Robotics and Autonomous Systems 142 (2021) 103798

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Vision-based manipulation of deformable and rigid objects using subspace projections of 2D contours

Jihong Zhu [a,b,c,*], David Navarro-Alarcon [d], Robin Passama [a], Andrea Cherubini [a]

[a] LIRMM - Université de Montpellier CNRS, 161 Rue Ada, 34090 Montpellier, France
[b] Department of Cognitive Robotics, Delft University of Technology, Netherlands
[c] Honda Research Institute, Europe, Germany
[d] The Hong Kong Polytechnic University, Department of Mechanical Engineering, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 3 August 2020
Received in revised form 21 March 2021
Accepted 21 April 2021
Available online 10 May 2021

Keywords:
Visual servoing
Sensor-based control
Deformable object manipulation

ABSTRACT

This paper proposes a unified vision-based manipulation framework using image contours of deformable/rigid objects. Instead of explicitly defining the features by geometries or functions, the robot automatically learns the visual features from processed vision data. Our method simultaneously generates – from the same data – both visual features and the interaction matrix that relates them to the robot control inputs. Extraction of the feature vector and control commands is done online and adaptively, and requires little data for initialization. Our method allows the robot to manipulate an object without knowing whether it is rigid or deformable. To validate our approach, we conduct numerical simulations and experiments with both deformable and rigid objects.

1. Introduction

Humans are capable of manipulating both rigid and deformable objects. However, robotic researchers tend to consider the manipulation of these two classes of objects as separate problems. Unless otherwise mentioned, object rigidity is an implicit assumption in most manipulation tasks. On the other hand, methods designed for deformable object manipulation [1], are never applied on rigid objects. This paper presents our efforts in formulating a generalized framework for vision-based manipulation of both rigid and deformable objects, which does not require prior knowledge of the object's mechanical properties.

In the visual servoing literature [2], vector $\boldsymbol{s}$ denotes the set of features selected to represent the object in the image. These features represent both the object's pose and its shape. We denote the process of selecting $\boldsymbol{s}$ as *parameterization*. The aim of visual servoing is to minimize, through robot motion, the feedback error $\boldsymbol{e} = \boldsymbol{s}^* - \boldsymbol{s}$ between the target $\boldsymbol{s}^*$ and the current (i.e., measured) feature $\boldsymbol{s}$.

One of the initial works on vision-based manipulation of deformable objects is presented in [3] to solve a knotting problem by a topological model. Smith et al. developed a relative elasticity model, such that vision can be utilized without a physical model for the manipulation task [4]. A classical model-free approach in manipulating deformable objects is developed in [5]. More recent research [6,7] proposes a method for online estimation of the deformation Jacobian, based on weighted least square minimization with a sliding window. In [8,9], the vision-based deformable objects manipulation is termed as *shape servoing*. An expository paper on the topic is available in [10]. A recent work on vision-based shape servoing of plastic material was presented in [11].

For a detailed survey on shape servoing we refer readers to [1]. For *shape servoing*, commonly selected features are curvatures [8], points [12] and angles [13]. Laranjeira et al. proposed a catenary-based feature for tethered management on wheeled and underwater robots [14,15]. A more general feature vector is that containing the Fourier coefficients of the object contour [9,16]. Yet, all these approaches require the user to specify a model, e.g., the object geometry [8,12,13] or a function [9,14,16] for selecting the feature. Alternative data-driven (hence, model-free) approaches rely on machine learning. Nair et al. combine learning and visual feedback to manipulate ropes in [17]. Li et al. approximate the deformation and camera model using a neural network [18]. The authors of [19] employ deep neural networks to manipulate deformable objects given their 3D point cloud. All these methods rely on (deep) connectionist models, which invariably require training through an extensive data set. The collected data has to be diverse enough to generalize the model learnt by this type of networks. Instead of relying on algorithmic solutions, [20] utilizes a vision-based tactile sensor (GelSight) for manipulating cables.

* Corresponding author at: LIRMM - Université de Montpellier CNRS, 161 Rue Ada, 34090 Montpellier, France.
E-mail addresses: jihong.zhu@lirmm.fr (J. Zhu), david.navarro-alarcon@polyu.edu.hk (D. Navarro-Alarcon), robin.passama@lirmm.fr (R. Passama), andrea.cherubini@lirmm.fr (A. Cherubini).

https://doi.org/10.1016/j.robot.2021.103798
0921-8890/

It is noteworthy that some of the above mentioned methods may apply to rigid objects. Yet, none of the previous works has investigated the possibility of this extension nor reported its experimental validation, as we do in this paper.

The trend in visual servoing, when *controlling the pose of rigid objects* is to find features which are independent from the object characteristics. Following this trend, [21] proposes the use of image moments. More recently, researchers have proposed direct visual servoing (DVS) methods, which eliminate the need for user-defined features and for the related image processing procedures. The pioneer DVS works [22,23] propose using the whole image luminance to control the robot, leading to "photometric" visual servoing. Bakthavatchalam et al. join the two ideas by introducing photometric moments [24]. A subspace method [25] can further enhance the convergence of photometric visual servoing, via Principal Component Analysis (PCA). This method was first introduced for visual servoing in [26]. In that work, using an eye-in-hand setup, the image was compressed to obtain a low-dimensional vector for controlling the robot to a target pose. Similarly, the authors of [27] transformed the image into a lower dimensional hyper surface, to control the robot position via in-hand camera feedback. However, DVS generally considers rigid and static scenes, where the robot controls the motion of the camera (eye-in-hand setup) to change only the image viewpoint, and not the environment. These constraints on the setup avoid breaking the Lambertian hypothesis that is needed, since DVS relies on the raw image luminance, which should not vary with the viewpoint. For this reason, to our knowledge, DVS was never applied to object manipulation, since changes in the pose and/or shape of the object would break the Lambertian assumption. This is not the case of feature-based methods (such as the one we present here), as long as the feature is chosen to be reliable even when the viewpoint and/or scene change.

Compared with the above-mentioned works, our paper presents the following original contributions:

1. We propose to use a feature vector – based on PCA of sampled 2D contours – for model-free manipulation of both deformable and rigid objects.
2. We exploit the linear properties of PCA and of the local interaction matrix, to initialize our algorithm with little data — the same data for feature vector extraction and for interaction matrix estimation.
3. We report experiments using the same framework to manipulate objects with different unknown geometric and mechanical properties.

The paper is organized as follows. Section 2 presents the problem. Section 3 outlines the framework. Section 4 elaborates on the methods. In Section 5, we analyze and verify the methods by numerical simulations. Then, Section 6 presents the robotic experiments and we conclude in Section 7.

2. Problem statement

In this work, we aim at solving object manipulation tasks with visual feedback. We rely on the following hypotheses:

- The shape and pose of the object are represented by its 2-D contour on the image as seen from a camera fixed in the robot workspace (eye-to-hand setup). We denote this contour as

$$\boldsymbol{c} = [\boldsymbol{p}_1 \ \cdots \ \boldsymbol{p}_K]^T \in \mathbb{R}^{2K}, \tag{1}$$

  where $\boldsymbol{p}_j = [u_j \ v_j] \in \mathbb{I}$ denotes the jth pixel of the contour in the image $\mathbb{I}$.
- The contour is always entirely visible in the scene and there are no occlusions.
- One of the robot's end-effectors holds one point of the object (we consider the grasping problem to be already solved). At each control iteration i, its pose is $\boldsymbol{r}_i \in \mathbb{SE}(3)$, and it can execute motion commands $\delta\boldsymbol{r}_i \in \mathbb{SE}(3)$ that drive the robot so that $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i + \delta\boldsymbol{r}_i$.
- The target constant shape (i.e., contour) of the object, $\boldsymbol{c}^*$, is physically reachable with shaping motions of the grasping point $\boldsymbol{r}$. To ensure this hypothesis, one can first command the robot to verify that it can move the shape to $\boldsymbol{c}^*$.

Problem Statement. Given a target shape of the object, represented by a constant contour vector $\boldsymbol{c}^*$, we aim at designing a vision-based controller that generates a sequence of robot motions $\delta\boldsymbol{r}_i$ to drive the initial contour to the target one.

The controller should work without any knowledge of the object physical characteristics, i.e., for both rigid and deformable objects. In the latter case, we assume that the deformation is homogeneous. Since rigid and deformable objects behave differently during manipulation, we set the following manipulation goals:

- Rigid objects: move them to a target pose (see Fig. 1(a)).
- Deformable objects: move them to a target pose with a target shape (see Fig. 1(b)).

The formulation of the problem is general, but due to challenges in perception (discussed in Section 7), we carried out the cases of study with movements in $\mathbb{SE}(2)$.

3. Preliminary

In this section, we present an overview of the proposed approach, motivated by the problem analysis. Throughout the paper, we use $\boldsymbol{c}$ to indicate the *object contour* and $\boldsymbol{s}$ as the *feature vector* obtained from the contour. The subscript i indicates the instance of the variable at iteration i (e.g., $\boldsymbol{c}_i$ is the contour at iteration i).

We can work directly on the object shape space by selecting the contour as the feature vector $\boldsymbol{s} \equiv \boldsymbol{c} \in \mathbb{R}^{2K}$. With image and data processing, we can extract a fixed number of ordered (i.e., identified) contour points to represent the shape/pose of the object. However, this will result in an unnecessarily large dimension of the feature vector (e.g., if $K = 50$, $\boldsymbol{s}$ has 100 components). The high dimensional feature vector increases the computation demand and complicates the control due to the high under-actuation of the system. Therefore, instead of working on this feature vector, we work on one with smaller dimensions. To this end, we split the problem into two sub-problems: *parameterization* and *control*, see Fig. 2.

Parameterization consists in representing the contour via a compact feature vector $\boldsymbol{s} \in \mathbb{R}^k$, such that $k \ll 2K$. We denote this representation as $\boldsymbol{s} = \boldsymbol{g}(\boldsymbol{c})$. We introduce the method for parameterization in Section 4.1.

Control consists in computing robot motions $\delta\boldsymbol{r}_1, \delta\boldsymbol{r}_2, \ldots$, so that the object's representation $\boldsymbol{s}$ converges to the target $\boldsymbol{s}^*$. *Control* can be broken down to solving the optimization problem:

$$\boldsymbol{r}^* = \arg\min_{\boldsymbol{r}} (\boldsymbol{f}(\boldsymbol{r}) - \boldsymbol{s}^*) \tag{2}$$

where $\boldsymbol{s} = \boldsymbol{f}(\boldsymbol{r})$ denotes the mapping between robot pose and feature vector, which is assumed to be smooth and generally nonlinear. The smoothness assumption requires that the objects' contour is at least twice differentiable with respect to the robot motion. If we know the analytic solution to $\boldsymbol{f}(\boldsymbol{r})$, we can solve (2) and obtain the target shape in a single iteration by commanding $\boldsymbol{r}^*$.

