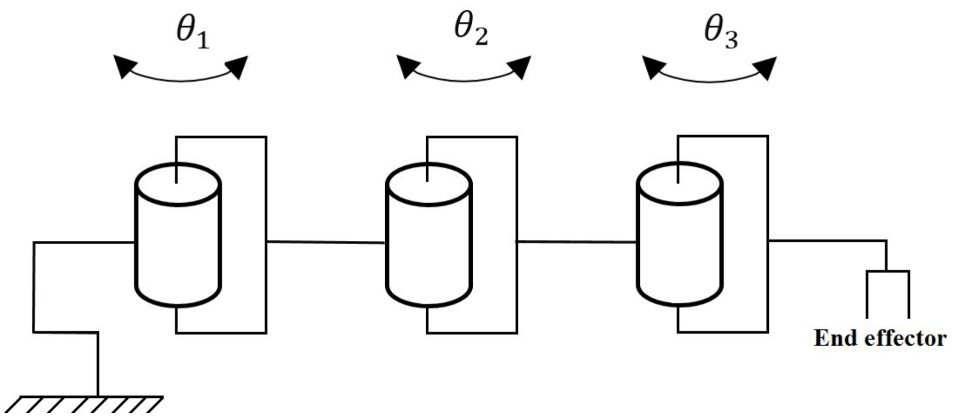


Figure 2.8: SCARA Robot Scheme

They are widely used for high-speed applications, with a cycle characterized by repetitive movements (point-to-point) such as assembling operations: a peculiar characteristic is that they are weaker about x and y axes and stronger about the z axis. [11]

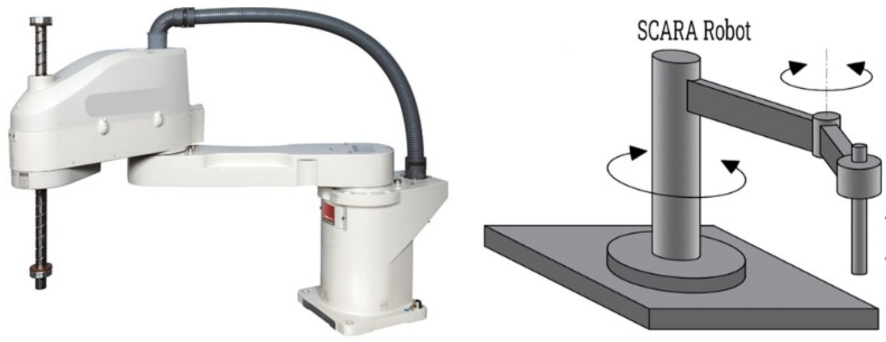


Figure 2.9: SCARA Robot

- **Delta (Parallel Link) Robots**: they consist of a fixed basement connected to its parallelogram linkage systems (4 bars): the end-effector must be composed by at least two chains, each one having a minimum of one actuator (one per each end-effector degree of freedom). [12]

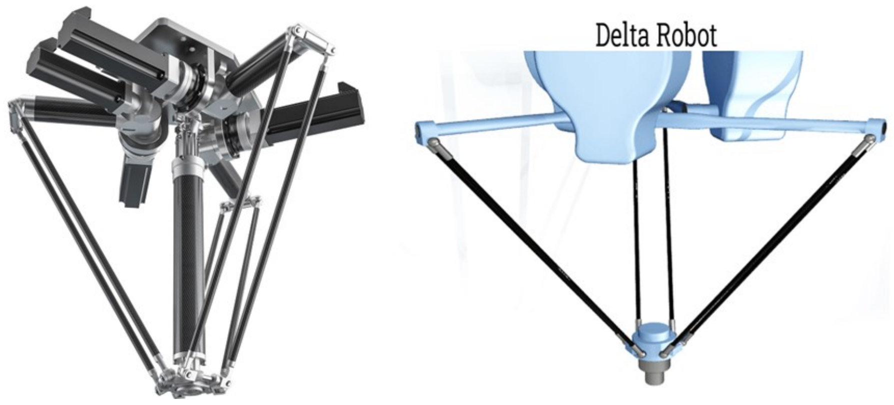


Figure 2.10: Delta Robot

Due to their highest configurations, such industrial Robots can move objects up to 6 degrees of freedom (DoF), determined by 3 translational (3T) and 3 rotational (3R) coordinates for a complete 3T-3R mobility. Anyway, in many cases, manipulation tasks require less than 6 degrees of freedom: it is so possible to use lower mobility manipulators that may bring advantages in terms of architecture, control, motions and costs. [4]

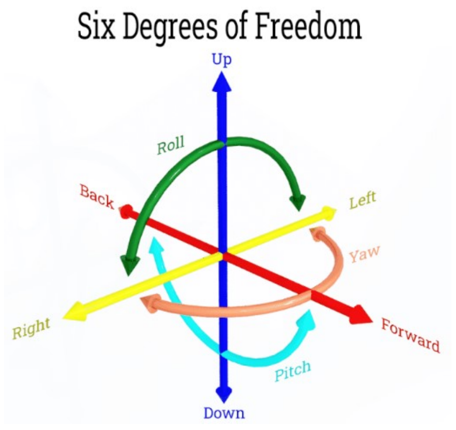


Figure 2.11: Six Degrees of Freedom 3-D Schematic

2.2 CARTESIAN ROBOTS

- **What Cartesian Robots are?**

Cartesian coordinate geometry is an optimal method to map 3-Dimensional space in a simple, understandable numerical system composed by three coordinate axes (perpendicular to each other) that cross in the so-called "origin": any point in the 3-Dimensional space is characterized by three numbers (x, y, z).

Mechatronic Robots that use linear axes to move along are called Cartesian Robots, Linear Robots, or Gantry Robots: they have an overall part that controls the motion along the horizontal plane and a Robotic Arm that performs vertical motions. They can move along x-y plane or x-y-z plane. The Robotic Arm presents an end-effector (generally a mechanical tool) attached on its end, depending on the function they are used for. [13]

- **Characteristics and Functionalities**

Cartesian Robots can only move through linear motions, controlled by servomotor drives. The drive system can be:

- Belt driven
- Cable driven
- Screw driven
- Pneumatic driven
- rack-and-pinion driven
- linear-motor driven

The very big advantage of Cartesian Robots is that they are very easy to program, having only linear motions to be controlled: due to that, complex arrays of PLCs and microchips are not necessary.

Compared to other typologies, Cartesian Robots have a higher payload carrying capacity that, combined with lower costs and simple programming, makes them suitable for a large variety of industrial applications. The linear Robots' movements range can also be extended by adding compatible modules, making them longer in an industrial life and much more versatile.

Having no need to arrange rotary motion, they are also characterized by a high level of accuracy and precision (with respect of other typologies): their tolerances are in the range of micrometers (µm), while others normally are in the range of millimeters (mm). [13]

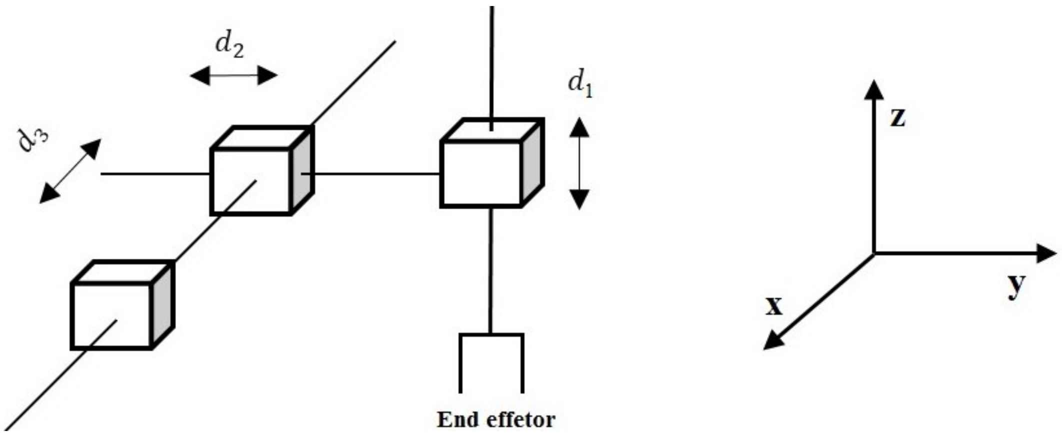


Figure 2.12: Cartesian Robot Scheme

The linear movements performed by the Cartesian Robot can be expressed by the following expression:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix}$$

Where $q \epsilon \mathbb{R} n \times 1$ is the joints vector. The movements performable by the end-effector to reach a specific point can be expressed by the following expression:

$$q_d = \begin{bmatrix} q_{d_1} \\ q_{d_2} \\ \vdots \\ q_{d_n} \end{bmatrix}$$

Where q_d is the joints vector of this reachable point and $\tilde{q}_{\mathrm{i}} = q_{d\mathrm{i}} - q_{\mathrm{I}}$ is the end-effector final positions vector. [14]

- **Most Common Applications**

Cartesian Robots are suitable for many applications in the industrial sector.

The most diffused are:

  - ***PICK AND PLACE:*** The Robotic arm can identify different components through a vision device and pick these objects, then sorting them into different bins.

  - ***PROCESS-TO-PROCESS TRANSFER:*** Using Dual-Drive linear Robots it is possible to transfer goods from a place to another: for that, time-synchronization systems can be used, depending on the process.

  - ***ASSEMBLING SYSTEM:*** Assembling a product's parts, when same steps must be repeated many times, Cartesian Robots can be widely used to automate such tasks.

  - ***PRECISION SPOT WELDING:*** In specific manufacturing processes a precise welding is often required: with their welding arms modules Cartesian Robots can accurately achieve it among microscopic (μm) locations on the work plane.

  - ***PALLETAZING AND DEPALLETAZING:*** Cartesian Robots are also useful to automatically both placing and taking products on and from pallets to easily transport goods throughout all the process chain.

  - ***CNC MACHINE TOOLING:*** Computerized numerical control-based machines are widely used to make specific products, depending on the designs made through engineering software. [13]

2.3 “CAMPETELLA” RHEA PRIME ROBOT

For the development and testing of the project, we used the model “RHEA PRIME” from “Campetella Robotic Center S.R.L”, that is equipped with three servo-guided electric axes:

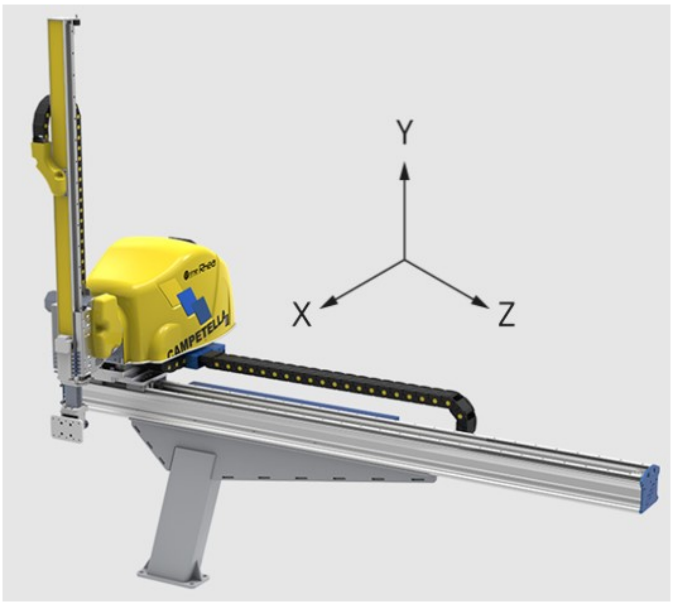


Figure 2.13: “Campetella” Rhea Prime Robot

It is characterized by:

- Configurable vertical axis of the A-type
- Z axis stroke from 1200 to 1500 mm
- X axis stroke about 300 mm
- Y axis stroke about 800 mm
- Payload that the end effector can effort up to 2 kg
- Dry Cycle of 5”

Table 1: Rhea Prime Robot Specific

	RHEA - 1 A	RHEA - 2 A
Maximum payload [kg]:	2	2
Vertical axis:	Direct	Direct
Z axis stroke – Horizontal [mm]:	1200	1500
X axis stroke - Extraction [mm]:	300	300
Y axis stroke - Vertical [mm]:	800	800
X,Y,Z axes motion:	AC synchronous brushless servomotors	
X,Y,Z axes guidance system:	Sliding blocks with roll-by-balls on a hardened steel prismatic guide	
Positioning repeatability [mm]:	± 0,1	
Pneumatic C axis rotation [deg]:	0°/ 90°	
Control unit:	Campetella PRIME proprietary system	

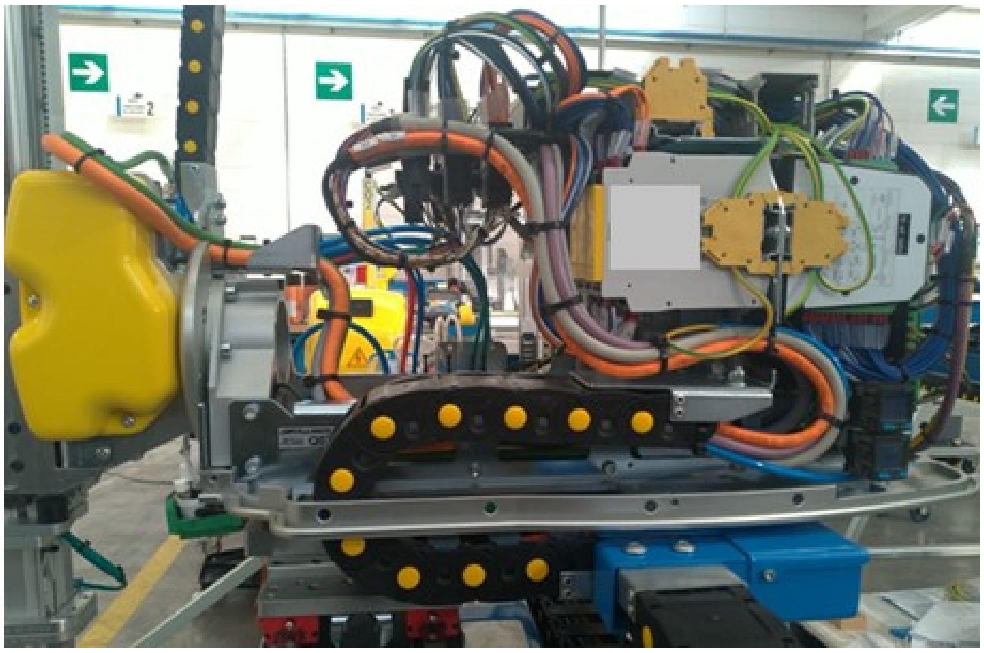


Figure 2.14: Rhea Prime Robot Internal Details

Chapter 3

DEVELOPED PROJECT

3.1 TASK MODIFICATION FOR ROBOT AXIAL CONTROL

The implementation of this project started with considering the chosen industrial Robot native code, written in CodeSys Structured Text (ST) language and property of "Campetella Robotic Center S.R.L." that kindly conceded to publish it into this treatise: it is possible to entirely view it in the 'Appendix'. The first aim is to properly modify it, so that to ensure the Robot to achieve the wanted behaviors.

In particular, the following tasks were chosen:

- Performing movements along the three axes: Rightwards (*z*+), Leftwards (*z*-), Downwards (*y*+), Upwards (*y*-), Frontwards (*x*+), backwards (*x*-)
- Entering JOG Mode
- Entering TEACH Mode
- Entering STEP & AUTO Mode
- Performing STOP & RESET Emergency Commands

According to such choice, the native code was modified in the following way:

```
//******************************************************************************************************
//--------------------------------------- Axial JOG Vocal commands ----------------------------------------
// When vocal command is received:
// 1) selection of the axis to move
// 2) raised up the front "startPIU" or "startMENO"
// 3) calculation of arrival range
//
*******************************************************************************************************
```

```
IF difuCmdVocale THEN // Axis selection in Teach Mode
        IF    codiceComando = CMD_DX THEN
                  AsseInTeach  := 1;
                  startPIU         := TRUE;
                  PIUinCorso    := TRUE;
                  apppoTargetPos:= QuotaReale[1] + deltaQuotaTeachVocale;
        ELSIF  codiceComando = CMD_SX THEN
                  AsseInTeach  := 1;
                  startMENO      := TRUE;
                  MENOinCorso   := TRUE;
                  apppoTargetPos:= QuotaReale[1] - deltaQuotaTeachVocale;
```

Starting with the "*z*" axis, if the received command from the vocal control is a Rightwards Movement (CMD_DX) then the axis associated with 1 (*z*) is selected and the new coordinate to reach in positive sense is the previous one (QuotaReale[1]) + the given one (deltaQuotaTeachVocale).
Otherwise, if the received command from the vocal control is a Leftwards Movement (CMD_SX) then the axis associated with 1 (*z*) is selected and the new coordinate to reach in negative sense is the previous one (QuotaReale[1]) - the given one (deltaQuotaTeachVocale).

```
        ELSIF  codiceComando = CMD_AVANTI THEN
                  AsseInTeach  := 2;
                  startPIU         := TRUE;
                  PIUinCorso    := TRUE;
                  apppoTargetPos:= QuotaReale[2] + deltaQuotaTeachVocale;
        ELSIF  codiceComando = CMD_INDIETRO THEN
                  AsseInTeach  := 2;
                  startMENO      := TRUE;
                  MENOinCorso    := TRUE;
                  apppoTargetPos:= QuotaReale[2] - deltaQuotaTeachVocale;
```

Going on with the "*x*" axis, if the received command from the vocal control is a Frontwards Movement (CMD_AVANTI) then the axis associated with 2 (*x*) is selected and the new coordinate to reach in positive sense is the previous one (QuotaReale[2]) + the given one (deltaQuotaTeachVocale).
Otherwise, if the received command from the vocal control is a Backwards Movement (CMD_INDIETRO) then the axis associated with 2 (*x*) is selected and the new coordinate to reach in negative sense is the previous one (QuotaReale[2]) - the given one (deltaQuotaTeachVocale).

```
ELSIF  codiceComando = CMD_BASSO THEN
                    AsseInTeach   := 3;
                    startPIU          := TRUE;
                    PIUinCorso    := TRUE;
                    apppoTargetPos:= QuotaReale[3] + deltaQuotaTeachVocale;
         ELSIF  codiceComando = CMD_ALTO THEN
                    AsseInTeach   := 3;
                    startMENO      := TRUE;
                    MENOinCorso   := TRUE;
                    apppoTargetPos:= QuotaReale[3] - deltaQuotaTeachVocale;
```

Ending with the *"y"* axis, if the received command from the vocal control is a Downwards Movement (CMD_BASSO) then the axis associated with 3 (*y*) is selected and the new coordinate to reach in positive sense is the previous one (QuotaReale[3]) + the given one (deltaQuotaTeachVocale).
Otherwise, if the received command from the vocal control is an Upwards Movement (CMD_ALTO) then the axis associated with 3 (*y*) is selected and the new coordinate to reach in negative sense is the previous one (QuotaReale[3]) - the given one (deltaQuotaTeachVocale).

```
         ELSIF  codiceComando = CMD_STOP THEN
              PIUinCorso    := FALSE;
        MENOinCorso   := FALSE;
```

If a STOP Command is given (CMD_STOP), then the variables related to positive (PIUinCorso) and negative (MENOinCorso) movements are reset and the Robot stops

```
         ELSIF  codiceComando = CMD_JOG THEN
              PIUinCorso    := FALSE;
        MENOinCorso   := FALSE;
                    JogTeach      := deltaQuotaTeachVocale;
         END_IF
   difuCmdVocale :=0;
END_IF
```

If the JOG Mode Command is given, the Robot enters in JOG Mode and the new coordinate to reach is just the given one (deltaQuotaTeachVocale), independently from the starting point

```
//to simulate button releasing condition since the axis is in target range
                    IF asseInMovimento[AsseInTeach] AND PIUinCorso AND
QuotaReale[AsseInTeach]>= apppoTargetPos-1 THEN
                           PIUinCorso := FALSE;
                    END_IF

                    IF asseInMovimento[AsseInTeach] AND MENOinCorso AND
QuotaReale[AsseInTeach]<= apppoTargetPos+1 THEN
                           MENOinCorso := FALSE;
                    END_IF

                    IF asseInMovimento[AsseInTeach] AND (NOT(PIUinCorso) AND
(NOT(MENOinCorso))) THEN
                           Assiif[AsseInTeach].Input.CmdStop:=TRUE;
```

The referring to a specific axis is given by: 'AsseInTeach=1' for *z*-axis, 'AsseInTeach=2' for *x*-axis and 'AsseInTeach=3' for *y*-axis. If one them is moving (asseInMovimento) in positive sense (PIUinCorso) and the actual range value (QuotaReale) reaches the chosen value (apppoTargetPos-1), then the positive moving stops (PIUinCorso := FALSE) and the same is happening when one axis is moving in negative sense (MENOinCorso); if both values (PIUinCorso & MENOinCorso) are absent, then the moving according to that specific axis stops (Assiif[AsseInTeach].Input.CmdStop:=TRUE).

