

Politecnico di Torino

Master's degree in Mechatronic Engineering

A.a. 2020/2021

Sessione di Laurea Luglio 2021



**Politecnico
di Torino**

**Creation and simulation of an
autonomous warehouse model
using a logic controller and OPC
communication**

Relatore:

Luigi Mazza

Candidato:

Roberto Serio

INTRODUCTION

This thesis stems from the desire to simulate and recreate a renovated laboratory bench in a virtual reality. Subsequently modify it again by expanding its functions.

In order to do this, different software have been used. In particular “Automation Studio”, by Famic, has been explored extensively in order to exploit its potential to the maximum and fill the weaknesses with other software.

In this thesis the process of creating the various simulation files is described, deepening some aspects to understand them better.

The first part of thesis describes the recreation phase of the existing model in SolidWorks, with explanation of the movements.

In the second part, various bench modifications via CAD are described with the new desired behavior. After which, the following aspects are described: the 2D simulation processes in Automation Studio, the 3D Simulation process in Automation Studio and finally the most real simulation with the help of Unity3D.

Finally, the logic of C# in Unity programming and ladder logic is described.

The goal of this thesis is to obtain the most accurate simulation with the best method, which allows the control of real-physical part and the simulated part.

INDEX

INTRODUCTION	2
1. Preface	4
1.1. Programmes.....	4
Automation Studio	4
SIMATIC	4
TIA PORTAL	5
S7-PLCSIM	5
KEPServerEX 6	5
NetToPLCSim	5
Adobe Photoshop	6
Unity3D	6
Visual Studio.....	6
1.2. Ladder Logic	7
1.3. AUTOMATION STUDIO PRESENTATION & LADDER LOGIC.....	8
1.4. Playful example.....	9
2. REAL BEANCH.....	13
2.1. Project analysis.....	14
CAD	14
2.2. Automation Studio 2D	17
Considerations	19
2.3. 3D manager	19
3. CAD CHANGES	21
3.1. Final version.....	27
3.2. Parts and sensors	28
3.3. Desired behaviour of the model	30

3.4.	VARIABLES:	39
3.5.	Lists of necessary theoretical variables:.....	44
4.	Unity	49
4.1.	References	50
	Scenes	51
	Hierarchical relationships	51
4.2.	Objects.....	52
	Objects> centre	52
4.3.	RigidBody	53
4.4.	Collider.....	54
4.5.	Materials and Physical Materials	56
	Material.....	56
	Physical material.....	56
4.6.	Other key elements.....	56
5.	Scripts	57
5.1.	Movement2	57
5.2.	Movement1 with Led	60
5.3.	MovimentoVacuum.....	62
5.4.	Rotation.....	63
5.5.	Deviation wall appearance	65
5.6.	Optical sensors simulation.	66
5.7.	Potentiometric sensors simulation.....	68
5.8.	Conveyor.....	69
5.9.	Polytechnic's symbol, moulding	70
5.10.	Suction Cup simulation	71
5.11.	SW	72

5.12.	Cameras	73
5.13.	Camera movement.....	75
5.14.	Random Spawn.....	76
5.15.	AS and Unity connection.....	78
5.16.	Unity variables/AS	79
5.17.	PUSHBUTTON Unity/AS	83
5.18.	DEBUG	84
5.19.	Variables create:	85
6.	Siemens.....	88
6.1.	Grafcet.....	88
6.2.	TiaPortal.....	90
7.	OPC.....	94
7.1.	NetToPLCsim	94
7.2.	KEPServerEx6	95
7.3.	Automation studio.....	95
8.	Conclusion	96
8.1.	Diagram.....	96
8.2.	Advantages.....	97
8.3.	Drawbacks.....	97
9.	Bibliography	98

1. Preface

This chapter presents the programmes used and summarises the basics.

1.1. Programmes

Automation Studio

Automation Studio is a circuit design, simulation and project documentation software for fluid power systems and electrical project conceived by Famic Technologies Inc.

It is used for CAD, maintenance, and training purposes.

Automation Studio can be applied in the design, training and troubleshooting of hydraulics, pneumatics, HMI, and electrical control systems.

In this thesis project it is used this software to create different simulations. It is simulated: actuators and pneumatic circuits, the actions of the real project in 2D, the actions of the reworked project in 2D and 3D, PLC and ladder logic.

SIMATIC

SIMATIC is a series of programmable logic controller and automation systems, developed by Siemens. As with other programmable logic controllers, SIMATIC devices are intended to separate the control of a machine from at the machine's direct operation, in a more lightweight and versatile manner than controls hard-wired for a specific machine.

TIA PORTAL

Every automation system needs a program to control a machine. The Siemens's software is Tia Portal. Thanks to this program we can read Input (from sensors) and generate action (actuators).

In this thesis project it is used to write the final and optimized ladder.

S7-PLCSIM

In order to not be bound by a physical PLC, S7-PLCSIM is used to simulate its functions.

KEPServerEX 6

KepServerEX is a connectivity platform that provides a single source of industrial automation data. This platform allows users to connect, manage, monitor, and control diverse automation devices and software applications through one user interface via OPC.

In this thesis project it is used to connect via OPC NetToPLCSim to Automation Studio.

NetToPLCSim

Having chosen not to use a physical PLC, the S7-PLCSIM simulator is not directly compatible with KepServerEX. NetToPLCSim using different IP allows you to get around the obstacle.

In this thesis project it is used to connect via server S7-PLCSIM to KepServerEX. Thanks to these programs it is possible to connect Tia Portal to Automation Studio.

Tia Portal-S7-PLCSIM- NetToPLCSim- KepServerEX-Automation Studio.

Adobe Photoshop

Photoshop is a raster graphics editor developed and published by Adobe Inc. In this thesis project it is used to isolate component in the 2D simulation.

Unity3D

Unity is a cross-platform game engine developed by Unity Technologies.

In this thesis project it is used its engine to create a virtual reality. Thanks to it, it is possible: to recreate the fundamental physical laws and to create machine movements that can be controlled by Boolean variables.





Visual Studio

Microsoft Visual Studio is an integrated development environment from Microsoft. Visual Studio includes a code editor and support several languages like C#, C++, Java and other.

In this thesis project it is used to create the scripts in C# for Unity.

1.2. Ladder Logic

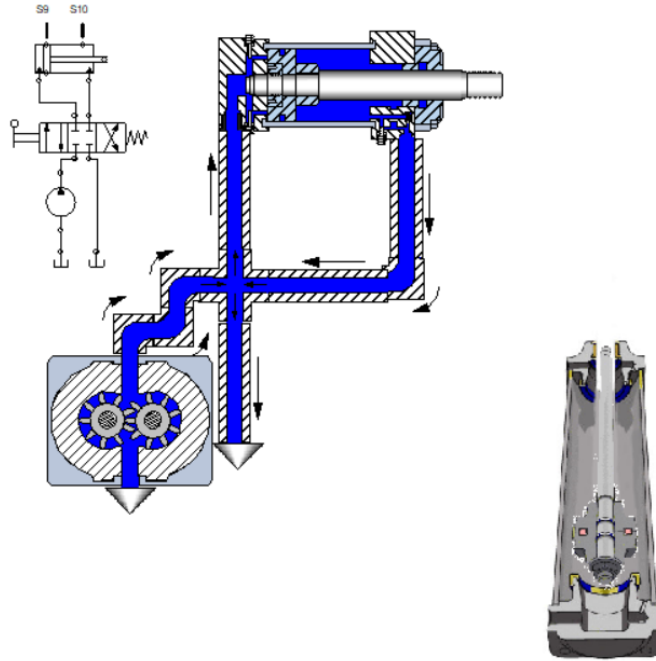
Ladder logic uses switch or relay contacts to implement Boolean expressions. It is a graphical programming language and is used to develop software for programmable logic controls (PLC) used in industrial control applications. The main programming elements are:

-  – Normally open contact, closed whenever its corresponding coil or an input which controls it is energized (send 1).
-  – Normally close contact, closed whenever its corresponding coil or an input which controls it is not energized (It is a NOT mathematician and send a 0).
-  – Normally inactive coil, energized whenever its rung is closed. (inactive at rest).
-  – Normally active coil, energized whenever its rung is closed. (active at rest)

If placed in parallel they form a mathematical OR, if in series an AND. When combined together they can form other basic blocks such as timers, counters, equalities, etc.

1.3. AUTOMATION STUDIO PRESENTATION & LADDER LOGIC

The main purpose of Automation Studio is to provide both a graphic and technical platform, to simulate and understand the behavior of what you create.

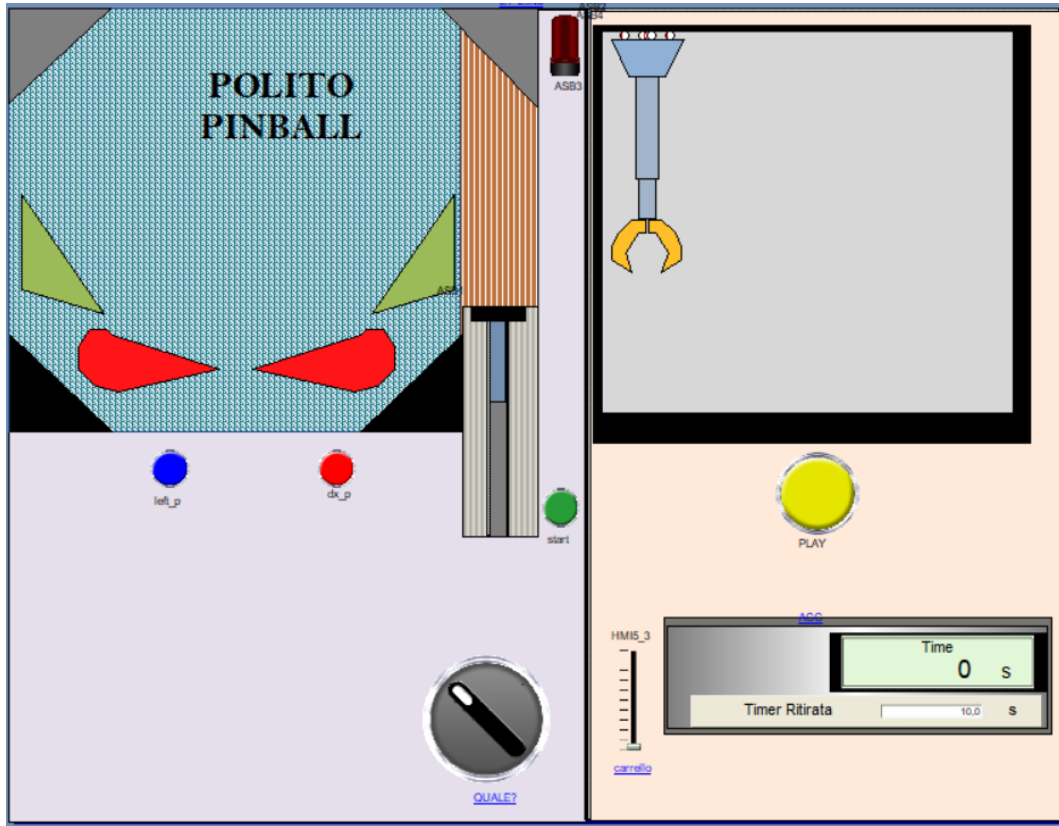


1.3.1 Figure- Cross Section of a circuit

Figure 1.3.1 is an example of functionality. There is a schematic pneumatic circuit, a “cross section” version and a 3D section. Everything is animated and linked via variables.

Other main functions used in Automation Studio are now described with some examples.

1.4. Playful example

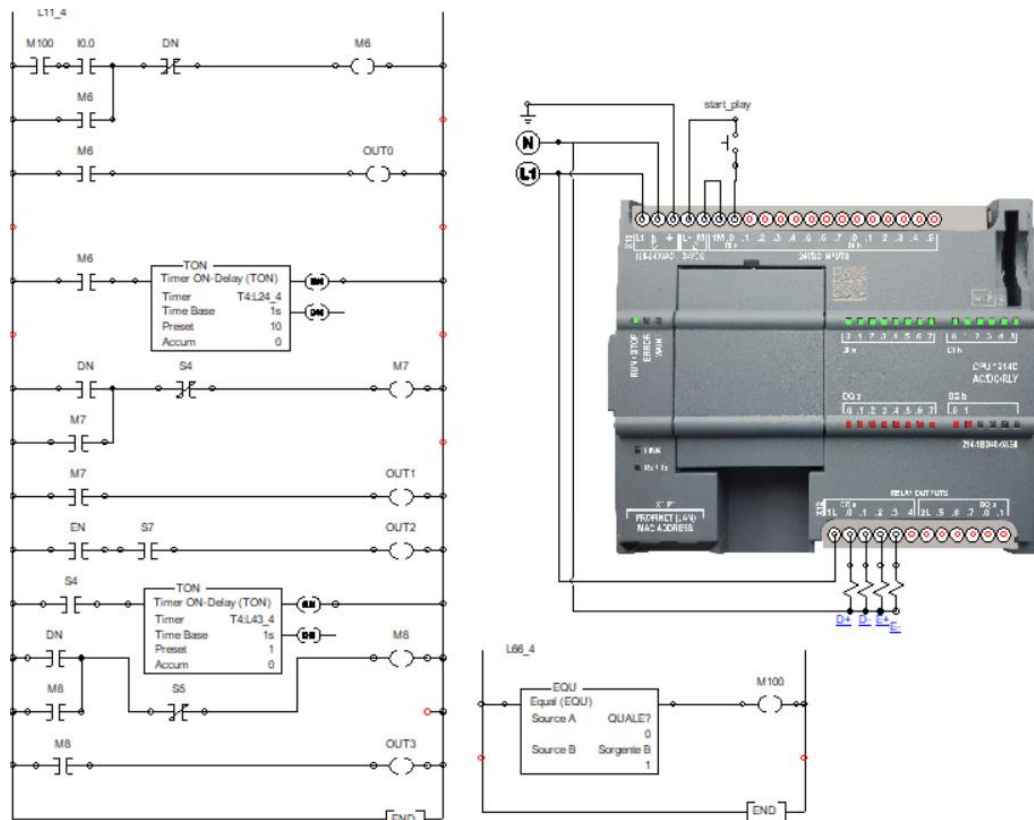


1.4.1 Figure- Example AS

This example is built using Automation Studio tools, it recalls the famous Pinball games and the hook of luck. The control HMI facade can be seen.

Thanks to this software, we can associate a Boolean variable with each input, while an actuator can be associated with each output movement.

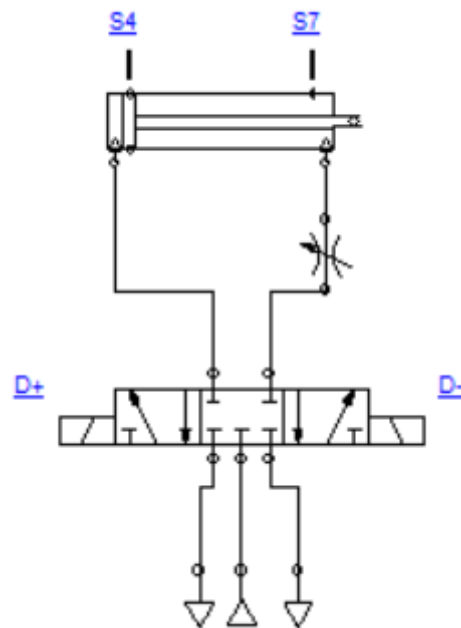
To perform the desired behavior in the graphic design, it is possible to use several options including grafcet and ladder language.



1.4.2 Figure- Example AS ladder logic

Figure 1.4.2 shows the hook ladder programming, the I/O are connected to a simulation interface of a PLC, in this case a Siemens S7-1200. Ground, power, supply, output solenoids and an input switch can be seen. Autoamtion Studio allows you to create Boolean variables associated with the HMI. In this case the yellow pushbutton is link to I0.0, if it is press AND the safety knob is turned on (in the rung on bottom right in figure 1.4.2), the sequence of actions can begin. In detail, to continue, memories are inserted to energize the wire and activate the various sequences. There are two timers to adjust the time between actions, adjustable from the panel.

Other inputs are given by the position sensors (Sn) positioned on the actuators.



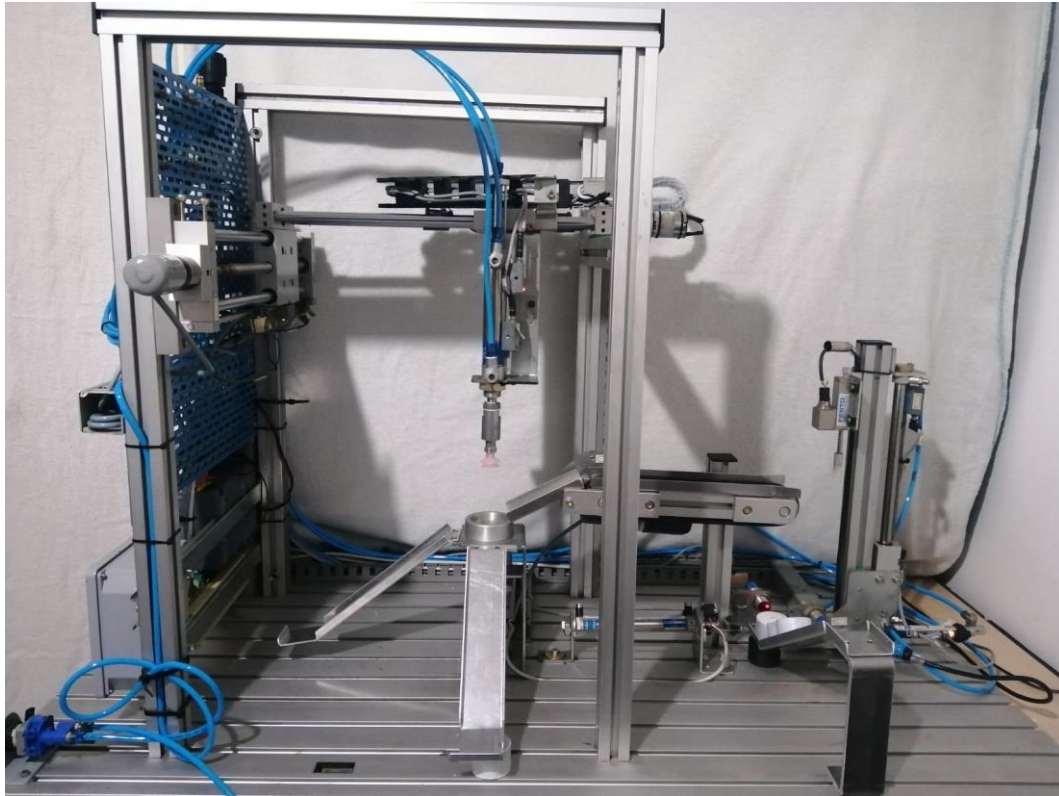
1.4.3 Figure- Example pneumatic circuit

As mentioned, the PLC outputs are connected to solenoids which activate flow valves. The pneumatic feeder powers the actuator, which carries out the movement. It is therefore possible to adjust the speed of movement either by choosing the right actuator (each feature is editable) or by inserting an adjustable butterfly valve on the pipeline.

Figure 1.4.3 also shows the positioning of some position sensors.

2. REAL BEANCH

The first part of the work consists in recreating the real model of the bench, analyzing its behaviour and simulating it.



2.0.0 Figure- real bench

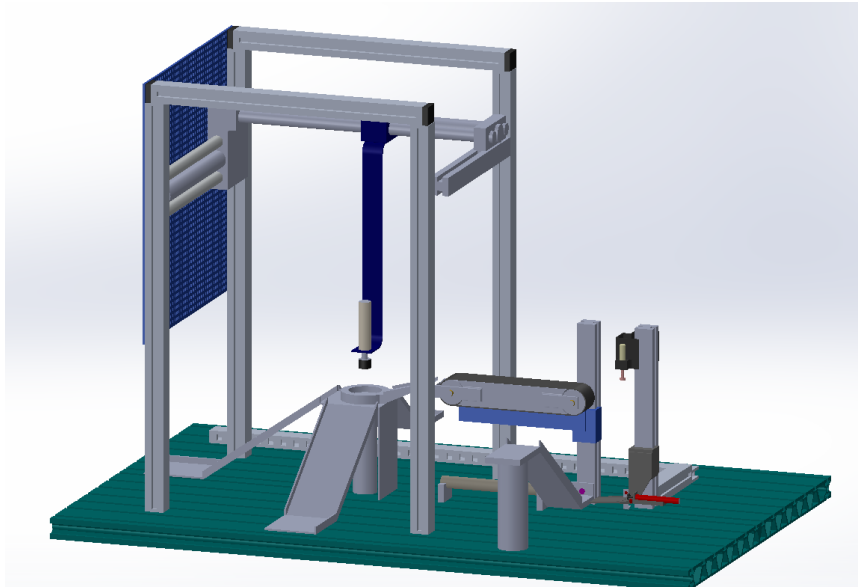
The figure 2.0.0 shows the physical bench of the laboratory renovated by engineer Caggese.

2.1. Project analysis

The main goal of the bench is to separate objects based on height and material. A position sensor ensures that cylindrical objects are in the correct position. A piston pushes the object onto a lifter which transports it to a conveyor belt. Along the way, the object is analyzed by an inductive sensor, an optical and a linear potentiometric sensor. Thanks to the information of the sensors, the piece will be positioned by a sacution cup on the right slide.

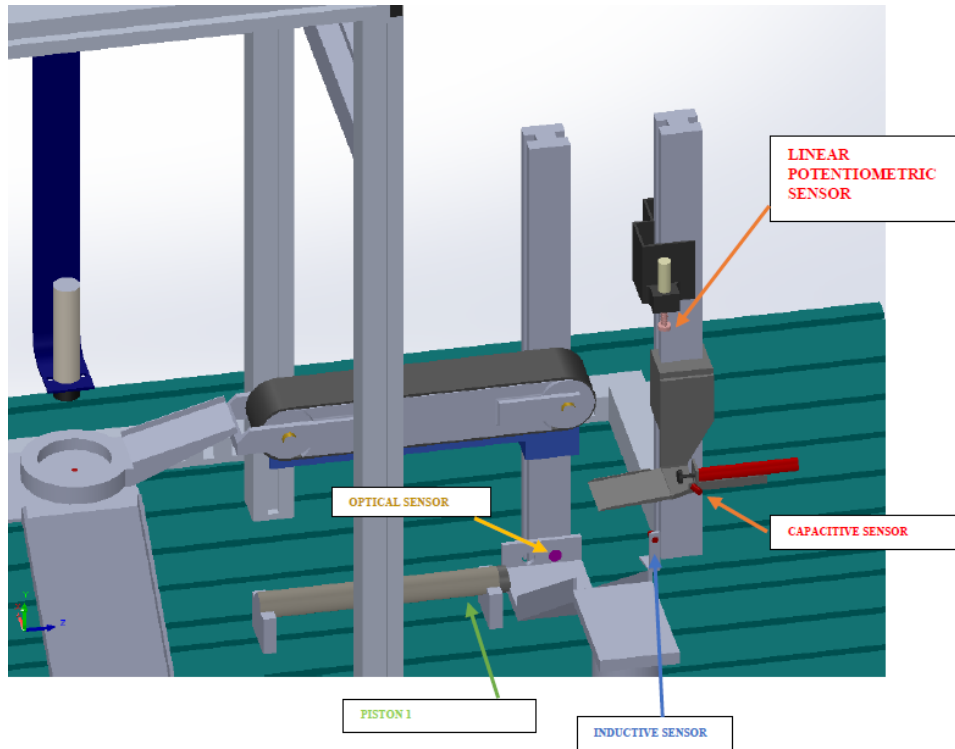
CAD

This sub-chapter contains some technical views and the 3D cad of the model.

