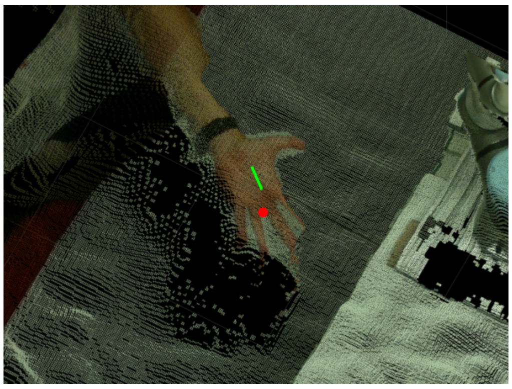


Vision based collaborative task planning and execution with a UR3 robotic manipulator

Tomás Moreira Furtado Carvalho Fernandes

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. Mário António da Silva Neves Ramalho

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Mário António da Silva Neves Ramalho
Member of the Committee: Prof. João Carlos Prata dos Reis

December 2021

Acknowledgments

First, I would like to thank my supervisor professor Mário Ramalho for his guidance, support and availability throughout this year to answer all my questions.

I would also like to thank Mr. Raposeiro for designing and printing the part necessary to secure the camera to the ceiling and Engineer Camilo for the help setting up the UR3 robot and connecting it to the Internet.

My family, who always backed my decisions and motivated me to do better and for all the support and encouragement during this period.

To my friends from high school, Hugo, Miguel and João, a special thank you for always being by my side all these years, for the good times and the bad, which we always overcame together.

I could not forget about the PSEM team where I spent most of my time in Técnico and all my teammates during those amazing years. A big part of what I know today I learned with you, working through endless days and sleepless nights. Special mention to Fred, Chico, Gonçalo, Catarina, Rita, Henrique, PD, Dias, Pimenta and Mário who made me feel very welcome when I entered the team. Also an individual acknowledgement to Miguel, Ribeira, Lino, Mendes, Inês, Diogo, Ferreira, Adegas, Tomás and Freira who were by my side through the most challenging times. Professor João Dias also deserves a recognition for all the support and experience shared with us, particularly during the competitions in England.

Last but not least, I want to thank Pedro and Fábio, who have been with me in IST from the start.

Resumo

Neste trabalho procurou-se desenvolver uma interface colaborativa entre uma bancada de automação e um operador. Pretendeu-se que o conjunto seja capaz de recolher e entregar peças a um operador quando este está numa zona especifica do espaço de trabalho. Para isso foi utilizada uma câmera profundidade Intel D435 e um robô colaborativo UR3. O algoritmo desenvolvido utiliza imagens a cores e informação de profundidade fornecidas pela câmera para gerar um caminho a ser percorrido pelo robô. O algoritmo mostrou uma correta segmentação das mãos do operador e das peças a serem transportadas, exceto em casos extremos e consequentemente pouco usuais em funcionamento normal. O robô mostrou um seguimento correto aos caminhos gerados, com exceção das mudanças de direção mais abruptas e na mudança entre controlo no espaço das juntas e controlo no espaço da ferramenta, onde a harmonização da trajetória não é efetuada tal como estipulado. Obtiveram-se disparidades entre as posições reportadas pela câmera e pelo robô inferiores a 15 mm no plano XY e 40 mm em profundidade. Estes erros devem-se ao método de transformação de coordenadas da câmera para coordenadas do robô e a erros intrínsecos da câmera. A colaboração foi conseguida, o algoritmo mostrou-se apto a responder em tempo real aos pedidos do utilizador e os erros de posicionamento da ferramenta revelaram-se aceitáveis para as tarefas propostas.

Palavras-chave: robótica colaborativa, visão computacional, câmera de profundidade, handover

Abstract

This thesis aimed at developing a collaborative interface between an automation bench and an operator. It is intended for the system to be able to collect and deliver objects from the hands of the operator. An Intel D435 depth camera an a UR3 collaborative robot were used. The algorithm developed employs the colour images and depth information from the camera to generate a path for the robot to follow. The algorithm was able to correctly segment the hands of the operator and the objects to be carried in normal operation conditions. The robot followed correctly the paths generated, except when sharp changes in direction occur or when transitioning from control on joint space to control in tool space. In these situations, the blend radii selected was not respected. The discrepancy between the positions reported by the camera and the robot were under 15 mm in the XY plane and 40 mm in terms of depth. These errors are due to the camera and the transformation between camera coordinates and robot coordinates. Collaboration was achieved since the algorithm reacted in real time to demands from the operator and the tool deviation observed was acceptable for the task at hand.

Keywords: collaborative robotics, computer vision, depth camera, hand-over

Contents

Acknowledgments . . . iii
Resumo . . . v
Abstract . . . vii
List of Tables . . . xi
List of Figures . . . xiii
Nomenclature . . . xv

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Topic Overview . . . 1
1.2.1 Collaborative robotics . . . 1
1.2.2 Depth estimation . . . 2
1.2.3 Real-time robot manipulation . . . 3
1.3 Objectives . . . 4
1.4 Thesis Outline . . . 4

2 Background **5**
2.1 Robotics . . . 5
2.1.1 Transformation matrix . . . 5
2.1.2 Rotation matrix . . . 6
2.1.3 Euler angles . . . 6
2.1.4 Axis-angle representation . . . 6
2.1.5 Unit quaternion . . . 7
2.1.6 Convert Euler angles to axis-angle . . . 7
2.1.7 Rotation from one vector into another . . . 8
2.2 Computer Vision . . . 8
2.2.1 Colour spaces . . . 8

3 Implementation **11**
3.1 UR3e Robot . . . 13
3.1.1 Third party communication . . . 14
3.1.2 URSim . . . 14

3.2 RG2 Gripper . . . 15
3.3 Intel RealSense D435 . . . 16
3.3.1 Camera calibration . . . 17
3.3.2 Camera output . . . 19
3.4 Software Architecture . . . 20
3.4.1 Camera . . . 21
3.4.2 Robot . . . 26
3.4.3 Main . . . 29

4 Results **31**
4.1 Segmentation results . . . 31
4.1.1 Hand segmentation . . . 31
4.1.2 Segmentation of objects in the hand . . . 32
4.1.3 Segmentation of objects on the table . . . 32
4.1.4 Computation times . . . 33
4.2 Accuracy testing . . . 34
4.2.1 Experimental setup . . . 34
4.2.2 Preliminary results . . . 34
4.2.3 Influence of the calibration . . . 35
4.2.4 Calibration Alternatives . . . 36
4.2.5 Camera depth error . . . 37
4.2.6 Camera repeatability . . . 38
4.2.7 Robot repeatability . . . 39
4.3 Paths . . . 40
4.3.1 Approach the object . . . 40
4.3.2 Carry an object to a specified position . . . 41
4.3.3 Transport of an object to a surface . . . 45
4.3.4 Return to rest position . . . 47
4.4 Collaboration Example . . . 49

5 Conclusions **55**
5.1 Achievements . . . 55
5.2 Conclusions . . . 56
5.3 Future Work . . . 56

Bibliography **57**

List of Tables

3.1 Main characteristics of UR3e robot 13
3.2 Mass and center of mass position of RG2 gripper 16
3.3 Outcome of the camera calibration 18
3.4 Results of the camera calibration with MATLAB 19

4.1 Computation times of Camera class methods 33
4.2 Mass and center of mass position for the repeatability test 39
4.3 Robot repeatability test result 40

List of Figures

3.1 Hardware used and its connection 11
3.2 Experimental setup 12
3.3 Draft of the workspace: top view (bottom) and operator’s perspective (top) 13
3.4 Quick changer and RG2 gripper 15
3.5 Intel RealSense D435 camera 16
3.6 Pattern used for calibration 17
3.7 Examples of camera output 20
3.8 Pointcloud obtained from combining depth map and RGB image 20
3.9 Illustration of **isPresent** method 23
3.10 Procedure for object segmentation 24
3.11 Centroid and orientation of objects 24
3.12 Procedure to segment the hand 25
3.13 Hand orientation and off-load location 25
3.14 Procedure to segment an object in the operator’s hand 26
3.15 Held object’s centroid and orientation 26
3.16 Examples of generated paths 29
3.17 Flowchart of the endless loop of the main algorithm 30

4.1 Examples of hand segmentation results 32
4.2 Examples of segmentation for objects in hand 32
4.3 Examples of object segmentation results 33
4.4 3D printed part used to assess accuracy 34
4.5 First accuracy results 35
4.6 Comparison of calibration results 36
4.7 Comparison of calibration methods 37
4.8 Depth error estimate 38
4.9 Camera repeatability results 38
4.10 Robot repeatability test result 39
4.11 Comparison of generated and actual object approach path 41
4.12 Influence of blend radius on path discrepancy 41
4.13 Montage from video of robot approaching object 42

4.14 Comparison of generated and actual deliver in place path 43
4.15 TCP speed for path of robot placing object in specified position 43
4.16 Montage from video of robot transporting an object to a specified position 44
4.17 TCP speed for path of robot placing object on the table 45
4.18 Comparison of generated and actual path of robot placing object on the table 45
4.19 Montage from video of robot placing object on the table 46
4.20 Comparison of generated and actual path of robot returning to rest position 47
4.21 Montage from video of robot returning to rest position 48
4.22 Operator changes hand position 50
4.23 Operator changes hand position to an unreachable position 51
4.24 Operator removes his hand from the workspace 52
4.25 Operator changes receiving hand 53
4.26 Robot placing finished product on the table 54

Nomenclature

HRI	Human-robot interaction
HSV	Hue, Saturation, Value
IFR	International Federation of Robotics
IR	Infrared
UR	Universal Robots

Greek symbols

$\alpha, \beta, \gamma, \theta$	Angles

Roman symbols

$\boldsymbol{I}$	Identity matrix
$\boldsymbol{p}_j^i$	Translation vector from frame *i* to frame *j*
$\boldsymbol{R}_j^i$	Rotation matrix from frame *i* to frame *j*
$\boldsymbol{T}_j^i$	Transformation matrix from frame *i* to frame *j*
$\mathbf{K_u}$	Cross product matrix of vector **u**
$\mathbf{t}$	Binarization threshold
$\mathbf{u}, \mathbf{n}, \mathbf{v}$	Vectors
J, Q, S	Quaternions
q_w	Scalar component of quaternion Q
q_x, q_y, q_z	Vectorial components of quaternion Q
u_x, u_y, u_z	Cartesian components of vector **u**

Chapter 1

Introduction

1.1 Motivation

The first industrial robot called UNIMATE was introduced in 1961 [1]. Since then, the majority of robots are usually confined in a cage where humans are not allowed to enter to prevent accidents. More recently, collaborative robots were introduced. They differ because they possess caracteristics that make them more suited to operate outside of cages. This allows the robot to operate together with people and in a dynamic environment, since the collision detection prevents the robot from seriously injuring someone or damaging itself when contacting with other equipment.

By adding other devices into the environment such as cameras, LIDAR or torque sensors, it is possible to achieve collaboration in a wider sense. This means having the robot and the operator working together. This cooperation can be very useful in situation where eliminating human presence can increase safety or quality. A prime example would be an operating rooms where a robot could act as a instrumentalist. By replacing a nurse and consequently lowering the risk of contamination, it would also liberate the nurse to execute more complex tasks. This concept can be extended to other contexts such as biosafety laboratories and electronics or composite cleanrooms.

1.2 Topic Overview

1.2.1 Collaborative robotics

Cobots, short for collaborative robots, are a particular kind of robots designed to interact directly with humans. They were developed to get rid of physical barriers between humans and robots on industrial environments, allowing for a greater production performance. According to the International Federation of Robotics (IFR) [2], the standard to be considered for these applications is the ISO 10218. It frames the safety of Human-robot interaction (HRI) by stating risk reduction approaches such as force, speed and power limitations for the robots.

Because the presence of robots in their workspace increases the stress operators are subject to, steps should be taken to reduce it as much as possible. This includes placing the robot as far as possible from the operator, reducing the maximum end-effector speed and notifying the operator whenever the robot is about to move [3].

There have been defined four types of HRI [4]:

- **Coexistance**, where the fences are removed but Human and robot don't share a workspace.
- **Synchronized**, where the Human and robot share a workspace but only one of them can be present at any time. The robot should stop in case of violation of the workspace by an operator;
- **Cooperation**, where both Human and robot share a workspace and can be present at the same time, but they work on a part sequentially;
- **Collaboration**, where Human and robot share a workspace and work together on the same product. The robots reacts to the actions of the operator in real time.

Universal Robot (UR) is one of the companies in developing cobots for industrial applications. It's robots are famous for achievements such as the UR5e robot who was the first robot to ring the New York Stock Exchange bell [5], and the UR3 who was capable of performing a Boeing 737's landing procedure [6].

1.2.2 Depth estimation

To estimate depth with one or more cameras, several approaches exist. Price used to be a great factor when deciding which technology to use for a given application. Today, these devices are affordable and the nature of the applications is the sole factor of choice.

Time of flight cameras measure the distance by emitting a modulated light and measuring the reflection's delay. The Kinect V2 from Microsoft is an example of a popular camera of this type . The modern models rely on a single observation to generate the depth map, as opposed to older models who required multiple observations and thus were prone to artifacts due to motion in dynamic scenes. The main problem of this type of camera is multipath interference, which happens when the modulated light arrives to the sensor through more than one path [7]. Another disadvantage is that the presence of other cameras of the same type can harm the results obtained [8].

Structured light cameras have a light projector and a camera. The camera captures an image of the scene with a pattern coming from the projector on it. The distortion of the pattern can be used to compute the depth map using a process called triangulation. These cameras don't suffer from multipath interferences, but because the pattern projected must be known in advance, they can have interferences if two or more projectors overlap in part of the image [9].

Stereo vision uses two cameras and a triangulation process. Pixels in both images are matched and the position disparity is used to compute depth. However, the pixel matching in low texture regions is difficult which lead to poor depth maps [10]. To solve this problem, active stereo was introduced. Active stereo measures depth from a pair of cameras where a pattern is projected. This artificially adds texture to the scene, improving the pixel matching. The main advantage of this method is that the pattern doesn't need to be known beforehand and therefore there is no problem with multiple cameras of this type interfering with each other.

Estimating the depth from images can also be accomplished with a single camera. If the scene is static, a single camera can be used to take two pictures of the same scene from different perspectives, provided the translation and rotation between the shots is known [11]. If the scene is not static, the process is more complex as there can be several independently moving bodies. In that case, it is necessary to use optical flow in combination with multiple view geometry [12].

A system of mirrors can also be used to mimic the presence of two cameras when only one is available [13]. This system uses four mirrors arranged such as the image obtained can be partitioned in half to obtain two different points of view from the same scene.

In addition, machine learning with supervised learning can be employed [14]. The training requires images with their corresponding depth maps to be provided.

1.2.3 Real-time robot manipulation

The idea of manipulating a robot in real time using visual cues is not recent. It is used for robot manipulators and mobile applications. Sports is one of the fields where cameras have been used in combination with robots, such as ping-pong [15] to track the ball or football [16] to identify the ball, the other players and the current position on the pitch. Tasks can be relatively simple, such as catching a ball with a cup [17], that a robot can easily accomplish due to it's speed and accuracy when a correct estimate of the ball's position is provided. The remote operation of a robot to map and evaluate the state of Chernobyl's Unit 4 [18] is another example of what can be achieved when combining computer vision and robotics.

Vision systems have been employed with UR robots in a multitude of applications. In [19], three cameras are used to detect markers that simulate an obstacle. The robot is then steered away from this obstacle using the method of artificial potentials. In [20], two Kinect cameras are used to estimate the pose of the arm of the operator to allow a robot to perform a hand-over application. The skeleton of the operator is directly extracted from the Kinect, therefore no segmenting operation is needed. Cameras are also be used directly attached to the robot as seen in [21]. This camera placement ensures the tool is captured by the camera at all times and usually leads to better accuracy, however the robot has to step back when a general perspective is needed.

1.3 Objectives

The objective of this work is to implement a collaborative interface between an automation bench and a user. For that a UR3e robot will be used supported by visual cues from a depth camera. It will be necessary to verify the feasibility of this approach in terms of hardware compatibility and environment condition, particularly in terms of lighting. It is intended for the overall system to be accurate enough to handle objects with a length of 30 mm and a height of 10 mm. Also important is the fact that this system will have to be able to answer in real time to inputs from the user, therefore short computational times are required.

1.4 Thesis Outline

In Chapter 2 the concepts of robotics and computer vision used are briefly explained. Chapter 3 describes the hardware used and its arrangement on the workstation. It also explains in depth the developed interface between the robot and the depth camera. Chapter 4 has a review of the algorithms developed, including experiments accomplished to improve the solution proposed. Chapter 5 presents conclusions and some suggestions in how the work done can be improved.

Chapter 2

Background

2.1 Robotics

Industrial robots are generally modelled as a sequence of links considered to be rigid bodies. Links have relative motion from one another by mean of mechanical joints that can be of two types, prismatic or revolute. To describe their position and orientation, we attach to each link an orthonormal reference frame. Among others, the Denavit-Hartenberg convention can be used to express their relative position and orientation [22]. This convention establishes a set of rules regarding how the frames are placed so that there is always four parameters that define their pose relative to the previous reference frame. These parameters can be used to construct a matrix called transformation matrix, which relate the poses of a reference frame to another. Knowing each of the links position and orientation depending on the joint angles is called direct kinematics. The inverse procedure, finding the joint angles given a desired pose of the end effector, is called inverse kinematics.

2.1.1 Transformation matrix

As transformation matrix $\boldsymbol{T}$ is a 4x4 matrix used to perform transformations from one frame to another in three dimentional space. The matrix $\boldsymbol{T}_0^1$, expressing the transformation from reference frame 0 to reference frame 1 is most often of the form:

$$\boldsymbol{T}_1^0 = \begin{bmatrix} \boldsymbol{R}_1^0 & \boldsymbol{p}_1^0 \\ \boldsymbol{0}^\top & 1 \end{bmatrix}, \tag{2.1}$$

where $\boldsymbol{R}_0^1$ is the rotation matrix from frame 0 to frame 1 and the vector $\boldsymbol{p}_0^1$ is the translation between the two reference frames expressed in the coordinates of frame 0. The last line is all zeros except the last column which is a one. If the frames have the same origin, then $\boldsymbol{p}_1^0$ is a null vector. Transformation matrices can be multiplied sequentially to express new transformations. In general, we have:

$$\boldsymbol{T}_n^0 = \prod_{k=1}^{n} \boldsymbol{T}_k^{k-1}. \tag{2.2}$$

2.1.2 Rotation matrix

A matrix that describes the change in orientation between two reference frames is called a rotation matrix. The matrix representing the change from frame 0 to frame 1 can be written as $\boldsymbol{R}_1^0$. Its columns are the direction cosines of the axis of frame 1 expressed in frame 0.

Because the reference frames are orthonormal, rotation matrices are orthonormal matrices. This means both columns and rows are orthonormal vectors, which lead to:

$$\boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}, \tag{2.3}$$

where $\boldsymbol{I}$ is the identity matrix. As a result:

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^\top. \tag{2.4}$$

Therefore, to reverse a given rotation, it suffices to re-multiply by the transpose of the rotation matrix used previously due to the transpose being equal to the inverse.

2.1.3 Euler angles

Rotation matrices have nine parameters which are not independent. The minimal number of parameters to fully characterize a rotation is three, called Euler angles. There are twelve sets of Euler angles, the one used in this work is the ZYX combination. The intrinsic rotations convention was chosen, which means elemental rotations are performed around the axis of the moving body. Therefore, the first parameter is a rotation about the Z axis, then a rotation about the new Y axis and finally a rotation about the final X axis. To obtain the same final result when performing the rotations around the fixed original axes, it suffices to use the reverse set of Euler angles, XYZ in our case.

2.1.4 Axis-angle representation

In this representation, a rotation of and angle θ is made about the vector $\boldsymbol{u}$. This notation is not unique as a rotation of $-\theta$ about $-\boldsymbol{u}$ yields the same rotation. Another drawback of this notation is a singularity when θ is zero, because in that case the vector $\boldsymbol{u}$ is arbitrary.

It is possible to reduce this notation to a single vector by multiplying $\boldsymbol{u}$ by θ. The resulting vector will have the same direction as the original one and the rotation angle can be retrieved by calculating the magnitude of the resulting vector. This procedure also eliminates the non-uniqueness of the axis-angle representation.

2.1.5 Unit quaternion

Unit quaternion is a four parameter representation that can be separated into scalar and vectorial parts:

$$Q = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}, \tag{2.5}$$

where q_w is the scalar part and q_x, q_y, q_z form the vectorial part. A quaternion can be retrieved from the axis angle representation as

$$Q = \begin{bmatrix} cos(\theta/2) & \mathbf{u} sin(\theta/2) \end{bmatrix}. \tag{2.6}$$

Therefore, this representation is unique since a rotation of θ about the vector ***u*** yields the same quaternion as a rotation of θ about the vector $-$***u***. It also eliminates the singularity since the vectorial part of the quaternion is null for a rotation angle θ equal to zero. To add two rotations represented by two quaternions Q and J, we use the following formula:

$$S = \begin{bmatrix} q_w j_w - q_x j_x - q_y j_y - q_z j_z \\ q_w j_x + q_x j_w + q_y j_z - q_z j_y \\ q_w j_y - q_x j_z + q_y j_w + q_z j_x \\ q_w j_z + q_x j_y - q_y j_x + q_z j_w \end{bmatrix}^\top. \tag{2.7}$$

2.1.6 Convert Euler angles to axis-angle

There is no direct way to convert from the Euler angles to the axis-angle notation. One of the workarounds is to convert Euler angles into a quaternion, before being put into axis-angle form.