3.2 WORKING TEST ON REAL ROBOT SIMULATOR

After having properly modified the Robot internal code to achieve the required tasks, it was necessary to test the efficiency of the new code during every single behavior to be sure that no problems or errors were performed. So, to do that, a simulator of the real Robot was assembled and connected to the Robot Opus A3 CPU (where the code was uploaded). This Control Unit has the characteristic to be unpluggable and re-pluggable in every Robot that supports it: the Robot itself, in fact, is an empty machine (hardware) unable to move without the connection of the CPU, containing the main code (software) to be executed.

The assembled simulator is composed by:

- 1x Opus A3 Central Control Unit (Robot Prime)
- 1x Power Supply 95 W 220 V (IN) – 24 V (OUT)
- 3x Driver Micro Eco EWDU (Selema)
- 3x COM RIO Digital I/O EWDU (Esa Automation)
- 3x Brushless Motors (24 V)

A schematic picture of the simulator is shown ahead:

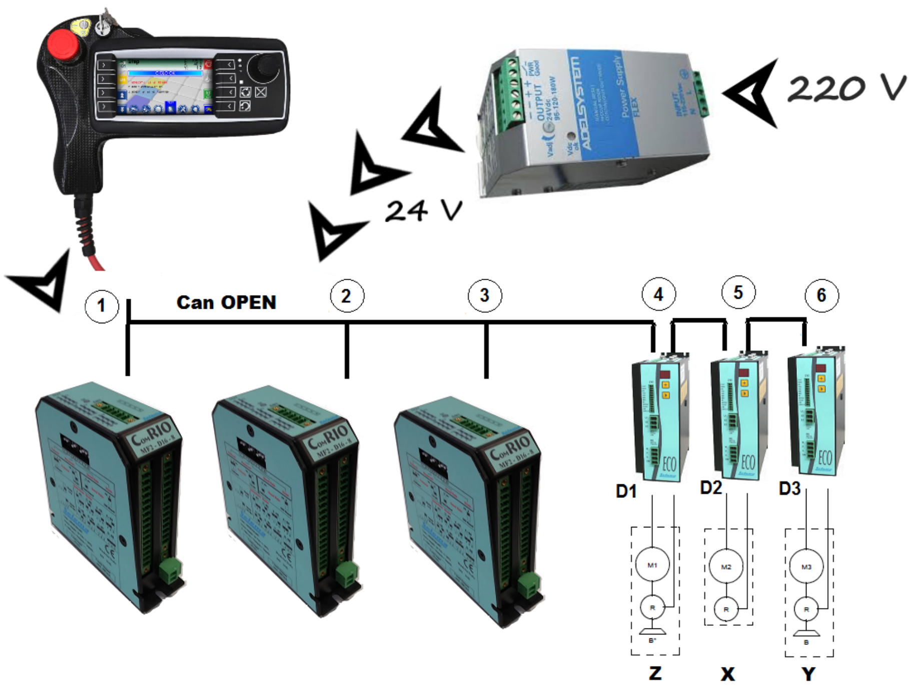


Figure 3.1: Robot Simulator Scheme

To build it up, we started from choosing the three brushless motors used in the real Robot configuration that were useful to emulate the real ones connected to the three Robotic Axes (*x*, *y*, *z*). Such motors require three specific Drivers (one each) to be piloted and they are connected through three control signal wires (u, v, w) that, with combinations of two of the three signals make the brushless motor windings to energize and perform a progressive angular rotation.

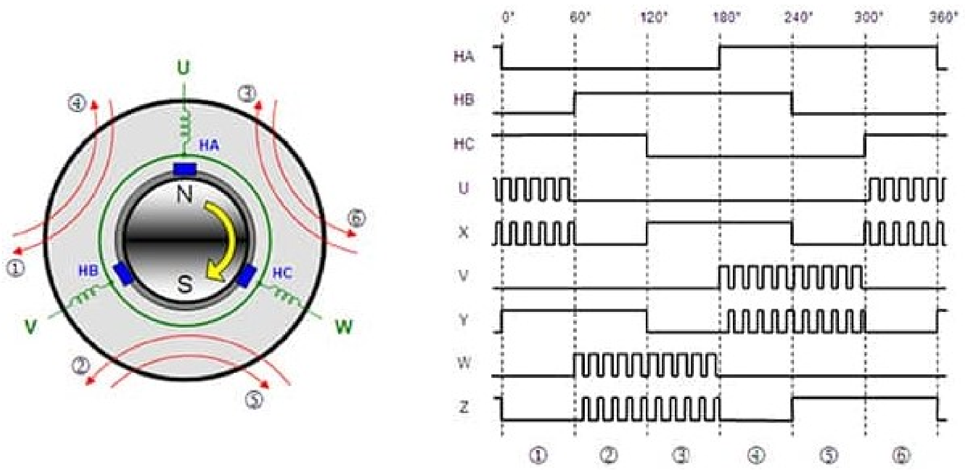


Figure 3.2: Brushless Motor Working Scheme

The Opus A3 Controller, trough the CANOpen field bus, is commanding the Drivers making the brushless motors to move and, at the same time, is reading the physical inputs coming to the COM RIOs from all the sensors placed in the Robot.
Furthermore, it is activating the transistor outputs, relays and electro-valves that compose the Robot electropneumatic circuit.
The CANOpen bus requires a line impedance of 60 Ohm, present within the CAN-H cable and the CAN-L cable: inside the simulator this impedance is obtained through a 120 Ohm resistor, applied upstream of the first COM RIO and combining with the one present inside the Controller.
This is a necessary configuration due to the impossibility of putting a termination resistor into the last driver (final part of the bus), used as an access point for the programming and monitoring cable of the individual motors' drivers.
During the simulations, COM RIOs are not used because they should acquiring informations from the Robot sensors, not present in the simulator; moreover, brushless motors require resetting signals (coming back to position zero) that are simulating during these tests.
The Ethernet cable is connected to the Control Unit USB port (via USB-to-Ethernet converter), used to connect the Robot to the factory network in a Smart Industrial optics.
To work properly, all components are supplied by a 95 Watt Power Supply, that converts the 220 V of Input into the 24 V of output required by Drivers and COM RIOs.

A picture of the real Robot simulator is shown below:

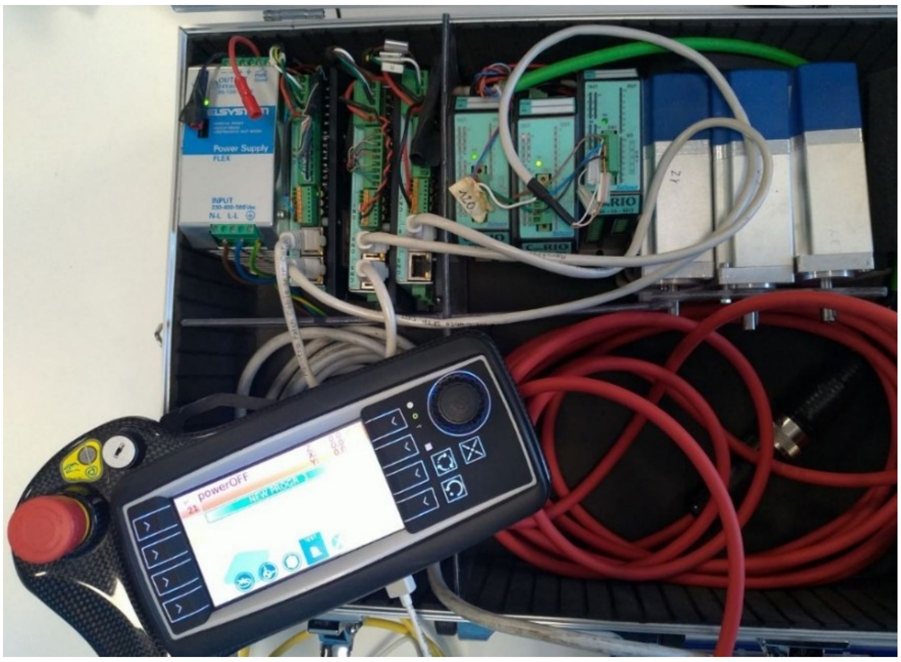


Figure 3.3: Robot Simulator

After having completed the assembling process, the modified CodeSys code was uploaded; due to the Modbus protocol communication, it was essential to declare the Modbus holding registers required to store the vocal commands data: by manually writing them, it was possible to test the chosen tasks.

```
IF   HoldingRegister[R_CH_CMD] > 0 THEN   // vocal input
          codiceComando := WORD_TO_INT(HoldingRegister[R_CH_CMD]);
          deltaQuotaTeachVocale := WORD_TO_REAL(HoldingRegister[R_QUOTA]);

// rising up axial JOG command front and put the register value to zero
          HoldingRegister[R_CH_CMD]:=0;
          difuCmdVocale := TRUE;
END_IF
R_QUOTA : INT := 27;    // Holding Register acquiring the range value of the reaching point

  deltaQuotaTeachVocale : REAL;
  apppoTargetPos : REAL;

  R_CH_CMD  : INT:= 26;    // Holding Register acquiring the movement direction command
```

As we can see in the previous lines, if a numerical value is received ([R_CH_CMD] > 0) the word is converted into an integer number (WORD_TO_ INT) and stored in the holding register #27 (R_QUOTA : INT := 27); otherwise if a command word is received ([R_CH_CMD] > 0) it is converted into a real number (WORD_TO_REAL) and stored in the holding register #26 (R_CH_CMD : INT:= 26).

3.3 MODBUS PROTOCOL

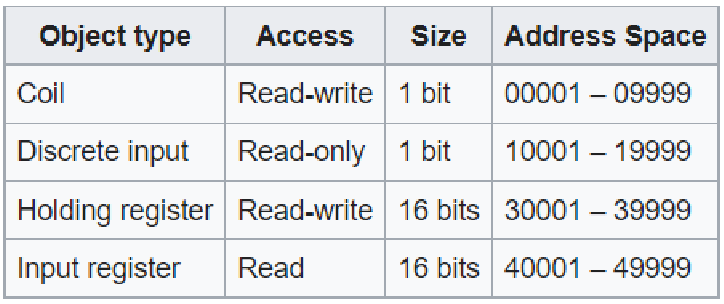
With the name "Modbus" we refer to a Communications Protocol born in 1979 to be used with PLCs. Though the past years, it became a standard communication protocol to connect industrial electronic devices.

Regarding industrial applications, it is very easy to be compared to other protocol standards and it is characterized by many restrictions about transmitted data format.

Besides, Modbus can handle multiple communications within devices connected to the same Ethernet network. [17]

Object Types provided from a Server to a Client (Modbus devices):

Table 2: Modbus Protocol Object Types



Object type	Access	Size	Address Space
Coil	Read-write	1 bit	00001 – 09999
Discrete input	Read-only	1 bit	10001 – 19999
Holding register	Read-write	16 bits	30001 – 39999
Input register	Read	16 bits	40001 – 49999

There are different versions of Modbus protocols that support the Internet Protocol Suite: RTU, ASCII, UDP, RTU-IP, TCP-IP.

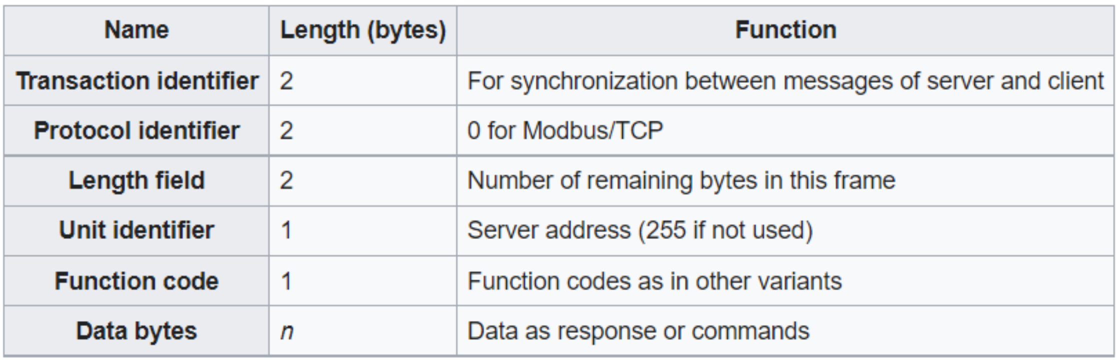
Modbus commands can be used to:

- modify the value of a register, written into Coils and Holding registers
- read an I/O port: data read from an Input or a Coil
- command to receive back values contained in Coils and Holding registers

Those commands contain the Modbus addresses of the device (1 to 247) and checksum information to help the recipient to find transmission errors. [17]

In this project we decided to use the TCP-IP standard (the most used on Ethernet networks) and the frame format is the sequent:

Table 3: Modbus Frame Format



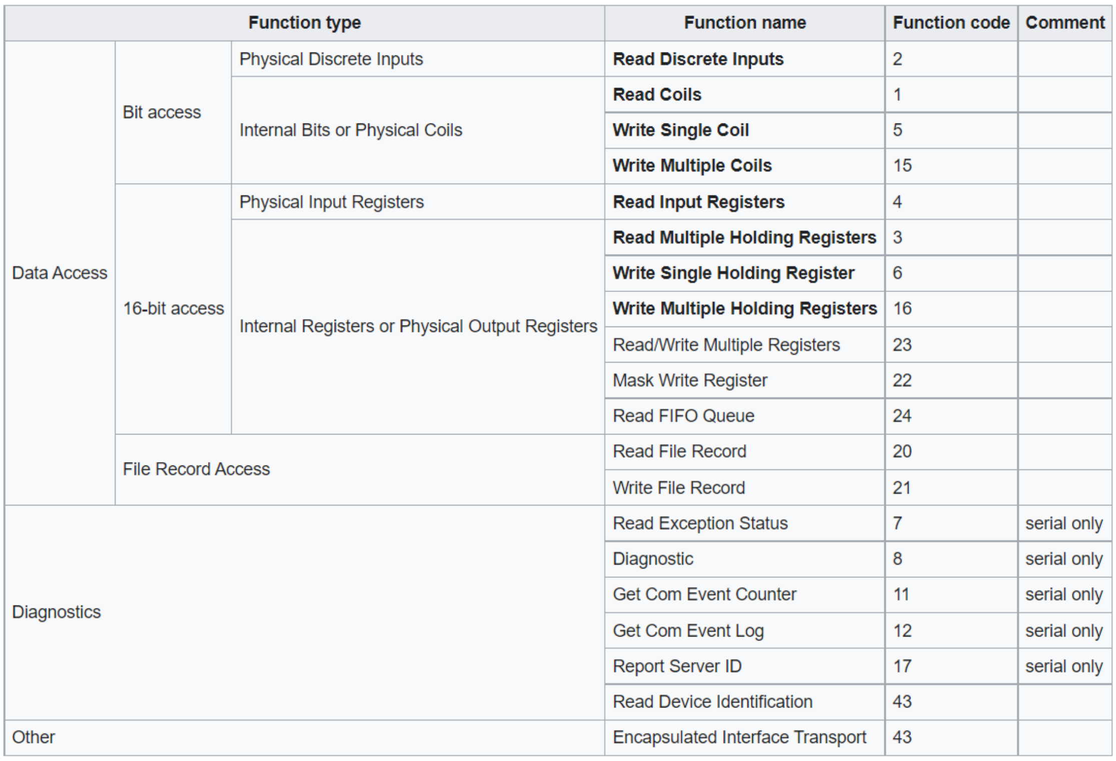
Name	Length (bytes)	Function
Transaction identifier	2	For synchronization between messages of server and client
Protocol identifier	2	0 for Modbus/TCP
Length field	2	Number of remaining bytes in this frame
Unit identifier	1	Server address (255 if not used)
Function code	1	Function codes as in other variants
Data bytes	*n*	Data as response or commands

