ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
INGEGNERIA BIOMEDICA, ELETTRICA E DEI SISTEMI

Ciclo 36

Settore Concorsuale: 09/G1 - AUTOMATICA

Settore Scientifico Disciplinare: ING-INF/04 - AUTOMATICA

Robotic Perception and Manipulation of Deformable Linear Objects

Presentata da: Alessio Caporali

Coordinatore Dottorato

Michele Monaci

Supervisore

Gianluca Palli

Co-supervisore

Claudio Melchiorri

Esame finale anno 2024

ALMA MATER STUDIORUM UNIVERSITY OF BOLOGNA

Abstract

School of Engineering and Architecture
Department of Electrical, Electronic and Information Engineering (DEI)
"Guglielmo Marconi"

PhD in Biomedical, Electrical and Systems Engineering

Robotic Perception and Manipulation of Deformable Linear Objects

by Alessio Caporali

Deformable objects are pervasive in the everyday life environment, in the form of clothes, cables, wires, ropes and many other objects. Despite their importance and widespread diffusion, there still exist many limitations when it comes to deploying robotic systems for interacting with deformable objects. This thesis presents a comprehensive exploration of research activities geared towards enhancing the perception and manipulation capabilities of a robotic system when dealing with deformable linear objects.

The activities are organized into two main research aspects, namely perception and manipulation. In the first part of this thesis, the focus is on developing perception solutions for deformable linear objects, primarily relying on visual data and exploiting deep learning techniques. Consequently, innovative methods are developed to address the dataset generation challenge with minimal to no human intervention. Furthermore, novel approaches are applied to tackle the instance segmentation task by combining deep learning techniques with graph-based representations of the object's configuration. The 3D reconstruction task is also addressed through a multi-view stereo reconstruction approach.

The second aspect of the research concentrated on the manipulation problem, specifically in predicting how robot actions affect the deformable linear object's configuration. This is achieved by employing a differentiable model of the object's dynamics that is used for planning the optimal manipulation action for achieving a target configuration. The same model is also used for estimating model parameters, thereby improving the prediction accuracy and consequently enhancing the robotic system's manipulation capabilities.

Finally, the perception methods developed in this thesis are extended to encompass the perception of deformable multi-linear objects, such as wire harnesses. To this end, a learning-based topological representation is conceived and applied in the context of a dual-arm disentangling manipulation task.

Contents

1 Introduction **1**
- 1.1 Deformable Objects in Robotics . . . 1
- 1.2 Deformable Linear Objects: Sensing and Manipulation . . . 2
- 1.3 Motivation and Contributions . . . 3
- 1.4 Thesis Structure . . . 4

2 Dataset Generation for Segmentation Tasks **5**
- 2.1 Introduction . . . 5
- 2.2 Related Works . . . 6
- 2.3 Chroma Separation . . . 8
- 2.4 Synthetic Images Rendering . . . 10
- 2.5 Weakly Supervised Annotation of Real Images . . . 11
- 2.6 Test Dataset . . . 15
- 2.7 Experiments . . . 16
- 2.8 Conclusions . . . 24

3 2D Perception: Instance Segmentation and Modeling **27**
- 3.1 Introduction . . . 27
- 3.2 Related Work . . . 28
- 3.3 The *Ariadne+* Algorithm . . . 30
- 3.4 The *FASTDLO* Algorithm . . . 38
- 3.5 The RT-DLO Algorithm . . . 44
- 3.6 Experimental Results . . . 53
- 3.7 Conclusions . . . 58

4 3D Shape Estimation: Combining 2D Perception and Multiple Views **61**
- 4.1 Introduction . . . 61
- 4.2 Related Works . . . 63
- 4.3 The *DLO3DS* Algorithm: Overview . . . 63
- 4.4 Instance Selection and Modeling . . . 64
- 4.5 Shape Estimation from Multiple Views . . . 65
- 4.6 Experimental Validation . . . 70
- 4.7 Conclusions . . . 73

5 Shape Control Task with Online Model Parameters Estimation **75**
- 5.1 Introduction . . . 75
- 5.2 Related Works . . . 76
- 5.3 Analytical Model and DLO State Representation . . . 78
- 5.4 Neural Network-based DLO Model . . . 79
- 5.5 Gradient-based Estimation of Action and Parameters . . . 81

5.6 Shape Control Task with Online Parameters Adaptation 83
5.7 Experiments 84
5.8 Conclusions 89

6 DMLOs Topology Representation Learning 91
6.1 Introduction 91
6.2 Related Works 93
6.3 Method Overview 95
6.4 Graph Initialization 95
6.5 Topology Learning 97
6.6 Solver 103
6.7 Topology-driven Manipulation 106
6.8 Experiments 107
6.9 Conclusions 114

7 Conclusions 115

Bibligraphy 117

List of Figures

1.1 Trend in scientific publications concerning deformable objects and robotics. Scopus search with query *"deformable AND objects AND robotics"*. . . 2

2.1 Outlined procedure for producing eight novel synthetic images through *chroma-separation* and domain randomization. 9
2.2 Images used to replace the background in the output dataset. 10
2.3 Synthetically generated RGB samples of DLOs with instance-wise labels. The samples are obtained exploiting the spline-based representation detailed in Sec. 2.4.1. 10
2.4 VR tracker-based weakly supervised dataset generation. 12
2.5 Transformations involved in the labeling procedure using the VR tracker. The transformations named ${}^{t}T_{r}$, ${}^{r}T_{c}$ are needed to transform the labeled point from the VR coordinates (P_i^t) to points in the robot coordinates (P_i^r) and in the camera coordinates (P_i^c). 13
2.6 Labeling flow performed for each captured image. On the sampled image (a) the input points of the VR tracker captured for each instance are projected (b) and smoothed with a spline curve (c). Thereafter, the CNN-based correction procedure is executed (d), the instance masks generated (e), and the output mask finalized (f). Colors meaning: input tracker points as white crosses, smoothed points in orange and corrected points in cyan. 14
2.7 CNN-based corrector predicting the approximated DLO center. 15
2.8 Samples of the *test set* organized by category and number of intersections. 16
2.9 Qualitative results of the semantic segmentation task on the *test set*. . 19
2.10 Boxplot of the Dice Coefficient computed across the *test set* images on the models trained with the different datasets. In all the tests the predictions are thresholded at 0.3. 19
2.11 Evaluation of w_r on *DeepLabV3+*. To the left is the change of average IoU score with different w_r. To the right is the change of training time over dataset size. 21
2.12 Evaluation of w_r on U^2PL with the comparison between supervised (red) and semi-supervised (blue) training. 21
2.13 Qualitative *test set* samples and predictions of *DeepLabV3+* when trained on 4K synthetic images (S4), 8K synthetic images (S8), a mix of 4K synthetic and 1K real (S4 + R1, $w_r = 0.20$) and a mix of 8K synthetic and 1K real (S8 + R1, $w_r = 0.11$). 22
2.14 In (a) sability measure with the SUS scale [10], the higher the better. In (b) workload measure with the NASA TLX [43], the lower the better. The comparison is established between chroma-key (*CK*), *RITM* [108] and *DLO-WSL*. 23

2.15 Evaluation of the performances in terms of semantic segmentation and number of clicks for the labeling approaches. The segmentation reports the outcomes with chroma-key (*CK*), *RITM* [108], *DLO-WSL* and raw data from the user (without the CNN correction) fitted with a spline (*SPL*). 23

3.1 FPS vs accuracy of 2D DLO-specific perception algorithms (*Ariadne+*, *FASTDLO* and *RT-DLO*) versus general-purpose baselines methods concerning the instance segmentation task. 28
3.2 The *Ariadne+* algorithm. 30
3.3 Superpixelization of an input image containing several DLOs in an industrial setting (a) with a crop of the image in (b). Graph edges computation in (c): superpixel mask before (top) and after (bottom) gradient operation. In (c-bottom) are shown the pixels of the neighboring superpixels in red and blue. 31
3.4 (a) the graph with intersection nodes highlighted in red, (b) the graph simplified, (c) the graph divided into intersection-free clusters (blue) and clusters with an intersection (yellow). 33
3.5 Example of input samples provided to *TripletNet*. The triplet loss is a distance-based loss that operates on three inputs: (a) anchor data; (b) positive data example (similar to the anchor); (c) negative data example. 34
3.6 Example of intersection node (a) and representative patches of the two classes (b,c) for the displayed intersection. *CrossNet* is employed to predict the class of each patch. 38
3.7 The *FASTDLO* algorithm. 39
3.8 Input images with background segmentation results and generated skeletons zoomed at the area of the intersections. To clarify the skeleton visualization, the mask colors in the right column are inverted. 40
3.9 Local neighbors possibilities of a skeleton pixel given a 3×3 kernel. To clarify the representation, the skeleton is dark colored while the background is white. 40
3.10 Endpoint-pair probability computation. For the general endpoint e_i, from the source image I_s and binary mask M_b a feature vector is created as x_i. The embedding vector z_i is obtained after the propagation of x_i in the similarity network layers. A Gaussian activation function on the embeddings L2-distance is employed for calculating the final score p_{ij} of the endpoints e_i and e_j. 42
3.11 (a) *segments* generated; (b) example of intersection processing of the region highlighted area in (a). 42
3.12 Example of the intersection layout estimation with DLOs having identical colors, in (a) and (b), and different colors (c). In all the examples the DLO at the top is blue labeled. 44
3.13 The *RT-DLO* algorithm. 45
3.14 Vertices sampling key elements: the mask M_b (a), M_{dist} (b) and $M_{\max}$ (c), the obtained vertices (d). The bright regions in (b) denote high-intensity values. 47
3.15 Edges processing main elements: (a) K_{nn} edges to obtain initial candidate edge set; (b) positive/negative edges illustration; (c) graph generated; (d) intersection subgraph extracted; (e) subgraph processing schema. . . . 48
3.16 DLOs instances extraction with and without consistency check in case of a problematic mask. 52
3.17 The connectivity graph (a) is processed to extract the DLOs instances and obtain the colored mask M_c (b). 52

3.18 Qualitative evaluation of *RT-DLO* versus *FASTDLO* and *Ariadne+* on the *test set* classes. 54
3.19 Evaluation of *RT-DLO*, *FASTDLO* and *Ariadne+* on the *test set* eroding M_b. 56
3.20 Qualitative comparison of *RT-DLO*, *FASTDLO* and *Ariadne+* given M_b eroded for 1, 2 and 3 iterations. 56
3.21 Evaluation of *RT-DLO*, *FASTDLO* and *Ariadne+* on the *test set* employing M_b obtained by SOS networks. 57
3.22 Qualitative comparison of the instances masks of *RT-DLO*, *FASTDLO* and *Ariadne+* given M_b from *EGNet*. 57
3.23 Analysis of superpixel parameter sensitivity in terms of the number of intersections (i.e. #1, #2 and #3) and average. The number of superpixels is varied between 10 and 90 whereas the reference value of 50 is shown with the dashed gray line. As metrics, the IoU is employed. . 59
3.24 Qualitative comparison of three different values for the number of superpixels for the same image. From left to right: low (10), normal (50), high (90). 59

4.1 Showcase of *DLO3DS* capabilities in reconstructing the shapes of DLOs in different scenarios. 62
4.2 3D shape estimation pipeline of *DLO3DS*. Data flow: the blue arrow denotes the image; the red arrow denotes the 2D spline. 64
4.3 Target spline selection approach based on distance computation. The symbol u denotes the spline-free parameter. 65
4.4 Scaling process. The shortest spline is selected as the reference and all the others are scaled to match the same DLO portion as closely as possible. 66
4.5 Tracking the same DLO after a forward motion by exploiting a distance-based computation on the overlap area. 69
4.6 Experimental setup composed of a Panda robot from Franka Emika and a low-cost eye-in-hand 2D USB camera. 70
4.7 Error distribution on the synthetic *test set* when varying a single parameter of *DLO3DS*. For the optimization plot: $fixed$ means fixed setup, b means just baseline, z means just distance, $b + z$ means both baseline and distance. 71
4.8 Comparison with baseline methods in the form of depth images and boxplot. Plot legend: 0) *DLO3DS*, 1) *SISTER* [28], 2) *CENSUS/SGM* [48] and 3) *MCCNN/SGM* [137]. The display of the *MCCNN/SGM* boxplot in the third row is avoided due to large errors. 72
4.9 Key-frames from a video sequence (available as supplementary material) showing the tracking test performed on DLOs of different types and diameters. Tester gripper diameter: black 6 mm, blue 10 mm. 73

5.1 Schematic view of the two phases composing the proposed manipulation framework: 1) training phase for dataset generation and NN training, and 2) online phase for simultaneous estimation of the best action and model parameters during the shape control task. 76
5.2 DLO analytical model representation. 78
5.3 Example sequence of $k = 5$ dataset samples generated employing a simulated DLO. V_{in} in red, V_{out} in light blue and action arrow in green. 80
5.4 Neural network architecture. With $\times$ the element-wise multiplication is denoted. 81

5.5 Scheme of the proposed gradient-based action and DLO parameters estimation. 82
5.6 Experimental robotic setup comprising different ropes and surfaces (a). Target shapes achieved with the *red* rope on the *cardboard* surface (b). 84
5.7 Outcomes of the shape control task involving online adaptation of model parameters, conducted across various rope types and surfaces. Average results across 5 repetitions per task (standard deviations confidence region intervals). With *cl* and *cb* the *cloth* and *cardboard* surfaces are denoted. 86
5.8 Comparing prediction errors using mid-range, online, and best model parameters across ropes and surfaces. 87
5.9 Prediction error for fix (*no params*) vs conditioning parameters across different models. For the latter, the symbol (*) denotes that the same shape is used for parameters estimation and forward prediction error, whereas with *(**)* the 4-fold cross-validation approach is denoted. With *fixed mid-range* and *varied* the two employed datasets are indicated. . . 88
5.10 The mean prediction error (log scale) of the input parameters (ours) vs neural network weights (RBF [134]) update, evaluated on the same shape (*) or on different shapes (**). 89

6.1 Extracted topology representation of a DMLO from a camera sample. The branch *sections* composing the DMLO are displayed in different colors while the *branch-points* and *intersection-points* are highlighted with red and blue dots. On the right side, the dual-arm robotic experimental setup. 92
6.2 Schema of the proposed approach. On the right side, the topology representation provided as output is shown with the individual branch *sections* of the DMLO denoted in different colors and the single *branch-point* and *intersection-point* highlighted as red and blue dots. 94
6.3 Vertices sampling procedure from segmentation mask: distance transform maks M_{dist}, local maximum mask $M_{\max}$, graph nodes $\mathcal{V}$. 96
6.4 Sample of the synthetic dataset. In (a) and (b) the rendered image and mask of the randomly generated DMLO. In (c), the colored curves represent the different sections and the red dots describe the *branch-points* used in the annotation process. In (d) the obtained ground truth graph is displayed. 98
6.5 Outcomes of the different stages of the pipeline. From the graph with the input set of edges $\mathcal{E}_{\text{knn}}$ (a), the predicted edge probabilities are displayed in (b) while the node orientations are in (c). The result of the node subgraphs classification task between *normal* (0) and *highly-connected* (1) nodes is shown in (d). 100
6.6 Effect of solver stages: (a) Obtained graph after the application of the edge filtering stage on the input graph of Fig. 6.5a; (b) The result of the high-degree nodes handling exploiting the estimated node classes. . . . 103
6.7 Node orientation solver updating the edges around node k in case its orientation is consistent with the nearby existing edge e_{ij} between nodes i and j. 104
6.8 Clusters formation for *branch-point* and *intersection-point* regions. High-degree nodes are denoted in black. 105
6.9 Schematic of the manipulation motion originated for the topology represented. 106
6.10 DMLOs real-world test samples. 108

6.11 Snapshots of the untangling manipulation experiment employing three different automotive DMLOs. The *branch-points* and *intersection-points* are shown as red and blue dots. 109
6.12 Offline evaluation results of the single learned components of the proposed pipeline for the different DMLOs composing the test set. With *ALL*, the entire set of samples is considered. 111
6.13 Comparisons of the proposed method against *RT-DLO* [15] and *mBEST* [24] performing the instance segmentation task on the respective proposed DLOs test datasets. 112
6.14 Ablation study of link prediction and node orientation tasks with components proposed in *RT-DLO*. 113
6.15 Limitations of the proposed method concerning topologies derived from masks that exhibit artifacts and gaps. To pinpoint the specific issue, a red arrow is employed. 114

List of Tables

2.1 The average Dice Coefficient computed across the *test set* images on the models trained with the different datasets. In all the tests the predictions are thresholded at 0.3. 18

2.2 Factorial design for the analysis of dataset mixtures. The table shows the size of real-world dataset given the synthetic dataset and ratio w_r. With † are denoted the configurations where U^2PL is trained also in a semi-supervised fashion. With * are denoted the configurations used only by *DeepLabV3+* and *YOLACT*. 20

3.1 *Ariadne+*, *FASTDLO* and *RT-DLO* versus baseline methods. Key-point denotes that the method also provides a representation of the detected DLOs as sequence of points. The symbol '*' indicates that the method is tested on a reduced dataset. 53

3.2 *Ariadne+* average execution timings of the different main parts and total computed over the test set. Values in milliseconds. 55

3.3 *FASTDLO* main procedures average execution times and total with respect to the number of intersections in the *test set* images, i.e. 1, 2, and 3. Values in milliseconds. 55

3.4 Average execution times [ms] of the main *RT-DLO* stages with respect to the number of intersections in the image, i.e. 1, 2, and 3. 55

3.5 Performances of *RT-DLO* when varying the vertices sampling ratio α and the number of K_{nn} nearest neighbors. In bold the values within 1% distance from the best one. 58

6.1 Offline evaluation results in terms of dice score for the link predicted in graph $\mathcal{G}''$ and detection accuracy for $\mathcal{B}$ and $\mathcal{I}$. 109

6.2 Timings of the main procedures of the approach across the test set. Values in milliseconds. 111

List of Abbrevations

DO Deformable Object

DLO Deformable Linear Object

DMLO Deformable Multi-Linear Object

NN Neural Network

CK Chroma Key

VR Virtual Reality

CNN Convolutional Neural Network

FPS Frames Per Second

DCNN Deep Convolutional Neural Network

RAG Region Adjacency Graph

SGM Semi-Global Matching

GNN Graph Neural Network

Chapter 1

Introduction

1.1 Deformable Objects in Robotics

Deformable Objects (DOs) refer to objects with the ability to change their shape when subjected to external forces. They are commonly found in everyday life, for instance, in the form of clothes, wires, cables, and ropes. However, they are also vastly present in other fields, such as the medical, agricultural and industrial domains.

The problem of DOs sensing and manipulation is an emerging research topic in robotics [132]. Since DOs are ubiquitous in our daily lives, the ability to perceive and manipulate them is a crucial skill for robots to possess [150]. For instance, robots can provide valuable assistance in elderly care by helping with tasks like dressing and handling textile objects [96]. DOs are also commonly found in industrial settings, where they are used for wiring all the electrical components of a vehicle [119, 53] or for the assembly of wire harnesses and wires in general [107]. Manipulating DOs with a robotic system can help to automate these tasks, thus reducing the need for human intervention and increasing productivity [150]. Fruits and vegetables are also DOs, and robots can help with harvesting and handling them [80]. Finally, the medical field is another sector where the manipulation of DOs can be beneficial. For instance, robots can perform tasks such as chaterization, suturing, and tissue manipulation [52, 91].

The research related to DOs spans every aspect of robotics, including perception, planning, and control [150]. The increasing trend in the number of publications related to DOs and robotics, as depicted in Fig. 1.1, underscores the growing interest and commitment of researchers in this field.

The field of DOs is vast and encompasses a wide range of objects with different characteristics. Therefore, it is common practice to categorize DOs based on their geometry and physical attributes [99].

With respect to their geometry, DOs can be classified as either *uniparametric*, *biparametric* or *triparametric*. Uniparametric objects are those with one dimension significantly greater than the other two, such as a cable where the length far surpasses its width and height. On the other hand, biparametric objects are characterized by one dimension considerably smaller than the other two as, for example, a sheet of paper whose thickness is negligibly small compared to its width and height. Triparametric objects refer to solid objects with substantial dimensions in all three aspects.

Concerning their physical attributes, DOs can be grouped into two categories: DOs with *no compression strength*, and DOs with *large strain*. DOs lacking compression strength, like ropes and clothes, deform and change their shape without offering any resistance when subjected to forces applied from opposite ends. In contrast, DOs with large strain have the capacity to deform under the influence of force but can also return

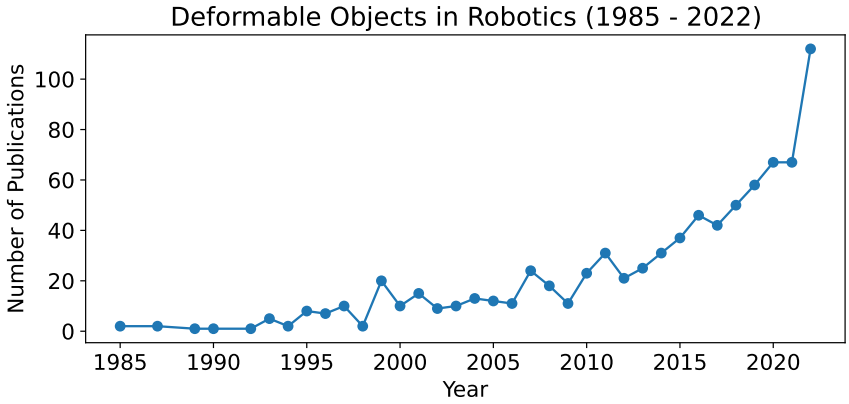


Figure 1.1: Trend in scientific publications concerning deformable objects and robotics. Scopus search with query *"deformable AND objects AND robotics"*.

to their original shape. Examples of such objects include springs and elastic wires.

Deformable Linear Objects (DLOs), which are the central focus of this thesis, fall into the category of uniparametric DOs. They can exhibit either *no compression strength*, similar to ropes, or *large strain*, as seen in cables and wires. The following section provides an overview of DLOs from the perspective of robotics.

1.2 Deformable Linear Objects: Sensing and Manipulation

The category of DLOs comprises a diverse array of items, including wires, cables, ropes, elastic tubes, and strings. Slightly differing from DLOs, there exists another class of objects known as Deformable Multi-Linear Objects (DMLOs), which encompass structures like wire harnesses, hoses, and other constructs made up of multiple interconnected DLO components. DMLOs share numerous attributes with DLOs, with the primary distinguishing factor being the presence of *branch-points*, which are points where two DLO components are interconnected. Therefore, within this thesis, the term DLOs is employed to encompass both DLOs and DMLOs. If necessary, explicit differentiation between these two categories is provided.

DLOs are commonly found in industrial scenarios, for instance, in the automotive [119, 53] and aerospace [107] industries. Despite their widespread use in industrial applications, automating processes involving DLOs and DMLOs remains a formidable challenge [119]. This is primarily attributed to the scarcity of effective solutions for perceiving and manipulating these objects.

The perception of DLOs can pose challenges due to their inherently ambiguous characteristics. The absence of distinctive shapes and colors makes it challenging to differentiate between various DLOs within a given scene and to distinguish DLOs from other objects. Furthermore, their relatively small dimensions, particularly in terms of diameters, present an additional hurdle when it comes to their 3D perception, which many sensors struggle to achieve, as highlighted in [26]. Finally, their deformable nature compounds the challenges associated with perception.

Manipulating DLOs can also pose a challenge as their intrinsic deformability results in a high-dimensional state space with complex and nonlinear dynamics [128, 77]. Therefore a deep understanding of their physical characteristics is required in manipulation tasks to predict and control their shape effectively.

1.3 Motivation and Contributions

The objective of this thesis is to develop robotic solutions tailored for the perception and manipulation of DLOs in diverse environments. These environments range from complex, unstructured settings like domestic spaces to semi-structured, yet demanding industrial environments.

Robotic applications typically require a reliable perception of target objects to effectively plan and execute manipulation tasks. Consequently, the perception system must be capable of furnishing 3D scene information. Typically, this is achieved by employing a 3D sensor such as an RGBD camera. Hence, the initial portion of this thesis is dedicated to addressing the challenge of achieving 3D perception of DLOs starting from 2D vision data.

Upon establishing a reliable solution for DLO perception, the subsequent task is to tackle the complex problem of DLO manipulation. Manipulating DLOs presents a formidable challenge due to their deformable nature, which renders their behavior during manipulation challenging to predict. Therefore, part of this thesis is dedicated to resolving some challenges associated with the prediction of the change in the DLO shape subject to the action of the robot.

The presence of *branch-points* in DMLOs makes them more demanding to perceive and manipulate compared to DLOs. Indeed, methods and algorithms developed for DLOs cannot be directly applied to DMLOs. Therefore, the final portion of this thesis is dedicated to extending the discussion to DMLOs.

1.3.1 Perception of DLOs

Solving the 3D perception problem for DLOs is challenging due to their small dimensions and deformability, as mentioned before. Therefore, in this thesis, the 3D perception of DLOs is accomplished by first solving the 2D perception problem. The 2D detection can then be extrapolated in the 3D domain by either relying on accurate 3D sensors capable of dealing with the DLOs' thin diameters or by combining the 2D results with a multi-view stereo approach, thus requiring only a simple 2D camera.

In images, deep learning approaches able to obtain precise and robust results are employed for detecting and segmenting DLOs. However, the problem of gathering the data required by the deep learning pipelines needs to be addressed. Manually annotating data containing multiple DLOs is a time-consuming and tedious task. Therefore, approaches for generating training data with zero or minimal human intervention have been developed.

Thereafter, the instance segmentation task is addressed. This task is crucial for the 3D shape estimation since it allows the association of the 2D detections with the corresponding 3D points, for instance by utilizing an RGBD camera. Instance segmentation means that each pixel of the image is assigned to a specific instance of the object class. This is a challenging task for DLOs since they are usually characterized by a lack of specific shape and color. Therefore, novel approaches for instance segmentation of DLOs have been formulated.

Finally, the 3D shape estimation task is tackled. The goal of this task is to provide a full description in the 3D space of the shape of the DLOs. Considering the difficulties that arise when it comes to detecting small and thin objects with the majority of 3D sensors, a novel approach is proposed. It combines the results of the 2D instance segmentation with a multi-view stereo approach for accurately reconstructing the 3D shape even of very thin DLOs.

1.3.2 Manipulation of DLOs

Manipulating DLOs is a challenging task for a robotic system due to their unpredictable configuration, high-dimensional state space and complex nonlinear dynamics. Model-based approaches are typically employed to tackle the manipulation problem. However, the high-dimensional state space and the complex dynamics of DLOs make it challenging to obtain an accurate and efficient model. Therefore, a data-driven approach is proposed. It relies on a Neural Network (NN) to learn the dynamics of the DLOs and to predict their behavior during manipulation. The NN is trained using an analytical model of the DLO dynamics. The complete task developed is the model-based shape control of DLOs with the simultaneous online gradient-based estimation of model parameters.

1.3.3 Extension to DMLOs

The extension of the proposed perception and manipulation strategies to the sub-category of DMLOs is also addressed. The main difference between DLOs and DMLOs is the presence of *branch-points*. Therefore, the perception problem is tackled by extracting a topological representation of the DMLO analyzed which is in turn employed for performing a manipulation task.

1.4 Thesis Structure

This thesis is organized as follows:

- **Chapter** 2 presents the data generation problem and proposes three different methods for collecting and annotating images. These methods are used to generate the datasets employed in Chapter 3.
- **Chapter** 3 tackles the issue of 2D detection and segmentation. It proposes three distinct learning-based approaches for solving the instance segmentation task. The approaches are presented in chronological order of development highlighting the improvements and differences between them.
- **Chapter** 4 addresses the problem of estimating the 3D shapes of DLOs via an approach employing only a simple 2D camera. The proposed solution combines the results of Chapter 3 with a multi-view stereo approach for accurately reconstructing the 3D shape even of very thin DLOs.
- **Chapter** 5 presents a manipulation framework for shaping DLOs toward a goal. Simultaneously to the manipulation, several model parameters are estimated for improving the capabilities of the robotic system.
- **Chapter** 6 extends the perception methods and algorithms developed for DLOs to the sub-category of DMLOs. Therefore, the perception problem is tackled by extracting a topological representation of the DMLO analyzed which is in turn employed for performing a manipulation task.
- **Chapter** 7 draws the conclusions about the overall work and reports open issues and possible future research developments.

Chapter 2

Dataset Generation for Segmentation Tasks

Deep learning approaches provide remarkable results in segmentation tasks. However, the problem of gathering the data required by the deep learning pipelines affects and limits their potential. Manually annotating data containing multiple DLOs is a time-consuming and tedious task. Therefore, approaches for generating training data with zero or minimal human intervention are developed.

2.1 Introduction

In computer vision, common tasks involve image classification, object detection and segmentation, either semantic-wise or instance-wise. Image classification consists of assigning a label or class describing the content of the image. Object detection is a harder task having the goal of assigning bounding boxes to different semantic objects. Finally, the segmentation task refers to the general problem of labeling each pixel of the image either by an object semantic class (semantic segmentation) or instance-wise (in the instance segmentation case).

The availability of big public datasets, e.g. [33, 12, 66], has promoted advances in deep learning algorithms in computer vision applications, such as image classification, object detection and semantic segmentation [44]. As a consequence, the key issue in modern computer vision deals more and more with gathering and labeling big amounts of data to be used for training deep learning models.

Usually, the process of segmenting and annotating training images is performed manually, and it is notoriously tedious, inaccurate and time-consuming. Moreover, the more complex the visual perception task is, the slower becomes the required annotation procedure. For instance, labeling a single image for 2D semantic segmentation can take several hours. Innovative companies, like Scale.ai, Superannotate.ai, Segments.ai and many others, are basing their business on advanced image labeling pipelines that can speed up and lighten the burden. These solutions often exploit a superpixel algorithm [1] which helps the user quickly select large portions of the image instead of individual pixels. Other new approaches rely on weakly supervised learning [148] that iterates between image labeling and model training to provide the user with initial and coarse labels for each new image instead of having it labeled from scratch.

The aforementioned big public datasets [33, 12, 66] usually concern general classes (e.g. person, car, tree, cat, dog, etc.) that may not suit the needs of a specific task. Robotic applications, especially in industrial settings, typically require detection or

segmentation with a very high success rate of a small but very specific set of object instances captured from different viewpoints in highly cluttered scenes.

DLOs, more than other objects, have some peculiarities that bring some interesting challenges on vision tasks: 1) they are deformable objects, which means that they are not characterized by a specific shape; 2) they are very lacking in features; 3) they aren't characterized by any particular color. Considering the peculiarities of DLOs and the final goal of this thesis, i.e. perception and manipulation methods for robotic applications, vision tasks like object detection or image classification are not relevant. Indeed, a bounding box is not sufficient to characterize the DLO configuration in the image. Instead, tasks like semantic or instance segmentation allow to properly set apart the DLO configuration. Thus, the rest of this chapter addresses the 2D perception problem by mostly tackling the segmentation of DLOs from images.

Since a DLO can feature a wide variety of shapes and colors, relying on standard computer vision methods is not sufficient. Instead, deep learning tools can be exploited since they can be trained to generalize well on the challenging class of DLOs. Learning this variance with a trained segmentation model is not an easy job and the success is strictly related to the quality of the training data. Thus, the generation of a large-scale dataset able to cover this variability as much as possible is of paramount importance.

This chapter, having motivated the lack of simple and effective solutions to generate big image datasets for training in the specific field of DLOs (see Sec. 2.2), proposes three approaches to generate training samples of DLOs with zero or minimal human intervention. The methods and results discussed in this chapter have been published as [136, 19]. Specifically, in Sec. 2.3 a method to generate images of DLOs by exploiting the chroma separation principle between background and foreground is presented. In Sec. 2.4, synthetic images of DLOs with varying shapes and colors are generated by exploiting a rendering engine. Finally, in Sec. 2.5 a method to collect real images of DLOs and perform the annotation process with a weak supervision is discussed.

2.2 Related Works

The annotation processes for semantic segmentation are labor-intensive using traditional methods [98, 66]. The efficient generation of ground truth segmentation masks, in general, is explored in many studies. Indeed, a lot of research effort has been spent on investigating alternative strategies to help the human operator in this task [149]. The research area is primarily motivated by the need for large data variability in training datasets to achieve performance saturation [111]. Furthermore, supervised methods, i.e., those trained on manually annotated data, currently best perform in public benchmarks [79]. However, the manual labeling process can become burdensome with large datasets. Therefore, to make the image labeling process more efficient, a broad range of methods come into consideration. Specifically, two major groups or methods are 1) weakly or semi-supervised segmentation detailed in Sec. 2.2.1, and 2) synthetic image generation and domain adaptation discussed in Sec. 2.2.2. Additionally, the robotic community has also developed some methods to generate datasets for object detection and pose estimation, as discussed in Sec. 2.2.3.

2.2.1 Weakly Supervised Learning

To minimize the need for pixel-wise annotated training data, the development of weakly supervised training methods is one of the most active research fields. Weakly supervised learning studies attempt to construct predictive models by learning with incomplete, inexact or inaccurate supervision [148]. Here, all kinds of weak supervision sources

are explored to generate segmentation masks, including image labels [88], point clicks [87], bounding boxes [9], or scribbles [65]. Alternatively, saliency detection can be employed [139], but it cannot differentiate among objects' instances. Furthermore, the performances of models trained in these ways are still significantly worse than that of models trained with fully annotated masks [79].

To correct faulty detections from weakly annotated data, interactive segmentation is introduced. In interactive segmentation frameworks [94] small portions of target objects are roughly highlighted by human operators through markers, called seeds. These seeds are used for a training stage that will produce some rough labels for all other images. The user can then produce more seeds and repeat the procedure until the desired quality level is reached.

In [3] the authors present a method that generates initial pixel-wise masks starting from a human-annotated bounding box. The predicted mask can be subsequently annotated with corrective clicks, which are used by the network to generate an improved mask. This process is repeated until the expectations of the human annotator are reached. More than that, the segmentation model generating the mask predictions is re-trained using the information from the initial bounding boxes and eventual corrections.

In [74] an industrial weakly supervised labeling tool is introduced where the image is first oversegmentated in superpixels and the user must label only a few superpixels per class. Then, based on a superpixel similarity metric, an algorithm annotates the remaining superpixels.

Another approach is an automatic adjustment of the given weak noisy labels which are corrected based on specific knowledge about their generation process [147]. In the work of [69] this approach is applied by using gradient guidance to automatically correct manual annotations of edge positions. A different idea, however, is proposed in [108] where the authors show that using pre-trained networks to infer object masks after the user labels a few points can diminish the labeling effort.

Despite these promising results, all the presented methods apply to single images only. Therefore, they are difficult to scale to big datasets while limiting user label points. Additionally, there is a certain gap between models trained by weak/semi-supervision and models trained by full supervision, although many researchers are making efforts to reduce this gap.

2.2.2 Synthetic Images and Domain Adaptation

A widely adopted strategy for efficient high-volume labeled data generation is to produce synthetically rendered images [31, 93]. With this approach, the generation of a virtually unlimited number of synthetic and photo-realistic images encompassing various types of objects with different sizes, shapes, compositions and 3D distributions is enabled. Thereby, the respective synthetic ground truth segmentation mask is automatically derived for each generated image reaching a segmentation accuracy comparable to human-made labeling. This method is already applied to the problem of three-dimensional pose estimation of DLOs tips by [36] but is also used in related research areas [118, 115, 117, 110, 100, 41].

However, simulated data can cause a domain gap (or shift) which refers to a loss in model performance due to a difference from training data to test data [90]. In order to reduce this shift, several *domain adaptation* techniques are usually applied [100, 97, 8]. Recent works [100, 32, 145] focus on developing ad-hoc adaptation techniques to close the performance gap between training and test distribution. Unfortunately, the performance achievable is still quite far from those obtainable training on real data or fine-tuning on a few annotated samples.

2.2.3 Datasets for Robotic Applications

Several approaches for creating datasets have been developed also within the robotics research community. A semi-automatic method to create labeled datasets for object detection is presented in [29]. The system leverages a 2D camera moved by means of a robot and an augmented reality pen to define the initial object bounding box. In [138] a 6D pose estimation system for Amazon Picking Challenge is presented, where a set of target objects is placed on the shelf and they are segmented and labeled from depth and multi-view information. This work, not only requires depth information but is also strongly tailor-made to the task's domain.

2.3 Chroma Separation

In this section, a novel method for generating a dataset of DLOs for semantic segmentation is presented. The proposed strategy employs a *chroma-key* (CK) technique to first label a set of images and then replace the background to randomize the domain and enlarge the dataset. The procedure is illustrated in Fig. 2.1.

2.3.1 Auto-labeling with Chroma Key

The CK is a technique widely used in movies and motion picture industries to combine two images, usually foreground and background. It requires a foreground image I_{fg} containing a target object that is overlapped with a background image I_{bg}. The target must be placed in front of a monochromatic panel, called *screen*, usually green or blue. The technique consists of a *chroma-separation* phase, where the target foreground object is isolated from the monochromatic panel, i.e. original background, and then an *image-overlay* phase, where the foreground is composed with a new background of choice. In the *chroma-separation* phase, a specific hue range is selected such that it contains solely the color of the screen (e.g. green) and excludes any other color belonging to the foreground. Then, by finding the pixels within that range, a mask for the target M_{t} and a complementary mask for the monochromatic background M_{c} are obtained. Thus, creating a dataset with this technique is straightforward and it can be done in just two steps:

1. record a high-quality video of the target object on a green screen, from which the input images are extracted;
2. find the chroma range of the pixels belonging to the monochromatic background and create the correspondent mask employing *chroma separation*.

In the DLOs dataset, while gathering the images, the DLO is held by its extremities and moved within the frame composing different shapes. To generalize more, the light setups are changed, as well as the DLO color and the number of DLOs in the scene. From a random video frame, the hue levels for the specific screen color are easily found. These levels, once found for one image, remain valid for any other image taken with the same light temperature setting and white balance. Hence, knowing the chroma range of the screen, it is possible to immediately obtain the mask for the DLO from each frame in the video.

2.3.2 Domain Randomization

The labeling procedure with *chroma separation* automatically generates labeled data ready to be used for training, but with low variability. In fact, in the gathering phase,

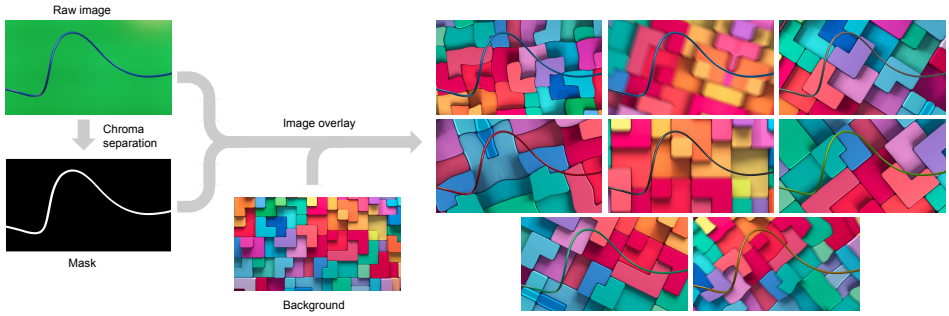


Figure 2.1: Outlined procedure for producing eight novel synthetic images through *chroma-separation* and domain randomization.

the scene featuring the target object is randomized in the following aspects: number of instances, color, size, position and shape. Nevertheless, the background is always uniform and monochromatic. The performance of a segmentation algorithm trained with images having homogeneous backgrounds would be significantly degraded when working in a complex and chaotic environment. Indeed, a cluttered background would easily confuse a learning-based segmentation algorithm, due to possible similarities between the target and the background, especially if the algorithm has never seen them during the training phase. This weakness can be readily overcome by replacing the background in the input images (i.e. *image-overlay* phase). In fact, by using the masks, the foreground can be combined with a random background that replaces the green screen. This process, known as domain randomization [8, 116], aims to provide enough synthetic variability in training data such that, at test time, the model is able to generalize to real-world data. Hence, the choice of background images is a key point for generalizing well to multiple real-world target domains without the need to access any target scenario data in training.

The backgrounds proposed for achieving a domain-independent dataset can be divided into three categories: (1) lowly textured images with shadows and lights; (2) highly textured images with color gradients and regular or geometric shapes; (3) highly textured images with chaotic and irregular shapes. In Fig. 2.2 a set of candidate background images following the proposed style are shown. These backgrounds introduce high variance in the environment properties that should be ignored in the learning task. The segmentation algorithm will learn to ignore shadows and cubic or spherical objects, while it will be forced to focus more on cylindrical shapes.

The presented method introduces two main difficulties that must be faced. The first evident issue of CK concerns the color of the target object. The color histogram of the target object should be clearly separated from the range reserved for the screen. For instance, green DLOs on a green screen are not possible. This implies that the segmentation algorithm will never see green DLOs in training, meaning that if it encounters a green DLO in a real scene, it would likely produce some false negative. The solutions to this issue are two: a different background for the green objects (e.g. a blue screen) can be employed or, the hue of the DLO can be randomized, trying to cover the missing color range (i.e. green).

Another issue is caused by the background replacement, which introduces a discontinuity in the synthetic image generated. This may be problematic for the learning, especially in the case of DLOs, since the algorithm will probably focus on that sharp feature to segment the object, compromising the prediction in a real image, devoid of

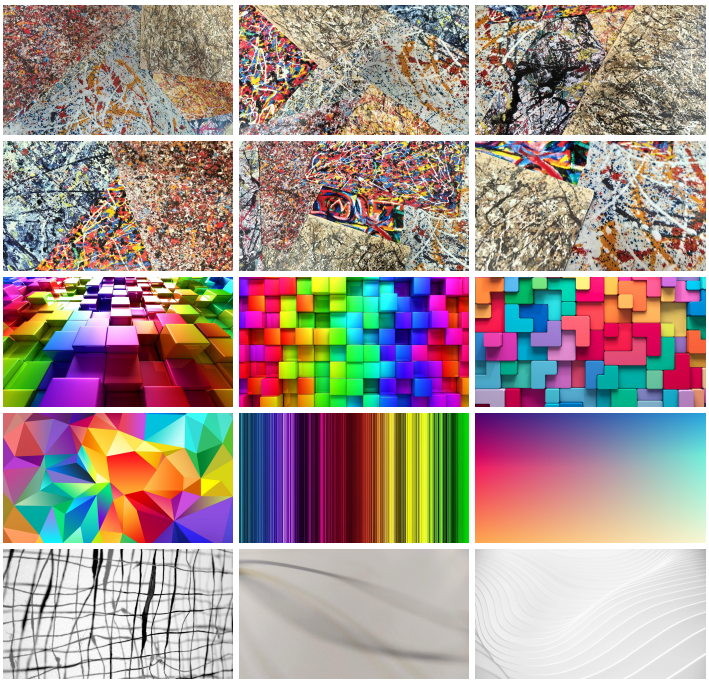


Figure 2.2: Images used to replace the background in the output dataset.

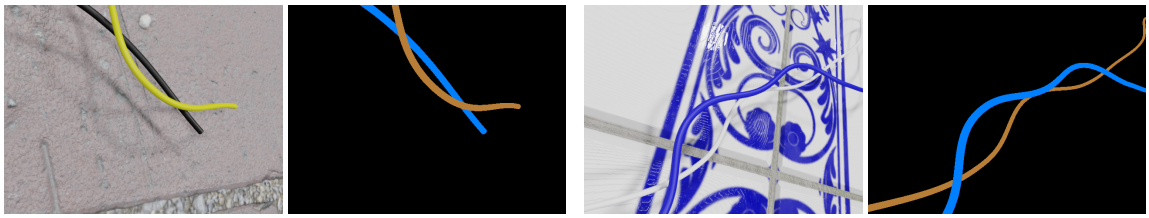


Figure 2.3: Synthetically generated RGB samples of DLOs with instance-wise labels. The samples are obtained exploiting the spline-based representation detailed in Sec. 2.4.1.

the learned discontinuity. To overcome this issue, the output image I_{out} is obtained according to the following formula

$$I_{\text{out}} = M_{\text{t}}^{\mathcal{G}} I_{\text{fg}} + (1_{h \times w} - M_{\text{t}}^{\mathcal{G}}) I_{\text{bg}}. \tag{2.1}$$

i.e. as a linear combination of the foreground I_{fg} and background I_{bg} images weighted respectively by the target mask processed by a Gaussian filter $M_{\text{t}}^{\mathcal{G}} = \mathcal{G}(M_{\text{t}})$ and its complement $(1_{h \times w} - M_{\text{t}}^{\mathcal{G}})$, where $1_{h \times w}$ is a unit matrix with the same size of the mask.

2.4 Synthetic Images Rendering

The CK method, detailed in Sec. 2.3, presents a robust approach for creating a DLO dataset for semantic segmentation with minimal human involvement, primarily in data collection and the initial labeling stages. However, it proves less suitable for generating datasets for instance segmentation tasks, as illustrated in Fig. 2.3, due to its inability to distinguish between distinct DLOs within the scene. To address these limitations, a novel

approach is introduced for constructing a DLO dataset. The method represents DLOs as spline curves and utilizes a rendering engine to generate entirely synthetic photo-realistic DLO images, complete with instance-level labels. The spline-based representation of DLOs is detailed in Sec. 2.4.1, while the rendering pipeline is described in Sec. 2.4.2.

2.4.1 Spline-based DLO Representation

A generic DLO shape can be represented in the Cartesian space by a 3rd-order spline basis as a function of a free coordinate u representing the position along the cable starting from an endpoint ($u = 0$) to the opposite end ($u = L$) being L the length of the DLO. That is:

$$q(u) = \sum_{i=1}^{n} b_i(u) q_i \tag{2.2}$$

where $q(u) = [x(u)\ y(u)\ z(u)]^T$ is the vector of Cartesian coordinates of each point along the DLO, $b_i(u)$ is the i-th elements of the spline polynomial basis used to represent the DLO shape and q_i are n properly selected coefficients, usually called *control points*, used to interpolate the DLO shape through the $b_i(u)$ function basis. Since open ended DLOs are considered, a simpler spline-based representation is preferred over more advanced parametric curves, such as NURBS. However, the proposed approach can be easily extended to NURBS-like curves.

The set of control points is constructed with an iterative propagation method. Being p_t the last element of the ordered set of already generated control points, the new point p_{t+1} is defined as $p_{t+1} = p_t + s\,d$ where s is the propagation step randomly selected between two boundary values and d describes a random direction vector pointing forward with respect to the existing sequence of points. The z component of the point is constrained within specific limits to ensure that the resulting curve closely matches an actual DLO shape. As a result, a random sequence of points is generated and subsequently interpolated into a spline curve, as described in eq. (2.2).

2.4.2 Synthetic Image Rendering

Concerning the rendering of the synthetic images, a novel pipeline making use of Blender is exploited [31]. A mesh object is created based on the DLO model generated through splines. Parameters such as DLO thickness, color, and stripes are defined during this process. These attributes can also be randomized to introduce greater diversity into the dataset. Additionally, random background textures and lighting settings are selected to add further variability, enabling the simulation of various shadow combinations. These techniques are essential for enhancing the generalization capabilities of data-driven methods and fall into the previously introduced domain randomization paradigm [116]. The final step of the rendering pipeline is the generation of the RGB image, which is performed by rendering the scene from a randomly sampled camera perspective.

Furthermore, the rendering pipeline not only generates images but also produces label data, usually in the form of mask images. For example, in Fig. 2.3, one can observe some images created using the proposed procedure, along with their corresponding labels.

2.5 Weakly Supervised Annotation of Real Images

The approaches of Secs. 2.3 and 2.4 can be susceptible to the domain gap issue. The first due to inaccuracies at the level of DLOs boundaries, the latter due to the artificial

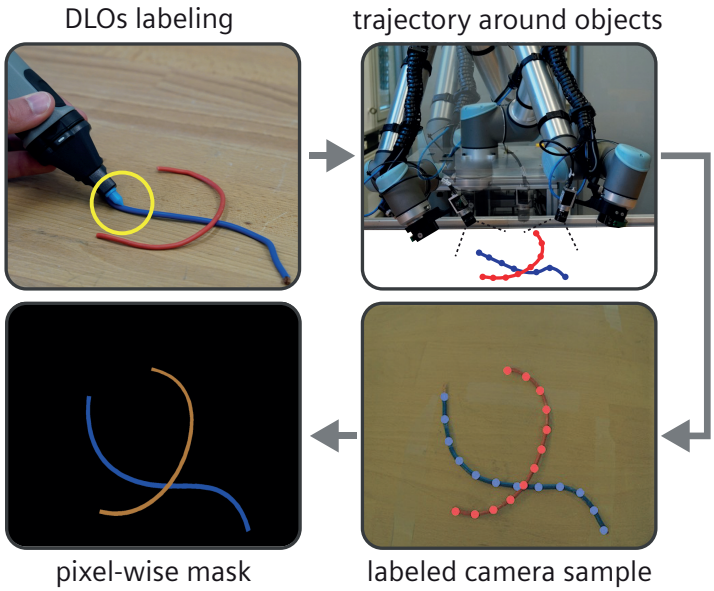


Figure 2.4: VR tracker-based weakly supervised dataset generation.

nature of the data. To address these constraints, an approach is introduced for creating a dataset of real images that come with instance-level labels.

The method is based on the use of a VR tracker to label DLOs in real images. A schematic view of the approach is detailed in Fig. 2.4. In particular, at first, the dataset is recorded as outlined in Sec. 2.5.1. Then, instances of DLOs are labeled via a VR tracker as described in Sec. 2.5.2. Finally, the generated labels are corrected to account for possible calibration and user input errors as illustrated in Sec. 2.5.3. The proposed weakly supervised labeling procedure is denoted in the following as *DLO-WSL*.

