TEXTURE-GENERIC DEEP SHAPE-FROM-TEMPLATE

David Fuentes-Jimenez
Department of Electronics
Universidad de Alcalá (UAH)
E-28871 Alcalá de Henares, Spain
e-mail: `d.fuentes@edu.uah.es`

Daniel Pizarro
Department of Electronics
Universidad de Alcalá (UAH)
E-28871 Alcalá de Henares, Spain
e-mail: `daniel.pizarro@uah.es`

David Casillas-Perez
Department of Signal Processing and Communications
Universidad Rey Juan Carlos (URJC)
Fuenlabrada 28933 Spain
e-mail: `david.casillas@urjc.es`

Toby Collins
IRCAD
Place de l'Hôpital
Strasbourg 67000, France
e-mail: `toby.collins@gmail.com`

Adrien Bartoli
EnCoV
Clermont-Ferrand 63000, France
e-mail: `adrien.bartoli@gmail.com`

May 21, 2021

ABSTRACT

Shape-from-Template (SfT) solves the registration and 3D reconstruction of a deformable 3D object, represented by the template, from a single image. Recently, methods based on deep learning have been able to solve SfT for the wide-baseline case in real-time, clearly surpassing classical methods. However, the main limitation of current methods is the need for fine tuning of the neural models to a specific geometry and appearance represented by the template texture map. We propose the first texture-generic deep learning SfT method which adapts to new texture maps at run-time, without the need for texture specific fine tuning. We achieve this by dividing the problem into a segmentation step and a registration and reconstruction step, both solved with deep learning. We include the template texture map as one of the neural inputs in both steps, training our models to adapt to different ones. We show that our method obtains comparable or better results to previous deep learning models, which are texture specific. It works in challenging imaging conditions, including complex deformations, occlusions, motion blur and poor textures. Our implementation runs in real-time, with a low-cost GPU and CPU.

Keywords Monocular, 3D Model, Image registration, 3D reconstruction, Wide-baseline, Dense, Deformable reconstruction, Shape-from-Template

1 Introduction

Image registration and image-based 3D reconstruction are fundamental problems extensively studied in Computer Vision. However, solving these problems with deformable objects remains challenges. In Shape-from-Template (SfT) [1, 2, 3, 4], the objective is to reconstruct the 3D shape of a deformable object from a single image and a reference 3D model of the object, known as the template. The template is a fundamental component of SfT that provides prior knowledge via three models. The first model is the *shape model*, typically represented as a triangulated 3D mesh, which gives the shape of the object in a known position (usually called the *reference shape*). The second model is the *deformation model* that determines how the object may deform from the reference shape. This is used to combat the general ill-posedness of 3D reconstruction from a single image, and it restricts the space of possible solutions to ones that are physically viable. The third model is the appearance model that represents the texture of the object's surface. This is required to relate the image with the template's surface at the pixel level though the object's texture. By far, the most common way to implement the appearance model is with texture mapping. This approach assigns each point on the template's surface to a pixel colour sampled from a discrete colour map known as the *texture map*.

The shape model and texture map are usually constructed with an optical acquisition system. There are two main approaches: The first approach uses an RGBD camera, where the depth information is used to construct the shape model and the RGB information is used to construct the texture map. The second approach uses several images taken with an RGB camera of the object in the reference position, which are then used to build the shape model and texture map using a combination of Structure-from-Motion (to provide the relative poses of the camera images) and multi-view stereo (to densely reconstruct the shape models and to build the texture map). The second approach is usually preferred in practice because the same camera can be used for constructing the template and running SfT [2, 5].

Using the template and an image of the deformed object, the general goal of SfT is to infer the 3D deformation of the template so that it matches the image. All three models in the template (shape, deformation and appearance) are essential to make SfT accurately solvable. Formally, two challenging and related problems are solved in SfT. The first challenge is to *register* the template to the image (*i.e.* to determine a dense spatial correspondence between the image and the template shape). The second challenge is to *reconstruct* the template's deformed shape (*i.e.* to determine the depth of each point on the template's surface with respect to the camera center).

SfT has numerous practical applications, and an important one is to facilitate Augmented Reality (AR) with deformable objects. Figure 1 shows the workflow diagram of a standard SfT+AR system. In this application, registration is required to correctly position virtual objects in the image to align with the deforming object's surface and reconstruction is required to correctly orient the virtual objects in 3D.

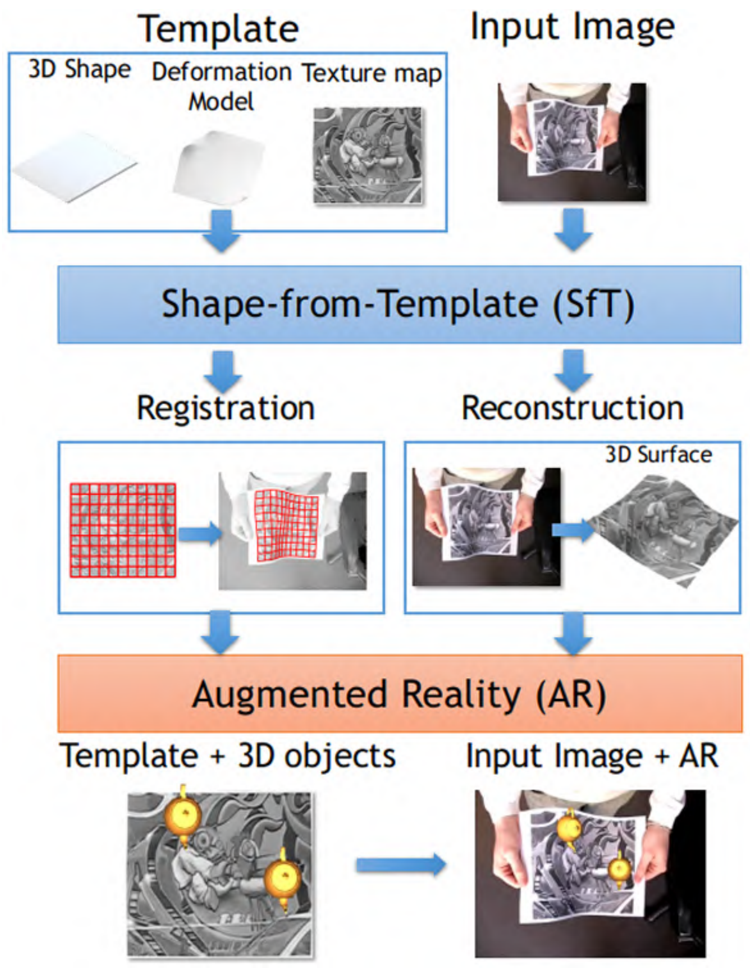


Figure 1: Principle of SfT and AR for deformable objects.

