Deformable One-Dimensional Object Detection for Routing and Manipulation

Azarakhsh Keipour, *Graduate Student Member, IEEE*, Maryam Bandari, and Stefan Schaal, *Fellow, IEEE*

***Abstract*—Many methods exist to model and track deformable one-dimensional objects (e.g., cables, ropes, and threads) across a stream of video frames. However, these methods depend on the existence of some initial conditions. To the best of our knowledge, the topic of detection methods that can extract those initial conditions in non-trivial situations has hardly been addressed. The lack of detection methods limits the use of the tracking methods in real-world applications and is a bottleneck for fully autonomous applications that work with these objects. This letter proposes an approach for detecting deformable one-dimensional objects which can handle crossings and occlusions. It can be used for tasks such as routing and manipulation and automatically provides the initialization required by the tracking methods. Our algorithm takes an image containing a deformable object and outputs a chain of fixed-length cylindrical segments connected with passive spherical joints. The chain follows the natural behavior of the deformable object and fills the gaps and occlusions in the original image. Our tests and experiments have shown that the method can correctly detect deformable one-dimensional objects in various complex conditions.**

***Index Terms*—Computer vision for automation, computer vision for medical robotics, deformable object detection, object detection, perception for grasping and manipulation, segmentation and categorization.**

I. Introduction

MANIPULATING non-rigid objects using robots has long been the subject of research in various contexts [1]. A specific class of non-rigid objects is called Deformable One-dimensional Objects (DOOs) or Deformable Linear Objects (DLOs) and includes objects such as ropes, cables, threads, sutures, and wires.

In order to achieve full autonomy in applications working with DOOs, one of the essential and challenging parts is perception. Many applications require complete knowledge of the object's initial conditions, even in the occluded parts. Moreover, routing and manipulation applications also require a representation model of the DOO that allows simulation and the computation of its dynamics.

There are many methods proposed in the medical imaging community that can find the DOOs (known as tubular structures in that context) in the image frame [2]–[6]. While these methods are perfected for low signal-to-noise images, and some can even work with self-crossings, they mainly only provide the region in the image containing the DOO and are considered segmentation methods.

Various algorithms have been proposed to track a DOO across the video frames, even in the presence of occlusions and self-crossings. These methods may devise different tools such as registration methods (e.g., Coherent Point Drift [7]), learning, dynamics models, and simulation to predict and correct the prediction across the consecutive frames [8]–[15]. While some of these methods can initialize the DOO in the first frame using trivial conditions (e.g., a straight rope in camera view), the other methods require even a simple DOO configuration to be provided to them a priori. Most tracking methods will fail to initialize in even slightly complex initial conditions.

The initial conditions are not generally provided in real-world applications, and pure segmentation is insufficient. For example, an aerial manipulator working with electrical wires at the top of a utility pole needs to detect the desired wire, including its occluded parts and crossings, and requires a model of the detected wire to perform any task on it.

The authors of [16] proposed a method that trains on rendered images and fine-tunes on real images to detect and track the state of a rope. The method requires tens of thousands of rendered and real images of the same rope for training and needs re-training and tuning for a new DOO. This approach may be acceptable for an industrial application focusing on DOOs of the exact same characteristics, but it may not be practical for many other applications.

This letter proposes a method to detect the initial conditions of a deformable one-dimensional object. The method fills the occluded parts and works with self-crossings. The output is a single object represented as a discretized model useful for routing, manipulation, and simulation. The detected object can be used to initialize other existing tracking methods to be used in applications such as manipulating DOOs into desired shapes and knots [17]–[20].

Sections II and III define the problem and present the proposed method for detection of DOOs; Section IV describes our implementation and shows the experiments and results. Finally,

Manuscript received September 9, 2021; accepted January 10, 2022. Date of publication January 31, 2022; date of current version February 24, 2022. This letter was recommended for publication by Associate Editor J. Borras and Editor C. C. Lerma upon evaluation of the reviewers' comments. This work was supported by X, The Moonshot Factory residency program. *(Corresponding author: Azarakhsh Keipour.)*

Azarakhsh Keipour is with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: keipour@cmu.edu; keipour@gmail.com).

Maryam Bandari and Stefan Schaal are with Google, Mountain View, CA 94040 USA (e-mail: maryamb@google.com; sschaal@google.com).

Digital Object Identifier 10.1109/LRA.2022.3146920

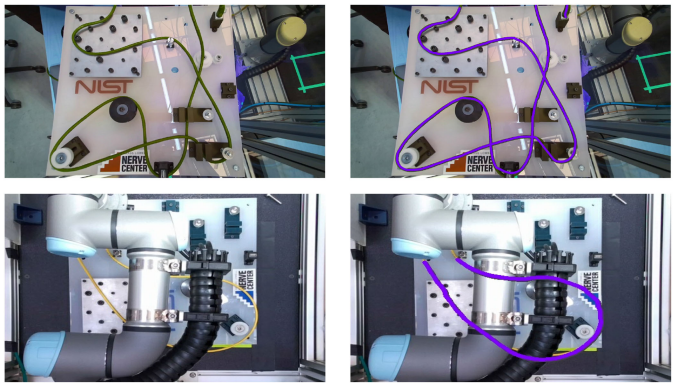


Fig. 1. The result of the proposed detection method on example inputs with occlusions and crossings. The detected cable (purple) overlaid on the frames on the right.

