`ROBUSfT`: Robust Real-Time Shape-from-Template, a C++ Library

Mohammadreza Shetab-Bushehri[a], Miguel Aranda[b], Erol Özgür[a], Youcef Mezouar[a], Adrien Bartoli[a]

[a] *CNRS, Clermont Auvergne INP, Institut Pascal, Université Clermont Auvergne, Clermont-Ferrand, F-63000, France*
[b] *Instituto de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, Zaragoza, E-50018, Spain*

Abstract

Tracking the 3D shape of a deforming object using only monocular 2D vision is a challenging problem. This is because one should *(i)* infer the 3D shape from a 2D image, which is a severely underconstrained problem, and *(ii)* implement the whole solution pipeline in real time. The pipeline typically requires feature detection and matching, mismatch filtering, 3D shape inference and feature tracking algorithms. We propose `ROBUSfT`, a conventional pipeline based on a template containing the object's rest shape, texture map and deformation law. `ROBUSfT` is ready-to-use, wide-baseline, capable of handling large deformations, fast up to 30 fps, free of training, and robust against partial occlusions and discontinuities. It outperforms the state-of-the-art methods in challenging video datasets. `ROBUSfT` is implemented as a publicly available `C++` library. We provide the code, a tutorial on how to use it, and a supplementary video of our experiments at `https://github.com/mrshetab/ROBUSfT`.

Keywords:
monocular non-rigid reconstruction, mismatch removal, SfT, validation procedure, `C++` library

1. Introduction

Problem and challenges. Tracking the 3D shape of a deforming object has important applications in augmented reality [1, 2], computer-assisted surgery [3–7] and robotics [8–10]. However, the existing methods are impractical. This is because of the following challenges: (`C1`) real-time implementability and (`C2`) robustness. Challenge `C1` occurs because the solution usually involves a computationally demanding multi-step pipeline. Challenge `C2` occurs because of noises, occlusions, the object being out of the sensor's field of view, large deformations and fast motions. Furthermore, in numerous applications of augmented reality, computer-assisted surgery and robotics, a 2D camera is the de facto sensor owing to its light weight, small size, and low cost [11]. The camera's perspective projection introduces an additional challenge, (`C3`) recoverability of depth from a 2D image. Challenge `C3` becomes extremely strong for deforming objects.

Shape-from-Template. Different priors and constraints have been proposed to resolve challenge `C3`. The most common ones are the object's 3D rest shape, texture map and deformation law, and the camera's calibration. These are the base ingredients for a variety of methods. Among these methods, we are specifically interested in Shape-from-Template (SfT). SfT has been well studied for isometrically deforming objects [12–14] and has been shown to uniquely resolve the depth of each object point [15]. It uses a template formed by the abovementioned priors. SfT's input is a single image of the deformed object and its output is the object's 3D shape seen in the image. We adopt a conventional SfT pipeline shown in Figure 1 to solve the problem of tracking the 3D shape of deforming objects. The pipeline involves keypoint extraction and matching, mismatch filtering, warping, and 3D shape inference steps. We successfully made it real-time and robust by seamlessly integrating both novel and state-of-the-art algorithms. We next give an overview of the strengths and weaknesses of existing SfT methods.

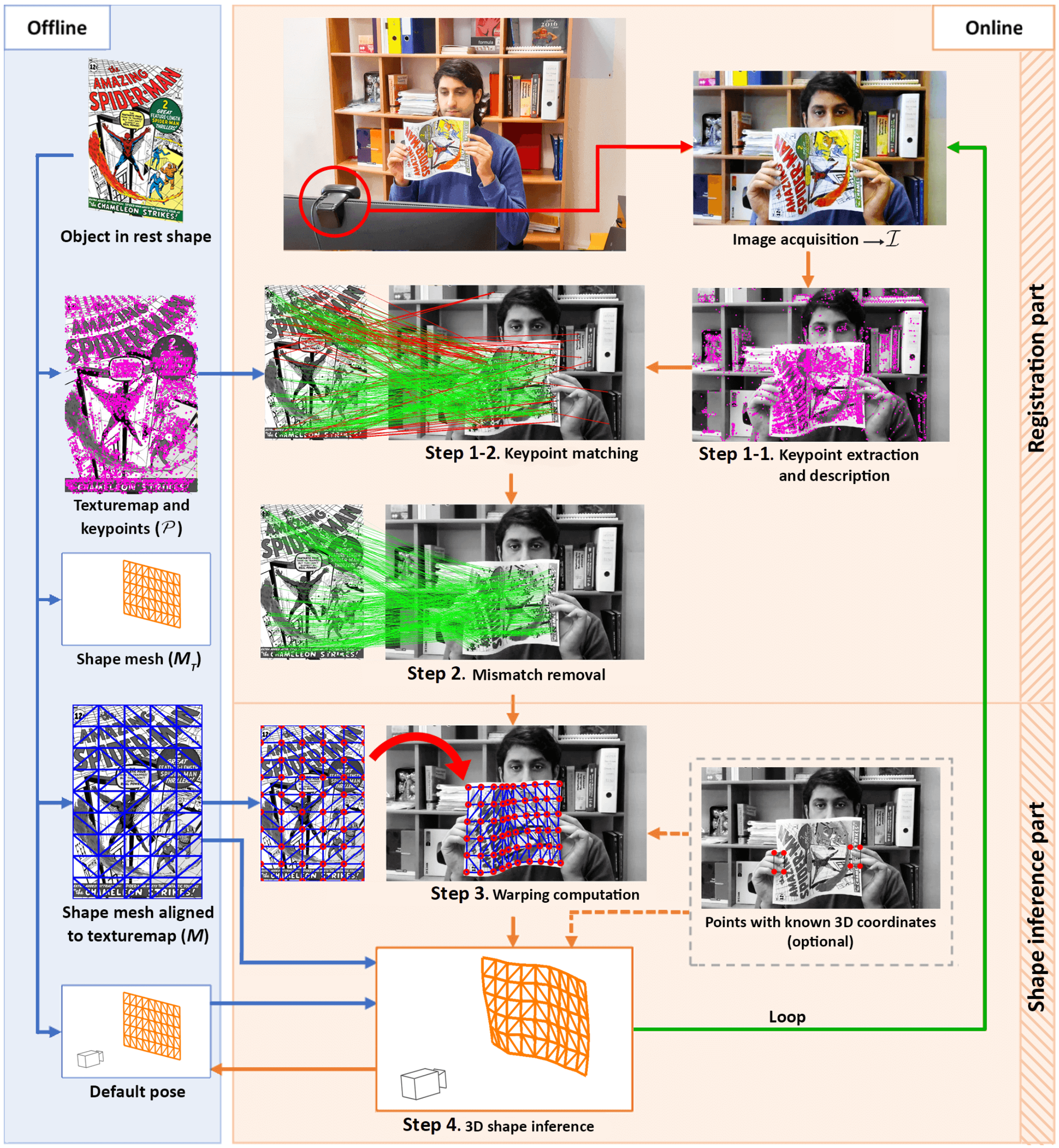

Offline
Online
Object in rest shape
Image acquisition $\longrightarrow \mathcal{I}$
Registration part
Texturemap and keypoints ($\mathcal{P}$)
Step 1-2. Keypoint matching
Step 1-1. Keypoint extraction and description
Shape mesh (M_T)
Step 2. Mismatch removal
Shape mesh aligned to texturemap (M)
Step 3. Warping computation
Points with known 3D coordinates (optional)
Shape inference part
Loop
Default pose
Step 4. 3D shape inference

Figure 1: Overview of ROBUSfT, the proposed SfT pipeline.

Existing SfT methods. SfT can be broken down into two main parts: registration and 3D shape inference. Following this, we categorize existing SfT methods into two groups: (G1) shape inference methods and (G2) integrated methods. G1 methods only cover the 3D shape inference part [10, 12, 13, 15–19]. In contrast, G2 methods cover both the registration and 3D shape inference parts [6, 20–24]. We also review Deep Neural Network

(DNN) based SfT methods, which have been recently introduced, as the third group (G3). G3 methods cover both the registration and 3D shape inference parts [25–29]. The majority of G1 methods are wide-baseline. However, they barely run in real time. Furthermore, a complete solution with registration would be even slower. The majority of G2 methods require an initialization close to the solution. This makes them short-baseline. These methods often fail against occlusions, fast motions and large deformations. Once they have failed, they need to be reinitialized. G3 methods are wide-baseline and run in real time. However, they are object-specific. They require a huge amount of training data and proper computational resources for each new object. This makes it difficult to consider them as general and ready-to-use solution methods. We therefore conclude that there does not exist an SfT method that is complete, real-time, robust and easily applicable to new objects.

Contributions. We list our contributions in four parts.

Contribution to SfT. We propose ROBUSfT, a complete real-time robust SfT pipeline for monocular 3D shape tracking of isometrically deforming thin-shell objects with matchable appearance. With the proposed CPU-GPU architecture, ROBUSfT can track up to 30 fps using 640×480 images on off-the-shelf hardware. It does not require initialization and implements tracking-by-detection. It is wide-baseline and robust to occlusions, the object being out of the field of view, large deformations, and fast motions. To use it with a new object, all it needs is a template of that object. It, thus, does not require any training or fine-tuning and is directly usable in many industrial applications and research studies. ROBUSfT outperforms the state-of-the-art methods in challenging datasets.

Contribution to mismatch removal. We introduce myNeighbor, a novel mismatch removal algorithm. This novel algorithm handles deforming scenes and a large percentage of mismatches. It is lightning fast, reaching 200 fps. It outperforms the existing mismatch removal algorithms in terms of accuracy and execution speed.

Contribution to experimental validation. We design a novel type of validation procedure, called Fake but Realistic Experiment (FREX). With just one run, FREX produces a large number of semi-synthetic scenes featuring an object undergoing isometric deformation under various conditions. The scenes come with 2D and 3D ground truth. This allows easily testing, evaluating, training and validating new algorithms that deal with isometrically deforming objects in tasks such as removing mismatches, registering 2D images and inferring 3D shapes. Unlike other artificially generated scenes of isometrically deforming objects, the images generated by FREX are the result of real object deformations. FREX is very simple to set up. All that is needed is a piece of paper with a set of Aruco markers printed on it.

Contribution to open-source. We release ROBUSfT as an out-of-the-box tool, in the form of a C++ library with a comprehensive tutorial for public use. The code, the tutorial, and a supplementary video of our experiments can be found at `https://github.com/mrshetab/ROBUSfT`.

Paper structure. Section 2 reviews previous work. Section 3 explains ROBUSfT. Section 4 presents FREX. Section 5 describes myNeighbor, conducts a series of experiments and evaluates the results of myNeighbor in comparison to previous work. Section 6 validates ROBUSfT through FREX and real data experiments, and compares the results with previous work. Finally, Section 7 concludes and suggests future work.

2. Previous Work

We review the methods for monocular shape inference of isometrically deforming objects, following the three categories mentioned above, namely, (G1) shape inference methods, (G2) integrated methods, and (G3) DNN-based SfT methods. For each category, we describe the assumptions, main characteristics, and limitations. We finally compare ROBUSfT to these methods.

2.1. *(G1) Shape inference methods*

These methods cover the 3D shape inference part. They assume that the registration between the template and the image was previously computed. For instance, they typically use keypoint matches between the template and the image, with generic mismatch removal methods [17, 30, 30–32]. Very few methods in this category could form a complete SfT pipeline by adding an existing registration solution [2, 17]. Three general groups are found in existing 3D shape inference methods: *(i)* methods using a convex relaxation of isometry called inextensibility [12, 13, 18], *(ii)* methods using local differential geometry [15–17], and *(iii)* methods minimizing a global non-convex cost function [10, 18, 19]. The methods in *(iii)* are the most precise ones; however, they are computationally expensive, and they require initialization. The first two groups of methods are often used to provide an initialization for the third group.

In the first group, Salzmann et al. [13] suggested a closed-form solution to non-rigid 3D surface registration by solving a set of quadratic equations accounting for inextensibility. Later, they replaced equality constraints with inequality constraints and thus sharp deformations could be better recovered [12]. Brunet et al. [18] formulated two shape inference methods based on point-wise and continuous surface models as Second Order Cone Programs (SOCP). In the second group, Bartoli et al. [15] showed that in addition to keypoint 2D coordinates in the image, their first-order differential structure can be used to estimate the depth. Instead of calculating a warp globally, which is time-consuming, Famouri et al. [17] estimated the depth locally for each match pair with respect to both local texture and neighboring matches. In each frame, the most recognizable matches were selected based on offline training. The execution speed of their algorithm is claimed to be up to 14 fps only for the 3D shape inference. In the third group, Brunet et al. [18] proposed a refining isometric SfT method by reformulating the isometric constraint and solving a non-convex optimization problem. The method required a reasonably accurate 3D shape of the deforming surface as the initializing guess. Özgür and Bartoli [19] developed Particle-SfT, which handles isometric and non-isometric deformations. A particle system is guided by deformation and reprojection constraints which are applied consecutively to the particle mesh. Similarly to [18], this algorithm needs an initial guess for the 3D position of the particles; however, for [19], the sensitivity to this initial guess is very low. The closer the guess to the true 3D shape, the faster the convergence. Aranda et al. [10] improved this algorithm in terms of execution speed and occlusion resistance and used it in real-time shape servoing of isometrically deforming objects. They used the 3D shape estimated in one frame as the initial guess for the next frame and thus improved the convergence speed of the algorithm to a great extent. They showed that their algorithm can track a paper sheet covered with markers and being manipulated by a robotic arm. To this end, they only needed to track a handful of markers. Knowing the 3D coordinates of several mesh points also has a significant effect on the convergence speed of the algorithm. The last step of `ROBUSfT` uses the same method to infer the 3D shape, as explained in Section III.

2.2. *(G2) Integrated methods*

These methods handle registration and 3D shape inference at the same time. They minimize a non-convex cost function in order to align the 3D inferred shape with image features. These features can be local [21, 22] or defined at the pixel-level [6, 23, 33].

Ostlund et al. [21] and later Ngo et al. [22] used the Laplacian formulation to reduce the problem size by introducing control points on the surface of the deforming object. The process of removing mismatches was performed iteratively during optimization by projecting the 3D estimated shape onto the image and disregarding the correspondences with higher reprojection errors. Using this procedure, they could reach up to 10 fps for 640×480 input images, restricting the maximum number of template and image keypoints to 500 and 2000, respectively.