2.5.1 Recording of Images with a Robot

To create a dataset via *DLO-WSL*, images along with the position of the camera in the world coordinate system are required. Therefore, a 2D RGB camera is mounted on the flange of a robotic arm in an eye-in-hand configuration, as shown in Fig. 2.4. This configuration offers a dual advantage: it allows for the rapid capture of multiple images and ensures that the camera's position is continuously known while mechanically connected to the robot

However, realizing these benefits required addressing two key challenges. First, the transformation from the camera's frame to its mounting position was determined through an iterative camera calibration process described in [64]. Second, a robot trajectory with an ellipsoidal shape was incorporated into the robot's control system to guarantee that the object always remained at the center of the trajectory when viewed by an inward-facing camera. The trajectory is calculated using eq. (2.3), where x, y, z represent the trajectory points, a, b, c are the ellipsoid parameters, θ denotes the zenith angle, ϕ represents the azimuth angle, and x_0, y_0, z_0 correspond to the initial position coordinates."

$$\begin{cases} x = a \sin\theta \sin\phi + x_0 \\ y = -b \sin\theta \sin\theta + y_0 \\ z = c \cos\theta + z_0 \end{cases} \tag{2.3}$$

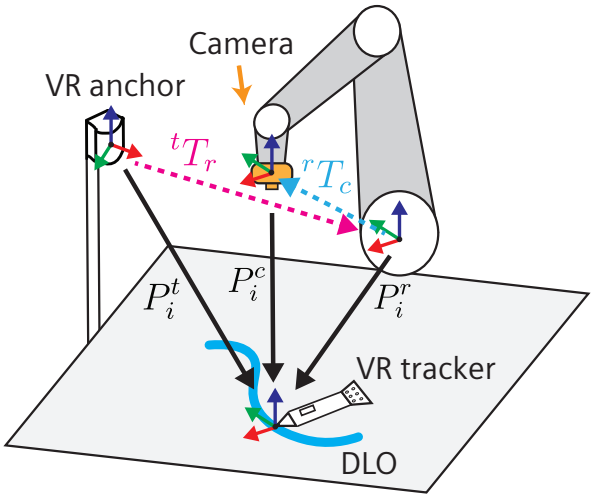


Figure 2.5: Transformations involved in the labeling procedure using the VR tracker. The transformations named ^tT_r, ^rT_c are needed to transform the labeled point from the VR coordinates (P_i^t) to points in the robot coordinates (P_i^r) and in the camera coordinates (P_i^c).

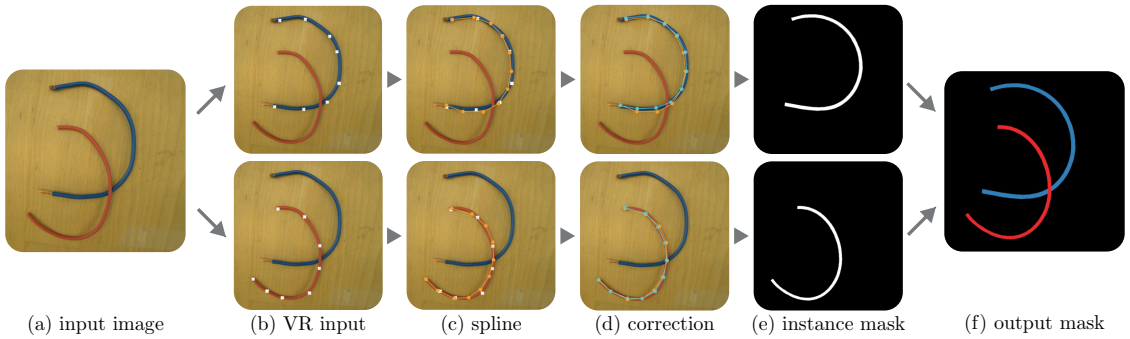
2.5.2 VR Tracker Labeling

A methodology involving a sensor tracked in space is selected to label instances of DLOs. Therefore, an input methodology similar to [29] is preferred. However, an alternative tracking technology is adopted to eliminate the need for an additional camera to determine the position of the tracked sensor, while also offering an industrial solution. More precisely, the Tracepen Virtual Reality (VR) pen is selected. This VR pen works on the basis of reflective photodiode sensors which, by receiving and mirroring an infrared signal, enable the calculation of the sensor's pose from the emitting station. Due to this working principle, the coordinates of the VR pen are expressed in reference to the emitting station (P_i^t) [85]. However, to obtain DLOs instance labels for the images, such positions had to be transformed in camera pixel coordinates (P_i^c). To solve this issue, homogeneous transformations between the emitting station and the camera position had to be considered as shown in Fig. 2.5. Hence, the approach of [143] is used and the transformation between the emitting station and the robot (^tT_r) is calculated to obtain VR tracker points in the robot coordinate frame (P_i^r).

Subsequently, these 3D points can be projected onto the 2D images to create training data. This projection is accomplished using eq. (2.4), where u and v represent the pixel coordinates, w' is the scaling factor, K corresponds to the intrinsic camera matrix obtained via camera calibration, ^rT_c denotes the transformation from the robot to the camera, and $[x, y, z]$ represents the 3D point that requires projection. This projection is achieved by calculating $u = u'/w'$ and $v = v'/w'$.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = K\, {}^rT_c \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.4}$$

Consequently, by employing this approach, images of real-world scenarios captured from various camera positions are obtained using a single labeled input. This process is iterated for each DLO instance requiring labeling.



(a) input image (b) VR input (c) spline (d) correction (e) instance mask (f) output mask

Figure 2.6: Labeling flow performed for each captured image. On the sampled image (a) the input points of the VR tracker captured for each instance are projected (b) and smoothed with a spline curve (c). Thereafter, the CNN-based correction procedure is executed (d), the instance masks generated (e), and the output mask finalized (f). Colors meaning: input tracker points as white crosses, smoothed points in orange and corrected points in cyan.

2.5.3 Weakly Supervised Semi-automatic Labeling

Unlike synthetic labeling, which is inherently error-free, during the labeling of real-world DLOs performed by a human operator, some level of error is expected. In particular, the major sources of errors are due to inaccuracies in 1) the calibration of the annotation tool and/or eye-in-hand camera; and 2) the labeling performed by the human operator. The presence of errors is more evident and severe, especially on very thin DLOs.

To overcome these problems, a fine-tuning step is applied after the human input to each labeled DLO instance. The main stages are shown in Fig. 2.6. First, the labeling points of one DLO instance are smoothed employing an approximating spline curve in the 2D pixel space, similar to the definition of eq. (2.2), see Fig. 2.6(c).

Then, an approach based on a Convolutional Neural Network (CNN) is applied to compute a correction offset for each labeled point. Given the source image I, the vertically oriented crop extracted around the i-th labeled point and having a size of $s \times s$ pixels is denoted as $\hat{I}_i$. With vertically oriented, the condition with the DLO having an almost vertical shape in $\hat{I}_i$ is denoted, see input crop in Fig. 2.7. Thus, the CNN-based network $H(\cdot)$ performs the following operation:

$$h = H(\hat{I}_i)$$

being $h \in \mathbb{R}^s$ the vector approximating the location of the DLO in the image along the horizontal axis, see the output in Fig. 2.7. In other words, h describes the probability of each image column, i.e. column 0 to column $s-1$, of corresponding to the center-line of the DLO in $\hat{I}_i$. Finally, the maximum of h is obtained as:

$$k = \text{argmax}(h) \;:\; \dot{h}_k = 0$$

being $\dot{h}_k$ the derivative of h evaluated at point k. Hence, the correcting offset δ for the crop $\hat{I}_i$ is computed as $\delta = k - s/2$, being s the crop size fixed to be 96 in the following.

The CNN structure is composed of a feature extractor, i.e. ResNet-18 [44], and two Fully Connected linear layers, i.e. $FC_{512,256}$ and $FC_{256,96}$. Binary cross-entropy is used as the loss function during the training stage to optimize the network weights.

The dataset for the training of this network is obtained from the synthetic samples of Sec. 2.4. Thus, crops of 96×96 pixels are randomly extracted from the synthetic images,

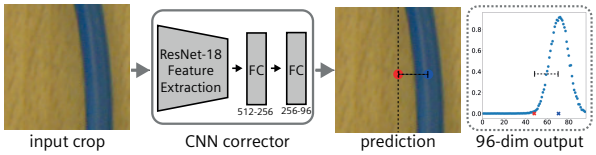


Figure 2.7: CNN-based corrector predicting the approximated DLO center.

given the available spline description, employing a fictitious offset to simulate the user error. Hence, the offset is converted with a Gaussian distribution centered at the offset value and with a variance of 8 pixels. In total, 40000 crops are used for the optimization with the typical 90-10 split in training and validation sets. The network is optimized for 50 epochs, employing a batch size of 128 and a learning rate of 5×10^{-5}. Adam is selected as optimizer with the final network weights chosen based on the validation loss. Having corrected the points, Fig. 2.6(d), the knowledge of the DLO thickness is exploited to construct a polygon that precisely follows the contour of the targeted DLO in each image plane. Thereafter, from the polygon, an instance mask can be easily drawn as shown in Fig. 2.6(e) and the final combined mask can be obtained as shown in Fig. 2.6(f).

2.6 Test Dataset

To evaluate and compare the performances on real data of the deep learning segmentation models trained with the datasets generated according to the proposed methods, a *test set* is constructed. The test set is composed of 135 real images of electrical wires with varying diameters and grouped into 3 categories, each consisting of 45 images defining a specific scenario, labeled as *C1*, *C2* and *C3*. The *test set* was originally deployed in [20] and extended in [17].

The image categories are defined as follows:

C1: scenes with only the target wires lying on a surface and no other disturbing objects. The difficulties in these scenes are the high contrast shadows of the wires, possible chroma similarities between the wires and the background, the dense crosses of wires, the light settings and the perspective distortions.

C2: scenes with the target wires on a highly-featured and complex background and no other disturbing objects. Here the challenge is to extract the wires correctly in a cluttered scene.

C3: scenes with the target wires in a realistic setting as an industrial one (e.g. an electric panel). These can be considered as an example of an application setting, where the difficulties may be given by the metallic surface reflecting the wires and other disturbing objects like commercial electromechanical components characteristic of these panels.

Each category is further divided into sub-classes based on the number of intersections between the DLOs present in the images, i.e. 1, 2, and 3, with 15 samples each. In Fig. 2.8 some samples of the *test set* are shown.

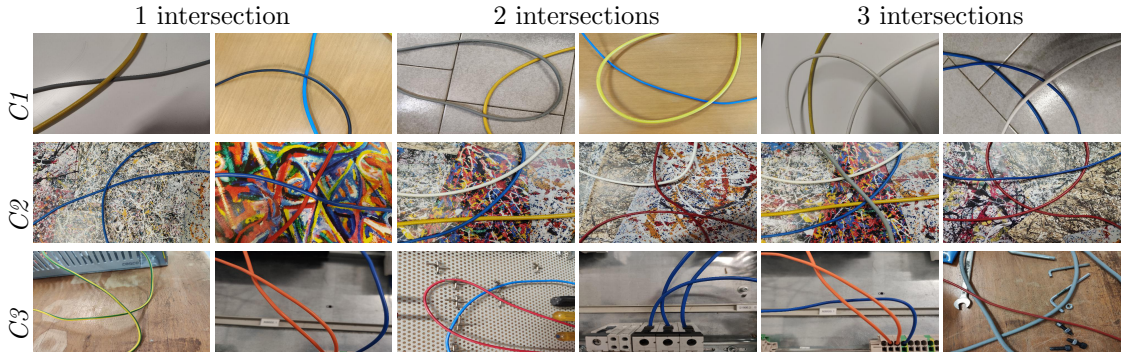


Figure 2.8: Samples of the *test set* organized by category and number of intersections.

2.7 Experiments

The dataset generation methods proposed in this chapter are evaluated from different perspectives in this section. In particular, the dataset obtained exploiting Sec. 2.3 and Sec. 2.4 are validated on the task of semantic segmentation of DLOs. The focus is on the segmentation of electric wires and cables, which is a fundamental task in the industrial automation domain. This evaluation is presented in Sec. 2.7.1. The dataset obtained with Sec. 2.5 is instead used to validate a dataset mixture scheme in which synthetic and real images are mixed to train a segmentation model. Indeed, the weakly supervised method of Sec. 2.5 is not suitable for generating an enormous amount of data. However, it can be used to generate a small dataset of real images that can be mixed with a larger synthetic dataset to train a segmentation model. Also, the mixture scheme is validated on the task of semantic segmentation of DLOs and the results are presented in Sec. 2.7.2. The experiments are performed on the *test set* introduced in Sec. 2.6. A video showcasing *DLO-WSL* is available as supplementary material[1].

Finally, in Sec. 2.7.3 the usability of the method of Sec. 2.5 is evaluated by comparing the annotation time with the one required by baseline methods, including the one of Sec. 2.3.

2.7.1 DLOs Semantic Segmentation

Two different datasets have been generated, one employing the CK method of Sec. 2.3 and a fully synthetic one according to Sec. 2.4. These datasets serve as training data for a deep learning semantic segmentation network. Subsequently, the resulting models are assessed using the *test set* introduced in Sec. 2.6.

Datasets Details

The strategy presented in Sec. 2.3.1 has been employed for generating a dataset of about 29000 RGB images, with a resolution of 720×1280 pixels. This dataset constitutes the *chroma-key dataset*. The dataset is obtained starting from 3176 images featuring blue, red, yellow, white and black wires, with different light setups and shapes. To improve the background and wire separation, besides the hue, also the saturation and value channels are augmented. For each raw image, a new background image is randomly picked among the pool of 15 shown in Fig. 2.2 and 8 new synthetic images are created, as visible in Fig. 2.1. In each new image, the foreground and background are separately

[1] https://www.youtube.com/watch?v=5F7tf9swhvw

augmented (by using the mask) before the merging is performed. In particular, the background is randomly flipped, shifted, scaled and rotated. Then, it is processed with motion blur and elastic transformation ($p = 0.2$), and it is randomly cropped at the dataset resolution. The foreground, instead, is transformed only by shuffling the channels ($p = 0.5$), converting to grey ($p = 0.1$) and randomizing the hue in the range of $[-100, 100]$ ($p = 0.5$). The *chroma-key dataset* dataset contains synthetic images obtained by *chroma-key* overlay. However, the reality gap in the resulting dataset is considerably small compared to those that might be obtained from rendering or simulation. In fact, the main visual discrepancy between real and output images is the object's contour, which has already been smoothed with the approach described in Sec. 2.3.2. To further reduce the reality gap, the input images with the original background are also added to the dataset.

Moreover, a dataset of synthetically generated wires is built by randomizing the shapes, radius, color, stripes, backgrounds and lights as presented in Sec. 2.4. Overall, a total of about $32,000$ images were rendered constituting the *synthetic dataset*.

Semantic Segmentation Network

The deep learning NN exploited to perform the training and testing needed to estimate the datasets' performances is DeeplabV3+ [23], a popular encoder-decoder architecture with proven performances especially in decoding precise object boundaries. As encoder, the original implementation of *DeepLabV3+* employs a modified *ResNet* [44] backbone with atrous convolutions, instead of the common convolutions, allowing the explicit control of the computed features resolution via the output stride parameter. In this section, a comparison also exploiting other state-of-the-art backbone architectures is provided. The selected backbones are *Swin-Transformer* [70] and *ConvNeXt* [71]. In summary, the following backbones are employed: *ResNet-50*, *ResNet-101*, *Swin-Transformer-T*, *Swin-Transformer-S*, *ConvNeXt-T* and *ConvNeXt-S*. The other backbones of *Swin-Transformer* and *ConvNeXt* are selected to match the model complexity of the *ResNet* ones.

The decoder consists of a simple yet effective module that refines the segmentation results along object boundaries. Here, the low-level features are concatenated to the bilinearly upsampled (4x) high-level features coming from the encoder. Several convolutions are performed to refine the features and a final upsampling (4x) is performed. This design choice of the decoder, compared to a direct bilinear 16x upsampling, provides improved performances [23]. The final output is a probability map of the same size as the input image, where each pixel is associated with a probability of belonging to the DLO semantic class or the background.

Training Procedure

The semantic segmentation models are trained with the following hyperparameters: batch size 4, Adam optimizer with learning rate 10^{-6} and polynomial learning rate adjustment policy to a minimum of 10^{-8}, with power 0.95. A warmup procedure is executed for the first 1000 steps with an initial learning rate of 10^{-8}. The *ResNet* backbones are trained with an output stride of 16 and separable convolutions. The training is executed for a maximum of 300.000 steps and the best weights are saved according to the validation loss. The models are implemented in PyTorch 1.10.0 and trained with an NVIDIA GeForce GTX 2080 Ti on an Intel Core i9-9900K CPU clocked at 3.60GHz. The training dataset is obtained from 90% of the original one, while the validation is done on the remaining 10%. The data augmentation scheme includes hue randomization, channel shuffling, flipping and finally resizing to 360×640 pixels.

Table 2.1: The average Dice Coefficient computed across the *test set* images on the models trained with the different datasets. In all the tests the predictions are thresholded at 0.3.

Backbone Family	Backbone Type	*chroma-key dataset*				*synthetic dataset*			
		C1	*C2*	*C3*	Tot.	*C1*	*C2*	*C3*	Tot.
ResNet [44]	ResNet50	78.946	68.777	81.372	76.365	79.562	75.241	76.383	77.062
	ResNet101	82.955	72.207	85.269	80.144	84.105	66.798	84.880	78.594
Swin Transformer [70]	SwinT	83.832	85.028	83.454	84.104	85.047	82.510	82.803	83.453
	SwinS	86.103	87.579	87.490	87.058	86.830	85.951	84.683	85.821
ConvNext [71]	ConvNextT	85.784	87.856	83.793	85.811	85.327	87.368	84.265	85.650
	ConvNextS	86.426	88.524	84.111	86.354	85.085	87.598	84.288	85.657

Evaluation Metric

The evaluation of the results is performed with the Dice coefficient by comparing the predicted mask with the ground truth one of the labeled *test set*. The Dice coefficient is defined as $\text{Dice} = 2\frac{|M_p \cap M_{gt}|}{|M_p|+|M_{gt}|}$ where M_p and M_{gt} are the predicted and ground truth masks, respectively. The Dice coefficient is computed for each image of the *test set* and then averaged across the images of each category and for the total.

Results Discussion

In Tab. 2.1 the average Dice obtained in the test dataset by DeepLabV3+ trained on the *chroma-key dataset* and the *synthetic dataset* and with the different set of backbones are reported. The results are shown for each category and the total across all the samples.

From the values reported in Tab. 2.1, the *synthetic dataset* exhibits a slight drop in performances compared to the *chroma-key dataset*. This is probably due to the reality gap between the synthetic and real images, being larger for the fully synthetic dataset of Sec. 2.4. However, the results are still comparable and the *synthetic dataset* can be considered a valid alternative to the *chroma-key dataset* for training the semantic segmentation models. Indeed, looking at the qualitative samples shown in Fig. 2.9, the difference between the two datasets is barely noticeable, with some predicted masks being better for one dataset and some for the other. To better compare the two datasets, in Fig. 2.10 the boxplot of the Dice Coefficient computed across the *test set* images on the models trained with the different datasets is shown. The boxplot shows that the *synthetic dataset* has a slightly lower median and a slightly larger interquartile range, but the difference is not significant. Concerning the backbone choice, the *SwinS* transformer seems to be the best choice for both datasets.

2.7.2 Synthetic and Real Samples Dataset Mixture

In this section, the impact of the dataset mixture of synthetic and real samples on the training of data-driven segmentation algorithms for DLOs is investigated. The mixture is obtained by combining the dataset generated in Sec. 2.4 with the real-world dataset of Sec. 2.5. The goal is to understand with which measure the inclusion of real-world data in the synthetic dataset, through *DLO-WSL*, helps the training of data-driven methods. The dataset mixture is obtained by combining the two datasets with different ratios, specifically:

$$F(x) = w_r P_r(x) + w_s P_s(x) \tag{2.5}$$

where, w_r and w_s are the ratios between real-world and synthetic datasets with $w_r = 1 - w_s$, $P_r(x)$ and $P_s(x)$ representing the distributions of the real-world and synthetic dataset respectively, and $F(x)$ is the resulting distribution. Therefore, during training, samples extracted from $F(x)$ are used to optimize the learning models. For

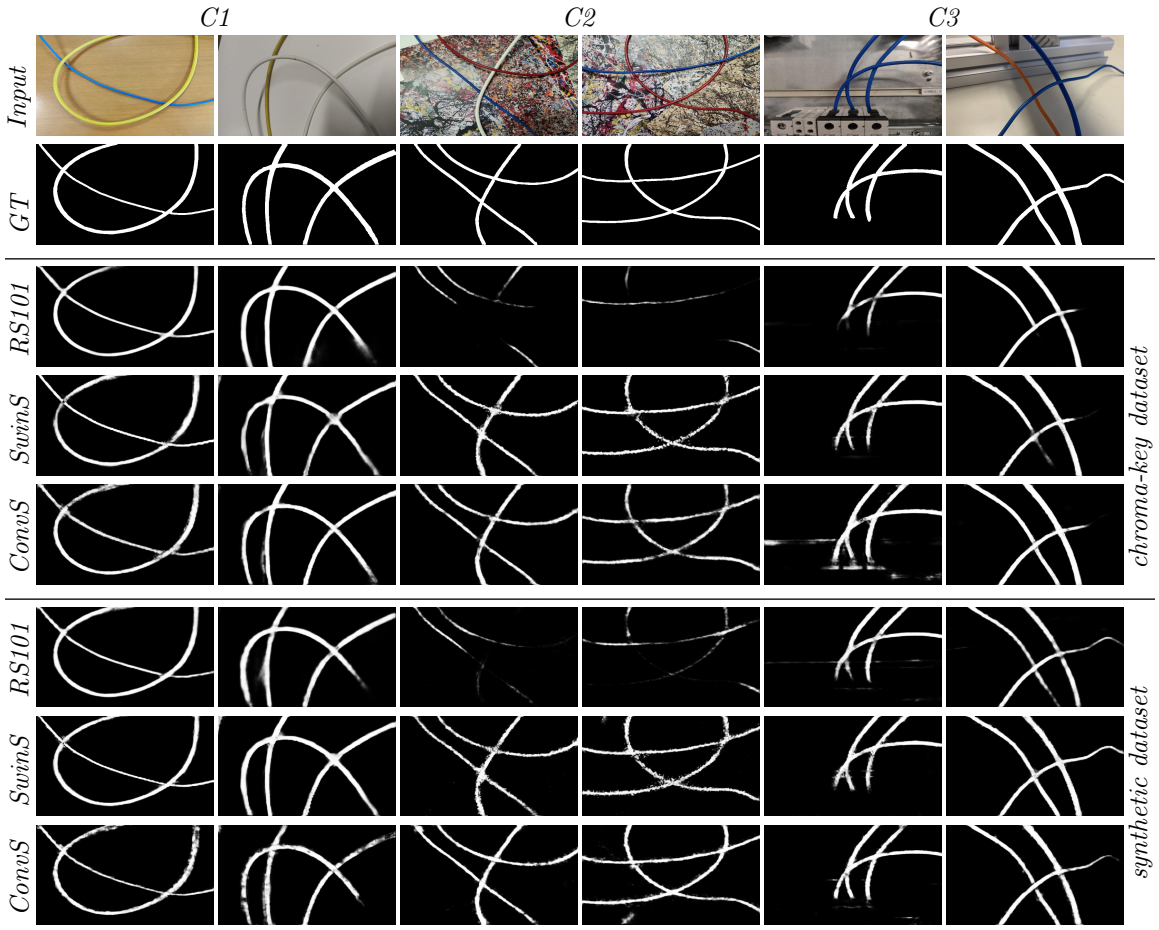


Figure 2.9: Qualitative results of the semantic segmentation task on the *test set*.

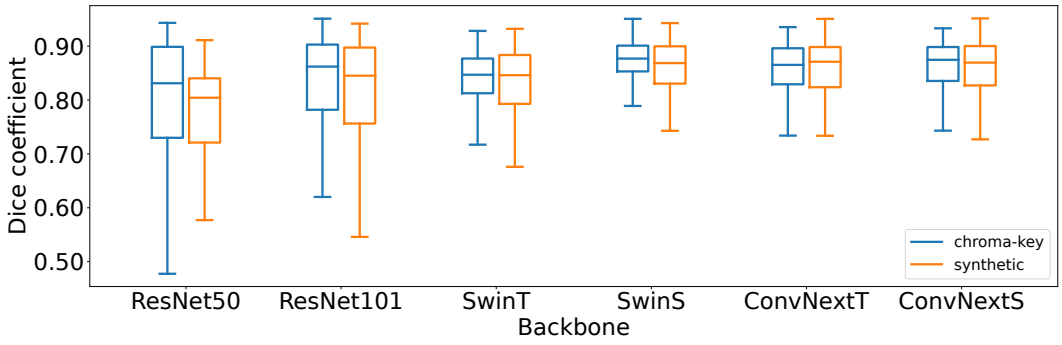


Figure 2.10: Boxplot of the Dice Coefficient computed across the *test set* images on the models trained with the different datasets. In all the tests the predictions are thresholded at 0.3.

the investigation of w_s and w_r, a full factorial experiment with the datasets as factors on three levels with three runs has been conceived. Specifically, the experiments with the respective w_r are shown in Tab. 2.2.

Table 2.2: Factorial design for the analysis of dataset mixtures. The table shows the size of real-world dataset given the synthetic dataset and ratio w_r. With † are denoted the configurations where U^2PL is trained also in a semi-supervised fashion. With * are denoted the configurations used only by *DeepLabV3+* and *YOLACT*.

Synthetic dataset size	Real-world dataset size given ratio w_r				
	0	0.05	0.11	0.20	0.33
4000	0	-	-	$1000^\dagger$	$2000^\dagger$
8000	0	-	$1000^\dagger$	$2000^\dagger$	-
16000	0*	1000*	2000*	-	-

Experiments Settings

For the semantic segmentation task, *DeepLabV3+*[23] and U^2PL[123] are used as networks for the tests. The former is introduced in Sec. 2.7.1. The latter is a recently proposed semi-supervised method. The usual setting of semi-supervised learning consists of: (i) a small or moderately sized 'labeled' dataset, and (ii) a much larger 'unlabeled' dataset. The goal is to leverage the unlabeled data to improve the performance of the model trained on the labeled data. Typically, the unlabeled data is used to generate pseudo-labels, which are then employed during the model optimization. Obviously, the quality of the pseudo-labels is far from that of the labeled data. However, the unlabeled data has two main advantages: 1) it can be easily collected in large numbers since labeling is avoided, and 2) it is usually closer in terms of domain to the one in which the model is expected to be deployed. For the experimental evaluation proposed in this section, the real-world dataset is used both as labeled (where the labels come from the proposed approach) and unlabeled. This approach allows for the evaluation of the benefits of labeling real-world data as opposed to a semi-supervised approach.

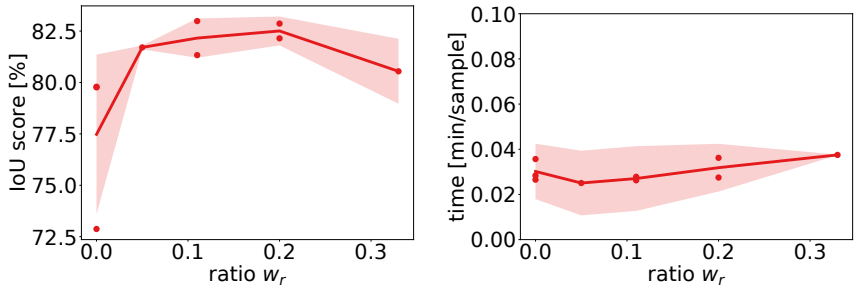
DeepLabV3+ is trained with a ResNet-101 [44] backbone with a batch size of 10, output stride of 16, separable convolutions, a polynomial learning rate of 10^{-5} with power 0.95, and Adam optimizer. Instead, U^2PL is trained with a ResNet-50 [44] backbone with a batch size of 2, a polynomial learning rate of 5×10^{-5} with power 0.9, and stochastic gradient descent (SGD) as optimizer. Both networks are optimized for 50 epochs with the final weights selected based on the validation loss. As augmentation scheme, the following is employed: channel shuffling; hue, saturation and value randomization; flipping; perspective distortions; random cropping; random brightness and contrast.

With reference to Tab. 2.2, for *DeepLabV3+* the training mixtures consist of a total of 9 configurations. For U^2PL, due to computation burden, the synthetic 16K configuration is avoided thus resulting in a total of 10 configurations accounting also the semi-supervised condition.

As evaluation metric, the IoU is used. The IoU is defined as $\frac{|M \cap M_{gt}|}{|M|+|M_{gt}|}$, where M is the predicted binary mask for the semantic segmentation task and M_{gt} is the ground truth.

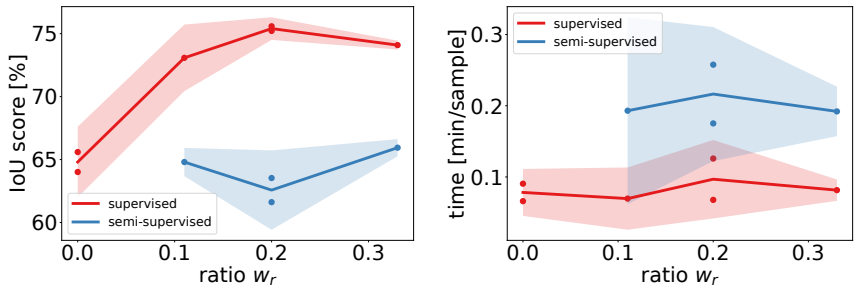
Discussion

In Fig. 2.11 results of the semantic segmentation task employing *DeepLabV3+* are shown. From the plot of Fig. 2.11a, it is evident that the introduction of real images helps in achieving higher accuracy in general. For instance, employing a total of 5K samples (4K synthetic and 1K real, $w_r = 0.20$) an IoU of M=82.83%, SD=0.54 is obtained compared to M=74.97%, SD=2.78 for the 4K synthetic only dataset (statistically significant $p < 0.05, CI = 95\%$). Similarly, when employing 1K real images on the 8K synthetic



(a) IoU score (b) time

Figure 2.11: Evaluation of w_r on *DeepLabV3+*. To the left is the change of average IoU score with different w_r. To the right is the change of training time over dataset size.



(a) IoU score (b) time

Figure 2.12: Evaluation of w_r on U^2PL with the comparison between supervised (red) and semi-supervised (blue) training.

dataset ($w_r = 0.11$), an overall higher IoU of M=83.93%, SD=0.17 is reached compared to M=81.60%, SD=0.42 for the 8K synthetic only dataset (statistically significant $p < 0.05, CI = 95\%$). For comparison, an IoU score of 81.9% is obtained on the same *test set* when employing the dataset from Sec. 2.3 for training. Concerning Fig. 2.11b, the training time shows a slight rise as w_r increases, as expected due to the growth of the overall dataset size.

In Fig. 2.13 a qualitative analysis of the results from *DeepLabV3+* is performed. Visually, both synthetic and real samples bring important benefits. The first ones permit an accurate segmentation of the DLO thanks to the availability of high-quality error-free ground truth labels. The latter allows, during the training stages, to recover the DLOs texture and appearance real information that is difficult to capture in synthetic renderings. Indeed, in Fig. 2.13, the DLOs with complex textures and lights are better handled once the real samples are employed (e.g., yellow DLO sample 1, blue DLO sample 2). The contribution of increasing the size of the synthetic dataset is beneficial but less significant.

The evaluation of the segmentation and time performances achieved by U^2PL is instead shown in Fig. 2.12. Focusing on Fig. 2.12a and comparing the scores for the same values of w_r, it is possible to observe how the semi-supervised training brings only very marginal benefits compared to the only synthetic fully-supervised one. Instead, the utilization of labels obtained exploiting *DLO-WSL* for the real samples boosts the IoU performances by about 10% with the highest result for $w_r = 0.20$. The timings of Fig. 2.12b depict how the semi-supervised training approach is much slower due to the

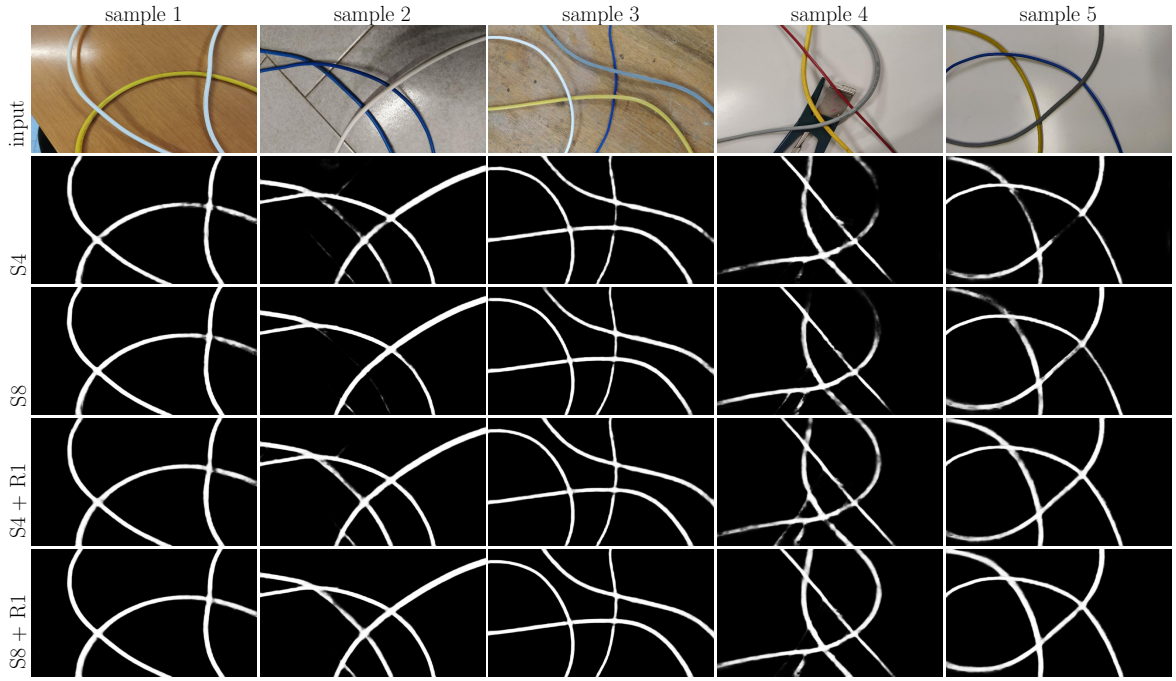


Figure 2.13: Qualitative *test set* samples and predictions of *DeepLabV3+* when trained on 4K synthetic images (S4), 8K synthetic images (S8), a mix of 4K synthetic and 1K real (S4 + R1, $w_r = 0.20$) and a mix of 8K synthetic and 1K real (S8 + R1, $w_r = 0.11$).

need to construct pseudo-labels at run-time.

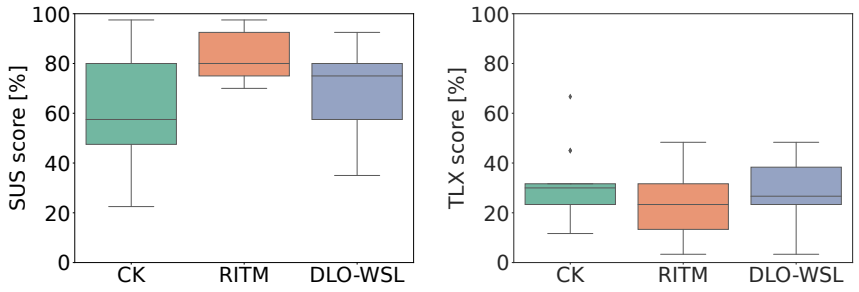
In conclusion, the results of the experiments show that the introduction of real samples in the synthetic dataset is beneficial for the training of data-driven segmentation algorithms for DLOs. In particular, the results show that the best results are obtained when the real samples are about 20% of the total dataset. Additionally, w_r seems to not drastically change the training time if differences between semi-supervised and supervised training are not considered.

2.7.3 *DLO-WSL* Perceived Usability and Performances

The goal of this section is to evaluate the perceived usability, required effort and labeling accuracy of the *DLO-WSL* method discussed in Sec. 2.5. In this regard, a user test with a balanced randomized order of three subsequent interactions was envisioned.

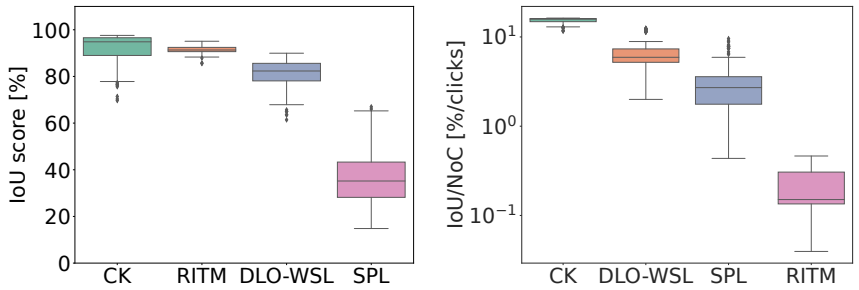
Concerning labeling methods for big datasets with uneven backgrounds and requiring annotation just for one image of the set, there is nothing in the literature to compare with. Therefore, the comparison is done against the *chroma-key* technique of Sec. 2.3 for its adequacy to generate multi-image datasets with single human intervention in even backgrounds, and *RITM* [108] due to its good performance in state-of-the-art single image weakly-supervised labeling. For these comparisons, the users were requested to label 10 images with three different methods. At the end of each interaction, usability and workload were measured through the System Usability Scale (SUS) [10] and the NASA-TLX [43]. Additionally, the number of clicks (NoC) to complete the labeling task was also recorded.

A total of 13 users, not experienced with labeling techniques, age mean (M) = 32.70 yrs, standard deviation (SD) = 9.23, participated in the study. All of them performed



(a) usability (b) effort

Figure 2.14: In (a) sability measure with the SUS scale [10], the higher the better. In (b) workload measure with the NASA TLX [43], the lower the better. The comparison is established between chroma-key (*CK*), *RITM* [108] and *DLO-WSL*.



(a) segmentation (b) segmentation and number of clicks

Figure 2.15: Evaluation of the performances in terms of semantic segmentation and number of clicks for the labeling approaches. The segmentation reports the outcomes with chroma-key (*CK*), *RITM* [108], *DLO-WSL* and raw data from the user (without the CNN correction) fitted with a spline (*SPL*).

the test correctly and no data were discarded.

Concerning the usability, the score for *chroma-key* is M=60.38%, SD=21.00, for *DLO-WSL* is M=69.61%, SD=16.26, and for *RITM* is M=82.30%, SD=9.54. A Mann-Whitney-U-Test is applied for statistical difference as long the normality pre-condition did not hold true $p > 0.05$ (CI=95%). The test reported $p > 0.05$ (CI=95%) when comparing *DLO-WSL* with the other methods. Fig. 2.14a shows the results of the SUS scores. Therefore, it is possible to conclude that the usability of *DLO-WSL* is good and comparable to *chroma-key* and *RITM*.

Regarding perceived effort, the score for *chroma-key* is M=30.51%, SD=15.08, for *DLO-WSL* is M=29.74%, SD=12.96, and for *RITM* is M=22.31%, SD=13.12. A Mann-Whitney-U-Test is applied for statistical difference as long the normality pre-condition did not hold true $p > 0.05$ (CI=95%). Fig. 2.14b shows the results of the NASA-TLX scores. The test reported $p > 0.05$ (CI=95%), therefore it is possible to conclude that the workload perceived in using *DLO-WSL* is comparable to the other methods.

Finally, to examine the performances of labeling, the average Intersection over Union (IoU) and IoU over the average Number of Clicks (NoC) for the dataset of 10 images are used. The IoU is defined as $\frac{|M \cap M_{gt}|}{|M|+|M_{gt}|}$, where M is the mask generated with a specific method and M_{gt} is the ground truth. The results are shown in Fig. 2.15. To further analyze the differences across the distributions, a pair-wise Mann-Whitney-U-Test is conducted due to the invalidity of the homogeneity of variance pre-condition. The

test reported $p < 0.05$ (CI=95%) when comparing *DLO-WSL* with the other methods. More precisely, IoU scores were M=91.68%, SD=6.56, M=91.32%, SD=1.52, M=81.05%, SD=6.22 and M=36.88%, SD=12.45 for *chroma-key*, *RITM*, *DLO-WSL* and *spline* respectively. IoU/NoC scores were M=15.31%/clicks, SD=1.09, M=6.54%, SD=2.78, M=3.16%/clicks, SD=2.05 and M=0.21%/clicks, SD=0.13 for *chroma-key*, *DLO-WSL*, *spline* and *RITM* respectively. Thus, *DLO-WSL* obtains a good average IoU while minimizing the number of clicks for uneven backgrounds.

Discussion

As shown from the results, the usability and the workload of *DLO-WSL* are comparable with the *chroma-key* and *RITM* approaches. However, *DLO-WSL* enables users to label more images with good average IoU while requiring fewer clicks in uneven backgrounds as shown by the IoU/NOC metric, thus lowering the overall effort.

Despite these results, *DLO-WSL* showed worse statistically significant performance in the average IoU of the labels, however still better than the *spline* approach. Therefore, some pitfalls might have been in the CNN fine-tuning step proposed in Sec. 2.5.3. To better evaluate its specific performances, a synthetic test set is generated. This set consisted of 1000 randomly cropped image regions obtained according to Sec. 2.4 where a random horizontal shift of the DLO is introduced to simulate user error. Afterward, this set was used to evaluate the CNN corrector by monitoring the offset prediction error computed as $\delta = O_p - O_{gt}$, where O_p is the offset predicted and O_{gt} is the ground truth. Out of this test, the resulting error δ can be approximated by a normal distribution with M=0.05 pixels, SD=1.57. Thus, considering that the test DLOs have diameters in the range of 10-20 pixels wide, one source of error resulting in a non-optimal IoU is due to the not-perfect matching of the DLOs edges by the CNN fine-tuning. Therefore, future work should address this issue and improve the pixel-wise labeling of the DLOs around the contour of the instances.

2.8 Conclusions

Three methods to generate training samples of DLOs with zero or minimal human intervention have been presented in this chapter. The first method exploits the chroma separation principle between background and foreground to generate labeled images of DLOs. The second method generates synthetic images of DLOs with varying shapes and colors by exploiting a photo-realistic rendering engine. Finally, the third method collects real images of DLOs and performs the annotation process with a weak supervision approach.

The generated datasets have been used to train several segmentation models and the results have been compared against real-world complex test images. The results show that the proposed methods are able to generate datasets of DLOs that can be used to train segmentation models that generalize well on real-world images.

Moreover, the chroma separation method and the weakly supervised annotation approach have been compared in terms of usability, effort and labeling accuracy. The results show that the weakly supervised labeling approach is as usable as the *chroma-key* method but drastically reduces the number of necessary clicks for uneven backgrounds.

Finally, the weakly supervised annotation approach has been used to generate a small dataset of real images that have been mixed with a larger synthetic dataset to train a segmentation model. The results show that the mixture scheme is able to improve the segmentation accuracy of the model trained only on synthetic data by introducing not modeled specific texture and appearance information in the dataset.

Despite the positive results, several points that can be further studied in future work are identified. In particular, the chroma separation method could be improved by introducing more sophisticated merging techniques to better handle the foreground-background high-frequency separation artifacts. Moreover, the synthetic-to-real domain gap of the segmentation models trained on synthetic data could be investigated more in-depth and the application of domain adaptation techniques could be studied. Concerning the weakly supervised annotation approach, the proposed CNN fine-tuning method used to correct the noisy user input in Sec. 2.5 could be improved in terms of annotation precision possibly exploiting the already corrected points to robustify the system. Finally, the mixture scheme presented in Sec. 2.7.2 could be further studied to better understand the impact of the real data on the segmentation accuracy.

Chapter 3

2D Perception: Instance Segmentation and Modeling

Segmenting DLOs instance-wise in images allows the extraction of key information that can be used for manipulation tasks. However, the instance segmentation of DLOs is a challenging task since DLOs are characterized by a high variability in shape and appearance, and it is often challenging to disentangle them from the background. Moreover, the lack of publicly available datasets and the difficulty of annotating DLOs instance-wise make the application of deep learning-based methods difficult. Therefore, novel approaches for instance segmentation of DLOs are developed.

3.1 Introduction

The term 2D perception is usually referred to the process of extracting information from an image. In robotics, 2D perception is one of the tools used to understand the environment surrounding the robot and consequently to plan the correct robot's actions. This chapter focuses on the 2D perception of DLOs, i.e. on the processing of images of the scene containing an unknown and variable number of DLOs. The overall goal is to extract information about the DLOs in the scene, possibly their configuration (or shape), and to distinguish them from the background and other objects. This information can be used to plan the robot's actions, e.g. where to grasp the DLOs, how to perform the manipulation or how to avoid other DLOs.

Among the several tasks of 2D perception, the instance segmentation task is the one that best fits the problem of DLOs perception. Instance segmentation is a challenging problem in computer vision, which consists of detecting and segmenting each instance of a class in the scene [81]. In practice, starting from the input image, an output mask is generated where each pixel is assigned to a specific instance of a semantic class. This is the general case for images with DLOs (i.e. one semantic class), where the goal is to segment each DLO from the background (another semantic class) and from each other (same semantic class but different instances). The instance segmentation task is usually addressed by means of deep learning-based methods, which are able to achieve state-of-the-art performances in common benchmarks [81]. However, applying these methods to DLOs is challenging due to the difficulty of DLO perception (as discussed in Chap. 1) and the dataset issue, particularly with regards to datasets availability and instance-wise annotation difficulties (as outlined in Chap. 2).

In this chapter, three algorithms for the instance segmentation of DLOs in images are presented. These algorithms are as follows:

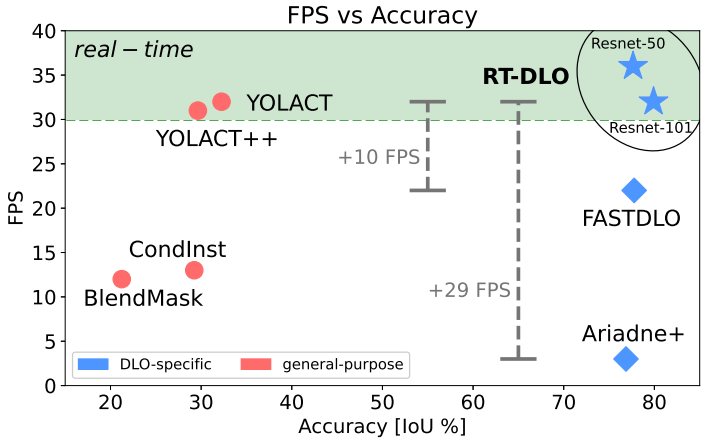


Figure 3.1: FPS vs accuracy of 2D DLO-specific perception algorithms (*Ariadne+*, *FASTDLO* and *RT-DLO*) versus general-purpose baselines methods concerning the instance segmentation task.

- *Ariadne+*: Drawing inspiration from the *Ariadne* algorithm [27], this approach involves first the semantic segmentation of the scene through a deep learning NN, which is then complemented by a superpixels-based over-segmentation to effectively handle DLOs within complex backgrounds. Additionally, color, curvature, and intersection layout predictors are utilized to accurately extract individual DLO instances from the semantic segmentation mask.
- *FASTDLO*: Enhanced version of *Ariadne+* that is capable of performing DLOs instance segmentation at a frame rate exceeding 20 frames per second (FPS). It employs a skeletonization approach to retrieve *sections* from the mask, followed by their proper connection based on a similarity network.
- *RT-DLO*: Enhanced version of *FASTDLO* achieving real-time DLOs instance segmentation performances at a frame rate exceeding 30 FPS. This algorithm utilizes a graph-based representation of DLOs within the scene proving to be both efficient and robust.

The methods and results discussed in this chapter have been published as [20, 17, 15]. Fig. 3.1 illustrates the performances of the proposed algorithms in terms of accuracy and frame rate. The algorithms are compared with several advanced deep-learning NN addressing the instance segmentation task.

The remainder of this chapter is organized as follows. In Sec. 3.2, the related work concerning the problem of 2D detection and segmentation of DLOs is presented. Then, Secs. 3.3, 3.4 and 3.5 describe the proposed *Ariadne+*, *FASTDLO* and *RT-DLO* algorithms. Sec. 3.6 presents the experimental results with cross-comparisons between the methods proposed and other additional baselines. Finally, Sec. 3.7 provides the conclusions of this chapter.

3.2 Related Work

3.2.1 General Problem of DLOs Identification

In the past, the problem of DLO identification has been solved in simple settings: in [54] the authors requires the presence of a single DLO in the scene and its segmentation

is based on a color threshold with a controlled background; in [128] a good contrast between the background and the DLO is again assumed; in [151] a threshold is applied in a controlled background to segment the cable. Also in [122], for instance, the tracking of DLOs is presented where the images are segmented relying on simple color information. In [128], instead, a good contrast between foreground and background is assumed to be available. Alternatively, in [53], the utilization of augmented reality markers for the detection of wiring harness endpoints is proposed.

Indeed, the major difficulties in DLO identification rely on its simplicity, which does not offer distinguishing features to be used for unambiguous detection. Moreover, the knowledge of the number of DLOs in the scene is usually assumed, e.g. in [54].

