POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Robust Shape Tracking of a Deformable Linear Object Manipulated by a Dual-Arm Robot

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Alessio Russo**

Student ID: 10618144
Advisor: Prof. Paolo Rocco
Co-advisors: Andrea Monguzzi, Prof. Andrea Maria Zanchettin
Academic Year: 2021-2022

Abstract

A large number of industrial, household, and medical scenarios involve the manipulation of deformable linear objects (DLOs) such as cables, ropes, wires, and hoses. Manipulating deformable objects is a challenging task for robots, considering that they have an infinite number of degrees of freedom. In order to achieve accurate, robust and efficient manipulation, tracking of the DLO shape during the manipulation is crucial.
This thesis proposes a vision algorithm to track in 3D the shape of a DLO, manipulated by a dual-arm robot in an industrial scenario, where different kinds of DLOs are scattered on the worktable. In particular, with reference to the occlusions caused by objects with the same color of manipulated DLO, this thesis proposes also a methodology to deal with this type of occlusion, beyond those caused by objects with different colors.
The creation of a depth filter allows to isolate the manipulated DLO from the background, which can dynamically change. Further to the construction of the point-cloud, the grippers poses acquired from the robot are added to it. The obtained points are fitted in 3D, solving two Lasso regression problems, in x-z and x-y planes, giving as output the tracked shape.
The proposed method does not rely on a physical simulation or physical model of the DLO, and uses only the acquired data by the camera and the grippers poses. The experiments performed prove the robustness of the method to several DLOs, which differ in color, length, and rigidity. In particular, during the experimental validation, they are manipulated by a real dual-arm robot in different configurations.

Keywords: Shape Tracking, Occlusion, Deformable linear objects manipulation, Robotics.

Abstract in lingua italiana

Un gran numero di scenari industriali, casalinghi e medici coinvolgono la manipolazione di oggetti deformabili lineari (DLOs) come cavi, corde, funi e tubi. La manipolazione dei DLO è un compito impegnativo per i robot, considerando che questi hanno un numero infinito di gradi di libertà. Al fine di ottenere una manipolazione accurata, robusta ed efficiente, il monitoraggio della forma dei DLO durante la manipolazione è cruciale.
Questa tesi propone un algoritmo di visione per rilevare in 3D la forma dei DLO, manipolati da un robot a due braccia in un ambiente industriale, dove sono sparsi differenti tipi di DLO sul tavolo da lavoro. In particolare, con riferimento alle occlusioni causate da corpi dello stesso colore del DLO manipolato, la tesi propone anche un metodo per gestire tale tipo di occlusioni oltre a quelle causate da un oggetto con colore diverso.
La creazione di un filtro di profondità permette di isolare il DLO manipolato dallo sfondo che può essere soggetto a cambiamenti. Successivamente alla costruzione della point-cloud, vengono aggiunte le posizioni dei gripper acquisite dal robot. I punti risultanti poi sono interpolati in 3D, risolvendo due problemi di regressione lineare (Lasso) nei piani x-z e x-y, rilevando in tal modo la forma tracciata.
Il metodo proposto non si affida a un modello fisico o a dei simulatori fisici del DLO, ma usa solo i dati acquisiti dalla telecamera e le posizioni dei gripper. I risultati sperimentali ottenuti mostrano la robustezza della metodologia per diversi tipi di DLO, che differiscono per colore, lunghezza e rigidezza. In particolare, durante la validazione, i DLO sono manipolati in diverse configurazioni da un robot a due braccia.

Parole chiave: Rilevamento della forma, Occlusione, Manipolazione di oggetti deformabili lineari, Robotica.

Contents

1 | Introduction

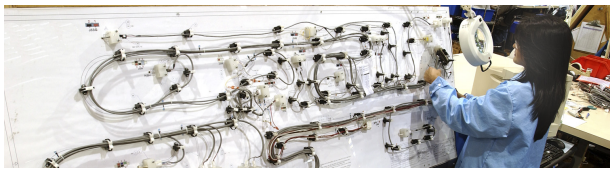
1.1. Robotic cable manipulation

Deformable Linear Objects (DLOs) are elements, like wires, pipes and ropes, with one dimension that is bigger than the other two.
The interest in robotic manipulation of deformable linear objects is growing rapidly, especially in the automotive and aerospace fields, in which numerous applications involving DLOs can be found. These include, for example, wiring operations, wire harness manufacturing or switchgear assembly.
In many industrial applications, manipulation of cables, wires or tubes is needed, but while the industrial manipulation of rigid objects has been automatized for a long time, the handling of deformable linear objects is usually performed manually (Figure 1.1).

(a)

(b)

(c)



(d)

Figure 1.1: Example of operation involving DLOs computed manually. (a) Wire harness factory. (b) Wire harness. (c) Aerospace wire harness. (d) Cable wiring.

The sequence of automated operations must be then interrupted to allow a human operator to manipulate the DLO, in this way the entire process becomes time-consuming.

For this reason, the DLO manipulation can be considered a bottleneck in industry frameworks.
Besides finding a solution to planning the robot manipulation, a clear need to determine a way to track the shape of the DLOs while they are manipulated has been identified. Indeed, shape tracking can provide feedback useful for defining a closed-loop manipulation strategy. Shape tracking is usually performed using vision sensors, which are efficient and intuitive to use. However, there are several challenges in the case of DLO tracking with respect to rigid object tracking. One of them is that, unlike the rigid object whose pose can be described by six degrees of freedom, a DLO has an infinite number of degrees of freedom in space. Moreover, the tracking based on a vision system can fail due to lower light conditions, dust, or the presence of a complex background with many DLOs with the same color or similar to the manipulated one.
In addition, the tracking needs to deal with the occlusion caused by objects in the environment (including the robot arm itself) and the self-occlusion of the object, in order to perform robustly the operation in a constrained environment. Moreover, to be useful for the robotic manipulation of deformable objects, the tracking algorithm needs to be fast enough to be executed in real time.

1.2. Thesis purpose and achievements

This thesis proposes the development of a tracking algorithm, which uses only the vision data by the camera and the poses of the grippers to track online in 3D the shape of different kinds of manipulated DLOs.
The main achievements of this thesis can be summarized in:

- A vision algorithm for tracking in 3D a DLO manipulated by a dual-arm robot, generating online for each frame a geometric estimation of the DLO shape.
- The development of a depth filter that allows to isolate the manipulated DLO from the workspace. Expanding the use of the proposed algorithm in the industry, where it can be a workspace with objects that can have the same shape and color as the tracked DLO, in addition, the workspace can change while the robot manipulates the DLO.
- A tracking algorithm robust to occlusion, that accounts for an issue not covered in the literature i.e. occluding objects with the same color of manipulated DLO. This problem is firstly softened by the use of a depth filter, which ensures robustness to those kinds of occlusions with an occluding object above the manipulated DLO. On the other hand, if the occluding object touches the DLO, the tracking is achieved

by making an online recovery using a sample video without occlusion.

- A potential industrial application, due to different types of DLOs tested, which differ in color, rigidity and length.

1.3. Thesis structure

The remaining of this thesis is organized as described in the following:

- **Chapter 2** introduces what object tracking is and then focuses on shape tracking for DLOs. In particular, summarizes the relevant literature works and it highlights the contributions of this work.
- **Chapter 3** deals with the setting of the problem, describing the used set-up and the kinds of tested DLOs, providing then some preliminary knowledge about the computer vision tools.
- **Chapter 4** describes the proposed tracking algorithm.
- **Chapter 6** reports the experimental validation.
- **Chapter 7** presents the conclusions of the work, as well as some future developments of the proposed tracking algorithm.

2 | State of the Art

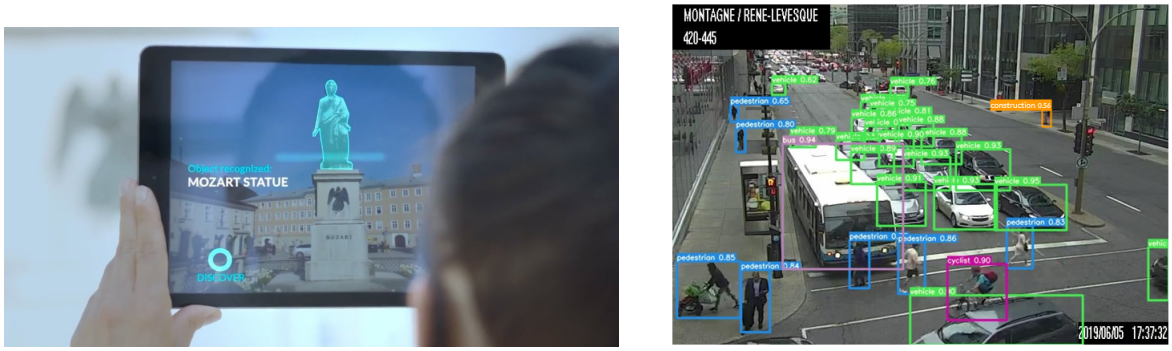
Object tracking is one of the most important tasks in computer vision, with a lot of applications in a different area as robotics, video surveillance, medical area, traffic monitoring etc. In this chapter, the current state of the art related to rigid object tracking and deformable linear object (DLO) tracking is discussed. Firstly a definition of object tracking is given, then we will focus on the deformable objects classifying them. Furthermore, we will discuss about the DLO's tracking, explaining the GMM-EM method used in most procedures for DLO tracking. The latter procedures will be explained by grouping them into ones that use physics simulation and ones without physics simulation. Finally, the contributions of this work with respect to the analyzed literature are highlighted.

2.1. Object tracking

Object tracking is a computer vision application where a program detects objects and then tracks their movements in space or across different camera angles. A first characterization of tracking can be made based on the input of the algorithm (both can be real-time or not):

- image tracking;
- video tracking.

In the first case, a two-dimensional input is given through the camera, the algorithm detects a two-dimensional planar image, which can be then used to superimpose a 3D graphical object. Once the 3D graph is overlaid, the user can move the camera without losing the trace of the 2D planar and graphical surface on it. This type of object tracking is often used in the field of augmented reality (AR, Figure 2.1a).
Differently in the video tracking is given a moving object as input, where its position changes in each frame, so the algorithm needs to be able to deal with this additional variable. An application can be video surveillance (Figure 2.1b).



(a) (b)

Figure 2.1: Example of image tracking (a) and of video tracking (b).

In this thesis, we will focus our attention on video tracking. Hence Going deeply into object tracking, it requires two steps:

- object detection, where the object is identified and the algorithm clusters the pixel of the object;
- object classification, where the objects are classified as fruit, animal, human etc. Based on its motion, texture, shape and/or color.

Balaji et al [1] proposed a formalization of the main technique in moving object detection, classification and tracking (Figure 2.2).

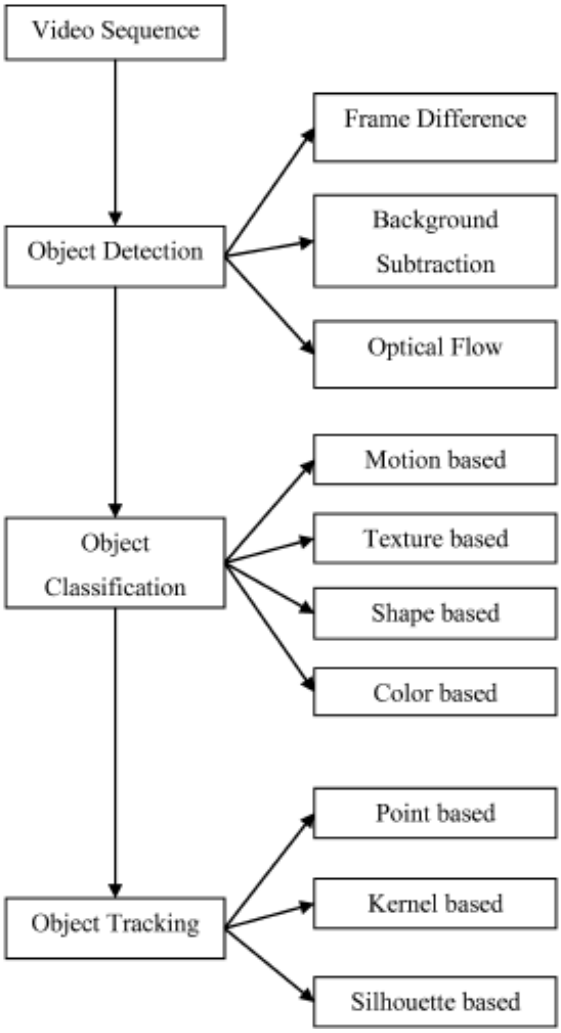


Figure 2.2: Phase and technique of object detection,classification and tracking [1].

We will focus only on a few of these methods that are prominent in the DLO tracking literature (Section 2.3).
Analyzing the object detection methods, in order to identify the moving object the frame difference method calculates the difference between two consecutive frames, even if it is very easy to implement, it requires a static background. In the Background subtraction method, it is necessary to first construct a background model in order to obtain a reference, that is compared with each video sequence to determine the possible variations. In the case of a recursive algorithm, based on each given input frame, it updates the background model. A method used in this technique is the Gaussian mixture, further detailed in Section 2.3.1.
Moreover, in the object classification methods, we can emphasize our attention in classification based on color (the technique that is used in most of the methods in Sections 2.3.2

and 2.3.3), due to the fact that the color is easy to acquire with respect to the other feature, but it still offers high accuracy with respect to the other methods.
The different object tracking methods are presented in Table 2.1:

Object tracking		
Point based	Kernel based	Silhouette based
Kalman filter	Simple Template Matching	Contour matching
Particle filter	Mean shift	Shape matching
Multiple hypothesis tracking	Support vector machine	
	Layer based tracking	

Table 2.1: Object tracking methods.

Point based groups together all the techniques where the moving objects during the tracking are represented by their feature points. One of the problems in tracking is the wrong detection of the object due to occlusion, a problem that can be avoided with this type of technique. In particular, in the case of the Kalman filter, the basic steps performed are the prediction of the state variable based on a set of observations and the updates of this variable for the next time instant, where a weight between the noisy measurements and the predicted variable is made by the Kalman filter, using a modeling state equation.
In the particle filter, the basic steps are the prediction and the update of the state as in the Kalman filter. Moreover, before the update of the variable, all models of that variable will be generated using the particle filter: this ensures tracking of multiple objects differently from the Kalman filter.
The point tracking method is preferred to the Kernel tracking and the Silhouette tracking due to the fact that these two methods do not consider or partially consider the occlusion.

2.2. Deformable Objects

Sanchez et al [2] proposed a classification of deformable objects based on their physical properties and their shapes. Considering the physical properties of the object, they may be divided in:

- object with no compression strength;
- object with large strain or present a large displacement.

The first are those objects which do not present any resistance when two opposite endpoints are pushed toward each other, an example are the ropes and clothes. The second

one are those objects with a low Young modulus, as a sponge or paper.
Considering the object's shapes, they can be divided in:

- uniparametric objects, characterized by one dimension significantly larger than the other two, an example is a cable where the length is much larger than its width or height ;
- biparametric objects, characterized by one dimension smaller that the other two, as a paper where the thickness is negligible with respect to the other dimensions;
- triparametric objects, that are the solid objects.

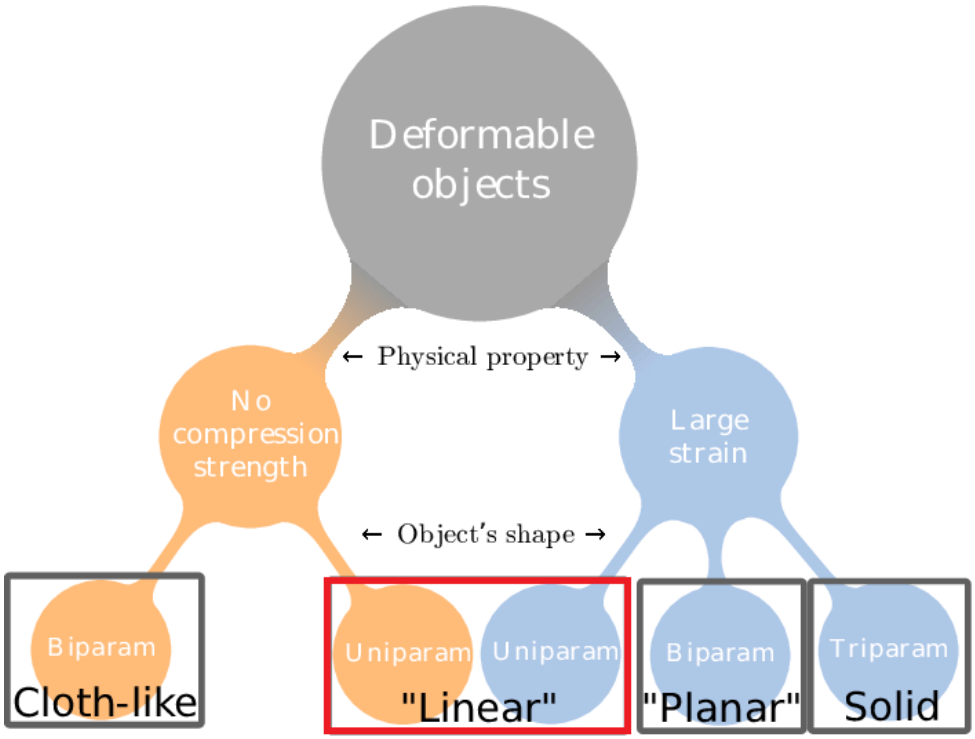


Figure 2.3: Classification of a deformable objects [2].

In this thesis, we will focus on the deformable linear object (DLOs), also known as deformable one-dimensional objects (DOOs) (Figure 2.3), and on semi-deformable linear objects (SDLO) (Section 3.1.1).
SDLOs are a subclass of DLO which are characterized by the presence of one or more rigid parts, usually connectors at the cable's ends, an example of it is showed in Figure 2.4b.

(a) (b)

Figure 2.4: Example of a DLO (a) and of a SDLO (b).

2.3. DLO tracking

In the last ten years, the interest in the tracking of deformable objects has grown, as we can think a large number of industrial, household, and medical scenarios involve the manipulation performed by a robot of deformable linear objects (DLOs) such as cables, ropes, wires, and strings. Hence it has never been more important to have information about the shape of the manipulated DLO, in particular in presence of occlusion allowing to perform operations in a constrained environment.
The literature dealing with DLO tracking can be divided in tracking methods that use the physics simulation (Section 2.3.2) and in ones without it (Section 2.3.3).
But there are other innovative methods, as the one presented in [3] that proposes a tracking of a thread in a surgical scenario. The thread has a multi-color pattern. The object is first modeled in 3D with a non-uniform B-spline, and the tracking problem is formulated as an energy minimization over the spline control points (Figure 2.5).

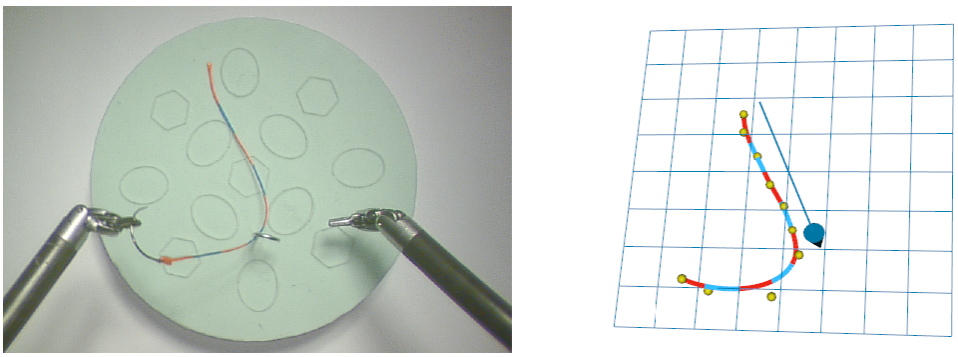


Figure 2.5: Manipulation of a thread with da Vinci instruments (left) and the 3D tracking model (right) [3].

Although this method is able to track efficiently a very thin object as a thread, it has a strong constraint on the presence of a multi-color DLO, this limit is too strict in an industrial application where the DLOs have a uniform color.
A novel approach called "slicing method", is proposed by Rastegarpanah et al [4]. The point-cloud captured is firstly filtered in order to isolate the DLO, then it is recursively sliced into several smaller node point-clouds, and finally, these resultant nodes are used in a trajectory step, where the output is a trajectory composed of waypoints that describe the current state of the DLO. Even though the algorithm needs only two parameters for the configuration reducing in this way the set-up time, it presents good results only in simulation with respect to the real-world experiment (Figure 2.6). Where a flexible pipe with reflective markers, which position is tracked by two optiTrack cameras, in order to produce a ground truth. Then the pipe is swung from a certain high and a depth camera is used to reconstruct the point-cloud. One of the limits of this algorithm is that it does not ensure robustness under occlusion.

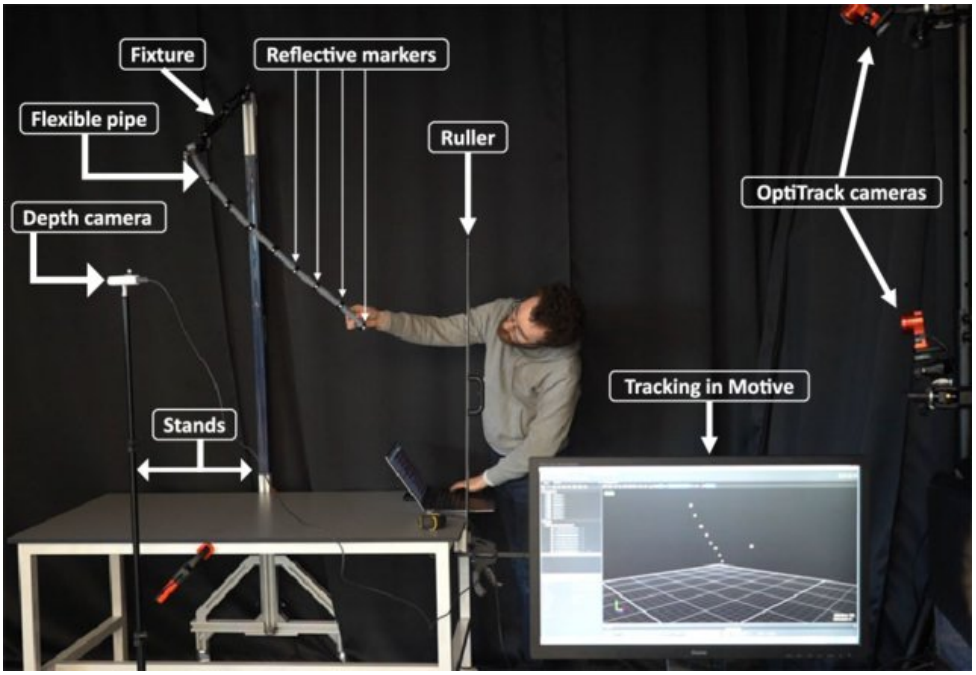


Figure 2.6: Experimental set up of Rastegarpanah et al [4].

The key common point in most DLO tracking techniques is the use of GMM-EM (Section 2.3.1), so before going deeply into them, we will first need to understand the GMM-EM and its use in DLO tracking.

2.3.1. Gaussian Mixture Model Expectation-Maximization (GMM-EM)

A Gaussian Mixture is a function that is composed of different Gaussians, where the number of Gaussians corresponds to the number of clusters. Each Gaussian is featured by the following parameters:

- the mean μ that defines its centre;
- the covariance Σ that define its width;
- the mixing probability ρ that defines how big or small the Gaussian function will be.

Figure 2.7 shows a case where we have three Gaussian functions.

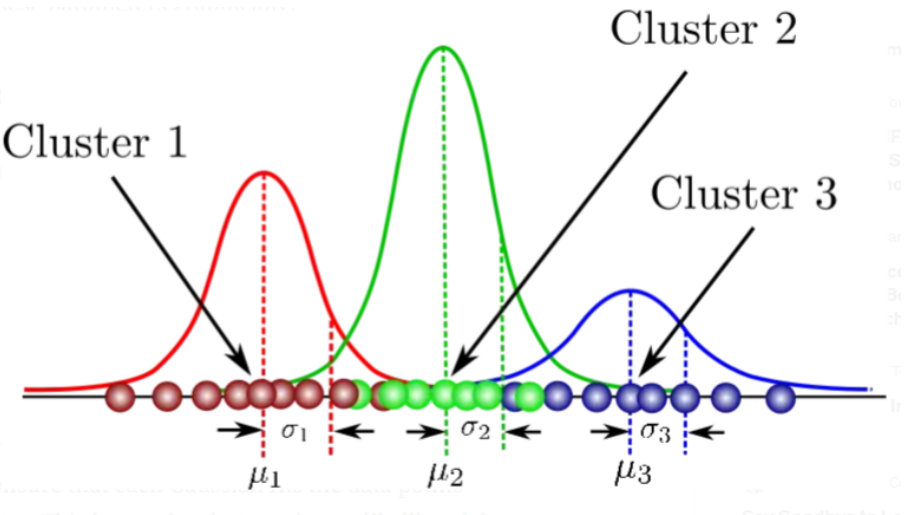


Figure 2.7: Gaussian mixture, composed by three Gaussian functions with mean μ_k and standard deviation σ_k ($k \in \{1, 2, 3\}$).

The idea is then to find the optimal values for the three parameters such that each Gaussian fits the data points belonging to each cluster.
Given the object's point-cloud at the time step t, $Y^t = \{y_1^t, y_2^t, y_3^t..., y_M^t\} \in R^{MxD}$ where D is the dimension of the data (in the considered case $D = 3$). We can assume that the deformable object can be discretized by N nodes, hence at the same time step t, we obtain $X^t = \{x_1^t, x_2^t, x_3^t..., x_N^t\} \in R^{NxD}$, usually with $N << M$. Due to the fact that between two time steps t and $t+1$, the change of deformable object is small, the tracking problem can be seen as a point-set registration problem, that consists in an estimate X^{t+1} aligning X^t towards Y^t. Hence following [5], this problem can be formulated using a GMM, where

X^t are the centroids of the Gaussian, and Y^t are the data points generated from the Gaussian mixture. y_m^t represents the m-th point in the point-cloud and correspondingly x_n^t is the position of the n-th node at time t. The probability distribution of point y_m^t is:

$$p(y_m^t) = \sum_{n=1}^{N} \frac{1}{N} \mathcal{N}(y_m^t; x_n^n, \sigma^2 I) = \sum_{n=1}^{N} \frac{1}{N} \frac{1}{(2\pi\sigma^2)^{D/2}} exp\left(-\frac{||y_m^t - x_n^t||^2}{2\sigma^2}\right) \tag{2.1}$$

Equation (2.1) assumes that the mixing probability ρ is fixed for each Gaussian equal to $\frac{1}{N}$ and they share the same isotropic covariance $\sigma^2 I$. In order to consider the noise and the outliers, we need to add an additional uniform distribution to the mixture model:

$$p(y_m^t) = \sum_{n=1}^{N+1} p(n) p(y_m^t | n) \tag{2.2}$$

Defining with ω the weight of uniform distribution :

$$p(n) = \begin{cases} (1-\omega)\frac{1}{N}, n = 1, ..., N \\ \omega, n = N+1 \end{cases} \tag{2.3a}$$

$$p(y_m^t | n) = \begin{cases} \mathcal{N}(y_m^t; x_n^n, \sigma^2 I), n = 1, ..., N \\ \frac{1}{M}, n = N+1 \end{cases} \tag{2.3b}$$

So in this way fixing the mixing probability ρ, the goal becomes to find the centroids x_n^t and the variance σ^2 that maximize the log-likelihood of equation Equation (2.2):

$$\mathcal{L}(x_n^t, \sigma^2 I | Y^t) = log \prod_{m=1}^{M} p(y_m^t) = \sum_{m=1}^{M} log\left(\sum_{n=1}^{N+1} p(n) p(y_m^t | n)\right) \tag{2.4}$$

Analyzing the Equation (2.4) due to presence of a summation inside the $\log(\cdot)$ the optimization problem:

$$(x_n^{t\,*}, \sigma^{2*}) = \arg\max_{x_n^t, \sigma^2} \mathcal{L}(x_n^t, \sigma^2 I | Y^t) \tag{2.5}$$

is not convex. For this reason, we can define a complete log-likelihood function Q:

$$Q(x_n^t . \sigma^2) = \sum_{m=1}^{M} \sum_{n=1}^{N+1} p(n | y_m^t)\, log(p(n) p(y_m^t | n)) \tag{2.6}$$

With the definition of the Equation (2.6), we can solve the optimization problem using a recursive algorithm: the Expectation Maximization (EM) algorithm.

E-step

In the E-step the posterior probabilities $p(n|y_m^t)$ is calculated using the current GMM parameters obtained from the last M-step. So using the Bayes rule we can obtain:

$$p(n|y_m^t) = \frac{exp\left(-\frac{||y_m^t - x_n^t||^2}{2\sigma^2}\right)}{\sum_{n=1}^{N} exp\left(-\frac{||y_m^t - x_n^t||^2}{2\sigma^2}\right) + \frac{(2\pi\sigma^2)^{D/2}\omega N}{(1-\omega)M}} \tag{2.7}$$

With the obtained value of $p(n|y_m^t)$, we can pass to the M-step.

M-step

During the M-step is found an estimation of x_n^t and σ^2 minimizing:

$$Q = -\sum_{m=1}^{M}\sum_{n=1}^{N+1} p(n|y_m^t)\frac{||y_m^t - x_n^t||^2}{2\sigma^2} - \frac{N_p D}{2} log(\sigma^2) \tag{2.8}$$

where $N_p = \sum_{m=1}^{M}\sum_{n=1}^{N} p(n|y_m^t)$. Then the E and M step are iterated until the function Q converges (Figure 2.8).

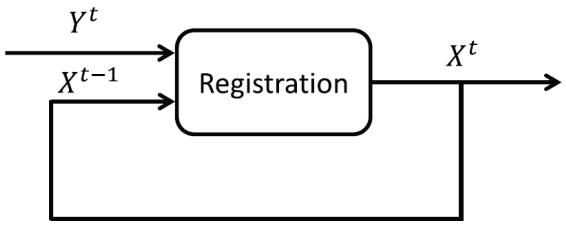
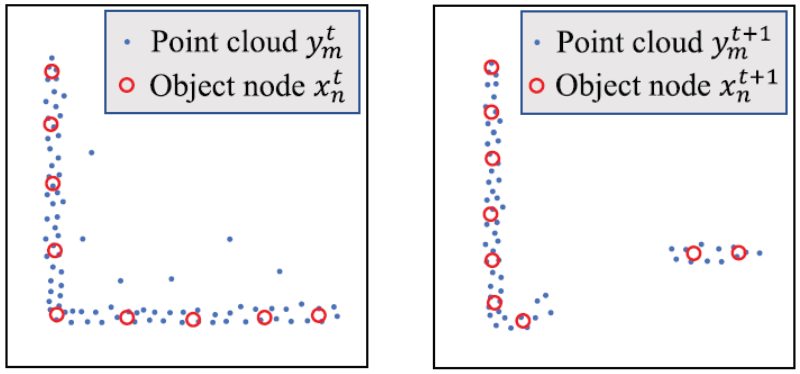


Figure 2.8: Point set registration scheme [5]. Due to the fact that the object shape between two time steps does not change too much, the estimation (x_n^{t-1}, σ^2) of the previous step is used to initialize the EM at time t.

The GMM-EM method is able to track the deformable object despite noise and outliers (Figure 2.9a), but fails in presence of occlusion (Figure 2.9b).



(a) (b)

Figure 2.9: Example of result obtained using the GMM-EM [5]. The blue dots composed the point-cloud, and the red circles are the estimated node positions. (a) At time step t, GMM is able to estimate the node state regardless of noise and outliers. (b) At time step $t+1$, occlusion happens. GMM is unable to estimate the status in the occlusion zone, and the estimate in other areas is also severely disturbed.

The Figure 2.9 underlines the necessity of an extension of this method in order to be robust to the occlusions.

2.3.2. DLO tracking using physics/simulation

Schulman et al [6] proposed a modified GMM-EM method (Section 2.3.1) where the nodes X^t are the point of a physical model, obtained from a simulation. In this method is added to the Gaussian mixture model the visibility variables $v_n, n = 1, 2, .., N$. This parameter defines if the node x_n^t is visible or not from the camera. Then is added a prior on x_n^t :

$$p(x_n^t) = e^{-\frac{1}{\eta}V_0(x_n^t)} \tag{2.9}$$

where $V_0(x_n^t)$ is the potential energy of the tracked object, which takes into account the gravitational and bending energy computed thanks to the physical model. Adding this prior means assuming a quasi-static scenario where the objects move slowly and are continually at low-energy configurations. Before passing to the E-step, at each time step the visibility variables v_n are firstly calculated, considering whether or not the line segment from the camera to the node is blocked by some modeled object. Then the E-step is applied, calculating the posterior probability influenced by v_n. The M step, is performed by applying a virtual force on the nodes and time-stepping a generic physics simulator. In this way, only the x_n^t which are visible are updated and the ones that are

not visible kept their previous position. Figure 2.10 shows some examples of tracking with this method. In spite of this method's account for physical constraints imposed by collision and by the material of the deformable object and it is robust to the occlusion, the use of a physics simulator is computationally heavy. In addition, it presents some failure when there is an occlusion of some critical part of the rope.