As for pixel-level alignment, Collins and Bartoli [23] introduced a real-time SfT algorithm which could handle large deformations and occlusions and reaches up to 21 fps. They combined extracted matches with physical deformation priors to perform shape inference. Collins et al. [6] later extended this algorithm and used it for tracking organs in laparoscopic videos. For achieving better performance, they also exploited organ boundaries as a tracking constraint. Recently, Kairanda et al. [33] have introduced a novel approach that utilizes a physics-based deformation model for reconstruction, simulating the object behavior within a simulator. They use a differentiable renderer to ensure that the reprojection of the object's inferred 3D shape precisely matches the observed images.

These methods are fast and can handle large deformation. Their main drawback, however, is that they are short-baseline. In case of tracking failure, they should be reinitialized precisely with a wide-baseline method. This restricts their usage to video streams.

Category	Method	Registration	Real-time	Wide-baseline	General geometry	Needless of training for new objects	Public access code
G1	Salzmann et al. [13]	×	NA	✓	✓	✓	×
	Brunet et al. [18]	×	×	✓	✓	✓	✓
	Bartoli et al. [15]	×	NA	✓	✓	✓	✓
	Ozgur et Bartoli [19]	×	×	✓	✓	✓	×
	Famouri et al. [17]	×	✓	✓	✓	✓	✓
	Aranda et al. [10]	×	✓	✓	✓	✓	×
G2	Ostlund et al. [21]	✓	✓	×	✓	✓	×
	Ngo et al. [22]	✓	✓	×	✓	✓	×
	Collins and Bartoli [23]	✓	✓	×	✓	✓	×
	Collins et al. [6]	✓	✓	×	✓	✓	×
G3	Pumarola et al. [25]	✓	×	✓	×	×	×
	Golyanik et al. [26]	✓	✓	✓	×	×	×
	Fuentes-Jimenez et al. [28]	✓	✓	✓	✓	×	×
	Shimada et al. [27]	✓	✓	✓	×	×	×
	Fuentes-Jimenez et al. [29]	✓	✓	✓	×	✓	×
	ROBUSfT	✓	✓	✓	✓	✓	✓

Table 1: Comparison of the existing SfT methods and ROBUSfT.

2.3. (G3) DNN-based methods

DNN-based SfT methods have been introduced in recent years. This coincides with the general trend to use deep learning to solve many computer vision problems. These methods are wide-baseline, fast, and cover both the registration and shape inference steps [25–29]. We classify these methods in two groups based on their type of output, which may be sparse or dense. The methods of the first group represent the SfT solution as the 3D coordinates of a regular mesh with a predefined size [25–27]. The usage of these methods is limited to thin-shell objects with rectangular shapes. The second group of methods gives a pixel-level depth map as output [28, 29]. They also apply a post-processing step based on the As-rigid-as-possible (ARAP) model [34] to the resulting depth map. This step recovers the whole object, including the occluded parts, as a mesh. The method in [28] reconstructs the shape of the object with different geometries and texture maps that the network is trained for. In [29], the proposed method can be applied to objects with new texture maps previously unseen by the network. The geometry of the objects is, nevertheless, limited to flat paper-like shapes. All the aforementioned methods in this category are object-specific. This means that they merely work for the object that they were trained for. An exception is [29], as it works for unseen texture maps but the applicability is still limited to flat rectangular objects. Therefore, in order to use a DNN-based method for a new object, the network should be fine-tuned for it. This demands proper computational resources and potentially a huge amount of training data, which are challenging to collect for deformable objects.

Lastly, apart from DNN-based SfT methods, there also exist deep object-generic monocular reconstruction methods which work by estimating pixel-wise depth [35–39]. The DNNs in these methods are trained on a diverse range of common objects, which eliminates the need for specific training for each new object. However, when dealing with deformable objects, their efficiency is limited and was shown to be subpar compared to SfT methods [28].

2.4. Positioning ROBUSfT compared to previous work

Existing methods all have one or several limitations, including not covering the whole pipeline, not being wide-baseline, being limited to a specific texture or geometry, requiring fine-tuning for a new object, being slow, and lacking public code access. This information is summarized in Table 1. In contrast, ROBUSfT covers the whole pipeline and runs fast. It can thus be used to develop applications requiring real-time 3D shape tracking. It can be instantly used for a new deformable object without training. Only a template containing information regarding the object's geometry, appearance and deformation law, as well as the intrinsic parameters of the monocular camera, are necessary. This is common to all existing and future SfT methods, by definition. In the next section, we describe `ROBUSfT` and all its steps.

3. ROBUSfT

3.1. Overview of the pipeline

The overview of our pipeline is presented in Figure 1. The pipeline is divided into an offline section and an online section. The offline section deals with the template. The online section includes four main steps: keypoint extraction and matching, mismatch removal, warp estimation, and 3D shape inference. The images coming directly from the camera are used as the inputs for the first step. In this step, the keypoints are extracted and matched with the ones that were previously extracted from the template's texture map. Then, the mismatches are detected and removed using our new mismatch removal algorithm `myNeighbor`. The list of estimated correct matches is then transferred to the next step where a warp is estimated between the template's texture map and the image. This warp transfers the template's registered mesh to the image space, which is finally used as input for the 3D shape inference algorithm. This process is repeated for each image. Since each image is analyzed independently, repeating the process in a loop enables tracking-by-detection.

In the following, both the offline and online sections of the pipeline are described in detail. Afterwards, an implementation permitting a fast execution of the pipeline is given.

3.2. Offline section: creating a template

We create a template for the surface of the deformable object that we want to track. We call this surface the tracking surface. The template of the tracking surface consists of the following elements:

- M_T: the triangular mesh covering the tracking surface at its rest shape.
- $\mathcal{P}$: the texture map of the tracking surface.
- M: the alignment of M_T to $\mathcal{P}$.

The first step in creating the template is to generate the 3D model of the tracking surface. The 3D model is merely the textured 3D geometry of the tracking surface in real dimensions in its rest shape. Once this 3D geometry is generated, we form M_T by triangulating it. The resolution of M_T should be high enough to be well aligned to the shape of the tracking surface. The next step is to take an image from the 3D model of the tracking surface while it is positioned perpendicular to the camera's optical axis in a simple texture-less background. In this image, $\mathcal{P}$ is formed by the projection of the texture of the tracking surface and M is formed by the projection of M_T. For simple rectangular thin-shell objects like a piece of paper, the whole process is straightforward. For other objects, including thin-shell objects with arbitrary shape, such as a shoe sole, and also volumetric objects, 3D reconstruction software like Agisoft Photoscan [40] can be used.

Next, we extract keypoints on $\mathcal{P}$. These keypoints will be matched with the ones that will be extracted from the input image in the online section. We use SIFT [41] for extracting keypoints but any other feature descriptor could be swapped in. As the final step, we initialize the pose of M_T in 3D space. This initial pose can be arbitrarily chosen as it will be used only once in Step 4 of the online section of the pipeline for the first input image. It will then be replaced by the inferred 3D shape in the next images.

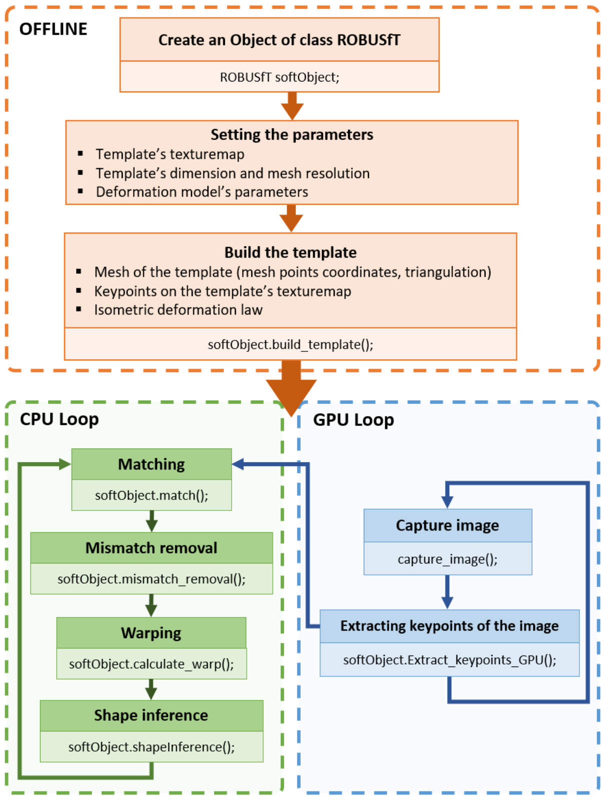
In order to use the `ROBUSfT` C++ library, first, an object of the class `ROBUSfT` should be created. The whole process of forming the template for this object is handled by the member function `build_template`(). This function possesses parameters for creating templates for rectangular and non-rectangular thin-shell objects as well as the tracking surface of volumetric objects. For thin-shell objects, the process of forming the template is automatic by just receiving a handful of inputs from the user. For the tracking surface of volumetric objects, however, M_T, M, and $\mathcal{P}$ should be prepared by the user and imported into the library.

3.3. Online section: shape tracking

Step 1: keypoint extraction and matching. The first step of the online section of the pipeline is to extract keypoints in the input image $\mathcal{I}$. To do so, we use the PopSift library [42], which is a GPU implementation of the SIFT algorithm. We then match these keypoints with the ones that were previously extracted from $\mathcal{P}$ by comparing descriptors, using winner-takes-all and Lowe's ratio test. Inevitably, a number of mismatches will be formed between $\mathcal{P}$ and $\mathcal{I}$. The mismatch points in $\mathcal{I}$ can be located on the surface of the deforming object or even in the background. This is shown as red lines in the *Matching* step of Figure 1. These mismatches will be eliminated in *Step 2* thanks to `myNeighbor` which can cope with a large rate of mismatches. As a result, in this step, the images coming from the camera can be used directly without pretraining on either the image, for segmenting the object from the background, or the matches, for preselection of the most reliable ones. In the library, the member function `extract_keypoints_GPU`() handles the keypoint extraction in $\mathcal{I}$. Then, the member function `match`() performs matching.

Step 2: mismatch removal. To remove the possible mismatches introduced in *Step 1*, a new mismatch removal algorithm, `myNeighbor`, was developed. The main principle used in this algorithm is the preservation of the neighborhood structure of correct matches on a deforming object. In other words, if all of the matches were correct, by deforming the object, the neighboring matches of each match should be preserved. On the contrary, mismatches lead to differences in the neighboring matches of each matched point in $\mathcal{I}$ in comparison to $\mathcal{P}$. This was used as a key indication to detect and remove mismatches. The whole process of `myNeighbor` is explained in Section V. In the library, the member function `mismatch_removal_algorithm`() handles the mismatch removal process. The output is a list of estimated correct matches.

Step 3: warp estimation. We use the estimated correct matches to estimate a warp W between $\mathcal{P}$ and $\mathcal{I}$. We then use W to transfer M to $\mathcal{I}$ and form $\widehat{M}$. The mesh points in $\widehat{M}$ will be used as sightline constraints in the 3D shape inference algorithm in *Step 4*. The precision of warping depends on the number of matches, their correctness, and their distribution over $\mathcal{P}$. Warp W can be estimated in the most precise way if all the matches are correct between $\mathcal{P}$ and $\mathcal{I}$. However, due to the smoothing nature of the warping algorithms, the transferring process can cope with a small percentage of mistakenly selected mismatches. It should be noted that W cannot be extremely precise in areas without matches. As a result, in these areas, the shape of $\widehat{M}$ might not be aligned well to the shape of the deforming object in $\mathcal{I}$. This is worse when the matchless area is located near the boundaries of $\mathcal{P}$ as the alignment cannot be guided by the surrounding matches. Hence, in order to use just well-aligned transferred mesh points of $\widehat{M}$ as the input for the 3D shape inference step, an assessment is performed over all of the mesh points and only the qualified ones are passed to *Step 4*. For this, we check M cell-by-cell. Only the mesh vertices for cells containing at least one correct match will be qualified as salient mesh points. The indices of these mesh points and their coordinates in $\widehat{M}$ are passed to *Step 4*. The other mesh points are disregarded.



OFFLINE
Create an Object of class ROBUSfT
ROBUSfT softObject;
Setting the parameters
Template's texturemap
Template's dimension and mesh resolution
Deformation model's parameters
Build the template
Mesh of the template (mesh points coordinates, triangulation)
Keypoints on the template's texturemap
Isometric deformation law
softObject.build_template();
CPU Loop
Matching
softObject.match();
Mismatch removal
softObject.mismatch_removal();
Warping
softObject.calculate_warp();
Shape inference
softObject.shapeInference();
GPU Loop
Capture image
capture_image();
Extracting keypoints of the image
softObject.Extract_keypoints_GPU();

Figure 2: Implementation of `ROBUSfT` distributed on the CPU and GPU. A pure CPU implementation is also available.

Representing and estimating W can be done with two well-known types of warp, the Thin-Plate Spline

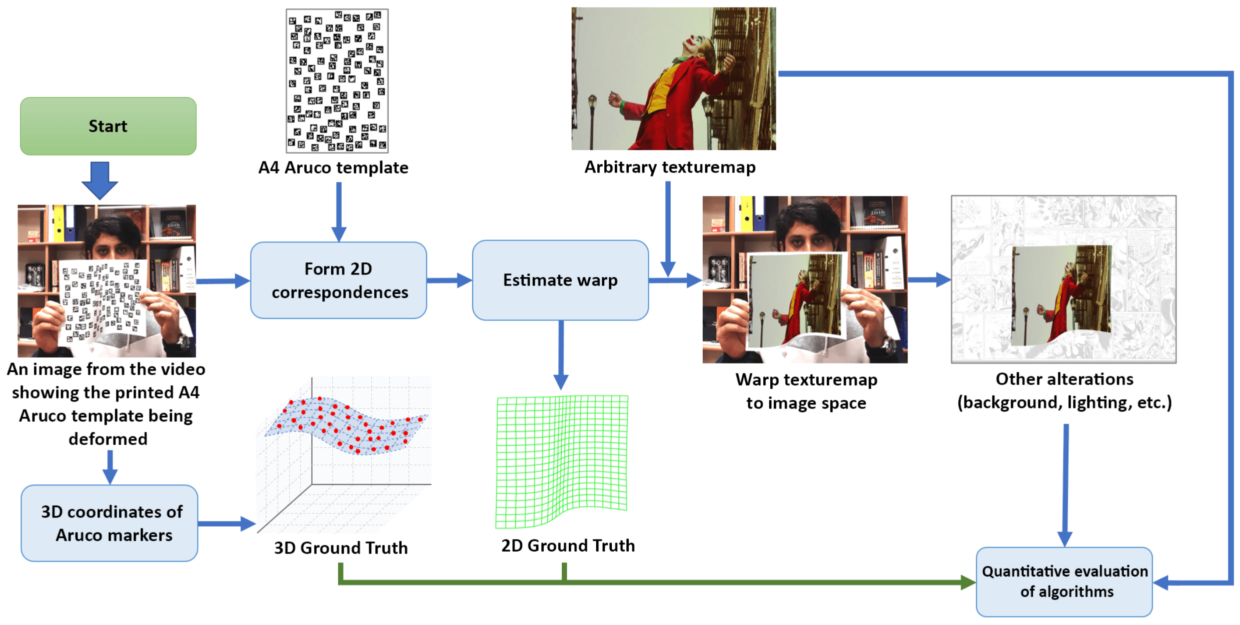

Start
A4 Aruco template
Arbitrary texturemap
Form 2D correspondences
Estimate warp
An image from the video showing the printed A4 Aruco template being deformed
Warp texturemap to image space
Other alterations (background, lighting, etc.)
3D coordinates of Aruco markers
3D Ground Truth
2D Ground Truth
Quantitative evaluation of algorithms

Figure 3: Flowchart of `FREX`, the proposed experimental protocol.