A solution to this problem is to approximate the full mapping $\boldsymbol{f}(\boldsymbol{r})$ from sensor observations. Classic deep learning-based

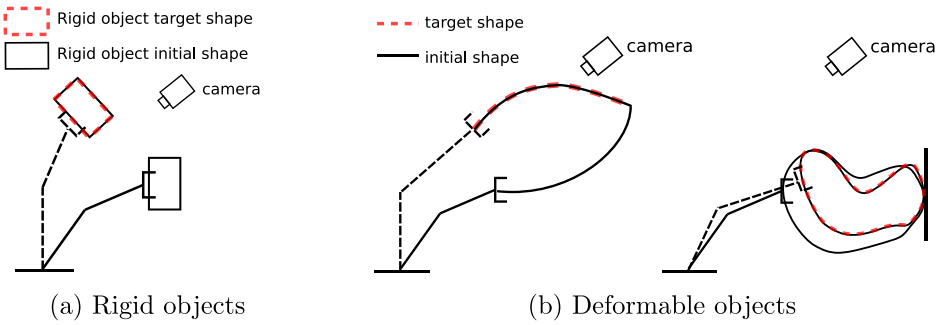


Fig. 1. Vision-based manipulation of rigid and deformable objects. For rigid objects (left): control pose (translation and rotation). For deformable objects (right): control the pose, and also shape.

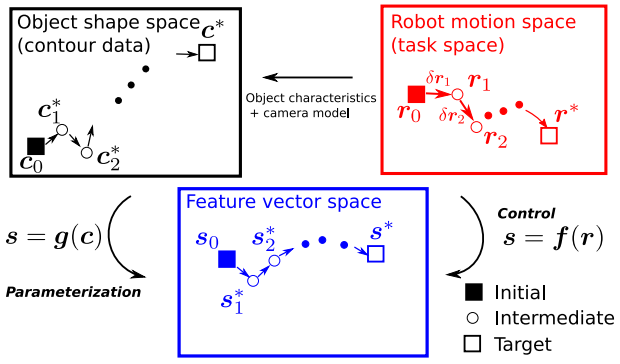


Fig. 2. Graphic representation of the vision-based manipulation problem, with its two sub-problems, *parameterization* and *control*.

approaches typically require a long training phase to collect vast and diverse data for approximating $\boldsymbol{f}(\boldsymbol{r})$. In some cases (for instance, robotics surgery), it is not possible to collect such data beforehand. Moreover, if the object changes, new data has to be collected to retrain the model, leading to a cumbersome process. In this paper instead, we aim at doing the data collection online, with minimum initialization.

Thus, instead of estimating the full nonlinear mapping $\boldsymbol{f}(\boldsymbol{r})$, we divide it into piece-wise linear models [28] at successive equilibrium points. The locality assumption refers to both the time and spatial dimensions. These models are considered time invariant in the neighborhood of the equilibrium points. We then compute the control law for each linear model and apply it to the robot end-effector. We will dedicate Sections 4.2 and 4.3 to the local models and Sections 4.4 and 4.5 to derive the control inputs and to analyze (local) stability.

4. Methodology

Given a target shape $\boldsymbol{c}^*$, we define an intermediate local target $\boldsymbol{c}_i^*$ at each $i = 1, 2, \ldots$ (see Fig. 2). At the ith iteration, the robot autonomously generates a local mapping $\boldsymbol{g}_i$ to produce the feature vector $\boldsymbol{s}_i = \boldsymbol{g}_i(\boldsymbol{c}_i)$. The robot then finds the local mapping $\boldsymbol{s}_i = \boldsymbol{f}_i(\boldsymbol{r}_i)$ online.

Consider at the current time instant i, the shape $\boldsymbol{c}_i$, the intermediate target $\boldsymbol{c}_i^*$ and the local parameterization $\boldsymbol{g}_i$. We can transform shape data into a feature vector by:

$$\boldsymbol{s}_i = \boldsymbol{g}_i(\boldsymbol{c}_i), \ \ \boldsymbol{s}_i^* = \boldsymbol{g}_i(\boldsymbol{c}_i^*). \tag{3}$$

The linearized version of $\boldsymbol{s} = \boldsymbol{f}(\boldsymbol{r})$ centered at $(\boldsymbol{s}_i, \boldsymbol{r}_i)$ is then:

$$\delta \boldsymbol{s}_i = \boldsymbol{L}_i \delta \boldsymbol{r}_i, \tag{4}$$

with

$$\begin{aligned} \boldsymbol{L}_i &= \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{r}} \, |_{\boldsymbol{r}=\boldsymbol{r}_i}, \\ \delta \boldsymbol{s}_i &= \boldsymbol{s}_{i+1} - \boldsymbol{s}_i, \\ \delta \boldsymbol{r}_i &= \boldsymbol{r}_{i+1} - \boldsymbol{r}_i. \end{aligned} \tag{5}$$

The matrix $\boldsymbol{L}_i$ represents a local mapping, referred to as the interaction matrix in the visual servoing literature [2]. If $\boldsymbol{L}_i$ can be estimated online at each iteration i, then, we can design one-step control laws to drive $\boldsymbol{s}_i$ towards $\boldsymbol{s}_i^*$.

After the robot has executed the motion command $\delta \boldsymbol{r}_i$, we update the next target to be $\boldsymbol{s}_{i+1}^*$, and so on, until it reaches the final target $\boldsymbol{s}^*$. Although the validity region of this local mapping is smaller than that of the original nonlinear mapping, it enables to use an online training approach that requires less data and reduced computational demand.

Fig. 3 shows the building blocks of the overall framework. In this section, we focus on the red dashed part of the diagram. We will elaborate on each red block in the subsequent subsections. The blue block represents the image processing pipeline that will be discussed in Section 6.1.

4.1. Feature vector extraction

There are many ways to parameterize $\boldsymbol{c}$ in order to reduce its dimension. One of the prominent dimension reduction methods is Principal Component Analysis (PCA). PCA finds a new orthogonal basis for high-dimensional data. This enables projection of the data to lower dimension with the minimal sum of squared residuals. It was used in image processing [29] and classification [30]. In visual servoing, the method was first introduced in [26]. PCA is proven to be an effective, yet easy to implement, algorithm for dimension reduction. By projecting to the new orthogonal space, each feature component is linearly independent. Besides,

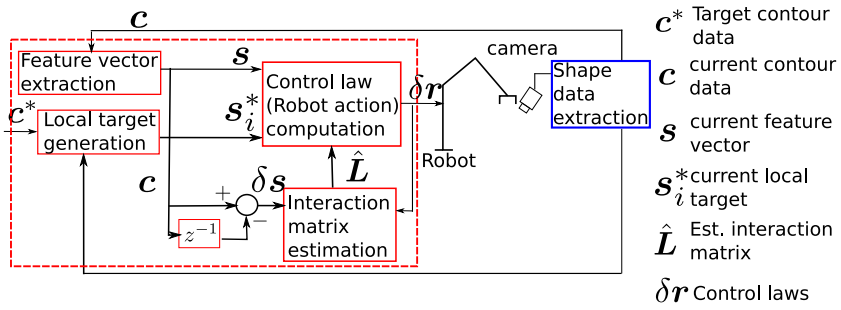


Fig. 3. The block diagram that represents the overall framework.

by checking the explained variance of a feature, we can intuitively measure if it represents the original shape.

We apply PCA to reduce $\boldsymbol{c} \in \mathbb{R}^{2K}$ to $\boldsymbol{s} \in \mathbb{R}^{k}$. To find the projection, we collect M images with different shapes of the object and construct the data matrix $\boldsymbol{\Gamma} = [\boldsymbol{c}_1\ \boldsymbol{c}_2\ \cdots\ \boldsymbol{c}_M] \in \mathbb{R}^{2K\times M}$. Then, we shift the columns of $\boldsymbol{\Gamma}$ by the mean contour $\bar{\boldsymbol{c}} = \sum_{i=1}^{M} \boldsymbol{c}_i/M$:

$$\bar{\boldsymbol{\Gamma}} = [\boldsymbol{c}_1 - \bar{\boldsymbol{c}}\quad \boldsymbol{c}_2 - \bar{\boldsymbol{c}}\quad \cdots\ \boldsymbol{c}_M - \bar{\boldsymbol{c}}] \in \mathbb{R}^{2K\times M}. \tag{6}$$

We then compute the covariance matrix $\boldsymbol{C} = \bar{\boldsymbol{\Gamma}}\bar{\boldsymbol{\Gamma}}^T$, and apply Singular Value Decomposition (SVD) to it:

$$\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T. \tag{7}$$

Once we have obtained the eigenvector matrix $\boldsymbol{U} \in \mathbb{R}^{2K\times 2K}$, we can move on to select the first k columns[1] of $\boldsymbol{U}$ denoted by $\boldsymbol{U}(k) \in \mathbb{R}^{2K\times k}$. Then, the $2K$-dimensional contour $\boldsymbol{c}$ can be projected into a smaller k-dimensional feature vector $\boldsymbol{s}$ as:

$$\boldsymbol{s} = \boldsymbol{U}^T(k)(\boldsymbol{c} - \bar{\boldsymbol{c}}) \in \mathbb{R}^{k}. \tag{8}$$

To assess the quality of this projection, we can compute the *explained variance* using the eigenvalue matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{2K\times 2K}$ in (7). By denoting the diagonal entries of $\boldsymbol{\Sigma}$ as $\sigma_1, \ldots, \sigma_{2K}$, the explained variance of the first k components is:

$$\Upsilon(k) = \frac{\sum_{j=1}^{k}\sigma_j}{\sum_{j=1}^{2K}\sigma_j}. \tag{9}$$

where Υ is a scalar between 0 and 1 (since $\sigma_j > 0, \forall j$), indicating to what extent the k components represent the original data (a larger Υ suggests a better representation).

Since PCA calculate features that lie on an orthonormal basis, these features are linearly independent. For controlling n DoF, at least the same number of independent visual features should be used. Therefore, we set $k = n$ features.

