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

Tomás Moreira Furtado Carvalho Fernandes
tomasmcfernandes@tecnico.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

December 2021

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 could 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 were acceptable for the task at hand.

Keywords: collaborative robotics, computer vision, depth camera, pick and place

1. Introduction

The first industrial robot called UNIMATE was introduced in 1961 [7]. Since then, the majority of robots used to be confined in a cage where humans were not allowed to enter to prevent accidents. More recently, collaborative robots were introduced. The fundamental difference is that they halt their movement if contact is detected. 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.

1.1. 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. 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 [12], and the UR3 who was capable of performing a Boeing 737's landing procedure [14].

1.2. 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 [3]. Another disadvantage is that the presence of other cameras of the same type can harm the results obtained.

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 thee 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 [5].

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 [2]. 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 [10]. 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.

In addition, machine learning with supervised learning can be employed [13]. The training requires images with their corresponding depth maps to be provided.

1.3. 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 to automate robots, such as ping-pong [16] to track the ball or football [11] 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 [15], 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 [1] is a more meaningful example of what can be achieved when combining computer vision and robotics.

2. Background

The majority of industrial robots can be decomposed in several 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 [6]. 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. Transformation matrix

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

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

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. 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:

$$\boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}, \tag{3}$$

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

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^\top. \tag{4}$$

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.

2.3. 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. 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.

2.4. 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. 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.

2.5. Unit quaternion

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

$$Q = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}, \tag{5}$$

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

$$Q = \begin{bmatrix} cos(\theta/2) & \mathbf{u}sin(\theta/2) \end{bmatrix}. \tag{6}$$

To add two rotations represented by two quaternions Q and J, we use the following formula:

$$S = \begin{bmatrix} q_wj_w - q_xj_x - q_yj_y - q_zj_z \\ q_wj_x + q_xj_w + q_yj_z - q_zj_y \\ q_wj_y - q_xj_z + q_yj_w + q_zj_x \\ q_wj_z + q_xj_y - q_yj_x + q_zj_w \end{bmatrix}^\top. \tag{7}$$

2.6. 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 6 to each of the elemental rotations, resulting in:

$$\begin{cases} Q_X = [cos(\alpha/2) \quad sin(\alpha/2) \quad 0 \quad 0], \\ Q_Y = [cos(\beta/2) \quad 0 \quad sin(\beta/2) \quad 0], \\ Q_Z = [cos(\gamma/2) \quad 0 \quad 0 \quad sin(\gamma/2)]. \end{cases} \tag{8}$$

The axis-angle representation can be retrieved by inverting equation 6:

$$\begin{cases} \theta = atan2(\left\| \begin{bmatrix} q_x & q_y & q_z \end{bmatrix} \right\|, \, q_w), \\ \mathbf{u} = \dfrac{\begin{bmatrix} q_x & q_y & q_z \end{bmatrix}}{\left\| \begin{bmatrix} q_x & q_y & q_z \end{bmatrix} \right\|}, \end{cases} \tag{9}$$

where the quaternion Q is obtained by adding the three elemental rotations in the order ZYX according to equation 7

2.7. Rotation from one vector into another

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

$$\begin{aligned} \mathbf{u} &= \mathbf{n} \times \mathbf{v} \\ &= \begin{bmatrix} n_yv_z - n_zv_y \\ n_zv_x - n_xv_z \\ n_xv_y - n_yv_x \end{bmatrix}, \end{aligned} \tag{10}$$

and the angle can be determined knowing that

$$\|\mathbf{n} \times \mathbf{v}\| = sin(\theta), \tag{11}$$

and

$$\mathbf{n} \cdot \mathbf{v} = cos(\theta), \tag{12}$$

which leads to:

$$\theta = atan2(\|\mathbf{n} \times \mathbf{v}\|, \, \mathbf{n} \cdot \mathbf{v}). \tag{13}$$

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

$$\mathbf{R} = \mathbf{I} + sin(\theta)\mathbf{K_u} + (1 - cos(\theta))\mathbf{K_u}^2, \tag{14}$$

where $\mathbf{K}$ is the cross product matrix of $\mathbf{u}$ defined as

$$\mathbf{K_u} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}. \tag{15}$$

2.8. Colour spaces

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

$$\begin{cases} V = \dfrac{max(R,G,B)}{255}; \\ S = \begin{cases} \dfrac{\Delta}{V} & \text{if } V > 0; \\ 0 & \text{if } V = 0; \end{cases} \\ H = \dfrac{1}{6} \times \begin{cases} \dfrac{G-B}{\Delta} \bmod 6 & \text{if } R > G, B; \\ \dfrac{B-R}{\Delta} + 2 & \text{if } G > R, B; \\ \dfrac{R-G}{\Delta} + 4 & \text{if } B > R, G; \\ 0 & \text{if } V = 0 \text{ or } S = 0, \end{cases} \end{cases} \tag{16}$$

with

$$\Delta = max(R,G,B) - min(R,G,B). \tag{17}$$

One of the formulas that have been defined to convert an RGB image to a greyscale image is:

$$\mathbf{G} = 0.299\mathbf{R} + 0.587\mathbf{G} + 0.114\mathbf{B}. \tag{18}$$

This formula was defined by the International Telecommunication Union and each colour has a different contribution due to the way humans perceive light [4].