(TPS) [43] and the Bicubic B-Spline (BBS) warps [44], which we both tested. The former is based on radial basis functions while the latter is formulated on the tensor product. Having the same number of matches as input, the TPS warp proved to be more precise than the BBS warp; nevertheless, its execution time rises exponentially with the number of matches. The execution time, however, remains almost constant for the BBS warp regardless of the number of matches. Thus, considering the criterion of fast execution of the code, the BBS warp was chosen as the warp function in this step and also in the mismatch removal step discussed in Section V. In the library, the process of warp estimation is performed by the function `warp`() that calls two functions: `BBS_Function`() and `BBS_Evaluation`(). The former estimates the warp W while the latter uses W to transfer M and form $\widehat{M}$. The process of selecting the salient mesh points is done by the member function `set_sightlines`().

Step 4: 3D shape inference. We use Particle-SfT [19] with the improvements for tracking proposed in [10]. In this algorithm, a particle system is defined from the points and edges in M_T. Then, the sightline and deformation constraints are applied consecutively on the particles until they converge to a stable 3D shape. As described in [10], in order to increase the convergence speed of the algorithm, the stable 3D shape for an image is used as initial guess for the next image. It should be noted that Particle-SfT can work even without a close initial guess. If the object is invisible in one or several images, the last inferred 3D shape can be used as the initial guess for the upcoming frame containing the object. This results in a slightly longer computation time in that image. For the next upcoming images the normal computation time is resumed. This capability brings about two of the major advantages of our pipeline, which are being wide-baseline and robust to video discontinuities. In the library, the whole process of shape inference is handled by the member function `shapeInference`().

As mentioned in [10], one of the optional input data that can significantly improve the convergence of Particle-SfT is the existence of known 3D coordinates for one or several particles. This is shown in Figure 1. The known 3D coordinates can be fixed in space or move on a certain trajectory. The latter happens when the deforming object is manipulated by tools with known poses in 3D space such as robotic end-effectors.

3.4. Implementation

In order to optimize the implementation of `ROBUSfT`, it was coded in C++ in two parallel loops: one on the GPU, and one on the CPU. The GPU loop handles keypoint extraction in the images. These keypoints are transferred to the CPU loop where the rest of the steps of the pipeline are executed. A pure CPU implementation is also available. This is shown in Figure 2. Any arbitrary resolution can be considered for the captured images; nevertheless, we obtained the best performance by using 640×480 images. The code runs on a Dell laptop with an Intel Core i7 2.60 GHz CPU and a Quadro T1000 GPU.

4. Fake but Realistic Experiment (`FREX`)

We introduce `FREX`, a novel experimental protocol, which we used for evaluating `myNeighbor` and `ROBUSfT` in comparison to existing methods. A single execution of this protocol provides a large collection of scenes of an isometrically deforming object in various conditions, with known 2D and 3D ground truth. This collection can be used to evaluate, compare, train, and validate new algorithms regarding isometrically deforming objects for tasks such as mismatch removal, 2D image registration, and isometric 3D shape inference. In contrast to other artificially generated scenes of an isometrically deforming surface, the generated images in our protocol are the result of real object deformations. Since it generates successive images with continuous deformation, `FREX` can also be used for algorithms which exploit feature and shape tracking. In addition, object occlusion and invisibility can be easily simulated, by dropping frames or overlaying an occluder.

The protocol flowchart is shown in Figure 3. First, we form the *Aruco template* by randomly distributing a set of Aruco markers all over a blank image. We then print the Aruco template on a standard A4 paper. These markers should be big enough to be recognizable by the user's camera at the desired distance. In order to improve recognition, there should be white space between the markers on the paper. In our experiments, we used 100 markers with a width of 1.4 cm each. The OpenCV library was used to identify the markers. These markers were recognizable by a 720p RGB camera from an approximate distance of 0.6 m. The next step is to deform the printed Aruco template in front of the camera. In each frame, the 2D and 3D coordinates of the markers' centers are estimated. Because each marker has its own unique ID, they can be used as correspondences between the Aruco template and each image of the video. We exploit the 2D coordinates of these recognized correspondences to estimate a warp with which we can transfer an arbitrary texture map to the video image space. This is done firstly by resizing the arbitrary texture to the size of the Aruco template. In order to keep the aspect ratio of the arbitrary texture map, white margins can also be added before resizing. Then, an inverse warping process with bilinear interpolation is used to transfer the pixel color information from the arbitrary texture map to the corresponding pixels in the video images. The whole procedure results in a scene with the arbitrary texture map being deformed exactly on top of the Aruco template. It is also possible to add further modifications; for instance, one can transfer the arbitrary texture map to another scene with any different background. Besides, as in [45], artificial lighting can also be added to form different variations of the scene.

For evaluating algorithms, one can use the 2D and 3D ground truth estimated in each frame of the video. Regarding the 2D ground truth, the estimated warp can be used to identify the 2D corresponding point of each pixel of the arbitrary texture map in the image. As for the 3D ground truth, one can exploit the 3D estimated coordinates of the Aruco markers in each frame, which can be obtained using the OpenCV library.

5. `myNeighbor`

We describe `myNeighbor`, our novel mismatch removal algorithm. It works based on two main principles:

- Given a sample set of correct matches between an image of a textured surface and another image of that surface undergoing a deformation, one can estimate a sufficiently accurate transfer function between the images such that the correctness of all the matches can be judged. Consequently, there is no need to remove all the mismatches.
- This sample set of correct matches can be extracted from the images considering that the neighborhood

structure among the points on a deforming surface is preserved.

We show that by using these two principles, the mismatches can be detected and removed in a fast and efficient way. The proposed algorithm is illustrated in Figure 4. It consists of three steps. First, a set of matches, which is highly probable to include only correct matches, is selected. This selection is done by forming two triangulations using match points, one in $\mathcal{P}$ and one in $\mathcal{I}$, and then choosing matches with high similarity in the list of their neighbors. Second, a small percentage of possible mismatches among the selected matches is identified and removed. This is done by transferring the selected match points from $\mathcal{P}$ to $\mathcal{I}$ and then removing those with large distances from their correspondences in $\mathcal{I}$. Third, all the match points are transferred from $\mathcal{P}$ to $\mathcal{I}$ using a warp estimated based on the clean set of selected matches from the second step. The distance between the transferred template match points and their correspondences in $\mathcal{I}$ is used as the criterion to distinguish estimated mismatches from estimated correct matches.

In order to analyze the performance of `myNeighbor` and calibrate the parameters in the different steps, we used synthetic data experiments. In the following section, we describe the design of these experiments. Afterwards, we describe in detail the different steps of `myNeighbor`.

5.1. Empirical parameter calibration

These experiments are conducted by synthetically forming two images of a mesh M_T and a series of matches between the two images. The first image shows M_T in its flat rest shape with all its keypoints on it. We call this image $\mathcal{I}_F$. In $\mathcal{I}_F$, the keypoints can be considered as the extracted keypoints from $\mathcal{P}$, and the 2D mesh is equivalent to M. The second image simulates $\mathcal{I}$ and shows M_T having undergone a random 3D deformation. We call this deformed mesh M_G. The keypoints in this image can be positioned in their correct locations on the mesh (correct matches) or be displaced in the image area (mismatches).

We consider M_T as a regular triangular mesh with 10×6 points in 3D space. In order to deform M_T, we use the same method as in [10]. Concretely, we apply two 3D deformations containing random translations and rotations to two mesh cells at both sides of M_T. The deformation is calculated in an iterative process based on position-based

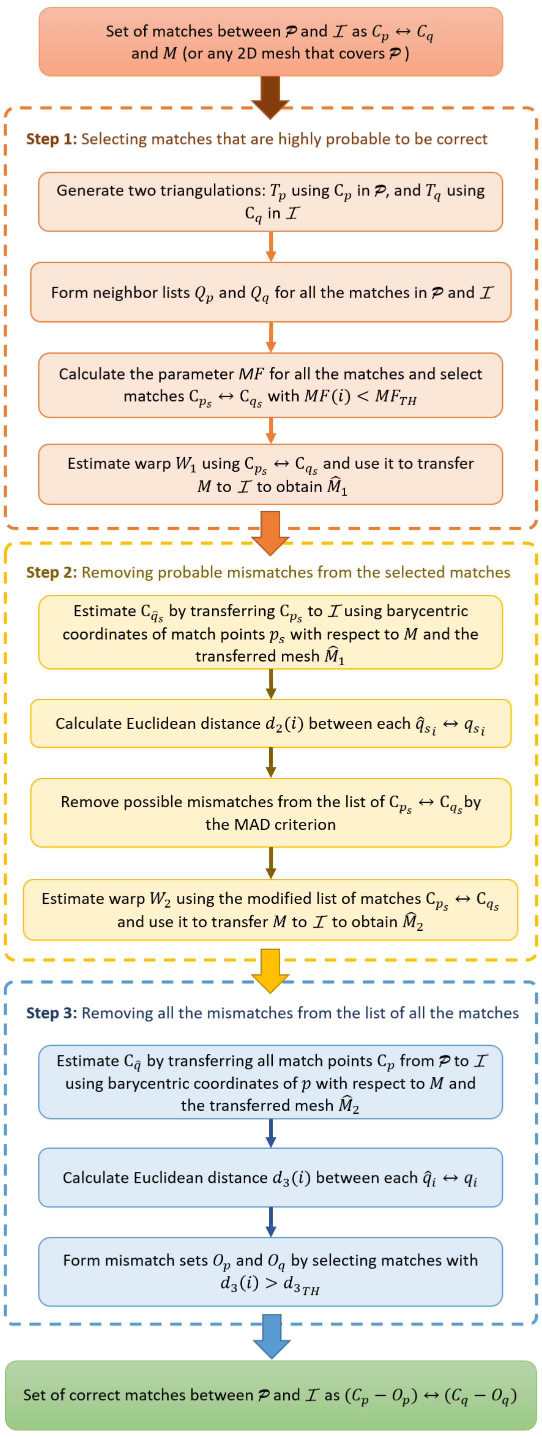

Set of matches between $\mathcal{P}$ and $\mathcal{I}$ as $C_p \leftrightarrow C_q$ and M (or any 2D mesh that covers $\mathcal{P}$)
Step 1: Selecting matches that are highly probable to be correct
Generate two triangulations: T_p using C_p in $\mathcal{P}$, and T_q using C_q in $\mathcal{I}$
Form neighbor lists Q_p and Q_q for all the matches in $\mathcal{P}$ and $\mathcal{I}$
Calculate the parameter MF for all the matches and select matches $C_{p_s} \leftrightarrow C_{q_s}$ with $MF(i) < MF_{TH}$
Estimate warp W_1 using $C_{p_s} \leftrightarrow C_{q_s}$ and use it to transfer M to $\mathcal{I}$ to obtain $\widehat{M}_1$
Step 2: Removing probable mismatches from the selected matches
Estimate $C_{\hat{q}_s}$ by transferring C_{p_s} to $\mathcal{I}$ using barycentric coordinates of match points p_s with respect to M and the transferred mesh $\widehat{M}_1$
Calculate Euclidean distance $d_2(i)$ between each $\hat{q}_{s_i} \leftrightarrow q_{s_i}$
Remove possible mismatches from the list of $C_{p_s} \leftrightarrow C_{q_s}$ by the MAD criterion
Estimate warp W_2 using the modified list of matches $C_{p_s} \leftrightarrow C_{q_s}$ and use it to transfer M to $\mathcal{I}$ to obtain $\widehat{M}_2$
Step 3: Removing all the mismatches from the list of all the matches
Estimate $C_{\hat{q}}$ by transferring all match points C_p from $\mathcal{P}$ to $\mathcal{I}$ using barycentric coordinates of p with respect to M and the transferred mesh $\widehat{M}_2$
Calculate Euclidean distance $d_3(i)$ between each $\hat{q}_i \leftrightarrow q_i$
Form mismatch sets O_p and O_q by selecting matches with $d_3(i) > d_{3TH}$
Set of correct matches between $\mathcal{P}$ and $\mathcal{I}$ as $(C_p - O_p) \leftrightarrow (C_q - O_q)$

Figure 4: Flowchart of `myNeighbor`.