Functions and commands:

- Coils: readable and writable, 1 bit (on/off)
- Discrete Inputs: read only, 1 bit (on/off)
- Input Registers: read only measurements and status, 16 bits (0 – 65,535)
- Holding Registers: readable and writeable values, 16 bits (0 – 65,535)

[17]

Table 4: Modbus Function Codes



Function type			Function name	Function code	Comment
Data Access	Bit access	Physical Discrete Inputs	**Read Discrete Inputs**	2	
		Internal Bits or Physical Coils	**Read Coils**	1	
			Write Single Coil	5	
			Write Multiple Coils	15	
	16-bit access	Physical Input Registers	**Read Input Registers**	4	
		Internal Registers or Physical Output Registers	**Read Multiple Holding Registers**	3	
			Write Single Holding Register	6	
			Write Multiple Holding Registers	16	
			Read/Write Multiple Registers	23	
			Mask Write Register	22	
			Read FIFO Queue	24	
	File Record Access		Read File Record	20	
			Write File Record	21	
Diagnostics			Read Exception Status	7	serial only
			Diagnostic	8	serial only
			Get Com Event Counter	11	serial only
			Get Com Event Log	12	serial only
			Report Server ID	17	serial only
			Read Device Identification	43	
Other			Encapsulated Interface Transport	43	

3.4 MODBUS PROTOCOL ON MICROCONTROLLER

The next goal to achieve is the software's development to be uploaded into the microcontroller: for that, the widely diffused Arduino Open-Source board was chosen to guarantee the easily finding of the required hardware modules to implement, just like software material to compare to.
The Modbus protocol requires communications via Ethernet Cable and to guarantee such connection, it was mandatory to add an Ethernet Shield board to the Arduino main board: a figure of both components is following

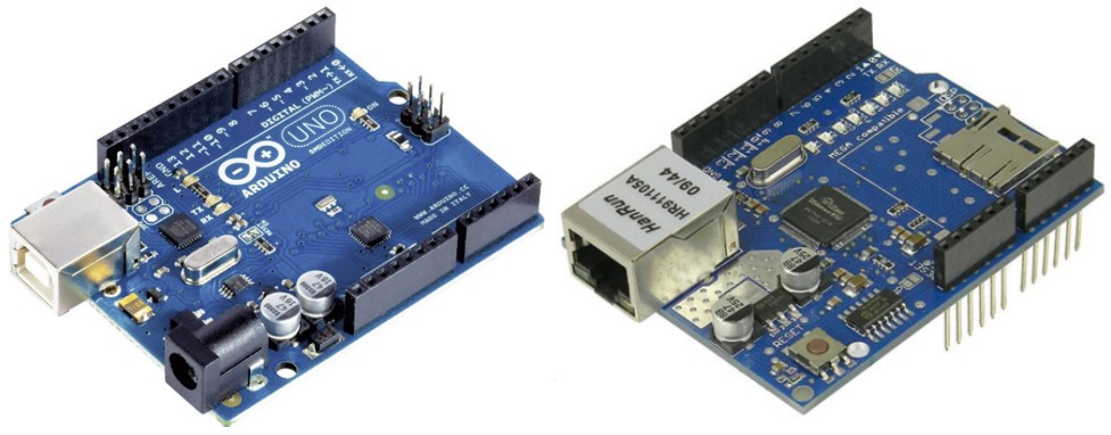


Figure 3.4: Arduino Uno (left) & Ethernet Shield (right)

To check the correct working of both devices, once joined together and connected via USB to the Computer, a basic TCP-IP communication setting was compiled and uploaded into the Microcontroller. This specific code is imposing a MAC address for the Ethernet Shield, then declaring the Client (Shield) and Server (Computer) TCP-IP addresses and finally print them on the Arduino's serial port. Due to some technical issues, the Ethernet Shield boards didn't work correctly and showed a specific error: setting the Client address always to 0.0.0.0

After some researches, a simple DIY solution was found welding two 100 Ohm resistors within pins 3-6 and 1-2 of the Ethernet connector, as shown below:

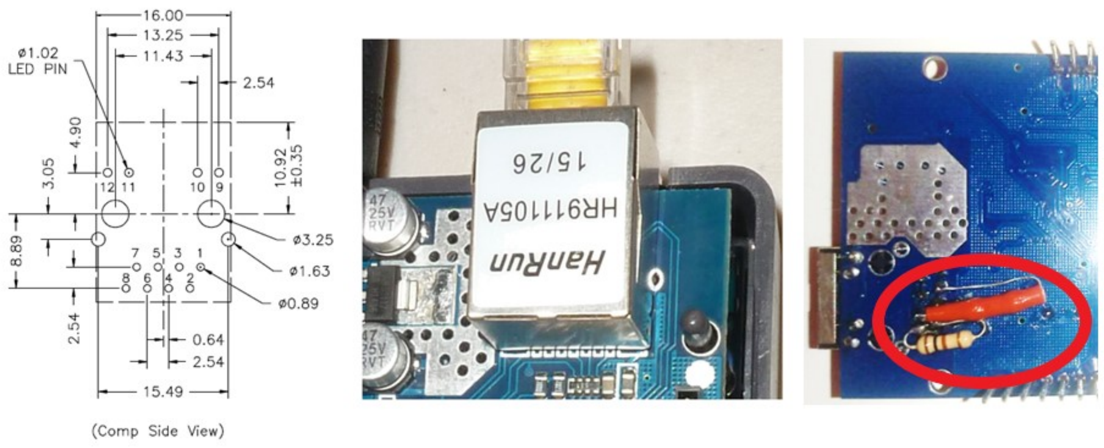
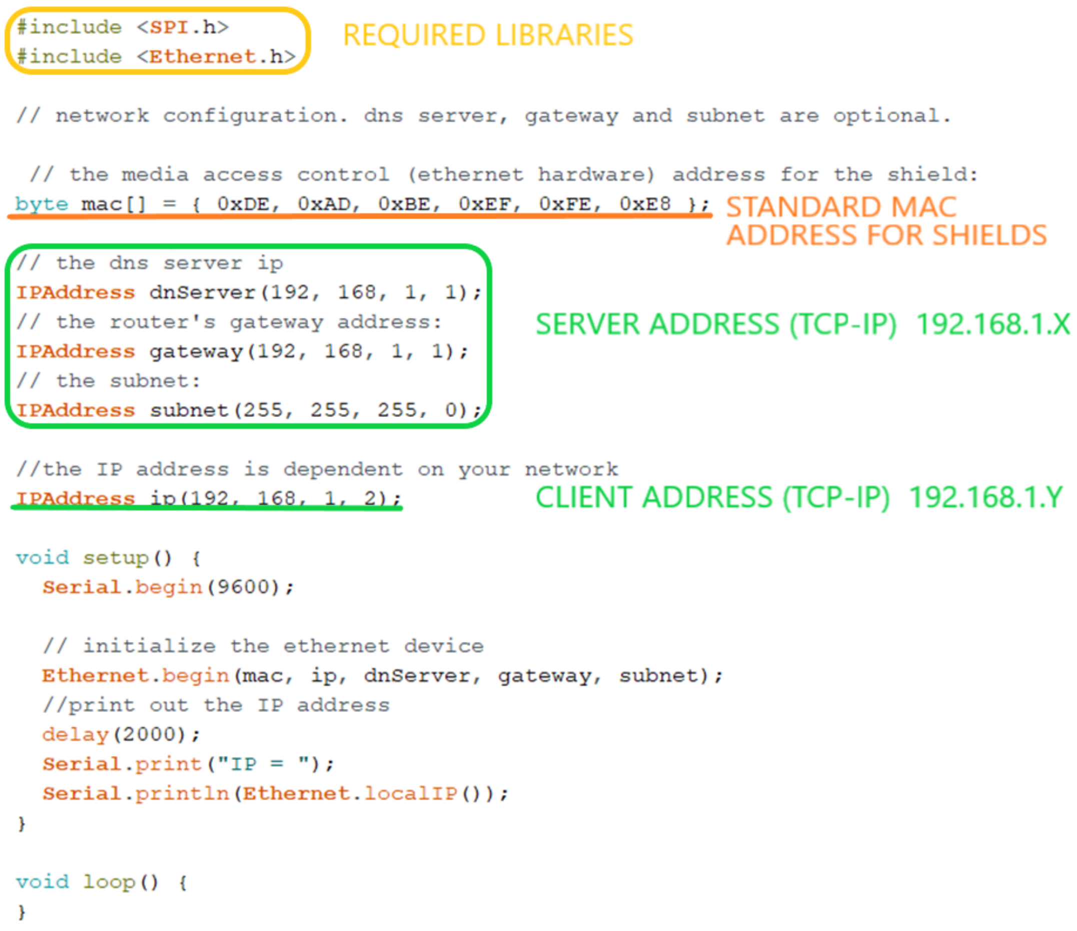


Figure 3.5: Ethernet Shield Connector Fixing

The adopted solution seemed to work properly: after that, the Client address was correctly set on the chosen value (192.168.1.2)


REQUIRED LIBRARIES
STANDARD MAC ADDRESS FOR SHIELDS
SERVER ADDRESS (TCP-IP) 192.168.1.X
CLIENT ADDRESS (TCP-IP) 192.168.1.Y

```
#include <SPI.h>
#include <Ethernet.h>

// network configuration. dns server, gateway and subnet are optional.

 // the media access control (ethernet hardware) address for the shield:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xE8 };

// the dns server ip
IPAddress dnServer(192, 168, 1, 1);
// the router's gateway address:
IPAddress gateway(192, 168, 1, 1);
// the subnet:
IPAddress subnet(255, 255, 255, 0);

//the IP address is dependent on your network
IPAddress ip(192, 168, 1, 2);

void setup() {
  Serial.begin(9600);

  // initialize the ethernet device
  Ethernet.begin(mac, ip, dnServer, gateway, subnet);
  //print out the IP address
  delay(2000);
  Serial.print("IP = ");
  Serial.println(Ethernet.localIP());
}

void loop() {
}
```

As we can see, the previous code is characterized by the following steps:

- Including the required libraries SPI (Serial Peripheral Interface) and Ethernet (standard TCP-IP communication protocol)
- declaring a standard MAC address for the Ethernet Shield (0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xE8) because it doesn't have its own printed on the board
- declaring dnServer, gateway and subnet addresses for the TCP-IP Server (Computer) and just the IP address for the TCP-IP Client (Shield).

The aim of this code is just to ensure that the Ethernet Shield is setting on the chosen address and properly communicating through the TCP-IP Client/Server Standard Protocol, therefore there is no VOID LOOP (Main) developed.

The next step is to introduce the Modbus TCP-IP Client/Server Protocol, adapted for Arduino boards: ahead is reported a list of all the useful commands [18]

`#include <ArduinoModbus.h>` (this library depends on the ArduinoRS485 library)

Modbus Client

• client.coilRead()	`int coilRead(int address)`
• client.holdingRegisterRead()	`holdingRegisterRead(int address)`
• client.coilWrite()	`coilWrite(int address)`
• client.holdingRegisterWrite()	`holdingRegisterWrite(int address)`

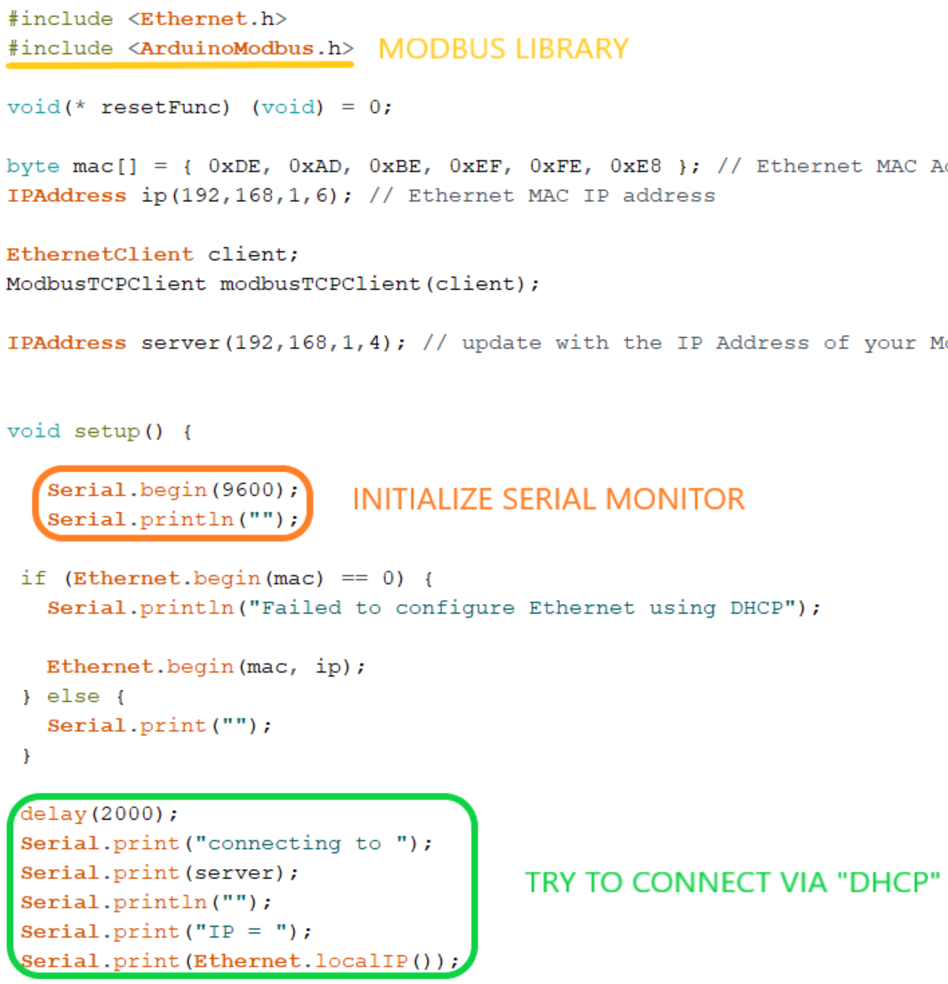
ModbusTCPClient Class

• ModbusTCPClient()	`ModbusTCPClient(client)`
• modbusTCPClient.begin()	`modbusTCPClient.begin(ip, port)`
• modbusTCPClient.connected()	`modbusTCPClient.connected()`
• modbusTCPClient.stop()	`modbusTCPClient.stop()`

ModbusTCPServer

• ModbusTCPServer()	`ModbusTCPServer()`
• modbusTCPServer.begin()	`modbusTCPserver.begin()`
• modbusTCPServer.accept()	`modbusTCPserver.accept(client)`

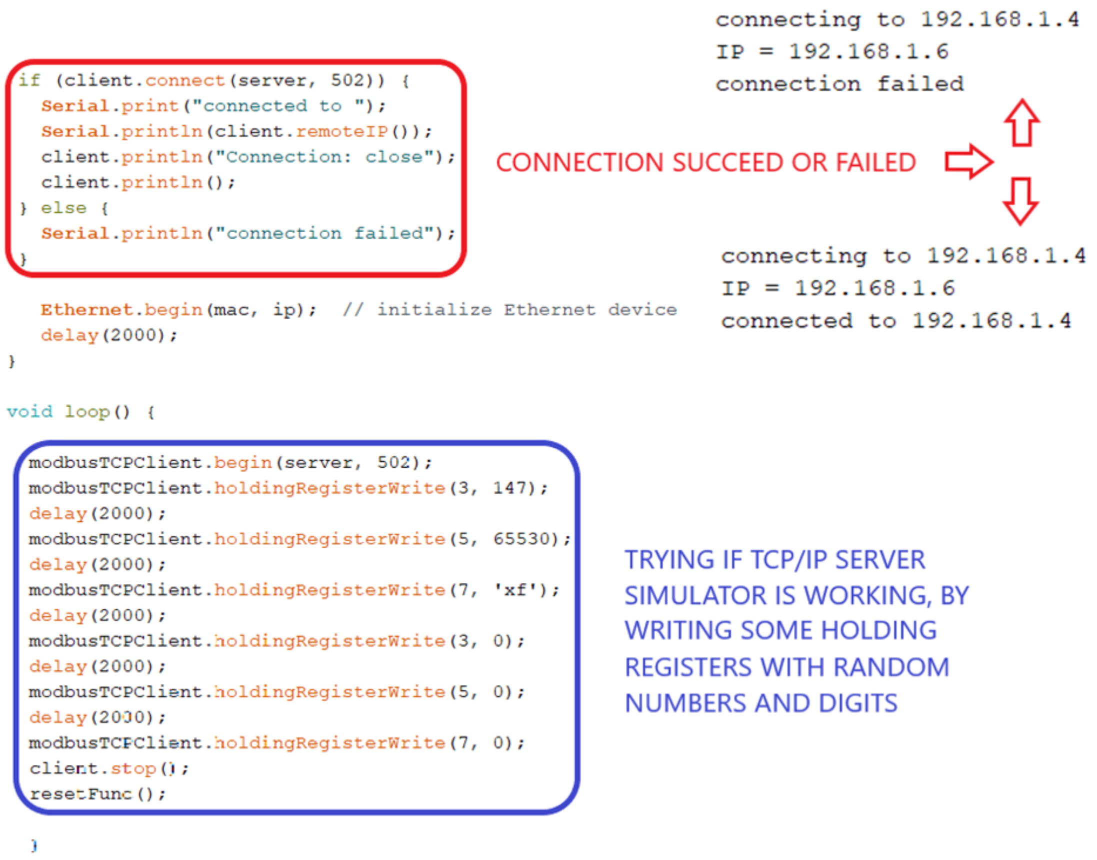
The real Robot is working through reading/writing holding registers, according to the Modbus communication protocol; therefore, the most useful commands are the holding register reading (holdingRegisterRead) and the holding register writing (holdingRegisterWrite).
To implement and test these commands line, the following code was assembled and checked through a free downloadable Modbus TCP-IP server simulator: the goal is to have a digital software simulating the real Robot holding registers behaviors on the computer, avoiding being every time connected with the simulator (heavy and bulky).



```
#include <Ethernet.h>
#include <ArduinoModbus.h>

void(* resetFunc) (void) = 0;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xE8 }; // Ethernet MAC Address
IPAddress ip(192,168,1,6); // Ethernet MAC IP address

EthernetClient client;
ModbusTCPClient modbusTCPClient(client);

IPAddress server(192,168,1,4); // update with the IP Address of your Modbus server


void setup() {

   Serial.begin(9600);
   Serial.println("");

  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");

    Ethernet.begin(mac, ip);
  } else {
    Serial.print("");
  }

 delay(2000);
 Serial.print("connecting to ");
 Serial.print(server);
 Serial.println("");
 Serial.print("IP = ");
 Serial.print(Ethernet.localIP());
```