Figure 2.10: Example of DLO tracking using the GMM-EM with a physics simulator [6].

Nevertheless, the estimation of obstacle position using the visibility variable v_n is not always possible and it is computationally expensive. Hence, Te Tang et al [5] proposed a method that is robust to occlusion without the use of a visibility variable: The Structured preserved registration (SPR).

SPR

The main idea of SPR is to introduce constraints on the Gaussian centroids such that they are registered in the area with higher likelihood but respecting the physical constraint of the tracked DLO. Hence both global and local regularization are introduced on GMM registration. The goal of the local regularization is to regularize the relative motions between neighbor points, this is made considering that any point at time step $t-1$ can be characterized by the weighted sum:

$$x_n^{t-1} = \sum_{i \in I_n} S_{ni} \cdot x_i^{t-1} \tag{2.10}$$

where I_n is the set for K nearest point to x_n^{t-1} that can be found efficiently by the K-nearest neighbor (KNN) algorithm and S_{ni} is the weight matrix that captures the local

topology between x_n^{t-1} and the surrounding node x_i^{t-1}. When at time t the DLO changes its shape, the tracking points can change but the local structure is maintained so:

$$x_n^t \approx \sum_{i \in I_n} S_{ni} \cdot x_i^t \tag{2.11}$$

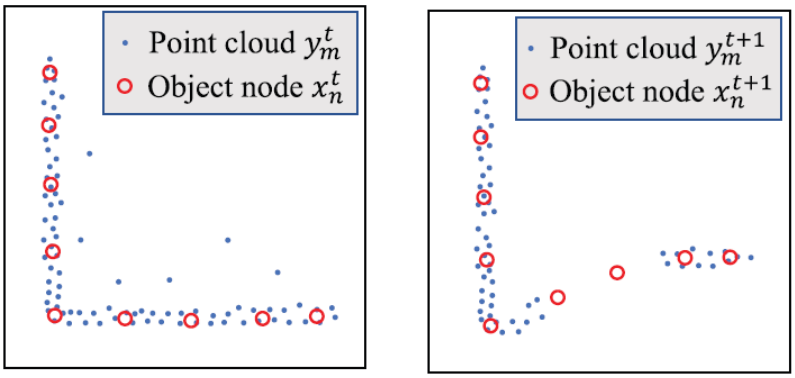
Where the optimal combination of weights S_{ni} can be performed by solving a constrained least square problem. There could be many sub-optimal weights due to the singularity of the matrix when solving least squares, so it is integrated all L sub-optimal weights to characterize the local structure. Further details can be found in [5]. Then, in order to maintain the local topology, E_{local} should be as small as possible. This means that there is not a local change between $t-1$ and t:

$$E_{local} = \sum_{n=1}^{N} \sum_{l=1}^{L} || \sum_{i=1}^{N} S_{ni}^{(l)} x_i^t || \tag{2.12}$$

Next global regularization is introduced, made following the CPD method [7], which regularizes the displacements of the entire body at neighbor time steps. Instead of modeling each point x_n^t as an independent Gaussian centroid, the frame-to-frame change is included in a spatial $x_n^t = \mathcal{T}(x_n^{t-1}, W^t)$ that maps every point in the space around our object of interest at time $t-1$ to another point at time t using parameter matrix $W^t \in R^{NxD}$. At last the Equation (2.6) is modified defining the likelihood function $\tilde{Q}$:

$$\tilde{Q} = Q(x_n^t, \sigma) - \frac{\tau}{2} E_{local} - \frac{\lambda}{2} E_{global} \tag{2.13}$$

where $\tau \in R^+$ and $\lambda \in R^+$ are trade-off weights that balance the regularization of local and global structure. In addition, E_{global} depends on the weight matrix W (for more detail see the appendix of [5]). Now as in 2.3.1 the EM algorithm can be performed but estimating the parameters (W,σ^2). The differences of use SPR instead of GMM can be noticed from Figure 2.11, where SPR under occlusion still works.



(a) (b)

Figure 2.11: Example of result obtained using SPR in presence of occlusion[5]. The blue dots composed the point-cloud, and the red circles are the estimated node positions. (a) At time step t, GMM is able to estimate the node state regardless of noise and outliers. (b) At time step $t+1$, occlusion happens. Differently from Figure 2.9b SPR is robust to the occlusion because the topological structure is preserved from the last time step.

The DLO as an object need to satisfy a series of physical laws, such as kinematics, dynamics and penetration constraints. In order to have a tracked object that follows these laws, the object states are first estimated by SPR and then the estimated states are sent to a dynamic simulation for further physical refinement. Indicating with $\tilde{X}^t = \{\tilde{x}_1^t, ..., \tilde{x}_n^t\} \in R^{NxD}$ the states of the virtual object, if there is a deviation between $\tilde{x}_n^t$ (the position of node obtained from simulation) and x_n^t (the position of the node obtained from SPR), a tracking force will be generated using an impedance controller and applied on virtual nodes. The final framework composed by SPR and dynamic simulation is summarized in Figure 2.12 and Figure 2.13 shows the tracking result obtained with SPR .

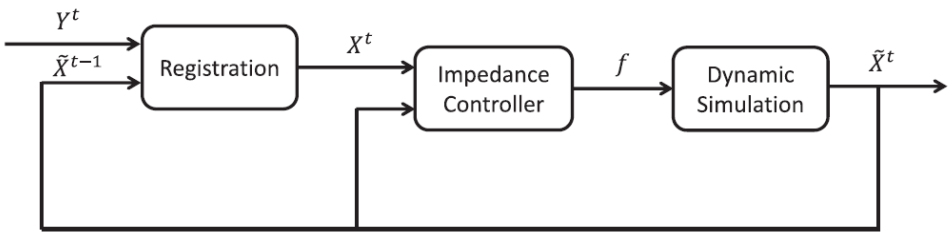


Figure 2.12: Scheme of the tracking method proposed by [5]

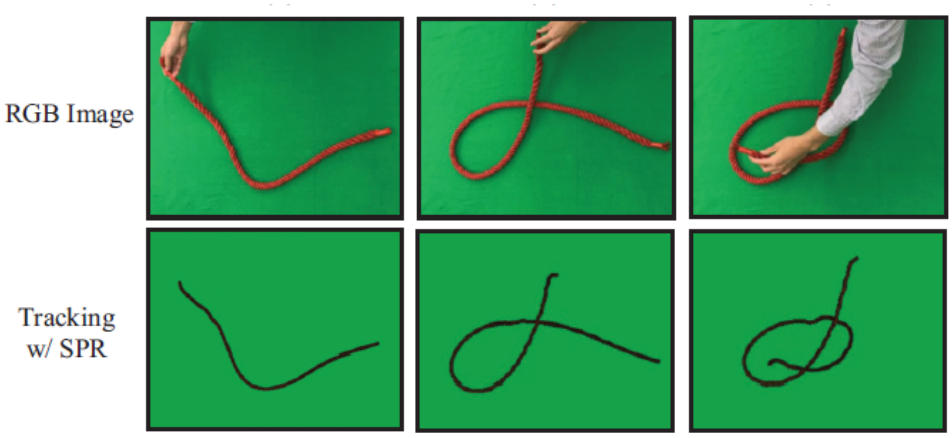


Figure 2.13: Tracking using SPR [5].

This method is robust when the occluded part is small or the tip of the cable is not occluded, in the opposite case SPR would fail to estimate the correct cable state. For this reason, Jin et al [8] proposed a point-cloud recovery inspired by the background subtraction method in computer vision (Section 2.1). In Figure 2.14 it can be noticed that at time step t, an occlusion happens (Figure 2.14c), so the idea is to complete the point-cloud in frame t (Figure 2.14c) using the foreground mask that underlines the part where the occlusion (Figure 2.14d) and the point-cloud at time $t-1$ that complete the occluded part (Figure 2.14e).

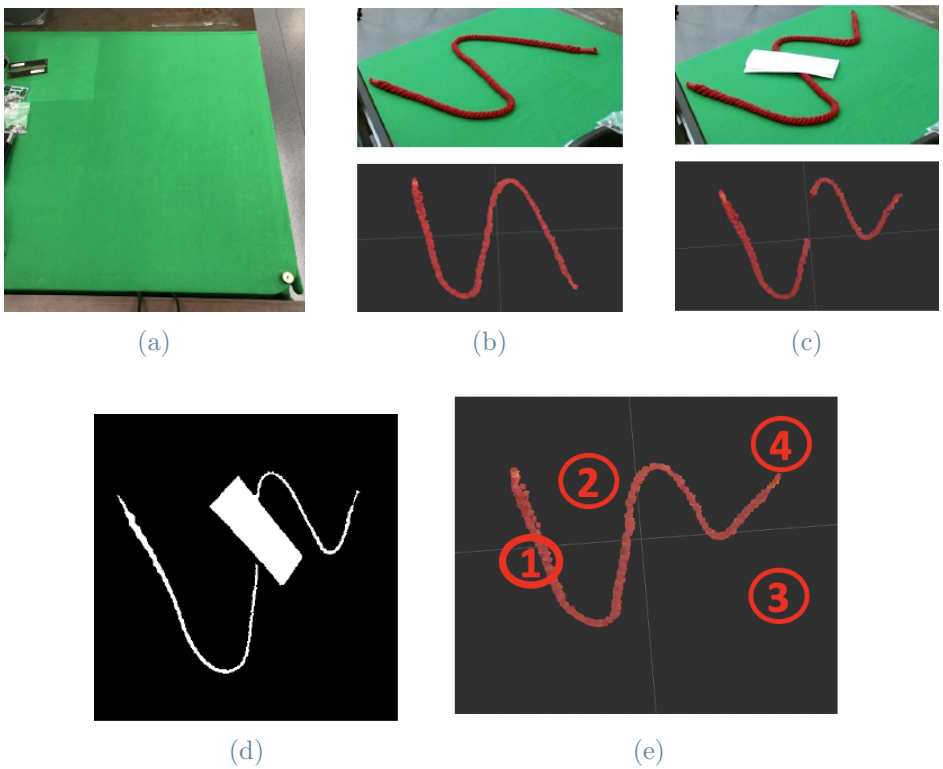


Figure 2.14: (a)Background. (b) RGB and point-cloud at time $t-1$. (c) RGB and point-cloud at time t. (d) foreground mask obtained subtracting the background from frame t. (e) recovered point-cloud [8].

The point-cloud recovery is then used to obtain the point cloud Y^t for the node registration (Figure 2.15).

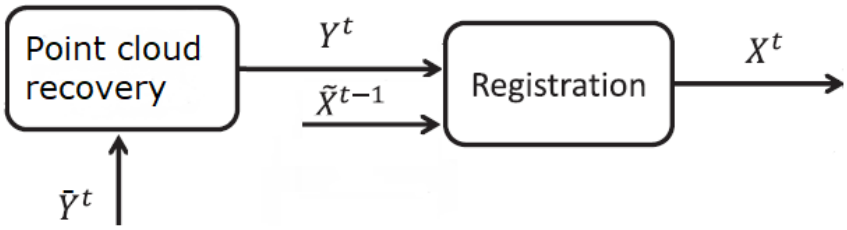


Figure 2.15: Block scheme of SPR with point-cloud recovery. $\bar{Y}^t$ is the point-cloud before the recovery, Y^t is the point-cloud after the recovery, $\tilde{X}^{t-1}$ is the state of virtual object obtained from the dynamic simulation and X^t are the tracked points.

Although this method increases the robustness at occlusion, the use of background subtraction limits this method to an environment with a static background, so suppose to have first a background free from DLOs like Figure 2.14a and then, while the DLO is

manipulated and tracked, the background changes adding, for example, another DLO, this recovery will fail. SPR [5] suffers from incorrect registration of branched DLO, in case a single branch is occluded. That's why Wnuk et al [9] proposed CAMP, a modified SPR where is introduced a branch-wise probability in the GMM. The result of CAMP implementation is shown in Figure 2.16.

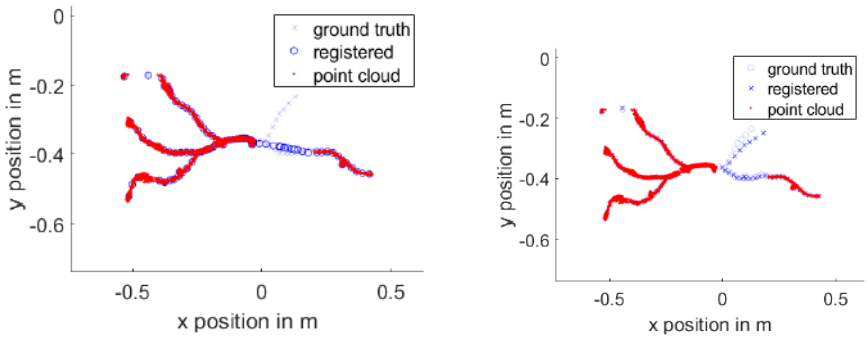


Figure 2.16: (a) SPR has difficulty to track the occluded branch, while (b) CAMP successfully tracks the occluded branch [9].

SPR [5] and all its extensions and modifications ([9] and [8]) are based on a combination of probability-based registration and dynamics-based simulation that provides an estimate with strong robustness to sensing noise, outliers and occlusion as well as satisfying physical constraints. The cons as in [6] is the use of the physics simulation that is computationally heavy.

2.3.3. DLO tracking without physics simulation

There are other methods that do not consider the use of a physics simulation. Jin et al [10] proposed to use first a Neural network (U-net [11]) to isolate the DLO: the interesting fact is that the U-net is trained using a single video of the DLO manipulation, and for each frame a DLO segmentation mask is generated automatically with the color filter. Then the registration step is applied using CPD [7], without adding a local regularization as SPR (Section 2.3.2) but simply connecting the registered nodes and re-sampling along the connected path to obtain equally distributed nodes. This limits the efficiency of this method only without heavy occlusion. Instead, Waltersson et al [12] proposed to first isolate the DLO from an RGB image using a color mask, then use the mask information and the depth image to obtain the point-cloud. The point-cloud is then used to perform the point registration for tracking, differently from SPR presented in the previous section

(Section 2.3.2) a simulator is not used. To deal with this lack that gives stable real-time tracking, the local regularization is kept constant from the initial estimation. Then for each call of SPR algorithm, the previous estimation is used as an initial guess (Figure 2.17).

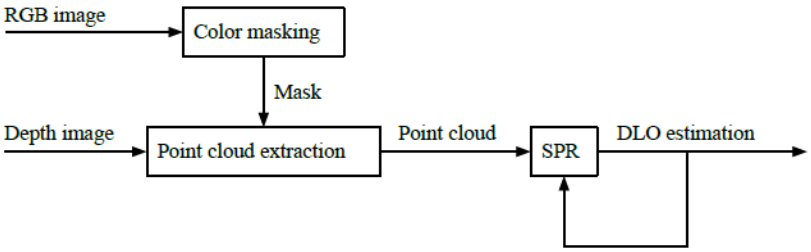


Figure 2.17: Scheme of tracking method proposed in [12].

On the other hand Jin et al [13] extended the SPR with RWLS (robust weighted least squares), which is used to calculate the local deformation model of the DLO under uncertainties. Physical simulation or physical model can be difficult to obtain in unstructured environments, for this Berenson et al [14] proposed a method that does not rely on these two things. The method is called CDCPD, as SPR is based on the GMM-EM and on local and global regularization. But differently from SPR is not considered that all Gaussian distributions have the same mixing factor $\rho = \frac{1}{N}$, because it means that for each Gaussian distribution is equally possible to generate a point of the point-cloud. It is easy to understand that this assumption fails especially in case of occlusions, hence as [6] it is considered a visibility variable $v(n)$ that penalizes the nodes below the visible point-cloud and the points farther away from the object. So the Equation (2.3a) becomes:

$$p(n) = \begin{cases} (1-\omega)v(n), n = 1, ..., N \\ \omega, n = N+1 \end{cases} \tag{2.14}$$

And the posterior probability in E-step (Equation (2.7)) becomes:

$$p(n|y_m^t) = \frac{v(n)exp\left(-\frac{||y_m^t - x_n^t||^2}{2\sigma^2}\right)}{\sum_{n=1}^{N} v(n)exp\left(-\frac{||y_m^t - x_n^t||^2}{2\sigma^2}\right) + \frac{(2\pi\sigma^2)^{D/2}\omega N}{(1-\omega)M}} \tag{2.15}$$

Then the E_{local} presented in SPR mitigates the topology consistency problem, but it does not take into account that the DLOs for the features presented in Section 2.2 are less deformable when being stretched than being compressed. In order to consider this limitation, a constrained optimization method is introduced to post-process the output of the GMM-EM, so that it allows the compression while keeping the distance between

points below a threshold. A very powerful innovation is the tracking failure recovery step, where firstly an energy function is constructed (J_{free}) that indicates the percentage of points that are in free space (the space between the ray emitted by the camera and the visible point) and how far they are from the non-free space, and this is useful to detect if a tracking failure happened and only apply the recovery when needed. Then a shape descriptor is used to find the most relevant previous state of the object after several occlusions.
After one year Wang et al [15] proposed an extension of the previous method: CDCPD2. It introduced a tracking method similar to Kalman filter method (Section 2.1). In particular, another term called $E_{Pred} = \sum_{n=1}^{N} ||x_n^{GMM} - x_n^{Pred}||^2$ is added in Equation (2.13), where x_n^{GMM} is n-th tracked point obtained using the GMM method, while x_n^{Pred} is a prediction of the n-th tracked point using a model of deformation of the object. The idea is to make a prediction with the motion model, and then use it to update the tracking result along with what is observed at the next frame (for further details about the implementation and type of motion model see [15]). This addition is able to avoid the shrinking problem. It happens when the tracked object has shrunk due to loose of the point due to occlusion, e.g. Figure 2.18a where a DLO is trailed and the end of the rope is occluded by a box. In addition, it is more robust with to respect the self-intersection and interaction with an obstacle, as we can see from Figure 2.18 .

(a)

(b)

Figure 2.18: (a) CDCPD demonstrates the difficulty to track the DLO when its end is occluded (left image) and when it interacts with the cylinder (right image). (b) While CDCPD2 demonstrates to have no problem in tracking the DLO in both cases. [9]

The CDCPD2 is the most efficient of the explained method, due to the fact it is able to track deformable objects in more realistic scenarios, where occlusion and clutter are available. In addition thanks to tracking failure recovery it is able to track the DLO

also in case of several occlusions, and all this without any physics simulation. Although the optimal results in the state of the art and the available code on GitHub [1], this implementation was not complete.
Since the aim of this thesis was to implement a robust method for tracking DLO, giving a contribution to the current state of the art, it was decided to implement a different tracking method, easier to implement.
Kangchen et al [16] proposed a data-driven method to robustly estimate the state of DLO from a single frame, also in case of occlusion, without the use of physics simulation and robot configuration. Figure 2.19 summarizes this method, a point-cloud, which can be fragmented due to occlusion, is given as input to a Neural network, PointNet++ encoder, that extract the point-cloud features. These features are given to two branches: End-to-End Regression, which focuses on global geometry information, and Point-to-Point Voting, which focuses on local geometry information. Then their estimations are fused to combine the advantages of the two branches, in this step is used CPD ([7]), without using the E-step (for further detail see [16]). Although this method ensures accurate tracking of the DLO in case of occlusion and not, the used model does not use temporal information, so the predicted shape might not be continuous across the adjacent frames.

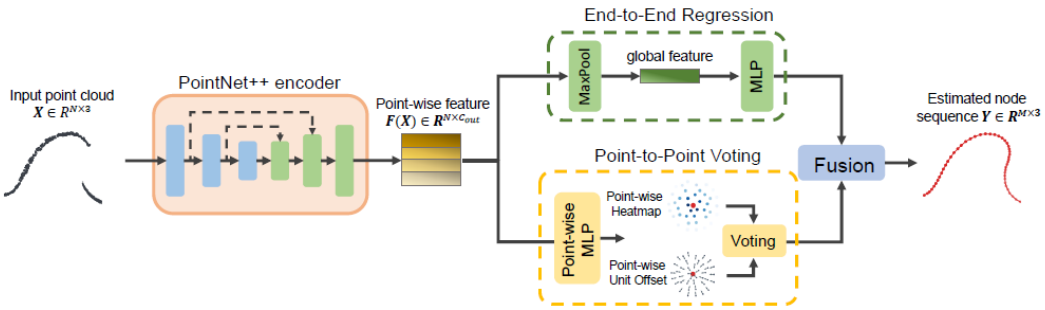


Figure 2.19: Scheme of the proposed method for occlusion-robustly estimating the 3-D states of DLOs[12].

[1] [15] code on GitHub: `https://github.com/UM-ARM-Lab/cdcpd`

2.4. Thesis contribution

The tracking solution proposed above (Sections 2.3.2 and 2.3.3) limits its use in an industrial scenario where the DLO is manipulated from the working table at a certain height, and other objects or DLOs also with the same color could be in the workspace. Moreover, [8] considers an occlusion recovery only in case of a static background, but the background can change during the manipulation. For example when the manipulated DLO is long enough such that the extremes of it touch the working table and will move during the operation. So in this scenario, the use of background subtraction would fail.
For this reason, this thesis tries to deal with these limitations by proposing to track different classes of DLOs and SDLOs (Section 3.1.1) in 3D, without using probabilistic methods or physics simulation, but using only vision algorithms. This algorithm is able to:

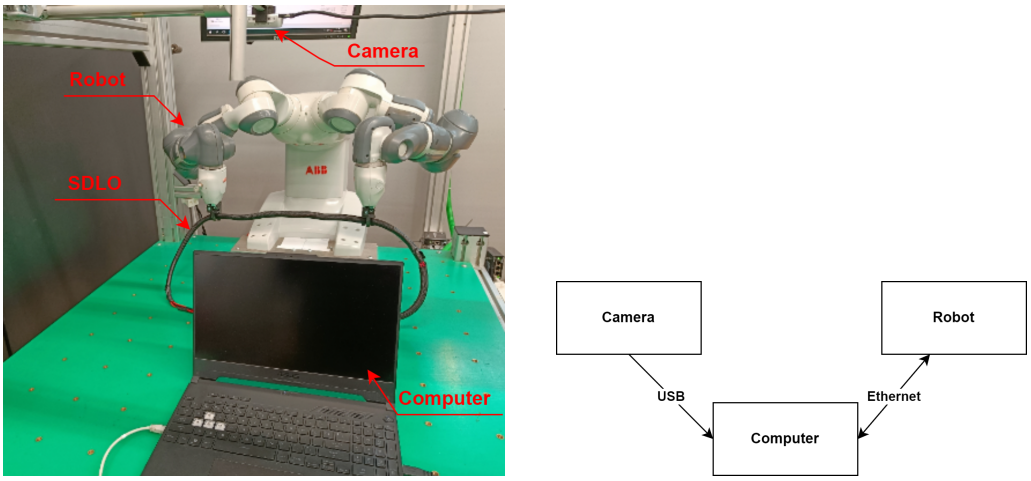
- deal with the limitation of the vision system as the impossibility to detect with good accuracy the position and the shape of an object in presence of dust, occlusions, particular light conditions or when similar shaped and colored objects are placed close one to each other in the working area;
- track performing a geometric estimation of the object, that will be reported in the reference system of robot in order to give complete information on the tracked object shape. In addition, it will be used in future work to implement a visual servoing algorithm;
- track robustly the DLO also in presence of dynamically changing background or other objects and/or DLOs in the working environment (that can have also the same shape and color of tracked DLO). This is possible using the depth information from the camera, in order to isolate only the manipulated DLO of interest, which does not have any limit on the length;
- track in case of occlusion, which can be a dynamic or static one, made by the dual-arm robot that manipulates the DLO or by the object in the scene. Differently from the presented literature, the occluding object can be also of the same color. This is possible thanks to the implementation of the position of the robot grippers in the geometric estimation of the object and thanks to the construction of a depth filter. Thanks to it the limit presented in SPR ([5]) on occlusion of the tip of the rope is mitigated.

3 | Setting of the problem

This Chapter deals with the setting of the considered problem, detailing the set-up exploited and the kinds of DLOs used. Some useful notions are also recalled.
In particular, we will first detail the exploited set-up and then explain how a digitized image is represented and what color models are used for it. In addition, an introduction on camera calibration will be given, and we will derive how to obtain a representation of a point in the robot base frame, from its representation in the camera frame.

3.1. Set-up

The hardware set-up is composed by a robot, a camera in eye-to-hand configuration and a computer (Figure 3.1a). The computer controls and exchanges information with the robot using an Ethernet connection (further details in Chapter 5), moreover the computer receives visual information by the camera and are connected with USB (Figure 3.1b).



(a) (b)

Figure 3.1: Hardware set up (a) and how it is connected (b).

The robot is an ABB IRB14000, also called YuMi, it is a dual-arm robot with 7 degrees of freedom for each arm. Thanks to this feature, it is capable to manipulate the DLO with

high precision and repeatability. Each flange is equipped with an electric parallel gripper on which custom 3D printed fingertips are mounted (Figure 3.2).

Figure 3.2: Gripper.

The camera is an Intel realsense D435i (see Table 3.1 for the tech specs), it is an RGB-D so has both an RGB sensor and depth sensors (further details on RGB and depth in Section 3.2.1). It is a stereo depth camera, which means that it has two sensors, spaced a small distance apart, and takes the two images from these two sensors and compares them. Since the distance between the sensors is known, these comparisons give depth information. To improve the accuracy of depth data it has also an infrared projector (Figure 3.3).
The eye-to-hand configuration is obtained by combining three rods of different lengths as shown in Figure 3.4. This configuration grants a fixed field of view as the robot moves, moreover, also the geometric relationship between the camera and the workspace is fixed, allowing an offline calibration. The cons of this configuration is that as the robot moves through the workspace, it can occlude the camera's field of view, for this reason in Chapter 4 we will propose a methodology dealing with this problem.

Depth Specs	Minimum depth distance (Min-Z) at max resolution:	∼28 cm
	Accuracy:	<2% at 2 m
	Output resolution:	Up to 1280 × 720
	Frame rate:	Up to 90 fps
RGB Specs	Frame resolution:	1920 × 1080
	Frame rate:	30 fps
	Sensor resolution:	2 MP

Table 3.1: Camera specifications.

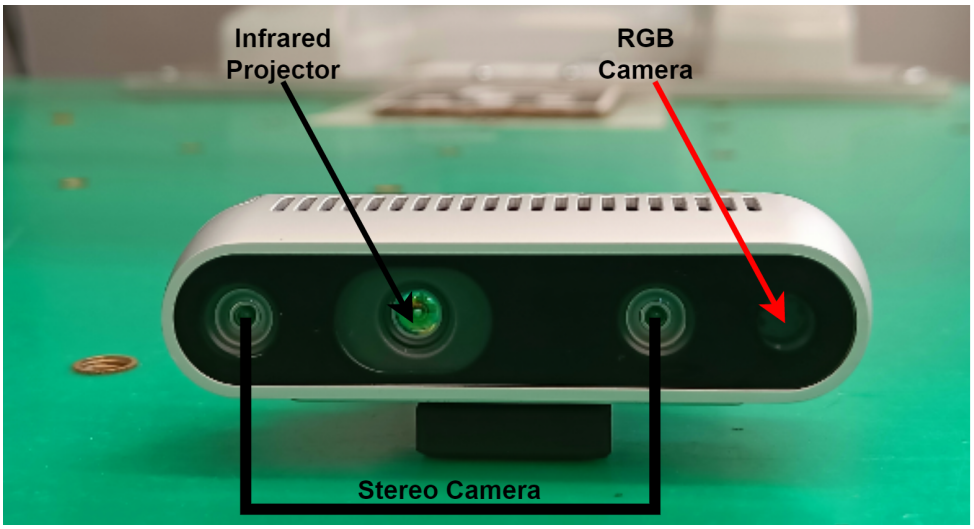


Figure 3.3: Camera Intel Realsense D435i.

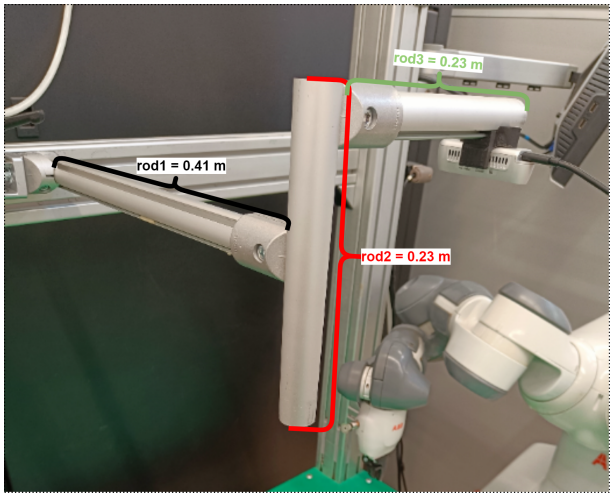


Figure 3.4: Eye-to-hand configuration with with the measures of the rods. The first rod is in black, the second one is in red, and the third one is in green.

3.1.1. Class of DLOs

To simulate an industrial scenario, the robot manipulates the DLO at a certain high from the working table, where there are positioned different kinds of DLO as shown in Figure 3.5.

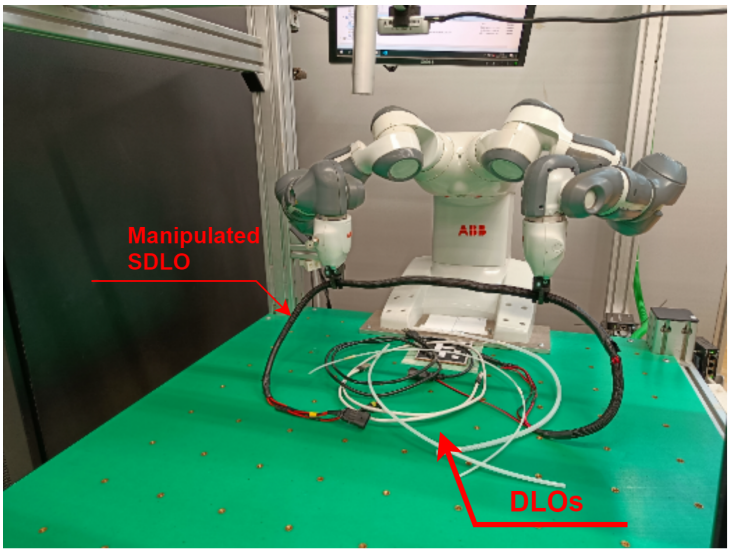


Figure 3.5: Manipulated DLO with the presence of DLOs in the background.

All the DLOs used during the manipulation or in the background can be seen in Figure 3.6 and they are described in Table 3.2. Considering that the DLOs are made of composite materials and we don't have any datasheet on their Young modulus, the submitted rigidity values are qualitative. They were defined as follows:

- low rigidity $\simeq 1e^6 Pa$;
- medium rigidity $\simeq 1e^7 Pa$;
- high rigidity $> 1e^7 Pa$.

