Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

ROBOT UR5 GUIADO POR VISIÓN ARTIFICIAL

Autor:

Horno Pérez, Pablo

Mª Isabel Sánchez Bascones

HAMK University of Applied Science

Valladolid, Junio 2018.

TFG REALIZADO EN PROGRAMA DE INTERCAMBIO

TÍTULO:	UR5 ROBOT cooperation with PLC
ALUMNO:	Pablo Horno Pérez.
FECHA:	01/06/2018
CENTRO:	HAMK, Valkeakoski
TUTOR:	Juha Sarkula

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Electrical and Automation Engineering
Valkeakoski

Author Pablo Horno **Year** 2018

Subject UR5 ROBOT ARM VISION GUIDED

Supervisor(s) Juha Sarkula

ABSTRACT

The aim of this project is to use a collaborative robot arm with six articulation points and a wide scope of flexibility, to feed a conveyor belt. These collaborative robot arms are designed to mimic the range of motion of a human arm and can be used with safety in the range of human workspace.

The main problem of this project is that the robot doesn´t know at any moment of the program the sizes, position nor orientation of the pieces it had to take. It will be accomplished with the help of a vision system which will guide the robot to locate and pick up the pieces from the pickup zone, and then place they in the beginning of the conveyor belt, but the conveyor belt and the base of the robot may not be always with the same offset, so the robot have to have a reference point of the conveyor belt to transforms and interpolate all the waypoints of the trajectories used to leave the piece on it.

To accomplish this objective, an URCap will be used, the Wirst camera from Robotiq which need a vision server running in the controller with the licence key in a USB and to keep the collaborative attribute of the robot, a collaborative gripper will be used in which can be set the force, velocity of the opening and closing operations.

As a conclusion the Wrist Camera, can detect really fast any thought object, and also dropped pieces and orientates them in the correct way, so the robot can pick them up.
Communications between PLC and robot works in real-time and the robot tries to keep the PLC always busy.

Keywords Artificial vision, robot UR5, URCap.

Pages 40 pages including appendices 74 pages

CONTENTS

1 INTRODUCTION

1.1 Introduction

A collaborative robot is a type of robot that controls the force and monitories it in real-time to make an emergency stop when the force surpasses the maximum amount of force configured by the user in the safety settings.
The controller of this robot interacts with a camera and artificial vision software to determinate where is a piece and differentiate it from the background. This software must share the relative coordinates with the controller to move the TCP (*Tool Center Point)* and grab it.

1.2 Problem statement

This project is being developed at the same time as a classifier of pieces. The main objective is to feed that classifier, taking the pieces from any part of the working space of the robot to the beginning of the conveyor belt.

The pieces given to the robot are place in the range of the camera, but it doesn´t have to have any reference in the robot program, so the robot will determinate all the waypoints necessaries to grab them and place it in the correct place. In addition, the robot arm will only feed the conveyor belt if it orders it, so it will act as a slave and the classifier will be the master, in the communication protocol.

The robot is also responsible of stacking the pieces that where discarded by the second conveyor belt.

1.3 Objectives

Here are all the objectives that were to be achieved with this thesis:

- Real-time locate pieces in camera field of view.
- Pick and place pieces to industrial process.
- Complex path to feed a conveyor belt with space limitations.
- Fully collaborative robotic system.
- Communication between robotic arm and Beckhoff PLC with digitals inputs and outputs.
- To learn about URScript programming and force control.
- Seek with force.

1.4 **Background of the project**

The main concepts to understand this thesis are the basics of robotics, understanding relative path, waypoints, trajectories, kinetics and TCP's.
The robot used is a six-axis, the rotation of each axis results in a linear, circular or elliptic path. This six-axes determinate the six degrees of freedom that the robot has, each axis has its own name in the robot, starting with the base and ending in the wrist.

Is necessary to know the movements that the TCP will follow when the controller set it to the six-axes of the robot.

1. **MoveJ** which is used for joint movement. The controller doesn´t know the path which the TCP will follow, but it's the fastest way to move the TCP from point A to point B.
2. **MoveL** this way of movement makes the actual TCP follows a rectilinear path from point A to B, interpolating the angle and the speed of each joint the controller makes the linear path.
3. **MoveP** this way of movement is the one which the robot will make circumferences, if needed.

This project involucrate a low-level communication between the UR controller and a PLC or any hardware capable of managing a 24-volt digital signal.

1.5 **Project roadmap**

This thesis has three main chapters as follows:

- Chapter II. Holds all about the differences between the collaborative robot's vendors, their characteristic, payload, reach, repeatability, weigh of the hole robot and some other specifications.
- Chapter III talks about the robot arm used, previous mounting process and installations parameters.
- Chapter IV hols all about the gripper, installation process, dimensions and specifications, electrical setup and how it grabs the pieces.
- Chapter V. Talks about the Vision system, which camera is used, its specifications, how to calibrate it and parameters.
- Chapter VI. Talks about all the programming done, explained step by step, from detection object to stack them with force.

2 COLLABORATIVE ROBOTS

2.1 Definition

The collaborative robots (*hereinafter co-bots*) are a good choice for small and medium industry, they are easy to program, free of complicated security systems that allows employees to work side by side with them. They are intended to physically interact with humans in a shared workspace.
Some of the main characteristic of the co-bots are the followings:

Easy to program: The co-bots are systems that can be programmed in an easy way, by non-qualified personnel and with no knowledge of programming. In lot of cases, we talk about teaching the robot rather than programming it, all the waypoints can be added by moving the robot manually with the *free mode*. (Garcia, D. 2014)

Security working side by side: As mentioned before, co-bots have the great advantage of working in a shared workspace with humans without the need of any security system, such as jails, motion detectors or photoelectric sensors acting as a watchdog. The actual normative reference to robotic systems are hold by the ISO 10218-1, ISO 10218-2. The incorporation of these robots to the industry have made new legislations like ISO/TS 15066 in which are defined the requirements of security of these co-bots. (Garcia, D. 2014)

2.2 **Types and brands**

Here are some of the most used co-bots now a day in the industry, from various vendors, categories, payload, complex and purposes.

2.2.1 ABB - YuMi

YuMi harnesses the enormous potential of human robot collaboration in small parts assembly. YuMi offers manufacturers a transformational new solution, the first dual arm robot purpose-built for the small parts space: inherently safe, extremely accurate. Each magnesium arm flexes on seven axes to mimic human-like movements with spatial efficiency. The robot was specifically designed to meet the flexible and agile production needs required by the consumer electronics industry. (ABB Robotics, n.d.)
(Figure 1)

Figure 1. ABB YuMi (*ABB Robotics*)

SPECIFICATIONS:

- Name: YuMi.
- Model ID**:** IRB 14000
- Payload: 500g
- Repeatability: ±0.02mm
- Reach: 555mm
- Weight: 38kg
- N° Axes: 7 per arm.

2.2.2 KUKA – LBR iiwa

The LBR iiwa is the world's first series-produced sensitive, and therefore HRC-compatible, robot. LBR stands for "Leichtbauroboter" (German for lightweight robot), iiwa for "intelligent industrial work assistant". This signals the beginning of a new era in industrial, sensitive robotics – and lays the foundations for innovative and sustainable production processes. The collaborative and sensitive LBR iiwa robot is available in two versions with payload capacities of 7 and 14 kilograms. (Kuka Robotics, n.d.)
It is shown in figure 2.

Figure 2. KUKA iiwa (*KUKA*)

SPECIFICATIONS:

- Name: LBR iiwa
- Payload: 7kg / 14kg
- Repeatability: ±0.1mm / ±0.15mm
- Reach: 800mm / 820mm
- Weight: 23.9kg / 29.9kg
- N° Axes: 7

2.2.3 MABI – SPEEDY 10

The "flexible manufacturing" often publicised today is the rationale for this development and it is based on a lightweight design with excellent damping characteristics. This 6-axis kinematics system with standard wrist is a lightweight in its class; nonetheless it offers high positioning precision for high-speed applications thanks to a high-resolution absolute feedback encoder. The SPEEDY 10 by MABI Robotic is an extremely flexible six-axis robot. Full engineering and installation at the Veltheim factory in Switzerland. (MABI, n.d.)
It is shown in figure 3.

Figure 3. MABI – SPEEDY 10 (*MABI Robotics*)

SPECIFICATIONS:

- Name**:** SPEEDY 10
- Payload: 10kg
- Repeatability: ±0.02mm
- Reach**:** 1384.5mm
- Weight: 28kg
- N° Axes: 6

2.2.4 Rethink Robotics

Rethink Robotics offered us the Baxter, an affordable and safe co-bot to operate around people, easily to integrate in productions environments. Baxter can handle a wide range of repetitive production tasks, including packaging, lifting material, landing, machine tools. Line works can train Baxter in minutes, with no software, robotics or engineering experience required. (Rethink Robotics, n.d.)
Baxter has an LCD screen, a 360° sonar and 3 vision cameras, force detection, tow arms with 7 degrees of freedom each. It also offers a series of accessories such as vacuum cups, parallel grippers as shown in figure 4.

Figure 4. Baxter cobot. (*Rethink Robotics*)

SPECIFICATIONS:

- Name: Baxter
- Payload: 2.2kg per arm.
- Repeatability: ±0.02mm
- Reach: 1210mm per arm.
- Weight: 74.8kg
- N° Axes: 7 per arm.

2.2.5 Rollomatic

NEXTAGE is the cobot of Rollomatic, a two-arm robot with a human-like geometry designed to perform tedious work. The cobot has four video cameras, two in the head and one in each arm.
Users can control and teach NEXTAGE what to do with a graphical user interface that can conveniently be handled by people without programming skills. (Rollomatic Robots, n.d) It is shown in figure 5.

Figure 5. NEXTAGE (*Rollomatic*)

SPECIFICATIONS

- Name: NEXTAGE
- Payload: 3kg / 5kg / 10kg
- Repeatability: ±0.03mm
- Reach: 500mm / 850mm / 1300mm
- Weight: 130kg
- N° Axes: 6

2.2.6 Universal Robots

Their main robots are the UR3, UR5 and UR10. They are easily integrated into existing production environments, with six articulation points and a wide scope of flexibility, these co-bots arks are designed to mimic the range of motion of a human arm. (Universal Robots, n.d) The family of UR robot is shown in figure 6.

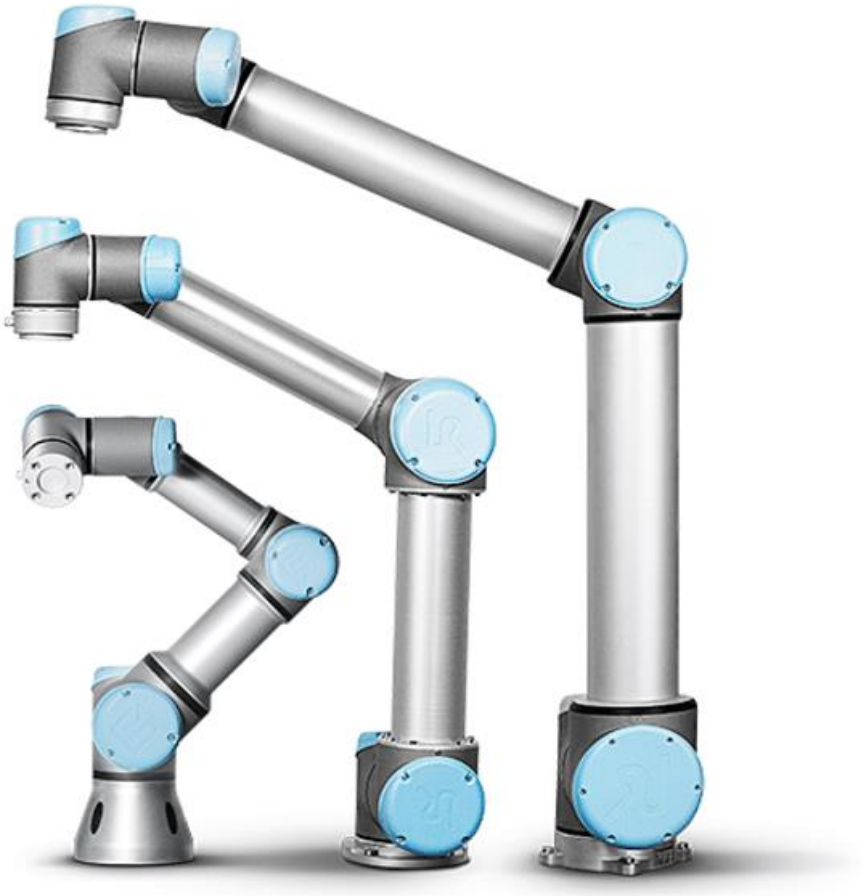


Figure 6. UR3 – UR5 – UR10. (*Universal Robots*)

SPECIFICATIONS:

- Name: UR 3 / 5 / 10
- Payload: 3kg / 5kg / 10kg
- Repeatability: ±0.1mm
- Reach: 500mm / 850mm / 1300mm
- Max TCP Velocity: 3m/s
- Max TCP Acceleration: 150m/s^2
- Weight: 11kg / 18.4kg / 28.9kg
- N° Axes: 6

3 UR5 ROBOT ARM

The UR 5 is the robot that is going to be used in this thesis to accomplish the objectives mentioned above, because of its repeatability, compatibility with URCap and the reach and payload are the best ones for this thesis.
The UR5 was boult to conquer task that still needs great precision and reliability.

Is the perfect choice for low-weight collaborative processes such as picking, placing or testing. Furthermore, the UR5 is very easy to set-up and program and offers one of the quickest payback times in the industry.

The UR5, like the other two robots in the UR family, follows the schema showing in figure 7. It has six axes as mentioned, three wrist, elbow shoulder and base.

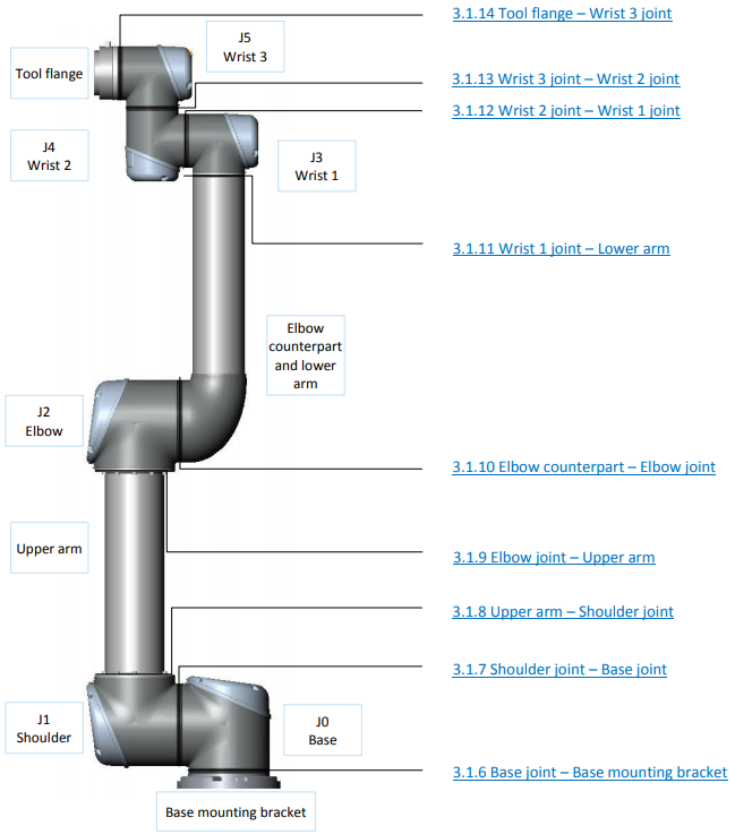


Figure 7. Joints and arms of UR5 (*Universal, R. (2018.), UR5 User manual*)

3.1 **Technical details**

Some of the technical details of the robot UR5, shown in figure 8, and its main components, such as the control box and the teach pendant, which are used in this thesis, are the followings

3.1.1 UR5

Figure 8. UR5 robot arm (*Universal, R. (2018.), UR5 User manual*)

PERFORMANCE

- Repeatability: ±0.1mm
- Ambient temperature range: 0-50°C
- Power consumption: Lowest 90W / Typical 150W / Maximum 325W

SPECIFICATIONS

- Payload: 5 kg
- Reach 850 mm without tool
- Degrees of freedom: 6 rotating joints

MOVEMENT

Table 1. Movements speeds and working range.