4.2. Local target generation

Let us now explain how we generate a local target contour $\boldsymbol{c}_i^*$ given a current contour $\boldsymbol{c}_i$ and final target contour $\boldsymbol{c}^*$. We also show this in Algorithm 1. The overall shape error is given by:

$$\boldsymbol{c_e} = \boldsymbol{c}^* - \boldsymbol{c}_i. \tag{10}$$

We define the intermediate target contour as:

$$\boldsymbol{c}_i^* = \boldsymbol{c}_i + \frac{1}{\eta}\boldsymbol{c_e}, \tag{11}$$

with $\eta = 1, 2, \ldots$ an integer that ensures that $\boldsymbol{c}_i^*$ is a "good" local target for $\boldsymbol{c}_i$ (i.e., the two are similar). Therefore, if we project the intermediate local data using the eigenvector matrix at the current iteration, $\boldsymbol{U}_i \in \mathbb{R}^{2K\times 2K}$ (note that we are using the full projection matrix and not just the first k columns), the projection $\boldsymbol{s}_i^p = \boldsymbol{U}_i(\boldsymbol{c}_i^* - \bar{\boldsymbol{c}}) \in \mathbb{R}^{2K}$ should fulfill:

$$\Psi\ (k) = \frac{\sum_{j=1}^{k}\left|s_{i,j}^p\right|}{\sum_{j=1}^{2K}\left|s_{i,j}^p\right|} \geq \epsilon, \tag{12}$$

with $\epsilon \in [0; 1]$ a threshold and $s_{i,j}^p$ the jth component of the projection. Then, we select the first k components in $\boldsymbol{s}_i^p$ to be the local target $\boldsymbol{s}_i^* \in \mathbb{R}^{k}$.

Algorithm 1 outlines the steps for computing the local intermediate targets, so that:

- they are near the final target,
- the corresponding feature vector can be extracted with the current learned projection matrix.

Remark 1. The reachability of a local target can only be verified with a global deformation model which we want to avoid identifying in our methods. We will further discuss this issue in the Conclusion (Section 7).

Algorithm 1 Local target generation.

```
localTargetFound = false
Ψ_0 = 0
η = 1
while not localTargetFound do
    c_i^* = c_i + (1/η)(c^* − c_i)
    s_i^p = U_i c_i^*
    Ψ_η = Σ_{j=1}^{k} |s_{i,j}^p| / Σ_{j=1}^{2K} |s_{i,j}^p|
    if Ψ_η ≥ ε or Ψ_η < Ψ_{η−1} then
        localTargetFound = true
        s_i^* = [I 0] s_i^p
    end if
    η = η + 1
end while
```

4.3. Interaction matrix estimation

Let us consider the current contour $\boldsymbol{c}_i$ and the local target $\boldsymbol{c}_i^*$. In this section, we show how we can implement the PCA and model estimation together and online. We denote the robot motions and corresponding object contours over the last M iterations (prior to iteration i, with $i \geq M$) as:

$$\begin{aligned}\boldsymbol{\Delta R}_i &= \left[\delta\boldsymbol{r}_{i-M+1}\ \delta\boldsymbol{r}_{i-M+2}\cdots\delta\boldsymbol{r}_i\right] \in \mathbb{R}^{n\times M}\\ \boldsymbol{\Gamma}_i &= \left[\boldsymbol{c}_{i-M}\ \boldsymbol{c}_{i-M+1}\ \boldsymbol{c}_{i-M+2}\cdots\boldsymbol{c}_i\right] \in \mathbb{R}^{2K\times(M+1)},\end{aligned} \tag{13}$$

with M the number of data samples collected during initialization, i.e., the size of the sliding window used for model adaptation (see Section 4.5).

[1] In the SVD algorithm, the first k columns correspond to the k largest eigenvalues of matrix Σ.

By selecting $k = n$ (note that n is also the number of DoFs of the robot manipulator we considered in the task execution), we compute the projection matrix $\boldsymbol{U}_i(n) \in \mathbb{R}^{2K\times n}$, from $\boldsymbol{\Gamma}_i$ and $\bar{\boldsymbol{c}}_i$ via (6) and (7). Then, using $\boldsymbol{U}_i(n)$, we project current contour $\boldsymbol{c}_i$, target contour $\boldsymbol{c}_i^*$ and shape matrix $\boldsymbol{\Gamma}_i$:

$$\begin{aligned}
&\boldsymbol{s}_i = \boldsymbol{U}_i(n)^T(\boldsymbol{c}_i - \bar{\boldsymbol{c}}_i) \in \mathbb{R}^n,\\
&\boldsymbol{s}_i^* = \boldsymbol{U}_i(n)^T(\boldsymbol{c}_i^* - \bar{\boldsymbol{c}}_i) \in \mathbb{R}^n,\\
&\boldsymbol{S}_i = \boldsymbol{U}_i(n)^T\bar{\boldsymbol{\Gamma}}_i = \begin{bmatrix}\boldsymbol{s}_{i-M} & \boldsymbol{s}_{i-M+1} & \cdots & \boldsymbol{s}_i\end{bmatrix} \in \mathbb{R}^{n\times(M+1)}.
\end{aligned} \tag{14}$$

In (14), $\bar{\boldsymbol{\Gamma}}_i$ is normalized by $\bar{\boldsymbol{c}}_i$ as in (6). We can then compute $\boldsymbol{\Delta S}_i$ from (5) and (14), by subtracting consecutive columns of $\boldsymbol{S}_i$:

$$\boldsymbol{\Delta S}_i = \begin{bmatrix}\delta\boldsymbol{s}_{i-M+1} & \delta\boldsymbol{s}_{i-M+2} & \cdots & \delta\boldsymbol{s}_i\end{bmatrix} \in \mathbb{R}^{n\times M}. \tag{15}$$

Using $\boldsymbol{\Delta S}_i \in \mathbb{R}^{n\times M}$ and $\boldsymbol{\Delta R}_i \in \mathbb{R}^{n\times M}$ we can now estimate the local interaction matrix $\boldsymbol{L}_i \in \mathbb{R}^{n\times n}$ at iteration i. We assume that near this iteration, the system remains linear and time invariant: $\boldsymbol{L}_i$ is constant. Using the local linear model (4), we can write the following:

$$\boldsymbol{\Delta S}_i = \boldsymbol{L}_i\boldsymbol{\Delta R}_i. \tag{16}$$

Our goal then is to solve for $\boldsymbol{L}_i$, given $\boldsymbol{\Delta S}_i$ and $\boldsymbol{\Delta R}_i$. Note that this is an overdetermined linear system (with $n \times M$ equations for n^2 unknowns). Let us consider $\boldsymbol{\Delta R}_i \in \mathbb{R}^{n\times M}$ has full row rank. Note this sufficiently implies $M \geq n$. With this prerequisite, $\text{rank}(\boldsymbol{\Delta R}_i) = n$. Therefore, $\text{rank}(\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T) = n$, and its inverse exists. We post multiply (16) by $\Delta\boldsymbol{R}_i^T$:

$$\boldsymbol{\Delta S}_i\boldsymbol{R}_i^T = \boldsymbol{L}_i\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T. \tag{17}$$

Then, since $\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T$ is invertible, the $\boldsymbol{L}_i$ that best fulfills (16) is:

$$\hat{\boldsymbol{L}}_i = \boldsymbol{\Delta S}_i\boldsymbol{\Delta R}_i^T(\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T)^{-1}. \tag{18}$$

If, in practice, the full row rank condition of $\boldsymbol{\Delta R}_i$ is not satisfied, $\text{rank}(\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T) < n$ and $\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T$ becomes singular. Then, instead of (18), we can use Tikhonov regularization:

$$\hat{\boldsymbol{L}}_i = \boldsymbol{\Delta S}_i\boldsymbol{\Delta R}_i^T(\boldsymbol{\Delta R}_i\boldsymbol{\Delta R}_i^T + \lambda\boldsymbol{I})^{-1}, \tag{19}$$

with λ an arbitrary (generally small) scalar.

Practically, this implies that one or more inputs motions do not appear in $\boldsymbol{\Delta R}_i$. Therefore, we cannot infer the relationship between these motions and the resulting feature vector changes. In this case it is better to increase M and obtain more data, so that $\boldsymbol{\Delta R}_i$ has full row rank.

Instead of computing the interaction matrix, it is also possible to directly compute its inverse, since this guarantees better control properties [31]. With the same data, one can re-write (16) as:

$$\boldsymbol{L}_i^{\oplus}\boldsymbol{\Delta S}_i = \boldsymbol{\Delta R}_i. \tag{20}$$

We can also solve (20) with Tikhonov regularization:

$$\hat{\boldsymbol{L}}_i^{\oplus} = \boldsymbol{\Delta R}_i\boldsymbol{\Delta S}_i^T(\boldsymbol{\Delta S}_i\boldsymbol{\Delta S}_i^T + \lambda\boldsymbol{I})^{-1}. \tag{21}$$

4.4. Control law and stability analysis

We can now control the robot, with either of the following strategies:

$$\delta\boldsymbol{r}_i = -\alpha\hat{\boldsymbol{L}}_i^{\dagger}(\boldsymbol{s}_i - \boldsymbol{s}_i^*), \tag{22}$$

if one estimates the interaction matrix with (19), where † denotes the pseudo-inverse, or:

$$\delta\boldsymbol{r}_i = -\alpha\hat{\boldsymbol{L}}_i^{\oplus}(\boldsymbol{s}_i - \boldsymbol{s}_i^*) \tag{23}$$

if one estimates the inverse of the interaction matrix with (21). In both equations, $\alpha > 0$ is an arbitrary control gain.