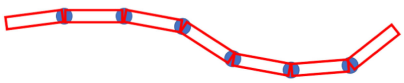


Fig. 2. The representation of a DOO as a chain of fixed-length cylinders connected by spherical joints.

Section V discusses how the proposed method can be further improved in the future.

II. Problem Definition

This work addresses the detection of cable-like deformable shapes. Unlike rigid objects, the shape of deformable objects can change, and some form of a flexible model is required to represent the current state of their shape. On the other hand, to predict the reaction of the deformable objects to the applied forces and moments, the representation model should facilitate the integration of a dynamics model.

In theory, a DOO represented by its pixels (or voxels) in the camera frame can be integrated with a dynamics model. However, the model would typically require finite element analysis and is computationally expensive, making it impractical for robotics applications. Simpler representations are commonly used in tasks such as manipulation, routing, and planning. Arriola-Rios *et al.* [1] provide an overview of the common representations for deformable objects.

A commonly-used approach for representing DOOs is to model them as a chain of fixed-length cylindrical segments connected by spherical joints. This simple model can easily integrate with an efficient dynamics model to simulate or predict the object's behavior. While the chosen model does not affect the ideas described in the proposed method, our method utilizes this model. In our application, the length of each segment is represented by l_s, and there is no gap between the segments (i.e., each segment starts precisely where the neighbor segment ends). Fig. 2 illustrates the fixed-length cylinder chain model used in this work.

The focus of this work is to provide the cylinder chain representation of a deformable one-dimensional object seen in the camera frame (which can be RGB or RGB-D/3-D). The output chain model should predict and fill the path taken by the DOO under the occlusions and return a single chain object. Fig. 1 shows this objective.

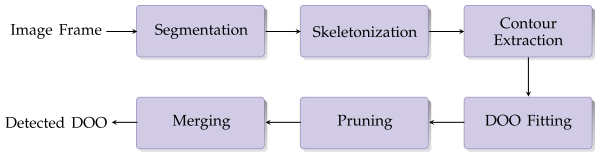


Fig. 3. The high-level overview of the proposed method for detection of deformable one-dimensional objects.

Algorithm 1: Deformable one-dimensional object detection.

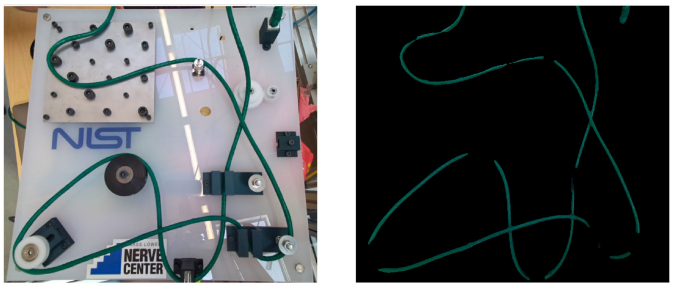
▷ `Detects and extract DOOs from the input image frame.`
function DETECTDOO(frame)
  ▷ `Segment the image to extract the DOO region`
  $segmented\_img \leftarrow$ SEGMENT$(frame)$
  ▷ `Extract the skeletons of the segmented regions`
  $thinned\_img \leftarrow$ SKELETONIZE$(segmented_img)$
  ▷ `Extract the contours from the skeletons`
  $contours \leftarrow$ EXTRACTCONTOURS$(thinned_img)$
  ▷ `Fit DOO chains to all contours`
  $Chains \leftarrow \emptyset$
  for each $c \in contours$ **do**
    $fitted\_chains \leftarrow$ FITDOO(c)
    $Chains$.insert$(fitted_chains)$
  end for
  ▷ `Prune all the overlapping segments`
  $Chains \leftarrow$ PRUNE$(Chains)$
  ▷ `Merge the DOO chains into a single DOO`
  while $Chains.length > 1$ **do**
    $C_1, C_2 \leftarrow$ FINDBESTMERGEMATCH$(Chains)$
    $C_{merged} \leftarrow$ MERGECHAINS(C_1, C_2)
    $Chains$.remove$(\{C_1, C_2\})$
    $Chains$.insert(C_{merged})
  end while
  return $Chains[0]$ ▷ `Return the final DOO chain`
end function

III. Proposed Method

The DOO detection method in this work takes the camera frame and performs a sequence of processing steps to output a single object in chain representation (described in Section II). Fig. 3 shows a high-level overview of the method.

The first three steps in the proposed method are well-known processes. Our algorithm can work with different segmentation, skeletonization, and contour extraction approaches. The choice depends on the task at hand. The last three processing steps are the contributions of our algorithm.

Algorithm 1 shows the pseudo-code of our approach. This section describes each of those steps in more detail.



(a) (b)

Fig. 4. The segmentation of a DOO in a camera frame. (a) The original image. (b) The segmentation result.

(a) (b)

Fig. 5. (a) The skeletonization of a segmented deformable one-dimensional object. (b) The contours extracted from the skeleton. Each contour is drawn with a different color.

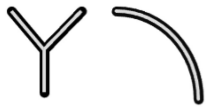


Fig. 6. Contour types extracted from a skeletonized image. The black area around the gray skeleton is the contour.

A. Segmentation

A vital step in extracting the complete DOO from the input camera image is segmentation. This step aims to filter the image data to extract the DOO portions and exclude all other data.

The simplest segmentation methods include color-based filtering and background subtraction, which can work in lab settings, but more complex methods are required for real-world applications.