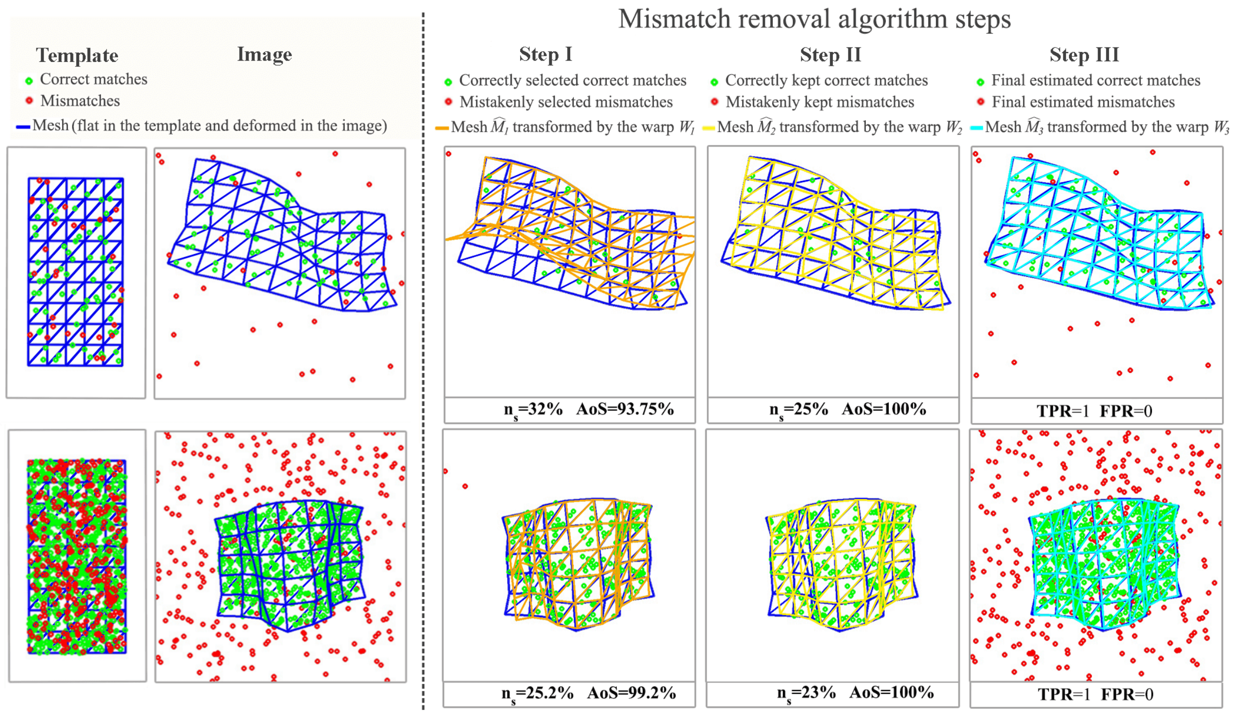

Mismatch removal algorithm steps
Template
Image
Correct matches
Mismatches
Mesh (flat in the template and deformed in the image)
Step I
Correctly selected correct matches
Mistakenly selected mismatches
Mesh $\widehat{M}_1$ transformed by the warp W_1
Step II
Correctly kept correct matches
Mistakenly kept mismatches
Mesh $\widehat{M}_2$ transformed by the warp W_2
Step III
Final estimated correct matches
Final estimated mismatches
Mesh $\widehat{M}_3$ transformed by the warp W_3
n_s=32% AoS=93.75%
n_s=25% AoS=100%
TPR=1 FPR=0
n_s=25.2% AoS=99.2%
n_s=23% AoS=100%
TPR=1 FPR=0

Figure 5: Two sample results of the steps for empirical parameter calibration. The first row is an experiment with 100 matches and a mismatch percentage of 30%. The second row is an experiment with 1000 matches and a mismatch percentage of 30%. The first and second columns represent $\mathcal{I}_F$ and $\mathcal{I}$ with correct matches in green and mismatches in red. The third column is the result of *Step I*. The wrongly chosen mismatches are shown in red. The fourth column is the result of *Step II*. The mismatches along with a small percentage of correct matches are removed. The fifth column is the separation of the estimated correct matches and the estimated mismatches from *Step III*. The transferred meshes $\widehat{M}_1$, $\widehat{M}_2$, and $\widehat{M}_3$ are shown in orange, yellow, and cyan for the three steps.

dynamics [46, 47]. As for generating keypoints, we first randomly place keypoints in the inner area of M in $\mathcal{I}_F$. In order to create the matches between $\mathcal{I}_F$ and $\mathcal{I}$, we then transfer the keypoints from $\mathcal{I}_F$ to $\mathcal{I}$ using a three-step process: calculating barycentric coordinates of the keypoints in M, transferring the keypoints to the 3D deformed mesh using the barycentric coordinates and the new 3D mesh points of the deformed M_T, and eventually projecting the transferred keypoints on $\mathcal{I}$. To generate mismatches, an arbitrary percentage of the transferred keypoints is corrupted by randomly distributing the keypoints all over the area of $\mathcal{I}$. Two samples of the generated images for 100 and 1000 matches each with 30% mismatches are shown in the two first columns of Figure 5.

5.2. *Methodology*

The algorithm `myNeighbor` is applied on N_m matches denoted as $C_p \leftrightarrow C_q$ between $\mathcal{P}$ and $\mathcal{I}$, with:

$$C_p = \{p_1, ..., p_{N_m}\}, \quad p_i = (x_i, y_i) \tag{1}$$

$$C_q = \{q_1, ..., q_{N_m}\}, \quad q_i = (u_i, v_i) \tag{2}$$

A pair (p_i, q_i) of points with the same index forms a match $p_i \leftrightarrow q_i$. We define the set of correct matches S_{in} as the collection of matches $p_i \leftrightarrow q_i$ where p_i and q_i point to the same location on the deforming surface in $\mathcal{P}$ and $\mathcal{I}$. On the contrary, when the pointing locations of the match points are different, they are categorized as mismatches S_{out}. The goal of `myNeighbor` is to form and remove the subsets $O_p \subset C_p$ and $O_q \subset C_q$ which have the largest possible number of matches belonging to S_{out} and smallest

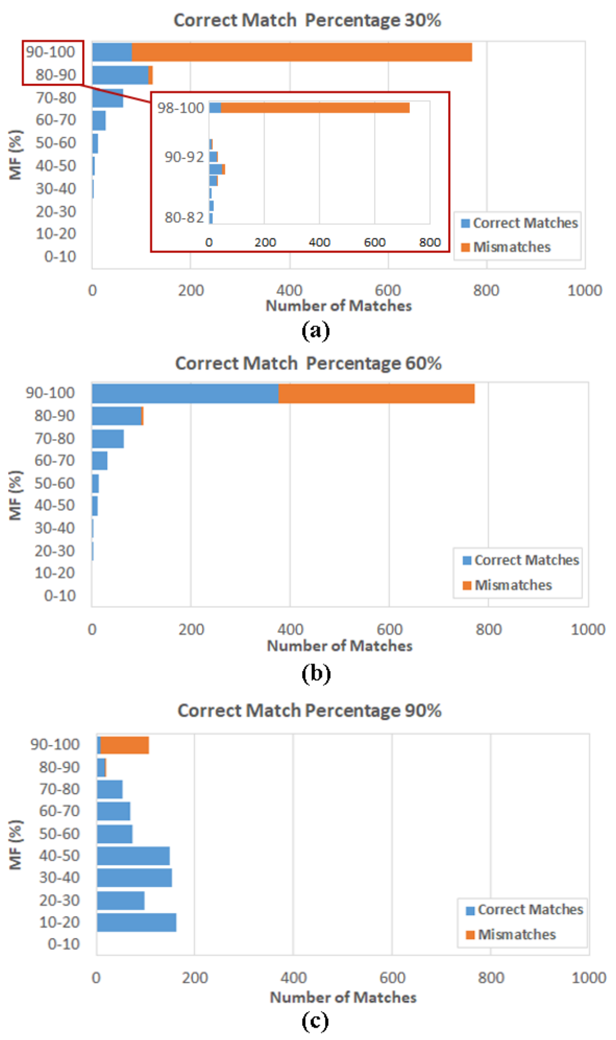


Figure 6: Histogram of MF values for three sample synthetic data experiments with 1000 matches and 30%, 60% and 90% of correct matches.

possible number of matches belonging to S_{in}. We explain the steps of our algorithm to fulfill this goal.

5.2.1. *Step I – Neighbor-based correct match selection*

We select subsets $C_{p_s} \subset C_p$ and $C_{q_s} \subset C_q$ which are highly probable to form correct matches. We start by defining W_G as the ground-truth warp between $\mathcal{P}$ and $\mathcal{I}$ that can transfer all the match points C_p from $\mathcal{P}$ to their correct locations in $\mathcal{I}$. With this definition, we have the set of correct matches S_{in} as:

$$S_{in} = \{(p_i, q_i) \mid i \in R\}, \tag{3}$$

where:

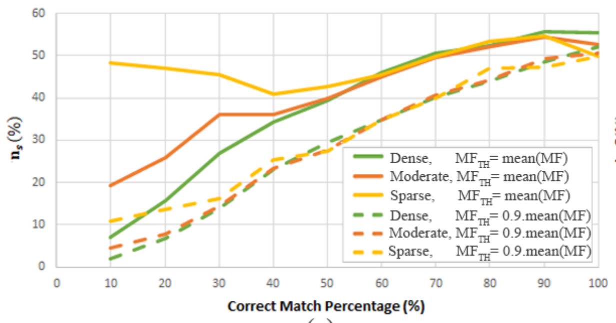
$$R = \{i \mid \|W_G(p_i) - q_i\| < \epsilon\}, \tag{4}$$

where ϵ is a small positive number. Warp W_G is an unknown composition of an isometric deformation mapping and a perspective projection mapping. The isometric deformation mapping preserves the geodesic distances among the points and their topological structure on the object's surface. However, with the addition of the perspective projection mapping, only the topological structure of the points remains preserved in the visible areas. This implies that by applying W_G, the neighborhood structure among the points on the object in $\mathcal{P}$ and $\mathcal{I}$ should be preserved. We exploit this characteristic of W_G to estimate $\widehat{R}$ as the set of indices of highly probably correct matches $C_{p_s} \leftrightarrow C_{q_s}$. To do so, first, we form two Delaunay triangulations, $T_p = D(C_p)$ in $\mathcal{P}$, and $T_q = D(C_q)$ in $\mathcal{I}$. Then, for each match i, we calculate two sets of first-order neighbors $Q_p(i)$ and $Q_q(i)$ in $\mathcal{P}$ and $\mathcal{I}$, respectively. We then define the *Mismatch Factor* (MF) criterion for match i as:

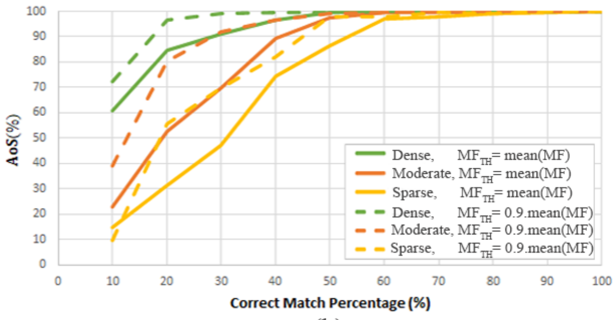
$$MF(i) = \frac{|Q_p(i) \cup Q_q(i) - Q_p(i) \cap Q_q(i)|}{|Q_p(i) \cup Q_q(i)|} \times 100 \tag{5}$$

For each match, MF measures the difference in the neighbor points between $\mathcal{P}$ and $\mathcal{I}$ as a percentage. Ideally, we expect that for all the matches $MF = 0$, which implies that there is no difference in the neighbors of each match during a deformation. However, in practice, there are two reasons which rather put MF values in a range from 0 to 100: the presence of mismatches and variations in triangulation. The presence of mismatches can affect the value of MF in two ways. First, when the match point i in $\mathcal{I}$ is a mismatch and thus located in a wrong location. And second, when the match point i in $\mathcal{I}$ is a correct match but one, several, or all of its neighbors are mismatches. Both of these cases result in different neighbors in $\mathcal{I}$ in comparison to $\mathcal{P}$. As for the two triangulations, it should be noted that even in the absence of mismatches, the neighborhood structures in T_p and T_q do not necessarily coincide. This is because of surface deformation, change in viewpoint, and occlusions.

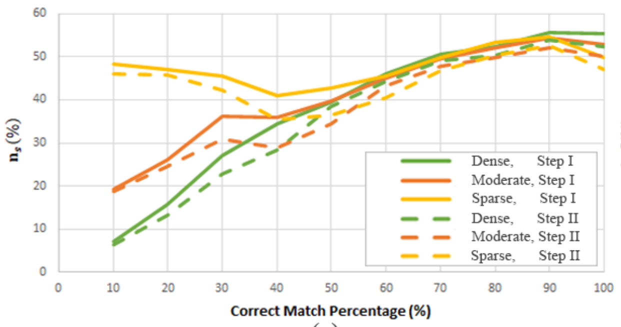
Calculating MF for all the matches, we can have a fair estimation regarding the state of the matches. A lower



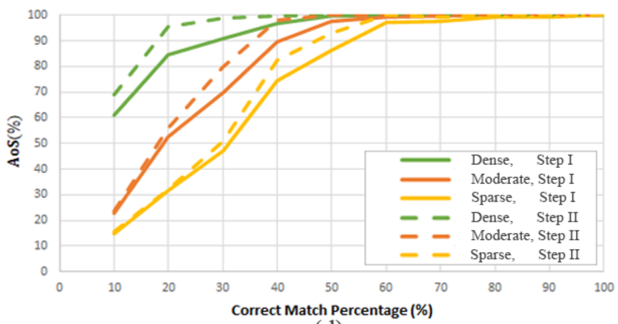
(a)



(b)



(c)



(d)

Figure 7: Results of applying the first two steps of the algorithm `myNeighbor` in synthetic data experiments in three different scenarios: Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each curve is the average result of 1000 trials. The first row gives n_s and AoS from *Step I* for two different values of MF_{TH}. The second row gives the results of *Step II* in comparison to the results of *Step I* with MF_{th} = mean(MF).