In this first part of the code, the following steps are performed:

- Including the required libraries Ethernet (standard TCP-IP communication protocol) and Arduino Modbus
- Setting a go-to line, when calling the Reset function by the program
- declaring a standard MAC address, Client and Server IP Addresses
- Initializing Serial Monitor and Ethernet Protocol, trying to connect via DHCP (Dynamic Host Configuration Protocol)



```
connecting to 192.168.1.4
IP = 192.168.1.6
connection failed
```

```
if (client.connect(server, 502)) {
  Serial.print("connected to ");
  Serial.println(client.remoteIP());
  client.println("Connection: close");
  client.println();
} else {
  Serial.println("connection failed");
}

  Ethernet.begin(mac, ip);  // initialize Ethernet device
  delay(2000);
}
```

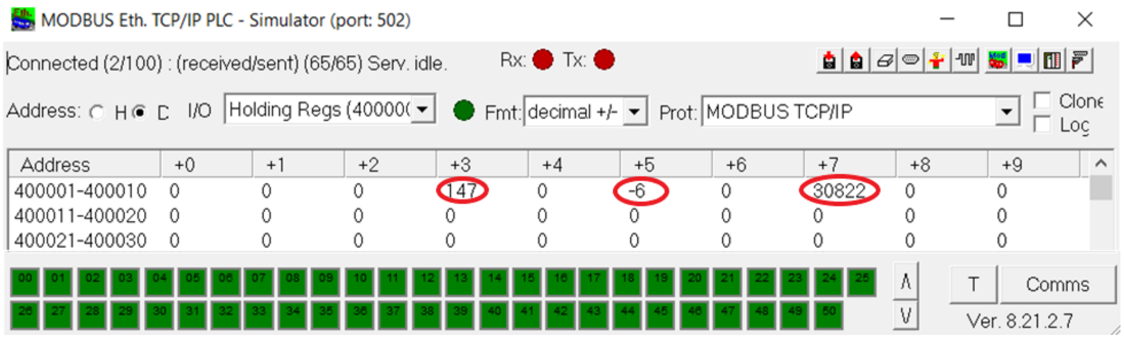
```
connecting to 192.168.1.4
IP = 192.168.1.6
connected to 192.168.1.4
```

```
void loop() {

  modbusTCPClient.begin(server, 502);
  modbusTCPClient.holdingRegisterWrite(3, 147);
  delay(2000);
  modbusTCPClient.holdingRegisterWrite(5, 65530);
  delay(2000);
  modbusTCPClient.holdingRegisterWrite(7, 'xf');
  delay(2000);
  modbusTCPClient.holdingRegisterWrite(3, 0);
  delay(2000);
  modbusTCPClient.holdingRegisterWrite(5, 0);
  delay(2000);
  modbusTCPClient.holdingRegisterWrite(7, 0);
  client.stop();
  resetFunc();
}
```

TRYING IF TCP/IP SERVER SIMULATOR IS WORKING, BY WRITING SOME HOLDING REGISTERS WITH RANDOM NUMBERS AND DIGITS

In the second part of the code, the following steps are performed:

- If the connection is failed: printing "connection failed" on serial monitor (above it is possible to see the real output) and resetting the program
- If the connection is reached: writing the holding register #3 with "147", writing the holding register #5 with "65530" (after two seconds), writing the holding register #7 with "xf" (after two seconds) and finally resetting them back to zero and resetting the program



MODBUS Eth. TCP/IP PLC - Simulator (port: 502)

Connected (2/100) : (received/sent) (65/65) Serv. idle. Rx: ● Tx: ●

Address: ○ H ● D I/O Holding Regs (40000(Fmt: decimal +/- Prot: MODBUS TCP/IP ☐ Clone ☐ Log

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	0	0	0	147	0	-6	0	30822	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	0	0	0	0

Figure 3.6: Modbus TCP-IP Server Simulator Interface

In the previous picture is shown the Modbus TCP-IP server simulator, composed by Coil Outputs (000000 – 099999), Digital Inputs (100000 – 299999), Analog Inputs (300000 – 399999) and Holding Registers (400000 – 465536).

The Holding Registers Screen is selected to check if the assembled code is properly working and writing the correct numbers inside the selected registers:

1 modbusTCPClient.holdingRegisterWrite(3, 147) command line wrote correctly a "147" integer in Register #3

2 modbusTCPClient.holdingRegisterWrite(5, 65530) command line wrote incorrectly a "-6" integer in Register #5, because the number "65530" exceeded the maximum value of 32768 (15 bits + 1 for the sign)

3 modbusTCPClient.holdingRegisterWrite(7, 'xf') command line wrote incorrectly a "30822" integer in Register #7, that's because the digits "xf" are not numbers

3.5 VOCAL CONTROL SETTINGS ON MICROCONTROLLER

The next goal is implementing a possible design of a vocal control for Arduino Board: due to Flash Memory problems, leading to a slowing down of the program computation, it has been necessary to choose a different board with more Flash Memory. For that, Arduino Mega was chosen:

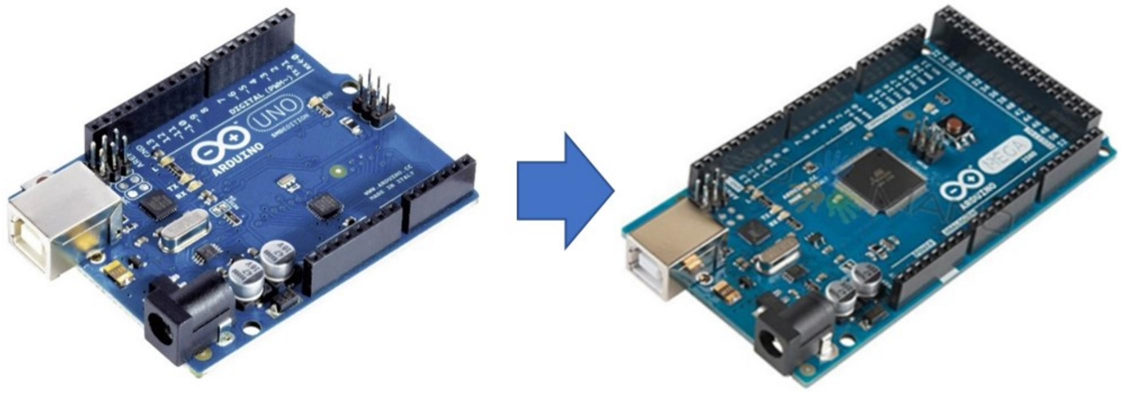


Figure 3.7: Arduino Uno (left) & Arduino Mega (right)

To perform a proper communication between Mobile Phone and Microcontroller, Bluetooth is indeed the fastest and most reliable solution: therefore, a HC-05 Bluetooth Transmitter/Receiver module compatible with Arduino was chosen for this aim. Furthermore, to avoid checking every time the serial monitor, a Liquid Crystal Display (LCD) with I2C interface was added to the main circuit.

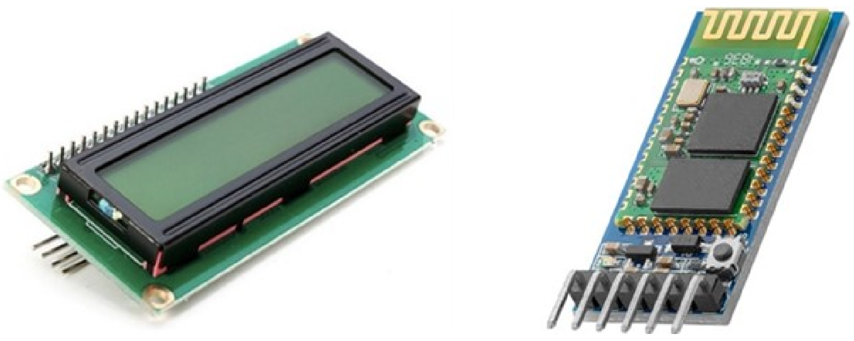


Figure 3.8: Liquid Crystal Display (left) & HC-05 Bluetooth Tx/Rx (right)

To connect all the previously listed modules with the main board, the following circuit configuration was developed:

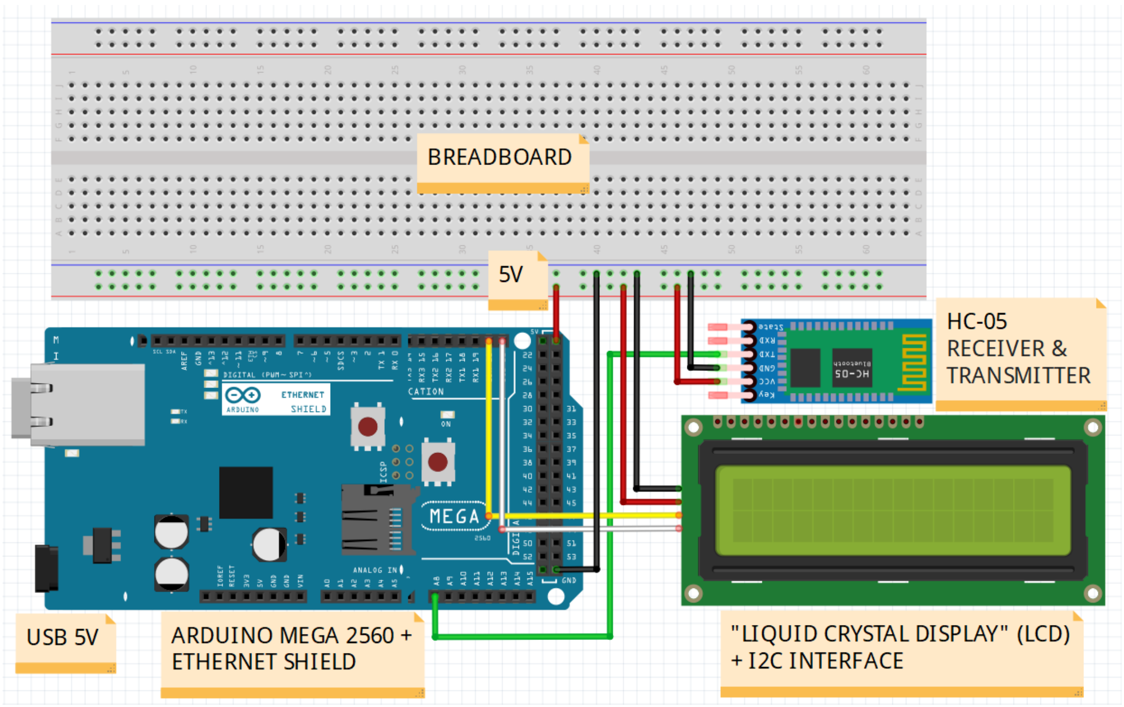

BREADBOARD
5V
HC-05 RECEIVER & TRANSMITTER
USB 5V
ARDUINO MEGA 2560 + ETHERNET SHIELD
"LIQUID CRYSTAL DISPLAY" (LCD) + I2C INTERFACE

Figure 3.9: Starting Circuit Scheme

This scheme is characterized by the following connections:

1 Ethernet Shield mounted on the Arduino Mega board

2 5V coming from Arduino's supply pins linked to one of the four breadboard supply lines (positive and **negative** pins)

3 HC-05 and LCD modules linked to the same breadboard 5V line to be supplied (positive and **negative** pins)

4 HC-05 transmitter pin connected to A8 pin (receiver) of Arduino board

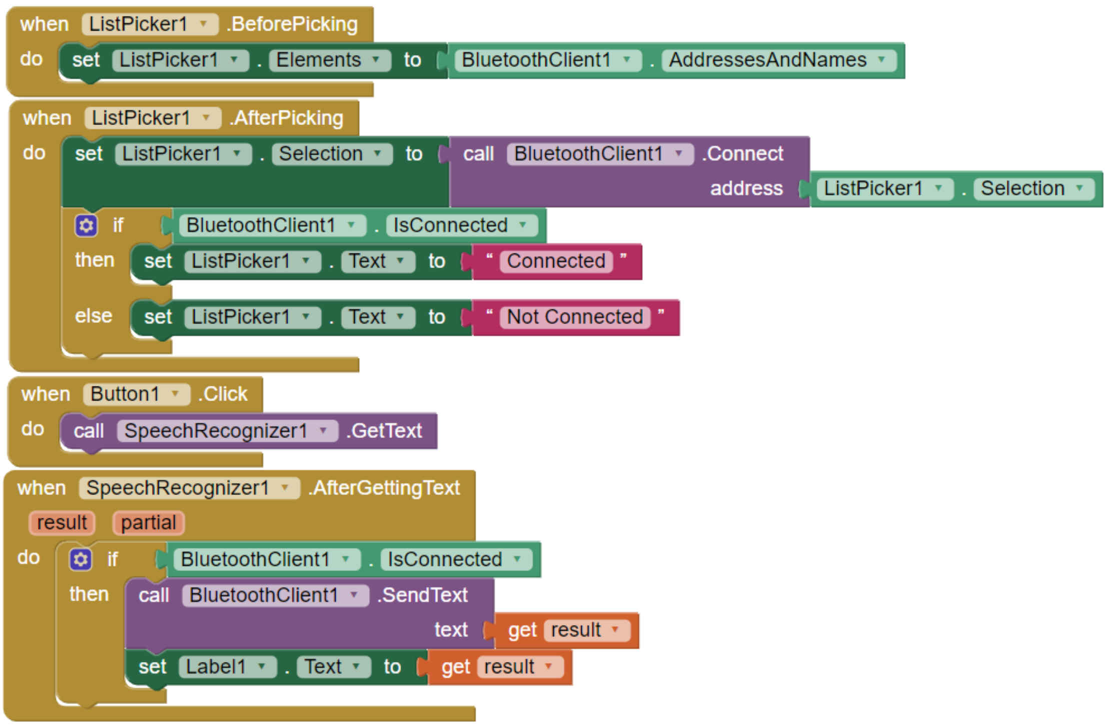
5 LCD Data Signal pin connected to Pin 20 (SDA Port)

6 LCD Clock Signal pin is connected to Pin 21 (SCL Port).

3.6 VOCAL CONTROL MOBILE APP BUILD

For simplicity reasons, to send vocal commands to the Bluetooth receiver, an Android mobile phone was selected; for this aim, a mobile phone vocal control App required to be implemented: one of the most direct and intuitive ways is indeed a block-structured free Android App editor (Open-Source) called "App Inventor". [19]

The revolution brought from this editor is that, instead of writing and compiling standard programming code, it provides a logical and faster way to compile instructions and functions blocks, showing a specific App screen in the connected device. The blocks configuration built is the sequent:



```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  BluetoothClient1 . AddressesAndNames

when ListPicker1 .AfterPicking
do  set ListPicker1 . Selection to  call BluetoothClient1 .Connect
                                                   address  ListPicker1 . Selection
    if   BluetoothClient1 . IsConnected
    then set ListPicker1 . Text to " Connected "
    else set ListPicker1 . Text to " Not Connected "

when Button1 .Click
do  call SpeechRecognizer1 .GetText

when SpeechRecognizer1 .AfterGettingText
 result  partial
do  if   BluetoothClient1 . IsConnected
    then call BluetoothClient1 .SendText
                             text  get result
         set Label1 . Text to  get result
```

Figure 3.10: Mobile App Blocks Configuration

As we can see in the previous figure, it is composed by four main blocks:

1 The first block is creating a list of all possible Bluetooth devices to connect to: before selecting one device (.BeforePicking), the block is setting the list (ListPicker1) with all the elements (.AdressesAndNames) coming from the mobile phone Bluetooth searching component (BluetoothClient1) implemented in it

2 The second block is performing a connection to the selected Bluetooth module: after have selected one device (.AfterPicking), the block calls a connection function (.Connect) linking to the chosen device address (.Selection) and, if the device is connected (.IsConnected), then set the text “Connected” into the label (ListPicker1); otherwise set the text “Not Connected” into it

3 The third block is converting pronounced words into text: when the SPEAK button is pressed, the block is calling the SpeechReconizer1 component by directly linking to the Android Google Vocal Assist and, through it, converting words in a text string

4 The fourth block is sending the acquired text via Bluetooth: after having gotten the text string (.AfterGettingText), if the Bluetooth device is still connected (.IsConnected) then send (.SendText) the text string (result) via Bluetooth and print the same text into the Label1

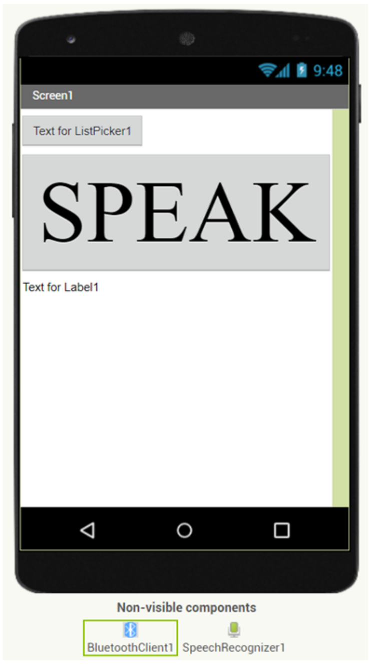


Figure 3.11: Vocal App Screen

The related App screen is composed by:

- "List Picker 1" to choose the Bluetooth device to connect to
- "SPEAK button" to start listening from the Vocal Assistant
- "Label 1" to check that the vocal input is correctly interpretated into a string

By touching "ListPicker1" button, a list of Bluetooth connectible devices appears (Addresses & Names), and it is possible to select the required one (HC-05). After having picked it up, the App ("BluetoothClient1") connects to the selected address (Selection) and changes the text of the "ListPicker1" button in "Connected" or "Not Connected", depending on the outcome of the connecting transition. When the "SPEAK" button is clicked, "SpeechReconizer1" component links to mobile phone vocal assistant and converts words in a text string (result). After that, the resulted text is sent via Bluetooth to the HC-05 receiver and to the "Text for Label1" label to check the correct interpretation of the words. By downloading the free "AI2 Companion" App on the Mobile device, it is possible to upload and compile the program to obtain such screenshot configurations (before and after):

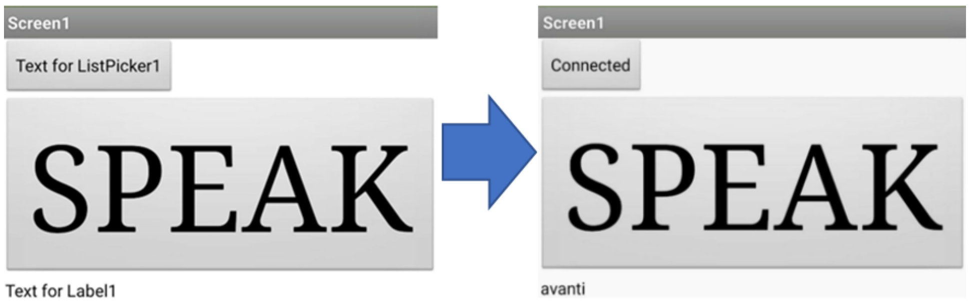


Figure 3.12: Vocal App Mobile Phone Screenshots

Starting from the initial screen, the following steps are performed:

1 Click on "Text for ListPicker1" label and select the HC-05 module
2 Waiting for connection ("Text for ListPicker1" will turn into "Connected")
3 Click on SPEAK button and pronounce the wanted command
4 Waiting for the Vocal Assistant to convert words in text and send it via Bluetooth ("Text for Label1" will turn into the command word, in this case "avanti")

Chapter 4

REAL IMPLEMENTATION

4.1 REAL CONFIGURATION BUILDING

We pass now to the real circuit (and relative code) building: once checked that the Mobile phone App is interpreting and sending words correctly via Bluetooth to the HC-05 receiver, the next step is to implement a 9V battery and an ON/OFF switch, to let the circuit be self-supplied and free from the USB cable (required just to upload the code): in this way it is possible to connect the complete circuit directly to the Robot Control Cabinet just with an Ethernet Cable.