Axis movement robot arm	Working range	Maximum speed
Base	±360°	±180°/seg
Shoulder	±360°	±180°/seg
Elbow	±360°	±180°/seg
Wrist 1	±360°	±180°/seg
Wrist 2	±360°	±180°/seg
Wrist 3	±360°	±180°/seg

As we can see in the table 1, all the joints have the same working range and maximum speed which will make the reach up to 850mm and a TCP velocity of 17m/seg.

FEATURES

- IP classification: IP54 (Dust/water resistance)
- ISO Class cleanroom: 5
- Noise: 72dB
- I/O ports
    - Digital in: 2
    - Digital out: 2
    - Analog in: 2
    - Analog out: 0
- I/O power supply in tool: 12/24 V 600mA in tool.

PHYSICAL

- Footprint: ø149mm
- Materials: Aluminium, PP plastics
- Tool connector type: M8
- Cable length robot arm: 6m
- Weight with cable 18,4kg

3.1.2 Control box

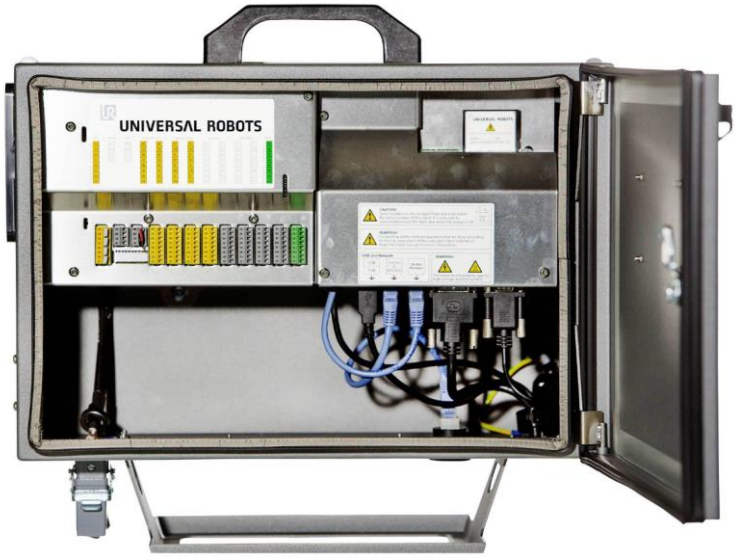


Figure 9. UR control box. (*Universal, R. (2018.), UR5 User manual*)

The control box, figure 9, is the power supply for the motors of the robot, also is the brain of the hole robot, the one which makes the links and connections between the inputs and outputs signals with the software. Provides the view of the teach pendant and have the safety cards and triggers on it.
Also, it controls the version, the new URCap licences. It runs on Debian a distribution of Linux, has a built-in FTP server, which can be accessed through the Ethernet port, as well as a socket listener to attend commands in real-time. (*Universal, R. (2018.), UR5 User manual*)

FEATURES

- IP CLASSIFICATION: IP20
- ISO class cleanroom: 6
- Noise: <65dB
- I/O ports:
    - Digital in: 16
    - Digital out: 16
    - Analog in: 2
    - Analog out: 2
- I/O power supply 24V@2A
- Communications:
    - TCP/IP 100 mb/s
    - Modbus TCP
    - Profinet
    - EthernetIP
- Power source: 100 ~ 240VAC @ 50 ~ 60Hz
- Ambient temperature range: 0~50°C

3.1.3 Teach pendant

Figure 10. Teach pendant (*Universal, R. (2018.), UR5 User manual*)

The teach pendant is where all the robot can be programmed and control with a touch screen. It also has three physical buttons; the emergency stop, the on/off button and the free mode button. The actual teach pendant is shown in figure 10. (*Universal, R. (2018.), UR5 User manual*)

- **Emergency stop:** When pressed this button will stop the program and robot movements (if was moving) and it will not run again until is released again.

- **On/off button:** To start and shutdown the robot.

- **Free mode button:** With this button teaching new positions is fast and easy, when pressed the robot will move in the direction of the applicated force. Notice that an incorrect set of the actual payload will cause the robot to move when no force applicated to it.

FEATURES
- IP classification: IP20

PHYSICAL
- Materials: Aluminium and PP.
- Weight: 1.5kg
- Cable length: 4.5m

3.2 **Installation**

3.2.1 Base

The robot base is set with a 45° in the Y axis and is mounted in a moving platform from Easy Robotics shown in figure 11, that is why we will need to define where the conveyor belt is from the robot.
The base weight is around 45kg and the robot itself 18.4Kg so the full setup weight is about 64 kg that are easy to move with the easy robotics mounting.

Figure 11. Moving robot base. (*Easy Robotics*, n.d)

The benefits of the UR5 is that each joint know its angle and the motors can be moved and no calibration will need, as long as the encoders still on. We can see an imagen of the interior of a joint in figure 12, the blue circle mark where is the break. There we can see the encoder, motor and gear system to transmit the movements of the motor to the shoulder.

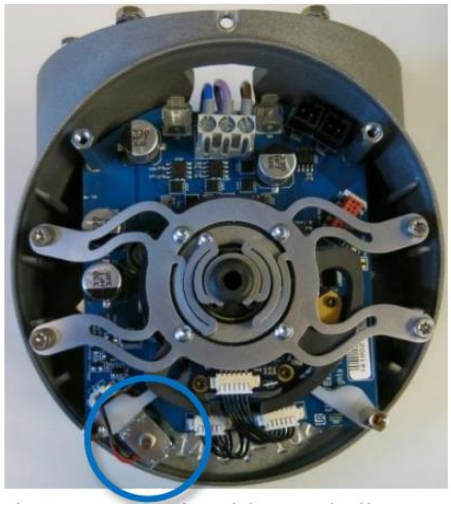


Figure 12. Base joint. (*Universal, R. (2018.), UR5 Service manual*)

3.2.2 Feature coordinates

A feature is a coordinate system consisting in 3 axes X, Y and Z. The robot has two predefined feature, the base feature located with origin in the center of robot base, and the tool feature located in the robot center current TCP. As default, a waypoint is saved as a coordinate position relative to the base feature.
The feature coordinates can be a point, a line or even a hole plane. For this installation up to six feature coordinates are used, and those are the followings:

- **MiddleScrew**: It marks the middle of the working table, it is used as way point in large trajectories between the pickup places drop place.

- **refConveyorBelt:** This point is the one used to calculate the relative position of all the paths and waypoints used in the conveyor belt pickup and drop. A socked has been designed to teach this point, because it is a point that need to be teach every time the relative position between the robot workspace changes with the conveyor belt. The 3D model of this socket is shown in figure 13.

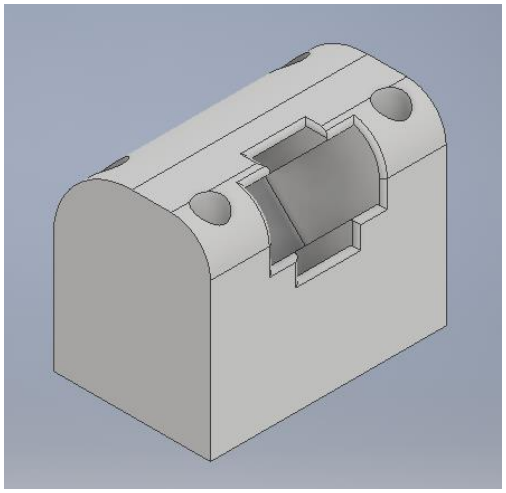


Figure 13. Socket of reference of conveyor belt

- **Home:** This reference point holds the pose to the home position of the robot, where the camera calibration has been made, where the object detections are made and the return point of most of the trajectories.

- **Sleep:** This pose holds a configuration where the robot stays in a position where it occupies a little space of the working place.

- **Table:** This is the only feature plane used in this project. It represents the actual real table on which the robot is. Below this plane no pose can be reached because first the tool will collide with the table.

3.2.3 Communications

The communications to the PLC are made with four digital signals, two for request to pick a piece from the two different entry point of the conveyor belt system, as shown in figure 16, and two more to tell the conveyor belt that the piece has been placed or taken.

INPUTS

The digitals input used are the followings:

- **Digital input 1 (*I[1]*),** determinates when the robot has to place a piece in the feeding conveyor belt (the left one in figure 16)
- **Digital input 2 (*I[2]*),** tells the robot when a piece is waiting for pick up at the exit of the conveyor belt.

OUTPUTS

The digitals output used of the robot are the followings:

- **Digital output 1 (O[1]),** when it rise up the conveyor belt system can start with the sorting process, it will be up during three seconds.
- **Digital output 2 (O[2]),** this output is used to speak to the PLC of the conveyor belt and tell that the piece that it was asking to be removed has been already been removed, so it can continues with it next process.

3.3 **Workspace**

The workspace of the robot is represented on the figure 14, on the 45° base and removing the all the points who are under the plane define by the feature plane *Table* which is shown in figure 15.
The figure 14 shows all the points that can be reached by the robot but not all of them can be reached with the same configuration, so some of the original path tough to the robot had to be edited because of some joint limit or the camera cable which is shown along the arm of the robot in figure 15.

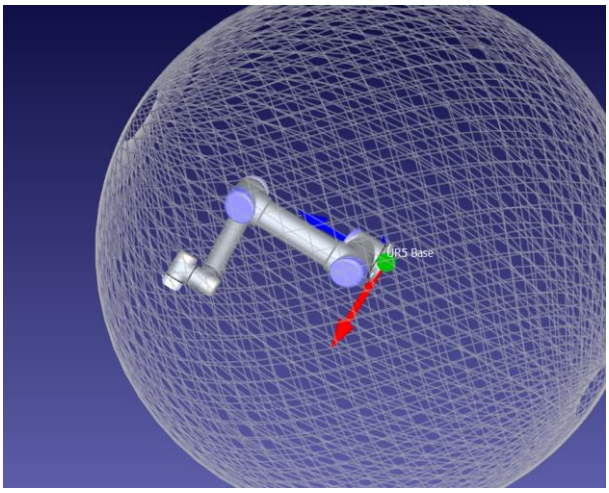


Figure 14. Robot base and workspace (*RoboDK*, n.d.)

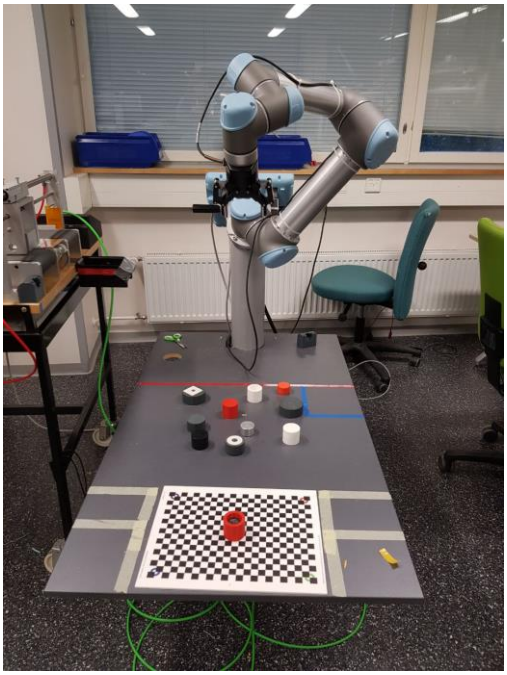


Figure 15. Real workspace

As we can see in figure 15, there are a red line which indicates the bottom limit of the range of the vision system, any piece closer to the robot behind that line won't be detected.
Also, we can see a blue square in the right side of the figure, there is the "*safe zone"* where no other piece can be at the run time of the program, it's a little space of the working table where the robot will perform some operation with the taken piece to place it correctly in the conveyor belt, with the right grab angle to be able to enter in between the limitations of the conveyor belt entry. Furthermore, it will change the grip angle from 90° to 45°.

In figure 15 we can also see how the cables needed on the tool travel along the robot arm fixed with white wrist.

3.4 **Conveyor belt**

The conveyor belt can be shown in figure 16. All about the conveyor belt is described in another thesis called "*Classification machine (2018)"* wrote by Marta Vidal.

In the left lower corner of the conveyor belt table we can see the real 3d printed socket for setting the feature point *refConveyorBelt*. Moreover, we can see the two entry points, as well, there is where the UR5 have to pick and place the pieces.

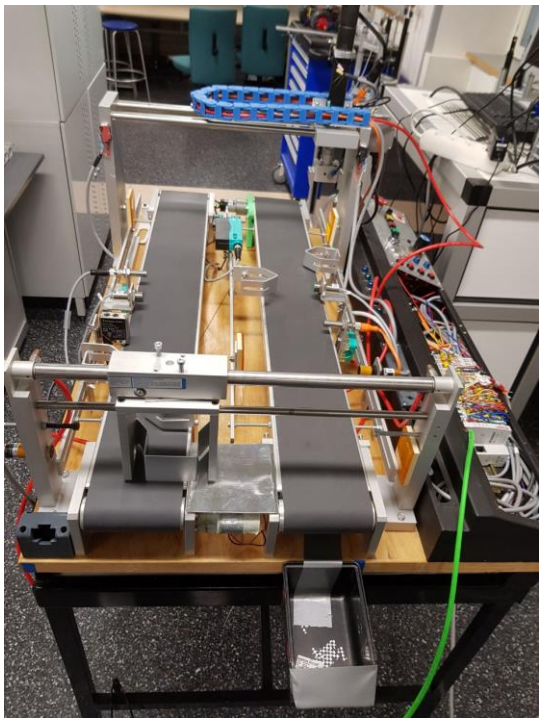


Figure 16. Conveyor belt system

In other to do so, first the conveyor selector will have to free space for the gripper to operate, that is why an input is used to know when the robot, physically, has room to reproduce the movements needs to place the requested piece.

The robot itself will be like a listener, waiting for the conveyor signals, trying to give priority to the place request as long as the robot has room and time to do so. The main objective to enhance the system and reduce queues is to leave the piece taken at the end of the conveyor belt in a second safe zone, then attend the feeding request of the conveyor belt, and when this one is busy, then start with the stack process of the piece.

4 GRIPPER

The gripper used in this project in collaborative with the UR5 must have control over the speed and force of the gripper since we want to keep the system collaborative and human work friendly. The manufactured chosen to accomplish these specifications is Robotiq, it has some adaptive grippers with different sizes and morphology.

4.1 RobotiQ

Robotiq's tools and know-how simplify cobots applications. Works with a global network of connected robot experts supporting their local manufactures. (RobotiQ, n.d) RobotiQ was chosen because its grippers are easy to use, versatile and is compatible with Universal Robots.

4.2 The gripper 2F-85

The chosen gripper is the 2-Finger Gripper 85. It has two articulated fingers that each have two joints (two phalanxes per finger), as shown in Figure 17 and some of their vantages are the followings:

- Built for Universal Robots
- Free programming software
- 3 wide-stroke options
- Multiple grip modes
- Built-in position feedback
- Fast and strong
- Precise and durable

Figure 17. Two-finger gripper (*RobotiQ 2F-85, Instruction manual*)

The Gripper can engage up to five points of contact with an object (two on each of the phalanges plus the palm). The fingers are under-actuated, meaning they have fewer motors than the total number of joints. This configuration allows the fingers to automatically adapt to the shape of the object they grip, and it also simplifies the control of the Gripper.