value of $MF(i)$ indicates that the match i is surrounded by similar matches in $\mathcal{P}$ and $\mathcal{I}$ and has a higher probability of being placed in its correct location and thus being a correct match. On the contrary, a higher value of $MF(i)$ can stem from the wrong location of the match i in comparison to its neighbors, which strengthens the possibility of it being a mismatch. The basic idea in this step is to form $C_{p_s} \leftrightarrow C_{q_s}$ by selecting pairs of highly probably correct matches $p_s \leftrightarrow q_s$. This is done by choosing the matches with lower values of MF. We examined the validity of this reasoning by evaluating three different synthetic data experiments, each with 1000 matches and different rates of correct matches (30%, 60%, and 90%). Figure 6 shows the histogram of MF for each case. We observe that the dispersion of MF spans a wider range as the value of the correct match rate grows. For higher numbers of correct matches, there are more similarities in the neighbor lists of each match and, consequently, MF decreases. Furthermore, regardless of the values of the correct match rate, the majority of the mismatches are accumulated in the top bins of the graphs, which correspond to higher values of MF. This is shown in more detail for the case with the correct match percentage of 30% by expanding the last two bins of the graph in Figure 6.a. This validates our prior reasoning that by selecting the matches with MF below a certain threshold MF_{th}, we can have a set of matches that is highly probable to include the correct matches. To quantify the appropriateness of this selection, we define two criteria, based on the following two quantities. The first quantity is n_s, which is the percentage of the selected matches compared to the total number of matches:

$$n_s = \frac{|C_s|}{N_m} \times 100, \tag{6}$$

where $C_s = \{(p_i, q_i) \mid i \in \widehat{R}\}$ is the set of selected matches. The second quantity is AoS, which is the Accuracy of Selection, defined as:

$$AoS = \frac{|C_s \cap S_{in}|}{|C_s|} \times 100. \quad (7)$$

Our goal is to choose the value of MF_{th} so that both of these quantities are as high as possible, which means selecting a high percentage of matches with high accuracy. However, practically, these two criteria work in reverse. By choosing a higher value for MF_{th}, more matches are selected (higher n_s) but with less accuracy (lower AoS) and vice versa. In order to choose the proper value for MF_{th}, we analyzed the behavior of these two criteria for a series of synthetic data experiments. We consider three scenarios for these experiments based on the number of matches, *i.e.* Dense, Moderate, and Sparse with in turn 1000, 200, and 50 total number of matches. The experiments were done in a wide range of correct match percentages (10% to 100%) for each scenario. Two different values of the criterion MF_{th} were studied: *mean* and $0.9 \times$ *mean*, where *mean* is the mean of all MF values in each experiment. The results are presented in Figures 7.a and 7.b. Each point in the graph is the average result of 1000 trials. The first point that should be noted here is that, generally, the proposed match selection method in this step is more reliable as the number of total matches grows. This can be deduced by comparing the higher values of AoS in the Dense case with the ones in the Moderate and Sparse cases. As for choosing MF_{th}, it should be noted that setting $MF_{th} = 0.9 \times mean$ leads to higher values of AoS in comparison to the case with $MF_{th} = mean$. Nevertheless, as shown in Figure 7.a, this sacrifices a high percentage of matches by reducing n_s significantly, which is undesirable. Hence, in this step, we choose *mean* as the value of MF_{th} and form $\widehat{R}$ as the set of indices of probably correct matches. While this choice implies a higher number of selected mismatches (lower AoS), we note that these mismatches can be removed in *Step II*.

As the final operation in this step, we estimate the warp W_1 between $\mathcal{P}$ and $\mathcal{I}$ using the selected matches $C_{p_s} \leftrightarrow C_{q_s}$. We then exploit this warp to transfer M to $\mathcal{I}$. We call this new mesh $\widehat{M}_1$. As can be seen in the third column of Figure 5, the mesh $\widehat{M}_1$ (shown in orange) may not be totally faithful to the deformation of M_G in $\mathcal{I}$, which is due to the inaccuracies in the calculation of the warp W_1. This stems from two main reasons: the existence of mismatches in the selection (shown as red dots), and the insufficient number of correct matches in some areas. In the next step, we exploit the transferred mesh $\widehat{M}_1$ to remove the possible remaining mismatches from the selected matches.

5.2.2. *Step II – Removing mismatches from the list of selected matches*

This step removes the possible mismatches from the selected matches $C_{p_s} \leftrightarrow C_{q_s}$. We first form the set $C_{\hat{q}_s}$ by transferring C_{p_s} to $\mathcal{I}$. This is done by finding the barycentric coordinates of each selected match $p_{s_i} \in C_{p_s}$ with respect to M and applying them on the transferred 2D mesh $\widehat{M}_1$ from *Step I*. We then use the following decision criterion to identify and remove possible mismatches one by one from the selected matches $C_{p_s} \leftrightarrow C_{q_s}$:

$$\left| d_2(i) - \text{median}(\{d_2(j)\}) \right| \geqslant 2.5\,\text{MAD}, \quad (8)$$

where $d_2(i) = \|\hat{q}_{s_i} - q_{s_i}\|$ with $i \in \widehat{R}$. MAD (Median of Absolute Deviations from Median) is calculated as:

$$\text{MAD} = k \ \text{median}\Big(\Big\{\Big| d_2(i) - \text{median}(\{d_2(j)\}) \Big|\Big\}\Big), \quad (9)$$

where $k = 1.4826$ is a constant number. The values of d_2 are relatively larger for mismatches in comparison to correct matches. This stems from two reasons. The first reason is the small percentage of mismatches compared to the great majority of correct matches coming from *Step I* and thus the smaller influence of mismatches in the estimation of warp W_1. The second reason is the inconsistent location of mismatches in $\mathcal{P}$ and $\mathcal{I}$. The decision criterion in equation (8) is chosen due to the distribution type of d_2, with the presence of just a small percentage of large values among the majority of small values. Figure 7.c and d illustrate the result of this step. As can be seen, unlike the previous strategy of choosing a smaller MF_{th}, this method results in improvement of AoS without losing a considerable percentage of selected matches. This can be clearly observed by comparing n_s in Figures 7.a and c.

As the last operation in this step, warp W_2 is calculated using the purified selected matches $C_{p_s} \leftrightarrow C_{q_s}$. This warp is then used to transfer M to the image space and form $\widehat{M}_2$. The result of removing possible mismatches in this step along with the transferred mesh $\widehat{M}_2$ is shown in the fourth column of Figure 5. As can be observed, in comparison to $\widehat{M}_1$, $\widehat{M}_2$ has a better compliance to M_G.

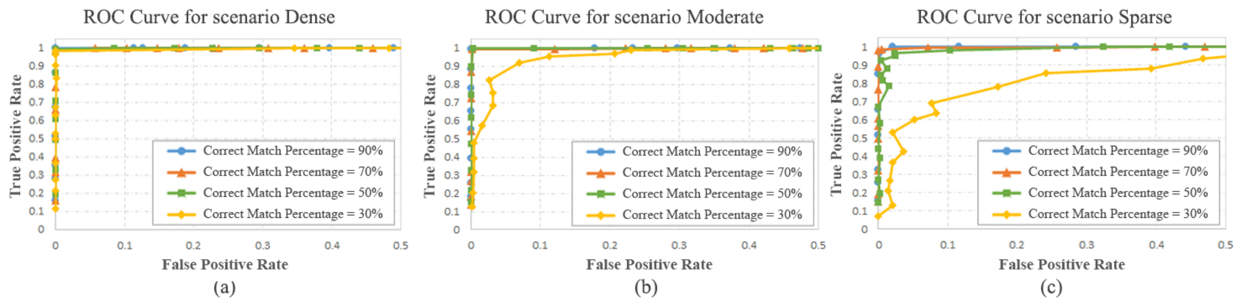


Figure 8: ROC curves resulting from the algorithm `myNeighbor` in synthetic data experiments in three scenarios: Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each point is the average result of 1000 trials calculated with a specific value of $d_{3_{th}}$.

5.2.3. *Step III – Extracting mismatches from the list of all the matches*

We exploit the transferred mesh $\widehat{M}_2$ to extract the mismatches $O_p \leftrightarrow O_q$ from the total matches $C_p \leftrightarrow C_q$. The process is similar to *Step II* except that this time all of the matches are checked. We first transfer the template match points C_p to the image space and form the set $C_{\hat{q}}$. This is done by calculating barycentric coordinates of all the match points C_p with respect to M and applying them on the new transferred mesh $\widehat{M}_2$. We define the following decision criterion to detect and remove mismatches:

$$d_3(i) = (\|\hat{q}_i - q_i\| \geqslant d_{3_{th}}) \tag{10}$$

Unlike *Step II* where we used the MAD criterion to remove just a small rate of mismatches, this time we use a constant threshold $d_{3_{th}}$. This is due to the higher percentage of mismatches compared to *Step II*. In order to make this distinction method more robust, we consider $d_{3_{th}}$ as the multiplication of a sample length l_s and a constant coefficient α_s. The sample length l_s is a measure of the size of the object in the image in pixels, and is calculated as the average distance between all the mesh points in the transferred mesh $\widehat{M}_2$. To choose a proper value for the constant coefficient α_s, synthetic data experiments with the same three scenarios as before (Dense, Moderate, and Sparse) and four different correct match rates were performed. The results are presented as ROC (Receiver Operating Characteristic) curves in Figure 8.a-c. Each point represents the average TPR (True Positive Rate) versus the average FPR (False Positive Rate) computed in 1000 trials using a specific value of α_s in the range [0, 1]. TPR is calculated as the number of selected true mismatches over the number of all true mismatches, and FPR is calculated as the number of true correct matches mistakenly selected as mismatches over the number of all true correct matches. Ideally, all the mismatches should be discarded (with a TPR of 100%) without discarding any correct matches (with an FPR of 0%). Hence, the most favorable α_s in a single ROC curve is the one that results in the maximum possible TPR leaving the FPR below a reasonable value. We choose $\alpha_s = 0.15$ which keeps TPR above 90% while FPR remains below 10% for most of the cases. The last column of Figure 5 illustrates the estimated correct matches (in green) and the estimated mismatches (in red) for each case. We also use the estimated correct matches to estimate warp W_3 and transfer M to $\mathcal{I}$ and form $\widehat{M}_3$ (shown in cyan). As can be seen, there is a high compliance between $\widehat{M}_3$ and M_G. It should be noted that estimating W_3 and $\widehat{M}_3$ is not necessary in `myNeighbor` and we merely estimate them to visually present the effectiveness of the algorithm in removing the mismatches. However, considering `myNeighbor` as a step in `ROBUSfT`, due to the fact that the final estimated correct matches are passed from this step to *Step 3* (warping) of `ROBUSfT`, W_3 and $\widehat{M}_3$ can also represent W and $\widehat{M}$ in the warping step, respectively.

5.3. *Mismatch removal results*

We study the efficiency of `myNeighbor` by evaluating its performance through various tests. We first compare the results of the algorithm with the existing ones by testing them through `FREX`. The experiment includes 60 frames of continuous deformation of the Aruco template in front of the camera. Five datasets were gen-

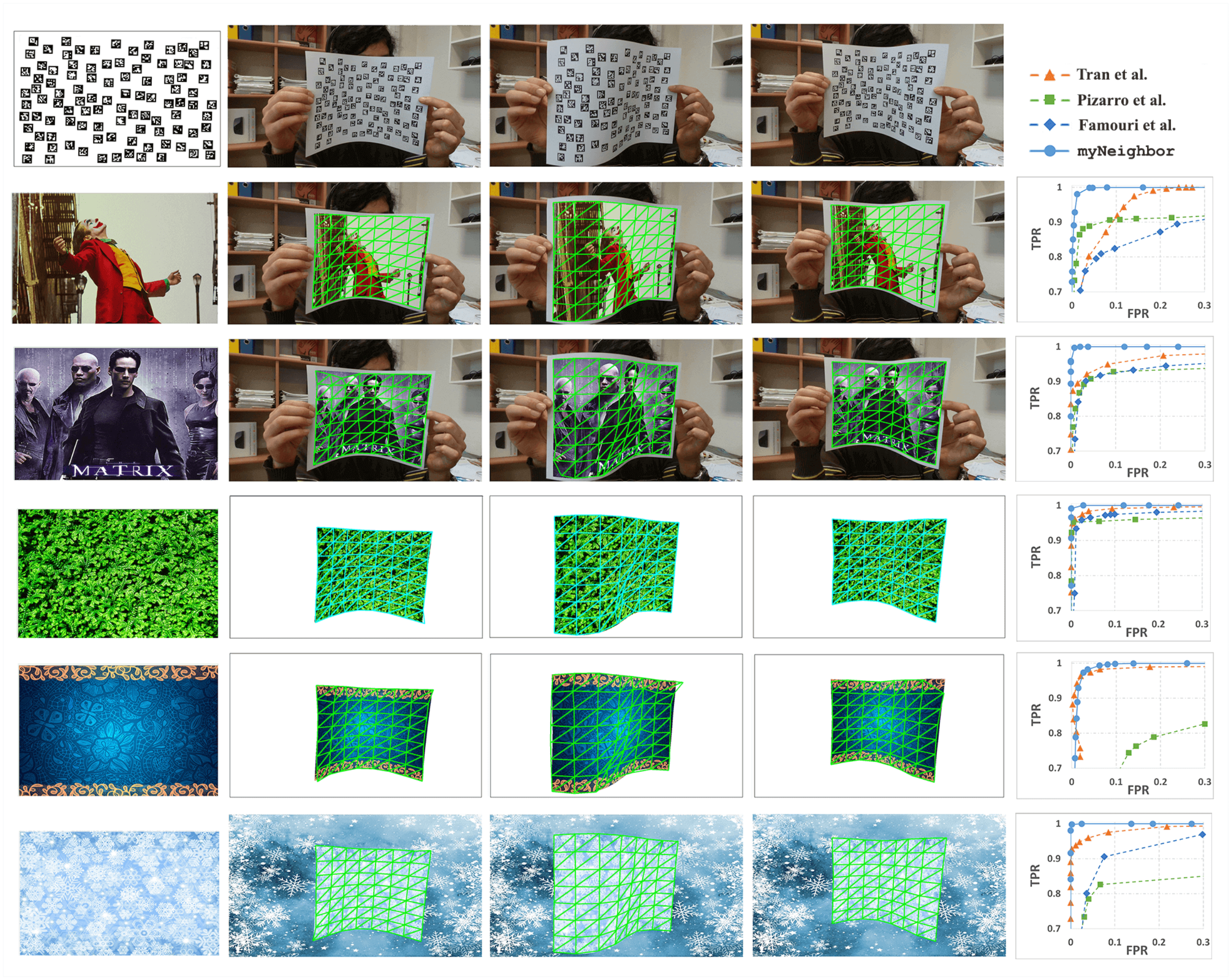


Figure 9: Performance evaluation of our mismatch removal method `myNeighbor` in comparison to the existing methods (Tran et al. [32], Pizarro et al. [30], Famouri et al. [17]) using the `FREX` protocol. The first row shows the Aruco template and three selected images (14, 47, 60) of the deformation of the printed Aruco template. The following rows show five datasets of generated scenes with the texture map in the first column, three generated images corresponding to the first row in the next columns, and the ROC curves of the mismatch removal algorithms in the last column. For each of the images $\widehat{M}_3$ from `myNeighbor` is overlaid.