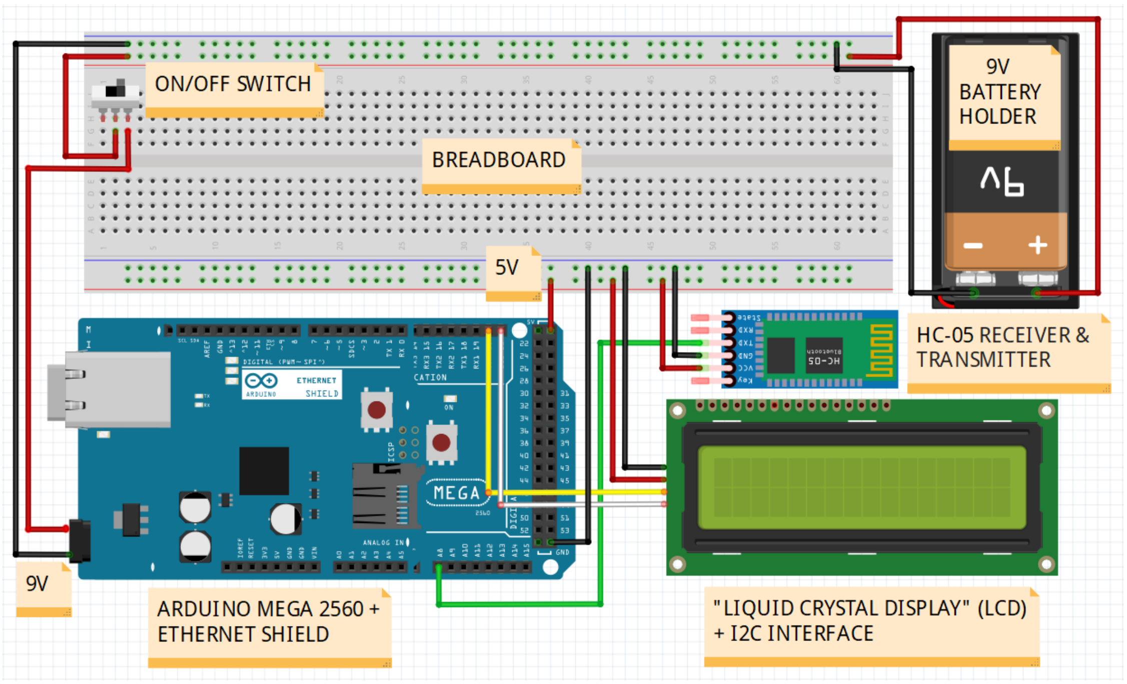

ON/OFF SWITCH
BREADBOARD
9V BATTERY HOLDER
5V
HC-05 RECEIVER & TRANSMITTER
9V
ARDUINO MEGA 2560 + ETHERNET SHIELD
"LIQUID CRYSTAL DISPLAY" (LCD) + I2C INTERFACE

Figure 4.1: Final Circuit Scheme

As we can see on the picture above, a 9V battery holder was added and connected to the secondary supply line of the breadboard, then connected to the switch and in the end, through a proper round connector, strictly linked to the Arduino 9V-12V supply port. The other part of the circuit staid the same as before, with the primary supply line of the breadboard connected to the 5V Arduino Supply Output and to the two modules (HC-05 and I2C LCD).
On Arduino Board, A8 Pin (Receiver Port) is connected to the HC-05 Tx Pin (Transmitter Port) and for the I2C LCD, the Data Signal Pin is connected to the Pin 20 (SDA Port) and the Clock Signal is connected to the Pin 21 (SCL Port).
In the next picture, it is possible to see the real circuit assembled and perfectly working:

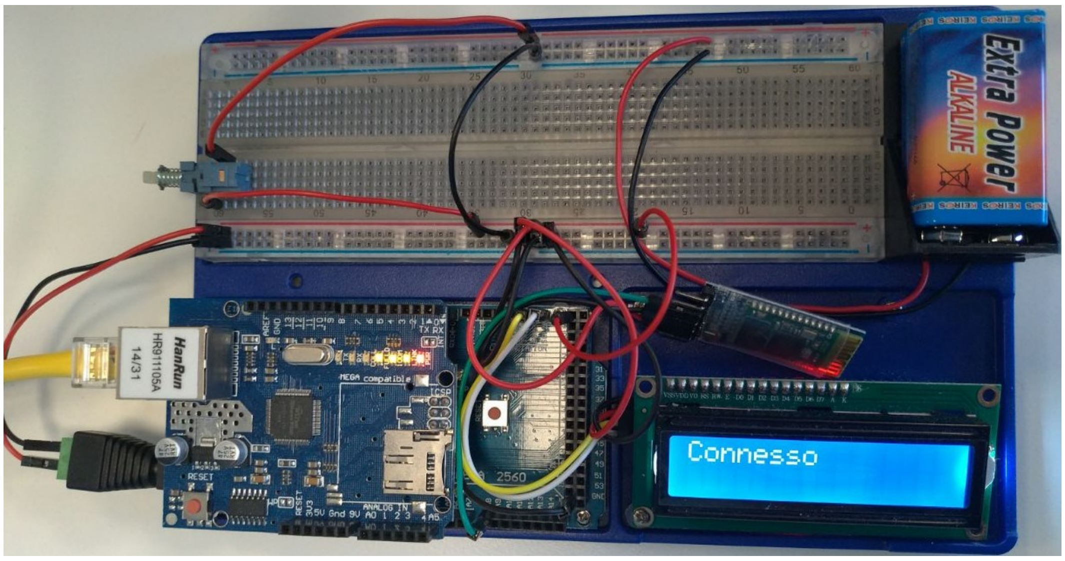


Figure 4.2: Final Circuit Configuration

To be able to connect via Ethernet all the devices together, it was necessary the usage of an Ethernet Switch, automatically acquiring all the addresses and letting the communication within each other.

The Ethernet Switch is connecting:

- the Opus A3 Robot Control Unit (via USB-to-Ethernet converter)
- the Notebook, where it is possible to supervise both Arduino's Serial Monitor and the Robot Control Unit (RCU) monitor (not mandatory connection)
- the implementation of Arduino board circuit (via Ethernet Shield)

The USB cable is connecting:

- the Notebook, to upload the assembled code
- Arduino board circuit

The Bluetooth is connecting:

- The mobile phone vocal control App (to perform voice recognition)
- HC-05 receiver, mounted on the Arduino board circuit

A scheme of these connections is shown in the next figure:

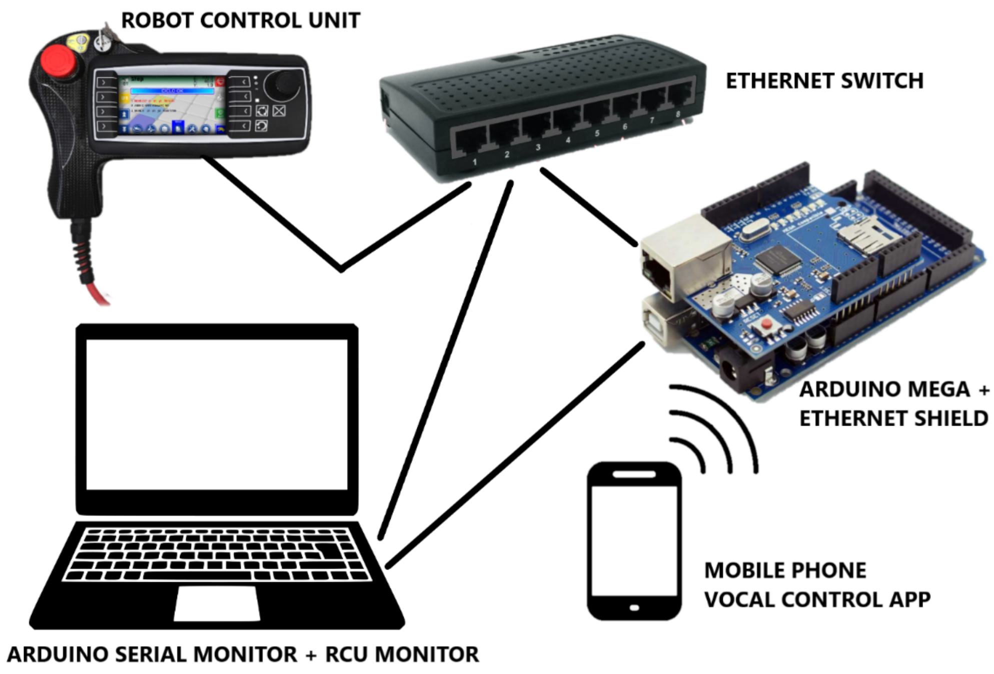
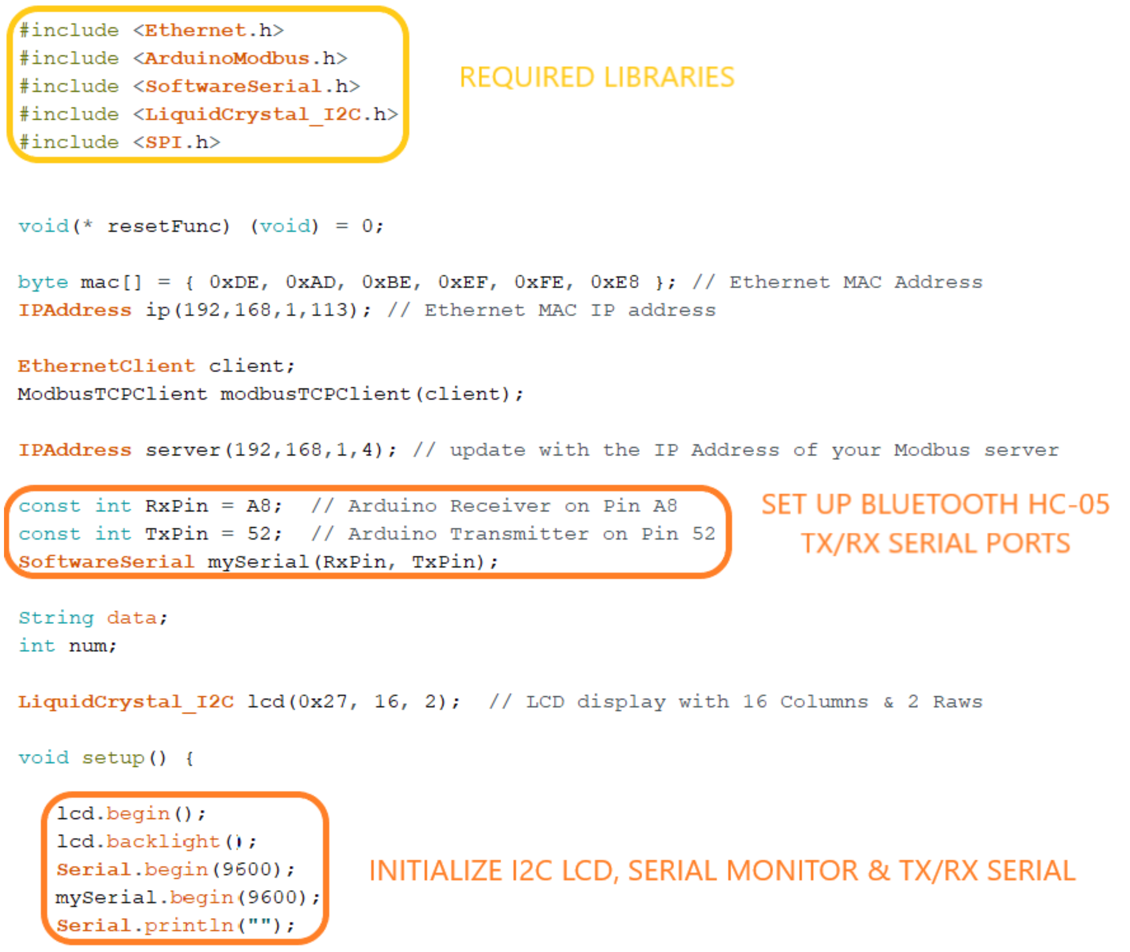

ROBOT CONTROL UNIT
ETHERNET SWITCH
ARDUINO MEGA + ETHERNET SHIELD
MOBILE PHONE VOCAL CONTROL APP
ARDUINO SERIAL MONITOR + RCU MONITOR

Figure 4.3: Connections Scheme

4.2 FINAL CODE ASSEMBLING

Regarding the up-loadable code, considering the previous codes developing, a final version is implemented in this way:



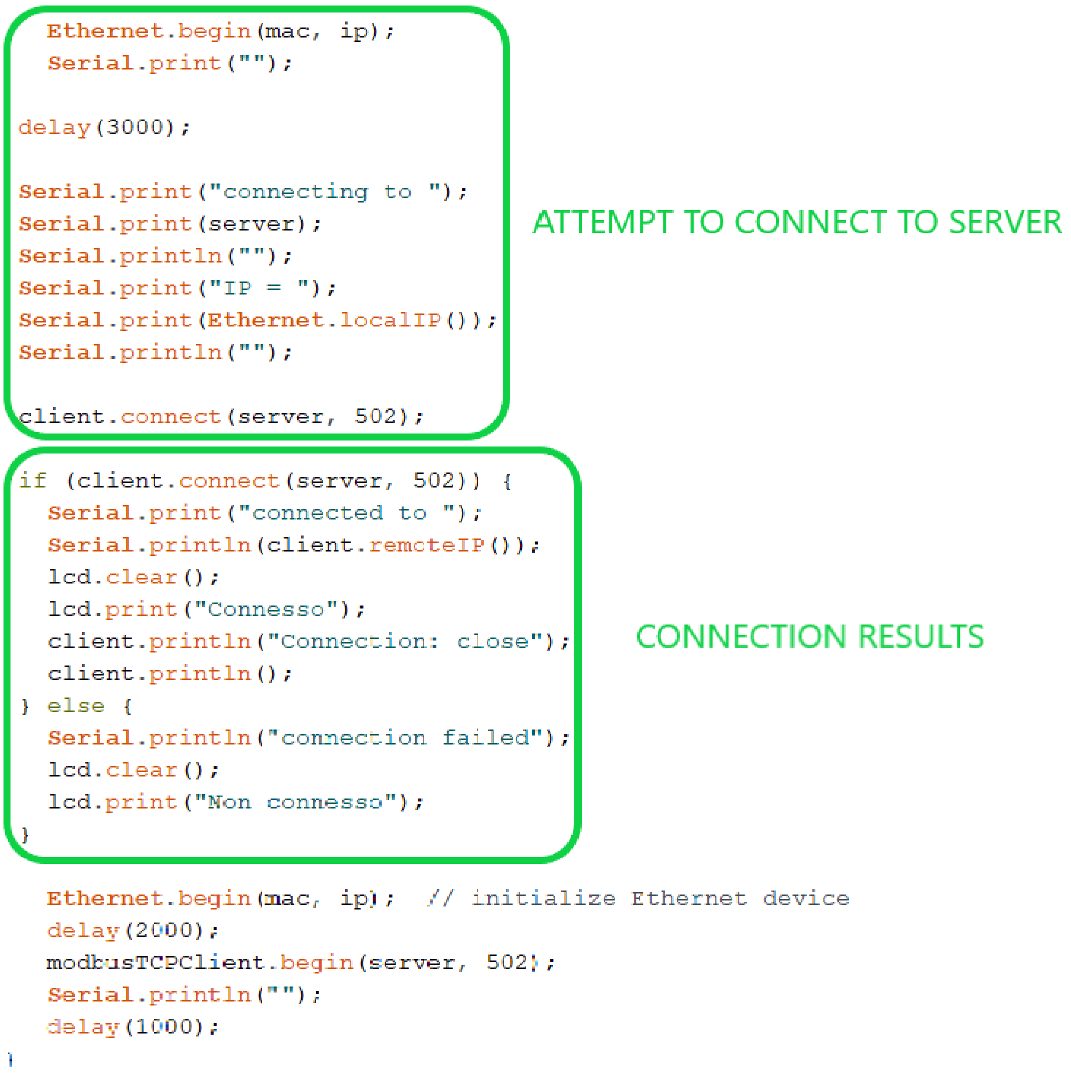
```
#include <Ethernet.h>
#include <ArduinoModbus.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>


void(* resetFunc) (void) = 0;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xE8 }; // Ethernet MAC Address
IPAddress ip(192,168,1,113); // Ethernet MAC IP address

EthernetClient client;
ModbusTCPClient modbusTCPClient(client);

IPAddress server(192,168,1,4); // update with the IP Address of your Modbus server

const int RxPin = A8;  // Arduino Receiver on Pin A8
const int TxPin = 52;  // Arduino Transmitter on Pin 52
SoftwareSerial mySerial(RxPin, TxPin);

String data;
int num;

LiquidCrystal_I2C lcd(0x27, 16, 2);  // LCD display with 16 Columns & 2 Raws

void setup() {

   lcd.begin();
   lcd.backlight();
   Serial.begin(9600);
   mySerial.begin(9600);
   Serial.println("");
```