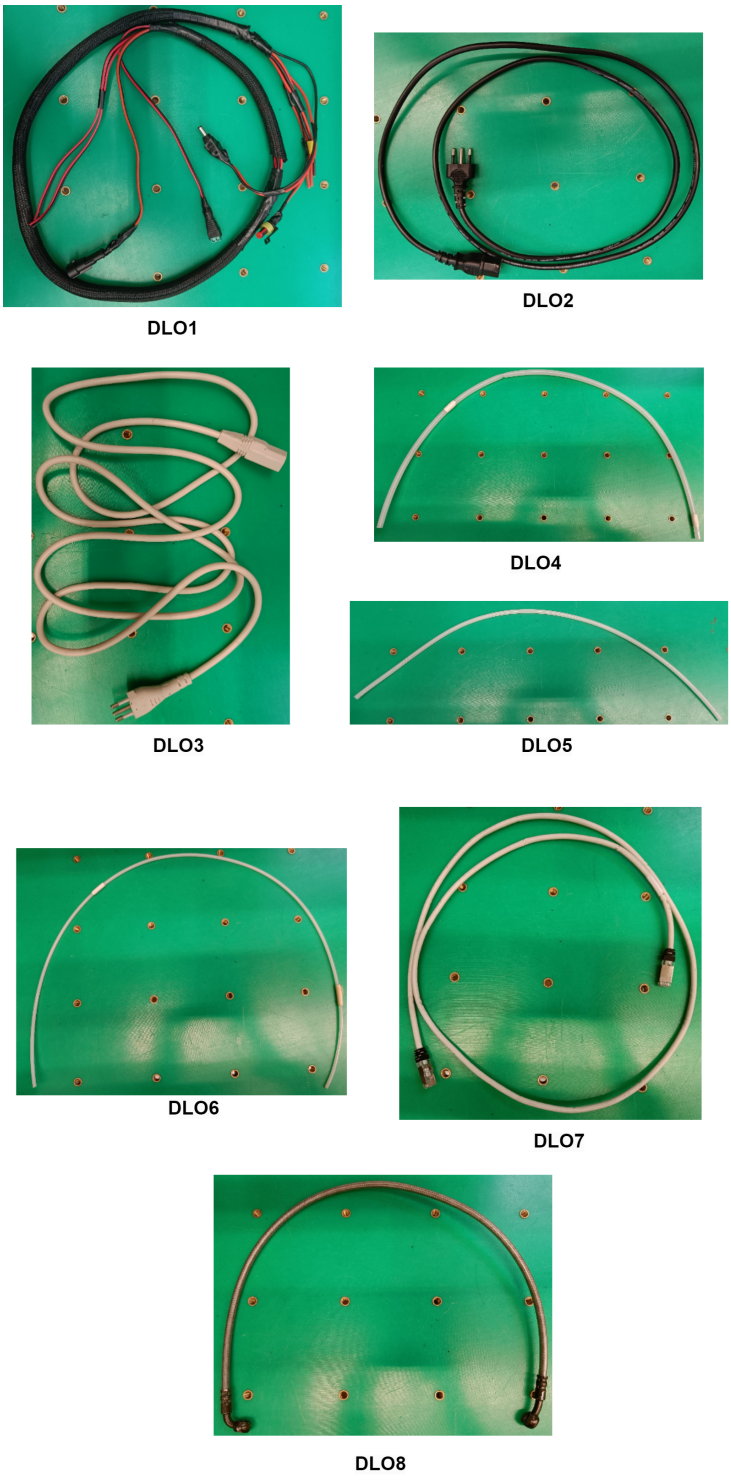


Figure 3.6: Class of DLOs.

	Diameter	Length	Peculiarity	Rigidity
DLO1	1.2 cm	1.55 m	black branched SDLO	low
DLO2	0.7 cm	1.95 m	black power cord	middle
DLO3	0.8 cm	2.36 m	white power cord	middle
DLO4	0.8 cm	0.78 m	hose for compressed air, translucent	high
DLO5	0.6 cm	0.61 m	hose for compressed air, translucent	high
DLO6	0.4 cm	0.85 m	hose for compressed air, translucent	high
DLO7	0.6 cm	1.53 m	white Ethernet cable	high
DLO8	0.75 cm	0.72 m	hose of a motorbike braking system, metallic	highest

Table 3.2: Specification of used DLOs.

3.2. Computer vision tools

Computer vision is a subfield of computer science that enables the extraction of meaningful information from digital images, videos, and other visual inputs.

3.2.1. Image representation

When an image is digitized, it is composed of a series of pixels represented in the "image plane" having the origin in the top left corner of the image (Figure 3.7).

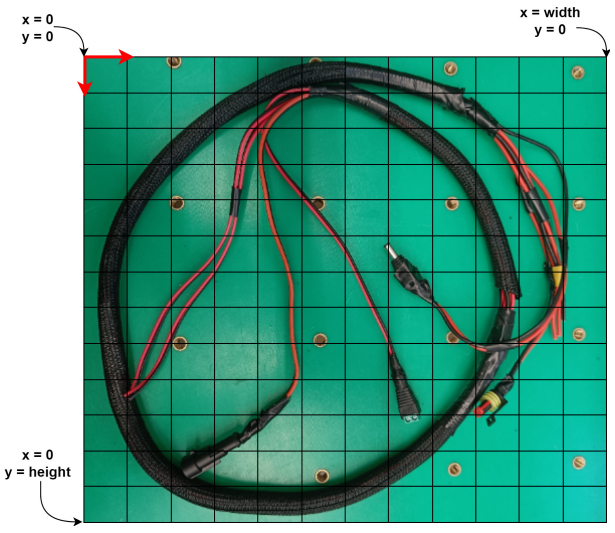


Figure 3.7: Digitized image in the "image plane".

The image dimension is described by the following tuple: (height, width, number of channels), where the number of channels depends on the type of color model used. In the case of grayscale, the number of channels is 1 and the image is represented by a 2D matrix, where each pixel assumes a value ranging from 0 to 255 that describes its color in a scale from white to black.
If colors are also taken into account, then the number of channels is 3, so the image is represented by a 3D matrix. The most used color model is the RGB, where each pixel defines in terms of a number ranging from 0 to 255 the amount of red, green, and blue to express its color. Alternative color models that will be used in this thesis (see Chapter 4) are the BGR and the HSV. The first one is like the RGB, but changes only the order of definition of the color. The HSV is a cylindrical color model that specifies each pixel color in terms of hue, saturation, and value (Figure 3.8). The hue represents the color, the saturation represents the amount of color used and the value represents the brightness. The hue range is [0,179], the saturation range is [0,255] and the value range is [0,255].

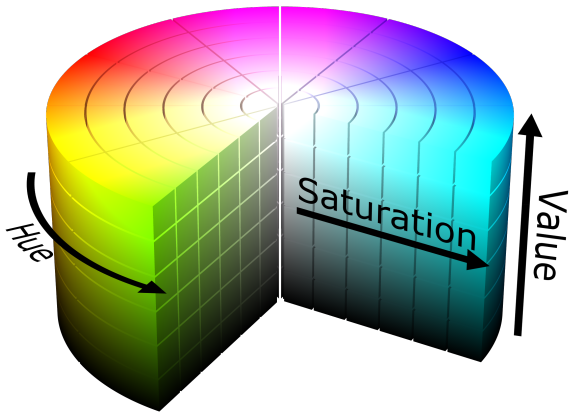
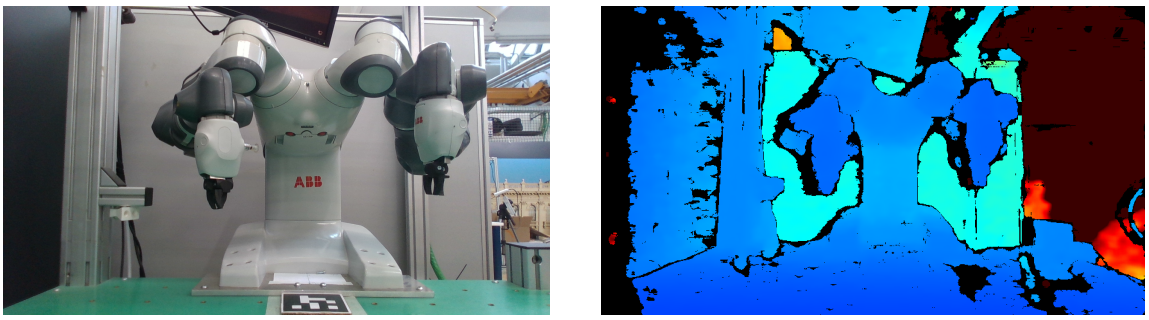


Figure 3.8: HSV color model [17].

Our camera is an RGB-D one, so it gives as output not only the color image but also the depth image, also called the depth map. It is an image where each pixel contains only information about the related distance of the scene objects from the camera viewpoint, for this reason, it has one channel. A depth map has different color representations that associate the distance to color, the one that will be used in this thesis is the *jet*.
(Figure 3.9) shows an example, the jet colormap goes from blue to red based on the distance from the camera and the black parts are the "holes".



(a) (b)

Figure 3.9: (a) RGB image. (b) Depth image in jet colormap.

The Holes are common in stereo depth camera and represent depth data unavailable or that did not meet the confidence metric, and instead of providing a wrong value, the camera provides a value of zero at that point. They commonly result from:

- Occlusions, the left and the right images do not see the same object due to shadowing;
- lack of texture, stereo matching relies on matching texture in the left and right images, so for texture-less surfaces like a flat white wall, the depth estimate can be challenging;
- multiple matches, this happens when there are multiple equally good matches, such as when looking at a very uniform periodic structure;
- no signal, this happens if the images are under-exposed or over-exposed.

However, there are strategies to deal with them that will be presented in Section 4.1.

3.2.2. Camera calibration matrix

The camera needs to be calibrated before use. There are two types of calibration:

- internal calibration: determination of intrinsic parameters of the camera;
- external calibration: determination of the extrinsic parameters of the camera like the position and the orientation of the camera with respect to a chosen frame (e.g. the base frame of the robot).

Since the intrinsic parameters were already available, in the following the external calibration is detailed. Let us introduce the homogeneous transformation matrix :

$$A_i^j = \begin{bmatrix} R_i^j & t_i^j \\ 0_{3\times1} & 1 \end{bmatrix}$$

A_i^j is a 4×4 matrix that relates the position and orientation of a point in the frame i with the position and orientation in the frame j. R_i^j is a 3×3 matrix is the rotation matrix of the frame i with respect to the frame j. t_i^j is a 3×1 vector that defines the translation of the frame i with respect to the frame j. $0_{3\times1}$ is a row vector composed of 3 zeros.

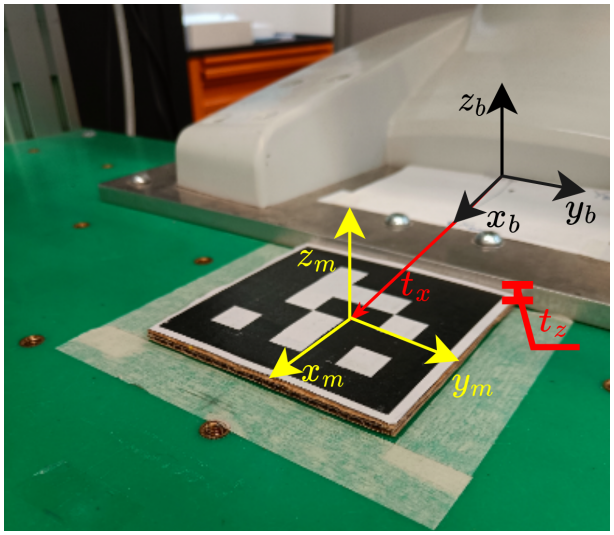


Figure 3.10: Aruco marker frame (yellow), robot base frame (black), translation vectors (red).

Considering Figure 3.10, with $t_x = 0,205\ m$ and $t_z = -0,005\ m$, we can define the homogeneous transformation matrix of the marker frame with respect to the robot base frame:

$$A_m^b = \begin{bmatrix} 1 & 0 & 0 & 0,205 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0,005 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

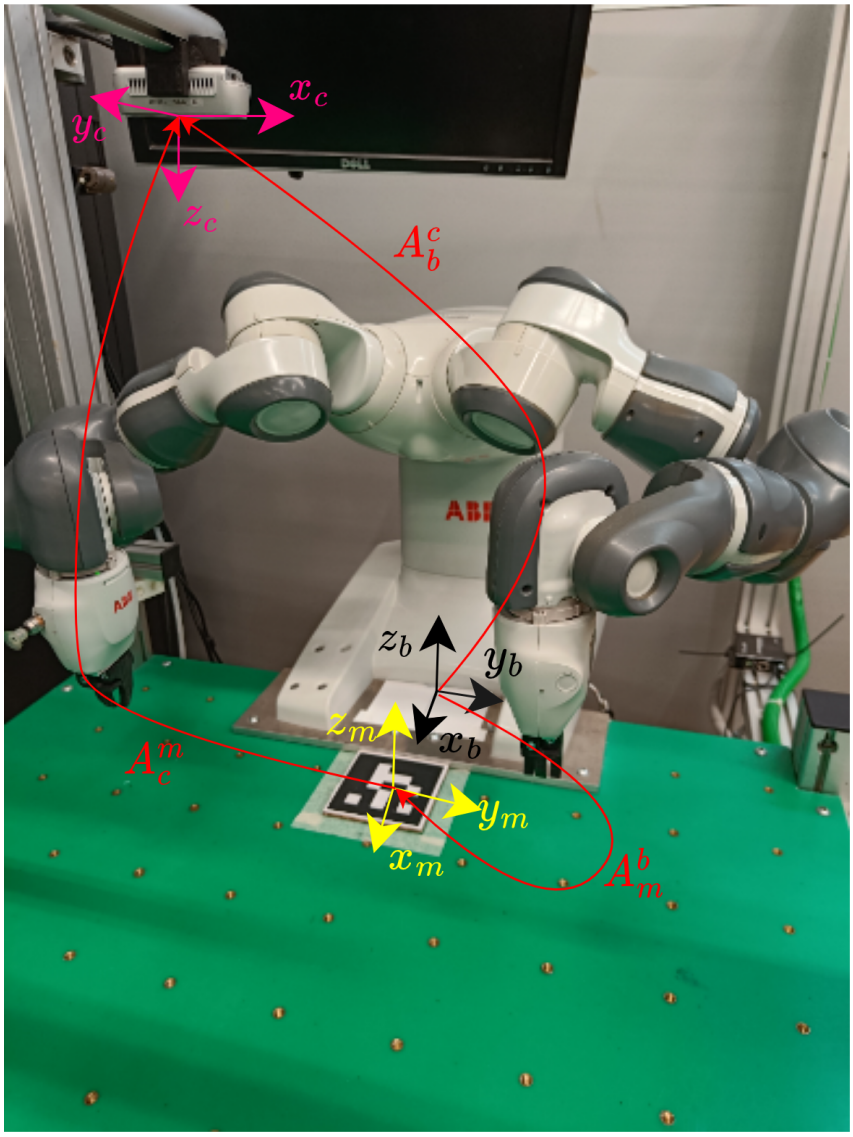


Figure 3.11: Marker frame (yellow), robot base frame (black), camera frame (pink), and homogeneous transformation matrices (red).

Since we are interested in expressing the tracked point in the coordinates of the base frame of robot and vice versa, the matrix A_c^b (the homogeneous transformation matrix of the robot base frame with respect to the camera frame) has to be computed. So by making reference to Figure 3.11, it follows:

$$A_c^b = A_b^m A_c^m \tag{3.1}$$

where A_c^m is the homogeneous transformation matrix of the camera frame with respect to the marker frame, which can be easily computed by detecting the marker with the camera (see Section 4.1 for details). It follows that, from eq. (3.1), given a point p^c in the camera

frame, it's possible to obtain its coordinate expressed in the robot base frame p^b, by:

$$p^b = A_c^b \cdot p^c \tag{3.2}$$

4 | DLO shape tracking algorithm

This chapter describes the proposed tracking algorithm, developed with the aim of tracking different types of DLOs (described in Section 3.1.1) after they are grasped at their ends and raised at a certain high from the working table. As described in Chapter 3, there are different kinds of DLOs on the working table.
The implementation was carried out using Python as the main programming language. The main-third party libraries used are:

- pyrealsense2[1] to interface with the Intel Realsense camera;
- Numpy [18] to convert the frames and the data as an array and to manage them;
- Open3D [19] for point-cloud creation and manipulation;
- Scikit-learn [20] for evaluation of the fitting error and for construction of the fitting problem;
- OpenCV [21] for the calibration of the camera and for the vision operation on the frames, as the color mask.

Figure 4.1 shows the overall flow chart of the proposed tracking strategy. In this chapter we will go deep into the implementation of these blocks, in the following order:

- In Section 4.1 we will describe the pre-processing step;
- in Section 4.2 we will focus on the segmentation of the DLO, a fundamental step for the point-cloud creation;
- Section 4.3 will explain the point-cloud creation and its manipulation in order to obtain only a few but important points;
- In Section 4.4 we will concentrate our attention on how to perform the fitting, in order to obtain a function describing the shape of the tracked DLO.

Afterward, we will focus on the occlusion problem.

[1] `https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.html`

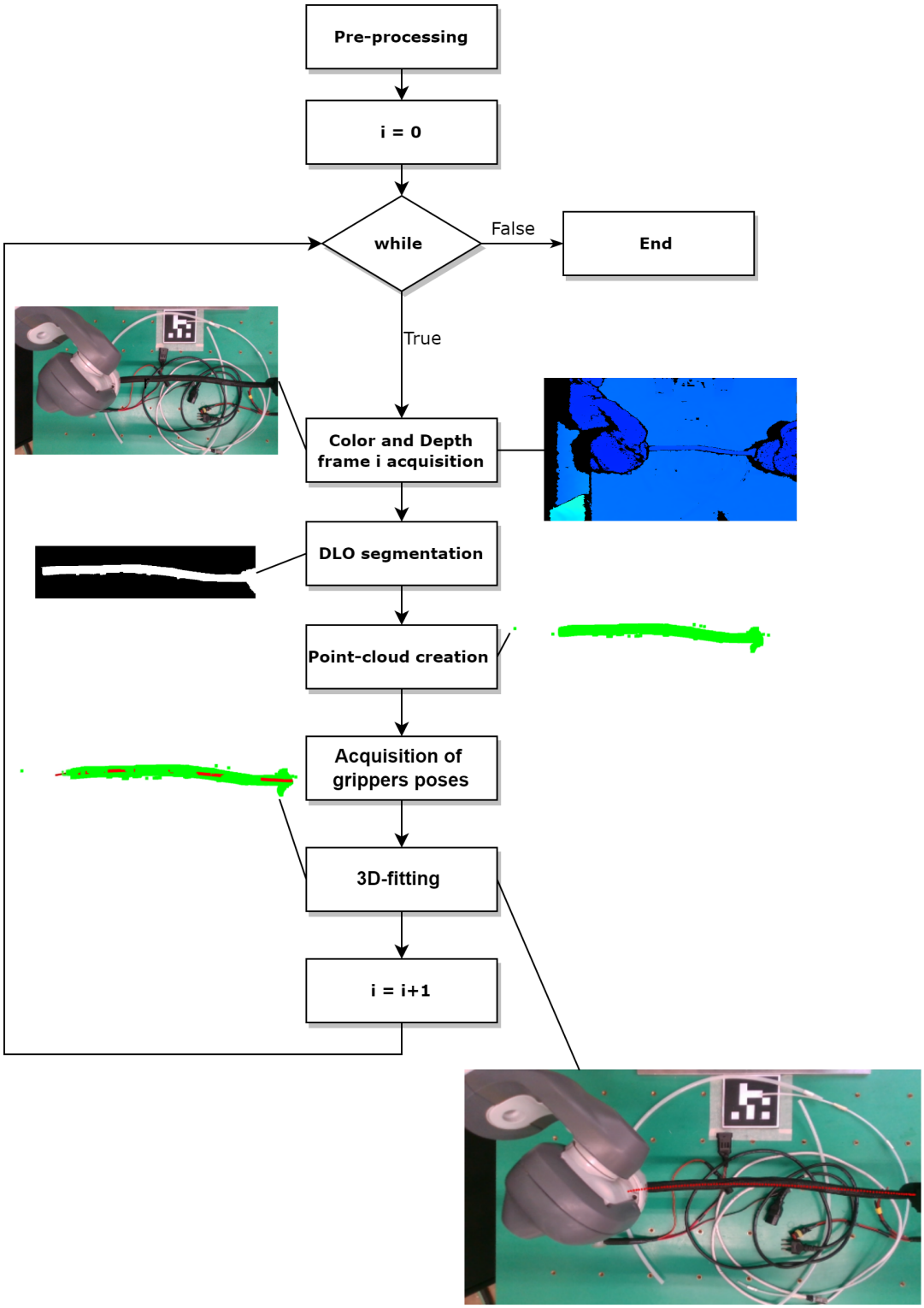


Figure 4.1: High-level flow chart of proposed tracking strategy. The red points represent the tracked shape of the DLO.

4.1. Pre-processing

In this phase, the initial settings for the tracking algorithm are defined.
Firstly we need to calibrate the camera in order to obtain the matrix $(A_c^b)^{-1}$. This matrix is obtained thanks to *cv2.aruco* library that takes as input a frame from the camera, it allows to detect the aruco marker (Figure 4.2) and to obtain R_m^c and t_m^c, respectively the rotation matrix and translation vector of the marker frame with respect to the camera frame. Obtaining them, the matrix A_m^c, the homogeneous transformation matrix of the aruco marker frame with respect to the camera frame, can be constructed. Moreover, knowing that $A_c^m = (A_m^c)^{-1}$ and following eq. (3.1), we can obtain A_c^b. This will be useful in Section 4.3.1. The pros of having a camera in eye-to-hand configuration is the fact that this step can be done offline or only for the first use of the algorithm.

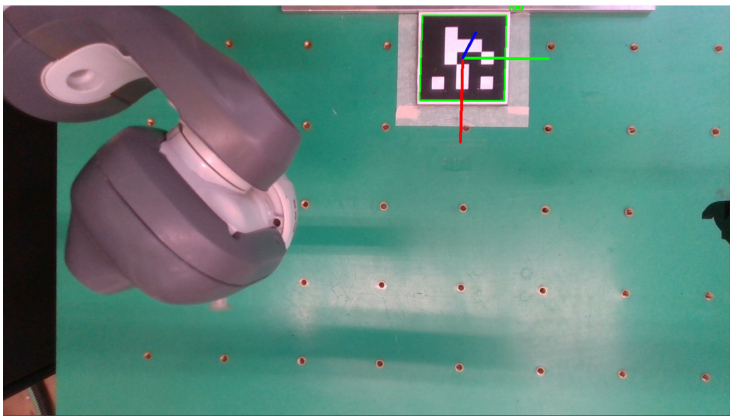
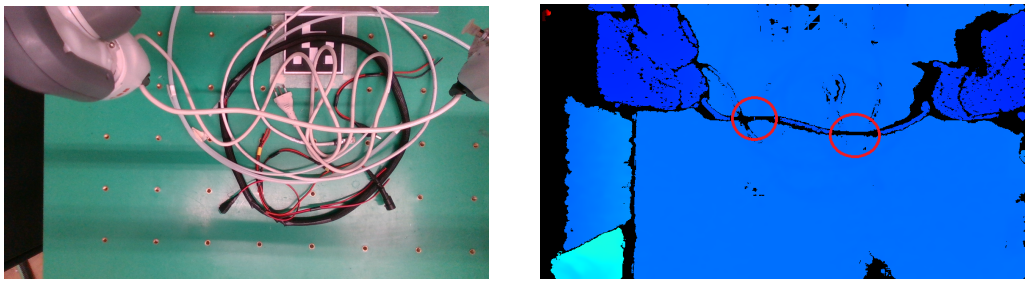


Figure 4.2: Detected aruco. The red line is the x-axis, the green line is the y-axis and the blue line is the z-axis.

Then we need to connect as a client through socket with the YuMi robot (further details in Section 5.1). Furthermore, the next step is the configuration of the camera.
Figure 4.3 shows an RGB frame and depth frame during a DLO manipulation. Focusing our attention on Figure 4.3b, the DLOs in the background "disappear" because they are at the same depth as the working table. In particular, we can notice the holes on the DLO circled in red, these holes need to be filled.



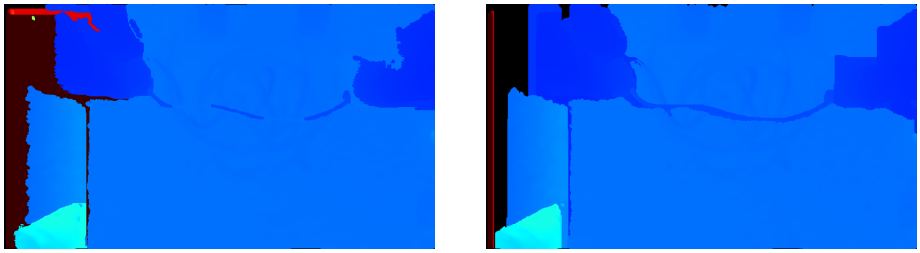
(a) (b)

Figure 4.3: (a) Color frame. (b) Depth frame in jet colormap with holes circled in red.

The holes problem can be mitigated using the $hole\_filling\_filter$ function implemented in the *pyrealsense2* library. This function takes as input a number in the range from 0 to 2, that corresponds to different filling logics:

- 0 is the 'fill from left', use the value from the left neighbour pixel to fill the hole (Figure 4.4a);
- 1 is the 'farest from around', use the biggest (farthest away) value among the valid five upper left and down pixel values (Figure 4.4b);
- 2 is the 'nearest from around', use the smallest among the valid five upper left and down pixel values (Figure 4.4c).

(a)



(b) (c)

Figure 4.4: Type of hole filling. (a) 'Fill from left'. (b) 'Farest from around'. (c) 'Nearest from around'.

In the considered case, the 'farest from around' method does not resolve the problem of the holes. Then the 'nearest from around' method takes as a reference more pixels around the hole, so it gives the same depth of the DLO also to the background. Differently, the 'fill from left' method, fills the hole with the best possible depth value taking only the left neighbour pixel. For this reason, the fill from left method is selected in our pipeline.
The other necessary camera preprocessing are:

- create and configure the pipeline to stream the color and depth information from the camera;
- align the depth stream to color stream, in order to have a one-to-one association between the pixels in the color frame and in pixels in the depth frame, necessary in the point-cloud creation;
- acquire the camera intrinsic parameters useful in Section 4.3;
- acquire the depth sensor scale useful in Section 4.2.1. The depth sensor scale is a constant number, which depends on the sensor, that maps the units of the depth image into meters:

$$depth\ image\ in\ meters = depth\ sensor\ scale \times depth\ image\ units$$

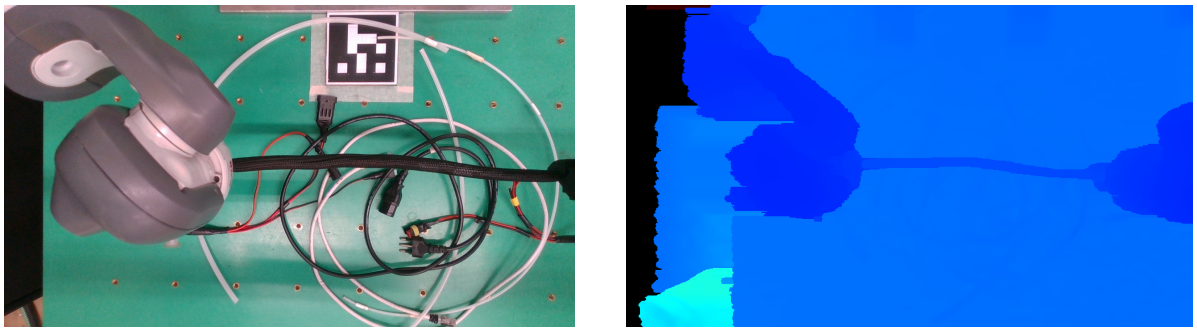
4.2. DLO segmentation

After the pre-processing step, we can enter in the description of the DLO segmentation algorithm. The acquisition of the color and the depth frame of Figure 4.1 can be split into different sub-steps:

- acquire the frame using the function $pipeline.wait\_for\_frames$ that wait until a new set of frames becomes available. If a frame is available by the time your processing finishes, it will return immediately, if not, it will block and free up the CPU until the next frame is available. In this way, the acquisition is less computationally heavy;
- apply the hole filling on these frames;
- apply the alignment of hole filled depth frame to the color frame;
- acquire the depth and the color frame and convert them in arrays using $numpy.asanyarray$;
- convert the color image from RGB to BGR, the color model used in the *OpenCV*

library.

Figure 4.5 shows the result after the acquisition of the color and the depth frame.



(a) (b)

Figure 4.5: Results after the acquisition of color frame and depth frame. (a) Color image after the BGR conversion. (b) Depth image after the hole filling.

4.2.1. Depth filtering

A challenge of the proposed algorithm is to track the DLO with other DLOs or objects, even those of the same color as the tracked DLO, in the background. Consider our assumption that the DLO is manipulated at a certain height from the working table, hence the idea is to isolate the DLO from what is above and below it. This isolation is made by implementing a depth filter whose steps and logic are shown in Figure 4.6.

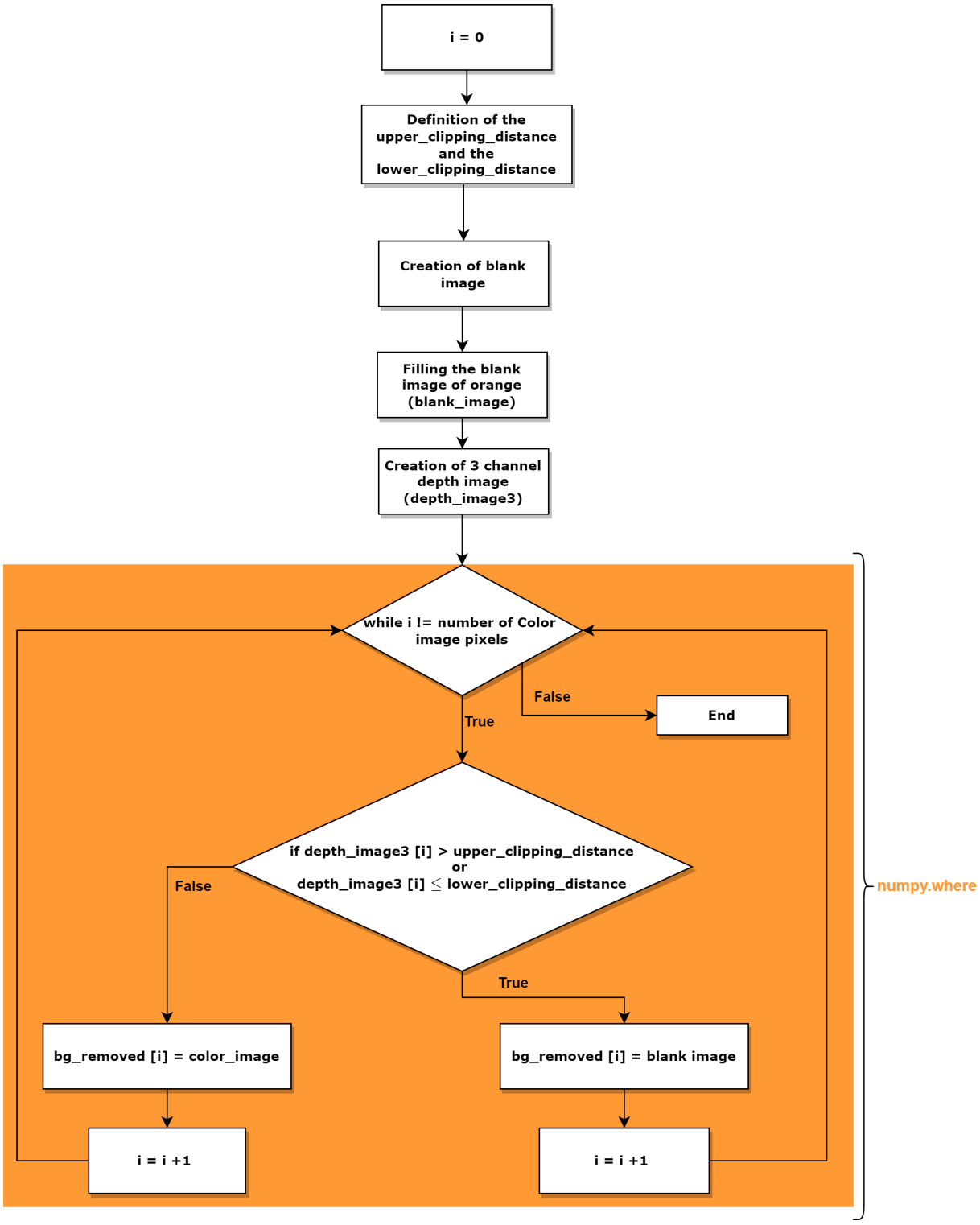


Figure 4.6: Depth filter flow chart, in orange the logic of the main used function: *numpy.where*.

Knowing the height in meters at which the DLO is manipulated, it's possible to obtain an upper and lower limit:

$$upper\_clipping\_distance = \frac{upper\ limit\ in\ meters}{depth\ sensor\ scale}$$

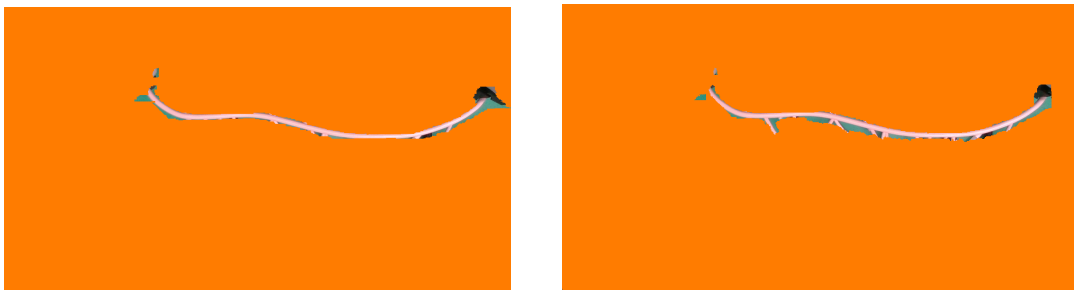
$$lower\_clipping\_distance = \frac{lower\ limit\ in\ meters}{depth\ sensor\ scale}$$

with the depth sensor scale acquired during the pre-processing. Note that the heights interval where the DLO is manipulated can be obtained considering the grippers positions. Subsequently, we need to create a blank image corresponding to a numpy array with the same dimension and data type of the color image. And then fill each element of this array with value (0, 124, 255), that correspond to the orange color in BGR. It was chosen this color because is different from the DLO considered in this thesis (Section 3.1.1) or in general from the DLO used in an industrial scenario.
The depth filter returns a new color image based on the depth one. Nevertheless, the depth image has only 1 channel, so is necessary to create a 3 channel depth image using *numpy.dstack*.
The main function is *numpy.where* (evidenced in orange in Figure 4.6) that compares each pixel of the 3 channel depth image with the upper_clipping_distance and the lower_clipping_distance, if this value is between these two limits, it will be assigned to this pixel the color image pixel, otherwise it will be assigned the blank image one. Iterating this logic, it will be created a new image called bg_removed with all in orange except the DLO (Figure 2.3).