The medical research community provides an extensive set of segmentation methods to address vein and vessel detection, which can be used here directly or with small modifications. Such methods include both model-based [2]–[5], [21], [22] and learning-based [6], [23] approaches and are often robust to clutter in the input image and can work in low signal-to-noise conditions.

The requirement for the segmentation method for our work is to filter the DOO data conservatively, i.e., ideally, it should eliminate all the unrelated data even if it removes some of the DOO data. Fig. 4 illustrates the segmentation of an example cable in the camera frame.

B. Topological Skeletonization

Skeletonization transforms each segmented connected component into a set of connected pixels with single-pixel width called a *skeleton*. It is commonly used in the pre-processing stage of various applications ranging from Optical Character Recognition (OCR) to human motion tracking, fingerprint analysis, and various medical imaging analysis [24], [25].

Our algorithm has two requirements for choosing the skeletonization method: a) the skeleton of a connected component should remain a connected component; b) only one branch should be returned per actual branch (i.e., multi-branching of a single skeleton branch should be avoided). The skeletonization algorithm chosen for this step should inherently respect the two constraints. Fig. 5(a) shows the skeletonization of the segmented example of Fig. 4.

C. Contour Extraction

A contour (a.k.a., boundary) is an ordered sequence of the pixels around a shape. Extracting contours from an image is utilized in many applications ranging from shape analysis to semantic segmentation and image classification [26], [27].

Ideally, the contour extraction method applied to the skeletons should result in one contour per skeleton branch. However, in practice, the contour extraction methods can result in several contours per branch and some contours containing multiple branches. Moreover, a contour contains the closed boundary *around* the skeleton and not the actual skeleton pixels. Fig. 6 shows different types of contours that can be extracted from a skeleton piece.

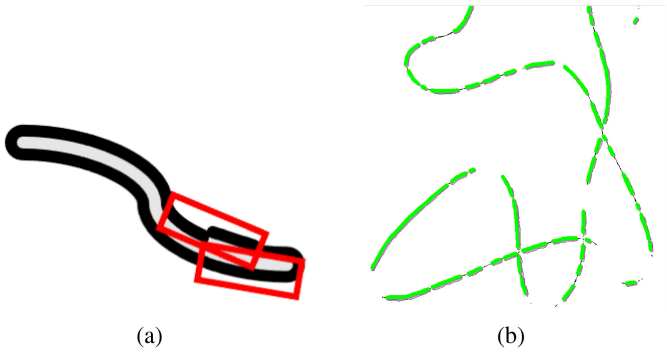
Our DOO detection method can handle the above-mentioned common issues raised from the contour extraction methods. Therefore, many of the existing contour extraction methods can be used with our algorithm regardless of their output limitations. Fig. 5(b) presents the result of contour extraction.

Contours facilitate traversing points along the skeleton and simplify understanding the connections in the branches. If an ordered set of pixels for each skeleton branch is obtained from the skeletonization method or other means, the contour extraction step can be skipped.

D. Fitting DOO Segments

The next step is to fit a chain of fixed-length segments to each contour. A contour can be a single branch, or it may contain multiple branches (see Fig. 5(b)). The pixel sequence for a contour returned by a typical contour extraction method starts from one of the tips and ends with a sharp turn back at the start point.

Let us call the latest added segment as s, the current segment as s', the first point in the contour sequence as the starting point p_s of s', and the point currently being traversed as p_c. Let us also define $\vec{s}$ as the vector in the direction of segment s, starting at its start point and pointing towards its end point. Therefore, starting with an empty DOO chain, the points are traversed while the distance $\|p_c - p_s\|_2$ is less than l_s. Then a new segment of length l_s is added to the DOO chain from point p_s in the



(a) (b)

Fig. 7. (a) An illustration of overlapping segments around a skeletonized deformable one-dimensional object. (b) The result of segment fitting from contours and pruning for a deformable one-dimensional object of Fig. 5(a).

direction of p_c, the new segment's end-point p_e is saved as the next segment's start point p_s, and the traversal continues.

Three conditions may happen during the traversal:

1) Vector $\vec{s'}$ is close to vector $\vec{s}$: The new segment is added to the current DOO chain in this case.
2) Vector $\vec{s'}$ is close to vector $-\vec{s}$: It means that the traversal has gone over a branch end. In this case, the current DOO is recorded without the new segment, and the new segment is discarded. The traversal continues from the branch tip with a new empty DOO chain.
3) The last point in the contour sequence is reached: it means that the traverse has returned to the start point, and the traverse can be terminated. There may be cases where the whole contour length is less than the segment size. In these cases, no new DOO chains will be generated.

Algorithm 2 shows the pseudo-code for the described steps.

Algorithm 2: Traversing contours for DOO chain creation.