The texture map is necessary as the purpose of SfT is two-fold: to register the template to the image and to reconstruct it in 3D. Both tasks are strongly interdependent, and only the texture map allows a precise registration at the pixel level using the specific texture details of the object of interest. Without a texture map, it would be impossible to exploit the object's specific texture characteristics, fundamentally preventing accurate registration and reconstruction. In practice, requiring a texture map is not a limitation because it is obtained as part of template construction using an optical system as described above.

SfT needs a deformation model to constrain the solution. Multiple deformation models have been studied, such as isometric [4, 6, 7], conformal [2, 8, 9], equiareal [10] and elastic [11, 12, 13] deformations. The most popular deformation models to solve SfT are isometry and quasi-isometry, also known as the as-rigid-as-possible (ARAP) model, which is a widely used relaxation of isometry. These models prevent the object from stretching or shrinking, and they make SfT well-posed in the general case.

SfT is a challenging problem when facing the conditions of a real application. The challenges are related to: *a)* the type and complexity of the object in terms of geometry (volumic or thin shell), deformations (low or high dimensional), and texture (rich or poor), *b)* the imaging conditions (including illumination changes, occlusions and motion blur) and *c)* the baseline (short or wide). In video sequences, there is temporal continuity between the frames, which corresponds to the *short-baseline* case. On the contrary, the *wide-baseline* case implies that each input image has to be treated individually, without temporal connection to previous images, due to large camera movements and sudden deformations. In practice, the wide-baseline conditions are more realistic, since occlusions and fast camera movements that break the short-baseline assumptions are frequent.

SfT has been extensively studied over the last decade, and only very recently, methods based on Deep Neural Networks (DNN) have been proposed. We divide classical (*i.e.*, non-DNN) SfT methods into two main categories. The first category of methods decouple SfT into a registration followed by a reconstruction step. The registration step is based on robust feature-based matching methods that are not specific to SfT [14, 15, 16]. The reconstruction step is based on shape inference methods, usually a combination of a non-iterative step [2, 4] followed by iterative refinement [6]. These approaches cope with volumic and thin shell objects and work under wide-baseline conditions. They are limited because feature-based registration fails under challenging image conditions and poorly textured objects. The second category of methods perform registration and reconstruction at the same time [3, 17, 18]. Methods in this category use iterative optimisation and only work in short-baseline conditions, as they require to be initialized close to the solution. They fail when the short-baseline tracking conditions are violated due to occlusions or fast motion. As in the first category, these methods also fail under challenging imaging conditions and poorly textured objects.

In recent years, the idea of using learning methods to solve SfT has been explored. These methods learn the mapping from the input image to the object 3D deformation [19, 20, 21, 22]. They are potentially able to solve SfT in wide-baseline conditions and without the need to run optimisation at run-time. However, existing DNN-based SfT methods face four main limitations at the moment. First, some of them only work with a specific template [22], requiring fine tuning for the specific template shape and texture map. Other works propose to solve SfT for a variety of texture maps [19, 20, 21], learning invariance to these. However, their results are far from satisfactory, still requiring fine tuning of the network to a specific template. This is because the templateless problem is highly ambiguous, making template-invariant methods unreliable. Second, in terms of template geometry, most existing methods tackle the thin-shell case [19, 20, 21], with the exception of the template-specific method [22]. Third, in terms of template complexity, most methods require a mesh with a reduced number of vertices (namely, 73×73 in [20, 21] and 10×10 in [19]), which is a strong limitation to work with complex templates or deformations. Fourth, they work for a specific camera configuration, not being able to adapt to dynamical camera parameters at run-time.

To sum up, a general DNN-based SfT solution is still missing, which would work for new templates without the need of fine tuning. The importance of pursuing such a solution resides in the complexity of generating training data for SfT, and the computational resources needed for fine tuning a network in a new dataset. It is thus a necessary step towards making DNN-based SfT methods widely applicable.

We make the following contributions to advance the state-of-the-art of DNN-based SfT. First, we propose the first DNN-based SfT method that takes the template texture map as a run-time input. This is used to condition the registration and reconstruction DNNs on the specific texture of the object of interest. Importantly, the texture map supplied at run-time can be completely novel in the sense that it was not (nor any similar texture map) used in the training data. Previous DNN-based methods $[18 - 21]$ that exploit the specific texture map of an object require their DNNs to be retrained for the specific texture map (either from scratch or by fine-tuning), which is highly impractical for real-world applications such as those running on smartphones. Indeed, the limits of requiring DNN retraining have completely prevented DNN-based SfT methods from being used in real-world applications. Our method does not suffer this limitation of texture-specific training, which is a significant advance of state-of-the-art towards practical real-world use of DNN-based SfT.

Second, our proposed architecture is divided into two neural networks, the *segmentation network* for pixel-based detection of the template, and the *registration-reconstruction network* to recover the SfT solution. The segmentation network is crucial to solve SfT in our generic template approach. We have proposed a new semantic segmentation architecture that allows us to add the template texture map as one of the inputs, which clearly differs from the classical category-level semantic segmentation methods, where the semantic categories are learned and fixed during training. The output of the segmentation network is fed into the registration-reconstruction network.

Third, our DNN models for template segmentation and registration-reconstruction networks are fully-convolutional encoder-decoder networks, that use residual structures and specific layers, created by us to address SfT. They allow our method to be computationally efficient. In addition to our base models, we propose a lightweight architecture for the registration-reconstruction network that can be used to run our method in real-time in low-cost GPUs, CPUs and embedded systems. It is based on a new custom decoding layer that implements the inverse Block Discrete Cosine Transform (DCT), which allows us to greatly reduce the number of network parameters, while easily controlling the loss of information induced by the new decoding layers. Our lightweight architecture requires 21.59% of the total parameters and is 214.65% faster than the base architecture while being only 19.21% less accurate. This new optimized architecture based on the DCT can potentially be applied to other problems different from SfT, effectively reducing the size of encoder-decoder DNN regression and classification models. Fourth, our proposed method works independently of the camera parameters, recovering the correct depth and scale of the deformed object. This fact eliminates the need to use the same camera during the training and testing stages, which affects other methods [19, 20, 21]. Finally, we show that our proposed methods outperform, both qualitatively and quantitatively, the representative state-of-the-art methods in terms of accuracy, speed and number of parameters. Our results include challenging scenarios in wide-baseline conditions and effects like motion blur, occlusions, illumination changes and weak texture. All the data and code presented in this work will be released for public use.

