GRAPE
Ground Robot for vineyArd monitoring and ProtEction

D4.1 - Pheromone dispenser manipulation techniques

Grant Agreement Number	601116 (corresponding to ECHORD++ Project)
Call identifier	FP7-ICT-2011-9
Project Acronym	GRAPE
Project Title	Ground Robot for vineyard monitoring and ProtEction
Funding Scheme	CP - Collaborative project (FP7)
Project Starting date	01/09/2016
Project Duration	18 months
Deliverable Number	4.1
Deliverable Title	Pheromone dispenser manipulation techniques
Nature of Deliverable	R
Dissemination Level	RE
Due date of deliverable	M14
Actual Date of deliverable	01/11/2017
Produced by	Polimi (Ehsan Fathi, Luca Bascetta, Federico Polito)
Validated by	Vitirover (--) Eurecat (Daniel Serrano, Jesús Pablo González) Polimi (Matteo Matteucci, Luca Bascetta)

ECHORD++

The European Coordination Hub for Open Robotics Development

eurecat Centre Tecnològic de Catalunya

VITIROVER MICRO ROBOTIQUE VITICOLE

POLITECNICO MILANO 1863

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement no. 601116

Document change record

Issue Date	Version	Author	Sections affected / Description
17/10/17	0.1	Luca Bascetta	All document – Creation and editing content
19/10/17	0.2	Luca Bascetta	Chapter 1 completed and revised
20/10/17	0.3	Luca Bascetta	Chapter 4 completed and revised
23/10/17	0.4	Luca Bascetta	Chapter 5 completed and revised
23/10/17	0.5	Ehsan Fathi	Chapters 2, 3 completed
25/10/17	0.5.1	Luca Bascetta	Chapters 2, 3 revised
27/10/17	0.6	Luca Bascetta, Federico Polito	Chapter 6 completed
28/10/17	0.6.1	Luca Bascetta	Chapter 6 revised
30/10/17	0.7	Luca Bascetta, Federico Polito	Chapter 7 completed and revised
31/10/17	0.8	Ehsan Fathi	Annex completed and revised
01/10/17	0.8.1	Luca Bascetta	Final revision

Glossary of terms and acronyms

Acronym	
UGV	Unmanned Ground Vehicle
ROS	Robot Operating System
OMPL	Open Motion Planning Library
POLIMI	Politecnico di Milano
IBVS	Image Based Visual Servoing

Content

Executive Summary

Among the GRAPE platform subsystems, the manipulation subsystem has the goal of grasping, manipulating and deploying the pheromone dispensers on the vineyard branches. The main criticalities of this subsystem are related to: (i) the low accuracy and repeatability of the arm, (ii) the size and kinematics of the hand that are not designed for dispenser manipulation, (iii) the fact that dispensers are small and flexible objects. To target the short delivery time of the GRAPE experiment, we adopted a prototype oriented methodology; starting from a baseline system, based on the ROS *MoveIt!* package, we prototyped trajectory planning and execution using proprioceptive sensors for tracking and a laser scanner or a camera for obstacle avoidance, and verifying the system behavior in a laboratory environment constituted by a prototype of a real vineyard. Suitable control algorithms, based on visual servoing and visual impedance, have been then developed in order to cope with the most critical tasks, i.e., dispenser grasping and deploying. Finally, as part of the manipulation system, suitable devices to ease the grasping operation and to house the dispensers on the robot in a graspable configuration, have been designed and realized.

In Chapter 1, we briefly review the arm and sensors used for the GRAPE project recalling the features which are relevant for dispenser grasping, manipulation, and deploying. Chapter 2 describes two algorithms that have been developed to determine the location of the arm on the platform that maximizes the arm manipulability. In Chapter 3 the different designs for the dispenser feeders and grasping devices are reported, explaining the adopted trial and error design procedure. Chapter 4 briefly reviews the two simulation environments that have been used to test the control law and the software. In Chapters 5 and 6 the architecture of the control system, including MoveIt! and the additional parts developed by POLIMI, is introduced, and the visual servoing and visual impedance control law are illustrated in mode details. Finally, Chapter 7 shows a preliminary set of results achieved in a laboratory environment constituted by a 3D printed mocap of a vineyard.

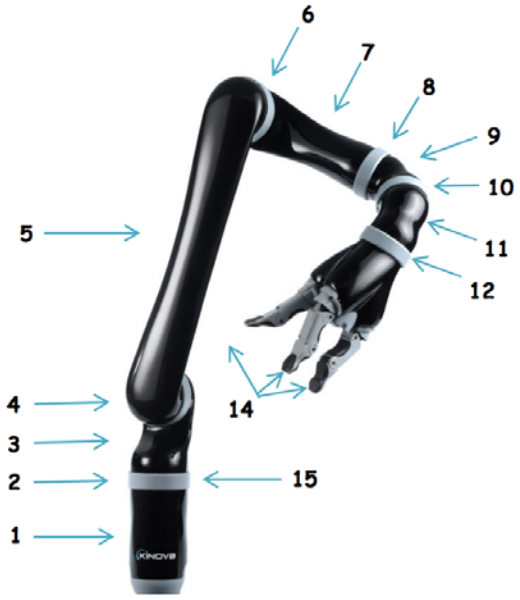
1 – Kinova Jaco 2 arm and sensors

In this chapter, the details of the arm selected for the project, that are relevant for the execution of the GRAPE deployment task are briefly reported, extracted from GRAPE Deliverable *D1.1 - Scenarios and Requirement Specifications*, and from Kinova technical documentation.
D1.1 is also integrated here with the description of the sensors adopted for the manipulation task.

1.2 - Kinova Jaco 2 arm

The arm selected for the GRAPE experiment is the Kinova Jaco2, that was acquired by POLIMI.

Jaco 2 is a 6DOF lightweight robotic arm, made of carbon fiber and aluminum, with a weight of 4.4kg and a reach of 90cm. The arm mounts a 3-finger gripper and is characterized by a non-spherical wrist. The design of the robot that includes the power electronics in the base, the possibility to power the arm with low voltage and the low power consumption makes this robot the best solution for mobile robotics.



PART ID	NAME
1	Controller
2	Actuator #1
3	Shoulder
4	Actuator #2
5	Arm
6	Actuator #3
7	Forearm
8	Actuator #4
9	Wrist #1
10	Actuator #5
11	Wrist #2
12	Actuator #6
13	Gripper
14	Fingers
15	Plastic Ring

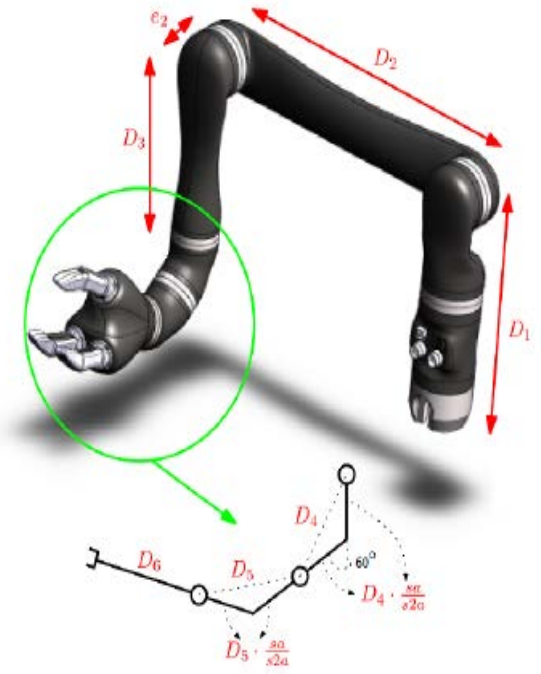
The control system can be re-parametrized and Jaco 2 is fully supported in ROS with community driven Open Source code and examples.

The following tables include the main specifications of the robot and all the parameters required to setup a kinematic/dynamic model of the arm, i.e., Denavit-Hartenberg parameters, link lengths, and inertial parameters.

GENERAL INFO	
TOTAL WEIGHT	5.2 Kg
PAYLOAD	1.6 Kg (mid-range continuous) 1.3 Kg (full-reach peak/temporar)

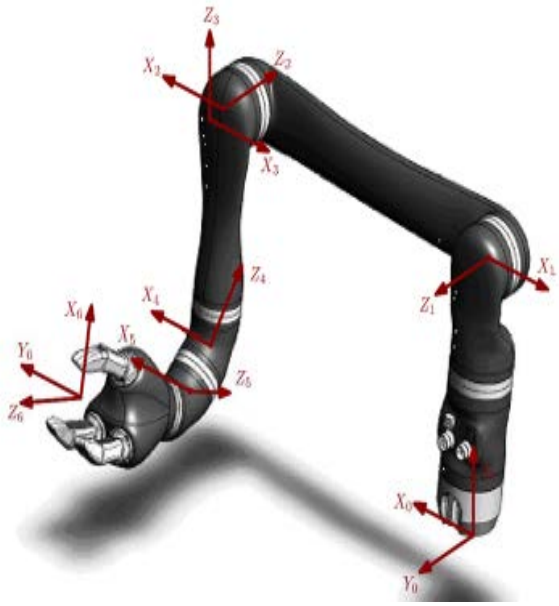
REACH	90 cm
MATERIALS	carbon fiber(links), aluminium(actuators)
JOINT RANGE (SOFTWARE LIMITATION)	±27.7 turns
MAXIMUM LINEAR ARM SPEED	20 cm/s
POWER SUPPLY VOLTAGE	18 to 29 VDC
AVERAGE POWER	25 W (5W in STANDBY)
PEAK POWER	100 W
COMMUNICATION PROTOCOL	RS485
COMMUNICATION CABLES	20 pins flat flex cable
WATER RESISTANCE	IPX2
OPERATING TEMPERATURE	-10 °C to 40 °C (continuous)

Robot length values (meters)		
D1	0.2755	Base to elbow
D2	0.4100	Arm length
D3	0.2073	Front arm length
D4	0.0741	First wrist length
D5	0.0741	Second wrist length
D6	0.1600	Wrist to center of the hand
e2	0.0098	Joint 3-4 lateral offset

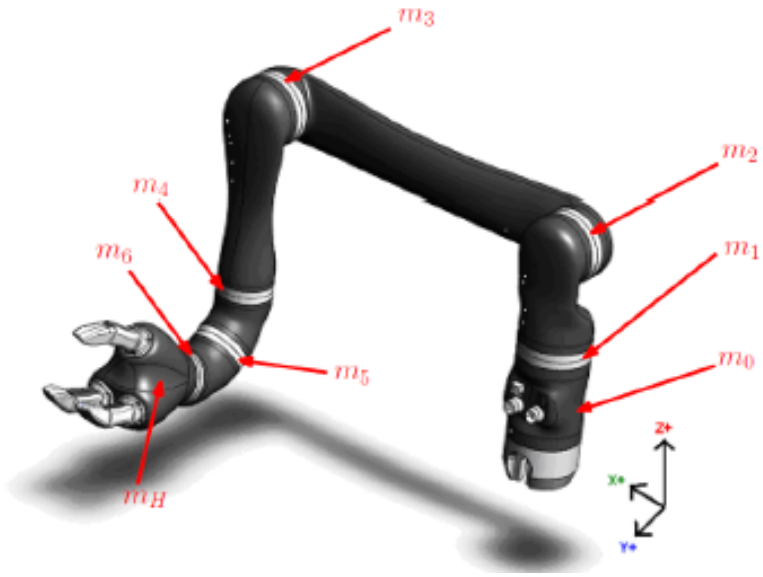


Link lengths

Classic DH parameters				
i	alpha(i-1)	a(i-1)	di	theta1
1	pi/2	0	D1	q1
2	pi	D2	0	q2
3	pi/2	0	-e2	q3
4	2*aa	0	-d4b	q4
5	2*aa	0	-d5b	q5
6	pi	0	-d6b	q6



Denavit-Hartenber parameters

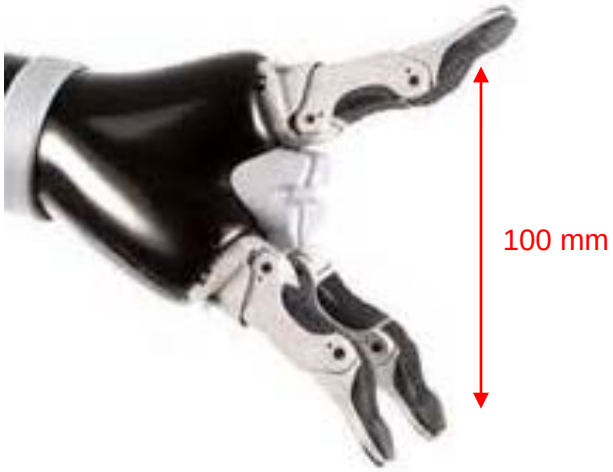


Inertial parameters	
m0	0.64 kg
m1	0.60 kg
m2	0.57 kg
m3	0.60 kg
m4	0.37 kg
m5	0.37 kg
m6	0.37 kg
mH	0.68 kg

Link inertial parameters

The hand is characterized by the following parameters (see also the figures):

- maximum diameter of a cylindrical object for a grasping operation, 100 mm;
- minimum diameter of a cylindrical object for a grasping operation, 45 mm;
- minimum object diameter for a pinching operation, 8 mm;
- maximum finger-to-finger force, 40 N.



1.3 – Hokuyo URG-04LX

The Hokuyo URG-04LX is a lightweight, low-power consumption, wide-range laser scanner that has been designed for a wide range of applications, from service robotics to automatic material handling systems.

Thanks to the low-weight the sensor can be easily mounted on the Jaco 2 wrist and has been used for indoor testing of the scanning task that allows to reconstruct the portion of vineyard in front of the arm in order to compute the dispenser deploying location.

The main sensor specifications are reported in the following table.

Power source	5VDC±5%(USB Bus power)
Light source	Semiconductor laser diode(λ=785nm) Laser safety class 1
Measuring area	20 to 5600mm(white paper with 70mm×70mm), 240°
Accuracy	60 to 1,000mm : ±30mm, 1,000 to 4,095mm : ±3% of measurement
Angular resolution	Step angle : approx. 0.36°(360°/1,024 steps)
Scanning time	100ms/scan
Noise	25dB or less
Interface	USB2.0/1.1[Mini B](Full Speed)
Command System	SCIP Ver.2.0
Ambient illuminance	Halogen/mercury lamp: 10,000Lux or less, Florescent: 6000Lux (Max)
Ambient temperature/humidity	-10 to +50 degrees C, 85% or less (No condensation, no icing)
Vibration resistance	10 to 55Hz, double amplitude 1.5mm each 2 hour in X, Y and Z directions
Impact resistance	196m/s2, Each 10 time in X, Y and Z directions
Weight	Approx. 160g

1.4 – Intel Realsense R200 camera

The R200 is a small (102x10x4 mm) USB3 camera, providing RGB color and stereoscopic IR, to produce depth, images. With the help of a laser projector, the camera does 3D scanning for scene perception having a range of approximately 0.5-3.5 meters inside and up to 10 meters outside.

The main camera specifications are reported in the following table.

Operating Range (Min-Max)	0.5m - 3.5m
Depth Resolution and FPS	480 x 360 @ 60fps
Depth Field of View	H: 59, V: 46, D: 70
Dimensions	101.6mm x 9.6mm x 3.8mm
Power	1.7 W
System Interface Type	USB3

The camera will be used to set up a visual servoing loop that allows to increase the robustness of the dispenser deploying task (see Chapter 6 for more details).

2 – Arm positioning on Husky platform

In many applications, the robot performance can be considerably improved by optimal evaluation of different parameters, such as the pose of the robotic manipulator relative to the task, the maximum velocities and accelerations of the actuators, the manipulator's configuration, etc.

The aim of the optimal placement problem is to find the optimum base pose and the joint configurations of the robot, maximizing the manipulability and dexterity of the end-effector to perform a given task.

This chapter introduces two algorithms to compute the best location for mounting the Jaco 2 arm on the Husky platform, in order to ease as much as possible the dispenser deployment and manipulation tasks.
Both approaches are based on a randomized algorithm that samples the Cartesian or the joint space in order to find the positioning of the arm that approximately maximizes a predefined kinematic index. The first approach leverages on the inverse kinematics relation in order to compute the index, while the second one is based on forward kinematics.

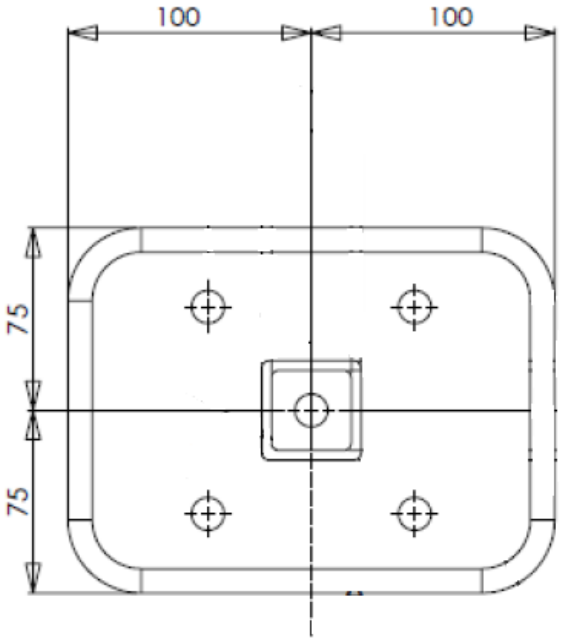
2.1 – Inverse kinematics-based approach

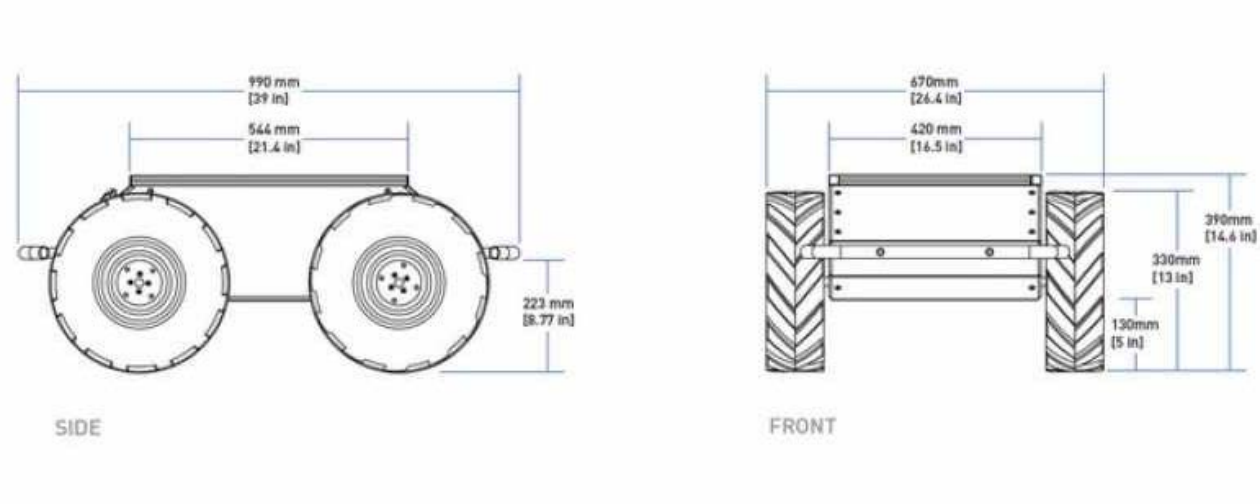
In order to determine the optimal placement of Jaco 2 arm, the manipulability index has been selected as a tool to assess the quality of the location. Then, a MATLAB code has been developed and used for the analysis.

Manipulability index, based on the computation of the robot Jacobian matrix, is the most well-known and commonly used index to measure kinematic performance. The index is a measure of the manipulation ability of a robotic arm in positioning and orienting the end-effector, and it allows also to quantify the proximity of the robot to a singularity.
The manipulability is defined as the square root of the determinant of the product of the manipulator Jacobian by its transpose, i.e.,

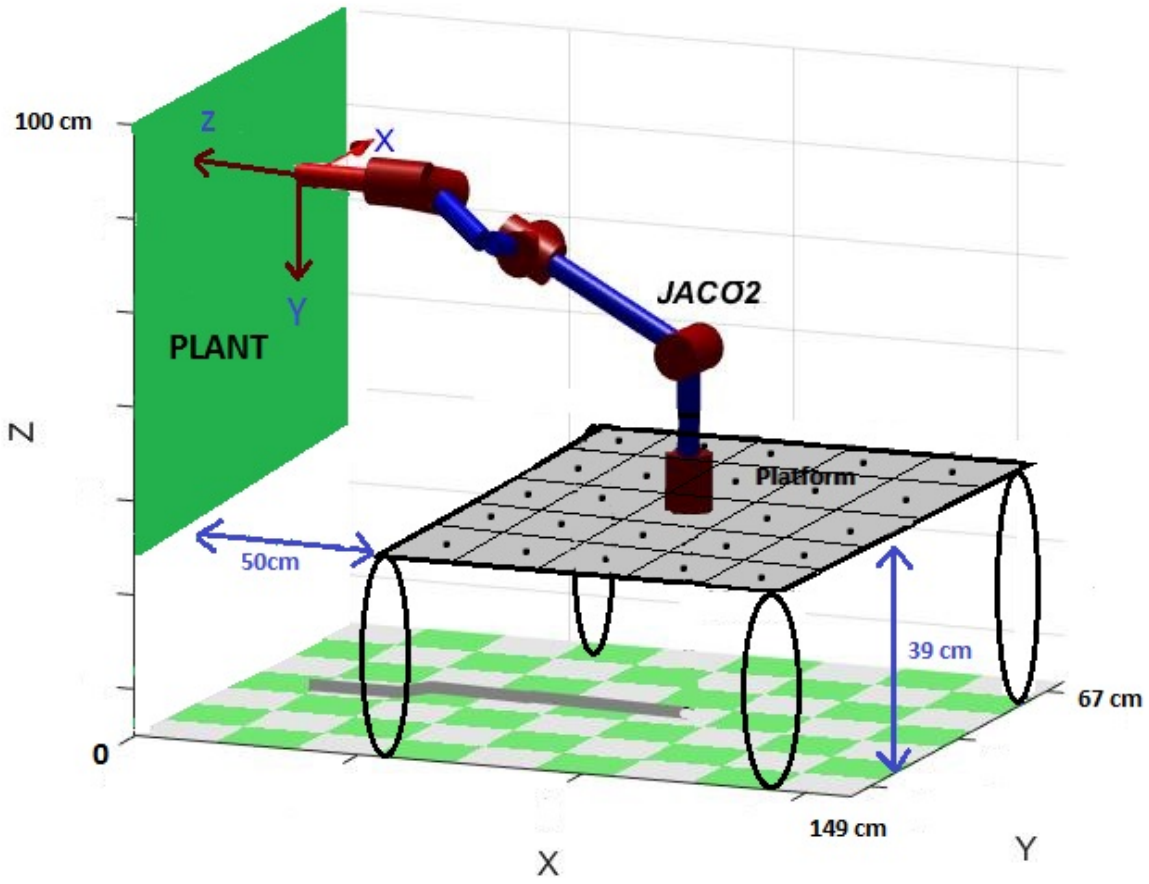
$$\mu = \sqrt{\det(JJ^T)}$$

In order to study the best location of the arm on the rover, all the required dimensions including Husky platform, arm base support, and vineyard standard working area, have been collected. The dimensions of the arm base support in millimeters, and of the Husky platform are shown in the figures below.





To simplify the analysis, the vineyard working area has been assumed to be a box (left green area in the following figure). Its dimensions are 61cm high (distance from the Husky platform to the highest point of working area), 67cm width (equal to the width of Husky platform) and 30cm depth.
Moreover, it is supposed that Husky stops by the vineyard, at a distance of 0.5 meter, to deploy the dispensers.



The first approach to optimize the location of the arm on the platform is structured as follows.

First, Husky platform base has been gridded into tiles with dimension 20cmx15cm, corresponding to the size of the arm plate support (see the figure above). The purpose of this approach is to investigate which grid tile is the best location for mounting the Jaco 2 arm.
To reduce the computational cost and simplify the analysis, the orientation of the end-effector has been assumed to be constant and equal to zero, while its position have been chosen randomly inside the vineyard working area plane.
The algorithm starts selecting as Jaco location the first tile of the grid.

Second, 10000 points have been chosen at random inside the working plane, as possible positions of the end-effector. For each random point, an inverse kinematics has been performed to obtain the joint coordinates.

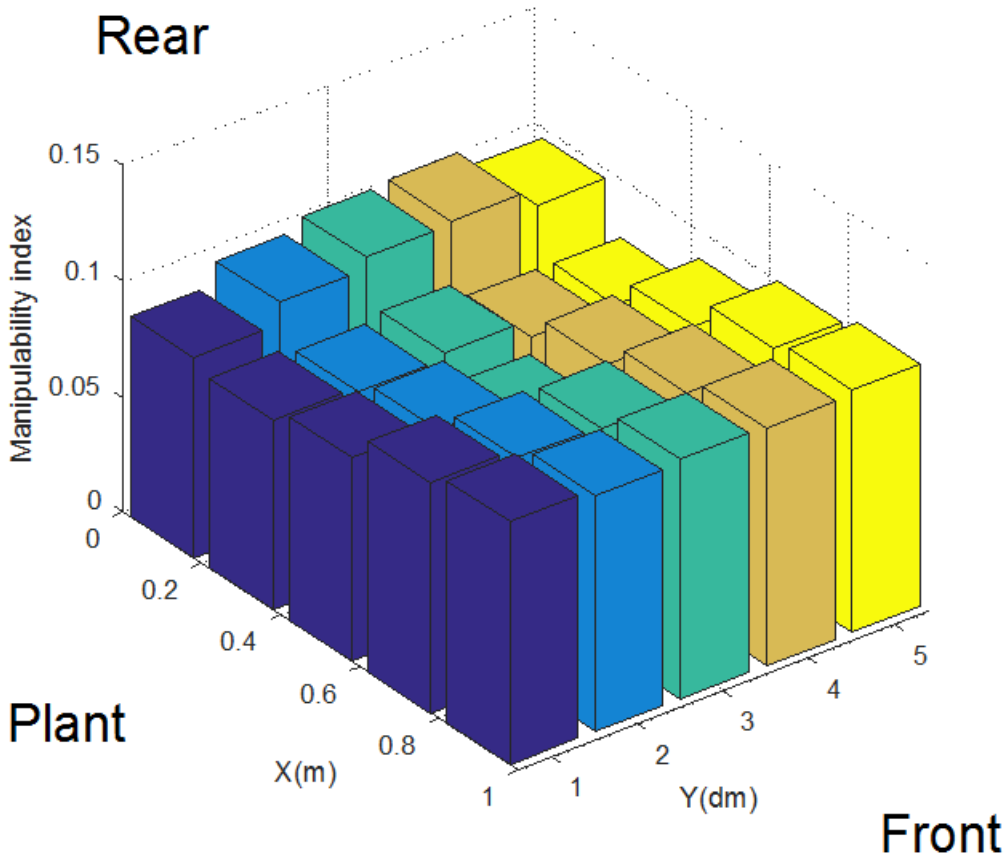
Third, the manipulability index of the obtained joint positions is computed. This procedure is repeated for each of the 10000 chosen random points, and an average of all manipulability indexes is determined and associated to the robot location.

Fourth, a different location is selected and the procedure is repeated until all the locations have been tested.