4.2.1 Dimensions

DIMENSIONS WHEN OPENED:

The figure 18, shows all the dimensions of the gripper that will allow us to know de TCP of the robot and some of the most important measures we will need, like the maximum aperture of the gripper, 85mm, and the length when closed which we will see in figure 19. (*RobotiQ 2F-85, Instruction manual*)

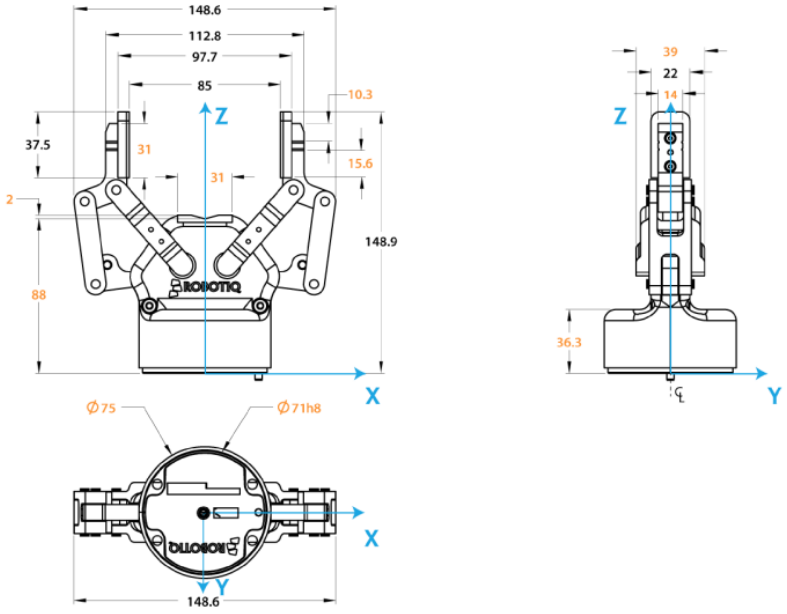


Figure 18. 2F-85 dimensions. (*RobotiQ 2F-85, Instruction manual*)

Dimensions when closed

The most important measure we will need, in figure 19, is the height of the gripper when closed because is the TCP point we will set in the settings of the installation.

Also, is needed the width of the gripper when closed to introduce it in the reference socket of the conveyor belt. (*RobotiQ 2F-85, Instruction manual*)

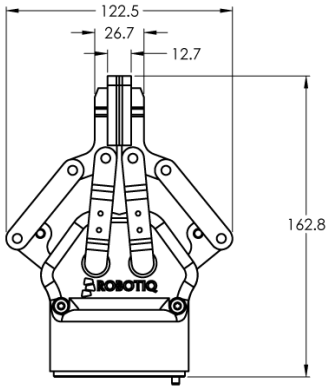


Figure 19. 2F-85 dimensions closed. (*RobotiQ 2F-85, Instruction manual*)

4.2.2 Object picking

The 2F-85 has a single actuator for opening and closing operations. The fingers automatically adapt to the shape of the object manipulated. Fingers will adopt either a parallel grip or encompassing grip as shown in figure 20. (RobotiQ, 2016)

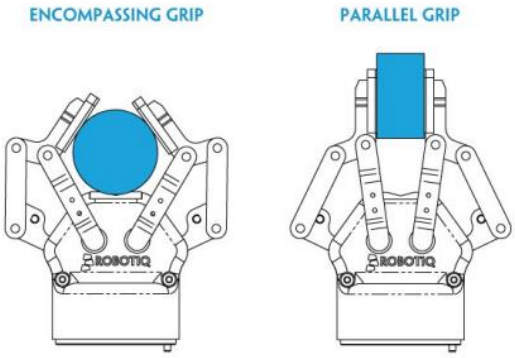


Figure 20. 2F-85 object picking (*RobotiQ 2F-85, Instruction manual*)

It is important to note that a fingertip grip can only be performed when the fingers touch the object with the upper section of the distal phalanxes first. Inversely, for an encompassing grip, the fingers must touch the object with the proximal or the lower section of the distal phalanxes first. Also, to ensure stability, the object should be held against the Gripper palm while performing an encompassing grip. (*RobotiQ 2F-85, Instruction manual*)

INSTALLING FINGERS ON THE 2F-85

The first step in the mechanical installation of the gripper is to mount the fingers. This model is the 85mm opening and the tools needed was the followings:

- 2mm Allen key
- 5~6mm snap ring pliers
- Medium strength Loctite (*248*)

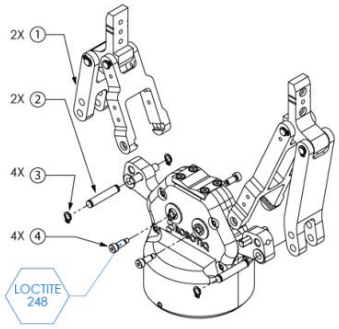


Figure 21. 2F-85 Finger installation. (*RobotiQ 2F-85, Instruction manual*)

INSTALLING THE FINGERTIPS ON THE 2F-85

We can choose in a wide option of fingertips for the 2F-85. To mount them we must follow the figure 22. Some of the tools needed are the followings:

- 2mm Allen key.
- Medium strength Loctite (*222*)

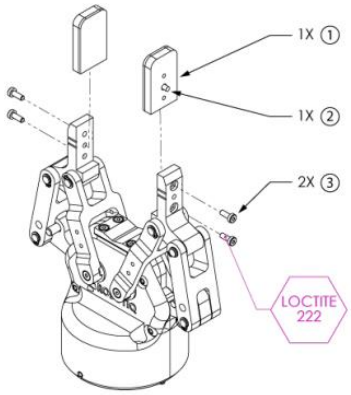


Figure 22. 2F-85 Fingertips installation
(*RobotiQ 2F-85, Instruction manual*)

Table 2. Screws of 2F-85

	AGC-TIP-XXX-002
1	85mm fingertip
2	M3X10 mm indexing pings
3	M3x10mm low head cap scresw

INSTALLING THE GRIPPER TO THE ROBOT WRIST.

The gripper must have a coupling to attach itself to the robot, but in these case, we will use de camera as a coupling attach, that is explained in the chapter V, so the camera will be between the robot wrist and the gripper mount. Mounting the gripper to the wrist only needs LOCTITE 248, four screws each one with their tooth lock washer, as sown in figure 23.
(*RobotiQ 2F-85, Instruction manual*)

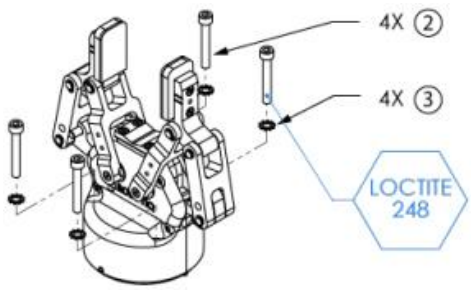


Figure 23. Mounting the gripper to the robot. (*RobotiQ 2F-85, Instruction manual*)

4.2.3 Electrical Setup

Power and communication are established with the 2-Finger Adaptive Robot Gripper via a single Device Cable. The Device Cable provides a 24V power supply to the Gripper and enables serial RS-485 communication to the robot controller as sown in figure 24.

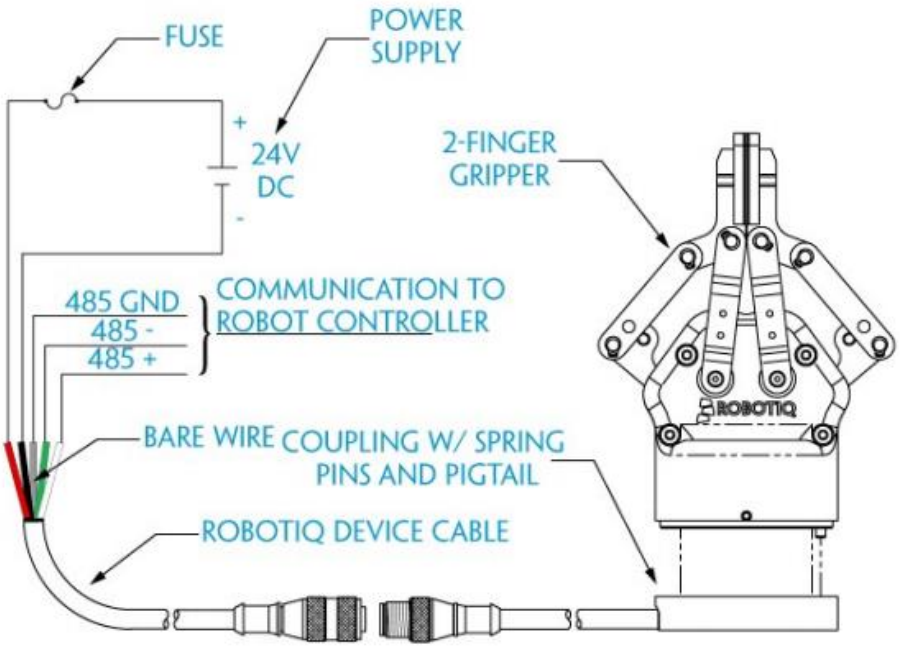


Figure 24. Electric schema. (*RobotiQ 2F-85, Instruction manual*)

Gripper grounding is optional and is done via the robot ground. The coupling indexing pin (dowel) is the ground connector. Gripper coupling, chassis and proximal phalanx are linked as illustrated in Figure 25. They link through the coupling indexing pin to the robot ground. Proximal bars, distal phalanx, fingertip base and fingertips are isolated. (*RobotiQ 2F-85, Instruction manual*)

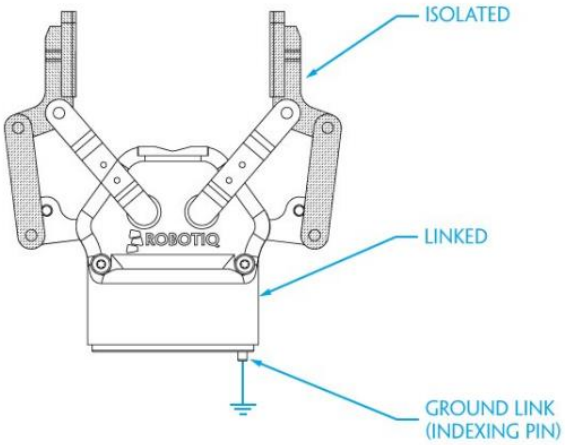


Figure 25. 2F-82 isolation and grounding. (*RobotiQ 2F-85, Instruction manual*)

In the figure 26 is shown how the cables are connected in the controller box. The red cable goes to the 24v output and the black to the ground. The white green and grey goes to the RS485 jumper converter, in that order, following the figure 26.

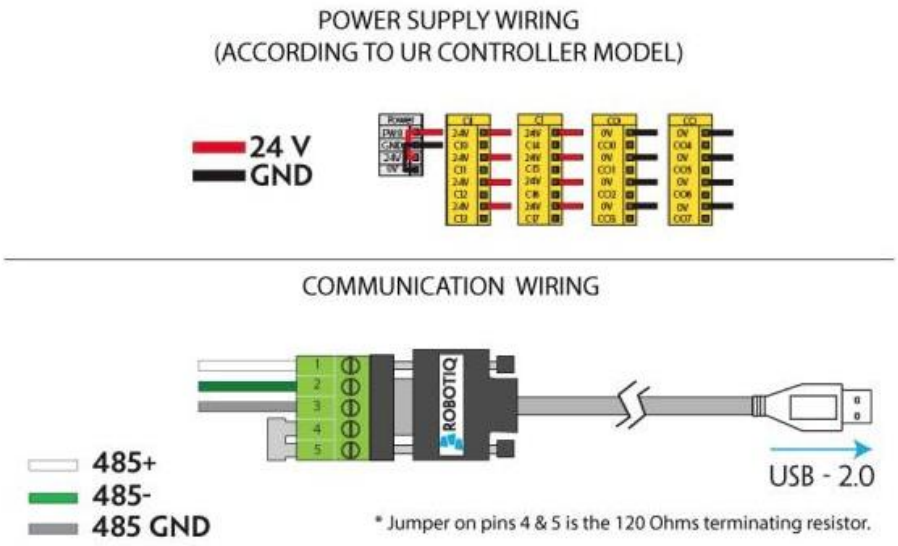


Figure 26. 2F-82 wiring. (*RobotiQ 2F-85, Instruction manual*)

5 VISION SYSTEM

The vision system is the one which has to identify the piece in the pickup place, get it shape from the background and the calculate its position and orientation. The pieces chosen for these projects are circular, so it wouldn't be necessary to get the orientation of the piece, but its implemented so a new object can be easily taught to the robot.
For accomplish this objective the camera used is the Wrist Camera from RobotiQ due to its easily implementation with Universal Robot.

5.1 Wrist Camera

The Wrist Camera is a light camera that have a resolution up to 5Mpx, frame rate from two to thirty frames per second and a focus from 70mm to infinity with automatic control. It has two sets of three led as an integrated lightning with automatic control. In figure 27 we can see the main feature of the Wrist Camera.

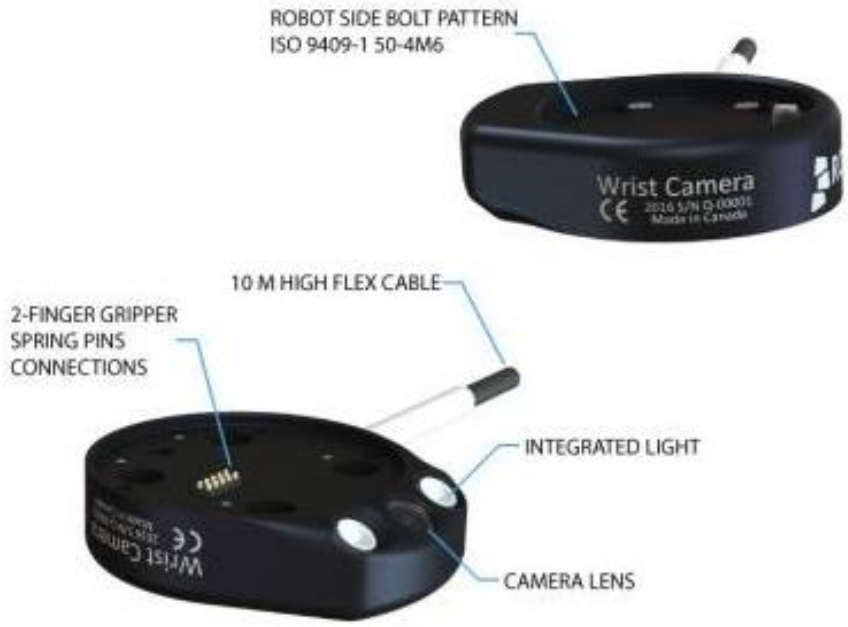


Figure 27. Main features of Wrist Camera. (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

The camera provides a direct mounting interface for the two-finger adaptive gripper, providing a mechanical interface, 24V power and communication to the Gripper. (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

5.1.1 Specifications

Here are the specifications of the Wirst camera. Some of the important data here is the added height because it will directly increase the TCP position, and that is used in the hole program every time.

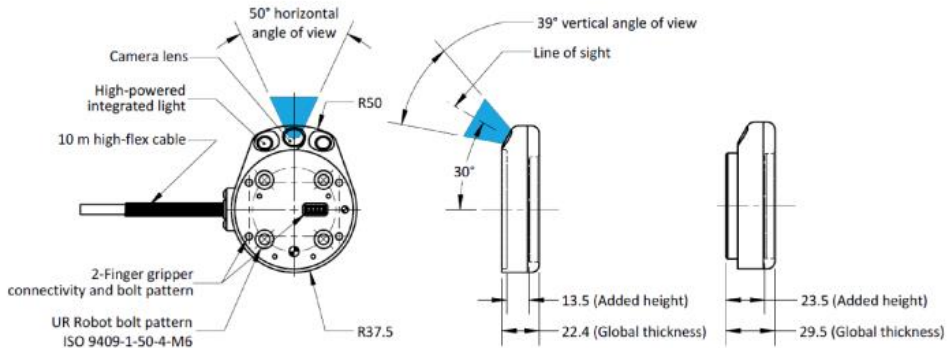


Figure 28. Camera dimensions (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

In table 1 is shown the specifications of the Wrist Camera, it weight is needed to calculate the actual force applied in the tool, so the robot can know the force needed to move the TCP to any position, and to know if any other force is there acting in the robot, and this directly affect to the security settings and limits. (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

Table 3. Wrist camera mechanical dimensions

Specification	**Value**	
Maximum load	10 Kg	40 Nm
Weight (without tool plate)	160 g	
Weight (with tool plate)	230 g	
Added height (for use with 2-Finger Gripper)	13.5 mm	
Global thickness (without tool plate)	22.4 mm	
Added height (with tool plate)	23.5 mm	

5.1.2 Electrical rating and performance

The wrist camera is connected to the controller box which will handle the power needed, as well as the communication throw the camera, gripper and robot.