Although for cables, and DLOs in general, the literature concerning perception algorithms is quite reduced, this is not the case for other domains where the treated objects may have some similarities with cables, such as vessel [82] and suture threads [73] segmentation in medical images. In fact, DLOs appear as tubular structures in images. Hence, algorithms developed for curvilinear structures can find potential applications for DLOs. Regarding the medical domain, in [82] the authors propose a new curvilinear structure segmentation network and show their application to vessel segmentation tasks. In [73] an approach consisting of the identification of the suture thread tips and a novel marching algorithm is presented aiming at segmenting the thread from the background.

Among simpler general "filters" for tubular structures, the most popular one, commonly addressed as *Frangi* filter, is [35], which is a multi-scale procedure able to highlight tubular structures. In addition, a well-known method is the *Ridge* filter [109], which is an algorithm used to extract image ridges, commonly applied to medical images showing vessel structures. Both the *Frangi* and *Ridge* filters are based on the Hessian matrix, hence many false positives can be detected in case of complex backgrounds [27]. Lastly, ELSD [89] is an algorithm developed for detecting line segments and elliptical arcs. Also ELSD suffers in case of complex backgrounds [27].

The assumptions about identification capability and the number of expected instances limit the applicability of the proposed solutions in real-case scenarios where DLOs are commonly involved.

3.2.2 DLOs Segmentation in Complex Images

Ariadne [27] is the most advanced method for cable detection and segmentation in images featuring complex backgrounds. It is able to perform both semantic segmentation and B-Spline modeling for multiple DLOs in the scene. However, it has a few drawbacks, the main ones being: it requires a NN (or a manual intervention) to specify the cables' start and endpoints; it is intrinsically slow due to the exploration process involved; its performances are heavily affected by perspective settings; it is not robust to specific background/foreground color combinations.

Regarding other data-driven approaches, the advancements of deep learning in the last years resulted in several Deep Convolutional Neural Networks (DCNNs) tailored for the general problem of instance segmentation task, e.g. *YOLACT* [6], *YOLACT++* [5], *BlendMask* [22] and *CondInst* [114]. A relevant problem in the application of data-driven approaches for instance segmentation of DLOs resides in the lack of good-quality publicly available datasets and, consequently, the difficulty in annotating a large set of images. However, some approaches have emerged focusing on synthetic data generation pipelines [31, 93] that tackle this problem, as described in Sec. 2.4.

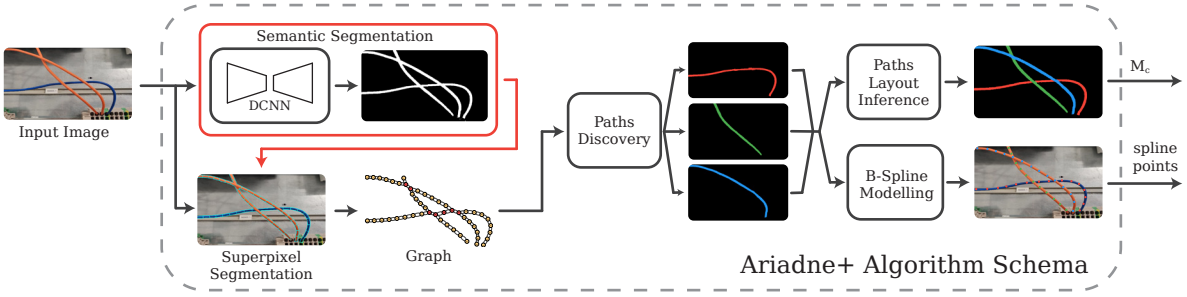


Figure 3.2: The *Ariadne+* algorithm.

3.3 The *Ariadne+* Algorithm

The *Ariadne+* algorithm represents an enhanced approach in comparison to the original *Ariadne* method introduced in [27]. This algorithm takes an input image of a scene and leverages a combination of deep learning and computer vision techniques to identify individual instances of each DLO within the scene.

The process begins with a DCNN performing semantic segmentation on the source image, yielding a binary mask as output. Subsequently, a computer vision pipeline processes both the input image and its binary mask. This pipeline includes a superpixel segmentation step to reduce complexity. The resulting superpixel structure and properties are efficiently managed using a Region Adjacency Graph (RAG). This RAG is subjected to simplification, refinement, and clustering, followed by a procedure to identify the path of each DLO. Ultimately, a task-specific DCNN is employed to assess the arrangement of the DLOs. The identified instances are then modeled using B-Splines, providing a valuable representation for manipulation tasks.

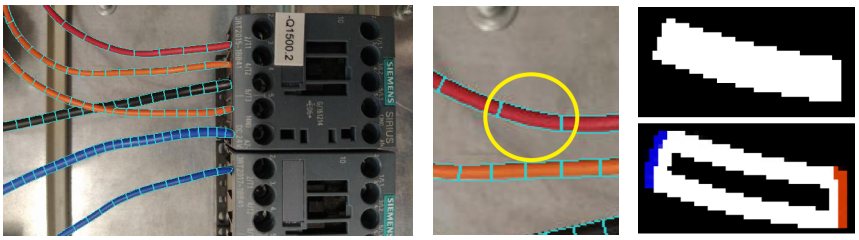
Regarding the enhancements introduced in *Ariadne+*, several key improvements have been made compared to the original *Ariadne* method:

- No Manual Initialization: The need for manual intervention to initialize the algorithm has been eliminated.
- Endpoint Independence: The approach is now capable of functioning even when the endpoints of the objects are not visible in the image.
- Efficiency Optimization: The superpixel segmentation and graph generation steps are no longer applied to the entire image, resulting in a significant reduction in execution time.
- Enhanced Robustness: The introduction of novel NN methods has significantly increased the algorithm's robustness, making it more adaptable to complex and diverse real-world scenarios.

In summary, *Ariadne+* outperforms *Ariadne* in terms of its broader applicability, improved accuracy, and reduced execution time.

In Fig. 3.2 an overview of the algorithm pipeline is provided. The main steps of the *Ariadne+* algorithm are summarized as follows:

1. **Semantic Segmentation**: A DCNN segments the input image and provides a binary mask M_b as output;
2. **Superpixels Segmentation**: The input image is segmented into superpixels taking advantage of M_b;



(a) (b) (c)

Figure 3.3: Superpixelization of an input image containing several DLOs in an industrial setting (a) with a crop of the image in (b). Graph edges computation in (c): superpixel mask before (top) and after (bottom) gradient operation. In (c-bottom) are shown the pixels of the neighboring superpixels in red and blue.

3. **Graph Generation**: A RAG is created to manage superpixels efficiently in terms of neighborhood search and endpoints identifications;
4. **Graph Simplification**: The graph's nodes degree is used to find intersection nodes, simplifying them to a single equivalent node;
5. **Graph Clustering**: The graph is divided into clusters based on a connectivity measure;
6. **Intersection Score Evaluation**: In case intersection nodes are present, the scores of their neighbor's couples are evaluated using a DCNN called *TripletNet*;
7. **Paths Finder**: A paths finding strategy is executed to discover the path of each DLO and maximize the coverage of the graph;
8. **Paths Layout Inference**: A DCNN is used to predict the layout of the DLOs in the intersection areas for creating correct instance masks.
9. **B-Spline Modelling**: The computed paths are approximated by B-Spline curves.

In the following, each of those steps is discussed in detail.

3.3.1 Semantic Segmentation

As starting point of the *Ariadne+* algorithm, a DCNN performs the semantic segmentation of the input image, producing a binary mask M_b of the DLOs. The *DeeplabV3+* DCNN model [23] is chosen for that purpose since it provides reliable semantic segmentation results concerning DLOs as detailed in Sec. 2.7.1. However, it is worth remarking that any model capable of extracting a good-quality mask from the scene can be employed in the pipeline since no specific constraints are introduced in this regard. As dataset, 29.000 samples of DLOs of different shapes and colors obtained exploiting the *chroma-key* method with background swapping (Sec. 2.3) is employed. Details and performances of the dataset are provided in Sec. 2.7.1.

3.3.2 Superpixels Segmentation

The superpixel segmentation of images consists of partitioning them into local meaningful subregions by capturing local similarity among the pixels. This procedure aims to simplify the image and speed up the processing. There exist two main branches of superpixel

segmentation algorithms: graph-based methods [34] and gradient-ascent approaches [120]. In the proposed *Ariadne+* algorithm, the Simple Linear Iterative Clustering (SLIC) segmentation algorithm [1] is adopted. This solution generates superpixels by clustering pixels based on their color and proximity in the image plane using a 5D space. Since a binary mask discerning the DLOs present in the image is available, the algorithm called MaskSlic [51] is employed. It is derived as an extension of the original SLIC and it is able to perform the clustering only within regions of interest. Thus, the region of interest $I_S^{'}$, i.e. the DLOs, on the input image I_S is partitioned into sub-regions denoted by R_i, i.e. the superpixels, such that $I_S^{'} = \cup R_i$ where $i = 1 \dots r$. From the computed superpixel labels, the centroid information is extracted from each superpixel. The superpixel centroids are then used in the last pipeline step as reference points for DLOs' B-Spline interpolation. Fig. 3.3 displays the result of the superpixelization on a sample image displaying several DLOs in an industrial setting.

3.3.3 Graph Generation

In this step, an undirected and non-weighted RAG is built from the image superpixel segmentation, where each superpixel becomes a graph node. Hence, the graph is composed of r nodes, where r is the original number of superpixels. The undirected unweighted RAG is denoted by $G = (V, E)$, where V is the set of nodes (or vertices) v_i, corresponding to each region R_i, and E is the set of edges $e_{j,k}$ such as that $e_{j,k} \in E$ if R_j and R_k are adjacent. With $\text{adj}(v_i)$ the set of nodes adjacent to v_i is denoted. The RAG generation and its usage are made as efficient as possible to make the pipeline's run-time execution faster.

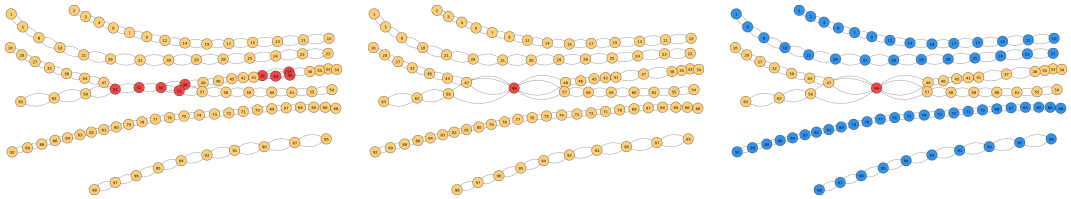
The edges of the graph are, instead, computed by exploiting the neighbors of each superpixel. Given a node of the graph, its superpixel label is retrieved and a binary mask of the superpixel is generated. A morphology gradient operation is performed on the mask to fetch pixels belonging to the neighboring superpixels. Consequently, from the labels of these neighbors, the knowledge of the neighboring relationship of the considered superpixels is obtained. Thus, an edge connection is established between the node considered and each associated neighboring node. Fig. 3.3c shows an outline of the approach on a sample superpixel.

In the following, the degree of a node, denoted by $d(v_i)$, represents the number of its neighbors, and the following classification is adopted for the nodes based on their degree:

- v_i is an outlier if $d(v_i) = 0$ and thus it is removed from the RAG;
- v_i is an endpoint if $d(v_i) = 1$;
- v_i is a segment if $d(v_i) = 2$;
- v_i is an intersection if $d(v_i) > 2$.

Each node is also augmented with the following *attributes*:

- *label ID*: to link each node to the original superpixel;
- *centroid coordinates*: defining the centroid point of the corresponding superpixel in the image;
- *intersection*: 0 (zero) if the node is not an intersection (degree < 2), 1 (one) otherwise.



(a) (b) (c)

Figure 3.4: (a) the graph with intersection nodes highlighted in red, (b) the graph simplified, (c) the graph divided into intersection-free clusters (blue) and clusters with an intersection (yellow).

3.3.4 Graph Simplification

Given the RAG initialized in the previous stage from the point of view of nodes and edges, the simplification of the graph is carried out in this pipeline step by exploiting the degree property of the nodes. An *intersection* is defined as an area of the graph composed of one or more intersection nodes. Exploiting the RAG, the intersections nodes are easily identified and organized, based on neighboring relationships, into groups (*intersections*) characterizing a given area of the image.

The graph is then simplified by replacing each intersection group with a single equivalent node that shares the same neighboring relationships as the original group, e.g. one of all the nodes combined. This simplification makes the path discovery on the graph simpler, in particular in the case of large intersections. Fig. 3.4b shows the result of the simplification applied to the initial situation depicted in Fig. 3.4a.

3.3.5 Graph Clustering

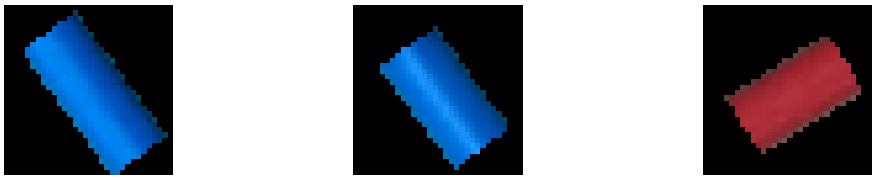
At this stage, RAG nodes appearing to be closer to each other based on some similarity measure are grouped. The similarity measure is usually a topological criterion, e.g. the graph structure. In the proposed method, the graph is clustered based on connected components, i.e. each cluster is connected if a path from any point to any other point of that given cluster exists. The result of the clustering is to split G into clusters (subsets) $\mathcal{C}_i$, $i = 1, \ldots, c$ where c is the number of clusters, such that $G = \cup \mathcal{C}_i$ that can be classified into two main groups:

- *intersection-free* clusters, i.e. not having any intersection node inside, shown in blue in Fig. 3.4c;
- clusters with *intersections*, shown in yellow in Fig. 3.4c with the intersection node marked in red.

Each cluster will be then processed as described in Sec. 3.3.7.

3.3.6 Intersection Score Evaluation

The evaluation of the intersections is performed through a partially learning-based predictor. To this end, the intersection nodes are collected in the set V_{int} and the nodes adjacent to an intersection are grouped into a set called $\mathcal{N} := \{v \in \mathcal{C} : d(v) \leq 2 \wedge \text{adj}(v) \cap V_{\text{int}} \neq \emptyset\}$, with $n_i \in \mathcal{N}$, $i = 1 \ldots m$, being m the number of elements in $\mathcal{N}$ (i.e. $m = \#\mathcal{N}$). Thereafter, a square matrix $\mathcal{W}_{\text{pred}} \in \mathbb{R}^{m \times m}$ is built and initialized to zero. In case no intersection nodes are present in the RAG, this procedure ends immediately and an empty $\mathcal{W}_{\text{pred}}$ is produced as output. Otherwise, $\mathcal{W}_{\text{pred}}$ will contain the *predictions* among each couple of neighbours of each intersection node $v_{\text{int}_j} \in V_{\text{int}}$,



(a) (b) (c)

Figure 3.5: Example of input samples provided to *TripletNet*. The triplet loss is a distance-based loss that operates on three inputs: (a) anchor data; (b) positive data example (similar to the anchor); (c) negative data example.

where the generic element w_{kl} of $\mathcal{W}_{\text{pred}}$ describes the value obtained for the pair of nodes $\{n_k, n_l\}$, where $n_k, n_l \in \mathcal{N}$. If follows that the matrix $\mathcal{W}_{\text{pred}}$ will have a strictly positive value (< 1 as described in the following) only if n_k and n_l are neighbors of the same intersection point v_{int_j}.

TripletNet Score

The values of $\mathcal{W}_{\text{pred}}$ are calculated employing a data-driven procedure, that is preferred with respect to hand-tuned criteria. For this reason, a DCNN called *TripletNet* is developed to perform this prediction. The name of this DCNN is inspired by the fact that the triplet loss criterion [105] is exploited for the required optimization. An example of the input samples of the network is provided in Fig. 3.5.

The structure of *TripletNet* is composed of a feature extractor (*ResNet18*) and a fully connected layer ($FC_{512,256}$) with an input dimension of 512 and outputting the embedding representation with a dimension of 256. The 18-layer version of *ResNet* [44] is selected as a good trade-off between model complexity and quality of the result.

At inference time, the patches x^{n_k} and x^{n_l} are obtained from input nodes n_k and n_l respectively by performing two crops of size 32×32 in the source image around each node centroid.

TripletNet computes the distance between each pair of patches, as $d = \|f(x^n) - f(x^{n_l})\|_2^2$. Then, a sigmoid activation function is used to translate the distance into a probability-like value constrained between 0 and 1, where probability 1 is associated with zero distance. Thus, s_{kl}^t denotes the score associated with the pairs of nodes n_k and n_l. In this way a score is assigned to each prediction coming from the network: in case of very similar patches their associated score is close to 1, otherwise it is near 0.

Curvature Score

In addition to the prediction performed by *TripletNet*, the curvature between n_k and n_l is calculated, similarly to what is shown in [27], and a score s_{kl}^c based on this curvature calculation is assigned. As a matter of fact, a small ordered sequence of nodes is built by looking at the single neighbors of n_k and n_l, here denoted as n_k^n and n_l^n. The constructed sequence is thus $\mathcal{P}_j = \{n_k^n, n_k, n_l, n_l^n\}$. If the full sequence can not be built (i.e. is not possible to retrieve n_k^n, n_l^n or both), a shorter version devoted to one or both of them can be adopted. Indeed, the only requirement for the sequence is to contain three nodes. Thus, in extreme cases, although very unlikely, the intersection node can be adopted as a replacement, obtaining $\mathcal{P}_j = \{n_k, v_{\text{int}_j}, n_l\}$. In general, a sequence without the intersection node is preferred since it better approximates the curvature of the DLO in the region, i.e. it is more robust with respect to the spurious location of the

nodes. By considering the consecutive edges between each pair of nodes in $\mathcal{P}_j$, the angles ϕ_r are calculated, with $r = 1 \dots \#\mathcal{P}_j - 2$. Then, the Von Mises distribution $\mathcal{M}(\cdot)$ is deployed for converting the angles information into a score value obtained as $s^c_{kl} = \frac{1}{\#\mathcal{P}_j - 2} \prod_r \mathcal{M}(\phi_r - \phi_{r+1})$.

Combining the Scores

The curvature score is used to penalize the score computed by *TripletNet* by multiplying the latter with the first:

$$s_{kl} = s^v_{kl}\, s^c_{kl} \tag{3.1}$$

with s_{kl} representing the final prediction score associated to nodes n_k and n_l and inserted to the corresponding entry k, l of $\mathcal{W}_{\text{pred}}$.

For a given intersection node v_{int_j}, the prediction is performed considering all the possible h pairs of neighbor nodes, with $h = \binom{t}{z}$, t denoting the number of items in the set, i.e. the number of neighbors of v_{int_j}, and z describing the number of items forming the combinatorial set (i.e. in our case equal to 2). Thus, the prediction score for every kl pair in h is computed and $\mathcal{W}_{\text{pred}}$ updated accordingly. Thereafter, the procedure is repeated for each intersection node in G.

3.3.7 Paths Finder

A path $\mathcal{P}$ over a generic cluster $\mathcal{C}$ is a sequence of distinct alternating nodes and edges, such as $(v_{i_1}, e_{i_{1,2}}, v_{i_2}, e_{i_{2,3}}, \dots, v_{i_{l-1}}, e_{i_{l-1,l}}, v_{i_l})$, where an edge $e_{i_{j,k}}$ connects nodes v_{i_j} and v_{i_k}. To simplify the notation, the i-th path is referred as $\mathcal{P}_i = \{v_{i_1} \dots v_{i_l}\}$ where l is the total number of nodes denoting path i. The goal is to extend the path node by node in such a way that every node introduced in the sequence belongs to the same DLO in the input image. Nodes v_{i_1} and v_{i_l} will be denoted as endpoints. It is worth mentioning that, as a result of Sec. 3.3.5, in the remainder of this section the focus is related to considering only a single cluster of nodes $\mathcal{C}$, which represent a subset of the entire set of nodes of graph G. The procedure explained is then carried out for every cluster in G.

With reference to Alg. 1, the generic cluster $\mathcal{C}$ is first scanned to find the set of candidate endpoints $\mathcal{E}$, i.e. the set of nodes having $d(\epsilon_i) = 1$, where $\epsilon_i \in \mathcal{E}$ is used to refer to the i-th endpoint candidates. In case $\mathcal{C}$ is an intersection-free cluster (see Sec. 3.3.5), the set of endpoints will have two elements only, and the path discovery is started directly from one of the two cluster endpoints indifferently. Then, the nodes are connected in sequence exploiting neighboring relations until the second endpoint of the cluster is reached, and the resulting path is added to the set of all the complete paths $\mathbb{P}$.

However, in case the cluster $\mathcal{C}$ presents some intersections, the set $\mathcal{N}$ including all the nodes that are neighbors of intersection nodes is considered. Thereafter, the procedure PartialPath creates a partial path $\mathcal{P}$ starting from an endpoint ϵ and adding neighbor nodes to this path until a node in $\mathcal{N}$ is reached. It is worth mentioning that this is the only option because, if an endpoint node would be reached, this means that this part of the cluster could be classified as intersection-free. Then, the last added node is removed from $\mathcal{N}$ and the partial path $\mathcal{P}$ is added to the set of all the partial paths $\mathbb{P}_{\text{tmp}}$. The procedure is then repeated for each endpoint in the cluster. This will cover all the partial paths from the endpoints to an intersection, but will not cover the partial paths between two intersections. For this reason, new partial paths are then created considering as starting nodes the ones remaining in $\mathcal{N}$ and added to the set partial paths $\mathbb{P}_{\text{tmp}}$. Once these steps are concluded, the set of neighbor nodes should be empty, i.e.

Algorithm 1: Paths Finder

Input: $\mathcal{C}, \mathcal{W}_{\text{pred}}$
Output: $\mathbb{P}$

$\mathcal{E} \leftarrow \{v \in \mathcal{C} : d(v) = 1\}$
$\mathcal{E}_{\text{tmp}} \leftarrow \mathcal{E}$
$V_{\text{int}} \leftarrow \{v \in \mathcal{C} : d(v) > 2\}$
$\mathcal{N} \leftarrow \{v \in \mathcal{C} : d(v) \leq 2 \wedge \text{adj}(v) \cap V_{\text{int}} \neq \emptyset\}$
$\mathcal{N}_{\text{tmp}} \leftarrow \mathcal{N}$
$\mathbb{P} \leftarrow \emptyset$
$\mathbb{P}_{\text{tmp}} \leftarrow \emptyset$
PartialPath($\mathcal{E}, \mathcal{N}, \mathbb{P}, \mathbb{P}_{tmp}$)
$\mathcal{E} \leftarrow \mathcal{E}_{\text{tmp}}$
$\mathcal{N} \leftarrow \mathcal{N}_{\text{tmp}}$
while $\mathcal{W}_{pred} \neq \emptyset \wedge \mathcal{W}_{pred} \neq 0^{k \times k}$ **do**
  $\{n_i, n_j\} \leftarrow \{n_i, n_j \in \mathcal{N} : \arg\max_{i,j} w_{i,j} \in \mathcal{W}_{\text{pred}}\}$
  $v \leftarrow \{v \in V_{\text{int}} : \{n_i, n_j\} \in \text{adj}(v)\}$
  $V_{\text{int}} \leftarrow V_{\text{int}} \setminus v$
  $\mathcal{P} \leftarrow \{\mathcal{P}_i \in \mathbb{P}_{\text{tmp}} : n_i \in \mathcal{P}_i\}$
  $\mathbb{P}_{\text{tmp}} \leftarrow \mathbb{P}_{\text{tmp}} \setminus \mathcal{P}_i$
  $\mathcal{P} \leftarrow \mathcal{P} \cup v$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\mathcal{P}_j \in \mathbb{P}_{\text{tmp}} : n_j \in \mathcal{P}_j\}$
  $\mathbb{P}_{\text{tmp}} \leftarrow \mathbb{P}_{\text{tmp}} \setminus \mathcal{P}_j$
  if $\#(\mathcal{P} \cap \mathcal{E}) = 2$ **then**
    $\mathbb{P} \leftarrow \mathbb{P} \cup \mathcal{P}$
  else
    $\mathbb{P}_{\text{tmp}} \leftarrow \mathbb{P}_{\text{tmp}} \cup \mathcal{P}$
  for $l \in \{1, \ldots, k\}$ **do**
    $\mathcal{W}_{\text{pred}}[i, l] = 0$
    $\mathcal{W}_{\text{pred}}[l, i] = 0$
    $\mathcal{W}_{\text{pred}}[j, l] = 0$
    $\mathcal{W}_{\text{pred}}[l, j] = 0$
return $\mathbb{P}$

$\mathcal{N} = \emptyset$, and the set of partial paths should cover the whole cluster but the intersection nodes, i.e. $\mathbb{P}_{\text{tmp}} = \mathcal{C} \setminus V_{\text{int}}$.

In the last phase of the algorithm, the partial paths are joined on the basis of the Intersection Score Evaluation previously performed, see Sec. 3.3.6. To this end, the nodes pair (n_1, n_2) associated to the maximum value of $\mathcal{W}_{\text{pred}}$ is selected together with the associated intersection node v_{int} and the two partial paths containing n_1 or n_2 as starting or ending node extracted from $\mathbb{P}_{\text{tmp}}$. These two partial paths are then joined via the intersection node v_{int}, then the resulting path is added to $\mathbb{P}$ if the starting and ending nodes are endpoints, otherwise it is added back to $\mathbb{P}_{\text{tmp}}$. Then, the rows and columns of $\mathcal{W}_{\text{pred}}$ associated with both the couple (n_1, n_2) and (n_2, n_1) are zeroed to remove them from the selection, and the procedure is repeated until non zero values are present in $\mathcal{W}_{\text{pred}}$.

3.3.8 Paths Layout Inference

The set of paths obtained from Sec. 3.3.7 is not sufficient for performing a correct instance segmentation in case of intersections. Indeed it is not possible to draw the boundary of each DLO's instance in the image since each intersection node is assigned to all the paths involved. The goal of this section is to present the deep learning-based approach that handles this last issue. As the source of information, image patches with size 64×64 are used. The patches are obtained by cropping the source image around the centroid points associated with the intersection nodes and then by post-processing

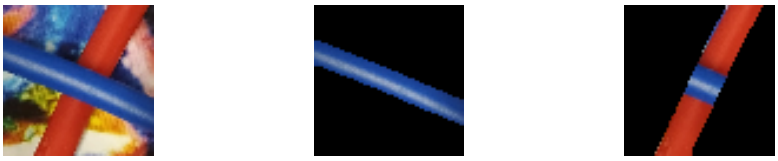
Procedure PartialPath($\mathcal{S}, \mathcal{N}, \mathbb{P}_1, \mathbb{P}_2$)

Input: $\mathcal{S}, \mathcal{N}, \mathbb{P}_1, \mathbb{P}_2$
Output: $\mathcal{S}, \mathcal{N}, \mathbb{P}_1, \mathbb{P}_2$
while $\mathcal{S} \neq \emptyset$ **do**
  $s \leftarrow \mathcal{S}$
  $\mathcal{P} \leftarrow s$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus s$
  $v \leftarrow \text{adj}(s)$
  while $v \notin \mathcal{S} \vee v \notin \mathcal{N}$ **do**
    $\mathcal{P} \leftarrow \mathcal{P} \cup v$
    $v \leftarrow \text{adj}(v) \setminus \mathcal{P}$
  if $v \in \mathcal{S}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \setminus v$
    $\mathbb{P}_1 \leftarrow \mathbb{P}_1 \cup \mathcal{P}$
  else
    $\mathcal{N} \leftarrow \mathcal{N} \setminus v$
    $\mathbb{P}_2 \leftarrow \mathbb{P}_2 \cup \mathcal{P}$
while $\mathcal{N} \neq \emptyset$ **do**
  $s \leftarrow \mathcal{N}$
  $\mathcal{P} \leftarrow s$
  $\mathcal{N} \leftarrow \mathcal{N} \setminus s$
  $v \leftarrow \text{adj}(s)$
  while $v \notin \mathcal{N}$ **do**
    $\mathcal{P} \leftarrow \mathcal{P} \cup v$
    $v \leftarrow \text{adj}(v) \setminus \mathcal{P}$
  $\mathcal{N} \leftarrow \mathcal{N} \setminus v$
  $\mathbb{P}_2 \leftarrow \mathbb{P}_2 \cup \mathcal{P}$
return $\mathcal{S}, \mathcal{N}, \mathbb{P}_1, \mathbb{P}_2$

further the crops. For ease of presentation, the following analysis is focused on a sample intersection, as the one depicted in Fig. 3.6a where the crossing is constituted by an intersection node shared between two different paths. For each path, a 2D spline curve is interpolated using the nodes' centroid coordinates as control points and, given the spline, a binary mask of the DLO corresponding to the path is generated. The crop shown in Fig. 3.6a is thus processed utilizing the computed masks obtaining Figs. 3.6b and 3.6c, where the spline-based masks are used to discard all the pixels of the crop not belonging to the considered DLO.

The obtained patches are provided as input to *CrossNet*, a deep neural network employed for their classification. It is composed of a feature extractor (*ResNet18*) combined with a fully connected layer ($FC_{512,1}$) with an input dimension of 512 and an output dimension of 1, followed by a sigmoid activation function. Thus, the network performs a binary classification task between two classes (e.g. *is_above* and *is_not_above*) and provides a single probability value as output: 1 (one) if the input patch is predicted to represent a DLO placed at the top of the intersection area, 0 (zero) otherwise.

Considering again the example shown in Fig. 3.6, the predicted probabilities computed by *CrossNet* for both patches of Fig. 3.6b and 3.6c are acquired. As DLO (and path) *is_above*, it is selected the one with the highest probability. So, the selection is performed based on an output comparison. In fact, since in the proposed framework there is always one *is_above* class sample among the patches when classifying an intersection, the utilization of a threshold for the probability value is avoided, making the approach more reliable for example in the case of difficult samples where fixing a threshold value can be challenging. The approach is not limited to just two DLOs crossing, but it applies also to three or more DLOs.



(a) intersection node (b) is_above class (c) is_not_above class

Figure 3.6: Example of intersection node (a) and representative patches of the two classes (b,c) for the displayed intersection. *CrossNet* is employed to predict the class of each patch.

3.3.9 B-Spline Modelling

The final paths obtained from Sec. 3.3.8 are employed for estimating B-Splines curves. For every path, the ordered sequence of nodes is translated into a sequence of 2D points by reading the nodes' centroid coordinates attributes. The centroid coordinates were computed at the superpixel segmentation stage (Sec. 3.3.2). A cubic B-Spline is fitted to this set of points. The obtained curve is thus discretized into an ordered fixed number of 2D points in pixel coordinates. In this way, a light model of each DLO in the image is provided that can be useful, for example, in robotic manipulation and tracking tasks.

3.4 The *FASTDLO* Algorithm

The *FASTDLO* algorithm, where the name stands for *FAst SegmenTation of Deformable Linear Objects*, is a novel method for a reliable, accurate and fast instance segmentation of DLOs assuming zero knowledge about the background and the number of objects in the scene.

Compared to *Ariadne+* of Sec. 3.3, *FASTDLO* introduces several improvements addressing the *Ariadne+* weaknesses. Indeed, *Ariadne+* throughput is limited to a few FPS providing a strong limiting factor for its applicability on real-world problems. Thus, instead of a paths discovery method based on the superpixelization that requires significant processing effort and the definition of the number of superpixels in an image, *FASTDLO* focuses directly on solving the intersection areas of the image between multiple DLOs to distinguish the instances, increasing the speed and accuracy of the results.

More in detail, *FASTDLO* takes as input an image and provides as output a colored mask where each DLO instance is denoted with a unique color. In addition, *FASTDLO* outputs a sequence of key points for each DLO instance, similarly to *Ariadne+*. From the input image, the background, i.e. pixels not corresponding to a DLO-like object, is removed utilizing DCNN, generating as output a binary mask. Thereafter, the binary mask is processed with a skeletonization algorithm and the ambiguous intersections between the DLOs are solved with a second data-driven approach based on a shallow similarity-based NN. Synthetically generated data are deployed in the learning-based methods allowing fast adaptability to every possible custom scenario. *FASTDLO* achieves a processing rate higher than 20 FPS with an image size of 640×360 pixels, employing, as hardware, a workstation with an Intel Core i9-9900K CPU clocked at 3.60GHz and an NVIDIA GeForce GTX 2080 Ti.

The *FASTDLO* pipeline, schematized in Fig. 3.7, consists of the following main steps:

1. **Background Segmentation** : A DCNN performs the segmentation of the source image discerning background pixels from DLOs pixels, outputting a binary mask M_b.

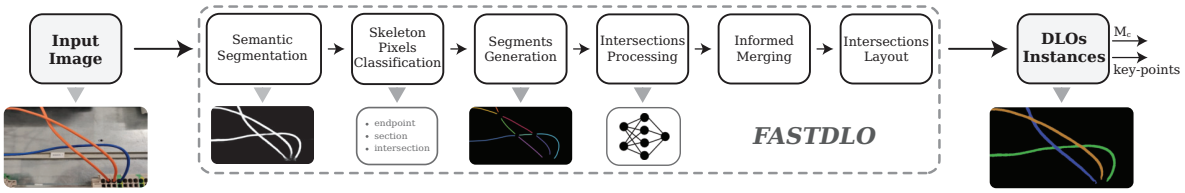


Figure 3.7: The *FASTDLO* algorithm.

2. **Skeleton Pixels Classification** : A skeleton M_s is generated from the mask M_b and its pixels are classified depending on their local neighborhoods.
3. **Segments Generation**: The intersection areas of M_s are filtered out and *segments* are generated;
4. **Intersections Processing**: A shallow NN is employed to predict connection probabilities among endpoint pairs;
5. **Informed Merging**: The *segments* are concatenated to recover the full description of each DLO employing the result of intersections processing;
6. **Intersections Layout**: The standard deviations of the DLOs instances RGB colors at the level of the intersections are used to assess the correct ordering at the intersection areas to create correct instance masks.

In the following, the aforementioned steps are analyzed in detail.

3.4.1 Background Segmentation

Like in *Ariadne+*, the generic input image I_s is processed employing a DCNN performing the semantic segmentation following the same methodology explained in Sec. 3.3.1. As dataset for the training of *DeepLabV3+* [23], synthetic images of DLOs rendered in Blender are used as detailed in Sec. 2.4 and 2.7.1.

In Fig. 3.8 an example of the segmentation process on real samples is shown: the shadows present in the input image do not affect the predicted mask and are successfully neglected; instead, the background object in the second row is more difficult to handle since it appears as a thin wire in the image, so few false positives can be found in the associated mask.

3.4.2 Skeleton Pixels Classification

The segmentation mask M_b is processed with a skeletonization algorithm consisting of a thinning iterative approach which erodes the input mask. Thus, a new mask M_s is obtained having the following properties: 1) same connectivity as the input mask; 2) 1-pixel width across the mask instead of the original mask thickness; 3) equidistant skeleton to the borders of M_b. In Fig. 3.8 (last column) the masks M_b and M_s are combined to highlight these properties.

From an image comprising several DLOs and exploiting the linearity property of the latter, for each pixel of the skeleton M_s, defining a small (3×3) kernel, only three types of local neighbors can be experienced. They are depicted in Fig. 3.9 with the target pixel at the center of the local region highlighted with a red contour. After the skeletonization, each pixel of M_s receives a label depending on its local neighborhood:

- *endpoint*: only one pixel in addition to the central one is contained in the local neighborhood, i.e. Fig. 3.9a;

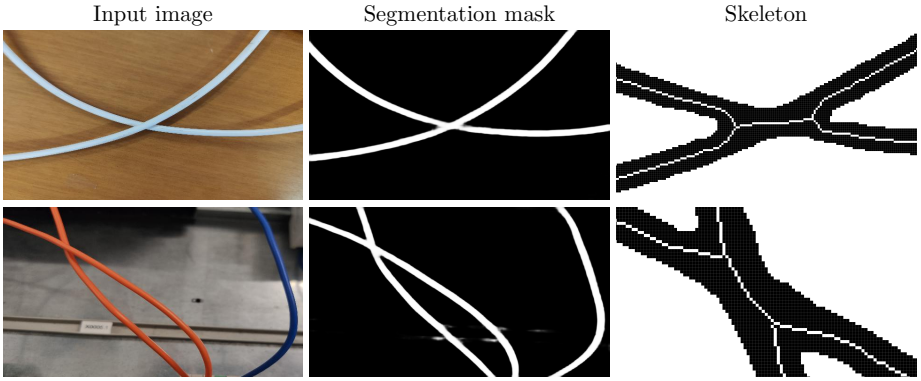
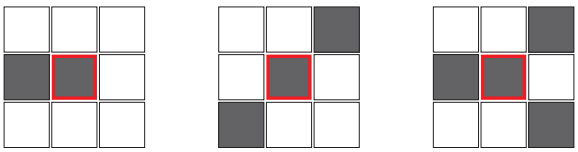


Figure 3.8: Input images with background segmentation results and generated skeletons zoomed at the area of the intersections. To clarify the skeleton visualization, the mask colors in the right column are inverted.



(a) endpoint (b) section (c) intersection

Figure 3.9: Local neighbors possibilities of a skeleton pixel given a 3×3 kernel. To clarify the representation, the skeleton is dark colored while the background is white.

- *section*: two more pixels are present in the neighborhood, i.e. Fig. 3.9b. The term *section* refers to the considered pixel being placed along a section of a DLO and not at its end.
- *intersection*: the central pixel is surrounded by three more pixels in the neighborhood forming, in general, a characteristic 'Y' shape [54], i.e. Fig. 3.9c. This condition arises in the case of binary masks describing DLOs crossing each other.

3.4.3 Segments Generation

The *intersection* pixels with their surrounding area are discarded from M_s since they correspond to a topologically misleading region of M_s due to the DLOs crossing. Indeed, these phenomena can be appreciated in Fig. 3.8 where the generated skeletons nearby the intersection pixels do not describe correctly the DLO topology, i.e. center line, as opposed to the skeleton pixels far away from it. The discard operation is performed based on the distance transform image of the local area considered. The distance transform is an operation that computes the distance, in pixel values, between a given pixel location to the nearest boundary [7], i.e. black pixels of M_b.

Because of the removal of the intersection areas, new *endpoint* pixels emerge in the updated skeleton. Thus, *segments* are generated between two connected *endpoints*. A *segment* is defined as an ordered sequence of pixels where the elements inside the sequence are *sections* whereas the extremities are *endpoints*. The *segment* sequence can be effectively obtained by sliding the skeleton with a 3×3 kernel from one of its *endpoints*, collecting the only pixel not already in the *segment* under construction and updating the kernel anchor to the added pixel location. In addition, for each *segment*, a common thickness is estimated based on a distance transform previously computed. The overall

Algorithm 2: Intersections Processing

Input: $\mathcal{C}$, $\mathcal{S}$, $\mathcal{E}$
Output: $\mathcal{Z}$

1 $\mathcal{P} \leftarrow \emptyset$ // endpoint-pairs collection
2 **foreach** $c \in \mathcal{C}$ **do**
3 $\quad \mathcal{E}_c \leftarrow \{e \in \mathcal{E} \cap e \in \mathcal{C}\}$
4 $\quad \mathcal{P}_c \leftarrow combination(\mathcal{E}_c, 2)$
5 $\quad \mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_c$
6 $\mathcal{Z} \leftarrow \emptyset$ // similarity network predictions
7 **foreach** $(e_i, e_j) \in \mathcal{P}$ **do**
8 $\quad x_i \leftarrow getFeatureVector(e_i)$
9 $\quad x_j \leftarrow getFeatureVector(e_j)$
10 $\quad z_i \leftarrow computeEbbendingVector(x_i)$
11 $\quad z_j \leftarrow computeEbbendingVector(x_j)$
12 $\quad p_{ij} \leftarrow e^{-\|z_i, z_j\|^2}$
13 $\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{e_i, e_j, p_{ij}\}$
14 return $\mathcal{Z}$

segment thickness is obtained by computing the median value of the distances gathered for each *segment's* pixel. The median allows gaining robustness against spurious values due to noisy boundaries in M_b. In Fig. 3.7 the *segments* generated for the considered image are denoted with unique colors.

3.4.4 Intersections Processing

The intersections among the DLOs are solved by comparing the feature vectors of the endpoints of two candidate segments via a shallow neural network, i.e. similarity network, predicting the probability of their connection. The computation of the connection probabilities is schematized in Alg. 2 showing the two main phases: endpoint-pairs collection; similarity network predictions. The inputs of the algorithm are the set of all the *intersections* in the image, i.e. $\mathcal{C}$, and, given the updated skeleton of Sec. 3.4.3, the sets of the *endpoints* and of the *segments*, i.e. $\mathcal{E}$ and $\mathcal{S}$ respectively.

The approach starts by collecting all the endpoint pairs whose connection needs to be evaluated by initializing $\mathcal{P}$ empty (line 1). Thus, for each intersection to be solved c, the endpoints of the *segments* associated to c, i.e. originally connected to c before removing the *intersection* pixels from the skeleton (see Sec. 3.4.3), are extracted from $\mathcal{E}$ and collected into $\mathcal{E}_c$, line 3. Then, the components of $\mathcal{E}_c$ are organized into combinations of 2 elements, i.e. endpoint-pairs, in $\mathcal{P}_c$, and the set of endpoint-pairs $\mathcal{P}$ is updated accordingly, lines 4 and 5.

The collected endpoint pairs $\mathcal{P}$ are now processed by a similarity network. The goal of this network is to transform an input feature vector into an embedding space where similar input vectors are close together and dissimilar ones are far apart. In the setup adopted in this work, the triplet loss [105], originally introduced in Sec. 3.3.6, is deployed for the required optimization of the network. The loss is computed between an anchor, a positive and a negative sample. The distance in the embedding space between anchor and positive is minimized, while the one between anchor and negative is maximized. The input feature vectors are obtained from the endpoints of the segments around a given intersection. As feature elements of the input vector $x \in \mathbb{R}^{d_i}$, the following values are used: RGB color of the local endpoint area; thickness of the segment associated with the endpoint; endpoint direction estimate.

In Alg. 2, the feature vector for the endpoints e_i and e_j are created at lines 8 and 9. Then, a forward pass in the network layers is performed to compute the embedding

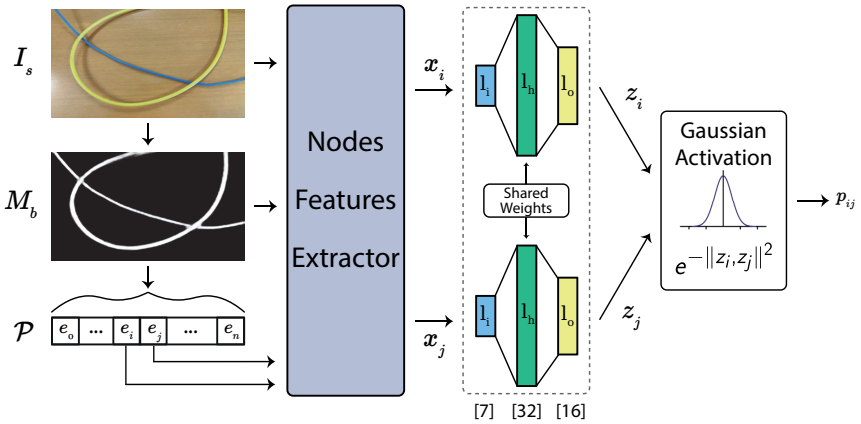
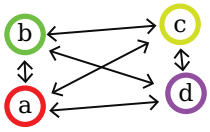


Figure 3.10: Endpoint-pair probability computation. For the general endpoint e_i, from the source image I_s and binary mask M_b a feature vector is created as x_i. The embedding vector z_i is obtained after the propagation of x_i in the similarity network layers. A Gaussian activation function on the embeddings L2-distance is employed for calculating the final score p_{ij} of the endpoints e_i and e_j.

(a) different *segments*



predictions		
b	d	0.987
a	c	0.928
d	a	0.290
b	a	0.185
c	d	0.249
c	b	0.160

(b) intersection processing with predictions of the endpoints pairs

Figure 3.11: (a) *segments* generated; (b) example of intersection processing of the region highlighted area in (a).

vectors z_i and z_j, lines 10 and 11.

As mentioned, the prediction is based on the distance of the embedding vectors which can be computed as $d_{ij} = \|z_i, z_j\|^2$, where $\|\cdot\|^2$ denotes the L2-distance. To obtain a probability-like value in the $[0, 1]$ range describing the likelihood of the connection, the distance is transformed through a Gaussian activation function as $p_{ij} = e^{-d_{ij}}$. This last step in the similarity network is provided at line 12 while an illustration schematizing the computation flow of the similarity network from its inputs till the predicted connection score is available in Fig. 3.10.

To conclude, for each endpoint pair, a probability value p_{ij} is computed and the set $\mathcal{Z}$ updated (line 13) with tuples of three values, i.e. endpoint pair (e_i,e_j) and connection probability p_{ij}. Although Alg. 4 describes the process for each element of $\mathcal{P}$, the actual implementation is based on batch processing enabling an efficient computation of the scores. In Fig. 3.11b an example of the processing for four segments (six endpoint-pairs) extracted from Fig. 3.8 (first row) is shown.

3.4.5 Informed Merging

Exploiting the endpoint pairs connection probabilities computed in Sec. 3.4.4 it is possible to concatenate *segments* obtaining the full description of each DLO in the image. This

Algorithm 3: Informed Merging

Input: $\mathcal{Z}$, $\mathcal{S}$
Output: $\mathcal{S}$

1 $\mathcal{Z} \leftarrow sorted(\mathcal{Z}$, "descending")
2 $\mathcal{E}_z \leftarrow \emptyset$
3 **foreach** $(e_i, e_j, p_{ij}) \in \mathcal{Z}$ **do**
4 **if** $e_i \notin \mathcal{E}_z$ *and* $e_j \notin \mathcal{E}_z$ **then**
5 **if** $p_{ij} > t_c$ **then**
6 $s_i \leftarrow getSegmentFromEndpoint(\mathcal{S}, e_i)$
7 $s_j \leftarrow getSegmentFromEndpoint(\mathcal{S}, e_j)$
8 $s \leftarrow s_i \cup s_j$
9 $\mathcal{S} \leftarrow \mathcal{S} \cup s \setminus \{s_i, s_j\}$
10 $\mathcal{E}_z \leftarrow \mathcal{E}_z \cup \{e_i, e_j\}$
11 return $\mathcal{S}$

concatenation process is addressed as informed merging and it is schematized in Alg. 3, showing how $\mathcal{Z}$ is employed to iteratively update $\mathcal{S}$ till each $s \in \mathcal{S}$ describes a whole DLO.

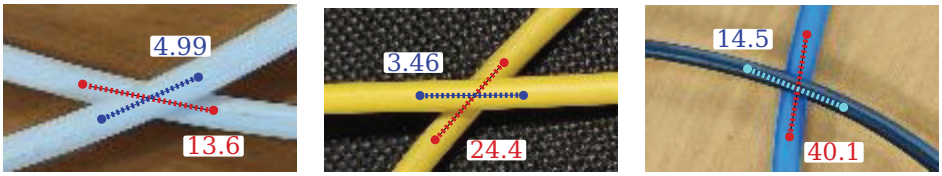
First, the elements of $\mathcal{Z}$ are sorted based on the connection probability values in descending order (line 1), thus prioritizing during the merging process the most probable connections. The set of nodes already processed, i.e. $\mathcal{E}_z$, is initialized to zero at line 2. An iteration on the elements of $\mathcal{Z}$ is performed and, starting from the highest score and moving toward the lowest one, the merging of the segments, i.e. Fig. 3.11a, is executed. Indeed, if both the endpoints retrieved from one of the elements of $\mathcal{Z}$ are not already processed (line 4), and their endpoint-pair connection probability is larger than a user-defined threshold t_c (line 5), the two corresponding *segments* associated to the endpoints are collected (lines 6 and 7), merged (line 8) and the *segments* set updated (line 9). The term *informed* merging is referred the high-level operation of performing the union between the two *segments* sets taking into consideration their ordering, e.g. head-tail, tail-tail and all the other combinations.

Consequently, the endpoint pairs having lower scores and with one of the two endpoint elements already associated are not considered and their merging is avoided. The described association continues for all the elements of $\mathcal{Z}$ having a connection probability larger than the threshold t_c, introduced to avoid merging endpoints with incompatible orientations, colors or thicknesses. This threshold is effectively used only in situations where the mask M_b is not reliable and borderline conditions occur. Instead, in normal settings, the merging process would result in the first association of high-probability endpoints thus making the low-probability ones already incompatible irrespective of the threshold value.

The presented merging process is performed directly on the existing *segments*, thus the operation is propagated by updating the set of *segments* accordingly. For instance, in case of a *segment* disputed by two different intersections, at the second merging process, the operation of joining the two candidates *segments* is performed on the new merged *segment* (obtained after the first merging process) and not on the initial one.

3.4.6 Intersections Layout

As additional information aiming at providing a complete and accurate solution of the scene, the order of the DLOs in a given intersection, i.e. which is the one at the top of the pile, is provided by comparing the standard deviation of the RGB colors along the line connecting the endpoint-pair previously solved. For example, given an intersection made of two DLOs, i.e. with four endpoints, and hence two endpoint-pairs predicted,



(a) example 1 (b) example 2 (c) example 3

Figure 3.12: Example of the intersection layout estimation with DLOs having identical colors, in (a) and (b), and different colors (c). In all the examples the DLO at the top is blue labeled.

the RGB color values along the two endpoint-pairs positions are collected and their mean standard deviation, i.e. the mean of the standard deviations computed for each channel, compared. The pair with the smallest standard deviation is assumed to be at the top of the intersection pile, while the highest standard deviation pair is below. The difference in the value is due to the change in the color along the line for the DLO not at the top or, in the case of DLOs with identical colors, mostly due to the shadows projected from the above DLO onto the one below. In the case of a cross composed of three or more DLOs, only the instance above them all can be identified since the continuity in the colors in the intersection region is the main deciding condition. This continuity is only met for the top DLO. The approach is quite simple and yet proves to be effective and inherently fast given the intersection solution provided in Sec. 3.4.4 is reliable.