To convert from a set of Euler angles to a quaternion, we apply equation 2.6 to each of the elemental rotations, resulting in:

$$\begin{cases} Q_X = [cos(\alpha/2) \quad sin(\alpha/2) \quad 0 \quad 0], \\ Q_Y = [cos(\beta/2) \quad 0 \quad sin(\beta/2) \quad 0], \\ Q_Z = [cos(\gamma/2) \quad 0 \quad 0 \quad sin(\gamma/2)]. \end{cases} \tag{2.8}$$

Adding the three elemental rotations in the order ZYX according to equation 2.7 yelds:

$$Q_{ZXY} = \begin{bmatrix} cos(\alpha/2)cos(\beta/2)cos(\gamma/2) + sin(\alpha/2)sin(\beta/2)sin(\gamma/2) \\ sin(\alpha/2)cos(\beta/2)cos(\gamma/2) - cos(\alpha/2)sin(\beta/2)sin(\gamma/2) \\ cos(\alpha/2)sin(\beta/2)cos(\gamma/2) + sin(\alpha/2)cos(\beta/2)sin(\gamma/2) \\ cos(\alpha/2)cos(\beta/2)sin(\gamma/2) - sin(\alpha/2)sin(\beta/2)cos(\gamma/2) \end{bmatrix}^\top. \tag{2.9}$$

The axis-angle representation can be retrieved inverting equation 2.6:

$$\begin{cases} \theta = atan2(\left\| \begin{bmatrix} q_x & q_y & q_z \end{bmatrix} \right\|, \ q_w), \\ \mathbf{u} = \frac{\begin{bmatrix} q_x & q_y & q_z \end{bmatrix}}{\left\| \begin{bmatrix} q_x & q_y & q_z \end{bmatrix} \right\|}. \end{cases} \tag{2.10}$$

2.1.7 Rotation from one vector into another

Let **v** and **n** be two unit vectors three dimensional space space. A rotation from **n** to **v** happens around an axis that is orthogonal to both vectors. To compute a vector along that axis, we can use the cross product operator:

$$\begin{aligned} \mathbf{u} &= \mathbf{n} \times \mathbf{v} \\ &= \begin{bmatrix} n_y v_z - n_z v_y \\ n_z v_x - n_x v_z \\ n_x v_y - n_y v_x \end{bmatrix}, \end{aligned} \tag{2.11}$$

and the angle can be determined knowing that

$$\|\mathbf{n} \times \mathbf{v}\| = sin(\theta), \tag{2.12}$$

and

$$\mathbf{n} \cdot \mathbf{v} = cos(\theta), \tag{2.13}$$

which leads to:

$$\theta = atan2(\|\mathbf{n} \times \mathbf{v}\|,\ \mathbf{n} \cdot \mathbf{v}). \tag{2.14}$$

From the axis-angle notation, a rotation matrix can be using Rodrigues' rotation formula [23]:

$$\mathbf{R} = \mathbf{I} + sin(\theta)\mathbf{K_u} + (1 - cos(\theta))\mathbf{K_u}^2, \tag{2.15}$$

where **K** is the cross product matrix of **u** defined as

$$\mathbf{K_u} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}. \tag{2.16}$$

2.2 Computer Vision

Computer vision is the field of computer science that deals with digital image processing. In computers, images are stored as multidimensional tables whose entries are called pixels.

Computer vision is used in a wide range of applications, from a simple image enhancement to more complex tasks such as recognizing and tracking objects.

2.2.1 Colour spaces

Colour spaces are means of representing colours by one or more number. Each number is the value of a channel, whose allowed range of values is defined by its colour depth. Colour spaces with more than one channel almost always have the same bit depth across all channels.

RGB; RGB stands for red, green and blue. It is the most known colourspace and it is based on the way humans perceive light with red, green and blue cones. The majority of modern digital cameras captures images with red, green and blue receptors and stores them in RGB format.

RGB-D; similar to RGB but with an extra channel, distance. Each pixel contains the distance of that world point to the plane of the camera. This additional channel can have or not the same bit depth than the others

HSV; HSV stands for hue, saturation and value. Hue measures the dominant wavelength in the mixture of light waves. Saturation is the purity of the colour, the lower the saturation, the more grey is mixed into the colour. Value depicts how dark a colour is.

To convert from RGB format to HSV, where the hue, saturation and value range from 0 to 1, we use the formula

$$\begin{cases} V = \dfrac{max(R,G,B)}{255}; \\ S = \begin{cases} \dfrac{\Delta}{V} & \text{if } V > 0; \\ 0 & \text{if } V = 0; \end{cases} \\ H = \dfrac{1}{6} \times \begin{cases} \dfrac{G-B}{\Delta} \mod 6 & \text{if } R > G, B; \\ \dfrac{B-R}{\Delta} + 2 & \text{if } G > R, B; \\ \dfrac{R-G}{\Delta} + 4 & \text{if } B > R, G; \\ 0 & \text{if } V = 0 \text{ or } S = 0, \end{cases} \end{cases} \tag{2.17}$$

with

$$\Delta = max(R,G,B) - min(R,G,B). \tag{2.18}$$

Greyscale; are images with a single channel colourspace, represented in tones of gray. One of the formulas that have been defined to convert an RGB image to a greyscale image is:

$$\mathbf{G} = 0.299\mathbf{R} + 0.587\mathbf{G} + 0.114\mathbf{B}. \tag{2.19}$$

This formula was defined by the International Telecommunication Union and each colour has a different contribution due to the way humans perceive light [24].

Binary images; are images were pixels can only have one of two values. These values are usually zero and one. When representing these images, black and white are usually chosen to represent pixels with values zero and one respectively. Binary images can be obtained by thresholding an image using

$$\mathbf{B} = \begin{cases} 0\,, & if \mathbf{I} \leq \mathbf{t}, \\ 1\,, & if \mathbf{I} > \mathbf{t}, \end{cases} \tag{2.20}$$

where **I** is the original image and **B** the obtained binary image. The variable **t** is an array of thresholds, one for each of the image's channels. The thresholds cans be the same in the entire image or vary according to the image's region they are being applied to.

Chapter 3

Implementation

In this chapter, the arrangement of the elements in the workbench and how they connect to each other is described. Next, a description of the hardware is made. Finally, the developed software is presented.

Figure 3.1 gives an overview of the hardware used and how it is connected. The integration between the robot and the camera is made through a desktop computer running MATLAB version 2018b. The host computer is responsible for receiving the information from the depth camera and choosing accordingly the appropriate actions to be executed by the robot. It is connected to the camera by a USB cable. The robot is connected to a router by an Ethernet cable and therefore instructions can be sent from any part of the world through the Internet.

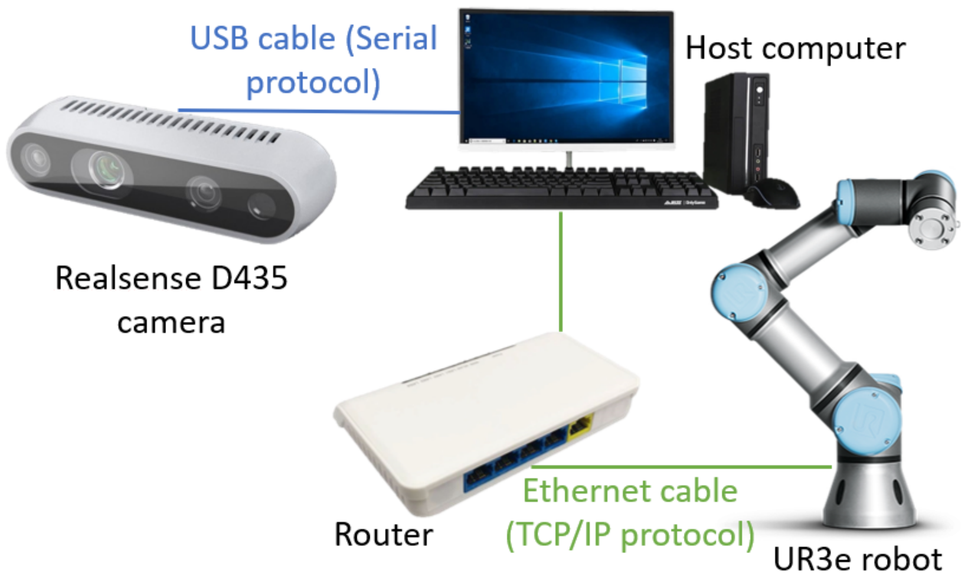


Figure 3.1: Hardware used and its connection

Figure 3.2 shows the layout of the experimental setup on the laboratory. The camera was attached to the ceiling so it can capture the entire workspace at all times. Since the goal is to use the robot as interface between the automation benches and the users, the robot is placed on the tabletop, adjacent to where the benches will be. This planned position is marked orange in the figure. A single robot will be used to for two automation benches, therefore the robot is located at the border between two workbenches.

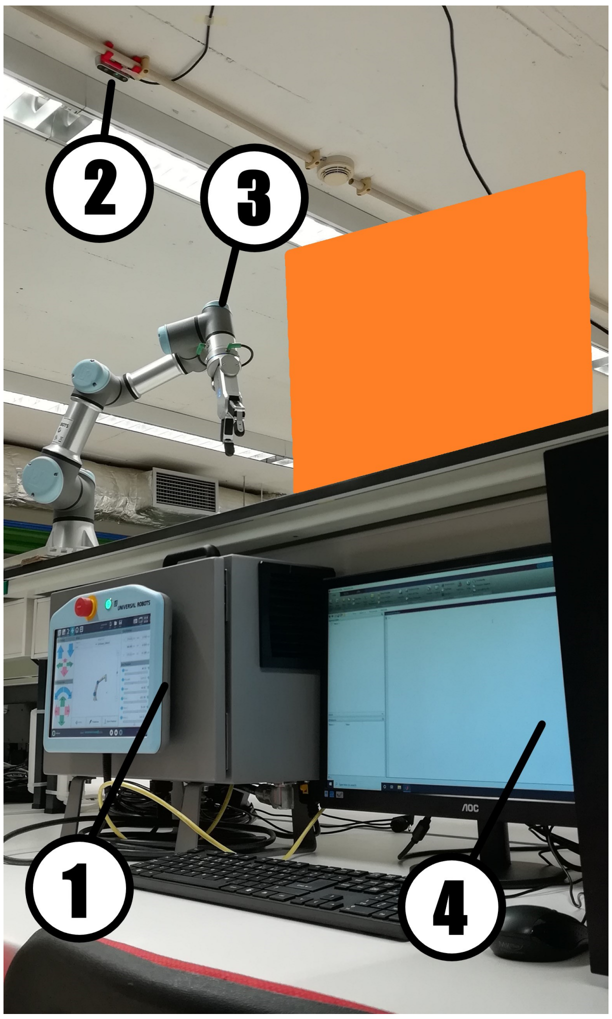


Figure 3.2: Experimental setup

1: Robot controller and teach pendant	2: D435 camera
3: UR3e robot and RG2 gripper	4: Host computer

Figure 3.3 is a sketch of the workspace. It displays the light positions in orange and approximate camera field of view in red.

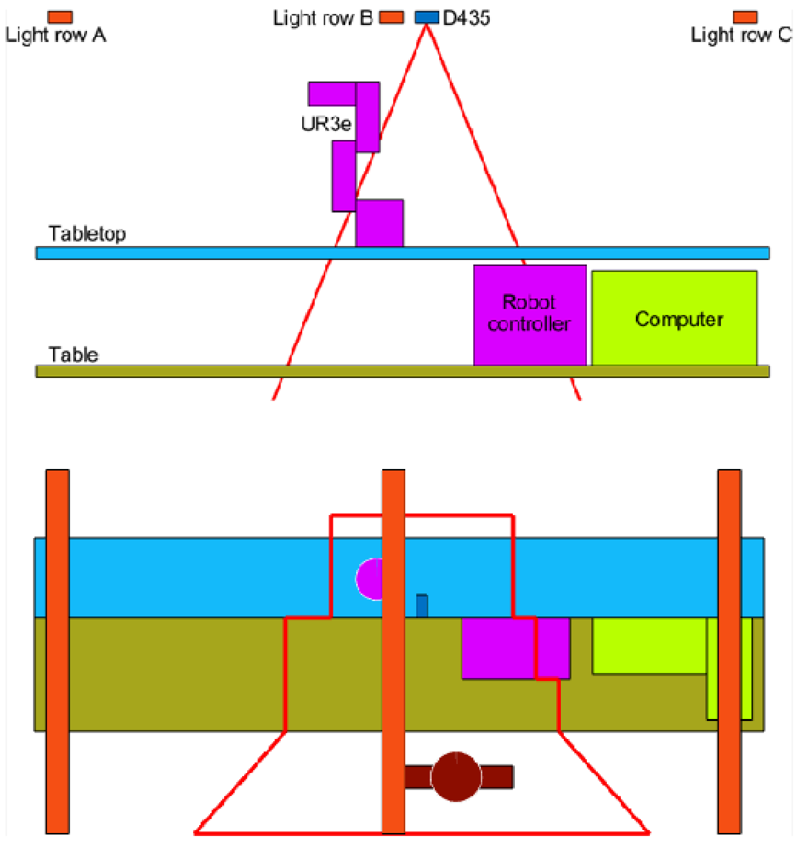


Figure 3.3: Draft of the workspace: top view (bottom) and operator's perspective (top)

3.1 UR3e Robot

The UR3e is a cobot produced by the danish company Universal Robots (UR), and is the smallest of their second generation of robots, designated e-Series. Table 3.1 lists its main attributes.

Table 3.1: Main characteristics of UR3e robot

Reach [mm]	Payload [kg]	Weight [kg]	Degrees of freedom	Type of joints	Repeatability [mm]
500	3.0	11.2	6	rotational	± 0.03

The robot uses a programming language from UR called URScript. Like every other language, it has variables, arithmetic operations and program control flow. The variables can be of eight types: *none*, *int*, *float*, *bool*, *array*, *matrix*, *pose* and *string*. The flow of control is ensured by *while* loops and *if* statements.

Besides the universal functions necessary to be considered a programming languages, URScript also has functions to manipulate and gather the robot's state. These include functions to define the tool's parameters, specify the gravitational acceleration in the robot's reference frame, move the robot in tool or joint space and calculate the inverse kinematics for a given end effector pose.

The teach pendant is the easiest way of programming the robot as it offers a graphical interface, designated PolyScope, that allows the user to program resorting to a Program Tree. The commands to be executed are added to the Program Tree as new nodes and the user programs entirely in pseudocode rather than programmings in URScript, the programming language used by UR robots. The URScript commands that are generated by the blocks of pseudocode in a program can be viewed by opening on a text editor the file of type *script* generated when saving said program. The teach pendant can also be used as a simulation environment, allowing the user to visualize how the robot will behave when executing a program before running it on the physical robot. This can be done by simply toggling a button on the footer on the interface.

3.1.1 Third party communication

The robot has a communication interface consisting of an Ethernet port is available to allow communication with other devices using MODBUS, PROFINET, Ethernet/IP or TCP/IP protocol. The connection can be made using an Ethernet cable to a router or directly to another device, that can be a computer, a PLC or another UR robot. All of these protocols allow to gather information about the robot such as speeds and positions of the joints, elbow and tool center point (TCP). The main difference between them is the inputs they can provide to the robot. MODBUS, PROFINET and Ethernet/IP can only be used to change registers, and therefore can only control the execution of programs through conditional statements or loops in previously written programs being run on the robot. TCP/IP has a broader range of applications and is also the protocol used for communication between the teach pendant and the control box of the robot.

TCP/IP offers different ways to interact with the robot, that acts as client or server depending on the interface used. The interface used was the The Real-time Client Interface (RTCI) as it is faster and more versatile than the others. It allows full programs to be sent to the robot to be executed, but also single commands. The RTCI also continuously provides clients with information about the robot's state, in lesser detail than other interfaces and with no possibility of customization but at a higher frequency of 500 Hz. Details about what information is sent can be found in [25].

In addition to the RTCI, the Dashboard Interface was also used. It allows communication between the teach pendant and a third party. Instructions can be sent to perform task that could have been performed using the teach pendant, including loading and executing programs that are already saved on the robot's memory, change safety configurations or check the robot's serial number. This interface does not continuously stream data, only answering queries from clients with a requested piece of information or with a message announcing the success or failure of a solicited task.

3.1.2 URSim

URSim is the simulation environment produced by Universal Robots. It was developed for Linux operating systems but can be executed in other operating systems resorting to a virtual machine. The software mimics the interface of the teach pendant and all the base functionalities are present. When used offline, only the simulation mode can be used, but when plugged by an Ethernet cable to a robot it can be used to control the physical robot as a substitute of the teach pendant.

The same communication protocols can be used between the URSim and a third party, the setup is similar to the procedure used for the physical robot. The simulation environment was used as a first verification step when generating new programs. However, some things are impossible to replicate in the simulator, such as the presence of obstacles. Therefore, a correct execution of the program in the simulation environment is no guarantee that the program will run smoothly on the physical robot.

3.2 RG2 Gripper

The RG2 is a gripper manufactured by OnRobot that can be attached to robots of fourteen brands, including the ones from UR. The gripper is not attached directly to the robot, but to a mounting bracket that is screwed to the robot's arm. OnRobot has three different mounting brackets that allow to attach one or two tools depending on the model. The simplest one was used in this work, it allows only one tool to be connected at the time and has no integrated force or torque sensor. The mounting bracket is connected electrically to the robot using an eight pin cable and to the tool through eight contact pads. The setup can be seen in Figure 3.4. A tool change happens in under a minute due to a quick release system in the gripper.

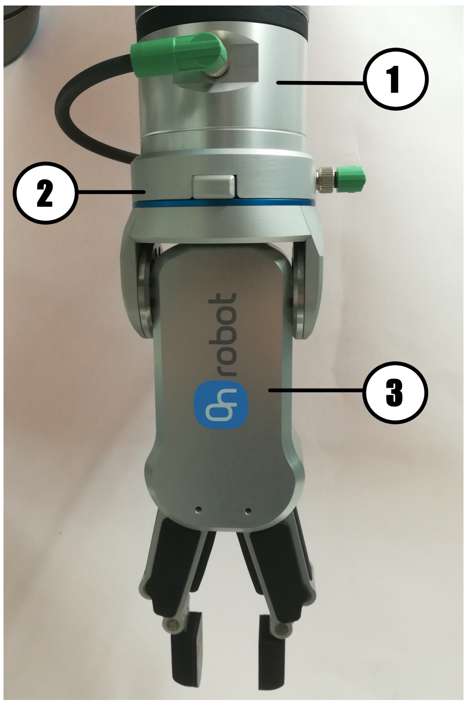


Figure 3.4: Quick changer and RG2 gripper

1: UR3e robot 2: Quick changer 3: RG2 gripper

The RG2 gripper can be powered at 12 V or 24 V, the latter being the recommended working voltage because it allows the gripper to operate at its full speed and force range. The fingers can be adjusted to internal or external grip and the maximum stroke is around 110 mm, depending on the grip direction and type of fingertips used. The gripping force can be set between 3 N and 40 N, the gripping speed is conditioned by the selected grip force and the gap between fingers.

The control of the RG2 gripper is simplified by installing an URCaps provided by OnRobots. URCaps are add-on software applications for PolyScope, which allow for easier integration of accessories with the robots. When a URCap is active, it adds new nodes that can be included in the programming tree. In the case of the RG2 gripper, it also adds a new tab in PolyScope's header that allows to open and close the gripper with a specified force, or set the gripper's opening to a certain width, without the need of a program. At a script level, URCaps are blocks of code at the begining of the program that define new variables and subroutines that assist the correct functioning of an accessory.

The tool increases the mass of the robot and therefore modifies its inertia. To cope with this change, the operator needs to introduce the new payload and center of gravity of the tool on Polyscope. The correct values can be read from a spreadsheet provided by OnRobots in which the user selects the mounting and tool used. Table 3.2 indicates the values indicated when selecting the quick changer mounting and the RG2 gripper.

Table 3.2: Mass and center of mass position of RG2 gripper

mass [kg]	center of mass [mm]		
	X	Y	Z
0.84	0	0	110.9

3.3 Intel RealSense D435

The RealSense D435 camera seen in Figure 3.5 is a depth camera from Intel, it connects to the host computer through a USB cable. It has two infrared (IR) cameras that constitute a stereo pair (first and third opening from the left), a RGB camera (rightmost opening) and an IR projector. The IR cameras are used at 848x480 resolution. This is the largest resolution possible and gives the most precise results. The RGB camera is used at 640x480 resolution because higher resolutions have lower frame rates and the increase in field of view would be fruitless, since those zones are not relevant for the task at hand.

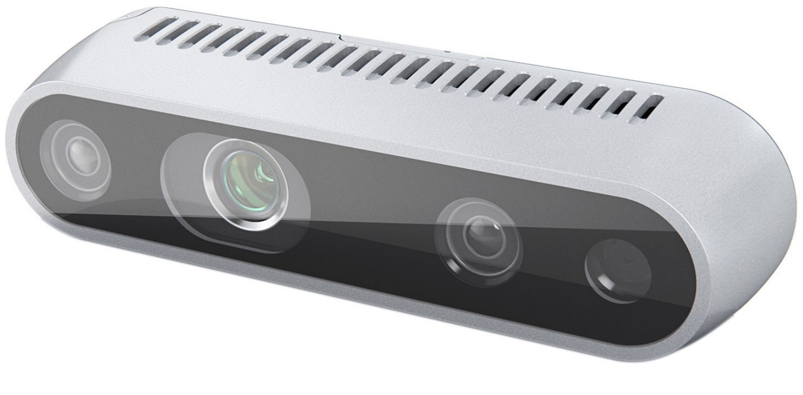


Figure 3.5: Intel RealSense D435 camera

3.3.1 Camera calibration

Before usage, it is essential to calibrate the cameras. To do that, Intel provides a calibration software that estimates the intrinsic and extrinsic parameters of the three cameras. The calibration software uses a pattern, shown in Figure 3.6, that can be printed or displayed on a mobile phone screen, where an app ensures the correct pattern size is displayed. The black and white bars in the middle have a width of 10 mm and their number can vary depending on the device used, as can their length which remains always a multiple of 10.