REQUIRED LIBRARIES
SET UP BLUETOOTH HC-05 TX/RX SERIAL PORTS
INITIALIZE I2C LCD, SERIAL MONITOR & TX/RX SERIAL

The final code is performing the following steps:

- Including of all the required libraries: Ethernet protocol (“Ethernet.h”), Modbus protocol (“ArduinoModbus.h”), Bluetooth Transmitting/Receiving serial protocol (“SoftwareSerial.h”), I2C LCD Display (“LiquidCrystal_I2C.h”), Serial Peripheral Interface (“SPI.h”)
- A Reset function is declared so that, when called, the code will restart from this line on

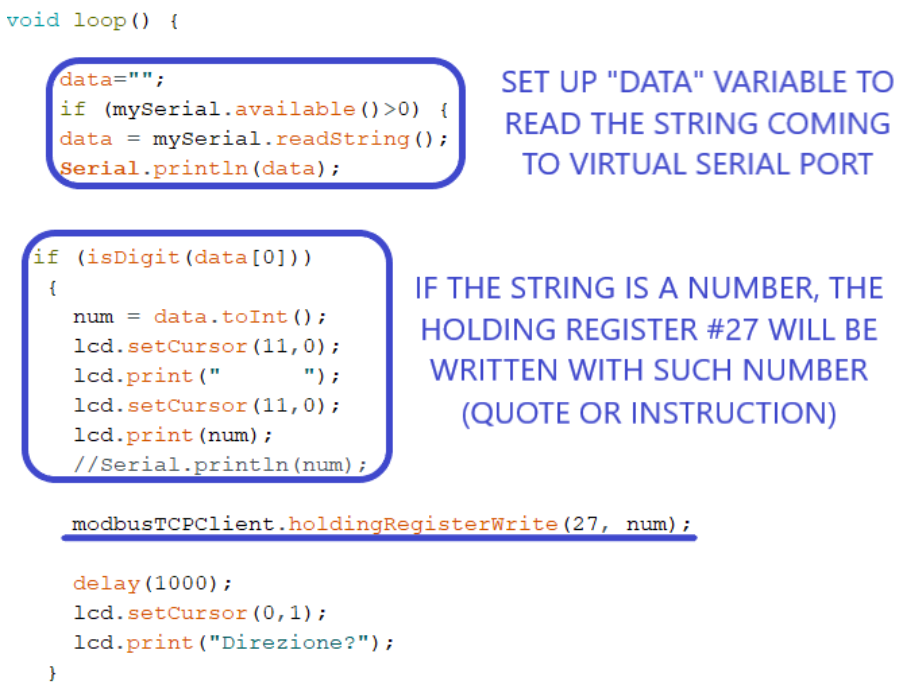
- Declaring a standard MAC address, Client and Server IP Addresses
- Initializing Serial Monitor and Ethernet Protocol, trying to connect via DHCP (Dynamic Host Configuration Protocol)
- Setting up HC-05 Transmitting and Receiving Serial Ports and serial function
- Declaring variables (“data” as a string and “num” as an integer)
- Declaring LCD with I2C (0x27) interface (16 columns and 2 rows)
- Initialing LCD, serial monitor and data receiving serial function



```
  Ethernet.begin(mac, ip);
  Serial.print("");

delay(3000);

Serial.print("connecting to ");
Serial.print(server);
Serial.println("");
Serial.print("IP = ");
Serial.print(Ethernet.localIP());
Serial.println("");

client.connect(server, 502);
if (client.connect(server, 502)) {
  Serial.print("connected to ");
  Serial.println(client.remoteIP());
  lcd.clear();
  lcd.print("Connesso");
  client.println("Connection: close");
  client.println();
} else {
  Serial.println("connection failed");
  lcd.clear();
  lcd.print("Non connesso");
}
  Ethernet.begin(mac, ip);  // initialize Ethernet device
  delay(2000);
  modbusTCPClient.begin(server, 502);
  Serial.println("");
  delay(1000);
}
```

Continuing with:

- Initializing the Ethernet TCP-IP protocol
- Modbus Client (Arduino) trying to establish a connection with the Modbus Server (Robot Control Unit)
- Depending on the result, printing on both serial monitor and LCD the current status (Connected or not)
- Reached the connection, Ethernet communication and TCP Client are set and ready for the main part (void loop) to start



```
void loop() {

  data="";
  if (mySerial.available()>0) {
  data = mySerial.readString();
  Serial.println(data);

 if (isDigit(data[0]))
  {
    num = data.toInt();
    lcd.setCursor(11,0);
    lcd.print("       ");
    lcd.setCursor(11,0);
    lcd.print(num);
    //Serial.println(num);

    modbusTCPClient.holdingRegisterWrite(27, num);

    delay(1000);
    lcd.setCursor(0,1);
    lcd.print("Direzione?");
  }
```

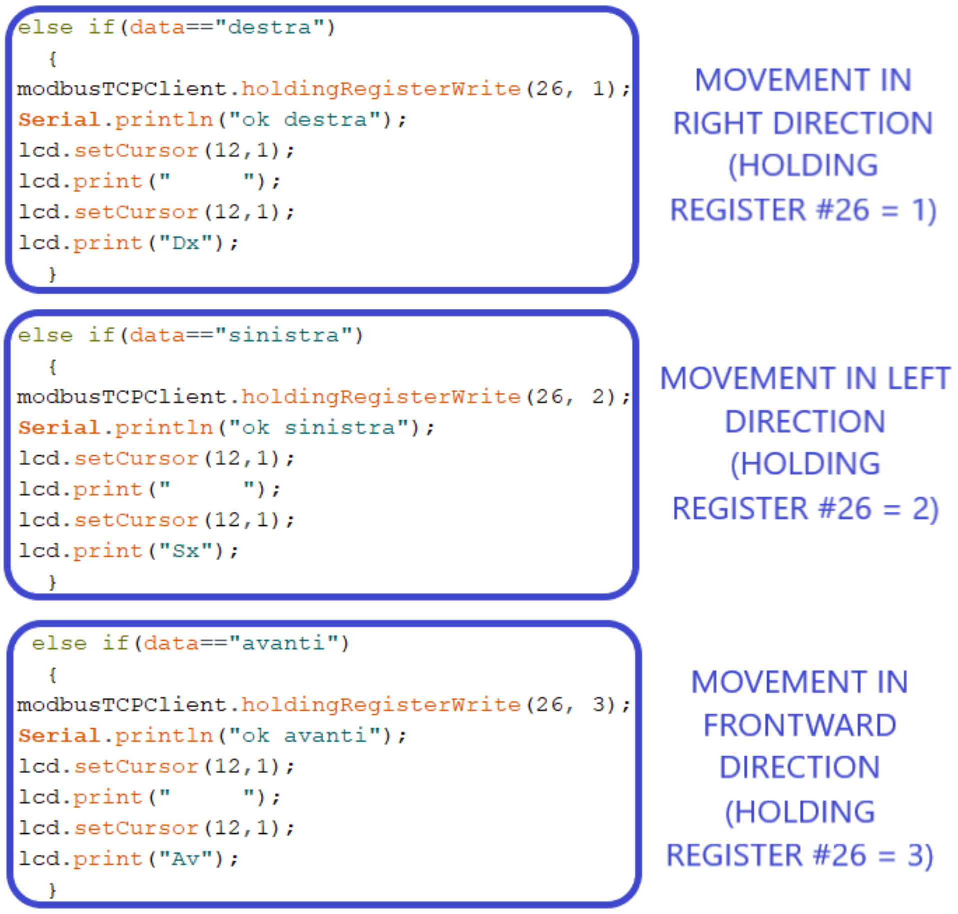
SET UP "DATA" VARIABLE TO READ THE STRING COMING TO VIRTUAL SERIAL PORT

IF THE STRING IS A NUMBER, THE HOLDING REGISTER #27 WILL BE WRITTEN WITH SUCH NUMBER (QUOTE OR INSTRUCTION)

Inside the Void Loop, every time a new word is pronounced, the “data” variable must be reset (data = "";) and get ready to read a new string (words or numbers) coming from the Bluetooth transmitter (data = mySerial.readString();).

If the first digit of the string is a number, “data” variable (string) will be converted into “num” variable (integer) and written into holding register #27 (this register can contain specific instruction or axial range in the form of positive numbers).

In this case, the Robot Control Unit is going to wait for a specific request (such as an axial movements) and, after having received it, both Serial monitor and LCD will print the text “Direction?”, to let the user know that after a specific range (in millimeters) it is required a direction and a sense to be performed.



```
else if(data=="destra")
  {
modbusTCPClient.holdingRegisterWrite(26, 1);
Serial.println("ok destra");
lcd.setCursor(12,1);
lcd.print("     ");
lcd.setCursor(12,1);
lcd.print("Dx");
  }
```

MOVEMENT IN RIGHT DIRECTION (HOLDING REGISTER #26 = 1)

```
else if(data=="sinistra")
  {
modbusTCPClient.holdingRegisterWrite(26, 2);
Serial.println("ok sinistra");
lcd.setCursor(12,1);
lcd.print("     ");
lcd.setCursor(12,1);
lcd.print("Sx");
  }
```

MOVEMENT IN LEFT DIRECTION (HOLDING REGISTER #26 = 2)

```
 else if(data=="avanti")
  {
modbusTCPClient.holdingRegisterWrite(26, 3);
Serial.println("ok avanti");
lcd.setCursor(12,1);
lcd.print("     ");
lcd.setCursor(12,1);
lcd.print("Av");
  }
```

MOVEMENT IN FRONTWARD DIRECTION (HOLDING REGISTER #26 = 3)

Regarding basic movements, once given the range (converted into a number), it is possible to choose a direction along the three axes:

- Rightward direction: after having pronounced the word “destra” (right), the Robot will move along *z* axis (positive sense), by the writing of the holding register #26 with “1” (related to a rightward movement) and of the holding register #27 with the specific range

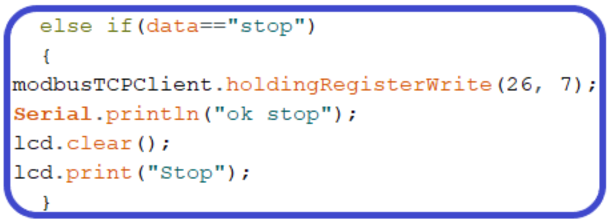
- Leftward direction: after having pronounced the word “sinistra” (left), the Robot will move along *z* axis (negative sense), by the writing of the holding register #26 with “2” (related to a leftward movement) and of the holding register #27 with the specific range

- Frontward direction: after having pronounced the word “avanti” (ahead), the Robot will move along *x* axis (positive sense), by the writing of the holding register #26 with “3” (related to a frontward movement) and of the holding register #27 with the specific range

- Backward direction: after having pronounced the word “indietro” (back), the Robot will move along *x* axis (negative sense), by the writing of the holding register #26 with “4” (related to a backward movement) and of the holding register #27 with the specific range

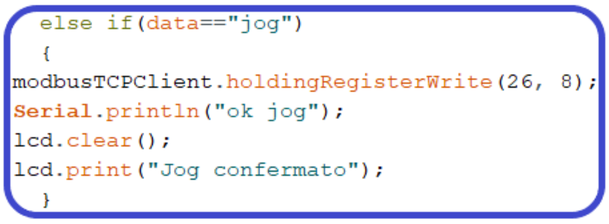
- Downward direction: after having pronounced the word “basso” (down), the Robot will move along *y* axis (positive sense), by the writing of the holding register #26 with “5” (related to a downward movement) and of the holding register #27 with the specific range

- Upward direction: after having pronounced the word “alto” (up), the Robot will move along *y* axis (negative sense), by the writing of the holding register #26 with “6” (related to an upward movement) and of the holding register #27 with the specific range



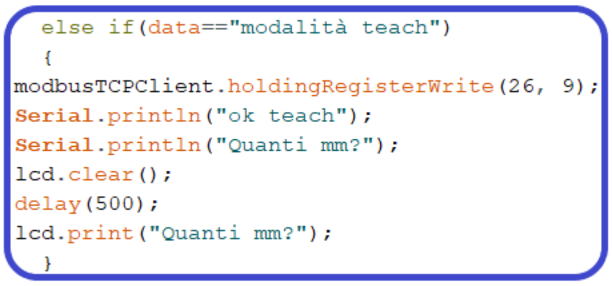
```
  else if(data=="stop")
  {
modbusTCPClient.holdingRegisterWrite(26, 7);
Serial.println("ok stop");
lcd.clear();
lcd.print("Stop");
  }
```

STOP COMMAND (HOLDING REGISTER #26 = 7)



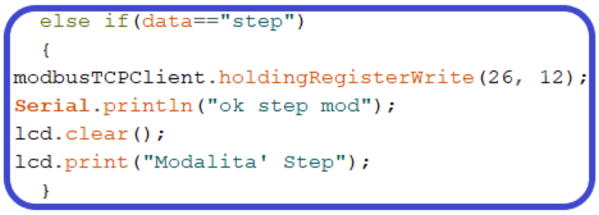
```
  else if(data=="jog")
  {
modbusTCPClient.holdingRegisterWrite(26, 8);
Serial.println("ok jog");
lcd.clear();
lcd.print("Jog confermato");
  }
```

JOG MODE (HOLDING REGISTER #26 = 8)



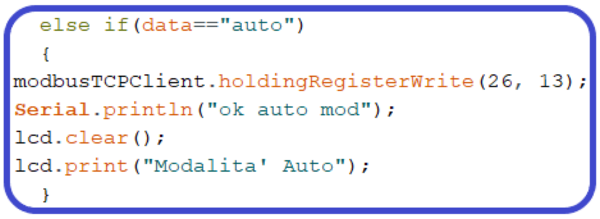
```
  else if(data=="modalità teach")
  {
modbusTCPClient.holdingRegisterWrite(26, 9);
Serial.println("ok teach");
Serial.println("Quanti mm?");
lcd.clear();
delay(500);
lcd.print("Quanti mm?");
  }
```

TEACH MODE (HOLDING REGISTER #26 = 9)



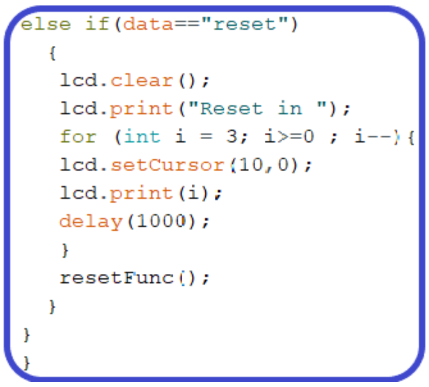
```
  else if(data=="step")
  {
modbusTCPClient.holdingRegisterWrite(26, 12);
Serial.println("ok step mod");
lcd.clear();
lcd.print("Modalita' Step");
  }
```

STEP MODE
(HOLDING
REGISTER #26 = 12)



```
  else if(data=="auto")
  {
modbusTCPClient.holdingRegisterWrite(26, 13);
Serial.println("ok auto mod");
lcd.clear();
lcd.print("Modalita' Auto");
  }
```

AUTO MODE
(HOLDING
REGISTER #26 = 13)



```
else if(data=="reset")
  {
   lcd.clear();
   lcd.print("Reset in ");
   for (int i = 3; i>=0 ; i--){
   lcd.setCursor(10,0);
   lcd.print(i);
   delay(1000);
   }
   resetFunc();
  }
}
}
```

RESET FUNCTION
(WHEN CALLED, THE
PROGRAM WILL RESTART
FROM THE BEGINNING)

Regarding main modes entering, once given an instruction (converted into a number), one of the following possibilities can be performed:

- STOP Command: in any case, after having pronounced the word “Stop”, the Robot will strictly stop, independently from the performing action, by the writing of the holding register #26 with “7”

- JOG Mode: after having pronounced the word “Jog”, the Robot will enter in the so-called JOG Mode, in which it is moving according to the selected coordinate system, by the writing of the holding register #26 with “8”

- TEACH Mode: after having pronounced the words “Modalità Teach” (Teach Mode), the Robot will enter in the so-called TEACH Mode, in which it is instructed in its specific motions and trajectories, by the writing of the holding register #26 with “9”

- STEP Mode: after having pronounced the word “Step”, the Robot will enter in the so-called STEP Mode, in which it is moving in a step-by-step way, performing the selected path, by the writing of the holding register #26 with “12”

- AUTO Mode: after having pronounced the word “Auto”, the Robot will enter in the so-called AUTO Mode, in which it is moving automatically, performing the selected trajectories, by the writing of the holding register #26 with “13”

- RESET Function: after having pronounced the word “reset”, the entire program will restart from the very beginning, simulating a manual reset (by pressing the dedicated button).

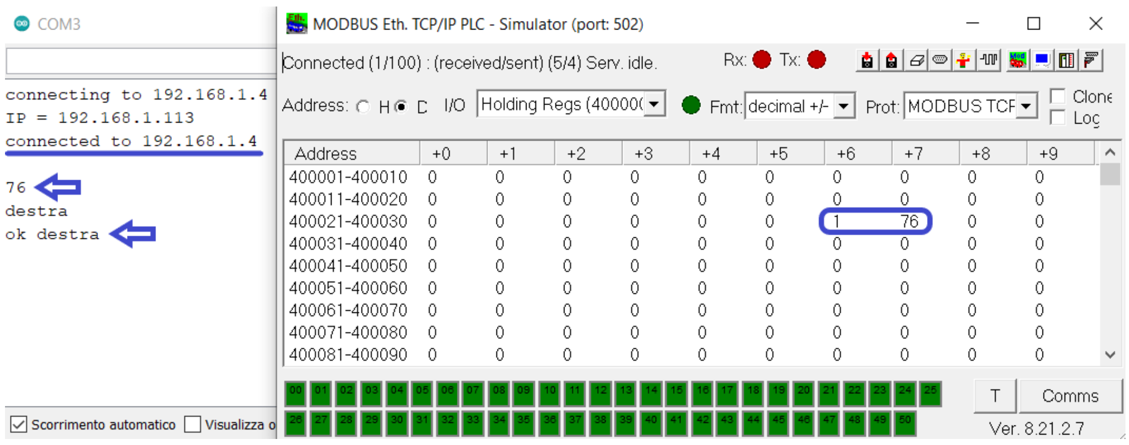
Chapter 5

FINAL TESTS

5.1 FINAL CODE WORKING TEST

The aim of this part is performing the final working tests, starting from the built code: by receiving some commands, the serial monitor must print the correct words and in the TCP Modbus simulator the specific holding registers must be written.