Finally, the grid tile whose average manipulability index is maximum, is the best location for mounting the Jaco 2 arm.

2.1.1 – Optimization results

The results of the application of the previously described procedure are reported in the following figure, in which the height of each 3D bar represents the average manipulability index of the associated grid tile.
Due to the use of inverse kinematics in the algorithm, which is not accurate, the height of 3D bars are very close to each other and making it difficult to find the optimal placement of Jaco 2 arm.



2.2 – Forward kinematics-based approach

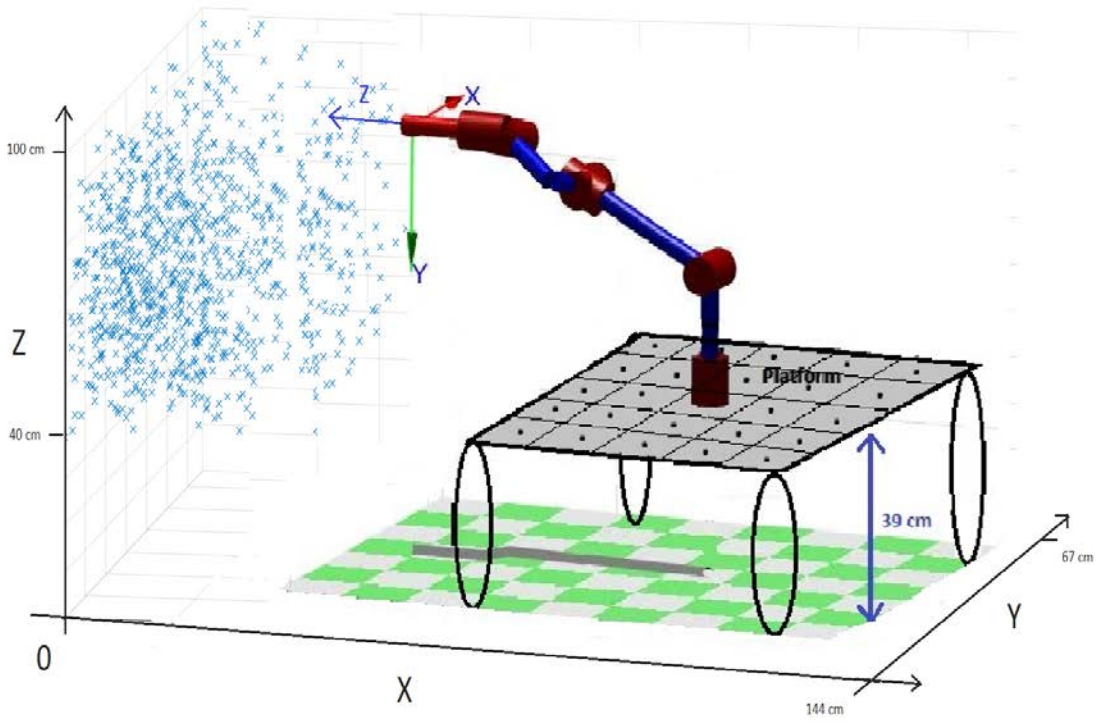
The second approach is similar to the first one, but forward kinematics has been adopted instead of inverse kinematics in order to increase the computation accuracy and solve the previous issues.

The second approach is structured as follows.

First, the algorithm starts selecting as Jaco location the first tile of the grid.

Second, 10000 joint position vectors are chosen randomly and forward kinematics is performed to compute the position and orientation of the end-effector.

Third, those end-effector positions which are inside the desired region, i.e., the vineyard working plane, are accepted, the others are discarded (see figure below).



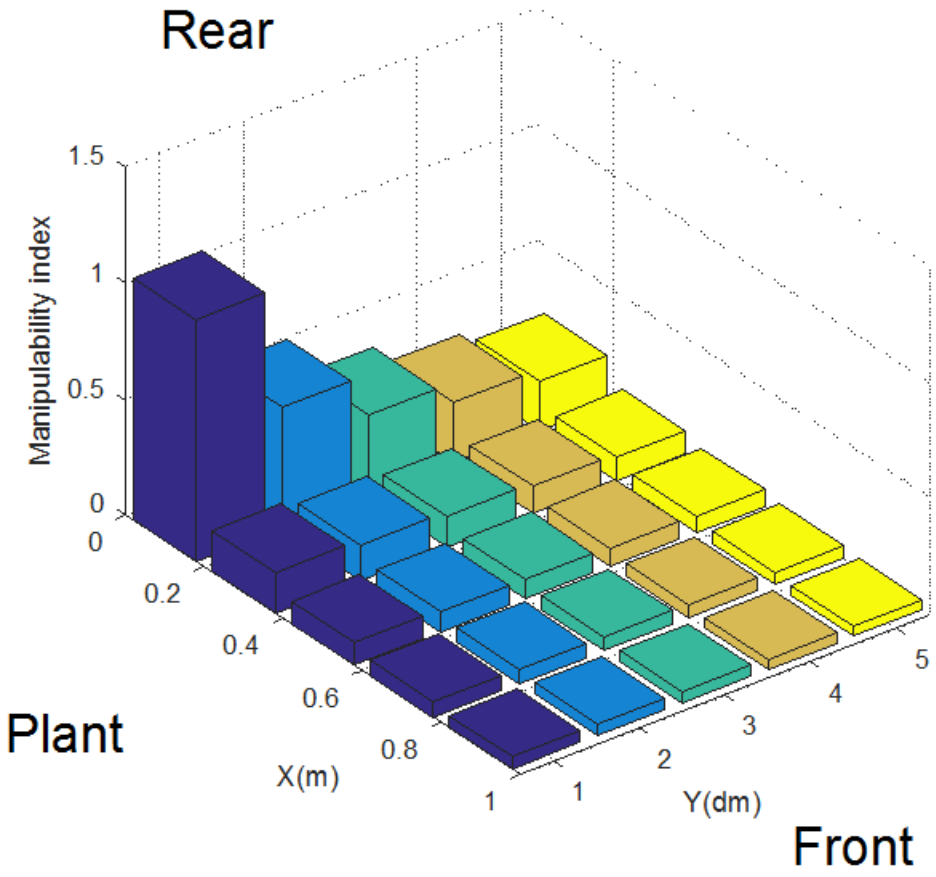
Fourth, the manipulability index for the chosen joint coordinates, corresponding to the accepted end-effector positions, is computed, and an averaged value is determined and associated to the robot location.

Fifth, a different location is selected and the procedure is repeated until all the locations have been tested.

Finally, the grid tile whose average manipulability index is maximum is the best location for mounting the Jaco 2 arm.

2.2.1 – Optimization results

The results of the application of the second procedure are reported in the following figure, in which the height of each 3D bar represents the average manipulability index of the associated grid tile.
As it is clear, first grid tile has the maximum average manipulability index. Therefore, the optimal placement of Jaco 2 arm is located on the rear right corner of the Husky platform.


Rear
Manipulability index
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
X(m)
1
2
3
4
5
Y(dm)
Plant
Front

3 – Dispenser manipulation devices

This section describes the devices that have been developed in order to manipulate the dispensers with the Jaco2 hand.
An example of these dispensers, the ISONET L E manufactured by Biogard (http://www.biogard.it), is shown in the figure below. A dispenser is composed by two flexible plastic capillaries, welded at the end, full of pheromones, with a length of approximately 15 cm. Pushing the two ends the dispenser can be deformed, enlarging the two capillaries, so that it takes the shape of a ring and it can be hang to a branch of the vineyard.

Different versions of dispenser feeder, i.e., a device that allows to store dispensers keeping them open so that they can be taken one by one using Jaco 2 hand, and nails, i.e., devices that are mounted on the fingers in order to make dispenser grasping operation easier and more robust, have been designed and realized in order to ease the manipulation, make it more robust and increasing the number of dispensers that can be manipulated. Nonetheless, this process has been performed as part of a research project and, thus, with any aim of devising a solution that is ready to be commercialized. One of the most important reason that makes this solution still far from a market-ready device is that the robot initially selected for the project, and in particular its hand, are general purpose devices not optimized for the specific application. Therefore, the length of the fingers is too large and, probably, the third finger interferes with instead of aiding the manipulation task.
For all these reasons, the design of the feeder and nails has been targeted to the creation of tools that allow to increase the manipulation robustness being able to support a reasonable demonstration of the GRAPE robot functionalities.
Feeder and nail prototypes have been then realized using 3D printing and choosing, as printing material, ABS (Acrylonitrile-Butadiene Styrene). More details can be found in Annex 1.

3.1 – Feeder design

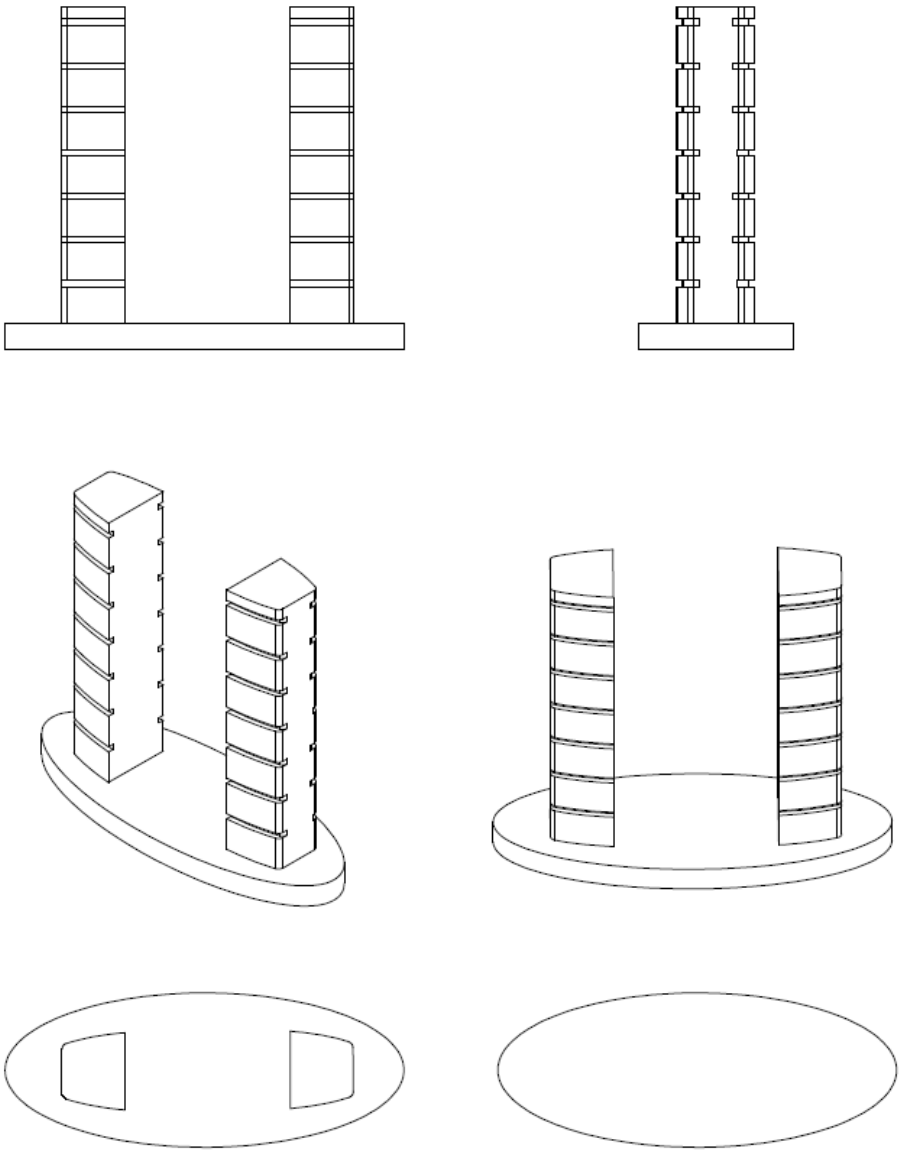
As already mentioned, different feeders have been designed and tested, following a trial and error procedure, until a satisfactory one has been devised.

Feeders' design has been driven by two main requirements. First, feeders should be able to support the dispensers, holding them in a predefined position. Second, feeders must keep the dispensers open in order to prepare them for a successful grasping.

In the following, the different feeder designs are described in detail. Further details, including part sizes, are reported in Annex 1.

3.1.1 – First design

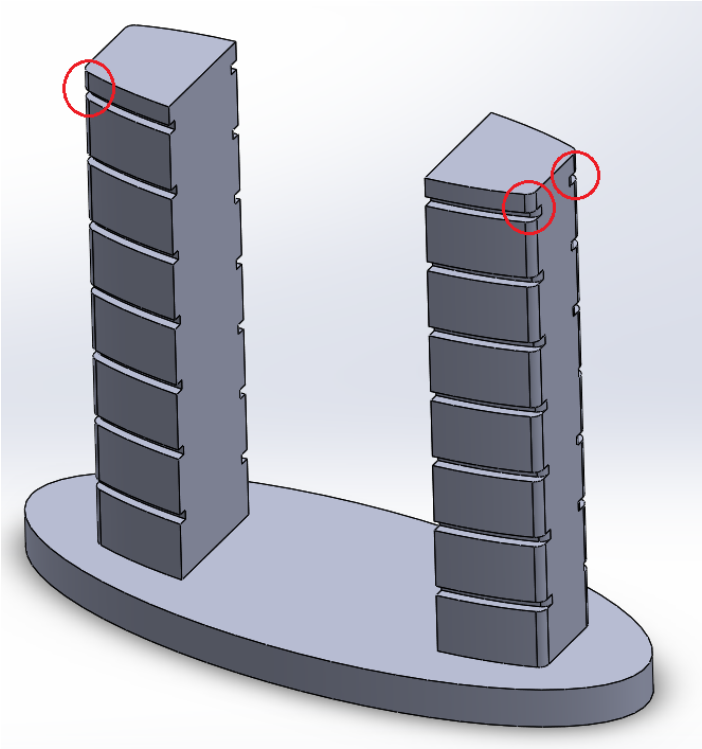
To support the dispensers, two columns with indentations on the external surface have been designed. With this design, dispensers can be located in the carved part and fixed there.
The figure below shows a schematic view of the first design.



To keep dispensers open in a position that can ease the grasping task, column sections have been designed as part of an ellipsoid, as it can be seen from the top view in the figure, as this is the same shape the dispensers have in the open position.

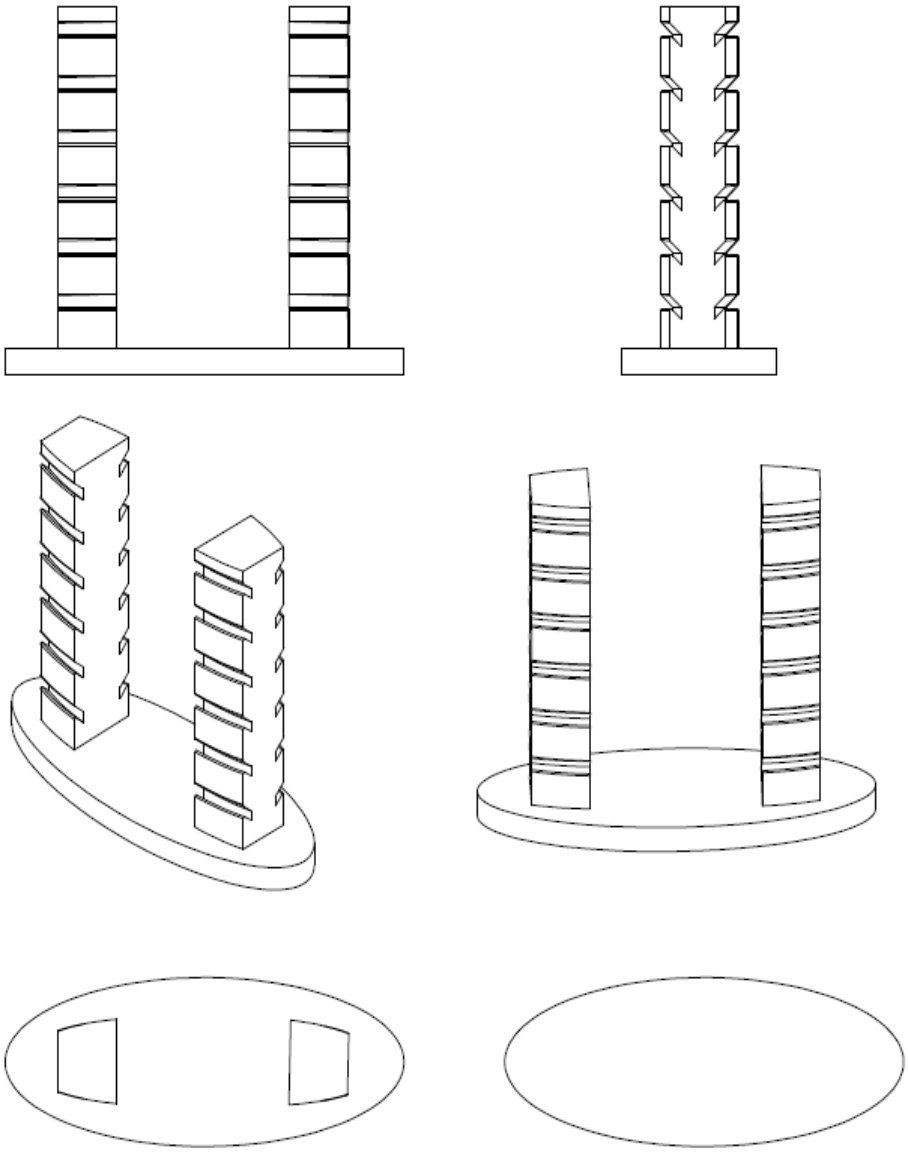
Using this design, a prototype has been produced by 3D printing, and grasping tests have been performed. The results of these tests show that, although the dispensers can be located and remained fixed in the desired position on the feeders, and they are kept open for a successful grasping, they may get stuck in the edges of the feeders during the detaching operation, causing the grasping task to fail.

The figure below shows a 3D model of the feeders and the red circles emphasize the more critical parts of this design.

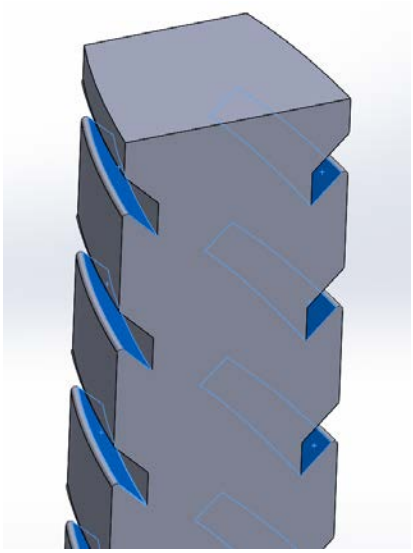


3.1.2 – Second design

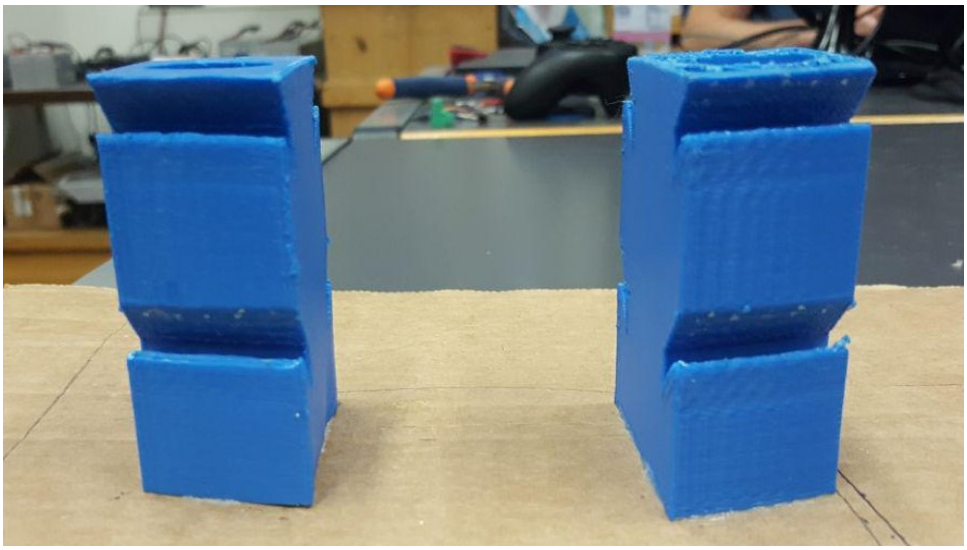
In order to get rid of the drawbacks of the previous design, the carved parts were modified adding an inclination to the floor and ceiling of each hollow, in order to ease the dispenser detaching operation. In this second design, it has been assumed that the dispensers can be detached from the feeders by sliding on the inclined surfaces, without getting stuck into the edges as in the first design.
The figure below shows a schematic view of the second design.



The two figures below shows a zoom of the 3D model and the 3D printed prototype of the second design, showing the inclination of the hollow parts. In particular, in the 3D model the blue surfaces represent the critical planes for the detaching operation, due to the high friction.



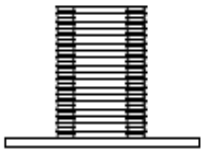
.

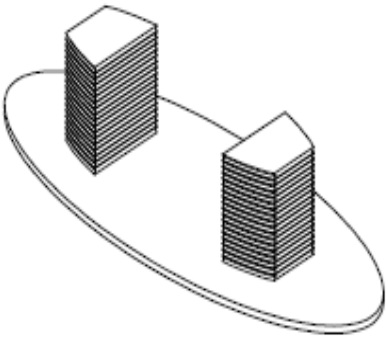


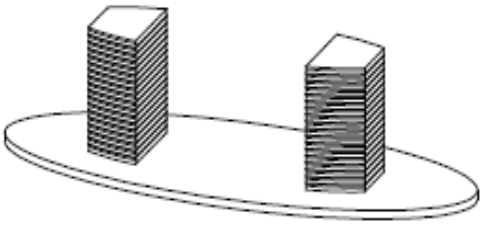
As for the first design, a prototype has been produced using 3D printing, and grasping tests have been performed. The results of these tests show that, in spite of the modifications of this second design, the dispenser detaching operation was not fully successful, due to the high friction of the inclined surface that is probably caused by the rough material used to print the feeders.

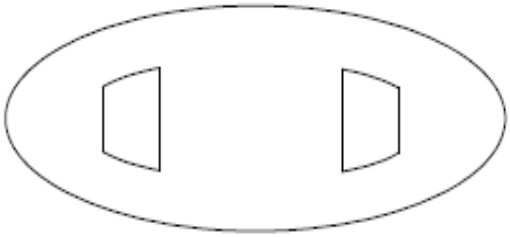
3.1.3 – Third design

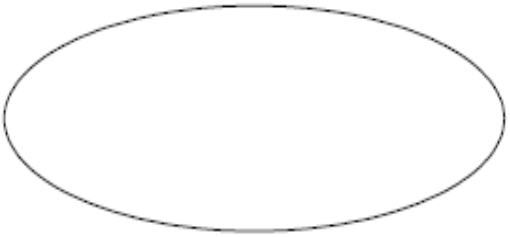
After the failures of the previous designs, the carved parts were modified again substituting the hollow parts with a continuous vertical surface characterized by a zigzag pattern, i.e., a saw-tooth profile. The zigzag pattern aims at introducing enough friction to hold the dispensers in the desired positions while its small indents should not allow them to get stuck in the feeder during the detaching operation. This new feeder design not only solves the previous drawbacks, but can also house a larger number of dispensers.
The figures below show a schematic view, a 3D model and the 3D printed prototype of the third design.

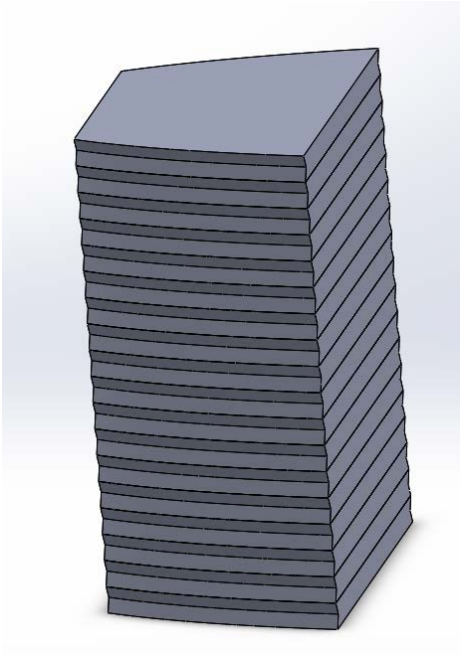


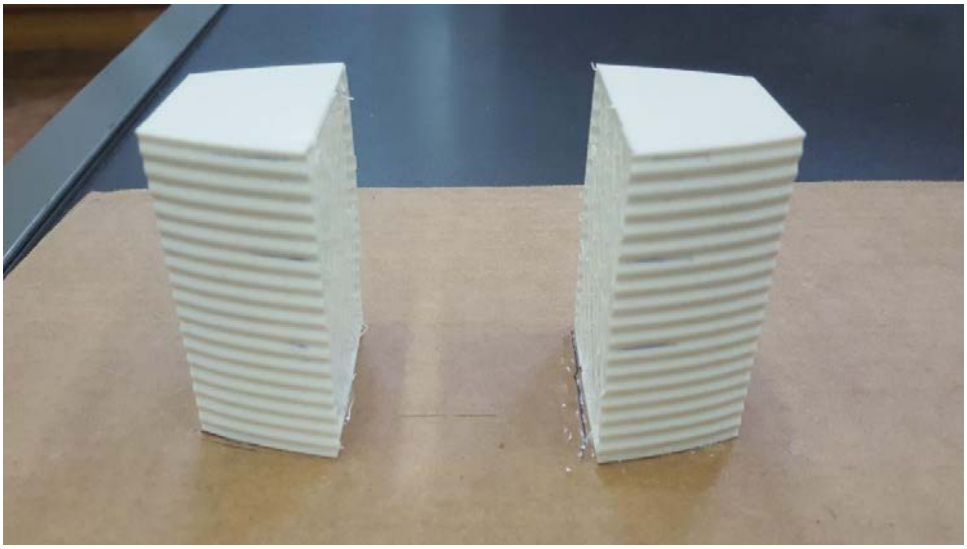








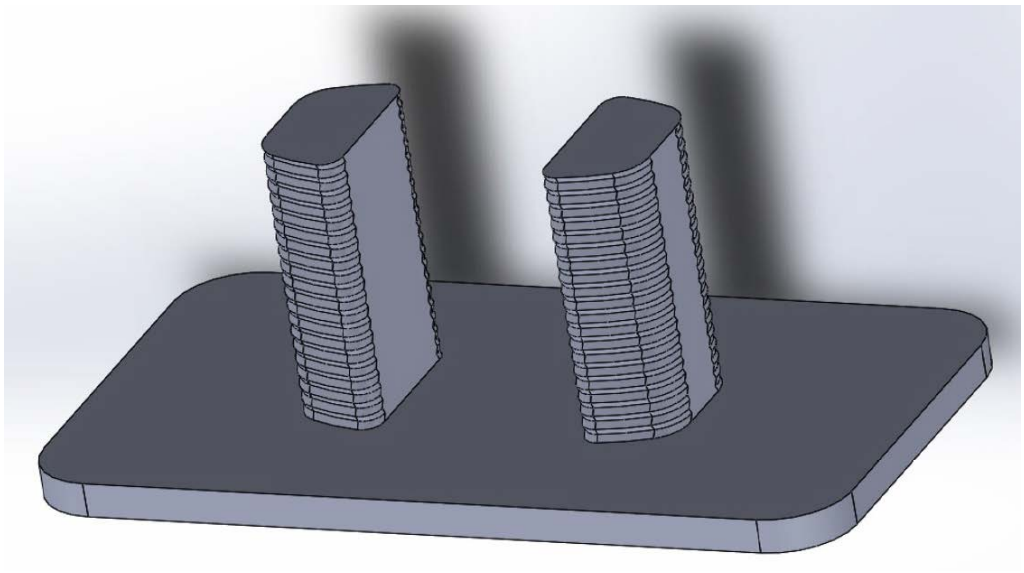


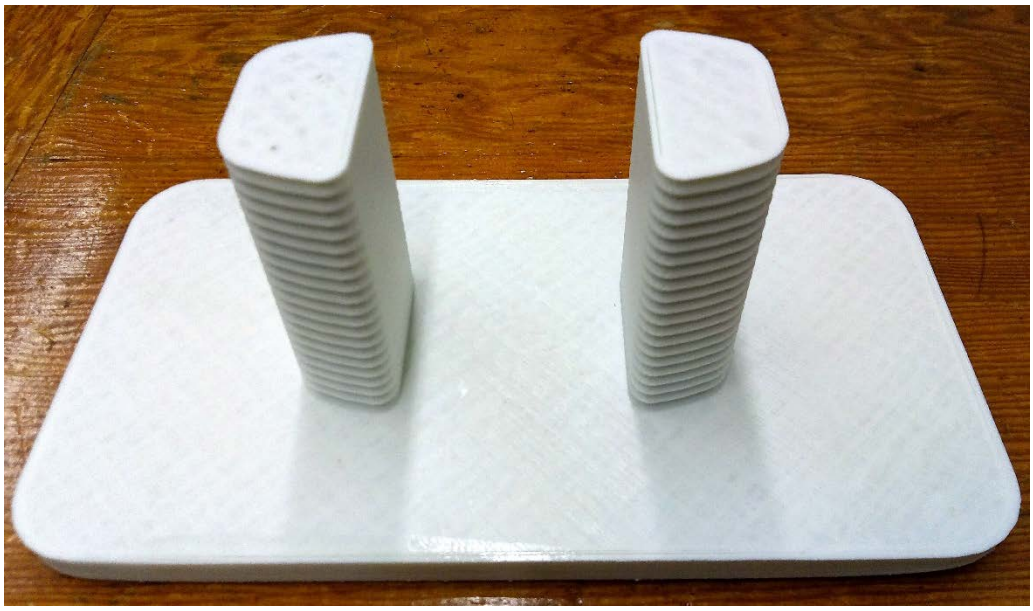


As for the other designs, a prototype has been produced using 3D printing, and grasping tests have been performed. The results of these tests show that this feeder design allows to successfully perform the grasping operation in a robust way.