Figure 4.7: DLO1 (defined in Section 3.1.1) after the depth filtering.

The depth filter ensures that the tracking algorithm is completely unaffected by the background, which can also dynamically change. The application of the hole filling before the

depth filtering guarantees that the DLO is not divided into different pieces by the filter. In addition, Figure 4.8 gives demonstrate that the nearest from around hole filling does not isolate only the DLO3 (defined in Section 3.1.1) but also the background.



(a) Fill from left. (b) Nearest from around.

Figure 4.8: Comparison of the hole filling methods after the depth filtering.

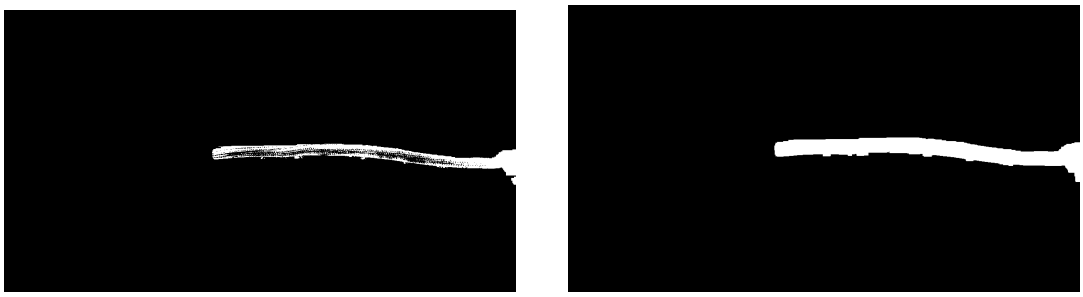
4.2.2. Color mask

Once the DLO is partially isolated by the depth filter, is possible to apply a color mask. The color mask searches in each frame the pixels with a certain specified color, putting in white all the pixels with this feature and in black the ones without it. It is easy to understand the importance of applying before the depth filter and then the color mask, without the depth filter in this step the color mask would segment all objects of the same color of the manipulated DLO. The color mask gives as output a 1 channel image, that allows the segmentation of the DLO in each frame (Figure 4.9).

Figure 4.9: Result of the color mask applied to Figure 4.7.

A key phase that makes easiest the color masking is the conversion from the BGR model to the HSV model. The HSV has the advantage of separating the luma, or the image

intensity, from the chroma, or color information. This means that in the case of an image with a shadow, if we remain in RGB or BGR model the segmentation would fail because the part with the shadow has very different features than the part without it, it would be segmented only the part with the shadow or without it. Instead in the HSV color space, the two parts have a similar hue component, due to the fact it represents the color without brightness, while the shadow influences the other two components. Consequently, the HSV model ensures a more uniform segmentation of the DLO.
Afterward, the conversion is necessary to define a lower and an upper threshold based on the color of the DLO in terms of hue, saturation, and value. Using then the function *cv*2.*inrange* all the pixels between these limits are put in white, the others are put in black. The result shown in Figure 4.10a, is a mask image that colors in black anything that is not very close to DLO color.



(a) (b)

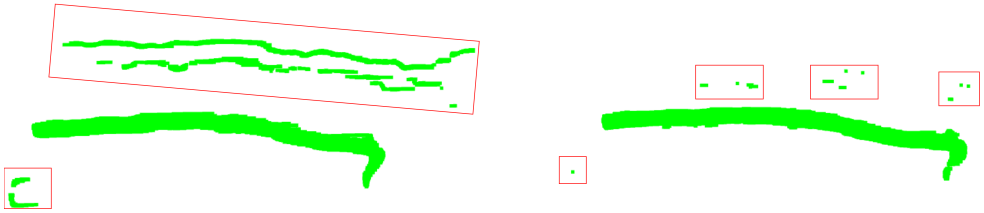
Figure 4.10: (a) Result of *cv*2.*inrange* function. (b) Result after the dilation transformation.

As can be noticed from Figure 4.10a, this operation creates black holes in the object, which can be resolved by applying the closing morphological transformation. This operation is composed of two subsequent transformations:

- firstly a dilation, where defining a kernel of a certain size, slides through the image. A pixel is considered white if at least one pixel under the kernel is white, otherwise is black. So after this operation, the holes are filled but the DLO became thicker (Figure 4.10b);
- Then is applied an erosion, a pixel is considered white only if all the pixels under the kernel are white, otherwise it is black. So after this operation, the DLO returned the original thickness but with the holes filled (Figure 4.9).

Even though the Figure 4.10b seems very close to the result obtained after the erosion step (Figure 4.9), the erosion ensures the construction of the point-cloud (further details

on the construction in Section 4.3) with a lower number of outliers (Figure 4.11).



(a) (b)

Figure 4.11: (a) Point-cloud applying only the dilation, with the outliers in red rectangles. (b) Point-cloud applying the closing morphological transformation, with the outliers in red rectangles.

4.2.3. Contour extraction

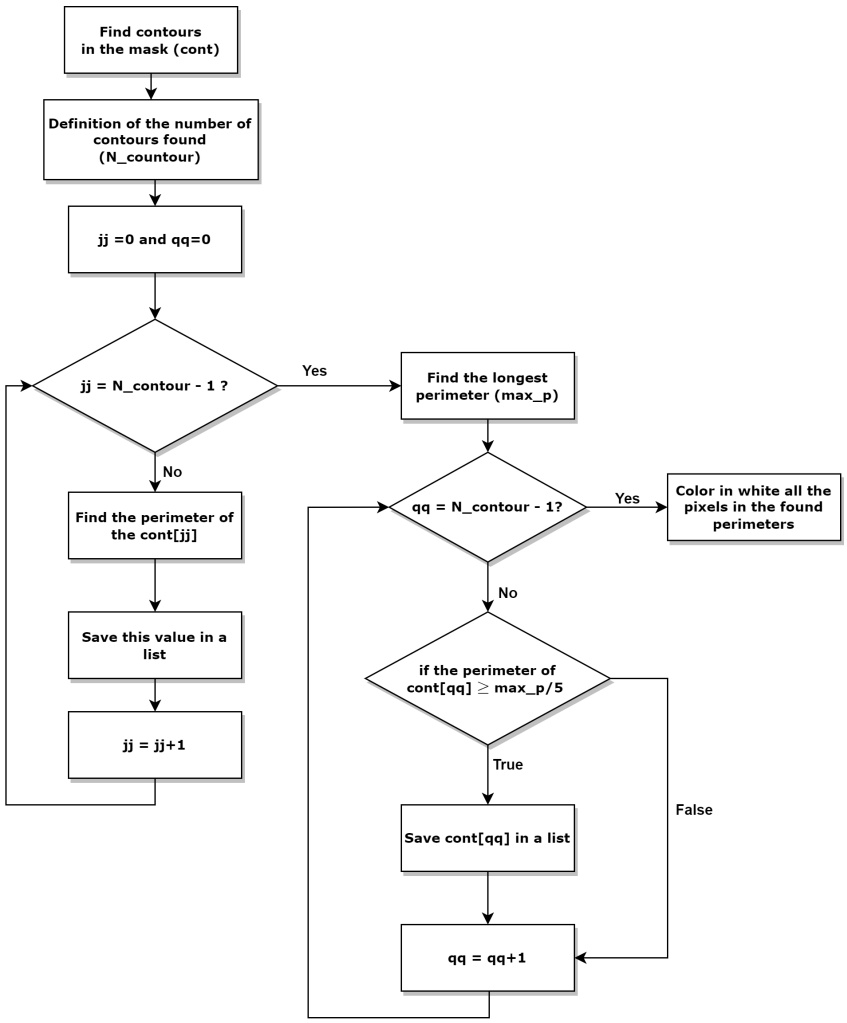
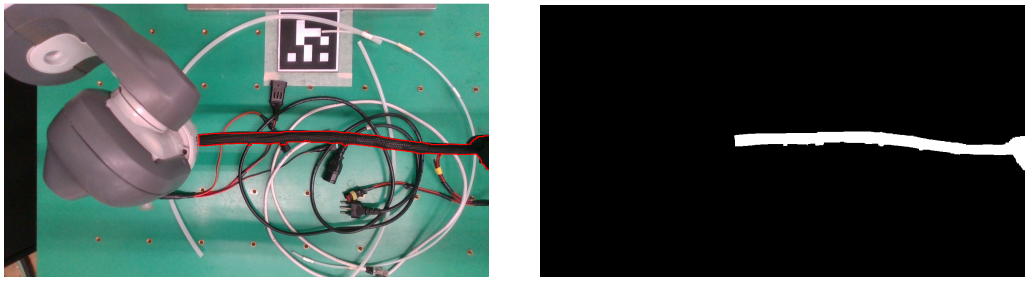


Figure 4.12: Contour extraction flowchart.

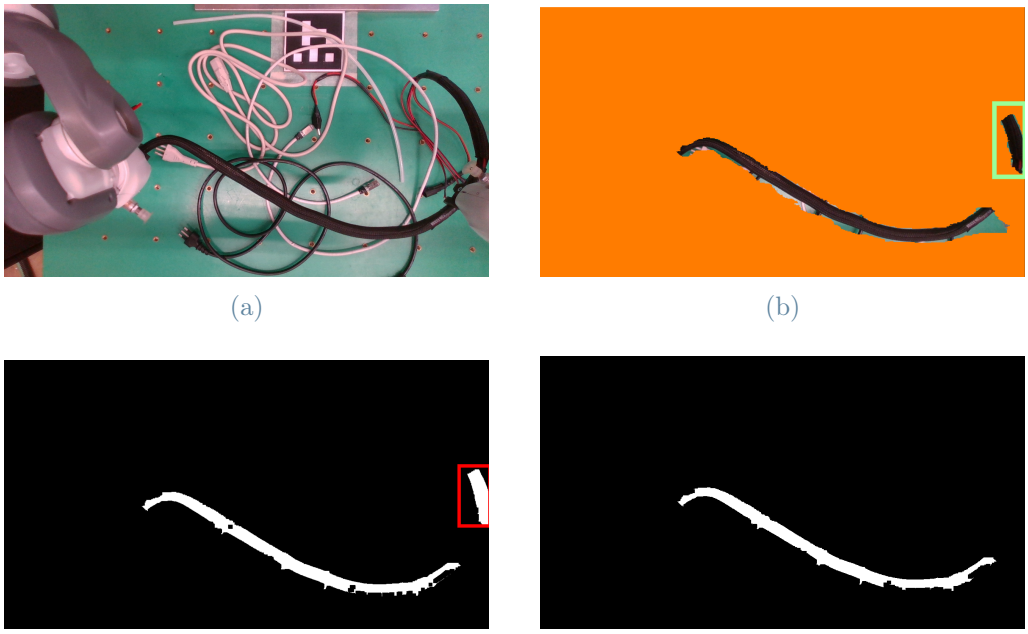
Once obtained the output of the color mask, we can apply a contour extraction algorithm on it. The algorithm is schematized in Figure 4.12. It starts finding all the contours in the 1 channel image (Figure 4.9) using the function $cv2.findcontours$, specifying none as an approximation method, this means that it will save all points along the boundaries, having the same color and intensity. It then records all the discovered contours' perimeters in a list, which it will use to identify the longest one that is certain to correspond with the DLO (Figure 4.13a) due to previous phases. After are found all the perimeters that are bigger or equal $\frac{1}{5}$ of the longest perimeter. Finally, all the pixels in the found perimeters are colored in white and the image is transformed in grayscale (Figure 4.13b), in this way the proposed online algorithm becomes more computationally efficient.



(a) (b)

Figure 4.13: (a) The biggest found perimeter colored in red on the color frame. (b) The output of the contour extraction algorithm.

The contour extraction step ensures the segmentation only of the DLO between the two grippers, extending the use of this tracking methodology to different kinds of DLOs without any limit on their lengths. Figure 4.14a, shows the DLO1 manipulated to follow a sinusoidal shape (Section 6.2), the part of it after the right gripper has the same depth value as the part between the grippers. For this reason, the depth filter isolate also the part after the right gripper (in green in Figure 4.14b). Therefore, if we only use the color mask, it would segment both parts (Figure 4.14c) and this will affect the final tracked shape. Instead, the contour extraction (Figure 4.14d) ensures the segmentation only of DLO between the gripper since the part after the right gripper has a perimeter too small with respect to the perimeter of the DLO between the gripper.

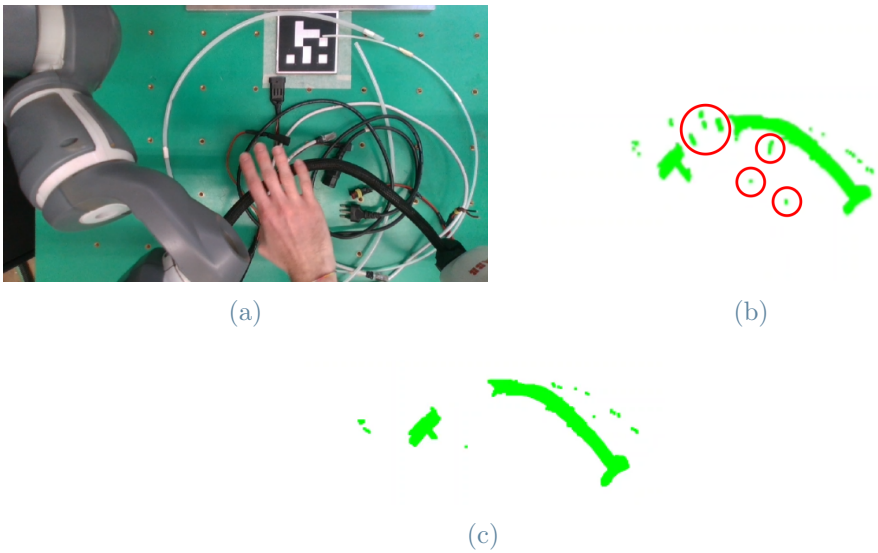


(a) (b)

(c) (d)

Figure 4.14: Difference between the use or not of the contour extraction step. (a) Color frame. (b) Output of the depth filter, in the green rectangle the DLO part after the right gripper. (c) The output of the color mask, in the red rectangle the DLO part after the right gripper. (d) The output of the contour extraction step.

In addition, it ensures the construction of the point-cloud (further details in Section 4.3) with a lower number of outliers, due to the fact they are filtered by the perimeter threshold. This can be noticed from Figure 4.15, where a hand occludes the DLO. In Figure 4.15b the outliers generated by the hand are circled in red, while applying the contour extraction step (Figure 4.15c) these outliers are deleted. In addition, it adds robustness to the occlusions, due to the fact that during the occlusion the DLO becomes piece-wise, and with the contour-extraction step we have a more faithful reconstruction of the DLO.



(a) (b)

(c)

Figure 4.15: Difference between the use or not of the contour extraction step in case of occlusion. (a) Color frame. (b) Point-cloud obtained by skipping the contour extraction step. (c) Point-cloud obtained with the contour extraction step.

4.3. Point-cloud creation

A point-cloud is a representation of the object in space, where each point position has its set of Cartesian coordinates (X_s, Y_s, Z_s). So we need to pass from a point in the image plane $p_i = (y_i, x_i)$ to its corresponding point in the space $p_s = (X_s, Y_s, Z_s)$ described by the geometric coordinates in meters.
After the execution of the previous phase, the pixels of the DLO are in white (Figure 4.13b), so the pixel coordinates can be extracted from it. Knowing that our color frame and depth frame are aligned, for each pixel the Z coordinate shall be calculated, using the aligned depth frame information.
Finally, the function $pyrealsense2.rs2\_deproject\_pixel\_to\_point$ for each extracted pixel is used, which takes as input the intrinsic parameter of the camera (calculated previously in the pre-processing step), the pixel coordinates in the image plane and the calculated Z, and gives as output the point in the geometric coordinates, constructing a $P \times 3$ matrix that contains all the coordinates of the points.
The Open3D library was used to visualize the point-cloud, it constructs the point-cloud object associating to it the founded points and coloring them in green. As shown in Figure 4.16a, it is necessary to apply a rotation of the point-cloud reference frame in order to have a coherent visualization of the point-cloud with the camera frames (Figure 4.16b).

This is possible by applying the 4×4 homogeneous transformation matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which correspond to a rotation of $-\pi$ respect to the x axis.

(a) (b)

Figure 4.16: (a) Point-cloud without transformation. (b) Point-cloud after the application of the transformation.

Then a downsample of the point-cloud is necessary, in order to make the fitting (Section 4.4) computationally efficient. This is done using the Open3D function *voxel_down_sample* which has only one parameter to specify the voxel size. The points are contained in voxels, then each occupied voxel generates only one point by averaging all points inside. The result obtained with a voxel size equal to 0.01 is shown in Figure 4.17a, which are also highlighted in red all the outliers. The outlier can be removed using the functions *statistical_outlier_removal* that removes points that are further away from their neighbor compared to the average for the point-cloud. The inputs parameter are:

- the number of neighbors that are taken into account in order to calculate the average distance for a given point;
- the standard ratio, defines a threshold based on the standard deviation of the average distance across the point-cloud. The lower this number the more aggressive the filter will be.

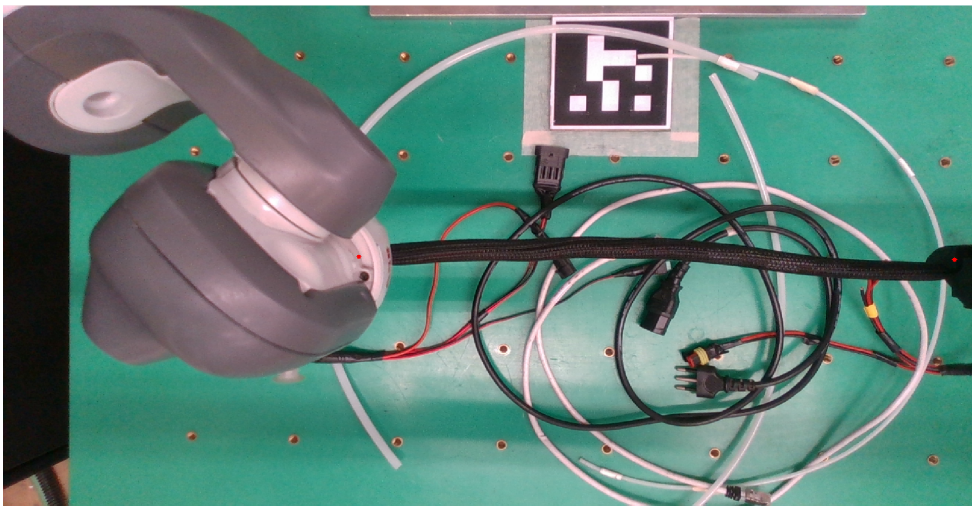
Subsequently is applied the *select_by_index* function that outputs only the selected point in the previous step. Figure 4.17b showed the result with the number of neighbors equal to 30 and the standard ratio equal to 0.1.

(a) (b)

Figure 4.17: (a) Application of the voxel down sampling, in red are highlighted the outliers. (b) Result after the outlier removal.

4.3.1. Acquisition of grippers poses

The application of the voxel downsample and of statistical outlier removal, with the chosen value for the parameter, ensures the right balance between the number of points and meaningful points in the point-cloud for all the DLO considered. However, there can be cases where these points are not enough for good tracking, due to: occlusion, the presence of a particular light condition, or a particular characteristic of the DLO.
One of the ways to resolve these problems is to acquire during the tracking the grippers poses. For this reason, in the color mask phase (Figure 4.9) we have decided to isolate the gripper pixels as well, in order to have more points at the end of the DLO. In addition, the DLO during the manipulation can be occluded from the robot arm (Figure 4.5a). For this reason, we acquire from the robot the points where it grips the DLO (further details will be given in Chapter 5). This way during the tracking, we have two points (one for each gripper), that are not influenced by the previous problems. These points are in the robot base frame, for this reason following they are firstly multiplied by $(A_c^b)^{-1}$ (eq. (3.2)) in order to be reported in the camera frame and then are added to the filtered point-cloud of the previous sub-section (Figure 4.18b). Figure 4.18a shows the acquired gripper point in red after that they are reported in the image plane.



(a)

(b)

Figure 4.18: (a) Color frame with the acquired grippers points in red. (b) Filtered point-cloud with grippers points in red.

4.4. 3D-Fitting

After the previous phase, we obtained a point-cloud that represents the DLO in space. The idea is to use these points to construct a geometric estimation of the DLO shape, by applying a 3D-fitting. The 3D-fitting is applied by choosing the x-axis as fixed and then fits the other two sets of coordinates with respect to the x-axis. In this way, the fitting problem becomes more simple and computationally efficient. Figure 4.19 shows the result of the 3D-fitting, as can be seen, while in the color frame (Figure 4.5a) the DLO seems without any changing on the z-axis, in the 3D plane there are some changes of the order of the cm that are highlighted, consequently these changes are also acquired by our tracking algorithm.

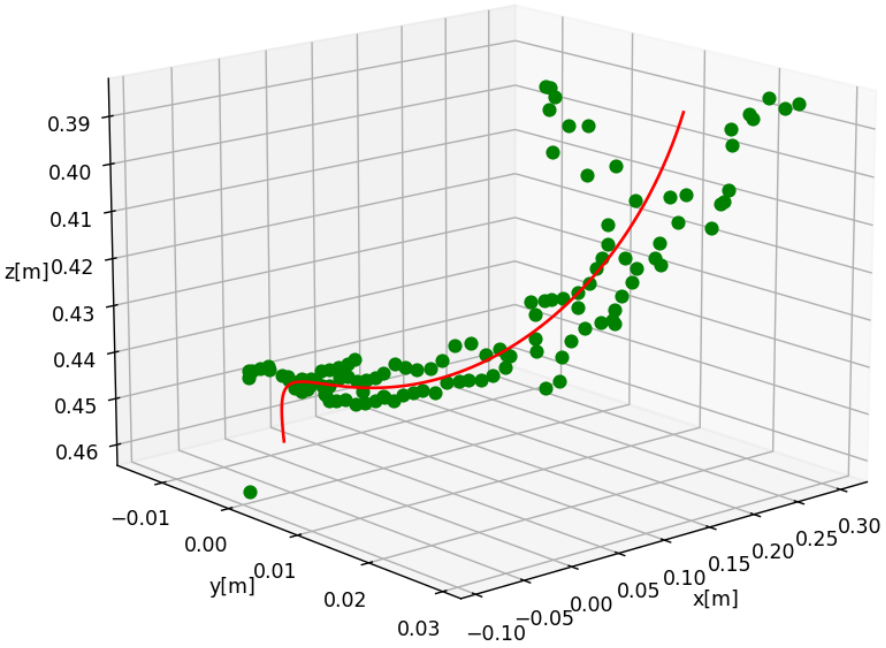
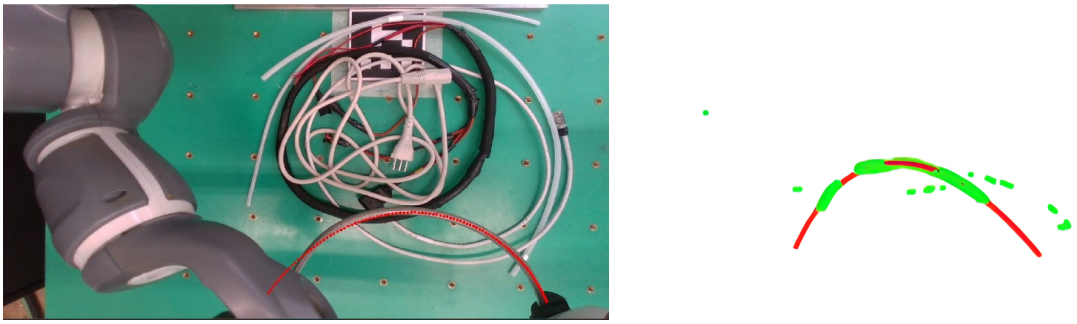


Figure 4.19: 3D-fitting plot. In green, the points acquired after the downsampling, the outlier removal and the acquisition of gripper pose. In red, the tracked shape of the DLO as a result of the 3D-fitting.

In addition, the proposed methodology can also track shape with a higher variation on the z-axis as it showed in Figure 4.21. Where the DLO8 is manipulated to follow a quadratic function shape (further details on the shapes considered in the Chapter 6) in a plane parallel to the working table, due to the high rigidity of this DLO it is deformed upward (Figure 4.20).



(a) (b)

Figure 4.20: DLO8 quadratic function shape. (a) Tracked points (in red) plotted on the color frame. (b) Point-cloud (in green) with the tracked shape (in red).

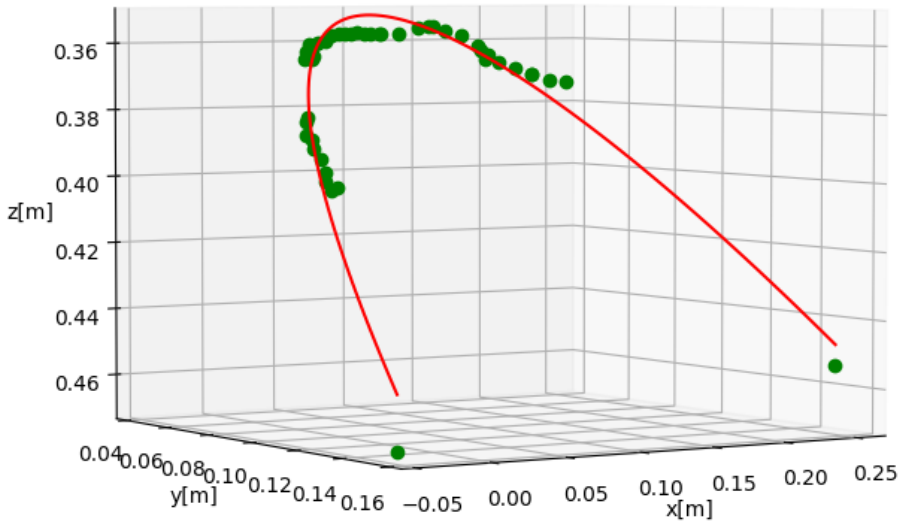


Figure 4.21: 3D-fitting plot of DLO9 in a parabola shape. In green the points acquired after the downsampling, the outlier removal and the acquisition of gripper pose. In red the tracked shape of the DLO as a result of the 3D-fitting.

4.4.1. Lasso Regression

The fitting problem is formalized as a linear regression problem with a regularization, in order to prevent overfitting.
Given the points described by (X, Y) in the 2D geometric plane, with $X = [X_1, X_2 ... X_P]^T$ and $Y = [Y_1, Y_2 ... Y_P]^T$, the fitting problem is formalized as the unconstrained optimization problem:

$$\min_{c} \quad \frac{1}{2P}||Y - Vc||_2^2 + r \tag{4.1}$$

where P is the number of points, r is the regularization term, c is the vector of the polynomial coefficient, $c = [c_0, c_1, c_2 ... c_P]^T$, and V is the Vandermonde matrix:

$$V = \begin{bmatrix} 1 & X_1 & X_1^2 & ... & X_1^{P-1} \\ 1 & X_2 & X_2^2 & ... & X_2^{P-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_P & X_P^2 & ... & X_P^{P-1} \end{bmatrix}$$

In machine learning, the most common regularization are :

- L1-regularization, where $r = \alpha||c||_1$, with α a positive constant and $||c||_1$ the $l_1 - norm$ of the coefficient vector;
- L2-regularization, $r = \alpha||c||_2^2$, with α a constant and $||c||_2^2$ the squared $l_2 - norm$ of the coefficient vector.

Differently from L2-regularization, L1 offers a built-in features selection, so this means that shrinks to zero the less important coefficient. For this reason, we decided to apply a Lasso regression (linear regression with L1-regularization), with the maximum degree of the Vandermorde matrix equal to four. We decided to stop up to the 4^{th} order degree polynomial because was experimentally tested which is sufficient to describe the shape that a DLO can assume. This means that, for each acquired frame, it was decided what is the best polynomial, from grade zero to four, that describes the DLO shape.
As we explain at the beginning of this section, the 3D fitting is applied by choosing the x-axis as fixed one, and then we fit the other two coordinates with respect to the x-axis. Accordingly, the 3D fitting problem is decomposed into two 2D Lasso regression problems one in the x-y plane and another for the x-z plane:

$$\begin{aligned} \min_{c_y} \quad & \frac{1}{2P}||Y - Vc_y||_2^2 + \alpha||c_y||_1 \\ \min_{c_z} \quad & \frac{1}{2P}||Z - Vc_z||_2^2 + \alpha||c_z||_1 \end{aligned} \tag{4.2}$$

formalized following Equation (4.1), with $Z = [Z_1, Z_2...Z_P]^T$ the vector of z coordinates of the point-cloud points. Solving the Lasso problems (Equation (4.2)), we can construct two polynomials one for the x-y plane and another for the x-z plane:

$$\begin{aligned} P_{x-y} &= c_{y4}X^4 + c_{y3}X^3 + c_{y2}X^2 + c_{y1}X + c_{y0} \\ P_{x-z} &= c_{z4}X^4 + c_{z3}X^3 + c_{z2}X^2 + c_{z1}X + c_{z0} \end{aligned} \tag{4.3}$$

which will describe the final tracked shape in 3D.
The custom optimization function *opt*, which is summarized in Figure 4.22, was defined for this purpose.

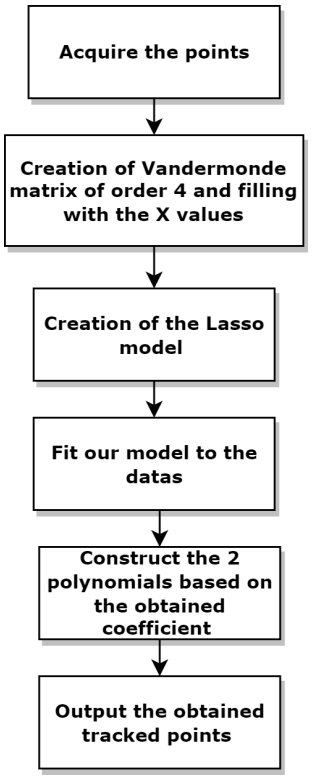


Figure 4.22: Flowchart of the *opt* function.

First, as explained by the chart in Figure 4.22, the points from the point-cloud are acquired, and each coordinate is saved in the vectors: $X = [X_1, X_2...X_P]^T$, $Y = [Y_1, Y_2...Y_P]^T$, $Z = [Z_1, Z_2...Z_P]^T$. Then is created the Vandermonde matrix with a maximum degree equal to 4, using the class $sklearn.preprocessing.PolynomialFeatures$ and fill it with the X vector:

$$V = \begin{bmatrix} 1 & X_1 & X_1^2 & X_1^3 & X_1^4 \\ 1 & X_2 & X_2^2 & X_2^3 & X_2^4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_P & X_P^2 & X_P^3 & X_P^4 \end{bmatrix}$$

The Lasso regression problems can be written according to eq. (4.2). So we can create our Lasso model, instantiating an object of class $sklearn.linear\_model.Lasso$ with parameters:

- *alpha*, that is the constant that multplies the l_1 norm. So it defines the importance of the regularization term. It is defined equal to $1e^{-7}$, which was proved experimentally that gives the right equilibrium between the least-squares penalty and the l_1 penalty;
- $fit\_intercept$, whether to calculate the intercept for this model. It is set to False in order to have also the zero-order term in the polynomial;

- *max_iteration*, the maximum number of iterations for the coordinate descent algorithm, that is used to fit the model. It was set equal to $1e^6$, such that the objective function converges with default tolerance $1e^{-4}$.