Binary images can be obtained by thresholding an image using

$$\mathbf{B} = \begin{cases} 0\,, & if \mathbf{I} \leq \mathbf{t}, \\ 1\,, & if \mathbf{I} > \mathbf{t}, \end{cases} \tag{19}$$

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.

3. Implementation

This thesis was developed using a UR3 collaborative robot from Universal Robots and a D435 depth camera from Intel. The integration is made through a desktop computer running MATLAB version 2019a. 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.

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.

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. 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.

3.2. 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. A tool change happens in under a minute due to a quick release system in the 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.

3.3. Intel RealSense D435

The RealSense D435 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, a RGB camera and an IR projector. The IR cameras are used at 848x480 resolution. This is the resolution that 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.

Before usage, it is essential to calibrate the cameras. To do that, Intel provides a calibration software that estimates the intrinsic parameters of the IR cameras, the baseline, the principal points and the rotation matrices between the cameras' and the world's frame. The calibration software uses a pattern, that can be printed or displayed on a mobile phone screen, where an app ensures the correct pattern size is displayed.

3.4. Software Architecture

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.

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.

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.

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, where it 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.

4. 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. Hand segmentation

The hand segmentation algorithm uses a mix of 3D and colour cues. Figure 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.

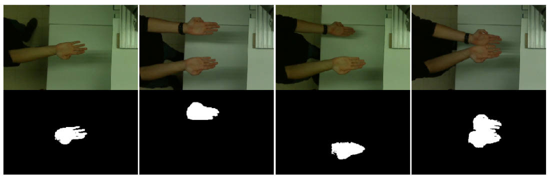


Figure 1: Examples of hand segmentation results

4.2. Segmentation of objects in the hand

Figure 2 depicts the original RGB images and the corresponding object segmentation below for six different cases. The first two images 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 frame 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 image, the sleeve and the object are detached from one another. Therefore, the object is correctly identified. However, in the fifth image the sleeve is overlapping the object. The algorithm is deceived into considering as the sleeve as part of the object. Finally, the rightmost image shows how the hand being too close to the table can cause issues. The shadow of the hand makes the algorithm recognize part of the table as belonging to the object.

4.3. Segmentation of objects on the table

Figure 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

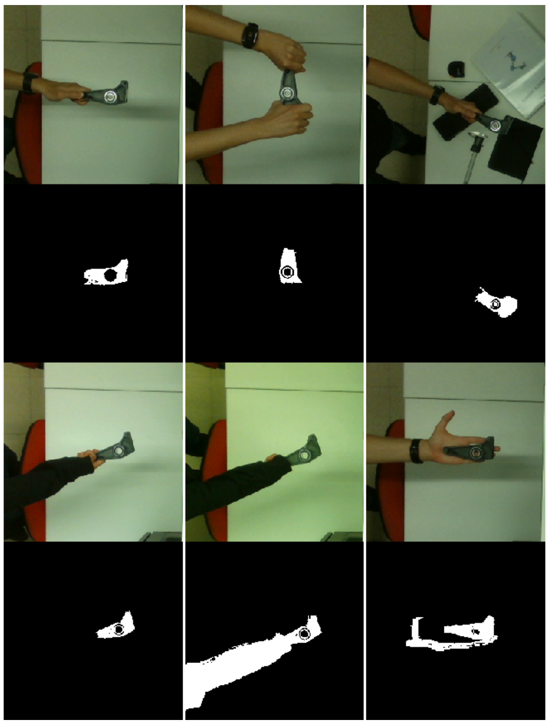


Figure 2: Examples of segmentation for objects in hand

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.