3.1.4 – Fourth design

Though the previous design has passed the first tests, the experience gained after a huge number of detaching operations suggests to simplify it, softening the external part of the ellipsoid shape of the feeders. This allows to decrease the probability that the dispenser get stuck during detaching, without compromising the stability of the contact with the feeder.
The figures below show a schematic view, a 3D model and the 3D printed prototype of the third design.





3.2 – Nail design

As for the feeders, different nails have been designed and tested, following a trial and error procedure, until a satisfactory one has been devised.

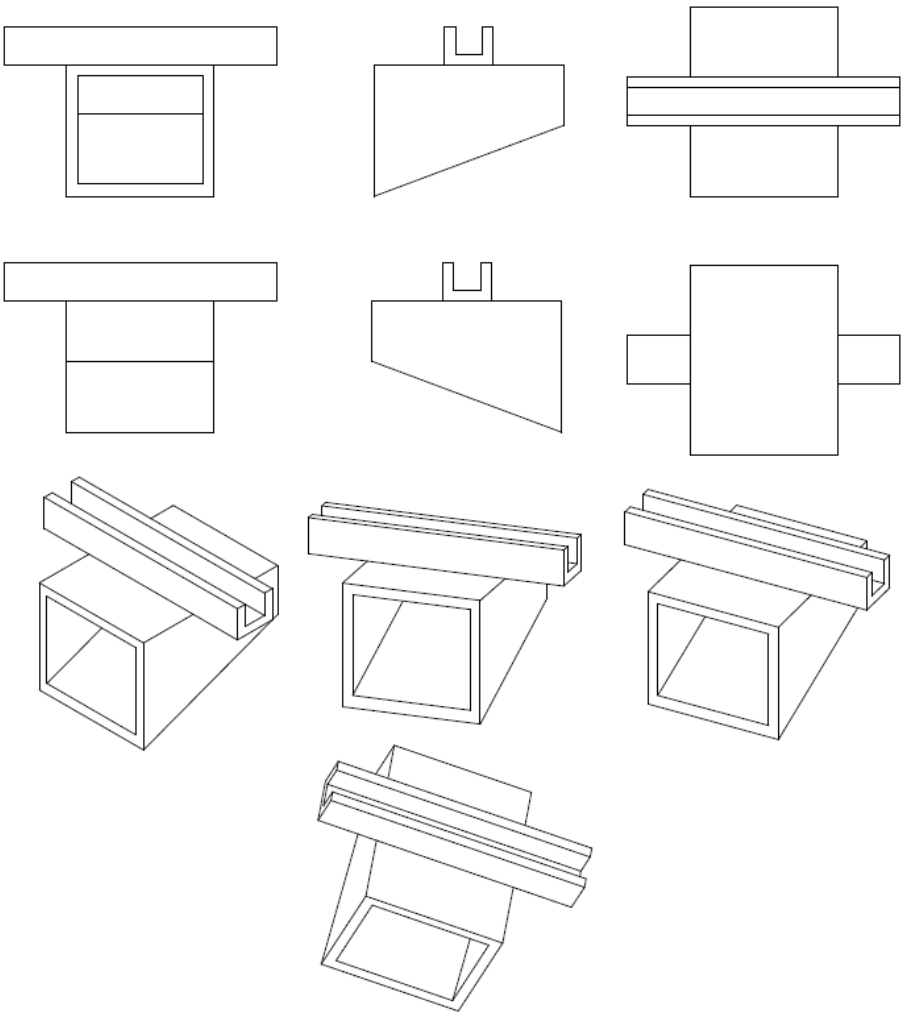
Nails' design has been driven by two main requirements. First, nails should be able to robustly grasp the dispensers from the feeders. Second, nails must robustly hold the dispenser during the manipulation and deployment tasks.

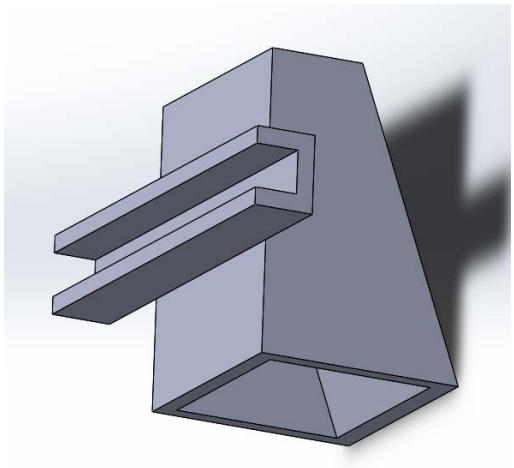
In the following, different nail designs are described in detail. Further details, including part sizes, are reported in Annex 1.

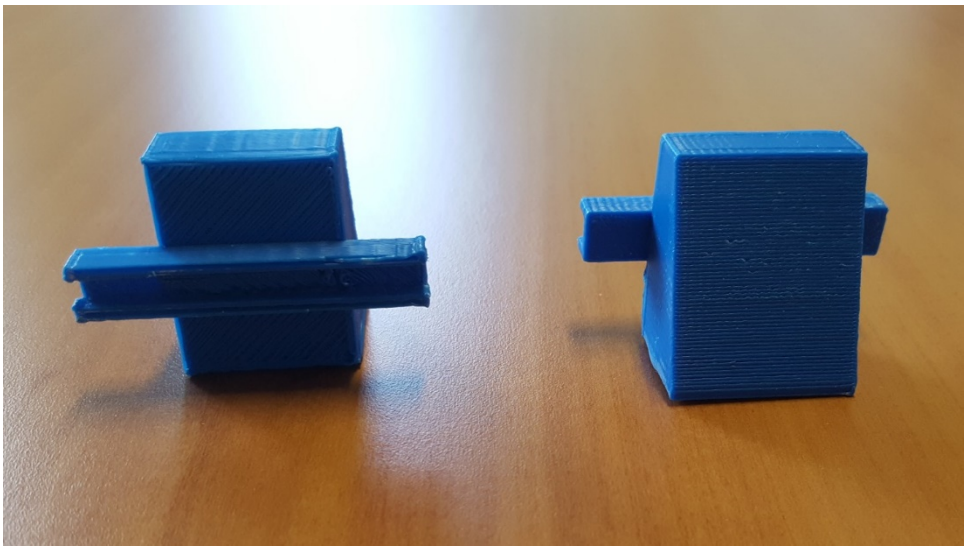
3.2.1 – First design

This nail design is composed of two parts.

The first is a trapezoidal part that can be put on the robot fingers. For a convenient installation, an extruded cut, which is based on the robot finger's shape, has been done inside the trapezoidal part.
The second part is a rectangular piece with an extruded cut inside, that has been designed to hold the dispenser.
The figures below show a schematic view, a 3D model and a picture of the printed prototypes of the first nail design.



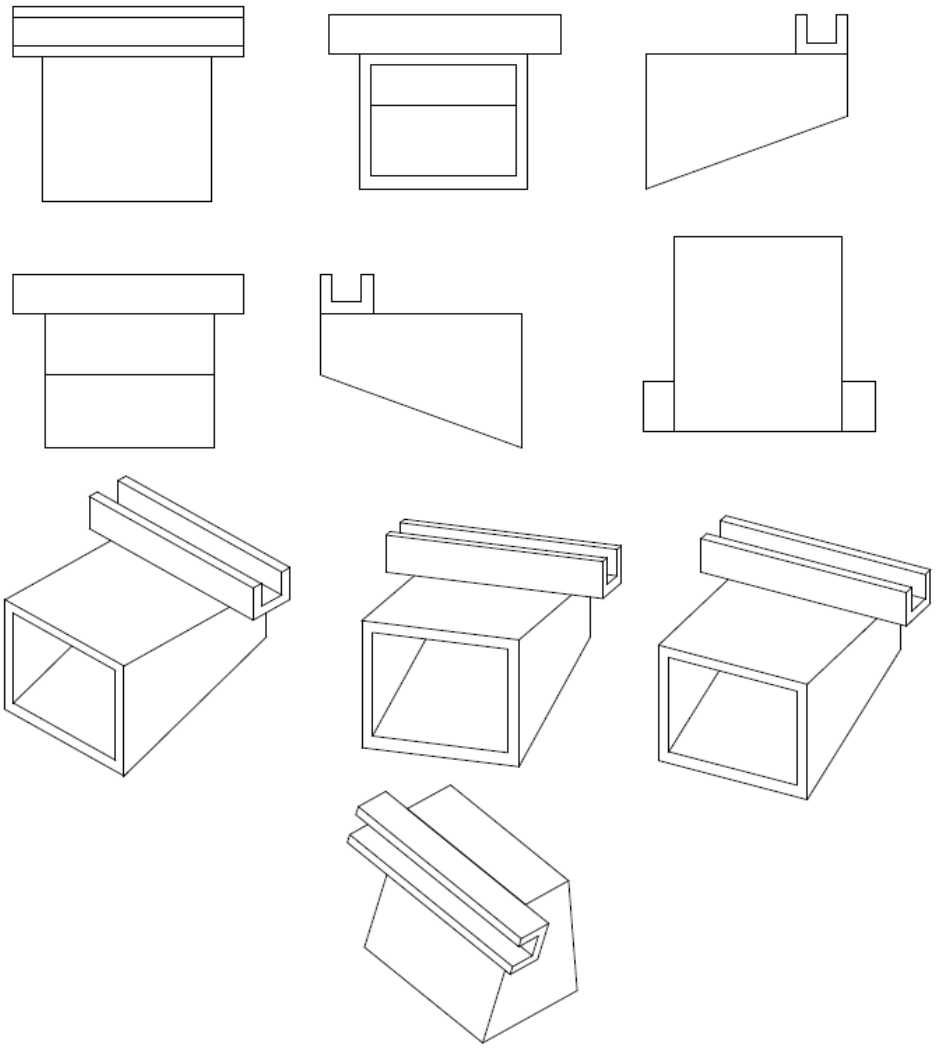


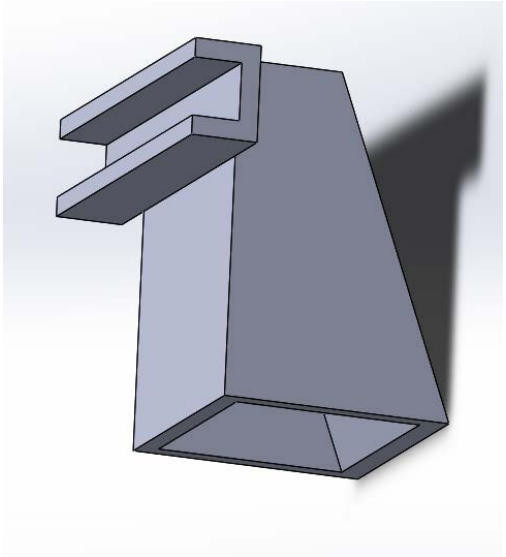


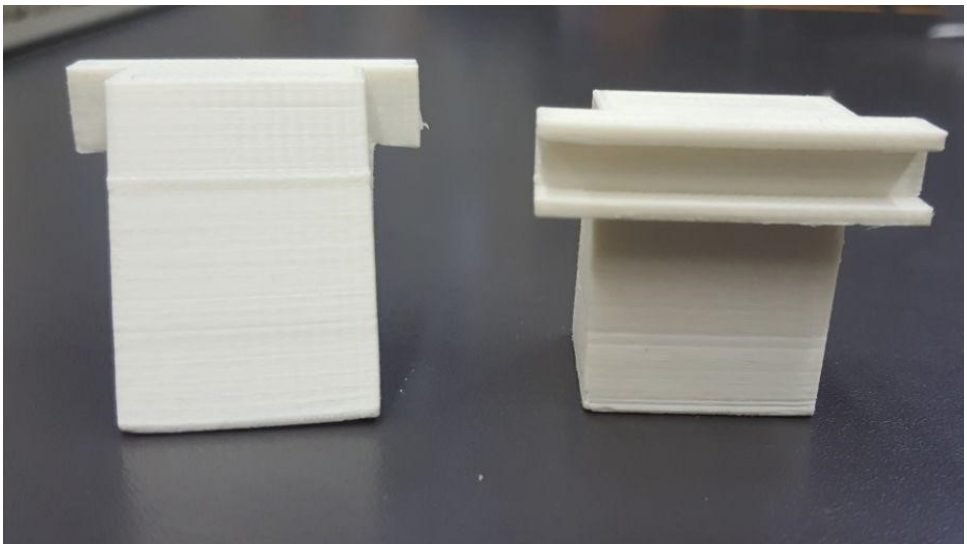
As it was done for the feeders, using this design a prototype has been produced by 3D printing, and grasping tests have been performed. The results of these tests show that, with this design dispensers can be taken and held only in ideal conditions, i.e., when the positioning is perfect. In other words, the grasping operation fails in case of positioning errors of the robot end-effector with respect to the dispenser. Considering the robot positioning accuracy this design does not allow to execute a robust grasping of dispensers.
Furthermore, this design allow to grasp only up to six dispensers from the feeder.

3.2.2 – Second design

This second nail design is very similar to the previous one, but in order to increase the maximum number of dispenser that can be grasped, the rectangular guide is shifted to the fingertip.
The figures below show a schematic view, a 3D model and a picture of the printed prototypes of the second nail design.



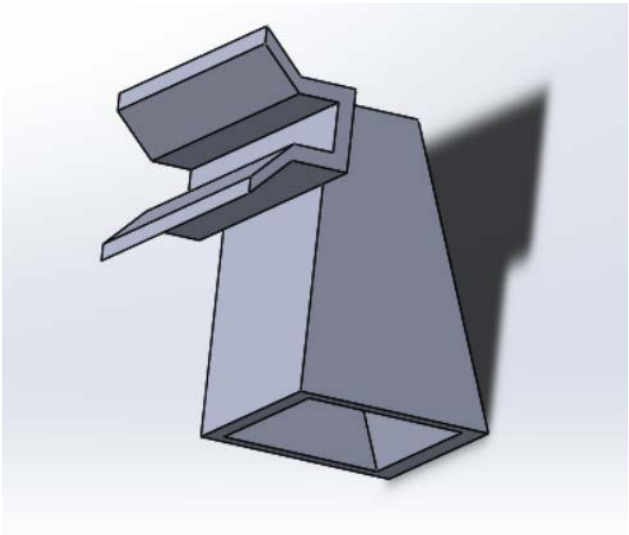


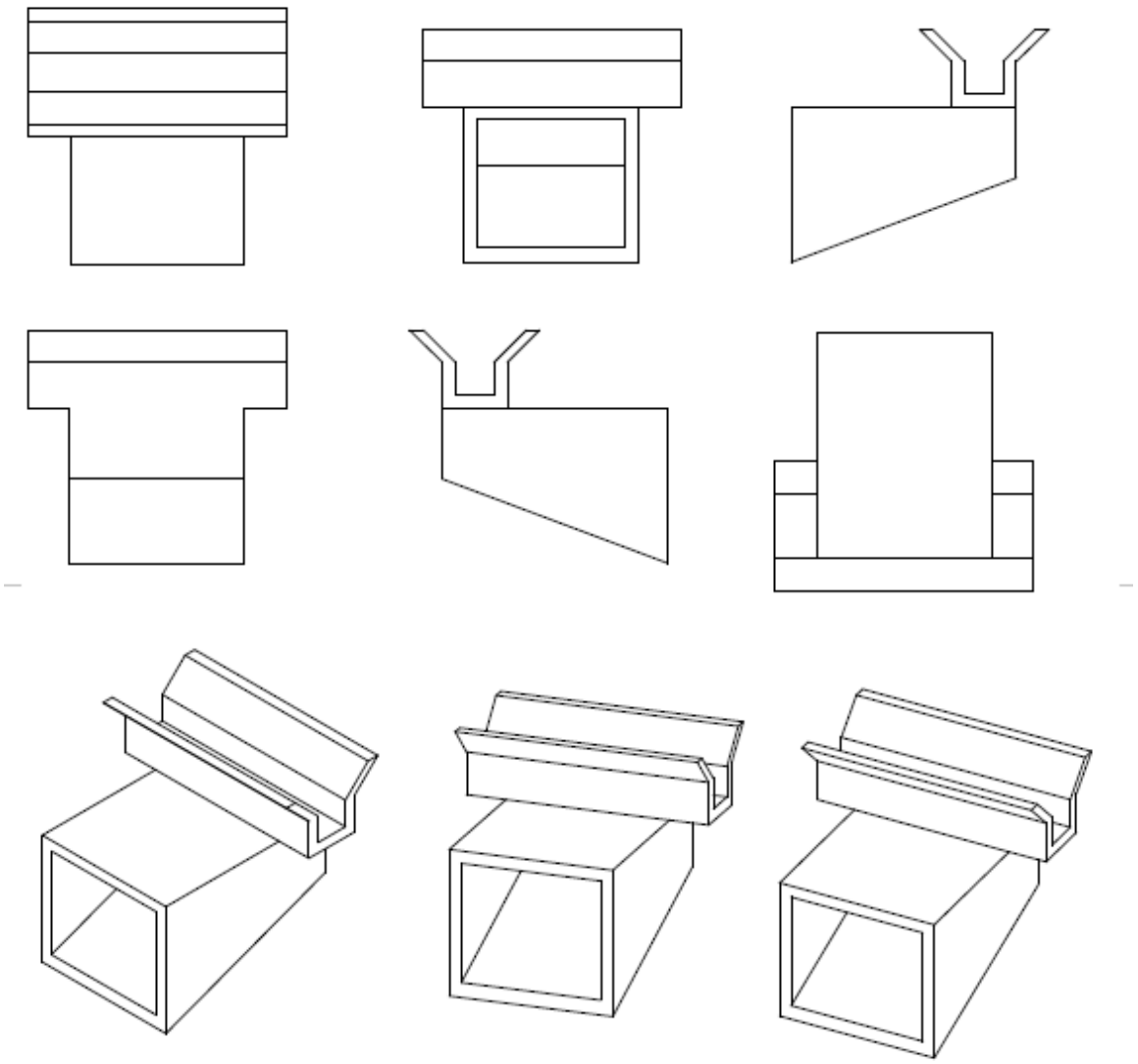


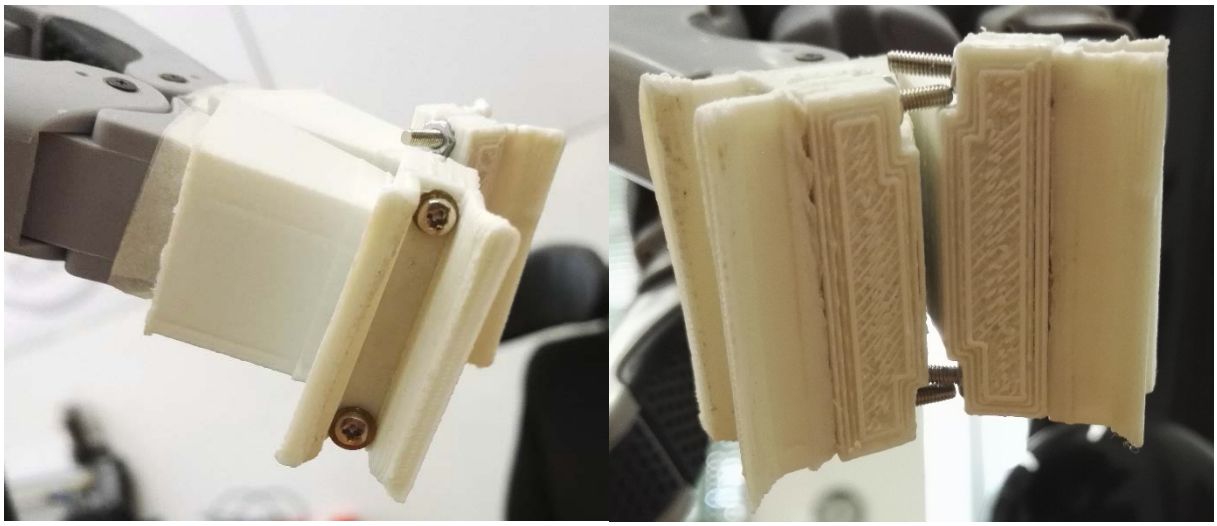
Using this design a prototype has been produced by 3D printing, and grasping tests have been performed. Considering that this design aims only at increasing the number of dispensers that can be grasped from the feeder, but not at solving the issues revealed by the first design, the tests were satisfactory as the maximum numbers of graspable dispenser has been increased.

3.2.3 – Third design

In order to solve the issue revealed by the first design, i.e., to compensate for possible positioning errors between the robot end-effector and the dispenser, the second nail design were modified by adding two inclined edges to the rectangular guide. In the case of a small positioning error, this new design eases the dispenser entering inside the rectangular guide.
The figures below show a schematic view, a 3D model and a picture of the printed prototypes of the third nail design.







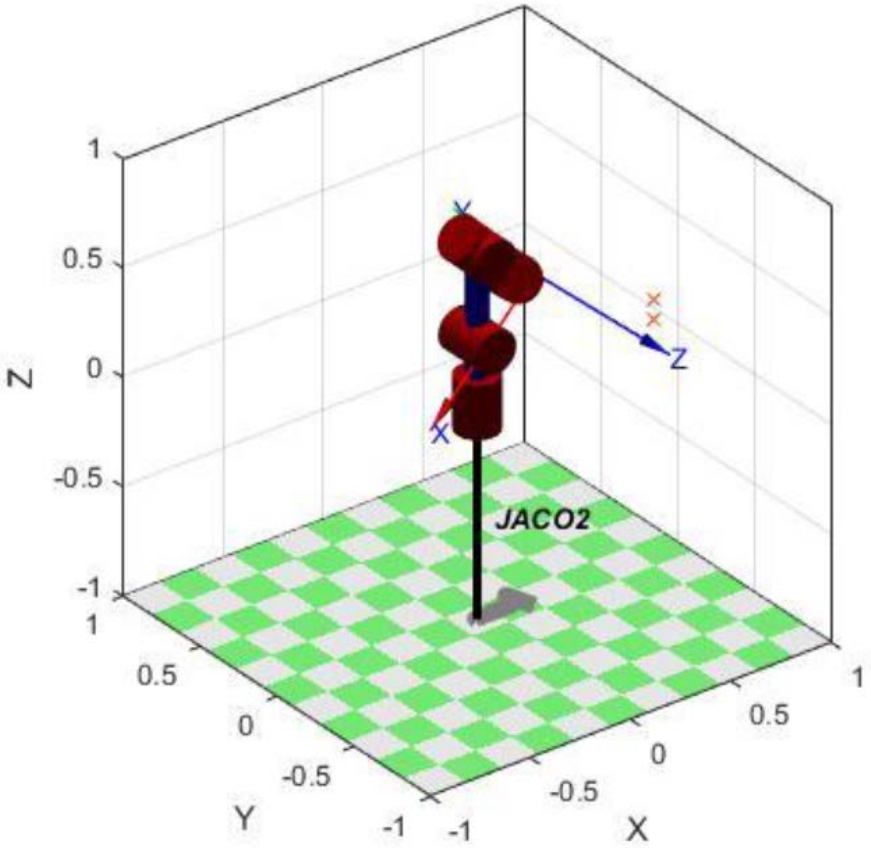
Using this design a prototype has been produced by 3D printing, and grasping tests have been performed. The results of these tests show that, this nail design is able to successfully and robustly grasp the dispenser, manipulate it and deploy it even in the presence of small errors in the positioning of the robot end-effector.

4 – Jaco 2 simulation platform

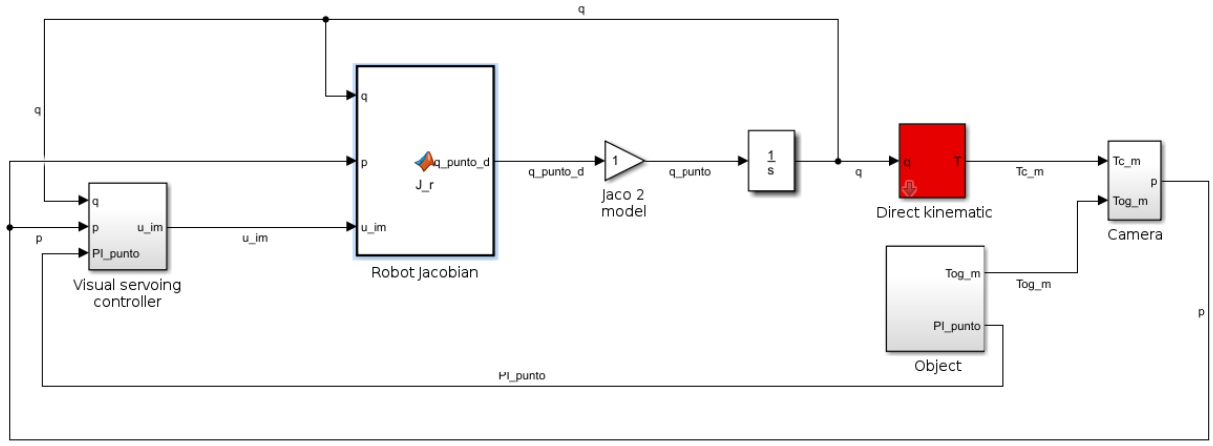
In order to study and validate the control algorithm, before deploying and testing it on the experimental platform, a Matlab/Simulink simulator and a Gazebo simulator has been set up.