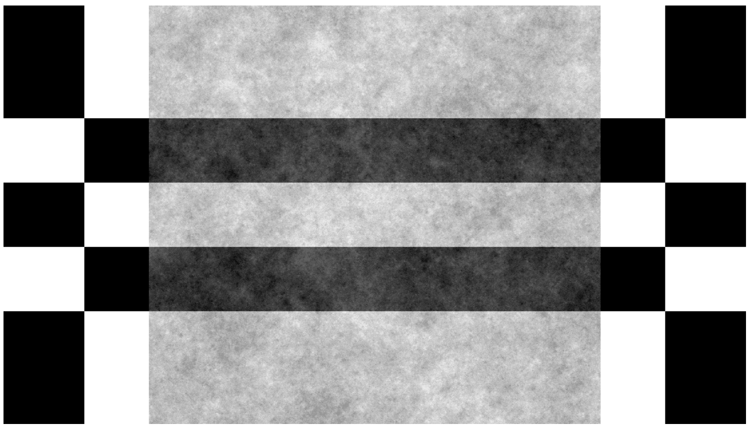


Figure 3.6: Pattern used for calibration

The calibration process requires the user to point the camera at the target from a distance ranging from 650 to 1000 mm. First, the stereo pair is calibrated. To do that, the camera must be moved so that the target is captured by the camera in a region that appears in the calibration software's screen. The same procedure must then replicated for the RGB camera. After each step, a verification is made by asking the user to place the target in ten different locations on the image.

Table 3.3 displays the results obtained after the calibration process for the stereo pair and the RGB camera. The intrinsic matrices of the two IR cameras are exactly equal, which can only mean the calibrator software is programmed to estimate only one matrix for both cameras. Given that the α_u and α_v values are equal, we can conclude that the pixels in the cameras are square. The values x_0 and y_0 are, as expected, near 424 and 240 respectively since the image resolution used was 848x480. The translation vector $\mathbf{p}_W^L$ being null means the origin of the world frame is the same as the origin of the left's camera frame. The last two entries of vectors $\mathbf{p}_W^R$ and $\mathbf{p}_W^{RGB}$ are zero which means Intel assumes the camera planes are all coplanar.

Table 3.3: Outcome of the camera calibration

left IR camera	right IR camera
$\mathbf{K}^L = \begin{bmatrix} 423.296 & 0 & 425.056 \\ 0 & 423.296 & 244.049 \\ 0 & 0 & 1 \end{bmatrix}$	$\mathbf{K}^R = \begin{bmatrix} 423.296 & 0 & 425.056 \\ 0 & 423.296 & 244.049 \\ 0 & 0 & 1 \end{bmatrix}$
$\mathbf{R}_L^W = \begin{bmatrix} 1 & -0.006 & 0.003 \\ 0.006 & 1 & -0.001 \\ -0.003 & 0.001 & 1 \end{bmatrix}$	$\mathbf{R}_R^W = \begin{bmatrix} 1 & -0.003 & 0.003 \\ 0.003 & 1 & 0.001 \\ -0.003 & -0.001 & 1 \end{bmatrix}$
$\mathbf{p}_W^L = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$	$\mathbf{p}_W^R = \begin{bmatrix} 49.907 & 0 & 0 \end{bmatrix}$
RGB camera	
$\mathbf{K}^{RGB} = \begin{bmatrix} 598.105 & 0 & 327.551 \\ 0 & 598.105 & 240.791 \\ 0 & 0 & 1 \end{bmatrix}$	
$\mathbf{R}^{RGB} = \begin{bmatrix} 1 & -0.001 & -0.002 \\ 0.001 & 1 & -0.001 \\ 0.002 & 0.001 & 1 \end{bmatrix}$	
$\mathbf{p}_W^{RGB} = \begin{bmatrix} -14.935 & 0 & 0 \end{bmatrix}$	

The calibration previously obtained was compared to a calibration using MATLAB which has an integrated camera calibration application. However, MATLAB does not have the option to turn off radial distortion estimation. Because the calibration is performed over an undistorted image, the obtained calibration matrices can't be equal. Table 3.4 shows the result of the calibration process using MATLAB. The intrinsic matrices of the IR cameras are different from one another since they are estimated separatly. MATLAB considers the frame of the left IR camera as the world frame since the translation between the two is null and the rotation matrix between the two is the identity matrix.

Table 3.4: Results of the camera calibration with MATLAB

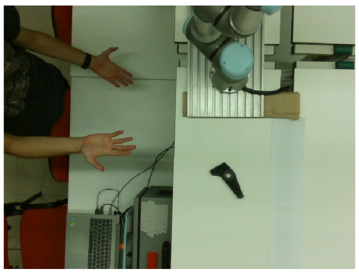
left IR camera	right IR camera
$\mathbf{K}^L = \begin{bmatrix} 421.696 & 0 & 426.488 \\ 0 & 422.118 & 245.209 \\ 0 & 0 & 1 \end{bmatrix}$	$\mathbf{K}^R = \begin{bmatrix} 421.161 & 0 & 425.371 \\ 0 & 421.550 & 245.012 \\ 0 & 0 & 1 \end{bmatrix}$
$\mathbf{R}_L^W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\mathbf{R}_R^W = \begin{bmatrix} 1 & 0 & -0.002 \\ 0 & 1 & 0.001 \\ 0.002 & -0.001 & 1 \end{bmatrix}$
$\mathbf{p}_W^L = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$	$\mathbf{p}_W^R = \begin{bmatrix} 50.057 & 0.062 & 0.0550 \end{bmatrix}$
RGB camera	
$\mathbf{K}^{RGB} = \begin{bmatrix} 596.501 & 0 & 330.606 \\ 0 & 595.775 & 239.722 \\ 0 & 0 & 1 \end{bmatrix}$	
$\mathbf{R}^{RGB} = \begin{bmatrix} 0.999 & 0.011 & 0.001 \\ -0.011 & 0.999 & 0 \\ -0.001 & 0 & 1 \end{bmatrix}$	
$\mathbf{p}_W^{RGB} = \begin{bmatrix} -15.178 & 0.117 & 0.178 \end{bmatrix}$	

3.3.2 Camera output

The camera has four possible outputs, the two IR images, the RGB image and the depth map. The IR and RGB images are received as a vector. The former only has one channel so each entry is the value of a pixel. The latter has three channels, hence the vector has the values of each pixel sequentially, wherein channels are in the order red, green and then blue. Figures 3.7(a) and 3.7(b) show examples of images obtained from the RGB and IR cameras, where the larger resolution and field of view of the IR camera is clearly seen. Figure 3.7(c) is the depthmap obtained from the stereo pair, the grey shade depends on the distance to the camera. Black pixels represent areas where the depth information could not be computed. These figures, as all subsequent, were exported using the function *export_fig* [26].

Figure 3.8 is an example of a point cloud that can be obtained when merging the depth map with the RGB image. From the image it is possible to see that flat surfaces are modelled wavy and that small edges are smoothen. This can be especially seen with the object on the table, whose height is only 12 mm and is indistinct from the table if only depth information is considered. Another example of this phenomena can be seen looking at both arms. The right arm is farther from the table and the transition is clean. As for the left arm that is closer to the table, the boundary between the two is more ambiguous. Also noticeable is a shadow below the right arm, this area is invisible to the camera.

(a) Image from left IR camera


(b) Image from RGB camera

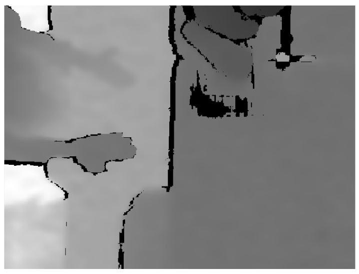

(c) Depth map

Figure 3.7: Examples of camera output

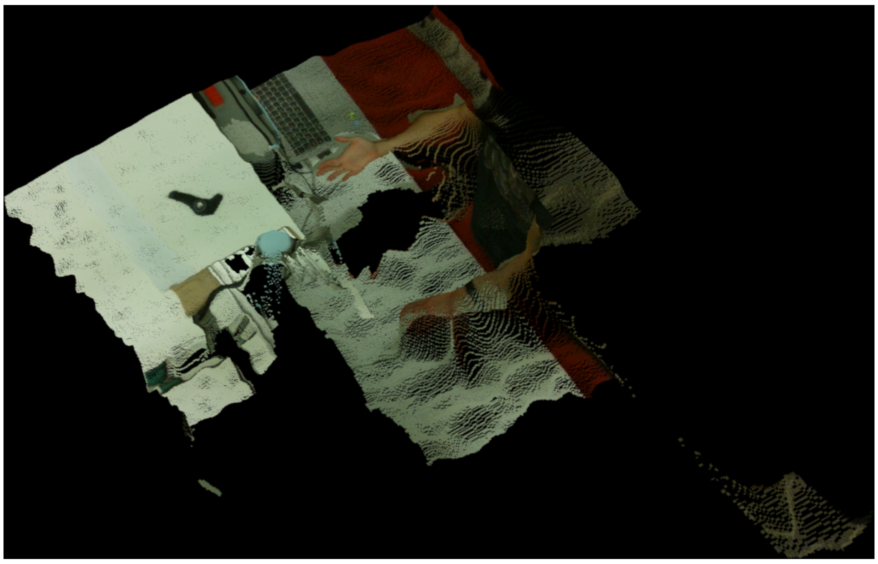


Figure 3.8: Pointcloud obtained from combining depth map and RGB image

To ensure the resulting frames are as homogeneous as possible, the exposure times of all cameras have been manually set. To select the appropriate value, several frames were captures with auto exposure enabled. Then, the exposure times of those frames were noted. For the IR cameras, the time is set in microseconds and any value can be specified. As for the RGB camera, the exposure time is indicated in milliseconds and only certain values are accepted. According to the exposure times observed, the values 1300 microseconds and 39 milliseconds were chosen as exposure times for the IR cameras and RGB camera respectively.

3.4 Software Architecture

There was no problem integrating the camera and the robot since they use a different communication protocol, which are both compatible with MATLAB, and are connected to the host computer through different cables.

The functions employed to treat the point cloud coming from the camera belong to the Computer Vision toolbox. Regarding the communication with the robot, the functions used belong to the Instrument Control toolbox. A detailed explanation of the methods of each class is provided since it is the first time the camera and the robot are used together in the laboratory.

3.4.1 Camera

The interface with the camera is accomplished with a Software Development Kit provided by Intel for several programming languages, including MATLAB. The snippets provided make use of MATLAB's object programming facet, defining a superclass *realsense*. This class has a set of subclasses, which allow all the actions to be performed within MATLAB instead of resorting to Intel's proprietary software.

The camera was implemented as a class *Camera*, which is responsible for interacting with the camera and detecting objects or the operator's hand depending on the required task to be performed. The position and orientation of the interest regions are calculated and transformed to the robot's reference frame before being outputted. Below is presented a description for each of the classes' methods.

Camera, is the class constructor. Its only input, *cameraSettings*, is of type *struct* and contains values of the camera parameters that were chosen not to remain at their default values. The method initiates the communications with the camera and passes on the parameters, which are retained until the camera is plugged off. Given the environment, the parameters chosen were the laser projector's power and the exposure times of both the stereo pair and RGB camera. It isn't necessary to disable auto exposure since defining a custom exposure time already has that effect. The method then calibrates the camera with the scene, in a procedure that will be explained further ahead.

delete, is the class destructor. It insures the communication with the camera is properly terminated when the object is destroyed.

takePic, is a private method and the only one who receives information from the camera. The two variable received from the camera are the RGB frame as a vector of 8 bits unsigned integers, and the 3D points as a three column matrix in which line are the Cartesian coordinates in the camera's reference frame. The vector is then transformed in a three dimensional matrix, the format adopted by MATLAB to represent RGB images. As for the matrix of Cartesian coordinates, it is passed into a *pointCloud* class.

calibrate, is the method that finds the transform matrix between the camera's reference frame and the robot's reference frame. First, the **takePic** method is called. The camera's Z axis and the robot's Z axis are approximately parallel, but point in opposite directions, therefore the cloud of points is rotated 180 degrees around the camera's Y axis, which is described by the homogeneous transformation:

$$\boldsymbol{T}_1^C = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.1}$$

To make the two XY planes coincide, the tabletop is used as it is a planar surface. A plane is created using the points belonging to that surface with the inbuilt function *pcfitplane*. The normal of the obtained plane is the desired Z axis. The rotation matrix to be applied can be calculated using the procedure described in section 2.1.7.

The distance to be travelled along the Z axis is obtained by averaging the Z component of the points belonging to the plane. However, the robot is not placed directly above that surface, but bolted to an elevated surface. Therefore, the height of the surface (38 mm) must be subtracted to that value and the resulting homogeneous transformation is of the form:

$$\boldsymbol{T}_3^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \Delta Z & 1 \end{bmatrix}. \tag{3.2}$$

The difference between the two reference frames is now only a translation in the XY plane and a rotation around the Z axis. To compute the translations in the X and Y axes, the light blue colour of the robot above joint zero is used. The points belonging to that surface are selected by converting the RGB values to the HSV colour space and using the constrains

$$\begin{cases} H < 0.55, \\ H > 0.4, \\ S > 0.2, \\ V > 0.5, \end{cases} \tag{3.3}$$

where H is the hue and V is the value of each point. Then, the centroid of the region is calculated and its position on the XY plane is used to determine the translation needed. The resulting transformation matrix is of the form:

$$\boldsymbol{T}_4^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta X & \Delta Y & 0 & 1 \end{bmatrix}. \tag{3.4}$$

Finally, the frame is rotated 45° around the Z axis:

$$\boldsymbol{T}_R^4 = \begin{bmatrix} 0.707 & -0.707 & 0 & 0 \\ 0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.5}$$

To get the homogeneous transformation from the camera's frame to the robot's frame, all the previously calculated matrices are multiplied sequentially:

$$\boldsymbol{T}_R^C = \boldsymbol{T}_1^C \cdot \boldsymbol{T}_2^1 \cdot \boldsymbol{T}_3^2 \cdot \boldsymbol{T}_4^3 \cdot \boldsymbol{T}_R^4. \tag{3.6}$$

The obtained matrix is passed into an *affine3d* object, which is MATLAB's way of storing a three dimensional geometric transformation.

getOrientation, is the method used to calculate the orientation of a body. The method's only input is a binary image, in which the the body of interest is represented by ones. The image is passed into the inbuilt function *regionprops* to find its orientation. The output is the angle in degrees between the major axis of the object and the Y axis of the camera's reference frame in the interval] -90 ; 90]. The goal being to align the Y axis of the gripper with the major axis of the body, it suffices to add 45° to the angle obtained. The resulting angle is the rotation about the Z axis necessary in a 'ZYX' set of Euler angles. To obtain the final orientation, a rotation of 180° is performed around the Y axis to make the Z axis point towards the object. The Euler angles are then converted into the axis notation, as seen in section 2.1.6.

isPresent checks if there the operator is in the shared workspace. Its output is a boolean value, false if the shared workspace is empty, true otherwise. Figure 3.9 illustrates the function behaviour of this method. In the left image, the hand is outside of the cuboid, thus the output of the method will be a boolean value false. In the right image, the arm is inside the workspace and the output of the method will be a boolean value true.

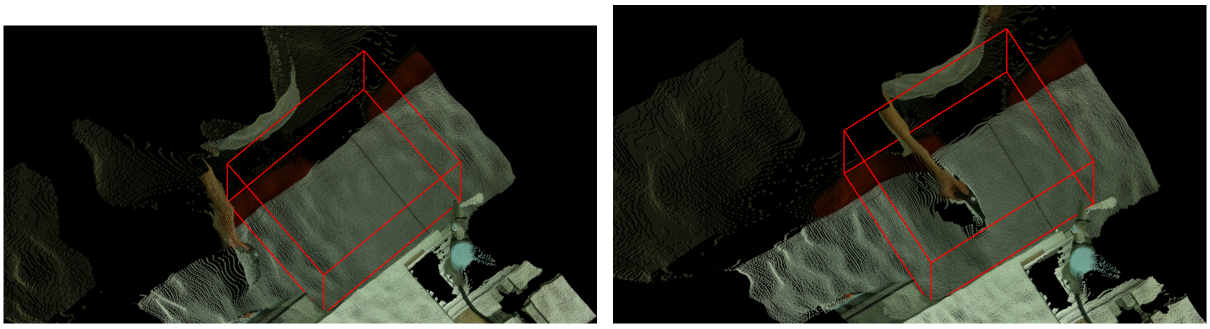

(a) Hand outside the workspace (b) Hand inside workspace

Figure 3.9: Illustration of **isPresent** method

getObjectOnTablePosition, is the method that outputs the position, orientation and colour of all the objects placed on the table. First, a rectangular cuboid where the objects can be located is defined. All the points of the point cloud located inside that cuboid are retrieved. Given that there is a biunivocal relation between the pixel index in the RGB image and the point index in the point cloud, a binary images is created where the pixels with value one correspond to 3D points that belong to the predefined zone. This binary image can be seen in Figure 3.10(b).

Then, another binary image is created using the original RGB image. The pixels with value one on this second binary image are the ones corresponding to white colour on the RGB image. To do that, the RGB image is converted to the HSV colourspace and the following constrains are applied

$$\mathbf{B} = \begin{cases} S < 0.2, \\ V > 0.5. \end{cases} \tag{3.7}$$

The resulting image can be seen in Figure 3.10(c). Next, the second binary image is subtracted from the first. The result is an image where only the parts on the table correspond to ones, like the one in Figure 3.10(d).

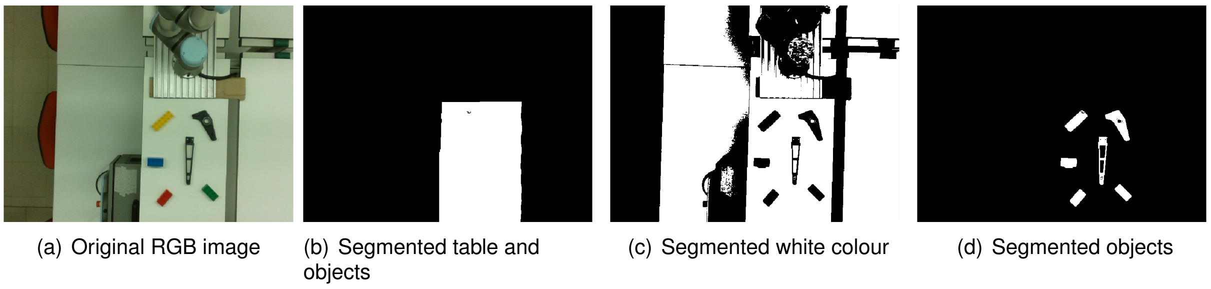

(a) Original RGB image (b) Segmented table and objects (c) Segmented white colour (d) Segmented objects

Figure 3.10: Procedure for object segmentation

The colour of each part is computed in the HSV colourspace, assessing the saturation and value to determine if the object is black. If not, the Hue is compared to a sample to determine the colour of the object. A vector of strings containing the colour of each object is one of the outputs of the method.

The position of each object is estimated by computing the average of the Cartesian coordinates of the 3D points of each object. The method **getOrientation** is called to obtain the orientations of the objects. Finally, the second output of the method is constructed by concatenating the positions and the orientations in a matrix, where each line corresponds to an object's pose. A visual representation of the positions and orientations of each object can be seen in Figure 3.11. The centroids are displayed as a red dot and the orientations as a green line.

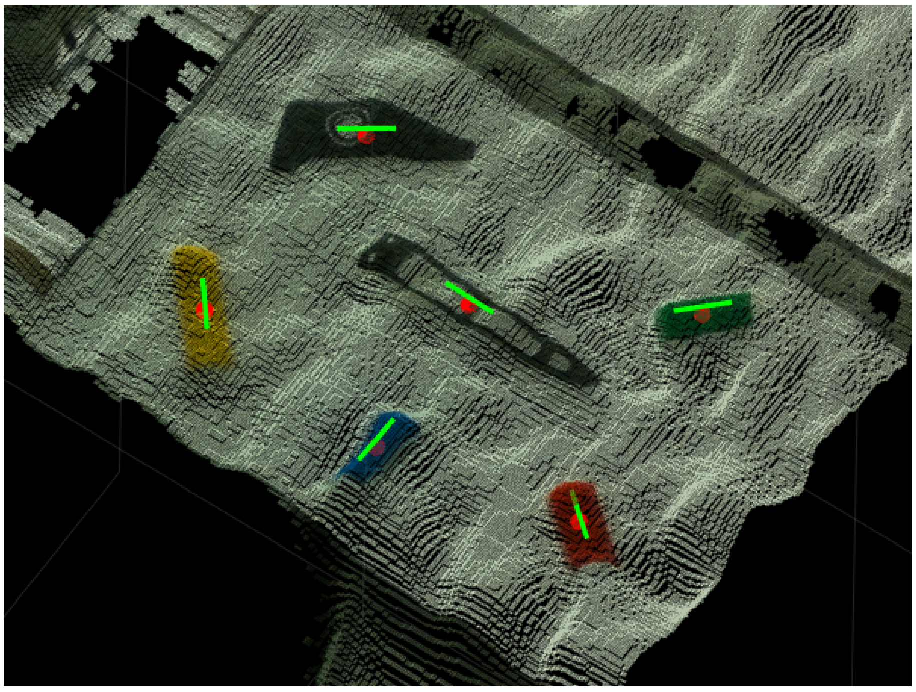


Figure 3.11: Centroid and orientation of objects

getHandPosition, is the method that estimates an operator's empty hand's position and orientation. The detection starts by designating a rectangular cuboid where the hand can be present, this region is the shared workspace between the robot and the operator. Again, the biunivocal relation between pixels of the RGB image and points in the point cloud is used to create a binary image where all the pixels corresponding to points inside of the cuboid are ones.