DLOs ordering in an intersection is particularly needed in case this approach is integrated into a larger manipulation pipeline with a robotic system for routing or pick and place tasks. The information about the layout of the DLOs in an intersection is missing both in [27] and [54]. Instead, the solution initially proposed in Sec. 3.3.8 is based on a data-driven classification approach requiring specific training and constrained at precise image crop resolutions. On the contrary, the approach proposed with *FASTDLO* exploits the accurate DLOs center line localization obtained from the skeleton, as opposed to the superpixels centroids used in Sec. 3.3.2, avoiding then the introduction of additional data-driven approaches. In Fig. 3.12 some example intersections are displayed with the computed values.

3.5 The RT-DLO Algorithm

The *RT-DLO* algorithm, the last approach proposed after *Ariadne+* (Sec. 3.3) and *FASTDLO* (Sec. 3.4), is a real-time capable and robust approach concerning the instance segmentation of DLOs. The name *RT-DLO* stands for Real-Time instance segmentation of Deformable Linear Objects.

RT-DLO, as the previous two methods, does not require any assumption about the background and the number of DLOs present in the scene. Again, similarly to *Ariadne+* and *FASTDLO*, as input it acquires the RGB image of the scene while providing as output a pixel-mapped colored mask where each DLO is represented by a unique color identifying its ID. In addition, being the DLO instances modeled as a sequence of key points, a representation of the scene with spline curves can be easily obtained, e.g. for manipulation tasks employing a state-space representation different from the image space [128], see Sec. 3.3.9.

Compared to *Ariadne+* and *FASTDLO*, *RT-DLO* employs an efficient and informative graph representation of the scene as opposed to the skeleton originated *segments*-based approach of *FASTDLO* and superpixel-based one of *Ariadne+*, resulting in

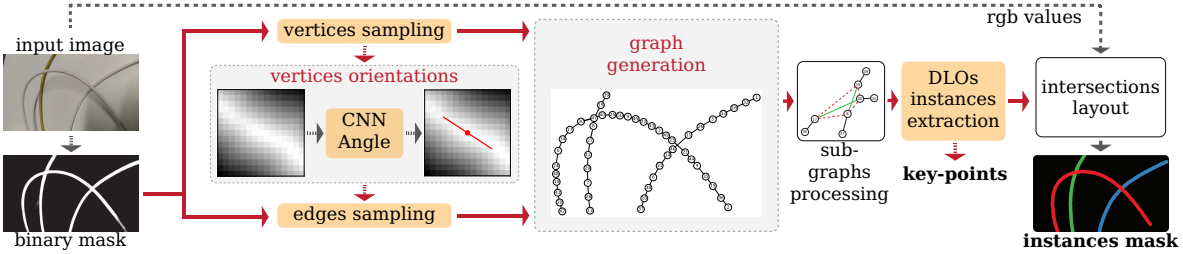


Figure 3.13: The *RT-DLO* algorithm.

faster processing times and improved accuracy, especially at the DLOs intersection. Indeed, *RT-DLO* can handle degraded masks more effectively since the continuity of the segmentation mask foreground along a DLO is not required.

In *RT-DLO*, as a pre-processing step, the input RGB image is propagated through a DCNN trained on synthetically generated data aiming at segmenting the background, i.e. pixels not representing a DLO, and providing as output a binary mask. Then, a graph representation of the scene is constructed by efficiently sampling the vertices from the segmentation mask. The edges connecting the graph's vertices are instead computed by reasoning about the topology expressed by the mask, with an approach that considers both the proximity and orientation constraints among the vertices. Ideally, only a maximum of two edges per vertex should be sampled. In the case of intersections of DLOs resulting in the presence of high-degree vertices in the graph, sub-graphs around the target vertices are extracted and further processed to disentangle the DLOs in the graph. Finally, the single DLOs are extracted from the graph based on an analysis of the graph connectivity. *RT-DLO* achieves a processing rate higher than 30 FPS with an input image of 640×360 pixels employing the same hardware of *Ariadne+* and *FASTDLO*.

To summarize, the idea exploited in *RT-DLO* is to model the current configuration of the DLOs present in the image with a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and then to extract the DLO instances from the obtained graph. The approach, schematized in Fig. 3.13, can thus be subdivided into six main steps:

1. **Mask Generation**: obtaining a binary mask M_b from the input color image via a DCNN;
2. **Vertices Sampling**: processing M_b, with vertices orientation characterization based on a CNN;
3. **Edges Sampling**: exploiting the proximity among the vertices and the orientation between vertices and edges;
4. **Intersections Processing**: disentangling the DLOs in the graph representation via sub-graphs analysis;
5. **DLOs Instances Extraction**: computing pixel-wise DLOs instances masks in the image plane;
6. **Intersections Layout**: assessing the correct instances locally at the intersections.

In the remainder of this section, the procedures for obtaining the graph representation and extracting coherently DLOs instances from it are presented. First, the binary mask M_b generation is discussed in Sec. 3.5.1. Then, concerning the graph formation process, the vertices are examined in Sec. 3.5.2 while the edges are in Sec. 3.5.3. Thereafter,

the algorithm employed for processing problematic regions of the graph is provided in Sec. 3.5.4. Finally, the extraction of the DLO instances, given the graph representation, is presented in Sec. 3.5.5 while the approach for analyzing their layout is in Sec. 3.5.6.

3.5.1 Mask Generation

The mask generation step can be considered a pre-processing phase of *RT-DLO* since the graph representation of the DLOs is obtained employing only the binary mask M_b of the scene and not the RGB image. As in Sec. 3.4.1, DeepLabV3+ [23] trained on synthetically generated data (Sec. 2.4) is employed as DCNN. This choice is convenient since 1) good performances are shown in Sec. 2.7.1 concerning the semantic segmentation capabilities of this method; 2) a simplification on the comparison of *RT-DLO* against *FASTDLO* is achieved. Therefore, a binary mask M_b is obtained by setting the pixels predicted to belong to a DLO to 1 and the remaining ones to 0.

It is worth mentioning that *RT-DLO* is independent of the method used to obtain the semantic segmentation mask. Different approaches can be employed depending on application requirements.

3.5.2 Vertices

First, vertices of the graph $\mathcal{G}$ are sampled from the binary mask M_b and then characterized in terms of local orientation by a CNN.

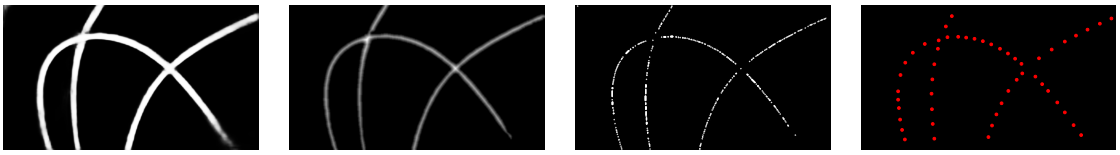
Vertices sampling

The set $\mathcal{V} = \{v_i\}_{i=1}^{n}$ contains the n vertices of the graph efficiently sampled from the binary mask M_b. First, the distance transform operator is executed on M_b obtaining M_{dist}. This operator computes the Euclidean distances between the non-zero values of M_b and the nearest boundaries (zero/black values) [7], thus assigning an intensity value to each pixel based on the computed distance. In Fig. 3.14b, M_{dist} originated from M_b (Fig 3.14a) is shown where M_{dist} is color-mapped on the grayscale level from dark (zero distance) to bright (maximum distance).

Then, M_{dist} is dilated with a small square kernel (i.e. 3×3). The dilation operation is a maximum locating morphology operation. Indeed, as the kernel is convolved over the target image, the maximal pixel value overlapped by the kernel is computed and the corresponding image pixel at the anchor position is replaced. Dilation is usually applied on binary masks to enlarge the foreground (white) portion. Instead, in *RT-DLO*, the dilation operation is applied to the mask M_{dist} which contains intensities values, i.e. M_{dist} is not binary, obtaining M_{dil}. The local maximums of M_{dist} are retrieved by comparing pixel-wise M_{dist} and M_{dil} masked using M_b, as follows:

$$M_{\text{max}}(i,j) = \begin{cases} 1 & \text{if } M_{\text{dil}}(i,j) = M_{\text{dist}}(i,j) \text{ and } M_b(i,j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Indeed, if the value of pixel (i,j) in M_{dist} and M_{dil} is the same, this means that the considered pixel is a local maximum. By assigning the pixel value of 1 to the maximums and 0 to the rest of the pixels, a new mask is obtained, denoted with M_{max}, and illustrated in Fig 3.14c. It is worth mentioning that, by construction, M_{max} approximates the center lines of the DLOs in the mask. The dilation-based approach resemble the non-maximum suppression (NMS) operation employed in the context of object detection [4]. Indeed, the NMS operation is used to remove redundant detections by keeping only the most



(a) M_b (b) M_{dist} (c) $M_{\max}$ (d) vertices

Figure 3.14: Vertices sampling key elements: the mask M_b (a), M_{dist} (b) and $M_{\max}$ (c), the obtained vertices (d). The bright regions in (b) denote high-intensity values.

confident ones. In *RT-DLO*, it is employed to remove redundant pixels in the mask M_{dist} by keeping only the local maximums.

The set of maximum pixels of $M_{\max}$, i.e. pixels whose value is equal to 1, is denoted as $\mathcal{V}_{\max}$. The cardinality of $\mathcal{V}_{\max}$ is relatively large and not really tractable in case real-time applications are sought. Thus, the farthest point sampling (fps) algorithm [92] is employed for down-sampling $\mathcal{V}_{\max}$. A sampling ratio of $\alpha \in [0, 1]$ is used to specify the amount of down-sampling. The set of vertices $\mathcal{V}$ of the graph $\mathcal{G}$ is obtained as $\alpha\mathcal{V}_{\max}$. In Fig. 3.14d the vertices extracted from the sample mask of Fig. 3.14a with $\alpha = 0.15$ are depicted.

Vertices Orientations

In the context of linear objects and linear shapes representation, for each given vertex of the graph, an orientation characterization can be performed. The objective is to describe locally the section of the linear object in the vicinity of the vertex as an orientation attribute of the vertex itself. Thus, the local orientation θ of a given vertex at pixel coordinates (x, y) is derived from a local patch of size $\delta \times \delta$ pixels, centered at (x, y) and with intensity values extracted from the distance transform image M_{dist}.

A CNN is used to estimate an angular value from a given patch. Predicting an angular value via a learning-based method can become quite a complex task due to the periodicity of the angular data resulting in inaccurate distance representations when computing the loss function. Indeed, an angle of 2° describes an orientation quite close both to 5° and 179°, although the corresponding loss values when applying common losses, e.g. L1-loss or MSE-loss, are quite different. An approach pioneered in [129] is thus employed to address the angular periodicity and ambiguity in the loss computation. A given angular value θ in the range $[0°, 180°]$ is encoded as a 180-dimensional vector with entries defined by applying a Gaussian function centered at θ and with variance σ. In this way, the angle θ is propagated smoothly in its proximity enabling benefits during the loss computation. The network structure is composed of two convolutional layers followed by a fully connected linear layer. Each convolution layer comprises a 2D convolution followed by batch normalization. Between the two layers, a max-pooling operation takes place. After the convolution layers, the embedded data are flattened and the fully connected layer is used as an output to classify the patch in the 180-dimensional vector. Binary cross entropy is used as loss function during the training stages, effectively shaping the learning task as a classification problem of the angular value in one of the 180 available classes. Consequently, the predicted angle is easily obtained from the 180-dimensional vector as the index of the vector associated with the maximum probability. This angular value characterizes the orientation of the vertex associated with the processed patch.

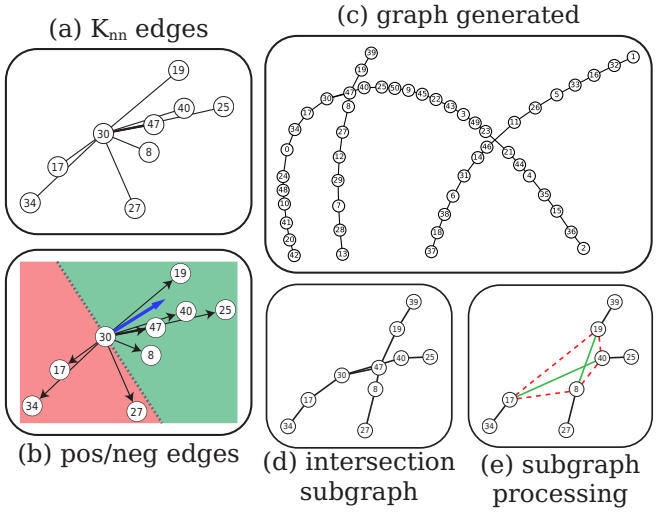


Figure 3.15: Edges processing main elements: (a) K_{nn} edges to obtain initial candidate edge set; (b) positive/negative edges illustration; (c) graph generated; (d) intersection subgraph extracted; (e) subgraph processing schema.

3.5.3 Edges

The set $\mathcal{E} = \{e_j\}_{j=1}^{m}$ contains the m edges of the graph. Identifying the correct edges to be inserted in the graph is a complex task. Indeed, the connections between the vertices should consider both their relative proximity as well as orientation constraints, the latter in the form of vertex orientation and edge orientation. The vertices orientations were described in Sec. 3.5.2. For convenience, a matrix $E \in \mathbb{R}^{m\times 2}$ describing the edge set $\mathcal{E}$ as organized tuples is introduced.

The relative proximity between vertices is exploited to obtain an initial candidate set of edges, denoted as $\mathcal{E}_{\text{knn}} = \{e_j\}_{j=1}^{m_{\text{knn}}}$. That is, for each vertex, the K_{nn} nearest neighbors in $\mathcal{V}$ are retrieved as edges. The value of K_{nn} is a user-defined parameter and it follows that $m_{\text{knn}} = n \times K_{nn}$ if the edges are considered as directed. In addition, $E_{\text{knn}} \in \mathbb{R}^{m_{\text{knn}}\times 2}$ is the matrix description of $\mathcal{E}_{\text{knn}}$. The K_{nn} nearest neighbor case with $K_{\text{nn}} = 8$ for a sample vertex is depicted in Fig. 3.15a.

Vertex-Vertex Similarity

The orientation constraints between two general vertices v_1 and v_2 are evaluated by assigning a score to their connection by means of the cosine similarity defined as

$$s(\mathbf{d_v^1}, \mathbf{d_v^2}) = \frac{\mathbf{d_v^1}^T \mathbf{d_v^2}}{\|\mathbf{d_v^1}\|\|\mathbf{d_v^2}\|} \tag{3.2}$$

In particular $\mathbf{d_v^1}$ is obtained as $[\cos(\theta_1), \sin(\theta_1)]^\top$, where θ_1 is the orientation of v_1 obtained from Sec. 3.5.2. For $\mathbf{d_v^2}$ the derivation is similar. In eq. (6.2), the product of the norms is denoted at the denominator. The cosine similarity is then used to score the orientations between two vertices pair.

For efficiency reasons, the cosine similarity is evaluated by means of matrix operations. Given the matrix $D_v \in \mathbb{R}^{n\times 2}$ of vertices orientations in the form of direction vectors obtained from the predicted angles, i.e. for vertex i as $\mathbf{d_v^i}/\left\|\mathbf{d_v^i}\right\|$, the cosine similarity

between each pair of vertices of the set $\mathcal{V}$ can be obtained as

$$S_{v,v} = |D_v D_v^T| \tag{3.3}$$

being $S_{v,v} \in \mathbb{R}^{n \times n}$ and $|\cdot|$ denoting the absolute value.

Vertex-Edge Similarity

Similarly to the vertex-vertex case, the matrix $D_e \in \mathbb{R}^{m_{\text{knn}} \times 2}$ of edges orientations can be defined. It contains the direction vectors obtained by subtracting the coordinates of the associated vertices followed by a normalization by their distance. The cosine similarity between each vertex of $\mathcal{V}$ and each edge of $\mathcal{E}_{\text{knn}}$ is obtained as:

$$S_{v,e} = D_v D_e^T \tag{3.4}$$

with $S_{v,e} \in \mathbb{R}^{n \times m_{\text{knn}}}$ being the obtained similarity matrix between vertices and edges.

Combining $S_{v,v}$ and $S_{v,e}$

At the current stage, because of the dimensions mismatch, it is not possible to combine $S_{v,v}$ (eq. (3.3)) and $S_{v,e}$ (eq. (3.4)). Thus, an augmented similarity vertices score matrix $\bar{S}_{v,v} \in \mathbb{R}^{n \times m_{\text{knn}}}$ is introduced. This matrix is obtained by mapping the values of $S_{v,v}$ in a column vector employing the entries of E_{knn} as row-column pairs to access $S_{v,v}$. Then, a matrix is constructed by repeating the column vector n times along the rows. Notice that this is a valid operation since $S_{v,v}$ is a symmetric matrix. The complete similarity score matrix is obtained as:

$$S = S_{v,e} \odot \bar{S}_{v,v} \odot B \tag{3.5}$$

where $B \in \mathbb{R}^{n \times m_{\text{knn}}}$ is the oriented incidence matrix and $\odot$ the Hadamard product. The matrix B is used to inject into the scores the knowledge of the edge existence (entries 0) and direction (entries ± 1). i.e. source vertex to target vertex. This information is very helpful since it allows the discrimination of the edge set based on the sign of their similarity score, i.e. the entries of S. An illustration of the two possible situations that can occur is provided in Fig 3.15b. The cosine similarity between the sample vertex 30 and its K_{nn} neighbors can provide both positive values, in case the edge direction vectors and the vertex orientation vector of 30 are both in the green region or negative values if instead, they lay in the red region.

Based on the scores contained in the similarity matrix S, a positive and a negative edge for each vertex of $\mathcal{V}$ are sought, being the characterization of an edge as positive or negative related to the sign of the associated score in S. Notice that it may happen that a positive or negative edge for a given vertex does not exist, e.g. in the presence of a vertex describing the terminal region of a DLO. The calculus to extract the positive and negative edges from the similarity matrix S of eq. (3.5) is provided in the following.

Positive Edges

Given $B_+ \in \mathbb{R}^{n \times m}$ as the positive incidence matrix where the entries -1 of B are set to zero, i.e. B_+ contains values of the set $\{0, +1\}$. Defining also a row vector $\mathbf{d} \in \mathbb{R}^{1 \times m_{\text{knn}}}$ containing the lengths of the edges. A matrix $D \in \mathbb{R}^{n \times m_{\text{knn}}}$ can be created stacking n times $\mathbf{d}$ along the rows. Thus, the entries of D can be filtered out based on B_+ as $D_+ = D \odot B_+$. Then, a generic entry (i, j) of S is weighted based on the associated edge length as $w_+^{ij} = 1 - \frac{d_+^{ij} - \min(\mathbf{d}_+^{\mathbf{i}})}{\max(\mathbf{d}_+^{\mathbf{i}})}$. The vector $\mathbf{d}_+^{\mathbf{i}}$ denotes the i-th row of D_+. The matrix

containing all the computed weights is denoted as $W_+ \in \mathbb{R}^{n \times m_{\text{knn}}}$. The presence of B_+ makes W_+ sparse since only the entries associated with an entry $+1$ in B_+ will have a weight different from zero. It follows that $S_+ = S \odot W_+$, where S_+ is the similarity matrix associated with the positive incidence matrix. Finally, an edge, if it exists, is selected for each row of S_+ as the edge associated with the maximum entry of S_+ along the considered row. Thus, considering the generic vertex i, i.e. row i of S_+, its positive edge e^i_+ is obtained as $e^i_+ = \{E_{\text{knn}}\}_{j^*}$, with $j^* = \text{argmax}(\mathbf{s^i_+})$, $s^{ij^*}_+ > \mu$, where with $\mathbf{s^i_+}$ is denoted the i-th row of S_+, with $\{E_{\text{knn}}\}_{j^*} \in \mathbb{R}^{1\times 2}$ the column vector at index j^* containing the indices of the source and target vertices and with μ a small threshold to avoid selecting edges with a very low similarity score.

Negative Edges

Following a similar discussion to the one of positive edges, let's define $B_- \in \mathbb{R}^{n\times m}$ as the negative incidence matrix where the entries $+1$ are set to zero, i.e. B_- contains values $\{-1, 0\}$. The entries of D can be filtered out based on B_- as $D_- = D \odot B_-$. The weight matrix associated with D_- can be defined as $W_- \in \mathbb{R}^{n\times m_{\text{knn}}}$ where only the entries associated with -1 in B_- are different from zero. A generic entry w^{ij}_- of W_- is obtained as $w^{ij}_- = 1 - \frac{d^{ij}_- - \min(\mathbf{d^i_-})}{\max(\mathbf{d^i_-})}$. It follows that $S_- = S \odot W_-$, obtaining S_- as the similarity matrix associated with the negative incidence matrix. Finally, an edge, if it exists, is selected for each row of S_- as the edge associated with the minimum entry of S_- along the considered row. The generic edge e^i_- is obtained as $e^i_- = \{E_{\text{knn}}\}_{j^*}$, with $j^* = \text{argmin}(\mathbf{s^i_-})$, $s^{ij^*}_- < -\mu$.

Edge Set

The obtained positive and negative edges are combined into a single edge set denoted as $\mathcal{E}$ with which the graph $\mathcal{G}$ is generated (Fig. 3.15c).

3.5.4 Intersections Processing

Although the graph $\mathcal{G}$ should contain vertices having a degree, i.e. number of neighbors, of only 1 or 2, depending on if the considered vertex is an endpoint, vertices having a higher degree, i.e. 3 or more, are still possible. This happens if the considered vertex is placed at the intersection area between multiple DLOs resulting in several ambiguous edge connections, e.g. Fig. 3.15d. To address this problem, Alg. 4 is employed: it detects the problematic vertices, extracts subgraphs around each of them, and by employing the cosine similarity approach it finds the correct edges.

With more details, Alg. 4 takes as input the graph $\mathcal{G}$ just created and provides as output the updated graph $\mathcal{G}'$ where the ambiguous vertices are removed and their edges redistributed correctly in their local subgraphs.

First, the ambiguous vertices are detected as those vertices with a degree larger than 2 and collected in $\mathcal{V}_{\text{int}}$, line 1. Then, for each v in $\mathcal{V}_{\text{int}}$, the neighbor vertices are collected (lines 2 to 5).

In case one or more vertices of one set of neighbors overlap with another one, those sets are merged (line 6) grouping all vertices and treating the problematic area as the composition of the original ones. Each set $\mathcal{N}$ of $\bar{\mathcal{N}}$ defines a subgraph around the problematic area.

For each subgraph defined by the vertices in $\mathcal{N}$, the number of connections (edges) to establish is determined by k_{conn} as the integer division between the cardinality of $\mathcal{N}$ and

Algorithm 4: Intersections Processing

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: $\mathcal{G}'$

1 $\mathcal{V}_{\text{int}} \leftarrow \{v \in \mathcal{V} : \deg(v) > 2\}$
2 $\bar{\mathcal{N}} \leftarrow \emptyset$
3 **foreach** $v \in \mathcal{V}_{int}$ **do**
4 $\quad \mathcal{N} \leftarrow \text{neighbors}(v)$
5 $\quad \bar{\mathcal{N}} \leftarrow \bar{\mathcal{N}} \cup \mathcal{N}$
6 $\bar{\mathcal{N}} \leftarrow \text{merge\_overlapping}(\bar{\mathcal{N}})$
7 $\mathcal{E}_{\text{new}} \leftarrow \emptyset$
8 **foreach** $\mathcal{N} \in \bar{\mathcal{N}}$ **do**
9 $\quad k_{\text{conn}} = |\mathcal{N}|$ **div** 2
10 $\quad \mathcal{C} \leftarrow \text{combinations}(\mathcal{N}, 2)$
11 $\quad \mathcal{Z} \leftarrow \text{edge\_solver}(C)$
12 $\quad \mathcal{Z} \leftarrow \text{sorted}(\mathcal{Z}, \text{"descending"})$
13 $\quad \mathcal{V}_{\text{done}} \leftarrow \emptyset$, $c \leftarrow 0$
14 $\quad$ **while** $c \leq k_{conn}$ **do**
15 $\quad\quad$ **foreach** $(v_i, v_j, s_{ij}) \in \mathcal{Z}$ **do**
16 $\quad\quad\quad$ **if** $v_i \notin \mathcal{V}_{done}$ *and* $v_j \notin \mathcal{V}_{done}$ **then**
17 $\quad\quad\quad\quad \mathcal{E}_{\text{new}} \leftarrow \mathcal{E}_{\text{new}} \cup (v_i, v_j)$
18 $\quad\quad\quad\quad \mathcal{V}_{\text{done}} \leftarrow \mathcal{V}_{\text{done}} \cup \{v_i, v_j\}$
19 $\quad\quad\quad\quad c \leftarrow c + 1$

2 (line 9). The combinations of 2 elements of the vertices contained in $\mathcal{N}$ are collected in the set $\mathcal{C}$ (line 10). These tuples of elements can be considered edge candidates for the subgraph. For instance, in Fig. 3.15e, the candidate edges of the subgraph under analysis are depicted in red (wrong) and green (valid). Thus, an edge solver (line 11) is employed to assign a score to each of those. In particular, given two sample vertices, i.e. v_1 and v_2, which connection should be scored, the direction of the edge connecting them is computed as $\mathbf{d_e^{1,2}} = v_1 - v_2$. Then, the connection cosine similarity score, as in eq. (6.2), is evaluated as

$$s_{\text{int}}(\mathbf{d_v^1}, \mathbf{d_v^2}, \mathbf{d_e^{1,2}}) = \left| s(\mathbf{d_v^1}, \mathbf{d_e^{1,2}})\, s(\mathbf{d_v^2}, \mathbf{d_e^{1,2}}) \right|$$

where with $\mathbf{d_v^1}$ and $\mathbf{d_v^2}$ the vertices orientations are denoted. Notice that the absolute value of the similarity is employed since the interest is not in its sign, but only in its magnitude. Each (v_i, v_j) of $\mathcal{C}$ is therefore augmented by the computed score s_{ij} as (v_i, v_j, s_{ij}) and collected by the set $\mathcal{Z}$ which is then sorted based on the score values in descending order (line 12). Finally, an interactive procedure takes place to loop through the elements of $\mathcal{Z}$ and collect the k_{conn} new edges into $\mathcal{E}_{\text{new}}$ as those defined by vertices not being already assigned to other edges (lines 13 to 19). The sample subgraph is solved by obtaining the final graph depicted in Fig. 3.17a.

3.5.5 DLOs Instances Extraction

The single instances of the DLOs present in the scene are retrieved considering the connectivity of the graph, i.e. each DLO is represented as an isolated sub-graph from the initial global graph. For each subgraph, the path from one endpoint (a vertex with degree 1) to the other is extracted. A path $\mathcal{P}_t$ can be denoted as an ordered sequence of vertices as $\mathcal{P}_t = \{v_1^t, v_2^t, \dots v_{t_n}^t\}$. The extracted path denotes the sequence of key points describing the DLO instance. From these key points, a spline curve can be fitted to better approximate the DLO shape and then an estimate of the DLO thickness can

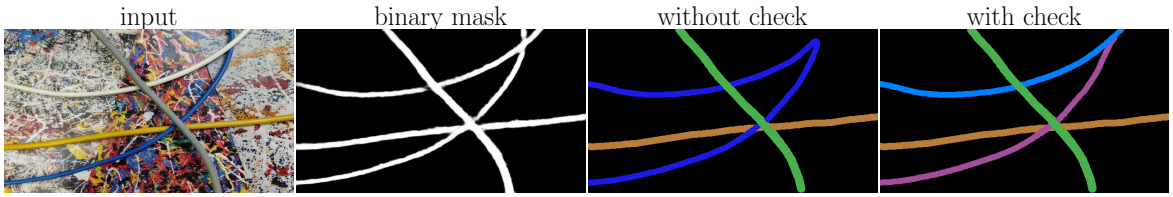


Figure 3.16: DLOs instances extraction with and without consistency check in case of a problematic mask.

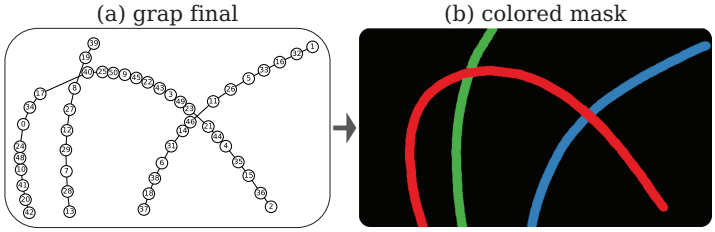


Figure 3.17: The connectivity graph (a) is processed to extract the DLOs instances and obtain the colored mask M_c (b).

be obtained from the distance transform mask M_{dist}. Thus, a colored mask M_c can be drawn as shown in Fig. 3.17.

In some cases, two or more DLO instances may be effectively denoted by a single path. This situation can occur in case, for instance, the intersection between two DLOs happens along the border of the image. *RT-DLO*, employing only the mask image, tries to solve this scene by connecting jointly the two distinct DLOs, see as an example Fig. 3.16 showing the obtained DLOs instances given the source image and mask. To handle this condition, as a final consistency check along the obtained path, the cosine similarity is computed between each vertex of the path and its two neighbors. In particular, given a sample vertex v_i^t, $i \in [2, t_n - 1]$ belonging to path $\mathcal{P}_t$. Its two neighboring vertices are v_{i-1}^t and v_{i+1}^t while the two edges directions are $\mathbf{d_e^{i,i-1}}$ and $\mathbf{d_e^{i,i+1}}$. According to eq. (6.2), the cosine similarity between $\mathbf{d_v^i}$ and $\mathbf{d_e^{i,i-1}}$ can be denoted as $s_{i,i-1} = s(\mathbf{d_v^i}, \mathbf{d_e^{i,i-1}})$, where $\mathbf{d_v^i}$ describes the orientation of vertex v_i^t. Similarly, $s_{i,i+1} = s(\mathbf{d_v^i}, \mathbf{d_e^{i,i+1}})$. If the product $s_{i,i+1}\, s_{i,i-1}$ is negative, it means that the path is not smooth at vertex v_i^t. Thus, the path $\mathcal{P}_t$ is detached at vertex v_i^t into two different paths, see Fig. 3.16.

3.5.6 Intersections Layout

To correctly assign the DLOs IDs in the intersection areas among two or more DLOs, additional color information is required. Indeed, only from the binary mask M_b and the corresponding constructed graph, this information is not achievable. In *RT-DLO*, the approach first described in [17] is deployed: the standard deviation of the RGB color along the edge connecting two vertices in the area of the intersection is used. For a given intersection, all the involved edges are collected and the standard deviation of the RGB values along the edges is compared. The edge corresponding to the smallest value is selected as the one placed at the top of the pile. Therefore, the mask M_c is drawn taking into account this information.

Table 3.1: *Ariadne+*, *FASTDLO* and *RT-DLO* versus baseline methods. Key-point denotes that the method also provides a representation of the detected DLOs as sequence of points. The symbol '*' indicates that the method is tested on a reduced dataset.

Group	Method	Backbone	Key-points	FPS ↑	Time [ms] ↓	IoU [%] ↑
general purpose	YOLACT	ResNet-50	✗	44	23	32.15
	YOLACT	ResNet-101	✗	32	31	35.25
	YOLACT++	ResNet-50	✗	42	24	29.96
	YOLACT++	ResNet-101	✗	31	32	29.64
	BlendMask	ResNet-50	✗	15	66	15.92
	BlendMask	ResNet-101	✗	12	81	21.24
	CondInst	ResNet-50	✗	16	62	23.29
	CondInst	ResNet-101	✗	13	78	29.24
DLO specific	Ariadne	-	✓	< 1	> 1000	*23.80*
	Ariadne+	ResNet-50	✓	3	354	73.96
	Ariadne+	ResNet-101	✓	3	360	76.87
	FASTDLO	ResNet-50	✓	23	44	73.89
	FASTDLO	ResNet-101	✓	22	46	77.77
	RT-DLO	ResNet-50	✓	36	27	77.65
	RT-DLO	ResNet-101	✓	32	31	79.91

3.6 Experimental Results

The proposed algorithms are evaluated on the *test set*, introduced in Sec. 2.6, addressing both general scenarios as well as industrial-related ones. The performances are compared both within the proposed algorithms and with general-purpose DCNNs, namely *YOLACT* [6], *YOLACT++* [5], *BlendMask* [22] and *CondInst* [114]. For these baselines, the synthetic dataset of Sec. 2.4 is employed for their training, since it allows the generation of a large number of images with a precise instance-wise annotation of the DLOs.

The evaluation of the performances is provided in Sec. 3.6.1 while the timings are in Sec. 3.6.2. Moreover, an analysis of the robustness of the proposed algorithms is presented in Sec. 3.6.3 and sensitivity to user-defined parameters for *RT-DLO* and *Ariadne+* are discussed in Secs. 3.6.4 and 3.6.5.

The experiments are performed employing, as the hardware platform, a workstation with an Intel Core i9-9900K CPU clocked at 3.60GHz and an NVIDIA GeForce GTX 2080 Ti. A video showcasing RT-DLO in action is available as supplementary material[1].

3.6.1 Performances Evaluation

The evaluation of the performances is addressed by employing as metric the Intersection over Union (IoU) defined as $\text{IoU} = \frac{|M \cap M_{gt}|}{|M| + |M_{gt}|}$. In the definition, M is the predicted mask of the DLOs instances and M_{gt} is the ground truth. In M, each DLO instance is denoted by a unique color and the IoU score is computed as the average score across the instances of the image.

The comparison of *Ariande+*, *FASTDLO* and *RT-DLO* against the baseline methods is presented in Tab. 3.1 by means of the IoU score computed starting from the color masks provided as output by each method. The table also provides details about the average inference time, FPS, and key-points availability as output.

The results show that the proposed algorithms outperform the baseline methods in terms of IoU score. In particular, *RT-DLO* shows the best performances with an IoU score of 79.91% and 77.65% when employing the ResNet-101 and ResNet-50 backbones respectively.

[1] https://www.youtube.com/watch?v=W4rfY-Ap_eE

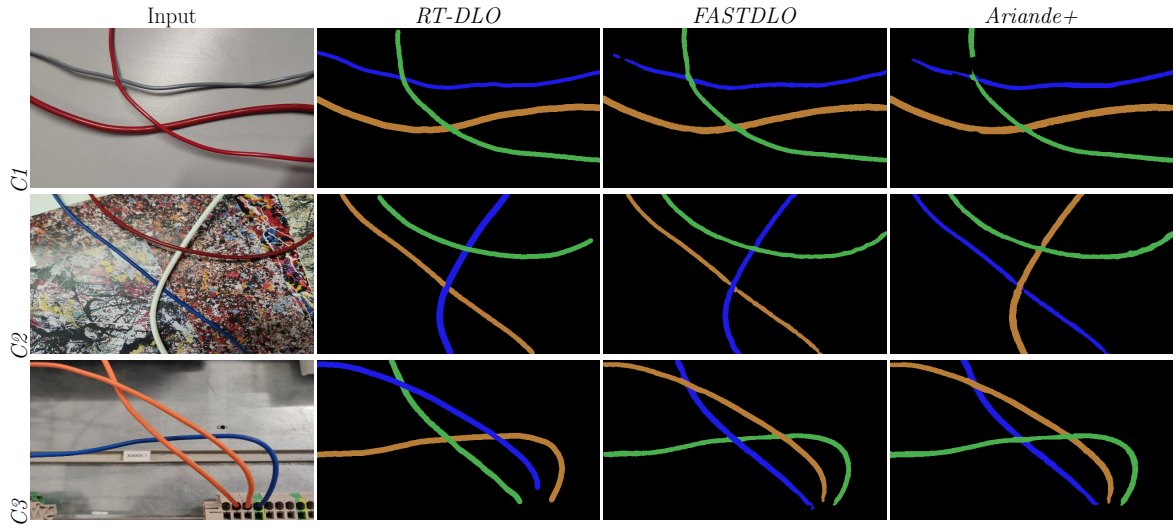


Figure 3.18: Qualitative evaluation of *RT-DLO* versus *FASTDLO* and *Ariadne+* on the *test set* classes.

Among the DLO-specific methods, clear improvements are shown by *Ariadne+* with respect to *Ariadne*, i.e. +50.16% and +62.07% improvements when employing the ResNet-50 and ResNet-101 backbones respectively. Instead, *FASTDLO* and *RT-DLO* show a slight improvement over *Ariadne+*, i.e. +3.91% and +2.05% improvements when employing the ResNet-50 backbone, and +0.91% and +1.04% improvements when employing the ResNet-101 backbone respectively. But, focusing on the efficiency, the differences between *Ariadne+*, *FASTDLO* and *RT-DLO* are more sharp. It is worth mentioning that *RT-DLO* can provide the same level of performance of *FASTDLO* employing a lighter backbone, thus making it possible to reach a frame-rate of 36 FPS, +13 FPS over *FASTDLO*.

A qualitative comparison on a few samples of the *test set* among *Ariande+*, *FASTDLO* and *RT-DLO* is provided in Fig. 3.18, where the superiority of *RT-DLO* is especially visible at the intersections. Indeed, the major advantage of *RT-DLO* against the other proposed approaches resides in its graph representation which is based on M_b but is less susceptible to the degraded area as opposed to the skeleton approach of *FASTDLO* and mask-guided superpixels method of *Ariadne+*. In this regard, deeper analysis on *RT-DLO* robustness capabilities is reported in Sec 3.6.3, where the requirement of an accurate segmentation mask M_b is experimentally relaxed.

3.6.2 Inference Time Evaluation

The inference time of the proposed algorithms is evaluated on the *test set* images. The results for *Ariadne+*, *FASTDLO* and *RT-DLO* are reported in Tabs. 3.2, 3.3 and 3.4 respecively. The timings are computed on the same hardware platform described at the beginning of Sec. 3.6, allowing a fair comparison among the algorithms.

As a segmentation network, *DeepLabV3+* with a ResNet-101 or ResNet-50 backbone is employed for all the algorithms. The inference time of the segmentation network is about 20 and 15 ms depending on the choice of the two backbones when processing the 640×360 images of the *test set*. Thus, for all methods, faster overall processing times can be achieved by deploying the lighter backbone ResNet-50, saving several milliseconds in the binary segmentation phase.

In Tab. 3.2, the timings of the different stages of *Ariadne+* are reported. The

Table 3.2: *Ariadne+* average execution timings of the different main parts and total computed over the test set. Values in milliseconds.

Procedure	Time [ms]
Superpixels Segmentation	167
Graph Generation	46
Graph Simplification and Clustering	47
Paths Discovery	20
Paths Layout Inference	17
Total	**342**

Table 3.3: *FASTDLO* main procedures average execution times and total with respect to the number of intersections in the *test set* images, i.e. 1, 2, and 3. Values in milliseconds.

Procedure	Number of Intersections		
	1	2	3
Skeleton Generation	12.27	13.26	14.25
Endpoint-pairs Predictions	0.80	1.04	1.21
Informed Merging	15.13	17.09	18.81
Total	**28.20**	**31.39**	**34.27**

Table 3.4: Average execution times [ms] of the main *RT-DLO* stages with respect to the number of intersections in the image, i.e. 1, 2, and 3.

Procedure	Number Intersections		
	1	2	3
Graph Generation	6.69	8.23	9.17
Intersections Processing	0.57	0.80	0.95
DLOs Instances Extraction	1.25	1.55	1.79
Intersections Layout	0.61	1.13	1.65
Output Colored Mask	0.37	0.45	0.51
Total	**9.50**	**12.15**	**14.07**

superpixels segmentation stage is the most time-consuming one, requiring about 167 ms. Overall, with a total processing time of 360 ms on average, *Ariadne+* is the slowest method of the group and is not suitable for real-time applications.

Tab. 3.3 reports the average execution times of the different stages of *FASTDLO* with respect to the number of intersections in the image. The total processing time excluding the initial segmentation is about 28 ms on average, with a maximum of 35 ms for highly complex scenes. The inference performed by the similarity network is very fast and does not suffer significantly from the increase in the number of intersections to process thanks to the possibility of employing batch inference. Compared to *Ariadne+*, *FASTDLO* is about 13 times faster, thanks mostly to the skeleton representation of the DLOs. *FASTDLO* can thus be considered a quasi-real-time algorithm, with a frame rate higher than 20 FPS considering also the initial semantic segmentation stage.

Concerning *RT-DLO*, Tab. 3.4 presents the timings of the different stages of the algorithm with respect to the number of intersections in the image, as for *FASTDLO*. The graph generation stage is the most time-consuming one, requiring about 8 ms. But, compared to the previous approaches, the graph representation of *RT-DLO* highlights its efficiency. Overall, the total processing time excluding the initial semantic segmentation is about 12 ms on average, with a maximum of 14 ms for highly complex scenes. If the colored mask is not required, for instance in the case of robotic manipulation tasks where the key-points representation is more useful, the last two stages can be skipped shortening the computation time by 1 to 2 milliseconds depending on the number of intersections, as shown in Tab. 3.4. A similar timing, of about 13.5 ms, is obtained deploying *RT-DLO* on a consumer laptop (Intel Core i7-12700H CPU). Indeed, high computation power is mostly required only for the DCNN. Thus, if the application does not require a complex deep model for scene semantic segmentation, the hardware specifications can be relaxed or higher overall FPS can be achieved.

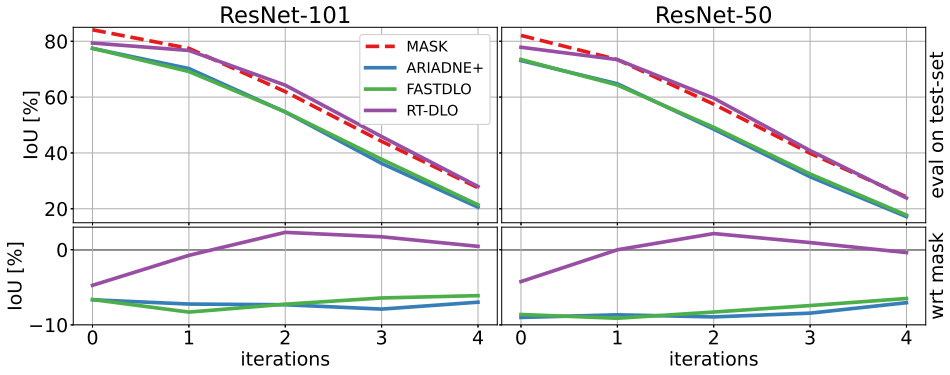


Figure 3.19: Evaluation of *RT-DLO*, *FASTDLO* and *Ariadne+* on the *test set* eroding M_b.

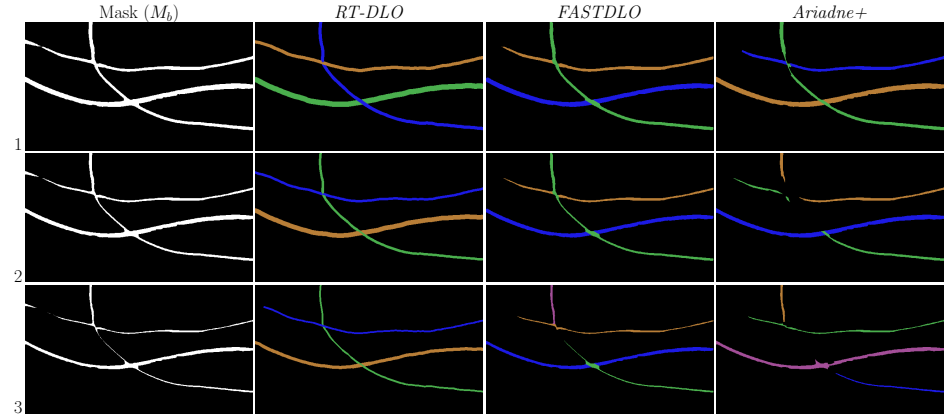


Figure 3.20: Qualitative comparison of *RT-DLO*, *FASTDLO* and *Ariadne+* given M_b eroded for 1, 2 and 3 iterations.

3.6.3 Masks Degradation and Different Segmentation Back-Ends

The improvements of *RT-DLO* against *Ariadne+* and *FASTDLO*, are not only in the form of faster processing time and better accuracy. Indeed, an important benefit of the fully graph-based representation approach of *RT-DLO* is its ability to better handle degraded semantic segmentation masks M_b. To illustrate the graph-based advantage of *RT-DLO*, a two-fold study is conducted. On one hand, the performance drop of the proposed algorithms is evaluated after an erosion process is applied on M_b. On the other hand, different segmentation networks trained on public datasets, i.e. not DLO-specific ones, are employed. Thus, the predicted masks M_b are not as accurate as the ones obtained with the DLO-specific segmentation network and the performance drop of the proposed algorithms is evaluated.

Masks with Erosion Process

Concerning the first study, the masks M_b of the *test set* are iteratively eroded, that is the process consisting in thinning the foreground area of a binary mask, with a kernel of 3×3 pixels to simulate the effects of less precise segmentation masks. The evaluation is performed by comparing *Ariadne+*, *FASTDLO* and *RT-DLO* on the masks obtained from the two different backbones, i.e. ResNet-50 and ResNet-101, see Fig. 3.19. From

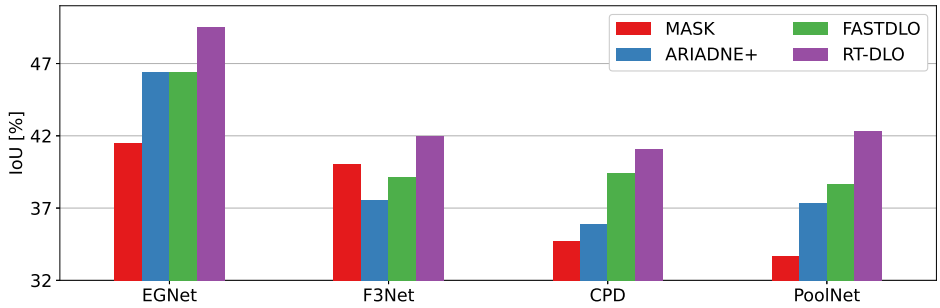


Figure 3.21: Evaluation of *RT-DLO*, *FASTDLO* and *Ariadne+* on the *test set* employing M_b obtained by SOS networks.

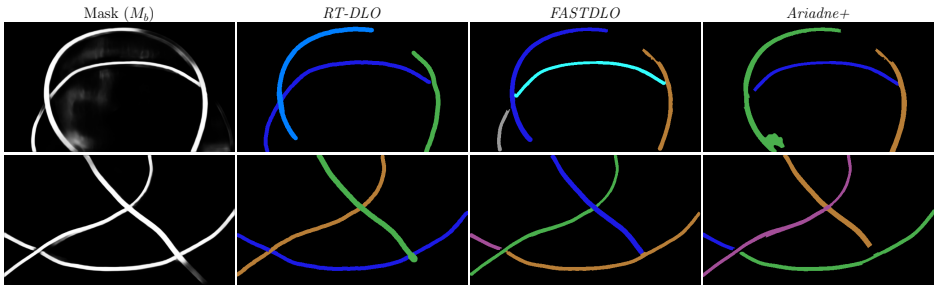


Figure 3.22: Qualitative comparison of the instances masks of *RT-DLO*, *FASTDLO* and *Ariadne+* given M_b from *EGNet*.

the plots of Fig. 3.19, *RT-DLO* shows the capability of maintaining an almost steady performance after the first round of the erosion process, followed by a drop in the scores in the subsequent iterations. On the contrary, the drop of scores associated with *FASTDLO* and *Ariadne+* is significant from the very first iteration. Considering the mask IoU score as an upper bound, *RT-DLO* is capable of maximizing its score as opposed to the compared approaches. The images of Fig. 3.20 allow to catch better the effects of the erosion process and the *RT-DLO* advantages on the test sample *C1* of Fig. 3.18.

Different Segmentation Networks

A study about replacing the DLO-specifically trained segmentation back-end with ImageNet pre-trained salient object segmentation (SOS) approaches is conducted in Fig. 3.21, thus avoiding the need for a specific DLO-based dataset and training procedure. The SOS architectures tested are: *EGNet* [146], *F3Net* [124], *CPD* [126] and *PoolNet* [68]. When evaluated on the *test set*, *RT-DLO* continues to achieve strong performances compared to *Ariadne+* and *FASTDLO*, see Fig. 3.21. The advantages of *RT-DLO* in the case of degraded masks are even more apparent for the sample images of Fig. 3.22 which show how *RT-DLO* is able to minimize the number of extracted instances.

3.6.4 RT-DLO: Parameters Sensitivity

RT-DLO employs two user-defined parameters that can affect the method performances, the vertex sampling ratio α and the number of K_{nn} nearest neighbors. In Tab. 3.5, the performances of *RT-DLO* on the *test set* are compared by varying α and K_{nn}.

Table 3.5: Performances of *RT-DLO* when varying the vertices sampling ratio α and the number of K_{nn} nearest neighbors. In bold the values within 1% distance from the best one.