Then the created Lasso model is fitted with respect to the X-Y and X-Z vectors, solving the optimization problem of Equation (4.2). As a result, the coefficient vectors are acquired and are saved in two different 5×1 vector, $c_z = [c_{z0}, c_{z1}, c_{z2}, c_{z3}, c_{z4}]^T$ and $c_y = [c_{y0}, c_{y1}, c_{y2}, c_{y3}, c_{y4}]^T$.
Obtained the coefficients, it is defined a new 1×100 column vector:
$X_{new} = [X_{new1}, X_{new2}...X_{new100}]^T$, with 100 element that goes from the minimum to the maximum of X, and finally are defined the tracked points in the 100×3 matrix
$p_p = (X_{new}, Y_{new}, Z_{new})$. Where:

$$P_{x-y} = Y_{new} = c_{y4}X_{new}^4 + c_{y3}X_{new}^3 + c_{y2}X_{new}^2 + c_{y1}X_{new} + c_{y0}$$

$$P_{x-z} = Z_{new} = c_{z4}X_{new}^4 + c_{z3}X_{new}^3 + c_{z2}X_{new}^2 + c_{z1}X_{new} + c_{z0}$$

obtained applying Equation (4.3). The result is shown in Figure 4.19, where the red points are the tracked points, then they can be plotted on the point-cloud (Figure 4.23).

Figure 4.23: Tracked points (in red) plotted on the point-cloud (green).

These points can be reported in the robot base frame in order to give complete information on the tracked DLO shape or to be used in future works to apply a visual servoing control. Following eq. (3.2):

$$p_p^b = A_c^b \cdot p_p \tag{4.4}$$

Finally reporting these points in the image plane using the function
pyrealsense2.rs2_project_point_to_pixel, which takes as input the intrinsic parameter of the camera and the point in the 3D geometric plane, and it gives as output the corresponding point in the image plane. Hence the tracked point can be plotted on the color frame (Figure 4.24).

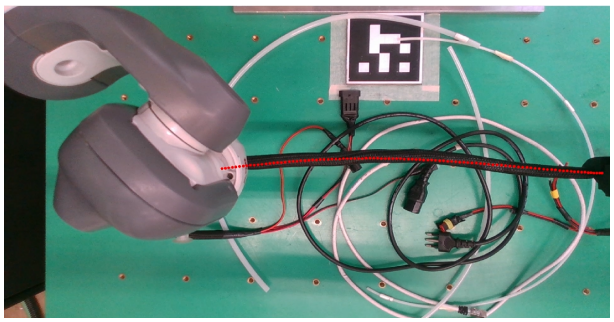
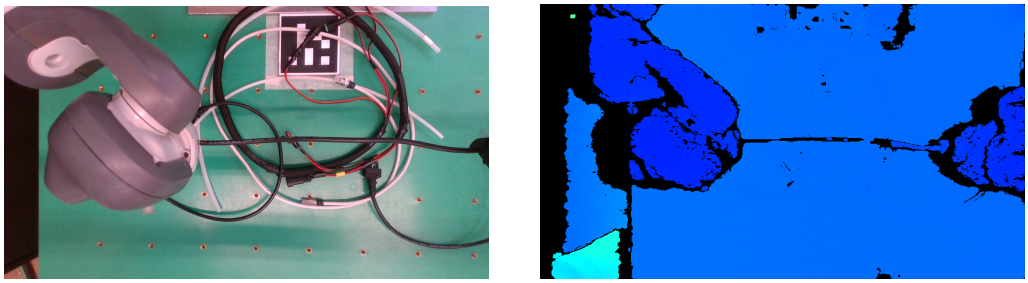


Figure 4.24: Tracked points (in red) plotted on the color frame.

4.5. Occlusion problem

While the dual-arm robot manipulates the DLO, it can occlude the end of the DLO with one of the two arms, as can be noticed in the Figure 4.5a, where the left arm does it. In addition, this thesis focuses the attention on tracking in an industrial scenario, so in this case can happen occlusion in all the parts of the DLO caused by different types of objects, e.g. an operator that touches the DLO during the manipulation (Figure 4.15a). The concept of occlusion can be defined as general as the loss of a considerable part of DLO pixels, and this can happen not only in case of occlusion by an object, but also in case of holes in the depth generated by shadows, a particular color of the DLO, like for the DLO2, DLO5, DLO6 (see Figure 3.6) that being very thin, can give problem in acquiring the depth information (Figures 4.25a and 4.25b).

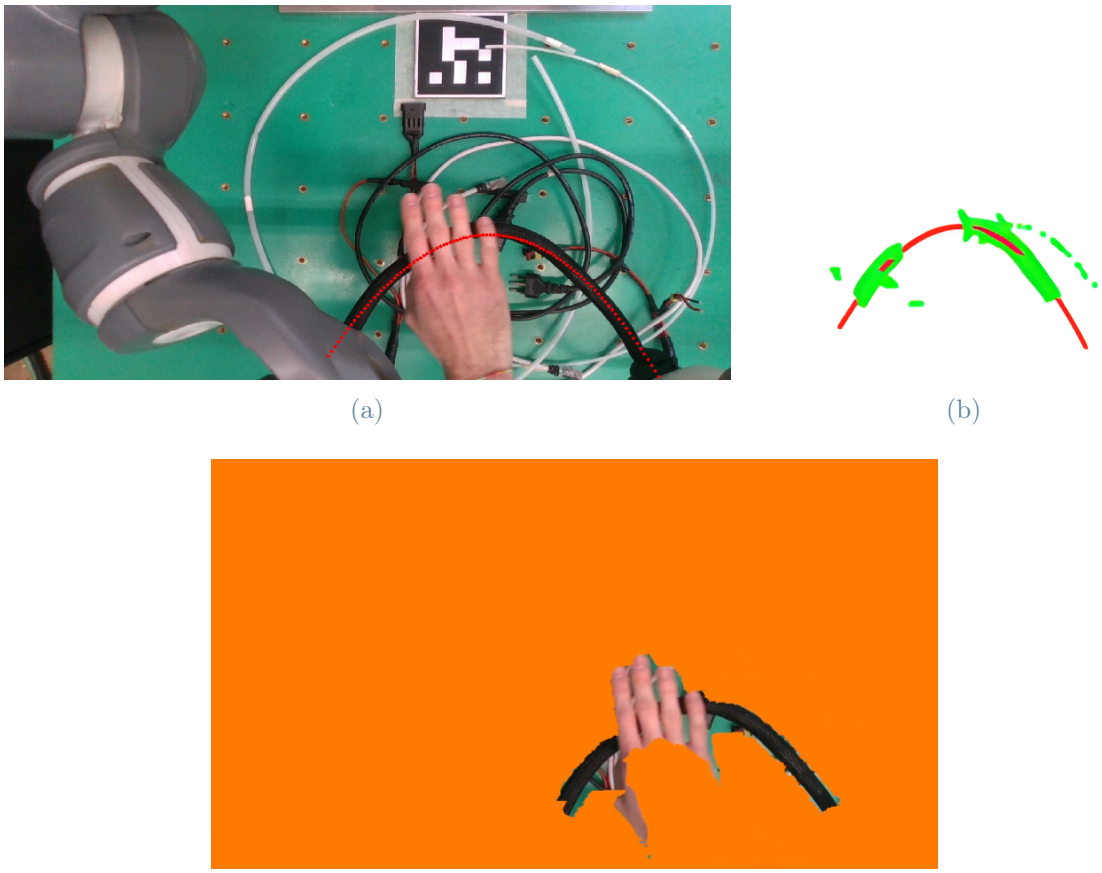


(a) (b)

(c)

Figure 4.25: (a) DLO2 color frame. (b) DLO2 depth frame, with holes. (c) Depth frame after the application of the $hole\_filling\_filter$ with fill from left logic.

In all these cases the application of the $hole\_filling\_filter$ is useless, due to the fact the holes are very large so we cannot take depth information from the neighbor pixels (Figure 4.25c).
But the proposed method described in the previous section is robust to all these types of occlusion. Since the DLO segmentation phase (Section 4.2) ensures complete isolation and extraction of the DLO pixels. As soon as the DLO pixels are reported in the 3D, the gripper points are also added to these points, ensuring more robustness in the tracked shape by the Lasso regression (Section 4.4.1). Figure 4.26, shows an example: the left arm occluded the left end of the DLO, but thanks to the acquisition of the grippers poses, the occluded part can be estimated. In addition, it shows an example of the tracking result in case of a hand occludes the DLO1 touching it. Note that the depth filter (Figure 4.26c) isolate also the hand since the hand touches the DLO, but the hand has different color with respect to the DLO, so the color mask (Section 4.2.2) will segment only the DLO ensuring the tracking of the shape.

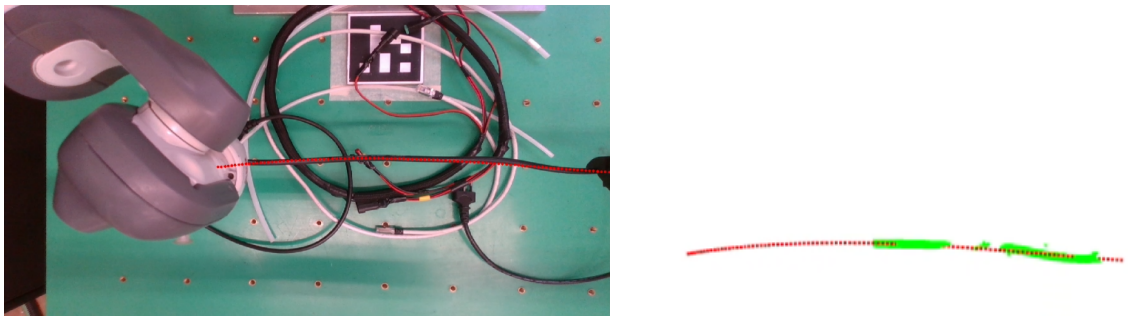


(a) (b)

(c)

Figure 4.26: (a) DLO1 color frame with a hand occlusion, in red the tracked points. (b) DLO1 point-cloud (in green) with a hand occlusion, in red the tracked points. (c) Depth filter output isolates the DLO and the hand.

Then in case of holes problem, like the one previously saw in Figure 4.25, the proposed method ensures a good geometric estimation of the DLO (Figure 4.27).



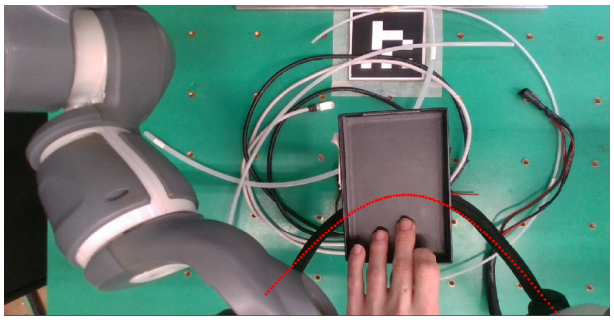
(a) (b)

Figure 4.27: (a) DLO2 color frame with tracked points (in red). (b) DLO2 point-cloud (in green) with tracked points (in red).

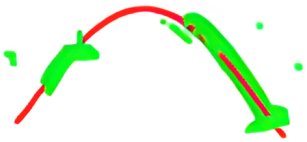
4.6. Occlusion Limitations

4.6.1. Sensitivity to the occluding object color

One of the limitations of the presented methodology, as in ones presented in the literature (Chapter 2), is the use of a color mask in order to segment the DLO. So this means that, in the case of an occluding object with different color with respect to the DLO, the tracking algorithm gives a good estimation of the DLO shape (Figure 4.26). Differently, if the occluding object has the same color as the manipulated DLO then the tracking algorithm generates a wrong DLO shape estimation (Figures 4.28d and 4.28e). Note that the methods proposed in the literature consider only occluding objects with different colors with respect to respect the tracked one. In the proposed methodology this limitation is mitigated, thanks to the application of the depth filter (Section 4.2.1) before the color mask. Indeed if the occlusion happens above the manipulated DLO of approximately 4 cm, the depth filter will cut off the occluding object (Figure 4.28c), ensuring good tracking results (Figures 4.28a and 4.28b).
Meanwhile, if the black object is on the DLO1 touching it, the depth filter isolate also the black object (Figure 4.28f), consequently the color mask and the contour extraction acquires also the black object pixels, so they are considered in the fitting causing a wrong estimation of the DLO's shape (Figures 4.28d and 4.28e).

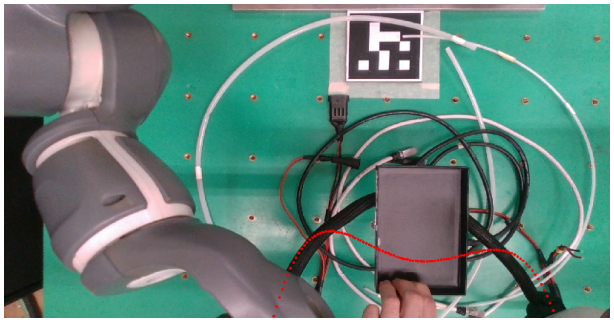


(a)



(b)

(c)



(d)

(e)

(f)

Figure 4.28: DLO1 tracking results in case of black occluding object. (a) DLO1 color frame with tracked points (in red) with black object 4 cm above the DLO. (b) DLO1 point-cloud (in green) with tracked points (in red), in case of black object 4 cm above the DLO. (c) Depth filter output in case of black object 4 cm above the DLO. (d) DLO1 color frame with tracked points (in red) in case of black object on the DLO touching it. (e) DLO1 point-cloud (in green) with tracked points (in red) in case of black object on the DLO touching it. (f) Depth filter output in case of black object n the DLO touching it.

We have this limit on the upper threshold of the depth filter because the DLOs have infinite degrees of freedom, so they can also deform upward during the manipulation. This deformation depends on the rigidity of the DLO (Table 3.2), but the proposed algorithm tracks the DLOs without the use of rigidity information. So choosing an upper threshold that is too strict can delete important parts of the DLO.
This thesis is developed to ensure tracking of a manipulated DLO in an industrial scenario. So thinking of an industry where the robot needs to manipulate the DLO repeating the same operation, is reasonable to assume that there is a sample video of the manipulation without any occlusion, in order to demonstrate the manipulation and the final desired shape. The mentioned problem can be completely solved using this sample video.
As previously explained, taking as an example the Figure 4.28, the main problem is that the Lasso regression considers also the points of the object causing the occlusion. The solution is to delete all the occluding object points before constructing the point-cloud. The idea is to run in parallel to the online acquired video from the camera, also the sample video. Applying on both videos all the pre-processing (Section 4.1) and segmentation (Section 4.2) steps.

Figure 4.29 shows respectively the color frame and the result of the contour extraction step, the last step before the construction of the point-cloud, of the sample video without occlusion (Figures 4.29a and 4.29b) and online video with occlusion (Figures 4.29c and 4.29d). Applying an AND between the two gray images, using the function *cv2.bitwise_and*, all the white pixels that are not present in both images are deleted (Figure 4.29e).

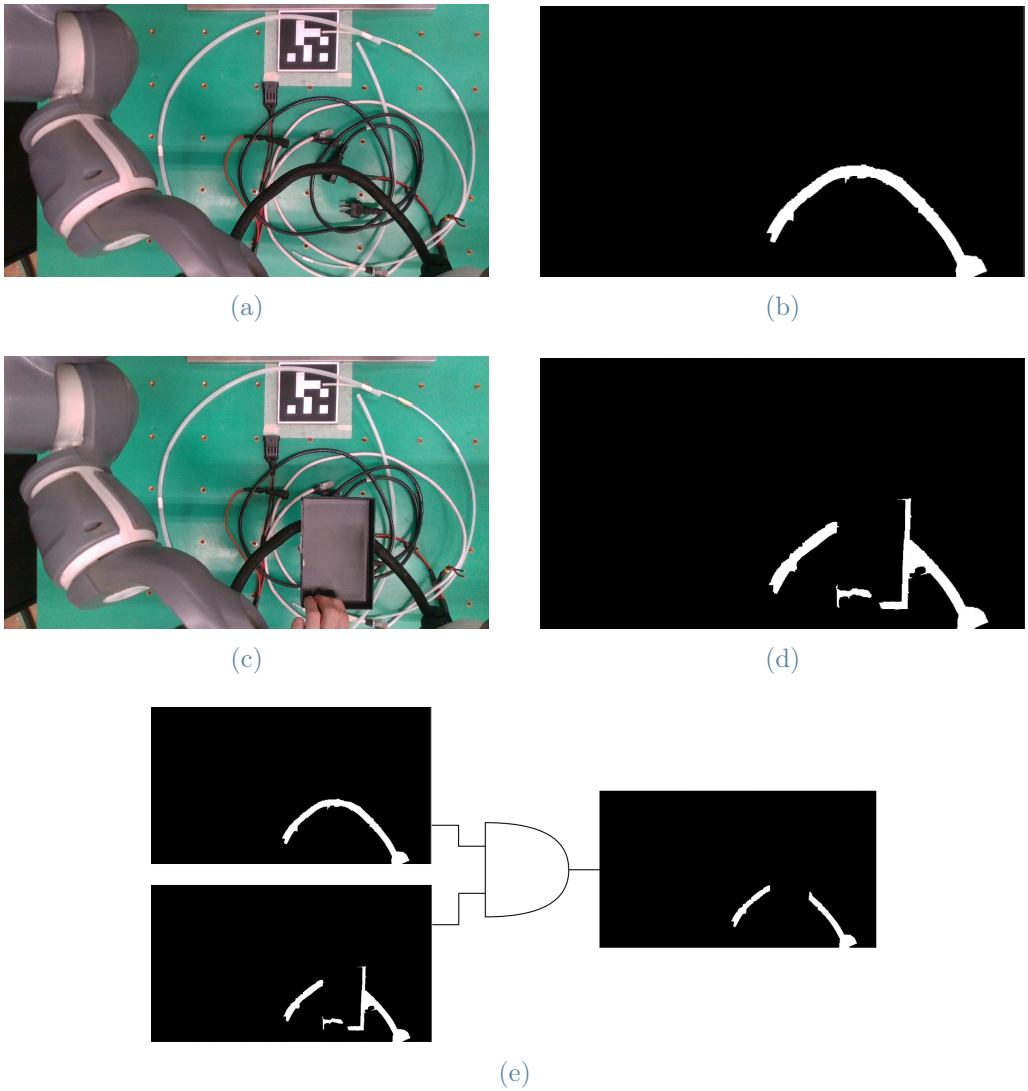
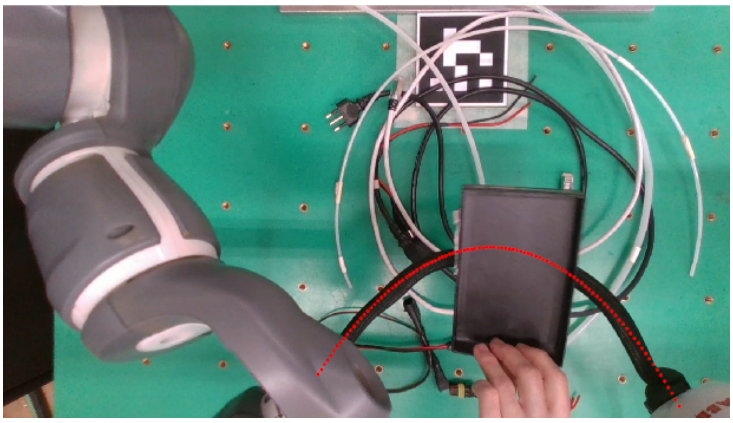
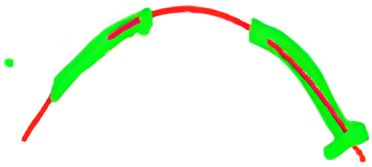


Figure 4.29: (a) Color frame of the sample video without the black occluding object. (b) Result of the contour extraction on the sample video without the black occluding object. (c) Color frame of the online video with a black occluding object. (d) Result of the contour extraction on the online video with a black occluding object. (e) AND between (b) and (d), filter out the object points.

Figure 4.30 shows the tracking result during a dynamic occlusion, once the object points are filtered out by the AND operation, the tracking methodology is able to estimate DLO1, differently from Figures 4.28d and 4.28e.



(a)



(b)

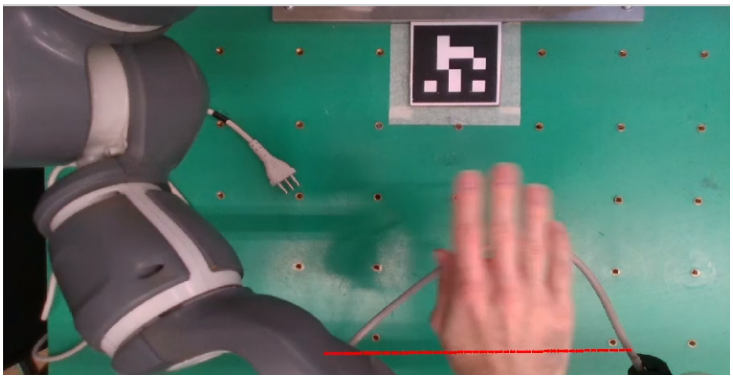
Figure 4.30: DLO1 tracking result in case of black occluded object on it. (a) DLO1 color frame with tracked points (in red). (b) DLO1 point-cloud (in green) with tracked points (in red).

Despite this method is based on the application of the DLO segmentation step two times, one for the online video and one for the sample video, there is no worsening of the computational time for each cycle of the entire algorithm (Figure 4.1) as it will be seen in Section 6.4.

4.6.2. Occluding object dimension

Another limitation is that the occluding object cannot be too big, or too close to the camera, because in this case, the free parts of the DLO are fewer. Those free portions may also not give points, due to holes or big shadows that create a different color that is out of the HSV threshold (Section 4.2.2). Hence the only points that are given to the Lasso regression are the gripper one, so it will approximate the shape with a polynomial

of order zero or one (Figure 4.31), based on the desired shape, and then the algorithm will stop. This limitation weighs more on the thin DLO, like DLO2, where there is just the holes issue which causes a lower number of points as we discussed at the beginning of this section.



(a)

(b)

Figure 4.31: DLO3 approximation with a straigh line due to lower number of points. (a) DLO3 color frame with tracked points (in red). (b) DLO3 point-cloud (in green) with tracked points (in red).

4.7. Sum-up

Figure 4.32 sums up the steps described in this Chapter, applying the proposed algorithm to track the DLO1 in a sinusoidal function shape. Note that as explained in Section 4.2.3 thanks to the contour extraction phase, the part of the DLO after the right gripper is filtered out, ensuring the tracking of the final shape.

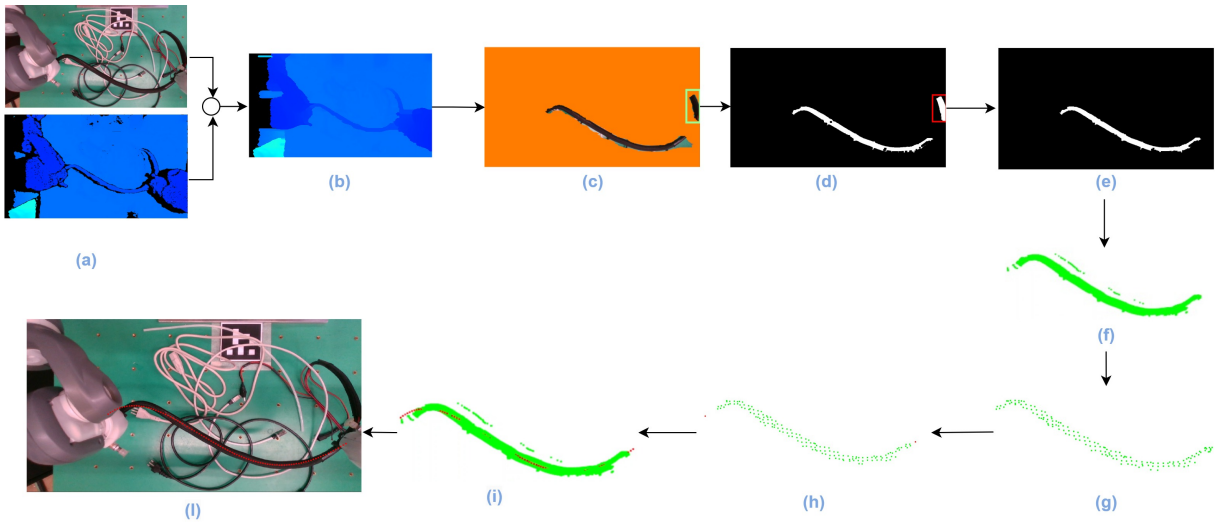


Figure 4.32: (a) Acquisition of the color and depth frame. (b) Application of the hole filling filter. (c) Depth filtering, in the green rectangle the DLO part after the right gripper. (d) Color mask, in the red rectangle the DLO part after the right gripper. (e) Contour extraction, which filters out the DLO part after the right gripper. (f) Point-cloud construction. (g) Point-cloud down-sampling and outliers removal. (h) Acquisition and addition of the grippers poses (red points) to the point-cloud. (i) 3D-fitting, in red the tracked shape. (l) DLO tracked shape (in red) on the Colour frame.

5 | Communication between the tracking algorithm and a dual-arm robot

While in Chapter 4 the attention was focused on the Computer vision algorithm, in this Chapter we will focus on the communication between the tracking algorithm and a dual-arm robot, which robot is programmed in RAPID.
As presented in Section 3.1, the computer receives visual information by the camera and is connected with USB, moreover the computer controls and exchange information with the robot using an Ethernet connection (Figure 3.1). In particular, the computer and the robot communicate via socket. For this reason, in this chapter will be first explained the main principle of socket communication. Then the program in RAPID will be introduced, detailing then the chosen program structure. Finally, the communication between the computer and the robot is described.

5.1. Socket

A socket is an endpoint that allows a two-way communication link between remote hosts or between local processes. In our case, the two remote hosts are:

- the robot, that is the server;
- the computer, that is the client.

The fact that is a two-way communication link means that the server and the client can both receive and sent packets, so information, on the network using the socket. The server socket is programmed in RAPID (further details in Section 5.2) which supports the stream type socket, meaning that is based on TCP (Transmission Control Protocol). This protocol has the main characteristic of being reliable, implying that in the event of lost packets in the network, this will be detected and the packet retransmitted.
The socket is identified by an IP address and a port number, which is opened by the

server and closed at the end by it. The communication between the server (the robot) and the computer (the client) is schematized in Figure 5.1.

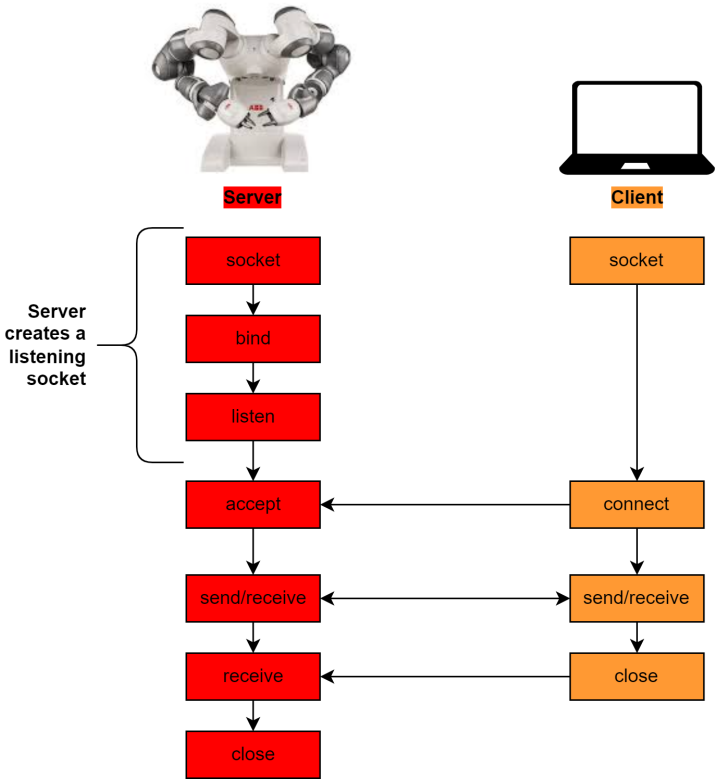


Figure 5.1: Server-client communication.

The server first creates the socket, binds the specified server IP address and port number to the socket and then listens, so waiting that a client requests the connection. Once the request for the connection by the client is accepted, the server and the client can exchange information.
In general, the computer vision algorithm sends an integer number that corresponds to applying a command to the robot, while the robot sends information about its gripper position. The server (the robot) is programmed in RAPID, while Python is used on the client to develop the Computer vision algorithm.

5.2. RAPID program structure

RAPID is a high-level programming language used to control ABB robots. A RAPID application is called a task, a task is composed of a set of modules. The module can contain:

- different data;

- a function, that returns a value;
- a procedure, that does not return any value, e.g used to specify motion instruction;
- a trap, used to deal with interrupt. When an interrupt occurs, the normal program execution is suspended, e.g. due to an error, and are executed the instruction in the trap routine.

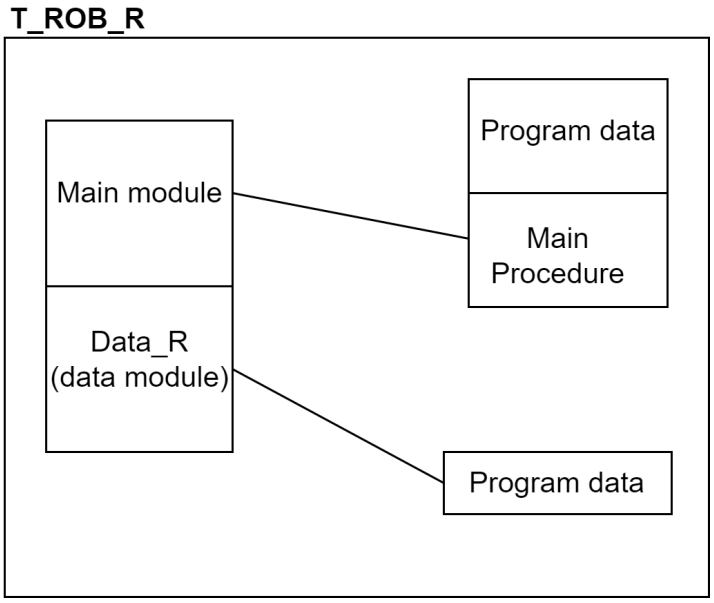


Figure 5.2: Right arm task.

Figure 5.2 shows an example of task. In fact, since YuMi is a dual-arm robot, can be defined a task for each arm. Each task contains:

- A main module, in which we defined the movement of the arm using a procedure and the data useful for it;
- A data module, in which we defined useful data, e.g. the points to which the arm needs to pass.

RAPID allows multitasking, which means running two or more tasks in parallel. For this reason, a task for each robot arm was created, in order to manipulate synchronously the DLO. Then to those 2 foreground tasks, a background task is added (Figure 5.3).

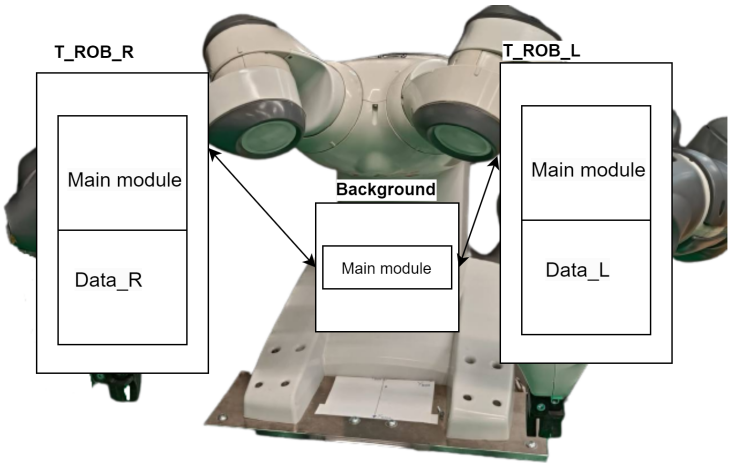


Figure 5.3: Chosen Multitasking RAPID configuration.

In the main modulus of the foreground tasks, the movement of the two arms is defined by procedures, using the *MoveJ* RAPID instruction, in case of movement not along a straight line (Sections 6.2, 6.3 and 6.5), otherwise the *MoveL* instruction is used (Section 6.1). These two functions in addition to the velocity of the TCP (Tool Center Point), in mm/s, and to the zone data, take as input the point to which the arm needs to pass and the considered TCP. These two data are saved in the data modules.
The considered TCP (tool_grip) coincides with the points where the fingertips grip the DLO (tool_grip). This point is calculated by translating downward of 130 mm tool0, that is the predefined TCP which is centered in the flange (Figure 5.4).

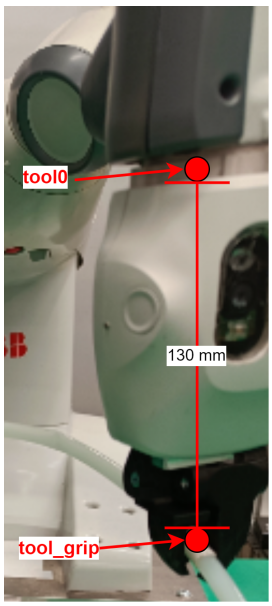


Figure 5.4: tool0 and tool_grip.

The background task is used to handle the robot movement and the socket communication while the two foreground tasks run in parallel. In particular, the background contains the procedures to create the listening socket, accept the connection of the client and then send the data (left column of Figure 5.1).