The rest of this paper is organized as follows. Section 2 discusses previous work. Section 3 describes the SfT problem and the proposed methods with the DNN architectures. Section 4 explains the training process, the loss function, and the setup carried out. Section 5 explains the setup, datasets and methods used for the experiments before showing the obtained results, both graphically and numerically, for various experimental settings. Finally, section 6 gives our conclusions and future work.

2 Previous Work

We divide the SfT state-of-the-art into two main categories, the classical SfT methods, which include the vast majority of existing work, and the DNN-based SfT methods. Before explaining these two categories, we introduce other vision problems that are related to SfT. After that we start with the classical methods, dividing them into two sub-categories, the decoupled and the integrated methods. The former solves registration and reconstruction as two independent problems and the latter solves registration and reconstruction jointly. Finally, we review the DNN-based methods and categorize them.

2.1 Vision problems related to SfT

There are several vision problems that relate to SfT and have been recently attempted by DNN methods. These are *1)* optical flow computation [23, 24], *2)* scene flow computation [25, 26], *3)* monocular depth reconstruction [27, 28, 29, 30], *4)* human pose estimation [31, 32] and *5)* Shape-from-Shading [33, 34]. None of these methods or their combination compete with SfT-specific solutions or solve SfT under general conditions. For instance *1)* solves registration between frames in a short-baseline sequence without including depth, *2)* solves registration between frames in 3D, mainly from depth or stereo cameras and thus has not the same inputs as SfT, *3)* solves depth for a general scene from a single image and is limited to selected types of scenes (roads or indoor). Besides, it does not compute the registration with the template, which is fundamental in SfT, *4)* can be seen as a specialisation of SfT for a template that represents the human body. The deformation model in this case is low-dimensional, parametrized with a few joint angles, favouring the use of learning methods. In SfT, the number of degrees-of-freedom can be much larger than in the human skeleton, and it changes with the type of template, which makes it much more challenging than *4)*. Finally, *5)* uses radiometric models to recover 3D surfaces from shading cues detected in the images. It requires the scene to be lit by highly controlled light sources and, as in *3)*, does not include registration to the template.

2.2 Classical decoupled SfT methods

The decoupled methods first obtain registration and then 3D reconstruction as two independent processes. The advantage of this approach is the reduction of complexity. The main drawback is that it does not include the constraints existing between registration and reconstruction. It usually solves wide-baseline registration using feature-based matching methods like SURF [35] and SIFT [36], which are then cleaned from mismatches using specific methods for deformable registration [14, 37]. Feature-based registration methods fail with repetitive or poorly textured objects and challenging image conditions, such as motion blur, strong viewpoint distortion or low resolution images. As a consequence, these methods are limited in challenging scenarios. The decoupled methods are usually categorised by the deformation model, isometry being the most common one. In terms of the solver strategy used in the reconstruction step, these methods typically follow one of the following approaches: *1)* the inextensibility model, a convex relaxation of isometry, and the maximum depth heuristic [1, 6, 38, 39], *2)* local differential geometry [2, 4], and *3)* minimisation of a non-convex cost function [6, 40]. Methods in *3)* are computationally expensive and require initialisation. They are mainly used as refinement methods for approaches in *1)* or *2)*, which are convex but less accurate in general. Other works study non-isometric deformations, such as non-linear elasticity [10, 41, 42, 43], linear elasticity [11, 12] or angle-preserving conformal models [2]. The main drawback of all these methods is that they require boundary conditions, usually in the form of 3D known points, to make the reconstruction problem well posed. Whether these non-isometric models can solve SfT uniquely without additional cues remains an open question.

2.3 Classical integrated SfT methods

The integrated methods jointly perform registration and reconstruction. The majority of them are restricted to short-baseline scenarios, which imply the use of video streams [3, 17, 44]. They minimize a non-convex cost function that jointly aligns the 3D solution with diverse image cues, such as feature point correspondences [3] or by using a pixel-level photo-consistency model [17, 44]. Integrated methods are effective and can recover complex deformations, some of them in real-time. However, they fail when the short-baseline conditions are violated, and require initialization with a wide-baseline method.

2.4 DNN-based SfT methods

DNN-based SfT methods have appeared in the last few years. We establish a difference between monocular reconstruction methods, that do not use a template but are specialised in deformable objects [30, 45, 46], and methods that genuinely solve SfT's registration and reconstruction [19, 20, 21, 22]. We are interested in methods that belong to the second group. They are divided by type of output representation (dense or discrete) and whether they are specific to a template or work for a generic category of templates.

Regarding the type of output, the majority of methods represent the SfT solution with the 3D vertex coordinates of a regular mesh. The mesh vertex count varies between existing methods. [19] uses 10×10 meshes. Their approach is based on obtaining 2D belief maps for the image position of each of the mesh vertices, inspired by DNN-based human pose computation methods. This approach strictly limits the amount of vertices. Other approaches [20, 21] use three-channel 2D maps to model the coordinates of a regular mesh, reaching 73×73 vertex counts. We proposed a different approach [22], by recovering pixel-level depth and registration maps, allowing one to represent 3D objects and complex shapes of arbitrary topologies. In addition we implemented a post-processing step based on the ARAP model to recover the hidden parts of the surface, not explicitly reconstructed by the neural network. This paper is an extension of our previous work [22]. It uses fundamentally new constraints and scope but relies on the same parametrisation.

In terms of template specificity, all existing DNN-based methods are trained for a specific template shape, while they deal with the template texture map differently. We thus divide existing DNN-based methods in the following categories: *1) Single-texture methods* are trained for a specific template texture map. They exploit texture information to accurately solve SfT, and this information is built-in the DNN weights [22]. *2) Multi-texture methods* work similar to *1)*, but the DNN is trained for several texture maps. Although there is currently no existing method falling in *2)*, one could easily be created by combining object detection with several instances of a method from *1)*.