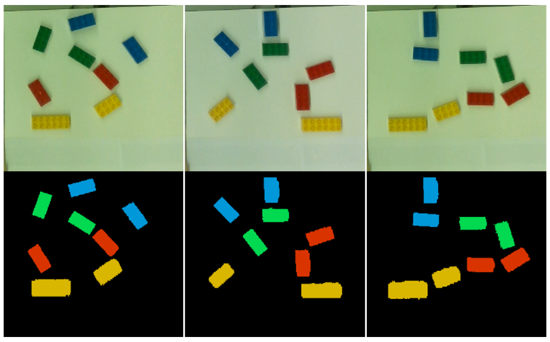


Figure 3: Examples of object segmentation results

4.4. 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. The median running time of the methods was determined using the inbuilt function *timeit*. The fastest method was the hand segmenting, with a median time of 23 ms, then the object in hand segmenting with 25 ms. Retrieving a frame from the camera and segmenting objects on the table were the slowest methods with a median running time of 36 ms. The latter greatly depends on the number of objects on the table, each additional object results in an increase of approximately 4 ms in processing time. 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.

5. 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.

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

The errors were split in error in the XY plane and error in the depth error, as shown in Figure 4. 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.

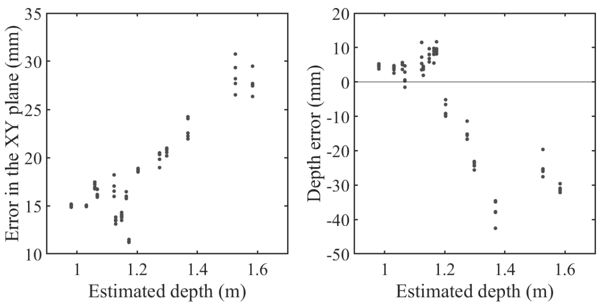


Figure 4: First accuracy results

The calibration process was examined, and found to have a considerable variability. To attempt at lowering the obtained errors, two other 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.

One of the proposed 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. To adjust for the reduced incident light, the exposure times of the RGB and IR cameras were increased. 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.

Another solution proposed was to add texture to the tabletop by placing a pattern on it. However, this approach was considerably worse. Figure 5 compares the best achieved result from this methods and the best calibration previously obtained.

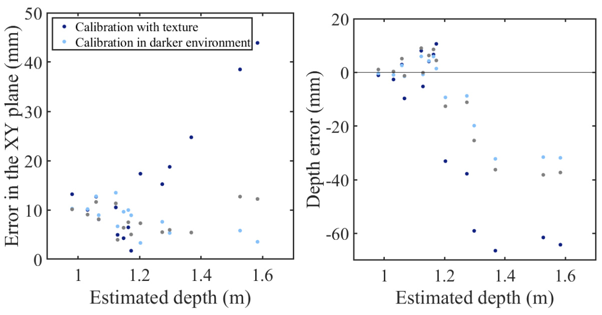


Figure 5: Comparison of calibration methods

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

$$\boldsymbol{T}_R^C = \begin{bmatrix} -0.7069 & 0.7069 & -0.0246 & 0.2332 \\ 0.7071 & 0.7071 & 0.0016 & 0.0544 \\ 0.0186 & -0.0163 & -0.9997 & 1.0936 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}. \quad (20)$$

5.1. 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. The latter 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 6 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 Figure 5

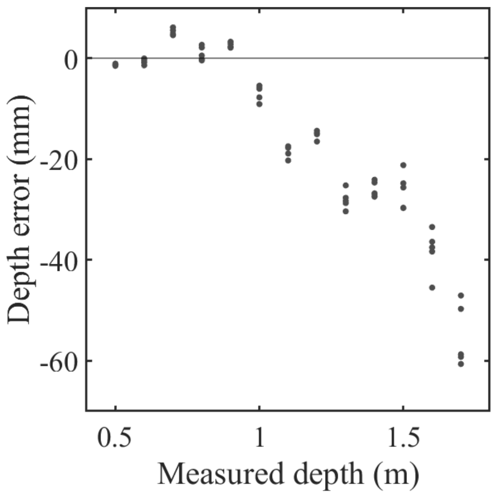


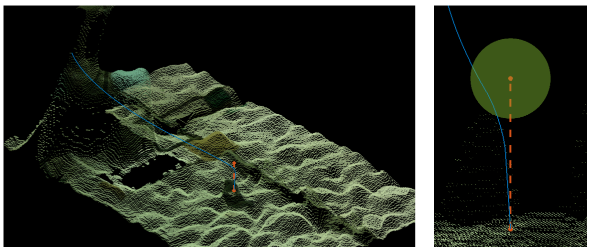
Figure 6: Depth error estimate

6. 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. Since the last two are almost an exact reverse of the first two, only the first two will be evaluated.

6.1. 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 7(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 7(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. Further testing has shown that this phenomena happens for all blend radii tested.



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

Figure 7: Comparison of generated and actual object approach path

Figure 8 is a montage of still images taken from [9], a video of the robot following approaching an object.