Then, the RGB image is scanned looking for skin and a binary image where skin pixels on the original image correspond to ones is created. This is done converting the image to HSV colourspace and applying the constraints

$$\mathbf{B} = \begin{cases} H < 0.15 \quad OR \quad H > 0.95, \\ V < 0.8, \end{cases} \tag{3.8}$$

where **B** is the obtained binary image and H is the hue and V is the value of the original image.

The two binary images are aggregated resorting to the logical operation AND. There are cases where this image can have more than one blob, such as if there is more than one hand in the workspace or if the operator is using a bracelet or a watch that would separate the hand from the rest of the arm. Therefore, position of each blob is computed and the closest to the robot is chosen. The orientation is obtained by calling the method **getOrientation**. Because parts are rather manipulated with fingers and the centroid of the hand is located in the palm, a new point is created, called off-load location. This spot is a translation of the centroid along the hand's orientation of 50 mm. The outuput of the method is depicted in Figure 3.13, the red dot represents the off-load location and the green line the orientation.

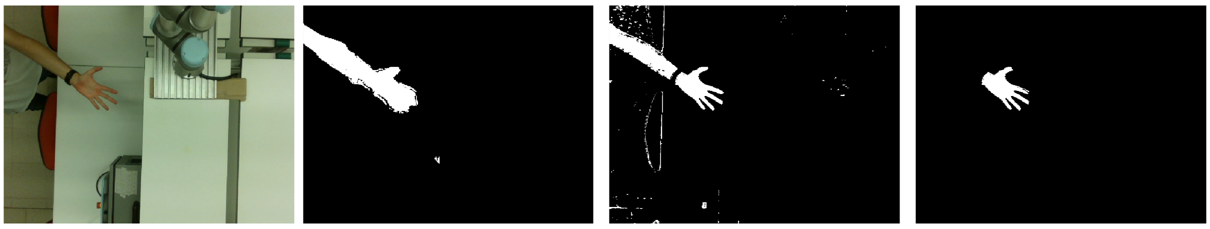

(a) Original RGB image (b) Segmented body inside workspace (c) Skin detection (d) Segmented hand

Figure 3.12: Procedure to segment the hand

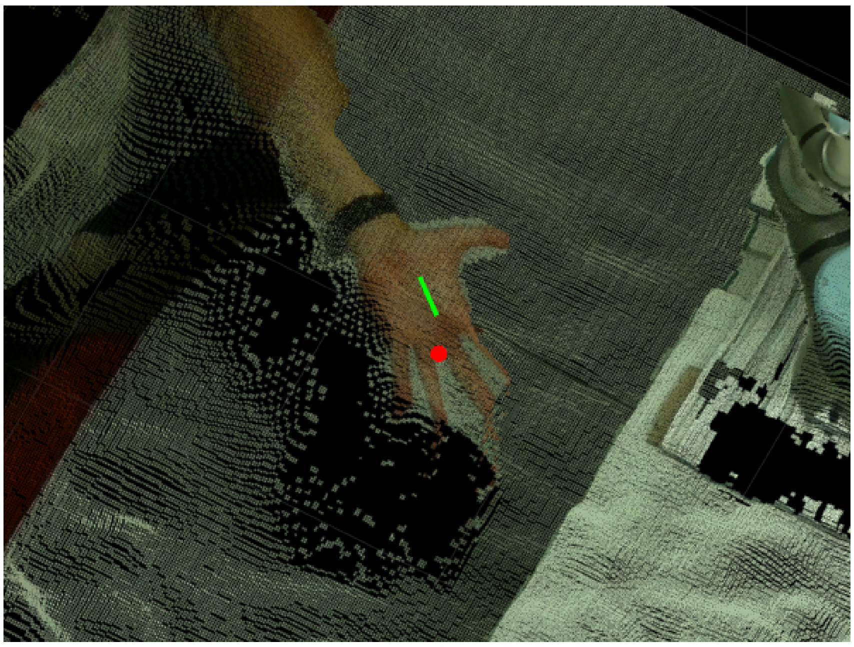


Figure 3.13: Hand orientation and off-load location

getObjectInHandPosition, is the method that estimates an object's position and orientation when it is being held by an operator. The procedure is similar to the one used in the method **getHandPosition**. Again, the bodies inside the workspace are segmented as can be seen in Figure 3.14(b). However in this case, the binary image that identifies skin colour is subtracted from the first image obtained. The result is a binary image where all non-skin objects in the workspace are highlighted as can be seen in Figure 3.14(c).

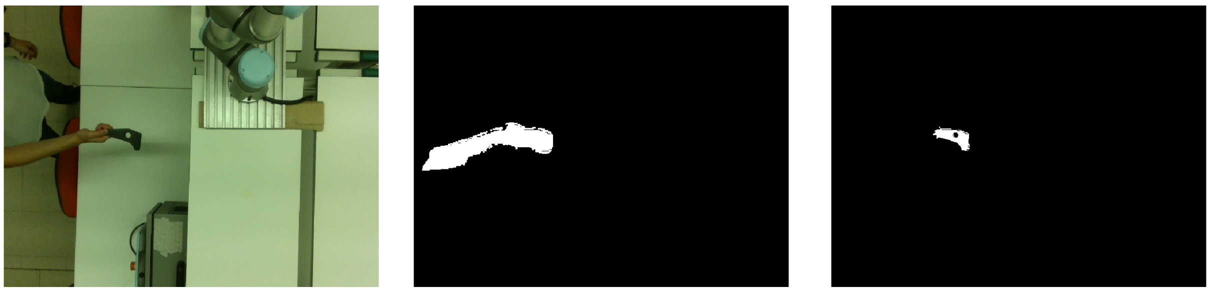

(a) Original RGB image (b) Segmented body inside workspace (c) Segmented object being held

Figure 3.14: Procedure to segment an object in the operator's hand

Again, several blobs can result from this operation so the same step of finding every object's position and calculate their distance to the robot is performed. The method **getOrientation** is once again used to estimate the orientation of the chosen object. Figure 3.15 shows a visual representation of the output of this method, with the estimated centroid position of the object in red and its orientation in green.

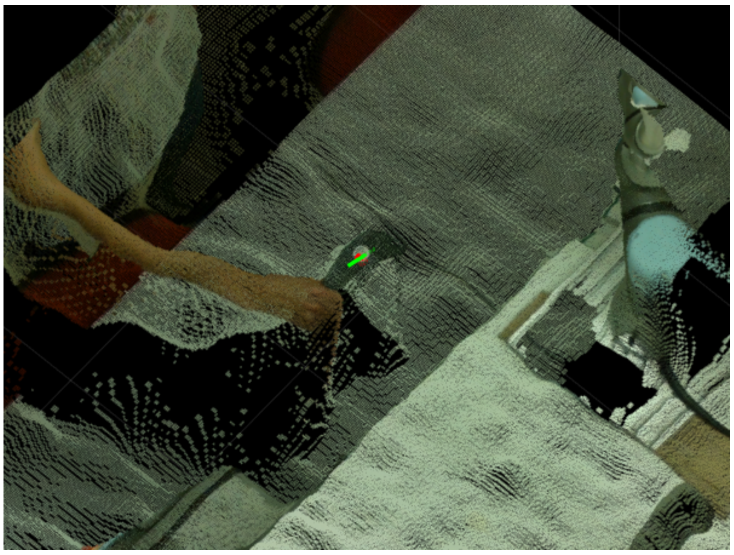


Figure 3.15: Held object's centroid and orientation

3.4.2 Robot

Similarly to the camera, the robot was also implemented as a class The interface with the robot consists in opening the necessary communication routes through TCP/IP protocol, generating and sending strings of text with the URScript commands to be executed by the robot and receiving information about the robot's state.

URRobot is the class constructor. The input *inputRobot* is of type *struct* and can contain up to five fields. The robot's IP adress is passed through the field *robotIP* as a *string*. The fields *accelerationMod* and *speedMod* are two *doubles* act as modifiers for acceleration and speed respectively. They act as a lever to control accelerations and velocities as they are multiplied by some predefined values. The default values for both of these parameters is one. The field *radius* is the blend radius used to transition between two consecutive move commands, its default value is 10mm. Finally, the value of *clearance* set the minimum distance between the closest point on the trajectory and the origin of the robot's reference frame. The purpose of this value is to prevent contact between the robot's base and the tool or the object being manipulated and it's default value is 200mm.

Using the inputs, the RTCI and Dashboard interfaces are initiated. The constructor also reads a text file with a snippet of code that is constant, to be used when placing an object on a hard surface. This code does not depend on any variables and therefore can be stored as a property for when it is necessary.

delete is the class destructor. It moves the robot to the home position, before closing any communication established.

calibrate, is the method used when calibrating the camera. The robot is placed in a configuration where the camera has a clear view over the workbench and the robot's base as those are the necessary areas to perform the calibration.

goToRestPosition is used to instruct the robot to adopt the rest position. This position was defined as the robot being upright with the tool's tip pointing down. This configuration ensures the robot has a small footprint on the camera's images.

isRunning is a method that enquires the robot through the Dashboard interface to know if a program is running. If a program is running, and therefore the robot or the tool is in motion, the method returns a logical true. Otherwise, the method returns a logical false.

isInPosition checks if the robot is in a given position. The position to be checked is the only input of the method. First, the robot's position is read from the RTCI interface. Then, the difference between the desired and the real positions and orientations are computed separately. Tolerances are defined in terms of position and orientation, if the current robot position is within that tolerance of the requested position, the method asserts the robot is in the desired pose. The position tolerance has been set to 1mm. As for the angle, a 5° angle is allowed between the desired and real rotation axes. This is equivalent to having a dot product between the two angles of at least 0.996.

sendProgram is a private method that sends a program to be executed by the robot. The only input is of type *string* and the method only adds the header and an *end* statement needed to make it an appropriate URScript function. The functions is then conveyed to the robot through the RTCI interface.

savedProgram instructs the robot to run a program it has in memory. This method is private and its only input is the name of the program to be executed. The request is made using the Dashboard interface, first by instructing the robot to load the program, then by ordering its execution. Currently, this method is called to open and close the gripper. During these operations, the fingers have vertical movement in addition to horizontal movement. The URCap provided by OnRobot can drive the robot during the gripper motions to maintain the TCP pose. This option was enabled when clenching because it prevent the fingers from colliding into the table or the hand of the operator. Conversely, this option is disabled when releasing an object since the vertical movement moves the fingers away from any obstacle.

pickObject is used when the robot has to pick an object on the table or from an operator's hand. The only input of the method is a six entry vector with the position and orientation of the part to be picked from the table. An intermediate position above the part is generated, to ensure the tool approaches the object vertically.

To comply with the recommendation to notify the operator of robot's motion, the gripper is opened to its maximum finger gap before any movement. The method **savedProgram** is called, and the robot executes a function that only moves the gripper.

Before, any motion, the robot checks that the waypoints are reachable using the internal function *is_within_safety_limits*. The result is passed as the condition of an *If* statement, and no movement is performed if the conditions is not verified.

Because the rest position is upright, and therefore a singularity, the first movement has to be done on the joint space. The desired joint coordinates are calculated using the robot's internal function *get_inverse_kin*, whose input is the pose to achieve. A second optional input is also used, to ensure the robot reaches the object in a elbow up configuration. The functions will compute the joint angles corresponding to the desired pose that are closest to the joint angles provided as second input.

placeObjectInPosition is the method used to place an object on a defined position. The current and target, positions and orientations are passed as six entry vectors. Two waypoints are calculated above each of those positions to ensure a vertical approach and departure. Additionally, a third waypoint may be added if the path is too close from the robot's base. The minimum distance between the path and base is computed, and if it is less than the property *clearance*, a waypoint is created along the direction from the base to the nearest path point. This behaviour is illustrated in Figure 3.16(b), where an additional waypoint is generated to ensure a minumum clearance between the tool and the robot's base. In Figure 3.16(a), the direct path already satisfies the minimum gap, therefore no new waypoint is added.

Finally, the robot returns to its rest position. As with the first movement of the method **pickObject**, it has to be done in joint space.

placeObjectOnTable is called when the object is to be placed on a rigid surface. It is similar to the **placeObjectInPosition** method but it doesn't require the surface's height to be known with exactitude. First, the robot hovers over the surface, then it is brought down at a reduced speed until contact is detected. The robot is promptly stopped and the object is released. Lastly, the robot is returned to its rest position.

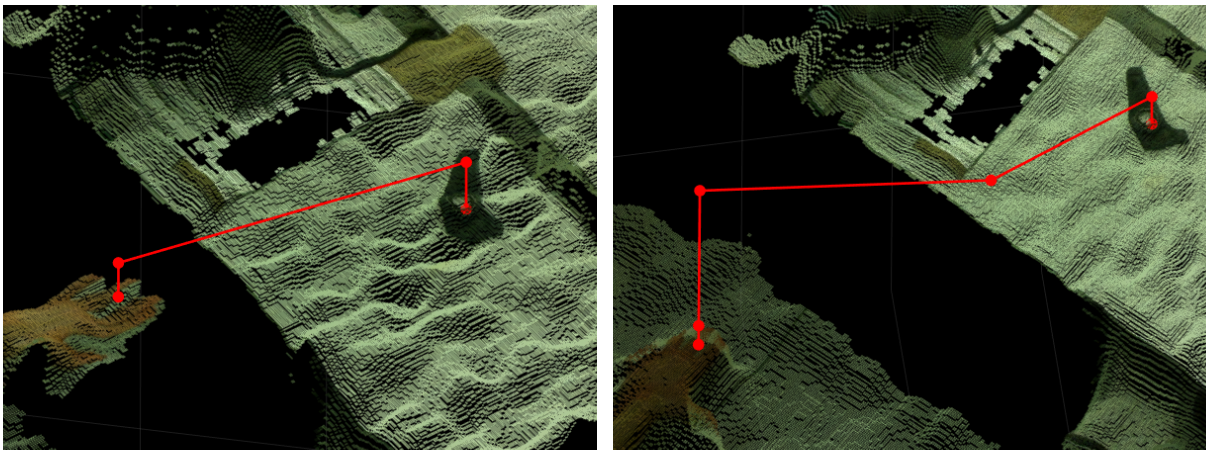

(a) Direct path between the part and the hand (b) Waypoint added to prevent collisions

Figure 3.16: Examples of generated paths

3.4.3 Main

The main script acts as an intermediary between the camera, the robot and the user. First, the script creates two objects from classes *Camera* and *URRobot*. Since the constructors of the classes need input parameters, those must be defined beforehand. Then, it proceeds to the calibration of the system. For that, the robot is moved to a position where the camera has an unobstructed view of the top of the base of the robot. Next, a frame is retrieved and the transformation matrix between the camera reference frame and the robot reference frame is estimated using the method *calibrate*. At this stage, the system is ready to operate therefore the robot place in its rest position.

The script then enters an endless loop, a flowchart describing the can be seen in Figure 3.17. The script is continuously monitoring the shared workspace to determine if the operator is present resorting to the method *isPresent*. When the operator is detected, it is necessary to instruct which operation is to be executed. Two operations are possible, retrieve an object from the hands of the operator and place it on the tabletop or pick an object on the table and give it to the operator.

If the operator chooses to give the robot an object so it can place it on the table, the camera checks that the operator has an object in its hands. If an object is found, the robot checks if the part is reachable and if so the object is carried to the hand of the operator. Finally, the robot returns to its rest position. If there is no object or the object is not reachable, no action is taken.

If the user wants an object that is on the table, the camera checks how many object there are. If no object is found, no action is performed. Otherwise, if there are multiple objects, the user has to select which object is to be transported. If the object is reachable, the robot grips it. The camera is then used once again to verify that the operator is still inside the shared workspace and that its hand is reachable. If so, the object is delivered to the operator and then returns to its rest position. Otherwise, the robot is sent back to its rest position without moving the object.

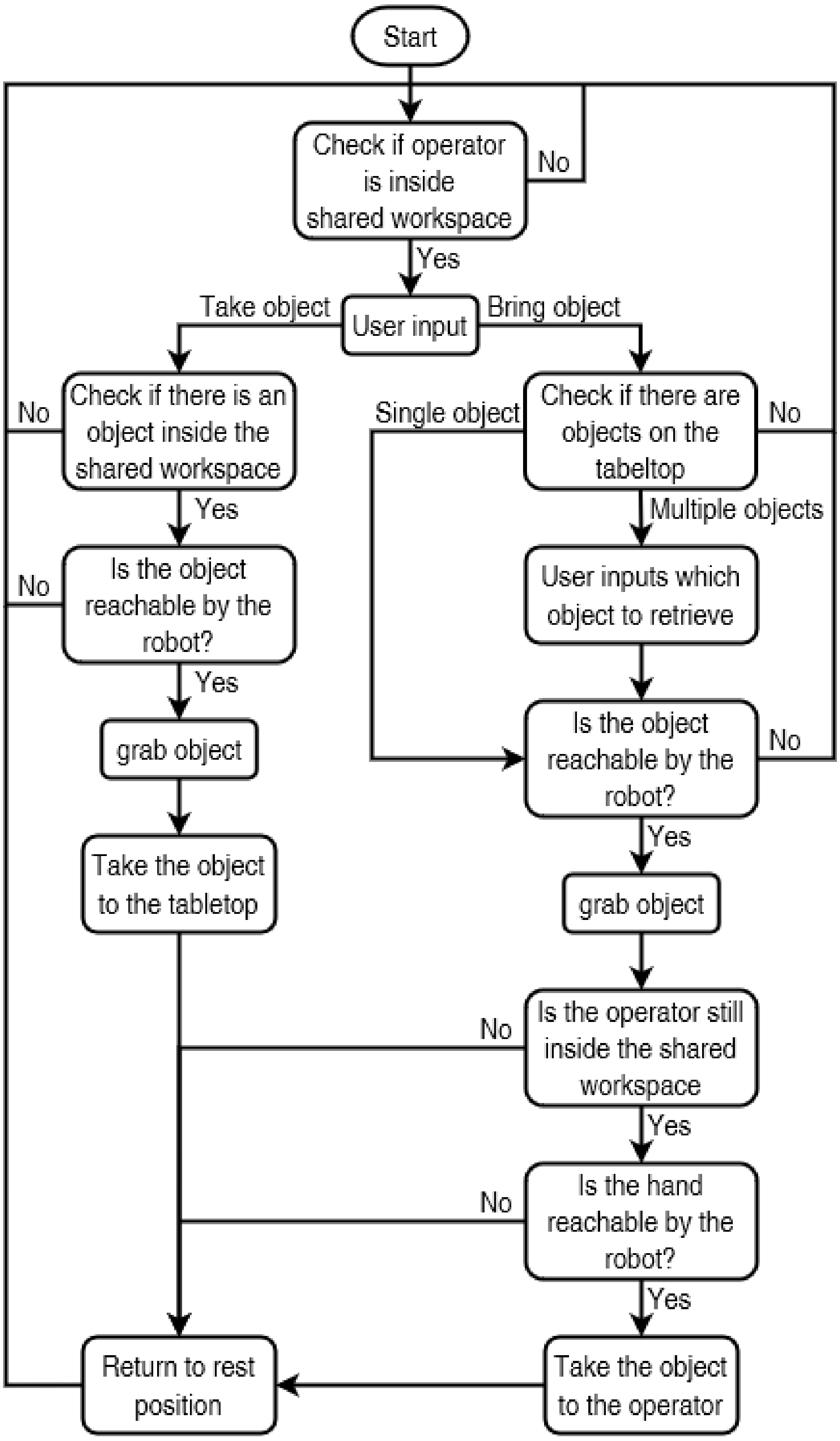

Start
Check if operator is inside shared workspace
No
Yes
User input
Take object
Bring object
Check if there is an object inside the shared workspace
No
Yes
Is the object reachable by the robot?
No
Yes
grab object
Take the object to the tabletop
Check if there are objects on the tabeltop
Single object
No
Multiple objects
User inputs which object to retrieve
Is the object reachable by the robot?
No
Yes
grab object
Is the operator still inside the shared workspace
No
Yes
Is the hand reachable by the robot?
No
Yes
Take the object to the operator
Return to rest position

Figure 3.17: Flowchart of the endless loop of the main algorithm

Chapter 4

Results

This chapter presents the results obtained after the implementation described in the previous chapter. First, a review of the segmentation algorithms. Then, the accuracy of the overall system was tested. Next, the generated paths are compared to the actual robot paths. Finally, an example of collaboration achieved is shown.

4.1 Segmentation results

To achieve correct results, it is paramount that the objects and the hands of the operator are correctly detected. The parameters used for segmentation were tuned according to the conditions of the laboratory, particularly lighting conditions.

4.1.1 Hand segmentation

As seen in section 3.4.1, the hand segmentation algorithm uses a mix of 3D and colour cues. Figure 4.1 displays the segmentation results a few hand configurations. The top row shows the original RGB images and the bottom row the obtained segmentation result.

In the first column, a single hand is present, the algorithm had no problem correctly segmenting the hand from the rest of the arm at the wrist. In the second column, there are two hand in the image approximately at the same depth. The left hand is closer to the base of the robot and hence is chosen. Third column has the left hand below the right hand. The latter becomes the closest hand to the base of the robot and therefore is preferred. Regarding the last column, the hands are at different depths but appear as a single entity in the RGB image. As a result, the segmenting algorithm fails and yields a binary image where both hands are present.