4.1 – Matlab/Simulink simulator

Using the kinematic and dynamic parameters reported in Chapter 1, and the Simulink Robotics Toolbox (http://petercorke.com/wordpress/toolboxes/robotics-toolbox), a kinematic and a dynamic model of Jaco 2, including a standard RGB-D camera has been set up.
The picture below shows a 3D representation of the robot.



Considering that a visual servoing control loop is usually closed around the robot position/velocity loops, a realistic simulation aiming at visual servoing control validation can be performed on a simple kinematic model. The picture below shows an example of a Simulink diagram including a visual servoing controller, a kinematic model of the robot, a camera model and the model of a simple object.



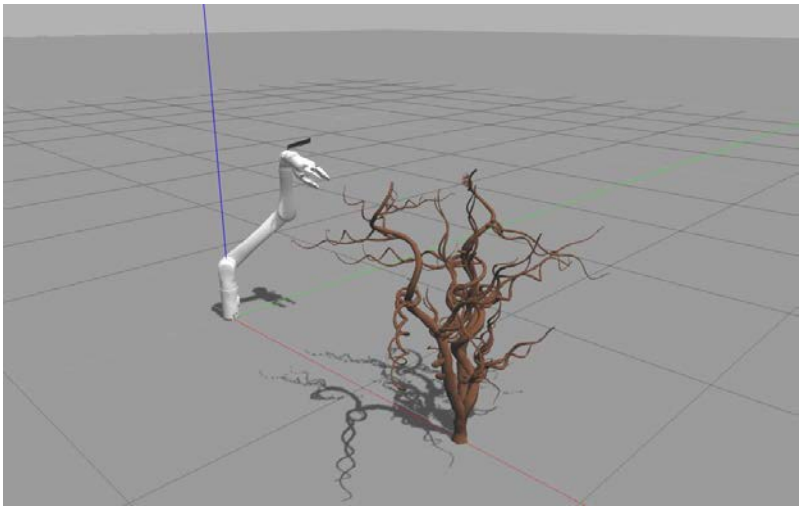
The kinematic model is composed of: the robot Jacobian, computed by way of a suitable function provided by the Robotics Toolbox, to convert end-effector Cartesian velocities into joint velocities; a vector of integrators, to compute joint positions from joint velocities; forward kinematics, computed by the Robotics Toolbox, to determine end-effector pose from joint positions; a model of the camera, provided by the Robotics Toolbox, representing perspective projection.

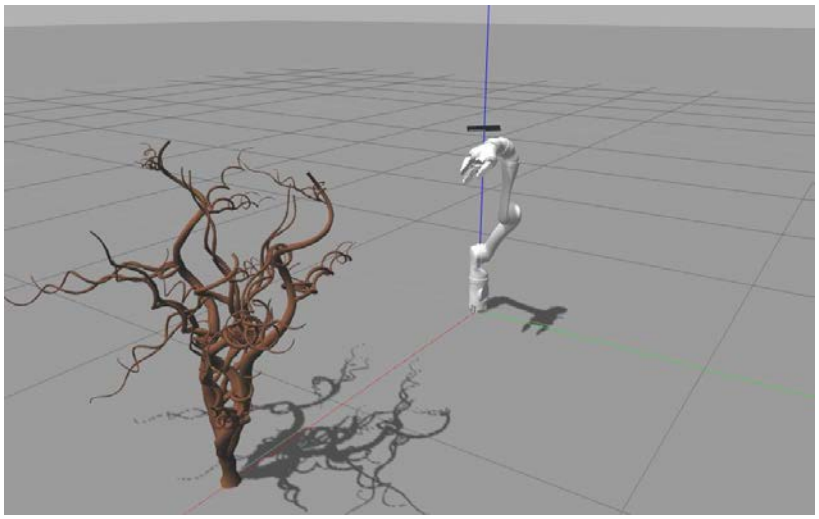
4.2 – Gazebo simulator

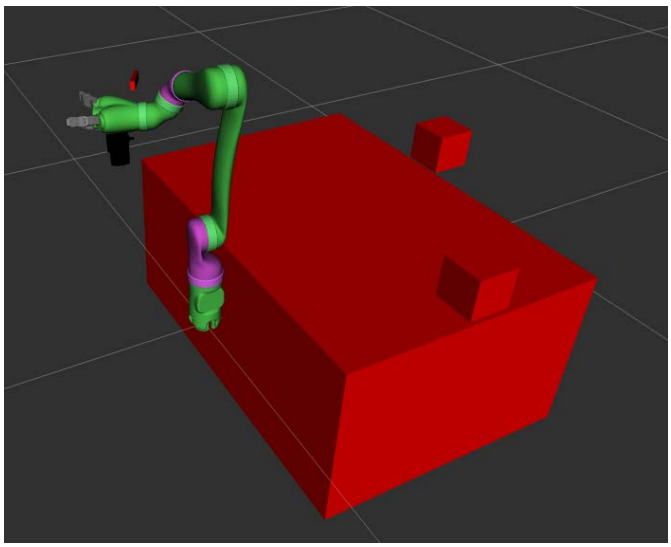
To help the development of the control software and perform a preliminary testing in simulation, a Gazebo simulator has been exploited.

The simulator and the software developed by POLIMI for dispenser manipulation and deploying run on a machine mounting Ubuntu 16.04 LTS, and they have been developed using ROS Kinetic Kame (http://wiki.ros.org/kinetic) and Gazebo 7.5 (http://gazebosim.org).

The simulation environment exploits the official Jaco 2 packages created by Kinova, including some standard launch and world files for the simulation of Jaco 2 in Gazebo and the visualization, control and monitoring of the robot throw Rviz GUI (http://wiki.ros.org/rviz).
Moreover, sensors, like the Hokuyo laser scanner and the Realsense camera, have been added to the simulation environment and to the Rviz GUI together with some static obstacles representing the Husky platform with the navigation sensors, in order to test the execution of the arm tasks.







Gazebo environment has been also extended adding a model of a vineyard.

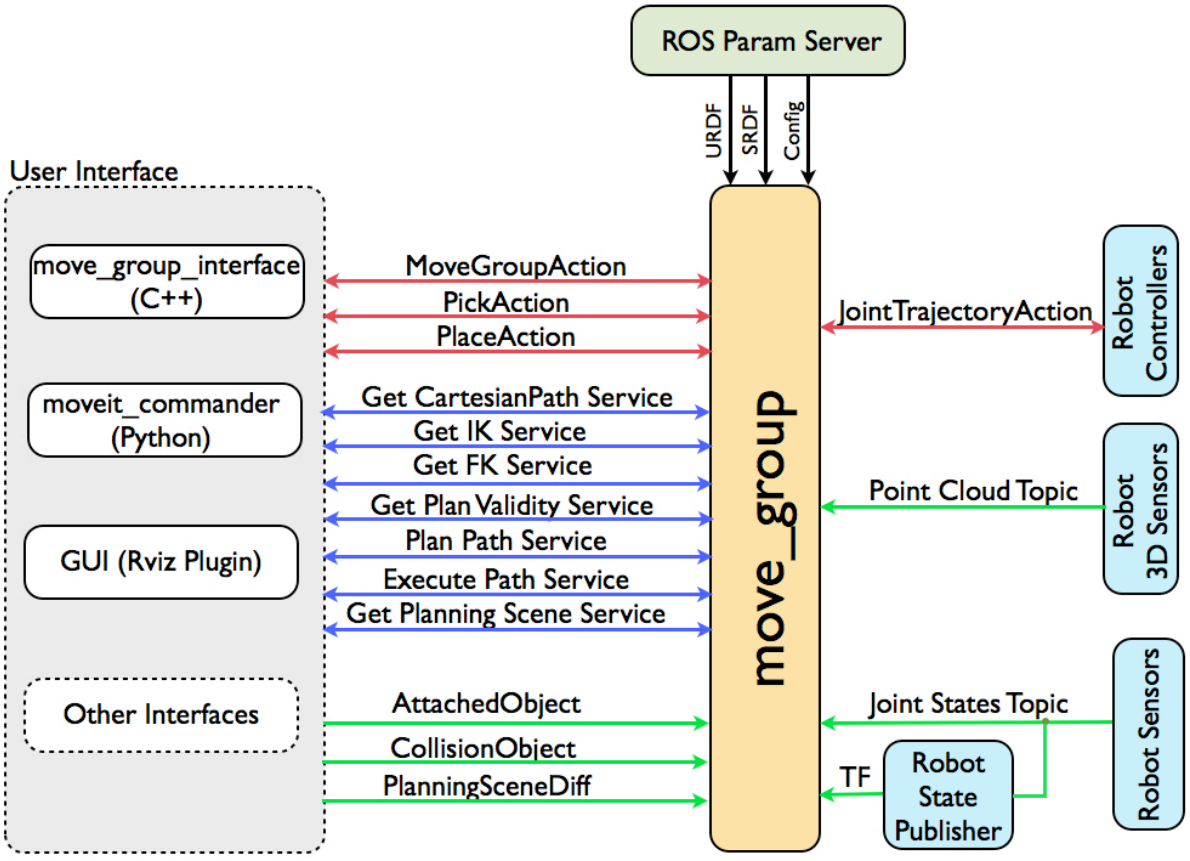
5 – Jaco 2 control architecture

The arm control architecture is based on the ROS package MoveIt! (http://moveit.ros.org/) and on a part of code developed by POLIMI to support visual servoing control. This chapter describes MoveIt! architecture and the functionalities provided to perform dispenser manipulation and deployment tasks. The next Chapter, instead, introduces the details on the visual servoing algorithm.

5.1 – MoveIt! architecture overview

MoveIt! is the state of the art and most widely used ROS package for mobile manipulation, including the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. Thanks to the already implemented and ready-to-use planning and control algorithms, and to the drivers that allow to communicate with 65 commercial robots, it ease the development of manipulation and mobile manipulation tasks.

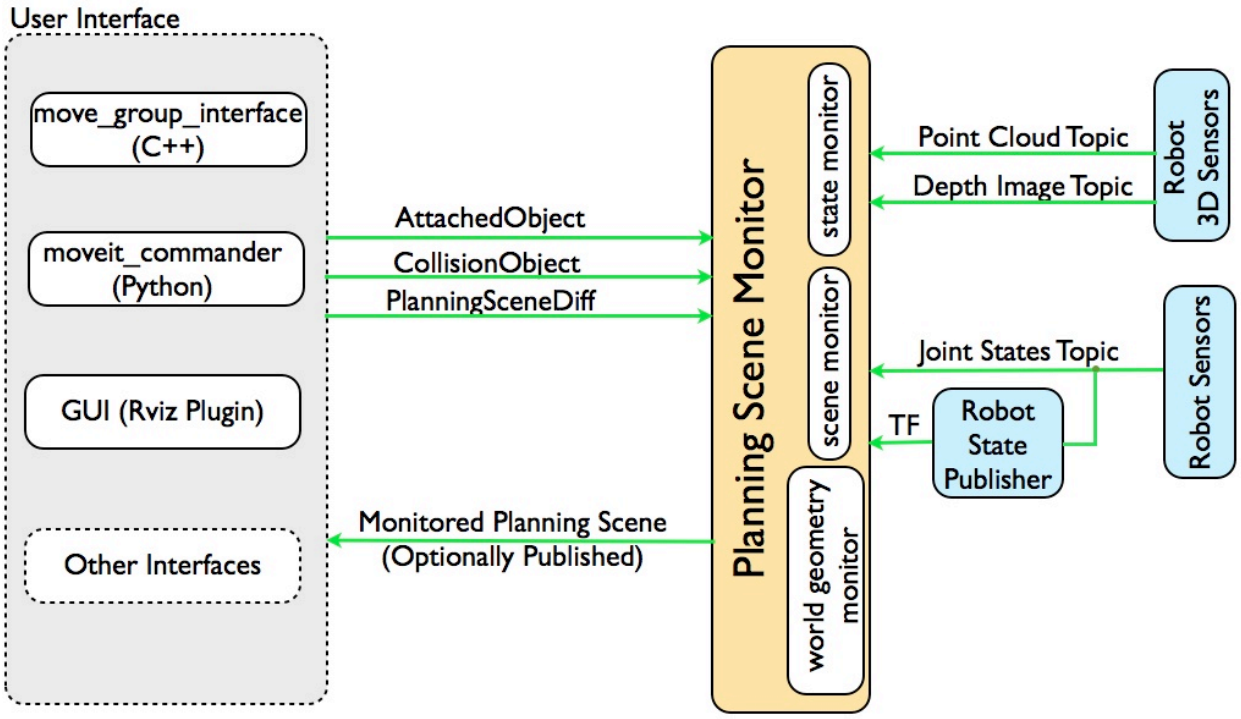
The figure below shows the high-level architecture of MoveIt! that is characterized by the *move_group* node, the most important part of the library, pulling all the individual components together to provide a set of ROS actions and services.



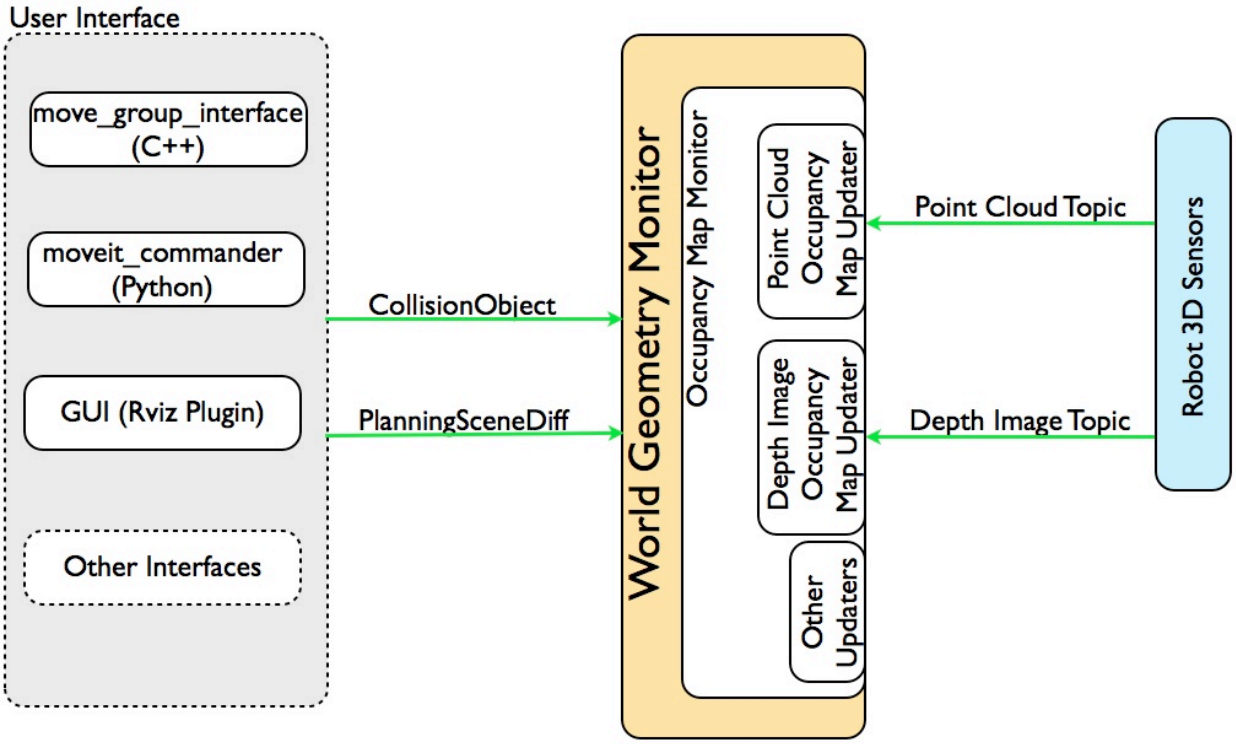
These actions and services can be accessed by way of a user interface that, in the standard version of the package, supports C++, through the *move_group_interface*, Python, through the *moveit_commander*, and a Rviz GUI, thanks to a suitable plugin. Moreover, *move_group* can

access the ROS parameter server, from which it gets the URDF and SRDF models for the robot, and all the configuration parameters.
On the other side, as it can be seen in the figure, *move_group* communicates with the robot to get the current state (*/joint_states* topic), with the controllers on the robot to command motion actions (*FollowJointTrajectoryAction* interface), and with all the exteroceptive sensors to acquire information about the environment.

The main functionalities provided by MoveIt! are trajectory planning and execution. Both functionalities are based on the Planning Scene Monitor that allows to maintain a planning scene, i.e., a representation of the world and the current state of the robot.

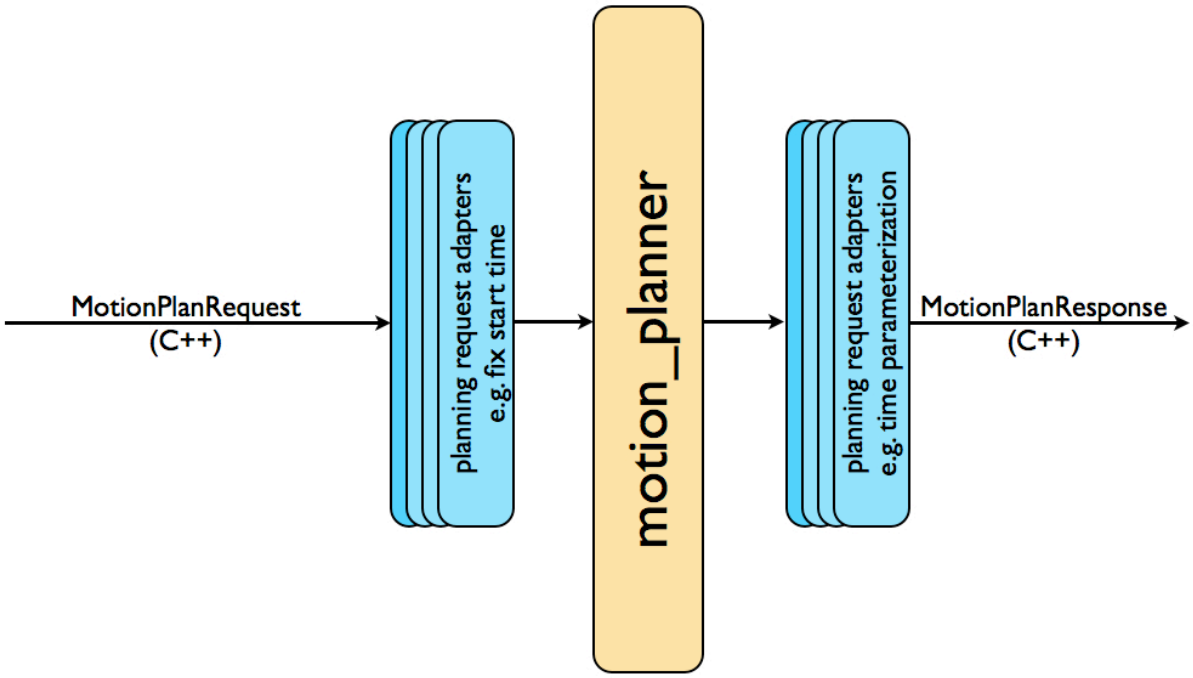


As shown in the figure above, the planning scene is maintained by the *planning scene monitor*, that is part of the *move_group* node. The planning scene monitor collects information on the robot state, from the */joint_states* topic, on the environment, using the world geometry monitor, and exploits a geometry model of the world created by the user.
The world geometry monitor builds a geometry model of the world using information coming from sensors on the robot and from user input.



A 3D representation of the environment around the robot, based on an Octomap (http://octomap.github.io/), is maintained by the occupancy map monitor. This component is based on a plugin architecture, allowing to handle different kind of sensor inputs (in particular, MoveIt! has inbuilt support for handling point clouds and depth images).
The information made available by the occupancy map is automatically exploited by the MoveIt! collision checking library.

As already mentioned, the aim of MoveIt! is to provide trajectory planning and execution in an unstructured and a-priori unknown environment.
Concerning motion planning, MoveIt! allows the adoption of different planning techniques, as motion planning libraries are interfaced to MoveIt! through a *plugin* interface. The default motion planning library, that is the one used in Grape project, is the Open Motion Planning Library (http://ompl.kavrakilab.org) that provides many state-of-the-art sampling-based motion planning algorithms.
Through the planning plugin, MoveIt! allows to plan a trajectory to move an arm to a different location in joint space or to a new pose of the end-effector. As collision checking, including self-collisions, is performed by MoveIt!, the motion planner can take into account any object, i.e., a camera or a laser scanner, attached to any part of the robot, providing that it has been properly defined in the robot model.
Finally, motion planning constraints, i.e., position constraints, orientation constraints, visibility constraints, joint constraints, or user-specified constraints, can be easily included in the planning problem.
The result of the motion planner is a trajectory that moves the arm to the desired location using the desired maximum velocities and accelerations. This trajectory can be further processed introducing in the motion planning pipeline a *planning request adapter*, as can be seen in the following figure, that can be used, for example, to assign a different time behavior to the trajectory generated by the planner.



Planning request adapters can be used to pre-process the planning request as well, for example to adjust a start state for the robot that is slightly outside the specified joint limitations.

5.2 – Arm task overview

The aim of the arm control software is to provide the platform a set of services to perform the following tasks:

- vineyard scan, the arm executes a predefined motion to perform a scan of the part of the vineyard in front of the platform in order to select the dispenser deploying location;
- homing to navigation position, during platform navigation the arm should take a configuration that is safe for itself and for the sensors mounted on it, and should maximize the platform stability with respect to roll-over/tip-over issues;
- dispenser deployment, the arm executes a partially predefined motion to take a dispenser from the feeder and deploy it at a predefined location on the vineyard;
- arm teleoperation, the remote operator can teleoperate the arm, using a keyboard or a joystick, in order to perform a manual inspection of the vineyard with a camera or a specific sensor.

The first two tasks, i.e., vineyard scan and homing to navigation position, are performed using MoveIt! planning and execution services.

Once a vineyard scan request is received, the robot starts from the navigation position, plans and executes a trajectory in joint space to reach the scanning position (during this phase the laser scanner is used to ensure obstacle detection and avoidance) and, finally, plans and executes a semicircular trajectory, from left to right and back, to perform a scan of the vineyard

in front of the platform. The activity is concluded planning and executing a trajectory back to the navigation position.

The vineyard scan data are processed by a further algorithm developed by Eurecat, that computes the deploying location and asks for the execution of the dispenser deployment task.

The dispenser deployment task is the most critical the arm has to perform due to the uncertainties in the position of the dispenser feeder and the deployment location, and to the elasticity/deformability of the dispenser that makes it an object difficult to be grasped and manipulated. These issues are particularly critical as the size and characteristics of Jaco 2 hand do not fit for the purpose of dispenser manipulation, as already explained in Chapter 3, and because the positioning accuracy of Jaco 2, probably due to problems in the kinematic inversion performed by the internal controller, is too rough to allow for a solution, based on calibration and proprioceptive sensors only, to grasp the dispenser from the feeder.
For this reason, a first set of tests has been performed using MoveIt! planning and execution algorithms with predefined calibrated grasping and deploying positions. As these tests reveal the aforementioned issues and the MoveIt! solution turns out to be not sufficiently robust, a different strategy has been adopted.
The task has been decomposed into the following parts:

- approaching the feeder, the arm starts from the navigation position and, using MoveIt!, a trajectory is planned and executed to move the end-effector to a pose close to the grasping zone;
- grasping the dispenser, to increase the robustness with respect to positioning errors, the last part of the approaching trajectory and the grasping operation are executed using an eye-in-hand-camera and a visual servoing control law (for further details see Chapter 6);
- approaching the deploying location, once the dispenser has been grasped and detached from the feeder the robot, using again MoveIt!, plans and executes a trajectory that connects the grasping zone to the deploying zone, i.e., a neighborhood of the selected deploying location;
- deploying the dispenser, being this another critical task, dispenser deploying is performed using visual servoing;
- homing to navigation position, once the deploying operation ends, the robot, using again MoveIt!, plans and executes a trajectory to the navigation position.

These macro-tasks are now decomposed, describing the specific activities that are included in each of them.

5.2.1 Dispenser grasping

The dispenser grasping task, that has been already decomposed into two activities, approaching the feeder and grasping the dispenser, is now described in more details.

The task can be decomposed into the following sub-tasks/activities:

1. approaching the feeder zone, using MoveIt! a trajectory is planned and executed from the actual robot position to a position located approximately 45 cm above the feeders. This position has been selected in accordance with the Realsense camera specifications, in order to take a RGB and a depth image of the feeder zone.
2. measuring the graspable dispenser, from the position reached at step 1 a RGB and a depth image are acquired. Processing these images, the height Z_d associated to the plane at which the graspable dispenser (the highest one on the feeders) belongs with respect to the world frame is determined. Moreover, an ellipse is fitted on the visible part

of the dispenser, and the X_d, Y_d positions of the center of the ellipse with respect to the world frame is determined.

3. measuring the marker position, a set of markers, whose geometry is known, is located at the base of the feeders and is used to guide the grasping operation during the visual servoing control phase. The RGB-D image considered in step 2 is here used to determine the 3D marker positions with respect to the world frame.
4. computing the visual servoing reference image, using the world position of the ellipse center (X_d, Y_d, Z_d), the end-effector pose given by the robot, the camera pose with respect to the end-effector frame, and the position of the nails in the grasping configuration with respect to the camera frame, the desired end-effector and/or camera position (with respect to the world frame) required to grasp the dispenser are computed. Using the camera intrinsic parameters and the 3D marker positions computed in step 3, a desired positions of the marker points on the image plane can be determined, as the image of the markers seen by a virtual camera located at the desired grasping position.
5. approaching the grasping zone, using MoveIt! a trajectory is planned and executed from the actual robot position to a position located approximately 25 cm above the feeders.
6. moving to the grasping position, from the robot position reached at the end of step 5, the visual servoing control law described in Chapter 6, aiming at taking the robot to the grasping position, is applied and the robot is moved to the grasping position.

5.2.2 Dispenser deploying

The dispenser deploying task, that has been already decomposed into two activities, approaching the deploying location and deploying the dispenser, is now described in more details.