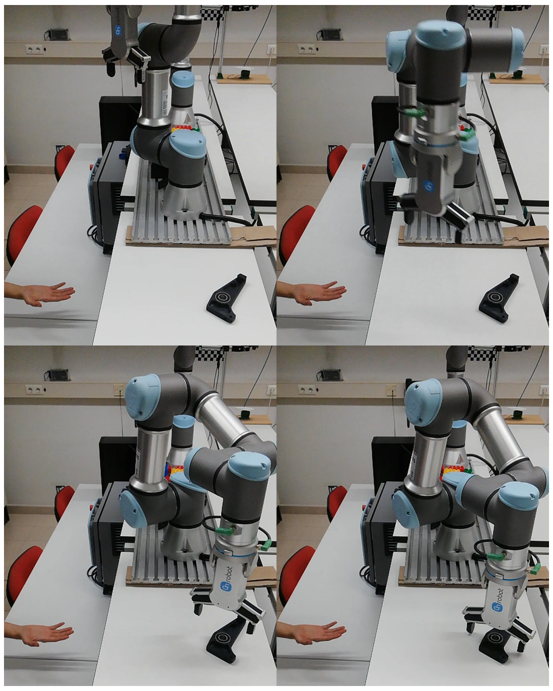
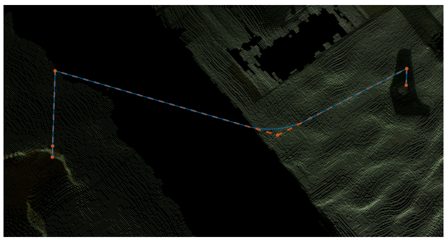


Figure 8: Montage from video of robot approaching object

6.2. Carry an object to a specified position

Regarding the transport of an object to a defined position, the following is more accurate. Figure 9(a) demonstrates that the path followed by the robot in blue coincides with the planned path in orange. Figure 9(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. 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. 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.


(a) Generated and actual robot path

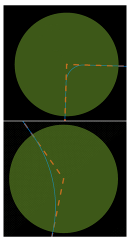

(b) Zoom on the effect of blend radii

Figure 9: Comparison of generated and actual deliver in place path

A video of the robot can be seen in [8], a montage of still images taken from such video is presented in Figure 10.

7. Conclusions

Regarding the setup arrangement on the workbench, the position of the robot is ideal since it 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. 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. 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 to estimate the transformation matrix between the camera and robot coordinates proved to be unreliable. 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 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.

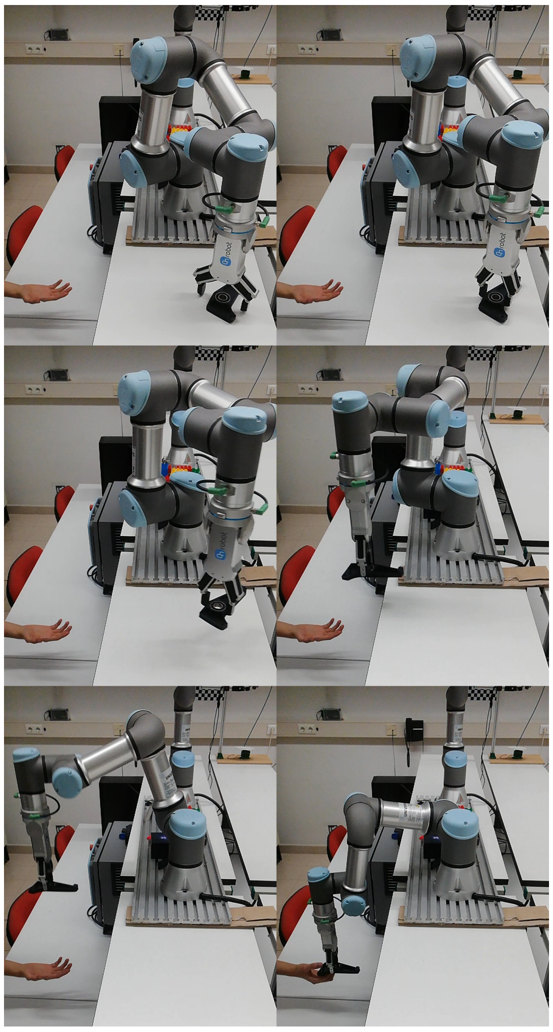


Figure 10: Montage from video of robot transporting an object to a specified position

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.

References

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

[2] K. Atsuta, K. Hamamoto, S. Kondo, et al. A robust stereo matching method for low texture stereo images. In *2009 IEEE-RIVF Interna-*

tional Conference on Computing and Communication Technologies, pages 1–8. IEEE, 2009.

[3] 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.

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

[5] 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.

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

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

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

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

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

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

[12] U. robots. Universal robots' ur5e rings the nyse closing bell.

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

[14] A. F. Sciences. Robotic co-pilot flies and lands a simulated boeing 737.

[15] 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.

[16] 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.