Proposition 1. *Consider that locally, the model (4) closely approximates the interaction matrix $\boldsymbol{L}_i = \hat{\boldsymbol{L}}_i$. For M number of linearly independent displacement vectors $\delta\boldsymbol{r}$ such that the interaction matrix $\hat{\boldsymbol{L}}_i$ is invertible, the update rule (22) asymptotically minimizes the error $\boldsymbol{e}_i = \boldsymbol{s}_i^* - \boldsymbol{s}_i$, where $\boldsymbol{s}_i^*$ is the local target.*

Proof. With $\delta\boldsymbol{s}_i = \boldsymbol{s}_{i+1} - \boldsymbol{s}_i$, we can write (4) in discretized form as

$$\boldsymbol{s}_{i+1} = \boldsymbol{s}_i + \boldsymbol{L}_i\delta\boldsymbol{r}_i. \tag{24}$$

From the definition of $\boldsymbol{e}_i$ we have (Note here the target $\boldsymbol{s}_i^*$ is not updated with i since we want to prove local convergence to a constant target):

$$\begin{aligned}
\boldsymbol{e}_i &= \boldsymbol{s}_i^* - \boldsymbol{s}_i\\
\boldsymbol{e}_{i+1} &= \boldsymbol{s}_i^* - \boldsymbol{s}_{i+1}
\end{aligned} \tag{25}$$

Taking (24) into (25):

$$\begin{aligned}
\boldsymbol{e}_{i+1} &= \boldsymbol{s}_i^* - \boldsymbol{s}_{i+1}\\
&= \boldsymbol{s}_i^* - \boldsymbol{s}_i - \boldsymbol{L}_i\delta\boldsymbol{r}_i\\
&= \boldsymbol{e}_i - \boldsymbol{L}_i\delta\boldsymbol{r}_i.
\end{aligned} \tag{26}$$

We replace $\delta\boldsymbol{r}_i$ in (26) with the control (22), the error dynamic is then:

$$\begin{aligned}
\boldsymbol{e}_{i+1} &= \boldsymbol{e}_i - \alpha\boldsymbol{L}_i\boldsymbol{L}_i^{-1}(\boldsymbol{s}_i^* - \boldsymbol{s}_i)\\
&= \boldsymbol{e}_i - \alpha\boldsymbol{e}_i = (1-\alpha)\boldsymbol{e}_i.
\end{aligned} \tag{27}$$

is asymptotically stable for $\alpha \in [0; 1]$. This can be proved by considering the Lyapunov function

$$\mathcal{V}(\boldsymbol{e}) = \boldsymbol{e}^T\boldsymbol{e}. \tag{28}$$

Using the error dynamic (27), one can derive:

$$\begin{aligned}
\Delta\mathcal{V} &= \mathcal{V}(\boldsymbol{e}_{i+1}) - \mathcal{V}(\boldsymbol{e}_i)\\
&= \boldsymbol{e}_i^T((1-\alpha)^2 - 1)\boldsymbol{e}_i < 0.
\end{aligned} \tag{29}$$

This proves the local asymptotic stability of the error $\boldsymbol{e}$ using our inputs. ■

4.5. Model adaptation

Since both the projection matrix $\boldsymbol{U}^T(n)$ and the interaction matrix are local approximations of the full nonlinear mapping, they need to be updated constantly. We choose a receding window approach with window size M.

At current iteration i, we estimate the projection matrix $\boldsymbol{U}_i^T$ and local interaction matrix $\boldsymbol{L}_i$ with M samples of the most recent data. Using the interaction matrix and the local target $\boldsymbol{c}_i^*$, we can derive the one-step command $\delta\boldsymbol{r}_i$ by (22). Once we execute the motion $\delta\boldsymbol{r}_i$, a new contour data $\boldsymbol{c}_{i+1}$ is obtained. We move to the next iteration $i+1$. A new pair of input and shape data $[\delta\boldsymbol{r}_i, \boldsymbol{c}_{i+1}]$ is obtained. We shift the window by deleting the oldest data in the window and add in the new data pair. Then, using the shifted window, we compute one step control at iteration $i+1$.

The receding window approach ensures that, at each iteration, we are using the latest data to estimate the interaction matrix. The overall algorithm is initialized with small random motions around the initial configuration. First, M samples of shape data and the corresponding robot motions are collected. With this initialization, we can simultaneously solve for the projection matrix and estimate the initial interaction matrix using the methods described in Sections 4.1 and 4.3. Using the projection matrix and the initial/target shapes, we can then find an intermediate target (see Section 4.2).

We consider quasi-static deformation. Hence, at each iteration the system is in equilibrium and can be linearized according

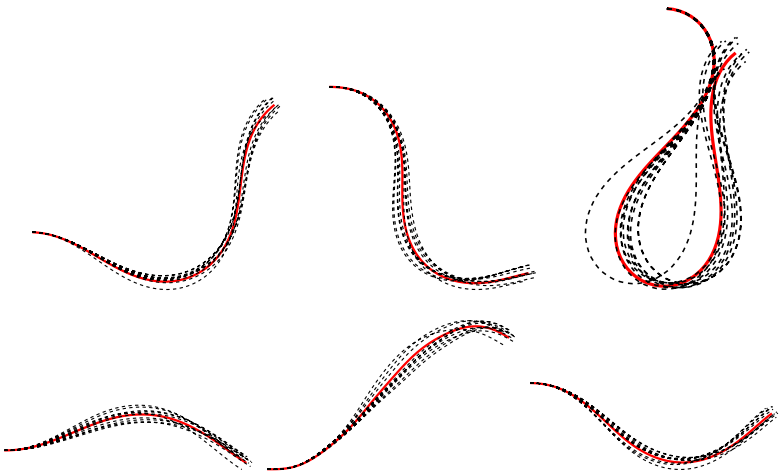


Fig. 4. Six trials conducted to test various choices of feature dimension k for a cable. In each sub-figure, the solid red lines are the initial shapes and the dashed black are the shapes resulting from 10 random motions of the right tip (translations limited to $\pm 5\%$ of the length, rotations limited to $\pm 5°$).

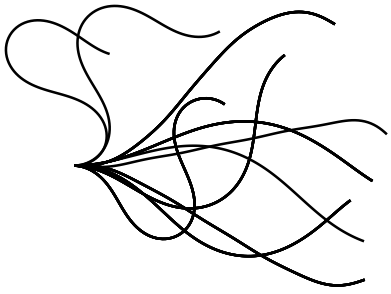


Fig. 5. Ten distinctive cable shapes generated by large motion: angle variation: $[-\frac{\pi}{2}, \frac{\pi}{2}]$, maximum translation: 106% of the cable length.

Table 1
Explained variance $\Upsilon(k)$ for the 6 trials with small motion.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6
$k = 1$	0.727	0.795	0.871	0.847	0.847	0.705
$k = 2$	0.992	0.995	0.996	0.997	0.997	0.994
$k = 3$	0.999	0.999	0.999	0.999	0.999	0.999

Table 2
Explained variance $\Upsilon(k)$ computed with large motion.

k	0	1	2	3	4	5
$\Upsilon(k)$	0	0.5444	0.7218	0.8927	0.9919	0.9990

to (4). The data that best captures the current system are the most recent ones. The choice of M is a trade-off between locality and richness. For fast varying deformations,[2] we would expect to reduce M since a larger M will hinder the locality assumption. Yet, if M is too small, it affects the estimation of $\hat{\boldsymbol{L}}_i$ (refer to the detailed discussion in Section 4.3).

5. Simulation results

In this section, we present the numerical simulations that we ran to validate our method.

5.1. Simulating the objects

We ran simulations on MATLAB (R2018b) with two types of objects: a rigid box and a deformable cable, both constrained to move on a plane. The rigid object is represented by a uniformly sampled rectangular contour. The controllable inputs are its position and orientation. For the cable, we developed a simulator, which is publicly available at https://github.com/Jihong-Zhu/cableModelling2D. The simulator relies on the differential geometry cable model introduced in [32], with the shape defined by solving a constrained optimization problem. The underlying principle is that the object's potential energy is minimal for the object's static shape [33]. Position and orientation constraints (imposed at the cable ends) are input to the simulator. The output is the sampled cable. Figs. 4–6, 9 and 10 show simulated shapes of cables and rigid boxes. We choose $K = 50$ samples for both rigid objects and cables. The camera perspective projection is simulated, with optical axis perpendicular to the plane.

5.2. Selecting the feature dimension k

To check whether choosing $k = n$ can represent the shape accurately, we simulate 6 trials with distinct initial shapes of a cable. The dimension of the robot motion vector $\delta \boldsymbol{r}$ is $n = 3$ (two translations and one rotation of the right tip), and the motions are limited: each translation to $\pm 5\%$ of the cable length and the rotation to $\pm 5°$. This range of motion gives a rule of thumb, which we have used for generating random movements throughout our experiments. For each trial, we command $M = 10$ random motions around the initial shape using our simulator. Fig. 4 shows the 6 initial cable shapes (solid red) and the resulting shapes from 10 random movements (dashed black).

For each trial, we apply PCA to map the cable contour $\boldsymbol{c} \in \mathbb{R}^{2K}$ to feature vector $\boldsymbol{s} \in \mathbb{R}^k$, as explained in Section 4.1. We do this for $k = 1$, 2 and 3 and for each of these 18 experiments, we calculate the explained variance $\Upsilon(k)$ with (9). Table 1 shows these explained variances. In all 6 trials, $k = n = 3$ yields explained variances very close to 1. This result confirms that choosing $k = n$ as the dimension of the feature vector gives an excellent representation of the shape data. It is also possible to select $k = 2$, since the first two components can represent more than 99% of the variance. Nevertheless, the simulation is noise-free. Therefore, although $\Upsilon(k)$ increases little from $k = 2$ to $k = 3$, this increase is not related to noise but to an actual gain in data information.

[2] The notion of fast or slow varying depends on both the speed of manipulation, and on the objects deformation characteristics (which affect the rate of change in shapes) with regard to the image processing time.

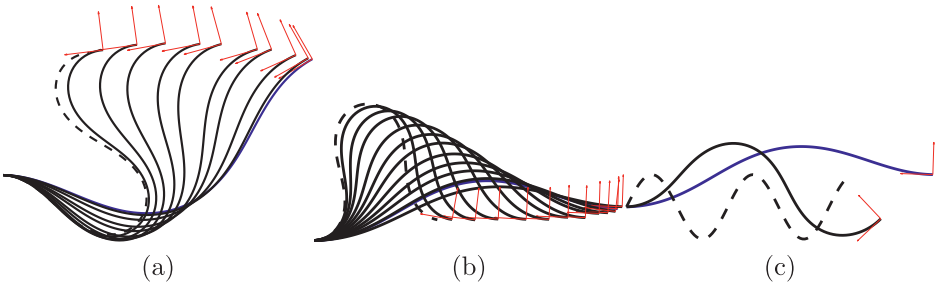


Fig. 6. Cable manipulation with a single end-effector, moving the right tip. (a): a reachable target, the blue and black lines are the initial and intermediate shapes, respectively, and the dashed black line is the target shape. The red frame indicates the end-effector position and orientation generated by our controller. (b): Adding Gaussian noise with zero mean and 0.01 standard deviation to the shape data with a reachable target. (c): An unreachable target and the final shape obtained with our controller. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

At this stage, it is legitimate to ask: *how does this scale to larger movements*? Fig. 5 illustrates 10 cable shapes generated by large movements (angle variation: $[-\frac{\pi}{2}, \frac{\pi}{2}]$, maximum translation: 106%). Again, we apply PCA ($M = 10$); Table 2 shows the $\Upsilon(k)$ resulting from various values of k.

Comparing Tables 1 and 2, it is noteworthy that $\Upsilon(4)$ with large motion is smaller than $\Upsilon(2)$ with small motion. There are two possible explanation here. One is that when shapes stays local, the local linear mapping $\boldsymbol{L}$ in (4) remains constant and we need less features to characterize it; the more the shape varies, the more features we need. Another possible explanation is that for larger motions, $M = 10$ shapes may be insufficient for PCA. Likely, the larger the changes, the larger the number of shapes M needed.