3) Texture-agnostic methods are trained with many texture maps. Hence, they learn to solve SfT using general image cues that are independent of the texture map, such as occlusion boundaries and shading [19, 20, 21]. Methods in *3)* are significantly less accurate than methods in *1)* and *2)*, producing coarse reconstruction, as shown in our experiments. Strictly speaking, methods in *3)* are not SfT methods, since they do not use the template texture map to solve SfT. Our work exploits the template texture map as in *1)* and *2)*, but using it as an input in the DNN. It can adapt to use new texture maps unseen during training and thus belongs to a new category of DNN methods, namely *4) Texture-generic methods*. We show in Section 5 that our method is only slightly less accurate than methods in *1)* and *2)*, while being much more applicable and flexible. Methods in *3)*, being independent of the texture map, have also high applicability as in *4)*. However, they fail to produce accurate reconstructions in general.

Summarizing, solving SfT with DNNs is a promising line of research, but the existing methods show important limitations. Our proposal can be classified as the first texture map-generic method, and so belongs to a new template-generic category of methods. Unlike existing methods, it exploits the texture map as an additional input to the DNN, allowing it to adapt to different texture maps at run-time. Our method uses all the available information, similarly to template-specific methods, thus guaranteeing that the SfT solution is well-posed. In terms of outputs, we use the same parametrisation as in our previous work [22], which copes with complex deformations and is not limited by the vertex count of the output mesh. Finally, we standardise the camera used during training by following [22], so that it is not necessary to use the same camera parameters during training and testing. These advances allow our method to deal with practical scenarios and contribute significantly to generic DNN-based SfT.

3 Methodology

3.1 Scene geometry

Figure 2 shows the components of the scene geometry in the SfT problem. **Template.** It is composed of a known 3D

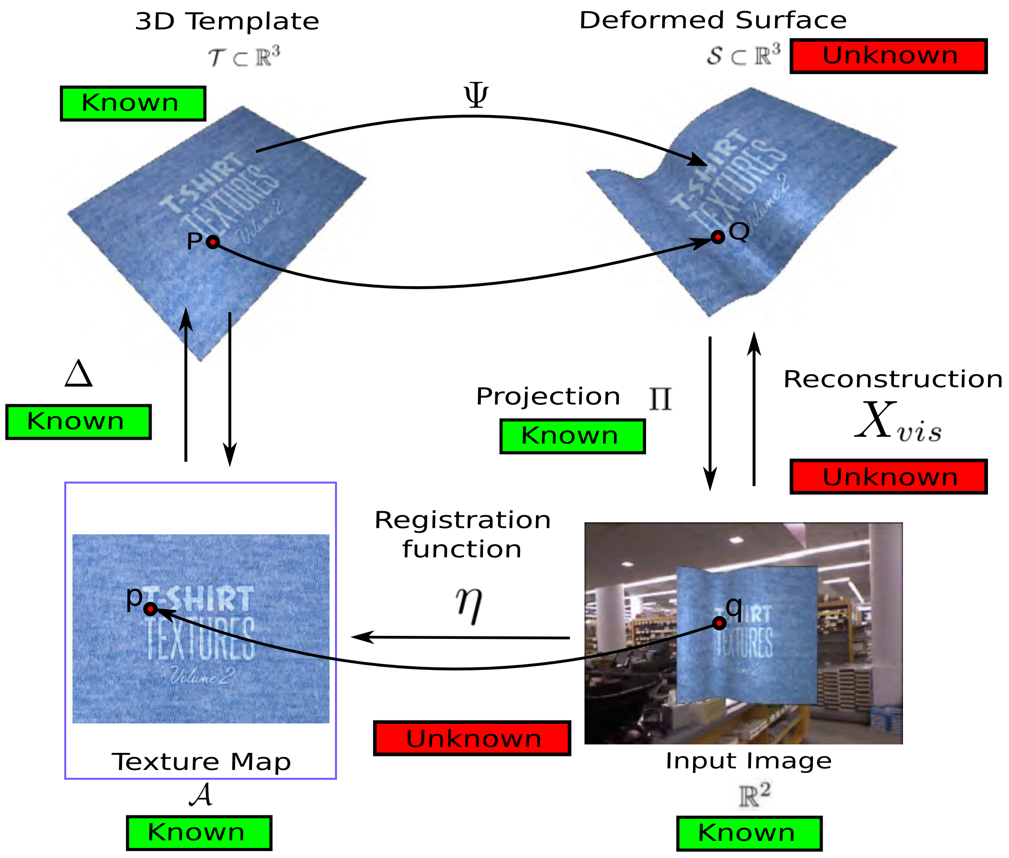


Figure 2: Geometric model of SfT.

surface $\mathcal{T} \subset \mathbb{R}^3$ and an appearance model represented by a texture map $\mathcal{A}_\mathcal{T} = (\mathcal{A}, A)$, where $\mathcal{A} \subset \mathbb{R}^2$ is a 2D subset and $A\colon \mathcal{A} \to (r, g, b)$ is a function that maps $\mathcal{A}$ to RGB values. In $\mathcal{A}$ normalized coordinates are used. The known $\Delta : \mathcal{A} \longmapsto \mathcal{T}$ parametrisation is a bijection that relates points of the texture *map* $\mathcal{A}$ to the *surface* $\mathcal{T}$.

Deformation. The template $\mathcal{T}$ undergoes an unknown *quasi-isometric* deformation that results in the unknown surface $\mathcal{S} \subset \mathbb{R}^3$, represented from $\mathcal{T}$ by the unknown map $\Psi\colon \mathcal{T} \to \mathcal{S}$.

Camera projection. The input image is defined as a colour intensity function that contains 3 channels $I\colon \mathbb{R}^2 \to (r, g, b)$, in a discrete pixel grid. The camera model is represented with the perspective projection:

$$(x, y, z) \longmapsto \left(\frac{x}{z}, \frac{y}{z}\right) = (u, v). \tag{1}$$

We assume that the camera is intrinsically calibrated, so we know its aspect ratio, focal length and radial distortion, which in SfT methods is a reasonable and ordinary assumption. The coordinates (u, v) can be easily obtained through the image coordinates and are known as retinal coordinates.