▷ This function traverses a contour and returns all DOO chains created from the contour.
function TRAVERSECONTOUR(*contour*)
  ▷ Initialize the collection of DOO chains
  $Collection \leftarrow \emptyset$
  ▷ Initialize the start point and the next DOO chain
  $p_s \leftarrow contours[0], \quad chain \leftarrow \emptyset$
  ▷ Initialize the tip point
  $p_t \leftarrow p_s, \quad update\_tip \leftarrow True$
  ▷ Traverse over all the points in the contour
  for each $p_c \in contour$ **do**
    ▷ Create a new segment if p_c is far enough
    if DIST$(p_c, p_s) \geq l_s$ **then**
      $p_e \leftarrow p_s + l_s \times (p_c - p_s)/\|p_c - p_s\|$
      $s' \leftarrow$ CREATESEGMENT(p_s, p_e)
      if $chain = \emptyset$ or ANGLE(s, s') is small **then**
        ▷ Add the segment if it is the first in chain
        ▷ or if the direction has not changed much
        $chain$.Insert(s')
        $s \leftarrow s', \quad p_s \leftarrow p_e, \quad p_t \leftarrow p_e$
      else
        ▷ Start a new chain if direction has changed
        $Collection$.Insert$(chain)$
        $chain \leftarrow \emptyset, \quad p_s \leftarrow p_t$
      end if
      ▷ Start updating tip point
      $update\_tip \leftarrow True$
    else if DIST$(p_c, p_s) \geq$ DIST(p_t, p_s) **then**
      ▷ Update the tip point
      if $update\_tip = True$ **then** $p_t \leftarrow p_c$
    else
      ▷ Stop updating tip point
      $update\_tip \leftarrow False$
    end if
  end for
  ▷ Add the last generated chain
  $Collection$.Insert$(chain)$
  return $Collection$ ▷ Return all the chains at the end
end function

E. Pruning

The segment fitting algorithm of Section III-D returns multiple overlapping DOO chains for each part of the object. It is desired to prune the overlapping segments to reduce the total number of segments and simplify the further steps by assuming that no two segments overlap.

To define the overlap of two segments, each segment can be assumed as a rotated rectangle in the 2-D case and a square cuboid (a cuboid with two square faces) for the 3-D case. The length of the rectangle and cuboid is the segment length l_s, and the rectangle's width is 3 pixels or higher. The reasoning for the choice of the width is that the width of the skeleton is generally 1 pixel with occasional width of 2 pixels (depends on the choice of the thinning method). The contour is the boundary around the skeleton, and for the two rectangles on the two sides of the skeleton to overlap, they need to be at least 3 pixels wide. In practice, any small number greater than 3 pixels should work well for pruning the overlapping segments. For the 3-D case, the width of the cuboid can be chosen similarly, i.e., it should be at least the total of the width of the contour layer and the maximum width of the skeleton. Once the segments are defined, the geometric intersection of two rotated rectangles or cuboids is used to find the overlap. Fig. 7(a) shows how two segments can overlap for the same skeleton.

A heuristic that has shown performance improvements in the subsequent stages removes the segment from the shorter chain when two segments overlap. This heuristic will result in many chains being quickly emptied, which reduces the overall number of DOO chains in the collection.

Fig. 7(b) shows the chains resulting from segment fitting and then pruning of the contours in Fig. 5(b).

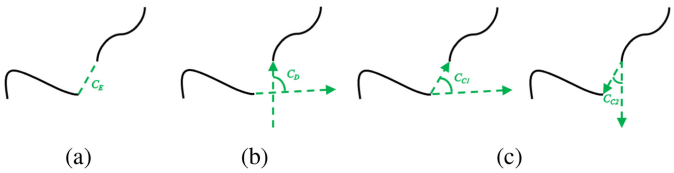


Fig. 8. Illustration of different partial costs of merging two chain ends. (a) Euclidean cost. (b) Direction cost. (c) Curvature costs from first chain to the second and from the second chain to the first.

F. Merging

Once we have a collection of DOO chains, they should be merged to fill the gaps and form a single object. A gap can result from an occlusion or an imperfect segmentation and should be filled in a way that follows the natural curve of the deformable object.

The merging process can be performed iteratively, connecting two chains at a time until all the chains are merged into a single deformable one-dimensional object. Each iteration can be broken down into two steps:

1) Choose the best two chains for merging
2) Connect the selected chains

The following subsections describe choosing the best chains and properly connecting them with their natural bend.

G. Merging: Choosing the Best Matches

To choose the best chains to connect, we define a new cost function $C_M(\cdot)$ that calculates the cost of connecting any two chain ends. Considering that there are two ends for each chain, there will be four cost values for connecting the two ends for any two chains. The lowest among the four values is the cost of connecting the two chains.

Given an end segment s_1 of the first chain and an end segment s_2 of the second chain, three separate partial costs are defined and then combined to create the total cost function $C_M(\cdot)$:

- Euclidean Cost C_E: This measure incurs costs to two chain ends based on their Euclidean distance to deter the early connection of far away ends (Fig. 8(a)). Having the end segments s_1 and s_2, this cost can be calculated as:

$$C_E\left(s_1, s_2\right) = \|s_1.end - s_2.end\|_2, \tag{1}$$

  where $\|\cdot\|_2$ is the norm of the resulting vector and $s.end$ is the end point of the segment (end point of the chain).
- Direction Cost C_D: This measure incurs costs to two chain ends based on the difference of the direction of the first with the opposite direction of the second (Fig. 8(b)). This cost discourages the connection of chains that are not facing each other. Having the end segments s_1 and s_2, this cost can be calculated as:

$$C_D\left(s_1, s_2\right) = \left|\arccos\left(\frac{-\vec{s_1}\cdot\vec{s_2}}{\|s_1\|\|s_2\|}\right)\right|, \tag{2}$$