5.3. Manipulation of deformable objects

With our cable simulator, we can now test the controller to modify the shape from an initial to a target one. Again, the left tip of the cable is fixed, and we control the right tip with $n = 3$ degrees of freedom (two translations and one rotation). Using the methods described in Section 4, we choose window size $M = 5$, the Tikhonov factor $\lambda = 0.01$, the local target threshold $\epsilon = 0.8$, the control gain $\alpha = 0.01$. To quantify the effectiveness of our algorithms in driving the contour to $\boldsymbol{c}^*$, we define a scalar measure: the Average Sample Error (ASE). At iteration i, with current contour $\boldsymbol{c}_i$ it is:

$$\text{ASE} = \frac{\|\boldsymbol{c}_i - \boldsymbol{c}^*\|_2}{2K}. \tag{30}$$

A small ASE indicates that the current contour is near the target one. In Section 4.4, we have proved that our controller asymptotically stabilizes the feature vector, $\boldsymbol{s}$ to $\boldsymbol{s}^*$. Hence, since we have also shown that $\boldsymbol{s}$ is a "very good" representation" of $\boldsymbol{c}$, we also expect our controller to drive $\boldsymbol{c}$ to $\boldsymbol{c}^*$, thus ASE to 0. This measure is also used in the real experiments.

Using the cable simulator, we compare the convergence of two control laws proposed in our paper (22) and (23) against a baseline algorithm in [16] which uses Fourier parameters as feature. To make methods compatible, we choose first order Fourier approximation. Note that this results in a feature vector of dimension of 6 (see [16]) which is still twice the number k used in our method. We also normalize the computed control action and then multiply by the same gain factor 0.01.

We also introduced artificial noise to the contour data, to test the robustness of our method. For a unit length cable, we add Gaussian noise of zero mean and 0.01 standard deviation to the contour sample points. Fig. 6(b) shows that our algorithm converges in these conditions as well. It is worth mentioning that in robotic experiments, as shape data is obtained and sampled from the images, signal noise is inevitable. Yet, our framework is robust enough to still converge to the target shape/pose (see Section 6 for detailed real robot experiments).

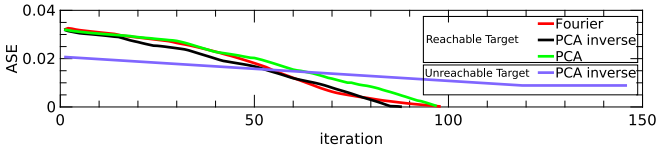


Fig. 7. The evolution of the ASE of the simulated cable manipulation using our method against the Fourier-based method as baseline and the ASE of the unreachable target with (23). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 6 shows two other simulation results: on the left, a reachable target and on the right an unreachable one. In Fig. 6(a), the cable shape successfully evolves towards the target thanks to our controller (23). Fig. 6(c) shows the starting shape (blue), unreachable target (dash black) and final shape (solid black) obtained using (23). Note that the controller gets stuck in a local minimum.

Fig. 7 compares the evolution of ASE with our methods against the Fourier-based method for the reachable target; in the same figure, we also plot the evolution of ASE using (23) for the unreachable target. We can observe that our method provides faster convergence using half the features than [16]. Also, directly computing the inverse (23) provides faster convergence than (22). It is noteworthy to point out that the Fourier-based method requires a different parameterization for closed and open contours (see [9,16]), whereas in our framework, the parameterization can be kept the same. Last but not least, our approach is the only one among the three, which has been validated on both rigid and deformable objects.

5.4. Comparison with the Broyden update law

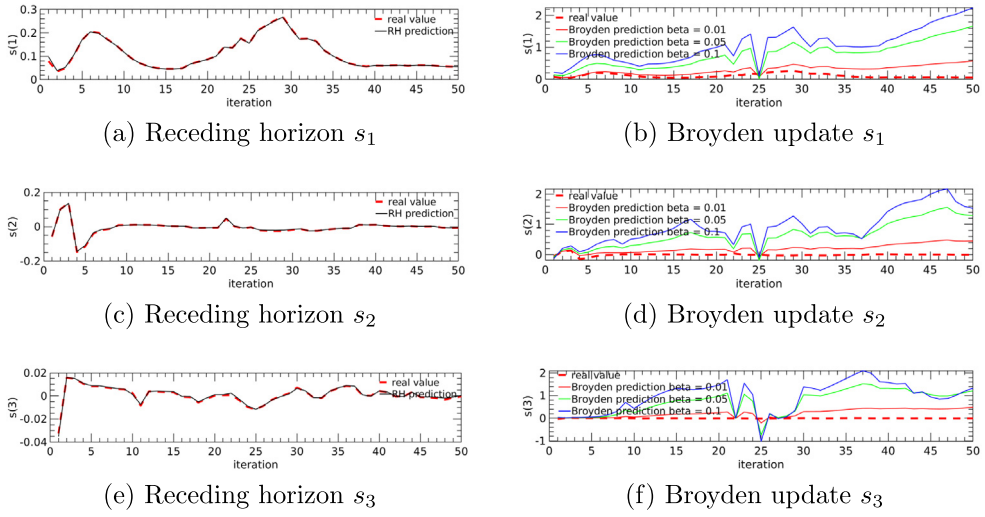
The Broyden update law [34], has been used to update the interaction matrix in classic visual servoing [35–37] and shape servoing [38].

In this section, we compare it with our method for updating the interaction matrix (19), which relies on a receding horizon. We will hereby show why the Broyden update law is not applicable in our framework.

The Broyden update is an iterative method for estimating $\boldsymbol{L}_i$ at iteration i. Its standard discrete-time formulation is:

$$\hat{\boldsymbol{L}}_i = \hat{\boldsymbol{L}}_{i-1} + \beta \frac{\delta \boldsymbol{s}_{i-1} - \hat{\boldsymbol{L}}_i \delta \boldsymbol{r}_{i-1}}{\delta \boldsymbol{r}_{i-1}^T \delta \boldsymbol{r}_{i-1}} \delta \boldsymbol{r}_{i-1}^T, \ \forall \boldsymbol{r}_{i-1} \neq \boldsymbol{0} \tag{31}$$

with $\beta \in [0; 1]$ an adjustable gain. Using our simulator, we estimate the interaction matrix using both Broyden update (with three different values of β) and our receding horizon method



(a) Receding horizon s_1 (b) Broyden update s_1

(c) Receding horizon s_2 (d) Broyden update s_2

(e) Receding horizon s_3 (f) Broyden update s_3

Fig. 8. Comparison – for estimating $\boldsymbol{s}$ – of the receding horizon approach (RH, left) and of the Broyden update (right, with three values of β). The topmost, middle and bottom plots show the one step prediction of s_1, s_2 and s_3, respectively. In all plots, the dashed red curve is the ground truth from the simulator. The plots clearly show that the receding horizon approach outperforms all three Broyden trials. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

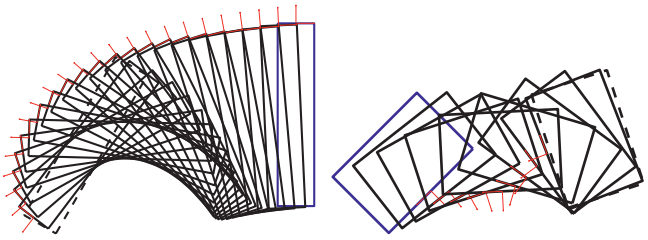


Fig. 9. Manipulation of a rigid object with a single end-effector (red frame). The initial, intermediate and target contours are respectively blue, solid black and dashed black. Note that in both cases, our controller moves the object to the target pose. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

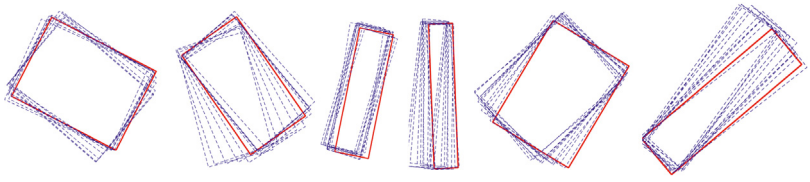


Fig. 10. From an initial (red) pose, we generate 10 (dashed blue) random motions of a rigid object. This figure shows multiple examples of different rectangular rigid objects. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

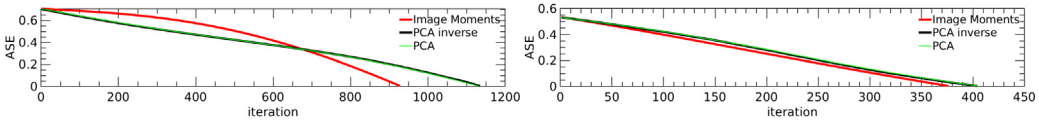


Fig. 11. Evolution of ASE of the simulated rigid object manipulation using our method against image moments. Left: simulation in Fig. 9, Right: simulation in Fig. 9. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

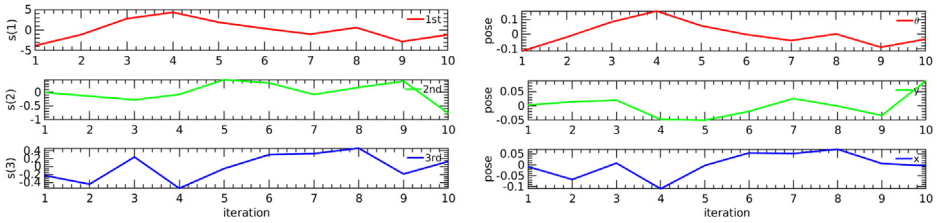


Fig. 12. Progression of the auto-generated feature components (row 1, 3, 5: s_1, s_2, s_3) vs. object pose (row 2, 4, 6: x, y, θ). We have purposely arranged the variables with high correlation with the same color. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(19). We then compare (with $\hat{\boldsymbol{L}}$ estimated with either method) the one-step prediction of the resulting feature vector:

$$\hat{\boldsymbol{s}}_{i+1} = \hat{\boldsymbol{L}}_i \boldsymbol{r}_i + \boldsymbol{s}_i, \tag{32}$$

with the ground truth $\boldsymbol{s}_{i+1}$ from the simulator. The results (plotted in Fig. 8) show that receding horizon outperforms all three Broyden trials. One possible reason is that the components of $\boldsymbol{s}$ fluctuate since (at each iteration) a new matrix $\boldsymbol{U}$ is used. These variations cause the Broyden method to accumulate the result from old interaction matrices, and therefore perform badly on a long term. This result contrasts with that of [38], where the Broyden method performs well since there is a fixed mapping from contour data to feature vector. Another advantage of the receding horizon approach is that it does not require any gain tuning.