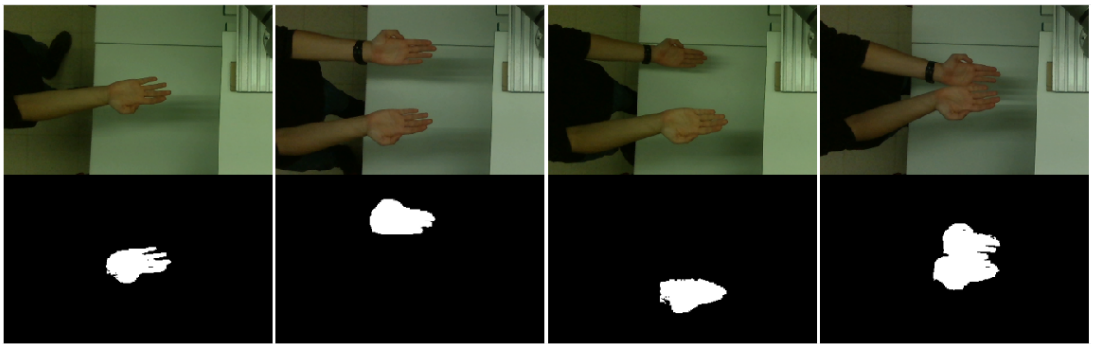


Figure 4.1: Examples of hand segmentation results

4.1.2 Segmentation of objects in the hand

Figure 4.2 depicts the original RGB images and the corresponding object segmentation below for six different cases. The first two columns demonstrate that the way the operator is holding the object is unimportant. As long as the object is visible, the algorithm can identify it and isolate it. The third column demonstrates that a cluttered background is not a problem for the algorithm, as it was able to correctly segment the held object even though several other objects were scattered on the table below. The next two columns show how clothes can affect the result obtained. In the fourth column, the sleeve and the object are detached from one another. Therefore, the object is correctly identified. However, in the fifth column the sleeve is overlapping the object. The algorithm is deceived into considering as the sleeve as part of the object. Finally, the rightmost column shows how the hand being too close to the table can cause issues. The noisy transition between the hand and the table, as already seen in Figure 3.8, and the shadow of the hand makes the algorithm recognize part of the table as belonging to the object.

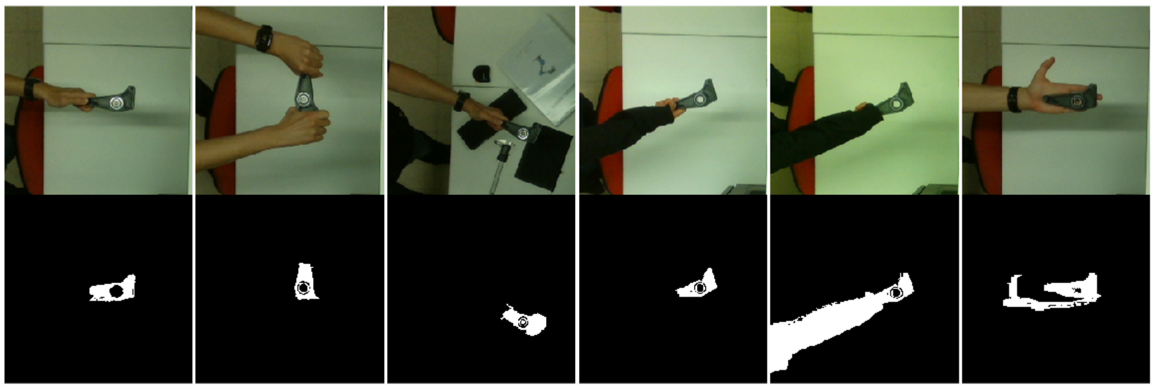


Figure 4.2: Examples of segmentation for objects in hand

4.1.3 Segmentation of objects on the table

Figure 4.3 shows the outcome of the object segmentation in three different cases. The blobs in the bottom row are coloured according to the colour the algorithm detects on the RGB image, black objects appear in white.

In the first two columns show a correct segmentation and colour choice. However, the rightmost column shows a case where two objects close together are mistaken as a single object. The algorithm recognizes a single yellow objects instead of two distinct yellow and red objects since the latter is smaller in size.

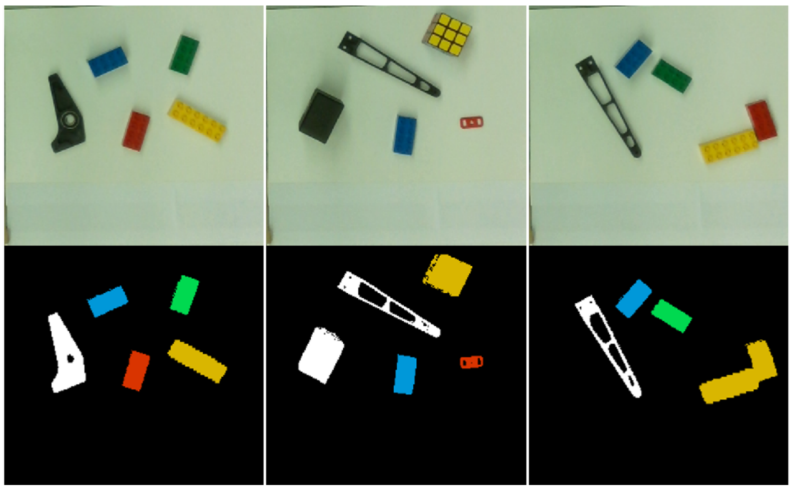


Figure 4.3: Examples of object segmentation results

4.1.4 Computation times

In order to ensure a fluid interaction between the robot and the operator, the robot has to act in real time which requires low computational times. Table 4.1 displays the median running time of the methods determined using the inbuilt function *timeit*. The time needed to retrieve a frame indicates the acquisition is made at 30 Hz. This is a limitation from having connected the camera to the host computer through a USB2 cable. Regarding the segmentation of objects on the tabletop, the time is dependent on the number of objects present. Each additional object results in an increase of approximately 4 ms in processing time. This explains the higher variance observed, since the measurements were taken with a varying number of objects on the table.

Table 4.1: Computation times of Camera class methods

	Retrieve frame	Hand segmentation	Object in hand segmentation	Object on table segmentation
Median [ms]	36	23	25	36
Variance [ms^2]	5	6	7	23

4.2 Accuracy testing

To test the accuracy of the overall system, an experiment was designed. In this analysis several points were considered and the position reported by the robot was compared to the position yielded by the camera and transformation matrix.

4.2.1 Experimental setup

For this experiment a red 3D printed part shown in Figure 4.4. This piece proved to be ideal given its small size and since its center part has a width of 15 mm, the same as the fingers of the gripper. This makes it easy to place the center of the part at the TCP, thus minimizing any human contribution to the error obtained. In addition, its height is only 3 mm so any sidewalls captured by the cameras will have a minimal contribution towards the observed inaccuracy. Finally, its red colour is well saturated making it easy to locate on the image.

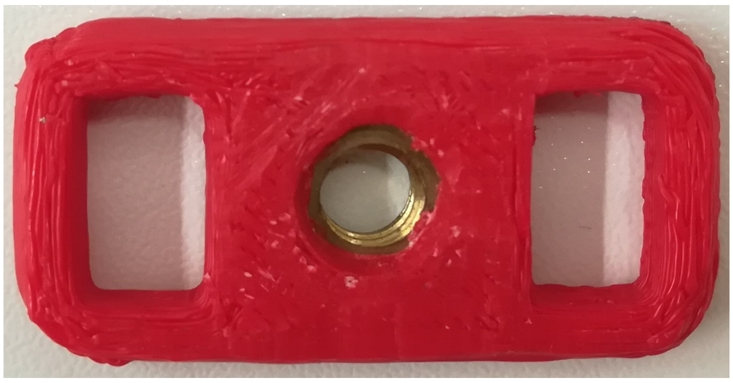


Figure 4.4: 3D printed part used to assess accuracy

The part was placed in 15 different locations inside our two areas of interest, the tabletop where objects can be placed to be picked up by the robot and the shared workspace. For each position, five frames were captured. The transformation matrix used was:

$$\boldsymbol{T}_R^C = \begin{bmatrix} -0.7071 & 0.7071 & 0.0014 & 0.2537 \\ 0.7070 & 0.7070 & -0.0146 & 0.0455 \\ -0.0113 & -0.0094 & -0.9999 & 1.0967 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}. \tag{4.1}$$

4.2.2 Preliminary results

The errors were split in error in the XY plane and error in the depth error, as shown in Figure 4.5. This graph was redesigned using the function *butplotwell* [27]. In both cases, the error increases with the distance to the camera. The distance to the camera is inferior to 1.2 m for objects placed on the table. In these cases, the depth error under 10 mm and an error in the XY plane under 20 mm is acceptable for this application. In regard to the accuracy for objects inside the shared workspace, the results deteriorate quickly with the increase of the distance to the camera. In that situation, even though the operator can adjust it's hand position to cope with errors from the robot, the errors were deemed too significant. Therefore, the source of such errors had to be determined to mitigate it if possible.

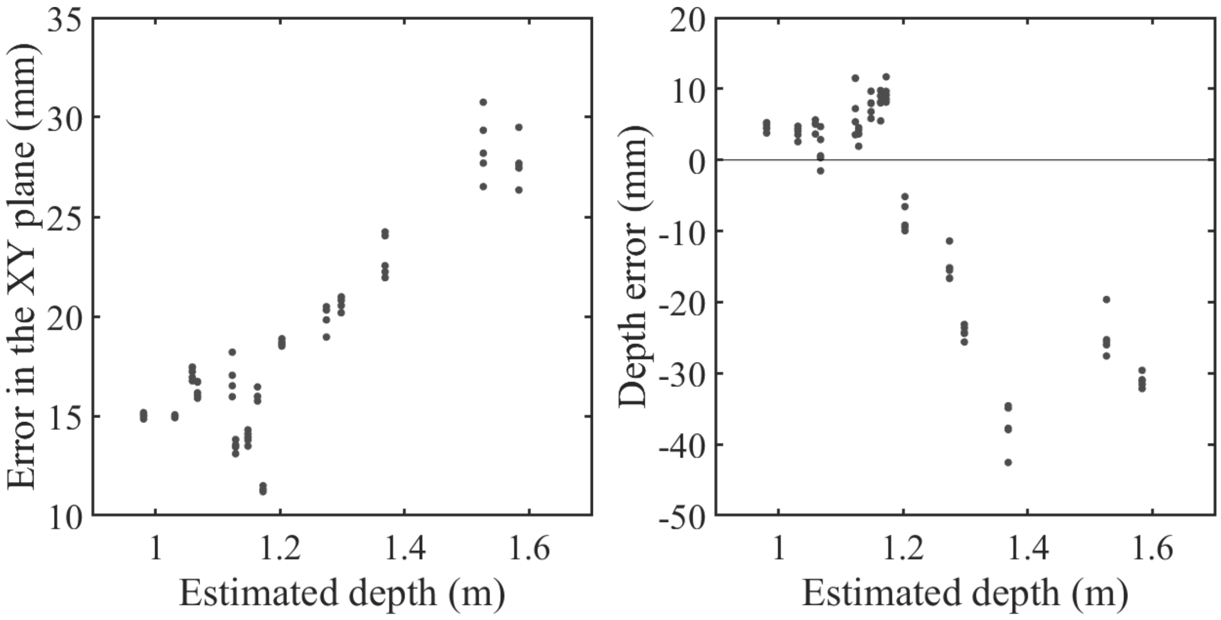


Figure 4.5: First accuracy results

4.2.3 Influence of the calibration

The calibration process is paramount, because it ensures that the robot is steered to the right coordinates. Without a correct calibration, the performance of the overall system will be poor even if the segmentation phase performed correctly.

To determine the influence of the calibration process in the errors obtained, ten calibrations were performed and the obtained results were compared. Figure 4.6 reveals the outcome of this experiment. The left graph shows that a correct calibration is paramount regarding accuracy on the XY plane. For the best calibration obtained the errors are constant, independently of the distance to the camera. This yields an error of 13mm at most, and compared to the worst calibration the error is less than a fifth for the positions farther from the camera. The right graph shows that the calibration is also important for the depth error, however the difference between best and worst calibrations is lesser. The depth error for the best calibration is about half the error for the worst calibration along the entire scale considered.

The transformation matrix for the best obtained calibration was:

$$\boldsymbol{T}_R^C = \begin{bmatrix} -0.7070 & 0.7071 & -0.0156 & 0.2305 \\ 0.7070 & 0.7071 & 0.0101 & 0.0431 \\ 0.0181 & -0.0039 & -0.9998 & 1.0931 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}. \tag{4.2}$$

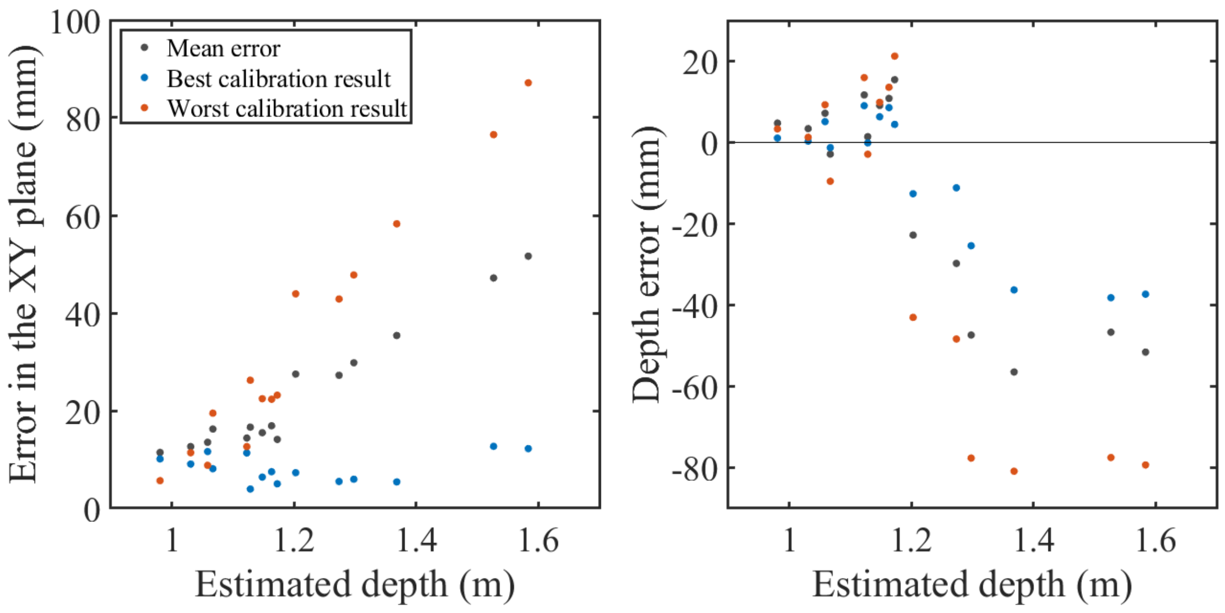


Figure 4.6: Comparison of calibration results

4.2.4 Calibration Alternatives

To attempt at lowering the obtained errors, two other automatic calibration methods were devised. The surface used as a reference for the XY plane of the robot is a glossy white colour and is placed directly underneath two lamps. Light is reflected to the cameras resulting is lens flare. As a consequence, the IR projector pattern is not visible which reduces the camera accuracy because of the lack of texture.

The first solution for this problem was to perform the calibration in a darker environment. Only the set of lights farther from the surface was kept on for this trial. This corresponds to light row A in Figure 3.3 To adjust for the reduced incident light, the exposure times of the RGB and IR cameras were increased. The second idea was to add texture to the surface by placing a checkerboard pattern printed on a sheet on top of the surface. This adds texture without changing the height. Ten calibrations were carried out for each new solution.

Regarding the calibration with less lighting, the best obtained result was similar to the best one previously obtained. The improvements are negligible, both in terms of error on the XY plane and depth error. However, the worst results obtained are significantly better than before. Comparing the worst cases, the errors are approximately half. This indicates that the outcome from low light calibration is more homogeneous.

Concerning the calibration with added texture, the results are poor. This alternatives proves not to be viable since it produces the worst results of all methods tested. This might be explained by the fact that the point cloud obtained does not have points at the center of the patterns.

Figure 4.7 compares the best achieved result from these two methods and the best calibration previously obtained.

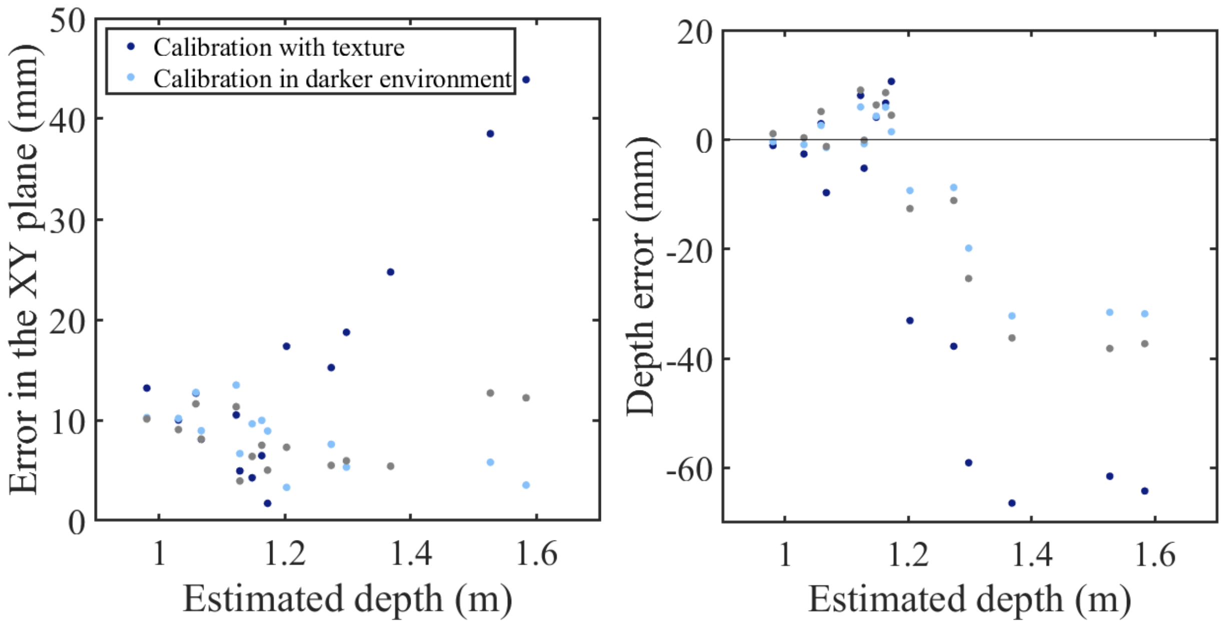


Figure 4.7: Comparison of calibration methods

The transformation matrix that provided the best results, resulting from calibration with dimmed light, used through the remaining of this work is:

$$\boldsymbol{T}_R^C = \begin{bmatrix} -0.7069 & 0.7069 & -0.0246 & 0.2332 \\ 0.7071 & 0.7071 & 0.0016 & 0.0544 \\ 0.0186 & -0.0163 & -0.9997 & 1.0936 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}. \tag{4.3}$$

4.2.5 Camera depth error

In an attempt to determine if the depth errors achieved are inherent from the camera or a systematic error from the calibration method devised, the depth error of the camera alone was estimated. For that purpose, the same red part was used. It was placed at varying distance from the camera that was mounted on a tripod to guarantee it would not move. The red part was placed in a vertical surface, parallel to the plane of the camera. This surface was place at a distance ranging from 0.5 to 1.7 meters from the camera with increments of 100 mm. A measuring tape was used to place the camera at the correct depth.

Figure 4.8 depicts the errors observed. One can note how the depth is predominantly overestimated for closer distances and underestimated for the farthest positions. This same phenomena can be noticed in Figures 4.5, 4.6 and 4.7

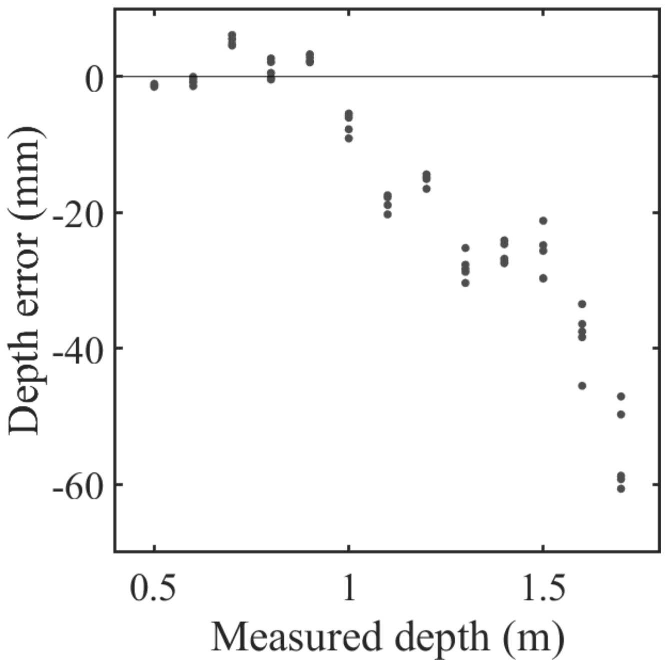


Figure 4.8: Depth error estimate

4.2.6 Camera repeatability

To test the repeatability of the camera, the same images captured previously were used. The position of the object in each of the five images was computed and the largest distance between any two of them was selected as the repeatability. Figure 4.9 reveals the repeatability obtained as a function of the distance to the camera. Once more, two different graphs are display. One for the repeatability on the XY plane and another for depth repeatability. The results suggest that the repeatability on the XY plane increases linearly with the distance to the camera. As for the depth repeatability, there is no clear correlation between the value observed and the distance to the camera

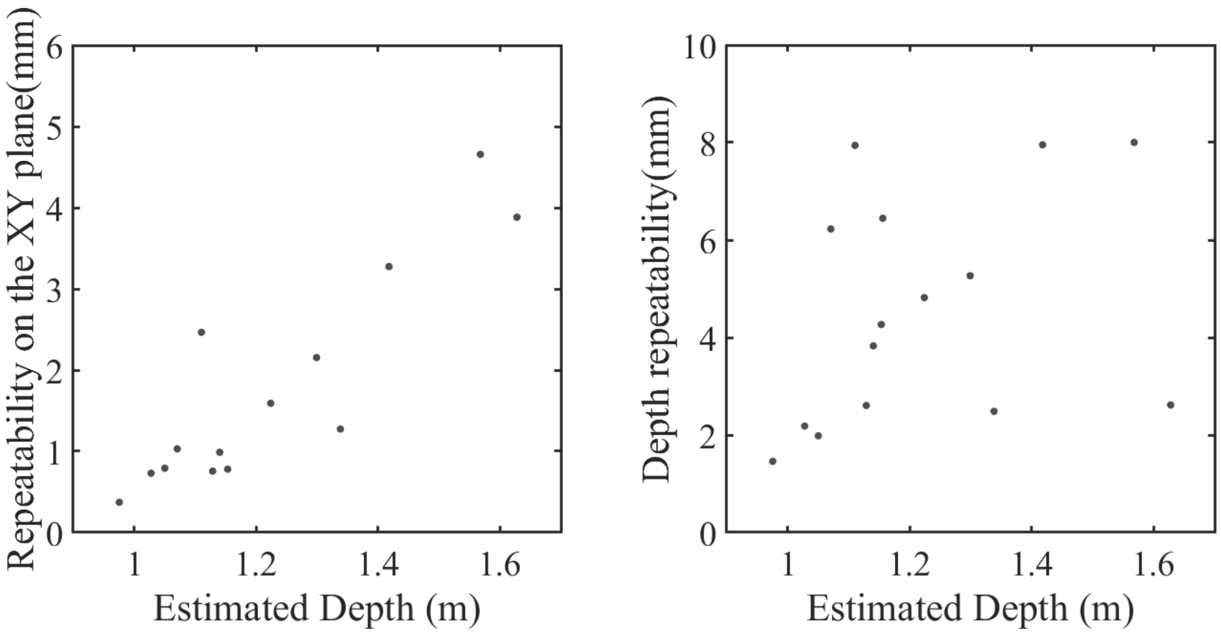


Figure 4.9: Camera repeatability results

4.2.7 Robot repeatability

The advertised repeatability of the UR3e robot is 30 microns, this value is tested following the norm ISO 9283. The norm establishes that the robot must be carrying the maximum payload and visit four positions before returning to the position where measuring is performed. The measurement is performed with high precision devices such as a laser tracker. Here, the objective is to evaluate if the real repeatability is enough for the task at hand.