  where $|\cdot|$ is the absolute value, $\|\cdot\|$ is the norm of the vector (size of the segment), $\vec{s}$ is the vector along the segment s starting from the start of the segment and ending at the end of the segment (i.e., the end of the chain), and $\cdot$ is the inner product operator.
- Curvature Cost C_C: This measure incurs costs to two chain ends based on how much curvature is needed to connect them. It is calculated as the higher cost between bending the first chain's end segment towards the second chain's end segment and vice versa (Fig. 8(c)). The measure is defined to discourage connections requiring excessive bending and to encourage smooth connections. Having the end segments s_1 and s_2, this cost can be calculated as:

$$\begin{aligned} C_{C1}\left(s_1, s_2\right) &= \left|\arccos\left(\frac{\vec{s_1}\cdot\vec{s_{21}}}{\|s_1\|\|s_{21}\|}\right)\right| \\ C_{C2}\left(s_1, s_2\right) &= \left|\arccos\left(\frac{\vec{s_2}\cdot\vec{s_{12}}}{\|s_2\|\|s_{12}\|}\right)\right| \\ C_C\left(s_1, s_2\right) &= \max\left(C_{C1}, C_{C2}\right), \end{aligned} \tag{3}$$

  where s_{nm} is a shorthand for $s_n.end - s_m.end$.

Having the three cost values for the ends of two chains, the total cost of these ends is computed as:

$$\begin{aligned} &C_M(s_1, s_2) \\ &\quad = \mathscr{F}\left(C_E\left(s_1, s_2\right), C_D\left(s_1, s_2\right), C_C\left(s_1, s_2\right)\right), \end{aligned} \tag{4}$$

where $\mathscr{F}$ is the function combining the three values. In practice, we learned that the weighted sum of the values works well, and even after manually choosing a simple weight set, the algorithm works for almost all kinds of situations (see Section IV for our test values). With the weighted sum, the Equation 4 reduces to:

$$\begin{aligned} &C_M(s_1, s_2) \\ &\quad = w_e \cdot C_E\left(s_1, s_2\right) + w_d \cdot C_D\left(s_1, s_2\right) + w_c \cdot C_C\left(s_1, s_2\right) \end{aligned} \tag{5}$$

After calculating the four costs of all end combinations of the two chains, the minimum of those costs is the cost of merging the two DOO chains. Once the costs for all pairs of chains are calculated, the two chains with the lowest total merging cost are chosen for merging.

Note that the choice for the cost function of Eq. 4 is to encourage the connection of closer chains that align well and can connect smoothly. Choosing a single measure such as minimum curvature would result in unwanted connections of farther chains that align perfectly over closer chains that are slightly misaligned.

H. Merging: Connecting Two Chains

Once two chains C_1 and C_2 are selected for connection (see Section III-G), the gap between the two chains should be filled with a new chain C_{new} in a way that it follows the expected curve of the deformable object. Our experiments show that any deformable object can take almost any curve given different pressure points, forces, tensions, and the object's condition. However, it is possible to have an educated *guess* on how the object behaves. For this purpose, we calculate the "natural" curvature required for the new chain C_{new}, which connects the desired end of C_1 to the desired end of C_2.

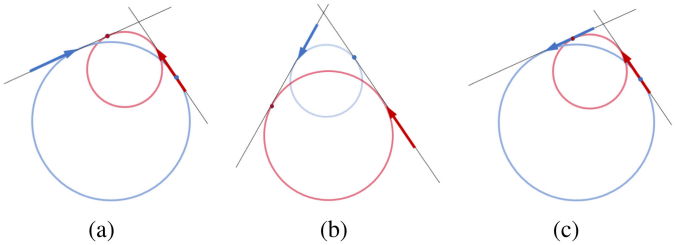


Fig. 9. Illustration of different merging scenarios with the two circles tangent to the line passing the end points of the two chains, each circle passing through one of the end points. Arrow ends and directions represent the end points and end directions of the chains. (a) Exactly one of the circles passing through the end point of a chain is touching the other line ahead of the other chain. (b) Both the circles passing through the end points of the chains are touching the other line ahead of the other chain. (c) None of the circles passing through the end point of a chain are touching the other line ahead of the other chain.

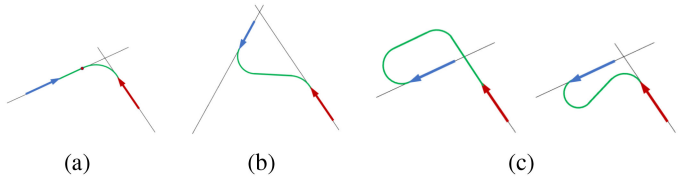


Fig. 10. Suggested solutions for different merging cases illustrated in Fig. 9.

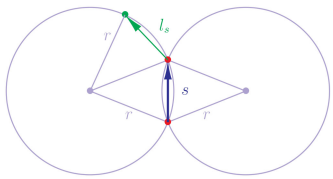


Fig. 11. Adding a new DOO segment with length l_s with a desired constant turn radius of r to the end of segment s.

To compute the "natural" curvature, we assume that the new chain C_{new} starts in the same direction as the two desired chain ends. In other words, at each end, C_{new} initially follows the direction of the last segment of the chain to which it is connected. On the other hand, we assume that when it is possible, the deformable one-dimensional object will follow a curve with a constant turn rate (i.e., constant radius). With these assumptions, we can find two circles tangent to the lines passing through the two chain ends, each passing through one of the chain end-points. Based on triangle similarity theorems, the radii of the two circles are proportional to the distances of the chain ends from the intersection point. Fig. 9 illustrates this idea.