5.2.1. Client-server communication

The client socket is created and the server connects at beginning of the pre-processing step (Section 4.1). As we saw the camera need some step to be ready for the online tracking algorithm, so while the camera does these steps the server wait, until a command $record = 3$, is received.
While the robot manipulates the DLO, in the background task the socket continues to wait for a command from the client. Each time the client finishes the downsampling and the outlier removal of the point-cloud, it acquires the gripper position (Section 4.3.1), sending the $command = 0$. This command read by the background task corresponds to acquiring the right and left grippers positions. This information is acquired using the RAPID instruction $CRobT$, which takes as input the task of the arm and the reference TCP, and gives the position and orientation of the TCP in the space. But the positions given by this instruction are not the real one, since the robot is not calibrated with absolute accuracy, so the measures of the tool0 are not right, and this influences also the defined tool_grip. Note that even if the fingertips (Figure 3.2) are designed in order to center the DLO when are closed, the point where the DLO is gripped can vary due to:

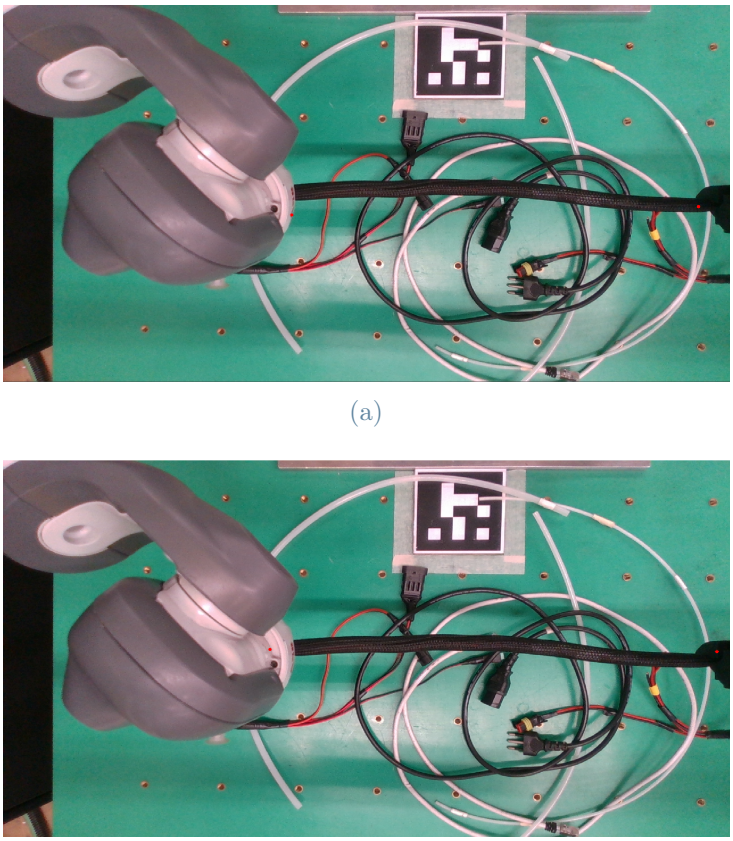
- the diameter of the DLO;
- the friction between the 3D printed fingertips and the DLO;
- wear of the fingertips.

For this reason, are added to the position given by $CRobT$ the offset shown in Table 5.1.

	X_{offset}	Y_{offset}	Z_{offset}
Right arm	-20 mm	-20 mm	+10 mm
Left arm	-10 mm	+25 mm	

Table 5.1: tool_gripper offset

The offsets are found experimentally with a trial-and-error method. For the X_{offset} and Y_{offset} checking the points with these offset in the image plane and plotting them on the acquired video from the camera (Figure 5.5). Then for the Z_{offset} was checked the sent Z position with respect to the one measured on the robot.



(a)

(b)

Figure 5.5: Gripper points with and without the offsets. (a) Received gripper points (in red) from the server without the offsets. (b) Received gripper points (in red) from the server with the offsets.

After that the offsets are added, they are firstly multiplied by 1000 in order to not lose important digit of the measurement, and then they are saved in two 1×3 vectors:

$$gripper_R = [X_r, Y_r, Z_r]$$

$$gripper_L = [X_l, Y_l, Z_l]$$

Then they are packed and sent to the client, which does the inverse action, unpacking them it and dividing for 1000, then are used just described in Section 4.3.1.

6 | Experimental analysis

A series of experimental tests were performed to validate the proposed tracking methodology in an industrial scenario, where the DLO is manipulated at a certain high from the working table. In particular, different kinds of DLOs are scattered on the working table. Note that several DLOs of the same color as the manipulated one are present in the scene to test the robustness.

This chapter analyzes the performed tests, which consist of the manipulation of the DLOs presented in Section 3.1.1 in different shapes, from the less complex one, as the linear shape, to the more complex one, as the sinusoidal shape. The considered DLOs differ one from the other in stiffness, color, length, and diameter. The proposed methodology does not have any information about the stiffness of the different DLOs. For this reason, for each DLO in the different shapes the depth threshold values must be specified (Section 4.2.1), which was estimated by the gripper position and the possible deformation of DLO. For each experiment and for each DLO, was analyzed the averaged mean squared error (AMSE) in the x-y and x-z planes:

$$AMSE_{x-y} = \frac{1}{n_{frames}} \sum_{j=1}^{n_{frames}-1} \sum_{i=1}^{P-1} (Y_i - Y\_true_i)^2$$

$$AMSE_{x-z} = \frac{1}{n_{frames}} \sum_{j=1}^{n_{frames}-1} \sum_{i=1}^{P-1} (Z_i - Z\_true_i)^2$$

with

$$Y\_true = c_{y4}X^4 + c_{y3}X^3 + c_{y2}X^2 + c_{y1}X + c_{y0}$$

$$Z\_true = c_{z4}X^4 + c_{z3}X^3 + c_{z2}X^2 + c_{z1}X + c_{z0}$$

where X, Y, and Z are the acquired points coordinates from point-cloud. P is the number of points, c_y and c_z are the results of the Lasso regression (Equation (4.2)). In addition, the proposed methodology is an online fitting algorithm, so for each manipulation also the mean computational time for each tracked frame was considered.

The specifications of the computer used are:

Component	Value
CPU	Intel i7-12650H
GPU	NVIDIA GeForce RTX 3060
RAM	16 GB
OS	Windows 11

Table 6.1, shows the HSV upper and lower threshold used for the color mask (Section 4.2.2) in each experiment.

	Lower Threshold			Upper Threshold		
	Hue	**Saturation**	**Value**	**Hue**	**Saturation**	**Value**
DLO1	0	0	0	179	255	53
DLO2	106	0	0	179	255	135
DLO3	138	14	150	179	255	255
DLO4	138	14	125	179	255	255
DLO5	138	0	125	179	255	255
DLO6	138	0	125	179	255	255
DLO7	138	14	125	179	255	255
DLO8	0	16	34	20	84	255

Table 6.1: HSV values for each DLO presented in Section 3.1.1

The user-defined parameters in the vision algorithm are:

Parameter	Value
kernel size	10×10
voxel size	0.01
number of neighbors	30
standard ratio	0.1
α	$1e^{-7}$
$max\_iteration$	$1e^{6}$
$fit\_intercept$	False

The configuration of the camera is shown in Figure 6.1

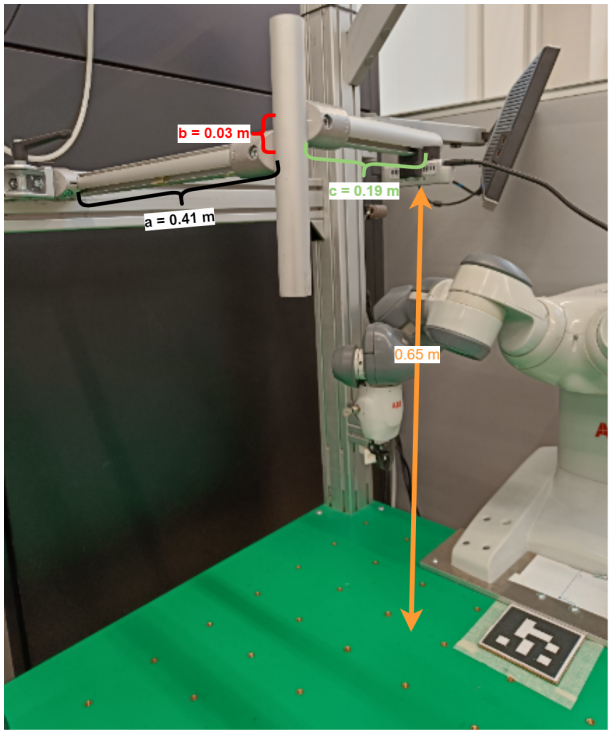


Figure 6.1: Camera configuration.

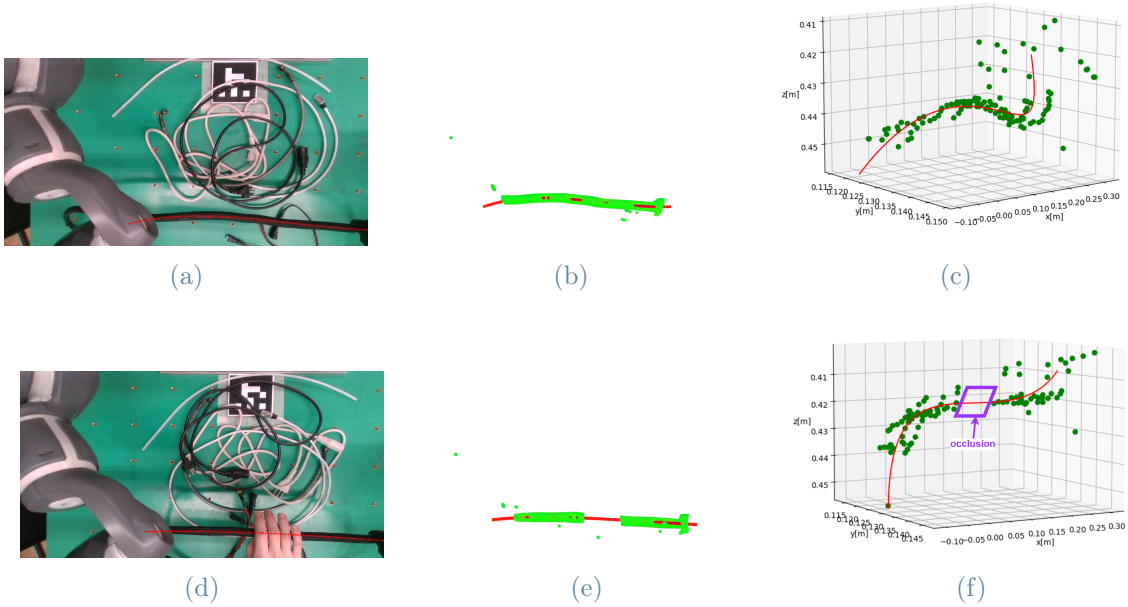
6.1. Test 1: Linear shape

The first tested shape is the easiest one, the linear shape. The robot moves linearly the DLO from an initial to a final point, with a velocity of 20 mms/s. This test was repeated with three kinds of DLO that differ one from the other in color, rigidity, and diameter. In this kind of shape, there is not a high deformation of the DLOs along the z-axis, for this reason, the same depth threshold value for the three DLOs is used (Table 6.2).

	Depth thresholds	
	upper limit in meters	**lower limit in meters**
DLO1	0.49	0.40
DLO7	0.49	0.40
DLO4	0.49	0.40

Table 6.2: Values of the depth filter thresholds, in case of linear shape.

Consider DLO1: it has a low rigidity but a higher diameter. Figures 6.2a to 6.2c shows the obtained result without occlusion. While Figures 6.2d to 6.2f show the results in case of hand occlusion. In both cases, the DLO is correctly tracked thanks also to the high diameter of the DLO, which gives a bigger number of points in the point-cloud. Note that, due to lower rigidity there are some changes of shape along the z-axis, highlighted in the 3D plot (Figures 6.2c and 6.2f).

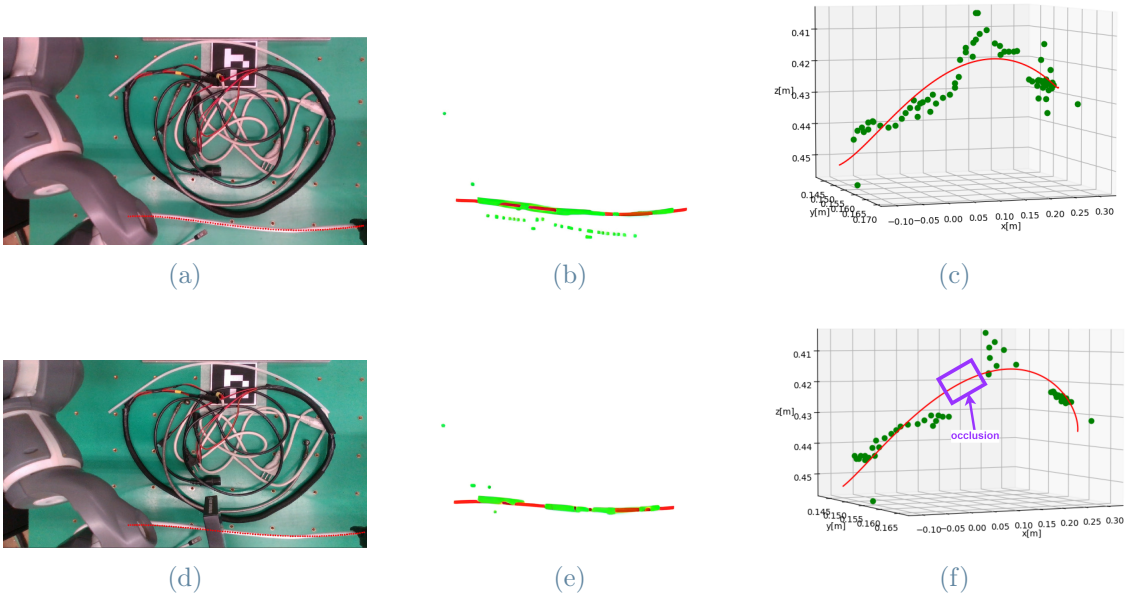


(a) (b) (c)

(d) (e) (f)

Figure 6.2: DLO1 linear manipulation without and with occlusion. (a) DLO1 color frame with tracked shape (in red). (b) DLO1 point-cloud (in green) with tracked shape (in red). (c) DLO1 3D-fitting plot. (d) DLO1 color frame with tracked shape (in red) in case of a hand occlusion. (e) DLO1 point-cloud (in green) with tracked shape (in red) in case of a hand occlusion. (f) DLO1 3D-fitting plot, with a hand occlusion highlighted in purple.

Consider now the tests involving DLO7. Figure 6.3 shows the result in case of occlusion with a black box and without it. Differently from DLO1, DLO7 is thinner, so there are fewer points in the extracted point-cloud, but it is still tracked.



(a) (b) (c)

(d) (e) (f)

Figure 6.3: DLO7 linear manipulation without and with occlusion. (a) DLO7 color frame with tracked shape (in red). (b) DLO7 point-cloud (in green) with tracked shape (in red). (c) DLO7 3D-fitting plot. (d) DLO7 color frame with tracked shape (in red) in case of occlusion. (e) DLO7 point-cloud (in green) with tracked shape (in red) in case of occlusion. (f) DLO7 3D-fitting plot, with occlusion highlighted in purple.

The last DLO considered for this test is the DLO4, the worst-case scenario for this shape. This DLO is a hose for compressed air, characterized by high rigidity and it is translucent. These two features influence the tracking of this DLO because it does not assume a completely linear shape due to high rigidity, as can be noticed by Figure 6.4c. In addition, forcing it to follow this shape creates a high number of depth holes, due to its particular material, which gives as result a lower number of DLO points in the point-cloud (green points in Figure 6.4b). Even with all these issues given by the particular type of DLO, it is correctly tracked without occlusion (Figures 6.4a to 6.4c) and with an occlusion made by a black box (Figures 6.4d to 6.4f). In this worst-case scenario, the acquisition of the grippers poses compensates for the low number of points, allowing tracking of the correct shape.

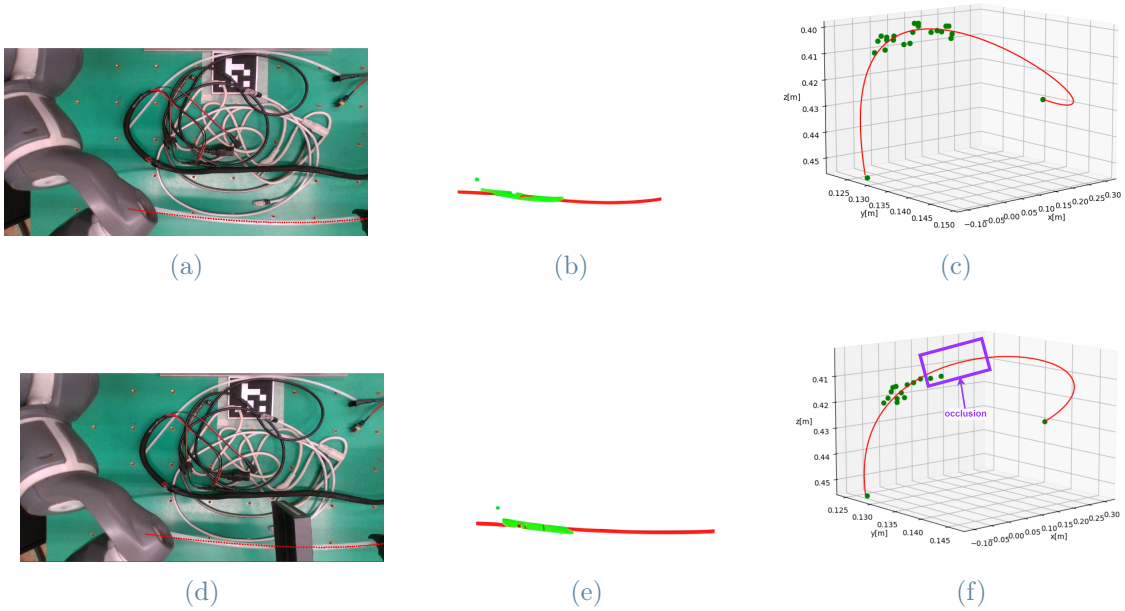


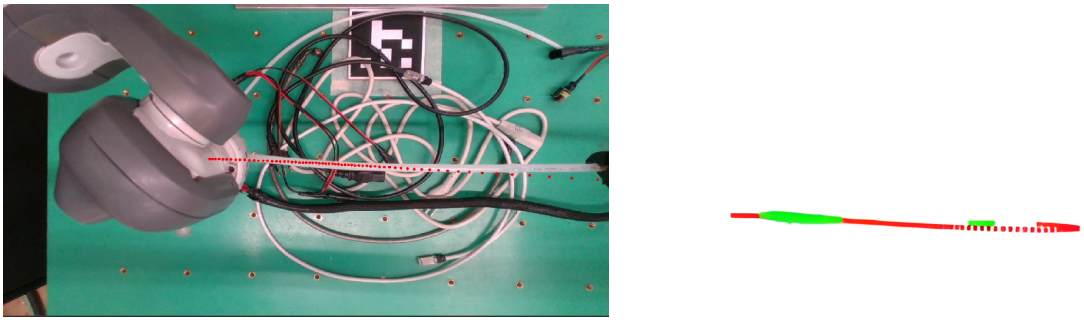
Figure 6.4: DLO4 linear manipulation without and with occlusion. (a) DLO4 color frame with tracked shape (in red). (b) DLO4 point-cloud (in green) with tracked shape (in red). (c) DLO4 3D-fitting plot. (d) DLO4 3D-fitting plot, with occlusion highlighted in purple. (e) DLO4 color frame with tracked shape (in red) in case of occlusion. (f) DLO4 point-cloud (in green) with tracked shape (in red) in case of occlusion.

Table 6.3 shows the mean computational time for each tracked frame. In general, it can be noticed that, in the case of occlusion, the time is smaller. This happens because the number of points to fit is lower due to occlusion, so the Lasso regression problem is faster. The association between the lower time and the number of points can be seen also comparing the time results for the 3 DLOs, from the DLO1 where there is a bigger number of points in the point-cloud to the DLO4 where the number of points in the point-cloud is lower.

Mean computational time for each tracked frame		
	without occlusion	**with occlusion**
DLO1	0.139 s	0.134 s
DLO7	0.093 s	0.091 s
DLO4	0.053 s	0.052 s

Table 6.3: Mean computational time for each tracked frame with occlusion and without it (linear shape).

Meanwhile, in Table 6.4 the averaged MSE tracked error is shown. In general, the error is of the same order of magnitude for the three DLOs, since the shape remains the same (straight line) during all the robotic manipulation. However, note that the DLO4 error in the x-z plane (highlighted in red in Table 6.4) is 10 times bigger with to respect the other DLO errors in the x-z plane. This error is generated because is not considered only the error in the last frame showed in Figures 6.4d to 6.4f where the shape is correctly tracked, but it is also considered the error in the previous frames that can be big due to loss of depth information and the occlusion which generate a lower number of points. Figure 6.5, showed one of the initial frames where there is a big error in the x-z plane.



(a) (b)

Figure 6.5: DLO4 point-cloud with big error in x-z plane during the linear shape

	$AMSE_{x-y}$		$AMSE_{x-z}$	
	without occlusions	**with occlusions**	**without occlusions**	**with occlusions**
DLO1	0.252 cm^2	0.429 cm^2	0.1 cm^2	0.104 cm^2
DLO7	0.0552 cm^2	0.0416 cm^2	0.141 cm^2	0.109 cm^2
DLO4	0.280 cm^2	0.14 cm^2	3.5 cm^2	2.3 cm^2

Table 6.4: Averaged MSE for each tracked frame (linear shape).

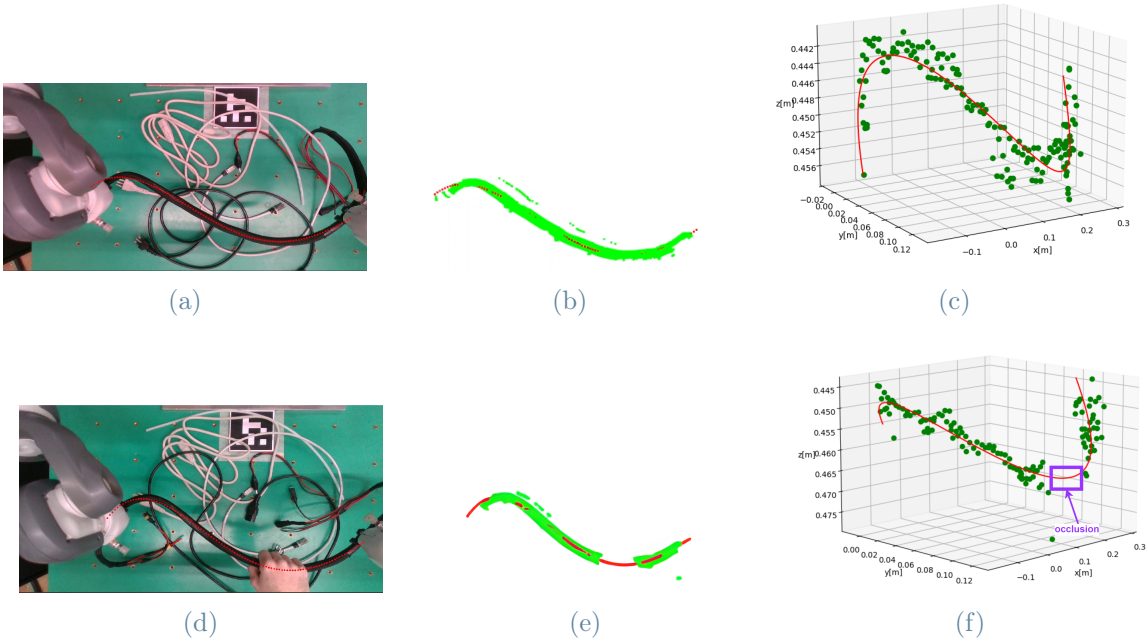
6.2. Test 2: 2D-Sinusoidal shape

Another tested shape is the sinusoidal shape. The robot moves, with a velocity of 80 mms/s, changing the joint configuration in order to reach the final sinusoidal shape. In this case, all the DLO of Table 3.2 are tested, except the one with too high rigidity due to the impossibility to have this shape with these kinds of DLO. Differently from the previous shape, is necessary to assign different depth thresholds to the DLO1, since it is the less rigid one and it tends to deform downward during the manipulation (Table 6.5).

Depth thresholds		
	upper limit in meters	**lower limit in meters**
DLO1	0.47	0.43
DLO2	0.49	0.42
DLO3	0.49	0.42
DLO7	0.49	0.42

Table 6.5: Values of the depth filter thresholds, in case of sinusoidal shape.

Firstly, the DLO1 was tested (Figure 6.6). Even if in the acquired frame there is also a cable part after the right gripper (Figure 6.6a), the tracking algorithm estimates correctly its shape. As explained in Section 4.2.3 the pixel of this part are not extracted to construct the point-cloud, thanks to the contour extraction phase (Figure 4.14). Figure 6.6d shows a test where the DLO is occluded by a hand that holds it and slides along it. It is interesting to notice that between Figure 6.24c and Figure 6.6f, we measure different variations along the z-axis. We rigorously define each variation along the z-axis as $\tilde{Z} = Z_{max} - Z_{min}$, where Z_{max} and Z_{min} are the maximum and minimum tracked Z values, respectively. Indeed, in Figure 6.24c, $\tilde{Z} = 1.6\ cm$, while in Figure 6.6f, $\tilde{Z} = 2\ cm$. This difference is caused, inevitably, by the operator's hand interacting with the DLO1.

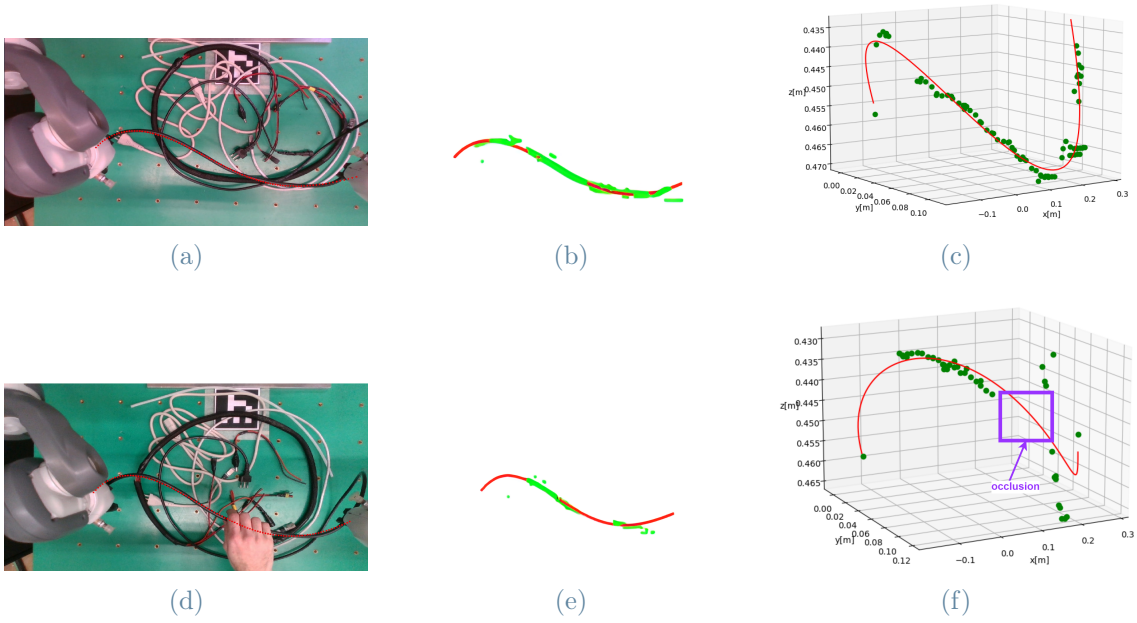


(a) (b) (c)

(d) (e) (f)

Figure 6.6: DLO1 sinusoidal manipulation without and with occlusion. (a) DLO1 color frame with tracked shape (in red). (b) DLO1 point-cloud (in green) with tracked shape (in red). (c) DLO1 3D-fitting plot. (d) DLO1 color frame with tracked shape (in red) in case of a hand occlusion. (e) DLO1 point-cloud (in green) with tracked shape (in red) in case of a hand occlusion. (f) DLO1 3D-fitting plot, with occlusion highlighted in purple.

Figure 6.7 shows the sinusoidal shape tested on the DLO2, that is thinner and more rigid with respect to DLO1. The influence of these two properties can be noticed especially in the case of occlusion. In fact, like in the previous experiment, a hand holds the DLO, which is pushed down while the hands slide on it. Unlike the DLO1, there are a lower number of points in the point-cloud, due to the fact it is thinner. Note that $\tilde{Z} = 3\ cm$ in the case without human occlusion (Figure 6.7f), while the interaction of the human hand generates a variation along the Z-axis $\tilde{Z} = 2.3\ cm$ (Figure 6.7e). However, this does not influence the tracking of the DLO with and without occlusion.



(a) (b) (c)

(d) (e) (f)

Figure 6.7: DLO2 sinusoidal manipulation without and with occlusion. (a) DLO2 color frame with tracked shape (in red). (b) DLO2 point-cloud (in green) with tracked shape (in red). (c) DLO2 3D-fitting plot. (d) DLO2 color frame with tracked shape (in red) in case of a hand occlusion. (e) DLO2 point-cloud (in green) with tracked shape (in red) in case of a hand occlusion. (f) DLO2 3D-fitting plot, with occlusion highlighted in purple.

Figure 6.8 shows the sinusoidal shape tracked on the DLO3, that is a power cord as the DLO2. Figures 6.8a and 6.8b, highlights the tracked shape in the 2D plane: it is possible to note that the shape is very similar to the DLO2. However, in Figure 6.8c it can be noticed that $\tilde{Z} = 1\ cm$, so $\frac{1}{3}$ respect to the one measured for DLO2. This is due to the different flexibility of the two DLOs. While the interaction with the black occluding object generated a $\tilde{Z} = 1.7\ cm$. Note that the proposed tracking algorithm can capture this difference in the tracking without any information on these features.