- Movement to the rightward direction about 76 mm



```
connecting to 192.168.1.4
IP = 192.168.1.113
connected to 192.168.1.4

76
destra
ok destra
```

Connected (1/100) : (received/sent) (5/4) Serv. idle.

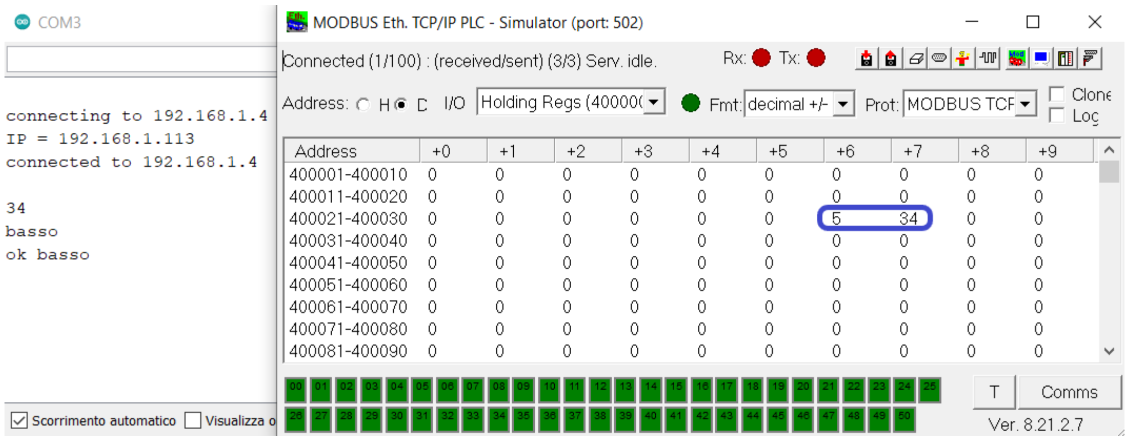
Address: H D I/O Holding Regs (40000(Fmt: decimal +/- Prot: MODBUS TCP

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	0	0	0	0	0	0	0	0	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	1	76	0	0
400031-400040	0	0	0	0	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0	0	0	0	0
400051-400060	0	0	0	0	0	0	0	0	0	0
400061-400070	0	0	0	0	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0	0	0	0	0
400081-400090	0	0	0	0	0	0	0	0	0	0

Figure 5.1: Modbus TCP-IP Server Simulator (Rightward Movement)

After having successfully connected to the server simulator (192.168.1.4), it was first pronounced the word “76”, that the program recognized as a number and wrote it on the Holding Register #27; after that it was pronounced the word “destra” (right) that the program recognized as a rightward movement command and wrote the Holding Register #26 with the number 1.

- Movement to the downward direction about 34 mm



```
connecting to 192.168.1.4
IP = 192.168.1.113
connected to 192.168.1.4

34
basso
ok basso
```

Connected (1/100) : (received/sent) (3/3) Serv. idle.

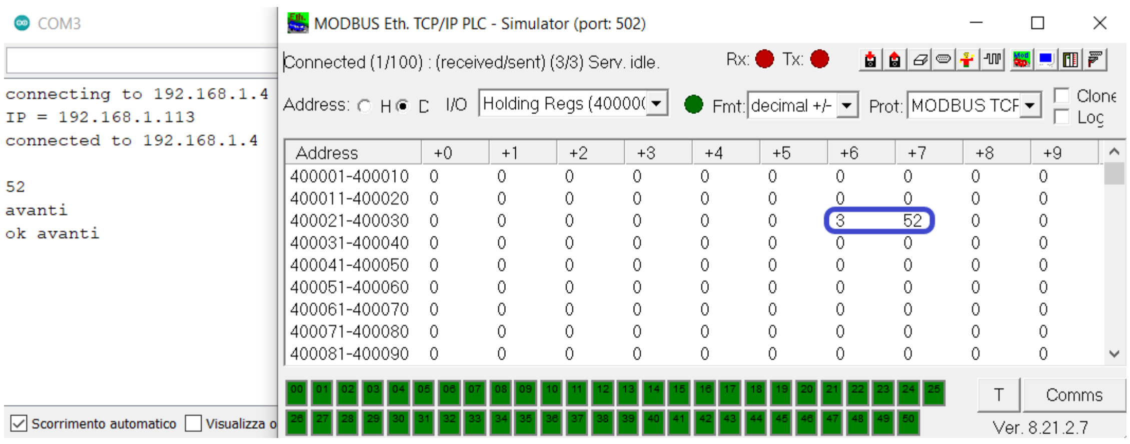
Address: H D I/O Holding Regs (40000(Fmt: decimal +/- Prot: MODBUS TCP

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	0	0	0	0	0	0	0	0	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	5	34	0	0
400031-400040	0	0	0	0	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0	0	0	0	0
400051-400060	0	0	0	0	0	0	0	0	0	0
400061-400070	0	0	0	0	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0	0	0	0	0
400081-400090	0	0	0	0	0	0	0	0	0	0

Figure 5.2: Modbus TCP-IP Server Simulator (Downward Movement)

This time it was first pronounced the word “34”, that the program recognized as a number and wrote it on the Holding Register #27; then it was pronounced the word “basso” (down) that the program recognized as a downward movement command and wrote the Holding Register #26 with the number 5.

- Movement to the frontward direction about 52 mm



```
connecting to 192.168.1.4
IP = 192.168.1.113
connected to 192.168.1.4

52
avanti
ok avanti
```

MODBUS Eth. TCP/IP PLC - Simulator (port: 502)

Connected (1/100) : (received/sent) (3/3) Serv. idle.

Address: H D I/O Holding Regs (40000(Fmt: decimal +/- Prot: MODBUS TCF

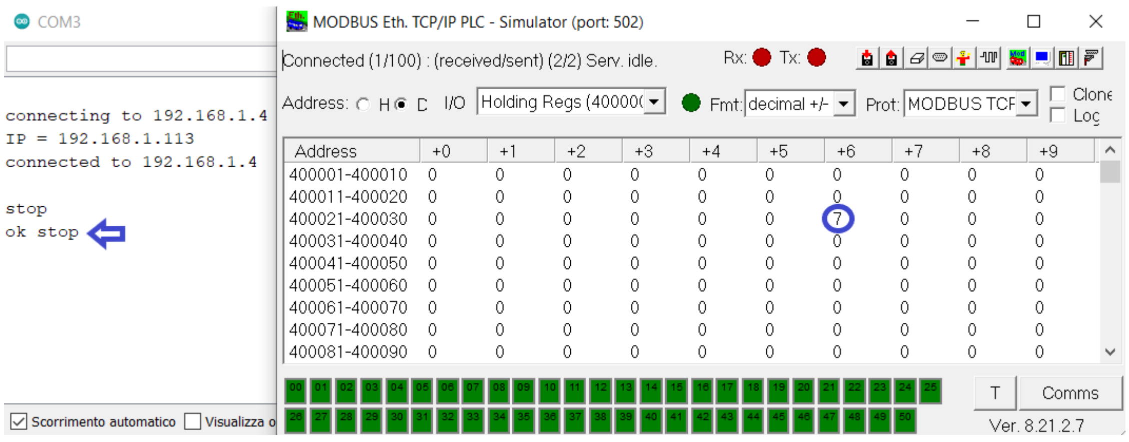
Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	0	0	0	0	0	0	0	0	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	3	52	0	0
400031-400040	0	0	0	0	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0	0	0	0	0
400051-400060	0	0	0	0	0	0	0	0	0	0
400061-400070	0	0	0	0	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0	0	0	0	0
400081-400090	0	0	0	0	0	0	0	0	0	0

Ver. 8.21.2.7

Figure 5.3: Modbus TCP-IP Server Simulator (Frontward Movement)

Just as before, Holding Register #27 was written with the range number (52) and Holding Register #26 with the number 3 (associated to a frontward movement).

- STOP Command



```
connecting to 192.168.1.4
IP = 192.168.1.113
connected to 192.168.1.4

stop
ok stop
```

MODBUS Eth. TCP/IP PLC - Simulator (port: 502)

Connected (1/100) : (received/sent) (2/2) Serv. idle.

Address: H D I/O Holding Regs (40000(Fmt: decimal +/- Prot: MODBUS TCF

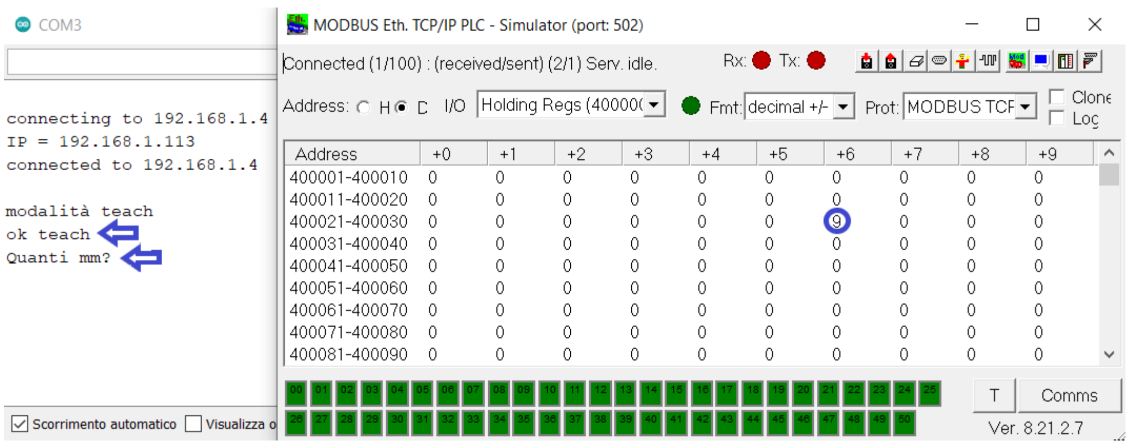
Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	0	0	0	0	0	0	0	0	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	7	0	0	0
400031-400040	0	0	0	0	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0	0	0	0	0
400051-400060	0	0	0	0	0	0	0	0	0	0
400061-400070	0	0	0	0	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0	0	0	0	0
400081-400090	0	0	0	0	0	0	0	0	0	0

Ver. 8.21.2.7

Figure 5.4: Modbus TCP-IP Server Simulator (STOP Command)

Safety word “STOP” must work properly: after having pronounced it, the Holding Register #26 is written with the number 7 (associated to a stopping action).

- TEACH Mode Command



```
connecting to 192.168.1.4
IP = 192.168.1.113
connected to 192.168.1.4

modalità teach
ok teach
Quanti mm?
```

[✓] Scorrimento automatico [ ] Visualizza o

Connected (1/100) : (received/sent) (2/1) Serv. idle.

Address: ◯ H [✓ D] I/O Holding Regs (40000(Fmt: decimal +/- Prot: MODBUS TCF [ ] Clone [ ] Log

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	0	0	0	0	0	0	0	0	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	9	0	0	0
400031-400040	0	0	0	0	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0	0	0	0	0
400051-400060	0	0	0	0	0	0	0	0	0	0
400061-400070	0	0	0	0	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0	0	0	0	0
400081-400090	0	0	0	0	0	0	0	0	0	0

Ver. 8.21.2.7

Figure 5.5: Modbus TCP-IP Server Simulator (TEACH Mode Command)

The Teach Mode is useful to choose a specific point in the cartesian plane to reach by the Robot’s end-effector: after having pronounced the word “Modalità Teach” (Teach Mode) the program write the Holding Register #26 with the number 9, waiting for a new range value (in millimeters).

5.2 CIRCUIT AND MOBILE APP WORKING TEST

The same already done tests must now be performed on the circuit implementation and the uploaded mobile app.

- Movement to the rightward direction about 76 mm

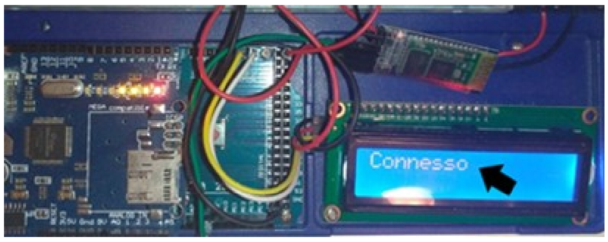


Figure 5.6: Microcontroller (Rightward Movement 1)

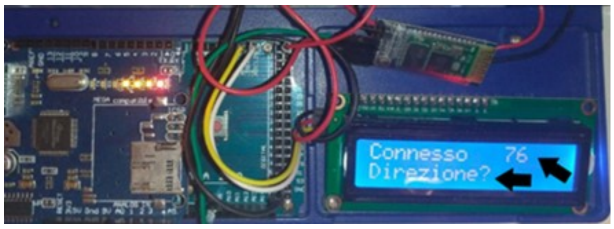


Figure 5.7: Microcontroller (Rightward Movement 2)

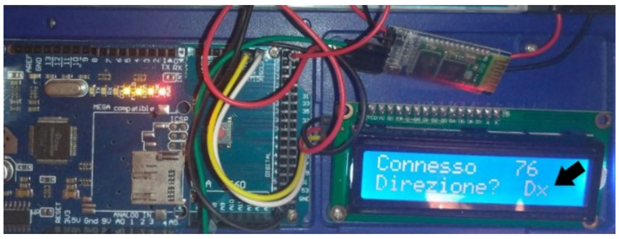


Figure 5.8: Microcontroller (Rightward Movement 3)

The program starts and the Microcontroller tries to establish a connection with the Modbus TCP server (virtual or real simulator): when correctly connected, the word “Connesso” (Connected) will be printed on the LCD. Now, to perform a movement, it is firstly required to specify the precise range value (in millimeters): such value will appear and, after a bit, the word “Direzione?” (Direction?) will be printed on the LCD. Finally, it must be declared the direction and a pair of letters (Dx = right) will appear on the LCD.

- STOP Command

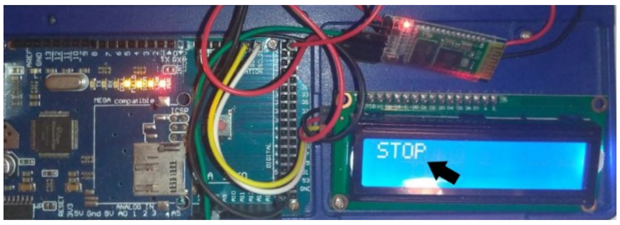


Figure 5.9: Microcontroller (STOP Command)

To perform a Stopping action, after having pronounced the related word, the word "STOP" will be printed on the LCD.

- TEACH Mode Command

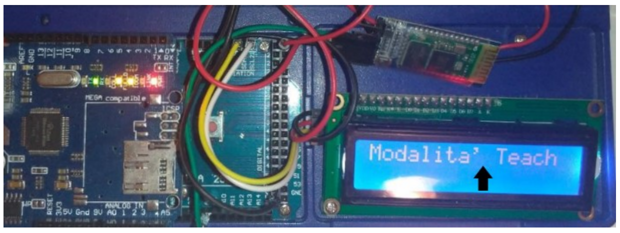


Figure 5.10: Microcontroller (TEACH Mode Command)

To enter in TEACH Mode, after having pronounced the related word, the words "Modalità Teach" (Teach Mode) will be printed on the LCD.

- RESET Command

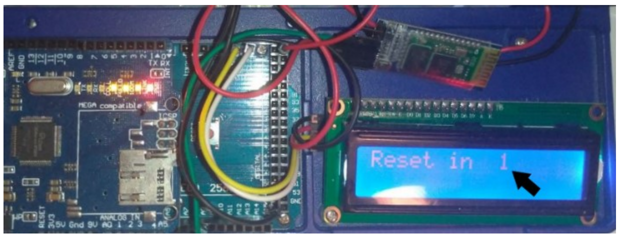


Figure 5.11: Microcontroller (RESET Command)

To perform a RESET action, after having pronounced the related word, the words "Reset in" will appear on the LCD, followed by a countdown from 3 to 0.

At the end of the count, the microcontroller will self-reset from the very begging, trying again to connect to the server.

5.3 FINAL TESTS AND RESULTS WITH REAL ROBOT

To perform the last and most important tests, the real Rhea Prime Cartesian Robot was settled down on a fixed structure surrounded by empty space to be easily controlled, avoiding accidental crashes with possible adjacent surfaces.