Let us define the end-points we desire to connect on chains C_1 and C_2 as e_1 and e_2, respectively. We call the lines passing through e_1 and e_2 in the direction of C_1 and C_2 ends as l_1 and l_2, respectively. Finally, we define the circles passing through e_1 and e_2 as c_1 and c_2, and the points they touch on the other line as t_1 and t_2, respectively. Note that points e_1 and t_2 will be lying on line l_1, while points e_2 and t_1 are on line l_2.

Without loss of generality, let us assume that in Fig. 9, circle c_1 is the red circle, the blue circle is c_2, the red dot is t_1, the blue dot is t_2, the red arrow's end point (arrow side) is e_1, the blue arrow's end point is e_2, the line passing e_1 is l_1 and the line passing e_2 is l_2.

It can be proven that the distance between e_2 and t_1 is equal to the distance between e_1 and t_2. However, each t_1 and t_2 can be lying on lines l_2 and l_1 ahead or behind e_2 and e_1, creating three different situations:

- Either t_1 is ahead of e_2 or t_2 is ahead of e_1, but not both (Fig. 9(a)). Not surprisingly, a majority of connections in a typical application would be of this type. In this case the blue circle c_2 that touches l_1 at point t_2 behind e_1 is discarded and we use the radius of the red circle c_1 for the turn radius of the new chain C_{new}. This chain will be composed of the arc of c_1 from e_1 to t_1 and the line from t_1 to e_2. Fig. 10(a) shows this scenario's solution.
- Both t_1 and t_2 are ahead of e_2 and e_1 (Fig. 9(a)). In this case, the new chain C_{new} is composed of an arc on each end (e_1 and e_2) and a line tangent to the arcs. The turn radius is as desired or can be experimentally determined for the DOO, and it should be large enough to allow the "natural-looking turn." However, the radius should be small enough so that the direction of the line tangent to the two arcs is close to the direction of the line connecting e_1 to e_2. Finally, we suggest the same turn radius for both ends. Fig. 10(b) shows this scenario's solution.
- Both t_1 and t_2 are behind of e_2 and e_1 (Fig. 9(c)). This case has two suggested solutions that depend on the conditions. In both solutions, similar to the previous case, the new chain C_{new} is composed of an arc on each end (e_1 and e_2) and a line tangent to the arcs. The turn radius is as desired or can be experimentally determined for the DOO. However, depending on external conditions, C_{new} can fill the gap from outside or inside the region between the two chains. Fig. 10(c) shows this scenario's solutions.

Note that, in all scenarios, there can be infinite correct solutions, which depend on the conditions. In practice, the suggested solutions result in good fits with the ground truth and can be used in most conditions without any modification.

Once the new chain C_{new} is obtained to fill the gap between the two chain ends C_1 and C_2, it can be added to the ends of the chains to connect them. C_{new} is a combination of constant-radius arcs and lines. Adding a DOO segment for the line sections is trivial and will not be explained. To add a segment that follows the desired turn, we use the last segment s on the chain. Knowing the start and end-points of this segment s, we can calculate two circles with the desired radius that pass through these points. Knowing the direction of the turn, one circle is eliminated, and the new point on the remaining circle at the segment distance l_s of the segment's end-point is used to create the new segment s' that is added to the end of the chain. Fig. 11 shows how the new segment can be added with the desired turn radius.

I. Notes on the Proposed Method

The merging process continues connecting the chains, two at a time, until all the chains are merged into one single chain, which is the detected DOO represented by the chain of fixed-length cylindrical segments connected by passive spherical joints. This

representation can be used as the input to routing and manipulation systems for the desired application.

Each segmented image will be processed into a single DOO. When there are multiple DOOs present in the camera frame, the easiest way to detect them as separate DOOs is to have separate segmentation for them. For example, if there are multiple cables with different colors in the image, a color-based segmentation can have two segmented images. In case there are multiple DOOs that cannot be segmented separately, the proposed algorithm can still help process them into separate DOO outputs. Note that the algorithm is greedy in the sense that it first goes for the best-matched chains. With multiple DOOs, there is a high chance that the chains related to separate DOOs do not give a good fit. As a result, for example, when there are only two chains left, there is a high chance that the two chains are the two separate DOOs. Therefore, it is enough to stop the merging process when the desired number of chains is left.

The proposed method is general and can be used with both 2-D and 3-D image data to provide the deformable object's representation in 2-D or 3-D.

Finally, the final output of the proposed method is a single chain of segments that does not keep track of the parts seen in the frame vs. the occluded parts. There are two ways to mark the parts of the chain that are related to the occlusions:

1) Map the segmented parts on the final detection to determine the occluded parts of the DOO chain.
2) During the process, when merging two chains, if the gap is equal or longer than the segment length l_s, the newly added chain C_{new} is marked as occluded. The reason for skipping smaller gaps is that many gaps shorter than l_s are created during the pruning process.

Both approaches ultimately depend on the accuracy of the segmentation. The first approach is simple but may mislabel an occluded part as visible in a multilayer setup. On the other hand, the second approach tends to be more accurate in multilayer settings but may skip more minor occlusions.

IV. Experiments and Results

The proposed was implemented for 2-D images in Python 3. We used the color-based segmentation of the DOO region. This approach generally tends to include extra areas around the DOO and other regions with similar color hues to the DOO. We chose conservative thresholds to exclude any non-DOO regions. This results in some DOO data being excluded; However, our experiments have shown the DOO detection to have challenges when extra regions are included but to work when some data is lost in segmentation. The same principle is advised for other segmentation methods choices, and those methods' parameters should be chosen conservatively to remove the irrelevant regions.