5.5. Manipulation of rigid objects

The same framework can also be applied to rigid object manipulation. Consider the problem of moving a rigid object to a certain position and orientation via visual feedback. This time, the shape of the object does not change, but its pose will (it can translate and rotate). We use the same M, λ, ϵ and α as for cable manipulation. We compare the convergence of two control laws proposed in our paper (22) and (23) against a baseline using image moments [21]. The translation and orientation can be represented with image moments and the analytic interaction matrix can be computed as explained in [21]). To make methods compatible, we normalize the computed control and then multiply it by the same factor 0.01.

Fig. 9 shows two simulations where our controller successfully moves a rigid object from an initial (blue) to a target (dashed black) pose using control law (23). Fig. 11 compares convergence of our methods against the image moments method. We can observe that our method provides a slightly slower convergence. Directly computing the inverse (23) provides a convergence similar to (22). Later, we will show why our method is slower. Yet, the fact that it can be applied on both deformable and rigid objects makes it stand out over the other techniques.

5.6. Feature analysis for rigid objects

In this section, we analyze locally what each component of the feature vector represents, in the case of rigid object manipulation. To this end, we apply $M = 10$ random movements (rotation range $[-0.11, 0.09]$, maximum translation 15% of the width) to multiple rigid rectangular objects (see Fig. 10). We compute the projection matrix as explained in Section 4.1, and transform the contour samples to feature vectors. Then, we seek the relationship – at each iteration – between the object pose x, y, θ and the components of the feature vector $\boldsymbol{s}$ generated by PCA. To this end, we use bivariate correlation [39] defined by:

$$\rho = \frac{E[(\xi - \bar{\xi})(\zeta - \bar{\zeta})]}{\sigma_\xi \sigma_\zeta}, \tag{33}$$

where ξ and ζ are two variables with expected values $\bar{\xi}$ and $\bar{\zeta}$ and standard deviations σ_ξ and σ_ζ. An absolute correlation $|\rho|$ close to 1 indicates that the variables are highly correlated. All the simulations in Fig. 10 exhibit similar correlation between the computed feature vector and the object pose. In Table 3, we show one instance (Left first simulation in Fig. 10) of the correlation between variables, with high absolute correlations marked in red. It is clear from the table that each component in the feature vector relates strongly to one pose parameter. We further demonstrate the correlation in Fig. 12, where we plot the evolution of object poses and feature components. Note that s_2 and θ are negatively correlated. The slower convergence could be a result of the fact that, in contrast with image moments, here the extracted features and object pose are not completely decoupled. Yet, the main contribution of our method is that it can be directly used for both rigid and deformable objects. Therefore, it can be expected to be slower than methods specifically designed for rigid objects (such as image moments).

Table 3
Correlation ρ between s_1, s_2, s_3 and x, y, θ.

	x	y	θ
s_1	−0.2819	−0.3343	0.9887
s_2	0.2607	−0.8547	−0.0465
s_3	0.9230	0.3629	−0.1426

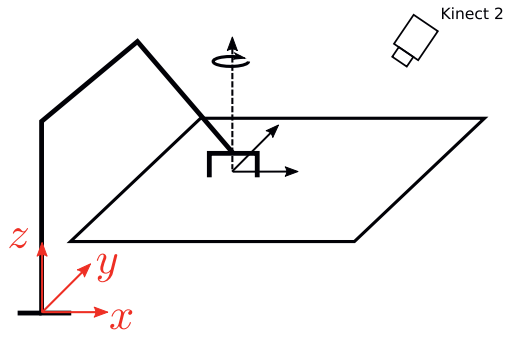


Fig. 13. Overview of the experimental setup.

6. Experiments

Fig. 13 outlines our experimental setup. We use a KUKA LWR IV arm. We constrain it to planar ($n = 3$) motions $\delta\boldsymbol{r}$, defined in its base frame (red in the figure): two translations δx and δy and one counterclockwise rotation $\delta\theta$ around z. A Microsoft Kinect V2 observes the object.[3] A Linux-based 64-bit PC processes the image at 30 fps. In the following sections, we first introduce the image processing for contour extraction, then present the experiments.

6.1. Image processing

This section explains how we extract and sample the object contours from an image. We have developed two pipelines, according to the kind of contours (See Fig. 14): open (e.g., representing a cable) and closed. We hereby describe the two.

6.1.1. Open contours

The overall pipeline for extracting an open contour is illustrated in Fig. 15 and **Algorithm** 2. On the initial image, the user manually selects a Region of Interest (ROI, see Fig. 15(a)) containing the object. In this ROI, we apply thresholding, followed by a morphological opening, to obtain a binary image as in Fig. 15(b). This image is dilated to generate a mask (Fig. 15(c)) used to compute the new ROI for the following image. Fig. 15(e) is the object after a small manipulation motion and 15(f) shows the mask (in gray color) which contains the cable. The OpenCV *findContour* function is applied to binary image, then two contours are extracted based on the two known ends of the cable, both are re-sampled (with same value of K) using **Algorithm** 2 and finally merged (by interpolation, for each sample, between the two contours' corresponding point) into the uniformly sampled open contour $\boldsymbol{c}$ (see Fig. 15(d), where the green box indicates the end-effector).

[3] We only use the RGB image – not the depth – from the sensor.

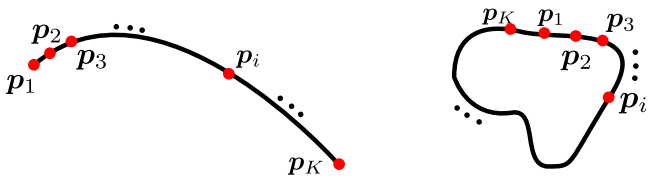


Fig. 14. Open (left) and closed (right) contours can be both represented by a sequence of sample pixels in the image.

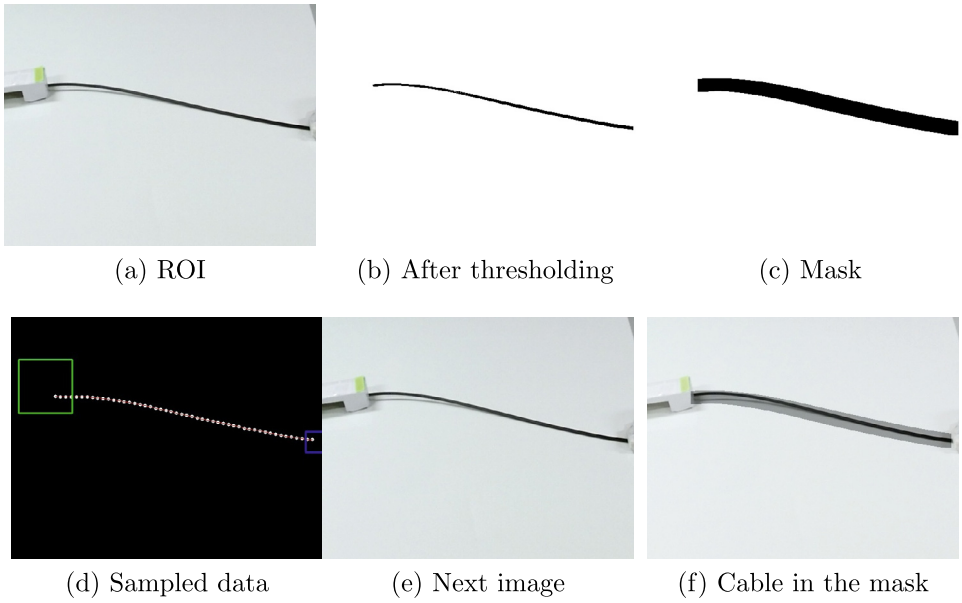


Fig. 15. Image processing steps needed to obtained the sampled open contour of an object (here, a cable). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

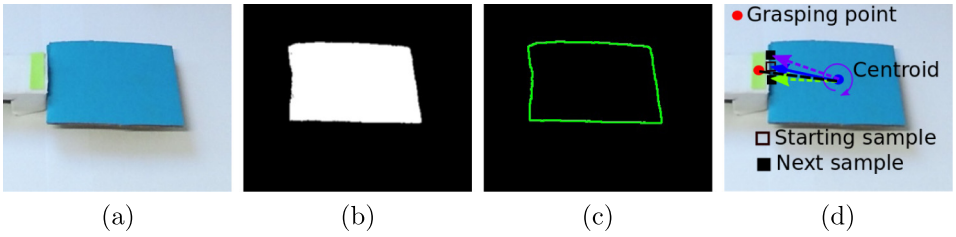


Fig. 16. Image processing for getting a sampled closed contour: (a) original image, (b) image after thresholding and Gaussian blur, (c) extracted contour, (d) finding the starting sample and the order of the samples. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

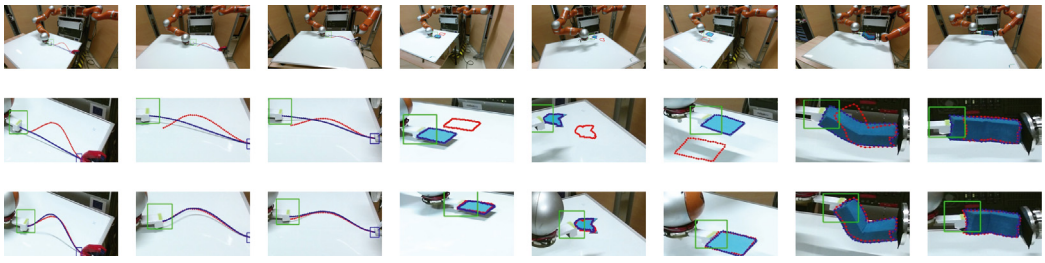
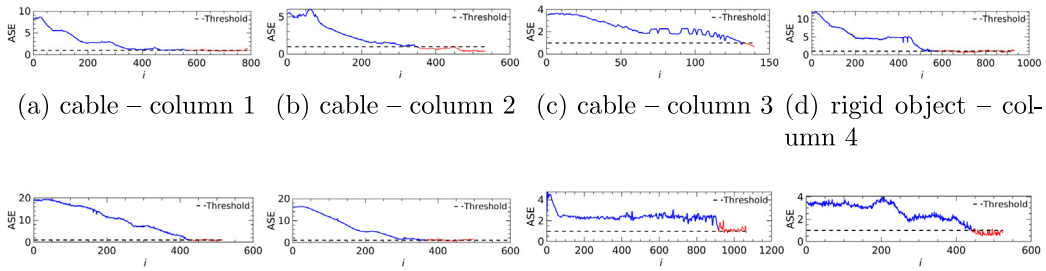