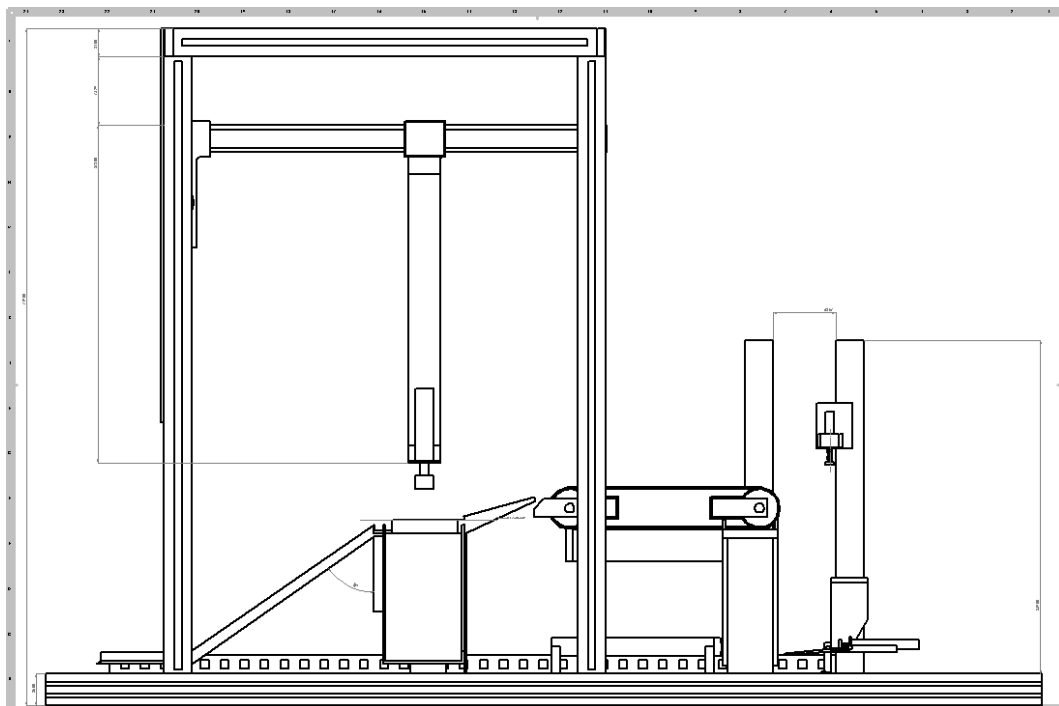


2.1.0 Figure- real bench CAD

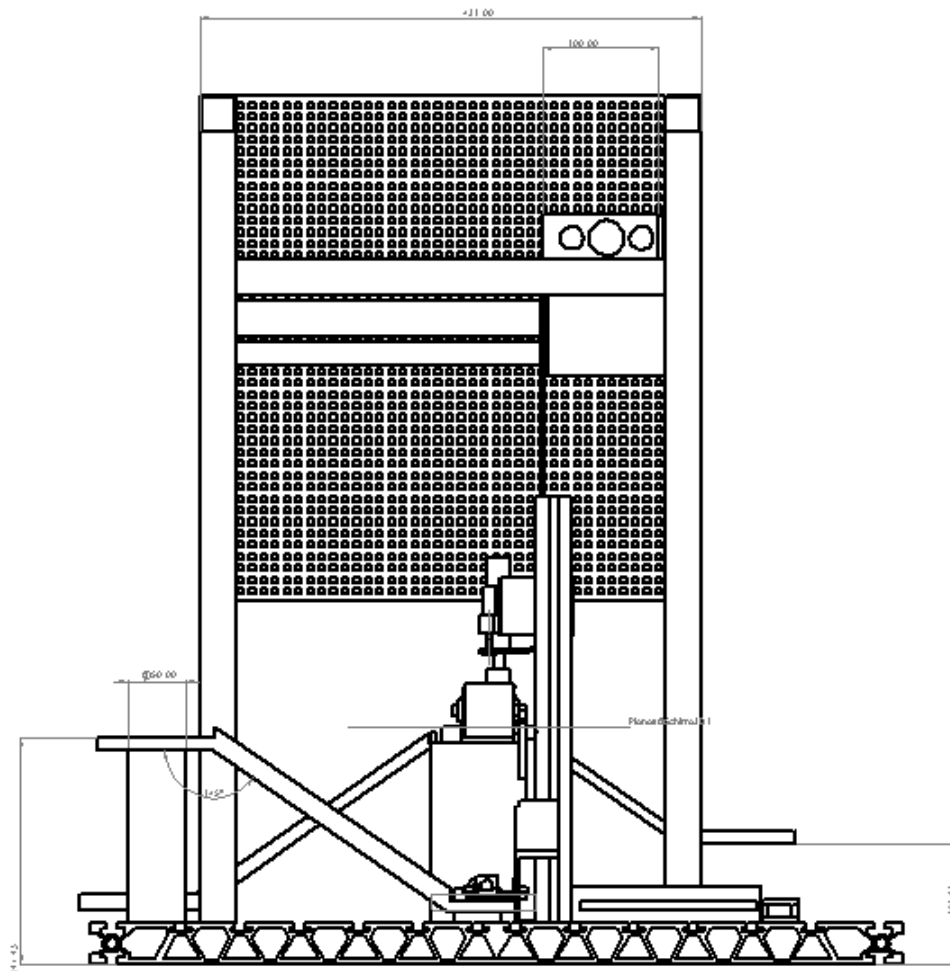
Figure 2.1.0 shows the CAD built with SolidWorks.



2.1.1 Figure- Bench's sensors



2.1.2 Figure- lateral view



2.1.3 Figure- Frontal view

2.2. Automation Studio 2D

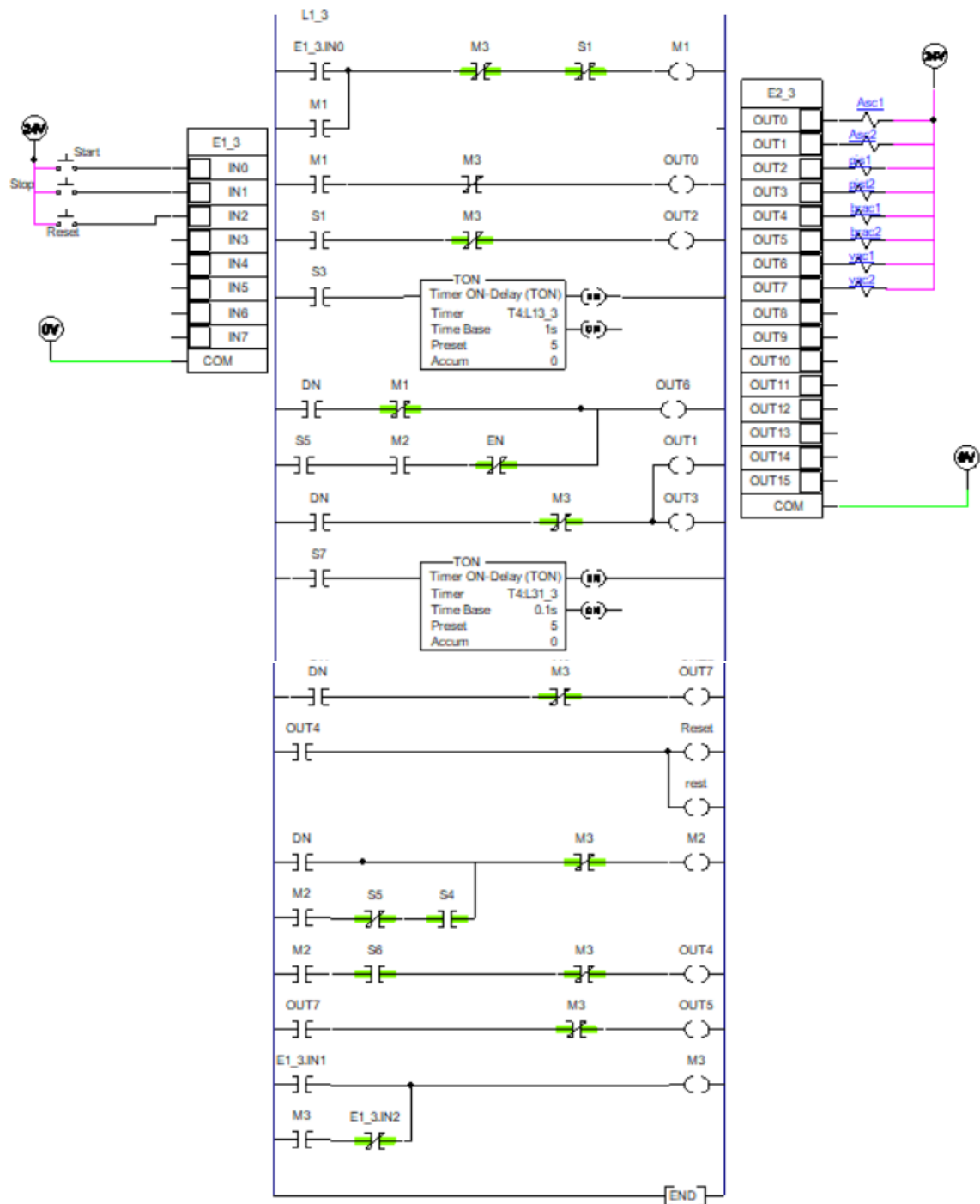
To get a first idea of the ladder programming of the project, it was chosen to perform a 2D simulation of the movements.

Automation studio allows this kind of simulation but does not have a suitable graphic engine to separate the various elements of an image. Photoshop is used to separate the different parts of the image, dividing it according to moving and fixed parts. In particular, a .png file was created for each dynamic object with the central component on a green background. Automation Studio makes it possible to eliminate a specific colour shade, resulting in only the component.



2.2.0 Figure- AS 2D

The figure 2.2.0 shows the final result with the addition of HMI control panel.



2.2.1 Figure- Ladder logic 2D

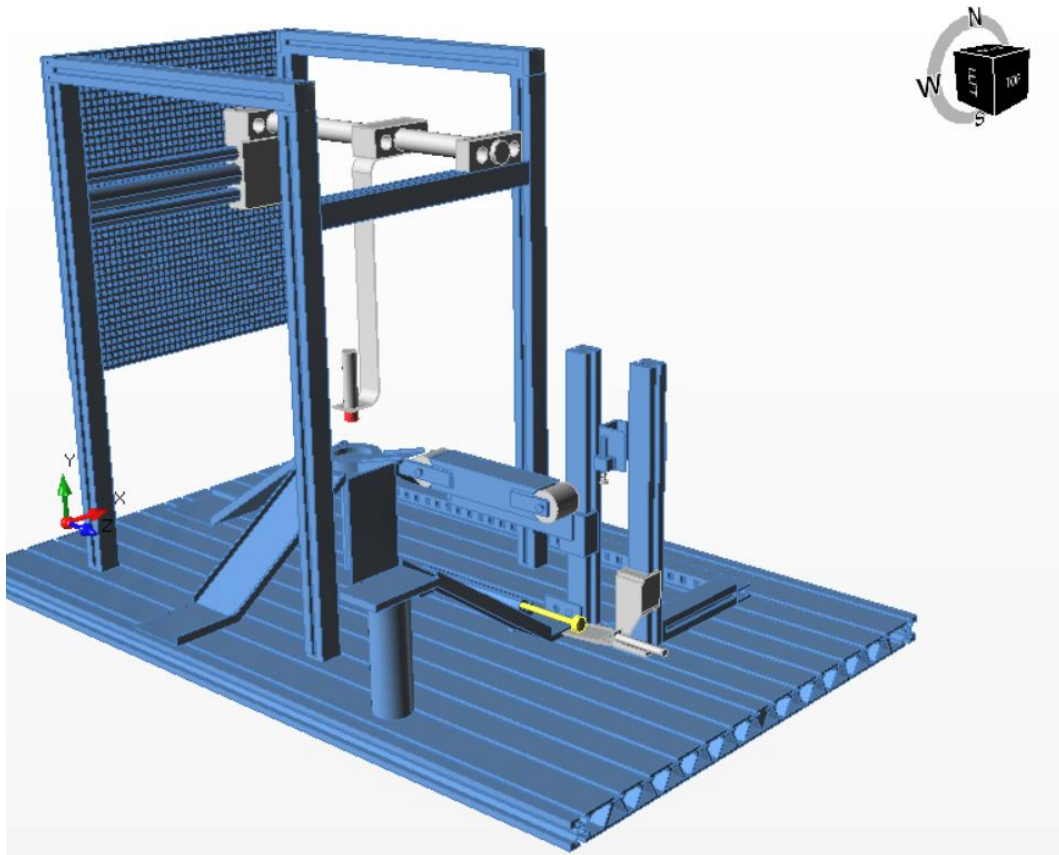
The complexity of the model has increased, it can be observed, that after pressing the “start” button a sequence starts that leads to the pushing of the piston A, raising of the lift, transport and positioning of the object.

Considerations

To get an approximate idea of machine movements, this is an appropriate tool. Although it has limitations on the recognition of the pixels to be eliminated and the obvious limitation of 2D on one dimension of motion. From this, the need to add spatiality.

2.3. 3D manager

To create a 3D model to simulate the machine, Automation Studio provides the “3D manager” tool.



2.3.0 Figure- 3D Manager

"3D Manager" allows you to import parts in .stl format. The strategy adopted to optimise the process consists of exporting all static parts from SolidWorks, in one single file. This allows a faster assembly of the components. The 3D Manager graphic system does not permanently bind the various parts but allows momentary

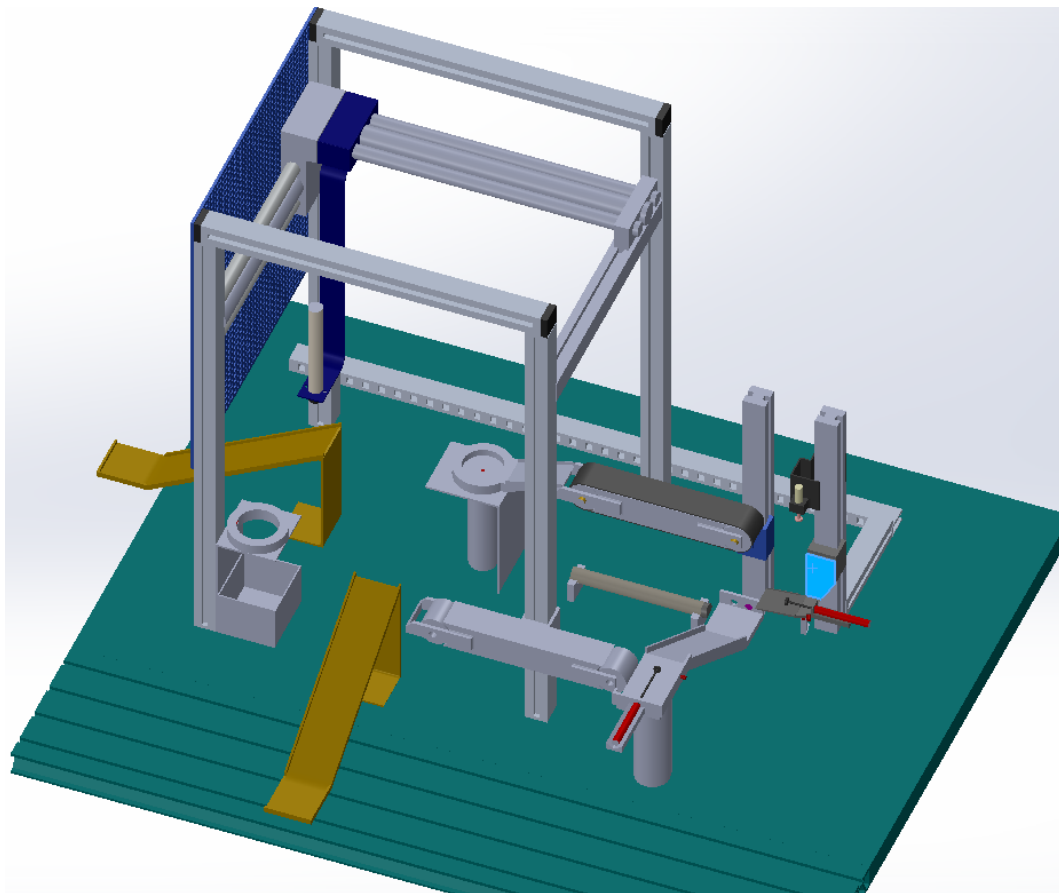
associations to orientate the components.

It is possible to create a series of constraints on movements: "carriage", "hinge" and "joint".

Pneumatic actuators can be associated with the above constraints, controlled, for example, by solenoids. it is therefore possible to create a ladder that simulates machine movements. "3D manager" is an excellent tool for simulating and understanding the spaces required by the machine, any intersections and the programming logic. A major limitation of this tool is the impossibility of simulating physics. It is not possible to simulate gravity, contact and sensors. It is not recommended to use it for accurate simulations, but only for a first evaluation. To get a better simulation, it is decided to use Unity3D to create the virtual reality of the model and then recreate the ladder programming for the desired behaviour.

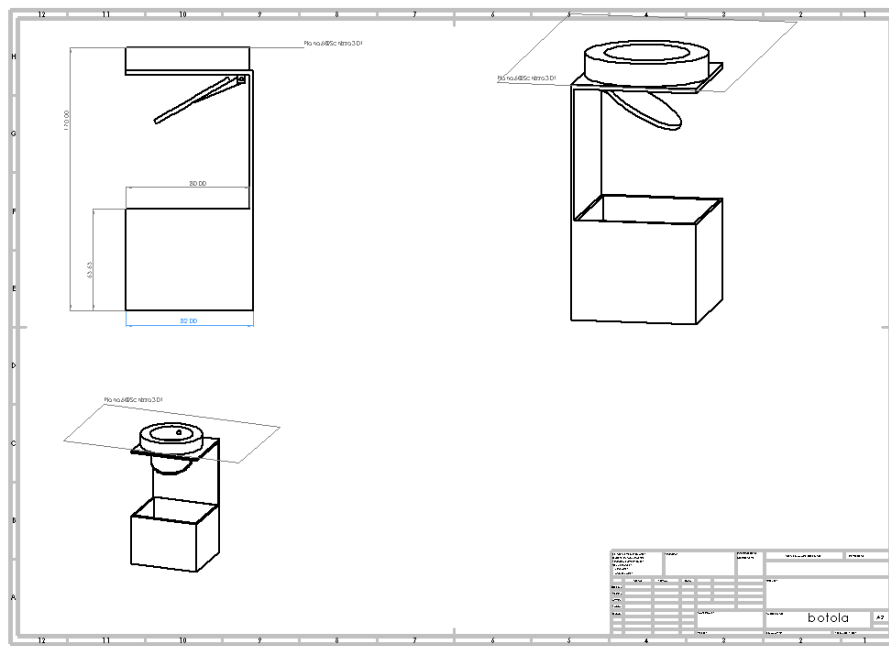
3. CAD CHANGES

Several models are created to understand which changes are really possible and which functions should be added.

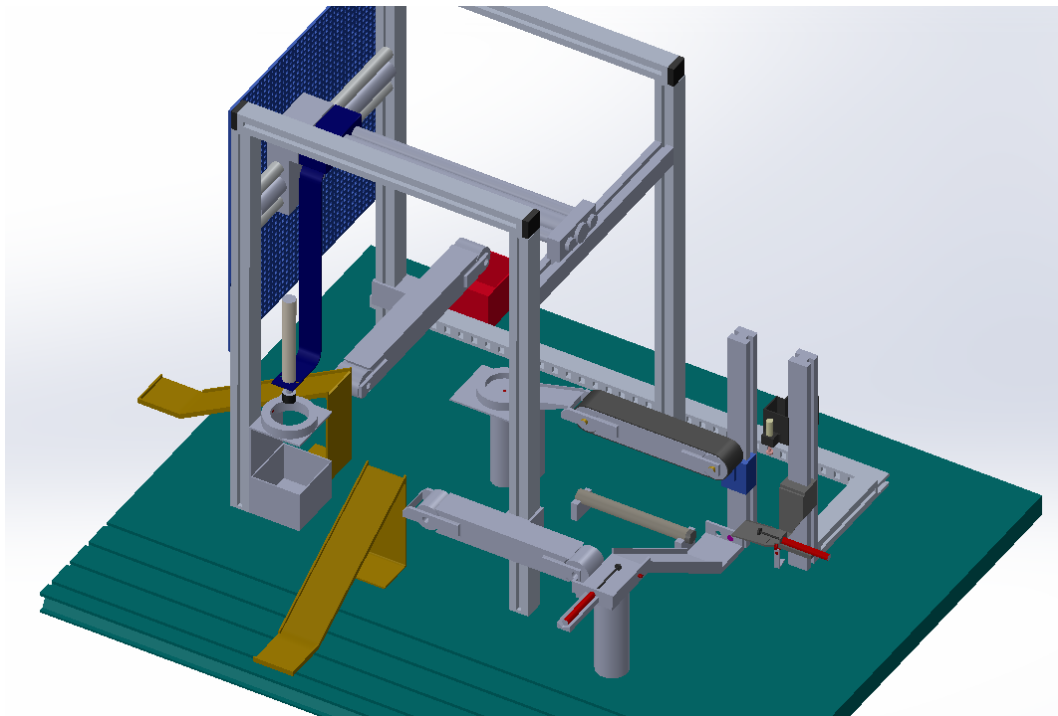


3.0.0 Figure- first remodel

In image 3.0.0, there are few changes. The slides are moved away from the piece support in order to make the best use of the area of the Cartesian arm. A hatch is added (see image 3.0.1) so that the part can be placed, assessed and stored underneath. Finally, a conveyor belt is added to load the part and a piston to push it down the chute. The downside is that the cartesian arm is not fully utilised and there is no automatic removal of the part.

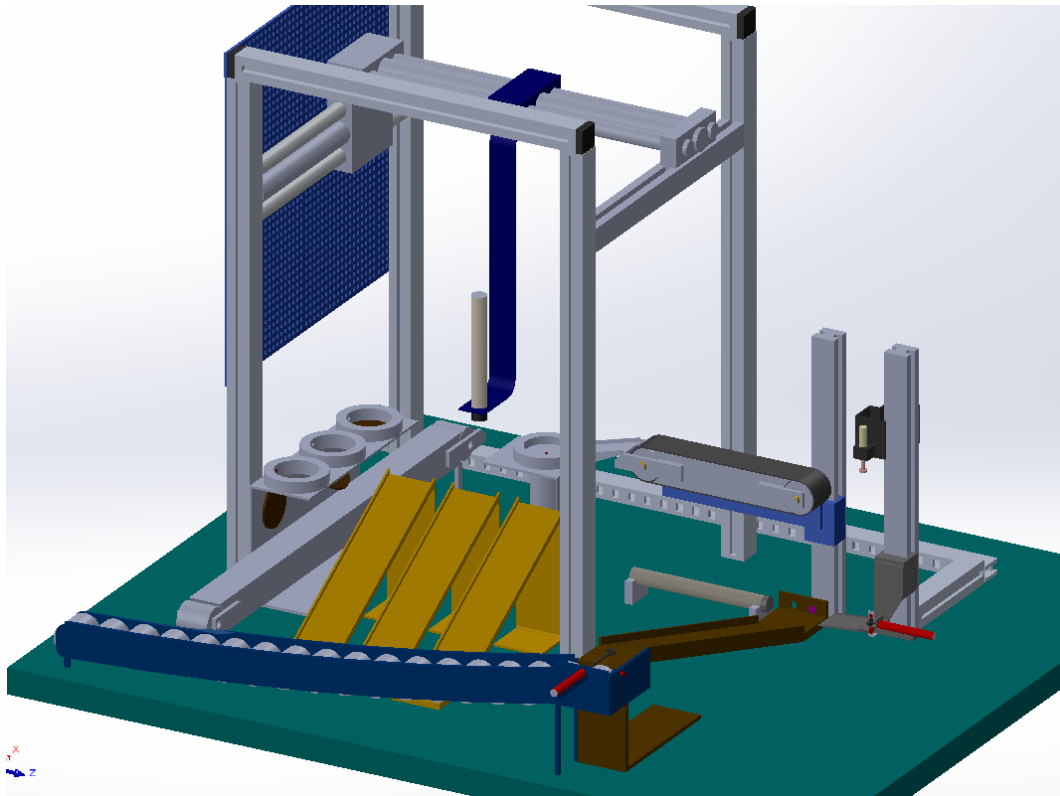


3.0.1 Figure-First type of Vertical warehouse with hatch



3.0.2 Figure- Second remodel

In image 3.0.2, an outgoing conveyor belt to a warehouse is then added.