erated in this experiment, each with an arbitrary texture with a challenging pattern. Three different types of backgrounds were also considered for these five cases; specifically, two original backgrounds, two white backgrounds, and a background with a pattern similar to one of the texture maps. We apply all the mismatch removal algorithms on all datasets. For each dataset, the corresponding arbitrary texture was used as the texture map for the mismatch removal algorithms. The matches between the texture map and each image of the dataset are extracted using SIFT. The results are presented in Figure 9. The first row illustrates the Aruco template and also three selected original images of its deformation in front of the camera. The lower rows represent the five datasets generated by `FREX`. Each row shows the arbitrary texture of the dataset in the first column, the three selected gener-

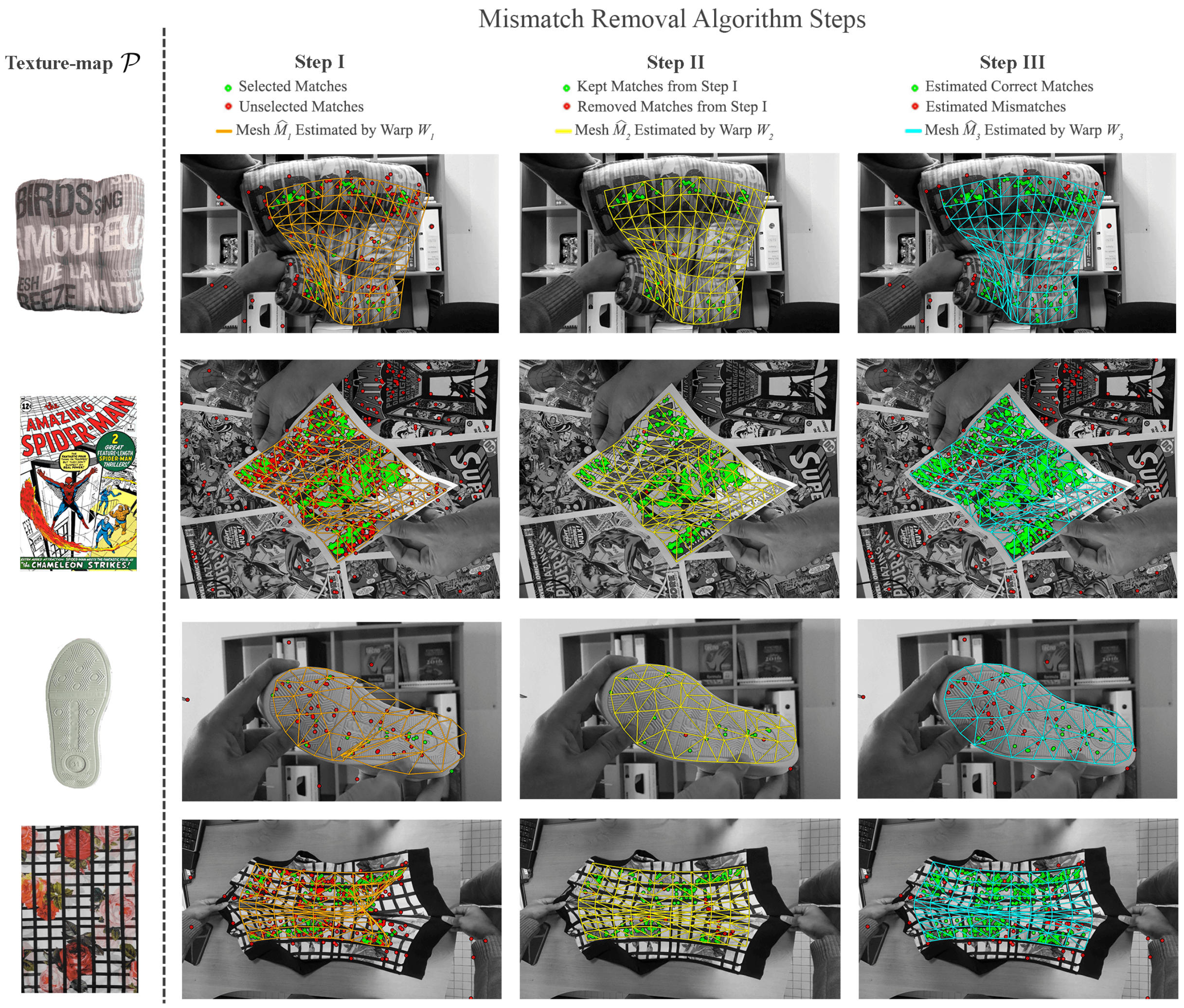

Mismatch Removal Algorithm Steps
Texture-map $\mathcal{P}$
Step I
Selected Matches
Unselected Matches
Mesh $\widehat{M}_1$ Estimated by Warp W_1
Step II
Kept Matches from Step I
Removed Matches from Step I
Mesh $\widehat{M}_2$ Estimated by Warp W_2
Step III
Estimated Correct Matches
Estimated Mismatches
Mesh $\widehat{M}_3$ Estimated by Warp W_3

Figure 10: Applying `myNeighbor` on four real cases: a cushion, a Spiderman poster, a shoe sole, and an elastic shirt. The first column shows the texture maps. The second column shows *Step I*. All the matches are shown in this column while the selected matches in *Step I* are shown in green. These selected matches are transferred to column three that shows *Step II*. In this column, those matches which are chosen as possible mismatches are shown in red. The last column is the distinction between the estimated correct matches (in green) and the estimated mismatches (in red) in *Step III*. The meshes $\widehat{M}_1$, $\widehat{M}_2$, and $\widehat{M}_3$ are overlaid to illustrate the computed warps.

ated images, and eventually the resulting ROC curves for all the mismatch removal algorithms on the dataset. In the ROC curves, for a certain algorithm and a certain dataset, each point is the average value of TPR and FPR over all 60 images of that dataset using a specific value for the threshold used in the algorithm. As can be seen, in all cases, our algorithm outperforms the other algorithms. In order to show the performance of our algorithm visually, for each dataset, we overlaid $\widehat{M}_3$ for the three selected frames. As can be observed, the transferred meshes are visually well-aligned to the 2D deformed shape of the object. In some cases, a small number of irregularities can be observed in

Method	Average run-time (s)
`myNeighbor`	0.0139
Tran et al. [32]	0.0206
Pizarro et al. [30]	1.8925
Famouri et al. [17]	0.0171

Table 2: Comparison of the average run-time of the mismatch removal algorithms for processing all the images of all the datasets.

certain areas (for example in the Matrix poster). This is because of the presence of a small number of mismatches in our list of estimated correct matches and the lack of matches in those areas. As for comparing the execution speed of different mismatch removal algorithms, the processing run-times for all the frames of all datasets were averaged and tabulated in Table 2. This comparison shows that our algorithm is faster than the others. It should be however noted that our algorithm is implemented in C++ while the others are in Matlab.

After validating the efficiency of `myNeighbor` in comparison to the existing algorithms, we evaluate its performance in real cases. To this end, we applied our algorithm to four real deforming objects, shown in Figure 10. We chose these cases in such a way that each one is challenging in a special way. The cases include a cushion with non-smooth surface and severe deformation, a Spiderman poster deformed in a scene with background covered with very similar posters, a shoe sole with an almost repetitive texture, and a shirt with elastic deformation. The texture maps are shown in the first column of Figure 10. The second to fourth columns show the results of *Step I* to *Step III* of `myNeighbor`. In each step, the alignment of the corresponding transferred mesh to the 2D shape of the deforming object indicates the correctness and abundance of the estimated correct matches. Like in the synthetic data experiments, this alignment improves progressively in the steps of our algorithm. Importantly, the shirt (the last case in Figure 10) is elastic. We exert a non-isometric deformation on it by pulling from both sides, and `myNeighbor` still works. This is due to the fact that we did not make any assumptions regarding isometry. In fact, the only assumption that we made is the preservation of the neighborhood structure in the deforming object. As a result, `myNeighbor` also works with non-isometric deformations which preserve just the neighborhood structure.

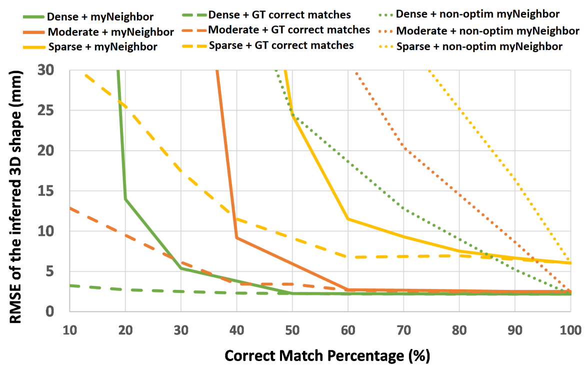


Figure 11: Results of applying `ROBUSfT` to experiments performed with `FREX` in three different scenarios: Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each curve represents the average result of all 60 frames of `FREX` with 20 trials for each frame. The solid lines indicate the use of `myNeighbor` to estimate correct matches, while the dotted and dashed lines represent the use of a non-optimal version of `myNeighbor` and the ground-truth correct matches, respectively.

6. Experimental Results

We evaluate the performance of `ROBUSfT` on different deforming objects in various conditions. We divide this section into three main parts: first, analyzing the robustness of `ROBUSfT` using `FREX`, then comparing the results with the state-of-the-art methods and finally evaluating `ROBUSfT` in several other challenging cases.

6.1. Robustness analysis

We analyze the robustness of `ROBUSfT` through several experiments using `FREX`. For these experiments, we utilize 60 images of the deforming Aruco marker paper sheet from Section 5.3. The experiments are designed with three different scenarios based on the number of matches, namely Dense, Moderate, and Sparse, with 1000, 200, and 50 matches, respectively. The richness of the object's texture, the level of occlusion, the lighting conditions, the choice of keypoint detection and matching algorithm, and the sensor noise can affect the number and correctness of matches in different parts of the image. Therefore, in each frame, we use a random selection of matches with a varying correct match rate, ranging from 10% to 100%. `ROBUSfT` is then applied to these matches to infer the 3D shape of the 8×8 Aruco template mesh. The Root Mean Squared Error (RMSE) in millimeters between the

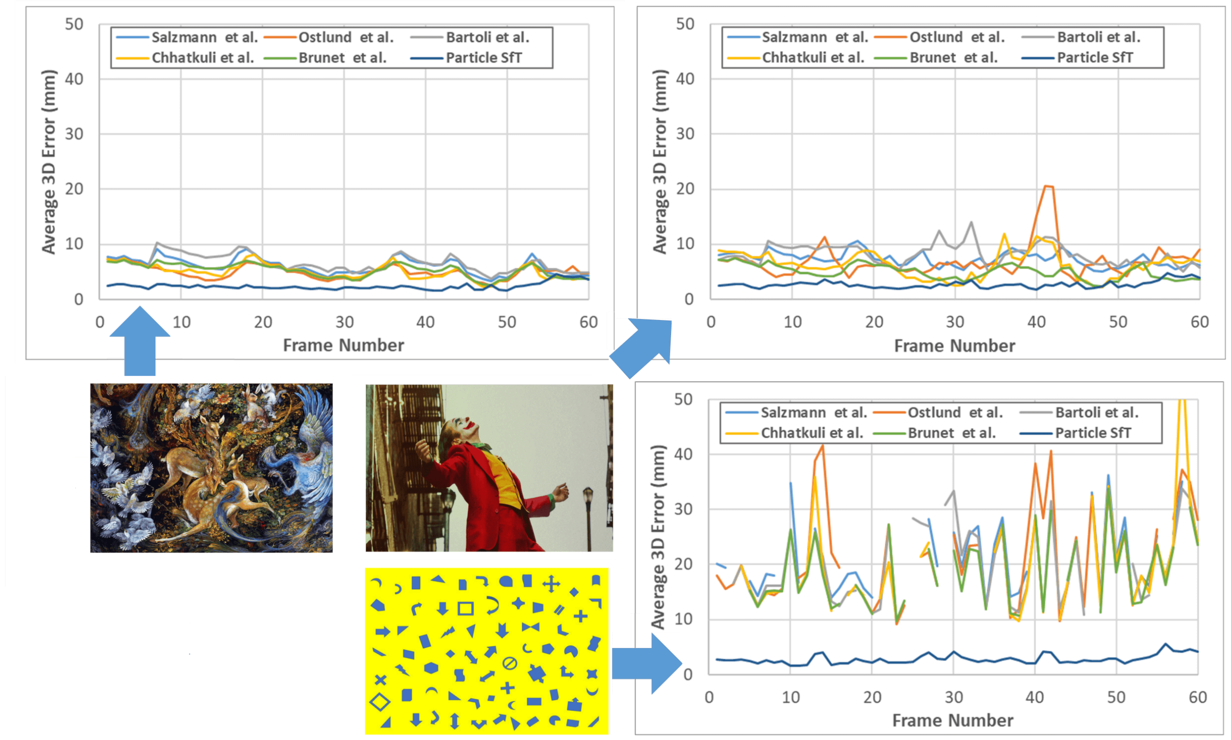
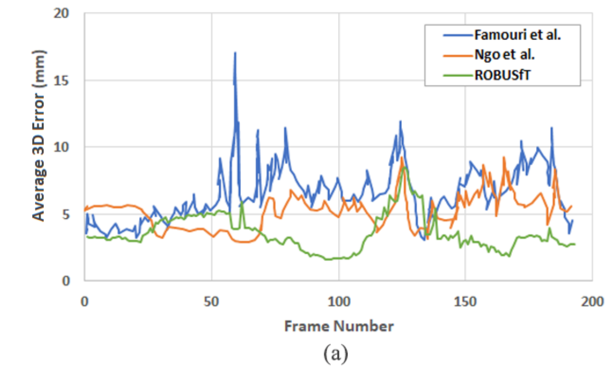


Figure 12: Comparing the accuracy of the 3D shape inference methods with Particle-SfT using three datasets obtained by `FREX`. The 3D shape inference methods are Brunet et al. [18], Chhatkuli et al. [48], Bartoli et al. [15], Ostlund et al. [21], and Salzmann et al. [20].

inferred shape and the ground-truth deformed 3D mesh is calculated. Each experiment is repeated 20 times, and we report the average error of all frames and trials as the final error. The results are shown in Figure 11 as solid lines. Our findings demonstrate that `ROBUSfT` exhibits outstanding performance in all scenarios, with a 3D error lower than 10 mm with only 30%, 40% and 60% of correct matches, in the Dense, Moderate and Sparse scenarios respectively. This remarkable robustness can be attributed to the efficiency of `myNeighbor` in handling large percentages of mismatches.