- Electrical specifications:
  - Input voltage: 24V DC ±20%
  - Quiescent power: 1W
  - Maximum power: 22W
  - Com interface: USB 2.0
- Cameras specifications
  - Maximum resolution 5Mpx @ 2fps 2560x1920
  - Maximum frame rate 30fps at 0.3Mpx 640x480
  - Active array size 2592x1944
  - Focus range 70mm to infinity

5.1.3 Installation

MECHANICAL INSTALLATION

The camera is mounted between the robot wrist and the two-finger gripper, as shown in figure 29. The gripper can´t be mounted without the camera or a mounting plate.

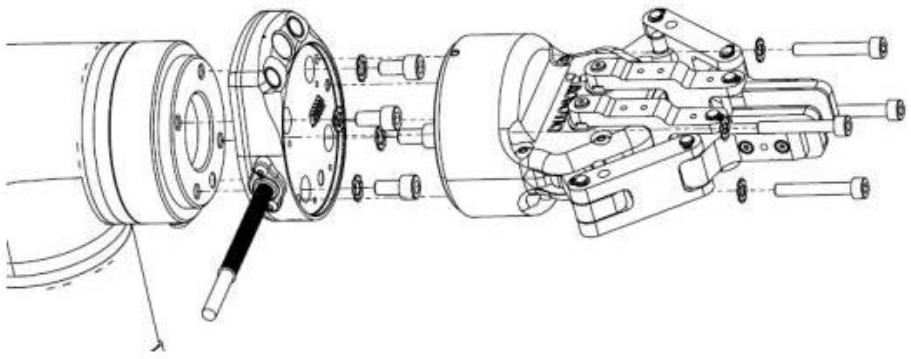


Figure 29. 2F-85 and Wrist installation(*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

The cable for the USB 2.0 communication will be grabbed with a flange along the two bigger arms of the robots, trying to keep it loose so it doesn't get damage when the robot is moving. (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

As mentioned before the Wrist Camera takes 24V DC, which will be given by the controller box following the diagram on figure 30.
The power and communication are established with the Wrist Camera via the high-flex device cable enabling USB 2.0 communication with the Universal Robot controller.

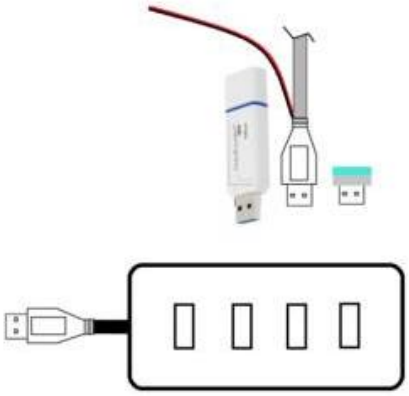


Figure 30. USB rack

In the figure 30 we can see how the USB rack is set in the interior of the controller box. From right to left, the first USB is the licence of the software, the next one is the Wrist Camera USB, through which all communication will be made, from this same USB there are two wires, one black, one red, which are connected to ground and 24V respectively, the last one is the USB memory stick which contains all the software installations needed to run it. (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

5.2 **Camera calibration**

Before doing the calibration, the Wrist Camera must be mounted correctly in the robot wrist. Ones it is correctly mounted the software needs a featured point called, snapshot position. For the calibration of the Wrist Camera on new background we will need the followings:

- Snapshot position.
- Calibration board for UR5.
- Calibration between the snapshot position and calibration board.

Snapshot Position: This position is the one in which all the movements relatives to the vision detection are based on. In the "*Installation"* tab of the Polyscope interface is where it must be defined as a featured point, as we can see in figure 31. This point must be variable to be modified by the robot software for each object detection.

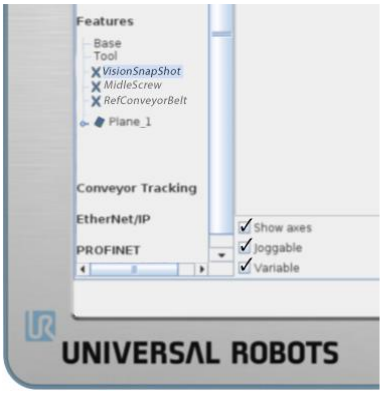


Figure 31. Definition of Snapshot position.

During the snapshot position definition, the ambient light must be of approximately 500 lux. The snapshot position will determine the field of view of the camera and thus the workspace used afterwards. The distance to the workplane will determinate the available workspace and the size of pieces it can detect. Reduced distance between the workplane and the snapshot position will allow to locate smaller pieces but in contras the available workspace will be reduced as well. (*RobotiQ. (n.d.). Wrist Camera Instruction Manual*)

Calibration board: It's shown in figure 32. These is the one used for the UR10 and the UR5 so it's the one used in this project. It has 4 coloured circles in each corner to determinate the size of it and then calculate the relative distance between the workplane and the snapshot position.

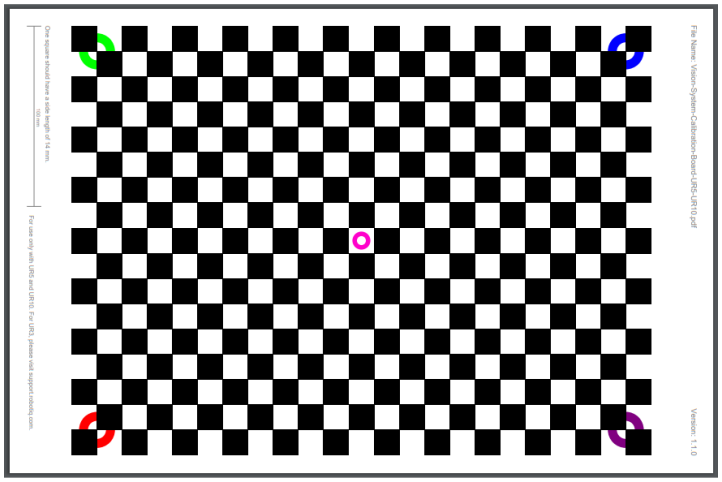


Figure 32. Calibration board (*RobotiQ.Instruction Manual*)

Calibration process: Ones we have set the snapshot position and the calibration board it's time of making the calibration between theses and get the real-world parameters in to the software.
Before continuing this process, the followings point must be checked out:

- 500 lux in constant light ambient during the calibration process.
- Maximum distance between workplane and snapshot position must be less than 700 mm.
- Minimum distance between workplane and snapshot position must be over 340 mm.
- Field of View (FoV) between 640 mm by 480mm and 100mm by 75mm
- Maximum part size can be detected is 60% of the FoV.
- Minimum part size detection 10% of the FoV.

Ones we set all of those, we can begin with the actual calibration process:

- First the software needs to link the snapshot position and the calibration point. To do so we have to navigate to:
    - *Camera* options in the Polyscope interface.
    - Then go to the *Snapshot positions* tab.
    - Next in the drop-down list we will set our Featured point.

  (In this program is set the *visionSnapShot* as shown in figure 31)

- Next step is launching the wizard:
    - First the robot will move to the actual Snapshot position (the calibration point doesn´t have to be the same as the snapshot position, but in this project is used the same to easy understood).
    - Ones is in place we will see the actual camera view in the Polyscope interface. We will need to check the followings:
        - The camera can see the whole board.
        - The board orientation should match the screen.
        - The board is well focused by the camera.
        - No reflexions are shown in the calibration board.
        - Make sure the robot workspace is clear.
        - The calibration board is clean to the view of the camera.
    - Next step is tap Calibrate to begin the calibration. The next steps will be done by the robot automatically (It will take up to 8 minutes depending on the robot configuration in the beginning of the calibration).
        - Vision System will centre on the board and take 27 poses of the board.
        - After that it will take 9 more photos for validating those.
        - When the process is done, it will ask to Accept or Re-Calibrate. (Figure 33)

Ones the calibration is done by the vision system will show the 9 validating poses each one with colours in the grid of the calibration board (Figure 33), as well as a colour chart where the dark blue colour mean a local accuracy of ±0mm and the dark red mean a local accuracy of ±4mm and over.

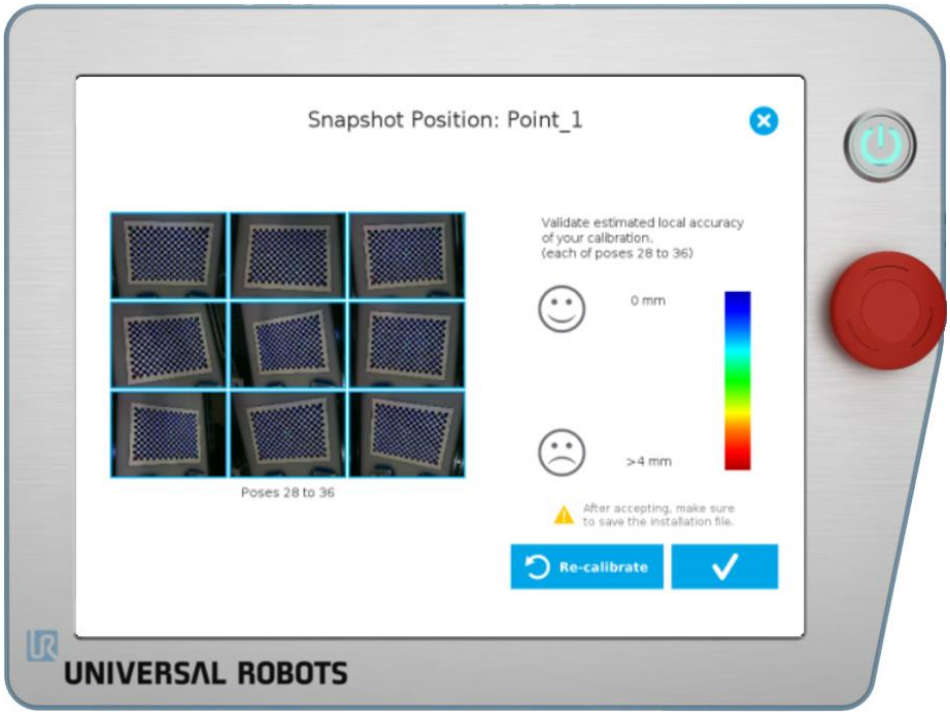


Figure 33. Calibration result. (*RobotiQ. WC Instruction Manual*)

If the calibration went so wrong that the accuracy is larger than ±4mm a message will inform to perform the calibration again.
Once the calibration has been accepted, the Snapshot Position will appear in the Snapshot Positions tab with the name of the Feature Point previously created.
Other snapshot positions can be defined, as long as new feature points are defined.

6 PROGRAM

6.1 Movements

As most others robot arms there are three typical types of movements, the linear move (*hereinafter moveL*), which makes the TCP going from point A to point B following a linear path, the programmed move (*hereinafter moveP*) which is used to follow circular path and last a joint move (*hereinafter moveJ*), which makes smoother joint movements and reaches more TCP speeds, but the path which describes the TCP can be more difficult to handle in reduced space, that's way moveL is used in approaches and moveJ in long distant paths.
So, some advantages and disadvantages of the types of moves are the followings:

6.1.1 Linear movement

Moves the tool linearly between waypoints. This means that each joint performs a more complicated motion to keep the tool on a straight-line path. The shared parameters that can be set for this movement type are the desired tool speed and tool acceleration specified in mm/s and mm/s2, respectively, and also a feature. The selected feature will determine in which feature space the tool positions of the waypoints are represented in.
Of specific interest concerning feature spaces are variable features and variable waypoints. Variable features can be used when the tool position of a waypoint need to be determined by the actual value of the variable feature when the robot program runs.

6.1.2 Joint movement

Makes movements that are calculated in the robot arm joint space.
Each joint is controlled to reach the desired end location at the same time. This movement type results in a curved path for the tool. The shared parameters that apply to this movement type are the maximum joint speed and joint acceleration to use for the movement calculations, specified in deg/s and deg/s^2, respectively.
If it's desired to have the robot arm move fast between waypoints, disregarding the path of the tool between those waypoints, this movement type is the best choice.

6.1.3 Programmed movement

This one moves the tool linearly with constant speed with circular blends, and is intended for some process operations, like gluing or dispensing. The size of the blend radius is by default a shared value between all the waypoints. A smaller value will make the path turn sharper whereas a higher value will make the path smoother.
While the robot arm is moving through the waypoints with constant speed, the robot control box cannot wait for either an I/O operation or an operator action. Doing so might stop the robot arm's motion or cause a protective stop.

- Circle move can be added to a moveP to make a circular movement. The robot starts the movement from its current position or start point, moves through a ViaPoint specified on the circular arc, and an EndPoint that completes the circular movement. A mode is used to calculate tool orientation, through the circular arc, as shown in figure 34 The mode can be:
    - **Fixed**: only the start point is used to define tool orientation.
    - **Unconstrained**: the start point transforms to the EndPoint to define tool orientation.

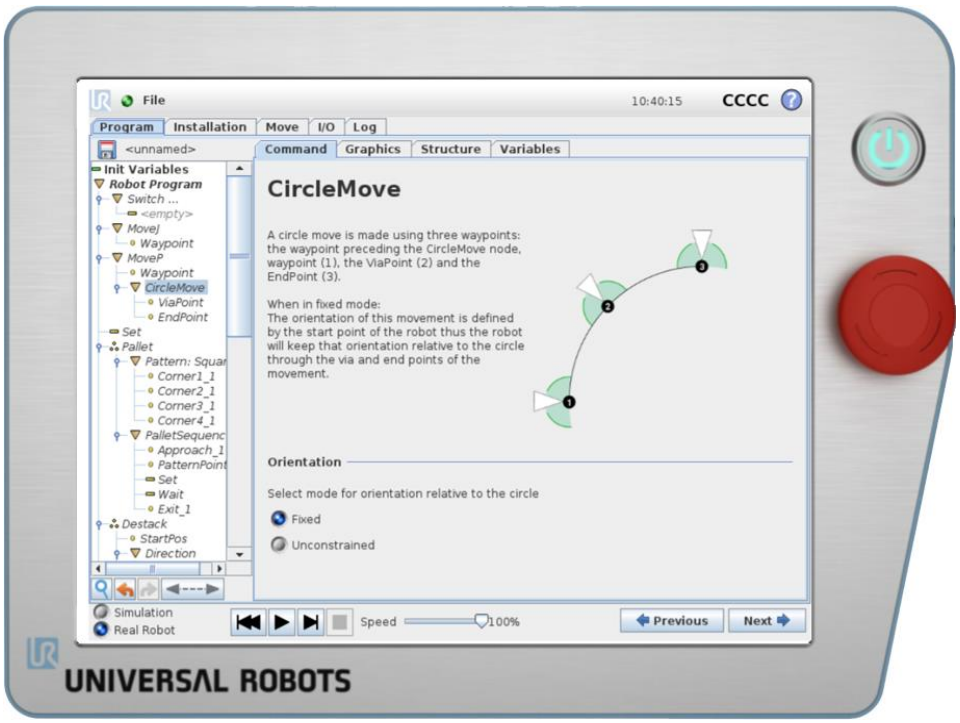


Figure 34. Circular move on Teach pendant (*Universal, R. (2018.), UR5 User manual*)

6.1.4 Feature selection

For moveL and moveP, it is possible to select which feature space the waypoints under the move command should be represented when specifying these waypoints. This means that when setting a waypoint, the program will remember the tool coordinates in the feature space of the selected feature.

There are a few circumstances that need detailed explanation:

- **Relative waypoints:** The selected feature has no effect on relative waypoints. The relative movement is always performed w.r.t. to orientation of the Base.
- **Variable waypoints:** When the robot arm moves to a variable waypoint, the tool target position is calculated as the coordinates of the variable in the space of the selected feature. Therefore, the robot arm movement for a variable waypoint changes if another feature is selected.
- **Variable feature:** If any of the features in the currently loaded installation are selected as variable, these corresponding variables are also selectable in the feature selection menu. If a feature variable (named with the name of the feature suffixed by " var") is selected, robot arm movements (except to Relative waypoints) are relative to the actual value of the variable when the program is running. The initial value of a feature variable is the value of the actual feature as configured in the installation. If this value is modified, then the movements of the robot change.