K_{nn}	vertices sampling ratio α					
	0.05	0.1	0.15	0.2	0.25	0.3
4	51.20	75.06	77.10	76.70	75.87	75.32
8	57.98	78.19	**79.80**	**79.20**	77.50	77.38
16	56.68	78.72	**79.91**	**79.86**	78.86	78.83
32	56.31	77.78	**79.42**	**79.13**	78.79	78.25

RT-DLO maintains remarkably strong performances across a wide range of values for α, i.e. between 0.1 and 0.3. On the contrary, selecting α as 0.05 results in a quite reduced number of vertices, hurting the description power of the graph. The selection of K_{nn} is also not critical with a value of 8 already sufficient to reach top performances.

3.6.5 Ariadne+: Superpixels Parameters Sensitivity

As described in Sec. 3.3.2, *Ariadne+* relays on a superpixel segmentation algorithm to construct the graph representation. Although the employed algorithm to this end [1, 51] exposes several tunable parameters, the only parameter in the *Ariadne+* method that needs to be adjusted to extract the best result is the number of superpixels in the image. In all the experiments reported in Tab. 3.1, this value is fixed at 50. Instead, for the evaluation reported in Fig. 3.23, the number of superpixels is varied between 10 and 90 with a step of 10 to describe the effects on the overall scores. In particular, the analysis is carried out for each number of intersections highlighting how as the complexity of the scene increases, a higher number of superpixels improves the performance. The plot shows that *Ariadne+* can provide consistent results for a wide range of superpixel numbers, i.e. from 40 to 90, resulting in an almost steady IoU score. Generally speaking, a low number of superpixels oversimplifies the scene whereas an unnecessarily high value causes the introduction of false intersections. Fig. 3.24 provides examples for three conditions. It is worth mentioning that the sensitivity of the result to this parameter is somehow mitigated by the graph simplification described in Sec. 3.3.4.

3.7 Conclusions

In this chapter, the task of instance segmentation of DLOs in images has been analyzed. Three algorithms, namely *Ariadne+*, *FASTDLO* and *RT-DLO*, developed to accomplish this task robustly and efficiently have been proposed.

The *Ariadne+* algorithm is the first one developed and exploits a DCNN for scene semantic segmentation and a superpixel-based graph representation to extract the DLOs instances. The *FASTDLO* algorithm is the second one developed and replaces the superpixel approach with a skeleton-based method for improved efficiency. A similarity network is employed to predict the merging between different disjoint DLO sections. The *RT-DLO* algorithm is the last one developed and is based on a fully graph-based representation to extract the DLOs instances. The graph is preferred for its robustness against degraded segmentation masks and its efficiency.

The three algorithms have been evaluated on a dataset of images of DLOs in real-world scenes. The experiments allowed to catch the differences and drawbacks of each method. In particular, the *RT-DLO* algorithm showed the best performances in terms of accuracy, and efficiency. Moreover, the *RT-DLO* algorithm showed the best robustness against degraded segmentation masks and the possibility to employ

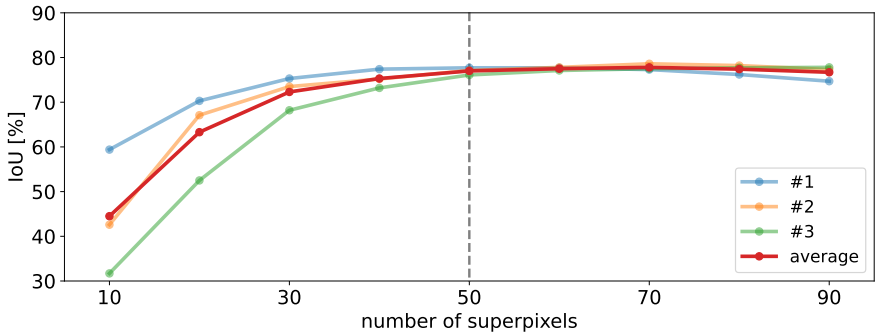


Figure 3.23: Analysis of superpixel parameter sensitivity in terms of the number of intersections (i.e. #1, #2 and #3) and average. The number of superpixels is varied between 10 and 90 whereas the reference value of 50 is shown with the dashed gray line. As metrics, the IoU is employed.

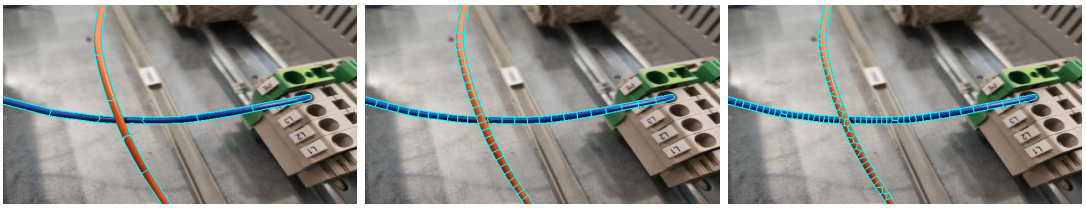


Figure 3.24: Qualitative comparison of three different values for the number of superpixels for the same image. From left to right: low (10), normal (50), high (90).

different segmentation networks trained on public datasets. Finally, the sensitivity of the *RT-DLO* and *Ariadne+* algorithms to a few user-defined parameters has been also evaluated, showing that both algorithms are able to provide consistent results for a wide range of values.

In future research endeavors, there are numerous opportunities for enhancing and extending the proposed algorithms. For instance, the *RT-DLO* algorithm could be expanded to incorporate a 3D graph representation, utilizing depth information derived from a 3D camera. Additionally, the use of video sequences, rather than individual images, could be explored as source media for all the methods. This would enable the utilization of information from previous frames, potentially leading to more efficient and rapid generation of segmentation masks. Furthermore, investigating a tracking system to match DLO instances across video sequences is also a promising direction for future investigation.

Chapter 4

3D Shape Estimation: Combining 2D Perception and Multiple Views

Robotic solutions require an accurate estimation of the 3D shape of DLOs for several tasks, such as grasping, manipulation, and tracking. In this chapter, the problem of 3D DLOs perception is analyzed and a multi-view stereo-based algorithm is proposed. First, 2D images of the scene are acquired from different viewpoints and processed according to Chap. 3. Thus, a spline-based matching followed by a triangulation procedure are employed for obtaining the 3D information.

4.1 Introduction

Sensors commonly employed for robotics applications are 2D cameras, 3D cameras, and laser scanners. Among these sensors, only 3D cameras and laser scanners are able to provide 3D information.

Laser scanners are usually employed in industrial applications where high accuracy is required. However, they are expensive, bulky, and heavy. Moreover, they are not able to detect transparent materials [45], such as in the case of medical hoses manufacturing, i.e. a type of DLOs. Among 3D cameras, it is possible to distinguish *passive* and *active* devices [55, 45]. The first group of cameras exploits the stereo vision principle and matching techniques to infer the depth of the scene [103]. The second group, instead, employs a projector to illuminate the scene with a special pattern [141] or a time-of-flight sensor to measure the time of flight of the emitted signal [42]. In both cases, a depth map is obtained and the 3D information is inferred.

Popular *active* camera options are from the RealSense and PrimeSense families [138]. These devices are classified as *general purpose consumer* cameras due to their affordable price. Despite their popularity, almost all *general purpose consumer* 3D cameras fail in perceiving thin objects like DLOs [26], irrespective of the specific 3D depth technology employed. The only category of 3D active cameras that can reliably detect the shape of very thin cylindrically shaped objects like DLOs with a diameter as low as 2-3 mm is the high-end one, consisting of devices like Zivid One+/Two and Photoneo MotionCam3D [26]. In fact, these devices can reach sub-millimeter depth accuracy, but, on the other hand, they show several limitations in terms of pricing, bulkiness, and working constraints. Thus, they are usually placed at a fixed position and not at the end-effector level, increasing the risk of occlusions and reducing the flexibility of the robotic application. If semi-transparent materials are taken into account

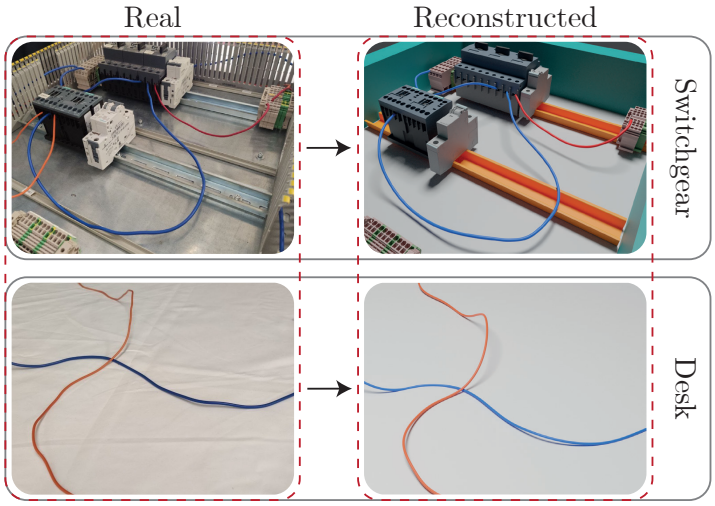


Figure 4.1: Showcase of *DLO3DS* capabilities in reconstructing the shapes of DLOs in different scenarios.

even high-end 3D sensors are not able to correctly detect those materials because of transparency, refraction and internal reflections [26].

In contrast, *passive* 3D devices, for instance 2D cameras arranged in a stereo (or multi-view stereo) setup, could potentially be more effective in detecting thin DLOs. However, these devices have limitations in terms of baseline (which is fixed and optimized for distant objects) and usually struggle in case of changes in lights and non-textured areas. DLOs, having small dimensions and lacking relevant textures, represent a difficult object to tackle for passive stereo cameras.

To address the drawbacks of both 3D active and passive cameras, a single 2D camera mounted on a robotic arm could be deployed. Utilizing just a 2D camera brings many beneficial effects including price, weight and size. Additionally, they have a wide range of resolutions and fields of view. Finally, with the 2D camera placed on the robotic arm, the high repeatability and accuracy of the latter can be exploited to avoid occlusions while, at the same time, enabling immense flexibility in terms of baselines and distances from the target.

In this chapter, a method to infer the 3D shape of DLOs in static scenes is presented. The method and results presented are based on the following publications [13, 14]. The approach makes use of the 2D perception algorithms discussed in Chap. 3 and exploits the mobility of the robotic arm to emulate a multi-baseline system. The proposed method is referred to as *DLO3DS* in the following. *DLO3DS* is able to reconstruct the 3D shape of DLOs from multiple images taken at known viewpoints without any prior knowledge of the DLOs or the surrounding scene, independently from the background. The DLO instances, after being modeled as B-spline curves, are matched by exploiting a triangulation-based method. *DLO3DS* provides reliable results where standard stereo-matching algorithms [48] fail due to the peculiar characteristics of DLOs previously discussed. Finally, the availability of the robotic arm is exploited by optimizing at run-time the baseline and distance from the target, thus reducing, even more, the estimation error. In Fig. 4.1 the capabilities of *DLO3DS* in two different scenarios are shown.

In the following, Sec. 4.2 reviews the closest contributions and methods dealing with multi-view stereo matching. Then, the proposed method is detailed in Sec 4.3, with the processing and estimation of a spline for each captured image in Sec. 4.4 and details on how the different k splines are matched and exploited for obtaining the final 3D shape

of a given DLO in Sec. 4.5. In Sec. 4.6, the experimental validation of *DLO3DS* in simulation and with real data is reported. Finally, Sec. 4.7 presents some final remarks and future research directions.

4.2 Related Works

The 3D shape reconstruction of objects from 2D images is a complex and extensively analyzed problem in computer vision. In this chapter, the concept of multi-view stereo is exploited for the 3D estimation of DLOs. The goal of multi-view stereo is to reconstruct a complete 3D object model from a collection of images taken from known camera viewpoints [37].

In order to achieve high accuracy in the 3D reconstruction, both the *disparity* error and the *geometric* error [39] should be tackled. The first is related to correspondence algorithms while the latter is related to physical parameters like baseline and distance from the objects. In the context of correspondence algorithms, stereo approaches are usually classified between local and global methods [103]. The latter are usually slower but more effective than the first in the case of non-textured areas. Among the many existing approaches, Semi-Global Matching (SGM) [48] is the most widely used approach due to its balance between quality, efficiency and scalability. However, its limitations in the case of non-textured areas are well-known [104] and several works have tried to address its weaknesses, such as time execution with a GPU implementation [47]. With the rise of deep learning, several approaches have been proposed for the computation of correspondence by employing SGM with, for instance, learned parameters [137] or learned matching cost [106]. A complete end-to-end learning approach [21] has been also proposed. Learning methods could potentially solve several challenges of traditional stereo algorithms, although the problem of dataset generation and model deployment in the real world remains to be evaluated.

Concerning the *geometric* error, it is not possible to adjust the baseline in case of a fixed stereo setup, as well as with commercial solutions. Thus, only the distance of operation (and possibly the resolution) can be modified for reducing the *geometric* error. In this context, some works exploit multiple 2D cameras mounted with different baselines to combine the advantages of short and wide baseline systems [49]. Other works employ a single 2D camera and a robot to emulate a multi-baseline system [28].

DLO3DS tackles both the *disparity* and *geometric* errors. The first is addressed by the reliable processing of the 2D images and the matching of splines. The latter is tackled by exploiting the robotic arm optimizing at run-time the baseline and the distance from the target.

4.3 The *DLO3DS* Algorithm: Overview

The 3D shape estimation pipeline of *DLO3DS* is illustrated in Fig. 4.2. The algorithm takes as input a set of k images of the scene, where each image contains one or more DLOs. *DLO3DS* consists of two main steps: (i) the processing and estimation of a spline for each captured image detailed in Sec. 4.4 and (ii) the matching of the splines and the triangulation of the 3D shape of the DLOs provided in Sec. 4.5.

The set of k images is processed by an instance selection algorithm for extracting the DLOs from each individual image. A B-spline model is computed for each detected instance (Sec. 4.4.1). Therefore, a single 2D target spline is selected from each image sample (Sec. 4.4.2). The set of selected B-splines is matched (Sec. 4.5.1) before performing the triangulation procedure (Sec. 4.5.2), obtaining a 3D spline describing the DLO shape

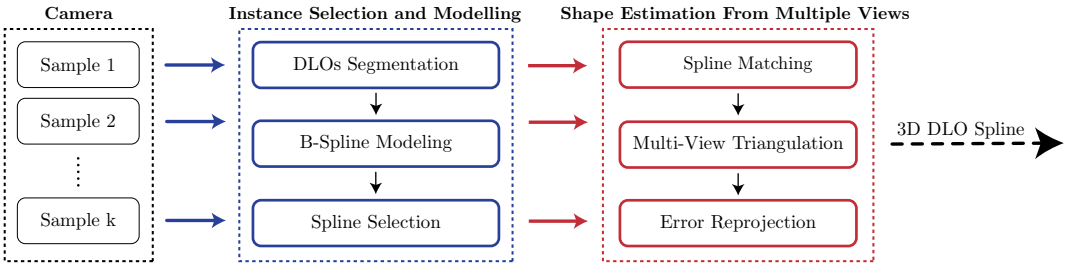


Figure 4.2: 3D shape estimation pipeline of *DLO3DS*. Data flow: the blue arrow denotes the image; the red arrow denotes the 2D spline.

in world coordinates. The 3D spline is then filtered to eliminate outliers and overlaps produced by subsequent acquisitions. Finally, the 3D spline is reprojected on each image and the difference with respect to the input 2D spline provided by the instance selection algorithm is computed (Sec. 4.5.3). The mean error norm is used to evaluate the quality of the estimation result. The availability of the robotic arm is exploited by optimizing at run-time the baseline and distance from the target, thus reducing, even more, the estimation error (Sec. 4.5.4). In the following, the details of the different steps of the algorithm are presented.

4.4 Instance Selection and Modeling

This section reports the details about the processing and estimation of a spline for each captured image. The estimated spline is employed both for computing the 3D shape of the DLO but also for aligning the camera with the target DLO main direction through principal component analysis. Indeed, in order to increase the portion of the same DLO visible in every sample, it is assumed to have the camera oriented along the DLO main axis and to record the samples by sliding orthogonally to it, see Fig. 4.3 for an example of the sliding direction with respect to the DLOs orientation.

4.4.1 DLOs Segmentation and B-spline Modeling

DLO3DS exploits existing approaches for segmenting the DLOs from an image. In this chapter, the learning-based algorithm named *FASTDLO* of Sec. 3.4 is employed, taking as input the RGB image of the scene and providing as output both an instance mask, where each DLO is denoted with a unique color identifying the assigned ID, and a sequence of 2D coordinates in the image plane for each detected DLO. A cubic B-spline is fitted to these coordinate points obtaining a continuous representation of the considered DLO. The considered spline is addressed as $q(u)$, where $u \in [0, 1]$ is the free parameter, i.e. the normalized position along the spline neutral axis. The computed curve is then discretized into a fixed number n_s of points. The utilization of a learning framework in *FASTDLO* allows to intrinsically deal with changes in lights and textureless areas, partially solving the limitations discussed in Sec. 4.1. However, other image processing pipelines can be employed for increased robustness and depending on the application scenario.

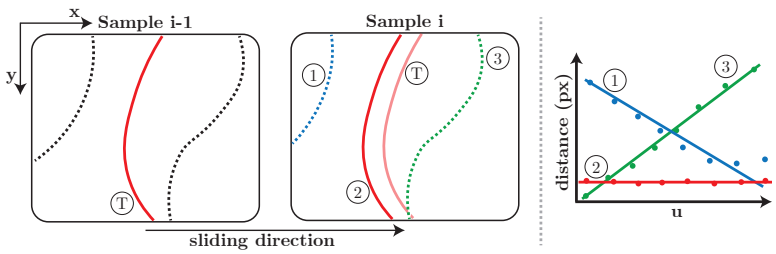


Figure 4.3: Target spline selection approach based on distance computation. The symbol u denotes the spline-free parameter.

4.4.2 Spline Selection

The spline selection is performed in case an image contains multiple DLOs. Indeed, all the instances extracted from an image are modeled in Sec. 4.4.1. However, in the following, a single DLO spline per image is expected. Thus, a regression-based distance approach is employed for retrieving the target DLO in sample i, based on sample $i-1$, with $i = 2, \ldots, k$. In particular, the point-to-point distance between the target spline T of a sample $i-1$ and each newly detected spline of a sample i is computed, as shown in Fig. 4.3. Then, a line is regressed for each distance curve and the spline associated with the smaller slope line is selected. Indeed, due to the orthogonal sliding direction, the same portion of DLO is assumed to be visible in each sample, thus an overall constant distance between the two curves given by the motion of the baseline step is expected.

4.5 Shape Estimation from Multiple Views

This section details how the different k splines are matched and exploited for obtaining the final 3D shape of a given DLO, see Fig. 4.2. In Sec. 4.5.1 the matching of the splines is discussed, while in Sec. 4.5.2 the triangulation approach is detailed. In Sec. 4.5.3 the possibility of employing the reprojection error for evaluating the quality of the estimation is presented. In Sec. 4.5.4, the optimization of the baseline and distance from the target is described. Finally, in Sec. 4.5.5, the applicability of *DLO3DS* in a DLO tracking framework is analyzed.

4.5.1 Splines Matching

A spline $q_i(u)$ can be sampled by defining a suitable vector u of n_s equally-spaced free parameter values in the interval [0, 1] (see Sec. 2.4.1). Thus, n_s 2D pixel points along the DLO for the i-th view are retrieved.

Let's denote with $p_{ij} = [p_{x_{ij}} \; p_{y_{ij}}]^T$ the j-th spline sample on the i-th image plane, with $i = 1, \ldots, k$ and $j = 1, \ldots, n_s$. To assess the accurate 3D location of a generic point seen from multiple images at pixel coordinates p_{ij}, the precise computation of the corresponding points p_{ij} is required. For this purpose, both the constraints embedded in the case of a normal stereo setup and the availability of the splines modeling the DLO are exploited.

The first step consists of sampling all the splines over the same DLO section by defining suitable vectors u_i, one for each spline. The length l_i of each spline is measured by summing the distance in pixels among adjacent points. Then, the index r of the

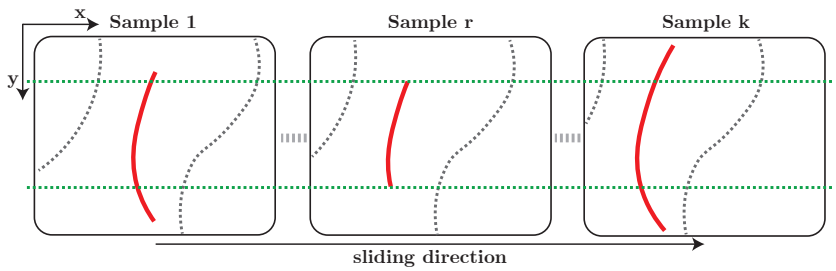


Figure 4.4: Scaling process. The shortest spline is selected as the reference and all the others are scaled to match the same DLO portion as closely as possible.

shortest spline is taken as a reference

$$r = \arg\min_{i} \{l_i : \ i \in 1, \ldots, k\}$$

Thus, the splines are re-sampled according to the redefined vectors of free parameters

$$u_i \leftarrow u_i \frac{l_i}{l_r} + \frac{d(q_i(0), q_r(0))}{l_i}$$

where the function $d(\cdot, \cdot)$ provides the distance in pixels between two points. As a consequence, the spline samples $q_i(u_i)$ provide a coarse matching across the different views.

The n_s spline samples of the shortest spline need to be precisely matched in all the other splines $q_i(u)$, $i = 1 \ldots k \setminus r$. In this regard, the corresponding j-th point on the i-th image plane p_{ij} is searched along the row coordinate of p_{rj}, empowering the basic constraints of epipolar lines in case of a normal stereo rig, as the intersection point with the spline $q_i(u)$. In the eventuality of multiple matches between the spline curve and the epipolar line, a smoothness constraint is also employed enforcing the most consistent point based on the past matches.

The aforementioned procedure is depicted in Fig. 4.4. The spline samples p_{ij} are then used to compute the DLO 3D shape as detailed in Sec. 4.5.2.

4.5.2 Multi-View Triangulation

For the sake of simplicity, the discussion is focused first on just one target point, i.e. $j = 1$. Let us consider the case in which a single unknown point p in the Cartesian space expressed with respect to the world reference frame is observed by the camera mounted on the robot from multiple points of view. Provided that the camera frame with respect to the world frame at the i-th points of view is

$${}^{w}T_{c_i} = \begin{bmatrix} {}^{w}R_{c_i} & {}^{w}t_{c_i} \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

where ${}^{w}R_{c_i}$ is the rotation matrix and ${}^{w}t_{c_i}$ is the position of the camera frame origin in world coordinates obtained from the kinematics of the robot and the extrinsic parameter of the camera calibration. It is assumed that the point p is seen in the image related to the i-th points of view at $p_i = [p_{x_i} \ p_{y_i}]^T$, being p_{x_i} and p_{y_i} the point pixel coordinates in the image.

A so-called unit ray v_i passing through the image reference frame origin and p can be expressed in the image frame considering the pixel coordinates p_i and the camera focal distance f

$$v_i' = \begin{bmatrix} p_{x_i} - c_x \\ p_{y_i} - c_y \\ f \end{bmatrix}, \quad v_i = \frac{v_i'}{\|v_i'\|}$$

where c_x and c_y are the pixel coordinates of the image center (assuming the camera frame is centered with respect to the image). Then, v_i can be expressed in the world frame by

$${}^w v_i = {}^w T_{c_i} v_i$$

Provided that k distinguished points of view are available, the estimation $\tilde{p}$ of the unknown point p can be obtained by looking for the point having the minimum distance from all the rays. By defining the symmetric V_i matrix

$$V_i = I - {}^w v_i {}^w v_i^T \tag{4.1}$$

providing the semi-norm on the ray distance, the point location estimate $\tilde{x}$ is provided by the nearest point search algorithm, i.e.

$$\tilde{p} = \left(\sum_{i=1}^{k} V_i \right)^{-1} \left(\sum_{i=1}^{k} V_i \, {}^w t_{c_i} \right)$$

The aforementioned algorithm is thus applied to estimate the DLO segment employing as input the spline samples $p_{ij} = p_i(u_j)$, $j = 1, \ldots, n_s$, $i = 1, \ldots, k$. The vector of control points $q_v = [q_1 \ \cdots \ q_{n_s}]^T$ of the 3D spline $q(u)$ that optimally approximated the set of point estimates p_{ij} can be defined as

$$q_v = B^{\#} \tilde{x}_v$$

where $^{\#}$ represents the matrix pseudo-inverse and

$$B = \begin{bmatrix} b_1(u_1) & \cdots & b_{n_u}(u_1) \\ b_1(u_2) & \cdots & b_{n_u}(u_2) \\ \vdots & \vdots & \vdots \\ b_1(u_{n_s}) & \cdots & b_{n_u}(u_{n_s}) \end{bmatrix}$$

$$\tilde{x}_v = \begin{bmatrix} \left(\sum_{i=1}^{k} V_{i1} \right)^{-1} \left(\sum_{i=1}^{k} V_{i1} \, {}^w t_{c_i} \right) \\ \left(\sum_{i=1}^{k} V_{i2} \right)^{-1} \left(\sum_{i=1}^{k} V_{i2} \, {}^w t_{c_i} \right) \\ \vdots \\ \left(\sum_{i=1}^{k} V_{in_s} \right)^{-1} \left(\sum_{i=1}^{k} V_{in_s} \, {}^w t_{c_i} \right) \end{bmatrix}$$

being V_{ij} the matrix computed according to eq. (4.1) for the j-th sample provided by the i-th image.

4.5.3 Evaluation of Estimation Error by Reprojection

To evaluate the estimation error, the 3D DLO B-spline obtained in Sec. 4.5.2 is reprojected on each image and the difference with respect to the input 2D spline provided by Sec. 4.4.1 is computed. Considering a generic 3D spline sample $q(u_j) = B\,q_v$, its homogeneous representation is provided by $\bar{q}(u_j) = [q(u_j)^T\ 1]^T$. The projected coordinates $\tilde{p}_{ij} = [\tilde{p}_{x_{ij}}\ \tilde{p}_{y_{ij}}]^T$ of the j-th spline sample on the i-th image plane can be written as

$$\tilde{p}'_{ij} = \begin{bmatrix} \tilde{p}'_{x_{ij}} \\ \tilde{p}'_{y_{ij}} \\ \tilde{p}'_{z_{ij}} \end{bmatrix} = A\,[{}^{w}R_{c_i}^{T} \mid -{}^{w}R_{c_i}^{T\,w}t_{c_i}]\,\bar{q}(u_j)$$

$$\tilde{p}_{ij} = \begin{bmatrix} \tilde{p}_{x_{ij}} \\ \tilde{p}_{y_{ij}} \end{bmatrix} = \begin{bmatrix} \tilde{p}'_{x_{ij}}/\tilde{p}'_{z_{ij}} \\ \tilde{p}'_{y_{ij}}/\tilde{p}'_{z_{ij}} \end{bmatrix}$$

where

$$A = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

is the camera matrix containing the camera intrinsic parameters, such as the focal length f and center point coordinates c_x and c_y. Then, the overall error is provided by collecting all together in a single vector the error related to every single image, i.e. $e = [\cdots\ e_{ij}\ \cdots]^T$, $j = 1, \ldots, n_s$, $i = 1, \ldots, k$, where $e_{ij} = \|p_{ij} - \tilde{p}_{ij}\|$ is the distance between the corresponding initial spline sample provided by Sec. 4.4.2 and the projection on the image plane of the estimated 3D spline sample. Finally, the mean error norm $\|e\|_{n_s k} = \sqrt{e^T e}/(n_s\,k)$ can be used to evaluate the quality of the estimation result.

4.5.4 Online Reconstruction Optimization

In a general stereo setup, the two sensors are fixed and, as a consequence, their baseline can not be modified. Here, instead, the mobility of the robot can be exploited in order to find the best baseline and distance from the object corresponding to the minimum depth error. Indeed, both the baseline b and the distance from the target object z are responsible for the overall depth estimation error arising in triangulation methods, with the well-known relationship [39]:

$$\epsilon = \frac{z^2}{b\,f}\,\epsilon_d \tag{4.2}$$

where ϵ denotes the depth error, f the focal length of the camera and ϵ_d the disparity error (assumed to be within one pixel in the following). Thus, given a set of points in the 3D space $p : \{p_i = (x_i\ y_i\ z_i)^T,\ i \in [1, n_s]\}$, the optimization problem aiming at minimizing the depth error can be implemented, having the following cost function:

$$\min_{\delta_z, b} \frac{1}{n} \sum_{n=0}^{n_s} \frac{(z_i + \delta_z)^2}{b\,f}$$

where δ_z denotes the camera distance increment from the object, a value that can be either positive or negative.

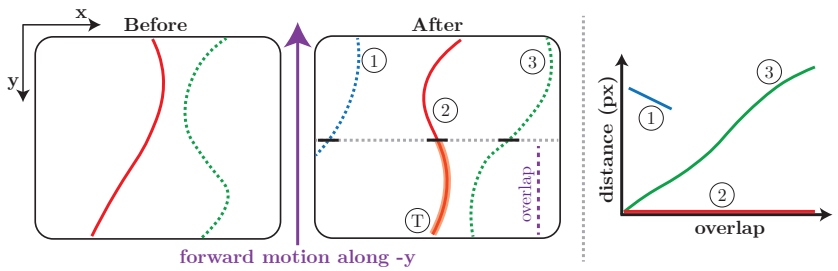


Figure 4.5: Tracking the same DLO after a forward motion by exploiting a distance-based computation on the overlap area.

This multi-variable optimization problem is subjected to a set of bounds and constraints that limit the admissible search space. The bounds are thus defined as :

$$b \geq 0$$
$$\delta_z^{\min} \leq \delta_z \leq \delta_z^{\max}$$

whereas the constraints are :

$$z_{\min} - z_i \leq \delta_z$$
$$z_i \left(-p_{x_i} + \sigma\right) \leq b\, f + \delta_z \left(p_{x_i} + \sigma\right) \tag{4.3}$$
$$b\, f + \delta_z \left(p_{x_i} + \sigma - w\right) \leq z_i \left(-p_{x_i} - \sigma + w\right) \tag{4.4}$$

where p_{x_i} is the row pixel value corresponding to the 3D coordinate p_i, σ is a safe offset in pixel coordinates to avoid regions of the image near the borders, w is the image width and $z_{\min}$ denotes the minimum distance of the camera from the 3D point. The solution to this minimization problem provides the optimal pair of baseline b and camera distance increment δ_z. Notice that eq. (4.3) and (4.4) restrict the value of the parameters b and δ_z such that all the points p are inside the k images taken using the optimal parameters.

The optimization routine requires as input an initial guess of the depth values z_i. Thus, a coarse guess should be utilized or an initial execution of *DLO3DS* with fixed default parameters for b and δ_z is required for computing the initial guess. Moreover, in the case of tracking the DLO shape (see Sec. 4.5.5), the values of the previous section can be used as an initial guess.

4.5.5 Tracking

In order to achieve a precise estimation of the DLO shape, *DLO3DS* is executed with camera samples captured in the proximity of sections of the DLO, e.g. the depth error is proportional to z (eq. (4.2)). Thus, if the estimation of a long DLO shape is sought, a different approach is required. In this section, the steps employed for applying *DLO3DS* in a tracking framework are described. Thus, the full 3D shape of a DLO combining individual estimations of small sections is achieved. In particular, after the estimation of a given section of the DLO, the camera is moved forward along the DLO principal direction and centered with respect to the estimated points. Thus, based on the overlap parameter n_o, a given percentage of previous points are still visible in the next DLO section and they are used for keeping track of the DLO under reconstruction, even in the presence of multiple DLOs in the scene, as shown in Fig. 4.5.

Upon completion of the tracking process along a DLO, the 3D points calculated for each segment of the analyzed DLO are aggregated into a unified vector. This collective

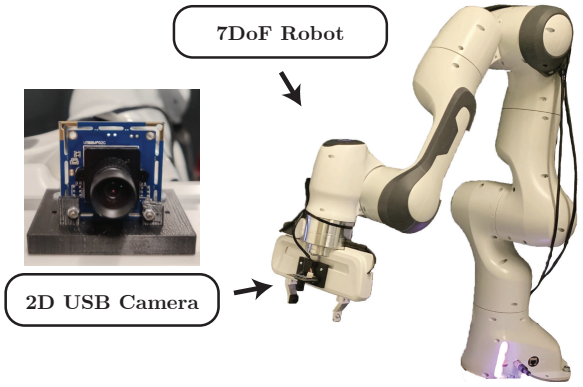


Figure 4.6: Experimental setup composed of a Panda robot from Franka Emika and a low-cost eye-in-hand 2D USB camera.

data is then employed to generate a single spline curve capable of representing the overall 3D shape of the DLO.

Furthermore, to enhance the accuracy of the estimation, these points undergo a filtering process to remove any outliers and overlaps that might result from successive acquisitions. For this purpose, the LOWESS algorithm [25], a locally weighted regression method that functions by defining a window in the sample data, is employed for the ultimate point filtering.

4.6 Experimental Validation

DLO3DS is validated experimentally employing a 7DoF robotic arm, the Panda from Franka Emika, equipped with an eye-in-hand 2D low-cost camera having a resolution of 640×480 pixels. The camera is both intrinsically and extrinsically calibrated, as shown in Fig. 5.6. The experiments are performed both with simulated and real data, in Sec. 4.6.1 and Sec. 4.6.2 respectively. Moreover, in Sec. 4.6.3, *DLO3DS* is characterized in terms of processing time. A video of the experiments is available as supplementary material[1].

4.6.1 Evaluation in Simulation

To perform a proper evaluation of *DLO3DS*, ground truth data is needed. Considering that it is quite difficult to obtain an error-free 3D ground truth shape of a real DLO, synthetically generated data [31] is exploited to assert the *DLO3DS* performances. Thus, a *test set* of 10 randomly shaped reference synthetic DLOs of 0.8 meters in length is generated resembling the shape and appearance of real DLOs. They are accompanied by ground truth data in the form of 3D points describing their center line.

Influence of DLO3DS parameters and DLO diameter

The *test set* is rendered using three different reference diameters $\phi = 2.0$, 3.5, and 5.0 mm to analyze how the DLO thickness may affect the performance of *DLO3DS*. In addition, the influence of the number of views, the percentage of overlap during the tracking (Sec. 4.5.5), and the contribution of the online optimization approach compared

[1] https://www.youtube.com/watch?v=beS9JgxJem8

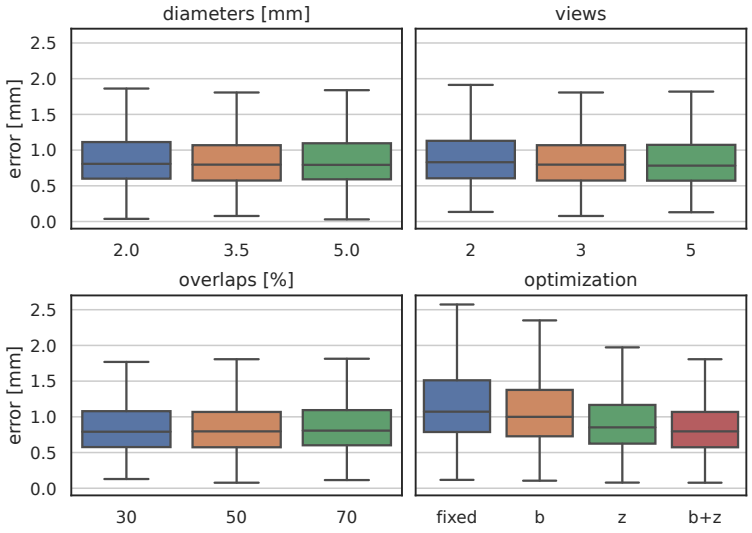


Figure 4.7: Error distribution on the synthetic *test set* when varying a single parameter of *DLO3DS*. For the optimization plot: $fixed$ means fixed setup, b means just baseline, z means just distance, $b + z$ means both baseline and distance.

to a fixed stereo parameter setup or a partial optimization is analyzed. When otherwise not specified, the default values are cable diameter 3.5 mm; number of views 3; overlap 50%; optimization of baseline and distance from the object ($b+z$). With default settings, the estimation mean error is 0.821 mm whereas the reprojection mean error is 0.731 pixels.

The box plots resulting from this analysis are depicted in Fig. 4.7, where the error values are computed on the *test set* samples between the estimated 3D points and the ground truth ones. From the plots, it is possible to conclude that the diameter of the DLO does not play a major role in the estimation error. The same can be said for the overlap percentage, with the only remark that in a real estimation, a bigger overlap may help to compensate for calibration errors. On the contrary, the slight drop in the error between 2 and 3 views is noticeable. Indeed, *DLO3DS* is commonly deployed using 3 views since the increase in the algorithm processing time is negligible and can be mostly compensated by its execution in masked time, as detailed in Sec. 4.6.3.

Ultimately, online optimization does play a major role in bringing the interquartile range of the reconstruction error between 1 and 0.5 mm. The contribution of optimizing just the baseline corresponds to an error drop of 9 % compared to the fixed setup. Instead, the optimization of the camera distance provides a drop of 22 %. The joint optimization makes the error drop of 29 %. The major relative improvement of z as opposed to b compared to the fixed setup is due to the changing of the *virtual* baseline, i.e. the baseline virtually increases when the camera is moved closer to the object. Thus, in the z experiment there is an actual minor coupling with b making its result closer to the $b + z$ configuration.

Comparison with Baseline Methods

A comparison between *DLO3DS*, established methods like Semi-Global Matching (*SGM*) [48], and more recent approaches like *SISTER* [28] is provided by rendering the sample number 1 of the *test set* with different backgrounds and colors. For the estimation

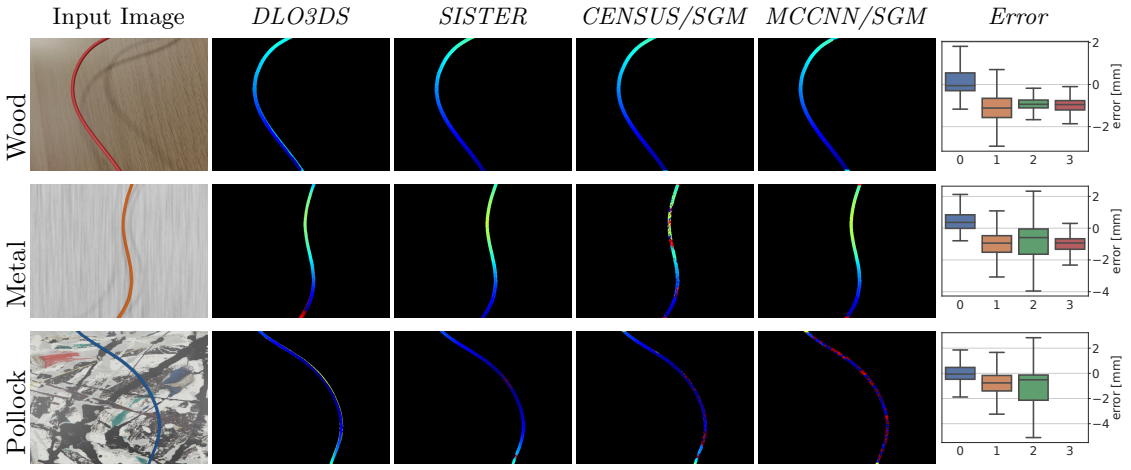


Figure 4.8: Comparison with baseline methods in the form of depth images and boxplot. Plot legend: 0) *DLO3DS*, 1) *SISTER* [28], 2) *CENSUS/SGM* [48] and 3) *MCCNN/SGM* [137]. The display of the *MCCNN/SGM* boxplot in the third row is avoided due to large errors.

performed by *DLO3DS*, 3 views are used. Concerning the *SGM* method, just 2 views are employed and the matching cost is computed one time via Census Transform (denoted as *CENSUS/SGM*) and a second time via a learned similarity measure [137, 102] (denoted as *MCCNN/SGM*). Finally, for *SISTER*, 5 views are employed as detailed in [28]. Aiming at a fair evaluation, in all the experiments the baseline was set to 25 mm, and the not-optimized fixed setup was employed, see Sec. 4.6.1.

Fig. 4.8 shows the computed depth images normalized between the min and max values of the ground truth one. Both *SISTER* and *SGM* provide as output a disparity image. Thus, those images are converted into a depth image given the known baseline and focal length. Instead, *DLO3DS* provides as output just 3D points describing the DLO center line. In order to compute the depth image, the estimated 3D points and the colored mask of Sec. 4.4.1 are used to first estimate the radius of the DLO in world coordinates. Then, the original center line description is over-sampled and used to reconstruct the DLO surface keeping into consideration its radius. The result is a dense depth image of the DLO. For a fair comparison, the methods are evaluated only for what concerns the depth values belonging to the DLO, the ground truth mask was used to select those points. The error between each method and the ground truth depth is computed by subtracting the latter from the first and it is shown using a box plot capturing the error distribution. From the figure, it is clear that *DLO3DS* provides an overall better estimate of the depth with wrong estimates only along the DLO boundaries due to prediction error in the segmentation mask.

4.6.2 Real-world Evaluation

To establish a boundary value of the estimation error in a real application, an experiment is performed using two types of purposely designed gripper fingers that, once closed, provide a hollow circle with a diameter of 6 and 10 mm respectively. First, *DLO3DS* is applied to estimate the DLO shape, then the center of the circumference is used as the reference frame for the generation of the motion: the robot should successfully follow the DLO without touching it, despite the shape of the DLO and changes in the z values. For the sake of generality, this experiment is performed both with electrical

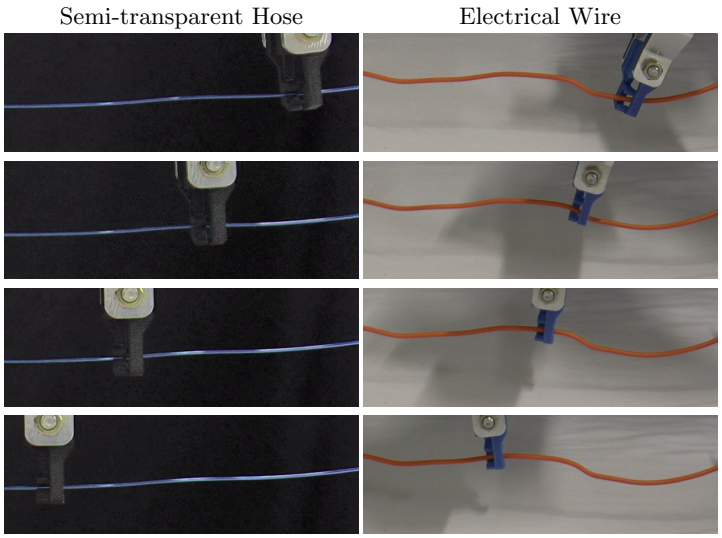


Figure 4.9: Key-frames from a video sequence (available as supplementary material) showing the tracking test performed on DLOs of different types and diameters. Tester gripper diameter: black 6 mm, blue 10 mm.

cables having a diameter of 3.5 mm and also with a different type of DLO, a polymeric hose for medical applications with an external diameter of 1.2 mm. The material of this hose is semi-transparent, such that it is almost invisible to commercial 3D sensors (including high-end ones) and laser scanners [26]. In Fig. 4.9, key-frames from a video sequence showing the experiment are reported. The cable of 3.5 mm is tested with the 10 mm gripper, while the hose with the one of 6 mm. Despite the complexity of the task, the cheap 2D camera used in this work is able to provide a reliable reconstruction of the sample objects, allowing for correct tracking without touching in both experiments.

4.6.3 Timings

The execution timings of *DLO3DS* are affected, other than the specific computing resources used, by the instance segmentation, modeling and selection performed in Sec. 4.4, and by the triangulation procedure of Sec. 4.5. The timings of the first are mostly correlated to the choice of the image processing algorithm. By employing *FASTDLO* (Sec. 3.4), 20 FPS are guaranteed for processing a single image when deployed on a workstation equipped with an Intel i9-9900K CPU and Nvidia 2080Ti. The performances of the triangulation procedure of Sec. 4.5 is affected by the number of points (n_s) at which the spline is evaluated. The following values are obtained for some configurations: $n_s = 10$, 7.5 ± 3.3 ms; $n_s = 20$, 19.5 ± 9.3 ms; $n_s = 40$, 27.2 ± 12.1 ms. Overall, *DLO3DS* provides competitive performances. It is worth mentioning that the data processing on a real setup can be mostly executed in masked time while the robot is moving toward the next pose.

4.7 Conclusions

In this chapter, an algorithm for the 3D shape estimation and tracking of DLOs, dubbed *DLO3DS*, is presented. *DLO3DS* leverages multiple 2D acquisitions within a multi-view stereo framework to achieve precise 3D shape estimation of DLOs. This algorithm

serves as a fundamental tool for enabling the manipulation of DLOs by robots, all without the need for costly, bulky, and restrictive 3D sensors. Consequently, *DLO3DS* holds particular significance for industrial applications seeking cost-effective and efficient solutions to intricate manufacturing tasks that involve the handling of cables, hoses, wires, ropes, and similar objects.

As it stands, *DLO3DS* primarily deals with static scenes, where the DLOs remain stationary between image acquisitions and may be influenced by the quality of the extracted splines. Consequently, promising avenues for future research include: 1) addressing dynamic scenes by developing a tracking algorithm for DLOs, and 2) enhancing the spline extraction algorithm to better handle complex and cluttered scenes.

Chapter 5

Shape Control Task with Online Model Parameters Estimation

Manipulating DLOs is a challenging task for a robotic system due to their unpredictable configuration, high-dimensional state space and complex nonlinear dynamics, as already outlined in Chap. 1. In this chapter, a framework is presented that targets the task of manipulating DLOs to achieve a target shape via a model-based approach. During the task execution, an online gradient-based estimation of model parameters is performed.

5.1 Introduction

Robotic solutions involving DLOs are highly complex, presenting various challenges from the perspectives of perception and manipulation, as detailed in Chap. 1. The first has been thoroughly investigated in Chaps. 2, 3 and 4.

Regarding the manipulation, DLOs prove to be challenging due to their unpredictable configuration behavior, high dimensional state-space, and complex nonlinear dynamics [134, 130]. Therefore, a deep understanding of their physical characteristics is essential for predicting and controlling their shape effectively [77].

In this chapter, a manipulation framework exploiting a physical prior of DLOs dynamics is proposed. The method and results presented are based on the following publication [18]. Based on an analytical DLO model, a data-driven learned model of the DLOs' dynamics is developed to predict the DLO behavior under manipulative actions. The prediction is made using a NN trained to approximate the dynamics of a class of DLOs, by conditioning its predictions on the set of the analytical model parameters.

First, the DLO's dynamics is modeled by a set of differential equations describing the DLO as point masses connected by axial and torsional springs [76], obtaining a so-called analytical DLO model. Then, the analytical DLO model is used to generate a comprehensive dataset by systematically sampling a variety of model parameters, diverse DLO configurations, and various manipulation actions. Consequently, a NN is trained utilizing this generated dataset, as visualized in the training phase depicted in Fig. 5.1. Notably, the NN is conditioned over the model parameters, such that it can be easily adapted to match different real-world DLOs.

The obtained NN model is employed during the online phase in Fig. 5.1 to estimate the manipulation actions to steer the DLO from its initial to a final target configuration, performing a task commonly referred to as shape control. In this context, the adoption of the network model is preferred over the analytical model due to its computation efficiency, stability, and scalability.

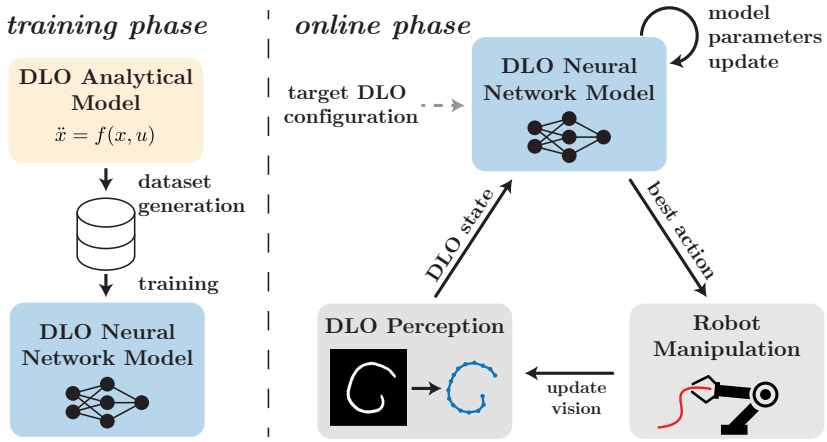


Figure 5.1: Schematic view of the two phases composing the proposed manipulation framework: 1) training phase for dataset generation and NN training, and 2) online phase for simultaneous estimation of the best action and model parameters during the shape control task.

The proposed method employs a gradient-based approach to estimate the best manipulation action to achieve a desired target configuration, by minimizing the error between the network prediction and the desired DLO configuration. A similar gradient-based approach is exploited to estimate the model parameters of the DLO, where the estimation is performed by minimizing the error between the model prediction and the observed real-world DLO configuration obtained after a manipulation action.

The unique characteristics of the proposed framework enable its direct application to various DLOs being manipulated on diverse surfaces, eliminating the need to: generate every time new task-specific data as in [130], introduce complex online adaptation controllers as in [134, 121], perform cumbersome and not intuitive parameters identification procedures as in [77, 67]. In summary, the main characteristics of the proposed framework are: 1) NN-based DLO dynamics approximation conditioned on several analytical model parameters allowing adaptability to different real-world scenarios; 2) efficient differentiable formulation to perform a gradient-based optimization aiming for either action prediction or parameters estimation employing the same learned NN model;

In the following, the related works are discussed in Sec. 5.2. The DLO analytical model and the DLO state representation are introduced in Sec. 5.3.1. The NN model is presented in Sec. 5.4 whereas the gradient-based estimation of the action and model parameters is discussed in Sec. 5.5. The shape control task with online parameters adaptation is presented in Sec. 5.6. The experimental evaluation of the method is provided in Sec. 5.7 and the conclusions are drawn in Sec. 5.8.