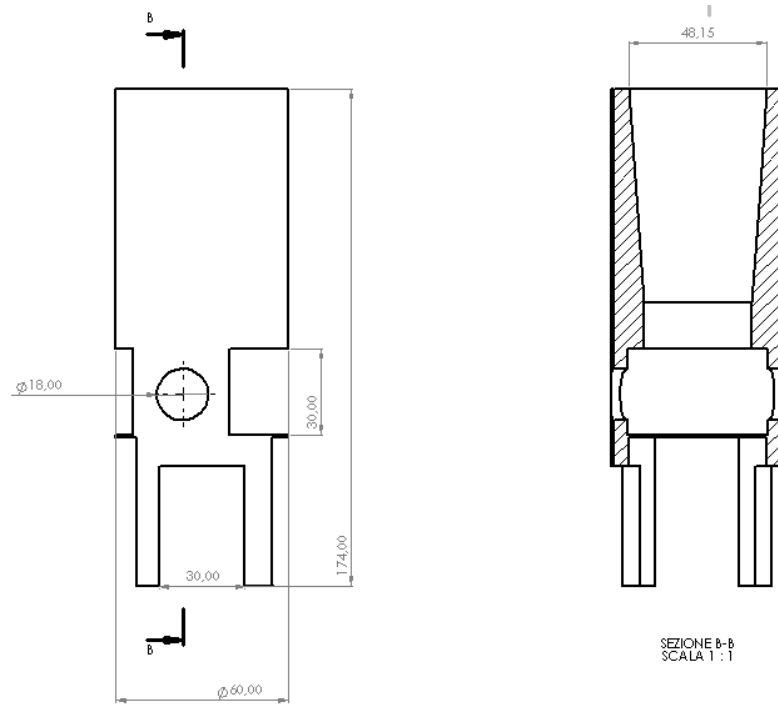
3.0.3 Figure-Third remodel

A disconnected exit from the system is preferred to a re-entry into the system (imagine 3.0.3). Therefore, a conveyor belt is placed underneath the hatches leading to a roller conveyor which re-introduces the object into the system. The slides are moved.

To power an uphill roller conveyor belt, it would need a motor for each roller, which would make it impractical.

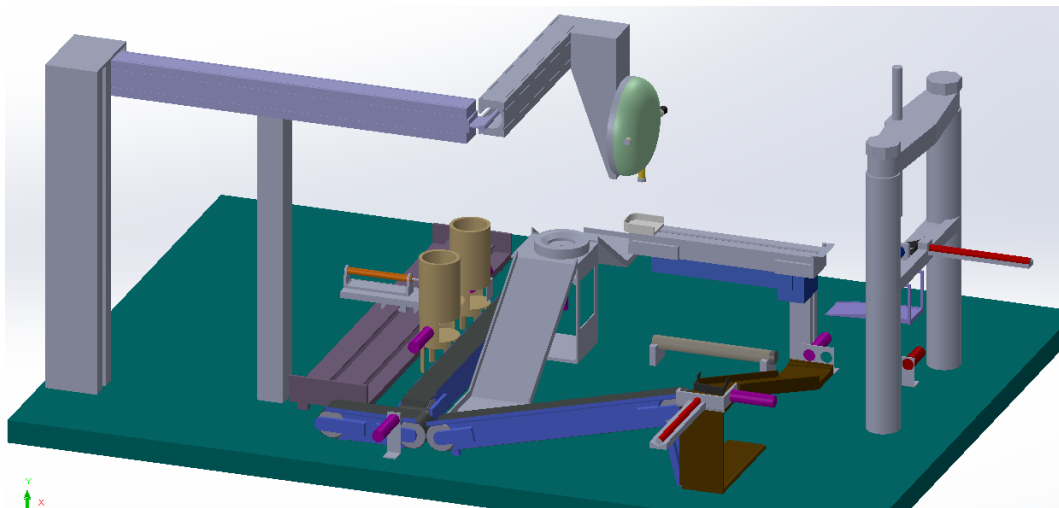
In addition, the disadvantage is that there is no way to store more than one object.

This problem is solved by the introduction of a vertical warehouse, figure 3.0.4.



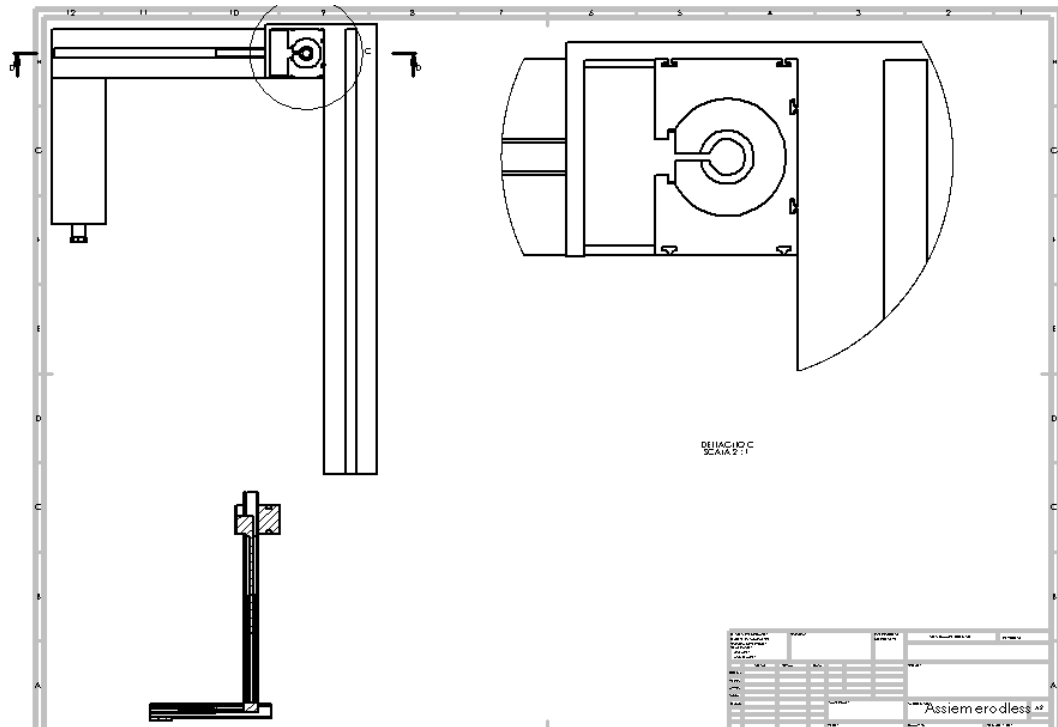
3.0.4 Figure- Vertical Warehouse

In figure 3.0.4, the vertical warehouse, capable of storing 4 objects, is shown. From section B-B it is possible to see the conical introduction, functional for easier positioning. On the left we note the hole for the optical positioning sensor.



3.0.5 Figure- Fourth remodel

In figure 3.0.5, the new model solution is shown. the cartesian arm is removed and a series of rod-less pistons is introduced (figure 3.0.6).



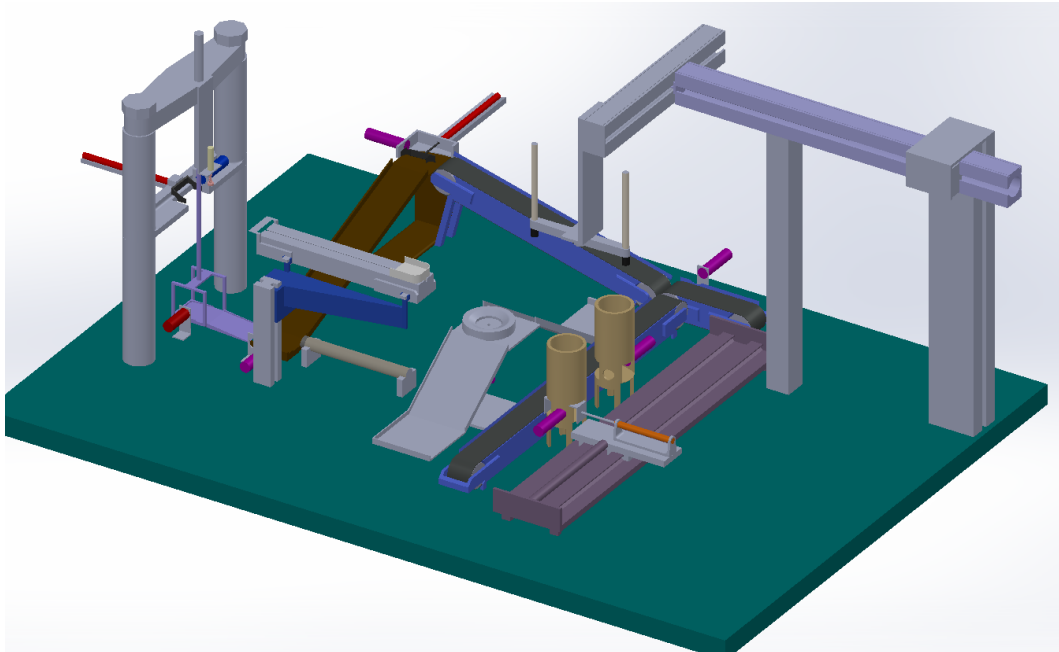
3.0.5 Figure- Rod less particular

The roller conveyor belt is replaced by two belts. This is so that it is not necessary to supply power to all the rollers but only to one per belt.

A guide is inserted on which there is a piston. Thanks to this, it is possible to push the part (after a call from the HMI) and reinsert it into the system. The belt after the lift is replaced by a slide driven by a piston.

A new function, stamping, is introduced. The stamping piston is on the same end as the vacuum on a device that can rotate along the z-axis.

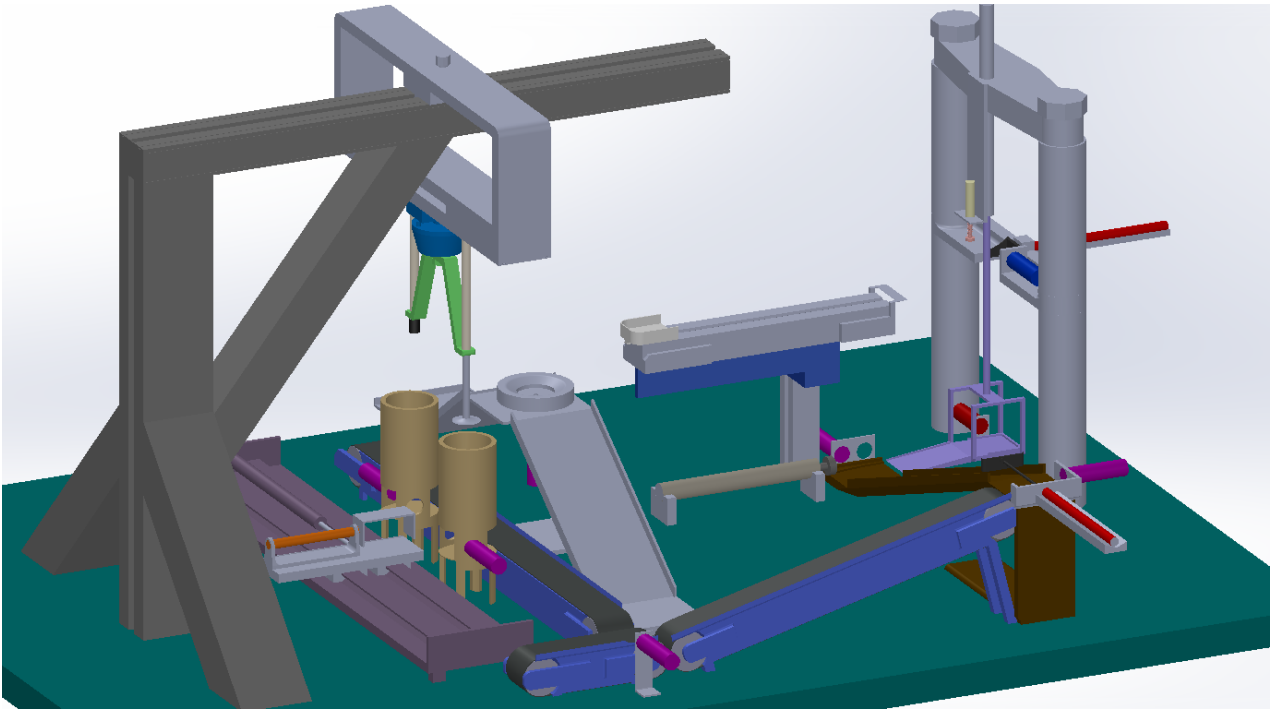
The latter is difficult to implement and will therefore be replaced, figure 3.0.6.



3.0.6 Figure- Fifth remodel

This rotary solution, figure 3.0.6, is also not optimal from the point of view of space.

3.1. Final version

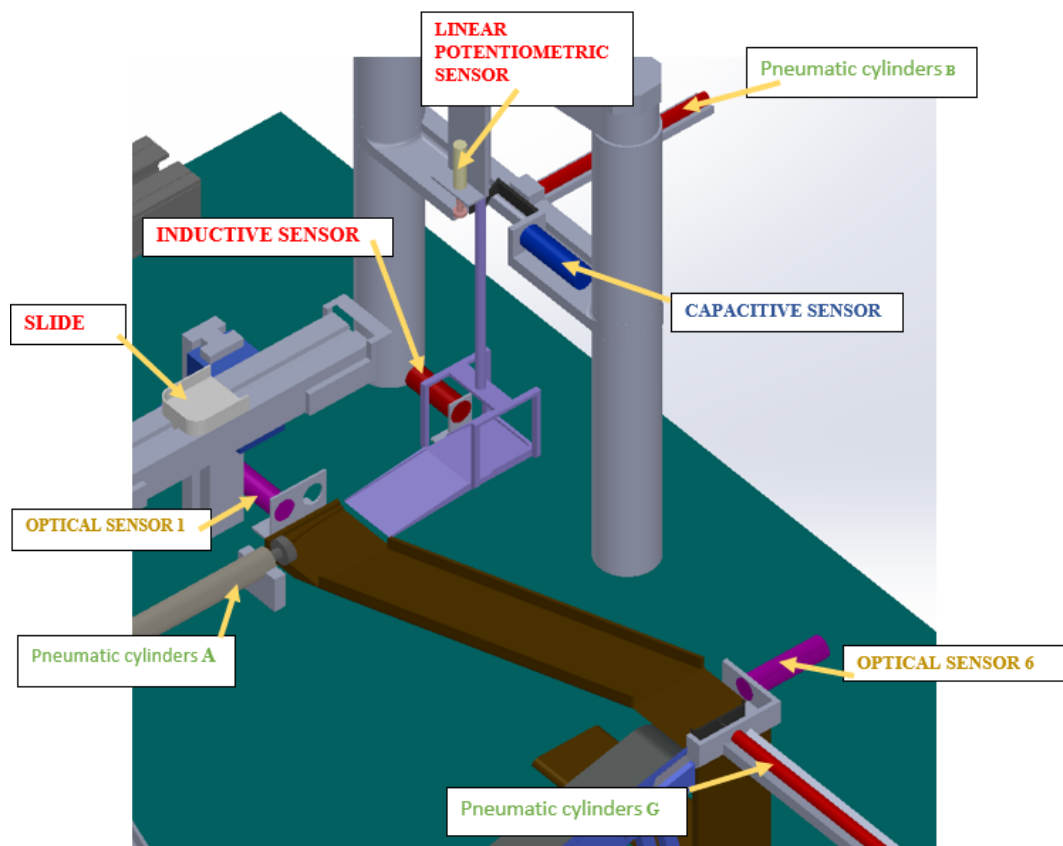


3.1.0 Figure- Final model

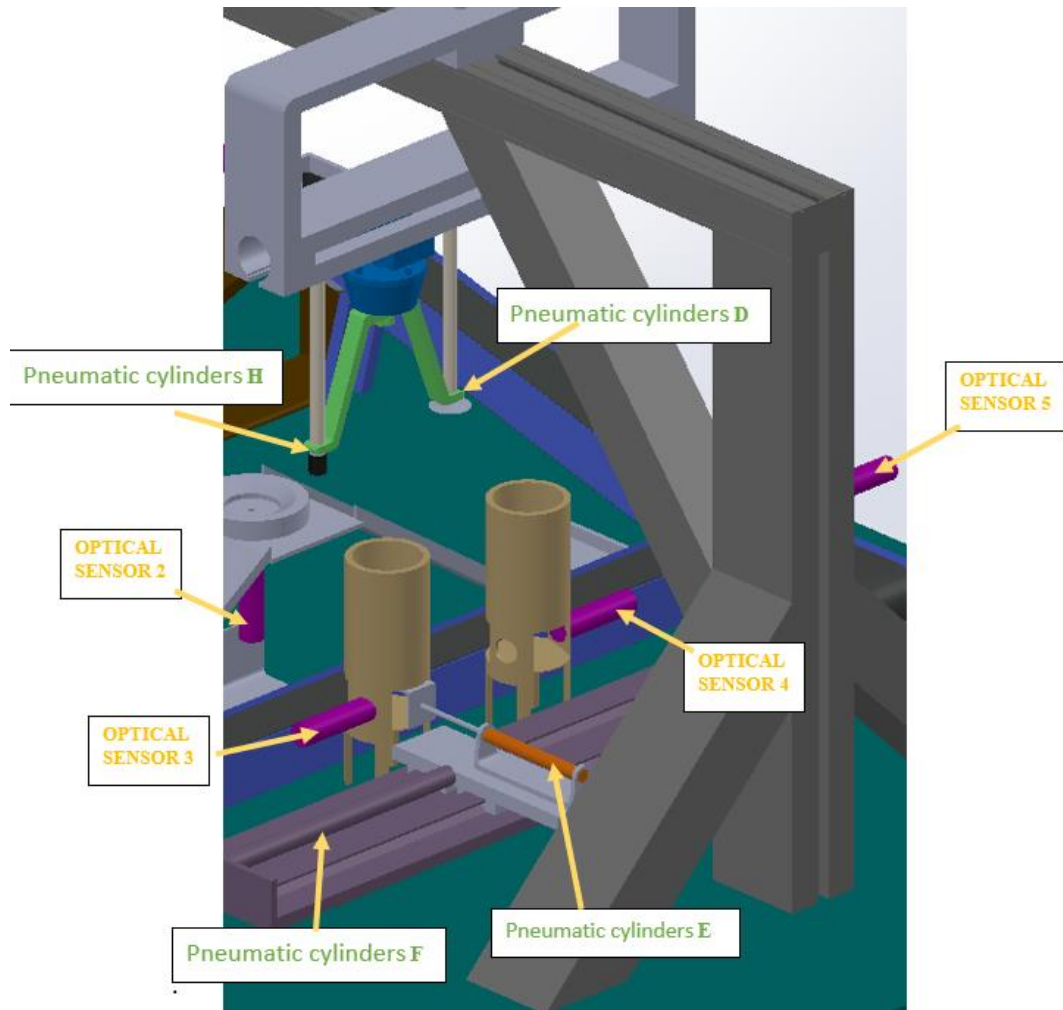
In figure 3.1.0, to improve the structural stability of the rod-less pistons, a hanging beam is inserted.

The mould and vacuum pistons are supported by a V-shaped support to avoid longitudinal bulk.

3.2. Parts and sensors



3.2.0 Figure- Model sensor/component pt.1



3.2.1 Figure- Model sensor/component pt.2

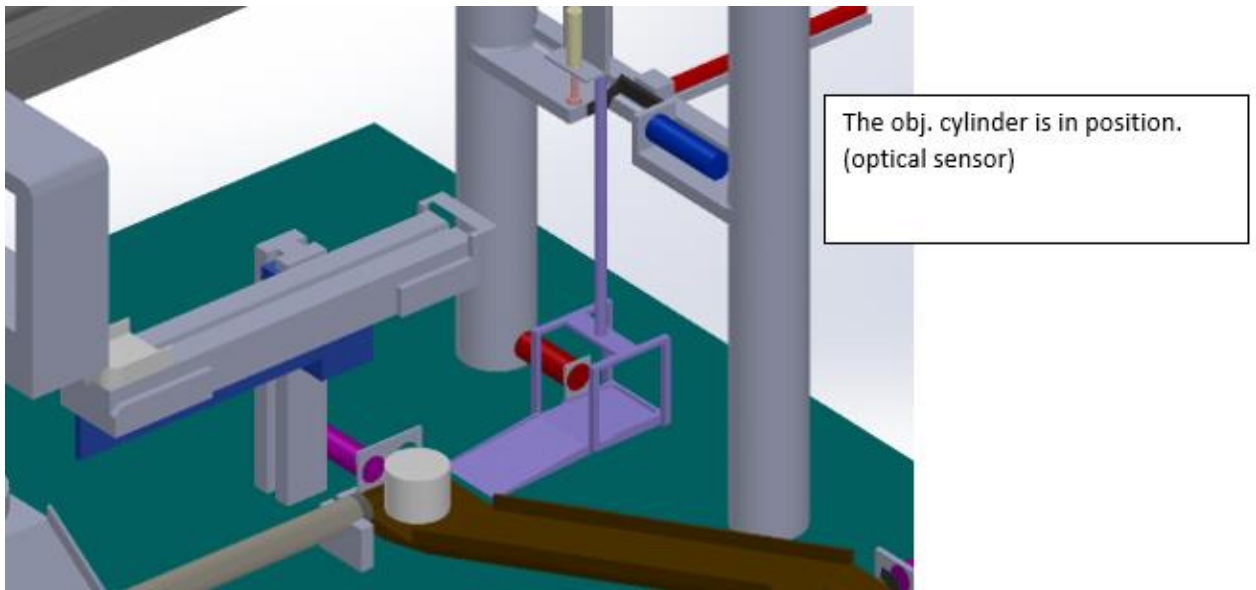
In figures 3.2.0 and 3.2.1, sensors and important parts are shown. The optical sensor indicates if the object (cylinder) is in position (state 1 if is in 0 otherwise).

The inductive sensor indicates if the object (cylinder) is on the lift (state 1 if is in 0 otherwise).

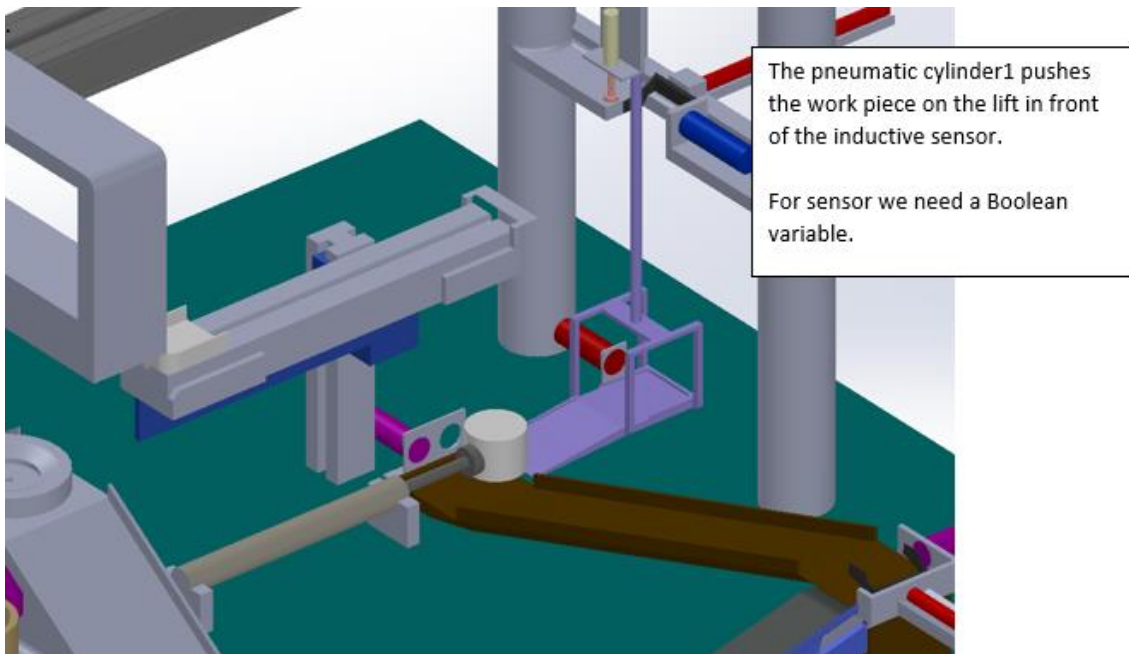
The capacitive sensor indicates if the object (cylinder) is iron or plastic (state 1 for plastic 0 otherwise).

The linear potentiometric sensor measures the height of the workpiece. There are two types of workpiece: tall and short. By using the in-range function, for example, it is possible to distinguish between tall and short workpieces.