6.1.5 Speed profile for a motion

A motion will follow the figure 35 curve. It is divided into three segments: acceleration, cruise and deceleration. The level of the cruise phase is given by the speed setting of the motion, while the steepness of the acceleration and deceleration phases is given by the acceleration parameter.

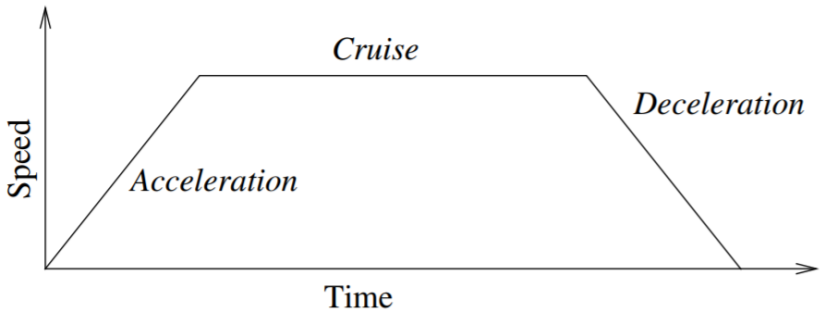


Figure 35. Speed vs Time for a motion

6.1.6 Blending

Blending enables the robot to smoothly transition between two trajectories, without stopping at the waypoint between them as we can see in the differences between the figure 36 and 37.

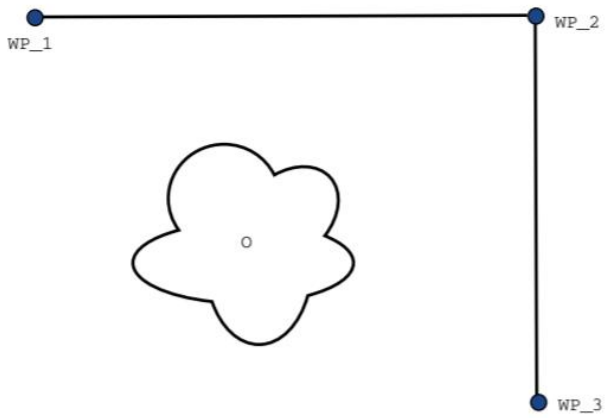


Figure 36. Non-blended trajectory (*Universal, R. (2018.), UR5 User manual*)

Blend parameters, apart from the waypoints, the followings multiple parameters will influence:

- The blend radius (r)
- The initial and final speed of the robot (at positions p1 and p2, respectively)
- The movement time (e.g. if setting a specific time for a trajectory this will influence the initial/final speed of the robot)
- The trajectory types to blend from and to (MoveL, MoveJ)

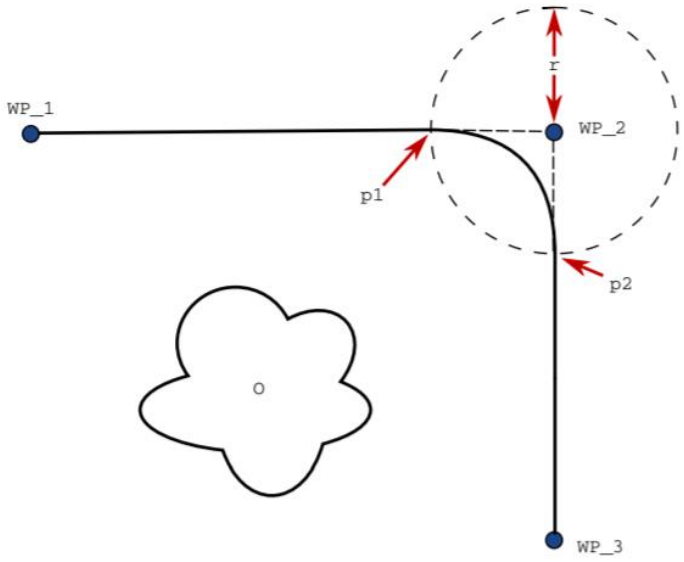


Figure 37. Blended trajectory (*Universal, R. (2018.), UR5 User manual*)

6.2 **Threads**

A thread is a parallel process to the robot program, it can be used to control an external machine independently of the robot arm. An important feature of the threats is that they share all the variables set in it and can access all the others declared in the main program. It's recommended not to use heavy computational operations in threats and leave those to the main program.
The threats are good raising up flags or setting some real-time values.

The use of threats in this project are aiming to check in every cycle of the controller, which is the real force applied to the robot and assign it to a variable called "*F"* so, in the stack with force subprogram we will have access in real-time to the force applied to the TCP.

6.3 **Stack with force**

Stack with force is used to know when the robot should stop when certain force is stopping the vertical movement, either because it has reached the bottom of the stack pile or it has found the last stacked piece in the pile.

There is more than one option to accomplish this objective, such as the built-in pallet command, but for make that run we will need to know the height of the piece in order to get the next one in the correct place. To accomplish this objective the diagram shown in figure 38 is going to be followed. It's a simple movement with the thread working in the background.

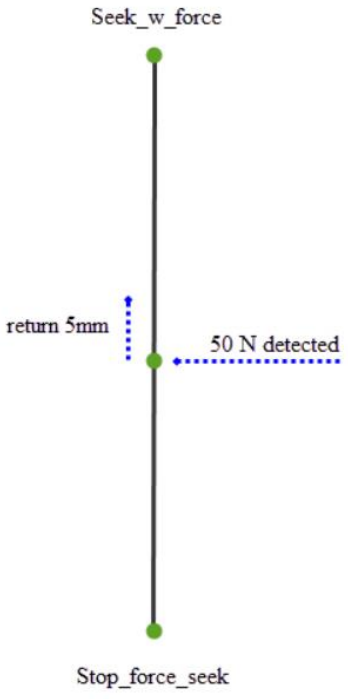


Figure 38. Seek with force diagram.

So, using the threads, we can know when the TCP force reach a defined amount of force, with a simple move command, setting as waypoint the bottom of the stack pile, and an if command evaluating the variable who has the actual forcer assigned by the thread and with the property "*Check expression continuously"* active, so if the expression is evaluated to True, at any point of the trajectory, the robot should retract 5mm (due to the fact that the table isn`t ideal rigid) and leave the piece there without continuing any further. Figure 39 shows the program explained above.

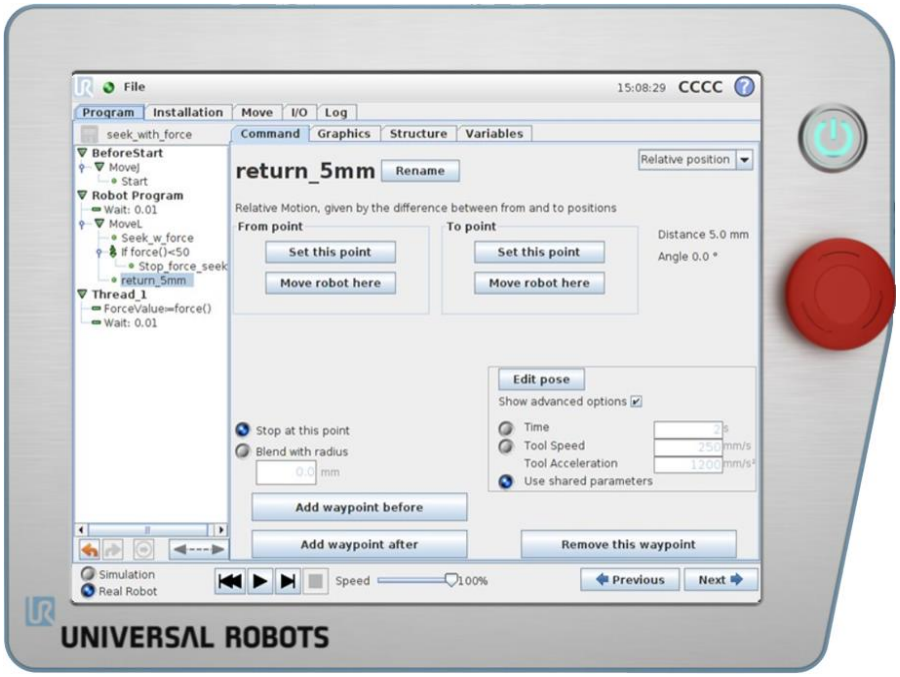


Figure 39. Seek with force program setup. (*Universal, R. (2018.), UR5 User manual*)

6.4 **Camera locate**

The camera locate node is valid for a single object, object teaching is linked to the snapshot position, if the snapshot positions change, the teaching will have to be performed again.

6.4.1 The background

Workplane around the object must be planar, mostly uniform and clear of any other objects. At runtime, the work space conditions can change, the object detection threshold can be used to adjust detection settings according to those conditions. The background around the object must be a uniform, continuous shape with a single colour, for that, the wrist camera where provided with a single colour non-brightness carpet.

6.4.2 Teaching a new object

Once a snapshot position is defined, and the robot in that position, only is needed to configure the Camera locate node. There are two ways of teaching a new object:

- Automatic method: Builds a model based on photos and a scan of the object. Best for complex and irregular shapes. This method is used if the object orientation has to be detected with one of its features.

- Parametric method: Builds a model based on parameters of a basic 2D shape (circle, ring, square or rectangle). This method is faster and allows the vision system to recognize and locate with high robustness objects that have few distinctive features such as raw material blanks. Usually gives best results than the Automatic method for simple geometry and highly reflective objects.

The method used to teach the pieces to takes is the parametric method because of the lack of borders, irregularities and remarkable shapes of the pieces to take. As shown in figure 40. Setting the height and diameter of the piece to teach, it will make a parametric object with that specifications and will be the one which the vision system will look for.

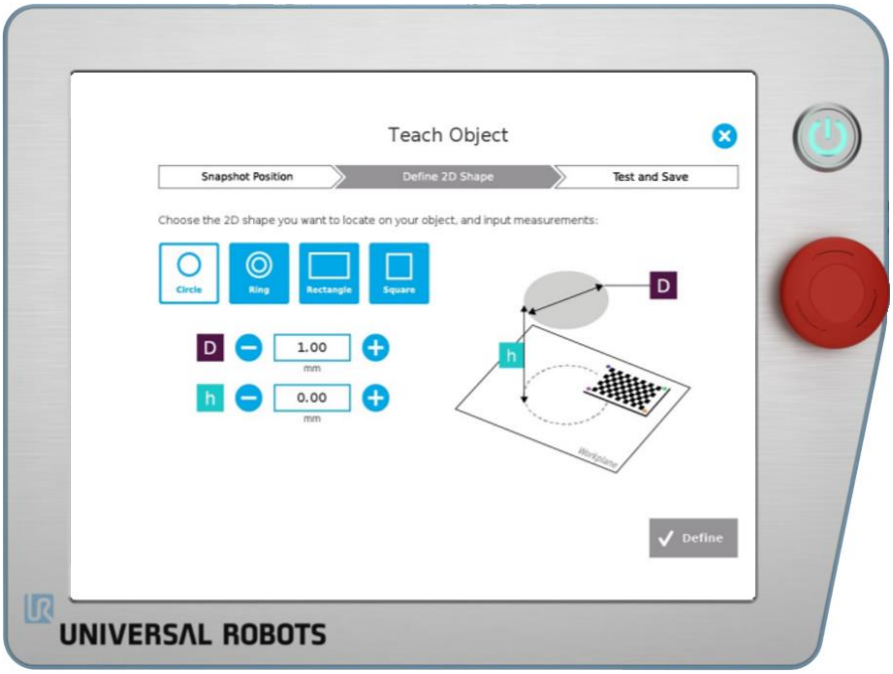


Figure 40. Introducing parametric cylinder. (*RobotiQ. WC Instruction Manual*)

7 CONCLUSION

In conclusion, the UR5 is able to feed the conveyor belt with no problem, once the offset with it have been set correctly.
It detects the face up pieces and dropped ones as well, what is an extra feature, but it is highly recommended to use constant ambient light to make the detection more accuracy and reliable.
It stacks with no matter of the height because it seeks with force, so the objective of stack piece is accomplished and no need to insert any information about the height of the pieces.
The hole setup can be reproduced in multiples stations due to its easy installation and copy paste of the program itself, so it is easy and fast to setup with any offset with the conveyor belt, but always remember to copy the program with the installation folder as well.

REFERENCES

ABB Robotics - Industrial Robots IRB 14000 YuMi (n.d.). Retrieved May 17, 2018, from http://new.abb.com/products/robotics/industrial-robots/yumi

Baxter Collaborative Robots for Industrial Automation. (n.d.). Retrieved May 18, 2018, from https://www.rethinkrobotics.com/baxter/

García, D. (2014, October 26). Robots Colaborativos - infoPLC. Retrieved May 17, 2018, from http://www.infoplc.net/blogs-automatizacion/item /102143 -robots-colaborativos

Kuka Robotics - LBR iiwa. (n.d.). Retrieved May 17, 2018, from https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

MABI Speedy 12 CobotHuman-robot collaboration (HRC). (n.d.). Retrieved May 17, 2018, from http://mabi-robotic.com/en/products /mab i-speedy-12

RobotiQ. (n.d.). Wrist Camera Instruction Manual. Retrieved June 5, 2018, from https://assets.robotiq.com/production/support_documents/document/Vision_System_PDF_20180509.pdf

RobotiQ. (n.d.). Retrieved May 20, 2018, from https://robotiq.com/

RobotiQ. (2016).*2-Finger Adaptive Robot Gripper Instruction Manual*[PDF]. *From UR.* [online] Available at: https://assets.robotiq.com/production/support_documents/document/2-Finger_PDF_20180510.pdf

Universal-robots.com. (2018). *Collaborative Industrial Robotic Robot Arms | Cobots from UR*. [online] Available at: https://www.universal-robots.com/ [Accessed 17 May 2018].