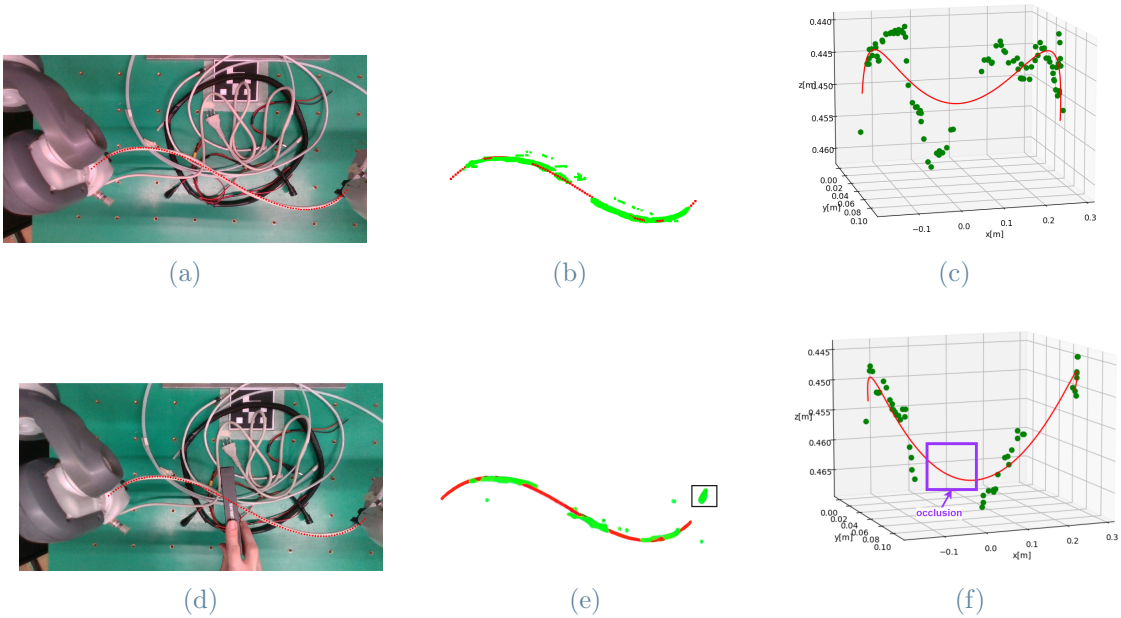
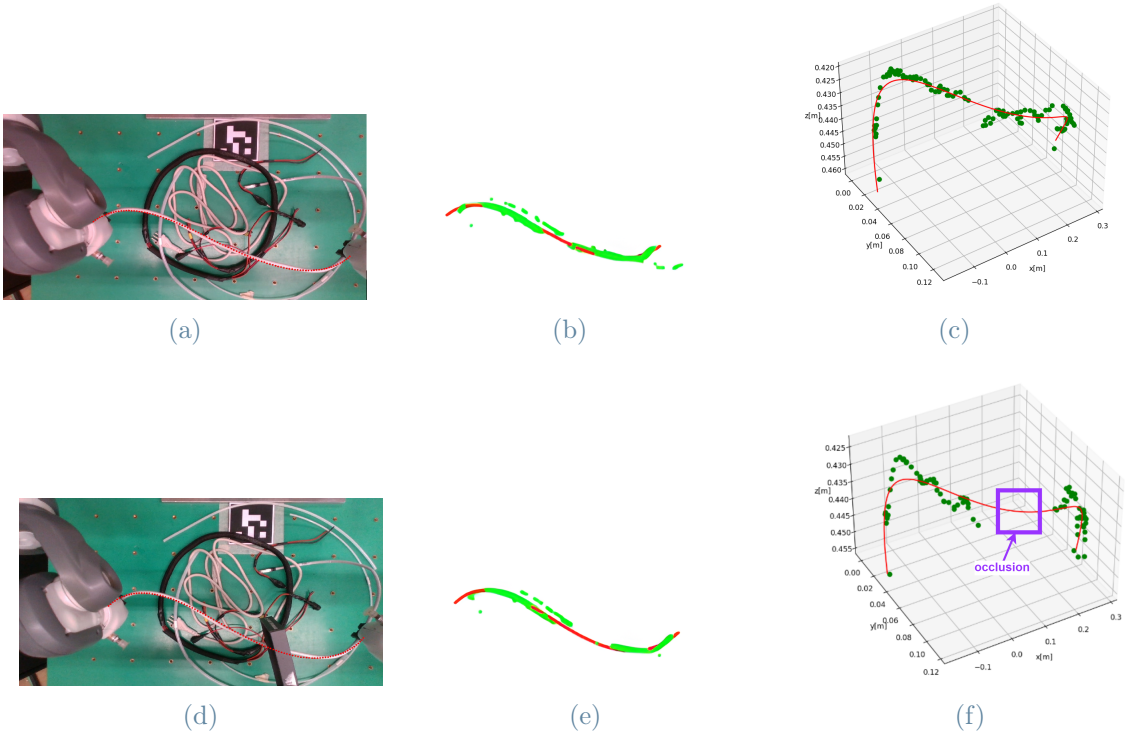


Figure 6.8: DLO3 sinusoidal manipulation without and with occlusion. (a) DLO3 color frame with tracked shape (in red). (b) DLO3 point-cloud (in green) with tracked shape (in red). (c) DLO3 3D-fitting plot. (d) DLO3 color frame with tracked shape (in red) in case of occlusion. (e) DLO3 point-cloud (in green) with tracked shape (in red) in case of occlusion, the right connector is highlighted in the black rectangle. (f) DLO3 3D-fitting plot, with occlusion, highlighted in purple.

From Figure 6.8d it can be noticed that in the color frame, the right connector of the DLO appears, and this part is also reported in the point-cloud (Figure 6.8e in the black rectangle). This happens because, differently from DLO1 where the contour extraction deletes the part over the left gripper (Figure 4.14), in this case, the maximum perimeter is lower, thus the connector is bigger of $\frac{1}{5}$ of the maximum perimeter, hence it is extracted by the contour extraction step (Section 4.2.3). Nevertheless, the connector is deleted by the downsampling and the outlier removal applied on the point-cloud (Figure 6.9), for this reason, the connector points are not considered in the fitting, ensuring in this way a good tracking result.

Figure 6.9: Output of downsampling and outliers removal on the DLO3 point-cloud in sinusoidal shape.

The last DLO tested is the DLO7 (Figure 6.10). Also if it is thinner with respect to the previous DLOs, it generates enough points to be correctly tracked. Differently from the previous DLOs, it has higher rigidity, which causes a slightly upward deformation. As the DLO2, it generates a variation along the z-axis $\tilde{Z} = 3\ cm$ in the case without occlusion (Figure 6.10c). Although, the DLO2 (Figure 6.7c) has $\tilde{Z} = 3\ cm$, due to a $Z_{max} = 4.7\ cm$, caused by the middle part of the DLO which is deformed downward due to its lower rigidity. On the other hand, the DLO3 has this $\tilde{Z}$, due to its lower $Z_{min} = 4.3cm$, caused by its slightly upward deformation.



(a) (b) (c)

(d) (e) (f)

Figure 6.10: DLO7 sinusoidal manipulation without and with occlusion. (a) DLO7 color frame with tracked shape (in red). (b) DLO7 point-cloud (in green) with tracked shape (in red). (c) DLO7 3D-fitting plot. (d) DLO7 color frame with tracked shape (in red) in case of occlusion. (e) DLO7 point-cloud (in green) with tracked shape (in red) in case of occlusion. (f) DLO7 3D-fitting plot, with occlusion highlighted in purple.

As we just noticed in the linear shape, if the number of acquired points is lower, then the entire algorithm in tracking the acquired frame is faster. Hence in (Table 6.6), the occlusion case has a lower time respect to the case without it. Moreover note that for the DLO1 the time with respect to the linear shape is decreased by 35 ms for the case without occlusion, and is decreased by 31 ms for the case with occlusion.

Mean computational time for each tracked frame		
	without occlusion	**with occlusion**
DLO1	0.104 s	0.103 s
DLO2	0.080 s	0.079 s
DLO3	0.099 s	0.096 s
DLO7	0.093 s	0.091 s

Table 6.6: Mean computational time for each tracked frame with occlusion and without it (sinusoidal shape).

Considering then the AMSE (Table 6.7), can be noticed in general for all the DLO tested, except for the DLO1, a bigger value of the error in the x-z plane respect the x-y plane. This is caused firstly by the $\tilde{Z}$ which is bigger respect the one in the linear shape. Secondly, this is due to the lower number of points acquired. The bigger error is in fact the one generated by the DLO3 (highlighted in red in Table 6.7) which is the one with a lower number of points in the extracted point-cloud (Figure 6.8b). In addition, we can remember that the acquired grippers poses, are not precise, especially for the z position, due to the robot not being calibrated with absolute accuracy. Also, this generates the increase of the $AMSE_{x-z}$, since the tracked shape tries to fit a point that is not the real one. Although these errors, the tracking results for the sinusoidal shape are satisfactory.

	$AMSE_{x-y}$		$AMSE_{x-z}$	
	without occlusions	**with occlusions**	**without occlusions**	**with occlusions**
DLO1	0.389 cm^2	0.427 cm^2	0.054 cm^2	0.077 cm^2
DLO2	0.096 cm^2	0.6 cm^2	2.3 cm^2	1.47 cm^2
DLO3	0.150 cm^2	0.133 cm^2	3.16 cm^2	1.58 cm^2
DLO7	0.152 cm^2	0.105 cm^2	1.38 cm^2	1.96 cm^2

Table 6.7: Averaged MSE for each tracked frame (sinusoidal shape).

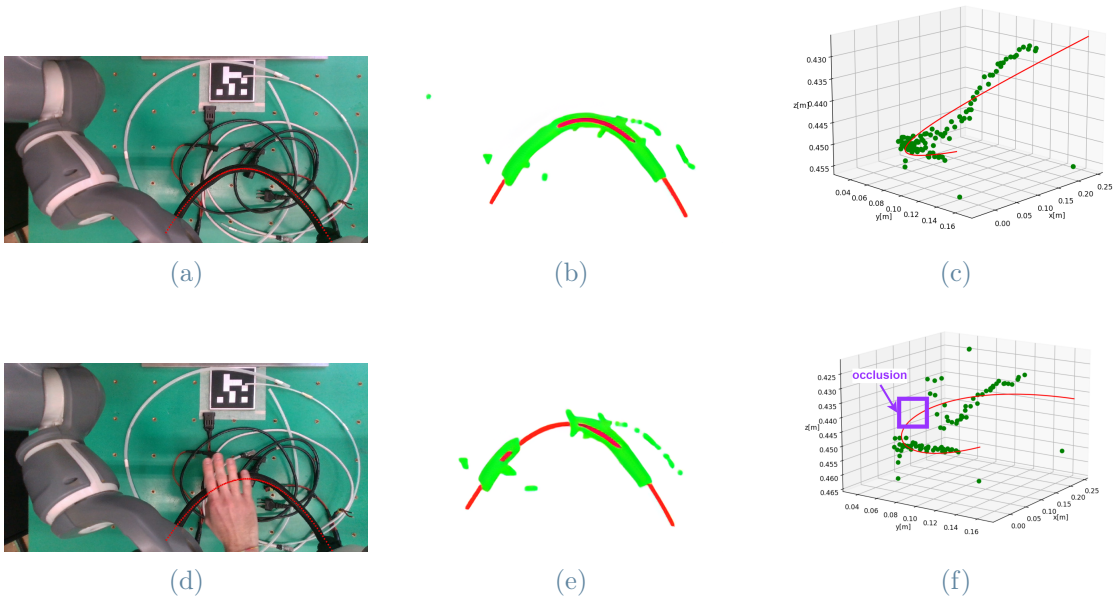
6.3. Test 3: Quadratic function shape

As for the sinusoidal shape, the robot moves with a velocity of 20 mm/s while changing its joint configuration in order to reach the final shape. But differently from the last two presented shapes, this can be tested for all the DLOs of Table 3.2, from the less rigid to the more rigid one. However, since the DLOs are characterized by different stiffness, it is necessary to define a depth threshold that considers for each DLO a possible deformation downward or upward (Table 6.8). In fact, the more rigid DLOs have a lower limit less strict (since during manipulation they can deform upwards), while the less rigid DLOs have an upper limit less strict (because is more probable that they deform downward during the manipulation).

	Depth thresholds	
	upper limit in meters	**lower limit in meters**
DLO1	0.49	0.42
DLO2	0.49	0.42
DLO3	0.49	0.4
DLO4	0.45	0.38
DLO5	0.45	0.38
DLO6	0.45	0.38
DLO7	0.49	0.38
DLO8	0.43	0.35

Table 6.8: Values of the depth filter thresholds, in case of quadratic function shape.

The first tested DLO is the DLO1. Figure 6.11 shows the result, due to the higher diameter with this DLO there is not any problem linked to the number of acquired points. But as it can be noticed due to the lower rigidity of this DLO, it will deform downward with $\tilde{Z} = 2.5\ cm$ (Figure 6.11c). While Figures 6.11d to 6.11f shows that the tracking result is robust also to a hand occlusion.

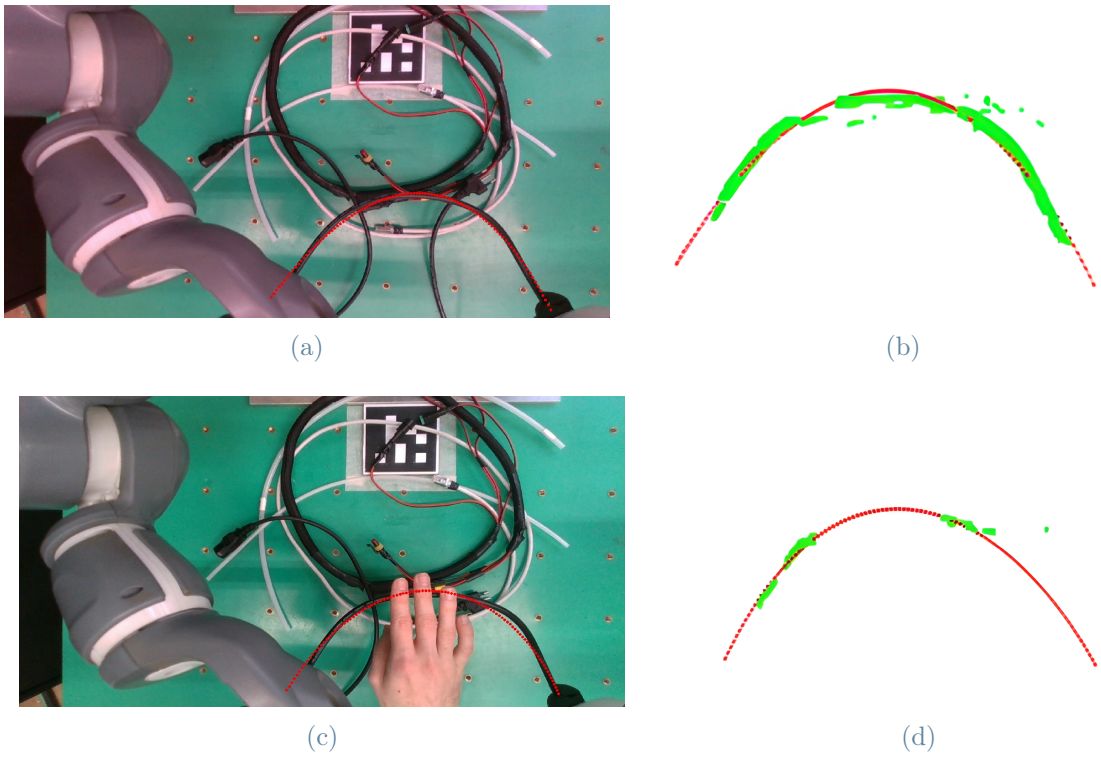


(a) (b) (c)

(d) (e) (f)

Figure 6.11: DLO1 quadratic function shape manipulation without and with occlusion. (a) DLO1 color frame with tracked shape (in red). (b) DLO1 point-cloud (in green) with tracked shape (in red). (c) DLO1 3D-fitting plot. (d) DLO1 color frame with tracked shape (in red) in case of hand occlusion. (e) DLO1 point-cloud (in green) with tracked shape (in red) in case of hand occlusion. (f) DLO1 3D-fitting plot with hand occlusion, highlighted in purple.

Figure 6.12 shows the results of the tested shape applied on DLO2. Note that in Figure 6.12a, in the background, other sections of the DLO that are not between the gripper are present. Thanks to the depth filter, these parts are filtered out and are not reported in the point-cloud (Figure 6.12b). Therefore, both in the image plane and in the point-cloud satisfactory results are obtained. Then in the case of occlusion, the number of points became lower (Figure 6.12d) due to the hand that occludes and pushes the DLO, but still, the final shape is well approximated.

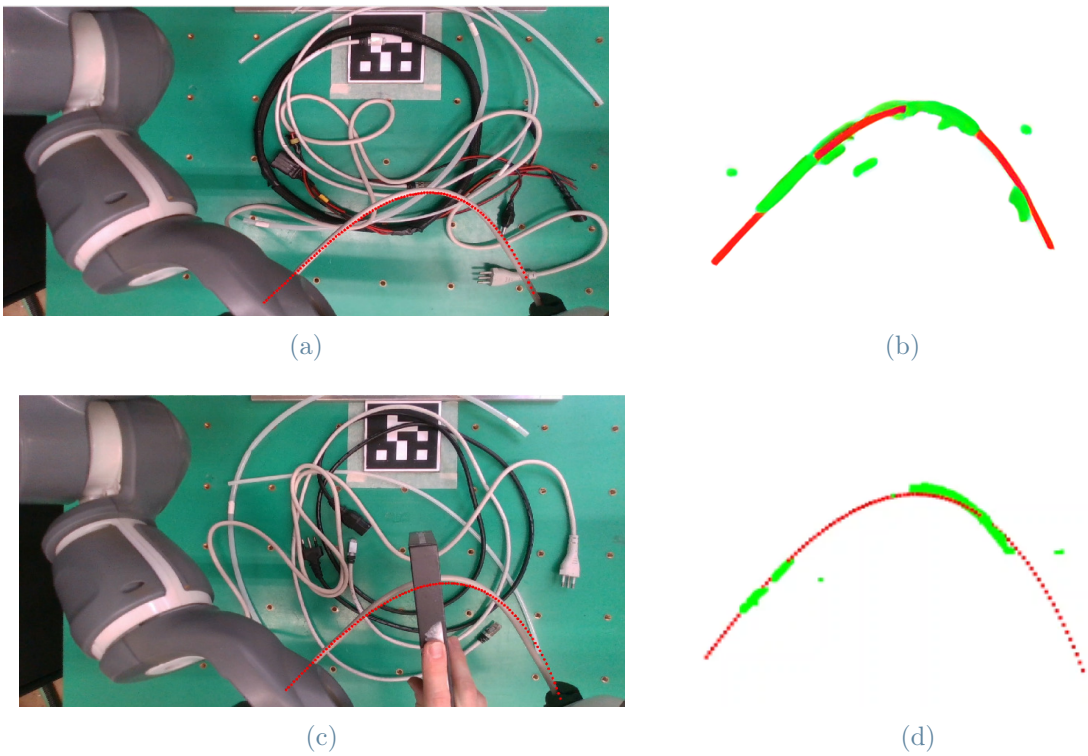


(a) (b)

(c) (d)

Figure 6.12: DLO2 quadratic function shape manipulation without and with occlusion. (a) DLO2 color frame with tracked shape (in red). (b) DLO2 point-cloud (in green) with tracked shape (in red). (c) DLO2 color frame with tracked shape (in red) in case of hand occlusion. (d) DLO2 point-cloud (in green) with tracked shape (in red) in case of hand occlusion.

The DLO3, due to its different rigidity with respect to the two previous DLOs, has instead a final shape slightly different from the others obtained previously (Figure 6.13). However, this shape is correctly tracked by the proposed method, both in the case of occlusions and without it.



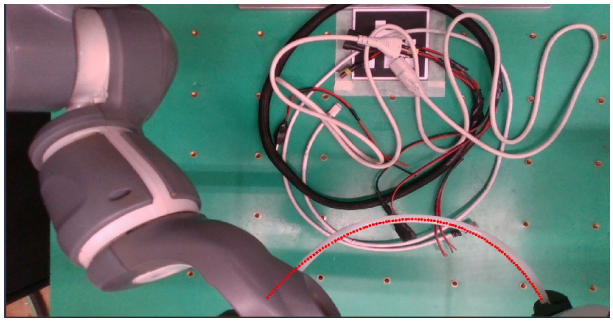
(a) (b)

(c) (d)

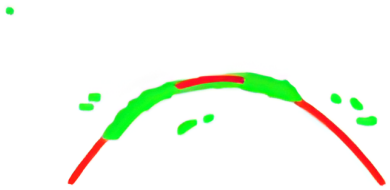
Figure 6.13: DLO3 quadratic function shape manipulation without and with occlusion. (a) DLO3 color frame with tracked shape (in red). (b) DLO3 point-cloud (in green) with tracked shape (in red). (c) DLO3 color frame with tracked shape (in red) in case of occlusion. (d) DLO3 point-cloud (in green) with tracked shape (in red) in case of occlusion.

The next DLO analyzed is the DLO4, Figure 6.14 shows the results. Differently from the linear case (Figure 6.4), the number of points grows up in case without occlusion, but the right part is tracked without extracted points (Figure 6.14b). This is caused by the particular translucent material, this implies that, in addition to the holes in the depth map, the infrared rays (generated by the infrared projector of the camera (Figure 3.3)), can pass through the DLO giving as depth value the one linked to the worktable as shown in Figure 6.14c. The part with holes (that have depth value = 0) and the part obtained due to the infrared projector, are filtered out by the depth filter (Section 4.2.1) that uses the threshold values of Table 6.8. It follows that, unfortunately, the small part between the holes and the result of the infrared projector is not extracted and reported in the point-cloud, because it is deleted by the contour extraction phase (Section 4.2.3) due to its smaller perimeter with respect to the other part of the DLO. However, despite of the mentioned issues the shape is correctly tracked.
The number of points decreases in case of occlusion (Figures 6.14d and 6.14e): this

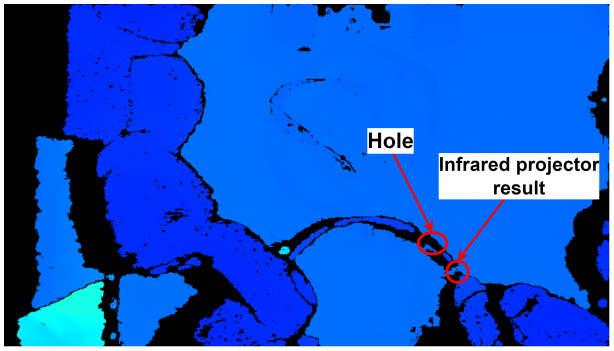
generates a slight shift to the left of the fitting points near the right gripper. Anyway also in case of occlusion, the final shape is correctly tracked.



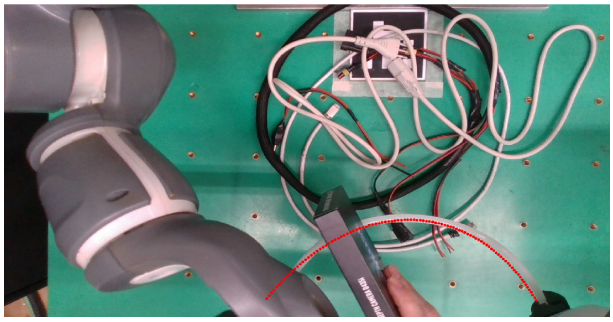
(a)



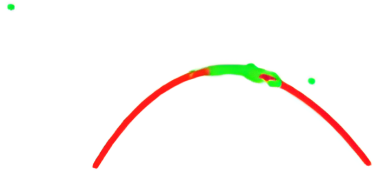
(b)



(c)



(d)



(e)

Figure 6.14: DLO4 quadratic function shape manipulation without and with occlusion. (a) DLO4 color frame with tracked shape (in red). (b) DLO4 point-cloud (in green) with tracked shape (in red). (c) DLO4 depth map with hole and infrared projector result circled in red. (d) DLO4 color frame with tracked shape (in red) in case of occlusion. (e) DLO4 point-cloud (in green) with tracked shape (in red) in case of occlusion.

Considering now the DLO5 (Figure 6.16). It is a hose for compressed air, as the previous DLO, but with a diameter lower than 0.2 cm. As in the previous case, the right part of the DLO is not reported in the point-cloud (Figure 6.16b) due to holes and the infrared result (Figure 6.15).

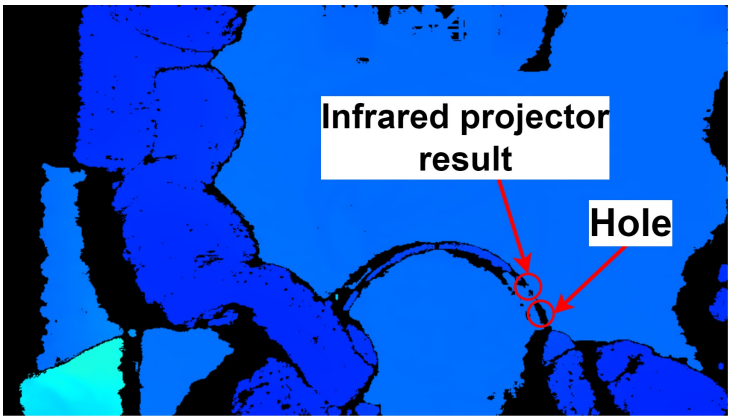
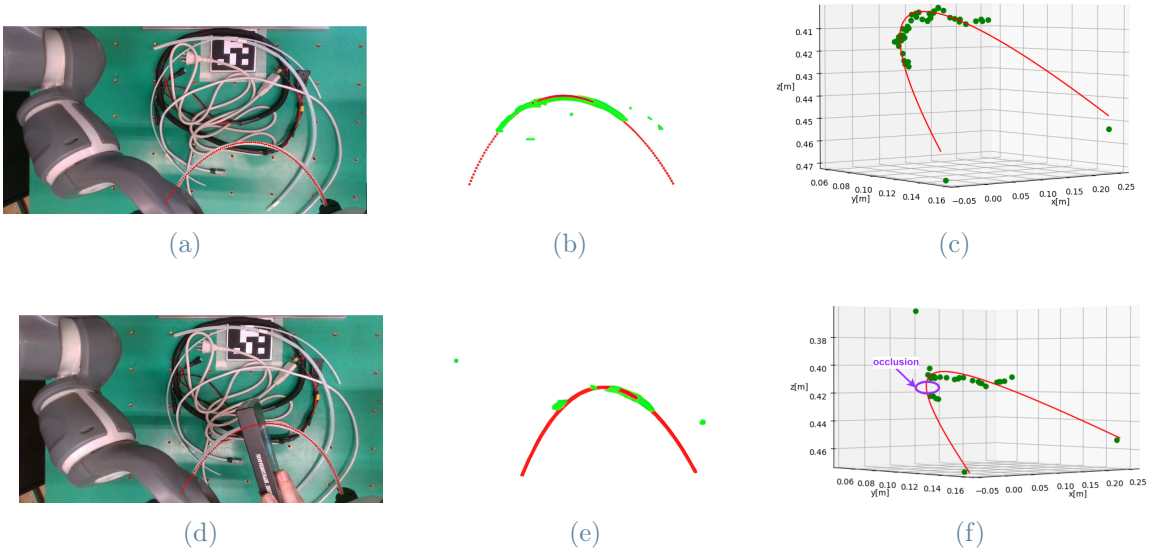


Figure 6.15: DLO5 depth map with hole and infrared projector result highlighted in red.

From Figures 6.16c and 6.16f, it is interesting to notice that $\tilde{Z} = 6\ cm$, that happens because the robot forces the DLO to assume a shape parallel to the working table, but due to the stiffness it will deform upward. The proposed tracking algorithm is able to track correctly both cases without occlusion and with them (Figures 6.16a and 6.16d), highlighting also the deformation upward without requiring any information about the DLO rigidity.

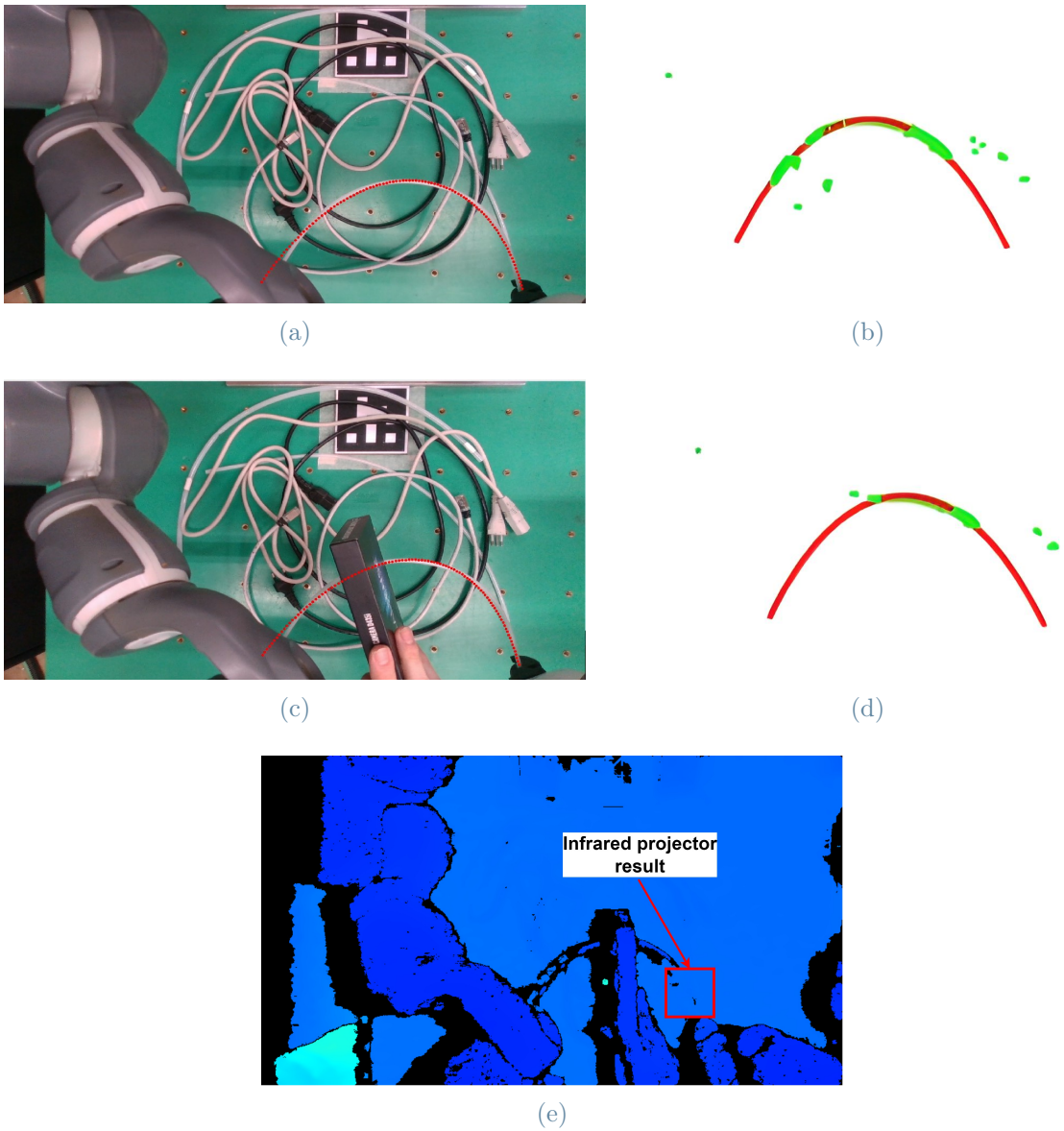


(a) (b) (c)

(d) (e) (f)

Figure 6.16: DLO5 quadratic function shape manipulation without and with occlusion. (a) DLO5 color frame with tracked shape (in red). (b) DLO5 point-cloud (in green) with tracked shape (in red). (c) DLO5 3D-fitting plot. (d) DLO5 color frame with tracked shape (in red) in case of occlusion. (e) DLO5 point-cloud (in green) with tracked shape (in red) in case of occlusion. (f) DLO5 3D-fitting plot with occlusion, highlighted in purple.

The last DLO of the family of the hose for compressed air is the DLO6, which has a diameter that is half of the DLO4. The infrared projector gives to all the right parts of the DLO the depth value of the working table due to small diameter of the DLO (Figure 6.17e). But also in the case of lower diameter the tracking algorithm is able to track the DLO in case of occlusion and without it (Figure 6.17).



(a) (b) (c) (d) (e)

Figure 6.17: DLO6 quadratic function shape manipulation without and with occlusion. (a) DLO6 color frame with tracked shape (in red). (b) DLO6 point-cloud (in green) with tracked shape (in red). (c) DLO6 color frame with tracked shape (in red) in case of occlusion. (d) DLO6 point-cloud (in green) with tracked shape (in red) in case of occlusion. (e) DLO6 depth map in case of occlusion, with infrared projector result in red square.

The other tested DLO is the DLO7 (Figure 6.18). It has a rigidity that can be estimated between the DLO3 and the hoses for compressed air, in fact, as can be noticed by Figures 6.18c and 6.18f, while it reaches the final shape, it will deform upwards of more than 5 cm due to its rigidity. But differently from the compressed air hoses, it gives more

points, that ensure the correct tracking of this DLO.