Assuming the deploying position is known, i.e., selected by the user or computed by another algorithm, the task can be decomposed into the following sub-tasks/activities:

1. approaching the vineyard, using MoveIt! a trajectory is planned and executed from the actual robot position to a position located approximately 45 cm from the selected deploying location. This position has been chosen in accordance with the Realsense camera specifications in order to take a RGB and a depth image of the deploying zone.
2. modelling the deploying branch, the camera is moved up-and-down and left-and-right along two circular paths, keeping the same distance from the deploying location, and 5 (or more) pictures are taken from different point of views. A point cloud, representing the deploying branch, is then generated.
3. measuring the deploying location, using the points of the point cloud belonging to a spherical region around the deploying location, an ellipsoid is fitted and the center and principal axis are computed.
4. approaching the deploying zone, considering that the positions of the foci of the ellipse, with respect to the end-effector (and thus the camera) frame, that represents the shape of a dispenser during the grasping phase, are approximately known; the equation of the major axis of the ellipsoid modelling the deploying location and the deploying location are known; using MoveIt! a trajectory is planned and executed from the actual robot position to a position located along the ellipsoid major axis at approximately 25 cm from the selected deploying location.
5. moving to the deploying position, from the robot position reached at the end of step 6, the robot is moved along the end-effector Z axis, following the ellipsoid major axis, of approximately 25 cm, while a visual impedance control is applied on the end-effector

X,Y-plane to keep the dispenser centered with respect to the deploying branch. A detailed description of this control algorithm is reported in Chapter 6.

6. dispenser deploying, from the robot position reached at the end of step 5 a precomputed motion, constituted by a linear motion along the end-effector Z axis and a rotation around the end-effector X and Y axis to make the end-effector Z axis parallel to the world Z axis is executed, and the fingers are closed in order to deploy the dispenser.

6 – Visual servoing methods

This chapter introduces the visual servoing control algorithm develop by POLIMI to increase the accuracy of dispenser grasping and deployment tasks, and the procedure devised to calibrate the Realsense camera.

6.1 – RGB-D camera calibration

As discussed in Chapter 1, the camera adopted for the visual servoing control is the Intel Realsense, a RGB-D camera that provides a color and a depth image of the scene.

Before introducing the visual servoing control law, the procedure that has been set up to calibrate the camera is described.
First of all, it must be noticed that the intrinsic camera parameters, and the parameters that correlate the color to the depth image have been calibrated by the manufacturer, and can be retrieved using the camera driver. For this reason, these parameters has not been recomputed in the calibration procedure.
On the other side, a calibration procedure to compute the extrinsic camera parameters, i.e., the pose of the camera frame with respect to the end-effector frame, has been devised.
Though many calibration procedures exist in the literature, we decided to set up a new one being simple, able to exploit RGB and depth data and determining only the extrinsic parameters.

The position P of a point in the 3D space with respect to the robot world frame is given by the following relation

$$P = p^o_{eef}(q) + R^o_{eef}(q)\, p^{eef}_c + R^o_{eef}(q) R^{eef}_c p^c_p$$

where

- $p^o_{eef}(q)$, is the position of the end-effector frame with respect to the world frame, corresponding to the joint configuration q;
- $R^o_{eef}(q)$, is the orientation of the end-effector frame with respect to the world frame, corresponding to the joint configuration q;
- p^{eef}_c, is the position of the camera frame with respect to the end-effector frame;
- R^{eef}_c, is the orientation of the camera frame with respect to the end-effector frame;
- p^c_p, is the position of the object with respect to the camera frame;
- q, is the vector of joint coordinates.

Assume now that the robot is moved to two different positions, keeping the same orientation, and two images are taken, one for each position, of the same object in the same position. Focusing on the position of the same point P we can write the following relations

$$P = p^o_{eef}(q_1) + R^o_{eef}(q_1)\, p^{eef}_c + R^o_{eef}(q_1) R^{eef}_c p^c_{p_1}$$

$$P = p^o_{eef}(q_2) + R^o_{eef}(q_2)\, p^{eef}_c + R^o_{eef}(q_2) R^{eef}_c p^c_{p_2}$$

where

- q_1, q_2, are the vector of joint coordinates describing the two robot positions;
- $p^c_{p_1}$, $p^c_{p_2}$, are the positions of point P with respect to the camera frame in the two robot positions;

- $R^o_{eef}(q_1) = R^o_{eef}(q_2)$, as the robot is moved keeping the same end-effector orientation.

Finally, taking the difference of the previous relations one obtains

$$P = p^o_{eef}(q_1) + R^o_{eef}(q_1)\, p^{eef}_c + R^o_{eef}(q_1) R^{eef}_c p^c_{p_1}$$

$$P = p^o_{eef}(q_2) + R^o_{eef}(q_2)\, p^{eef}_c + R^o_{eef}(q_2) R^{eef}_c p^c_{p_2}$$

$$0 = \left[p^o_{eef}(q_1) - p^o_{eef}(q_2)\right] + R^o_{eef}(q_1)\, R^{eef}_c (p^c_{p_1} - p^c_{p_2})$$

and, consequently

$$R^{eef}_o(q_1)\left[p^o_{eef}(q_2) - p^o_{eef}(q_1)\right] = R^{eef}_c (p^c_{p_1} - p^c_{p_2})$$

The previous relation can be used to estimate the camera orientation with respect to the end-effector frame. In fact, $p^o_{eef}(q_1)$ and $p^o_{eef}(q_2)$ are two known robot positions, $R^o_{eef}(q_1)$ is the known robot orientation, that can be computed from joint coordinates using the forward kinematics, $p^c_{p_1}$ and $p^c_{p_2}$ are the positions of point P in the camera frame, that can be computed from the camera measurements using the perspective projection and the known camera intrinsic parameters, and R^{eef}_c is the unknown. Consequently, the previous relation represents a set of 3 linear equations in 9 unknowns, the 9 elements of the rotation matrix.
If the same reasoning is extended to six points in the robot workspace, always keeping the same end-effector orientation, the system can be rewritten as 9 equations in 9 unknowns as follows

$$Ax = b$$

where

$$A = \begin{bmatrix} p^c_{p_1} - p^c_{p_2} \\ p^c_{p_3} - p^c_{p_4} \\ p^c_{p_5} - p^c_{p_6} \end{bmatrix} \qquad x = R^{eef}_c \qquad b = \begin{bmatrix} R^{eef}_o(q_1)\left[p^o_{eef}(q_2) - p^o_{eef}(q_1)\right] \\ R^{eef}_o(q_1)\left[p^o_{eef}(q_4) - p^o_{eef}(q_3)\right] \\ R^{eef}_o(q_1)\left[p^o_{eef}(q_6) - p^o_{eef}(q_5)\right] \end{bmatrix}$$

The orientation matrix can be thus computed as

$$x = A^{-1} b$$

The previous procedure can be easily extended to consider more than six robot positions, to increase the robustness of the estimate, solving the linear system by way of a lest-squares solution.

Due to noise and errors, the orientation matrix computed with the previous algorithm is not orthonormal, an orthonormalization step must be thus introduced in the algorithm.

Calling X and Y the first two columns of matrix R^{eef}_c, we can define the error as $X^T Y$ and, consequently, two orthogonal columns can be introduced as follows

$$X_{orthogonal} = X - \frac{X^T Y}{2} Y \qquad \text{and} \qquad Y_{orthogonal} = Y - \frac{X^T Y}{2} X$$

and

$$Z_{orthogonal} = X_{orthogonal}\, x\, Y_{orthogonal}$$

Finally, a set of orthonormal columns can be computed as

$$X_{orthonormal} = \frac{1}{2}(3 - X_{orthogonal} \cdot X_{orthogonal})X_{orthogonal}$$

$$Y_{orthonormal} = \frac{1}{2}(3 - Y_{orthogonal} \cdot Y_{orthogonal})Y_{orthogonal}$$

$$Z_{orthonormal} = \frac{1}{2}(3 - Z_{orthogonal} \cdot Z_{orthogonal})Z_{orthogonal}$$

and the orthonormal version of the rotation matrix is given by

$$R_c^{eef} = [X_{orthonormal} \quad Y_{orthonormal} \quad Z_{orthonormal}]^T$$

Once the camera orientation has been computed, the initial relation can be exploited to compute the camera position with respect to the end-effector frame

$$p_c^{eef} = (R_{eef}^o(q))^{-1}[P - p_{eef}^o(q) - R_{eef}^o(q)R_c^{eef}p_p^c]$$

6.2 – Visual servoing control law

The visual servoing approach herein described is a standard Image Based Visual Servoing control algorithm[1].

Define a feature point as the vector

$$f = \begin{bmatrix} u \\ v \end{bmatrix}$$

where (u, v) are the coordinate of the point on the image plane representing the image of the considered point in 3D space. In case more than one 3D point, and thus more than one feature point, is considered, a feature vector can be defined as follows

$$\boldsymbol{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

In IBVS the goal configuration is defined by a desired configuration $\boldsymbol{f}_d$ of the image features. Therefore, the image error function is given by

$$\boldsymbol{e}(t) = \boldsymbol{f}(t) - \boldsymbol{f}_d$$

The image-based control problem aims at finding a mapping from this error function to a commanded camera motion, under the following assumptions:

- the manipulator is a kinematic positioning device, i.e., manipulator dynamics are neglected;
- the trajectories generated by the IBVS controller can be tracked by a lower level manipulator controller;
- the target is fixed and the desired configuration $\boldsymbol{f}_d$ is constant.

The most common approach to solve the aforementioned problem is to compute a desired camera velocity and use this as the control input, assuming that the robot can be commanded by a set of linear and rotational Cartesian velocities.

[1] F. Chaumette, S. Hutchinson, Visual servo control, Part I: basic approaches, IEEE Robotics and Automation Magazine, December 2006, pp. 82-90

The camera velocity can be computed using the well-known relation, represented by the interaction matrix, between the velocity of the camera frame and the velocity of the features on the image plane

$$\dot{\boldsymbol{f}}(t) = L(f, \boldsymbol{q})\boldsymbol{v}_{cam}$$

where the interaction matrix of a feature point is given by

$$L(f, \boldsymbol{q}) = \begin{bmatrix} -\frac{f_u}{Z} & 0 & \frac{u}{Z} & \frac{uv}{f_u} & -\frac{f_u^2 + u^2}{f_u} & v \\ 0 & -\frac{f_v}{Z} & \frac{v}{Z} & \frac{f_v^2 + v^2}{f_v} & -\frac{uv}{f_v} & -u \end{bmatrix}$$

and $\boldsymbol{v}_{cam}$ is the vector of linear and rotational Cartesian velocities of the camera. In the case of a vector of feature points, the interaction matrix can be obtained stacking each interaction matrix associated to each point.
Assuming that the number of selected features is such that the whole interaction matrix as more rows than columns, the previous relation can be inverted using a least-squares solution as follows

$$\boldsymbol{v}_{cam} = (L^T L)^{-1} L^T \dot{\boldsymbol{f}}(t)$$

For the system

$$\dot{\boldsymbol{f}}(t) = L(f, \boldsymbol{q})\boldsymbol{v}_{cam}$$

with error defined by

$$\boldsymbol{e}(t) = \boldsymbol{f}(t) - \boldsymbol{f}_d$$

consider the candidate Lyapunov function

$$V(t) = \frac{1}{2}\boldsymbol{e}^T\boldsymbol{e}$$

The time derivative of this function is

$$\dot{V}(t) = \boldsymbol{e}^T\dot{\boldsymbol{e}}$$

If a controller can be designed such that

$$\dot{\boldsymbol{e}} = -\lambda\boldsymbol{e}$$

with $\lambda > 0$, the derivative of the Lyapunov function becomes

$$\dot{V}(t) = -\lambda\boldsymbol{e}^T\boldsymbol{e} < 0$$

and this would ensure exponential stability of the closed loop system, i.e., the system would be asymptotically stable even under small perturbations.
Finally, as the derivative of the error function is given by

$$\dot{\boldsymbol{e}}(t) = \frac{d}{dt}(\boldsymbol{f}(t) - \boldsymbol{f}_d) = \dot{\boldsymbol{f}}(t) = L(f, \boldsymbol{q})\boldsymbol{v}_{cam}$$

the following control law is obtained

$$\boldsymbol{v}_{cam} = (L^T L)^{-1} L^T \dot{\boldsymbol{e}}(t)$$

As the Kinova ROS interface allows to command the robot using a Cartesian velocity vector referred to the end-effector frame, the last step is the derivation of the end-effector velocity from the camera velocity.

Assuming that the pose of the camera frame with respect to the robot end-effector frame has been determined by way of a calibration process, the following relations can be written

$$^{o}p_c = {}^{o}p_e + {}^{o}R_e\ {}^{e}p_c$$
$$^{o}R_c = {}^{o}R_e\ {}^{e}R_c$$

where

- $^{o}p_c$ and $^{o}R_c$, are the position and orientation of the camera frame with respect to the world frame;
- $^{o}p_e$ and $^{o}R_e$, are the position and orientation of the end-effector frame with respect to the world frame;
- $^{e}p_c$ and $^{e}R_c$ are the position and orientation of the camera frame with respect to the end-effector frame.

Deriving the previous relations with respect to time, one obtains

$$^{o}\dot{p}_c = {}^{o}\dot{p}_e + {}^{o}\dot{R}_e\ {}^{e}p_c = {}^{o}\dot{p}_e + S({}^{o}\omega_e)\ {}^{o}R_e\ {}^{e}p_c$$
$$^{o}\dot{R}_c = {}^{o}\dot{R}_e\ {}^{e}R_c \quad \Rightarrow \quad S({}^{o}\omega_c)\ {}^{o}R_c = S({}^{o}\omega_e)\ {}^{o}R_e\ {}^{e}R_c$$

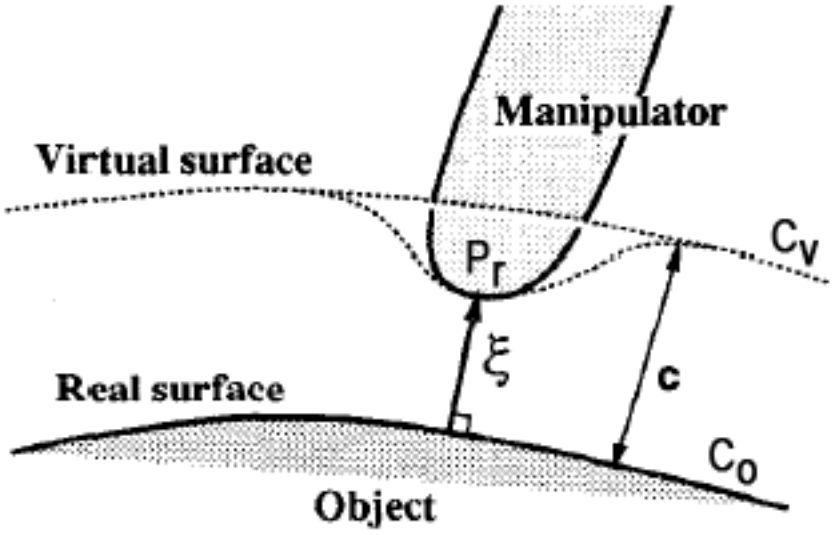
From which the relation between the camera linear and rotational velocities ($\boldsymbol{v}_{cam}$) and the end-effector linear and rotational velocities can be straightforwardly derived

$$^{o}\dot{p}_e = {}^{o}\dot{p}_c - {}^{o}\dot{R}_e\ {}^{e}p_c = {}^{o}\dot{p}_c - S({}^{o}\omega_e)\ {}^{o}R_e\ {}^{e}p_c$$
$$^{o}\omega_c = {}^{o}\omega_e$$

6.3 – Visual impedance control law

The control law here described is inspired by previous works on visual impedance[2].

Given two objects an impedance relation, based on their relative position on the image plane, can be defined as in the figure below (where one of the objects is the manipulator end-effector).



[2] Y. Nakabo, M. Ishikawa, “Visual impedance using 1ms visual feedback system”, IEEE International Conference on Robotics and Automation, 1998, pp. 2333-2338

Let C_0 be the pattern, extracted from the image, of an edge on the object, and $P(u,v)$ any point outside the object, where u,v are coordinates expressed in the image plane coordinate frame. The distance between P and the nearest point x on the edge can be expressed as follows

$$\varphi = min(|P - x|) \quad \forall x \in C_0$$

Given a constant c, a virtual surface is defined as

$$C_v = \{P_v | \varphi(P_v) = c\}$$

When the manipulator, or any other object, penetrate the virtual surface C_v a virtual contact occurs, and a contact vector is defined as

$$\boldsymbol{\xi} = \begin{cases} \big(c - \varphi(P_r)\big) \cdot \boldsymbol{n} & c \geq \varphi(P_r) \\ 0 \cdot \boldsymbol{n} & c < \varphi(P_r) \end{cases}$$

where $\boldsymbol{n}$ represents a suitable normal vector.

Introducing now a viscoelastic model of the virtual surface, a virtual force can be derived

$$\boldsymbol{F} = M\ddot{\boldsymbol{\xi}} + D\dot{\boldsymbol{\xi}} + K\boldsymbol{\xi}$$

A simple way to exploit the visual impedance approach in the contest of dispenser deploying is by applying implicit impedance or implicit position-force control, i.e., considering the visual impedance as a virtual force sensor.
In the case of position-force control the motion along the Z direction and in the X,Y plane are automatically decupled, applying position control along the Z direction and force control in the X,Y plane.

7 - Results

This chapter describes the laboratory experimental results related to the dispenser grasping and deploying tasks achieved by POLIMI using the Kinova Jaco 2 manipulator mounted on a table, the dispenser feeders and a model of a vineyard, that have been 3D printed by POLIMI.
Considering that a video showing the dispenser grasping and deployment operations has been already presented as means of verification of milestone M3, this chapter focuses on the results of the algorithms that have been adopted to robustify the approach. In particular, the most critical parts related to image processing are discussed.

7.1 – Vineyard model

In order to perform laboratory experiments of the dispenser grasping and deploying tasks in a realistic scenario, a point cloud model of two branches, acquired by Eurecat on a real vineyard, has been used to create a 3D printed model of the vineyard.

The figures below show the point cloud and the 3D printed model of the vineyard.

7.2 – Image processing and camera calibration

First of all, image processing is done using Realsense library (https://github.com/IntelRealSense/librealsense) and OpenCV library (https://opencv.org).

An example of image processing pipeline is given in the following.
RGB and depth images are both acquired, and the first one is used to search for an object characterized by a predefined color. The coordinates of the center of the object on the image plane are then computed using image moments

$$x = \frac{m_{10}}{m_{00}}, \qquad y = \frac{m_{01}}{m_{00}}$$

where pq-moment is defined as

$$m_{pq} = \sum_x \sum_y x^p y^q I(x,y)$$

$I(x,y)$ being a measure of the pixel intensity.
The depth associated to this point can be then extracted by the depth image using the stream `rs::stream::depth_aligned_to_color`. It is exploited to convert points on the image plane into points in the 3D space by way of the de-projection procedure, that is based on a distortion model calibrated by the manufacturer, and the perspective projection model of the camera

$$x = \frac{u - o_x}{f_x} z, \qquad y = \frac{v - o_y}{f_y} z$$

where

- u, v, are the coordinates of a point on the image plane expressed in pixel;
- x, y, z, are the coordinates of the corresponding point in 3D space in meters, with respect to the camera frame;
- o_x, o_y, are the coordinates of the image center in pixel;
- f_x, f_y, are the focal lengths in pixel/m.

To calibrate the camera pose with respect to the end-effector frame a target composed by a blue square market has been used, together with the procedure defined in Chapter 6.
Up to ten different robot positions have been considered, discarding those positions that after re-projection give rise to an excessive error, in order to minimize the effect of inaccuracies in robot positioning.
At the end the following pose for the camera frame with respect to the robot frame has been estimated

$$^e p_c = [-0.0823 \quad 0.0675 \quad -0.1375]^T$$
$$^e R_c = \begin{bmatrix} -0.9538 & -0.2864 & -0.0751 \\ 0.2937 & -0.9491 & -0.0084 \\ -0.0693 & -0.0302 & 0.9955 \end{bmatrix}$$

The calibration parameters have been then validated, acquiring other images with the marker located in known positions and estimating its position using the image data and the calibrated camera pose. The following table reports the results achieved on three different marker positions.

	Marker position	Estimated marker position	Error
X	0.0425	0.044633	-0.002133
Y	0.0000	0.000800	-0.000800
Z	-0.0350	-0.036000	0.001000
X	0.6700	0.6792	-0.0092
Y	0.0000	-0.0060	0.0060
Z	-0.0420	-0.0412	-0.0008
X	0.0000	0.0121	-0.0121
Y	0.6900	0.6965	-0.0065
Z	-0.0420	-0.0401	-0.0019

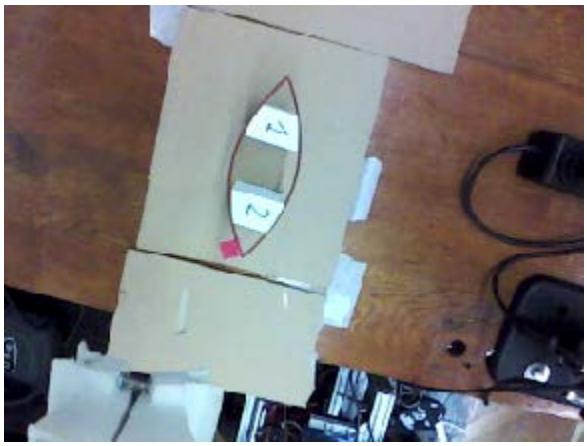
Despite the countermeasures adopted to increase the accuracy, the result is still affected by the low robot positioning accuracy.

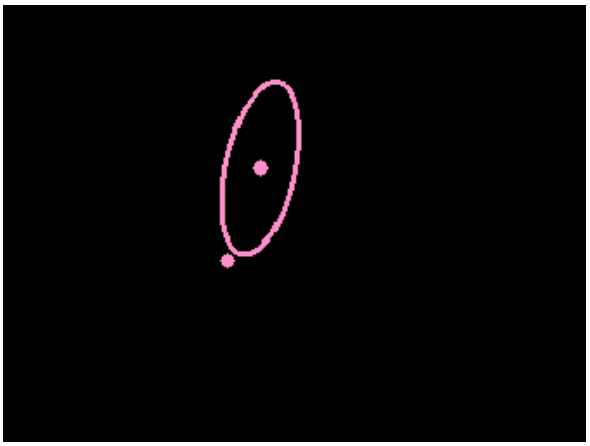
7.3 – Dispenser grasping

As already mentioned in Chapter 6, one of two most critical operations of the grasping, manipulation and deployment tasks is the grasping and detaching of the dispenser from the feeder. In fact, due to robot positioning errors and to the dispenser flexibility, the detaching operation can easily fail.

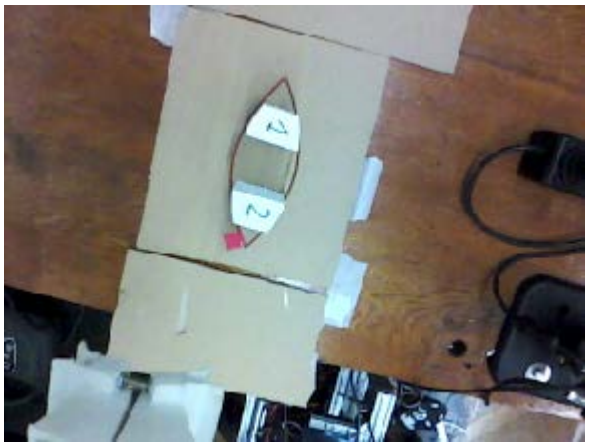
First of all, the robot starts framing the feeder zone from a distance of approximately 45 cm. Using OpenCv functionalities the visible parts of the dispenser are detected in the RGB image and an ellipse is fitted on this parts. Then, using the depth image the position of the ellipse center (X_d, Y_d, Z_d) with respect to the world frame is computed.

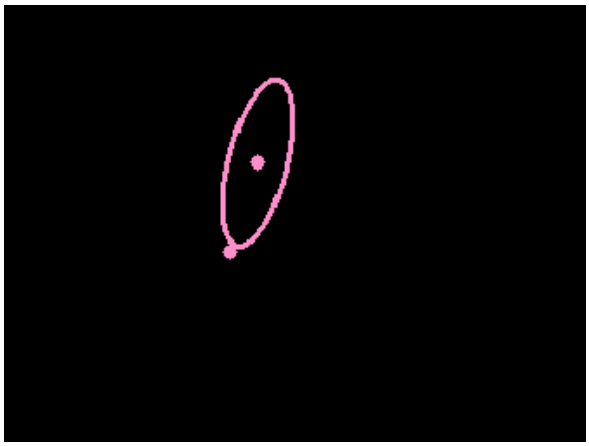
The figures below show three examples of dispenser detection and measuring. In particular, the RGB image, the depth image, and a synthetic image generated plotting the estimated ellipse and its center (whose coordinate are reported at the right of the depth image) are shown.