Additionally, we compare these results with two other series of experiments. *(i)* Experiments using a non-optimal version of `myNeighbor` (indicated by dotted lines). In these experiments, we change the value of α_s from an optimal value of 0.15 to a non-optimal value of 0.8 (see Section 5.2.3). This adjustment causes `myNeighbor` to detect and remove a lower number of mismatches. As can be observed, this has a pronounced negative impact on the performance of `ROBUSfT`. *(ii)* Idealized experiments where all correct matches are accurately identified (indicated by dashed lines). In these experiments, we directly use the ground-truth correct matches. In fact, these experiments represent the usage of an ideal mismatch removal algorithm. Figure 11 effectively demonstrates the significance of our novel mismatch removal algorithm, `myNeighbor`, in enhancing the accuracy of `ROBUSfT`. Comparing the outcomes of `ROBUSfT` with the idealized experiments highlights that `ROBUSfT` attains its optimal performance when the percentage of correct matches surpasses certain thresholds for each scenario. Specifically, these thresholds are 50%, 60%, and 80% for the Dense, Moderate, and Sparse scenarios, respectively. These threshold values are both reasonable and applicable in real-world situations.



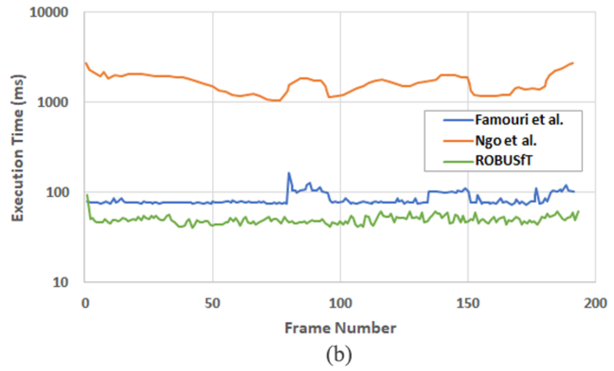


Figure 13: Comparison between the results of applying `ROBUSfT` and the integrated methods, *i.e.* Famouri et al. [17] and Ngo et al [22] on the public dataset provided in [45]. (a) Average 3D error between the inferred shape and the ground truth. (b) Execution time in milliseconds.

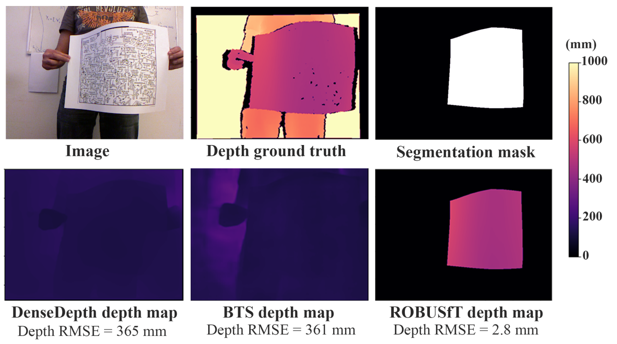


Figure 14: Comparison between the depth map from `ROBUSfT` and the deep object-generic monocular reconstruction methods, *i.e.* DenseDepth [35] and BTS [36], on one frame of the dataset provided in [45]. To make a fair comparison, the depth RMSE is calculated using the pixels from the segmentation mask. This segmentation mask is estimated in the registration step of `ROBUSfT`.

6.2. *Comparison to existing methods*

We compare `ROBUSfT` with existing methods through two different tests. The first test is conducted among the shape inference methods (`G1`). The second test is carried out among the integrated methods (`G2`) and the DNN-based methods (`G3`).

Comparison to `G1` methods. We use `FREX` to conduct this test. To this end, the same 60 images of a deforming Aruco marker paper sheet are used. We create three different datasets using three arbitrary texture maps and apply a white background to all the scenes. The arbitrary texture maps include a painting, the Joker poster, and a paper sheet filled with basic geometric shapes. These images are shown in Figure 12. In each dataset, we compare the result of the last two steps of `ROBUSfT` (warp estimation and 3D shape inference) with five other shape inference methods from Brunet et al. [18], Chhatkuli et al. [48], Bartoli et al. [15], Ostlund et al. [21], and Salzmann et al. [20]. A similar comparison was made in [19] on another dataset. However, in [19], a random 3D shape was used as the initial guess for the Particle-SfT algorithm in each image of the video; in contrast, we use the 3D inferred shape of the object in each image as the initial guess for the next image. In each dataset, the matches between $\mathcal{P}$ and each image are extracted using SIFT. We then separate the correct matches and use them as the input for all the methods. If required by a shape inference method, a BBS warp is estimated based on these correct matches and used as the input to that shape inference method. The results for all three datasets are presented in Figure 12 as the average 3D error between the 3D inferred shapes and the ground truth. As can be observed, Particle-SfT provides the lowest value of 3D error in comparison to the other methods. This is more apparent in the datasets with lower numbers of matches. In the last dataset, there are several discontinuities in the 3D error graph of existing methods. This is due to the failure of shape inference in those images of the video by those methods. Particle-SfT, however, successfully infers the 3D shape of the object in all of the images with a reasonable error.

Comparison to `G2` and `G3` methods. For the second test, we ran `ROBUSfT` on the public dataset provided in [45]. The dataset includes the 2D correspondences as well as

3D Kinect data of 193 consecutive images of a deforming paper. The paper is planar and no occlusion appears in the series of images. We compared our results with the integrated and DNN-based methods. This is shown in Figures 13 and 14, and Table 3. The integrated methods include Famouri et al. [17] and Ngo et al. [22]. We use the results of these methods on all the dataset frames provided in [17]. As can be observed in Figure 13, ROBUSfT is more precise in most of the frames. It is also faster than the compared methods. It should be noted that ROBUSfT uses images directly as input and covered the whole process from extracting keypoints to 3D shape inference. In contrast, the two integrated methods use the already available correspondences in the dataset. Next, we compare ROBUSfT with two deep object-generic monocular reconstruction methods, *i.e.* DenseDepth [35] and BTS [36]. The importance of these methods is that, similarly to ROBUSfT, they do not need to be trained for a new object. For both methods, we use the model pre-trained with the NYUDepth dataset [49], which is a collection of RGB-D images from indoor scenes with various common objects. We compare the resulting depth maps from ROBUSfT and these two methods. This comparison is shown for one frame of our test dataset in Figure 14. To make a fair comparison, the depth error is computed only for the object's surface and not the whole scene. This is done by considering the pixels from the segmentation mask which is estimated in the registration step of ROBUSfT. As can be observed, the deep object-generic monocular reconstruction methods are unable to predict the depth of the object. This is also discussed in [28], where it was shown that these methods require fine-tuning to be able to estimate the deformation of specific objects. Another issue with these methods is the lack of registration. In other words, these methods only provide the depth map and not the shape of the object in 3D space. Consequently, an additional registration method should be used to register the object to the depth map so that the 3D shape of the object can be estimated. Our last comparison is with the DNN-based SfT methods. In contrast to the deep object-generic monocular reconstruction methods, DNN-based SfT methods are trained for specific objects using synthetic and real datasets and are capable of handling both registration and reconstruction. The compared methods include HDM-Net [26], IsMo-GAN [27], DeepSfT [28], and RRNet [29]. The comparison is shown in Table 3.

Method	Average error (mm)	Average execution time (ms)
ROBUSfT	3.56	50
HDM-Net [26]	17.92	40
IsMo-GAN[27]	15.91	96
DeepSfT[28]	6.97	49
RRNet[29]	8.63	16

Table 3: Comparison between the average 3D error from applying ROBUSfT and the DNN-based SfT methods, *i.e.* HDM-Net [26], IsMo-GAN [27], DeepSfT [28], and RRNet [29], on 50 frames of the public dataset provided in [45].

We compare the average errors of applying these methods on 50 frames of the dataset [45] presented in [29] to the average error of applying ROBUSfT on the same frames. As can be seen, ROBUSfT is more precise than the DNN-based SfT methods. Importantly, as explained in [29], the DNN-based SfT methods are specifically trained on this dataset. ROBUSfT, however, does not need any prior training. In terms of execution speed, several of the DNN-based SfT methods are faster than ROBUSfT. We, however, note that in this test, we use a serial CPU-GPU architecture instead of a parallel one. This is done to make sure that the captured image that we analyze and the ground truth that we compare to are for the same image. This consequently reduces the execution speed of our code compared to the parallel architecture. In conclusion, considering generalizability, efficiency, and execution speed, ROBUSfT achieves better performance in comparison to existing methods.

6.3. *Evaluation of* ROBUSfT

Evaluation on daily objects. We first evaluate the efficiency of ROBUSfT in three real cases. These cases are shown in Figure 15. The tested objects are a Spiderman poster, a chopping mat, and a T-shirt. In each case, the object is deformed in front of a 3D camera with which we capture both an RGB image and the depth of each point on the object. We use the measured depth as ground truth for evaluating the reconstructed 3D shape. We use the Intel RealSense D435 depth camera and built-in libraries for aligning the depth map to the RGB image. For each case, four images of the experiment are shown in Figure 15. In the first case, we set the resolution of the camera to 640×480. In the second and third cases, we increased it to 1280×720 due to the insufficient number of detected keypoints using the previous resolution. Below

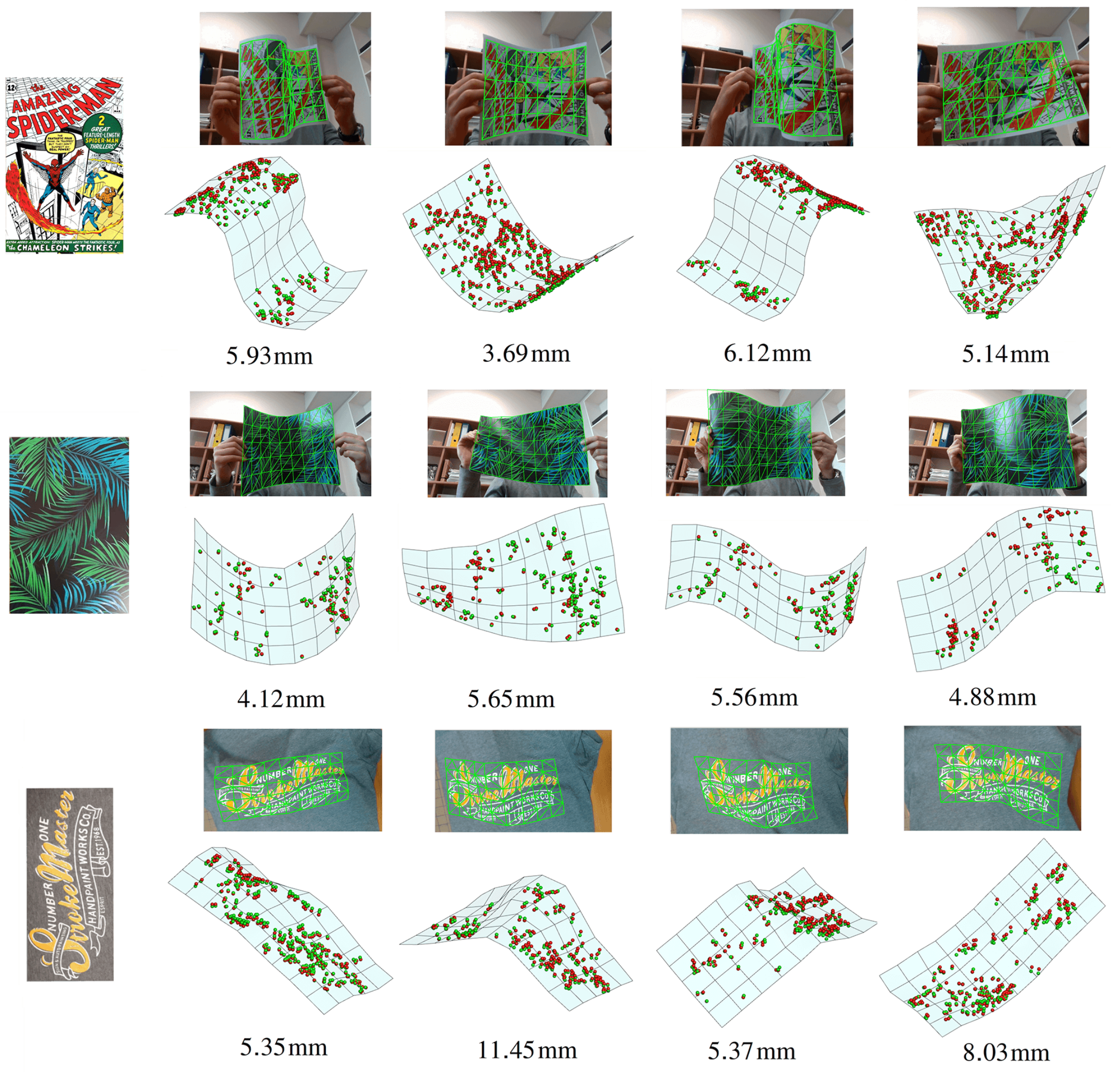


Figure 15: Evaluating `ROBUSfT` in three real data experiments: a Spiderman poster, a chopping mat, and a T-shirt. The texture maps of the templates are shown in the first column. For each case, four images are shown. Below each frame, the reconstructed 3D shape of the deforming object with the estimated 3D coordinates of the estimated correct matches (red particles) as well as their ground truth (green particles) are shown. The 2D projections of the 3D inferred shapes are also overlaid on the image. For each image, the median Euclidean distance between the estimated 3D coordinates of the estimated correct matches and their ground truth is given below the reconstructed shape.

each image, the reconstructed 3D shape of the deforming object along with the 3D coordinates of the estimated correct matches (red particles) as well as their ground truth (green particles) are shown. The 3D coordinates of the estimated correct matches are estimated by calculating their barycentric coordinates in $\mathcal{P}$ with respect to M and applying these coordinates on the 3D reconstructed mesh of the object. The number written below each frame is the median distance between the reconstructed 3D coordinates of the estimated correct matches and their ground truth. The

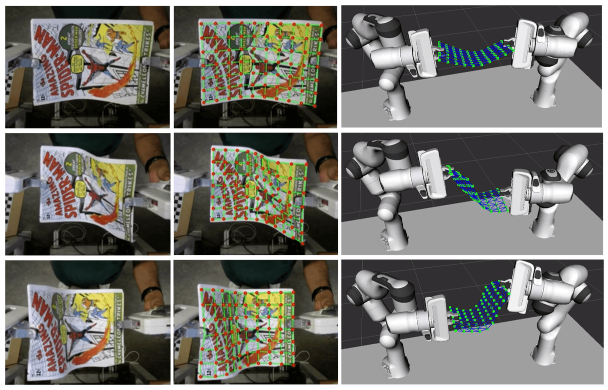


Figure 16: Evaluating ROBUSfT in a real data experiment with two robotic arms; soft constraints are applied to bind the constrained mesh points to the grippers. Each row shows three images: the original camera view, the projection of the 3D reconstructed mesh on the camera view, and the 3D reconstructed mesh with the robots in the RViz environment.

median is chosen due to the probable existence of mismatches among the list of estimated correct matches. In 3D space, the ground truth of these mismatches can be located in the background and not on the object itself. This significantly increases the 3D shape error. Using the median gives a better estimate of the 3D shape error considering the existence of this small percentage of mismatches with large 3D errors.