For the test, a pen attached to the gripper was used to tint a piece of paper placed on the table. First, three control marks were executed. Then, the robot dyed a fourth location 100 times, visiting four locations between each time. Considering that the pen is lightweight, additional weights were attached to the gripper to simulate a heavier payload. Since these weights modify the mass and position of the center of gravity of the end-effector, the Payload Estimation widget was used to estimate these new parameters. The predicted values of mass and center of gravity position are displayed in Table 4.2. Given the location where the masses have been attached, the estimated center of gravity position is unlikely to be the real value. This can be confirmed by using the widget to estimate the parameters for the unloaded gripper. The mass value obtained was 0.87 kg, which is similar to the real mass of the gripper as seen in Table 3.2. However, the position of the center of gravity in the Z direction was 59 mm, which is almost half of the real value. Nevertheless, the value obtained through the widget was used for this experiment.

Table 4.2: Mass and center of mass position for the repeatability test

mass [kg]	center of mass [mm]		
	X	Y	Z
1.53	-2	-1	75

Figure 4.10 displays the obtained blots, the three leftmost are the control marks where the pen has reached the paper only once, the square is present for size comparison purposes, it has 100 mm^2.

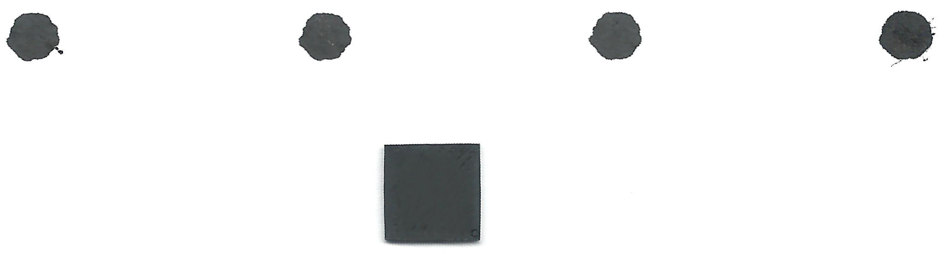


Figure 4.10: Robot repeatability test result

Counting the number of pixels of the circles and comparing to the number of pixels of the square, it is possible to estimate their area and diameter by cross multiplying. The values obtained are displayed in

Table 4.3. Comparing the diameters calculated, the repeatability can be considered to be between 130 and 200 μm. This value is adequate for the task performed.

Table 4.3: Robot repeatability test result

	Control 1	Control 2	Control 3	Test	Square
Number of pixels	11 286	11 007	11 104	11 902	56 642
Area [mm^2]	19.93	19.43	19.60	21.01	100.00
Diameter [mm]	5.04	4.97	5.00	5.17	X

4.3 Paths

This section is a review of the paths generated by the algorithm and the execution of such paths by the robot. These paths were generated with acceleration and speed modifier of 1, a blend radius of 20 mm and a base-to-TCP clearance of 300 mm. All actions consist in a sequence of elementary movements. First, the robot approaches the object. Then it transports it to the target position. Finally, the robot returns to its resting position. There are four elementary movements, the approach towards the object, the transport of an object to a defined position, the transport of an object to a surface and the return towards the rest position.

4.3.1 Approach the object

The paths generated by the algorithm are constructed to preserve the integrity of both the operator and the robot. Hence, it is necessary that robot follows a path as close as possible to the path provided. Figure 4.11(a) exhibits an example of approach towards an object created by the algorithm in orange and the path executed by the robot in blue. Regarding the generated path, only the part accomplished in the tool space is shown. Figure 4.11(b) is a zoomed perspective on the middle waypoint. Since there is a blend radius between sections of the path, the TCP never reaches this waypoint. The green sphere has a radius of 20 mm, the same as the blending radius. Outside the sphere, the two paths should coincide which is not the case.

To understand if this behaviour is influenced by the radius chosen, the same path was executed using different blending radii. The results from this experiment are shown in Figure 4.12. The data collected indicates that contrarily to what is instructed, the robot never follows a vertical approach towards the object. However, the path discrepancy is less than a tenth of the remaining distance to the target position.

Figure 4.13 is a montage of still images taken from [28], a video of the robot approaching an object.

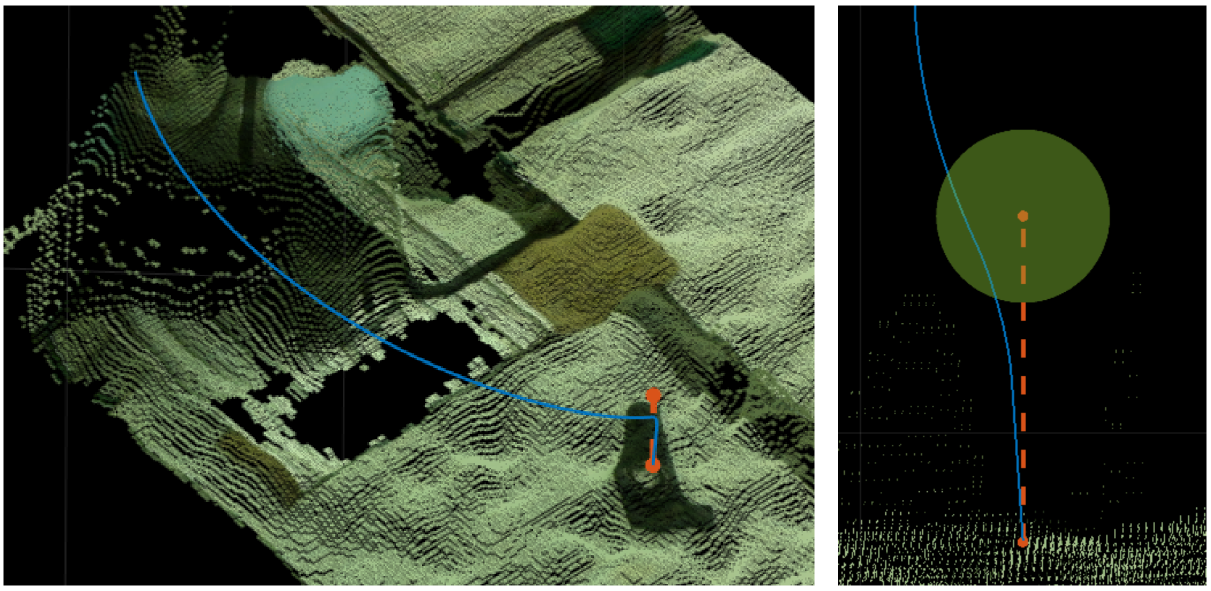

(a) Generated path and actual robot path (b) Zoom on the effect of blend radius

Figure 4.11: Comparison of generated and actual object approach path

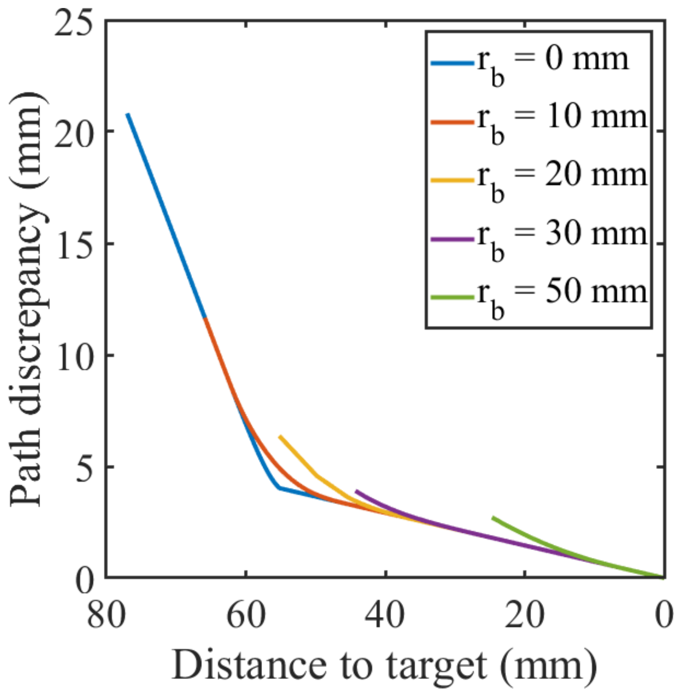


Figure 4.12: Influence of blend radius on path discrepancy

4.3.2 Carry an object to a specified position

Regarding the transport of an object to a defined position, the following is more accurate. Figure 4.14(a) demonstrates that the path followed by the robot in blue coincides with the planned path in orange. Figure 4.14(b) shows two examples of blending in this path. At the bottom, the blending near the waypoint created to guarantee a clearance between the tool and the base of the robot. As can be seen, the blend occurs with the correct radius and the robot followed the planned path outside of the blending region. At the top is the blending that occur when transitioning from vertical to horizontal motion.

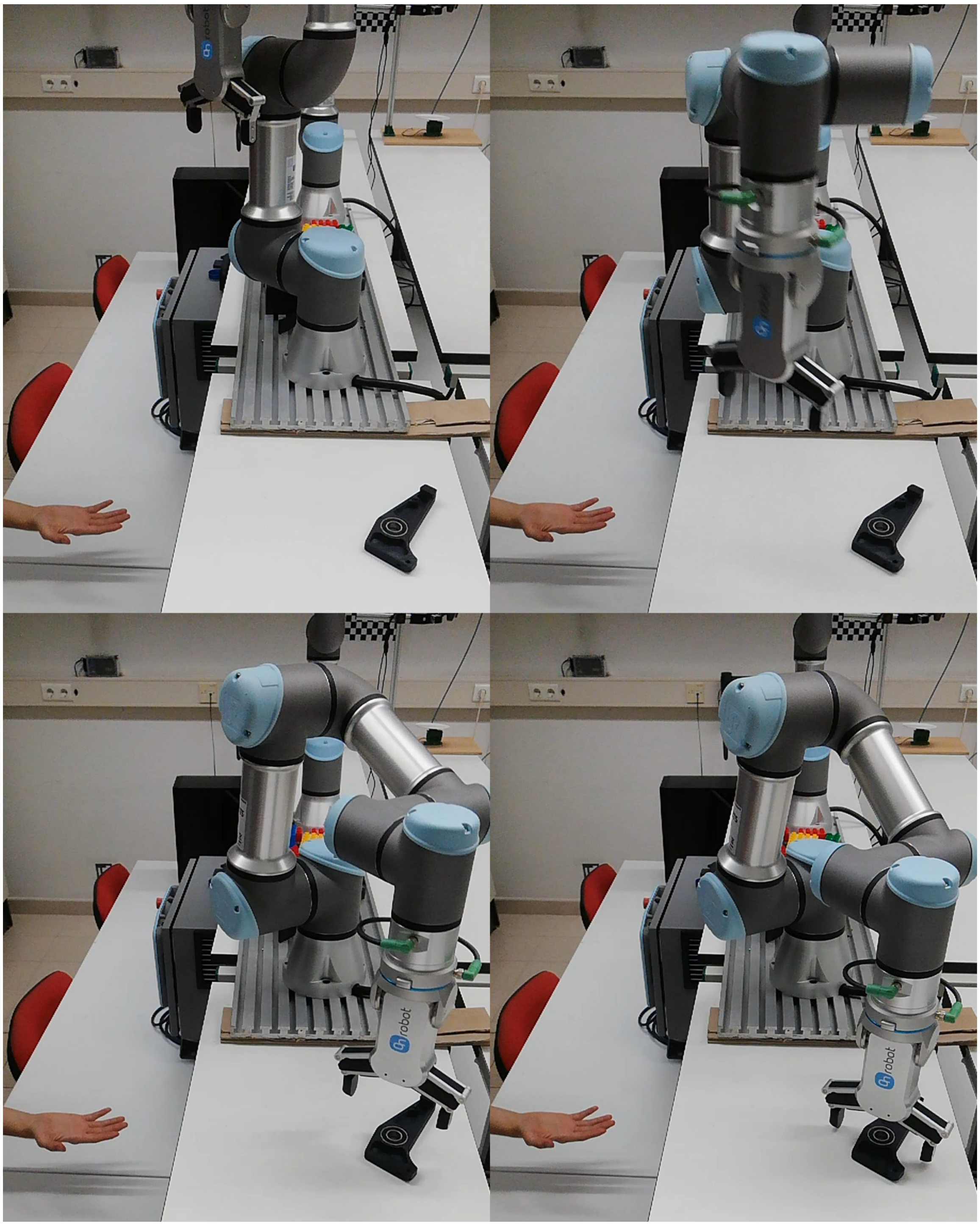
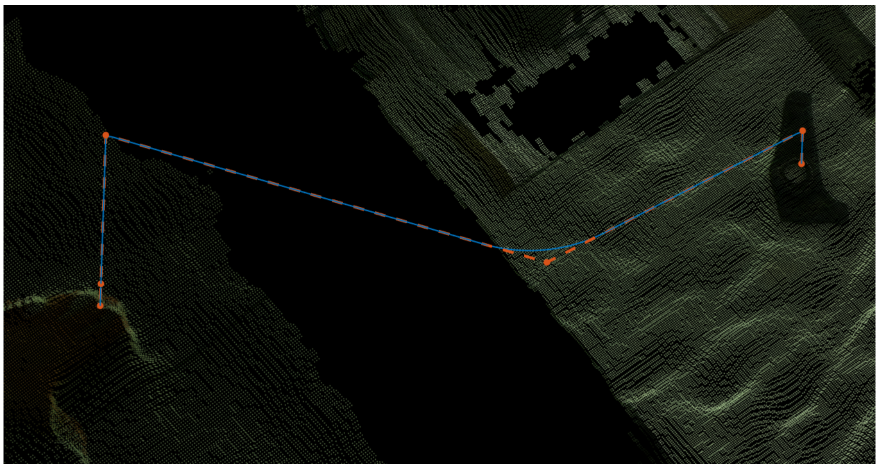


Figure 4.13: Montage from video of robot approaching object

The path following is suitable, however the actual blend radius is less than half of the one specified. This behaviour was observed independently of the blend radius chosen.


(a) Generated and actual robot path

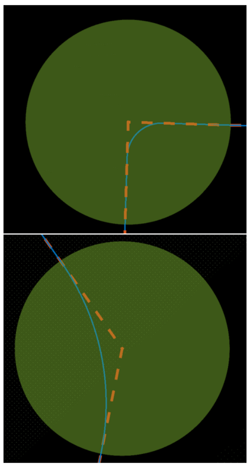

(b) Zoom on the effect of blend radii

Figure 4.14: Comparison of generated and actual deliver in place path

As a consequence of these narrow blends, the robot has to reduce its speed more than it would be necessary had it used the provided blend radii. Figure 4.15 depicts this issue as it can be seen that the TCP speed has to be decreased to around 0.11 m/s to navigate the corners. Additionally, the same path has been implemented using URSim to confirm the consistency between the simulation results and actual robot. The two curves overlap with the only difference being the noise present in the robot's data.

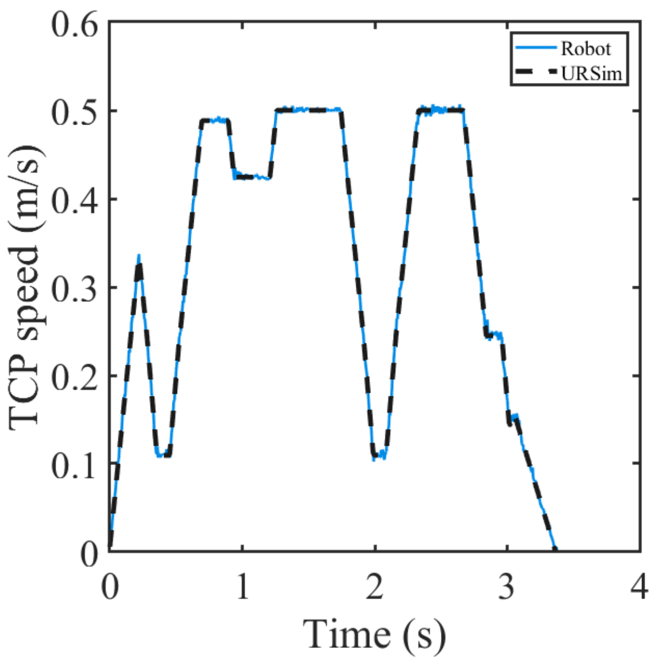


Figure 4.15: TCP speed for path of robot placing object in specified position

A video of the robot can be seen in [29], a montage of still images taken from such video is presented in Figure 4.16.

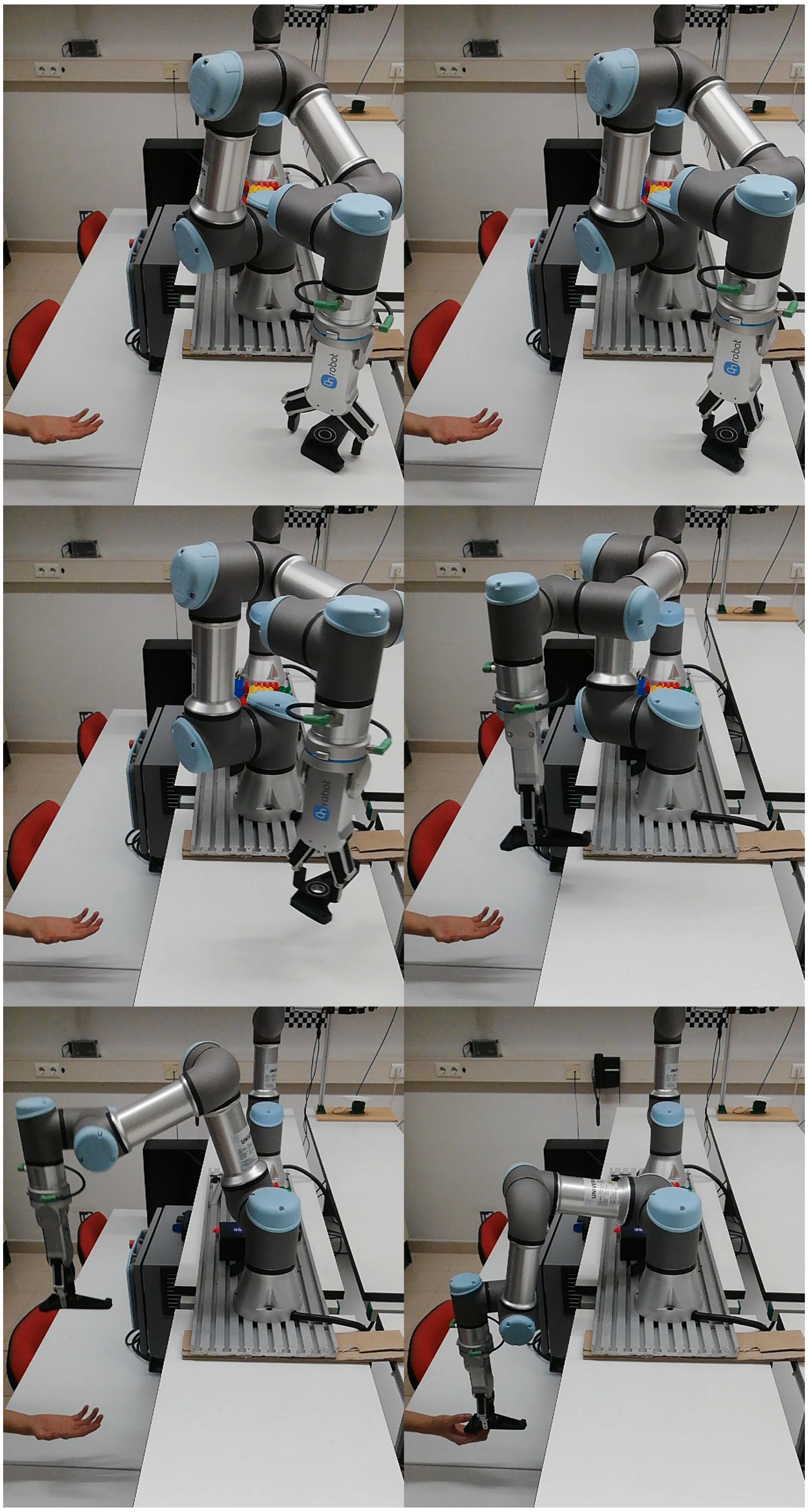


Figure 4.16: Montage from video of robot transporting an object to a specified position

4.3.3 Transport of an object to a surface

This path has the particularity that it cannot be tested resorting to URSim since it is not possible to replicate the presence of the surface. Depositing an object to a surface is similar to the previous one but the drop off location for the object is not completely defined. The final movement has to be done at a reduced speed as not to trigger a safety stop when the object impacts the surface. As can be seen in Figure 4.17, the speed during this stretch is only 50 mm/s.