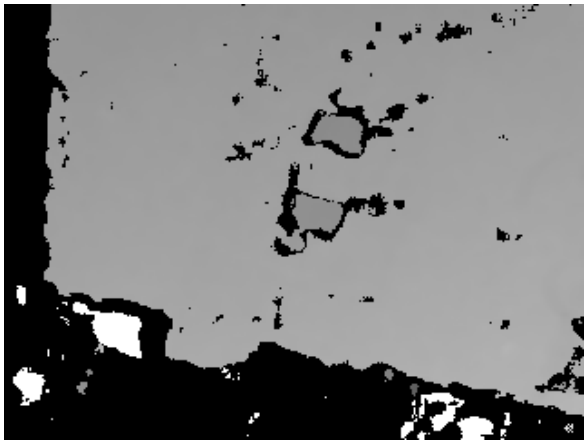




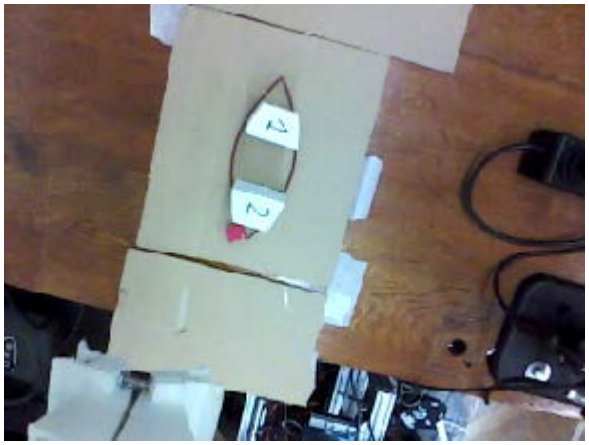
$X_d = 0.1504\ Y_d = 0.3872\ Z_d = 0.0247$

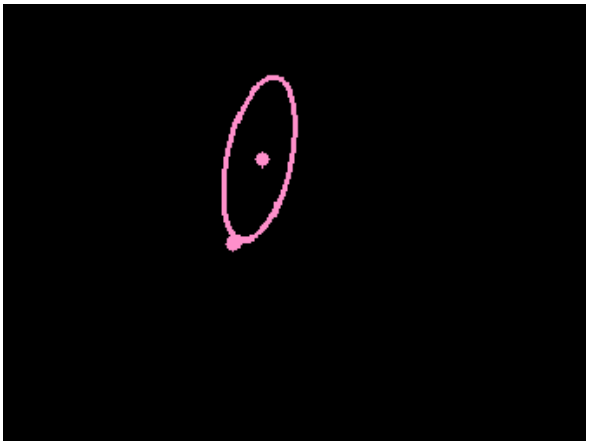


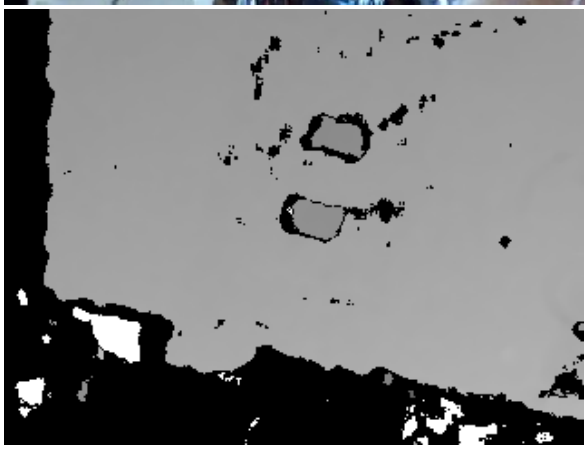




$X_d = 0.158\ Y_d = 0.3935\ Z_d = -0.0156$







$X_d = 0.163\ Y_d = 0.389\ Z_d = -0.0385$

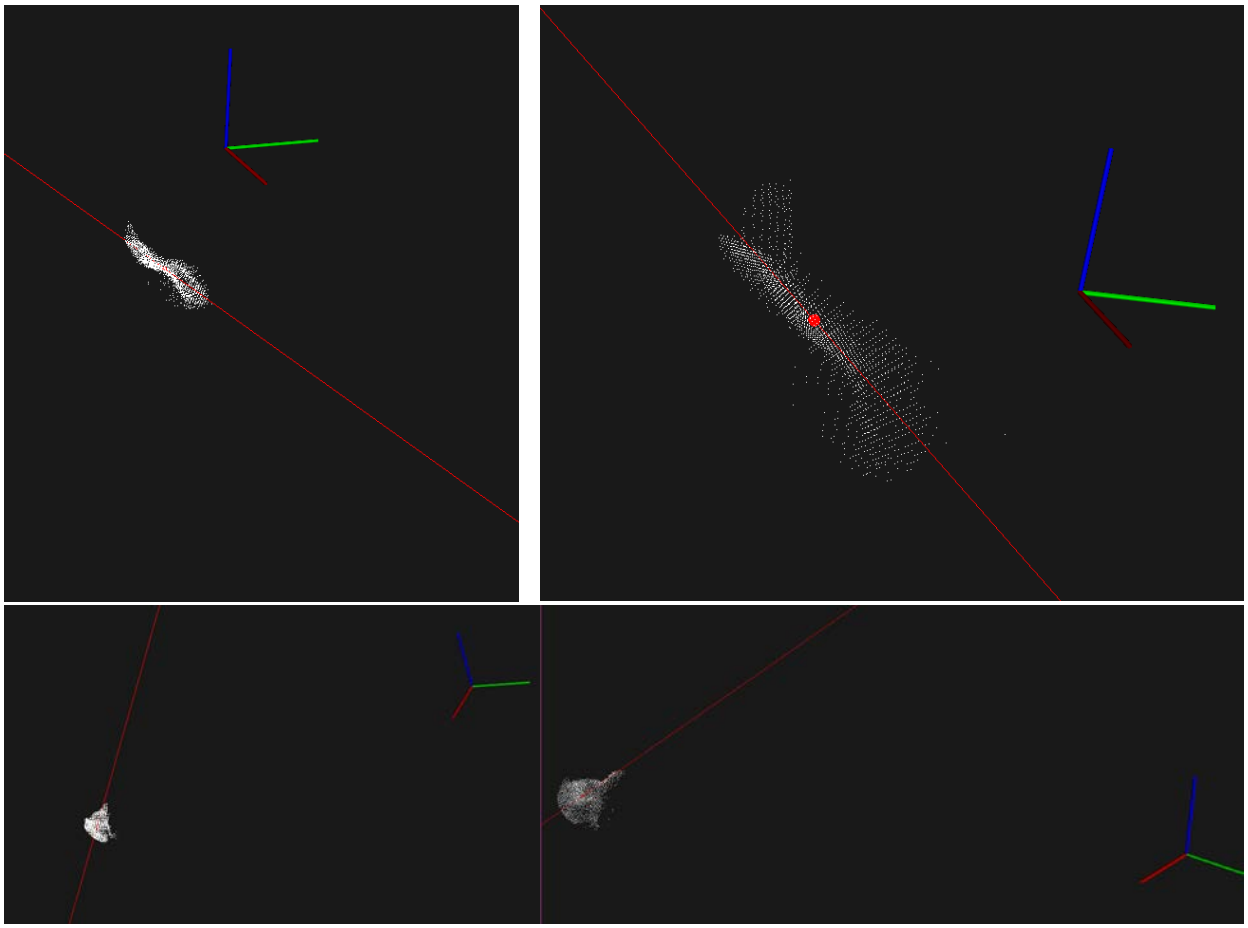
As can be seen from the above figures, a small piece of tape has been added to the dispenser. In fact, the dispenser capillaries are too short, compared to the resolution of the camera, to be seen by the depth image.

7.4 – Dispenser deploying

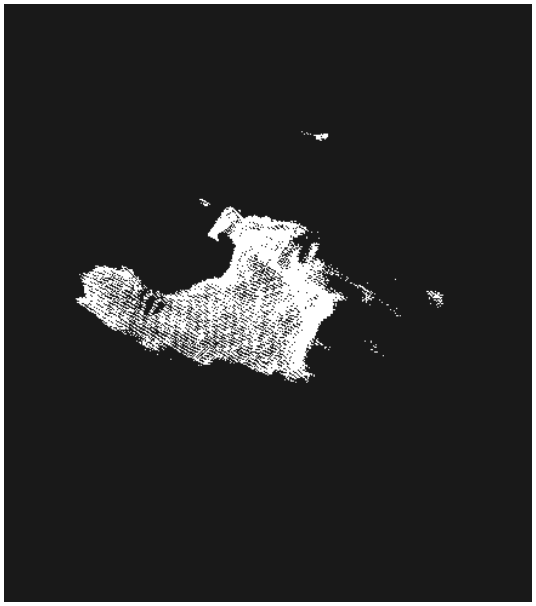
As already mentioned in Chapter 6, the second most critical operation of the grasping, manipulation and deployment tasks is the deploying of the dispenser on the vineyard. Though it is less critical than the previous one, as the dispenser has only to be released without entailing any manipulation, it must be considered that the geometry of the part of the branch on which the dispenser has to be deployed is completely unknown.

First of all, the robot starts framing the deploying zone from a distance of approximately 45 cm. Using the depth images taken from some positions around the deploying location, and PointCloud library (http://wiki.ros.org/pcl) functionalities, a model of the deploying location is determined. In particular, the neighborhood of the branch around the deploying location is approximated with an ellipsoid whose main axis and center are computed and used then to guide the robot during the approach and deploying phases.

The figures below show an example where a part of a branch of the mocap of the vineyard has been considered as deploying location and analyzed using the camera. Two different views of the point cloud describing the deploying region, and the related magnified images, are shown. The analysis of this point cloud determines the principal axis of the ellipse that is represented as a red line in the images.



The last two images show a close-up of the result of merging more point clouds already referred to the world reference frame.



7.5 – Conclusion

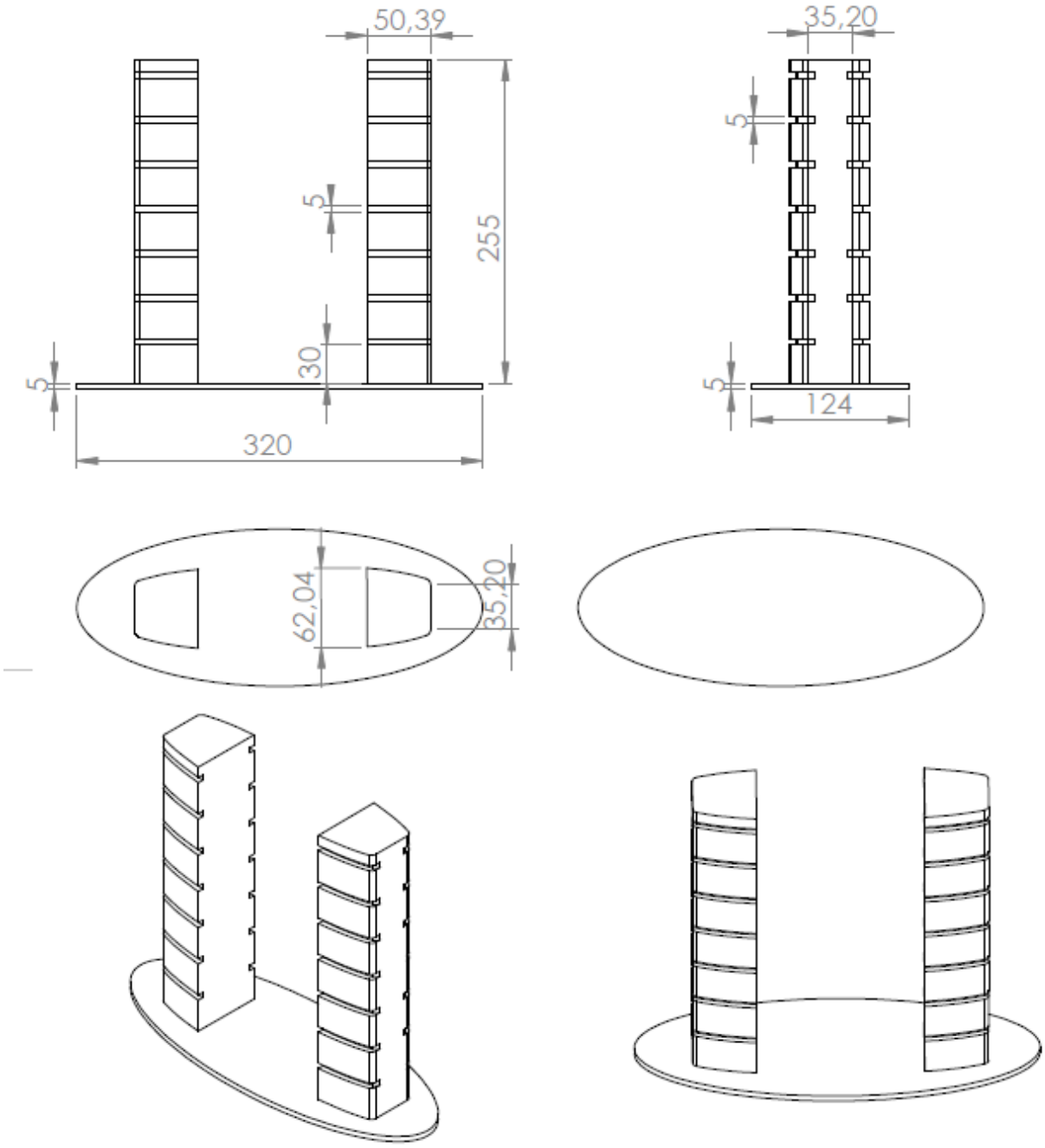
The procedures described in this deliverable effectively contributed making a dispenser grasping and deploying task more robust with respect to a simple position-control based solution.
Those procedures, however, have still to be improved and tested in a real outdoor environment.
Moreover, during the next integration step, the procedures here described to reconstruct the deploying zone may be substituted by an extension of the algorithm developed by Eurecat to select the deploying point.

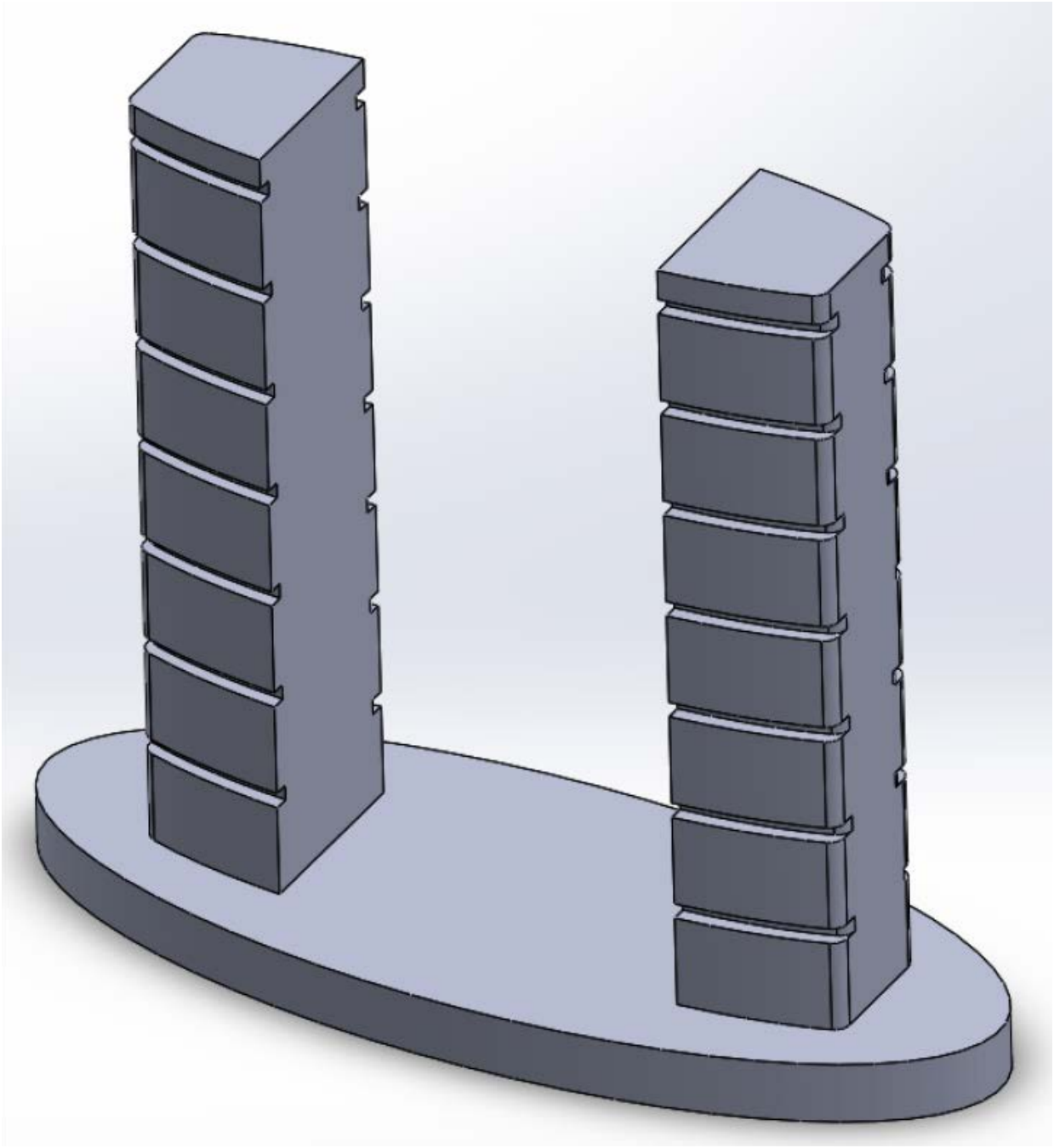
Annex I – Feeder and nail design

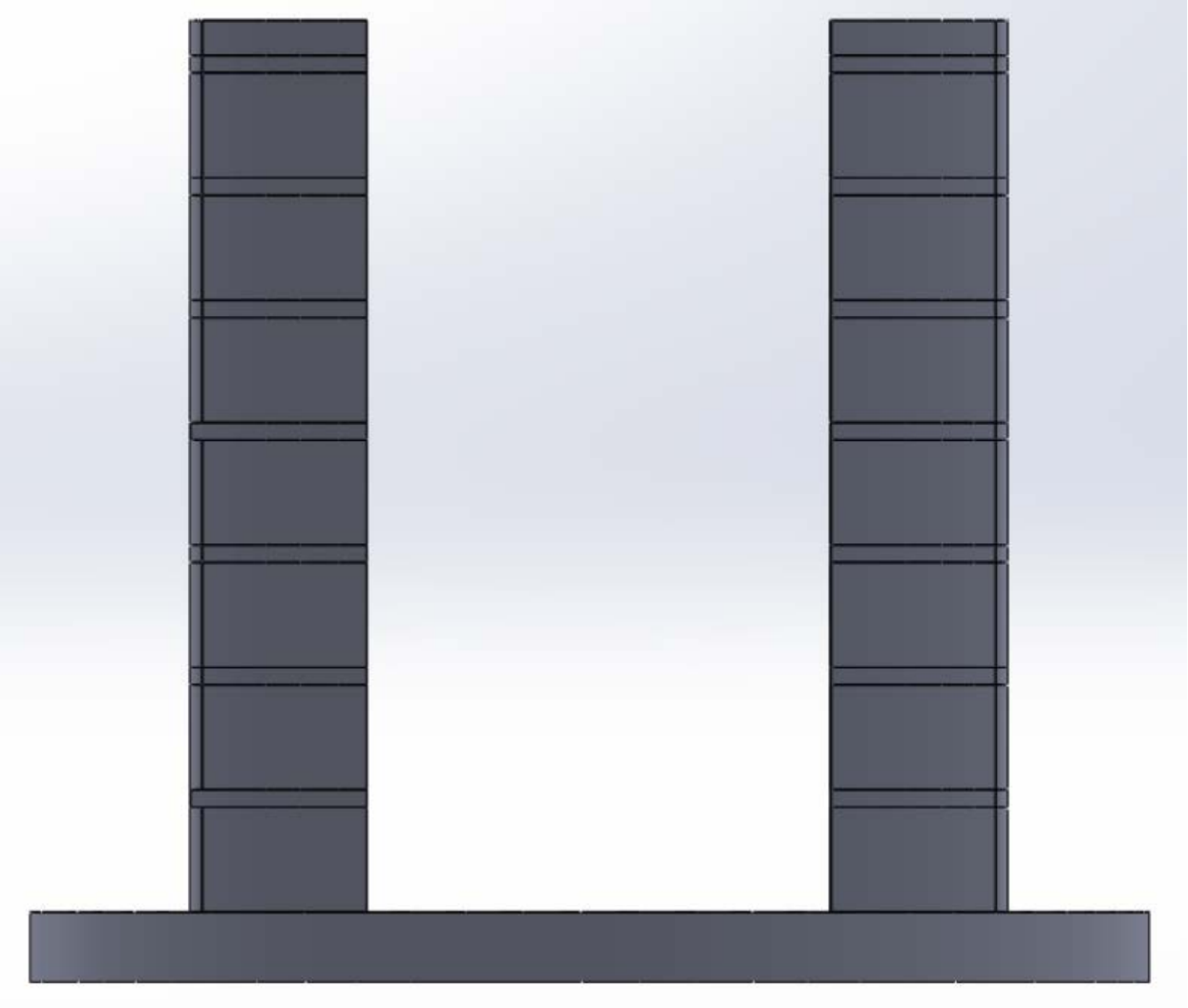
In this annex all the documentation related to feeder and nail design an 3D printing are reported.