Visible surface region and registration map. The visible surface $\mathcal{S}_{vis} \subset \mathcal{S}$ is represented by all the non-occluded areas of the camera image plane. Through the projection of this region in the image plane, we define a known bidimensional region $\mathcal{I} \subset \mathbb{R}^2$. $\mathcal{I}$ and $\mathcal{S}_{vis}$ are related by a perspective embedding function $X_{vis}\colon \mathcal{I} \to \mathcal{S}_{vis}$ with $X_{vis}(u,v) = \rho(u,v)\,(u,v,1)$. In this embedding function, the function $\rho\colon \mathcal{I} \to \mathcal{S}_{vis}$ is the unknown depth function that allows one to obtain the depth of $\mathcal{S}_{vis}$ for each pixel in camera coordinates. The registration map $\eta\colon \mathcal{I} \to \mathcal{A}$ is an injective map that associates the points of $\mathcal{I}$ to their correspondences in $\mathcal{A}$.

3.2 DNN architecture

Figure 3 shows the general diagram of our SfT solution. It involves two steps. First, we use the *Segmentation Network*, that takes as inputs the image and template texture map and produces a pixel-wise binary segmentation map that classifies each pixel as background or object. Second, we obtain the SfT solution with the *Registration-Reconstruction Network*, that takes the image, the template texture map, and the binary mask from the previous step as inputs and obtains pixel-wise maps that represent the object's registration and depth with respect to the template. Third, we use an ARAP post-processing step to recover the occluded surface parts.

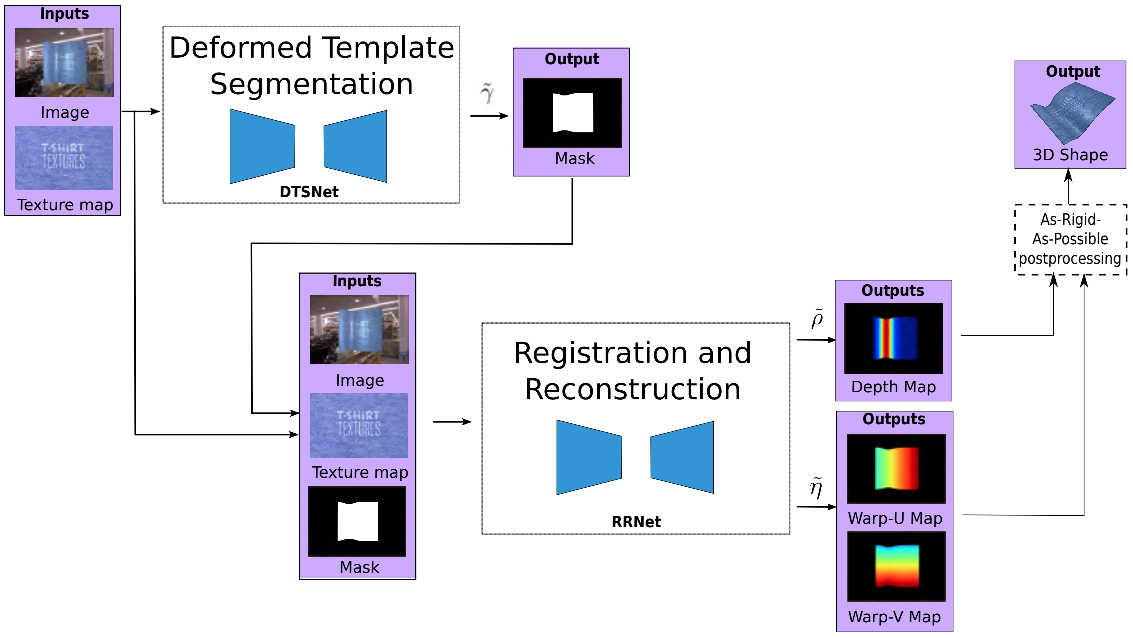


Figure 3: General diagram of our proposed DNN-based Texture-generic SfT method.

3.2.1 Segmentation Network

We propose a DNN model for the segmentation of the deformed template in the input image, named Deformed Template Segmentation Network (DTSNet) and defined with the following mathematical function:

$$\gamma = \mathcal{D}_{DTS}(I, A, \theta_{DTS}), \tag{2}$$

where I is the input image, A is the template texture map and θ_{DTS} is the weight vector. We resize I and A to match the input resolution of 135×240 pixels. The output $\gamma(u,v)$ represents the deformed template segmentation map, with the same size as the input image, where:

$$\gamma(u,v) = \begin{cases} 1 & (u,v) \in \mathcal{I} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

$\mathcal{I}$ is the region of the input image I where the deformed template is visible. Figure 4 and Table 20 of Appendix A show the proposed architecture for DTSNet. It is based on encoder-decoder blocks with skip connections, such as [47, 48], that are commonly used in semantic segmentation. The convolutional, identity and deconvolutional blocks are based on the feed-forward structures in the ResNet50 model [49].

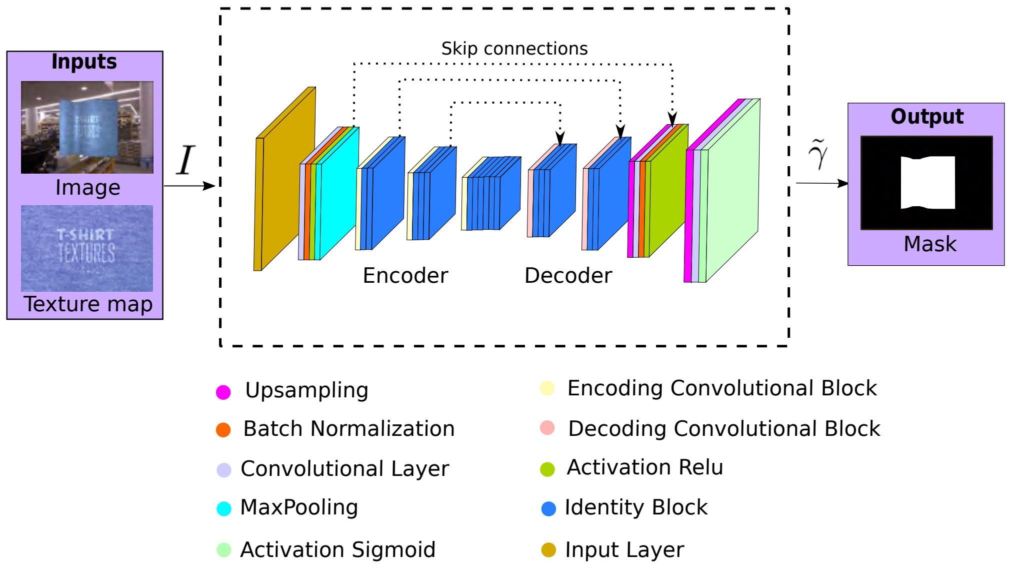


Figure 4: Proposed Deformed Template Segmentation architecture DTSNet.