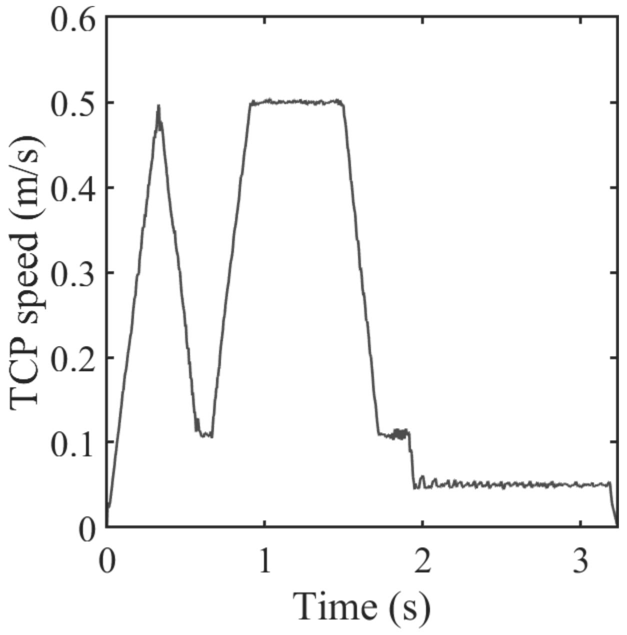
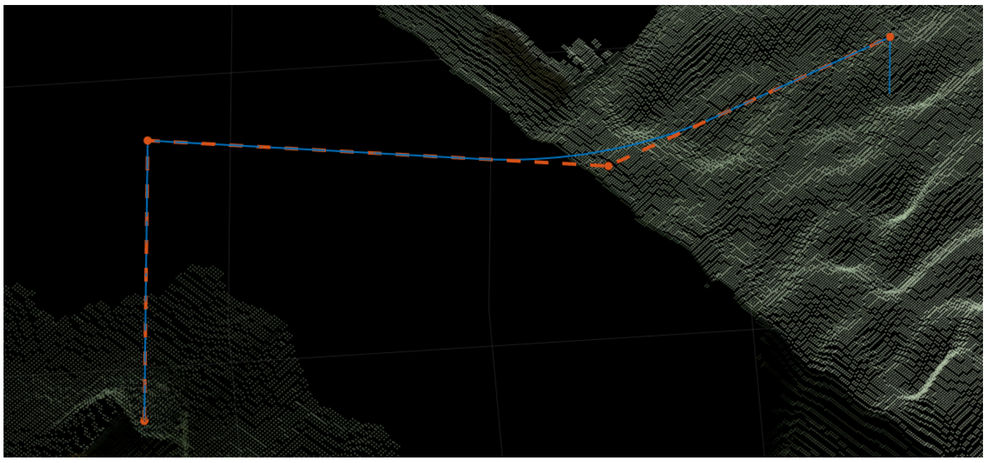
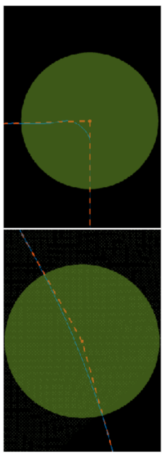


Figure 4.17: TCP speed for path of robot placing object on the table

Similarly to the previously reffered path and as suggested by the speed drop to around 0.11 m/s, the blends between the vertical and horizontal motions are performed with a lower radius than instructed. The other blend is executed with the radius provided, this can be verified in Figure 4.18(b). Apart from the blend radius nonconformity, the robot follows the path closely as exhibited in Figure 4.18(a).



(a) Generated and actual robot path



(b) Zoom on the effect of blend radii

Figure 4.18: Comparison of generated and actual path of robot placing object on the table

Figure 4.19 is a montage of still images taken from [30], a video of the robot following this path.

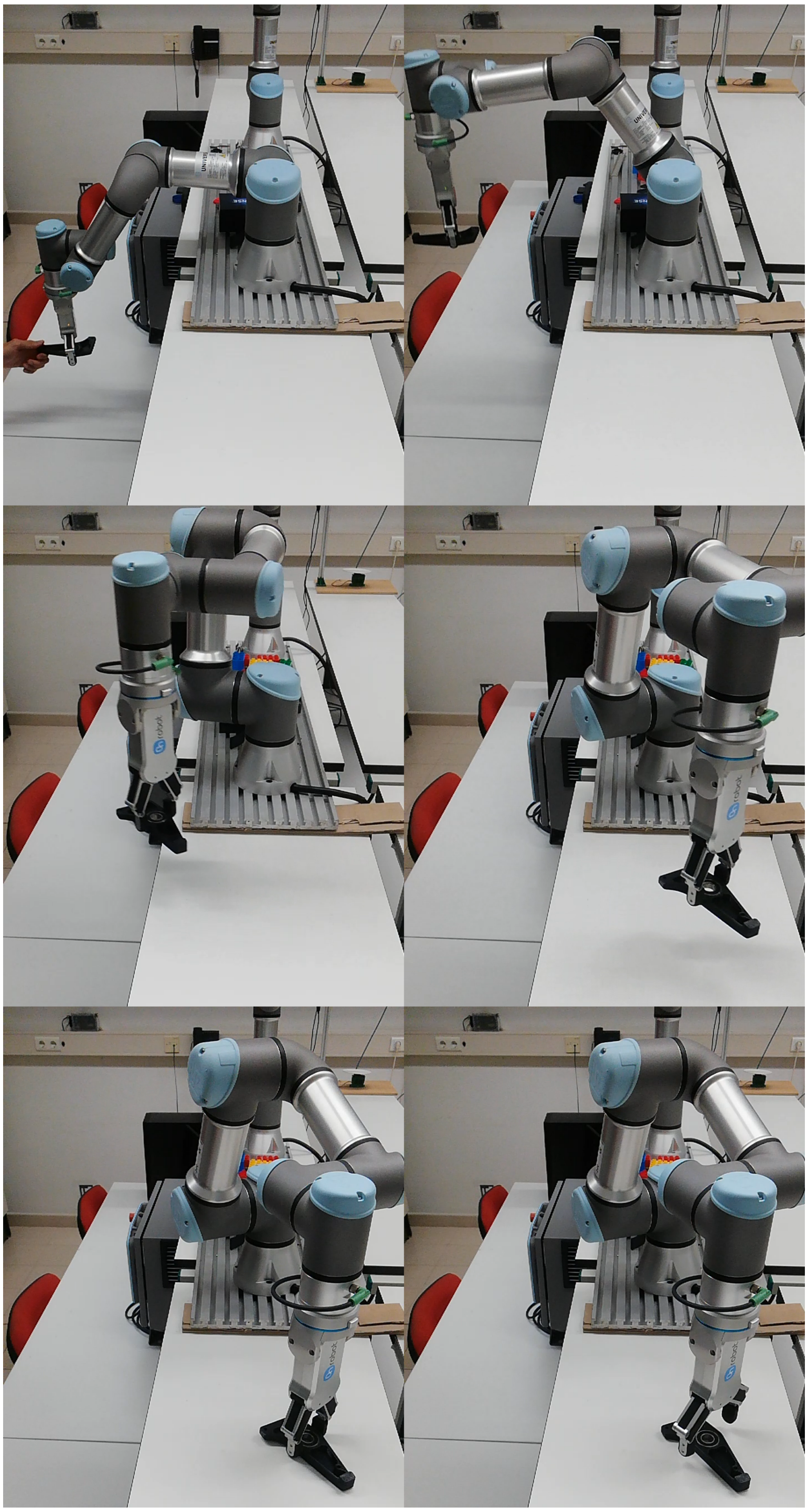
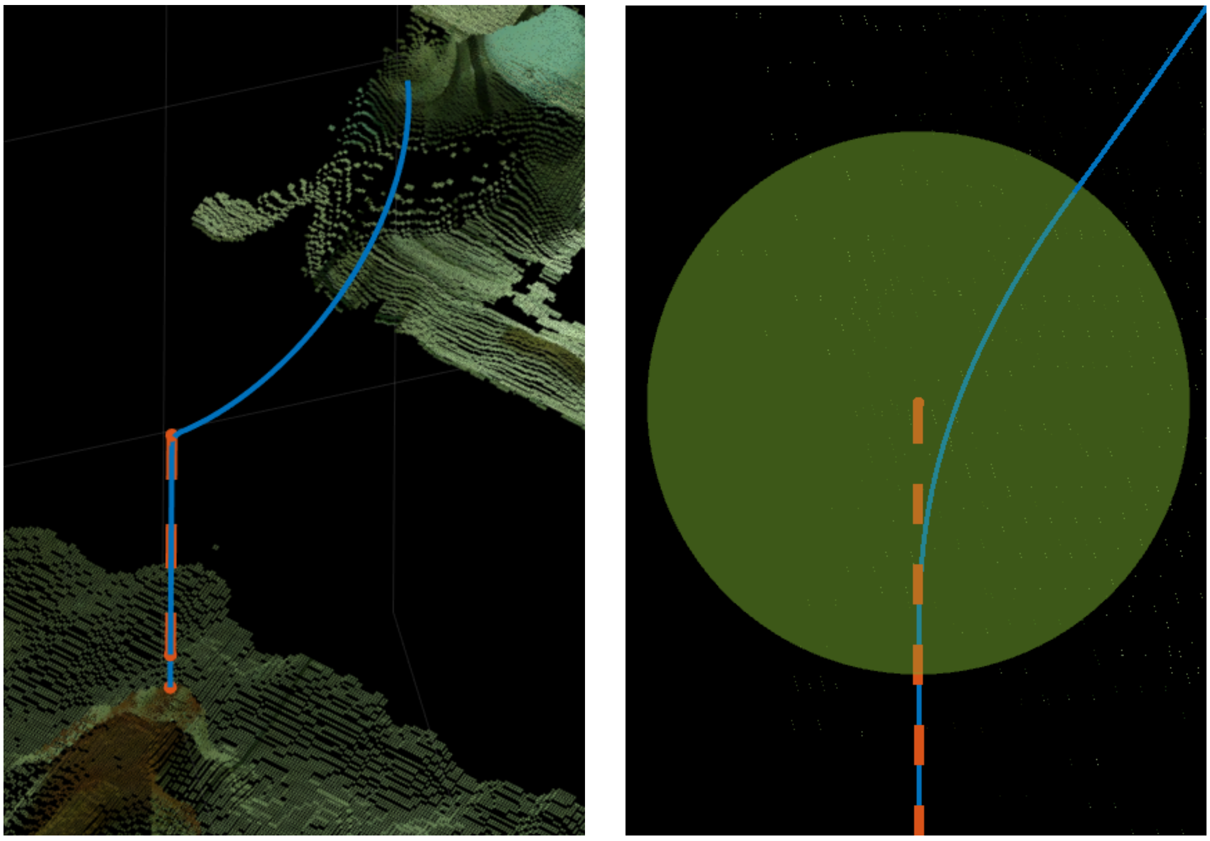


Figure 4.19: Montage from video of robot placing object on the table

4.3.4 Return to rest position

As for the return to rest position, the departure is vertical as instructed and the blend between the motion in Cartesian space and the movement in joint space is correctly executed as can be seen in Figure 4.20(b). Similarly to the first path mentioned, only the section of the path performed in tool space is displayed in Figure 4.20(a).


(a) Generated and actual robot path

(b) Zoom on the effect of blend radius

Figure 4.20: Comparison of generated and actual path of robot returning to rest position

Figure 4.21 is a montage of still images taken from [31], a video of the robot following this path.

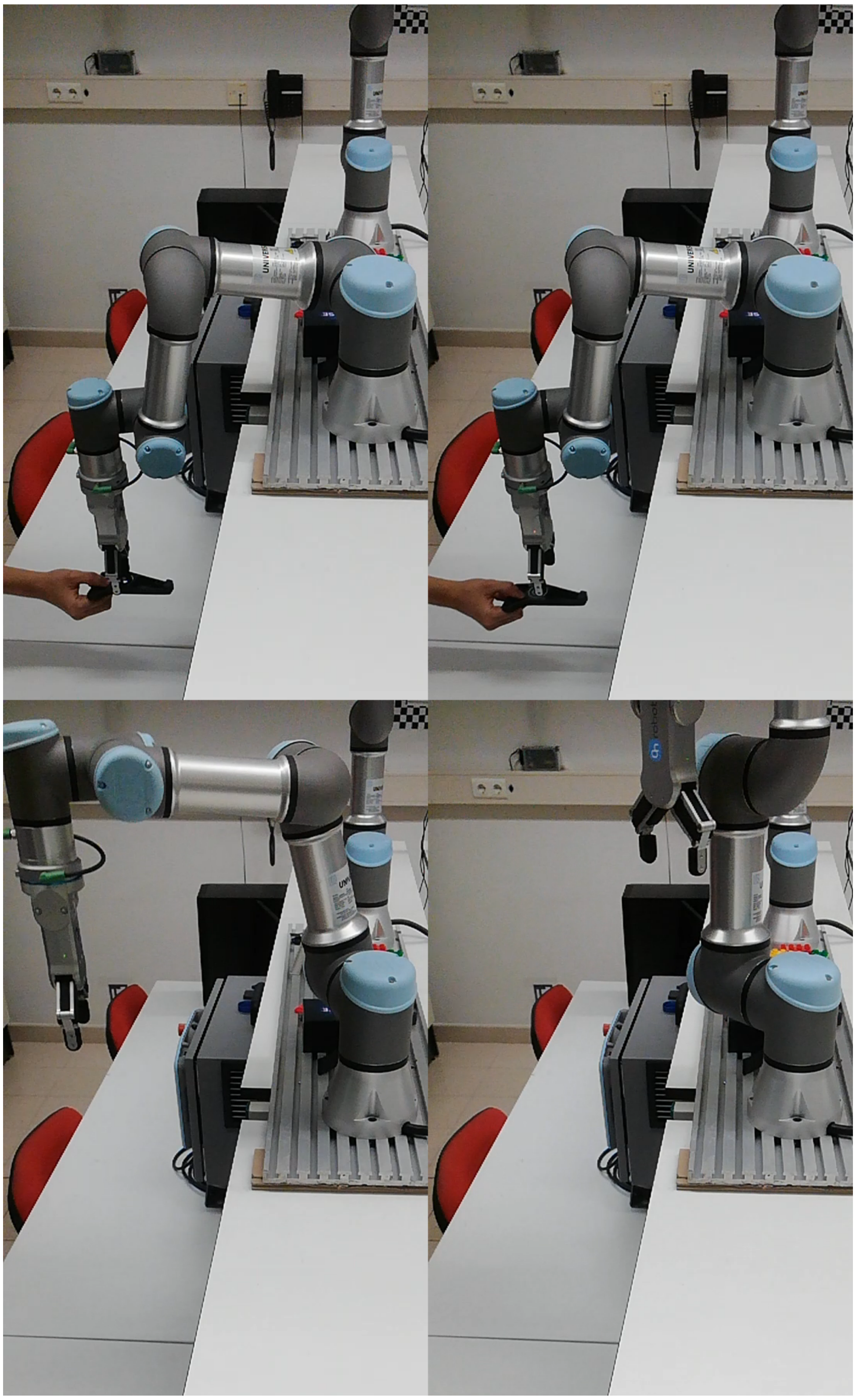


Figure 4.21: Montage from video of robot returning to rest position

4.4 Collaboration Example

In this example, the algorithm developed was used to build a house made of LEGO bricks. First, the bricks needed were scattered on the table. Then, the user requested the parts one by one while simultaneously building the house. A video of the entire activity can be found in [32].

During this experiment, the limits of the algorithm developed were tested. Given the fast computational times observed, the algorithm reacts if the operator change its hand position while the robot is moving to grasp the object. This can be observed in Figure 4.22, the position of the hand is changed while the robot is in motion towards the object and it is delivered to the operator at the new position. Similarly, if the operator moves his hand to a position outside the reach of the robot or even removes his hand from the workspace, the robot will not pick the object and will return to its rest position. These behaviours are illustrated in Figure 4.23 and 4.24 respectively. Figure 4.25 shows a situation where the operator changed the hand that is inside the workspace. In the first frame, the the left hand is inside the workspace. During the motion of the robot towards the object, the operator has withdrawn his left arm and extended his right arm. The robot delivers the object at the correct location regardless of this change. At the end, the operator asked the robot to collect the finished product and take it back to the tabletop as depicted in Figure 4.26.