Universal, R. (n.d.). UR5 User manual. Retrieved June 5, 2018, from https://s3-eu-west-1.amazonaws.com/ur-support-site/32402/UR5_User_Manual_en_Global-3.5.5.pdf

Utilisateur, S. (n.d.). Dual-Arm Industrial Robot NEXTAGE. Retrieved May 18, 2018, from http://www.rollomatic.ch/en/products/collaborative-robot/descriptions

Appendix 1

UR CODE

```
def TFG():
  set_standard_analog_input_domain(0, 0)
  set_standard_analog_input_domain(1, 0)
  set_tool_analog_input_domain(0, 0)
  set_tool_analog_input_domain(1, 0)
  set_analog_outputdomain(0, 0)
  set_analog_outputdomain(1, 0)
  set_tool_voltage(0)
  set_input_actions_to_default()
  set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
  set_payload(1.06, [0.0, 0.0, 0.0])
  set_gravity([8.503688472232543E-16, 6.943788591251896, 6.943788591251898])
  global Home=p[-0.008712207145643067,-
0.22933140625325948,0.340789229788094,2.853080753184346,-
0.013223854815498428,0.003628374359291273]
  global MiddleScrew=p[-0.013764025967751689,-0.6826962260164302,-
4.928410913848693E-4,2.355636967311725,-0.016979266610951598,-
0.041361241622899336]
  global Table=p[0.34172218295414813,-0.5630757538354465,-
0.1301816543004791,1.7333607688611792,-
1.751571664238621,0.7490707197234436]
  global VisionSnapShot=p[-0.008702949451920692,-
0.22931119664704608,0.3407646500876095 3,2.8531734364069745,-
0.013331211340499383,0.003723914740261046]
  global refConveyorbelt=p[-0.33015510778526747,-
0.4980123221118575,0.5508381800340578,2.1795627460264364,1.559893668399618,-1.608151265030982]
  # begin: URCap Installation Node
  #   Source: Robotiq_Wrist_Camera, 1.2.3, Robotiq Inc.
  #   Type: Camera

  ##############################################
  #######Vision urcap preamble start########

  logging_service = rpc_factory("xmlrpc","http://127.0.0.1:4747")
  # Converts a pose relative to the flange in the base frame.
  def get_T_in_base_from_flange(T_x_in_flange):
    T_flange_in_base = get_actual_tool_flange_pose()
    T_x_in_base = pose_trans(T_flange_in_base, T_x_in_flange)
    return T_x_in_base
  end
```

```
# Search pose cartesian (camera pose)
VisionSnapShot = p[-0.00901427, -0.162668, 0.565441, 2.8532, -0.0132514, 0.00370997]
T_camera_in_flange = p[0.0, 0.05, 0.05, -0.5, 0.0, 0.0]
snapshot_position_offset = p[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
ignore_snapshot_position = False

# Open connection with vision service
xmlrpc_server=rpc_factory("xmlrpc","http://127.0.0.1:4242")

#######Vision urcap preamble end##########
##########################################

# end: URCap Installation Node
# begin: URCap Installation Node
#   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
#   Type: Gripper

##########################################
#######Gripper URCap preamble start########
#######Version null########

#aliases for the gripper variable names
ACT = 1
GTO = 2
ATR = 3
ARD = 4
FOR = 5
SPE = 6
OBJ = 7
STA = 8
FLT = 9
POS = 10
PRE = 11

def rq_init_connection(gripper_sid=9, gripper_socket="1"):
                socket_open("127.0.0.1",63352, gripper_socket)
                socket_set_var("SID", gripper_sid,  gripper_socket)
                ack = socket_read_byte_list(3, gripper_socket)
end

def rq_set_sid(gripper_sid=9, gripper_socket="1"):
      socket_set_var("SID", gripper_sid,  gripper_socket)
      sync()
      return is_ack(socket_read_byte_list(3, gripper_socket))
end
def rq_activate(gripper_socket="1"):
                rq_gripper_act = 0
```

```
    if (not rq_is_gripper_activated(gripper_socket)):
      rq_reset(gripper_socket)
    end
        rq_set_var(ACT,1, gripper_socket)
end

def rq_activate_and_wait(gripper_socket="1"):
        rq_activate(gripper_socket)

        while(not rq_is_gripper_activated(gripper_socket)):
                # wait for activation completed
        end
end

def rq_stop(gripper_socket="1"):
        rq_set_var(GTO,0, gripper_socket)
end

def rq_reset(gripper_socket="1"):
        rq_gripper_act = 0
        rq_obj_detect = 0
        rq_mov_complete = 0

        rq_set_var(ACT,0, gripper_socket)
        rq_set_var(ATR,0, gripper_socket)
end

def rq_auto_release_open_and_wait(gripper_socket="1"):

        rq_set_var(ARD,0, gripper_socket)
        rq_set_var(ACT,1, gripper_socket)
        rq_set_var(ATR,1, gripper_socket)

        gFLT = rq_get_var(FLT, 2, gripper_socket)

        while(not is_FLT_autorelease_completed(gFLT)):
                gFLT = rq_get_var(FLT, 2, gripper_socket)
        end
end

def rq_auto_release_close_and_wait(gripper_socket="1"):
        rq_set_var(ARD,1, gripper_socket)
        rq_set_var(ACT,1, gripper_socket)
        rq_set_var(ATR,1, gripper_socket)

        gFLT = rq_get_var(FLT, 2, gripper_socket)
```

```
		while(not is_FLT_autorelease_completed(gFLT)):
			gFLT = rq_get_var(FLT, 2, gripper_socket)
		end
end

def rq_set_force(force, gripper_socket="1"):
		rq_set_var(FOR,force, gripper_socket)
end

def rq_set_speed(speed, gripper_socket="1"):
		rq_set_var(SPE,speed, gripper_socket)
end

def rq_open(gripper_socket="1"):
		rq_move(0, gripper_socket)
end

def rq_close(gripper_socket="1"):
		rq_move(255, gripper_socket)
end

def rq_open_and_wait(gripper_socket="1"):
		rq_move_and_wait(0, gripper_socket)
end

def rq_close_and_wait(gripper_socket="1"):
		rq_move_and_wait(255, gripper_socket)
end

def rq_move(pos, gripper_socket="1"):
		rq_mov_complete = 0
		rq_obj_detect = 0

		rq_set_pos(pos, gripper_socket)
		rq_go_to(gripper_socket)
end

def rq_move_and_wait(pos, gripper_socket="1"):
		rq_move(pos, gripper_socket)

		while (not rq_is_motion_complete(gripper_socket)):
			# wait for motion completed
			sleep(0.01)
			sync()
		end

		# following code used for compatibility with previous versions
		rq_is_object_detected(gripper_socket)
```

```
		if (rq_obj_detect != 1):
			rq_mov_complete = 1
		end
end

def rq_wait(gripper_socket="1"):
	# Wait for the gripper motion to complete
	while (not rq_is_motion_complete(gripper_socket)):
			# wait for motion completed
			sleep(0.01)
			sync()
		end

		# following code used for compatibility with previous versions
		rq_is_object_detected(gripper_socket)

		if (rq_obj_detect != 1):
			rq_mov_complete = 1
		end
end

def rq_go_to(gripper_socket="1"):
		rq_set_var(GTO,1, gripper_socket)
end

# reset the rGTO to prevent movement and set the position
def rq_set_pos(pos, gripper_socket="1"):
		rq_set_var(GTO,0, gripper_socket)

		rq_set_var(POS, pos, gripper_socket)

		gPRE = rq_get_var(PRE, 3, gripper_socket)
		pre = (gPRE[1] - 48)*100 + (gPRE[2] -48)*10 + gPRE[3] - 48
		sync()
		while (pre != pos):
	rq_set_var(POS, pos, gripper_socket)
			gPRE = rq_get_var(PRE, 3, gripper_socket)
			pre = (gPRE[1] - 48)*100 + (gPRE[2] -48)*10 + gPRE[3] - 48
			sync()
		end
end

def rq_is_motion_complete(gripper_socket="1"):
		rq_mov_complete = 0

		gOBJ = rq_get_var(OBJ, 1, gripper_socket)
		sleep(0.01)
```

```
		if (is_OBJ_gripper_at_position(gOBJ)):
			rq_mov_complete = 1
			return True
		end

		if (is_OBJ_object_detected(gOBJ)):
			rq_mov_complete = 1
			return True
		end

		return False

end

def rq_is_gripper_activated(gripper_socket="1"):
		gSTA = rq_get_var(STA, 1, gripper_socket)

		if(is_STA_gripper_activated(gSTA)):
			rq_gripper_act = 1
			return True
		else:
			rq_gripper_act = 0
			return False
		end
end

def rq_is_object_detected(gripper_socket="1"):
		gOBJ = rq_get_var(OBJ, 1, gripper_socket)

		if(is_OBJ_object_detected(gOBJ)):
			rq_obj_detect = 1
			return True
		else:
			rq_obj_detect = 0
			return False
		end
end

def rq_current_pos(gripper_socket="1"):
		rq_pos = socket_get_var("POS",gripper_socket)
		sync()
  return rq_pos
end

def rq_print_gripper_fault_code(gripper_socket="1"):
		gFLT = rq_get_var(FLT, 2, gripper_socket)
```

```
		if(is_FLT_no_fault(gFLT)):
			textmsg("Gripper Fault : ", "No Fault (0x00)")
		elif (is_FLT_action_delayed(gFLT)):
			textmsg("Gripper Fault : ", "Priority Fault: Action delayed, initialization must be completed prior to action (0x05)")
		elif (is_FLT_not_activated(gFLT)):
			textmsg("Gripper Fault : ", "Priority Fault: The activation must be set prior to action (0x07)")
		elif (is_FLT_autorelease_in_progress(gFLT)):
			textmsg("Gripper Fault : ", "Minor Fault: Automatic release in progress (0x0B)")
		elif (is_FLT_overcurrent(gFLT)):
			textmsg("Gripper Fault : ", "Minor Fault: Overcurrent protection tiggered (0x0E)")
		elif (is_FLT_autorelease_completed(gFLT)):
			textmsg("Gripper Fault : ", "Major Fault: Automatic release completed (0x0F)")
		else:
			textmsg("Gripper Fault : ", "Unkwown Fault")
		end
 end

 def rq_print_gripper_num_cycles(gripper_socket="1"):
		socket_send_string("GET NCY",gripper_socket)
		sync()
		string_from_server = socket_read_string(gripper_socket)
		sync()

		if(string_from_server == "0"):
			textmsg("Gripper Cycle Number : ", "Number of cycles is unreachable.")
		else:
			textmsg("Gripper Cycle Number : ", string_from_server)
		end
 end

 def rq_print_gripper_driver_state(gripper_socket="1"):
		socket_send_string("GET DST",gripper_socket)
		sync()
		string_from_server = socket_read_string(gripper_socket)
		sync()

		if(string_from_server == "0"):
			textmsg("Gripper Driver State : ", "RQ_STATE_INIT")
		elif(string_from_server == "1"):
			textmsg("Gripper Driver State : ", "RQ_STATE_LISTEN")
		elif(string_from_server == "2"):
			textmsg("Gripper Driver State : ", "RQ_STATE_READ_INFO")
```

```
		elif(string_from_server == "3"):
			textmsg("Gripper Driver State : ", "RQ_STATE_ACTIVATION")
		else:
			textmsg("Gripper Driver State : ", "RQ_STATE_RUN")
		end
	end

	def rq_print_gripper_serial_number():
		#socket_send_string("GET SNU",gripper_socket)
		#sync()
		#string_from_server = socket_read_string(gripper_socket)
		#sync()
		#textmsg("Gripper Serial Number : ", string_from_server)
	end

	def rq_print_gripper_firmware_version(gripper_socket="1"):
		socket_send_string("GET FWV",gripper_socket)
		sync()
		string_from_server = socket_read_string(gripper_socket)
		sync()
		textmsg("Gripper Firmware Version : ", string_from_server)
	end

	def rq_print_gripper_driver_version(gripper_socket="1"):
		socket_send_string("GET VER",gripper_socket)
		sync()
		string_from_server = socket_read_string(gripper_socket)
		sync()
		textmsg("Gripper Driver Version : ", string_from_server)
	end

	def rq_print_gripper_probleme_connection(gripper_socket="1"):
		socket_send_string("GET PCO",gripper_socket)
		sync()
		string_from_server = socket_read_string(gripper_socket)
		sync()
		if (string_from_server == "0"):
			textmsg("Gripper Connection State : ", "No connection
problem detected")
		else:
			textmsg("Gripper Connection State : ", "Connection problem
detected")
		end
	end

	# Returns True if list_of_bytes is [3, 'a', 'c', 'k']
	def is_ack(list_of_bytes):
```

```
		# list length is not 3
		if (list_of_bytes[0] != 3):
				return False
		end

		# first byte not is 'a'?
		if (list_of_bytes[1] != 97):
				return False
		end

		# first byte not is 'c'?
		if (list_of_bytes[2] != 99):
				return False
		end

		# first byte not is 'k'?
		if (list_of_bytes[3] != 107):
				return False
		end

		return True
end

# Returns True if list_of_bytes is not [3, 'a', 'c', 'k']
def is_not_ack(list_of_bytes):
		if (is_ack(list_of_bytes)):
				return False
		else:
				return True
		end
end

def is_STA_gripper_activated (list_of_bytes):

		# list length is not 1
		if (list_of_bytes[0] != 1):
				return False
		end

		# byte is '3'?
		if (list_of_bytes[1] == 51):
				return True
		end

		return False
end

# Returns True if list_of_byte is [1, '1'] or [1, '2']
```

```
# Used to test OBJ = 0x1 or OBJ = 0x2
def is_OBJ_object_detected (list_of_bytes):

		# list length is not 1
		if (list_of_bytes[0] != 1):
				return False
		end

		# byte is '2'?
		if (list_of_bytes[1] == 50):
				return True
		end

		# byte is '1'?
		if (list_of_bytes[1]  == 49):
				return True
		end

		return False

end

# Returns True if list_of_byte is [1, '3']
# Used to test OBJ = 0x3
def is_OBJ_gripper_at_position (list_of_bytes):

		# list length is not 1
		if (list_of_bytes[0] != 1):
				return False
		end

		# byte is '3'?
		if (list_of_bytes[1] == 51):
				return True
		end

		return False
end

def is_not_OBJ_gripper_at_position (list_of_bytes):

		if (is_OBJ_gripper_at_position(list_of_bytes)):
				return False
		else:
				return True
		end
end
```

```
def is_FLT_no_fault(list_of_bytes):

        # list length is not 2
        if (list_of_bytes[0] != 2):
                return False
        end

        # first byte is '0'?
        if (list_of_bytes[1] != 48):
                return False
        end

        # second byte is '0'?
        if (list_of_bytes[2] != 48):
                return False
        end

        return True

end

def is_FLT_action_delayed(list_of_bytes):

        # list length is not 2
        if (list_of_bytes[0] != 2):
                return False
        end

        # first byte is '0'?
        if (list_of_bytes[1] != 48):
                return False
        end

        # second byte is '5'?
        if (list_of_bytes[2] != 53):
                return False
        end

        return True
end

def is_FLT_not_activated(list_of_bytes):

        # list length is not 2
        if (list_of_bytes[0] != 2):
                return False
        end
```

```
                # first byte is '0'?
                if (list_of_bytes[1] != 48):
                                return False
                end

                # second byte is '7'?
                if (list_of_bytes[2] != 55):
                                return False
                end

                return True
end

def is_FLT_autorelease_in_progress(list_of_bytes):

                # list length is not 2
                if (list_of_bytes[0] != 2):
                                return False
                end

                # first byte is '1'?
                if (list_of_bytes[1] != 49):
                                return False
                end

                # second byte is '1'?
                if (list_of_bytes[2] != 49):
                                return False
                end

                return True

end

def is_FLT_overcurrent(list_of_bytes):

                # list length is not 2
                if (list_of_bytes[0] != 2):
                                return False
                end

                # first byte is '1'?
                if (list_of_bytes[1] != 49):
                                return False
                end

                # second byte is '4'?
                if (list_of_bytes[2] != 52):
```

```
                return False
        end

        return True

end

def is_FLT_autorelease_completed(list_of_bytes):

        # list length is not 2
        if (list_of_bytes[0] != 2):
                return False
        end

        # first byte is '1'?
        if (list_of_bytes[1] != 49):
                return False
        end

        # second byte is '5'?
        if (list_of_bytes[2] != 53):
                return False
        end

        return True

end

def rq_set_var(var_name, var_value, gripper_socket="1"):

        sync()
        if (var_name == ACT):
                socket_set_var("ACT", var_value, gripper_socket)
        elif (var_name == GTO):
                socket_set_var("GTO", var_value, gripper_socket)
        elif (var_name == ATR):
                socket_set_var("ATR", var_value, gripper_socket)
        elif (var_name == ARD):
                socket_set_var("ARD", var_value, gripper_socket)
        elif (var_name == FOR):
                socket_set_var("FOR", var_value, gripper_socket)
        elif (var_name == SPE):
                socket_set_var("SPE", var_value, gripper_socket)
        elif (var_name == POS):
                socket_set_var("POS", var_value, gripper_socket)
        else:
        end
```

```
		sync()
		ack = socket_read_byte_list(3, gripper_socket)
		sync()

		while(is_not_ack(ack)):

				textmsg("rq_set_var : retry", " ...")
				textmsg("rq_set_var : var_name = ", var_name)
				textmsg("rq_set_var : var_value = ", var_value)

				if (ack[0] != 0):
					textmsg("rq_set_var : invalid ack value = ", ack)
				end

				socket_set_var(var_name , var_value,gripper_socket)
				sync()
				ack = socket_read_byte_list(3, gripper_socket)
				sync()
		end
end


def rq_get_var(var_name, nbr_bytes, gripper_socket="1"):

		if (var_name == FLT):
				socket_send_string("GET FLT",gripper_socket)
				sync()
		elif (var_name == OBJ):
				socket_send_string("GET OBJ",gripper_socket)
				sync()
		elif (var_name == STA):
				socket_send_string("GET STA",gripper_socket)
				sync()
		elif (var_name == PRE):
				socket_send_string("GET PRE",gripper_socket)
				sync()
		else:
		end

		var_value = socket_read_byte_list(nbr_bytes, gripper_socket)
		sync()

		return var_value
end

##############################################
# normalized functions (maps 0-100 to 0-255)
##############################################
```

```
def rq_set_force_norm(force_norm, gripper_socket="1"):
  force_gripper = norm_to_gripper(force_norm)
  rq_set_force(force_gripper, gripper_socket)
end

def rq_set_speed_norm(speed_norm, gripper_socket="1"):
  speed_gripper = norm_to_gripper(speed_norm)
  rq_set_speed(speed_gripper, gripper_socket)
end

def rq_move_norm(pos_norm, gripper_socket="1"):
  pos_gripper = norm_to_gripper(pos_norm)
  rq_move(pos_gripper, gripper_socket)
end

def rq_move_and_wait_norm(pos_norm, gripper_socket="1"):
  pos_gripper = norm_to_gripper(pos_norm)
  rq_move_and_wait(pos_gripper, gripper_socket)
end

def rq_set_pos_norm(pos_norm, gripper_socket="1"):
  pos_gripper = norm_to_gripper(pos_norm)
  rq_set_pos(pos_gripper, gripper_socket)
end


def rq_current_pos_norm(gripper_socket="1"):
  pos_gripper = rq_current_pos(gripper_socket)
  pos_norm = gripper_to_norm(pos_gripper)
  return pos_norm
end

def gripper_to_norm(value_gripper):
  value_norm = (value_gripper / 255) * 100
  return floor(value_norm)
end

def norm_to_gripper(value_norm):
  value_gripper = (value_norm / 100) * 255
  return ceil(value_gripper)
end

def rq_get_position():
  return rq_current_pos_norm()
end
##########################################
rq_obj_detect = 0
rq_init_connection(9, "1")
```

```
connectivity_checked = [-1,-1,-1,-1]
status_checked = [-1,-1,-1,-1]
current_speed = [-1,-1,-1,-1]
current_force = [-1,-1,-1,-1]

#######Gripper URCap preamble end##########
##########################################

# end: URCap Installation Node
# begin: URCap Installation Node
#   Source: RG - On Robot, 1.2.1, On Robot ApS
#   Type: RG6 Configuration
global measure_width=0
global grip_detected=False
global lost_grip=False
global zsysx=0
global zsysy=0
global zsysz=0.10205
global zsysm=1.085
global zmasx=0
global zmasy=-0
global zmasz=0.23434
global zmasm=0
global zmasm=0
global zslax=0
global zslay=0
global zslaz=0
global zslam=0
global zslam=0
thread lost_grip_thread():
while True:
set_tool_voltage(24)
        if True ==get_digital_in(9):
                sync()
                sync()
                sync()
                if True == grip_detected:
                        if False == get_digital_in(8):
                                grip_detected=False
                                lost_grip=True
                        end
                end
        set_tool_analog_input_domain(0, 1)
        set_tool_analog_input_domain(1, 1)
        zscale = (get_analog_in(2)-0.026)/2.976
        zangle = zscale*1.57079633-0.0942477796
        zwidth = 8.4+160*sin(zangle)
        global measure_width = (floor(zwidth*10))/10-9.2
```

```
                end
                sync()
 end
 end
 lg_thr = run lost_grip_thread()
 def RG6(target_width=160, target_force=120, payload=0.0, set_payload=False,
depth_compensation=False, slave=False):
                grip_detected=False
                if slave:
                                slave_grip_detected=False
                else:
                                master_grip_detected=False
                end
                timeout = 0
                while get_digital_in(9) == False:
                 if timeout > 400:
                  break
                 end
                 timeout = timeout+1
                 sync()
                end
                def bit(input):
                 msb=65536
                 local i=0
                 local output=0
                 while i<17:
                  set_digital_out(8,True)
                  if input>=msb:
                   input=input-msb