3.2.2 Registration-Reconstruction Network

The *Reconstruction-Registration Network*, named RRNet, estimates $\rho(u, v)$ and $\eta(u, v)$, the main unknowns in SfT, using as inputs the input image I, the input texture map A and $\tilde{\gamma}$ provided by DTSnet. The RRNet function is denoted as $\mathcal{D}_{RR}$:

$$(\tilde{\rho}, \tilde{\eta}) = \mathcal{D}_{RR}(I, \tilde{\gamma}, A, \theta_{RR}), \tag{4}$$

where the network inputs are resized to 135×240 pixels and θ_{RR} are the network weights. The outputs $\tilde{\rho}$ and $\tilde{\eta}$ are the estimates of the depth and registration maps respectively, defined as follows:

$$\begin{aligned} \tilde{\rho}(u, v) &\approx \begin{cases} \rho(u, v) & (u, v) \in \mathcal{I} \\ -1 & \text{otherwise} \end{cases} \\ \tilde{\eta}(u, v) &\approx \begin{cases} \eta(u, v) & (u, v) \in \mathcal{I} \\ (-1, -1) & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

We propose two DNN models to implement the *Reconstruction-Registration Network*; RRNet is the base model and RRNet-DCT is a lightweight version that can work efficiently on low-cost devices.

3.2.3 The RRNet architecture

Figure 5 and Table 21 of Appendix A show the proposed network architecture for RRNet. Our proposal uses a single encoder and two different decoders for registration and reconstruction. We use skip connections between the encoder and each of the decoders. These connections reinforce the relationship between registration and reconstruction at several scales. This model uses three types of blocks that are shown in Figure 6 and employs identity, convolutional and deconvolutional residual feed-forwarding structures based on the ResNet50 model [49].

3.2.4 The RRNet-DCT architecture

Our proposal for the lightweight RRNet-DCT is based on a single encoder that connects to a shallow decoding layer based on the DCT [50].

The DCT [51] is a linear decomposition of vectors, or matrices in our case, with cosine functions at different frequencies as basis elements. It is widely used to compress 1D signals [52, 53] and images [54, 55, 56, 57]. In image compression, the input image is divided into non-overlapping blocks, and the DCT is computed for each block. Given A, an $M \times M$ block of information (typically $M = 8$), the 2D DCT of A is given by:

$$\Omega = TAT^{\top}, \tag{6}$$

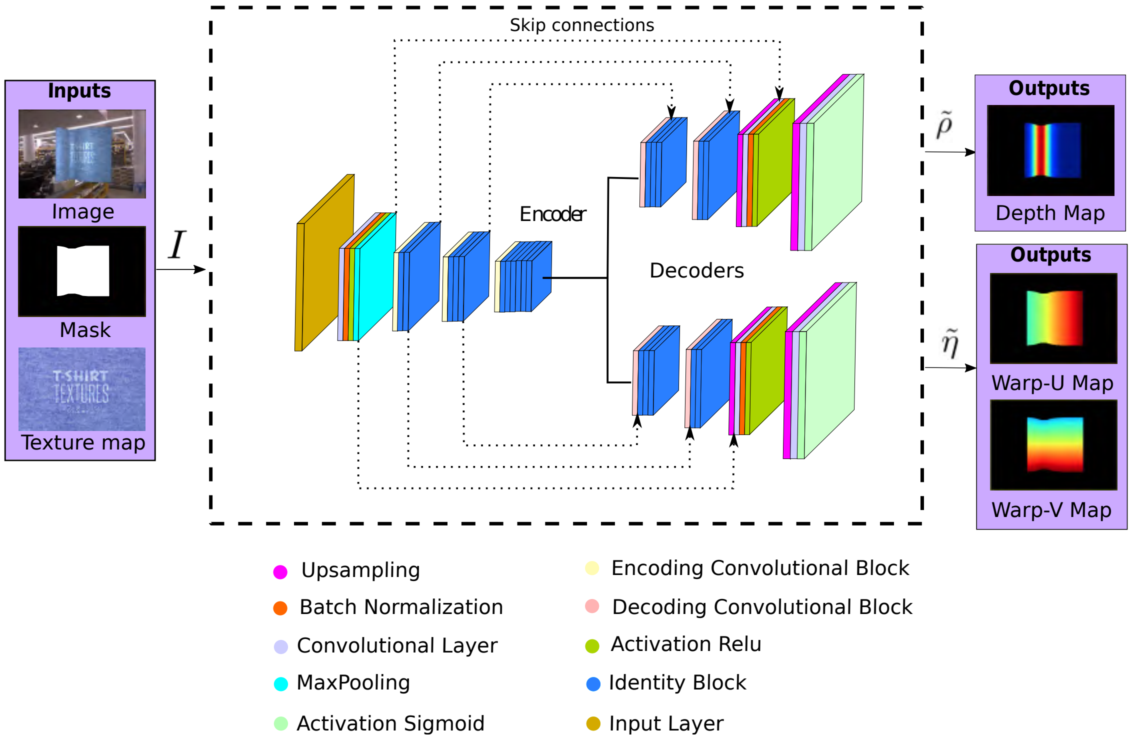


Figure 5: Proposed Registration and Reconstruction RRNet architecture.

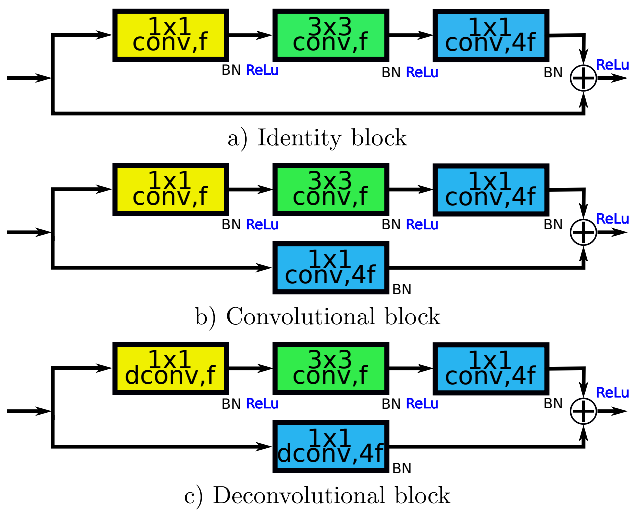


Figure 6: Identity, convolutional and deconvolutional residual blocks.