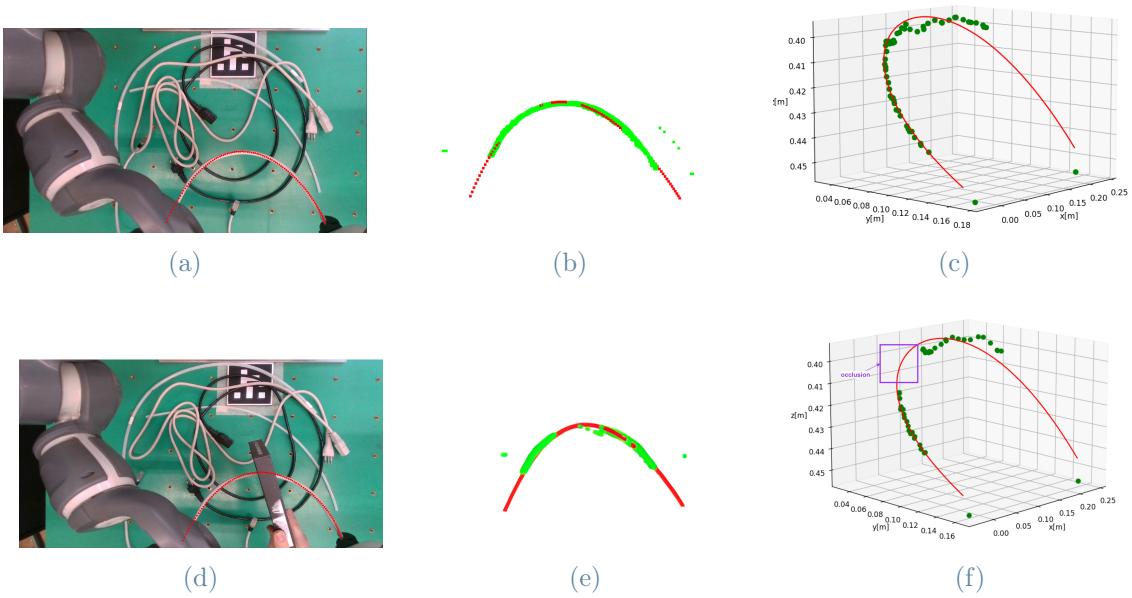
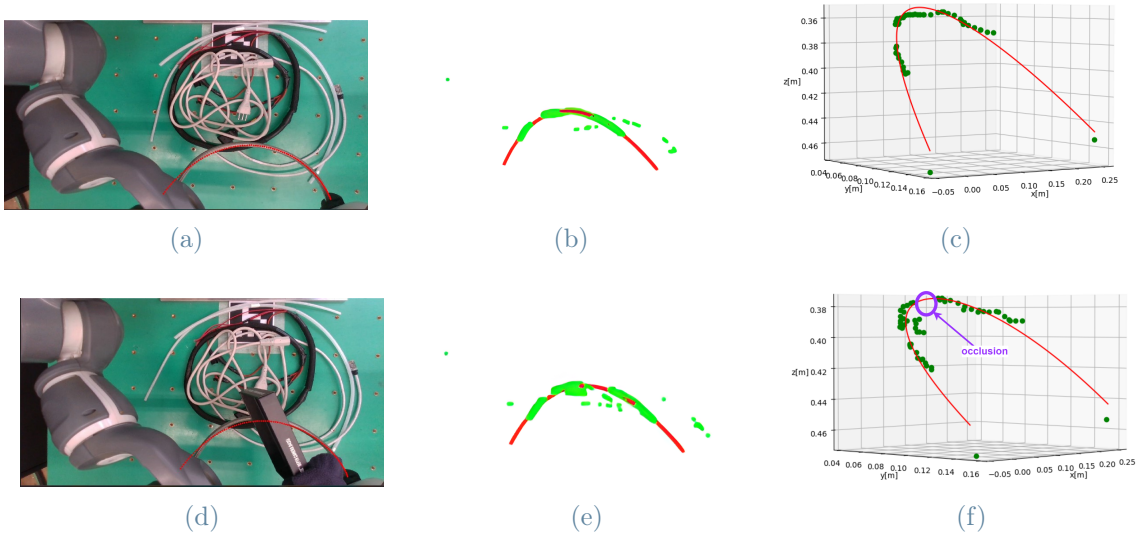


Figure 6.18: DLO7 quadratic function shape manipulation without and with occlusion. (a) DLO7 color frame with tracked shape (in red). (b) DLO7 point-cloud (in green) with tracked shape (in red). (c) DLO7 3D-fitting plot. (d) DLO7 color frame with tracked shape (in red) in case of occlusion. (e) DLO7 point-cloud (in green) with tracked shape (in red) in case of occlusion. (f) DLO7 3D-fitting plot with occlusion, highlighted in purple.

The last DLO analyzed is the DLO8 (Section 3.1.1). DLO8 is a hose of a motorbike braking system, characterized by the highest rigidity in the tested DLOs, and by a reflective material that makes difficult to acquire its contour by color mask.
Figure 6.19 shows the result with and without occlusion. Due to its high rigidity, forcing it to reach a final shape parallel to the working table generates a deformation upward of more than 10 cm (Figures 6.19c and 6.19f), more of 4 cm than the DLO5 (Figure 6.16). Considering the tests where the DLO was occluded, it must be pointed out that the human hand has to be covered by a glove (Figure 6.19d), otherwise it could interfere with the HSV values of the DLO in the image. The final tracked shape is slightly more rounded in this case respecting the case without occlusion, but anyway, the DLO shape is correctly estimated.



(a) (b) (c)

(d) (e) (f)

Figure 6.19: DLO8 quadratic function shape manipulation without and with occlusion. (a) DLO8 color frame with tracked shape (in red). (b) DLO8 point-cloud (in green) with tracked shape (in red). (c) DLO8 3D-fitting plot. (d) DLO8 color frame with tracked shape (in red) in case of occlusion. (e) DLO8 point-cloud (in green) with tracked shape (in red) in case of occlusion. (f) DLO8 3D-fitting plot with occlusion, highlighted in purple.

Analyzing the time result shown in Table 6.9, the lower time, evidenced in green, belongs to the tracking of the DLO2 in case of occlusion. This reconfirms what we explain for the previous shapes, namely that lower is the number of acquired points (Figure 6.12d) than lower is the time to fit these points by Lasso regression, which means lower time for each tracked frame. In fact, as in the previously tested shape, the DLO1 takes the longest mean computational time for each tracked frame, which is still only 0.12 s.

Mean computational time for each tracked frame		
	without occlusion	**with occlusion**
DLO1	0.12 s	0.116 s
DLO2	0.078 s	0.061 s
DLO3	0.077 s	0.068 s
DLO4	0.091 s	0.065 s
DLO5	0.068 s	0.066 s
DLO6	0.072 s	0.068 s
DLO7	0.081 s	0.076 s
DLO8	0.085 s	0.082 s

Table 6.9: Mean computational time for each tracked frame with occlusion and without it (quadratic function shape).

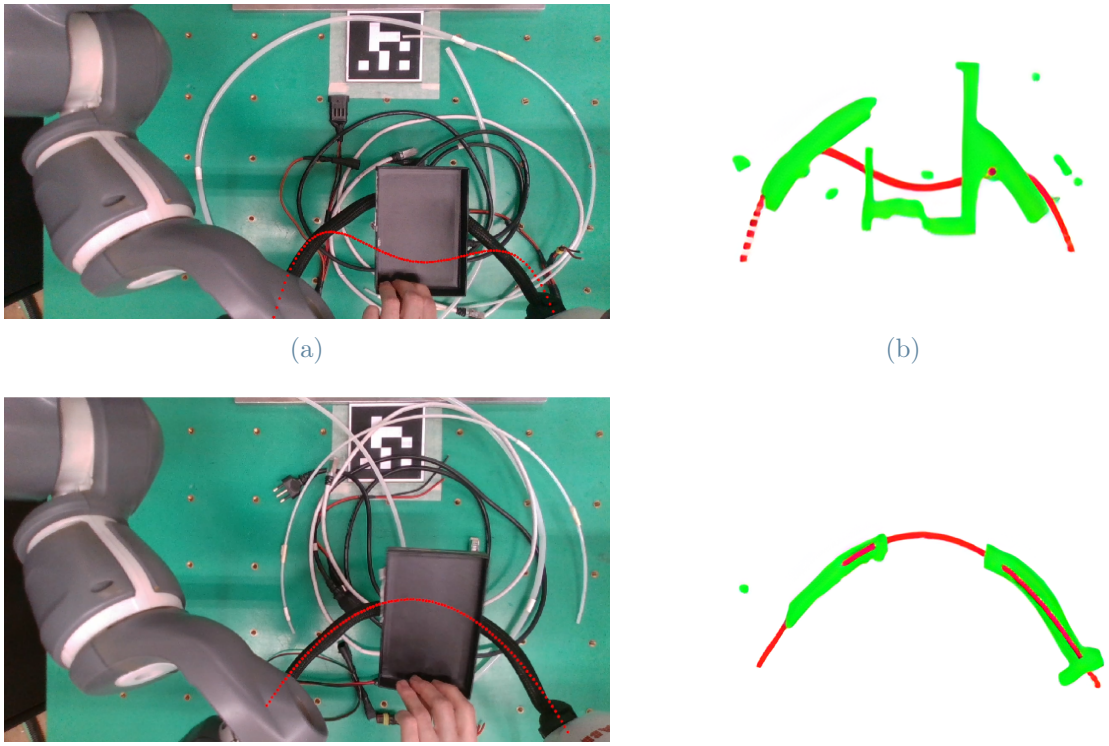
Table 6.10 shows the averaged mean squared error in the x-y and x-z plane. Firstly it can be noticed that the errors in x-z plane for the DLO2, DLO3, DLO7 are decreased with respect to the sinusoidal shape (Table 6.7) because there is a lower change in this plane for this DLOs.
The biggest fitting errors are the $AMSE_{x-z}$, and are generated by the DLOs with high rigidity. This happens because this error takes into account the fitting error from the initial shape which is a linear one, which as we have already seen in the linear case with the DLO5 Table 6.3, generates problems with this kind of DLO. In addition, there is also an error due to not accurate grippers' poses. The worst error is made by the DLO5, which is highlighted in red in Table 6.10. While the DLO7 errors are highlighted in green, since both in the x-y and x-z planes it has the lowest fitting error.

	$AMSE_{x-y}$		$AMSE_{x-z}$	
	without occlusions	**with occlusions**	**without occlusions**	**with occlusions**
DLO1	0.609 cm^2	0.56 cm^2	0.282 cm^2	0.284 cm^2
DLO2	0.09 cm^2	0.08 cm^2	1.4 cm^2	0.165 cm^2
DLO3	0.069 cm^2	0.07 cm^2	0.093 cm^2	0.069 cm^2
DLO4	0.12 cm^2	0.11 cm^2	1.8 cm^2	1.3 cm^2
DLO5	0.045 cm^2	0.089 cm^2	3.8 cm^2	3.3 cm^2
DLO6	0.033 cm^2	0.581 cm^2	0.7 cm^2	0.99 cm^2
DLO7	0.032 cm^2	0.035 cm^2	0.091 cm^2	0.23 cm^2
DLO8	0.045 cm^2	0.15 cm^2	1.5 cm^2	2 cm^2

Table 6.10: Averaged MSE for each tracked frame (quadratic function shape).

6.4. Limitation: Sensitivity to the occluding object color

As just presented in Section 4.6.1 one of the limits of the proposed methodology is that the occluding object, if it is at the same height as the DLO, cannot have the same color as the manipulated DLO. Nonetheless, we proposed a solution to this problem, which consists in run in parallel the online video with occlusion and a sample video without occlusion, and making the AND between the two videos (as detailed in Section 4.6.1).
The first DLO tested is the DLO1 (Figure 6.20). As it can be noticed in Figures 6.20a and 6.20b the black occluding object pixels are acquired by the segmentation phase (Section 4.2) and are reported in the point-cloud, generating a wrong tracked shape. On the other hand, using the AND between the online and the sample video, the occluding object points are correctly filtered out.

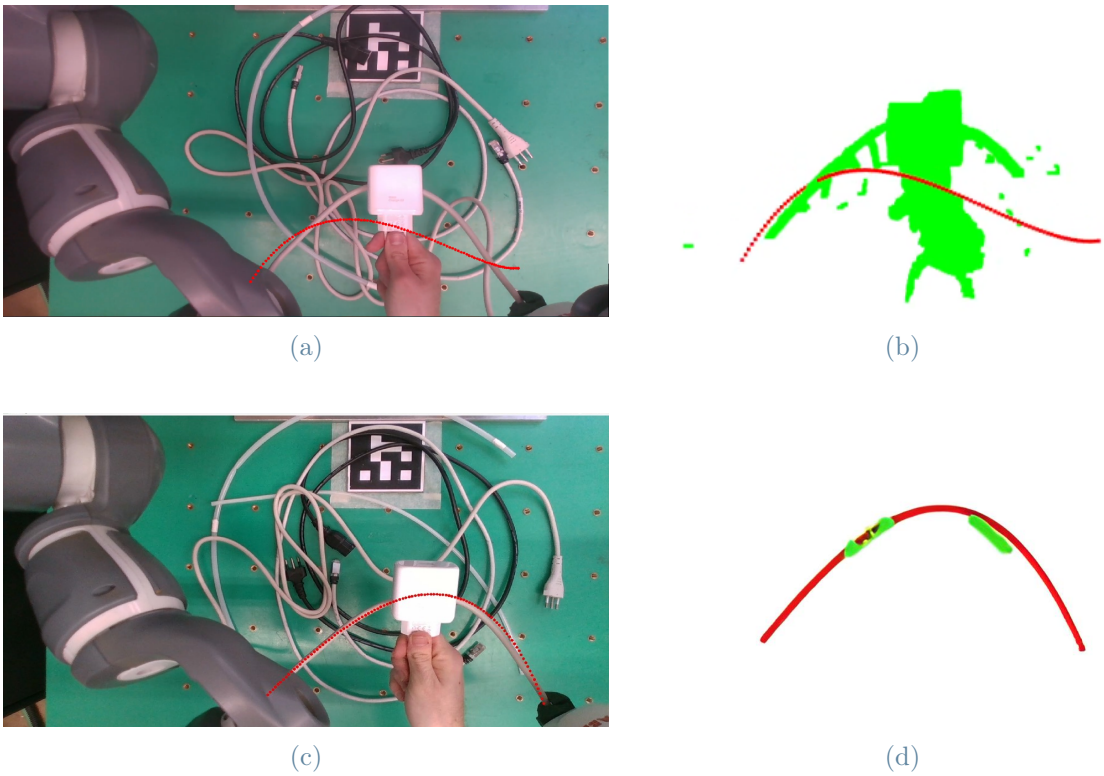


(a) (b)

(c) (d)

Figure 6.20: DLO1 tracking results in case of a black occluding object on it. (a) DLO1 color frame with tracked shape (in red), without the AND between the online and the sample video. (b) DLO1 point-cloud (in green) with tracked shape (in red), without using the AND between the online and the sample video. (c) DLO1 color frame with tracked shape (in red), using the AND between the online and the sample video. (b) DLO1 point-cloud (in green) with tracked shape (in red), using the AND between the online and the sample video.

This method is tested also on the DLO3, in case of an occlusion made by a white object. Figure 6.21a shows a white object that occludes the DLO, note that not only the pixels of the object are reported in the point-cloud but also the hand is reported (Figure 6.21b) generating a wrong tracked shape. However applying the AND method (Figures 6.21c and 6.21d) the tracked shape is equal to the one without and with occluding object color different from the DLO (Figure 6.13).

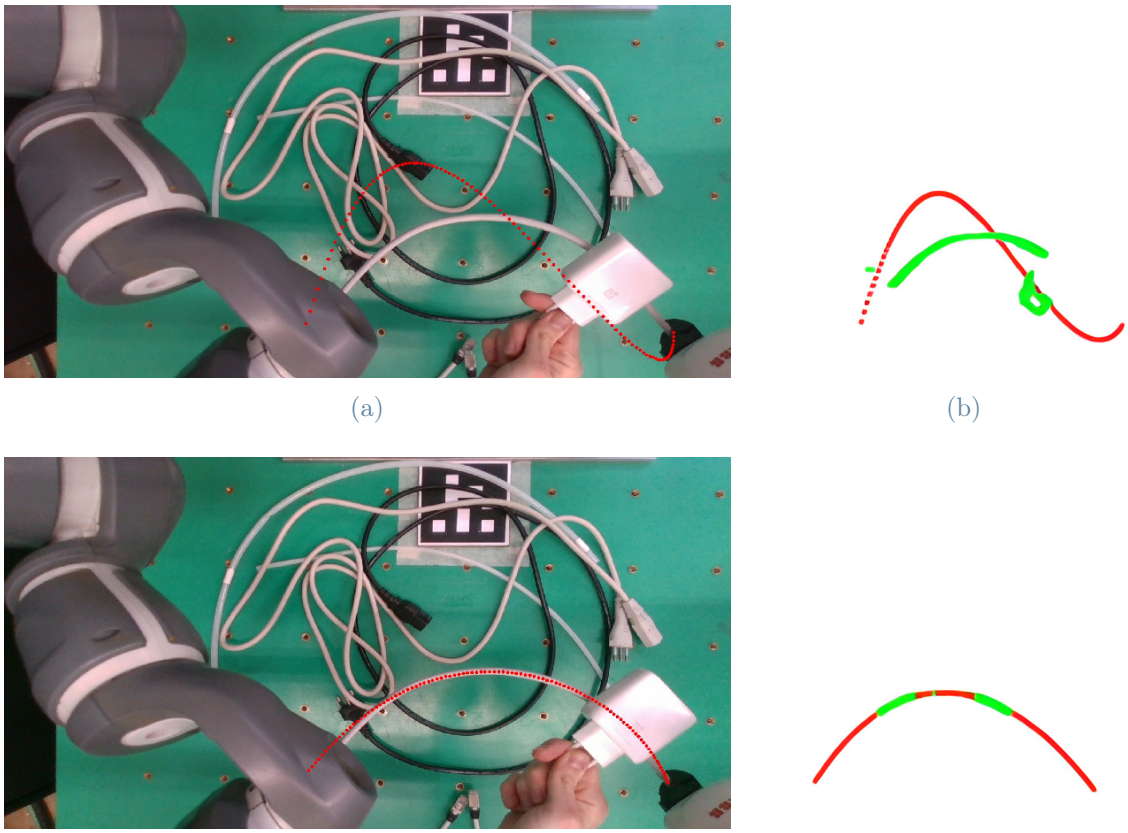


(a) (b)

(c) (d)

Figure 6.21: DLO3 tracking results in case of a black occluding object on it. (a) DLO3 color frame with tracked shape (in red), without the AND between the online and the sample video. (b) DLO3 point-cloud (in green) with tracked shape (in red), without using the AND between the online and the sample video. (c) DLO3 color frame with tracked shape (in red), using the AND between the online and the sample video. (b) DLO3 point-cloud (in green) with tracked shape (in red), using the AND between the online and the sample video.

The last tested DLO is the DLO7, the same white object of the previous test is used as occlusion. Figures 6.22a and 6.22b show that in case the object occludes the right part, Lasso generates a wrong fitting due to the fact, it tries to fit also the object points. By applying the AND between the sample video and the online one with the occlusion (Figures 6.22c and 6.22d), the object points are filtered out and ensures a good estimation of the DLO shape also if the available points are less with respect to the case shown in (Figure 6.18e).



(a) (b)

(c) (d)

Figure 6.22: DLO7 tracking results in case of white occluded object on it. (a) DLO7 color frame with tracked shape (in red), without using the sample video. (b) DLO7 point-cloud (in green) with tracked shape (in red), without using the sample video. (c) DLO7 color frame with tracked shape (in red), with the use of the sample video. (b) DLO7 point-cloud (in green) with tracked shape (in red), with the use of the sample video.

The proposed methodology applies the segmentation phase two times, one for the online video and one for the sample video, hence one of the main problems could be the degradation of the computational time. Table 6.11 shows that this kind of problem does not exist. In fact, the time only increases by 0.03 s for the DLO1, 0.03 s for the DLO3, and 0.025 s for the DLO7, with respect to the case with occlusion of different color or with the same color but at a certain height above the manipulated DLO. This happens because the object points are filtered out before the construction of the point-cloud, meaning that as soon as the color mask is applied, all operations are executed on gray images, so they are computationally efficient.

Mean computational time for each tracked frame			
	without occlusion	**with occlusion**	**with occlusion of same DLO color**
DLO1	0.12 s	0.116 s	0.15 s
DLO3	0.077 s	0.068 s	0.098 s
DLO7	0.081 s	0.076 s	0.101 s

Table 6.11: Comparison of the mean computational time for each tracked frame with and without occlusion, with use of the sample video.

Table 6.12 shows the $AMSE_{x-y}$ and $AMSE_{x-z}$. As can be noticed, the $AMSE_{x-y}$ decreases in case of occlusion of the same DLO color (highlighted in green), this is because the removed occluding object being on the DLO decreases the number of points, which generates a decreasing of the fitting error. While the $AMSE_{x-z}$ increases. As it happens in the previously tested shape, the fitting error is influenced by the gripper's poses which are not accurate, so having a lower number of points and the acquired gripper positions with lower accuracy, generate an increased error in the x-z plane.

$AMSE_{x-y}$			
	without occlusion	**with occlusion**	**with occlusion of same DLO color**
DLO1	0.609 cm^2	0.56 cm^2	0.34 cm^2
DLO3	0.069 cm^2	0.07 cm^2	0.032 cm^2
DLO7	0.032 cm^2	0.035 cm^2	0.024 cm^2

$AMSE_{x-z}$			
	without occlusion	**with occlusion**	**with occlusion of same DLO color**
DLO1	0.282 cm^2	0.284 cm^2	0.3 cm^2
DLO3	0.093 cm^2	0.069 cm^2	0.19 cm^2
DLO7	0.091 cm^2	0.23 cm^2	0.16 cm^2

Table 6.12: Comparison of AMSE for each tracked frame with and without occlusion, with use of the sample video.

The Frangi filter [22] could help with the problem of the occluding object color. It is optimized to detect tubular objects, so if we apply it after the color mask it will automatically filter out all the parts that are not DLO. The cons of this method is that its speed depends on the rescaling of the color frame, smaller is the frame, quicker is the

Frangi filter. However, in our case changing the dimension of the color frame creates a misalignment between the color and depth frame generating a wrong point-cloud. When Frangi is added to the general tracking algorithm in order to track the DLO1, the mean computational time for each tracked frame was 1.57 s, more than 10 times bigger without using it (Table 6.11). This causes the video to stutter, so this is unacceptable for an online tracking algorithm.
In general, another technique that would eliminate this problem can be the use of a Neural Network (NN). In this way is not necessary to use the color mask for the detection. However, the cons is that is necessary to train the NN for each DLO used, leading to expensive data collection and labeling in case of an industrial scenario with different types of DLOs.

6.5. Test 4: 3D-Sinusoidal shape

Differently from the previous shapes, where the change of high is small, and the configuration of the camera can remain the same. In this case, we will have a bigger chance of high, so the camera is reconfigured (Figure 6.23).

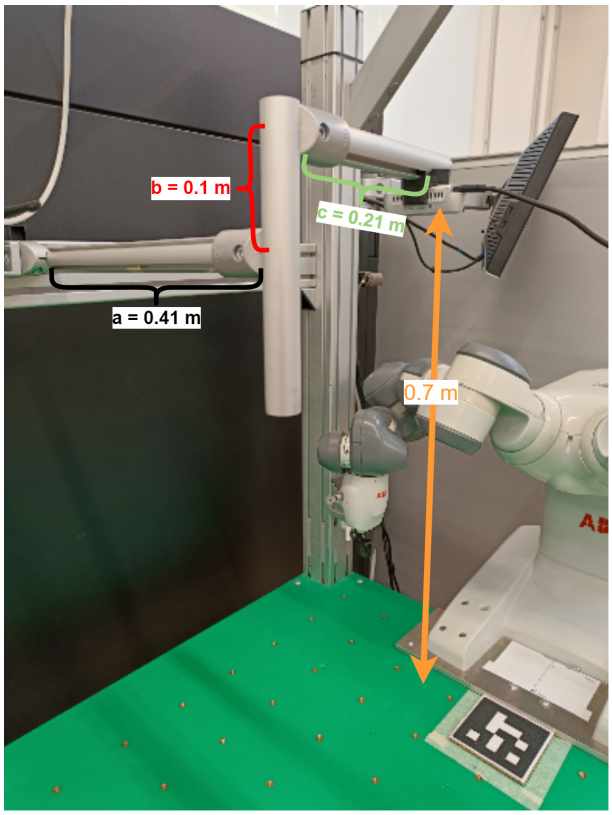


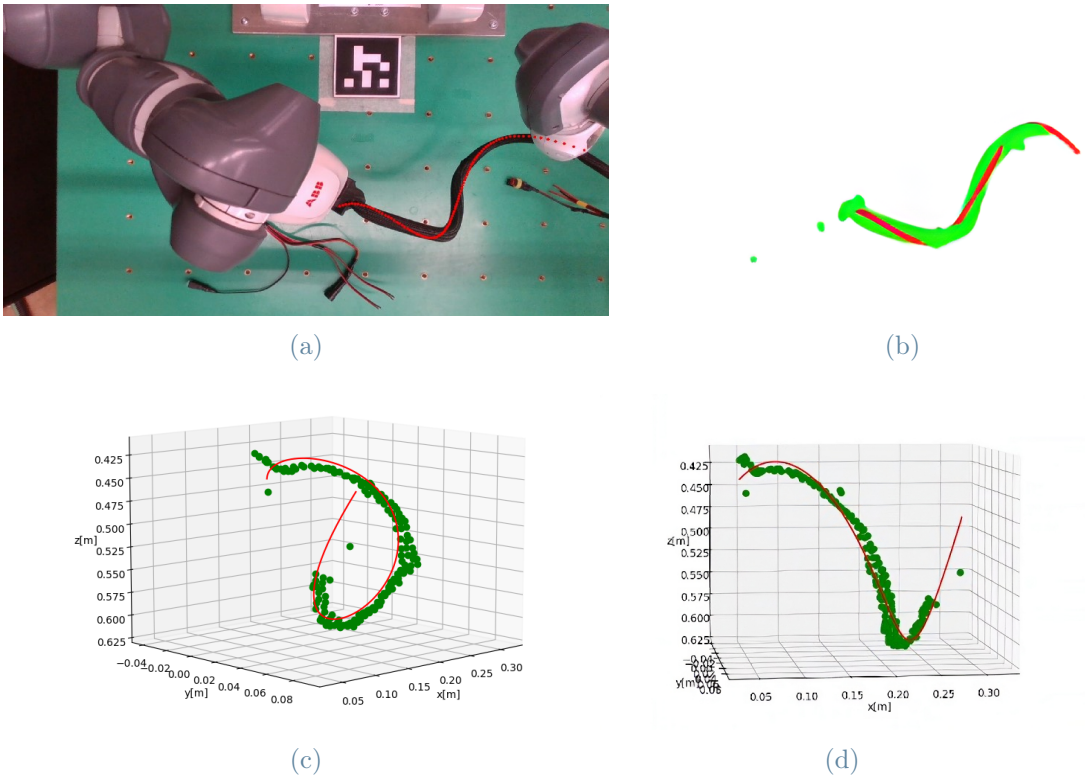
Figure 6.23: Used camera configuration in case of bigger variation along the z-axis.

In addition is also necessary to define the new depth threshold (Table 6.13).

	Depth thresholds	
	upper limit in meters	**lower limit in meters**
DLO1	0.65	0.38

Table 6.13: Values of the depth filter thresholds, in case of 3D sinusoidal shape.

The last tested shape is the 3D sinusoidal shape, it is similar to Section 6.2 but with a huge variation along the z-axis. Figure 6.24 shows the DLO1 result. As can be noticed by Figures 6.24c and 6.24d, $\tilde{Z} = 20\ cm$, so nearly 10 times the $\tilde{Z}$ that we had in the previous sinusoidal shape (Figure 6.6). The mean computational time for each tracked frame is 0.11 s, while the $AMSE_{x-y}$ is 0.5 cm^2, while the $AMSE_{x-z}$ is 0.4 cm^2.



(a) (b)

(c) (d)

Figure 6.24: DLO1 3D sinusoidal shape manipulation. (a) DLO1 color frame with tracked shape (in red). (b) DLO1 point-cloud (in green) with tracked shape (in red). (c) DLO1 3D-fitting plot. (d) DLO1 3D-fitting plot.

7 | Conclusions

The vision based tracking of deformable linear objects (DLOs) manipulated by a dual-arm robot presents several challenges, due to the high number of degrees of freedom of the DLO. In addition, the tracking algorithm has limitations, linked to the impossibility to detect with good accuracy the position and the shape of an object in presence of occlusions, particular lighting conditions, or when similarly shaped and colored objects are placed close to each other in the working area. Finally, an online tracking algorithm needs to be computationally efficient in order to give an estimated tracked shape with a minimum delay with respect to online robot manipulation.

In view of these challenges, this thesis proposes a tracking approach that differs from the literature, being based on the visual information, given by the camera, and on the robot grippers poses, used to define the parameters of the depth filter and to achieve more accurate and robust tracking of the DLO shape. Moreover, this work focuses on DLO shape tracking in industrial applications. For this reason, during the validation, the tracking of the shape of different kinds of DLOs was analyzed. The considered DLOs differ in color, length, and material.

In particular, thanks to the use of the depth filter the robustness during the tracking was achieved also in presence of other DLOs on the working table, which can have also the same shape and/or color as the manipulated DLO. Furthermore, the created filter allows to filter out the points that are not between the grippers, thanks to the implemented segmentation phase. In the case that the segmentation phase fails in this filtering, it was proved that point-cloud downsampling and outlier removal eliminates these points (Figure 6.9). This enlarges the use of the proposed tracking algorithm for different kinds of DLOs without any limits on their length.

Moreover thanks to the acquisition of the grippers poses, the tracking algorithm is robust to the occlusions which can be static or dynamic. In particular, the state of the art does not consider the case of manipulated DLO and occluding objects with the same color. This problem was considered and mitigated in this thesis. In fact, it was softened, considering that the occluding object can be above the manipulated DLO of 4 cm. On the other hand, if the occluding object touches the cable, the tracking is achieved using a

sample video without any occlusion, making an online recovery based on this video which allows the removal of the object points.
In the experimental analysis the tracking algorithm has achieved satisfactory tracking results maintaining a lower mean time for each tracked frame, also in the case the sample video is used and the algorithm needs to work on the online and on the sample video together. Without any information about the rigidity of the DLOs, the algorithm is able to track their shape in 3D, highlighting their change also along the z-axis in the different tested shapes.
Despite the good result, the visual algorithm evidenced some problems in the tracking hoses for compressed air when they are in a linear shape. In fact, due to their particular translucent material and rigidity, problems arise in the acquisition of their depth value by the stereo camera. This, creates for some frames a wrong-tracked shape, resulting in a higher averaged mean squared error (AMSE) in the x-z plane respecting the AMSE in the x-y plane.

7.1. Future Developments

Despite the good results obtained and the relevance of the method in an industrial context, some future works could improve the strategy.
The first improvement can be to run the code on a GPU, in order to have a faster tracking algorithm.
Another improvement can be the use of the proposed online tracking strategy with an offline planner. In fact, the planned shape can be used to have some additional information about the deformation of the DLOs during the manipulation, this can be used to update online the depth threshold of the depth filter, in order to ensure an adaptable threshold for each DLO during the manipulation. Moreover, the tracking algorithm can be used to realize visual servoing.
In addition, the vision algorithm can be improved by adding the Frangi filter, which can add robustness against occlusion, in particular in the case of an occluding object with the same color as the manipulated DLO. But a trade-off between the rescaling of the image and the computational time of the algorithm must be found.
Finally an improvement can be achieved by developing an online recovery method in order to deal with the sensitivity to the occluding object color, without using a sample video.

Bibliography

[1] S. R. Balaji and S. Karthikeyan. A survey on moving object tracking using image processing. In *2017 11th International Conference on Intelligent Systems and Control (ISCO)*, pages 469–474, 2017.

[2] Jose Sanchez, Juan Antonio Corrales, B. C. Bouzgarrou, and Youcef Mezouar. Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37:688–716, 2018.

[3] Nicolas Padoy and Gregory Hager. Deformable tracking of textured curvilinear objects. *BMVC 2012 - Electronic Proceedings of the British Machine Vision Conference 2012*, 01 2012.

[4] Alireza Rastegarpanah, Rhys Howard, and Rustam Stolkin. Tracking linear deformable objects using slicing method. *Robotica*, 40(4):1188–1206, 2022.

[5] Te Tang and Masayoshi Tomizuka. Track deformable objects from point clouds with structure preserved registration. *The International Journal of Robotics Research*, 41:599 – 614, 2019.

[6] John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. In *2013 IEEE International Conference on Robotics and Automation*, pages 1130–1137, 2013.

[7] A. Myronenko and Song. Point-set registration: Coherent point drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:2262–2275, 2010.

[8] Shiyu Jin, Changhao Wang, Xinghao Zhu, Te Tang, and Masayoshi Tomizuka. Real-time state estimation of deformable objects with dynamical simulation. page 11879–11883., 11 2020.

[9] Markus Wnuk, Christoph Hinze, Manuel Zürn, Qizhen Pan, Armin Lechler, and Alexander Verl. Tracking branched deformable linear objects with structure preserved registration by branch-wise probability modification. pages 101–108, 11 2021.

[10] Shiyu Jin, Wenzhao Lian, Changhao Wang, Masayoshi Tomizuka, and Stefan Schaal. Robotic cable routing with spatial representation. *IEEE Robotics and Automation Letters*, 7:1–1, 04 2022.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[12] Waltersson Gabriel Arslan, Rita Laezza, and Yiannis Karayiannidis. Planning and control for cable-routing with dual-arm robot. *IEEE International Conference on Robotics and Automation (ICRA)*, 04 2022.

[13] Shiyu Jin, Changhao Wang, and Masayoshi Tomizuka. Robust deformation model approximation for robotic cable manipulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6586–6593, 2019.

[14] Cheng Chi and Dmitry Berenson. Occlusion-robust deformable object tracking without physics simulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6443–6450, 2019.

[15] Yixuan Wang, Dale McConachie, and Dmitry Berenson. Tracking partially-occluded deformable objects while enforcing geometric constraints. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14199–14205, 2021.

[16] Kangchen Lv, Mingrui Yu, Yifan Pu, and Xiang Li. Learning to occlusion-robustly estimate 3-d states of deformable linear objects from single-frame point clouds, 2022.

[17] HSV color model scheme. `https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png`.

[18] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[19] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing, 2018. cite arxiv:1801.09847Comment: http://www.open3d.org.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[22] Alejandro F. Frangi, Wiro J. Niessen, Koen L. Vincken, and Max A. Viergever. Multiscale vessel enhancement filtering. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 130–137, 1998.

List of Figures

List of Tables