We used a well-known morphological thinning method for skeletonization [28]. The algorithm proposed by Suzuki and Abe [29] and provided in the OpenCV library is used to extract contours. All our tests use $w_e = 1$, $w_d = 100$ and $w_c = 100$ values for the cost function of Equation 5 and the segment length l_s is chosen as 10 pixels. The weights are chosen manually to focus on shorter distances while heavily discouraging non-matching segment directions and excessive bending. Different weights result in some types of incorrect connections increasing while the number of other types decreases. A better set can be found using a more methodical approach and optimization for the desired applications. Similarly, the segment length was not chosen optimally. In general, a shorter segment length can capture the contour ends better and create segments from smaller contours, leading to better curves in filling gaps; however, it increases the number of segments and the average number of incorrect connections. On the other hand, a longer segment length has the potential of not following the curves well and ignoring small contours. However, it tends to reduce the number of incorrect connections and improve the detection speed by reducing the number of segments.

TABLE I
Detection Results on 7 Video Sequences

	Total	Correct	Incorrect	Accuracy
Frames	4,230	3,542	688	83.7%
Occlusions	26,456	23,991	2,465	90.7%
Merges	583,743	581,130	2,613	99.6%

We have used the method for cable routing and manipulation tasks [30]. The viability is tested on 7 video sequences with a total of 4,230 frames of size 1280 $\times$ 720. Table I shows the quantitative results for the algorithm's accuracy on the whole cable in an image, for the occlusions filled, and for the merges performed. Mengyuan *et al.* [16] have used the root mean square of the Euclidean distance between their estimated and the ground-truth point positions on the DOO, which they reported as around 23 mm. Note that due to the lack of ground truth for the occluded areas and to focus on testing the key contributions of our proposed approach, we used a stricter measure that even when a single connection is incorrect, we counted the frame as incorrect detection. The occlusions are counted as incorrect when either a wrong connection is made or when the filled connection does not follow the actual cable's path. Finally, we noticed that the incorrect merges only rarely happen in places other than at occlusions, with only 148 cases, almost all of which happened at self-crossings.

Our method's average detection time per frame across all the sequences is 0.537 seconds on a system with Intel Core i9-10885H CPU and 64 GB DDR4 RAM. Figs. 1 and 12 show snapshots of some video sequences and the detection results.

V. Conclusions and Future Work

We presented a novel method for detecting deformable one-dimensional objects (e.g., ropes and cables) and showed the results. Our implementation is only 2-D, not tuned towards a specific condition, and is not optimally coded. Choices other than the weighted sum for the total cost function were not researched, and our selection of weights was not made optimally. Nevertheless, the results show promise with an almost 2 Hz detection rate on an HD image input.

The method is flexible and can be tuned for specific 2-D and 3-D applications to provide near-perfect results. On the other hand,

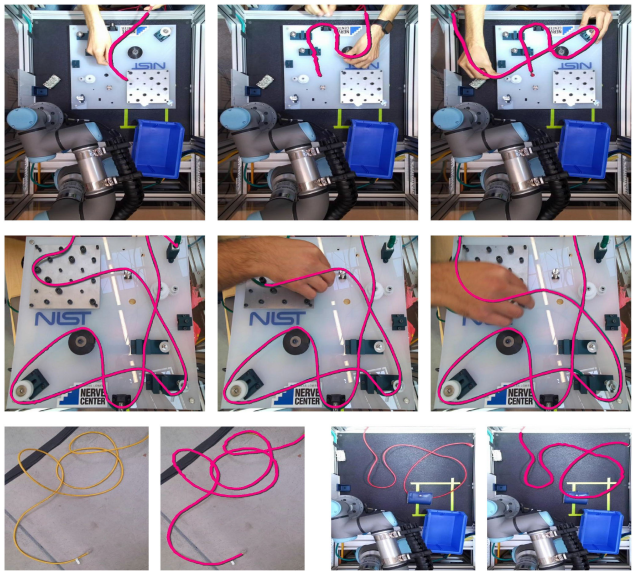


Fig. 12. Screenshots of example sequences with the overlaid detected cable (in magenta). The third row includes the original frame for comparison.

a more optimized implementation can take advantage of special data structures and parallelization to increase the method's speed by several orders of magnitude.

Note that it is not hard to find unstructured or adversarial situations with entanglements, occlusions, multiple close and parallel DOOs, and other complex scenarios that can easily confuse the proposed algorithm. This work aimed to provide a method that can assist in semi-structured situations rather than addressing those "crazy" scenarios.

The considerations for the 3-D case are provided for each step. However, we did not implement the 3-D case, and there may be unpredicted implementation challenges. In the future, the ideas of the method can be integrated with tracking methods to improve tracking accuracy. Its integration in a robotics pipeline can finally enable full autonomy in real-world robotics applications working with DOOs such as cables, surgical sutures, and ropes.

ACKNOWLEDGMENT

The authors would like to thank Olivier Pauly for great insights into approaching the problem from the medical imaging research point of view.

REFERENCES

[1] V. Arriola-Rios, P. Guler, F. Ficuciello, D. Kragic, B. Siciliano, and J. Wyatt, "Modeling of deformable objects for robotic manipulation: A tutorial and review," *Front. Robot. AI*, vol. 7, p. 82, 2020.