                   set_digital_out(9,False)
                  else:
                   set_digital_out(9,True)
                  end
                  if get_digital_in(8):
                   out=1
                  end
                  sync()
                  set_digital_out(8,False)
                  sync()
                  input=input*2
                  output=output*2
                  i=i+1
                 end
                 return output
                end
                target_width=target_width+9.2
                if target_force>120:
                target_force=120
```

```
end
if target_force<25:
target_force=25
end
if target_width>160:
target_width=160
end
if target_width<0:
target_width=0
end
rg_data=floor(target_width)*4
rg_data=rg_data+floor(target_force/5)*4*161
if slave:
rg_data=rg_data+16384
end
bit(rg_data)
if depth_compensation:
finger_length = 80.0/1000
finger_heigth_disp = 6.3/1000
center_displacement = 10.5/1000

start_pose = get_forward_kin()
set_analog_inputrange(2, 1)
zscale = (get_analog_in(2)-0.026)/2.976
zangle = zscale*1.57079633-0.0942477796
zwidth = 8.4+160*sin(zangle)

start_depth = cos(zangle)*finger_length

sync()
sync()
timeout = 0
while get_digital_in(9) == True:
 timeout=timeout+1
 sync()
 if timeout > 20:
  break
 end
end
timeout = 0
while get_digital_in(9) == False:
 zscale = (get_analog_in(2)-0.026)/2.976
 zangle = zscale*1.57079633-0.0942477796
 zwidth = 8.4+160*sin(zangle)
 measure_depth = cos(zangle)*finger_length
 compensation_depth = (measure_depth - start_depth)
 target_pose = pose_trans(start_pose,p[0,0,-compensation_depth,0,0,0])
 if timeout > 400:
```

```
                  break
                 end
                 timeout=timeout+1
               #   servoj(get_inverse_kin(target_pose), t=0.008, lookahead_time=0.033,
gain=1500)
               #  textmsg(point_dist(target_pose, get_forward_kin()))
               #end
               #textmsg("end gripper move!!!!!")
               #nspeedthr = 0.001
               #nspeed = norm(get_actual_tcp_speed())
               #while nspeed > nspeedthr:
               #   servoj(get_inverse_kin(target_pose), t=0.008, lookahead_time=0.033,
gain=1500)
               #  nspeed = norm(get_actual_tcp_speed())
               #  textmsg(point_dist(target_pose, get_forward_kin()))
               #end
               servoj(get_inverse_kin(target_pose),0,0,0.008,0.01,2000)
               if point_dist(target_pose, get_forward_kin()) > 0.005:
               popup("Lower grasping force or max width",title="RG-lag threshold
exceeded", warning=False, error=False, blocking=False)
               end
               end
               nspeed = norm(get_actual_tcp_speed())
               while nspeed > 0.001:
               servoj(get_inverse_kin(target_pose),0,0,0.008,0.01,2000)
               nspeed = norm(get_actual_tcp_speed())
               end
               end
               if depth_compensation==False:
               timeout = 0
               while get_digital_in(9) == True:
                timeout = timeout+1
                sync()
                if timeout > 20:
                  break
                end
               end
               timeout = 0
               while get_digital_in(9) == False:
                timeout = timeout+1
                sync()
                if timeout > 400:
                  break
                end
               end
               end
               sync()
               sync()
```

```
            sync()
            if set_payload:
            if slave:
            if get_analog_in(3) < 2:
            zslam=0
            else:
            zslam=payload
            end
            else:
            if get_digital_in(8) == False:
            zmasm=0
            else:
            zmasm=payload
            end
            end
            zload=zmasm+zslam+zsysm

            set_payload(zload,[(zsysx*zsysm+zmasx*zmasm+zslax*zslam)/zload,(zsys
y*zsysm+zmasy*zmasm+zslay*zslam)/zload,(zsysz*zsysm+zmasz*zmasm+zslaz*zslam)
/zload])
            end
            master_grip_detected=False
            master_lost_grip=False
            slave_grip_detected=False
            slave_lost_grip=False
            if True == get_digital_in(8):
                        master_grip_detected=True
            end
            if get_analog_in(3)>2:
                        slave_grip_detected=True
            end
            grip_detected=False
            lost_grip=False
            if True == get_digital_in(8):
                        grip_detected=True
            end
            zscale = (get_analog_in(2)-0.026)/2.976
            zangle = zscale*1.57079633-0.0942477796
            zwidth = 8.4+160*sin(zangle)
            global measure_width = (floor(zwidth*10))/10-9.2
            if slave:
            slave_measure_width=measure_width
            else:
            master_measure_width=measure_width
            end
            return grip_detected
 end
 set_tool_voltage(24)
```

```
  set_tcp(p[0,-0,0.23435,0,-0,0])
  # end: URCap Installation Node
  global F=0
  global MaxF=0
  global ObjectLocated=  False
  global PieceLeaved=  False
  $ 68 "Thread_1"
  thread Thread_1():
    while (True):
      sleep(0.01)
      global F=force()
      if (MaxF<F):
        global MaxF=F
      end
      if (PieceLeaved ==   True  ):
        set_standard_digital_out(1, True)
        sleep(3.0)
        set_standard_digital_out(1, False)
        global PieceLeaved=  False
      end
    end
  end
  threadId_Thread_1 = run Thread_1()
  while (True):
    $ 2 "Robot Program"
    $ 3 "MoveJ"
    $ 4 "Home_var"
    movej(Home, a=1.3962634015954636, v=1.0471975511965976)
    # begin: URCap Program Node
    #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
    #   Type: Gripper
    $ 5 "Gripper Open (1)"
    if (connectivity_checked[0] != 1):
      if not(rq_set_sid(9, "1")):
        popup("Gripper 1 must be connected to run this program.", False, False, True)
      end
      connectivity_checked[0] = 1
    end
    if (status_checked[0] != 1):
      if not(rq_is_gripper_activated("1")):
        popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it and
run the program again.", False, False, True)
      end
      status_checked[0] = 1
    end
    if (current_speed[0] != 100):
      rq_set_speed_norm(100, "1")
      current_speed[0] = 100
```

```
  end
  if (current_force[0] != 0):
   rq_set_force_norm(0, "1")
   current_force[0] = 0
  end
  rq_set_pos_norm(0, "1")
  rq_go_to("1")
  rq_wait("1")
  # end: URCap Program Node
  $ 6 "If digital_in[1]≟ True "
  if (get_standard_digital_in(1) ==  True  ):
   # begin: URCap Program Node
   #  Source: Robotiq_Wrist_Camera, 1.2.3, Robotiq Inc.
   #  Type: Cam Locate
   $ 7 "Camera Locate"

   ##########################################
   #######VisionLocate node start########

   # Offset in translation only.
   snapshot_position_offset[3] = 0
   snapshot_position_offset[4] = 0
   snapshot_position_offset[5] = 0
   T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour l'instant, on suppose que la camera est situee directement sur la flange.
   tool = get_T_in_base_from_flange(T_camera_in_flange)
   textmsg("actual tool flange : ", tool)
   tool = pose_sub(tool, snapshot_position_offset)
   textmsg("tool after offset : ", tool)
   snapshot_position = p[-0.00901427, -0.162668, 0.565441, 2.8532, -0.0132514, 0.00370997]
   textmsg("expected snapshot position : ", snapshot_position)
   diff = pose_sub(tool, snapshot_position)
   textmsg("diff = ", diff)
   textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
   textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
   is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
   textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
   is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) < 0.005)
   textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
   is_snapshot_position_offset = norm(snapshot_position_offset) != 0
   is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
   if not(is_at_snapshot_position):
    popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position before Camera Locate node.. Error code: [UCC-8]", False, False, True)
    halt
```

```
    end
    f = xmlrpc_server.findmodel("contextName-57422", tool[0], tool[1], tool[2], tool[3],
tool[4], tool[5])
    nbOccu = f[0]
    logging_service.publish("FIND_MODEL", f)
    object_teaching_location = p[0.052274725729417895, -0.6600366799538961,
0.124595725918615, 2.329619196230474, -0.0027105713369098465, -
0.004191786633438582]
    object_location = p[f[1], f[2], f[3], f[4], f[5], f[6]]
    textmsg("object_location before offset = ", object_location)
    object_location = pose_add(object_location, snapshot_position_offset)
    textmsg("object_location after offset = ", object_location)
    feature_teaching_reference = p[-0.008702949451920692, -0.22931119664704608,
0.34076465008760953, 2.8531734364069745, -0.013331211340499383,
0.003723914740261046]
    VisionSnapShot = pose_trans(object_location,
pose_trans(pose_inv(p[0.052274725729417895, -0.6600366799538961,
0.124595725918615, 2.329619196230474, -0.0027105713369098465, -
0.004191786633438582]), p[-0.008702949451920692, -0.22931119664704608,
0.34076465008760953, 2.8531734364069745, -0.013331211340499383,
0.003723914740261046]))
    if (nbOccu > 0.5):
     $ 8 "Grab"
     $ 9 "MoveL"
     $ 10 "aproachTop"
     set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
     movel(pose_trans(VisionSnapShot,
p[0.0665768905383662,0.3167619319644588,0.27313477215099846,-
0.5236374783766486,-4.1066383609840315E-4,2.917801491506903E-6]), a=1.2,
v=0.25)
     $ 11 "Catch"
     set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
     movel(pose_trans(VisionSnapShot,
p[0.06655190336940718,0.36159757507425694,0.3497705931926853,-
0.5238887278007582,-3.131708888788737E-4,2.340877640757373E-4]), a=1.2,
v=0.04)
     # begin: URCap Program Node
     #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
     #   Type: Gripper
     $ 12 "Gripper Close (1)"
     if (connectivity_checked[0] != 1):
      if not(rq_set_sid(9, "1")):
       popup("Gripper 1 must be connected to run this program.", False, False, True)
      end
      connectivity_checked[0] = 1
     end
     if (status_checked[0] != 1):
      if not(rq_is_gripper_activated("1")):
```

```
        popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it
and run the program again.", False, False, True)
       end
       status_checked[0] = 1
      end
      if (current_speed[0] != 0):
       rq_set_speed_norm(0, "1")
       current_speed[0] = 0
      end
      if (current_force[0] != 0):
       rq_set_force_norm(0, "1")
       current_force[0] = 0
      end
      rq_set_pos_norm(100, "1")
      rq_go_to("1")
      rq_wait("1")
      # end: URCap Program Node
      $ 13 "aproachTop"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(pose_trans(VisionSnapShot,
p[0.0665768905383662,0.3167619319644588,0.27313477215099846,-
0.5236374783766486,-4.1066383609840315E-4,2.917801491506903E-6]),          a=1.2,
v=0.25)
      $ 14 "ObjectLocated≔ True "
      global ObjectLocated=  True
     end
     # Restore snapshot position
     VisionSnapShot  =  p[-0.00901427,  -0.162668,  0.565441,  2.8532,  -0.0132514,
0.00370997]

     #######VisionLocate node end########
     ##########################################

     # end: URCap Program Node
     $ 15 "If ObjectLocated≟ True "
     if (ObjectLocated ==  True  ):
      $ 16 "SaveZone"
      $ 17 "MoveL"
      $ 18 "SafeZoneTopV"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(p[.241604121155,  -.497912511629,  .048729158262,  2.329640736093,  -
.012515989989, -.000316394571], a=1.2, v=0.25, r=0.05)
      $ 19 "Drop"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(p[.241295738149,  -.561817068077,  -.011821699782,  2.329562407353,  -
.012431415029, -.000206867711], a=1.2, v=0.04)
      # begin: URCap Program Node
      #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
```

```
    #   Type: Gripper
    $ 20 "Gripper Open (1)"
    if (connectivity_checked[0] != 1):
     if not(rq_set_sid(9, "1")):
      popup("Gripper 1 must be connected to run this program.", False, False, True)
     end
     connectivity_checked[0] = 1
    end
    if (status_checked[0] != 1):
     if not(rq_is_gripper_activated("1")):
      popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it
and run the program again.", False, False, True)
     end
     status_checked[0] = 1
    end
    if (current_speed[0] != 100):
     rq_set_speed_norm(100, "1")
     current_speed[0] = 100
    end
    if (current_force[0] != 0):
     rq_set_force_norm(0, "1")
     current_force[0] = 0
    end
    rq_set_pos_norm(0, "1")
    rq_go_to("1")
    rq_wait("1")
    # end: URCap Program Node
    $ 21 "SafeZoneTop70"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.242070374789,  -.498063447871,  .048856650245,  -2.001675971322,  -
1.063679027716, 1.725935616853], a=1.2, v=0.25, r=0.05)
    $ 22 "Catch70"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.241718765977,  -.560066125973,  -.009818850878,  -2.001883365460,  -
1.063513868113, 1.725929647872], a=1.2, v=0.04)
    # begin: URCap Program Node
    #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
    #   Type: Gripper
    $ 23 "Gripper Close (1)"
    if (connectivity_checked[0] != 1):
     if not(rq_set_sid(9, "1")):
      popup("Gripper 1 must be connected to run this program.", False, False, True)
     end
     connectivity_checked[0] = 1
    end
    if (status_checked[0] != 1):
     if not(rq_is_gripper_activated("1")):
```

```
        popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it and run the program again.", False, False, True)
       end
       status_checked[0] = 1
      end
      if (current_speed[0] != 0):
       rq_set_speed_norm(0, "1")
       current_speed[0] = 0
      end
      if (current_force[0] != 0):
       rq_set_force_norm(0, "1")
       current_force[0] = 0
      end
      rq_set_pos_norm(100, "1")
      rq_go_to("1")
      rq_wait("1")
      # end: URCap Program Node
      $ 24 "SafeZoneTop70"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(p[.242070374789, -.498063447871, .048856650245, -2.001675971322, -1.063679027716, 1.725935616853], a=1.2, v=0.25, r=0.05)
      $ 25 "AproachConveyor"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(p[-.322372708803, -.248185406625, .385233945263, -1.999369633337, -1.067645585119, 1.724779936665], a=1.2, v=0.25, r=0.05)
      $ 26 "AproachConveyor"
      $ 27 "MoveL"
      $ 28 "ZoneA"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(pose_trans(refConveyorbelt, p[-0.0978825201866439,0.05815607265858996,-0.21252699756521026,-1.941221458584907E-4,0.8130918147741518,3.0344590017637856]), a=1.2, v=0.1)
      $ 29 "AproachA"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(pose_trans(refConveyorbelt, p[-0.0952140018922576,0.11467797461102647,-0.1043851018572895,-0.012980343220842853,0.6633515206565507,3.062854138295442]), a=1.2, v=0.1, r=0.016999999999999998)
      $ 30 "DropA"
      set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
      movel(pose_trans(refConveyorbelt, p[-0.09543183025100865,0.10163513316927053,-0.09157221066065957,-0.013001489845497495,0.6634332438687037,3.0629484098587283]), a=1.2, v=0.02)
      # begin: URCap Program Node
      #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
      #   Type: Gripper
      $ 31 "Gripper Open (1)"
      if (connectivity_checked[0] != 1):
```

```
      if not(rq_set_sid(9, "1")):
       popup("Gripper 1 must be connected to run this program.", False, False, True)
      end
      connectivity_checked[0] = 1
     end
     if (status_checked[0] != 1):
      if not(rq_is_gripper_activated("1")):
       popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it
and run the program again.", False, False, True)
      end
      status_checked[0] = 1
     end
     if (current_speed[0] != 100):
      rq_set_speed_norm(100, "1")
      current_speed[0] = 100
     end
     if (current_force[0] != 0):
      rq_set_force_norm(0, "1")
      current_force[0] = 0
     end
     rq_set_pos_norm(0, "1")
     rq_go_to("1")
     rq_wait("1")
     # end: URCap Program Node
     $ 32 "AproachA"
     set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
     movel(pose_trans(refConveyorbelt,                                                p[-
0.0952140018922576,0.11467797461102647,-0.1043851018572895,-
0.012980343220842853,0.6633515206565507,3.062854138295442]),   a=1.2,   v=0.1,
r=0.016999999999999998)
     $ 33 "ZoneA"
     set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
     movel(pose_trans(refConveyorbelt,                                                p[-
0.0978825201866439,0.05815607265858996,-0.21252699756521026,-
1.941221458584907E-4,0.8130918147741518,3.0344590017637856]),   a=1.2,   v=0.1,
r=0.05)
     $ 34 "PieceLeaved≔ True "
     global PieceLeaved=  True
     $ 35 "ObjectLocated≔ False "
     global ObjectLocated=  False
    end
   else:
    $ 36 "ElseIf digital_in[2]≟ False "
    if (get_standard_digital_in(2) ==   False  ):
     $ 37 "MoveJ"
     $ 38 "ZoneB"
```

```
movej([0.21366808985936658, -2.2651272719940927, 2.512036184308844, -3.205293603353509, -1.6005141859864445, 0.8290921876002124], a=1.3962634015954636, v=1.0471975511965976, r=0.05)
$ 39 "MoveL"
$ 40 "AproachB"
movel([0.4930294697611215, -1.5908062045123827, 2.3573770233298403, -3.9143999821227307, -2.060659464847377, 0.8051030989469907], a=1.2, v=0.1, r=0.009999999999999998)
$ 41 "GrabB"
movel([0.5192380937514844, -1.5559004423823168, 2.3509764267055147, -3.94303548942343, -2.086723229433974, 0.8050748368950035], a=1.2, v=0.01)
# begin: URCap Program Node
#   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
#   Type: Gripper
$ 42 "Gripper Close (1)"
if (connectivity_checked[0] != 1):
 if not(rq_set_sid(9, "1")):
  popup("Gripper 1 must be connected to run this program.", False, False, True)
 end
 connectivity_checked[0] = 1
end
if (status_checked[0] != 1):
 if not(rq_is_gripper_activated("1")):
  popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it and run the program again.", False, False, True)
 end
 status_checked[0] = 1
end
if (current_speed[0] != 0):
 rq_set_speed_norm(0, "1")
 current_speed[0] = 0
end
if (current_force[0] != 0):
 rq_set_force_norm(0, "1")
 current_force[0] = 0
end
rq_set_pos_norm(100, "1")
rq_go_to("1")
rq_wait("1")
# end: URCap Program Node
$ 43 "AproachB"
movel([0.4930294697611215, -1.5908062045123827, 2.3573770233298403, -3.9143999821227307, -2.060659464847377, 0.8051030989469907], a=1.2, v=0.1, r=0.009999999999999998)
$ 44 "ZoneB"
movel([0.21366808985936658, -2.2651272719940927, 2.512036184308844, -3.205293603353509, -1.6005141859864445, 0.8290921876002124], a=1.2, v=0.1, r=0.05)
```

```
    $ 45 "SaveZoneReturn"
    $ 46 "MoveL"
    $ 47 "Home_var"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(pose_trans(p[0.0,0.0,0.0,0.0,0.0,0.0], Home), a=1.2, v=0.25, r=0.05)
    $ 48 "SafeZoneTopH"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.241670897778, -.497982517345, .048749019665, -1.781774373584, -
.701600745795, 1.768721387866], a=1.2, v=0.25, r=0.05)
    $ 49 "MoveL"
    $ 50 "DropH"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.241312357949, -.560234987318, -.010331072748, -1.781691334602, -
.701545118293, 1.768584140124], a=1.2, v=0.04)
    # begin: URCap Program Node
    #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
    #   Type: Gripper
    $ 51 "Gripper Open (1)"
    if (connectivity_checked[0] != 1):
     if not(rq_set_sid(9, "1")):
      popup("Gripper 1 must be connected to run this program.", False, False, True)
     end
     connectivity_checked[0] = 1
    end
    if (status_checked[0] != 1):
     if not(rq_is_gripper_activated("1")):
      popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it
and run the program again.", False, False, True)
     end
     status_checked[0] = 1
    end
    if (current_speed[0] != 0):
     rq_set_speed_norm(0, "1")
     current_speed[0] = 0
    end
    if (current_force[0] != 0):
     rq_set_force_norm(0, "1")
     current_force[0] = 0
    end
    rq_set_pos_norm(0, "1")
    rq_go_to("1")
    rq_wait("1")
    # end: URCap Program Node
    $ 52 "SafeZoneTopV"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.241604121155, -.497912511629, .048729158262, 2.329640736093, -
.012515989989, -.000316394571], a=1.2, v=0.25, r=0.05)
    $ 53 "Drop"
```