5.2 Related Works

5.2.1 DLO Shape Control Task

The term shape control of DLOs is typically used to refer to two different manipulation problems targeting the achievement of a final target shape: 1) the manipulation of a *soft* DLO with a sequence of pick and place actions [128, 63, 142], where the deformation of the DLO is held in place by the friction of the surface underneath. 2) the manipulation of *elastic* DLOs with one or more robotic arms and/or one end of the DLO fixed [134, 121, 130, 62, 58], where the second arm is used to achieve better control of the shape, e.g.

in the situation where the DLO's stiffness and the object exhibit rigid or plastic behavior. The outcome of the task can be assessed by comparing the achieved configuration to the target one in two possible ways [61]: by measuring their *relative similarity*; by evaluating their *absolute similarity* (i.e. a more challenging alternative considering also the final positioning in space). In this chapter, the latter approach is employed.

5.2.2 Model-free Approaches

One of the popular approaches for manipulating DLOs is to use methods that do not require nor create DLO models. Examples of these methods are those based on expert demonstration. In [113], shape similarity is used to determine which human demonstration should be replayed by the robot to achieve the goal. Instead, in [95], human demonstrations were used to create a set of control primitives from which one can build the manipulation plan. Whereas in [112], human expertise was used to learn the DLO manipulation policy. This kind of policy can be also learned without supervision in a reinforcement learning paradigm, however, it is typically done only in simulation [61], which limits the potential application to the real DLOs. To approach this *reality-gap* and improve the robustness, authors of [83, 151, 2] focused on the online adaptation of the DLO control strategy. In the case of [2], the parameters of the controller were adjusted online based on the tracking error. Whereas in [83, 151], the control law relies on the Jacobian that locally approximates how the movement of the grippers affects the DLO. These methods, which utilize local approximations of DLO motions and online adaptation of controller parameters, have the potential to enhance the system's manipulation abilities in the context of model-free approaches. Nevertheless, model-based approaches can typically strengthen the system's robustness through its better generalizability to diverse scenarios.

5.2.3 Learned DLO Models

The literature related to learning-based methods can be divided by the type of DLO to be manipulated. Concerning the manipulation of *soft* DLOs like ropes with pick and place actions, in [63] an image-based predictive DLO model is learned in a self-supervised manner. Instead, in [142], the image of the DLO is embedded in the latent space with linear dynamics imposed on it. Differently, [128] proposes learning the DLO dynamics in state-space while enforcing the physic priors via a biLSTM architecture modeling the DLO as a chain-like mass-spring system. In all these works, manipulation actions are sampled randomly and the best one is selected considering suitable cost functions. On the contrary, a gradient-based approach for estimating the best action is proposed in this work, where also a rotation component is estimated. In this way, a more complex manipulation action can be executed with respect to the simpler linear displacement operation. Regarding the manipulation of *elastic* DLOs, several works proposed a learning-based framework to predict the DLO dynamics. In [130, 134, 121] the DLO dynamic is approximated with data-driven approaches trained using simulation data. In [134] the authors propose an online adaptation of the DLO model to compensate for the *reality gap*. Similarly, in [121] a linear residual model is estimated online.

5.2.4 DLO Analytical Models and Physical Parameters

Many different physical models of DLOs have been proposed in the past [77], ranging from simpler mass-spring [76] and energy-based models to more accurate but computationally demanding elastic-rod, dynamic splines, and finite element models [77]. Other than the selection of a specific model, the choice of the integration method is crucial to achieve

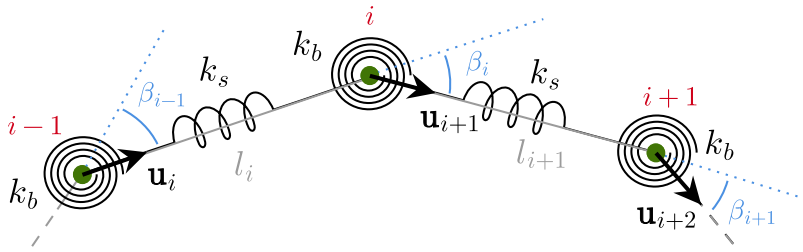


Figure 5.2: DLO analytical model representation.

a good trade-off between simulation accuracy and efficiency, and different integration approaches like Runge-Kutta and symplectic have been proposed [56]. As opposed to the mentioned force-based methods, a differentiable position-based simulation of DLOs is proposed in [67] where the integration steps are avoided leading to a more efficient and stable simulation.

Despite the many models available, their application in robotic systems is usually marginal due to the high computational cost and sensitivity to the choice of physical and simulation parameters. Indeed, their estimation is a cumbersome task, as can be seen in [77] where the physical model parameters are constantly adjusted to make the simulation results approach the experimental ones. Alternatively, in [67], the differentiability of the framework is exploited for the estimation of model parameters. However, the process is quite slow and can not be performed in an online framework.

Learning-based methods usually employ a DLO simulator to collect training data, where the simulator is generally constructed based on one of the mentioned DLO models. However, only a few learning-based works pay attention to DLO parameters. In [142] and [130] the DLO parameters employed in the simulator are estimated by optimizing the simulation against a small set of real samples, employing a sampling-based method [142] or a differential evolution strategy [130].

Compared to the parameters estimation procedures of [77, 67, 130, 142], this chapter proposes an efficient gradient-based procedure which can seamlessly be performed online during the execution of the shape control task.

5.3 Analytical Model and DLO State Representation

5.3.1 Analytical Model

A DLO can be physically modeled via a set of nodes having proper mass and axial springs connecting the nodes to create a serial chain [76], as shown in Fig. 5.2. In addition, the bending stiffness of the DLO is modeled by placing a torsional spring at each node. To improve the stability of the model, a viscous friction proportional to the velocity of the node is included as a damping term.

The dynamics of the generic node i can be written as:

$$m_i \ddot{\mathbf{p}}_i = -k_d \dot{\mathbf{p}} + \mathbf{f}_i^s + \mathbf{f}_i^b, \tag{5.1}$$

where $\mathbf{p}$ is the node coordinates, k_d a damping constant, f_i^s the force due to the axial effects and f_i^b the forces due to the bending effects. The axial effects f_i^s are computed as:

$$\mathbf{f}_i^s = -k_s(l_i - l_i^0)\mathbf{u}_i + k_s(l_{i+1} - l_{i+1}^0)\mathbf{u}_{i+1}, \tag{5.2}$$

where l_i and l_i^0 are the current and initial lengths of link i respectively, while u_i represent the unit vector of node i. With reference to Fig. 5.2, the bending effect can be written as:

$$\begin{aligned}\mathbf{f}_i^b =& k_b \frac{\beta_{i-1}}{l_i \sin\beta_{i-1}} \mathbf{u}_i \times (\mathbf{u}_{i-1} \times \mathbf{u}_i) - k_b \frac{\beta_i}{l_i \sin\beta_i} \mathbf{u}_i \times (\mathbf{u}_i \times \mathbf{u}_{i+1}) \\ & - k_b \frac{\beta_i}{l_{i+1} \sin\beta_i} \mathbf{u}_{i+1} \times (\mathbf{u}_i \times \mathbf{u}_{i+1}) + k_b \frac{\beta_{i+1}}{l_{i+1} \sin\beta_{i+1}} \mathbf{u}_{i+1} \times (\mathbf{u}_{i+1} \times \mathbf{u}_{i+2}),\end{aligned} \tag{5.3}$$

where

$$\beta_i = \arctan \frac{||\mathbf{u}_{i+1} \times \mathbf{u}_i||}{\langle \mathbf{u}_{i+1}, \mathbf{u}_i \rangle}$$

represents the angle between u_i and u_{i+1}.

The manipulation action executed on the DLO model is parametrized as a pick-and-place operation executed on the edge of the DLO, i.e. between two consecutive nodes. The decision to perform actions at the edge level is primarily influenced by the physical design of the gripper. In fact, the gripper does not interact with the DLO at a single node, but it engages with the DLO in a manner that can be more accurately described as the movement of the segment between two consecutive nodes. The DLO action parameters vector is defined by

$$a = [\alpha, \delta_x, \delta_y, \delta_\theta], \tag{5.4}$$

where α denotes the index of the edge to grasp, δ_x and δ_y are the linear displacements applied to the selected edge $\{\mathbf{p}_\alpha, \mathbf{p}_{\alpha+1}\}$ and δ_θ is the rotation applied to the initial edge orientation. The effect of the action introduced above is simulated using forward Euler method applied to the discretized version of eq. (5.1).

5.3.2 DLO Perception and State Representation

Since the DLO's dynamics is based on a mass-spring-damper system, an appropriate representation according to the chosen model is utilized. The DLO state V is represented as a sequence of n 2D points in the Cartesian space, i.e. $V \in \mathbb{R}^{n\times 2}$. In the simulation, each node represents the position of the simulated masses. The 2D coordinates of the DLO in the real scenario are obtained using the algorithm *RT-DLO* presented in Sec. 3.5. The input image is provided by a fixed 3D vision sensor. From the acquired point cloud, the workspace plane is segmented out to obtain the DLOs points in the scene. These points are then projected on the image plane by utilizing the camera's intrinsic parameters obtaining a binary mask of the DLO in the scene. The binary mask is forwarded to *RT-DLO* that performs the modeling of the DLO as a line graph representation of the DLO, i.e. a sequence of nodes and edges, as shown in Fig. 5.1. Therefore, the node coordinates in the line graph represent the state V.

5.4 Neural Network-based DLO Model

A NN is employed to approximate the DLO model and gain a significant boost in terms of computational efficiency. Indeed, the complexity of the analytical DLO model, i.e. eqs. (5.1)-(5.3), affects its performance, which makes using it in an online framework challenging. Instead, a NN can be trained to accurately replicate the DLO dynamics by exploiting a dataset of DLO movements, which can be generated offline using the

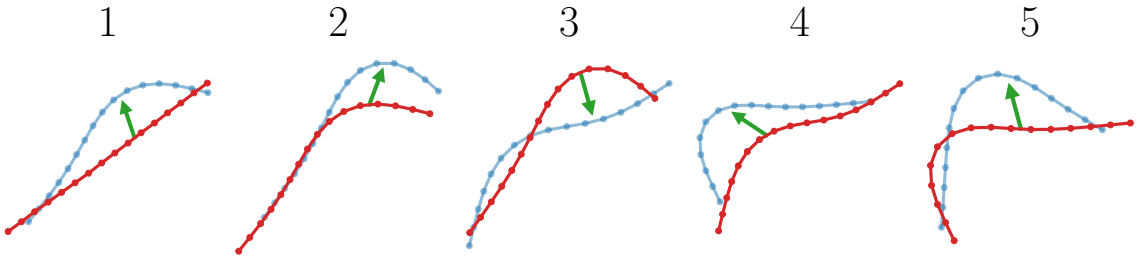


Figure 5.3: Example sequence of $k = 5$ dataset samples generated employing a simulated DLO. V_{in} in red, V_{out} in light blue and action arrow in green.

analytical DLO model. Thanks to the use of a relatively small neural network, a constant and short inference time is obtained which is more than an order of magnitude smaller than the time needed to evaluate the analytical DLO model. While the analytical model can be hard to differentiate, the NN model is easily differentiable with respect to the parameters, improving the possibility of optimizing all relevant tasks.

5.4.1 Dataset Generation

The dataset is generated by simulating the analytical DLO model subjected to a set of random actions. Each data sample consists of the DLO initial and final configurations, the performed action, and the employed model parameters.

The DLO initial and final configurations are sets of 2D points characterizing the DLO state, i.e. V_{in} and V_{out}, while the action is described by parameters introduced in eq. (5.4).

The DLO axial deformation is assumed to be negligible for the purposes of this chapter, thus the coefficient k_s is kept fixed to a high value. Instead, the damping term k_d, the bending term k_b, the length of the DLO, and the mass of the DLO change within predefined ranges. In particular, both the length and the mass are assumed to be known quantities since they can usually be measured. Eventually, the DLO length can be estimated online using the perception algorithm and the mass can be measured using force sensors mounted on the robot. The other two terms are instead difficult to measure, so they are estimated online.

Aiming to learn a general DLO model, both the action and model parameters are drawn from a broad range of values covering the expected real-world variability. In particular, each value is sampled from a uniform distribution with specific boundaries, except for the edge index which is sampled from a discrete uniform distribution.

To generate the dataset, the physical parameters are set to random values from the physically plausible ranges, and the simulated DLO is initialized with an almost linear initial configuration. Then, a set of k actions is sampled and the behavior of DLO is simulated after applying them sequentially. This procedure is exploited in order to build a diversified dataset in terms of DLO configuration complexity. In Fig. 5.3 an example sequence is shown. After the execution of each action, a dataset sample is saved containing the reached DLO configuration, the initial DLO configuration, the model parameters, and the performed action.

5.4.2 Data Augmentation

To improve the training efficiency and generalization capabilities of the NN model, several augmentation and normalization strategies are implemented on the data. The

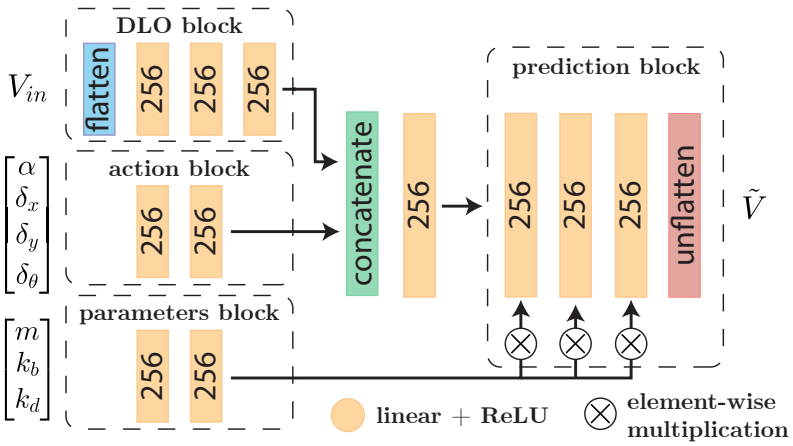


Figure 5.4: Neural network architecture. With $\times$ the element-wise multiplication is denoted.

idea is to exploit the symmetries in the DLO data to reduce the amount of information the NN has to learn.

The normalization is performed by finding a transformation that makes V_{in} aligned to the x-axis and mean-centered, and applying it to normalize both V_{in} and V_{out}. This transformation is composed of the translation equal to the negative geometrical center of the V_{in} and rotation that is required to make the first principal component of the points V_{in} aligned to the x-axis. The nodes are ordered from negative to positive x values by flipping V_{in} and V_{out} along the rows if necessary. The action index α is adjusted accordingly. In addition, the action parameters are scaled to be within the $[0, 1]$ range for α and $[-1, 1]$ range for the displacements. The model parameters are also normalized within the $[0, 1]$ range.

5.4.3 Neural Network Architecture

The neural network architecture is based on a set of Linear layers followed by ReLU activation functions. In detail, the network is composed of four main blocks illustrated in Fig. 5.4: the action block, the physical parameters block, the DLO block, and the prediction block. The input of the network is the initial configuration of the DLO V_{in}, the action parameters a, and the model parameters p. The length is not provided as input since it is implicit from V_{in}, thus p denotes only the model parameters provided as input, i.e. $p = [m, k_b, k_d]$. The output of the network, denoted as $\tilde{V}$, is the sequence of predicted changes of the 2D DLO coordinates from the initial configuration. The final predicted DLO configuration V_{pred} is expressed as

$$V_{\text{pred}} = \mathcal{F}(V_{\text{in}}, a, p) = \tilde{V}(V_{\text{in}}, a, p) + V_{\text{in}}. \tag{5.5}$$

The network is trained to minimize the mean squared error between the predicted V_{pred} and the expected V_{out} final configurations.

5.5 Gradient-based Estimation of Action and Parameters

The trained NN model is used to estimate both the next manipulation action and the parameters that allow for accurate approximation of the observed DLO behavior. These two estimation procedures are performed using numerical optimization of the loss

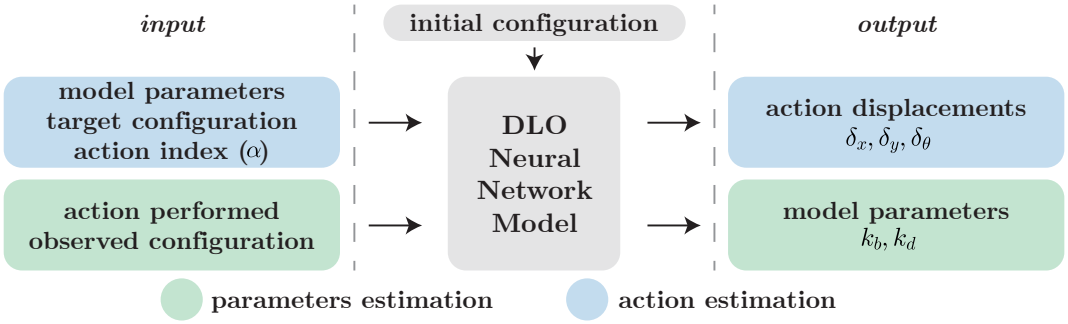


Figure 5.5: Scheme of the proposed gradient-based action and DLO parameters estimation.

function between two DLO states, e.g. V_1 and V_2. This loss is computed as the sum of L2 norms between corresponding points among the two states and can be defined by

$$\mathcal{D}(V_1, V_2) = \sum_{i=1}^{n} \|V_{1,i} - V_{2,i}\|. \tag{5.6}$$

An illustration of this idea is provided in Fig. 5.5. Since the NN model is intrinsically differentiable, a gradient-based approach can be used for the optimization of the above-mentioned loss function. In addition, thanks to the possibility of performing the optimization in batch, the mentioned gradient-based optimization can be performed quite efficiently.

5.5.1 Action Estimation

For the best action estimation given the current DLO state V_{in} and the model parameters p, the action parameters a minimizing the difference between the NN prediction $\mathcal{F}(V_{\text{in}}, a, p)$ and the target shape V_{tgt}, i.e. the goal to be reached in the shape control task, are sought. While this can be easily done using gradients for the edge displacement parameters, this is not the case for the edge index. Thus, the efficient batch processing capabilities of the NN model are employed, and $n-1$ optimizations are executed simultaneously, with the assumption that the edge index is held constant in each of them. Finally, the best action among the ones evaluated for each edge is selected. This optimization procedure can be in general described by

$$a^* = \underset{a}{\operatorname{argmin}}\ \mathcal{D}(\mathcal{F}(V_{\text{in}}, a, p), V_{\text{tgt}}), \tag{5.7}$$

where a^* denotes the action that maximally reduces the difference between V_{in} and V_{tgt} according to the used NN model $\mathcal{F}$. However, as mentioned, in practice first are performed $n-1$ optimizations of the form

$$a_j^* = [j, \underset{\delta_x, \delta_y, \delta_\theta}{\operatorname{argmin}}\ \mathcal{D}(\mathcal{F}(V_{\text{in}}, [j, \delta_x, \delta_y, \delta_\theta], p), V_{\text{tgt}})], \tag{5.8}$$

where a_j^* denotes the best action obtained for the fixed edge index j, and then the best action a^* is sought from the a_j^* by

$$a^* = \underset{a_j^* \text{ for } j \in \{1,2,\ldots,n-1\}}{\operatorname{argmin}}\ \mathcal{D}(\mathcal{F}(V_{\text{in}}, a_j^*, p), V_{\text{tgt}}). \tag{5.9}$$

Algorithm 5: Online Adaptation

Input: V_{tgt}
Output: V_{out}, k_d, k_b

1 $k_d, k_b \leftarrow k_{d,\text{start}}, k_{b,\text{start}}$
2 $\mathcal{V}_{in}, \mathcal{V}_{out}, \mathcal{A}^* \leftarrow \emptyset$
3 $V_{\text{in}} \leftarrow$ get_dlo_state()
4 $\epsilon \leftarrow \mathcal{D}(V_{\text{in}}, V_{\text{tgt}})/n$
5 **while** $\epsilon > \epsilon_{th}$ **do**
6 $\quad a^* \leftarrow$ best_action$(V_{\text{in}}, V_{\text{tgt}}, k_d, k_b)$
7 $\quad$ robot_manipulation(a^*)
8 $\quad V_{\text{out}} \leftarrow$ get_dlo_state()
9 $\quad \mathcal{V}_{\text{in}}, \mathcal{V}_{\text{out}}, \mathcal{A}^* \leftarrow$ update_dataset$(V_{\text{in}}, V_{\text{out}}, a^*)$
10 $\quad k_d, k_b \leftarrow$ best_parameters$(\mathcal{V}_{\text{in}}, \mathcal{V}_{\text{out}}, \mathcal{A}^*)$
11 $\quad \epsilon \leftarrow \mathcal{D}(V_{\text{out}}, V_{\text{tgt}})/n$
12 $\quad V_{\text{in}} \leftarrow V_{\text{out}}$

5.5.2 Parameters Estimation

Similarly to actions, the model parameters are estimated by searching for the ones that minimize the difference between the NN prediction V_{pred} and the observed DLO state V_{out}. This optimization can be written by

$$k_b^*, k_d^* = \underset{k_b, k_d}{\text{argmin}}\ \mathcal{D}(\mathcal{F}(V_{\text{in}}, a, p), V_{\text{out}}) \tag{5.10}$$

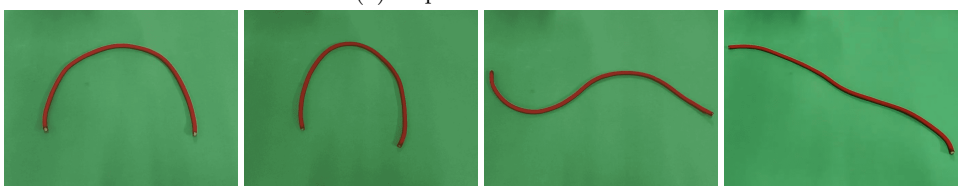
where k_b^*, k_d^* denotes the optimal values of the bending and damping coefficients, and $p = [m, k_b, k_d]$. Since the mass m can be measured, it is not optimized but measured and provided as input to the NN model.

5.6 Shape Control Task with Online Parameters Adaptation

To improve the manipulation capabilities of the robotic system in the case of the shape control task for a real previously unseen DLOs, an approach that utilizes both model-based DLO shape control and the online model parameters adaptation is proposed. This can be achieved by jointly using the gradient-based action and parameters optimization routines developed in Sec. 5.5. In Alg. 5, the proposed method is detailed. If there is no prior knowledge, the model parameters k_d and k_b can be initialized at the midpoint of the range used in the dataset generation, see Sec. 5.4.1. Therefore, given a target shape V_{tgt}, the robotic system iterates (line 5) executing a sequence of manipulation actions until the error between the current observed state V_{out} and the target shape V_{tgt} computed according to eq. (5.6) is below a user-defined threshold ϵ_{th}.

At each iteration, the camera system is first used to capture a new sample from the scene and process it via the perception system described in Sec. 5.3.2, thus obtaining an initial configuration (line 3). Then, the best action to move the DLO toward the target configuration is computed (line 6) according to eq. (5.7), and the result of the performed action on the real system is observed (line 8). The interaction with the real system is saved (line 9) into a task dataset, which is initialized empty at the beginning of the task. The task dataset is used for the best parameters estimation performed following eq. (5.10) (line 10). Finally, the configuration error is updated (line 11) by comparing the achieved shape to the target one.

(a) Ropes and surfaces



(b) Reached target shapes

Figure 5.6: Experimental robotic setup comprising different ropes and surfaces (a). Target shapes achieved with the *red* rope on the *cardboard* surface (b).

5.7 Experiments

The manipulation framework is evaluated in the context of a shape control task, with a robotic setup composed a Panda Robot equipped with a parallel-jaw gripper, a Photoneo Motioncam3D statically mounted on the robotic cell, and a selection of ropes and surfaces.

Three ropes are used in the experiments: a *white* rope (0.45 m, 0.02 kg, 0.01 m diameter); a *black* rope (0.42 m, 0.05 kg, 0.014 m diameter); and a *red* rope (0.50 m, 0.02 kg, 0.005 m diameter). Note, the *black* rope is the stiffest one, while the *red* rope exhibits a higher degree of bending elasticity compared to the *white* rope. Additionally, two planar surfaces with different physical properties are used: a *cloth* and a *cardboard* surface. The *cardboard* is smoother and more slippery than the *cloth*.

The experiments were executed employing as hardware an Ubuntu PC equipped with an Intel CPU i7-12700H clocked at 2.3GHz and an Nvidia GPU 3050Ti. A video of the experiments is available as supplementary material[1].

5.7.1 Optimization Details

The NN model (Sec. 5.4) is trained on a dataset of DLO manipulation samples, obtained by the analytical model (Sec. 5.3.1), comprising about 275K elements. The dataset is generated by employing the following boundary values for action and model parameter ranges. Regarding the action, $\alpha \in [0, 15]$; δ_x and δ_y are confined at ± 0.10 m; and δ_θ is within $\pm\pi/4$ rad. Concerning the model parameters, the damping coefficient $k_d \in [3, 30]$ Ns/m; the bending coefficient $k_b \in [0.05, 1.0]$ N/m; the DLO length within $[0.15, 0.50]$ m; the mass within $[0.02, 0.1]$ kg.

[1] https://www.youtube.com/watch?v=GWv3CFQqSzo

NN Model Training

A 90-10% split is employed to organize the dataset into training and validation sets. The network is trained for 100 epochs (batch size 128, learning rate 5×10^{-5}). The final weights are selected as the ones having the minimum mean squared error validation loss.

Action and Parameters Estimation

The gradient-based action estimation of Sec. 5.5 is performed for 500 optimization steps employing a learning rate of 0.005. Regarding the estimation of the parameter, the optimization is performed for 3000 steps with a learning rate of 0.01. In both cases, an early stopping procedure is implemented in case of the earlier convergence. The tanh and sigmoid activation functions are employed to limit the normalized action and model parameters, respectively, to the ranges of $[-1, 1]$ and $[0, 1]$ (Sec. 5.4.2). This ensures to obtain denormalized values consistent with the boundaries employed in the dataset.

5.7.2 Shape Control Task with Online Parameters Estimation

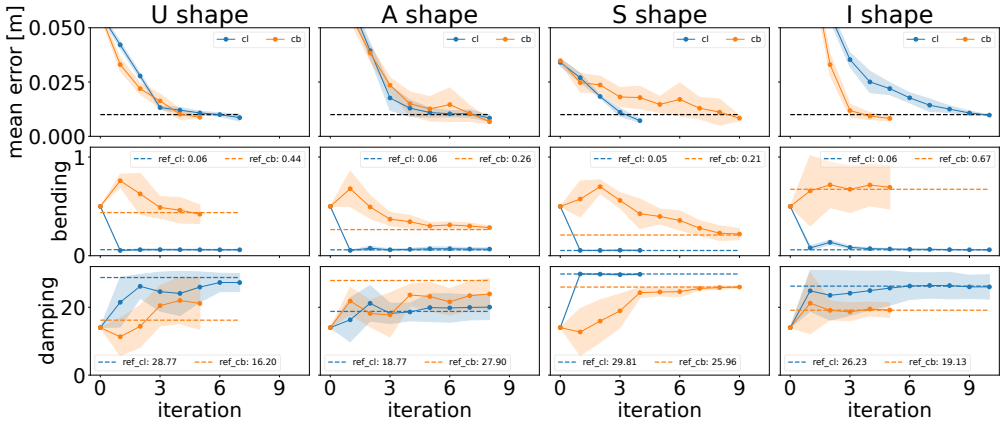
The shape control task involves the manipulation of four distinct target shapes: the U, A, S and I shapes (see Fig. 5.6b). Notably, the A shape differs from the U shape by requiring a more pronounced bending in its central region. Conversely, the S shape is characterized by two opposing and symmetric bends. The I shape is used to evaluate the situation of zero curvature target, where the I target is rotated by 90 deg with respect to the initial configuration. The reachability of all these shapes was ensured by rearranging the ropes between the target and initial configurations using a single human arm restricted to the motions available to the robot.

The task is performed following the online adaptation approach introduced in Sec. 5.6. The initial DLO configuration V_{in} is a straight line. The initial model parameters k_d, k_b are selected around the mid values of their ranges in the dataset, i.e. $k_d = 14$ and $k_b = 0.5$. The task is executed for each target shape V_{tgt} on each planar surface 5 times. The execution of the task is terminated once the error computed according to eq. (5.6) between V_{out} and V_{tgt} is below 0.01 m.

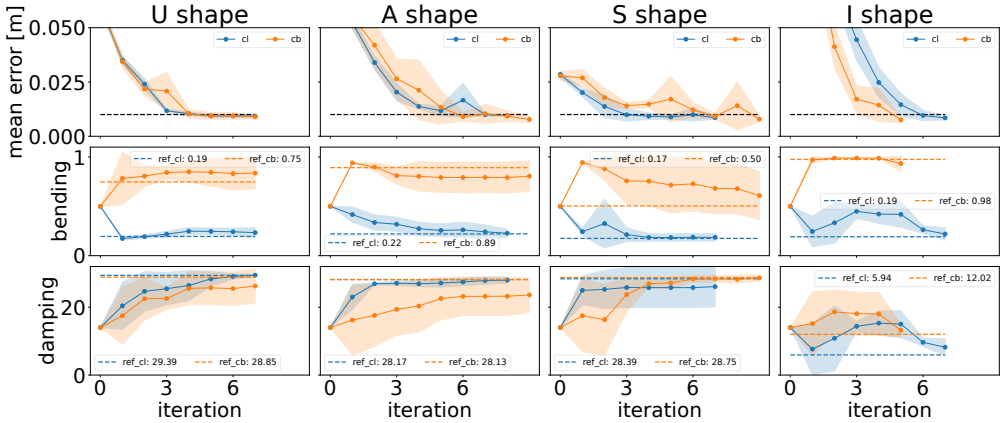
The results of the experiments are provided in Fig. 5.7, where, within each subplot of a specific rope, columns illustrate the task execution for specific target shapes, while rows provide an analysis of error and model parameters. In detail, the first row focuses on the mean error, with a dashed horizontal line denoting the 0.01 m threshold marking the completion of the task. The second and third rows delve into the examination of the bending parameter k_b and the damping parameter k_d respectively. Here, the dashed lines represent the estimated model parameters derived from all samples across all repetitions performed for a given shape. These values, in essence, serve as potential reference values for the specific parameters.

Analyzing the x-axis in the plots, iteration 0 represents the initial condition with a straight DLO configuration and model parameters at their initial values. An action is then executed by the robotic system, updating the observed DLO configuration. Model parameters are recalculated based on a single data sample, resulting in updated values at manipulation iteration 1. This iterative process continues until the specified termination condition is met. At manipulation iteration m, the parameter estimation is based on m data samples.

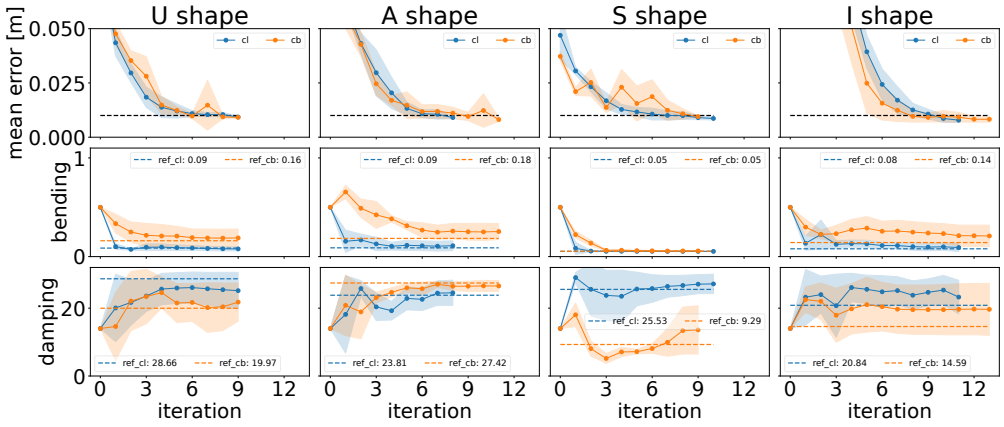
Examining the plots in Fig. 5.7, it is worth noting that similar bending parameters are consistently estimated for each specific rope on the *cloth* surface, regardless of the chosen target shape. The parameters estimated on the *cardboard* surface exhibit a higher degree of variability, indicating the presence of more complex dynamics due



(a) *white* rope



(b) *black* rope



(c) *red* rope

Figure 5.7: Outcomes of the shape control task involving online adaptation of model parameters, conducted across various rope types and surfaces. Average results across 5 repetitions per task (standard deviations confidence region intervals). With *cl* and *cb* the *cloth* and *cardboard* surfaces are denoted.

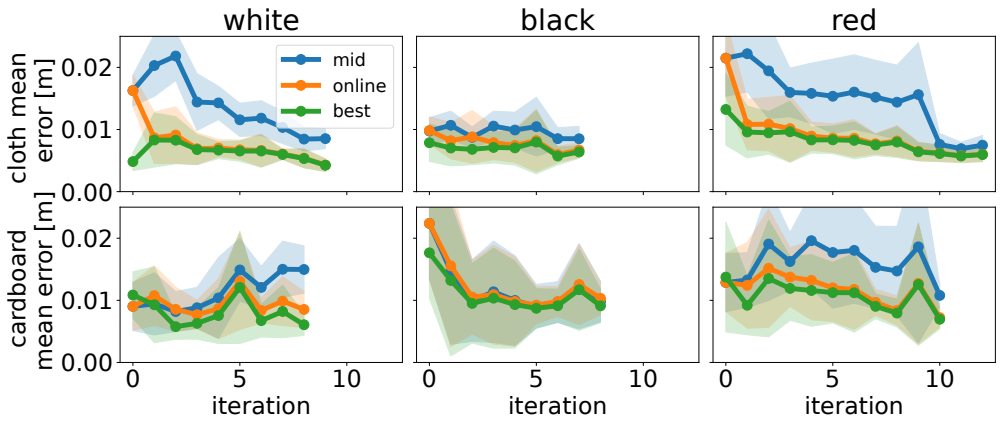


Figure 5.8: Comparing prediction errors using mid-range, online, and best model parameters across ropes and surfaces.

to increased slippage. The estimation of the damping term is less stable. In general, different pairs of k_d and k_b values are estimated for the same rope on different surfaces, highlighting the adaptation processes. The estimated bending parameters comparison confirms significantly different physical properties between the three ropes and that the *black* rope is the stiffest one, as initially predicted. For instance, on the *cloth* surface, the reference bending values are approximately 0.06 and 0.08 for the *white* and *red* ropes and about 0.19 for the *black* rope.

To gain a deeper insight into the impact of the online model parameters estimation, Fig. 5.8 presents a comparison among *mid-range*, *online* estimated, and *best* parameters. The latter refers to those estimated at the end of each task repetition, while the *mid-range* to the ones from which *online* estimation starts. These parameter setups were compared using the mean prediction error, denoted as $\mathcal{D}(V_{\text{pred}}, V_{\text{out}})$, computed after each iteration of the shape control task across all the target shapes. The plots illustrate how, within just a few iterations, the proposed method attains parameters that yield a mean error between V_{pred} and V_{out} comparable to the *best* scenario, and in most of the cases significantly better than for the *mid-range* parameters.

5.7.3 Comparison with state-of-the-art

The NN model of Sec. 5.4.3, and here shortly denoted as *NN*, is subjected to a comparative analysis against several previously proposed architectures concerning DLO dynamics prediction. These include the bi-directional LSTM (*BiLSTM*) [128], the interaction-network bi-directional LSTM (*INBiLSTM*) [130], the graph neural network architecture (*GNN*) [121], and the *RBF* network [134]. To ensure a fair comparison, a comparable number of parameters is employed across all the networks. The goals of this section are: (i) to compare different neural network-based DLO models, (ii) to show that the proposed approach to DLO modeling with conditioning on parameters and online adaptation is architecture-agnostic, (iii) to compare the performance of the proposed adaptation of the input model parameters against the direct adjustment of the neural network weights, as in [134].

The analysis is carried out on the data obtained in the shape control task of Sec. 5.7.2 on the *cloth* surface, by computing the mean prediction error. A 4-fold cross-validation approach is used. Fold 1 employs the data of the U shape for parameters estimation and the data of the A, S, and I shapes for forward prediction error. The same holds for sets 2, 3 and 4 for shapes A, S and I respectively.

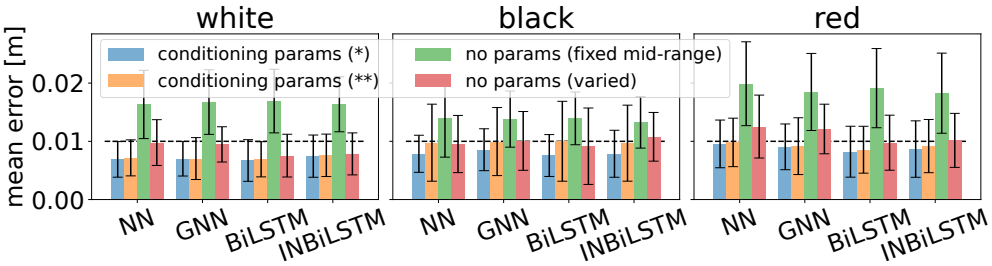


Figure 5.9: Prediction error for fix (*no params*) vs conditioning parameters across different models. For the latter, the symbol (*) denotes that the same shape is used for parameters estimation and forward prediction error, whereas with *(**)* the 4-fold cross-validation approach is denoted. With *fixed mid-range* and *varied* the two employed datasets are indicated.

Fixed vs Conditioning Parameters

To validate the choice of a NN model conditioned on the model parameters, a new dataset of 275K samples is generated with fixed mid-range parameters ($k_d = 14$, $k_b = 0.5$, mass of 0.02kg, and length in $[0.4, 0.5]$m). The comparison is performed by optimizing the models using three different approaches: 1) the mid-range dataset without considering parameters (*no params (fixed mid-range)*); 2) the varied parameters dataset of Sec. 5.7.1 without considering parameters (*no params (varied)*); 3) using modified architectures that incorporate model parameters as input (*conditioning params)*. Notably, the latter case implies estimating the parameters according to Sec. 5.5 before evaluating the prediction error. This is performed either employing the same shape for both estimation and prediction (*conditioning params (*)*) or with the introduced 4-fold cross-validation procedure (*conditioning params (**)*). The results are shown in Fig. 5.9. The plots show that conditioning the models on the parameters allows to achieve better accuracy than with the model that is trained to be robust to the range of the parameters (*no params (varied)*), or only with fixed parameters (*no params (fixed mid-range)*), for all considered architectures. Moreover, this is true also when optimized and tested for different shapes (compare *conditioning params (*)* and *(**)* cases, see also Sec. 5.7.3). Additionally, since all architectures show similar accuracy, a simpler and faster fully connected NN model is preferable (see Sec. 5.7.3).

Parameters vs Weights Update

Prior work proposed to directly update the weights of the model to achieve online adaptation [134]. In this section, a comparison against this approach is established in Fig. 5.10, where *fold* refers to the 4-fold cross-validation approach. Fig. 5.10 shows that updating the network weights directly, as proposed in the *RBF* approach [134], leads to overfitting to the specific target shape, resulting in a loss of generality (compare (*) and (**) cases). In contrast, when the model parameters update is considered, as employed in the proposed approach (NN), a higher level of generalization is observed, resulting in consistent outcomes across various tasks, regardless of the specific task performed during parameter estimation.

Architecture Efficiency

To provide an insight about the considered DLO model architecture efficiency, the time to perform forward and backward passes is measured: *NN* 0.20/0.65 ms; *BiLSTM* 0.48/0.98 ms; *INBiLSTM* 0.75/1.42 ms; *GNN* 0.63/1.05 ms. Taking into account the

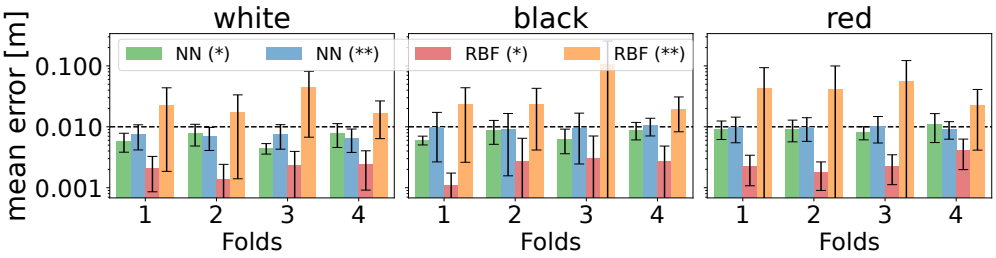


Figure 5.10: The mean prediction error (log scale) of the input parameters (ours) vs neural network weights (RBF [134]) update, evaluated on the same shape ($*$) or on different shapes ($**$).

time complexity of each architecture, it can be concluded that the proposed simple *NN* model emerges as the most favorable.

5.7.4 Limitations

The proposed framework exhibits several limitations. First, the action is predicted over a 1-step horizon, in contrast to other approaches [128, 121]. Nevertheless, the current gradient-based action estimation can be expanded to a N-step horizon using the same batch approach. However, encompassing all possibilities would result in an exponential growth of the task. Therefore, the introduction of sampling-based, Top-K, or other techniques is necessary to constrain the prediction task scale. Moreover, our approach is also a forward model of the DLO, so it can be used in any MPC-like framework to enable manipulation planning in longer horizons. The second limitation is related to the fixed number of nodes, e.g. 16 in this work, which would necessitate the generation of new data and the retraining of a new network with adapted dimensions if modified. A third limitation is to assume the DLO dynamics negligible during manipulation. Indeed, the analytical model provides a full trajectory of the DLO response to the pick-and-place action. However, the current NN model only captures the final response.

5.8 Conclusions

This chapter introduces a novel framework for DLO manipulation. The framework is based on a NN model that approximates the dynamics of DLOs, coupled with a gradient-based procedure for action and parameter estimation. The NN model is trained using a dataset comprising samples of DLO manipulation, obtained from an analytical model of DLO dynamics. The framework's performance is assessed in the context of a shape control task with real-time parameter adaptation within the model.

Experimental results demonstrate the framework's capability to effectively manipulate DLOs with various shapes, materials, and surface properties. When compared to several state-of-the-art NN architectures, the framework exhibits comparable performance while maintaining superior efficiency.

Currently, the framework is tailored for 2D DLO manipulation, with the potential for expanding its capabilities to encompass 3D DLO manipulation. Additionally, the analytical model of DLO dynamics, which relies on simplifying assumptions, may benefit from a more accurate representation.

Furthermore, an investigation is planned to extend the framework to scenarios involving dual-arm manipulation and manipulation tasks conducted in the presence of environmental obstacles. Finally, there is an interesting avenue for extending the framework to handle multiple interconnected DLOs, such as DMLOs, where the manipulation process must consider constraints arising from the presence of *branch-points*.

Chapter 6

DMLOs Topology Representation Learning

The representation of complex deformable multi-linear objects, such as wire harnesses, poses significant challenges in various applications, including robotic systems' perception and manipulation planning. By extending the solutions developed in Chap. 3 for deformable linear objects, this chapter proposes an approach to address the robust and efficient topological representation of deformable multi-linear objects leveraging a graph-based description of the scene exploiting graph neural networks.

6.1 Introduction

Deformable Multi-Linear Objects (DMLOs), also known as Branched DLOs [125, 152], are flexible objects that possess a linear structure similar to DLOs but also feature branching or bifurcation points where the object's path diverges into multiple branches. This complexity in their deformation sets them apart from simple DLOs. Examples of such objects include wire harnesses, as illustrated in Fig. 6.1, and branched hoses commonly encountered in manufacturing settings. Detecting and manipulating these objects require advanced techniques that can account for both their linear deformations and branching behavior. While several algorithms and approaches have been developed in recent years for general DLOs, addressing the perception and manipulation of DMLOs remains an ongoing challenge [119]. Some research exists in the literature of DMLOs domain, such as [57, 38, 131, 86, 46, 135, 144, 152, 125]. Nevertheless, these efforts often focus on specific scenarios and types of objects, thus presenting limitations in their general applicability.

In this chapter, the challenging task of describing the configuration of a DMLO via a graph-based representation is tackled. The proposed approach is currently under review as a journal publication [16]. A topological representation is particularly suitable for DMLOs since it allows to capture high-level information about the objects, which can be exploited easily in robotic manipulation tasks. The graph representation of DMLOs is derived through a multi-step process. Initially, graph nodes are extracted by sampling along the estimated centerlines of the objects on a binary mask of the scene. Subsequently, a data-driven pipeline is employed to acquire knowledge about how to assign graph edges between these nodes and determine the type of each node based on their local topology and orientation. Finally, by utilizing the learned information, a solver is applied to combine the predictions and generate a coherent representation of

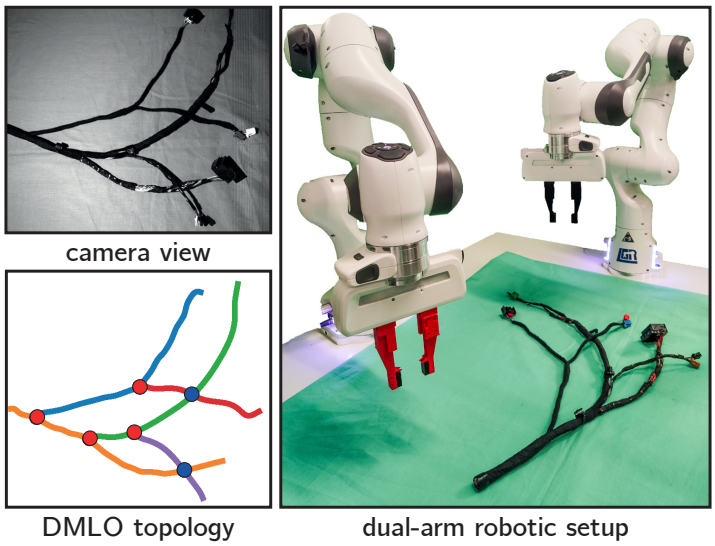


Figure 6.1: Extracted topology representation of a DMLO from a camera sample. The branch *sections* composing the DMLO are displayed in different colors while the *branch-points* and *intersection-points* are highlighted with red and blue dots. On the right side, the dual-arm robotic experimental setup.

the objects in the scene.

Obtaining a robust and efficient graph representation of the DMLOs entails addressing two pivotal challenges: link prediction, which involves predicting the existence of connections between nodes in the graph, and node characterization, which aims to understand and capture key nodes' information such as their local orientation and topology. These two challenges are tackled by exploiting Graph Neural Networks (GNNs) [127] to learn the topology of the DMLOs. GNNs have revolutionized the field of machine learning, enabling effective learning on graph-structured data [59]. They have demonstrated their prowess across a wide spectrum of tasks, encompassing node classification, link prediction, and graph generation. GNNs excel at capturing the structural information of graphs and leveraging node and edge features to learn powerful representations that encode the relationships and dependencies within the data. This makes them particularly suitable for the task of learning the topology of DMLOs.

The learning phases exploit an *inductive framework* [40]: The GNN weights are optimized based on a labeled training dataset while the model is tested and deployed on unseen data (i.e. graphs) samples. In particular, graphs synthesized in a simulated environment are employed in the training set, allowing the optimization of the model on a large number of graphs, while enabling important generalization capabilities in unseen real-world graphs.

The proposed method, thanks to the graph-based representation, preserves satisfactory levels of robustness in the presence of noise and disruptions in the input image mask, i.e. it is not affected by the possible presence of large connectors, clips, and other fixtures along the object branch *sections*. Additionally, there is no prior knowledge of the structure of the DMLO and the method is capable of simultaneously identifying *branch-points* and resolving *intersection-points* among different branch *sections*.

The evaluation is conducted by performing a disentangling manipulative operation with a dual-arm robotic setup, shown in Fig. 6.1. The DMLOs employed in the experiments are complex real-world wire harnesses from the automotive domain. The experiments demonstrate the effectiveness and accuracy of the framework in generating

a robust and efficient topological representation leading to a simplified manipulation task by the robotic system.

To summarize, the main contributions of this chapter are:

1. Description of the configuration of DMLOs with a topological graph obtained by exploiting a GNN model learned in an inductive framework to provide a robust and efficient representation;
2. Robustness to noise in the input mask and to the presence of connectors, clips, and other fixtures that can be connected to the branch *sections* of DMLOs;
3. Extensive evaluation in disentangling manipulative operations with complex real-world automotive wire harnesses, demonstrating its effectiveness in generating an accurate and efficient topological representation, which simplifies manipulation tasks.

6.2 Related Works

6.2.1 Graph Neural Networks

A graph is a mathematical structure consisting of a set of nodes (or vertices) and a set of edges (also called links) that connect pairs of nodes [11]. Graphs are a powerful tool to represent data that can be modeled as a set of objects/points (nodes) and their relationships (edges). Indeed, graphs found applications in many fields, ranging from social networks to chemistry and biology [127].

Graph Neural Networks (GNNs) [101] are a powerful extension of classical graph analysis methods where the graph's underlying structure is exploited in a learning framework. The idea of GNNs is to iteratively update node representations by aggregating information from their neighboring nodes of the graph. This allows GNNs to capture local relationships between the nodes, and through the propagation of the embedding via the network layers, it is also possible to capture more global relationships.