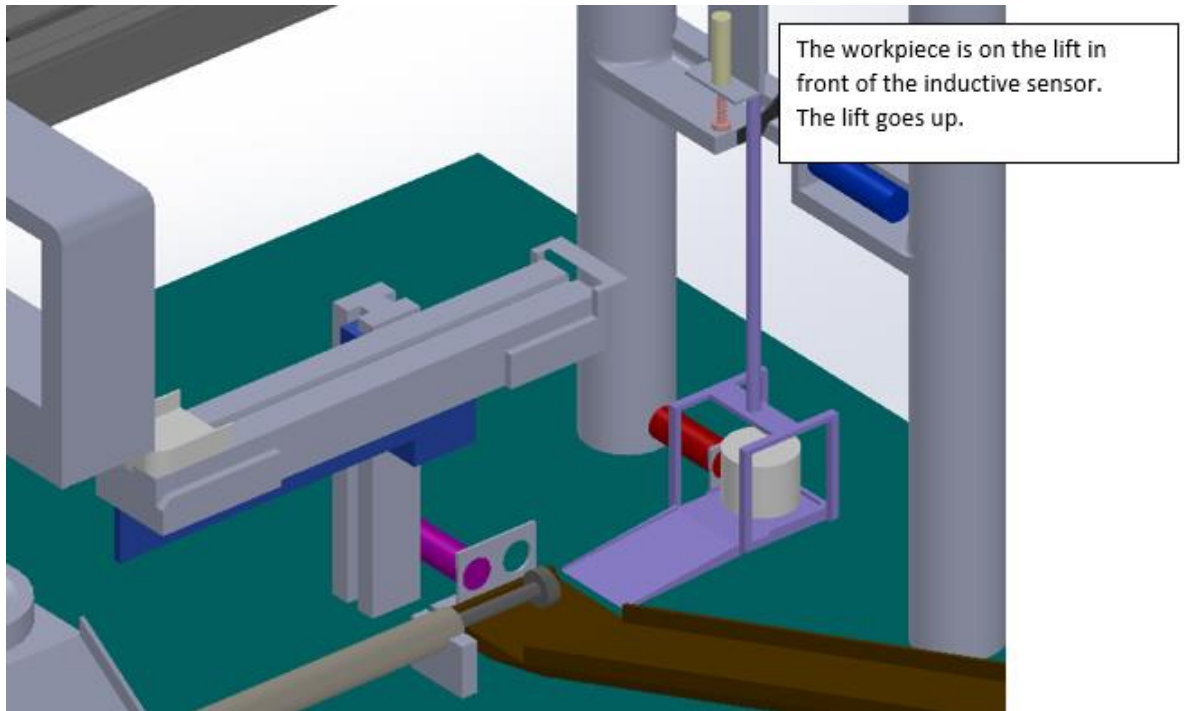
3.3. Desired behaviour of the model



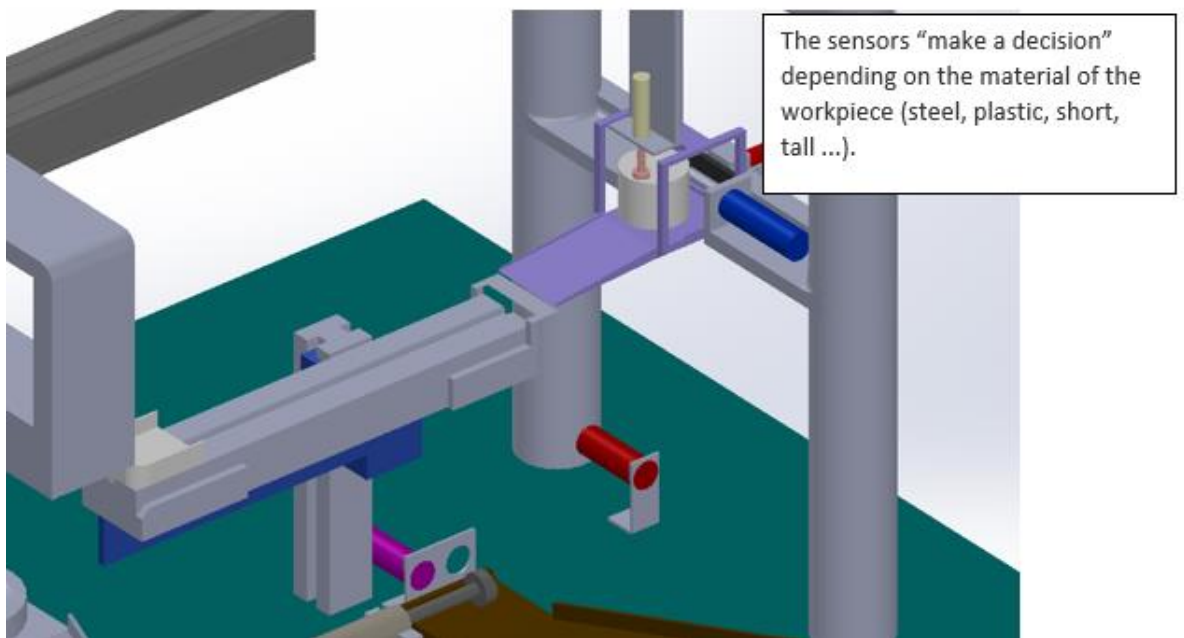
3.3.0 Figure-



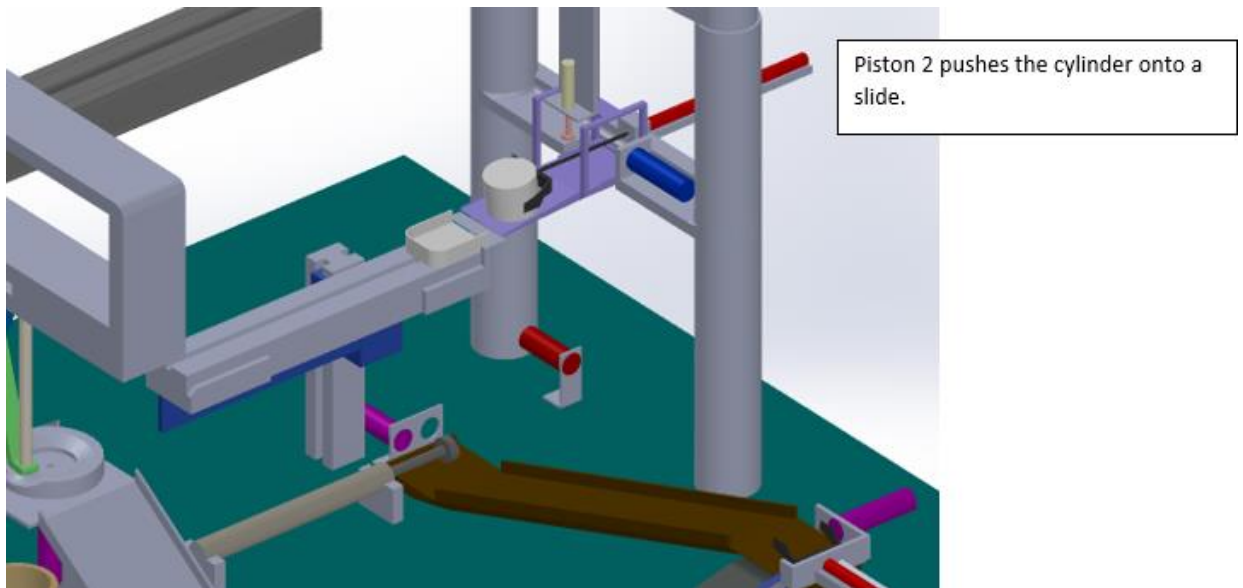
3.3.1 Figure-



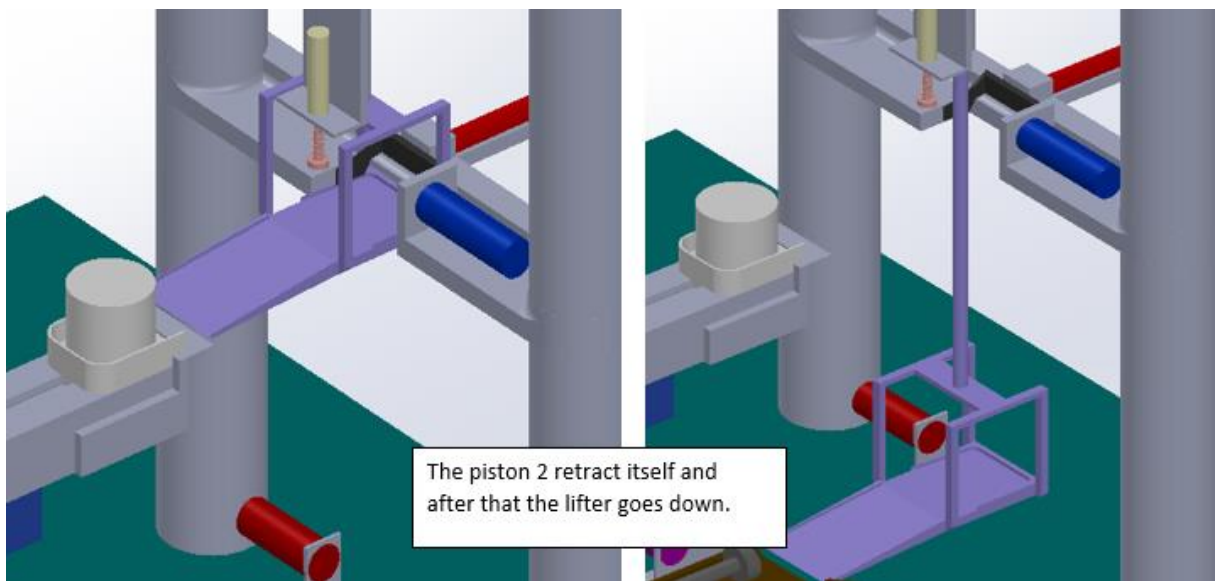
3.3.2 Figure-



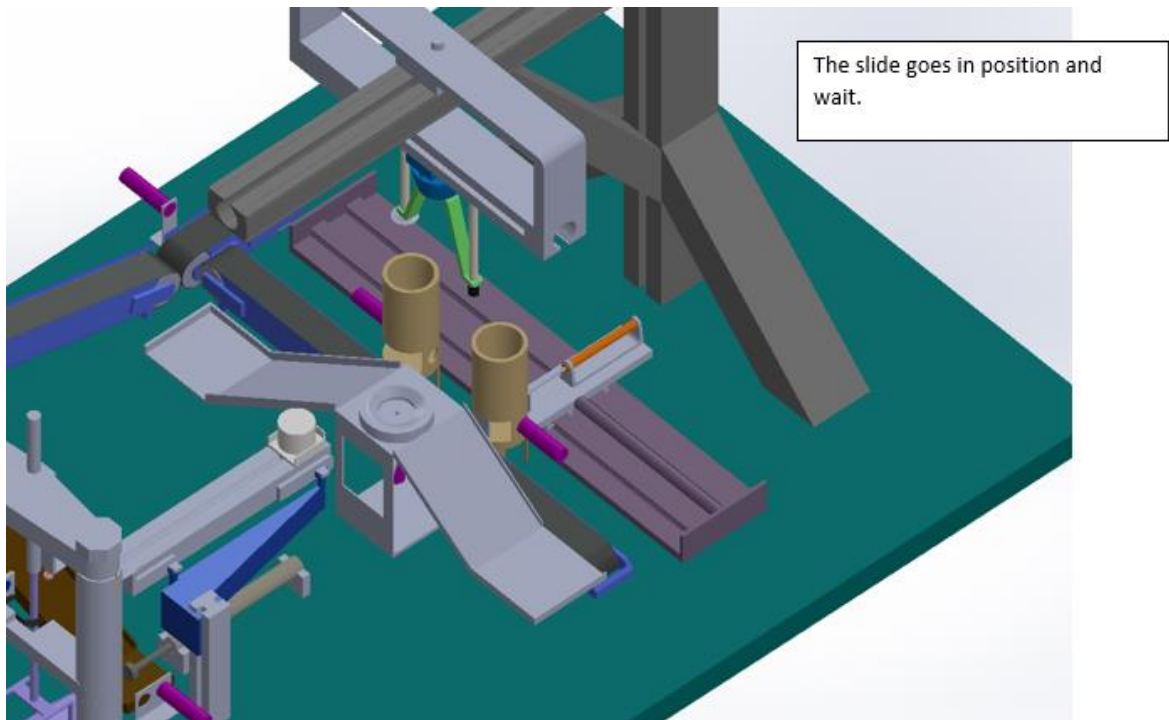
3.3.3 Figure-



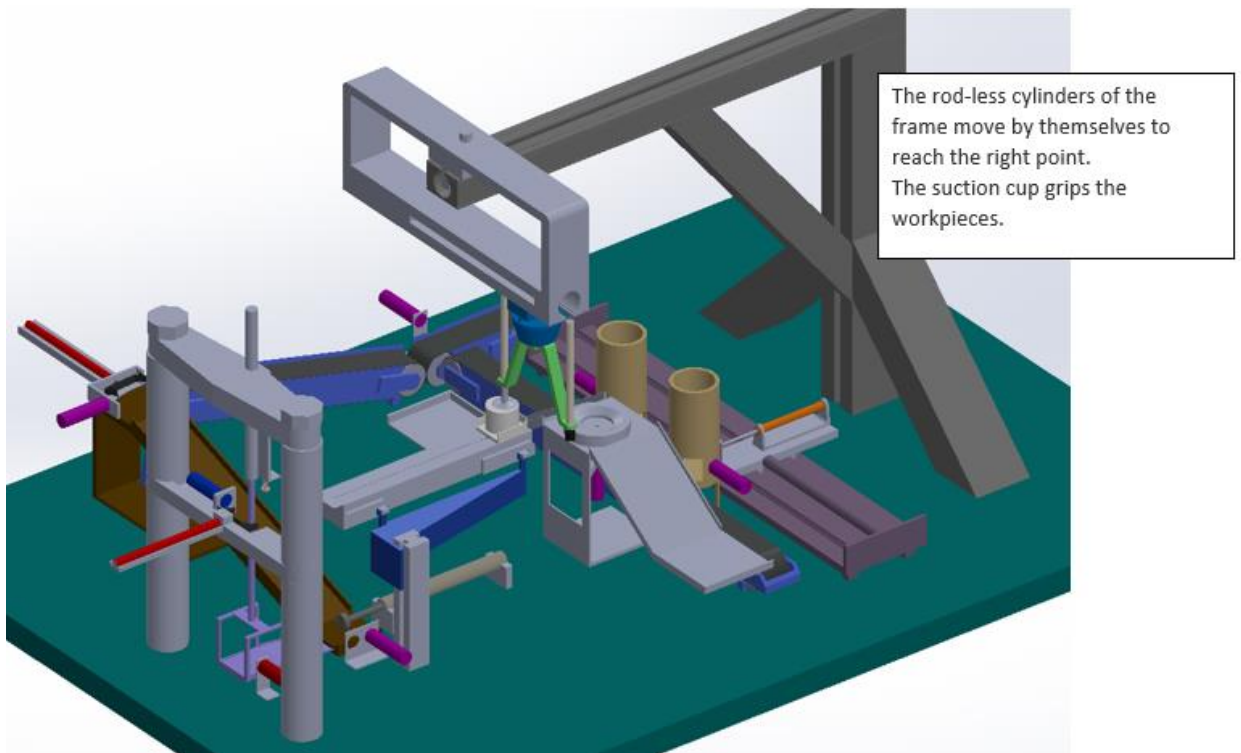
3.3.4 Figure-



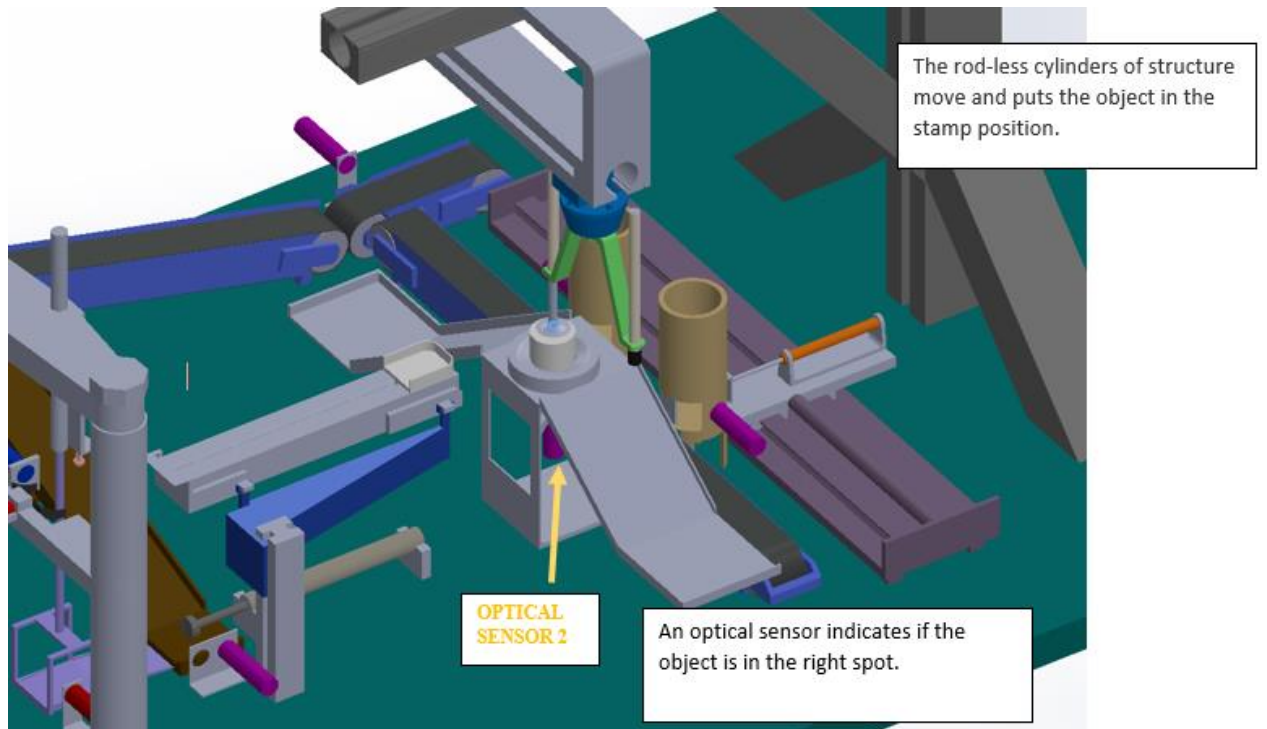
3.3.5 Figure-



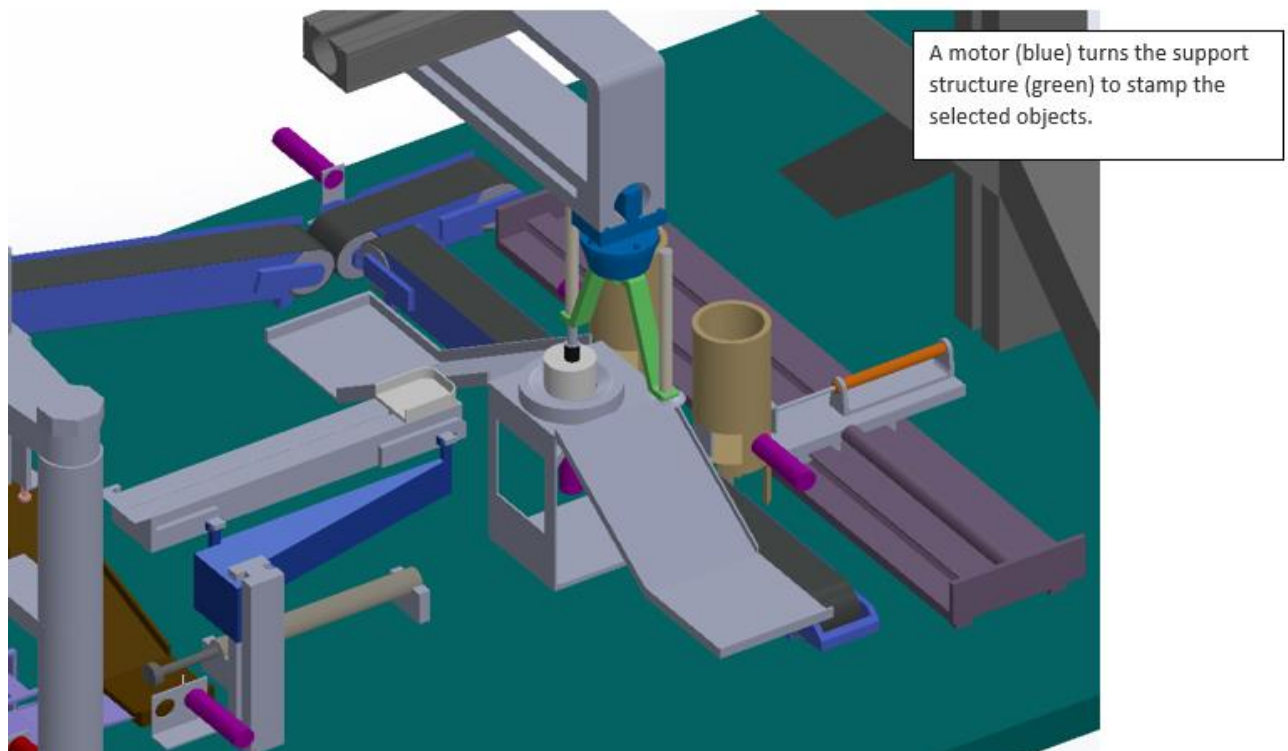
3.3.5 Figure-



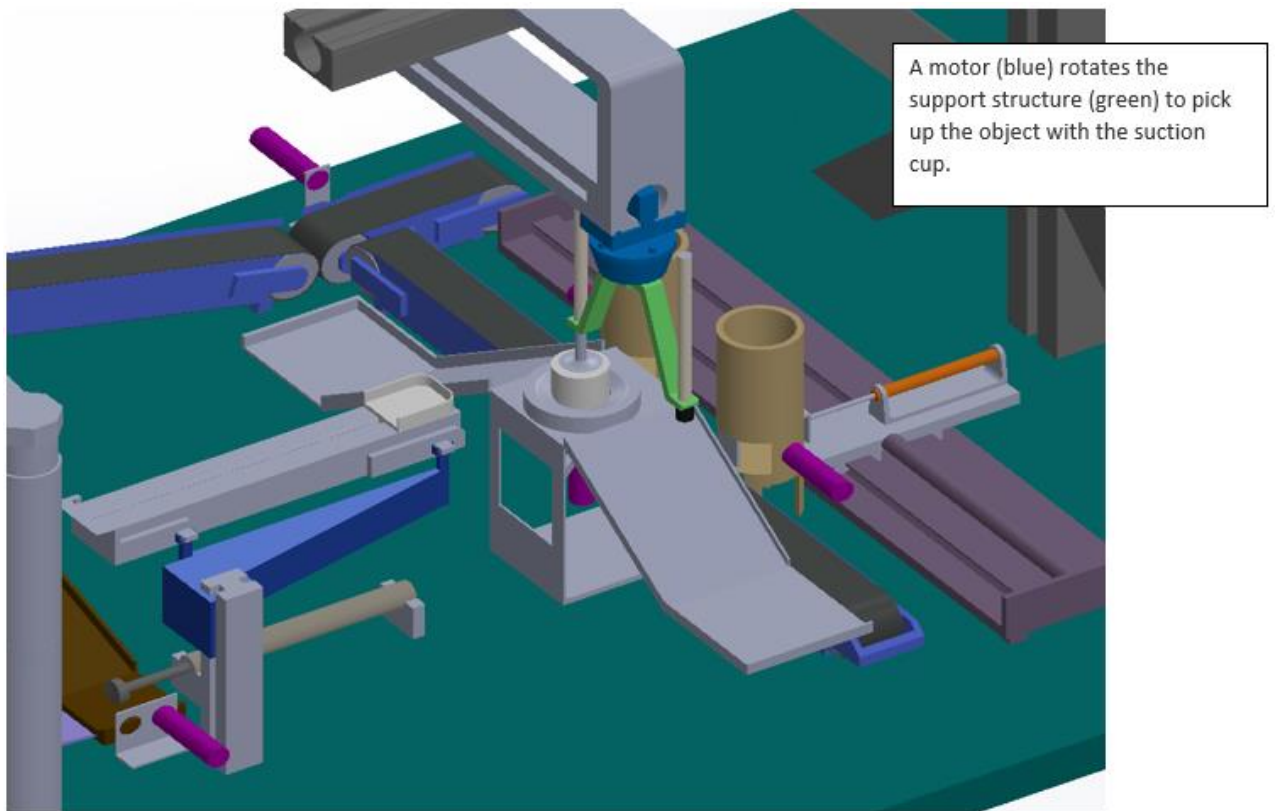
3.3.6 Figure-



3.3.7 Figure-

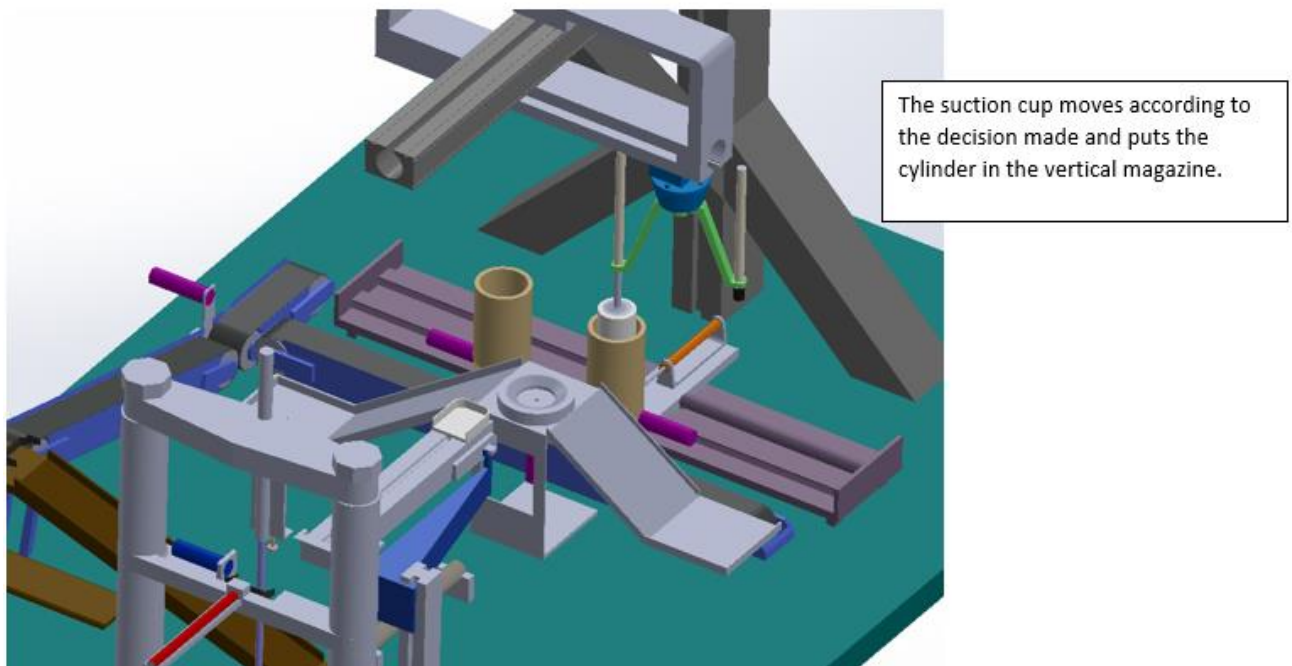


3.3.8 Figure-

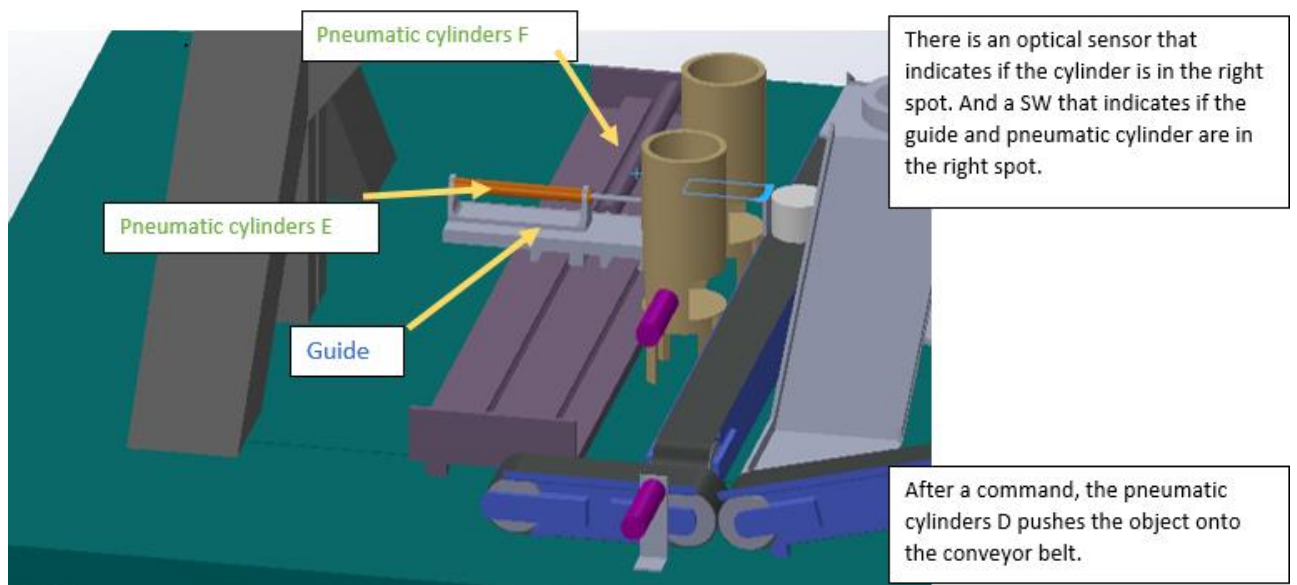


3.3.8 Figure-

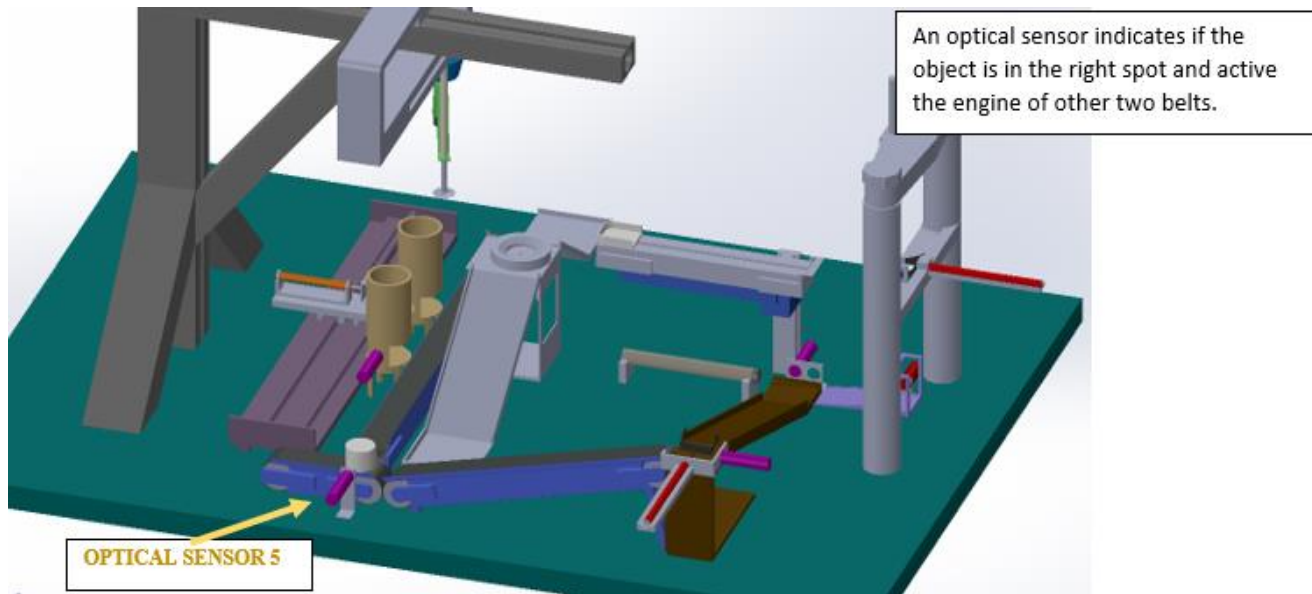
The idea is to move cylindrical objects of different heights up the chutes, while the others, print them out and place them in the vertical magazines. For example, in the magazines: cylinders of the same height but of a different colour or material.



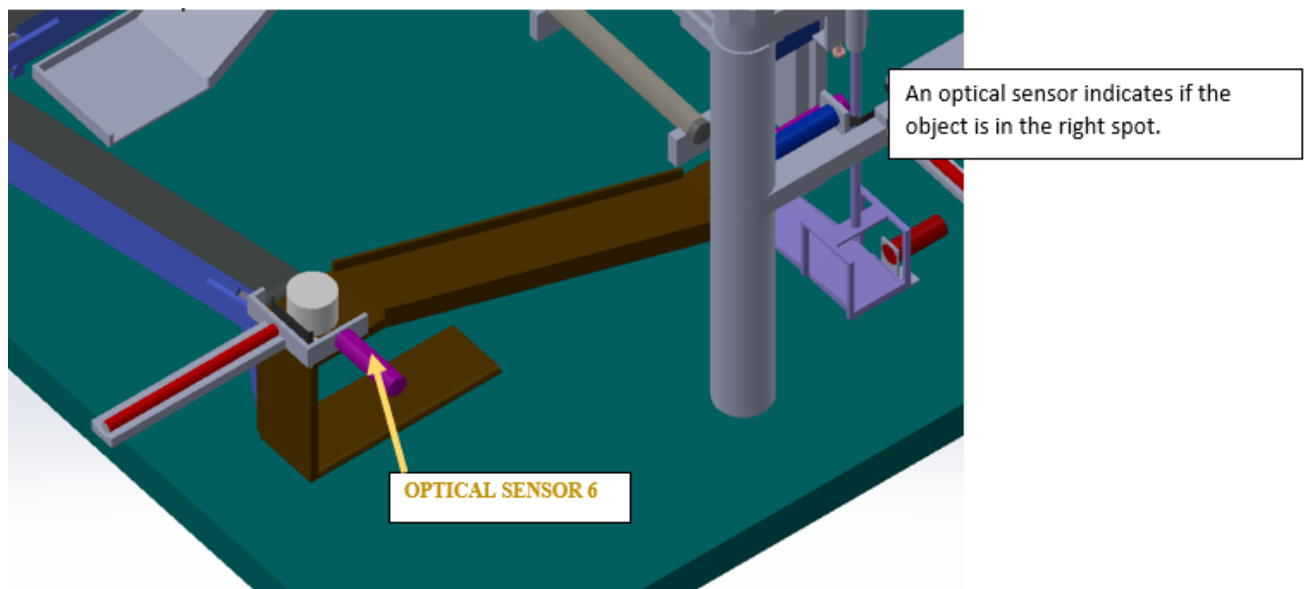
3.3.9 Figure-



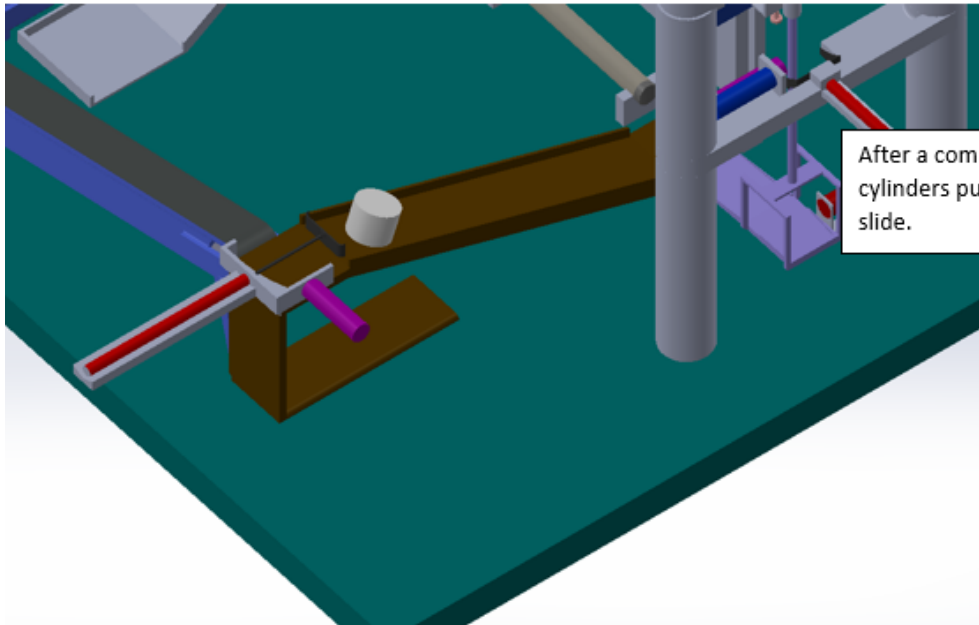
3.3.10 Figure-



3.3.11 Figure-



3.3.12 Figure-



After a command, the pneumatic cylinders push the object onto the slide.

3.3.13 Figure-

3.4. VARIABLES:

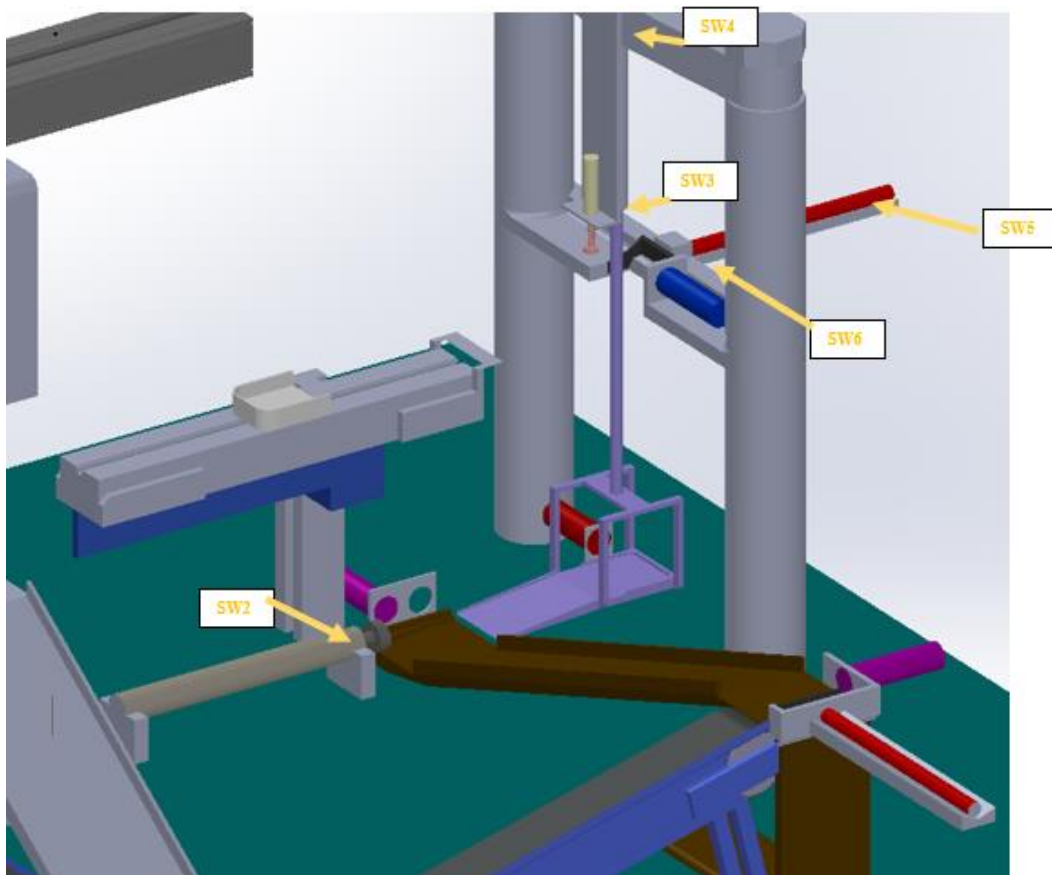