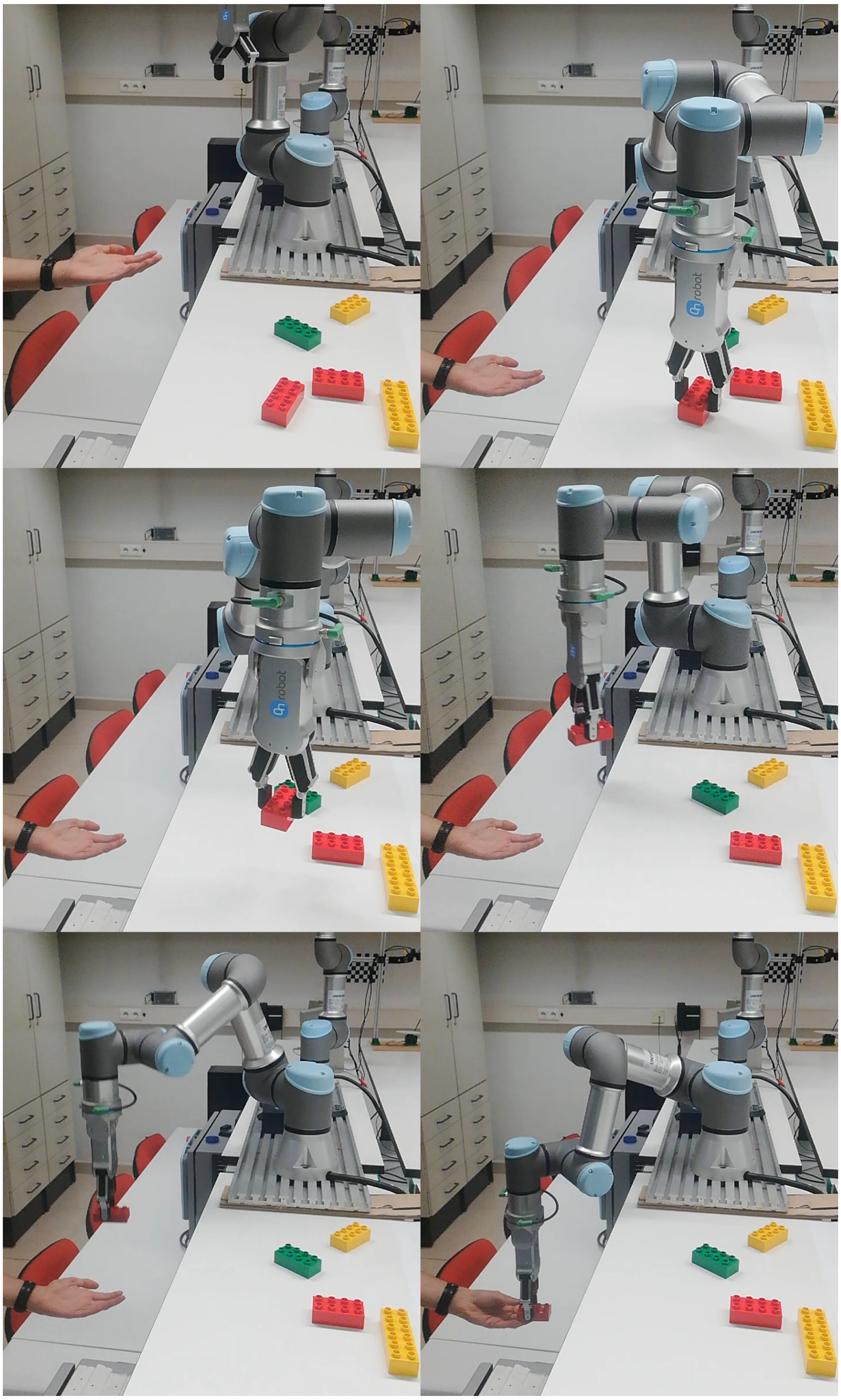


Figure 4.22: Operator changes hand position

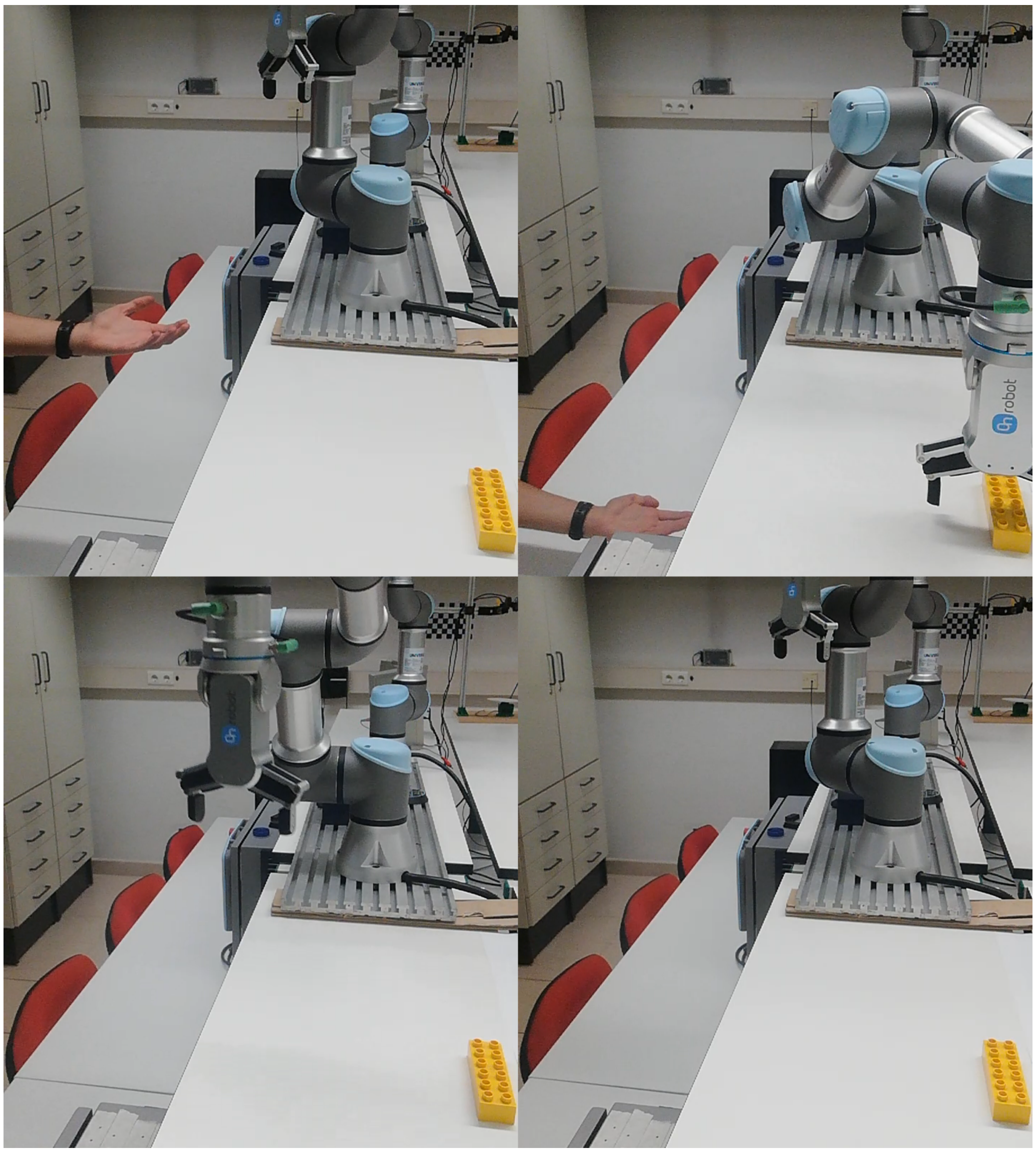


Figure 4.23: Operator changes hand position to an unreachable position

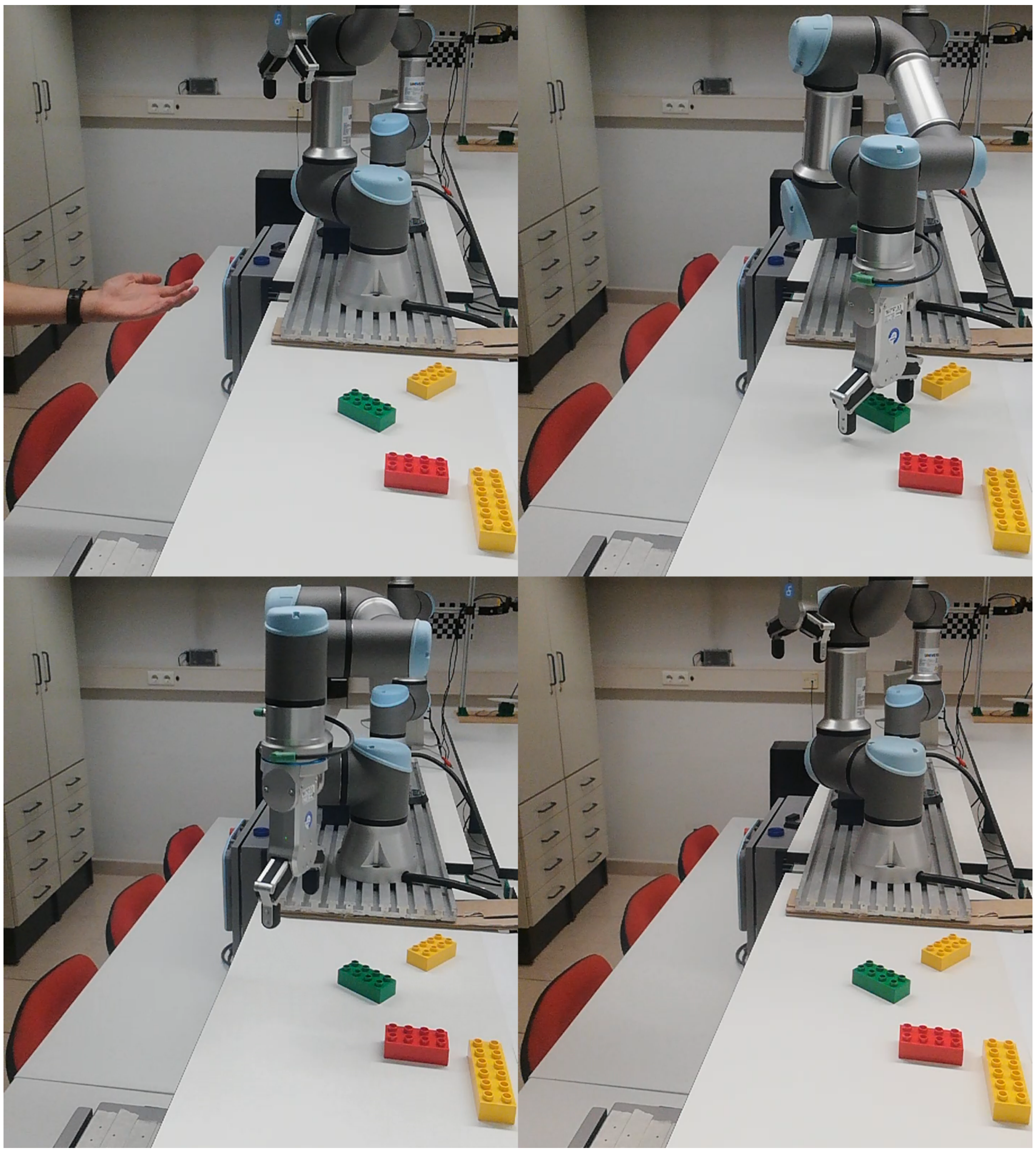


Figure 4.24: Operator removes his hand from the workspace

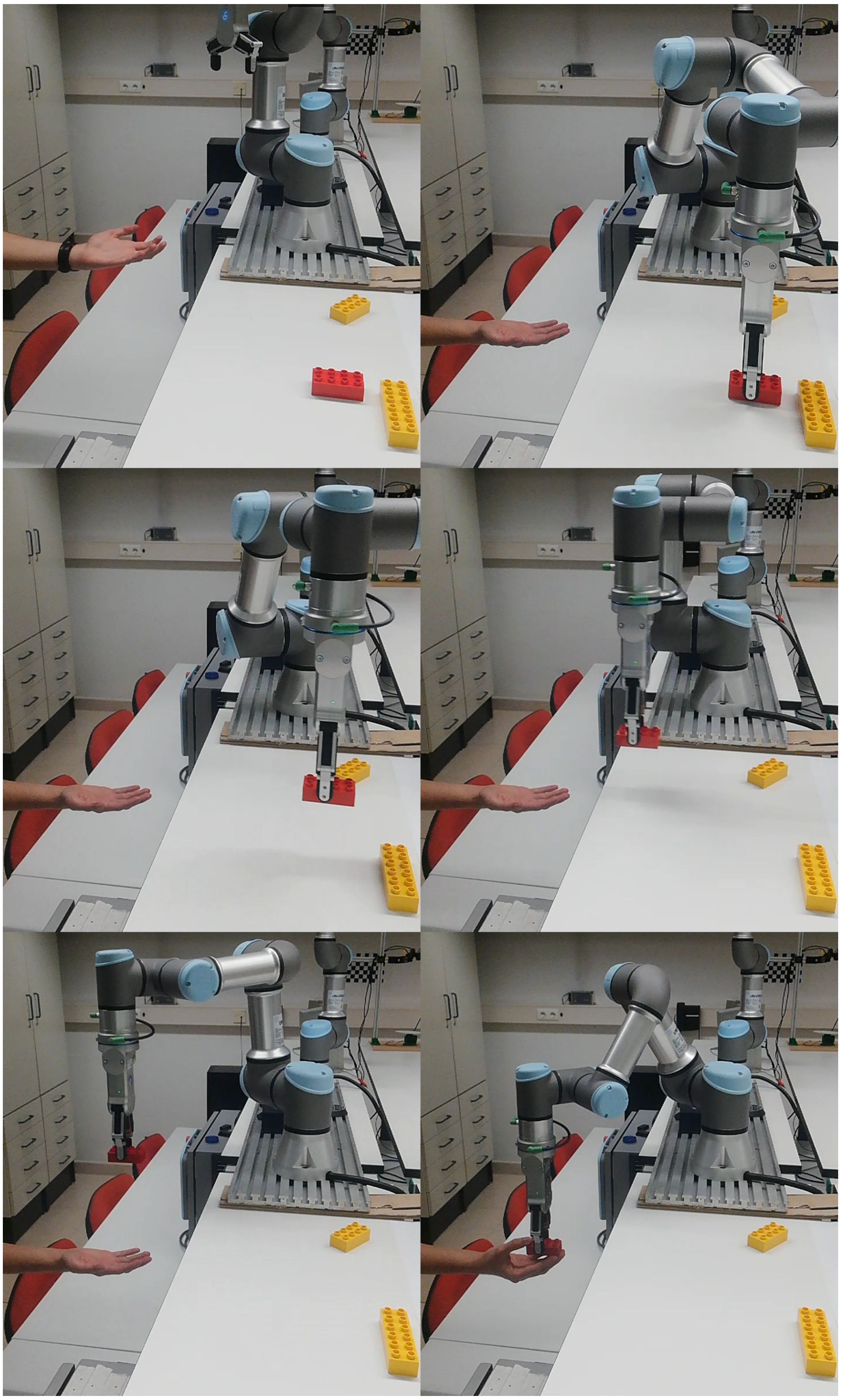


Figure 4.25: Operator changes receiving hand

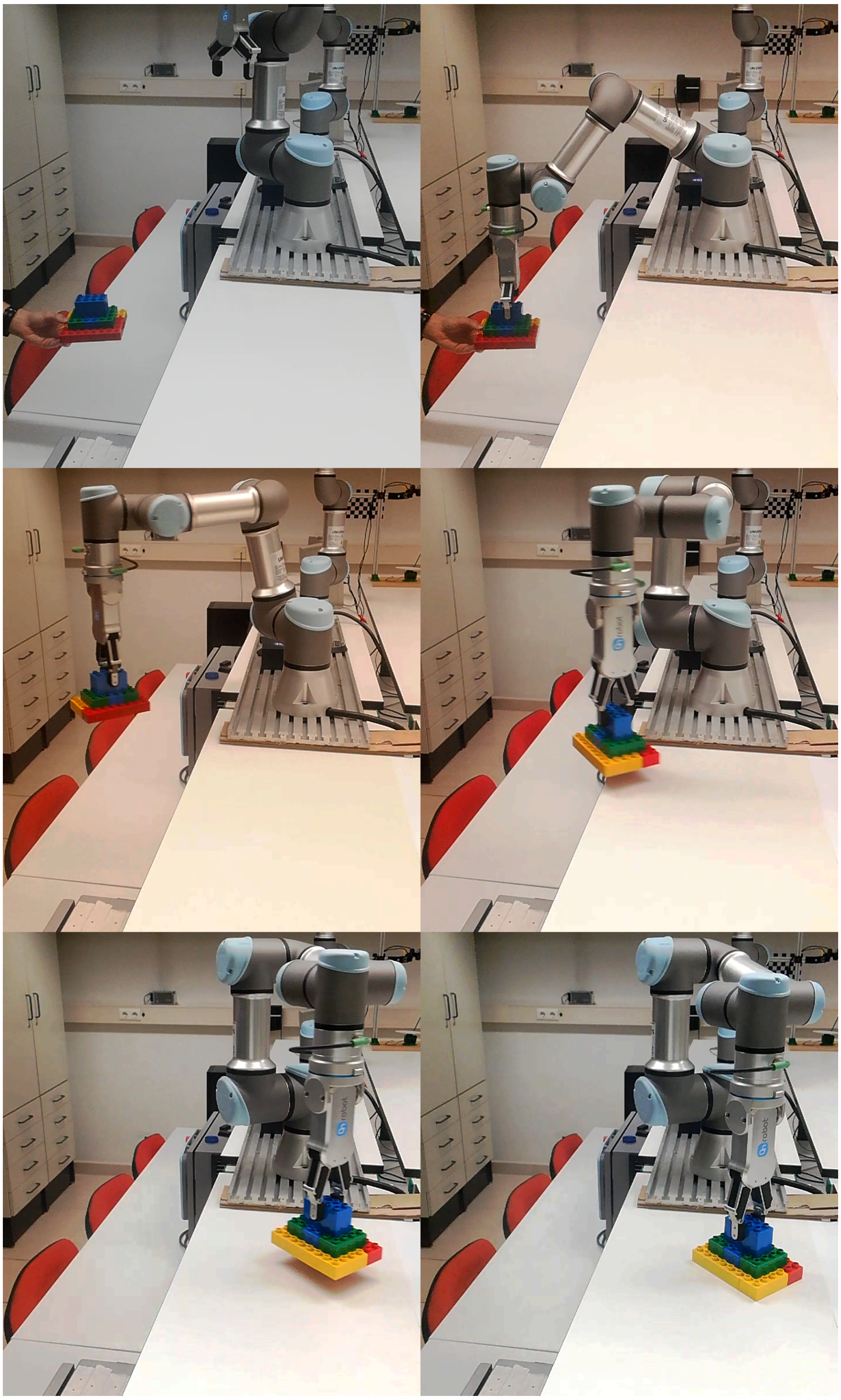


Figure 4.26: Robot placing finished product on the table

Chapter 5

Conclusions

5.1 Achievements

The present work focused on developing an algorithm to integrate a collaborative robot and a depth camera. Regarding the arrangement of the material used on the workbench, the position of the robot is as far away from the operator as possible, but still allowing a shared workspace dimension ample enough for the objective proposed. As for the camera, its elevated position allows an overlook over the entire working range of the robot. However it proved less accurate than desirable in terms of depth. This could have been mitigated using another fixed camera with a different point of view or a camera attached to the robot. Concerning the lighting, having only the farthest light switched on during the calibration proved to be the best option, but this would origin too many shadows in the shared workspace during normal operation. Apart from the calibration phase, having all three ceiling light switched gave the best results, even though shadows could still be noticeable when the arms of the operator were too close to the table.

The procedure created to estimate the transformation matrix between the camera and robot coordinates generates heterogeneous results. Although fast, it runs in under 2 seconds, the results display a high variability.

The desired accuracy was achieved when looking at the XY plane, with error bellow 15 mm for the best calibration obtained. Regarding depth, the accuracy is sufficient to achieve the proposed objectives when considering object on the tabletop but was poorer when considering the shared workspace, with an error up to 30 mm.

The paths generated allow a correct handle of the object from and to the hands of the operator, ensuring correct separation between objects being held, the robot and the environment.

5.2 Conclusions

The final example shows collaboration has been achieved. The algorithm correctly identifies the operator and the objects using the depth camera and the robot is able to answer in real time to the orders of the operator. The overall system provides sufficient accuracy for the task and is able to handle the operator changing hand position mid operation or even removing it from the workspace.

The applications for such systems are numerous, as it can be used anywhere an assistant is needed to manipulate and transport tools or other objects. Besides the examples of operating rooms, laboratories and cleanrooms mentioned earlier, it is also possible to use such system on a mobile platform. Applications such as clearing a table after a meal or retrieving tools and parts in a workshop are also plausible.

5.3 Future Work

After the tests and experiments carried out with the developed algorithm some improvements became can be suggested, such as:

- Considering the variability observed when estimating the transformation matrix, it would be relevant to design a procedure to verify the calibration obtained and recalibrate if the results are not adequate.
- The results collected seem to indicate a intrinsic depth error from the camera. Further test should be conducted to evaluate this claim and if it is confirmed, this error should be taken into consideration when generating paths for the robot.
- The objects have been approached vertically but this is not mandatory. Allowing the robot to approach the objects at an angle would increase the maximum reach.
- Some actions to be performed by the robot are selected by the operator using a keyboard, this occupies the hands of the operator. For this reason, the system could be further developed to allow the operator to have full control without resorting to a keyboard. Two possibilities emerge as the most promising to attend to this drawback, voice and gesture control.

Bibliography

[1] J. G. C. Devol. Programmed article transfer, June 13 1961. US Patent 2,988,237.

[2] International federation of robotics. `https://ifr.org/`. Accessed: 2020-12-03.

[3] T. Arai, R. Kato, and M. Fujita. Assessment of operator stress induced by robot collaboration in assembly. *CIRP annals*, 59(1):5–8, 2010.

[4] W. Bauer, M. Bender, M. Braun, P. Rally, and O. Scholtz. Lightweight robots in manual assembly—best to start simply. *Examining companies' initial experiences with lightweight robots, Stuttgart*, pages 1–32, 2016.

[5] U. robots. Universal robots' ur5e rings the nyse closing bell. `https://youtu.be/XaZoTlgjdIg`. Accessed: 28-11-2020.

[6] A. F. Sciences. Robotic co-pilot flies and lands a simulated boeing 737. `https://youtu.be/om18cOWFL3Q`. Accessed: 28-11-2020.

[7] A. Bhandari, M. Feigin, S. Izadi, C. Rhemann, M. Schmidt, and R. Raskar. Resolving multipath interference in kinect: An inverse problem approach. In *SENSORS, 2014 IEEE*, pages 614–617. IEEE, 2014.

[8] L. Li, S. Xiang, Y. Yang, and L. Yu. Multi-camera interference cancellation of time-of-flight (tof) cameras. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 556–560. IEEE, 2015.

[9] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim. Shake'n'sense: reducing interference for overlapping structured light depth cameras. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1933–1936, 2012.

[10] K. Atsuta, K. Hamamoto, S. Kondo, et al. A robust stereo matching method for low texture stereo images. In *2009 IEEE-RIVF International Conference on Computing and Communication Technologies*, pages 1–8. IEEE, 2009.

[11] R. Nevatia. Depth measurement by motion stereo. *Computer Graphics and Image Processing*, 5 (2):203–214, 1976.

[12] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun. Dense monocular depth estimation in complex dynamic scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4058–4066, 2016.

[13] M. Inaba, T. Hara, and H. Inoue. A stereo viewer based on a single camera with view-control mechanisms. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 3, pages 1857–1865. IEEE, 1993.

[14] A. Saxena, S. H. Chung, A. Y. Ng, et al. Learning depth from single monocular images. In *NIPS*, volume 18, pages 1–8, 2005.

[15] Y.-h. Zhang, W. Wei, D. Yu, and C.-w. Zhong. A tracking and predicting scheme for ping pong robot. *Journal of Zhejiang University SCIENCE C*, 12(2):110–115, 2011.

[16] A. F. Ribeiro, C. Machado, I. Costa, and S. Sampaio. Patriarcas/minho football team. 1999.

[17] C. Smith and H. I. Christensen. Using cots to construct a high performance robot arm. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4056–4063. IEEE, 2007.

[18] J. Abouaf. Trial by fire: teleoperated robot targets chernobyl. *IEEE Computer Graphics and Applications*, 18(4):10–14, 1998.

[19] L. S. Scimmi, M. Melchiorre, S. Mauro, and S. P. Pastorelli. Implementing a vision-based collision avoidance algorithm on a ur3 robot. In *2019 23rd International Conference on Mechatronics Technology (ICMT)*, pages 1–6. IEEE, 2019.

[20] M. Melchiorre, L. S. Scimmi, S. Mauro, and S. P. Pastorelli. Vision-based control architecture for human–robot hand-over applications. *Asian Journal of Control*, 23(1):105–117, 2021.

[21] A. You, F. Sukkar, R. Fitch, M. Karkee, and J. R. Davidson. An efficient planning and control framework for pruning fruit trees. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3930–3936. IEEE, 2020.

[22] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. 1955.

[23] O. Rodrigues. Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire. *Journal de mathématiques pures et appliquées*, 5(1):380–440, 1840.

[24] R. I.-R. BT et al. Studio encoding parameters of digital television for standard 4: 3 and wide-screen 16: 9 aspect ratios. 2011.

[25] Client interfaces. `https://s3-eu-west-1.amazonaws.com/ur-support-site/16496/Client_Interfaces_1.1.3.xlsx`. Accessed: 5-11-2020.

[26] Y. Altman. export_fig. https://github.com/altmany. GitHub. Retrieved August 19, 2021.

[27] S. Yanez-Pagans. Plot it,...but plot it well! https://github.com/sergioyapa. GitHub. Retrieved August 19, 2021.

[28] T. Fernandes. Video of the approach towards an object. https://vimeo.com/638588434, 2021.

[29] T. Fernandes. Video of carrying an object to a specified position. https://vimeo.com/638601757, 2021.

[30] T. Fernandes. Video of carrying an object to a surface. https://vimeo.com/638601908, 2021.

[31] T. Fernandes. Video return to rest position. https://vimeo.com/638602030, 2021.

[32] T. Fernandes. Video of collaboration example. https://vimeo.com/641493564, 2021.