GNNs have been successfully applied to a wide range of tasks, including node classification, link prediction, and graph classification [127]. In particular, graph classification and link prediction are the two tasks also addressed in this chapter. The first focuses on predicting the class or label of a set of nodes composing a graph. The task is usually performed by first learning node embeddings via a GNN, combining all the nodes embedding via a graph pooling operation, and then applying a classifier [127]. Link prediction aims to estimate the existence of connections between nodes in the graph. This task is usually performed by learning a similarity function between nodes and then predicting the existence of a link between two nodes based on their similarity [140].

Concerning deformable objects, GNNs were successfully adopted for the manipulation of linear and planar objects. In particular, GNNs are usually exploited for learning the interaction between key-points on clothes or ropes [78, 30, 72]. In [78], the GNN is combined with a recurrent neural network for tracking the key-points and an MPC for planning the manipulation action. In [30] a second GNN is used also for planning the action. A GNN is employed as a forward dynamics module for a planar deformable object in [72].

6.2.2 Deformable Objects State Representation

Deformable object perception and representation is still an open research problem [132]. Considering the complexity of the perception of such objects, many researchers use

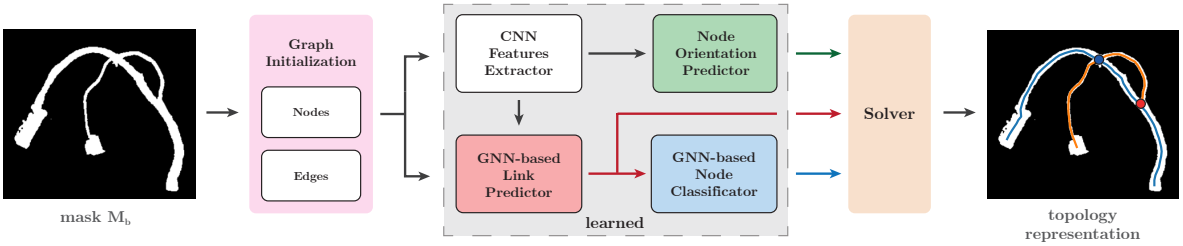


Figure 6.2: Schema of the proposed approach. On the right side, the topology representation provided as output is shown with the individual branch *sections* of the DMLO denoted in different colors and the single *branch-point* and *intersection-point* highlighted as red and blue dots.

simplified scenarios to segment the objects from the scene, e.g. by using color difference, controlled backgrounds, or markers, as highlighted in [19]. This clearly limits the possible utilization of these methods in real scenarios, where the inconsistency of the background or the sensitivity to the quality of the segmentation mask can restrain a method from being used. Learning-based methods have emerged as a key tool for tackling the perception problem of linear and planar deformable objects. Usually, key-points are automatically extracted from an image [30, 78].

Specifically for DLOs, several works have addressed the synthetic and real-world dataset generation [19], the instance segmentation task [15, 24], 3D shape estimation [14, 75], and crossing points detection [50]. It is clear however the lack of solutions for identifying more complex DMLOs. Indeed, methods developed for DLOs can not be directly applied to DMLOs, but modifications and extensions [152] are required.

6.2.3 Deformable Multi-Linear Objects

Despite some research have been conducted in the past, e.g. wire harness manufacturing automation in [86] and collaborative robots in wire harness assembly in [84], the state representation of DMLOs remains largely unexplored. Incorporating more structural information into the state representation of manipulated objects would greatly facilitate planning and manipulation tasks. For instance, in [46], an automatic path-planning algorithm for wire harness installations is presented. However, this approach requires complete knowledge of the entire topological structure and the initial state of the DMLO, which can be a significant limitation in real-world applications.

In an attempt to extract partial information, [135] developed a vision system that identifies the final connectors of a wire harness through image processing. However, they did not address other related tasks such as recognizing *branch-points* and *intersection-points*, leaving the geometric reconstruction of the main trunk and branch *sections* for future work. Another approach, presented in [57], involves classifying branch *sections* of the wire harness from raw RGBD images and augmenting the data using elastic transformation. However, the neural network trained in this work does not solely focus on specific connectors within each branch but also considers the entire branch *section* as a whole. This approach provides a black box solution encoding all information related to the branch *sections* and the DMLO into classification data, limiting its portability across tasks and/or different DMLOs.

Tracking of DMLOs has been investigated in several works [125, 131]. In [125] a modified registration method is combined with a topology model of the DMLO for its tracking under occlusions. The approach is evaluated on a wire harness featuring connectors and clips, similar to the one employed in this work. However, knowledge of the DMLO structure is required and the DMLO is assumed to be in a fully visible

and entirely disentangled configuration. Instead, in [131], tracking using depth images via particle filtering is proposed. Here a data-driven predictor is used to assess the likelihood of each particle based on the depth image. However, the method is evaluated on simplified DMLOs with no intersections between the branch *sections*, nor connectors and other disturbing objects along the branch *sections* themselves. Similarly to the tracking methods described above, in [152] the matching of an observed topological configuration of a DMLO with a model representation is performed via a cost function. A score evaluating the topological uniqueness of a DMLO is also proposed.

The disentangling of DMLOs is investigated in [144], where a bin-picking policy for grasping and extracting a DMLO in an untangled configuration is learned from real-world data.

This chapter proposes an approach capable of extracting and representing DMLO topological information in a graph representation that can be easily incorporated into a robotic manipulation framework. Noteworthy, in the approach, no prior knowledge of the DMLO structure is required.

6.3 Method Overview

The proposed approach aims at providing as output a topological representation of the DMLO, as schematized in Fig. 6.2. The approach can be subdivided into three main stages, which are:

1. **Graph Initialization**: the graph representation is initialized with a set of nodes and edges from an image of the scene;
2. **Topology Learning**: the graph edges are assigned a probability of existence; the nodes are classified according to their local topology and orientation;
3. **Solver**: the learned information is processed by a solver to obtain a coherent representation of the DMLOs in the scene.

In particular, the graph initialization phase is detailed in Sec. 6.4. Then, the topology learning phase is described in Sec. 6.5. The solver is described in Sec. 6.6. Therefore, the obtained representation can be used effectively for disentangling the DMLO branch *sections*, as detailed in Sec. 6.7.

6.4 Graph Initialization

An undirected graph of n nodes and m edges is initialized and denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the single node i is denoted as v_i. Given two nodes v_i and v_j, their edge is indicated as e_{ij}.

6.4.1 Semantic Segmentation

The input data to the proposed approach is a binary mask M_b of the scene where the pixels are classified to either belong to DMLOs (value of 1) or not (value of 0). An example of a mask M_b is provided in Fig. 6.2.

Obtaining M_b is not the focus of this chapter as it is considered a pre-processing step. For getting M_b, in the following a 3D sensor capturing a point-cloud of the scene is employed. Thus, a plane segmentation approach on point-cloud data is executed and an image mask is computed. However, the proposed approach is independent of the method used to obtain the semantic segmentation mask. Alternative approaches like the ones highlighted in Chap. 3 can be employed depending on application requirements.

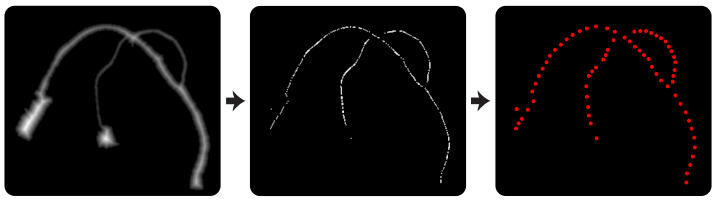


Figure 6.3: Vertices sampling procedure from segmentation mask: distance transform maks M_{dist}, local maximum mask M_{max}, graph nodes $\mathcal{V}$.

6.4.2 Vertices Sampling

The set $\mathcal{V} = \{v_i\}_{i=1}^{n}$ contains the n vertices of the graph efficiently sampled from the binary mask M_b. The sampling strategy follows the approach already presented in Sec. 3.5. First, the distance transform operator is executed on M_b obtaining M_{dist}. This operator computes the Euclidean distances between the non-zero values of M_b and the nearest boundaries (zero/black values) [7], thus assigning an intensity value to each pixel based on the computed distance. In Fig. 6.3, M_{dist} originated from M_b (left side of Fig. 6.2) is shown where M_{dist} is color-mapped on the grayscale level from dark (zero distance) to bright (maximum distance).

Next, M_{dist} is dilated using a small square kernel, i.e. 3×3, resulting in M_{dil}. The local maxima of M_{dist} are then extracted by comparing pixel-wise values of M_{dist} and M_{dil}, masked using M_b. This process generates a binary map M_{max}, which indicates the positions of local maxima, approximating the center lines of the DMLOs in the mask. The computation of M_{max} can be expressed as follows:

$$M_{\text{max}}(i,j) = \begin{cases} 1 & \text{if } M_{\text{dil}}(i,j) = M_{\text{dist}}(i,j) \text{ and } M_b(i,j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The set of maximum pixels of M_{max}, i.e. pixels whose value is equal to 1, is denoted as $\mathcal{V}_{\text{max}}$. The cardinality of $\mathcal{V}_{\text{max}}$ is quite high (see M_{max} in Fig. 6.3). Thus, the farthest point sampling algorithm [92] is employed for down-sampling $\mathcal{V}_{\text{max}}$. The ratio $\alpha \in [0, 1]$ is used to specify the amount of down-sampling. It is chosen such that the density of the nodes is kept mostly constant. This is important since different DMLOs can have very diverse amounts of branch *sections* and thus foreground pixels. The constant density is obtained by dividing the current mean point-to-point distance in $\mathcal{V}_{\text{max}}$ with the target distance to achieve. Therefore, the set of vertices $\mathcal{V}$ of the graph $\mathcal{G}$ is obtained as $\alpha\mathcal{V}_{\text{max}}$.

This vertices sampling process ensures an efficient representation of the DMLO's topology, reducing redundancy while preserving the key structural information necessary for subsequent analysis and modeling.

6.4.3 Edges Sampling

In order to create a set of initial edges, referred as $\mathcal{E}_{\text{knn}}$, the proximity between vertices is leveraged. This involves retrieving and using the K_{nn} nearest neighbors in $\mathcal{V}$ as edges for each vertex. The proposed method employs a value of 8 for K_{nn}. By utilizing a sufficiently large number of nearest neighbors as initial edges, comprehensive coverage of the local connectivity in the graph is guaranteed, which can be advantageous for the subsequent learning phases of Sec. 6.5.

6.5 Topology Learning

Learning on graphs using GNNs involves processing the graph structure, node features, and edge features.

In a graph with n nodes, the graph structure can be represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$. The adjacency matrix A encodes the connections between nodes, where $A_{ij} = 1$ if there is an edge between nodes i and j, and $A_{ij} = 0$ otherwise.

The node features in the graph are represented by a node feature matrix $F_{\text{v}} \in \mathbb{R}^{n \times h}$, where each row i corresponds to the h features associated with node n_i. The edge feature matrix, denoted as $F_{\text{e}} \in \mathbb{R}^{m \times d}$, represents instead the features associated with each edge in the graph. Here, d denotes the dimensionality of the edge features.

By incorporating the adjacency matrix A, the node feature matrix F_{v}, and the edge feature matrix F_{e}, GNNs can effectively process and learn from the structural, node, and edge information of the graph to perform various graph-based learning tasks.

In the context of this chapter, the graph learning strategy is based on an *inductive framework* with supervision provided by ground truth data [40]. A synthetic dataset of DMLO graphs is generated and exploited, as detailed in Sec. 6.5.1. The nodes and edges feature computations are discussed in Sec. 6.5.2. Hence, the link and node orientation prediction tasks are presented in Secs. 6.5.3 and 6.5.4. In Sec. 6.5.5 the node subgraph classification task is analyzed and Sec. 6.5.6 details the overall training strategy.

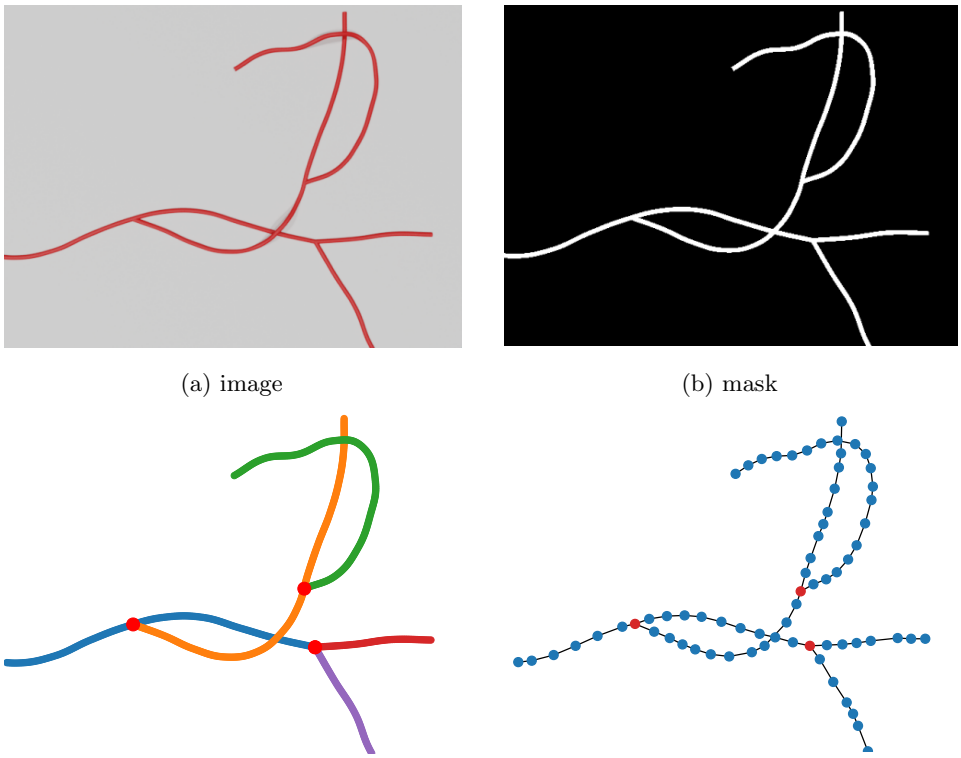
6.5.1 Synthetic Dataset

For training the GNN model, a synthetic dataset is exploited. Similarly to [19], each *section* of the synthetic DMLO is modeled as a 3rd-order spline curve and a rendering pipeline based on Blender [31] is used for obtaining the image samples.

First, a synthetic DMLO in a random configuration is generated. It is composed of a maximum of 5 main *sections*, organized by up to two *branch-points*. A *branch-point* is defined as an area of the DMLO where different *sections* are connected. Then, the camera location is randomized and an image of the scene is rendered. In Fig. 6.4a an example of the generated data is provided.

Along with the generated image, the randomly generated spline curves are saved and used for annotation purposes. Indeed, since the proposed learning strategy is supervised, for each generated mask sample (Fig.6.4b), annotation data is required. Therefore, a ground truth graph is generated based on the available annotations. The vertices of the ground truth graph are obtained by exploiting the sampling strategy of Sec. 6.4.2. The edges are instead computed by processing the ground truth spline curve used to synthesize the DMLO sample. These curves are projected from the cartesian space to the image plane by knowing the camera location in the synthetic scene, as shown in Fig. 6.4c. Then, each sampled node is associated with the correct curve. The ordering of the vertices and thus the sequence of edges to add is obtained by traversing the vertices on the corresponding ground truth curve. The only edges missing are the ones at the *branch-point* locations. In this case, since the structure of the DMLO is known, this information is exploited to add the proper edges. The result is a full graph representation of the scene provided by the mask, as shown in Fig. 6.4d. Given the association between the graph nodes and the initial spline curves, it is also possible to associate a ground truth value in terms of node direction as the tangent of the spline at the node location.

By generating synthetic data and annotating the ground truth graphs, a labeled dataset is created, which can be used to train the models of Secs. 6.5.3 and 6.5.5.



(a) image (b) mask

(c) annotation (d) ground truth

Figure 6.4: Sample of the synthetic dataset. In (a) and (b) the rendered image and mask of the randomly generated DMLO. In (c), the colored curves represent the different sections and the red dots describe the *branch-points* used in the annotation process. In (d) the obtained ground truth graph is displayed.

6.5.2 Encoding Nodes and Edges

The use of node features is crucial for enhancing the performance of GNNs [127]. Each node is given an input feature vector that originates from two sources: 1) the encoded normalized absolute position of the node in the image plane, 2) and a vector of values that are obtained by sampling the encoded mask M_{enc} at the node's location. Features are also computed for the edges. The details of the node and edge features are provided in the following.

Encoding of Node Positions

The node positions, normalized in the $(0, 1)$ range, are encoded through a linear layer with ReLU activation having an input dimension of 2 (the pixel locations of each node) and an output dimension of h. This results in a matrix F_{P} of size $\mathbb{R}^{n \times h}$. Each row of F_{P} corresponds to the encoded feature vector of a node, capturing its position information.

Encoding of Mask

For the mask encoding, the M_{dist} mask is first normalized and then processed through a two-layer CNN. Each convolutional layer uses a 5×5 kernel with a padding of 2 to

maintain the input image size. A batch normalization layer is included between the convolutional layers to stabilize the training process. Finally, a max pooling operation is performed with a 2×2 kernel and a stride of 2. The first convolutional layer expands the input channels from a dimension of 1 to $h/2$, while the second layer increases the channel's dimension further to h. This encodes the M_{dist} mask into a new mask M_{enc} with half the dimensions and h channels. The node features are then extracted from M_{enc} by sampling each node location (adjusted by the new dimensions) along all the h channels. This results in the matrix F_{M} of size $\mathbb{R}^{n \times h}$ where each row encodes the mask information at each node location.

The encodings obtained from F_{P} and F_{M} are concatenated column-wise, resulting in the combined node feature matrix $F_{\text{PM}} \in \mathbb{R}^{n \times 2h}$. Each row of F_{PM} represents the concatenated feature vector of a node, combining both the position information and the encoded mask information. A final non-linear transformation is performed on F_{PM} using a linear layer with ReLU activation for downsampling the feature dimension to h. The output of this transformation is the final node feature matrix $F_{\text{v}} \in \mathbb{R}^{n \times h}$.

Encoding of Edges

Similarly to nodes, edge features are of paramount importance for enhanced GNNs performance. Therefore, the features associated with the edges are obtained by encoding the edge direction and norm. In particular, for each edge in the set $\mathcal{E}_{\text{knn}}$, the norm is computed, normalized, and encoded via a linear layer from a dimension of 1 to $h/2$ followed by a ReLU activation function. Likewise, the edge direction is converted to an angular value and first encoded as a 180-dimensional vector with entries defined by applying a Gaussian function centered at the edge angle. This encoding is necessary to address the angular periodicity [129]. Then, the 180-dimensional vector is embedded by a linear layer from a dimension of 180 to $h/2$ followed by a ReLU activation function. Finally, the embedded norm and direction are concatenated and further processed by a second linear layer providing an output dimension of h. Therefore, an edge feature matrix $F_{\text{e}} \in \mathbb{R}^{m \times h}$ is obtained, where each row represents the feature vector of an edge in the graph.

6.5.3 Link Prediction

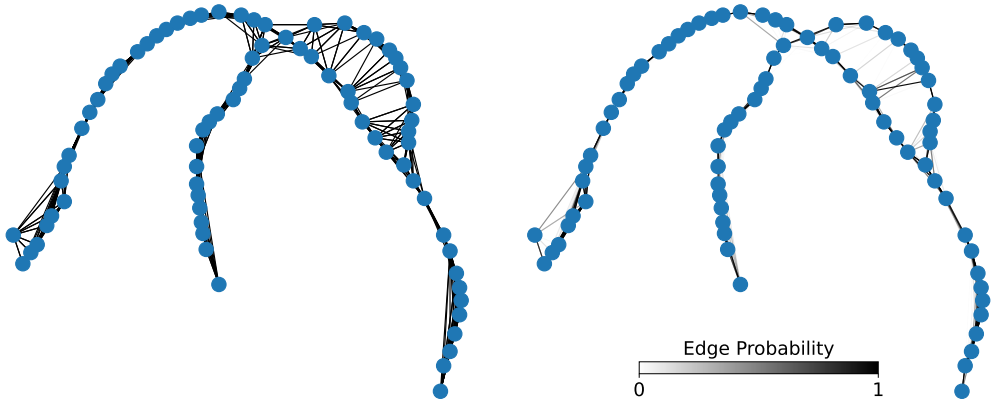
Link prediction in graph theory refers to predicting the likelihood of an edge (link) between two nodes. In this chapter, the task is to determine the probability of existence for each edge in the set $\mathcal{E}_{\text{knn}}$.

To accomplish this, a Graph AutoEncoder (GAE) [60] is utilized, which is characterized by an encoder-decoder structure. The encoder takes the initialized graph from Sec. 6.4 as input and uses GNN layers to aggregate and update the features of each node based on the extracted input features of Sec. 6.5.2. The encoder's purpose is to learn a latent representation that captures the underlying structure and patterns in the graph.

On the other hand, the decoder processes the node features obtained from the encoder through multiple neural network layers. When given two nodes for testing, the decoder produces a value that represents the probability of their connection. The decoder's aim is to rebuild the graph structure based on the learned node representations.

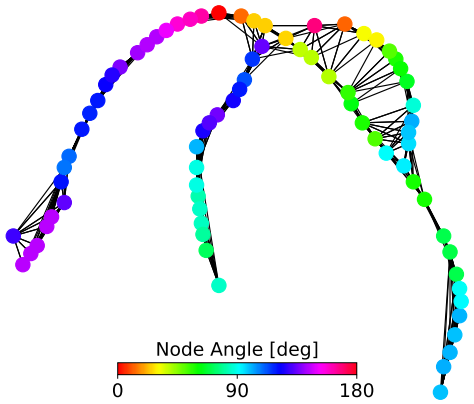
GNN Structure

To perform link prediction using GNNs, a 3-layer GNN is utilized to process the node and edge features along the graph structure. Each GNN layer aggregates and updates

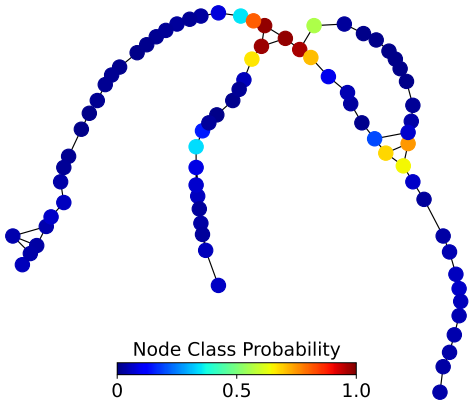


(a) initial edges $\mathcal{E}_{\text{knn}}$

(b) link probabilities $\mathcal{Z}_{\text{lp}}$



(c) node orientations $\mathcal{Z}_{\theta}$



(d) node class probabilities $\mathcal{Z}_{\text{cls}}$

Figure 6.5: Outcomes of the different stages of the pipeline. From the graph with the input set of edges $\mathcal{E}_{\text{knn}}$ (a), the predicted edge probabilities are displayed in (b) while the node orientations are in (c). The result of the node subgraphs classification task between *normal* (0) and *highly-connected* (1) nodes is shown in (d).

the node features followed by a non-linear transformation. The GNN layer operation, adapted from [133], can be expressed as follows:

$$x_i^{(k+1)} = \sigma\left(\sum_{u \in N(v_i)} (W x_u^{(k)} + B x_i^{(k)} + K f_e^{(u,i)}) + S x_i^{(k)}\right)$$

Here, $x_i^{(k+1)}$ represents the updated embedding vector of node v_i in the $(k+1)$-th GNN layer. The terms W, B, and K are trainable matrices that perform linear transformations of the neighbor embeddings, i.e. $u \in N(v_i)$, the current node embedding x_i^k, and the edge feature between the considered nodes $f_e^{(u,i)}$, respectively. The matrix S is a trainable matrix that accounts for a skip connection. The function σ represents a non-linear activation function, such as ReLU. Each GNN layer transforms the node features while maintaining their dimensions as h. Two batch normalization layers are

placed between the GNN layers to help stabilize the training process. For the first GNN layer, the embedding x_i^0 corresponds to the node features row-vector f_v^i extracted from matrix F_v.

Links Decoding

Following the GNN processing, the node embeddings obtained by the last layer are used to predict the probability of a connection for each edge in the set $\mathcal{E}_{\text{knn}}$. The link predictor decoder is conceived as a 3-layer MLP (Multi-Layer Perceptron) with an input and output dimension of h and ReLU activation functions between the layers, except for the last layer. Given two graph nodes v_i and v_j with their encoded node features x_i and x_j, the probability of a connection between them is obtained as:

$$z_{ij}^{lp} = \sigma(\text{MLP}(x_i \cdot x_j))$$

Here, σ represents a sigmoid activation function, ensuring that the predicted probability is within the range (0, 1). The MLP applies a series of linear transformations and non-linear activations to the dot product of the node features z_i and z_j before producing the final probability value. In Fig. 6.5b the link probabilities predicted for a sample input graph (Fig. 6.5a) are shown.

6.5.4 Node Orientation Prediction

When representing linear objects and shapes, it is possible to perform an orientation characterization for each vertex of the graph. This characterization aims to describe the local section of the linear object near the vertex by assigning an orientation attribute to the vertex itself. To achieve this, the features F_{M} extracted in Sec. 6.5.2 with dimensions h are mapped to a 180-dimensional vector through a single neural network linear layer followed by a softmax activation function. This process is similar to what is done in [15], with the fundamental difference being that the input features are extracted from the encoded mask M_{enc} instead of employing fixed-size crop images of M_{dist}. The resulting 180-dimensional vector represents the node orientation as a Gaussian function centered at the predicted orientation, similar to the encoding performed to the edges in F_{e} (see Sec. 6.5.2). To obtain the actual angular value, the argmax function is applied to determine the peak of the predicted distribution. An example of the estimated node orientations for a sample graph is shown in Fig. 6.5c.

6.5.5 Node Subgraph Classification

Until now all the nodes $v \in \mathcal{V}$ are regarded as similar with no distinction. However, in the context of DMLOs, it is possible to distinguish between two types of nodes: highly-connected nodes and normal nodes. A highly-connected node is defined as a node having a degree of more than 2, which instead is the limit for normal nodes, i.e. a node that is contained in a given *section* of the DMLO. An example of highly-connected nodes are *branch-points* and *intersection-points*. A *branch-point* is characterized by a degree of 3, while an *intersection-point* has a degree of more than 3.

The node classification task is employed to predict the type of each node in the graph among the two possible ones. Classifying directly the nodes of the graph is challenging, even exploiting the graph structure, since the condition of being a highly-connected node is due to an area around the node rather than the node itself. To better characterize the task, subgraphs of one hop around the considered node are computed and used in the classification task. The features of the nodes composing the subgraph are exploited

to enrich the set of data making the classification problem easier. A GNN is utilized to process the node features of the subgraph and predict the probability of the entire subgraph originating from a highly-connected (label 1) or normal (label 0) node.

As node features, the encoded position and mask information are exploited as described in Sec. 6.5.2. In addition, the node embedding obtained by the last GNN encoder layer of Sec. 6.5.3 is concatenated as well. Thus, a linear layer is used to process the node feature vector of dimension $3h/2$ to a dimension of h.

As edge features, the encoded norms and direction as detailed in Sec. 6.5.2 are exploited. The output of the decoder of Sec. 6.5.3 is processed by a linear layer with input dimension 1 and output dimension h followed by a ReLU activation function, obtaining F_{e}^{lp}. The final edge features are obtained by summing the entries of F_{e} and F_{e}^{lp}.

The structure of the GNN is similar to the one employed in Sec. 6.5.3. It is composed of one GNN layer with an input and output dimension of h and a ReLU activation function. Then, as output, an add-pooling operation is performed at the graph level to aggregate all the features of the nodes into a single embedding vector of dimension h. Finally, two linear layers compress the embedding first to a dimension of $h/2$ and then to an output dimension of 1. A sigmoid activation function is thus applied to the output to obtain the final probability value in the range (0-1). In Fig. 6.5d an illustration describing the predicted node classes for a sample graph is depicted.

6.5.6 Training Strategy

For all network training, the binary cross-entropy loss function is employed. The link prediction task of Sec. 6.5.3 and the node orientation prediction task of Sec. 6.5.4 are optimized together with a combined loss function, as:

$$\mathcal{L}_{\mathrm{link},\theta} = \mathcal{L}_{\mathrm{link}} + \mathcal{L}_{\theta} \tag{6.1}$$

The link prediction task is optimized by computing the loss among the positive (predicted value of 1) and negative (predicted value of 0) edges. The first relates to the edges present in the ground truth graph as described in Sec. 6.5.1. The latter is obtained starting from the set $\mathcal{E}_{\mathrm{knn}}$ by subtracting the ground truth edges.

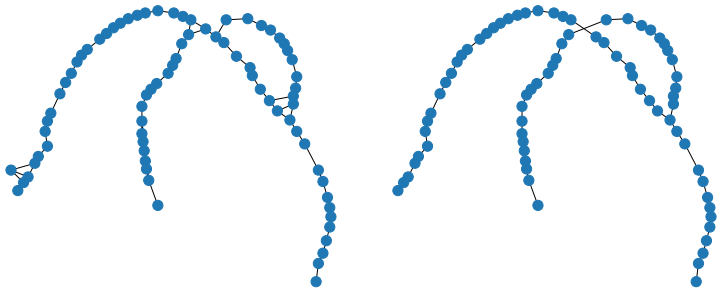
Regarding the node orientation prediction task, the smooth label strategy is employed for mapping the angular values from the $[0°, 180°]$ domain to a 180-dimensional domain in the form of a Gaussian distribution centered at the angular value [129]. Thus, this encoding is applied to the angular ground truth data of Sec. 6.5.1 and the network is trained to predict a similar output in the form of a classification task with respect to the 180-dim classes, i.e. the node orientation angles in degrees. The binary-cross entropy loss $\mathcal{L}_{\theta}$ is thus computed element-wise on the predicted and ground truth value across the vector dimension.

The node subgraph classification network of Sec. 6.5.5 is also trained with a binary-cross entropy loss between the node labels and the output provided by the network. The labels are obtained from the ground truth graph, where nodes in the proximity of *branch-points* and *intersection-points* locations are given a value of 1 instead of 0 assigned to all the others. Since the ratio between the 0-class and 1-class is quite unbalanced toward the first one, the 0-class is downsampled to roughly match the number of elements in the 1-class.

Algorithm 6: Solver

Input: $\mathcal{V}$, $\mathcal{E}_{\text{knn}}$, $\mathcal{Z}_{\text{lp}}$, $\mathcal{Z}_{\theta}$, $\mathcal{Z}_{\text{cls}}$
Output: $\mathcal{P}$, $\mathcal{B}$, $\mathcal{I}$

/* **Edges Filtering** */
$\mathcal{E}_{\text{lp}} \leftarrow$link_prediction_solver($\mathcal{V}$, $\mathcal{E}_{\text{knn}}$, $\mathcal{Z}_{\text{lp}}$)
$\mathcal{E}_{\text{f}} \leftarrow$ edges_simplification($\mathcal{V}$, $\mathcal{E}_{\text{lp}}$, $\mathcal{Z}_{\theta}$)
$\mathcal{G}' \leftarrow (\mathcal{V}, \mathcal{E}_f)$
/* **High-degree Nodes Handling** */
$\mathcal{E}_0 \leftarrow$ process_normal_nodes($\mathcal{V}$, $\mathcal{E}_{\text{f}}$, $\mathcal{Z}_{\text{cls}}$)
$\mathcal{E}_1 \leftarrow$ process_high_deg_nodes($\mathcal{V}$, $\mathcal{E}_0$, $\mathcal{Z}_{\text{cls}}$)
$\mathcal{G}'' \leftarrow (\mathcal{V}, \mathcal{E}_1)$
/* **Topology Output** */
$\mathcal{B} \leftarrow$ branch_points_extraction($\mathcal{G}''$)
$\mathcal{P} \leftarrow$ sections_extraction($\mathcal{G}''$)
$\mathcal{I} \leftarrow$ intersection_points_extraction($\mathcal{G}''$)
return $\mathcal{P}$, $\mathcal{B}$, $\mathcal{I}$



(a) $\mathcal{G}'$ (b) $\mathcal{G}''$

Figure 6.6: Effect of solver stages: (a) Obtained graph after the application of the edge filtering stage on the input graph of Fig. 6.5a; (b) The result of the high-degree nodes handling exploiting the estimated node classes.

6.6 Solver

To compute a final representation of the DMLO, the solver combines predicted link probabilities, node orientations, and node classes. The solver, outlined in Alg. 6, performs three main steps:

A) Edges filtering: the input graph edges are refined based on the predicted link probabilities (Sec. 6.5.3) and nodes orientations (Sec. 6.5.4) as described in Sec. 6.6.1;

B) High-degree nodes handling: based on the predicted node classes, high-degree nodes are interpreted and handled accordingly, as detailed in Sec. 6.6.2;

C) *Sections*, *branch-points* and *intersection-points* extraction: the final topology representation is computed, as detailed in Sec. 6.6.3.

In the following, each step is discussed in detail.

6.6.1 Edges Filtering

The first step of the solver is filtering the edges of the input graph. Indeed, the cardinality of $\mathcal{E}_{\text{knn}}$ is quite large since the majority of edges are either connecting two different

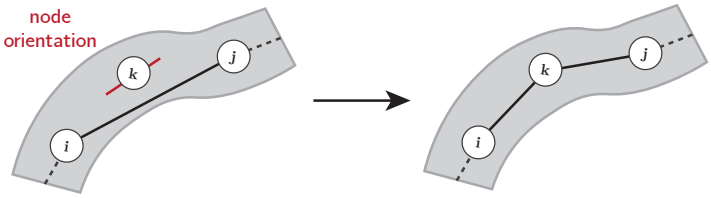


Figure 6.7: Node orientation solver updating the edges around node k in case its orientation is consistent with the nearby existing edge e_{ij} between nodes i and j.

DMLO *sections* or just redundant within the same *section*. The filtering is accomplished by first accounting the predicted link probabilities $\mathcal{Z}_{\text{lp}}$ (line 2), and then by considering also the node orientations $\mathcal{Z}_{\theta}$ (line 3).

Link Prediction Solver

The link prediction solver takes into account the predicted link probabilities $\mathcal{Z}_{\text{lp}}$ to select the edges that are more likely to be part of the same DMLO *section*. The link probabilities are computed by the GNN network of Sec. 6.5.3. The link prediction solver is employed to just remove all the very unlikely edges in the graph, thus a quite low threshold value (e.g. 0.05) on the computed probabilities is employed and only the edges having a probability greater than the threshold are kept. The resulting graph is $\mathcal{G}_{\text{lp}} = (\mathcal{V}, \mathcal{E}_{\text{lp}})$.

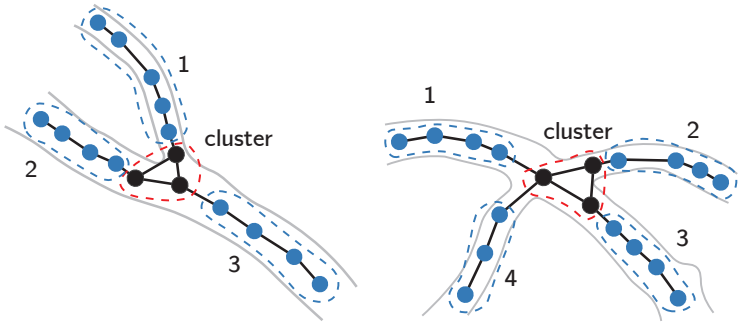
Node Orientation Solver

This step is performed to remove redundant edges that result, for instance, in graph cycles. The edges are checked for removing the ones that are mostly aligned with other nodes.

The alignment is evaluated by considering the node orientations $\mathcal{Z}_{\theta}$. In particular, the orientation of a node is an angular value, which is converted into a direction vector, e.g a direction vector of node v_i with orientation z_i^{θ} is obtained as $d_i = \left[\cos\left(z_i^{\theta}\right) \quad \sin\left(z_i^{\theta}\right)\right]^{\top}$. The cosine similarity is employed to evaluate the alignment between the direction vectors of the edges and the nodes. The cosine similarity for two general vectors a and b is defined as:

$$\text{cos\_sim}(a, b) = \frac{a^T b}{\|a\|\|b\|} \tag{6.2}$$

With reference to Fig. 6.7, let's consider one edge e_{ij} and a separate node v_k. The direction of edge e_{ij} is denoted as d_{ij}^{e}. The distance between the edge segment and node v_k is computed. If the distance is less than the estimated DMLO thickness at point v_k (this is easily obtained by M_{dist}), then the edge is removed if the orientation of node v_k is consistent with the one of edge e_{ij}, i.e. $|\text{cos\_sim}(d_k, d_{ij}^{e})| < 0.5$ where $|\cdot|$ denotes the absolute value. In particular, two additional edges e_{ik} and e_{kj} are inserted to compensate for the removal of e_{ij} as shown in Fig. 6.7. This process can be applied to multiple nodes simultaneously by projecting them onto the edge and placing new edges accordingly. Moreover, it is carried out for all nodes in $\mathcal{V}$ via matrix multiplications, resulting in efficient processing times. All the selected edges are stored in $\mathcal{E}_{\text{f}}$ obtaining the graph $\mathcal{G}'$ as shown in Fig. 6.6.



(a) *branch-point* cluster (b) *intersection-point* cluster

Figure 6.8: Clusters formation for *branch-point* and *intersection-point* regions. High-degree nodes are denoted in black.

6.6.2 High-Degree Nodes Handling

The predicted node classes $\mathcal{Z}_{\text{cls}}$ of Sec. 6.5.5 are exploited for further refining the graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}_{\text{f}})$. First, the predicted *normal* nodes are processed (Sec. 6.6.2). Then, high-degree nodes are clustered and processed, Sec. 6.6.2. Considering the predicted values $\mathcal{Z}_{\text{cls}}$, the nodes in $\mathcal{V}$ with a class score below a threshold value (e.g. 0.5) are considered *normal* nodes, $\mathcal{V}_{\text{normal}}$. The remaining nodes are considered *high-degree* nodes, $\mathcal{V}_{\text{high}}$.

Normal Nodes Processing

Among the *normal* nodes, those having a node degree higher than 2 in $\mathcal{G}'$ are processed in order to simplify the graph since their connectivity is in contrast with the predicted class. For a given node v_i characterized by a degree larger than 2, all the edges connected to v_i are retrieved and organized in combinations of two elements. Indeed, since v_i is assumed to be part of a DMLO *section*, it should have just two edges. Among the combinations, the one describing the smoothest connection is selected and the other edges are removed. The smoothness is evaluated by employing the cosine similarity as in eq. (6.2). In particular, considering two edges e_{ji} and e_{ik} of node v_i, their smoothness is obtained as $s_{jik} = \cos\_\text{sim}(d^e_{ji}, d^e_{ik})$. The final set of edges $\mathcal{E}_0$ is computed after processing all the *normal* nodes.

Cluster-based High-degree Nodes Processing

The nodes in $\mathcal{V}_{\text{high}}$ are processed by first organizing them into *clusters* considering as graph $\mathcal{G}_0 = (\mathcal{V}, \mathcal{E}_0)$. Each cluster describes one given DMLO area characterized by high-degree nodes. Indeed, since multiple nodes can describe the same *branch-points* and *intersection-points* area of the DMLO, neighboring nodes will have similar class predictions, as shown in Fig. 6.5d where the two high-degree areas of the graph show smooth values between 0 and 1. Thus, the clustering is performed by collecting the neighbors of each node in $\mathcal{V}_{\text{high}}$ and accounting for possible overlapping.

Then, each cluster is solved by connecting the individual *sections* in the most smooth way possible. Indeed, with reference to Fig. 6.8 showing the clustering results on a *branch-point* and *intersection-point*, the following assignments are made. For Fig. 6.8a, *sections* 2 and 3 are connected together. For Fig. 6.8b, the connection is established between *sections* 1-3 and 2-4. The merging is performed by evaluating proposals of merged sections and estimating a smoothness score similarly to what performed at the

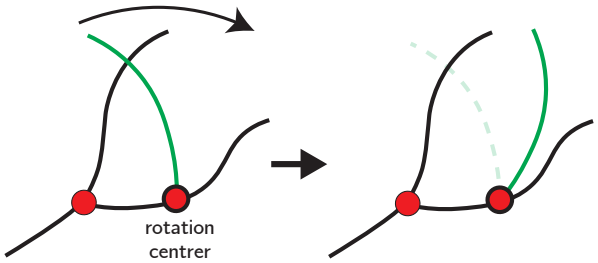


Figure 6.9: Schematic of the manipulation motion originated for the topology represented.

node-level on eq. 6.2.

The final set of edges $\mathcal{E}_1$ is thus obtained and the final graph $\mathcal{G}'' = (\mathcal{V}, \mathcal{E}_{\text{int}})$ is generated. In Fig. 6.6b an example of the computed graph $\mathcal{G}''$ is depicted.

6.6.3 Topology Output

The last step of the solver is the computation of the final topology representation. The *branch-points* are extracted from the graph $\mathcal{G}''$ by considering the nodes with degree 3 and stored in $\mathcal{B}$ (line 10 in Alg. 6). Then, the *sections* are computed by traversing the graph starting from the nodes with degree 1 or degree 3, and ending at the nodes with degree 1 or degree 3. The *sections* are stored in $\mathcal{P}$ (line 11). Finally, the *intersection-points* are obtained by checking the intersections between different *sections* in $\mathcal{P}$ and saving the result in $\mathcal{I}$ (line 12).

6.7 Topology-driven Manipulation

The usefulness and efficiency of the proposed DMLO topology representation approach are demonstrated by solving a manipulation task. Thus, the objective of this section is to introduce the proposed manipulation framework. More specifically, the problem of spreading a DMLO, i.e. removing all the *intersections* between different *sections* of the object, is analyzed.

In this context, the following assumptions are introduced: 1) the structure of the DMLO is not known, therefore no information from it is exploited, e.g. root location. 2) Given the stiffness of the DMLO considered, e.g. an automotive wire harness, the motion of a section of the DMLO is approximated by a rigid transformation constrained to rotate around the associated *branch-point*. To solve the manipulation task, a dual-arm robot system is employed. One arm is used to keep a part of the DMLO fixed in place. The second arm performs a pick-and-place operation with a motion trajectory around an estimated center of rotation, i.e. the *branch-point*, see Fig. 6.9.

The manipulation task, schematized in Alg. 7, works in the following way. As input, the topology representation of the DMLO contained in the scene under analysis is provided, in the form of *sections* $\mathcal{P}$, *branch-points* $\mathcal{B}$ and *intersection-points* $\mathcal{I}$. The goal is to solve all the possible intersections present in the DMLO (line 2), where each possible manipulation action computed is saved into $\mathcal{A}$ (line 1).

For a given intersection i, the segments characterizing it are retrieved at line 3. Given the two segments, the *section* of the DMLO above at the intersection i is evaluated from the 3D data (line 4). Thus, the *branch-point* constraining the motion of the *section* above is retrieved (line 5). This will give the rotation point of the movement performed by the second robot. Therefore, a full pose for the first arm is computed (line 6). The

Algorithm 7: DMLO Intersection Manipulation

Input: $\mathcal{P}$, $\mathcal{B}$, $\mathcal{I}$
Output: dual-arm manipulation to perform

1 $\mathcal{A} \leftarrow 0$
2 **for** $i \in \mathcal{I}$ **do**
3 $\quad s_{i,0}, s_{i,1} \leftarrow$ GetSegmentIntersection(i)
4 $\quad s_{\text{above}} \leftarrow$ getSegmentAbove$(s_{i,0}, s_{i,1})$
5 $\quad b \leftarrow$ getBranchPoint (s_{above})
6 $\quad p_{\text{center}} \leftarrow$ getCenterOfRotation(b)
7 $\quad \mathcal{T} \leftarrow$ getPickAndPlaceTrajcetory$(b,\ s_{\text{above}})$
8 $\quad \mathcal{A} \leftarrow [\mathcal{T}, p_{\text{center}}]$
9 $a^* \leftarrow$ getBestAction(A)
10 dual_arm_manipulation(a^*)

pick-and-place trajectory is computed considering also the center of rotation (line 7). The set of actions is updated at line 8.

The trajectory of the pick-and-place motion is obtained by simulating the motion of the section above with a circular trajectory around the obtained center (kept fixed by the other arm). The direction of rotation is computed by evaluating the direction in which the angle at the *branch-point* is increased, i.e. spreading objective. Given multiple *intersection-points*, the one to manipulate is selected as the most external in the graph topology (line 9), i.e. closer to graph terminal nodes with degree 1, since it is assumed to be easier to solve. Thereafter, the robots perform the manipulation (line 10). An illustration of the approach is provided in Fig. 6.9. The assignment of the task for each arm, either for the hold or motion action, is based on the distance between the robots and the target poses.

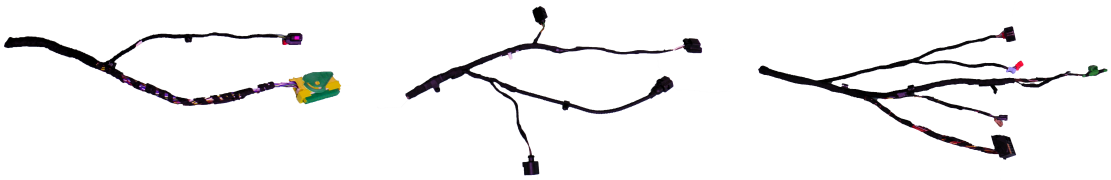
6.8 Experiments

The experiments involve the evaluation of the proposed topology representation pipeline in a real-world manipulation scenario. The experimental setup is composed of a robotic cell containing two Panda robots from Franka Emika shown in Fig. 6.1. The robotic cell is equipped with a Photoneo MotionCam3D structured-light camera statically mounted.

Details about the training dataset and hyperparameters are provided in Sec. 6.8.1 whereas the DMLOs employed in the experiments are presented in Sec. 6.8.2. Then, Sec. 6.8.3 discusses the manipulation experiment that is used to both evaluate the application of the proposed method for manipulation purposes and also to collect test samples. Sec. 6.8.4 analyses the collected samples. An analysis of the timing performances of the method is provided in Sec. 6.8.5. Comparisons with methods from the DLOs domain are performed in Sec. 6.8.6. The limitations are detailed in Sec. 6.8.7.

6.8.1 Training Details

The synthetic dataset introduced in Sec. 6.5.1 is exploited for the training of the method. It consists of a total of about 1600 graph samples used for train and 280 graph samples used for validation. First, the link predictor and node orientation predictor are trained. A learning rate of 10^{-3} is employed with the Adam optimizer. The batch size is set to 2, the hidden dimension h to 32, and the training is performed for 200,000 steps. The best model is selected based on the validation loss. Then, the node subgraph classificator is trained by employing the same dataset but also the previously trained predictors, which are kept frozen in this last training phase. A similar training process is employed. In



(a) *WH1* (b) *WH2* (c) *WH3*

Figure 6.10: DMLOs real-world test samples.

particular, a learning rate of 10^{-5} is used with the Adam optimizer. The batch size is set to 64, the hidden dimension h to 32, and the training is performed for 50,000 steps. The best model is selected based on the validation loss.

6.8.2 Test DMLOs

The experiments are performed using real-world automotive wire harnesses, shown in Fig. 6.10. Samples differ by increasing the complexity of the structures and the number of final endpoints. The samples, with reference to Fig. 6.10, are named and characterized in the following way:

- WH1: Plain taped cables with 1 *branch-points* and two end-points.
- WH2: Taped cables with side routing clips. A total of 4 *branch-points* and 5 end-points. Various final connectors with different sizes and shapes.
- WH3: Taped cables with side routing clips. A total of 6 *branch-points* with multiple end-points of different sizes.

6.8.3 Manipulation Experiment

The manipulation experiment is used to evaluate the performance of the proposed pipeline in a real-world scenario and to collect test samples. In particular, in the experiments, from the camera a point-cloud of the scene is acquired, the plane segmented, and the mask of the DMLO object generated. Therefore, the mask is processed by the proposed pipeline to extract the topological representation. The manipulation action is computed according to Sec. 6.7, and the robot is commanded to execute it. The DMLO is configured to have at least one *intersection-point*. The goal of the manipulation task is to remove the intersection, effectively disentangling the different sections of the DMLO. After the manipulation action is executed, the mask is processed again by the proposed pipeline to extract the obtained topological representation. The experiment is repeated 10 times for each of the three types of DMLOs, i.e., *WH1*, *WH2*, and *WH3*. In total, a set of 60 test samples is collected, 20 for each type of DMLO.

The sequence of manipulation steps for each type of the DMLO is displayed in Fig. 6.11. The figure contains snapshots collected during the execution of the experiments. The experiments show that the proposed pipeline is able to correctly identify all the *branch-points* and *sections* of the DMLOs which allowed a correct manipulation action to be computed. The results are also confirmed by the evaluation of the test samples, which is reported in Sec. 6.8.4.

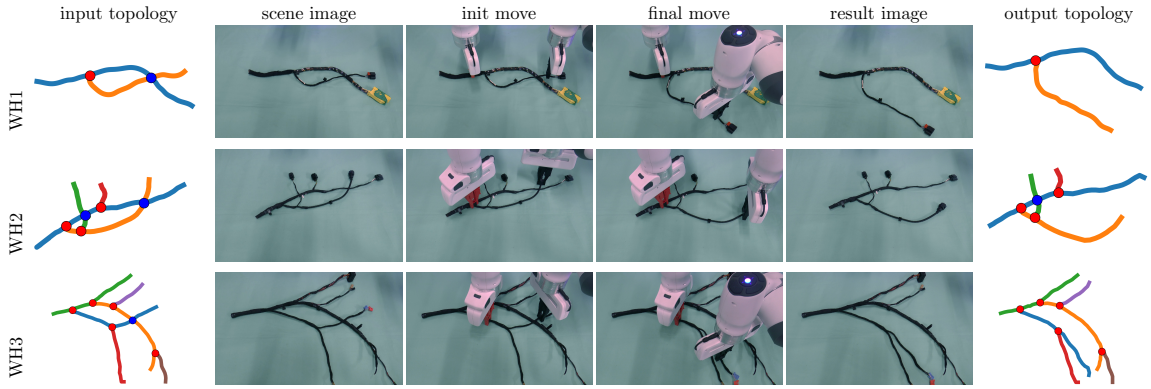


Figure 6.11: Snapshots of the untangling manipulation experiment employing three different automotive DMLOs. The *branch-points* and *intersection-points* are shown as red and blue dots.