In order to program the model's movements in Automation Studio, it is necessary to define control variables.

Component	Name
Linear Potentiometric sensor	S_LP
Inductive Sensor	S_i
Capacitive Sensor	S_c
Optical Sensor 1	S_op1
Optical Sensor 2	S_op2
Optical Sensor 3	S_op3
Optical Sensor 4	S_op4
Optical Sensor 5	S_op5
Optical Sensor 6	S_op6
Piston A	PA
Piston B	PB
Piston C - Lifter	PC
Piston D - suction cup	PD
Piston E	PE
Piston F - guide	PF
Piston G	PG
Piston H	PH

3.4.0 Table-

The **SW**x are limit switches on pneumatic cylinders. There are not design in the model.

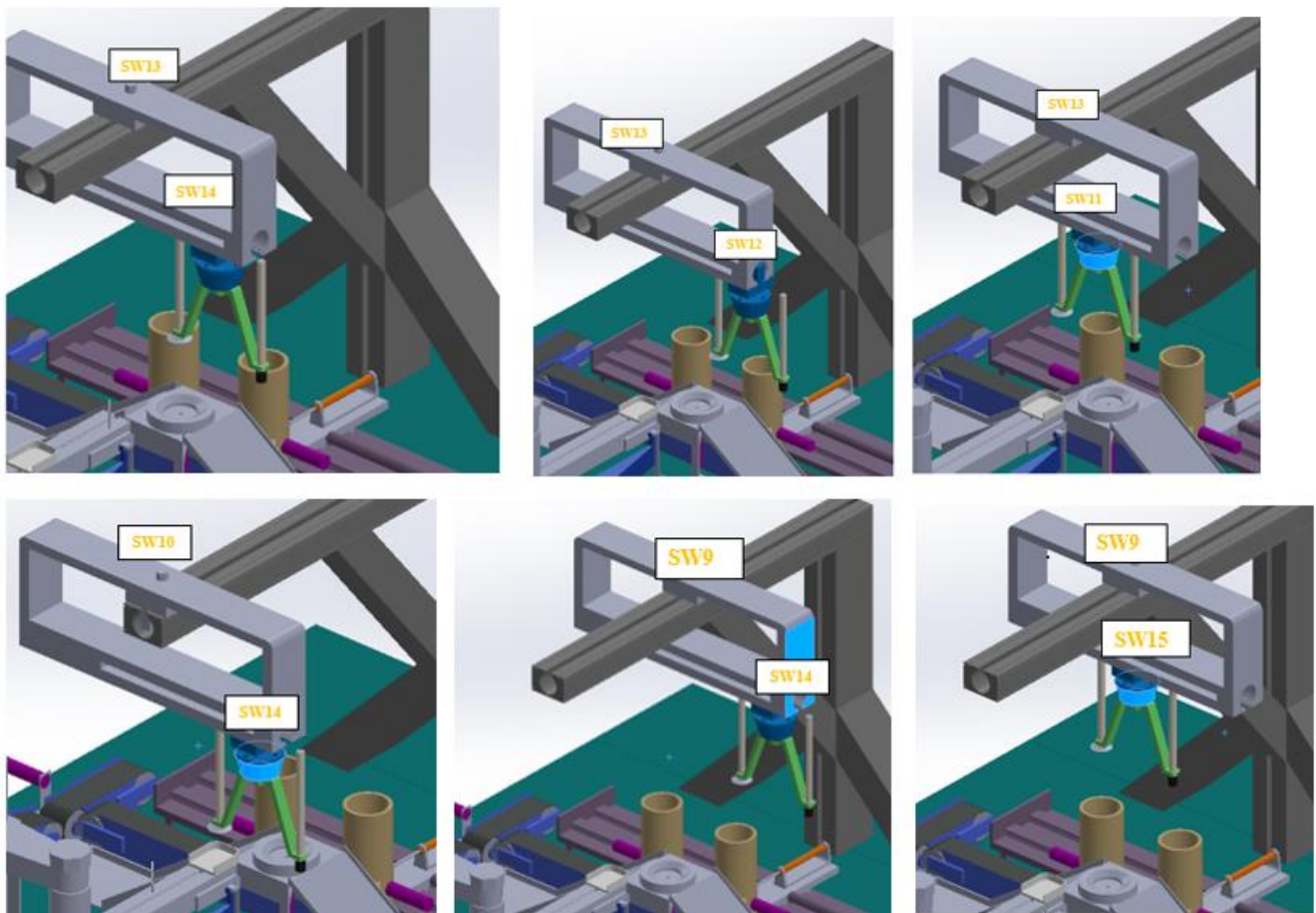
Ex indicates the command to start the engines of conveyors.



3.4.1 Figure- SW

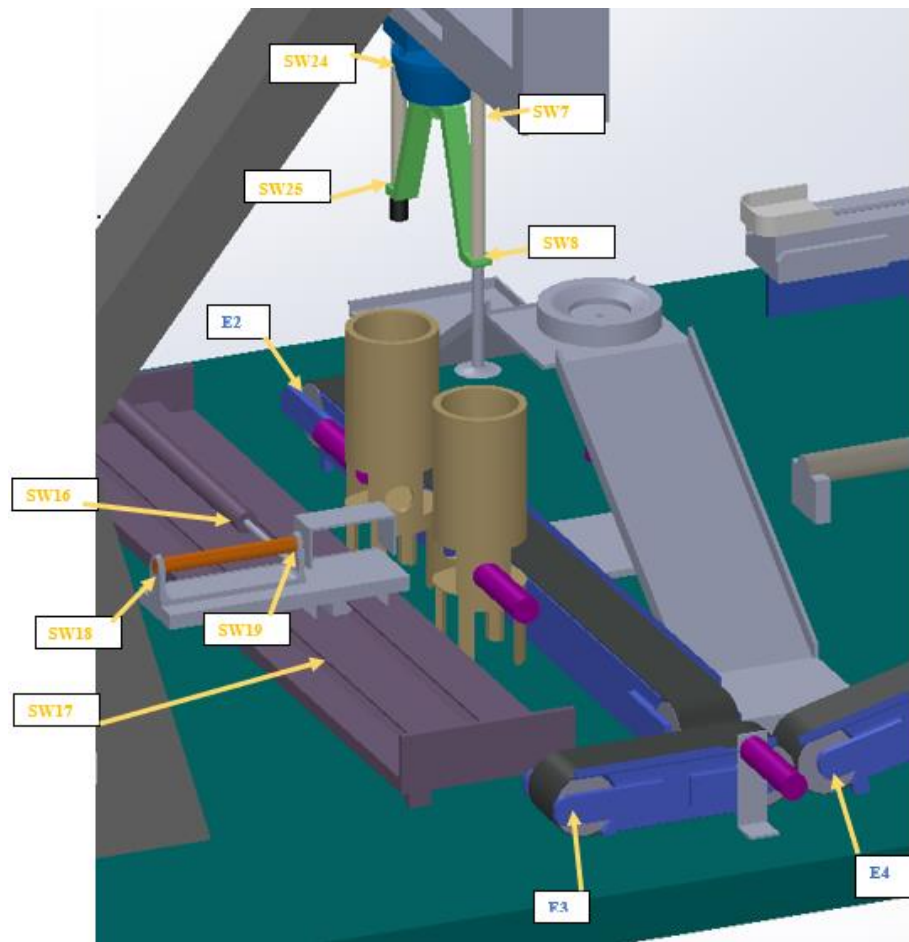
SW2	Outstroke of the pneumatic cylinders A
SW3	Instroke of the pneumatic cylinder of lifter
SW4	Outstroke of the pneumatic cylinder of lifter
SW5	Instroke of the pneumatic cylinders B
SW6	Outstroke of the pneumatic cylinders B

3.4.2 Table- SW



3.4.1 Figure- SW

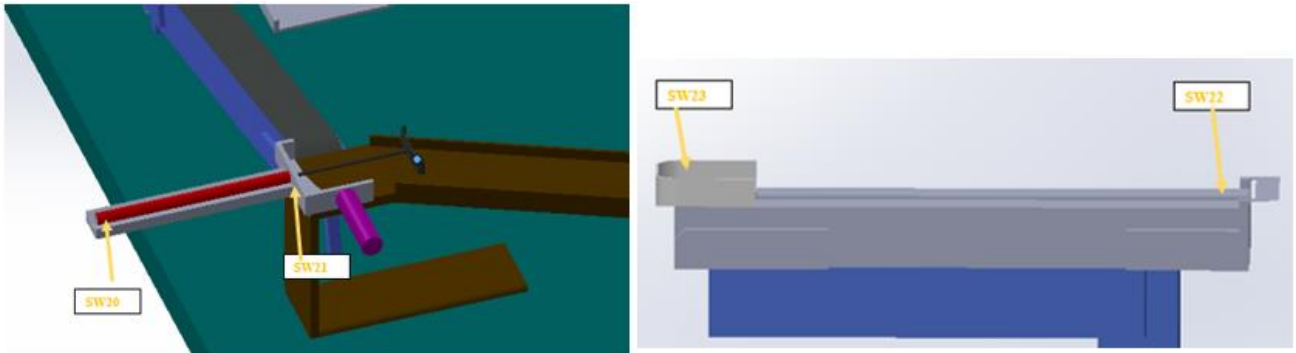
SW9	Endstroke for position 1 x
SW10	Endstroke for position 5 X
SW11	Endstroke for position 4 y
SW12	Endstroke for position 3 y
SW13	Rest position X axis
SW14	Rest position Y axis
SW15	Endstroke for position 2 y