where Ω is $M \times M$ and $T \in O(M)$ is the following DCT transformation matrix:

$$T_{uv} = \begin{cases} \frac{1}{\sqrt{M}} & u = 1, \quad 1 \leq v \leq M \\ \sqrt{\frac{2}{M}} \cos \frac{\pi(2v-1)(u-1)}{2M} & 2 \leq u \leq M, \quad 1 \leq v \leq M. \end{cases} \tag{7}$$

Conversely, the inverse DCT (iDCT) of Ω is given by $A = T^\top \Omega T$. The principle behind DCT compression is to truncate Ω, resulting in $\tilde{\Omega}$, and then recover an approximate value of A with the iDCT, namely $\tilde{A} = T^\top \tilde{\Omega} T$. In natural images one can recover $\tilde{A} \approx A$ with a small number of non-zero values in $\tilde{\Omega}$. Intuitively, coefficients in Ω associated to low frequencies (top-left corner of Ω) are more important than high frequencies (bottom-right corner of Ω). One of the most common truncation policies is based on sorting Ω from low to high frequencies, by traversing it in zig-zag as Figure 7 shows. In our case, we express the DNN outputs, that is the depth and registration maps, in terms of DCT blocks of size $M = 8$ whose coefficients have been truncated. This way, we can compress the amount of information handled by the DNN, and consequently its parameters. It is thus crucial to design which elements of $\tilde{\Omega}$ are set to zero, so that our DNN outputs are sufficiently accurate to represent the solutions. To do so, we select 20000 depth and registration maps from our datasets, described in Section 4.1, and divide them into blocks of size 8×8. We transform these blocks with the DCT, truncate the resulting coefficients and then recover them back with the iDCT. We evaluate the mean recovery error for registration and reconstruction separately for different numbers of non-zero elements in $\tilde{\Omega}$. Assuming we collect N_b blocks from the 20000 dataset examples, the average compression error is computed as:

$$\epsilon[M_b] = \frac{1}{N_b 64} \sum_{i=1}^{N_b} \|A_i - T^\top \tilde{\Omega}[M_b] T\|_F, \tag{8}$$

where M_b is the number of non-zero elements in $\tilde{\Omega}$, following the zig-zag order from low to high frequencies. Figure 7 shows ϵ for both registration and reconstruction maps in function of the percentage of non-zero elements in $\tilde{\Omega}$, computed as $100\frac{M_b}{64}$. The number of parameters selected in the DCT was fixed to $M_b = 18$ (around 28% of coefficients are non-zero). With this number we obtain an average error of 2 pixels in registration and 3 mm in reconstruction, which is a fair trade-off between accuracy and compression ratio, which is of 70%.

Our RRNet-DCT model uses an encoder stage, similar to the encoder used in RRNet, but without including the skip connections. The output of this encoder is directly the M_b non-zero coefficients in $\tilde{\Omega}$ for each 8×8 block in the output maps. The encoder output size is $(\lfloor \frac{H}{M} \rceil, \lfloor \frac{W}{M} \rceil, M_b N_c) \rightarrow (16, 30, 18 \cdot 3)$, where H and W are the height and width of the input images, M is the size of the DCT block, and N_c corresponds to the number of RRNet-DCT output channels (one for $\tilde{\rho}$ and two for $\tilde{\eta}$). The decoder is a single layer that computes the block iDCT of the encoder outputs to produce $\tilde{\eta}$ and $\tilde{\rho}$. This decoder layer does not add trainable parameters and it is based on the transformation matrix T, which is hard-coded in the layer parameters. The decoder uses zero-padding to complete the vertical dimension of the output, since $\frac{H}{M}$ is not an integer. Figure 8 and Table 22 of Appendix A show the RRNet-DCT diagram and its layer architecture respectively. The types of blocks used in the encoder are similar to the ones used in RRNet, and are presented in Figure 6.

3.2.5 Recovering occluded surface regions with ARAP shape completion

Our RRNet and RRNet-DCT models recover the SfT solution of the surface's visible part $\mathcal{S}_{vis}$, encoded in $\hat{\eta}$ and $\hat{\rho}$. Given that external or self-occlusions can represent an important part of the object, we use a post-processing stage, described in our template-specific method [22], that recovers the whole surface $\mathcal{S}_h$ using the As-Rigid-As-Possible (ARAP) prior. This is a very well known technique in mesh editing [58], and does not require training [7, 59]. Our ARAP post-processing step requires a mesh representation and involves minimizing a non-convex cost function with Gauss-Newton, an iterative second-order optimisation method. In general the algorithm converges after a few number of iterations (less than 10) and can efficiently handle meshes with a high vertex count, that can represent very complex deformations.

4 DNN Training

4.1 Datasets

4.1.1 Templates

We use 28 texture maps, shown in Table 1. We refer to these templates as DSX, with X the number of the texture map. The template shape is a rectangular flat shape. We train with 24 of the 28 different template texture maps and test our methods with 28 different template texture maps (dividing the test on seen and unseen template texture maps).

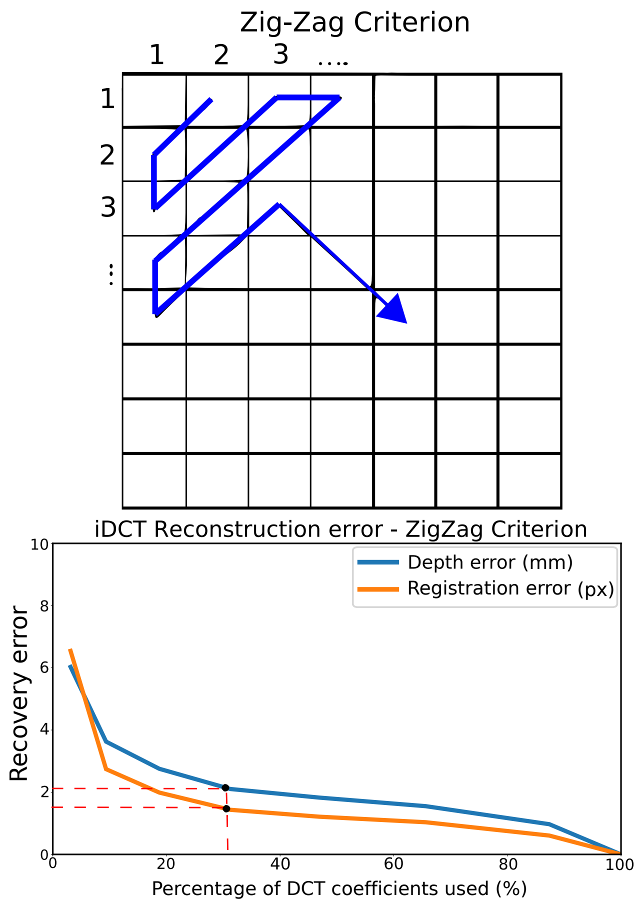


Figure 7: DCT parameters selection criterion. The y axis is the depth error in mm and the registration error in px.