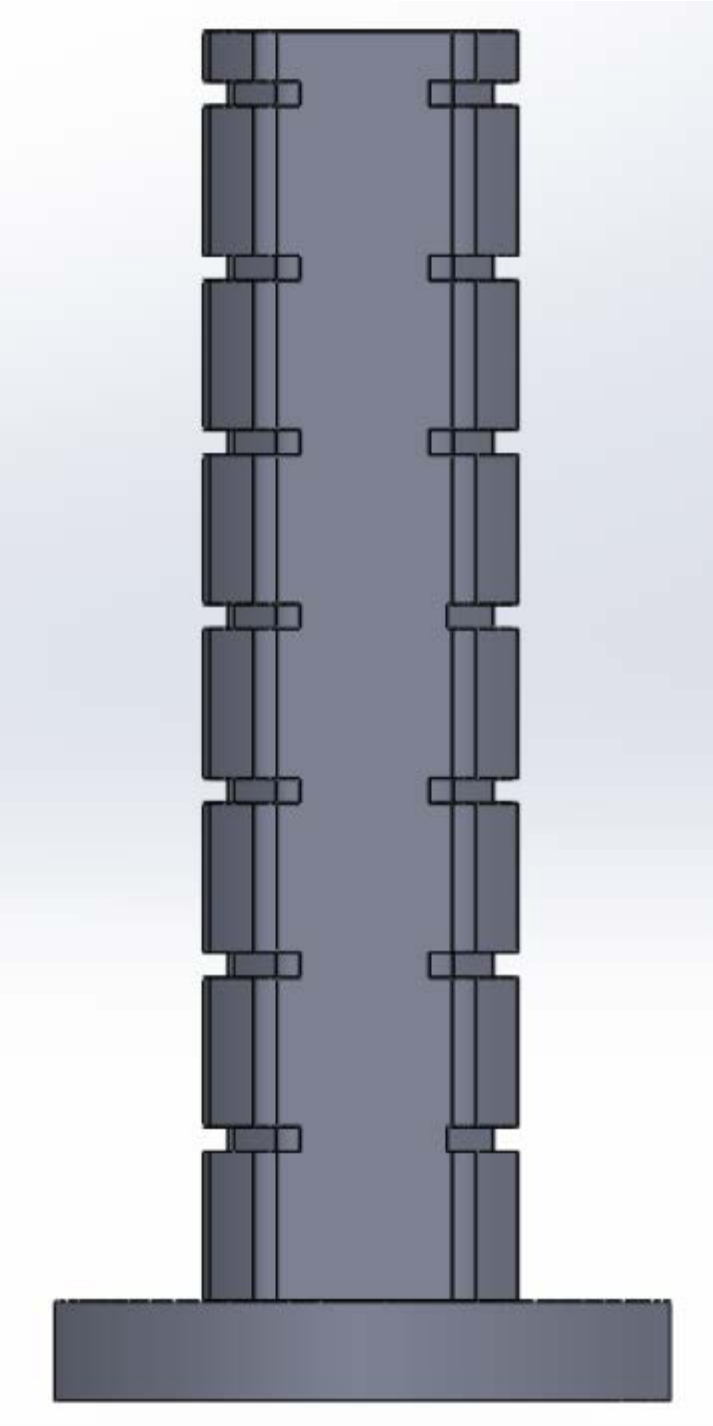
I.1 – Feeder design

I.1.1 – First design

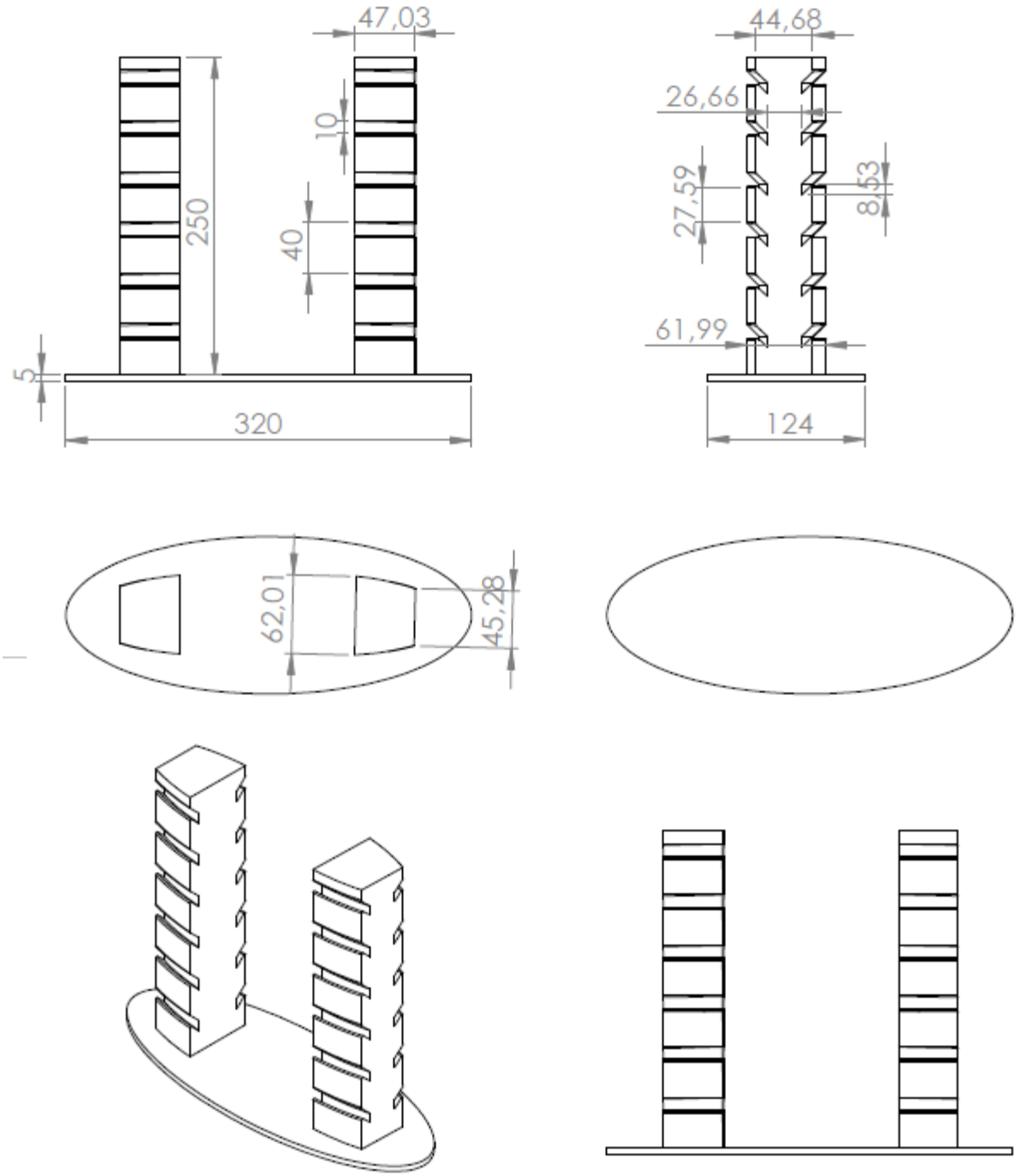


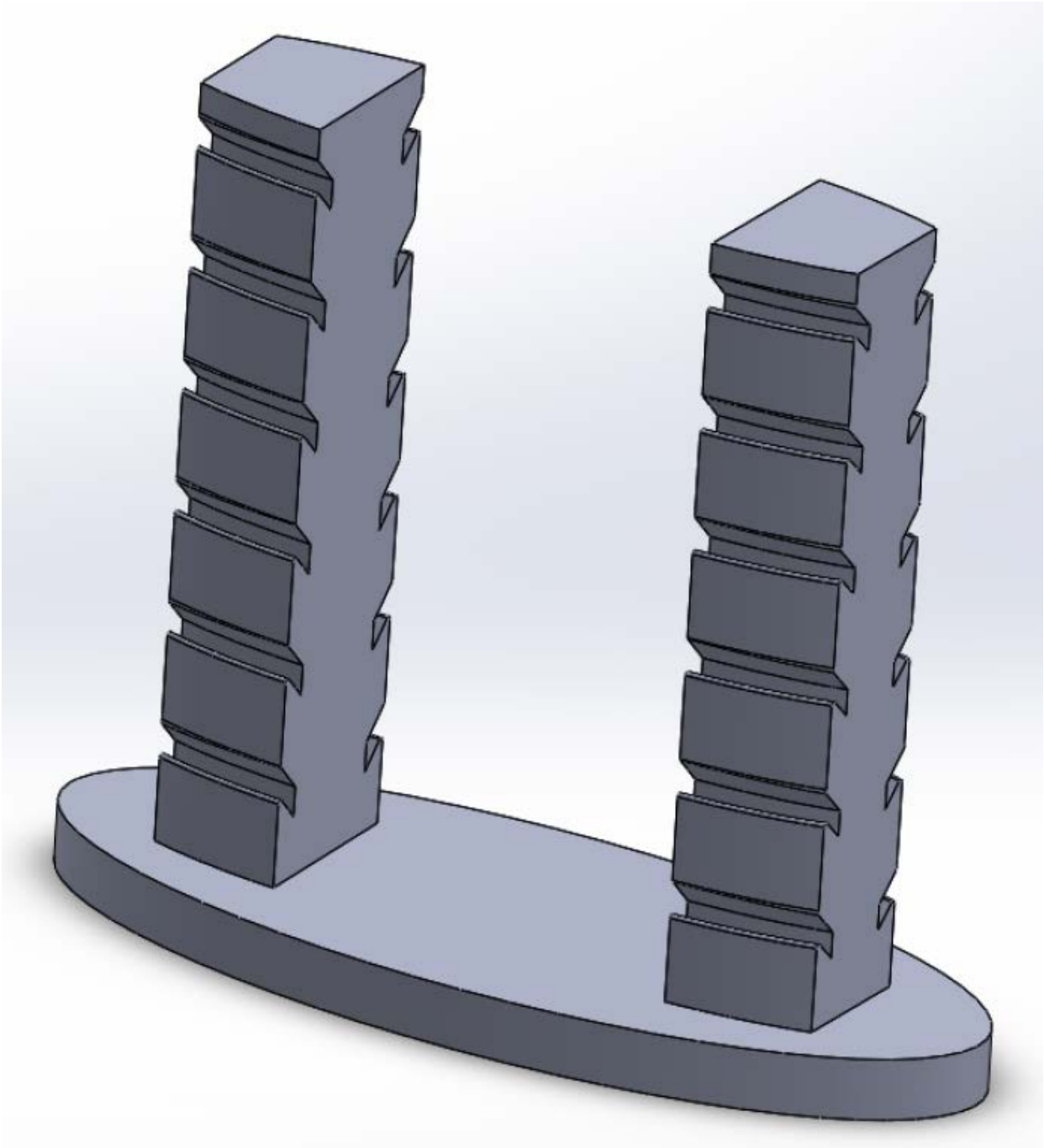


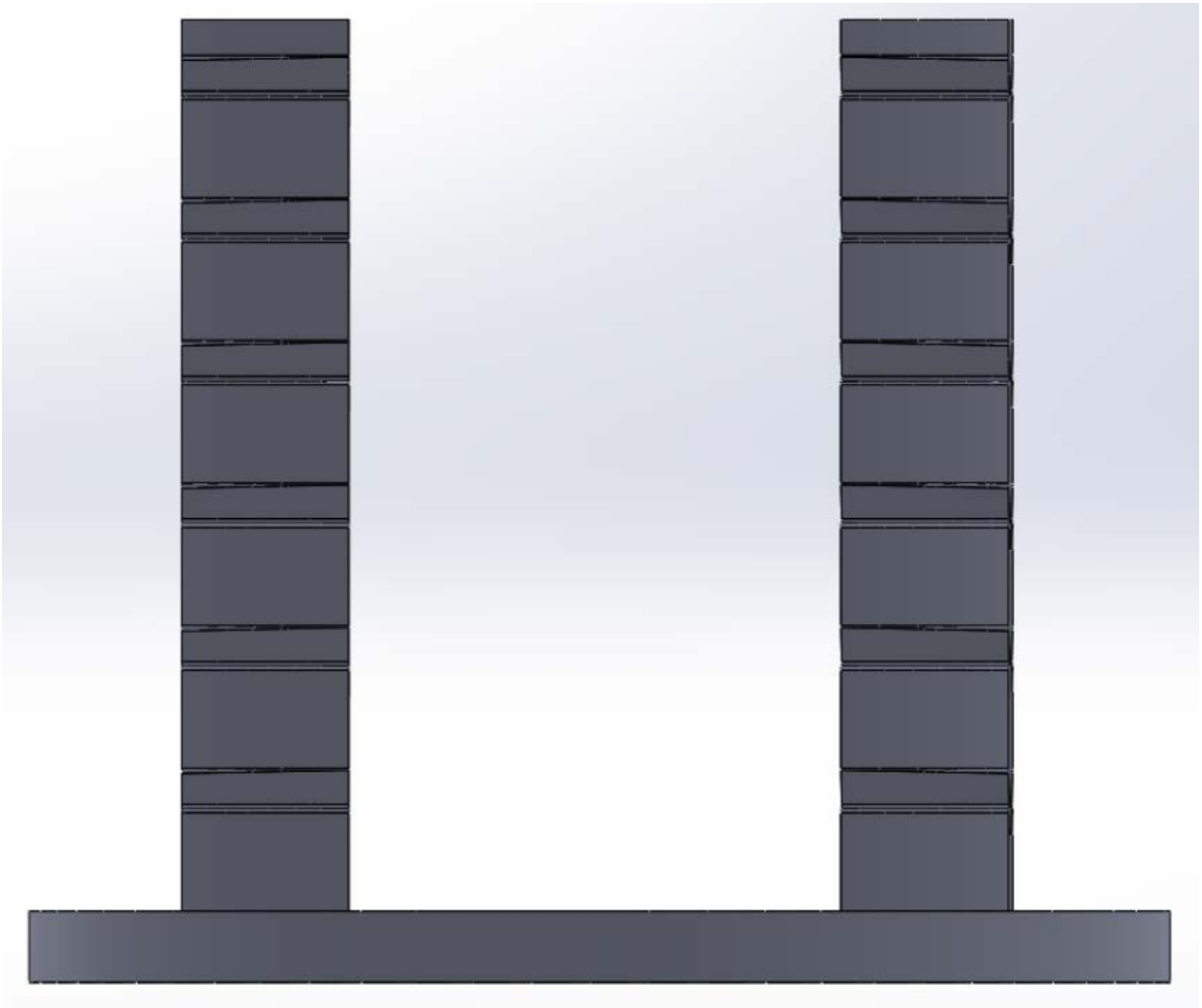


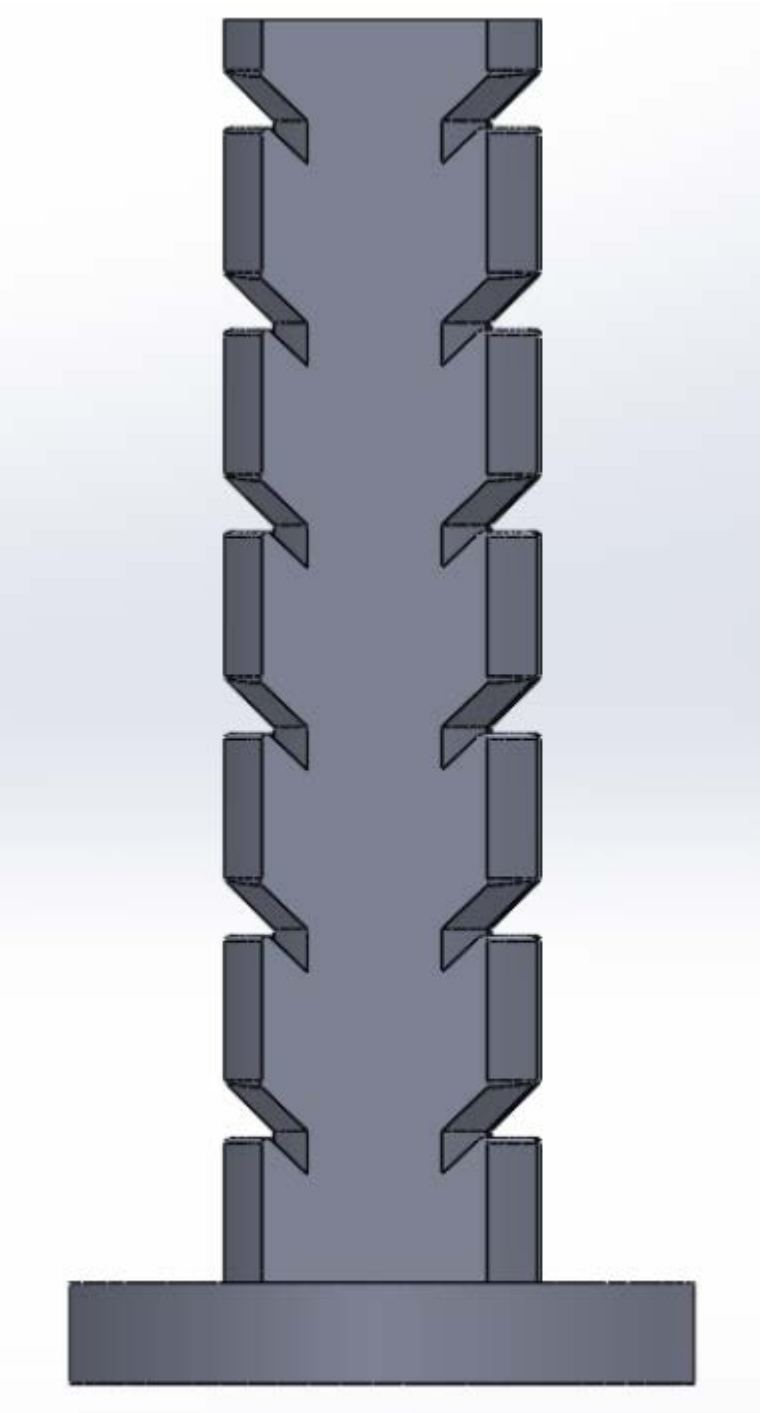


I.1.2 – Second design

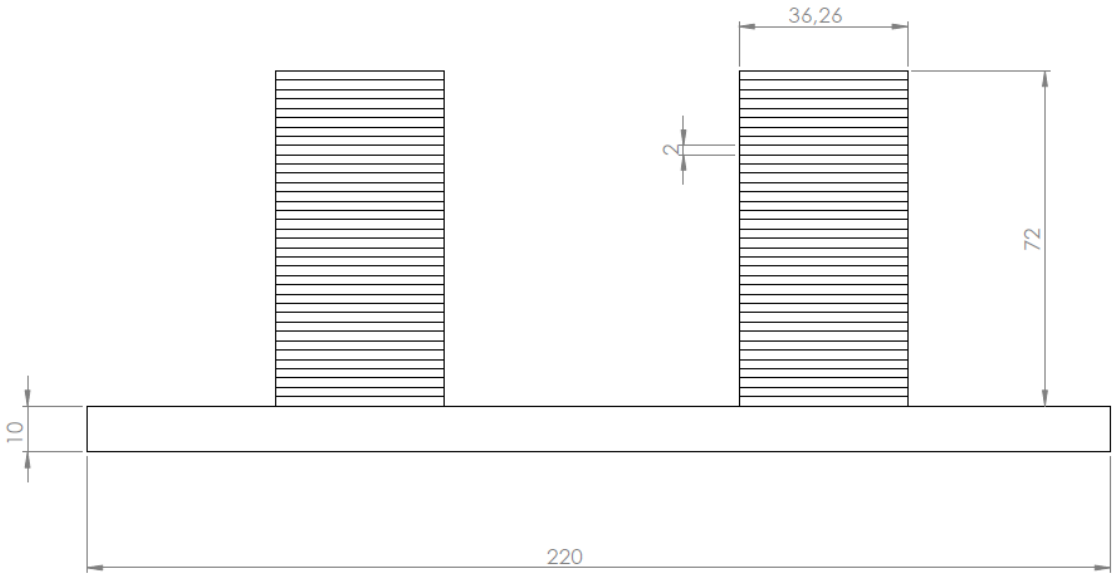


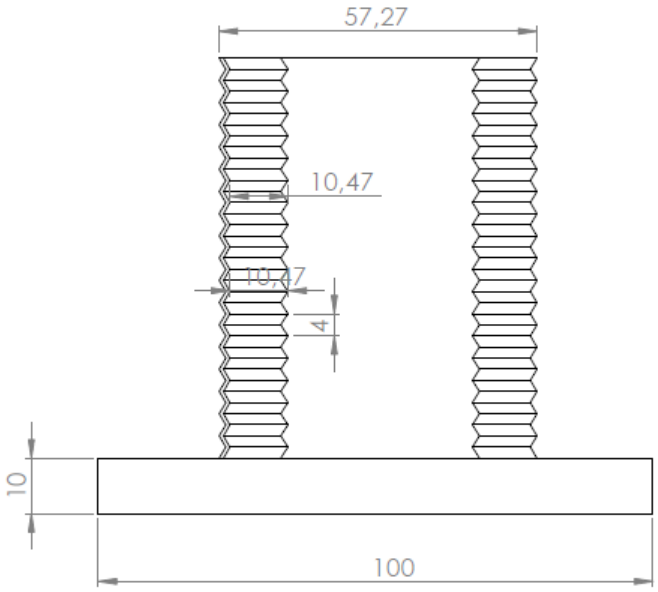


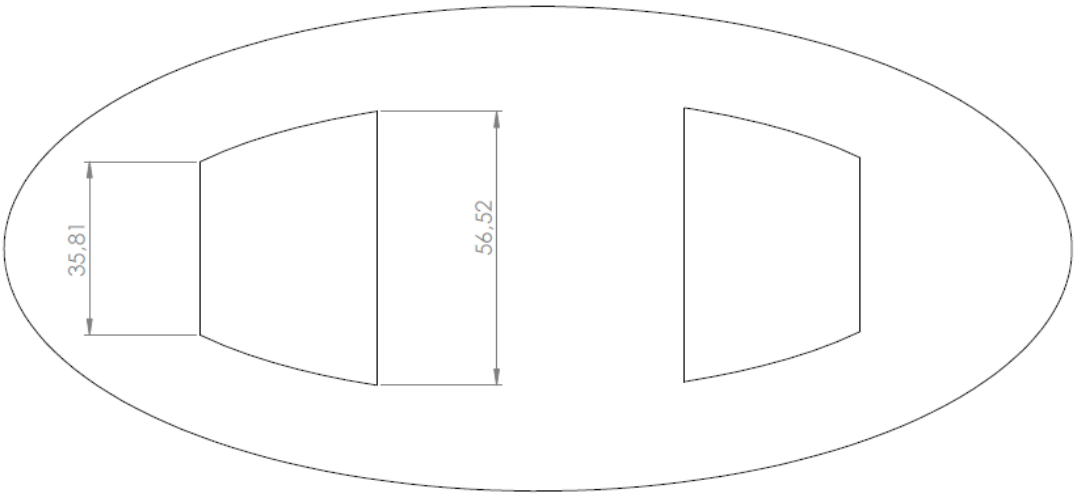


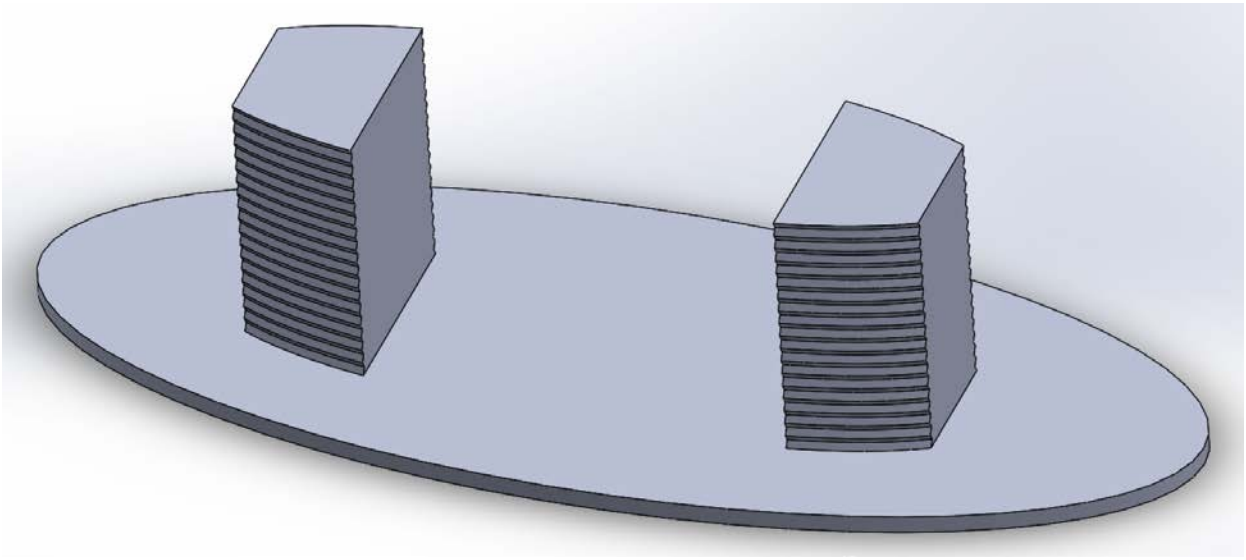


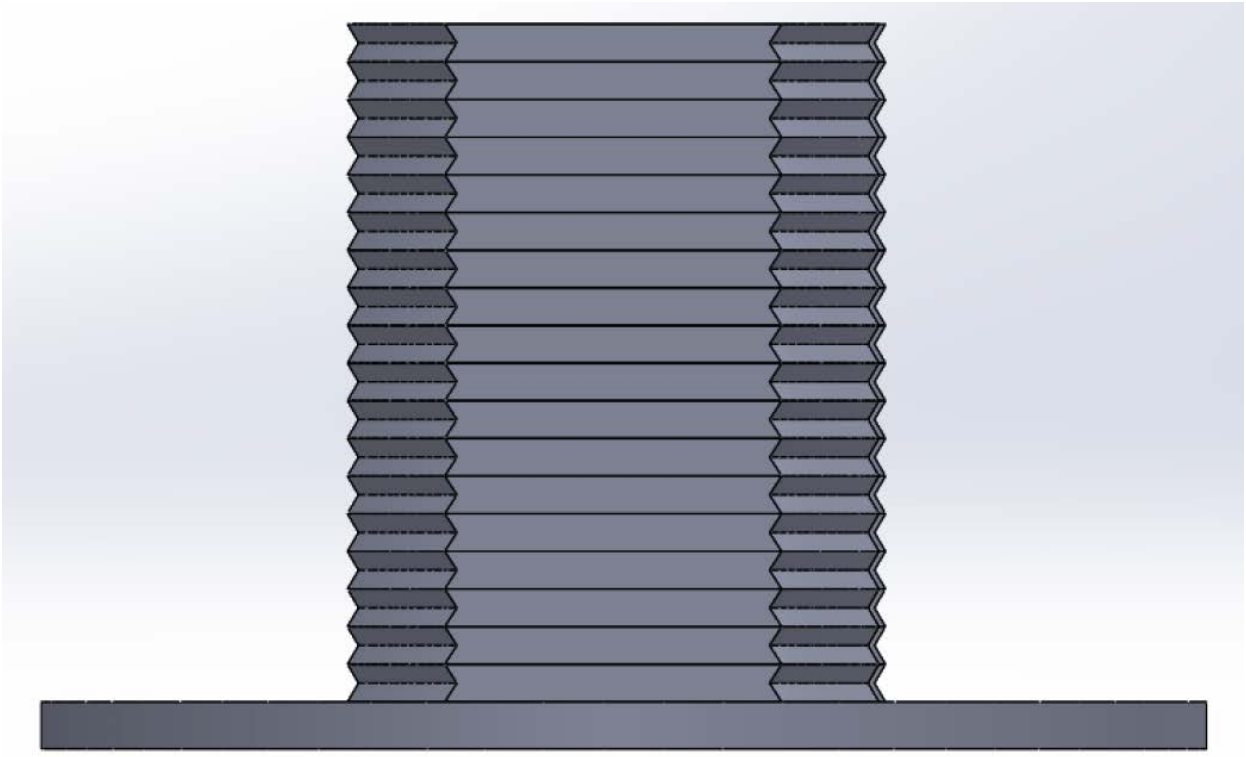
I.1.3 – Third design



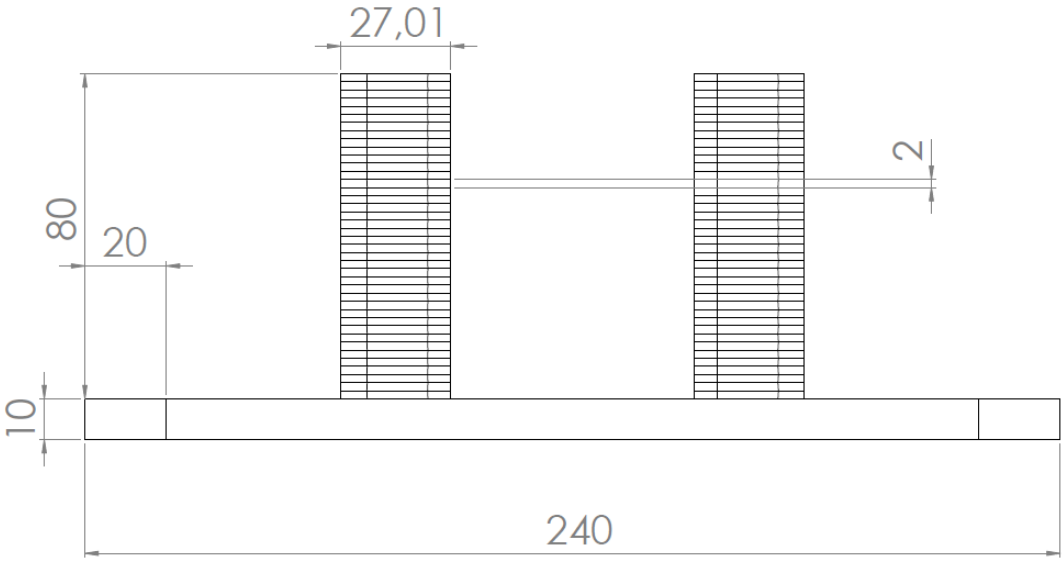


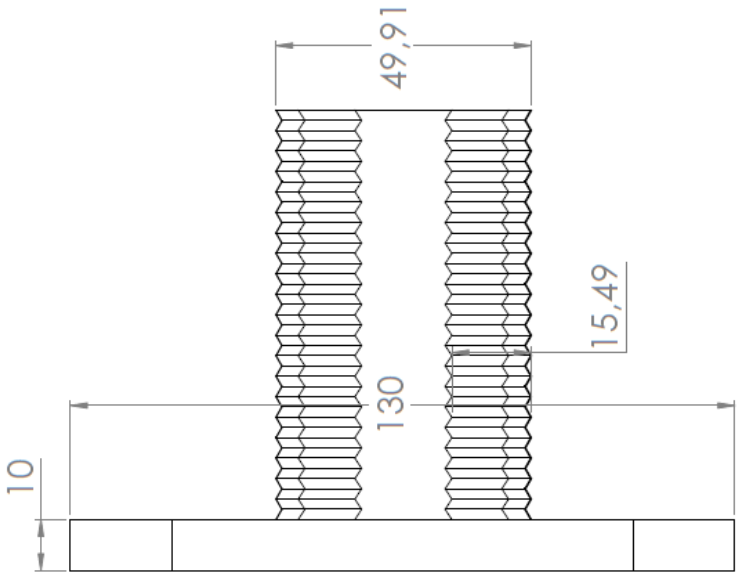


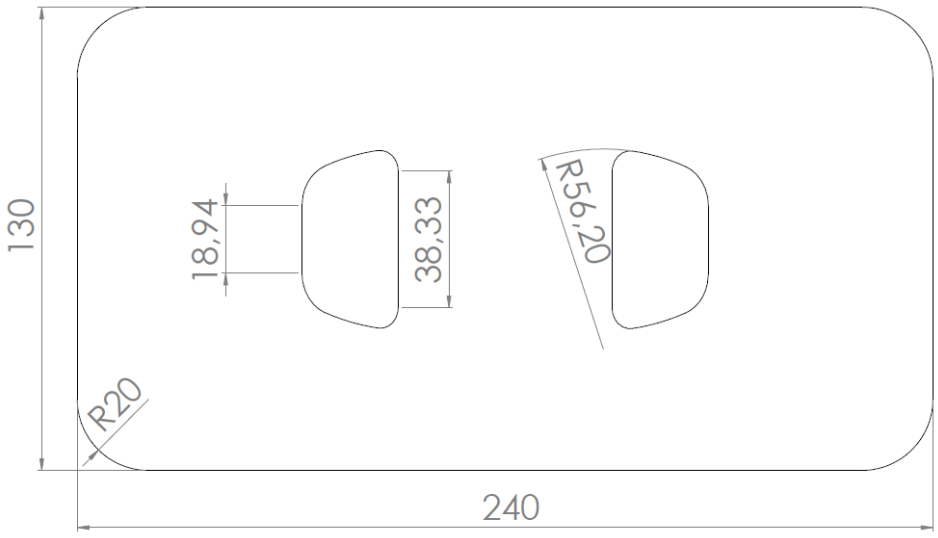


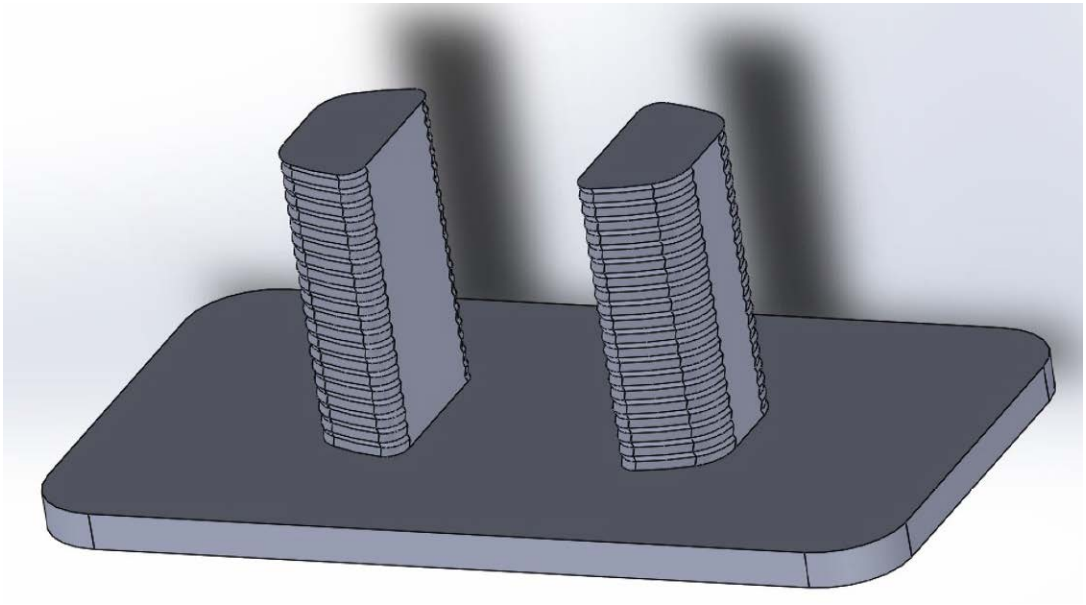


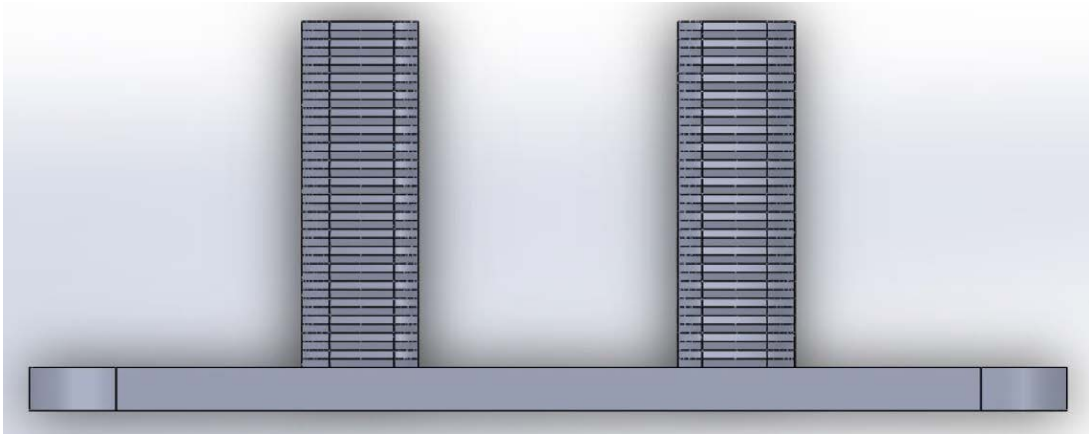
I.1.4 – Fourth design

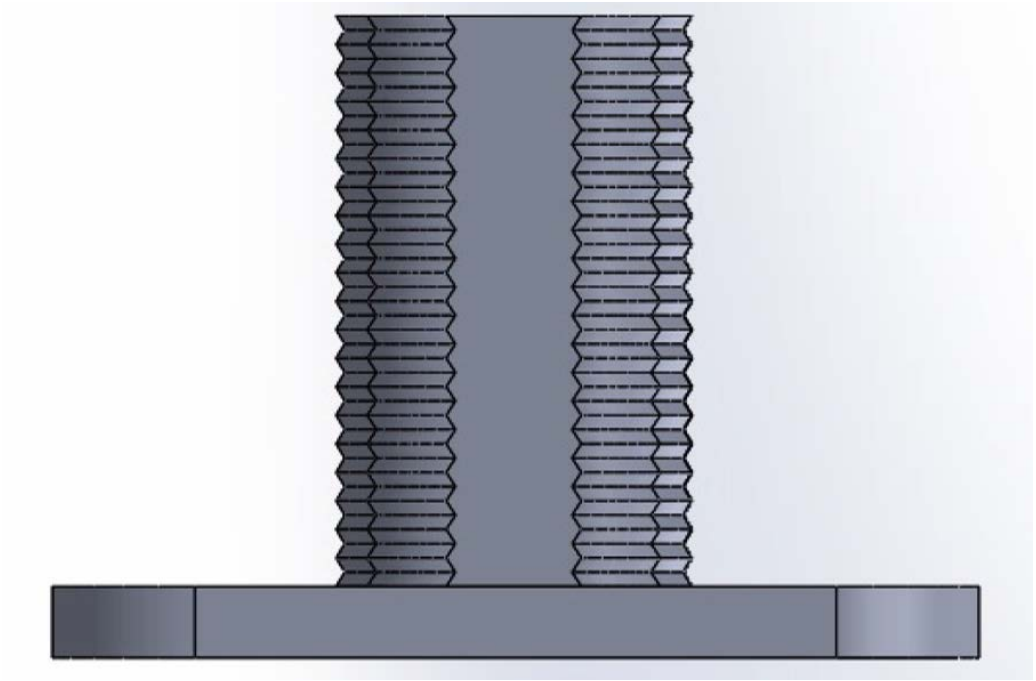






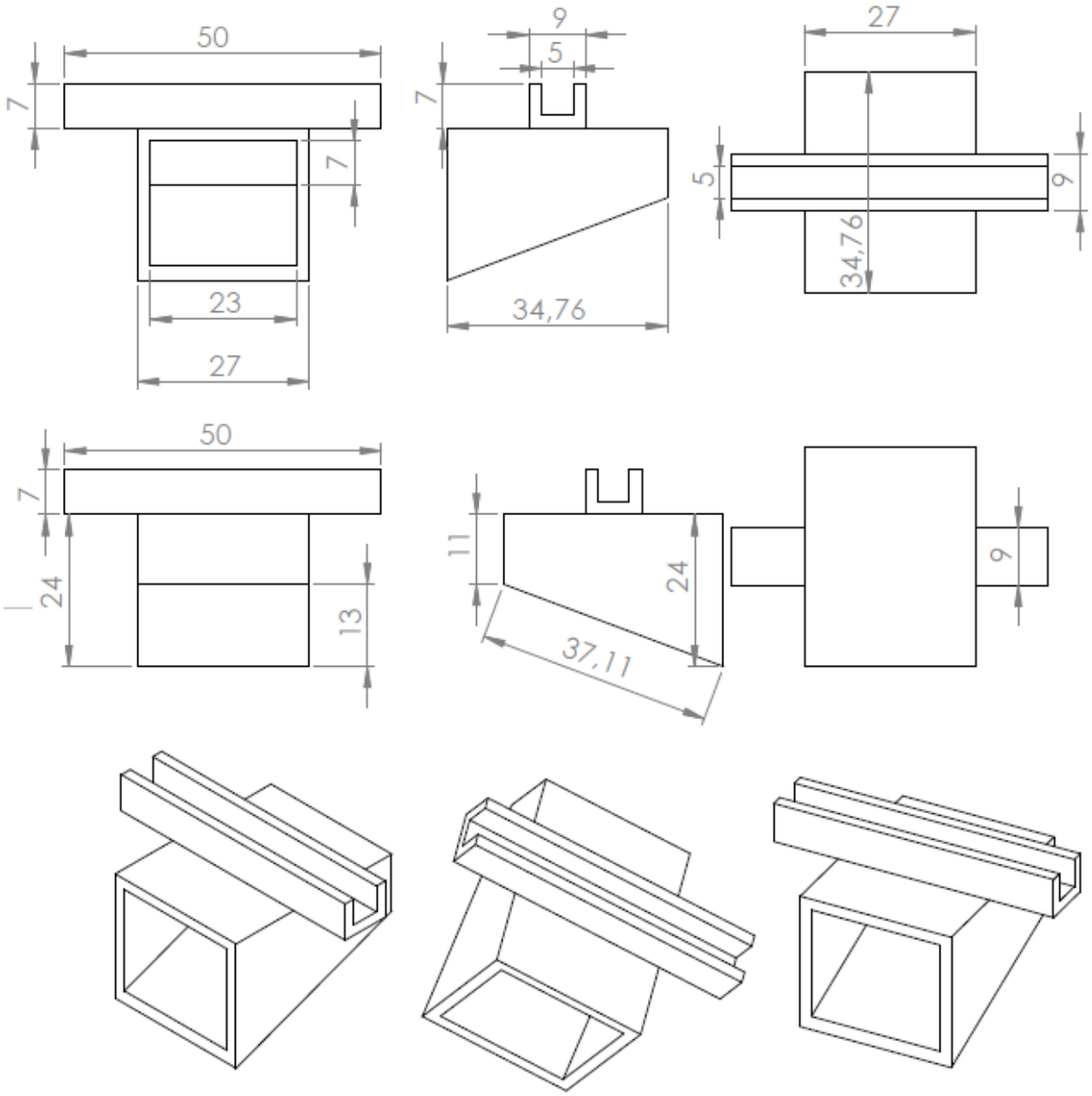


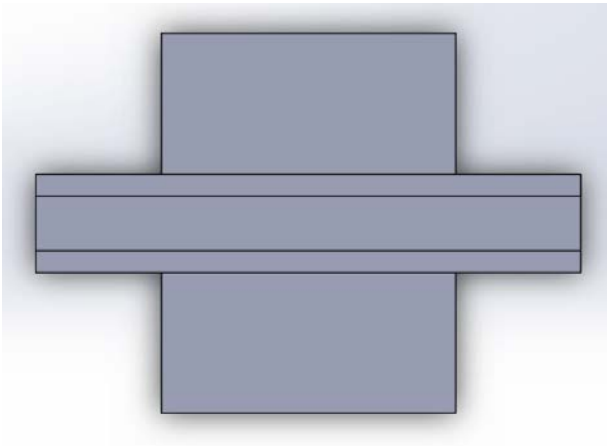


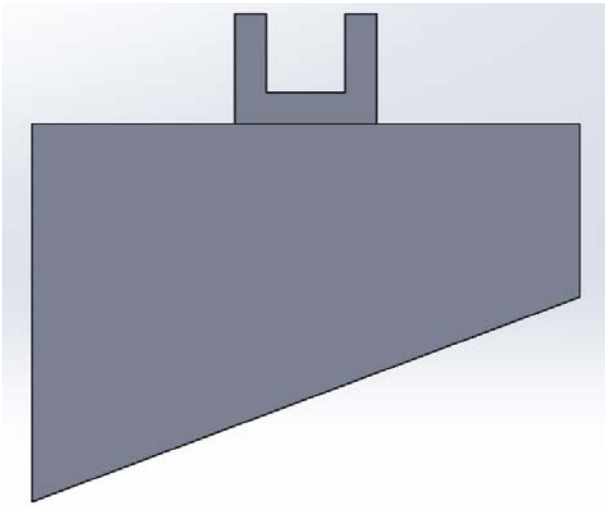


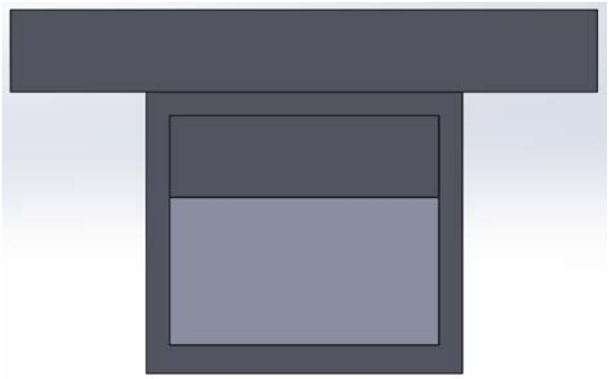
I.2 – Nail design

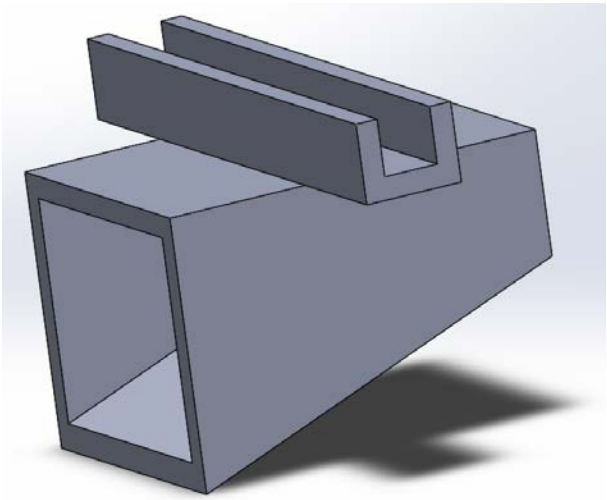
I.2.1 – First design



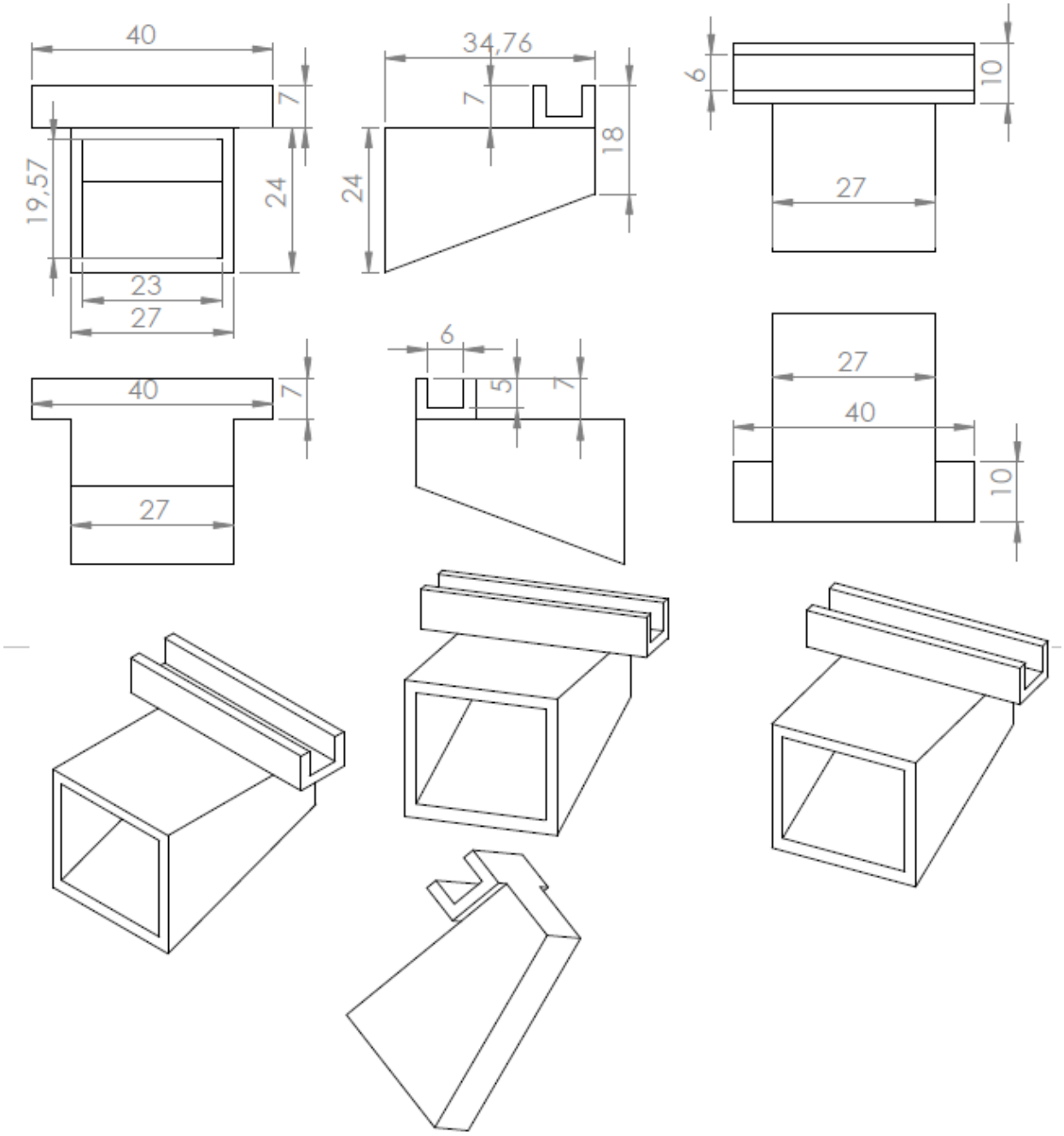


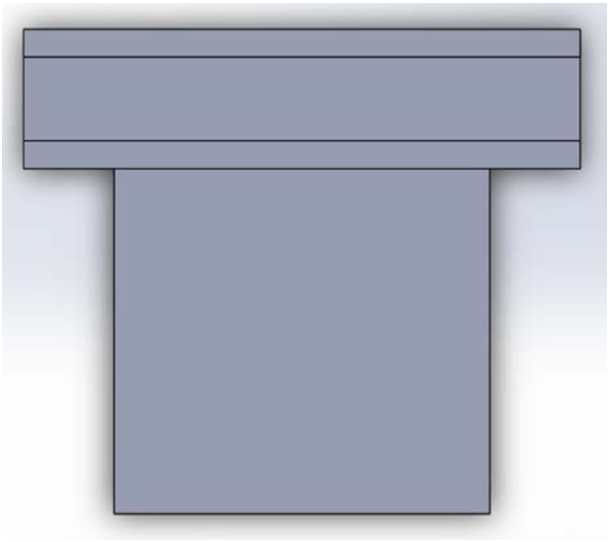


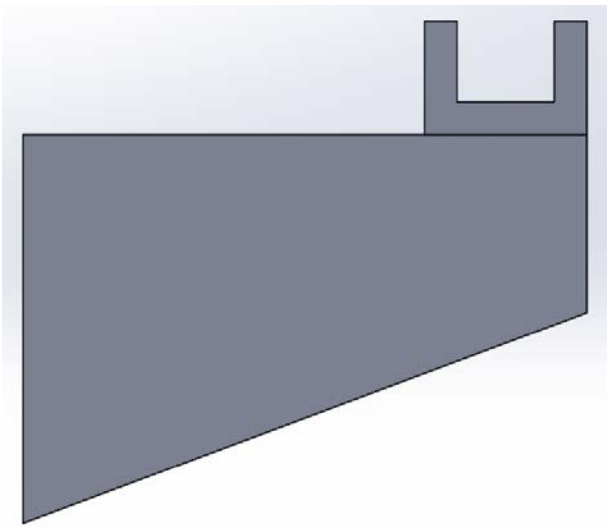


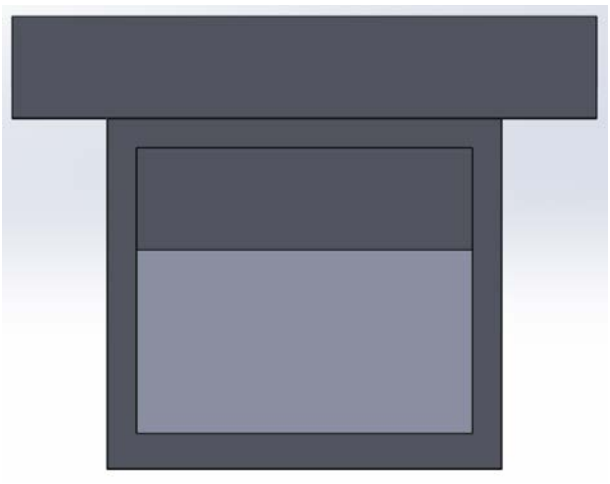


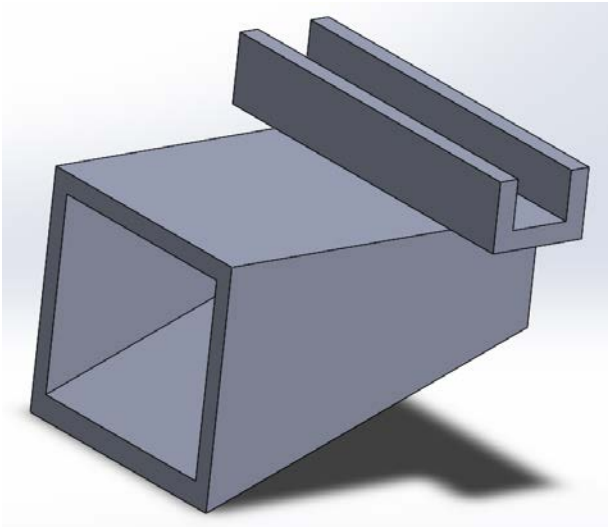
I.2.2 – Second design



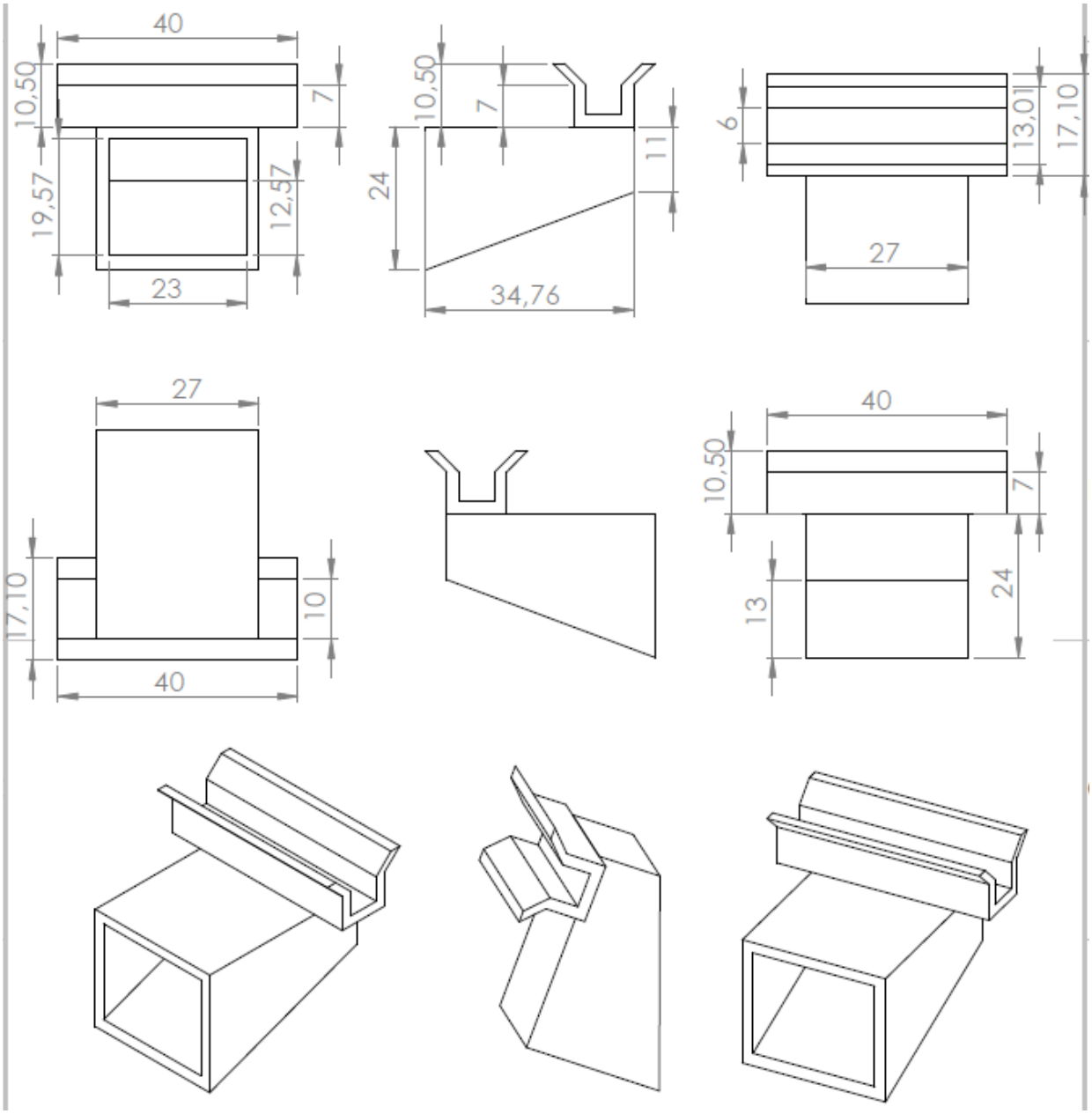


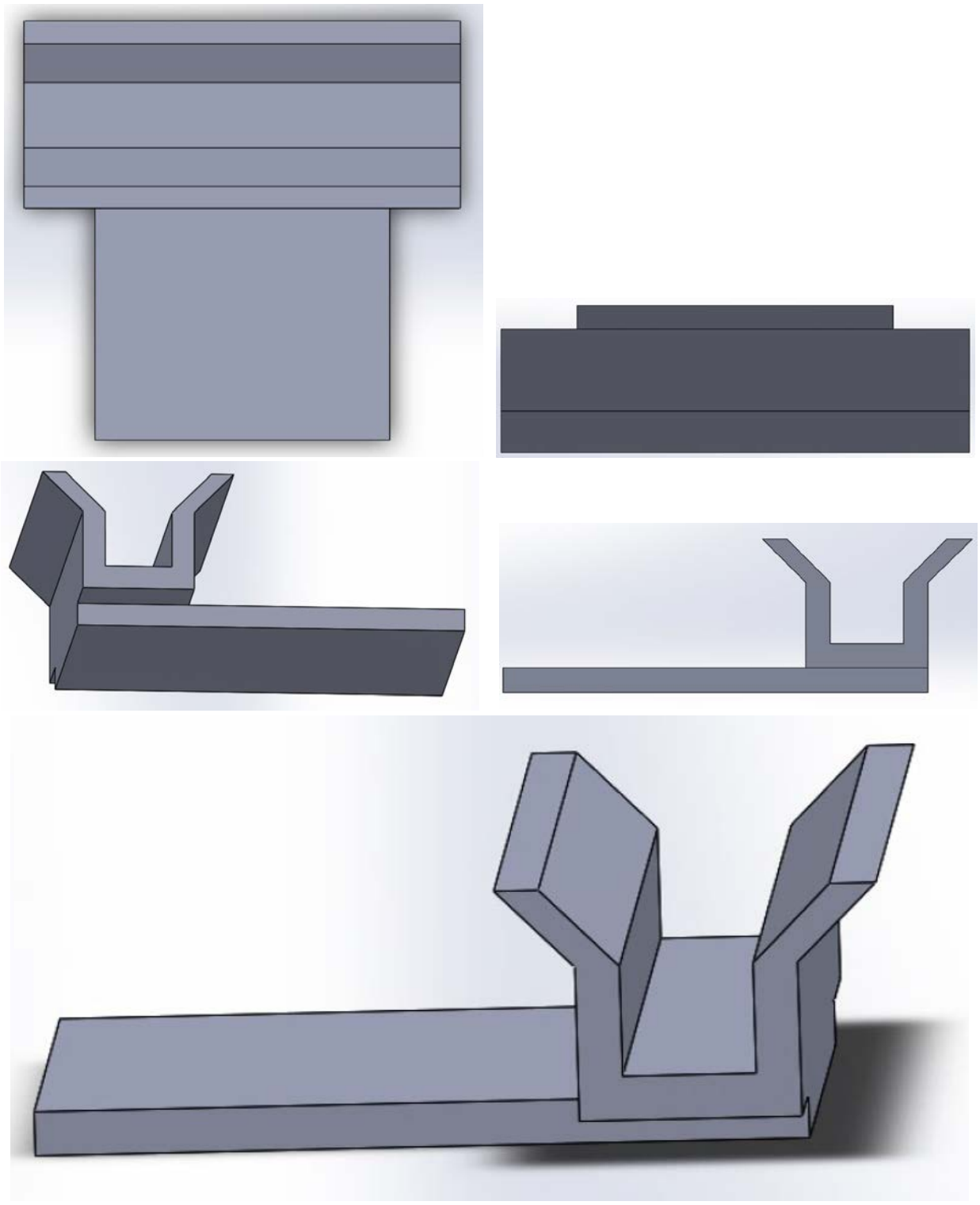






I.2.3 – Third design





I.3 – Material used to print feeder and nail