Fig. 17. Eight experiments with the robot manipulating different objects. From left to right: a cable (columns 1–3), a rigid object (columns 4–6) and a sponge (columns 7 and 8). The first row shows the full Kinect V2 view, and the second and the third columns zoom in to show the manipulation process at the first and last iterations. The red contour is the target one, whereas the blue contour is the current one. The green square indicates the end-effector. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.1.2. Closed contours

The procedure is shown in Fig. 16. For an object with uniform color (in the experiment blue), we apply HSV segmentation, followed by Gaussian blur of size 3, and finally the OpenCV *findContour* function, to get the object contour. The contour is then re-sampled using **Algorithm** 2. The starting point and the order of the samples is determined by tracking the grasping point (red dot in Fig. 16(d)) and the centroid of the object (blue dot). We



(a) cable – column 1 (b) cable – column 2 (c) cable – column 3 (d) rigid object – column 4

(e) rigid object – column 5 (f) cable – column 6 (g) sponge – column 7 (h) sponge – column 8

Fig. 18. Evolution of e_i at each iteration i, for the 8 experiments of Fig. 17. The black dashed lines indicate the threshold ASE = 1 pixel. The blue curves show e_i until the termination condition, whereas the red curves show the error until manual termination by the human operator. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

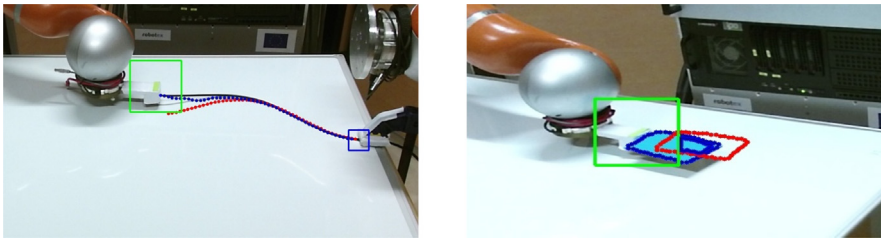
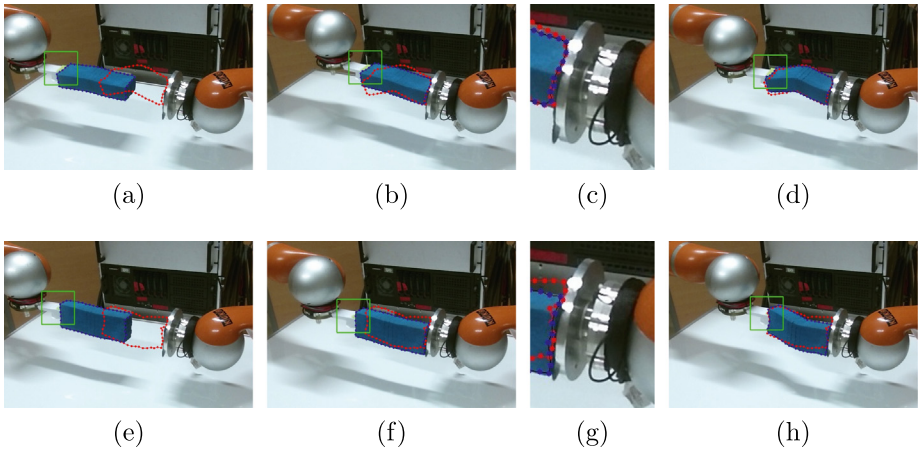


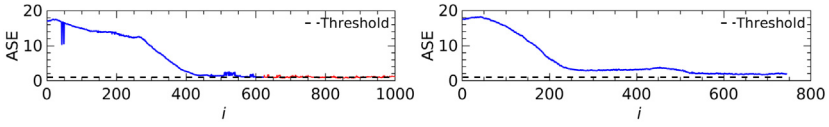
Fig. 19. False contour data from the image can cause noise in ASE.



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

(e) (f) (g) (h)

Fig. 20. Two "move and shape" experiments grouped into two rows. The target contour (red dotted) is far from the initial one. This requires the robot to (1) move the object, establish contact with the right – fixed – robot arm, (2) give the object the target shape, by relying on the contact. The first column shows the starting configuration, the second column presents the contact establishment, and the third column zooms in to show the alignment. The last column shows the final results.



(a) move and shape – row 1 (b) move and shape – row 2

Fig. 21. The evolution of e_i for the experiments of Fig. 20. The black dashed line indicates the threshold ASE = 1. The blue curves show e_i until the termination condition, whereas the red curves show the error until manual termination by the human operator. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

obtain the vector connecting the grasping point to the centroid. Then, the starting sample is the one closest to this vector, and we proceed along the contour clockwise. Therefore, the extracted contour is ordered clockwise and always starting from the same point. This solves the contour ambiguity for symmetrical objects.

6.2. Vision-based manipulation

In this section, we present the experiments that we ran to validate our algorithms, also visible at https://youtu.be/gYfO2ZxZ5KQ. To demonstrate the generality of our framework, we tested it with:

Algorithm 2 Generate fixed number of points with uniform spacing *Input* $\boldsymbol{K}$ = target number of data samples *Input* $\boldsymbol{\epsilon}$ = infinitesimal threshold for equality of distances *Output* $\boldsymbol{P}_O = [\boldsymbol{p}_O(1), \cdots, \boldsymbol{p}_O(K)]$: re-sampled data with uniform spacing.

compute the full length $\mathcal{L}$ of $\boldsymbol{P}_I$.
compute desired distance per sample: $\mu = \mathcal{L}/K$
$l = 1$, $dist = 0$
$\boldsymbol{p}_{curr} = \boldsymbol{p}_O(l) = \boldsymbol{p}_I(l)$
$h = l = l + 1$
while $l \leq N$ **do**
  $\boldsymbol{p}_{next} = \boldsymbol{p}_I(l)$
  $d = ||\boldsymbol{p}_{next} - \boldsymbol{p}_{curr}||_2$
  if $d + dist \leq \mu$ **then**
    $dist = dist + d$
    $\boldsymbol{p}_{curr} = \boldsymbol{p}_{next}$
    $l = l + 1$
  else
    $\boldsymbol{p}_{curr} = \boldsymbol{p}_O(h) = \boldsymbol{p}_{curr} + (\boldsymbol{p}_{next} - \boldsymbol{p}_{curr})\frac{\mu - dist}{d}$
    $h = h + 1$
    $dist = 0$
  end if
end while
if $|\mu - dist| < \epsilon$ **then**
  $\boldsymbol{p}_O(h) = \boldsymbol{p}_{curr}$
end if

- Rigid objects represented by closed contours,
- Deformable objects represented by open contours (cables),
- Deformable objects represented by closed contours (sponges).

We carried out different experiments with a variety of initial and target contours and camera-to-object relative poses. The variety of both geometric and physical properties demonstrates the robustness of our framework. The variety of camera-to-object relative poses shows that – as usual in image-based visual servoing [2] – camera calibration is unnecessary. The algorithm and parameters are the same in all experiments; the only differences are in the image processing, depending on the type of contour (closed or open, see Section 6.1).

We obtain the target contours by commanding the robot with predefined motions. Once the target contour is acquired, the robot goes back to the initial position, and then should autonomously reproduce the target contour. Again, we set the number of features $k = n = 3$, and use $K = 50$ samples to represent the contour $\boldsymbol{c}$. We set the window size $M = 5$, both for obtaining the feature vector $\boldsymbol{s}$ and the interaction matrix $\boldsymbol{L}$. We choose the control gain to be 0.01. A larger control gain may result in faster convergence, but could also lead to oscillation. The local target threshold ϵ is set to 0.8. A higher threshold will result in closer local target and vice versa. The Tikhonov factor used to ensure numerical stability for matrix inversion, is set to $\lambda = 0.01$.

At the beginning of each experiment, the robot executes 5 steps of small[4] random motions to obtain the initial features and interaction matrix.

For all the experiments, we set the same termination condition at iteration $i + 1$ using ASE defined in (30) such that:

1. $\text{ASE}_i < 1$ pixel and
2. $\text{ASE}_{i+1} \geq \text{ASE}_i$.

[4] The notion of small is relative, and usually dependent on the size of the object the robot is manipulating. Refer to Section 5.2 (especially Fig. 4) for a discussion on this.

In the graphs that follow, we show the evolution of ASE in blue before the termination condition, and in red after the condition (until manual stop by the operator).

Fig. 17 presents 8 experiments, one per column. Columns 1 – 3, 4 – 6 and 7 – 8 show respectively manipulation of: cable, rigid object and sponge. The first row presents the full RGB image obtained from Kinect V2. The second and third rows zoom in on the manipulation at the initial and final iterations. We track the end-effector in the image with a green marker for contour sampling. The target and current contours are drawn in red and blue, respectively.

Fig. 18 shows the decreasing trend of error ASE for each experiment. The initial increase of ASE in the experiments can be due to the random motion at the beginning of the experiments. In general, we found that ASE is more noisy for the closed than for the open contour. This discontinuity is visible in Figs. 18(c) and 18(d) (zigzag evolution). Such noise is likely introduced by the way we sampled the contour. Also, the noise in the contour extraction is more visible on Figs. 18(g) and 18(h). The two plots show that our framework can converge to the target in the presence of noisy image processing. When we have false contour data, the value of ASE may encounter a sudden discontinuity. Fig. 19 shows examples of these false samples, output by the image processing pipeline. Despite these errors, thanks to the "forgetting nature" of the receding horizon and to the relatively small window size ($M = 5$), the corrupted data will soon be forgotten, and it will not hinder the overall manipulation task. Yet, the overall framework would benefit from a more robust sensing strategy, as in [40].

Finally, since our framework can deal with both rigid and deformable objects, we tested it in two experiments where the same object (a sponge) can be both rigid (in the free space), and deformed (when in contact with the environment). These experiments require the robot to: 1) move the object, establish contact, 2) give the object the target shape, by relying on the contact. Fig. 20 presents these two original "move and shape" servoing experiments with the corresponding errors ASE plotted in Fig. 21. We use a second fixed robot arm to generate the deforming contact. As the figures and curves show, both experiments were successful.

The success of the "move and shape" task is largely dependent on the contact establishment. However, even when the initial contact has some misalignment (see Figs. 20(c) and 20(g)), our framework can still reduce the ASE to give a reasonable final configuration (see Fig. 20(h) and Fig. 21(b)).

7. Conclusion

In this paper, we propose algorithms to automatically and concurrently generate object representations (feature vectors) and models of interaction (interaction matrices) from the same data. We use these algorithms to generate the control inputs enabling a robot to move and shape the said object, be it rigid or deformable. The scheme is validated with comprehensive experiments, including a target contour that requires both moving and shaping. We believe it is unprecedented in previous research. Our framework adopts a model-free approach. The system characteristics are computed online with visual and manipulation data. We do not require camera calibration, nor a priori knowledge of the camera pose, object size or shape.