Table 6.1: Offline evaluation results in terms of dice score for the link predicted in graph $\mathcal{G}''$ and detection accuracy for $\mathcal{B}$ and $\mathcal{I}$.

category	link pred	*branch-points*		*intersection-points*	
	dice score	accuracy	(ratio)	accuracy	(ratio)
WH1	0.952	1.0	(20/20)	1.0	(10/10)
WH2	0.938	0.952	(60/63)	0.938	(15/16)
WH3	0.944	0.846	(93/106)	0.923	(12/13)
total	0.944	0.915	(173/189)	0.949	(37/39)

6.8.4 Offline Evaluation

The offline evaluation is performed on the 60 samples collected in the manipulation experiment of Sec. 6.8.3. The evaluation is performed by comparing the extracted topology representation with the ground truth one. The ground truth is manually annotated by an expert operator. The annotation process consists of manually providing *branch-points*, *intersection-points*, and *sections* labels.

Overall Results

The results of the offline evaluation are presented in Tab. 6.1, with values reported for each test class and also as the average across the entire test dataset.

When it comes to predicting the edges for $\mathcal{G}''$, the results are expressed through the *dice score*, emphasizing the effectiveness of the approach in generating reliable edges to construct the topology graph.

The proposed pipeline successfully identifies the majority of *branch-points* within the DMLO samples. Referring to Tab. 6.1, it is worth noting that for *WH2* and *WH3* samples, there are 3 and 7 instances of false positive *branch-points* detected, respectively. Additionally, there are 6 false negative *branch-points* detected for the *WH3* sample.

As for the *intersection-points*, there are inaccuracies in the *WH2* sample, with one *intersection-point* incorrectly identified as a *branch-point*, and in the *WH3* sample, where one *intersection-point* remains undetected due to graph-related inaccuracies.

Performance of the Single Components

To provide a more detailed analysis of the performance of each individual learned component within the approach, here separate evaluations for the link predictor (Sec. 6.5.3), node orientation predictor (Sec. 6.5.4), and node classifier (Sec. 6.5.5) are presented. The results are shown in Fig. 6.12. Since the first two components are involved in binary classification tasks, a common evaluation metric is employed — specifically, the *precision-recall* curve, which provides a graphical representation of a classifier's performance across various thresholds.

Ground truth data is utilzed to asses the performance of the single components. For the link predictor, ground truth edges are extracted from the reference graph and utilized for assessment. For the node orientation predictor, ground truth node orientations are obtained by fitting spline curves along the labeled sections and by determining the tangent lines to the curves at each node's position. The node classifier, on the other hand, is assessed by utilizing the labeled *branch-points* and *intersection-points*. It's worth noting that the location of the assigned high-degree node point may vary in relation to the node's position in the graph. Consequently, the classifier is expected to provide a one-class prediction within a specified radius, set at two times the thickness of the DMLO in the relevant area.

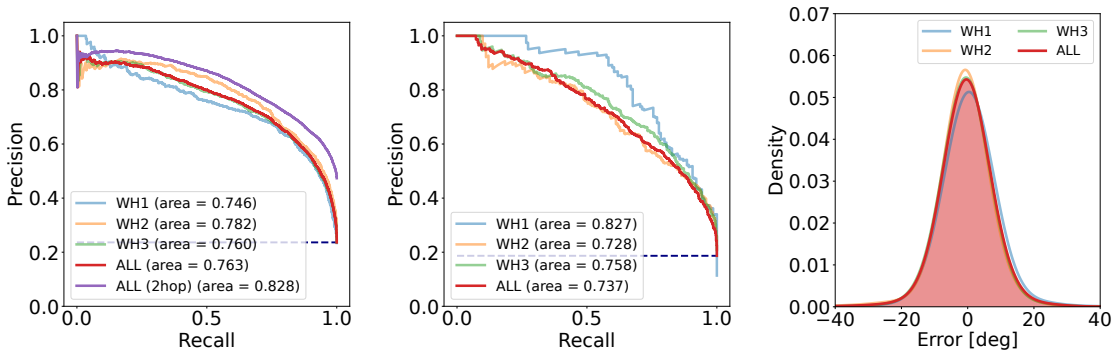
When examining the outcomes of the link prediction task (as shown in Fig. 6.12a), it's notable that the predictor faces the challenge of accurately classifying only the two closest neighboring nodes to the considered one. Rather than delivering a sharp distinction, the predictor often produces a more gradual prediction, occasionally assigning high probabilities to the two more distant nodes as well. Indeed, this is clearly illustrated by the *ALL* and *ALL (2hop)* curves in the figure. With *ALL*, an edge is considered positive if connects two nodes at a 1-hop distance along a section of the DMLO. Instead, with *ALL (2hop)* are considered positive edges both the 1-hop and 2-hops ones within the same section of the DMLO. In the proposed methodology, this behavior doesn't pose an issue, as it effectively filters out unnecessary edges, as elaborated in Sec. 6.6.1.

Concerning the node classification task, the plot in Fig. 6.12b provides clear evidence of the capabilities of the classifier in locating the areas of the graph subject to high connectivity.

Conversely, the orientation estimation error, as depicted in Figure 6.12c, exhibits a distribution akin to the normal curve, with statistical characteristics of approximately 0.63 ± 10.03 degrees. Notably, it is essential to highlight that this error is primarily influenced by incorrect edges within the mask and the presence of obtrusive objects such as clips and plugs.

6.8.5 Timings

In Tab. 6.2, the recorded timings for the primary procedures of the proposed method across the test set collected in Sec. 6.8.3 are presented. The timing values are expressed in milliseconds and have been obtained on a laptop PC running Ubuntu 20, equipped with an Intel Core i7-12700H processor, 16GB of RAM, and an Nvidia 3050Ti graphics card, all powered by PyTorch 1.10 with CUDA enabled. The table provides clear evidence of the efficiency of the approach when executed on a standard hardware configuration. Moreover, it is noteworthy to emphasize that the computational demands increase as the complexity of the scene grows. This is particularly evident in the average number of graph nodes across the *WH1*, *WH2* and *WH3* samples, which stand at 62, 82, and 140, respectively.



(a) *link prediction* (b) *node classification* (c) *node orientation*

Figure 6.12: Offline evaluation results of the single learned components of the proposed pipeline for the different DMLOs composing the test set. With *ALL*, the entire set of samples is considered.

Table 6.2: Timings of the main procedures of the approach across the test set. Values in milliseconds.

Procedure	WH1	WH2	WH3	All
Graph Generation	7.3	8.5	12.1	9.3
Link & Node Orientation Prediction	9.2	9.3	9.6	9.3
Node Classification	9.6	12.3	19.5	13.8
Edges Filtering	4.6	6.0	10.2	6.9
High-degree Node Solver	2.1	3.6	8.1	4.6
Total	32.8	39.7	59.7	43.9

6.8.6 Comparison with Methods from DLOs Domain

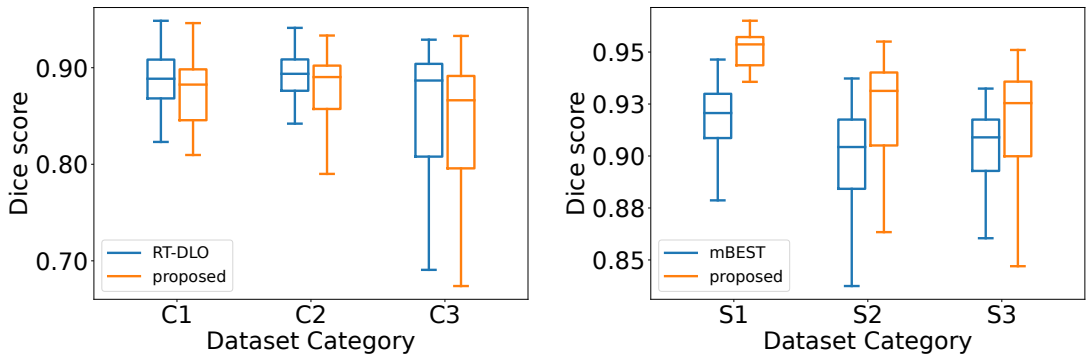
In the existing literature, there are no direct and comprehensive comparatives available for the proposed pipeline. However, it is possible to benchmark it against methods designed for similar tasks, particularly those related to the perception of DLOs. In this context, two methods emerge as possible benchmarks, namely *RT-DLO* (Sec. 3.5) and *mBEST* [24]. Both of these baseline methods focus on instance segmentation of DLOs: starting from a semantic segmentation mask, they produce a colored mask where each DLO is represented by a unique color corresponding to its ID. In particular, *RT-DLO* employs a graph-based representation of the scene similarly to the proposed framework. Instead, *mBEST* exploits a skeleton-based approach.

DLOs Instance Segmentation

The proposed method is adjusted to produce a similar output to the one of *RT-DLO* and *mBEST* for what concerns the instance segmentation task. Therefore, given the graph topology, a colored mask is generated based on the extracted *section* with mask thickness information obtained from the distance transform image M_{dist}.

For the evaluation, the same test dataset as the respective works is utilized, which includes dataset categories *C1*, *C2* and *C3* from [15] and *S1*, *S2*, and *S3* from [24]. The evaluation metric employed is the dice score.

The results of these comparisons are shown in Fig. 6.13 in the form of boxplots. From the plots, it is evident that the proposed method serves as a viable alternative to the baseline methods, with the primary distinction being the versatility of the proposed



(a) comparison against *RT-DLO*

(b) comparison against *mBEST*

Figure 6.13: Comparisons of the proposed method against *RT-DLO* [15] and *mBEST* [24] performing the instance segmentation task on the respective proposed DLOs test datasets.

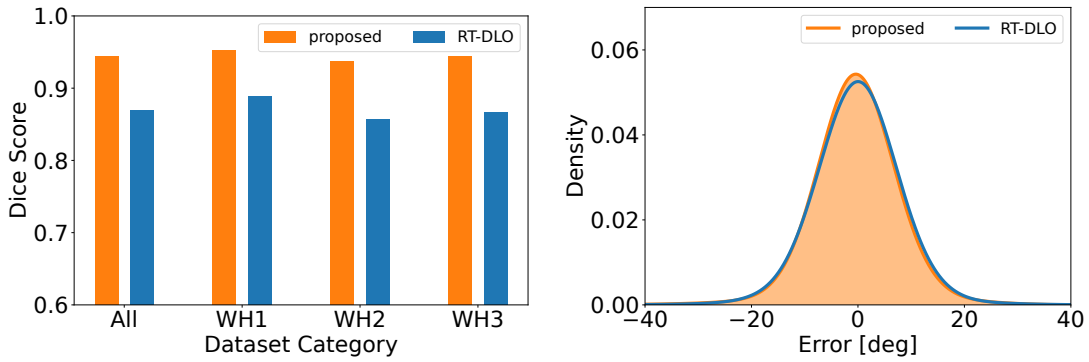
approach: it can be seamlessly applied in a context involving both DMLOs and DLOs. When compared to *mBEST*, it exhibits robust performance and the capability to handle self-loops without requiring any modifications compared to the case of DMLOs. The slight performance dip in comparison to *RT-DLO* is mainly attributed to unmerged connections between sections of the same DLO. This can be expected since the proposed method does not make any assumptions regarding high-degree regions in the graph, which can either be *branch-points* or *intersection-points*. This is in contrast to the DLO methods, where the presence of a high-degree region in the graph is due solely to an intersection thus the overall connectivity is enforced.

Ablation Study

The *RT-DLO* approach is similar to the proposed one in terms of graph-based representation. More specifically, the proposed method can be viewed as a generalization of *RT-DLO*, capable of encompassing both DLOs and DMLOs. This is because: 1) it avoids making assumptions about the nature of high-degree nodes in the graph, which can be either *branch-points* or *intersection-points*, and 2) the probability of connecting two nodes is learned from the data and is not fixed as a function of distance and node orientation alignments.

To better evaluate the differences between the two methods, a study about replacing several components of the proposed pipeline with the alternative ones of *RT-DLO* is conducted in this section. Specifically, the utilization of *RT-DLO*'s graph generation and link prediction approach to replace the proposed graph initialization stage (Sec. 6.4) and the GNN-based link prediction (Sec. 6.5.3) is evaluated. The results of this evaluation can be seen in Fig. 6.14a. One notable advantage of the proposed graph initialization stage is its ability to maintain consistent node density, even when dealing with highly diverse masks. In contrast, the sampling ratio of *RT-DLO* is manually adjusted to match the number of nodes in the proposed approach for a fair comparison. Overall, the proposed method demonstrates superior performance in predicting consistent edges within the node-set, highlighting the advantage of the learning-based approach.

Additionally, the node orientation prediction task in Sec. 6.5.4 is replaced with *RT-DLO*'s specific neural network approach, which accomplishes the same task but using image crops of M_{dist}. The results are presented in Fig.6.14b. The error in node orientation prediction is comparable between the two approaches, with the primary distinction lying in the architecture described in Sec. 6.5.4, which eliminates the need



(a) link prediction scores

(b) node orientation errors

Figure 6.14: Ablation study of link prediction and node orientation tasks with components proposed in *RT-DLO*.

for mask cropping and the associated computational overhead.

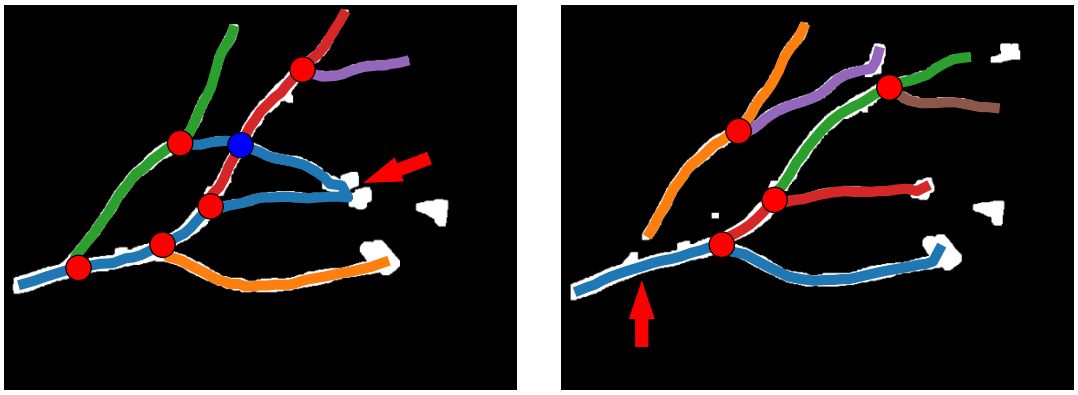
An attempt is also made to utilize the graph generated by *RT-DLO* along with the predicted node orientations for conducting node subgraph classification in Sec. 6.5.5. Nevertheless, the outcomes proved to be highly unsatisfactory, underscoring the crucial importance of the preliminary encoding stage carried out in Secs. 6.5.3 and 6.5.4.

6.8.7 Limitations

The proposed approach is not exempt from limitations and drawbacks, which manifest themselves in two primary areas: 1) the representation of topology and 2) the manipulation strategy. As for the former, key limitations are depicted in Fig.6.15. Specifically, Fig.6.15a illustrates a scenario where two distinct branch *sections* are merged into a single one, while Fig. 6.15b illustrates a situation where a *branch-point* is not detected. These issues arise because the DMLO's topology is reconstructed based solely on the input mask, making it susceptible to inaccuracies and suboptimal topologies due to gaps and other mask artifacts. While the graph-based approach does provide a degree of resilience against these issues, it is limited in its effectiveness up to a certain extent. Addressing the first limitation might be achievable by imposing a consistency constraint on the smoothness of the extracted *sections*. Nevertheless, this solution introduces a new parameter to tune and does not represent a definitive resolution.

Regarding the manipulation aspect, the principal limitation originates from the simplicity of the proposed manipulation strategy in Sec. 6.7. Indeed, the method can fail in the case of complex DMLOs, where the *branch-points* are not easily accessible or there is no space to perform the manipulation action. This simplicity is intentional, as the chapter's objective is to present a methodology for deriving a topological description of a DMLO without prior knowledge of its actual structure. Hence, a straightforward manipulation strategy is chosen to highlight the advantages of using this representation.

Both of these limitations could be mitigated by incorporating knowledge of the DMLO's structure and, consequently, implementing a form of matching between the extracted DMLO and the expected one. As a result, both the topology extraction process and manipulation could benefit substantially from such insights. However, these enhancements are deferred as future research work.



(a) merging of different *sections* (b) missing *branch-point*

Figure 6.15: Limitations of the proposed method concerning topologies derived from masks that exhibit artifacts and gaps. To pinpoint the specific issue, a red arrow is employed.

6.9 Conclusions

In this chapter, a new approach for the efficient and accurate estimation of a topological graph-based representation of DMLOs configurations is presented. The proposed method is based on a graph neural network that embeds node and edge features to fully characterize the DMLO structure in terms of node connectivity, orientation and class. The obtained topology representation is exploited to perform a dual-arm manipulation task, which is demonstrated in a real-world scenario using DMLOs from the automotive field. The experiments show that the proposed method is able to correctly identify the *branch-points* and *sections* of the DMLOs which allowed a correct disentangling manipulation action to be computed. Moreover, the proposed method is compared with methods from the DLOs domain, showing comparable performance in terms of instance segmentation capabilities with the main advantage of being able to handle DMLOs and DLOs in a unified framework. The method does not come without limitations, which are analyzed in the experiments. In particular, the proposed method is susceptible to inaccuracies and suboptimal topologies due to important gaps and other relevant mask artifacts. Furthermore, the manipulation strategy is simple and can fail in the case of complex DMLOs. Therefore, future works can be focused on the integration of knowledge about the topological structure of the DMLO. This will involve investigating methods for matching and aligning the extracted DMLO topology with the actual scene. Furthermore, a more sophisticated dual-arm manipulation strategy can be developed.

Chapter 7

Conclusions

The research activities presented in this thesis introduce some novel contributions in the field of robotic perception and manipulation of DLOs and DMLOs. These contributions encompass both perception and manipulation aspects, and a detailed description of the main results is provided in the following.

Percception

- Efficient and minimal human intervention is possible in generating datasets for training deep learning models through the use of synthetic data generation methods. This can be achieved by modeling the DLOs as curves and utilizing a rendering engine, or by leveraging the chroma separation principle. In both approaches, it is feasible to reduce the domain gap concerning the real world by enhancing the generated dataset with real images annotated using a weakly supervised methodology.
- Using deep learning models trained on purpose-generated datasets to perform semantic segmentation of DLOs represents an effective approach for simplifying the challenge of image perception.
- Combining the semantic segmentation step with a graph-based representation can be utilized for the instance segmentation of DLOs. This method enables the robust and precise extraction of DLO instances within an image, even when dealing with degradation and noise in the segmentation mask.
- The use of a graph-based representation of the DLOs in the image scene can be extended to DMLOs, effectively tackling the two perception problems with a common and unified approach.
- Accurate 3D reconstruction of DLOs can be attained by integrating the instance segmentation techniques developed with a multi-view stereo reconstruction approach. Specifically, a robot and a single 2D camera in an eye-in-hand configuration can be deployed to execute the reconstruction process, capitalizing on the robot's mobility to enhance reconstruction outcomes.

Manipulation

- Leveraging a topological representation of the DMLOs within the scene facilitates the execution of disentangling manipulation tasks using a straightforward yet effective manipulation strategy.

- A NN can be utilized to create an efficient predictive model of DLO dynamics, enabling the planning of manipulation actions. The NN model can be fully trained by simulating DLO dynamics through a physics-based analytical model.
- The utilization of a differentiable DLO model framework conditioned on model parameters, offers the opportunity to conduct gradient-based optimization on the same model for either the manipulation action or model parameters. In particular, during the execution of a manipulation task, the model parameters can be optimized to minimize the error between the NN model and the real DLO dynamics, thus improving the accuracy of the predictive model.

Despite the promising outcomes achieved, this thesis also paves the way for further research and exploration. Specifically, the following areas offer opportunities for future investigation:

- To reduce the domain gap arising from synthetic data usage, future work may involve leveraging domain adaptation techniques and enhancing the realism of synthetic image generation. Furthermore, improving the weakly supervised annotation methodology through the introduction of more intuitive and robust annotation tools can enhance the efficiency of the annotation process.
- The instance segmentation approach can be further improved in two different aspects: 1) Reducing the reliance on the quality of the semantic segmentation mask to enhance robustness against segmentation errors; 2) Enhancing the aggregation process for different sections of the same DLO by incorporating a model-based approach.
- Enhancements in 3D reconstruction methods could involve accounting for the dynamic nature of the scene. This would entail considering robot motion during the instance segmentation process, which could lead to more comprehensive DLO reconstruction in complex scenes and reduce sensitivity to instance segmentation errors.
- Improve the NN model of the DLO dynamics by extending it to 3D and by considering more sophisticated physical effects. Fixture and contact points could be also integrated into the NN model, thus enabling the execution of more complex manipulation through the utilization of the surrounding environment.
- Develop a more complex manipulation strategy for DMLOs, that would enable the execution of more complex manipulation tasks. In particular, the integration of the DLO dynamical model within the existing constraints posed by *branch-points* is an interesting aspect to be investigated.

Bibliography

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. "SLIC superpixels compared to state-of-the-art superpixel methods". In: *IEEE transactions on pattern analysis and machine intelligence* 34.11 (2012), pp. 2274–2282.

[2] Omid Aghajanzadeh, Miguel Aranda, Juan Antonio Corrales Ramon, Christophe Cariou, Roland Lenain, and Youcef Mezouar. "Adaptive Deformation Control for Elastic Linear Objects". In: *Frontiers in Robotics and AI* (2022).

[3] Rodrigo Benenson, Stefan Popov, and Vittorio Ferrari. "Large-scale interactive object segmentation with human annotators". In: *CVPR*. 2019.

[4] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. "Soft-NMS–improving object detection with one line of code". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5561–5569.

[5] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. "YOLACT++: Better Real-time Instance Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[6] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. "Yolact: Real-time instance segmentation". In: *Proceedings of the IEEE/CVF Conf. ICCV*. 2019.

[7] Gunilla Borgefors. "Distance transformations in digital images". In: *Computer vision, graphics, and image processing* (1986).

[8] Joao Borrego, Atabak Dehban, Rui Figueiredo, Plinio Moreno, Alexandre Bernardino, and José Santos-Victor. "Applying domain randomization to synthetic data for object category detection". In: *arXiv preprint arXiv:1807.09834* (2018).

[9] Yuri Y Boykov and M-P Jolly. "Interactive graph cuts for optimal boundary & region segmentation of objects in ND images". In: *ICCV*. IEEE. 2001.

[10] John Brooke. *SUS - A quick and dirty usability scale*. CRC Press, 1996.

[11] F. Bullo. *Lectures on Network Systems*. 1.6. Kindle Direct Publishing, 2022. ISBN: 978-1986425643. URL: `https://fbullo.github.io/lns`.

[12] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. "Yale-CMU-Berkeley dataset for robotic manipulation research". In: *The International Journal of Robotics Research* 36.3 (2017), pp. 261–268.

[13] Alessio Caporali, Kevin Galassi, and Gianluca Palli. "3D DLO Shape Detection and Grasp Planning from Multiple 2D Views". In: *2021 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE. 2021, pp. 424–429.

[14] Alessio Caporali, Kevin Galassi, and Gianluca Palli. "Deformable Linear Objects 3D Shape Estimation and Tracking From Multiple 2D Views". In: *IEEE Robotics and Automation Letters* (2023).

[15] Alessio Caporali, Kevin Galassi, Bare Luka Žagar, Riccardo Zanella, Gianluca Palli, and Alois C Knoll. "RT-DLO: Real-Time Deformable Linear Objects Instance Segmentation". In: *IEEE Transactions on Industrial Informatics* (2023).

[16] Alessio Caporali, Kevin Galassi, Riccardo Zanella, and Gianluca Palli. "Deformable Multi-Linear Objects Manipulation Leveraging on GNN Topology Representation Learning". In: *Submitted to Transactions on Automation Science and Engineering* (2023).

[17] Alessio Caporali, Kevin Galassi, Riccardo Zanella, and Gianluca Palli. "FASTDLO: Fast Deformable Linear Objects Instance Segmentation". In: *Robotics and Automation Letters* (2022).

[18] Alessio Caporali, Piotr Kicki, Kevin Galassi, Riccardo Zanella, Krzysztof Walas, and Gianluca Palli. "Deformable Linear Objects Manipulation with Online Model Parameters Estimation". In: *Submitted to IEEE Robotics and Automation Letters* (2023).

[19] Alessio Caporali, Matteo Pantano, Lucas Janisch, Daniel Regulin, Gianluca Palli, and Dongheui Lee. "A Weakly Supervised Semi-automatic Image Labeling Approach for Deformable Linear Objects". In: *IEEE Robotics and Automation Letters* (2023).

[20] Alessio Caporali, Riccardo Zanella, Daniele De Greogrio, and Gianluca Palli. "Ariadne+: Deep Learning–Based Augmented Framework for the Instance Segmentation of Wires". In: *TII* (2022).

[21] Jia-Ren Chang and Yong-Sheng Chen. "Pyramid stereo matching network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5410–5418.

[22] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. "Blendmask: Top-down meets bottom-up for instance segmentation". In: *Proceedings of the IEEE/CVF Conf. CVPR*. 2020.

[23] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

[24] Andrew Choi, Dezhong Tong, Brian Park, Demetri Terzopoulos, Jungseock Joo, and Mohammad Khalid Jawed. "mBEST: Realtime Deformable Linear Object Detection Through Minimal Bending Energy Skeleton Pixel Traversals". In: *arXiv preprint arXiv:2302.09444* (2023).

[25] William S. Cleveland. "LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression". In: *The American Statistician* (1981).

[26] Konrad P Cop, Arne Peters, Bare L Žagar, Daniel Hettegger, and Alois C Knoll. "New metrics for industrial depth sensors evaluation for precise robotic applications". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5350–5356.

[27] Daniele De Gregorio, Gianluca Palli, and Luigi Di Stefano. "Let's take a walk on superpixels graphs: Deformable linear objects segmentation and model estimation". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 662–677.

[28] Daniele De Gregorio, Matteo Poggi, Pierluigi Zama Ramirez, Gianluca Palli, Stefano Mattoccia, and Luigi Di Stefano. "Beyond the Baseline: 3D Reconstruction of Tiny Objects With Single Camera Stereo Robot". In: *IEEE Access* (2021).

[29] Daniele De Gregorio, Alessio Tonioni, Gianluca Palli, and Luigi Di Stefano. "Semiautomatic Labeling for Deep Learning in Robotics". In: *IEEE Transactions on Automation Science and Engineering* 17.2 (2019), pp. 611–620.

[30] Yuhong Deng, Xueqian Wang, and Lipeng Chen. "Learning visual-based deformable object rearrangement with local graph neural networks". In: *Complex & Intelligent Systems* (2023), pp. 1–14.

[31] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. "Blenderproc". In: *arXiv preprint arXiv:1911.01911* (2019).

[32] Carl Doersch and Andrew Zisserman. "Sim2real transfer learning for 3D human pose estimation: motion to the rescue". In: *Advances in Neural Information Processing Systems*. 2019, pp. 12949–12961.

[33] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[34] Pedro F Felzenszwalb and Daniel.P Huttenlocher. "Efficient graph-based image segmentation". In: *Int. J. Comp. Vision* 59.2 (2004), pp. 167–181.

[35] Alejandro F Frangi, Wiro J Niessen, Koen L Vincken, and Max A Viergever. "Multiscale vessel enhancement filtering". In: *Medical Image Computing and Computer-Assisted Intervention*. Springer. 1998, pp. 130–137.

[36] Simon Fröhlig, Maximilian von Fabris auf Mayerhofen, Moritz Meiners, and Jörg Franke. "Three-dimensional pose estimation of deformable linear object tips based on a low-cost, two-dimensional sensor setup and AI-based evaluation". In: *Procedia CIRP* (2022).

[37] Yasutaka Furukawa, Carlos Hernández, et al. "Multi-view stereo: A tutorial". In: *Foundations and Trends® in Computer Graphics and Vision* (2015).

[38] Kevin Galassi and Gianluca Palli. "Robotic wires manipulation for switchgear cabling and wiring harness manufacturing". In: *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2021, pp. 531–536.

[39] David Gallup, Jan-Michael Frahm, Philippos Mordohai, and Marc Pollefeys. "Variable baseline/resolution stereo". In: *2008 IEEE conference on computer vision and pattern recognition*. IEEE. 2008, pp. 1–8.

[40] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[41] A Handa, V Patraucean, V Badrinarayanan, S Stent, and R Cipolla. "Scenenet: understanding real world indoor scenes with synthetic data. arXiv preprint (2015)". In: *arXiv preprint arXiv:1511.07041* (2015).

[42] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud. *Time-of-flight cameras: principles, methods and applications*. Springer Science & Business Media, 2012.

[43] Sandra G. Hart and Lowell E. Staveland. "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research". In: *Advances in Psychology*. Elsevier, 1988.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[45] Kejing He, Congying Sui, Tianyu Huang, Rong Dai, Congyi Lyu, and Yun-Hui Liu. "3D Surface reconstruction of transparent objects using laser scanning with LTFtF method". In: *Optics and Lasers in Engineering* 148 (2022), p. 106774.

[46] Tomas Hermansson, Robert Bohlin, Johan S Carlson, and Rikard Söderberg. "Automatic assembly path planning for wiring harness installations". In: *Journal of manufacturing systems* 32.3 (2013), pp. 417–422.

[47] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M López. "Embedded real-time stereo estimation via semi-global matching on the GPU". In: *Procedia Computer Science* (2016).

[48] Heiko Hirschmuller. "Stereo processing by semiglobal matching and mutual information". In: *IEEE Transactions on pattern analysis and machine intelligence* 30.2 (2007), pp. 328–341.

[49] Dominik Honegger, Torsten Sattler, and Marc Pollefeys. "Embedded real-time multi-baseline stereo". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5245–5250.

[50] Xuzhao Huang, Dayuan Chen, Yuhao Guo, Xin Jiang, and Yunhui Liu. "Untangling multiple deformable linear objects in unknown quantities with complex backgrounds". In: *IEEE Transactions on Automation Science and Engineering* (2023).

[51] Benjamin Irving. "maskSLIC: regional superpixel generation with application to local pathology characterisation in medical images". In: *arXiv preprint arXiv:1606.09518* (2016).

[52] Jagadeesan Jayender, Rajnikant V Patel, and Suwas Nikumb. "Robot-assisted active catheter insertion: Algorithms and experiments". In: *The International Journal of Robotics Research* 28.9 (2009), pp. 1101–1117.

[53] Xin Jiang, Kyong-mo Koo, Kohei Kikuchi, Atsushi Konno, and Masaru Uchiyama. "Robotized assembly of a wire harness in a car production line". In: *Advanced Robotics* 25.3-4 (2011), pp. 473–489.

[54] Azarakhsh Keipour, Maryam Bandari, and Stefan Schaal. "Deformable One-Dimensional Object Detection for Routing and Manipulation". In: *Robotics and Automation Letters* (2022).

[55] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. "Intel realsense stereoscopic depth cameras". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 1–10.

[56] Alaa Khalifa and Gianluca Palli. "Symplectic integration for multivariate dynamic spline-based model of deformable linear objects". In: *Journal of Computational and Nonlinear Dynamics* (2022).

[57] Piotr Kicki, Michał Bednarek, Paweł Lembicz, Grzegorz Mierzwiak, Amadeusz Szymko, Marek Kraft, and Krzysztof Walas. "Tell me, what do you see?—interpretable classification of wiring harness branches with deep neural networks". In: *Sensors* 21.13 (2021), p. 4327.

[58] Piotr Kicki, Michał Bidziński, and Krzysztof Walas. *Learning Quasi-Static 3D Models of Markerless Deformable Linear Objects for Bimanual Robotic Manipulation*. 2023.

[59] Thomas N Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2016.

[60] Thomas N Kipf and Max Welling. "Variational graph auto-encoders". In: *arXiv preprint arXiv:1611.07308* (2016).

[61] Rita Laezza and Yiannis Karayiannidis. "Learning Shape Control of Elastoplastic Deformable Linear Objects". In: *IEEE Int. Conf. ICRA*. 2021.

[62] Romain Lagneau, Alexandre Krupa, and Maud Marchal. "Automatic shape control of deformable wires based on model-free visual servoing". In: *IEEE Robotics and Automation Letters* (2020).

[63] Robert Lee, Masashi Hamaya, Takayuki Murooka, Yoshihisa Ijiri, and Peter Corke. "Sample-efficient learning of deformable linear object manipulation in the real world through self-supervision". In: *IEEE Robotics and Automation Letters* (2021).

[64] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "EPnP: An Accurate O(n) Solution to the PnP Problem". In: *Int. Journal of Computer Vision* (2009).

[65] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, and Jian Sun. "Scribblesup: Scribble-supervised convolutional networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3159–3167.

[66] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[67] Fei Liu, Entong Su, Jingpei Lu, Mingen Li, and Michael C Yip. "Robotic manipulation of deformable rope-like objects using differentiable compliant position-based dynamics". In: *IEEE Robotics and Automation Letters* (2023).

[68] Jiang-Jiang Liu, Qibin Hou, Ming-Ming Cheng, Jiashi Feng, and Jianmin Jiang. "A simple pooling-based design for real-time salient object detection". In: *IEEE/CVF Conf. CVPR*. 2019.

[69] Jiawei Liu, Qiang Wang, Huijie Fan, Shuai Wang, Wentao Li, Yandong Tang, Danbo Wang, Mingyi Zhou, and Li Chen. "Automatic Label Correction for the Accurate Edge Detection of Overlapping Cervical Cells". In: *arXiv preprint arXiv:2010.01919* (2020).

[70] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. "Swin transformer: Hierarchical vision transformer using shifted windows". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.

[71] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. "A convnet for the 2020s". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 11976–11986.

[72] Alberta Longhini, Marco Moletta, Alfredo Reichlin, Michael C Welle, David Held, Zackory Erickson, and Danica Kragic. "Edo-net: Learning elastic properties of deformable objects from graph dynamics". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 3875–3881.

[73] Bo Lu, XB Yu, JW Lai, KC Huang, Keith CC Chan, and Henry K Chu. "A Learning Approach for Suture Thread Detection With Feature Enhancement and Segmentation for 3-D Shape Reconstruction". In: *IEEE Transactions on Automation Science and Engineering* 17.2 (2019), pp. 858–870.

[74] Benjamin Lutz, Lucas Janisch, Dominik Kisskalt, Daniel Regulin, and Jörg Franke. "Interactive Image Segmentation Using Superpixels and Deep Metric Learning for Tool Condition Monitoring". In: *Procedia CIRP* 118 (2023), pp. 459–464.

[75] Kangchen Lv, Mingrui Yu, Yifan Pu, Xin Jiang, Gao Huang, and Xiang Li. "Learning to estimate 3-d states of deformable linear objects from single-frame occluded point clouds". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 7119–7125.

[76] Naijing Lv, Jianhua Liu, Xiaoyu Ding, Jiashun Liu, Haili Lin, and Jiangtao Ma. "Physically based real-time interactive assembly simulation of cable harness". In: *Journal of Manufacturing Systems* (2017).

[77] Naijing Lv, Jianhua Liu, and Yunyi Jia. "Dynamic Modeling and Control of Deformable Linear Objects for Single-Arm and Dual-Arm Robot Manipulations". In: *IEEE Transactions on Robotics* (2022).

[78] Xiao Ma, David Hsu, and Wee Sun Lee. *Learning Latent Graph Dynamics for Visual Manipulation of Deformable Objects*. 2022. arXiv: `2104.12149 [cs.RO]`.

[79] Kevis-Kokitsi Maninis, Sergi Caelles, Jordi Pont-Tuset, and Luc Van Gool. "Deep extreme cut: From extreme points to object segmentation". In: *CVPR*. 2018.

[80] Rene J Moreno Masey, John O Gray, Tony J Dodd, and Darwin G Caldwell. "Guidelines for the design of low-cost robots for the food industry". In: *Industrial Robot: An International Journal* (2010).

[81] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image segmentation using deep learning: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3523–3542.

[82] Lei Mou, Yitian Zhao, Huazhu Fu, Yonghuai Liu, Jun Cheng, Yalin Zheng, Pan Su, Jianlong Yang, Li Chen, Alejandro F Frangi, et al. "CS2-Net: Deep learning segmentation of curvilinear structures in medical imaging". In: *Medical image analysis* 67 (2021), p. 101874.

[83] David Navarro-Alarcon, Hiu Man Yip, Zerui Wang, Yun-Hui Liu, Fangxun Zhong, Tianxue Zhang, and Peng Li. "Automatic 3-D Manipulation of Soft Objects by Robotic Arms With an Adaptive Deformation Model". In: *IEEE Trans. on Robotics* (2016).

[84] Gabriel E Navas-Reascos, David Romero, Johan Stahre, and Alberto Caballero-Ruiz. "Wire harness assembly process supported by collaborative robots: Literature review and call for R&D". In: *Robotics* 11.3 (2022), p. 65.

[85] Adrian K. T. Ng, Leith K. Y. Chan, and Henry Y. K. Lau. "A low-cost lighthouse-based virtual reality head tracking system". In: *IC3D*. IEEE, 2017.

[86] Huong Giang Nguyen, Marlene Kuhn, and Jörg Franke. "Manufacturing automation for automotive wiring harnesses". In: *Procedia CIRP* 97 (2021), pp. 379–384.

[87] Dim P Papadopoulos, Jasper RR Uijlings, Frank Keller, and Vittorio Ferrari. "Training object class detectors with click supervision". In: *CVPR*. 2017.

[88] Deepak Pathak, Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully convolutional multi-class multiple instance learning". In: *preprint arXiv:1412.7144* (2014).

[89] Viorica Pătrăucean, Pierre Gurdjos, and Rafael Grompone Von Gioi. "A parameterless line segment and elliptical arc detector with enhanced ellipse fitting". In: *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part II 12*. Springer. 2012, pp. 572–585.

[90] Xingchao Peng, Ben Usman, Kuniaki Saito, Neela Kaushik, Judy Hoffman, and Kate Saenko. "Syn2real: A new benchmark forsynthetic-to-real visual domain adaptation". In: *arXiv preprint arXiv:1806.09755* (2018).

[91] Jason Pile, George B Wanna, and Nabil Simaan. "Force-based flexible path plans for robotic electrode insertion". In: *Proc. of the ICRA*. 2014, pp. 297–303.

[92] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems* 30 (2017).

[93] Weichao Qiu and Alan Yuille. "Unrealcv: Connecting computer vision to unreal engine". In: *Proc. of the ECCV*. 2016, pp. 909–916.

[94] Hiba Ramadan, Chaymae Lachqar, and Hamid Tairi. "A survey of recent interactive image segmentation methods". In: *Computational Visual Media* (2020), pp. 1–30.

[95] Matthias Rambow, Thomas Schauß, Martin Buss, and Sandra Hirche. "Autonomous manipulation of deformable objects based on teleoperated demonstrations". In: *IEEE Int. Conf. IROS*. 2012.

[96] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. "Using depth and appearance features for informed robot grasping of highly wrinkled clothes". In: *Proc. of the ICRA*. 2012, pp. 1703–1708.

[97] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. "Playing for data: Ground truth from computer games". In: *European conference on computer vision*. Springer. 2016, pp. 102–118.

[98] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "" GrabCut" interactive foreground extraction using iterated graph cuts". In: *ACM transactions on graphics (TOG)* 23.3 (2004), pp. 309–314.

[99] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey". In: *The International Journal of Robotics Research* 37.7 (2018), pp. 688–716.

[100] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. "Learning from synthetic data: Addressing domain shift for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3752–3761.

[101] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[102] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. "High-resolution stereo datasets with subpixel-accurate ground truth". In: *German conference on pattern recognition*. Springer. 2014.

[103] Daniel Scharstein and Richard Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". In: *International journal of computer vision* 47 (2002), pp. 7–42.

[104] Daniel Scharstein, Tatsunori Taniai, and Sudipta N Sinha. "Semi-global stereo matching with surface orientation priors". In: *2017 International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 215–224.

[105] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[106] Akihito Seki and Marc Pollefeys. "Sgm-nets: Semi-global matching with neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 231–240.

[107] Ankit Shah, Lotta Blumberg, and Julie Shah. "Planning for manipulation of interlinked deformable linear objects with applications to aircraft assembly". In: *IEEE Tran. on Automation Science and Engineering* 15.4 (2018), pp. 1823–1838.

[108] Konstantin Sofiiuk, Ilya A Petrov, and Anton Konushin. "Reviving iterative training with mask guidance for interactive segmentation". In: *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2022, pp. 3141–3145.

[109] Joes Staal, Michael D Abràmoff, Meindert Niemeijer, Max A Viergever, and Bram Van Ginneken. "Ridge-based vessel segmentation in color images of the retina". In: *IEEE transactions on medical imaging* 23.4 (2004), pp. 501–509.

[110] Baochen Sun, Jiashi Feng, and Kate Saenko. "Return of frustratingly easy domain adaptation". In: *AAAI*. 2016.

[111] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. "Revisiting unreasonable effectiveness of data in deep learning era". In: *ICCV*. 2017.

[112] Priya Sundaresan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph Gonzalez, and Ken Goldberg. "Learning Rope Manipulation Policies Using Dense Object Descriptors Trained on Synthetic Depth Data". In: *IEEE Int. Conf. ICRA* (2020).

[113] Te Tang, Changhao Wang, and Masayoshi Tomizuka. "A Framework for Manipulating Deformable Linear Objects by Coherent Point Drift". In: *IEEE Robotics and Automation Letters* (2018).

[114] Zhi Tian, Chunhua Shen, and Hao Chen. "Conditional convolutions for instance segmentation". In: *Proceedings of the Conf. ECCV*. Springer. 2020.

[115] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IROS*. IEEE, 2017.

[116] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30.

[117] Jonathan Tremblay, Thang To, and Stan Birchfield. "Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation". In: *CVPR*. IEEE, 2018.

[118] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. *Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects*. Number: arXiv:1809.10790 arXiv:1809.10790 [cs]. 2018.

[119] Jerome Trommnau, Jens Kühnle, Jörg Siegert, Robert Inderka, and Thomas Bauernhansl. "Overview of the state of the art in the production process of automotive wire harnesses, current research and future trends". In: *Procedia CIRP* (2019).

[120] Andrea Vedaldi and Stefano Soatto. "Quick shift and kernel methods for mode seeking". In: *Proc. of the ECCV*. 2008, pp. 705–718.

[121] Changhao Wang, Yuyou Zhang, Xiang Zhang, Zheng Wu, Xinghao Zhu, Shiyu Jin, Te Tang, and Masayoshi Tomizuka. "Offline-online learning of deformation model for cable manipulation with graph neural networks". In: *IEEE Robotics and Automation Letters* (2022).

[122] 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)*. IEEE. 2021, pp. 14199–14205.

[123] Yuchao Wang, Haochen Wang, Yujun Shen, Jingjing Fei, Wei Li, Guoqiang Jin, Liwei Wu, Rui Zhao, and Xinyi Le. "Semi-Supervised Semantic Segmentation Using Unreliable Pseudo-Labels". In: *CVPR*. 2022.

[124] Jun Wei, Shuhui Wang, and Qingming Huang. "F^3Net: fusion, feedback and focus for salient object detection". In: *Proceedings of AAAI-20*. 2020.

[125] Markus Wnuk, Chistoph Hinze, Manuel Zürn, Qizhen Pan, Armin Lechler, and Alexander Verl. "Tracking branched deformable linear objects with structure preserved registration by branch-wise probability modification". In: *2021 27th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. IEEE. 2021, pp. 101–108.

[126] Zhe Wu, Li Su, and Qingming Huang. "Cascaded partial decoder for fast and accurate salient object detection". In: *IEEE/CVF Conf. CVPR*. 2019.

[127] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[128] Mengyuan Yan, Yilin Zhu, Ning Jin, and Jeannette Bohg. "Self-supervised learning of state estimation for manipulating deformable linear objects". In: *IEEE robotics and automation letters* 5.2 (2020), pp. 2372–2379.

[129] Xue Yang and Junchi Yan. "Arbitrary-oriented object detection with circular smooth label". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*. Springer. 2020, pp. 677–694.

[130] Yuxuan Yang, Johannes A Stork, and Todor Stoyanov. "Learning differentiable dynamics models for shape control of deformable linear objects". In: *Robotics and Autonomous Systems* (2022).

[131] Yuxuan Yang, Johannes Andreas Stork, and Todor Stoyanov. "Tracking Branched Deformable Linear Objects Using Particle Filtering on Depth Images". In: *Available at SSRN 4531786* (2023).

[132] Hang Yin, Anastasia Varava, and Danica Kragic. "Modeling, learning, perception, and control methods for deformable object manipulation". In: *Science Robotics* 6.54 (2021), eabd8803.

[133] Jiaxuan You, Zhitao Ying, and Jure Leskovec. "Design space for graph neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17009–17021.

[134] Mingrui Yu, Kangchen Lv, Hanzhong Zhong, Shiji Song, and Xiang Li. "Global model learning for large deformation control of elastic deformable linear objects: An efficient and adaptive approach". In: *IEEE Trans. on Robotics* (2022).

[135] Francisco Yumbla, Meseret Abeyabas, Tuan Luong, June-Sup Yi, and Hyungpil Moon. "Preliminary connector recognition system based on image processing for wire harness assembly tasks". In: *2020 20th International Conference on Control, Automation and Systems (ICCAS)*. IEEE. 2020, pp. 1146–1150.

[136] Riccardo Zanella, Alessio Caporali, Kalyan Tadaka, Daniele De Gregorio, and Gianluca Palli. "Auto-generated Wires Dataset for Semantic Segmentation with Domain-Independence". In: *Proc. of ICCCR*. 2021.

[137] Jure Zbontar, Yann LeCun, et al. "Stereo matching by training a convolutional neural network to compare image patches." In: *J. Mach. Learn. Res.* 17.1 (2016), pp. 2287–2318.

[138] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 1386–1383.

[139] Hongshuang Zhang, Yu Zeng, Huchuan Lu, Lihe Zhang, Jianhua Li, and Jinqing Qi. "Learning to detect salient object with multi-source weak supervision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[140] Muhan Zhang and Yixin Chen. "Link prediction based on graph neural networks". In: *Advances in neural information processing systems* 31 (2018).

[141] Song Zhang. "High-speed 3D shape measurement with structured light methods: A review". In: *Optics and lasers in engineering* 106 (2018), pp. 119–131.

[142] Wenbo Zhang, Karl Schmeckpeper, Pratik Chaudhari, and Kostas Daniilidis. "Deformable linear object prediction using locally linear latent dynamics". In: *IEEE Int. Conf. ICRA*. 2021.

[143] Wenzeng Zhang, Xiande Ma, Leqin Cui, and Qiang Chen. "3 Points Calibration Method of Part Coordinates for Arc Welding Robot". In: *Intelligent Robotics and Applications*. Springer Berlin Heidelberg, 2008.

[144] Xinyi Zhang, Yukiyasu Domae, Weiwei Wan, and Kensuke Harada. "Learning efficient policies for picking entangled wire harnesses: An approach to industrial bin picking". In: *IEEE Robotics and Automation Letters* 8.1 (2022), pp. 73–80.

[145] Yiheng Zhang, Zhaofan Qiu, Ting Yao, Dong Liu, and Tao Mei. "Fully convolutional adaptation networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6810–6818.

[146] Jia-Xing Zhao, Jiang-Jiang Liu, Deng-Ping Fan, Yang Cao, Jufeng Yang, and Ming-Ming Cheng. "EGNet: Edge guidance network for salient object detection". In: *Proceedings of the IEEE/CVF Conf. ICCV*. 2019.

[147] Guoqing Zheng, Ahmed Hassan Awadallah, and Susan Dumais. "Meta label correction for learning with weak supervision". In: (2019).

[148] Zhi-Hua Zhou. "A brief introduction to weakly supervised learning". In: *National Science Review* 5.1 (2018), pp. 44–53.

[149] Hongyuan Zhu, Fanman Meng, Jianfei Cai, and Shijian Lu. "Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation". In: *Journal of Visual Communication and Image Representation* 34 (2016), pp. 12–27.

[150] Jihong Zhu, Andrea Cherubini, Claire Dune, David Navarro-Alarcon, Farshid Alambeigi, Dmitry Berenson, Fanny Ficuciello, Kensuke Harada, Jens Kober, Xiang Li, et al. "Challenges and outlook in robotic manipulation of deformable objects". In: *IEEE Robotics & Automation Magazine* 29.3 (2022), pp. 67–77.

[151] Jihong Zhu, Benjamin Navarro, Philippe Fraisse, André Crosnier, and Andrea Cherubini. "Dual-arm robotic manipulation of flexible cables". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 479–484.

[152] Manuel Zürn, Markus Wnuk, Armin Lechler, and Alexander Verl. "Topology matching of branched deformable linear objects". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 7097–7103.