3.4.2 Figure- SW

SW7	Instroke of the pneumatic cylinder of suction cup
SW8	Outstroke of the pneumatic cylinder of suction cup to Warehouse
SW8.1	Outstroke of the pneumatic cylinder of suction cup to SLIDE
SW8.2	Outstroke of the pneumatic cylinder of suction cup to rest position
SW16	Instroke Position F Guide – first warehouse
SW17	Outstroke Position F Guide –second warehouse
SW18	Instroke pneumatic cylinder E
SW19	Outstroke pneumatic cylinder E
SW24	Instroke of the pneumatic cylinder of stamp to rest position
SW25	Outstroke of the pneumatic cylinder of stamp to rest position

3.4.3 Table- SW



3.4.4 Figure- SW

SW20	Instroke pneumatic cylinder G
SW21	Outstroke pneumatic cylinder G
SW22	Position on right slide
SW23	Position on left slide

3.4.5 Table- SW

3.5. Lists of necessary theoretical variables:

NAME	Type	Description	Action	I/O
P_STOP	Boo	Bound to HMI for stop the cycle		I
PRESTAR T	Boo	Bound to HMI for replace in rest position		
SW_2	Boo	Outstroke of the pneumatic cylinders A		I
SW_3	Boo	Instroke of the pneumatic cylinder of lifter	Lifter down	I
SW_4	Boo	Outstroke of the pneumatic cylinder of lifter	Lifter up	I
SW_5	Boo	Instroke of the pneumatic cylinders B		I
SW_6	Boo	Outstroke of the pneumatic cylinders B	Charge the conveyor	I
SW_7	Boo	Instroke of the pneumatic cylinder of suction cup		I
SW_8	Boo	Outstroke of the pneumatic cylinder of suction cup to Warehouse		I
SW_8.1	Boo	Outstroke of the pneumatic cylinder of suction cup to SLIDE		
SW_8.2	Boo	Outstroke of the pneumatic cylinder of suction cup to rest position		
SW_9	Boo	Endstroke for position 1 x	position 1 x	I
SW_10	Boo	Endstroke for position 5 X	position 1 with SW_9	I
SW_11	Boo	Endstroke for position 4 y	position 4 with SW_13	I
SW_12	Boo	Endstroke for position 3 y	position 3 with	I

			SW_13	
SW_13	Boo	Rest position X axis	rest position with SW_14	I
SW_14	Boo	Rest position Y axis	rest position with SW_13	I
SW_15	Boo	Endstroke for position 2 y	position 2 with SW_9	I
SW_16	Boo	Instroke Position F Guide – first warehouse		I
SW_17	Boo	Outstroke Position F Guide –second warehouse		I
SW_18	Boo	Instroke pneumatic cylinder E		I
SW_19	Boo	Outstroke pneumatic cylinder E		I
SW_20	Boo	Instroke pneumatic cylinder G		I
SW_21	Boo	Outstroke pneumatic cylinder G		I
SW_22	Boo	Position of rx slide		
SW_23	Boo	Position of lf slide		
SW_24	Boo	Instroke of the pneumatic cylinder of stamp to rest position		
SW_25	Boo	Outstroke of the pneumatic cylinder of stamp to rest position		
S_c		Capacitive sensor		I
S_i		Inductive sensor		I
S_op1	Boo	Optical sensor1		I
S_op2	Boo	Optical sensor2		I

S_op3	Boo	Optical sensor3		I
S_op4	Boo	Optical sensor4		I
S_op5	Boo	Optical sensor5		I
S_op6	Boo	Optical sensor6		I
S_LP	Boo	Linear Potentiometric sensor	Piezometric sensor for heights	I
V_y		External variable in order to change the velocity of y Vacuum		
V_x		External variable in order to change the velocity of x Vacuum		
V_z		External variable in order to change the velocity of z Vacuum		
V_lift		External variable in order to change the velocity of lifter		
W1	inter	External variable in order to change the velocity of conveyor1		
W2	Inter	External variable in order to change the velocity of conveyor2		
W3	Inter	External variable in order to change the velocity of conveyor3		
W4	Inter	External variable in order to change the velocity of conveyor4		
E2	Boo	Active engine conveyor2		O
E3	Boo	Active engine conveyor3		O

E4	Boo	Active engine conveyor4		O
EV_1-	Boo	Solenoid valve that retracts Pneumatic Cylinder A	Pa-	O
Ev_1+	Boo	Solenoid valve that extends Pneumatic Cylinder A	PA+	O
Ev_2+	Boo	Solenoid valve for upwards motion of pneumatic cylinder C	PC→ lifter up	O
Ev_2-	Boo	Solenoid valve for downwards motion of pneumatic cylinder C	PC →lifter down	O
Ev_3+	Boo	Solenoid valve that extends PB+		O
Ev_3-	Boo	Solenoid valve that retracts PB-		O
Ev_4+	Boo	Solenoid valve that extends the pneumatic cylinder of suction cup	PD	O
Ev_4-	Boo	Solenoid valve that retracts the pneumatic cylinder of the suction cup	PD	O
Ev_5+	Boo	Activates the suction cup		O
Ev_5-	Boo	Deactivate the suction cup		O
Ev_6+	Boo	Solenoid valve that extends axis X of suction cup system		O
Ev_6-	Boo	Solenoid valve that retracts axis X of suction cup system		O
Ev_7+	Boo	Solenoid valve that extends axis Y of suction cup system		O
Ev_7-	Boo	Solenoid valve that retracts axis Y of suction cup system		O
Ev_8+	Boo	Solenoid valve that extends Pneumatic Cylinder E		O
Ev_8-	Boo	Solenoid valve that retracts Pneumatic Cylinder E		O
Ev_9+	Boo	Solenoid valve that extends Pneumatic Cylinder F		O

Ev_9-	Boo	Solenoid valve that retracts Pneumatic Cylinder F		O
Ev_10+	Boo	Solenoid valve that extends Pneumatic Cylinder G		O
Ev_10-	Boo	Solenoid valve that retracts Pneumatic Cylinder G		O
Ev_11+	Boo	Solenoid valve that extends Pneumatic Cylinder H	Stamp	
Ev_11-	Boo	Solenoid valve that extends Pneumatic Cylinder H	Stamp	

3.5.0 Table- SW

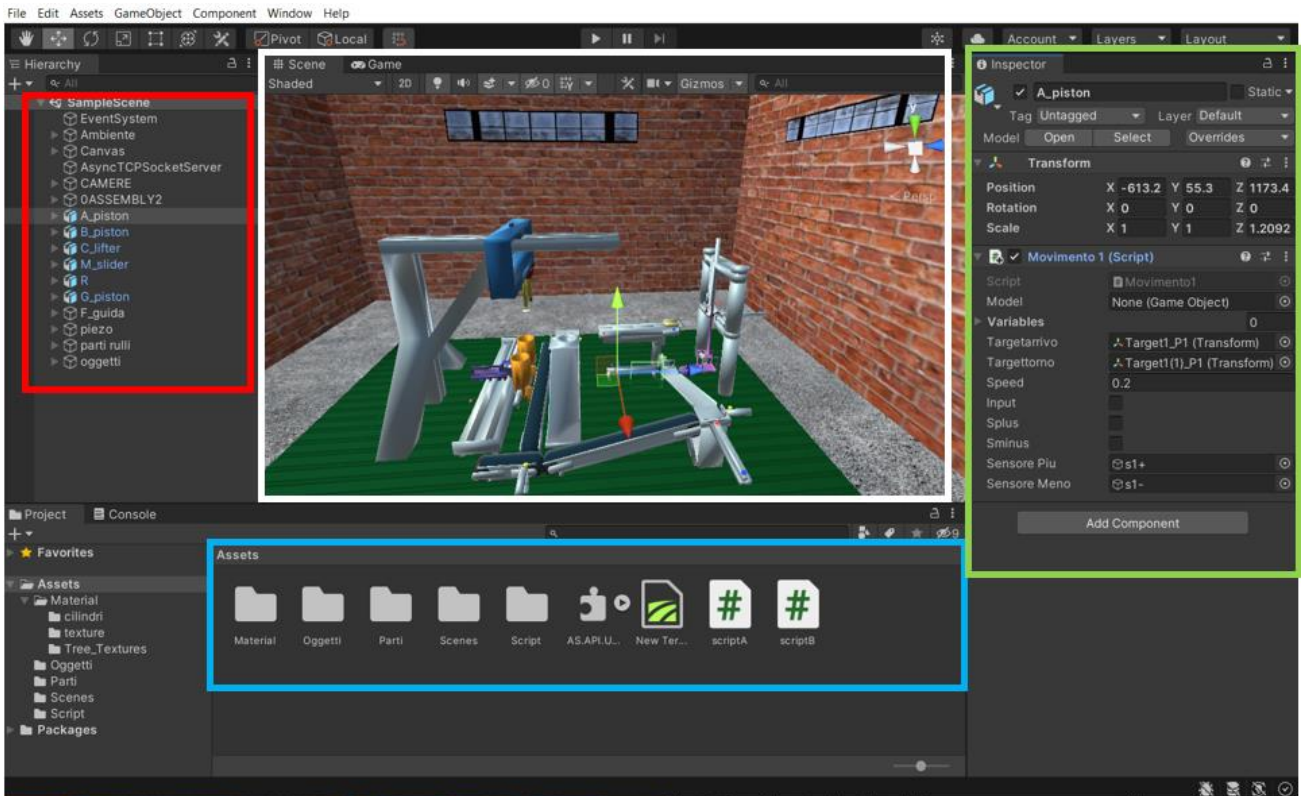
4. Unity

In order to simulate and create a working environment as close to reality as possible, the Unity programme was used.

Unity is an open source programme. This means that it is free and constantly being updated. For this thesis 2020.3.4F1LTS was used. Unity is used to create video games or simulate environments with augmented reality (i.e. program a VR).

To use Unity you need to know the C# or Java language. The writing program used is Visual Studio, which is also open source.

4.1. References



4.0.0 Figure- Unity interface

On the left are the "Scenes" in red, with the elements associated with them (as "children").

On the right are the "Inspector" in green, here are the characteristics associated with the component.

In the middle, in white, is the "composer" where you can manipulate the elements.

Above the Composer we can see a Play button. This allows you to switch between developer mode, which allows you to move and modify elements freely, and Game mode, which allows you to preview what you will be allowed to do once you have exported the file.

In blue is the Asset, i.e. the libraries that make up the file.

The strategy of dividing the asset into sub-folders (as shown in image x) was used from the outset.

Scenes

Scenes are the working environments that are created. In this case the stand-alone warehouse model. They are useful if you want more than one scene in the same file. (e.g. two different types of warehouse).

Hierarchical relationships

The parent-child concept consists of defining a hierarchy of subordination. It is possible to see in image 4.0.0 in the Scene sector that each element is a child (or subordinate) of the SampleScene element.

Another example:



4.0.1 Figure- Unity subordinated

In image 4.0.1, it is possible to see how the elements that should be moved together with other elements are subordinated.

For example, when the structure R (parent) moves it would be appropriate that all components P,V, the suction cup and the mould move together, on the contrary it would be not appropriate that when the mould piston descends along the z-axis it moves R. Thanks to this, a tree of hierarchies is created.

It is possible to build this relationship in the Scene section or via scripts.

4.2. Objects

It is not possible to insert every type of format on Unity. The recommended formats for objects are the .obj extensions.

To create the structural elements accurately, SolidWorks is used, which does not have the .obj extension as an export file in the version provided by the polytechnic. From SolidWorks, the individual parts are exported in the highest quality in .stl format. an open source program called MeshLab is therefore used to convert .stl formats into .obj. This is to preserve the precise technical characteristics obtained from SolidWorks machining.

The objects in the Unity environment are called GameObjects.

Once the objects have been imported into the Composer it is possible to rotate, translate and scale them.

As a working strategy, it has been decided not to import all the parts. It has been decided to group all the structural and static parts into a single part and to assemble only those that need to perform an action.

Objects> centre

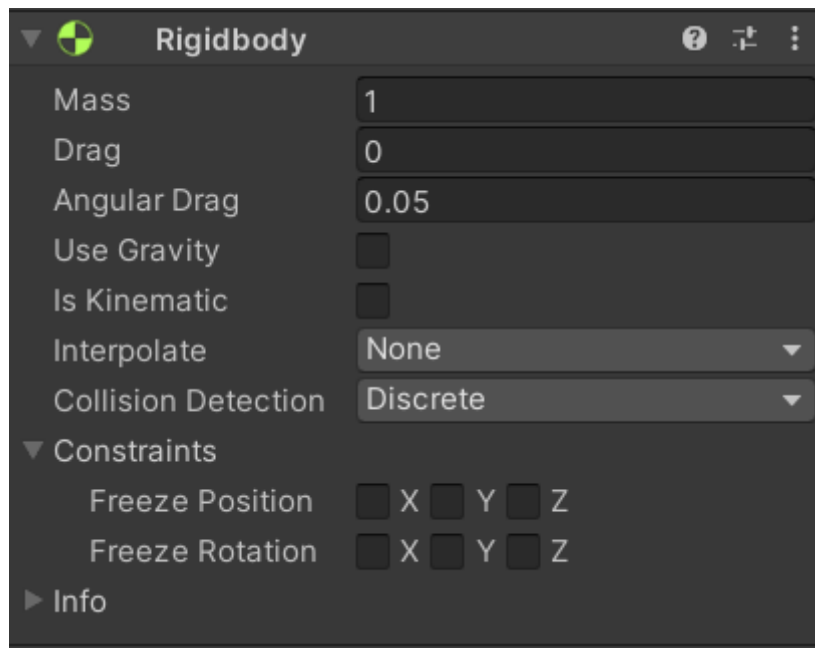
Once the object has been imported into the scene, it must be mounted by shifting its centre. However, since it is a converted object, the centre of the object will not coincide with the centre of inertia or geometry. The solution found is to create an empty object (B) and position its centre where desired and make object B the father of the object to be moved. At this point, it is possible control the object by moving the empty object B. To create an empty object, click in the scene panel on the left >create>EmptyObject.

This "technique" is used in this project to rotate the V-shaped structure on itself. In fact, for operations of this type, it is essential to have the centre of the object at the centre of mass.

4.3. Rigidbody

Once the object has been inserted, it is possible to assign characteristics to it via the inspector (add component).

"Rigidbody" is the characteristic that allows a GameObject to react in real time to the laws of physics.



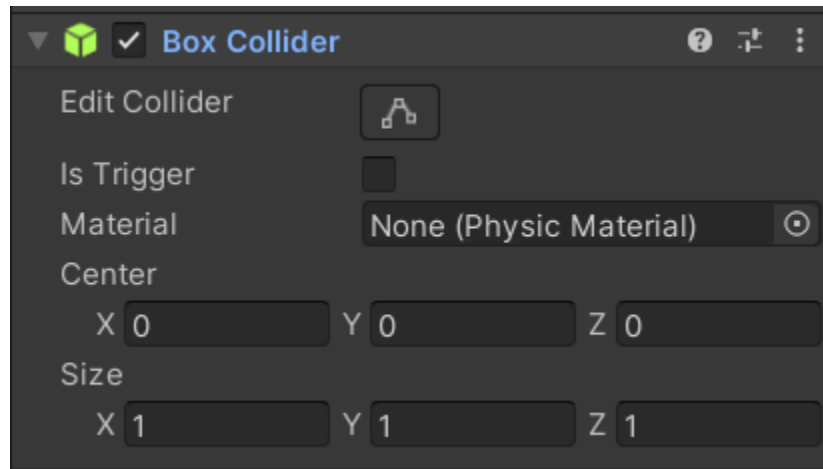
4.3.0 Figure- Unity Rigidbody

As can be seen from the image, it is therefore possible to assign a mass, a coefficient of resistance to angular and linear drag. (The actual coefficient of friction is not set here).

Checking the Gravity box will cause the object to be affected by gravity. Checking the Kinematic box means the object will not be subject to dynamic actions created by interaction with other pieces. It is possible to constrain along local axes.

In this project, each cylinder has a RigidBodies component so we set a mass, a coefficient of resistance to rotation (being cylinders they should not tip over easily) and be subject to gravity.

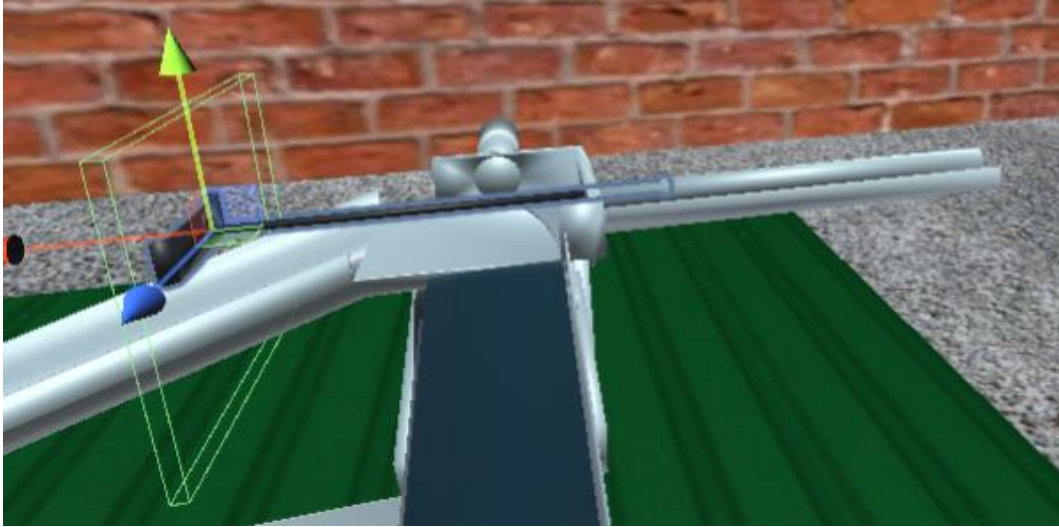
4.4. Collider



4.4.0 Figure- Unity collider

Another key component for many parts is the collider. As a strategy this feature is not used for every object but sometimes to simulate their behaviour.

Unity, not being an engineering programme, does not allow the precision of coupling that professional software does. Inserting the piston rod into the cylinder "manually" and adding the "Collider" feature to both could create clipping and unwanted behaviour due to internal collisions. The strategy used to avoid this is to create a cube the size of the piston head, place it in the piston head, make its mesh invisible and apply the collider to it. Thanks to this stratagem it is possible to associate the piston with a RigidBody (and characteristics) while making the collider only the head (which is the part that interacts with the other elements).



4.4.1 Figure- Unity collider simulate piston

Another important function of the collider is the "is trigger" tick. This function makes it possible to have a variable in the associated scripts that is triggered when something comes into contact with the collider.

4.5. Materials and Physical Materials

Material

By material is meant the aesthetic appearance of the object, so it is possible to associate a colour, define how metallic an object is (for reflection) and whether it emits light. It is possible to assign a texture.

Physical material

A physical material, on the other hand, allows you to assign physical characteristics to the material such as the dynamic and static friction coefficient.

It is therefore possible to simulate different friction coefficients for each object.

4.6. Other key elements

Before exporting the file, cameras and lights must be added.

The cameras are highly modifiable and can be adjusted in many ways with the Inspector. You can place them at fixed points or create the first-person view and move it as if you were in the environment. In this project, several cameras are positioned and it is possible to switch between them using scripts and a button made dependent on the perspective of the camera (UI canvas).

5. Scripts

5.1. Movement2

The scripts are written in C#.

The standard graphics systems and libraries used by Unity are called up automatically.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Other actions or libraries can be called in this part of the script.

These libraries allow the use of `public` and `private` variables.

Private variables are variables inside the script that cannot be read externally.

Public are variables that are read/modified outside the script. It is possible declare, for example, a `public GameObject` allowing the GameObject to be assigned later, making the script very flexible and reusable.

Again, it is possible define the position characteristics in space of an object as a `public Transform`.

For a better understanding, let's look at the "Movement2" script, which allows the movement of an object from point A to point B.

```
public class Movement2 : MonoBehaviour
{
    public Transform targetArrivo;    //B
    public Transform targetTorno;    //A
    public float speed;
    public bool input=false;
    public bool indietro=false;
```

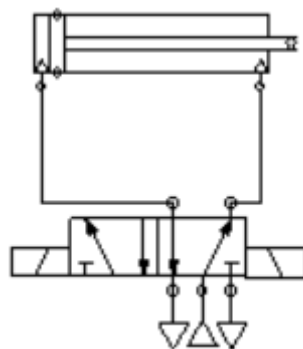
```

void FixedUpdate()
{
    if (input==true)
    {
        // Debug.Log("moving");
        Vector3 a= transform.position;
        Vector3 b= targetArrivo.position;
        speed=0.6f;
        transform.position =Vector3.MoveTowards(a,b,speed);
    }

    if (indietro==true)
    {
        //Debug.Log("come back");
        Vector3 a= transform.position;
        Vector3 b= targetTorno.position;
        speed=0.6f;
        transform.position =Vector3.MoveTowards(a,b,speed);
    }
}
}

```

Script 1



5.1.0 Figure- Pneumatic diagram of script 5/2 two solenoids

MonoBehaviour is used for standard scripts that do not need to communicate outside Unity and Visual studio.

As mentioned, this script is used for movement from point A to point B. Instead of creating a different script for each such movement, writing each arrival and return co-ordinate by hand, two public variables were used.

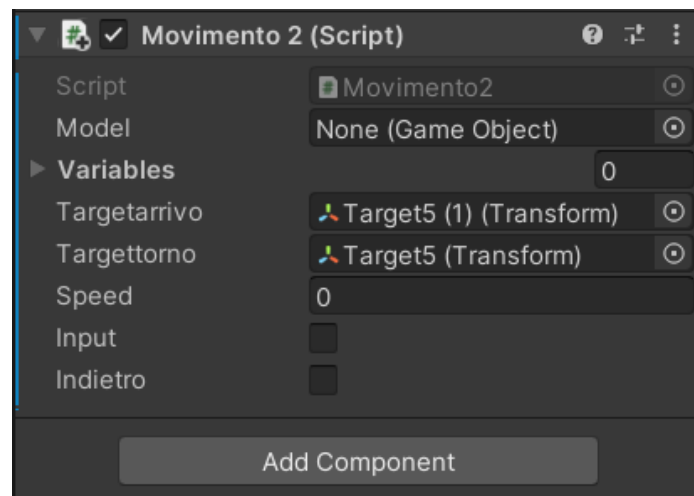
```

public Transform targetArrivo;    //B
public Transform targetTorno;    //A

```

The **Transform** typology makes it possible to extrapolate spatial characteristics, such as coordinates and degrees of relationship, from an assigned GameObject.

By associating this script to an object, the inspector will request references to these two variables. It has been chosen to create two empty objects by placing their centres at the desired positions and dragged their names from the scene section into the inspector.



5.1.1 Figure- Movimento 2 Inspector

It is therefore possible to use a single script for all the linear movements of each object. It is possible to drag the script to any desired inspector and create empty objects indicating the movement positions.

Thanks to the use of standard C# libraries, as shown in script 1, it is possible to use functions such as: `void FixedUpdate()`.

This function, like `void Update()`, is activated every frame of execution. The difference between the two is that `FixedUpdate` uses the physical engine of the program while `Update` uses the graphical engine.

In this case it is used because we want the piece to move at any moment of execution if the variables change.

It possible to observe:

```
if (input==true)
{
    // Debug.Log("moving");
    Vector3 a= transform.position;
    Vector3 b= targetArrivo.position;
    speed=0.6f;
    transform.position =Vector3.MoveTowards(a,b,speed);
}
```

Extract of Script 1

When *Input* is activated, the script saves in "a" the current position of the workpiece with which the script is associated. In "b" it saves the position of the previously positioned empty object.

Vector3 is a standard vector that contains the coordinates of the object.

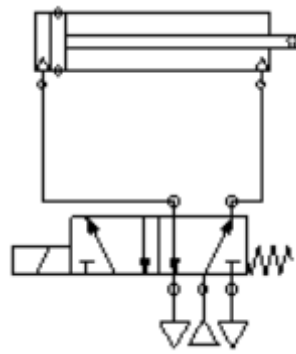
The last line of code transforms the current position "a" into "b" by interpolating it with a speed of 0.6f (float).