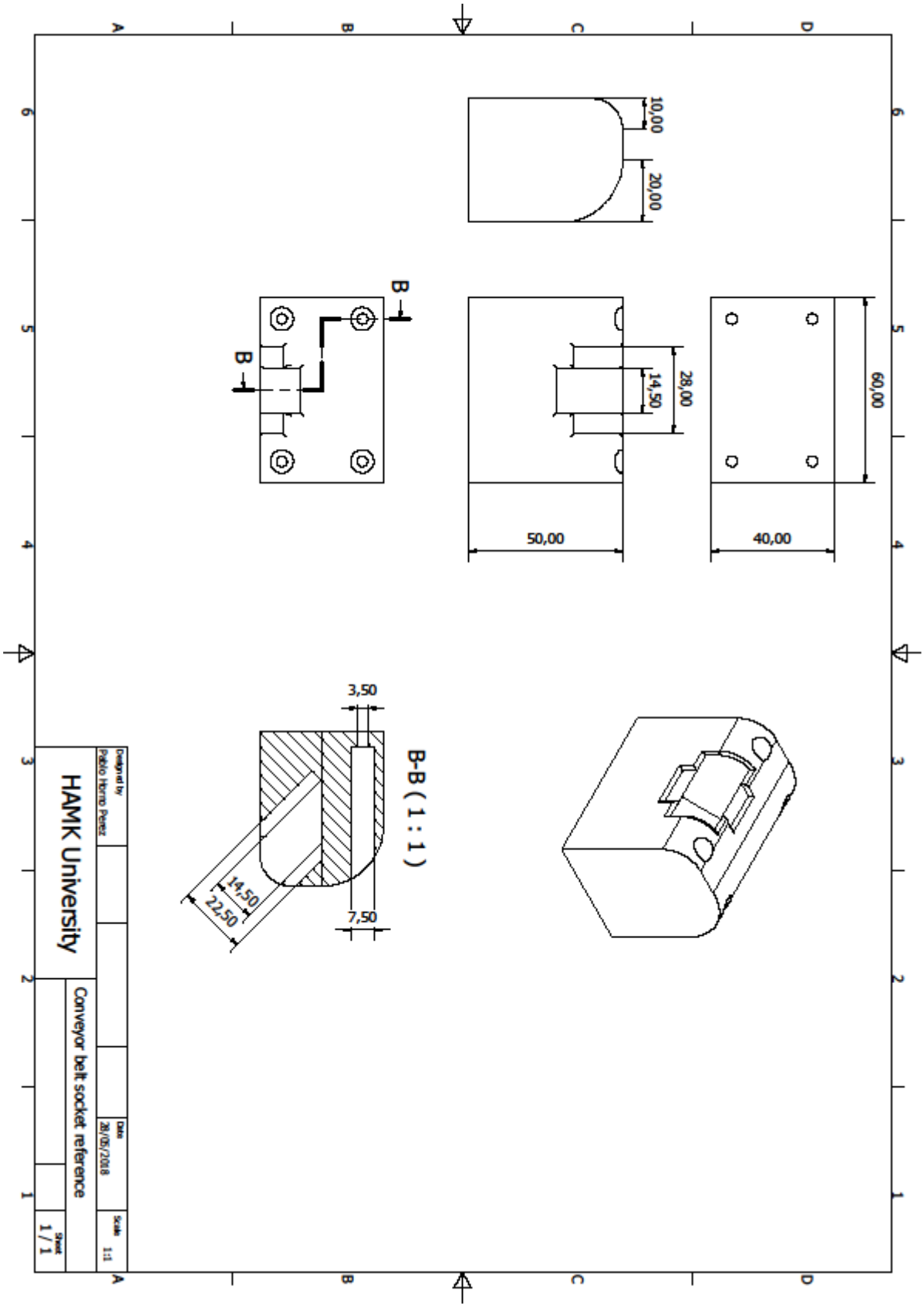
```
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.241295738149, -.561817068077, -.011821699782, 2.329562407353, -
.012431415029, -.000206867711], a=1.2, v=0.04)
    # begin: URCap Program Node
    #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
    #   Type: Gripper
    $ 54 "Gripper Close (1)"
    if (connectivity_checked[0] != 1):
     if not(rq_set_sid(9, "1")):
      popup("Gripper 1 must be connected to run this program.", False, False, True)
     end
     connectivity_checked[0] = 1
    end
    if (status_checked[0] != 1):
     if not(rq_is_gripper_activated("1")):
      popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it
and run the program again.", False, False, True)
     end
     status_checked[0] = 1
    end
    if (current_speed[0] != 0):
     rq_set_speed_norm(0, "1")
     current_speed[0] = 0
    end
    if (current_force[0] != 60):
     rq_set_force_norm(60, "1")
     current_force[0] = 60
    end
    rq_set_pos_norm(100, "1")
    rq_go_to("1")
    rq_wait("1")
    # end: URCap Program Node
    $ 55 "SafeZoneTopV"
    set_tcp(p[0.0,0.0,0.1625,0.0,0.0,0.0])
    movel(p[.241604121155, -.497912511629, .048729158262, 2.329640736093, -
.012515989989, -.000316394571], a=1.2, v=0.25, r=0.01)
    $ 56 "MoveJ"
    $ 57 "StackTopJ"
    movej([2.530723153855002, -1.8760585813040418, 2.5560740937700857, -
2.794439648144816, -2.2133683847120675, 3.922719881583569],
a=1.3962634015954636, v=1.0471975511965976)
    $ 58 "Wait: 1.0"
    sleep(1.0)
    $ 59 "MoveL"
    $ 60 "StackTop"
    movel([2.5122990401594407, -1.851969164000077, 2.5633000706659375, -
2.837342233930098, -2.203544481027687, 3.903164206470141], a=0.5, v=0.025)
    $ 61 "MaxF:=0"
```

```
  global MaxF=0
  $ 62 "If F<50"
  global thread_flag_62=0
  thread Thread_if_62():
   $ 63 "StackBottom"
   movel([2.0295354801377243, -0.7847142230232222, 2.291551654382868, -
3.834451383200472, -1.898177188515426, 3.4736348951154596], a=0.5, v=0.025)
   thread_flag_62 = 1
  end
  if (F<50):
   global thread_handler_62=run Thread_if_62()
   while (thread_flag_62 == 0):
    if not(F<50):
     kill thread_handler_62
     thread_flag_62 = 2
    else:
     sync()
    end
   end
  else:
   thread_flag_62 = 2
  end
  $ 64 "return_2mm"
  movel(pose_add(get_forward_kin(), pose_sub(p[.272468337773, -.482207083355,
-.103520321286, 2.329712558507, -.012419453146, -.000774961485],
p[.272461793546, -.483707412283, -.104961706486, 2.329727564861, -
.012294284317, -.000785040411])), a=0.5, v=0.025)
  # begin: URCap Program Node
  #   Source: Robotiq_2-Finger_Adaptive_Gripper, 1.1.3, Robotiq Inc.
  #   Type: Gripper
  $ 65 "Gripper Open (1)"
  if (connectivity_checked[0] != 1):
   if not(rq_set_sid(9, "1")):
    popup("Gripper 1 must be connected to run this program.", False, False, True)
   end
   connectivity_checked[0] = 1
  end
  if (status_checked[0] != 1):
   if not(rq_is_gripper_activated("1")):
    popup("Gripper 1 is not activated. Go to Installaton tab > Gripper to activate it
and run the program again.", False, False, True)
   end
   status_checked[0] = 1
  end
  if (current_speed[0] != 0):
   rq_set_speed_norm(0, "1")
   current_speed[0] = 0
  end
```

```
      if (current_force[0] != 0):
        rq_set_force_norm(0, "1")
        current_force[0] = 0
      end
      rq_set_pos_norm(0, "1")
      rq_go_to("1")
      rq_wait("1")
      # end: URCap Program Node
      $ 66 "MoveL"
      $ 67 "StackTop"
      movel([2.5122990401594407, -1.851969164000077, 2.5633000706659375, -
2.837342233930098, -2.203544481027687, 3.903164206470141], a=1.2, v=0.25)
    end
   end
  end
end
```

Appendix 2

APPENDIX HEADING


10,00
20,00
B
B
28,00
14,50
60,00
50,00
40,00
3,50
B-B (1 : 1)
14,50
22,50
7,50
HAMK University
Designed by
Pablo Herrro Perez
Conveyor belt socket reference
Date
28/05/2018
Scale
1:1
Sheet
1 / 1