[2] K. Krissian, G. Malandain, N. Ayache, R. Vaillant, and Y. Trousset, "Model-based detection of tubular structures in 3D images," *Comput. Vis. Image Understanding*, vol. 80, no. 2, pp. 130–171, 2000.

[3] T. Hachaj and M. R. Ogiela, "Segmentation and visualization of tubular structures in computed tomography angiography," in *Proc. Intell. Inf. Database Syst.*, 2012, pp. 495–503.

[4] B. Wang *et al.*, "A robust and efficient framework for tubular structure segmentation in chest CT images," *Technol. Health Care*, vol. 29, pp. 655–665, 2021.

[5] J. H. Noble and B. M. Dawant, "A new approach for tubular structure modeling and segmentation using graph-based techniques," in *Proc. Med. Image Comput. Comput.-Assist. Interv.*, vol. 14, no. Pt 3, 2011, pp. 305–312.

[6] C. Wang *et al.*, "Tubular structure segmentation using spatial fully connected network with radial distance loss for 3D medical images," in *Proc. Med. Image Comput. Comput. Assist. Intervention*, 2019, pp. 348–356.

[7] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 12, pp. 2262–2275, Dec. 2010.

[8] S. Javdani, S. Tandon, J. Tang, J. F. O'Brien, and P. Abbeel, "Modeling and perception of deformable one-dimensional objects," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 1607–1614.

[9] O. Pauly, H. Heibel, and N. Navab, "A machine learning approach for deformable guide-wire tracking in fluoroscopic sequences," in *Proc. Med. Image Comput. Comput.- Assist. Interv.*, 2010, pp. 343–350.

[10] Y. Wang, D. McConachie, and D. Berenson, "Tracking partially-occluded deformable objects while enforcing geometric constraints," in *Proc. Int. Conf. Robot. Automat.*, 2021, pp. 1–7.

[11] A. Rastegarpanah, R. Howard, and R. Stolkin, "Tracking linear deformable objects using slicing method," *Robotica*, pp. 1–19, 2021, doi: 10.1017/S0263574721001065.

[12] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 1130–1137.

[13] Y. Lai, J. Poon, G. Paul, H. Han, and T. Matsubara, "Probabilistic pose estimation of deformable linear objects," in *Proc. Int. Conf. Automat. Sci. Eng.*, 2018, pp. 471–476.

[14] N. Padoy and G. Hager, "Deformable tracking of textured curvilinear objects," in *Proc. 23rd Brit. Mach. Vis. Conf.*, 2012, pp. 1–11.

[15] T. Tang, Y. Fan, H.-C. Lin, and M. Tomizuka, "State estimation for deformable objects by point registration and dynamic simulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 2427–2433.

[16] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2372–2379, Apr. 2020.

[17] X. Li, Z. Wang, and Y.-H. Liu, "Sequential robotic manipulation for active shape control of deformable linear objects," in *Proc. IEEE Int. Conf. Real-Time Comput. Robot.*, 2019, pp. 840–845.

[18] T. Tang, C. Wang, and M. Tomizuka, "A framework for manipulating deformable linear objects by coherent point drift," *IEEE Robot. Automat. Lett.*, vol. 3, no. 4, pp. 3426–3433, Oct. 2018.

[19] H. Wakamatsu, E. Arai, and S. Hirai, "Knotting/unknotting manipulation of deformable linear objects," *Int. J. Robot. Res.*, vol. 25, no. 4, pp. 371–395, 2006.

[20] M. Saha and P. Isto, "Manipulation planning for deformable linear objects," *IEEE Trans. Robot.*, vol. 23, no. 6, pp. 1141–1150, Dec. 2007.

[21] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever, "Multiscale vessel enhancement filtering," in *Proc. Med. Image Comput. Comput.- Assist. Interv.*, 1998, pp. 130–137.

[22] O. Merveille, H. Talbot, L. Najman, and N. Passat, "Tubular structure filtering by ranking orientation responses of path operators," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 203–218.

[23] Y. Wang *et al.*, "Deep distance transform for tubular structure segmentation in CT scans," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3832–3841.

[24] P. K. Saha, G. Borgefors, and G. Sanniti di Baja, "Chapter 1 - skeletonization and its applications — A review," in *Skeletonization:Theory, Methods and Applications, Academic Press*, 2017, pp. 3–42.

[25] A. Keipour, M. Eshghi, S. M. Ghadikolaei, N. Mohammadi, and S. Ensafi, "Omnifont persian OCR system using primitives," 2013, *arXiv:2202.06371*.

[26] X.-Y. Gong, H. Su, D. Xu, Z.-T. Zhang, F. Shen, and H.-B. Yang, "An overview of contour detection approaches," *Int. J. Automat. Comput.*, vol. 15, no. 6, pp. 656–672, Dec. 2018.

[27] A. Keipour, G. A. Pereira, and S. Scherer, "Real-time ellipse detection for robotics applications," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 7009–7016, Oct. 2021.

[28] P. Maragos and R. Schafer, "Morphological skeleton representation and coding of binary images," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 5, pp. 1228–1244, Oct. 1986.

[29] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Comput. Vis., Graph., Image Process.*, vol. 30, no. 1, pp. 32–46, 1985.

[30] A. Keipour, M. Bandari, and S. Schaal, "Efficient spatial representation and routing of deformable one-dimensional objects for manipulation," 2022, *arXiv:2202.06172*.