5.2. Movement1 with Led

Script 2 uses the same concepts as previously explained, adding the switching on of a positional LED.

```
public class Movement1 : MonoBehaviour
{
    public Transform targetarrivo;
    public Transform targettornio;
    public float speed;
    public bool input = false;
    public bool splus = false;
    public bool sminus = false;
    public GameObject sensorePiu;
    public GameObject sensoreMeno;

    void FixedUpdate()
    {
        if (input)
        {
            Vector3 a = transform.position;
```

5.1.0 Figure- Pneumatic Script Scheme with 5/2-Way Valve

It is possible to see that the SetActive(true) command is used to switch on the LED.

The LEDs are declared as public objects and must therefore be associated with an element in the Inspector.

5.3. MovimentoVacuum

Script 3 is another motion script, associated with the vacuum piston. Having to go down to different levels, it is decided to use a script that moves along the z-axis until a collider detects something.

```
public class MovimeventoVacuum : MonoBehaviour
{
    public Transform targetarrivo;
    public float speed;
    public bool input=false;
    public bool indietro=false;
    public bool contatto=false;
    public Transform targettorno;
    void FixedUpdate()
    {
        if (input==true & contatto==false)
        {
            Vector3 a= transform.position;
            Vector3 b= targetarrivo.position;
            speed=0.6f;
            transform.position =Vector3.MoveTowards(a,b,speed);
        }
    }
}
```



```

        if (indietro==true)
        {
            Vector3 a= transform.position;
            Vector3 b= targettorno.position;
            speed=0.6f;
            transform.position =Vector3.MoveTowards(a,b,speed);
        }
    }
private void OnTriggerEnter (Collider other){
    contatto=true;
}
private void OnTriggerExit (Collider other){
    contatto=false;
}
}

```

Script 3

5.4. Rotation

In script 4 it is possible to see the last type of movement, not linear but rotational.

```

public class Rotazione : MonoBehaviour
{
    public bool rot=false;
    {
        if(rot==true){
            transform.Rotate(Vector3.up, 100*Time.deltaTime);
            Invoke("stopp",1.8f);
        }
    }
    void stopp()
    {
        rot=false;
    }
}

```

Script 4

It is possible to see that on activation of the variable "rot" the piece to which script 4 is associated will rotate one hundred times faster than the unit of time.

```
Invoke("stopp",1.8f);
```

This line of code makes it possible to call a personal function after 1.8f, taking into account the speed of rotation, obtaining a rotation of 180 degrees at the end of which the function stopp() will be called. It is not possible to declare functions with standard names because they probably already exist in the library, for example it is not possible use Stop().

This script is not recommended as it is controlled by a timer but acts in frames, which causes latency and unintended bugs. The proposed solution is:

```
void Update()
{
    if(rot==true)
    {
        rota=true;
    }
    if(rota==true && memoria==false)
    {
        rota=false;
        memoria=true;
        StartCoroutine( Rotate(Vector3.up, 180, 1.0f) );
    }
    if(rot==false)
    {memoria=false;
    rota=false;}
    private void Start ()
    {}
    IEnumerator Rotate( Vector3 axis, float angle, float duration = 1.0f)
    {
        Quaternion from = transform.rotation;
        Quaternion to = transform.rotation;
        to *= Quaternion.Euler( axis * angle );

        float elapsed = 0.0f;
        while( elapsed < duration )
        {
            transform.rotation = Quaternion.Slerp(from, to, elapsed / duration );
            elapsed += Time.deltaTime;
            yield return null;
        }
        transform.rotation = to;
    }
}
```

Script4.1

Quaternions are then used to obtain a rotation of 180 degrees each time "rot" becomes TRUE. This solves a number of latency problems.

5.5. Deviation wall appearance

```
public class Movimento_slitta : MonoBehaviour
{
    void Start()
    {
        GetComponent<BoxCollider>().isTrigger=true;
        GetComponent<Rigidbody>().isKinematic=false;
    }
    private void OnTriggerEnter(Collider other)
    {
        GetComponent<Rigidbody>().isKinematic=false;
    }
    private void OnTriggerExit(Collider other)
    {
        GetComponent<BoxCollider>().isTrigger=false;
        GetComponent<Rigidbody>().isKinematic=true;
    }

    void riprendiTriggher()
    {
        GetComponent<BoxCollider>().isTrigger=true;
    }
}
```

Script 5

When the Collider is crossed an invisible wall is activated which adjusts the trajectory of the cylinder.

5.6. Optical sensors simulation.

In order to simulate an optical sensor, a cube is created and placed where the position sensor is. Script 6 was associated with the cube.

```
public class Detection1 : MonoBehaviour
{
    public GameObject objectToActivateAndDeactivate;
    public bool inside = false;
    public bool red = false;
    public bool white = false;
    public bool shorrt = false;
    public bool tall = false;
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Cylinder1")
        {
            inside = true;
            white = true;
        }
        if (other.tag == "Cilindro2")
        {
            inside = true;
            red = true;
        }
        if (other.tag == "cilindro3Alto")
        {
            inside = true;
            tall = true;
        }
        if (other.tag == "Cilindro4basso")
        {
            inside = true;
            shorrt = true;
        }
    }
    private void OnTriggerExit(Collider other)
    {
        inside = false;
        shorrt = false;
        tall = false;
        white = false;
        red = false;
    }
}
```

```

void Update()
{
    if (inside == false)
    {
        objectToActivateAndDeactivate.SetActive(false);
    }
    if (inside == true)
    {
        objectToActivateAndDeactivate.SetActive(true);
    }
}
}

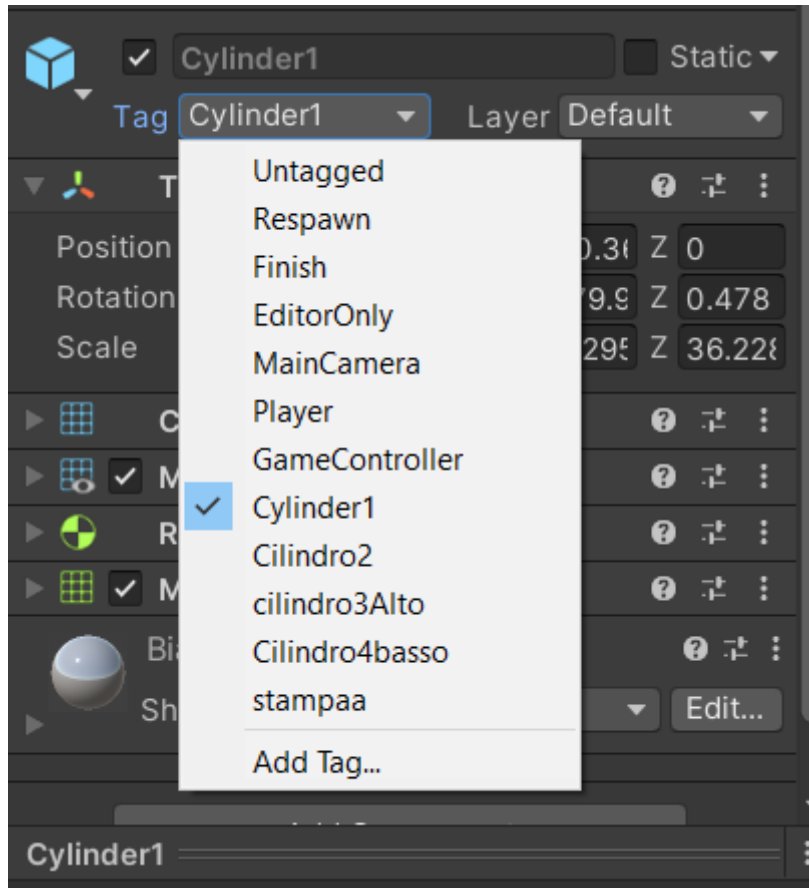
```

Script 6

It can be seen that when something enters the cube:

```
private void OnTriggerEnter(Collider other)
```

is analysed "other" in the script, in order to check which object it is and give an appropriate signal. To do this, a tag has been associated with the cylinders using the Inspector (image 6). it is possible to simulate an optical sensor in a more realistically way changing in an int value instead of a boolean.



5.6.0 Figure-Tag

5.7. Potentiometric sensors simulation.

In order to simulate the behaviour of a height sensor, script 6 is used to calculate the height difference that the sensor undergoes when it comes into contact with the workpiece.

```
Public class Altezza : MonoBehaviour
{
    public GameObject damisurare;
    public float dist=0;
    private float curPos=0;
    private float nuovaPos;
    void Start ()
    {
        curPos = damisurare.transform.position.y;
    }
    void Update ()
```

```

{
    if (curPos<damisurare.transform.position.y)
    {
        nuovaPos=damisurare.transform.position.y;
    }
    dist=(nuovaPos-curPos);
    dist=dist*11.037f;
    if(dist<-280)
    {
        dist=0;
    }
    if(dist>=8){
        Debug.Log(dist);
    }
}
}

```

Script 6

5.8. Conveyor

Script 7 associated with the conveyor belt allows the movement of what is above it, after the activation of the variable "startbelt". The endpoint object is an empty GameObject on the other side of the conveyor belt.

```

public class Belt : MonoBehaviour
{
    public GameObject conveyor;
    public float speed;
    public Transform endpoint;
    public bool startbelt=false;
    void OnTriggerStay(Collider other)
    {
        speed=20;
        if(startbelt==true){
            other.transform.position=Vector3.MoveTowards(other.transf
orm.position,endpoint.position,speed*Time.deltaTime);
        }
    }
}

```

Script 7

5.9. Polytechnic's symbol, moulding

In order to make the polytechnic symbol appear when the piston is touched, script 8 is associated with the symbol:

`GetComponent<Renderer>().enabled` makes the object visible.

```
public class Stamp : MonoBehaviour
{
    public bool trigger=false;
    public Renderer rend;

    void Start () {
        rend = GetComponent<Renderer>();
        rend.enabled = false;
    }
    private void OnTriggerEnter (Collider other){
        if (other.tag== "stampaa")
        {
            rend.enabled = true;
            var politico= gameObject.GetComponent<BoxCollider>();
            politico.enabled=false;
        }
    }
}
```

Script 8

“Var politico” is used to prevent the vacuum piston from touching the symbol but continuing to the cube.

5.10. Suction Cup simulation

A force could be created to simulate the vacuum suction of the object. But this is not the solution followed. In fact, script 9 uses the change of parentage and subordination to make the cylindrical object temporarily a child of the Vacuum (until the stop variable is activated).

```
public class Vacuum : MonoBehaviour
{
    GameObject cube;
    GameObject child;
    public bool stopvacuum;
    public bool contatto;
    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.CompareTag("Cylinder1") || other.gameObject
        .CompareTag("Cilindro2") || other.gameObject.CompareTag("cilindro3Al
        to") || other.gameObject.CompareTag("Cilindro4basso"))
        {
            //Debug.Log ("triggerato");
            child=other.gameObject.transform.GetChild (0).gameObject
;

            contatto=true;
            cube=other.gameObject;

            other.transform.SetParent(transform.parent);
            cube.GetComponent<Rigidbody>().isKinematic=true;
        }
        private void Update()
        {
            if(stopvacuum==true){
                cube.GetComponent<Rigidbody>().isKinematic=false;
                child.transform.SetParent(cube.transform);
                cube.transform.SetParent(genitore.transform);
                contatto=false;
            }
        }
    }
}
```

Script 9

The script transforms "other" (the object it collides with) into a child of the object that the script possesses. At the same time, it modifies some of its characteristics.

5.11. SW

The script 10, allows to create the switch variables.

```
public class SensorFinecorda : Item
{
    public bool sensore;
    private void Start()
    {
        sensore = false;
    }
    void Update()
    {
        variables[0].BoolValue = sensore;
    }
    private void OnTriggerEnter(Collider other)
    {
        sensore = true;
        Debug.Log("SensorePosizione");
    }
    private void OnTriggerExit(Collider other)
    {
        sensore = false;
    }
}
```

Script 10

5.12. Cameras

After adding the cameras, a script is created to allow the change of view between them. The Canvas method was used to create a button which is visible from all camera perspectives.



5.12.0 Figure-UI canvas

```
using UnityEngine.EventSystems;
This line of code allows Unity to use the Canvas system.
public class Button1 : MonoBehaviour, IPointerDownHandler, IPointerUp
Handler
{
    private bool ispressed=false;
    public Camera[] cameras;           //array
    private int currentCameraIndex;
    bool memoria=false;
    //Turn all cameras off, except the first default one
    void Start () {
        currentCameraIndex = 0;
        for (int i=1; i<cameras.Length; i++)
        {
            cameras[i].gameObject.SetActive(false);
        }
        if (cameras.Length>0)
        {
            cameras [0].gameObject.SetActive (true);
            Debug.Log ("Camera with name: " + cameras [0].GetCompon
ent<Camera>().name + ", is now enabled");
        }
    }
}
```

```

    }
}
void Update () {
    //If the button is pressed, switch to the next camera
    //Set the camera at the current index to inactive, and set the next
    one in the array to active
    //When we reach the end of the camera array, move back to the beginn
    ing or the array.
    if (ispressed==true & memoria==false)
    {
        memoria=true;
        currentCameraIndex ++;
        Debug.Log ("button has been pressed. Switching to the n
ext camera");
        if (currentCameraIndex < cameras.Length)
        {
            cameras[currentCameraIndex-
1].gameObject.SetActive(false);
            cameras[currentCameraIndex].gameObject.SetActive(tr
ue);
            Debug.Log ("Camera with name: " + cameras [currentC
ameraIndex].GetComponent<Camera>().name + ", is now enabled");
        }
        else
        {
            cameras[currentCameraIndex-
1].gameObject.SetActive(false);
            currentCameraIndex = 0;
            cameras[currentCameraIndex].gameObject.SetActive(true
);
            Debug.Log ("Camera with name: " + cameras [currentCame
raIndex].GetComponent<Camera>().name + ", is now enabled");
        }
    }
}

public void OnPointerDown(PointerEventData eventData)
{
    ispressed=true;
}

public void OnPointerUp(PointerEventData eventData)
{
    ispressed=false;
    memoria=false;
}
}

```

Script 11

Script 10 defines an Inspector expandable vector in which the Start() resets the index and each time the button is activated it sets and resets the next camera. Start() is a function that is only triggered the first time the script is activated.

5.13. Camera movement

Script 12 allows camera movement during execution.

```
public class CameraMouse : MonoBehaviour
{
    public float speedH = 0.6f;
    public float speedV = 0.6f;

    private float yaw = 0.0f;
    private float pitch = 0.0f;

    void Update()
    {
        yaw += speedH * Input.GetAxis("Mouse X");
        pitch -= speedV * Input.GetAxis("Mouse Y");

        transform.eulerAngles = new Vector3(pitch, yaw, 0.0f);
    }
}
```

Script 12

5.14. Random Spawn

Another UI button with Canvas method is created to spawn a random object from a collection.

Then a folder containing the four objects is created in the Asset.

Finally, script 13 is associated.

```
using UnityEngine.EventSystems;
```

This line of code allows Unity to use the Canvas system.

```
public class RandomSpawn : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    private bool ispressed=false;
    bool memoria=false;
    public GameObject[] spawnees;
    public Transform spawnPos;
    public bool tempo=false;
    int randomInt;
    void Update () {
        if(ispressed==true & memoria==false & tempo==false) {
            SpawnRandom();
            tempo=true;
            Invoke("wait",3f);
        }
    }

    void SpawnRandom() {
        randomInt = Random.Range(0, spawnees.Length);
        Instantiate(spawnees[randomInt], spawnPos.position, spawnPos
.rotation);
    }

    void wait()
    {
        tempo=false;
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        ispressed=true;
    }
    public void OnPointerUp(PointerEventData eventData)
    {
        ispressed=false;
        memoria=false;
    }
}
```

```
}  
}
```

Script 13

The wait() function is a personal "safety" function to avoid overload due to pressing the button too many times. An empty GameObject is used to establish the Spawn point.

The line of code:

```
public void OnPointerDown(PointerEventData eventData)
```

allows you to act on an event, in particular that of pressing the UI pushbutton.

Conversely, the next function allows you to control the actions on the event when it is released:

```
public void OnPointerUp(PointerEventData eventData)
```

To use these functions it is necessary to declare them:

```
public class RandomSpawn : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
```

5.15. AS and Unity connection

First of all, the "Developer Package V1.0 3D System" extension provided by the company must be installed.

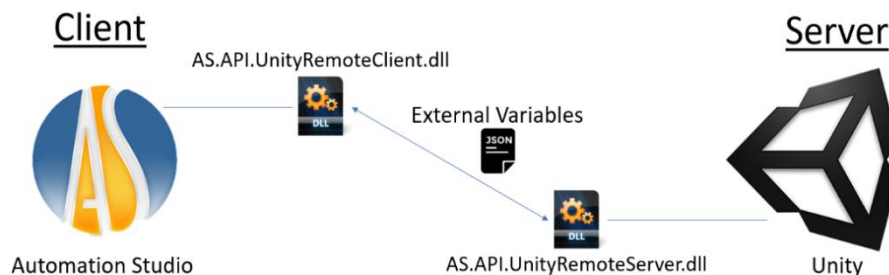
The URC and URS system is used to exchange data between the two programmes. The acronym URC stands for UnityRemoteClient and the acronym URS stands for UnityRemoteServer. The Automation Studio™ project acts as the client and the Unity project acts as the server.

The URC – URS are optional modules of Automation Studio™. They:

- Allow configuration of data exchange
- Allow consistency of operation with other modules of Automation Studio™
- Provide components that enable data exchange

The URC – URS allow the exchange of information between Automation Studio™ and Unity and form a connection between local Variables using External Variables in a JSON file.

Each External Variable essentially connects a Variable on the client side to a Variable on the server side. Each External Variable is given a Role of either Import or Export that defines whether their values are managed by the Client or Server.



5.15.0 Figure-UI canvas

It is required to import AS.API.UnityRemoteServer.dll in the Asset.

5.16. Unity variables/AS

After adding the .dll file, the scripts from which the control variables are extracted are modified.

The first change to be made is to call the library in the AS.API.UnityRemoteServer.dll file in order to use it in the script.

```
using AS.API.UnityRemoteClient;
```

A type of Awake() function must be added; this is activated regardless of whether or not the script is used at the start of execution.

```
#region ***Automation Studio variable creation***
//This section is where variables used to communicate between Un
ity and Automation Studio are created.
void Awake()
{
    if (variables.Count == 0)
    {
        this.Model = gameObject;

        variables.Add(new ExternalVariable("3D_" + name + "_Ew",
ExternalVariable.Role.Import, ExternalVariable.Type.Bool));
        variables.Add(new ExternalVariable("3D_" + name + "_S+",
ExternalVariable.Role.Export, ExternalVariable.Type.Bool));
    }

    VariableBinder.SendItemToBinder(this);
}
#endregion
```

Script 14

Script 13 shows the section defining the variables to be read by AS.

Note the difference between:

`ExternalVariable.Role.Import` and `ExternalVariable.Role.Export` the "import" extension allows AS to have that variable as input, while "export" only allows AS to read its value.

`ExternalVariable.Type.Bool` if it is boolean, and `.Float` or `.Int` for other variables.

Typing:

```
new ExternalVariable("3D_" + name + "_Ew" ...)
```

makes it possible to add prefix and suffix variables. The prefix of the transferred variable will be "3D_" to facilitate searching in AS, the main name will be the name of the object to which it is linked and the suffix will be an identification name chosen by the user.

It is clear that it is necessary to have access to these variables at each frame. it is necessary to use the basic `Update()` function.

```
void Update()
{
    input = variables[0].BoolValue //Import
    variables[1].BoolValue = splus; //Export
}
```

Script 15

Is very important the logic with write import/export variable in `Update()`.

Example rewrite:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using AS.API.UnityRemoteClient;

public class Movimento2 : Item
{
    public Transform targetarribo;
    public Transform targettornio;
    public float speed;
    public bool input=false;
    public bool indietro=false;

    #region ***Automation Studio variable creation***
    void Awake()
    {
        if (variables.Count == 0)
```

```

    {
        this.Model = gameObject;
        variables.Add(new ExternalVariable("3D_" + name + "_Ew+"
, ExternalVariable.Role.Import, ExternalVariable.Type.Bool));
        variables.Add(new ExternalVariable("3D_" + name + "_Ew-
", ExternalVariable.Role.Import, ExternalVariable.Type.Bool));
    }
    VariableBinder.SendItemToBinder(this);
}
#endregion

void FixedUpdate()
{
    if (variables.Count > 0)
    {
        input = variables[0].BoolValue;
        indietro=variables[1].BoolValue;
        if (input==true)
        {
            Vector3 a= transform.position;
            Vector3 b= targetarrivo.position;
            speed=0.6f;
            transform.position =Vector3.MoveTowards(a,b,speed);
        }
        if (indietro==true)
        {
            Vector3 a= transform.position;
            Vector3 b= targettornio.position;
            speed=0.6f;
            transform.position =Vector3.MoveTowards(a,b,speed);
        }
    }
}
}

```

Script 1.1

```

public class Rotazione : Item
{
    public bool rot=false;
    #region ***Automation Studio variable creation***
    void Awake()
    {
        if (variables.Count == 0)
        {
            this.Model = gameObject;

```

```

        variables.Add(new ExternalVariable("3D_" + name + "_rot"
, ExternalVariable.Role.Import, ExternalVariable.Type.Bool));

    }

    VariableBinder.SendItemToBinder(this);
}
#endregion

void Update()
{
    rot = variables[0].BoolValue;

    if(rot==true){
        transform.Rotate(Vector3.up, 100*Time.deltaTime);
        Invoke("stopp",1.8f);
    }
}
private void Start () {}
void stopp()
{
    rot=false;
}
}

```

Script 4.1

5.17. PUSHBUTTON Unity/AS

Finally, script 16 shows how to create a Pushbutton in Unity and read it from AS, like an HMI. This script combines all the techniques seen so far:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using AS.API.UnityRemoteClient;

public class VariabileOneshot : Item, IPointerDownHandler, IPointerUpHandler
{
    private bool vair=false;
    public bool trig=false;
    void Start()
    {

    }

    #region ***Automation Studio variable creation***
    void Awake()
    {
        if (variables.Count == 0)
        {
            this.Model = gameObject;
            variables.Add(new ExternalVariable("3D_" + name + "_var",
ExternalVariable.Role.Export, ExternalVariable.Type.Bool));
        }

        VariableBinder.SendItemToBinder(this);
    }
    #endregion

    void Update()
    {
        variables[0].BoolValue=trig;
        if(vair){trig=true;}else{trig=false;}
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        vair=true;
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        vair=false;
    }
}
```

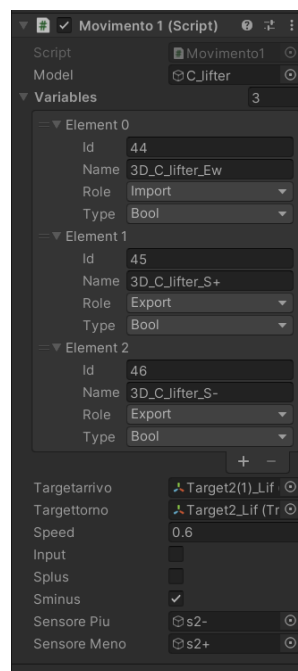
```
}
}
```

Script 16

Four Pushbuttons are created with this technique, two to create HMI buttons (stop and restart), two for recall from vertical magazines.

5.18. DEBUG

After the creation of the communication between the variables, it is not possible to debug. It is possible, however, check the names and creations of the variables by going to the Inspector of the GameObject that owns the script.