As can be observed, the pipeline successfully infers the 3D shape of the object in all cases. Regarding the Spiderman poster case, it should be noted that there are self-occlusions in the first and third illustrated images. In these images, the 3D shape of the object in the occluded areas is estimated by the deformation constraints implemented in Particle-SfT. These constraints preserve the geodesic distance between each pair of mesh points as equal to its initial value in M_T. Regarding the run-time, using the parallel architecture and 640×480 captured frames as the input (as in the Spiderman poster case), the execution speed reaches 30 fps.

Evaluation on a robotic use case. The last experiment is a practical use case with robots. The experiment aims at highlighting the advantage of using known 3D coordinates in ROBUSfT. As mentioned in *Step 4* and shown in Figure 16, these known coordinates are an optional input to the last step of ROBUSfT. Their usage can increase the robustness of the tracking process. The setup of this experiment is the same as in [50], where we applied ROBUSfT in a robotic use case; specifically, controlling the shape of deformable objects. The setup consists of two robotic arms grasping and manipulating the Spiderman poster from both sides and a top camera facing the manipulation area. The 3D positions of the two robotic grippers are known in camera coordinates, thanks to the known pose of each gripper in the robots' coordinate frames and also the external calibration between the robots and the camera. For each gripper, we consider the closest mesh point to the gripper as a constrained mesh point. These mesh points should be bound to their corresponding gripper and move with it. As described in [50], this binding is performed using a soft constraint. In this soft constraint, for each gripper, a sphere with a small radius centered at the gripper's 3D position is considered. Then, in each iteration of Particle-SfT, if the corresponding mesh point is outside this sphere, it will be absorbed to the closest point on the sphere surface. This soft constraint has two main advantages over rigidly binding the constrained mesh points to the grippers. First, it allows the position-based dynamic equations in Particle-SfT that preserve the distances between the mesh points to be applied on the constrained mesh points, which leads to a smoother reconstructed shape. Second, it allows one to cope with small possible errors in robot-camera calibration. In fact, a wrong robot-camera calibration leads to a wrong transfer of the grippers' 3D coordinates to the camera coordinate frame which eventually results in wrong coordinates of the constrained mesh points. By using the soft constraint and considering a sphere rather than a rigid bind, we give a certain degree of flexibility to the constrained mesh points to move in close proximity to the gripper's coordinates. This can compensate for slightly inaccurate coordinates of the grippers.

7. Conclusion

We have proposed ROBUSfT, a publicly available C++ library that can effectively track the 3D shape of an isometrically deforming object using a monocular 2D camera. ROBUSfT outperforms the state-of-the-art methods. The proposed pipeline addresses the well-known challenges in this area. These challenges include ambiguities in inferring the 3D shape of the deforming object from a single 2D image, and real-time implementation. We have introduced myNeighbor, a novel mismatch removal algorithm for deforming objects, which works based on the preservation of the neighborhood structure of matches. We validated the efficiency of myNeighbor in comparison to existing algorithms in numerous experiments. In order to compare ROBUSfT and myNeighbor with existing methods, we have presented a novel type of experimental protocol called FREX (Fake but Realistic Experiment). A single execution of this protocol provides a collection of scenes of an isometrically deforming object in various conditions with 2D and 3D ground truth. This collection can be used to evaluate, compare, and validate algorithms regarding isometrically deforming objects. In addition, the provided 2D and 3D ground truth may be used for training learning-based algorithms. In contrast to other artificially made scenes of an isometrically deforming surface, the generated images in FREX are the result of real isometric deformations.

The primary limitation of ROBUSfT is that it is restricted to surface object models, which can appropriately model thin-shell objects and the surface of volumetric objects. As future work, we plan to extend ROBUSfT to handle volumetric object models. This would allow ROBUSfT to exploit the deformation constraints that exist inside thick deformable objects, and should improve its accuracy compared to the current surface model. A secondary limitation is that ROBUSfT requires textured object surfaces to establish correspondences, even though it is able to cope with reasonably low numbers of correspondences. As future work, we plan to use the object's silhouette, in other words, the occluding contour, to complement the correspondences. Importantly, these two limitations also apply to the vast majority of existing SfT methods.

Acknowledgments

This work was supported by project SOFTMANBOT, which received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 869855. This work was also supported via a María Zambrano fellowship funded by the Spanish Ministry of Universities and by the European Union-NextGenerationEU.

References

[1] N. Haouchine, J. Dequidt, M.-O. Berger, S. Cotin, Single view augmentation of 3D elastic objects, in: IEEE International Symposium on Mixed and Augmented Reality, 2014.

[2] J. Pilet, V. Lepetit, P. Fua, Fast non-rigid surface detection, registration and realistic augmentation, International Journal of Computer Vision 76 (2) (2008) 109–122.

[3] M. Hu, G. P. Penney, D. Rueckert, P. J. Edwards, F. Bello, R. Casula, M. Figl, D. J. Hawkes, Non-rigid reconstruction of the beating heart surface for minimally invasive cardiac surgery, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, 2009.

[4] T. Collins, B. Compte, A. Bartoli, Deformable shape-from-motion in laparoscopy using a rigid sliding window, in: Medical Image Understanding and Analysis Conference, 2011.

[5] A. Malti, A. Bartoli, T. Collins, Template-based conformal shape-from-motion-and-shading for laparoscopy, in: International Conference on Information Processing in Computer-Assisted Interventions, 2012.

[6] T. Collins, A. Bartoli, N. Bourdel, M. Canis, Robust, real-time, dense and deformable 3D organ tracking in laparoscopic videos, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, 2016.

[7] J. Lamarca, S. Parashar, A. Bartoli, J. M. M. Montiel, DefSLAM: Tracking and mapping of deforming scenes from monocular sequences, IEEE Transactions on Robotics 37 (1) (2020) 291–303.

[8] Y. Li, Y. Wang, M. Case, S.-F. Chang, P. K. Allen, Real-time pose estimation of deformable objects using a volumetric approach, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014.

[9] B. Frank, C. Stachniss, R. Schmedding, M. Teschner, W. Burgard, Learning object deformation models for robot motion planning, Robotics and Autonomous Systems 62 (8) (2014) 1153–1174.

[10] M. Aranda, J. A. Corrales Ramon, Y. Mezouar, A. Bartoli, E. Özgür, Monocular visual shape tracking and servoing for isometrically deforming objects, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2020.

[11] E. Tretschk, N. Kairanda, M. BR, R. Dabral, A. Kortylewski, B. Egger, M. Habermann, P. Fua, C. Theobalt, V. Golyanik, State of the art in dense monocular non-rigid 3D reconstruction, Computer Graphics Forum 42 (2) (2023) 485–520.

[12] M. Salzmann, P. Fua, Reconstructing sharply folding surfaces: A convex formulation, in: IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[13] M. Salzmann, F. Moreno-Noguer, V. Lepetit, P. Fua, Closed-form solution to non-rigid 3D surface registration, in: European Conference on Computer Vision, 2008.

[14] M. Perriollat, R. Hartley, A. Bartoli, Monocular template-based reconstruction of inextensible surfaces, International Journal of Computer Vision 95 (2) (2011) 124–137.

[15] A. Bartoli, Y. Gérard, F. Chadebecq, T. Collins, D. Pizarro, Shape-from-template, IEEE Transactions on Pattern Analysis and Machine Intelligence 37 (10) (2015) 2099–2118.

[16] A. Chhatkuli, D. Pizarro, A. Bartoli, T. Collins, A stable analytical framework for isometric shape-from-template by surface integration, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (5) (2016) 833–850.

[17] M. Famouri, A. Bartoli, Z. Azimifar, Fast shape-from-template using local features, Machine Vision and Applications 29 (1) (2018) 73–93.

[18] F. Brunet, A. Bartoli, R. I. Hartley, Monocular template-based 3D surface reconstruction: Convex inextensible and nonconvex isometric methods, Computer Vision and Image Understanding 125 (2014) 138–154.

[19] E. Özgür, A. Bartoli, Particle-SfT: A provably-convergent, fast shape-from-template algorithm, International Journal of Computer Vision 123 (2) (2017) 184–205.

[20] M. Salzmann, P. Fua, Linear local models for monocular reconstruction of deformable surfaces, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (5) (2010) 931–944.

[21] J. Östlund, A. Varol, D. T. Ngo, P. Fua, Laplacian meshes for monocular 3D shape recovery, in: European Conference on Computer Vision, 2012.

[22] D. T. Ngo, J. Östlund, P. Fua, Template-based monocular 3D shape recovery using Laplacian meshes, IEEE Transactions on Pattern Analysis and Machine Intelligence 38 (1) (2015) 172–187.

[23] T. Collins, A. Bartoli, [POSTER] Realtime shape-from-template: System and applications, in: IEEE International Symposium on Mixed and Augmented Reality, 2015.

[24] Q. Liu-Yin, R. Yu, L. Agapito, A. Fitzgibbon, C. Russell, Better together: Joint reasoning for non-rigid 3D reconstruction with specularities and shading, arXiv preprint arXiv:1708.01654 (2017).

[25] A. Pumarola, A. Agudo, L. Porzi, A. Sanfeliu, V. Lepetit, F. Moreno-Noguer, Geometry-aware network for non-rigid shape prediction from a single view, in: IEEE Conference on Computer Vision and Pattern Recognition, 2018.

[26] V. Golyanik, S. Shimada, K. Varanasi, D. Stricker, HDM-Net: Monocular non-rigid 3D reconstruction with learned deformation model, in: International Conference on Virtual Reality and Augmented Reality, 2018.

[27] S. Shimada, V. Golyanik, C. Theobalt, D. Stricker, IsMo-GAN: Adversarial learning for monocular

non-rigid 3D reconstruction, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2019.

[28] D. Fuentes-Jimenez, D. Pizarro, D. Casillas-Pérez, T. Collins, A. Bartoli, Deep shape-from-template: Single-image quasi-isometric deformable registration and reconstruction, Image and Vision Computing 127 (2022) 104531.

[29] D. Fuentes-Jimenez, D. Pizarro, D. Casillas-Perez, T. Collins, A. Bartoli, Texture-generic deep shape-from-template, IEEE Access 9 (2021) 75211–75230.

[30] D. Pizarro, A. Bartoli, Feature-based deformable surface detection with self-occlusion reasoning, International Journal of Computer Vision 97 (1) (2012) 54–70.

[31] T. Collins, P. Mesejo, A. Bartoli, An analysis of errors in graph-based keypoint matching and proposed solutions, in: European Conference on Computer Vision, 2014.

[32] Q.-H. Tran, T.-J. Chin, G. Carneiro, M. S. Brown, D. Suter, In defence of RANSAC for outlier rejection in deformable registration, in: European Conference on Computer Vision, 2012.

[33] N. Kairanda, E. Tretschk, M. Elgharib, C. Theobalt, V. Golyanik, ϕ-sft: Shape-from-template with a physics-based deformation model, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022.

[34] O. Sorkine, M. Alexa, As-rigid-as-possible surface modeling, in: Eurographics Symposium on Geometry Processing, 2007.

[35] I. Alhashim, P. Wonka, High quality monocular depth estimation via transfer learning, arXiv preprint arXiv:1812.11941 (2018).

[36] J. H. Lee, M.-K. Han, D. W. Ko, I. H. Suh, From big to small: Multi-scale local planar guidance for monocular depth estimation, arXiv preprint arXiv:1907.10326 (2019).

[37] A. Agarwal, C. Arora, Attention attention everywhere: Monocular depth prediction with skip attention, in: IEEE/CVF Winter Conference on Applications of Computer Vision, 2023.

[38] W. Yuan, X. Gu, Z. Dai, S. Zhu, P. Tan, Neural window fully-connected CRFs for monocular depth estimation, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022.

[39] C. Liu, S. Kumar, S. Gu, R. Timofte, L. Van Gool, Single image depth prediction made better: A multivariate Gaussian take, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023.

[40] Agisoft, Agisoft PhotoScan.
URL `https://www.agisoft.com`

[41] D. G. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision 60 (2) (2004) 91–110.

[42] C. Griwodz, L. Calvet, P. Halvorsen, Popsift: A faithful SIFT implementation for real-time applications, in: ACM Multimedia Systems Conference, 2018.

[43] F. L. Bookstein, Principal warps: Thin-plate splines and the decomposition of deformations, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (6) (1989) 567–585.

[44] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. Hill, M. O. Leach, D. J. Hawkes, Nonrigid registration using free-form deformations: Application to breast MR images, IEEE Transactions on Medical Imaging 18 (8) (1999) 712–721.

[45] A. Varol, A. Shaji, M. Salzmann, P. Fua, Monocular 3D reconstruction of locally textured surfaces, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (6) (2012) 1118–1130.

[46] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff, Position based dynamics, Journal of Visual Communication and Image Representation 18 (2) (2007) 109–118.

[47] J. Bender, M. Müller, M. A. Otaduy, M. Teschner, M. Macklin, A survey on position-based simulation methods in computer graphics, Computer Graphics Forum 33 (6) (2014) 228–251.

[48] A. Chhatkuli, D. Pizarro, A. Bartoli, Stable template-based isometric 3D reconstruction in all imaging conditions by linear least-squares, in: IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[49] N. Silberman, D. Hoiem, P. Kohli, R. Fergus, Indoor segmentation and support inference from RGBD images, in: European Conference on Computer Vision, 2012.

[50] M. Shetab-Bushehri, M. Aranda, Y. Mezouar, E. Özgür, As-rigid-as-possible shape servoing, IEEE Robotics and Automation Letters 7 (2) (2022) 3898–3905.