The same main tasks already performed with the TCP Modbus simulator and the circuit implementation are now tested on this real Robot settlement:

- Movement to the rightward direction about 600 mm

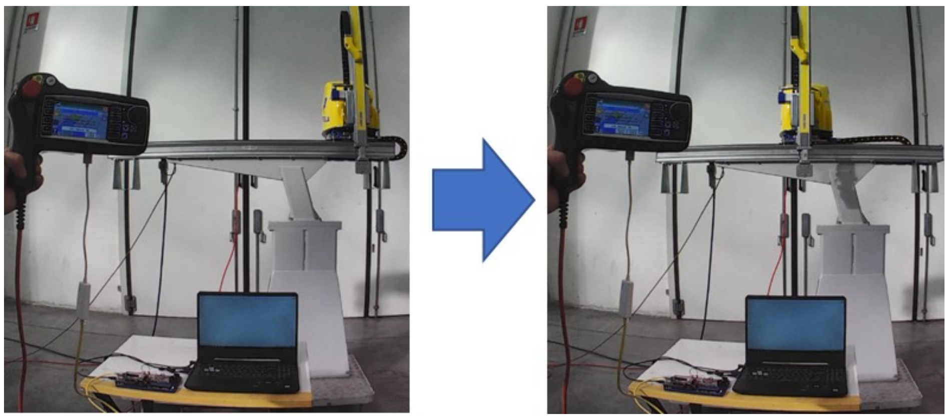


Figure 5.12: Real Robot (Rightward Movement)

- Movement to the downward direction about 200 mm

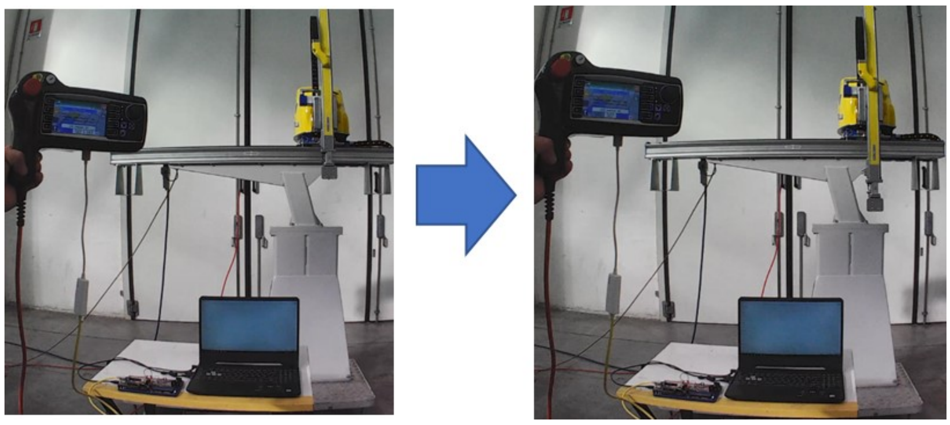


Figure 5.13: Real Robot (Downward Movement)

By directly performing movements, the starting point of the Robotic Cycle (the origin by default) will change according to the new chosen coordinates: once putted in Auto Mode in fact, the Robot will perform the cycle reaching the arrival point and coming back to the new starting point.

- TEACH Mode Command

Due to Robot internal safety reasons, once entered in Teach Mode, it is possible to give the Teach confirmation command just manually on the Control Unit Pad, after having inserted and turned a specific key: this command was therefore vocally avoided and every time given manually.

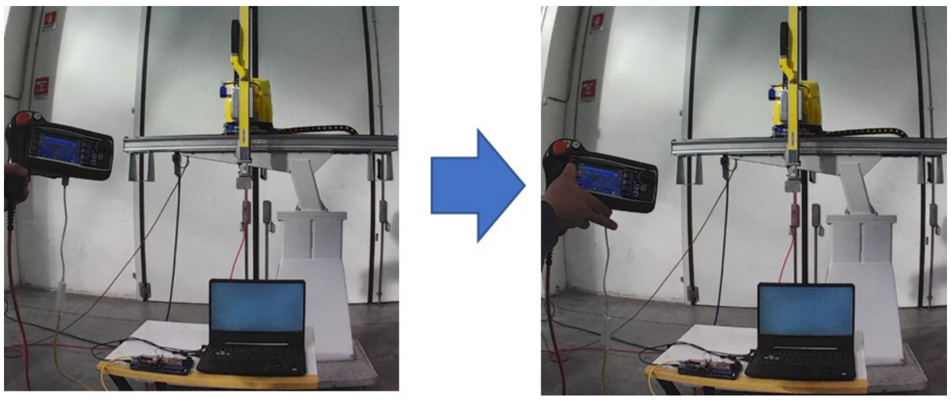


Figure 5.14: Real Robot (Manual TEACH Command Confirmation)

By giving movements commands in Teach Mode, the arrival point of the Robotic Cycle will change according to the new chosen coordinates: once putted in Auto Mode in fact, the Robot will perform the cycle reaching the new arrival point and coming back to the previous starting point.

- STOP Command

Eventually, the last tested task was the ability of the vocal control to promptly perform a STOP Emergency Command: unfortunately, after having pronounced the command word, the vocal App requires approximately 2 seconds to call the vocal assistant, convert the word into text and send it; furthermore, the microcontroller requires approximately 1 second to interpretate the command and write the related holding register. In total, a 3 seconds delay is required to effectively see the Robot stopping, after the given command: this is not conformed with the readiness which should characterized such an Emergency Command (maybe given in a dangerous situation).

CONCLUSIONS AND POSSIBLE APPLICATIONS

In this treatise, a vocal control developing, implementing and building procedure for an industrial cartesian Robot was discussed. It consisted in:

1. Vocal control settings on microcontroller and respective Robot programming modification to establish a proper Modbus communications protocol

2. Real configuration building (code, circuit and connections)

3. Final Tests on code, circuit and real Robot

To maintain the typical Cartesian Robot characteristics, the following main aspects should have been respected:

- Safety

  In industrial installations, Cartesian Robots are always working in safety conditions: the workspace is settled to be far from human presence (often due to physical barriers), with possibility of Stop and Emergency safety Commands to block the Robot in case of necessity.
  For the final tests on the real Robot, the same safety conditions were respected, with the addition of a further safer and faster way to block the Robot: human voice.

- Fast Response

  Cartesian Robots are characterized by a very fast response to a given command (centesimals of seconds); final tests showed that the built architecture requires around three seconds to perform a Stop Command action: this amount can be accepted to send movement commands but can't be accepted for Emergency ones where immediacy is essential.
  This result is mainly due to the Android integrated Google assistant, that requires around two seconds to understand that the user ended to speak, then interpretate the pronounced words and finally send them to the

microcontroller (very fast to elaborate received commands). The choice of a faster vocal assistant can speed up the acquiring process and therefore, the total interpretation speed.

- Accuracy

  Regarding industrial Robots, Cartesian Ones are certainly the most precise with a tolerance in the range of micrometers (µm); for convenience reasons, the Robot programming modifications were made to receive and perform movements in the order of millimeters (mm) but, acting directly on holding registers, the Robot would have anyway respected the same accuracy, even with micrometric movements commands.

A Man-Robot voice interaction proved to be a promising method for Industrial Robots trajectory planning in the next future. From a human point of view, this communicating typology is the most natural and intuitive: as we can see in many other fields (automotive for example), this type of vocal functionality is already available. In an industrial settlement, this solution can provide to operators an easier, faster and closer communication with machines (just like in every other application where Man-Robot cooperation is essential).

In conclusion, the proposed project showed that, even with a not so complex software and hardware implementation, it is practically possible to interact through voice commands with an industrial Robot that is normally configured as an automatic machine (fixed cycle): these results, open to the possibility of switching from the typical Robot Control Override (via Digital Control Unit) to a vocal interface that allows the user to be free handed and use just short words to give direct commands, such as the Emergency ones (Stop/Reset) that, in a situation of danger for example, are required with a certain readiness.

Furthermore, for what concerns the Smart Industry, the possibility of linking together all the machines present in an industrial settlement can make communications even faster, with the chance of vocally controlling not just a single Robot, but all the entire automatic production, further narrowing the boundary between humans and machines.

BIBLIOGRAPHY

[1] *https://en.wikipedia.org/wiki/Fourth_Industrial_Revolution*

[2] *Lin, I. H., Liu, C. Y., Chen, L. C., "Evaluation of Human-Robot Arm Movement Imitation", Proceedings of 8th Asian Control Conference (ASCC), pp.287-292, 2011*

[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7664672

[4] https://en.wikipedia.org/wiki/Industrial_robot

[5] International federation of robotics (IFR), "World Robotics", Switzerland: United Nation, 2005

[6] Kabuka, R. M., Glaskowsky, N. P., Miranda, J., "Microcontroller-Based Architecture for Control of a Six Joints Robot Arm", IEEE Transactions on Industrial Electronics, Issue 2, Vol. 35, pp.217-221, 1988

[7] Kuttan, A., "Robotics", India: I.K. International publishing house, 2007

[8] Saha, S. K., "Introduction to Robotics", India: Tata McGraw-Hill publishing company Ltd., 2008

[9] Bagad, V. S., "Mechatronics" India: Technical Publication Pune, 2008

[10] Roa, P. N., "CAD/CAM Principles and Applications 3rd Edition", India: Tata McGraw Hill Education private Ltd., 2010

[11] Nof, Y. S., "Handbook of industrial robotics, volume 1", Canada: John Wiley & Sons, 1999

[12] Merlet, J. P., "Parallel Robots: Second Edition", Netherlands: Springer, 2006

[13] https://control.com/technical-articles/what-are-cartesian-robots

[14] Sanchez, S. P., Cortes, R. F., "Cartesian Control for Robot Manipulators, Robot Manipulators Trends and Development", 2010

[15] Arian Faravar, "Design, Implementation and Control of a Robotic Arm Using PIC 16F877A Microcontroller", 2014

[16] https://www.campetella.com/en/cartesian-top-entry-robots/rhea-prime

[17] https://en.wikipedia.org/wiki/Modbus

[18] https://www.arduino.cc/reference/en/libraries/arduinomodbus

[19] https://appinventor.mit.edu

APPENDIX

FINAL BUILT CODE

```
#include <Ethernet.h>
#include <ArduinoModbus.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>


void(* resetFunc) (void) = 0;

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xE8 }; // Ethernet MAC Address
IPAddress ip(192,168,1,113); // Ethernet MAC IP address

EthernetClient client;
ModbusTCPClient modbusTCPClient(client);

IPAddress server(192,168,1,4); // update with the IP Address of your Modbus server

const int RxPin = A8;  // Arduino Receiver on Pin A8
const int TxPin = 52;  // Arduino Transmitter on Pin 52
SoftwareSerial mySerial(RxPin, TxPin);

String data;
int num;

LiquidCrystal_I2C lcd(0x27, 16, 2);  // LCD display with 16 Columns & 2 Raws

void setup() {

   lcd.begin();
   lcd.backlight();
   Serial.begin(9600);
   mySerial.begin(9600);
   Serial.println("");

   Ethernet.begin(mac, ip);
   Serial.print("");

 delay(3000);


 Serial.print("connecting to ");
 Serial.print(server);
 Serial.println("");
 Serial.print("IP = ");
 Serial.print(Ethernet.localIP());
 Serial.println("");

 client.connect(server, 502);

 if (client.connect(server, 502)) {
   Serial.print("connected to ");
   Serial.println(client.remoteIP());
   lcd.clear();
   lcd.print("Connesso");
   client.println("Connection: close");
   client.println();
 } else {
   Serial.println("connection failed");
   lcd.clear();
   lcd.print("Non connesso");
 }

   Ethernet.begin(mac, ip);  // initialize Ethernet device
   delay(2000);
   modbusTCPClient.begin(server, 502);
   Serial.println("");
   delay(1000);
}
```

```
void loop() {

    data = "";
    if (mySerial.available()>0) {
    data = mySerial.readString();
    Serial.println(data);


  if (isDigit(data[0]))
   {
     num = data.toInt();
     lcd.setCursor(11,0);
     lcd.print("      ");
     lcd.setCursor(11,0);
     lcd.print(num);
     //Serial.println(num);

     modbusTCPClient.holdingRegisterWrite(27, num);

     delay(1000);
     lcd.setCursor(0,1);
     lcd.print("Direzione?");
   }

  else if(data=="destra")
    {
  modbusTCPClient.holdingRegisterWrite(26, 1);
  Serial.println("ok destra");
  lcd.setCursor(12,1);
  lcd.print("    ");
  lcd.setCursor(12,1);
  lcd.print("Dx");
    }


else if(data=="sinistra")
  {
modbusTCPClient.holdingRegisterWrite(26, 2);
Serial.println("ok sinistra");
lcd.setCursor(12,1);
lcd.print("    ");
lcd.setCursor(12,1);
lcd.print("Sx");
  }

 else if(data=="avanti")
  {
modbusTCPClient.holdingRegisterWrite(26, 3);
Serial.println("ok avanti");
lcd.setCursor(12,1);
lcd.print("    ");
lcd.setCursor(12,1);
lcd.print("Av");
  }

  else if(data=="indietro")
  {
modbusTCPClient.holdingRegisterWrite(26, 4);
Serial.println("ok indietro");
lcd.setCursor(12,1);
lcd.print("    ");
lcd.setCursor(12,1);
lcd.print("In");
  }

  else if(data=="basso")
  {
modbusTCPClient.holdingRegisterWrite(26, 5);
Serial.println("ok basso");
lcd.setCursor(12,1);
lcd.print("    ");
lcd.setCursor(12,1);
lcd.print("Bx");
  }
  else if(data=="alto")
  {
modbusTCPClient.holdingRegisterWrite(26, 6);
Serial.println("ok alto");
lcd.setCursor(12,1);
lcd.print("     ");
lcd.setCursor(12,1);
lcd.print("Ax");
  }

  else if(data=="stop")
  {
modbusTCPClient.holdingRegisterWrite(26, 7);
Serial.println("ok stop");
lcd.clear();
lcd.print("STOP");
  }

  else if(data=="jog")
  {
modbusTCPClient.holdingRegisterWrite(26, 8);
Serial.println("ok jog");
lcd.clear();
lcd.print("Jog confermato");
  }

  else if(data=="modalità teach")
  {
modbusTCPClient.holdingRegisterWrite(26, 9);
Serial.println("ok teach mode");
Serial.println("Quanti mm?");
lcd.clear();
delay(500);
lcd.print("Modalita' Teach");
  }


//    else if(data=="conferma teach")
//    {
//  modbusTCPClient.holdingRegisterWrite(26, 10);
//  Serial.println("ok conferma teach");
//  lcd.clear();
//  lcd.print("Teach confermato");
//    }

//    else if(data=="Annulla teach")
//    {
//  modbusTCPClient.holdingRegisterWrite(26, 11);
//  Serial.println("ok annulla teach");
//  lcd.clear();
//  lcd.print("Teach annullato");
//    }

    else if(data=="step")
    {
  modbusTCPClient.holdingRegisterWrite(26, 12);
  Serial.println("ok step mod");
  lcd.clear();
  lcd.print("Modalita' Step");
    }

    else if(data=="auto")
    {
  modbusTCPClient.holdingRegisterWrite(26, 13);
  Serial.println("ok auto mod");
  lcd.clear();
  lcd.print("Modalita' Auto");
    }

  else if(data=="reset")
    {
     lcd.clear();
     lcd.print("Reset in ");
     for (int i = 3; i>=0 ; i--){
     lcd.setCursor(10,0);
     lcd.print(i);
     delay(1000);
     }
     resetFunc();
    }
  }
  }
```

UNMODIFIED ROBOT NATIVE CODE (STANDARD AXIAL JOG) [Property of “Campetella Robotic Center S.R.L.”]

```
// *****************************************************************************************************
// -------------------------------- Axial JOG movements through “AbsExecute comand --------------------
// *****************************************************************************************************

                IF AsseInTeach >0 AND NOT GoingToTRJT THEN
                        IF Assiif[AsseInTeach].Input.AbsExecute THEN
                                Assiif[AsseInTeach].Input.AbsExecute:=FALSE;

                                // Self-held to avoid reset if more than one axis moves

                                Z_Axis_Moved:=Z_Axis_Moved   OR   (AsseInTeach.0   AND
(NOT( AsseInTeach.1)));
                                X_Axis_Moved:=X_Axis_Moved   OR   (AsseInTeach.1   AND
(NOT( AsseInTeach.0)));
                                Y_Axis_Moved:=Y_Axis_Moved   OR   (AsseInTeach.0   AND
AsseInTeach.1);
                        END_IF


                        IF asseInMovimento[AsseInTeach] AND (NOT(common.KeysF[6].clk))
AND (NOT(common.KeysF[7].clk))   THEN
                          //assiif[AsseInTeach].Par.StopDec                              :=
assiif[AsseInTeach].Par.AbsDec*10;
                                Assiif[AsseInTeach].Input.CmdStop:=TRUE;
                                IF tomdelay > DERATA_MIN_STEP then   //30 THEN  7ms
                                        OK_NuovoStart := TRUE;
                                ELSE
                                        OK_NuovoStart := FALSE;
                                        tomdelay := tomdelay +1;
                                END_IF
                        END_IF

                        IF common.KeysR[6].Q  AND OK_NuovoStart THEN   //premuto PIU
                                Assiif[AsseInTeach].Input.CmdStop:= FALSE;
                                Assiif[AsseInTeach].Par.AbsVel:=
AssiIf[AsseInTeach].Par.Vel_Max              *              ((UINT_TO_REAL(assiPar[
AX_VELMAN].valPar[AsseInTeach]) / 100)) *(JogTeach /100);


                                IF tipoMeccanica =  EVO_ROBOT THEN
                                        Assiif[AsseInTeach].Par.AbsAcc:=
AssiIf[AsseInTeach].Par.Acc_Max * (JogTeach /100) * 10;
                                ELSE
```

```
                                        Assiif[AsseInTeach].Par.AbsAcc:=
AssiIf[AsseInTeach].Par.Acc_Max * (JogTeach /100);
                                        END_IF

                                        Assiif[AsseInTeach].Par.AbsPos:=
assiPar[AX_CORSAMAX].valPar[asseInTeach]/10;
                                        Assiif[AsseInTeach].Input.AbsExecute:=common.KeysR[6].Q;
                                        tomdelay := 1;
asseInMovimento[AsseInTeach] := TRUE;
                              END_IF


                                       IF common.KeysR[7].Q  AND OK_NuovoStart THEN//premuto
MENO
                                        Assiif[AsseInTeach].Input.CmdStop:= false;
                                        Assiif[AsseInTeach].Par.AbsVel:=
AssiIf[AsseInTeach].Par.Vel_Max              *               ((UINT_TO_REAL(assiPar[
AX_VELMAN].valPar[AsseInTeach]) / 100)) *(JogTeach /100);



                                        IF tipoMeccanica =  EVO_ROBOT THEN
                                                  Assiif[AsseInTeach].Par.AbsAcc:=
AssiIf[AsseInTeach].Par.Acc_Max * (JogTeach /100) * 10;
                                        ELSE
                                                  Assiif[AsseInTeach].Par.AbsAcc:=
AssiIf[AsseInTeach].Par.Acc_Max * (JogTeach /100);
                                        END_IF
                                        IF RobotStatus = def.MANUAL THEN // can't go under 0
                                          Assiif[AsseInTeach].Par.AbsPos:= 0;
                                        ELSE
                                          Assiif[AsseInTeach].Par.AbsPos:=                      -
(UINT_TO_INT(assiPar[AX_CORSAMAX].valPar[asseInTeach]/10));
                                        END_IF
                                        Assiif[AsseInTeach].Input.AbsExecute:=common.KeysR[7].Q;

                                        tomdelay := 1;
                                        asseInMovimento[AsseInTeach] := TRUE;
                              END_IF

                    END_IF

END_IF
```