5.15.0 Figure-Inspector Variables

5.19. Variables create:

The following variables are then extracted from these scripts:

Components	Name AS	Type in Unity	I/O	Status0
HMI WH1	3D_WH1_var	Export	I	
HMI WH2	3D_WH2_var	Export	I	
HMI Stop	3D_stop_var	Export	I	
HMI Restart	3D_restart_var	Export	I	
Piston A	3D_A_piston_Ew	Import	O	
	3D_A_piston_S+	Export	I	
	3D_A_piston_S-	Export	I	On
Piston B	3D_B_piston_Ew	Import	O	
	3D_B_piston_S+	Export	I	
	3D_B_piston_S-	Export	I	On
Lifter C	3D_C_lifter_Ew	Import	O	
	3D_C_lifter_S+	Export	I	
	3D_C_lifter_S-	Export	I	On
M_slider	3D_M_slider_Ew	Import	O	
	3D_M_slider_S+	Export	I	
	3D_M_slider_S-	Export	I	On
R	3D_R_UEw+	Import	O	
	3D_R_UEw-	Import	O	
Laser1y	3D_laser1y	Export	I	On
Laser2y	3D_laser2y	Export	I	
Laser3y	3D_laser3y	Export	I	
Laser4y	3D_laser4y	Export	I	

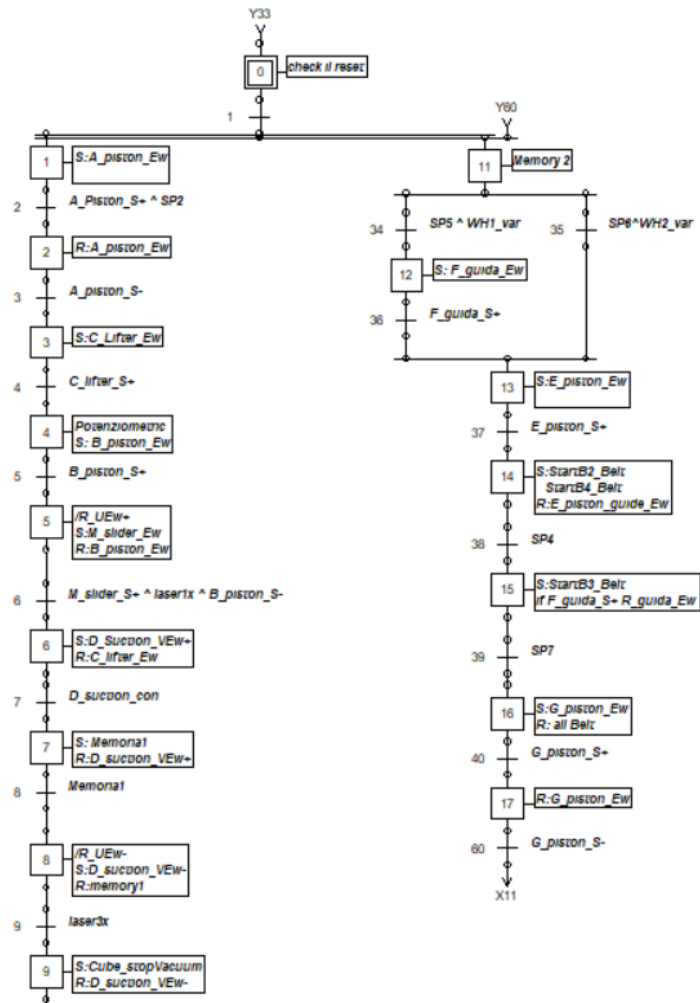
P_motor	3D_P_motor_Uew+	Import	O	
	3D_P_motor_Uew-	Import	O	
V	3D_V_rot	Import	O	
K_stamp	3D_K_stamp_Ew	Import	O	
	3D_K_stamp_S+	Export	I	
	3D_K_stamp_S-	Export	I	On
D_suction	3D_D_-suction_Ew	Import	O	
	3D_D_-suction_S+	Export	I	
	3D_D_-suction_S-	Export	I	On
	3D_D_-suction_VEw+	Import	O	
	3D_D_-suction_VEw-	Import	O	
	3D_D_-suction_con	Export	I	
	3D_Cube_stopVacuum	Import	O	
G_piston	3D_G_piston_Ew	Import	O	
	3D_G_piston_S+	Export	I	
	3D_G_piston_S-	Export	I	On
F_guide	3D_F_guida_Ew	Import	O	
	3D_F_guida_S+	Export	I	
	3D_F_guida_S-	Export	I	On
E_piston	3D_E_piston_guide_Ew	Import	O	
	3D_E_piston_guide_S+	Export	I	
	3D_E_piston_guide_S-	Export	I	On
Potenziometric	3D_O_piezometric_dist	Export Float	I	
Conveyor 2	3D_StartB3_Belt	Import	O	
Conveyor 1	3D_StartB2_Belt	Import	O	
Conveyor 3	3D_StartB4_Belt	Import	O	
SP1	3D_SP1_i	Export	I	

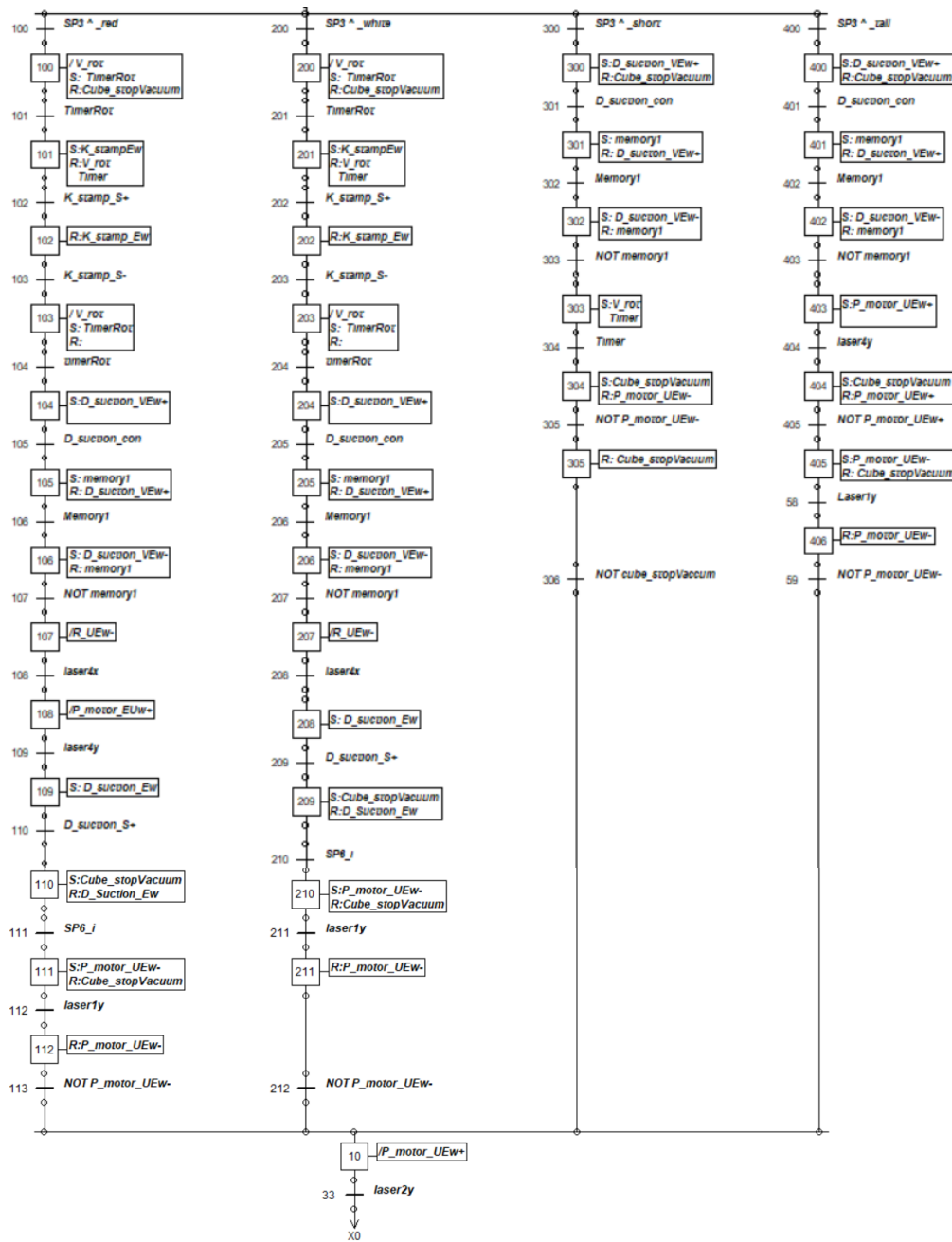
SP2	3D_SP2_i	Export	I	
SP3	3D_SP3_i	Export	I	
SP4	3D_SP4_i	Export	I	
SP5	3D_SP5_i	Export	I	
SP6	3D_SP6_i	Export	I	
SP7b	3D_SP7b_i	Export	I	
Laser2x	3D_laser2x	Export	I	
Laser1x	3D_laser1x	Export	I	
Laser3x	3D_laser3x	Export	I	
Laser4x	3D_laser4x	Export	I	On
Sensor	3D_red	Export	I	
	3D_white	Export	I	
	3D_short	Export	I	
	3D_tall	Export	I	

5.19.0 Table Variables

6. Siemens

6.1. Grafcet





6.1.0 Grafset

Image 6.1.0 represents the logic diagram of programming in the form of Grafset. The grafset allows to have explicit and schematized the actions to be performed (in the squares) and the conditions of transition from one action to another. It can be observed that there are two independent and parallel paths. Finally, can be observed the four branches and different paths based on sensor decisions.

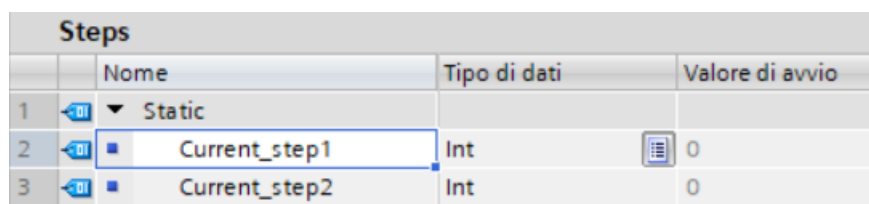
6.2. TiaPortal

The image 6.1.0 *Grafcet* has been translated into ladder language.

A variant of the batch technique has been used.

The batch technique requires the use of a memory for each executive step in order to have strict conditions on the execution sequence.

Two variables/memory of type INT are then created, one for each parallel path.



The screenshot shows a table titled 'Steps' with four columns: an index column, a 'Nome' column, a 'Tipo di dati' column, and a 'Valore di avvio' column. There are three rows. Row 1 has index '1', a dropdown arrow, and the text 'Static'. Row 2 has index '2', a square icon, the text 'Current_step1', 'Int' in the data type column, and '0' in the start value column. Row 3 has index '3', a square icon, the text 'Current_step2', 'Int' in the data type column, and '0' in the start value column. The 'Current_step1' row is highlighted with a blue border.

		Nome	Tipo di dati	Valore di avvio
1	▼	Static		
2	■	Current_step1	Int	0
3	■	Current_step2	Int	0

6.2.0 Memory of steps

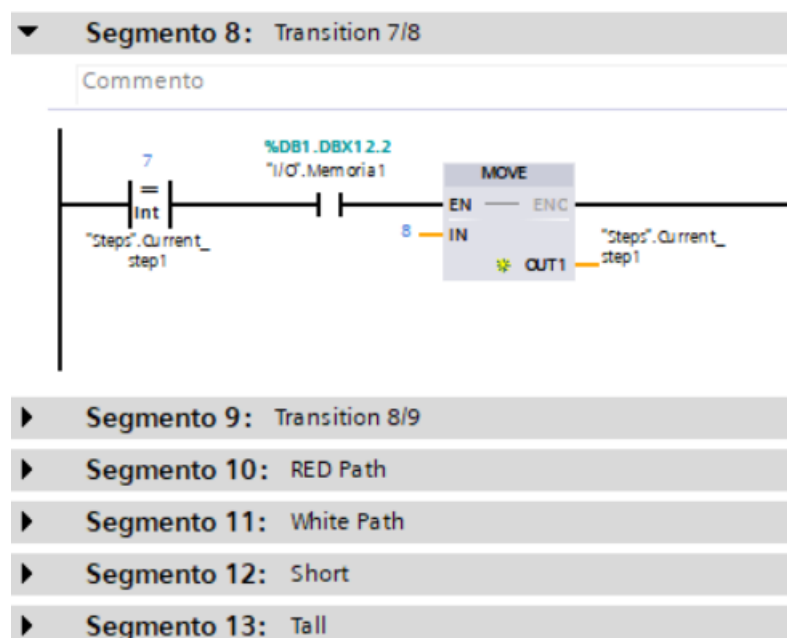
The programming logic is to increment the value of "current step" to match each value with an action.

Another data block has been created in which all the I/O are inserted.

I/O				
		Nome	Tipo di dati	Offset
3		WH2_var	Bool	0.1
4		Stop_var	Bool	0.2
5		restart_var	Bool	0.3
6		A_piston_Ew	Bool	0.4
7		A_piston_S+	Bool	0.5
8		A_piston_S-	Bool	0.6
9		B_piston_Ew	Bool	0.7
10		B_piston_S+	Bool	1.0
11		B_piston_S-	Bool	1.1
12		C_lifter_Ew	Bool	1.2
13		C_lifter_S+	Bool	1.3
14		C_lifter_S-	Bool	1.4
15		M_slider_Ew	Bool	1.5
16		M_slider_S+	Bool	1.6
17		M_slider_S-	Bool	1.7
18		R_UEw+	Bool	2.0
19		R_UEw-	Bool	2.1
20		Laser1y	Bool	2.2
21		Laser2y	Bool	2.3

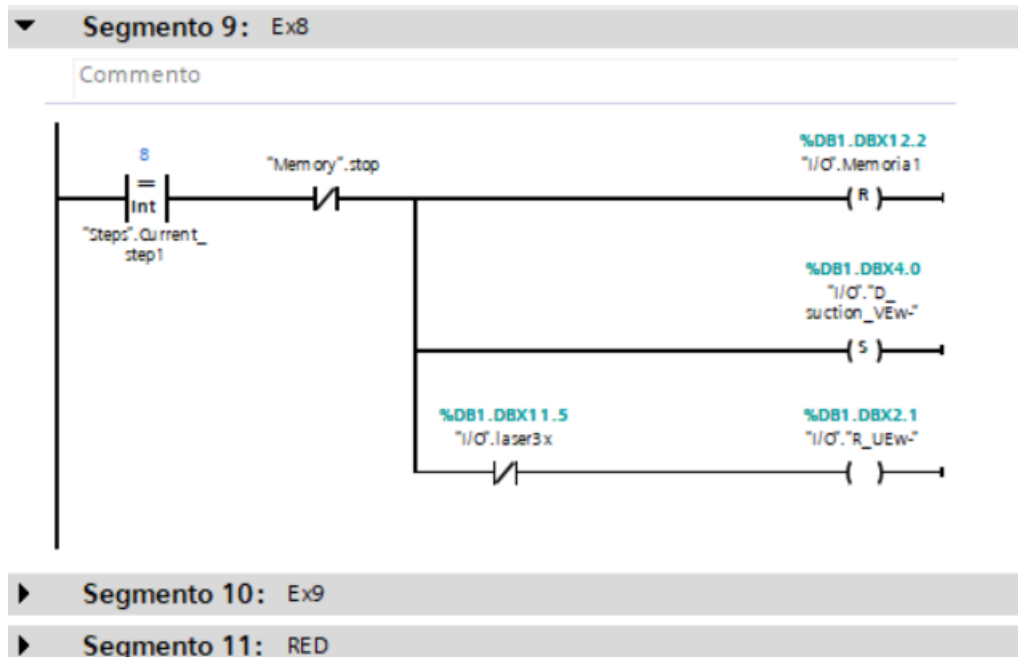
6.2.1 I/O

Several sub-functions have been created. Two control the stages of transitions:



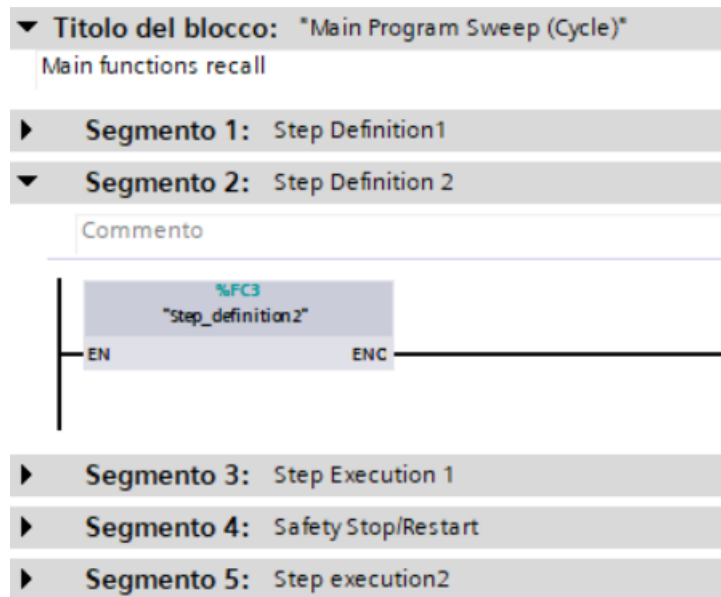
6.2.2 Transition Function

The logic consists of: comparing the value of "current step" with a constant value represented by the previous step, inserting transition conditions and finally a "move" block to update "current step". Image 6.2.2



6.2.3 Execution Function

Image 6.2.3 depicts a step in the execution function block. Then, it compares the value of "current step" with a constant value, if TRUE the set/reset actions are executed.



6.2.4 Execution Function

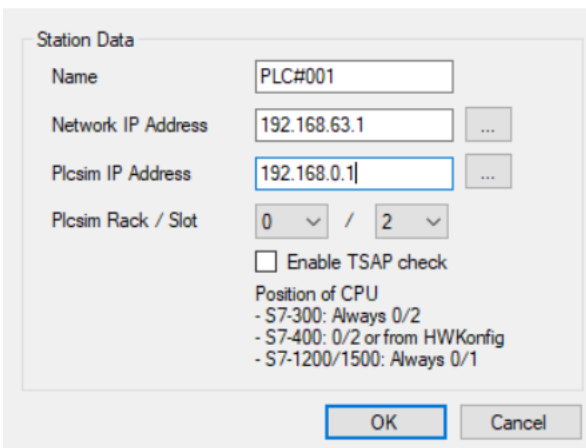
Function blocks and Start/Reset conditions must be inserted in logical order in the main. Imagine 6.2.4

7. OPC

The OPC technique allows the passage of information between three different programs via external servers.

7.1. NetToPLCsim

NetToPLCsim allows you to create a server that receives signals from PLCsim and allows them to be read by third parties.



The screenshot shows a dialog box titled "Station" with a close button (X) in the top right corner. Inside the dialog, there is a section labeled "Station Data" containing the following fields and options:

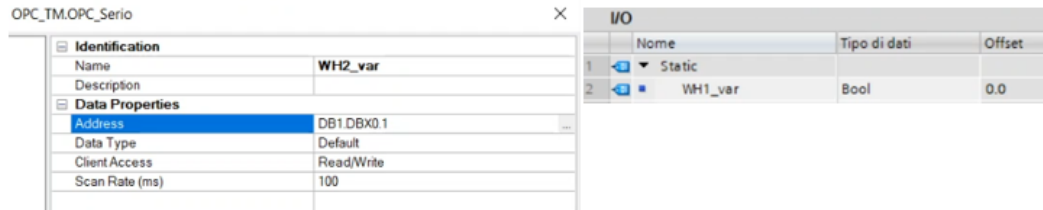
- Name:** A text field containing "PLC#001".
- Network IP Address:** A text field containing "192.168.63.1" with a browse button (three dots) to its right.
- Plcsim IP Address:** A text field containing "192.168.0.1" with a browse button (three dots) to its right.
- Plcsim Rack / Slot:** Two dropdown menus showing "0" and "2" separated by a slash.
- Enable TSAP check:** An unchecked checkbox.
- Position of CPU:** A section with the following text:
 - S7-300: Always 0/2
 - S7-400: 0/2 or from HWKonfig
 - S7-1200/1500: Always 0/1

At the bottom of the dialog are "OK" and "Cancel" buttons.

7.1.0 NetToPLCsim configuration

7.2. KEPServerEx6

By entering the IP address of NetToPLCsim in KEPServerEx6 as input, we can create links to the TiaPortal variables, using offset addresses.



7.2.0 KEPServerEx variables configuration

This operation is done for each variable.

Automation Studio and Siemens cannot connect only with KEPServer.

7.3. Automation studio

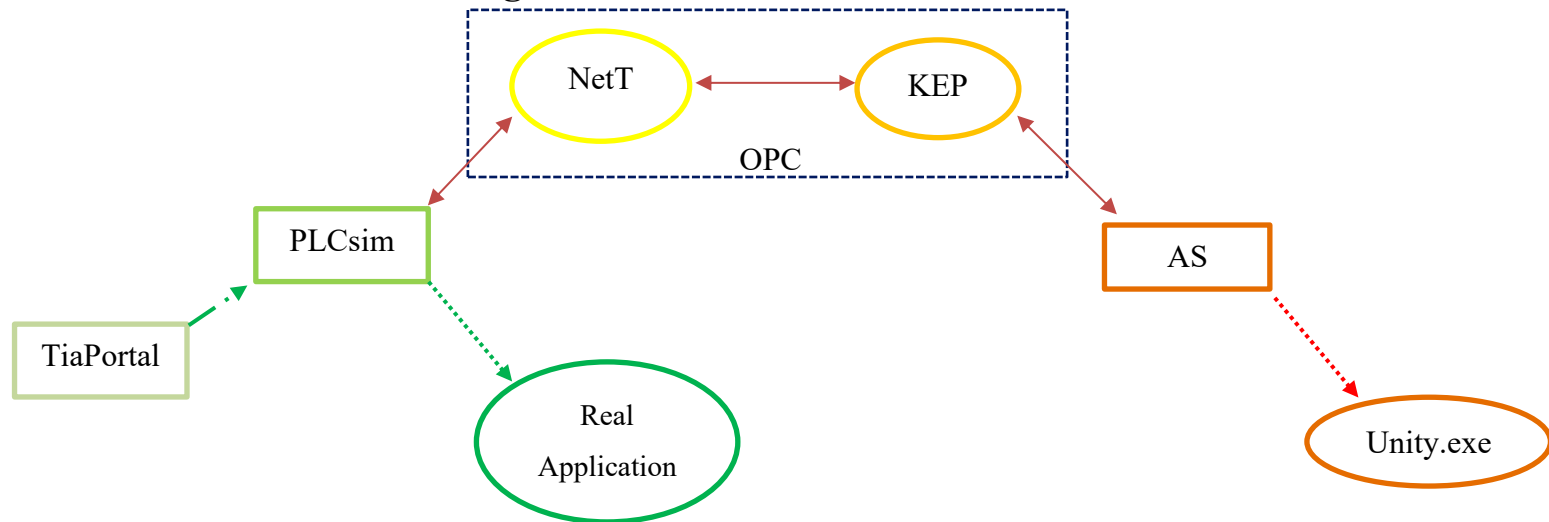
From the point of view of Automation Studio, an OPC can be created from the communication management. Specifying as input server that of the KEPServer. The same number and type of variables have been created.

The variables in Automation Studio are tied to those in the TiaPortal.

The variables are now controlled in Automation Studio using Siemens programs.

8. Conclusion

8.1. Diagram



8.1.0 Conceptual Diagram

Diagram 8 summarises the thesis process and what can be done with this technique. In fact, it is possible to control both the simulated model and the real model simultaneously with a single ladder logic program.

So, for particular systems, if the Unity simulation is accurate it is possible to safely test only on the simulation and then apply it without program changes to the real application.

8.2. Advantages

The advantages are that it is possible to operate and get feedback on your automation programme in a realistic environment. It also allows different programmes to be used, covering and compensating for each.

For example, the ladder part is not very comfortable to program in Automation Studio, whereas it is in TiaPortal. On the contrary, the visual part of TiaPortal cannot compete with Automation Studio.

Due to external restrictions (ex.Covid) it is still possible to work on a project without travelling to the site.

8.3. Drawbacks

Such a process is certainly time-consuming. Also, since the ladder logic has no instant feedback, it is not possible to use it to learn the method.

Operating in OPC and connecting different programmes requires considerable computer flash memory and CPU performance.

PLCsim is a soft simulator of a real PLC, so it may not run smoothly.

Possible solutions could be to use an external PLC to perform the logic and in case use an SSD to boost the computer's performance.

9. Bibliography

Pneumatic Control – *Joji P.*

Tiportal manual – *Programmazione dei passi di lavorazione*

Unity User Manual 2020.3

Automation Studio manual

Webnair Famic Thecnology

Alla mia famiglia e i miei amici più vicini,

radici di pietra

di idee ramificate.

Vi devo tutto.

Non sto più nella pelle, mama

Fuori di me,

exuvia,

spiego le ali, au revoir