The proposed approach has two major limitations: 1. *The challenge of extending it to 6 DoF motion*, 2. *Global convergence cannot be guaranteed*. Below, we discuss each limitation and present possible solutions.

An open question remains the management of 6 DOF motion of the robot. Indeed, while the proposed controller can be

easily generalized to 6 DOF motions, it relies on a sufficiently accurate extraction of feature vectors from vision sensors. A very challenging task is to generate complete and reliable 3D feature vectors of objects from a limited sensor set, due to partial views of the object and to occlusions. To extend it, the framework should benefit from robust deformation sensing. In addition, since the approach relies on local linear models, we expect that with higher DOF, the algorithm will more likely get stuck in local minima.

The second drawback is that the representation and model of interaction are local. Thus, they cannot guarantee global convergence. In addition, our framework cannot infer whether a shape is reachable or not. This drawback is solvable by using a global deformation model for control. But as we mentioned earlier, a global model usually requires an offline identification phase which we want to avoid. In fact, for different objects, we will need to re-identify the model. There is a dilemma in using a global deformation model.

Maybe one of the possible solutions to this dilemma is to have both our method and deep learning based methods run in parallel. While our scheme enables fast online computation and direct manipulation, the extracted data can be used by a deep neural network to obtain a global interaction mapping. Once a global mapping is learned, it can later be used for direct manipulation and to infer feasibility of the goal shape.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by the EU H2020 research and innovation programme as part of the project VERSATILE, France under grant agreement No 731330, by the Research Grants Council (RGC) of Hong Kong under grant number 14203917, and by the PROCORE-France/Hong Kong RGC Joint Research Scheme, France under grant F-PolyU503/18.

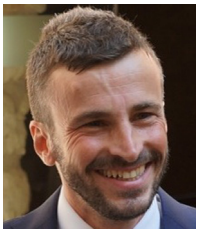
References

[1] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, Y. Mezouar, Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey, Int. J. Robot. Res. (2018).
[2] F. Chaumette, S. Hutchinson, Visual servo control, part I: Basic approaches, IEEE Robot. Autom. Mag. 13 (4) (2006) 82–90.
[3] H. Inoue, Hand-eye coordination in rope handling, in: Proc. of Int. Symposium on Robotics Research, MIT PRESS, 1984, pp. 163–174.
[4] P.W. Smith, N. Nandhakumar, A.K. Ramadorai, Vision based manipulation of non-rigid objects, in: IEEE Int. Conf. on Robotics and Automation, 4, IEEE, 1996, pp. 3191–3196.
[5] D. Berenson, Manipulation of deformable objects without modeling and simulating deformation, in: 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IEEE, 2013, pp. 4525–4532.
[6] R. Lagneau, A. Krupa, M. Marchal, Active deformation through visual servoing of soft objects, in: IEEE Int. Conf. on Robotics and Automation, 2020.
[7] R. Lagneau, A. Krupa, M. Marchal, Automatic shape control of deformable wires based on model-free visual servoing, IEEE Robot. Autom. Lett. 5 (4) (2020) 5252–5259.
[8] D. Navarro-Alarcon, Y. Liu, J.G. Romero, P. Li, On the visual deformation servoing of compliant objects: Uncalibrated control methods and experiments, Int. J. Robot. Res. 33 (11) (2014) 1462–1480.
[9] D. Navarro-Alarcon, Y.-H. Liu, Fourier-based shape servoing: A new feedback method to actively deform soft objects into desired 2D image contours, IEEE Trans. Robot. 34 (1) (2018) 272–279.
[10] D. Navarro-Alarcon, A. Cherubini, X. Li, On model adaptation for sensorimotor control of robots, in: Chinese Control Conference, 2019, pp. 2548–2552.
[11] A. Cherubini, V. Ortenzi, A. Cosgun, R. Lee, P. Corke, Model-free vision-based shaping of deformable plastic materials, Int. J. Robot. Res. 39 (14) (2020) 1739–1759.
[12] Z. Wang, X. Li, D. Navarro-Alarcon, Y.-h. Liu, A unified controller for region-reaching and deforming of soft objects, in: 2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).
[13] D. Navarro-Alarcon, Y. Liu, Uncalibrated vision-based deformation control of compliant objects with online estimation of the Jacobian matrix, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2013.
[14] M. Laranjeira, C. Dune, V. Hugel, Catenary-based visual servoing for tethered robots, in: 2017 IEEE Int. Conf. on Robotics and Automation (ICRA), IEEE, 2017, pp. 732–738.
[15] M. Laranjeira, C. Dune, V. Hugel, Catenary-based visual servoing for tether shape control between underwater vehicles, Ocean Eng. 200 (2020) 1–19.
[16] J. Zhu, B. Navarro, P. Fraisse, A. Crosnier, A. Cherubini, Dual-arm robotic manipulation of flexible cables, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2018.
[17] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, S. Levine, Combining self-supervised learning and imitation for vision-based rope manipulation, in: IEEE Int. Conf. on Robotics and Automation, 2017.
[18] X. Li, X. Su, Y. Gao, Y.-H. Liu, Vision-based robotic grasping and manipulation of USB wires, in: IEEE Int. Conf. on Robotics and Automation (ICRA), 2018.
[19] Z. Hu, T. Han, P. Sun, J. Pan, D. Manocha, 3-d deformable object manipulation using deep neural networks, IEEE Robot. Autom. Lett. 4 (4) (2019) 4255–4261.
[20] Y. She, S. Wang, S. Dong, N. Sunil, A. Rodriguez, E. Adelson, Cable manipulation with a tactile-reactive gripper, in: Robotics: Science and Systems, 2020.
[21] F. Chaumette, Image moments: a general and useful set of features for visual servoing, IEEE Trans. Robot. 20 (4) (2004) 713–723.
[22] C. Collewet, E. Marchand, F. Chaumette, Visual servoing set free from image processing, in: 2008 IEEE Int. Conf. on Robotics and Automation, IEEE, 2008, pp. 81–86.
[23] C. Collewet, E. Marchand, Photometric visual servoing, IEEE Trans. Robot. 27 (4) (2011) 828–834.
[24] M. Bakthavatchalam, F. Chaumette, E. Marchand, Photometric moments: New promising candidates for visual servoing, in: 2013 IEEE Int. Conf. on Robotics and Automation, IEEE, 2013, pp. 5241–5246.
[25] E. Marchand, Subspace-based direct visual servoing, IEEE Robot. Autom. Lett. 4 (3) (2019) 2699–2706.
[26] S.K. Nayar, S.A. Nene, H. Murase, Subspace methods for robot vision, IEEE Trans. Robot. Autom. 12 (5) (1996) 750–758.
[27] K. Deguchi, T. Noguchi, Visual servoing using eigenspace method and dynamic calculation of interaction matrices, in: Proc. of 13th IEEE Int. Conf. on Pattern Recognition.
[28] Q. Sang, G. Tao, Adaptive control of piecewise linear systems: the state tracking case, IEEE Trans. Autom. Control 57 (2) (2012) 522–528.
[29] L. Zhang, W. Dong, D. Zhang, G. Shi, Two-stage image denoising by principal component analysis with local pixel grouping, Pattern recognition 43 (4) (2010) 1531–1549.
[30] R. Zeng, J. Wu, Z. Shao, Y. Chen, B. Chen, L. Senhadji, H. Shu, Color image classification via quaternion principal component analysis network, Neurocomputing 216 (2016) 416–428.
[31] J.-T. Lapresté, F. Jurie, M. Dhome, F. Chaumette, An efficient method to compute the inverse jacobian matrix in visual servoing, in: IEEE Int. Conf. on Robotics and Automation, 2004.
[32] H. Wakamatsu, S. Hirai, Static modeling of linear object deformation based on differential geometry, Int. J. Robot. Res. 23 (3) (2004) 293–311.
[33] H. Wakamatsu, S. Hirai, K. Iwata, Modeling of linear objects considering bend, twist, and extensional deformations, in: IEEE Int. Conf. on Robotics and Automation, 1995.
[34] C.G. Broyden, A class of methods for solving nonlinear simultaneous equations, Math. Comput. 19 (92) (1965) 577–593.
[35] K. Hosoda, M. Asada, Versatile visual servoing without knowledge of true jacobian, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 1994.
[36] M. Jagersand, O. Fuentes, R. Nelson, Experimental evaluation of uncalibrated visual servoing for precision manipulation, in: Int. Conf. on Robotics and Automation, 1997.
[37] F. Chaumette, S. Hutchinson, Visual servo control, part II: Advanced approaches, IEEE Robot. Autom. Mag. 14 (1) (2007) 109–118.
[38] D. Navarro-Alarcon, Y. Liu, J.G. Romero, P. Li, Model-free visually servoed deformation control of elastic objects by robot manipulators, IEEE Trans. Robot. 29(6) (2013) 1457–1468.
[39] W. Feller, An Introduction to Probability Theory and Its Applications, 2, John Wiley & Sons, 2008.
[40] C. Chi, D. Berenson, Occlusion-robust deformable object tracking without physics simulation, in: 2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems.

Jihong Zhu received the M.Sc. in Systems and Control in 2015 from TU Delft and the Ph.D. in Robotics from University of Montpellier in 2020. He conducted his doctoral research at LIRMM. In 2019, he visited ROMI Lab hosted by Dr. David Navarro-Alarcon for 4 months. He is currently a postdoc at Cognitive Robotics department, TU Delft. His research interests include: sensor-based control, and manipulation of soft objects.

David Navarro-Alarcon received the Ph.D. degree in mechanical and automation engineering from The Chinese University of Hong Kong, NT, Hong Kong, in 2014. He was Research Assistant Professor at the CUHK Robotics Institute, from 2015 to 2017. Since 2017, he has been with The Hong Kong Polytechnic University, where he is Assistant Professor at the Department of Mechanical Engineering. His current research interests include perceptual robotics and control theory.

Robin Passama received the M.Sc. in Computer Science in 2002 and graduated in 2006 with a Ph.D. Degree in Computer Science, from the University of Montpellier. As a research engineer, he then worked on many projects and contributed in various experiments. He became a permanent CNRS member in 2012. His main area of expertise is software engineering for robotics.



Andrea Cherubini received the M.Sc. in Mechanical Engineering in 2001 from the University of Rome La Sapienza and a second M.Sc. in Control Systems in 2003 from the University of Sheffield, U.K. In 2008, he received the Ph.D. in Control Systems from La Sapienza. From 2008 to 2011, he was postdoc at INRIA Rennes. Since 2011, he is Associate Professor at Université de Montpellier. His research interests include: physical human–robot interaction, and manipulation of soft objects.