ABS (Acrylonitrile-Butadiene Styrene) is an oil-based plastic. It is a tough material that can be used to create robust plastic objects for everyday use, for example in cars, electrical equipment or even in the popular Lego bricks.

ABS has a strong resistance to corrosive chemicals and/or physical impacts. It is very easy to machine and has a low melting temperature making it particularly simple to use in injection molding manufacturing processes or 3D printing on an FDM machine. ABS is also relatively inexpensive. ABS plastic is not typically used in high heat situations due to its low melting point. All of these characteristics lead to ABS being used in a large number of applications across a wide range of industries.

ABS is easily machined, sanded, glued and painted. This makes it a great material for prototyping. It is possible also get good cosmetic finishes with ABS and it can also be colored relatively easily, unlike some other plastics.

Technical properties of ABS can be seen in the following table.

Property	Value
Technical Name	Acrylonitrile butadiene styrene (ABS)
Chemical Formula	(C8H8)x· (C4H6)y·(C3H3N)z)
Glass Transition	105 °C (221 °F)
Typical Injection Molding Temperature	204 - 238 °C (400 - 460 °F)
Heat Deflection Temperature (HDT)	98 °C (208 °F) at 0.46 MPa (66 PSI)
UL RTI	60 °C (140 °F)
Tensile Strength	46 MPa (6600 PSI)
Flexural Strength	74 MPa (10800 PSI)
Specific Gravity	1.06
Shrink Rate	0.5-0.7 % (.005-.007 in/in)

Table1: Technical properties of ABS.