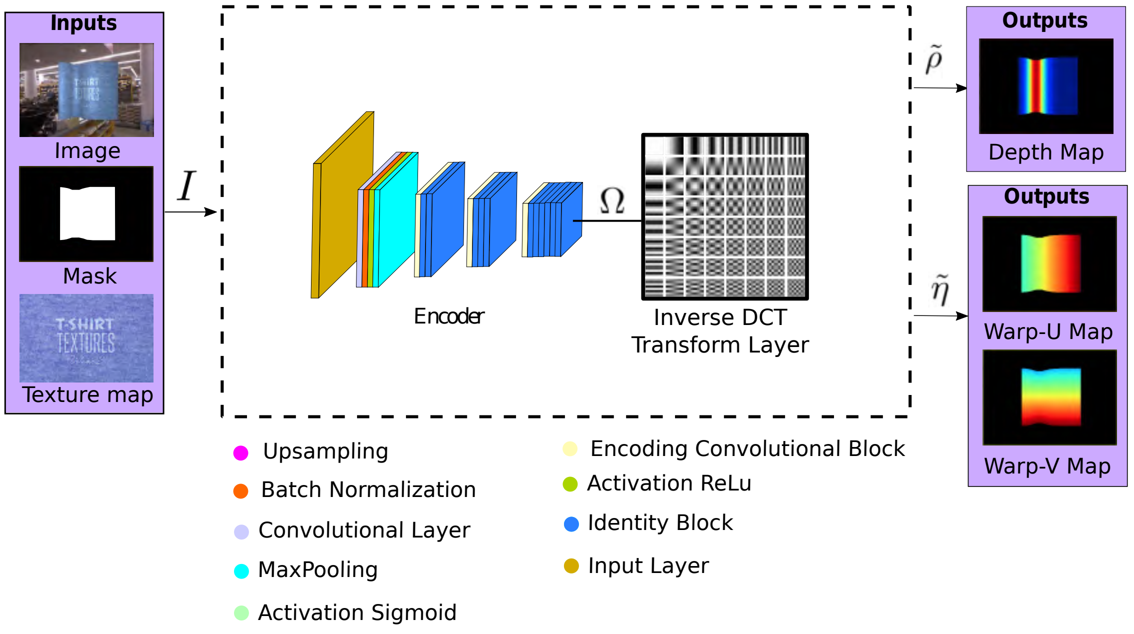


Figure 8: Proposed registration and reconstruction architecture RRNet-DCT.

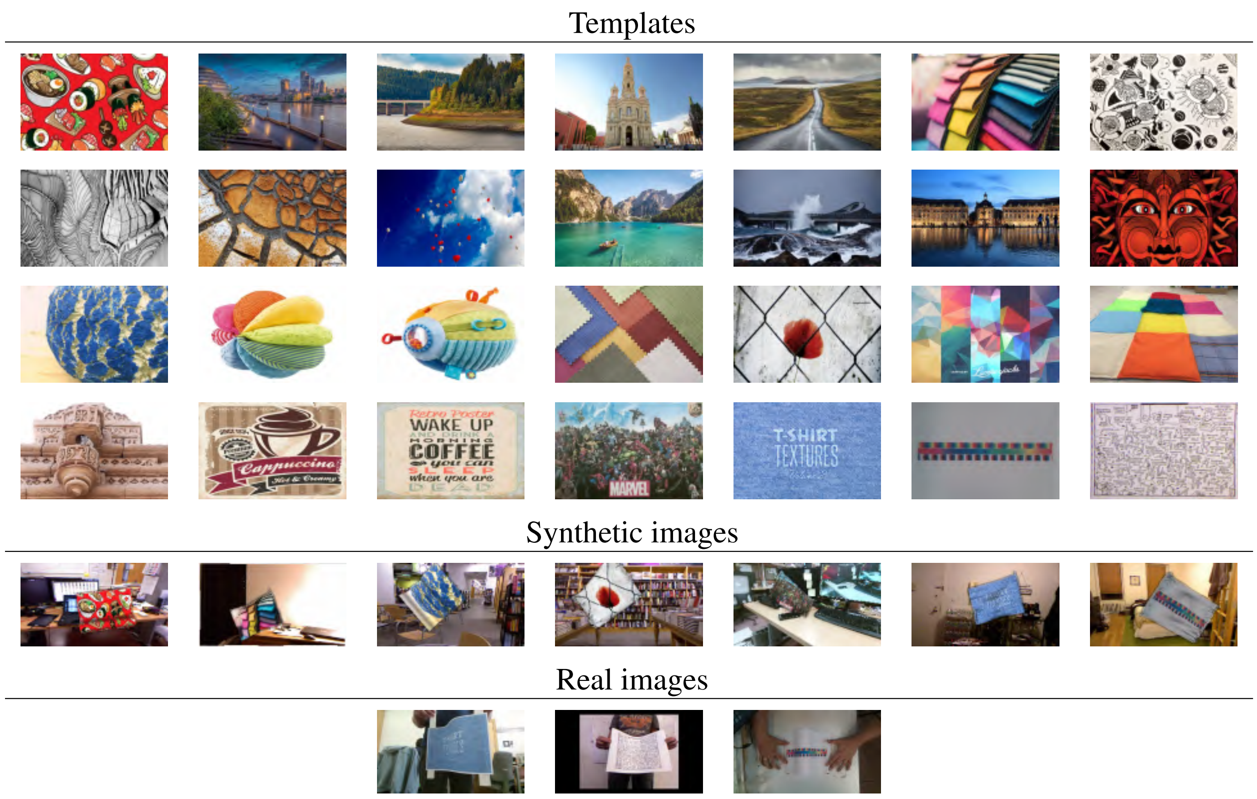


Table 1: Visualisation of templates and input images. Rows 1 to 4 show the 28 texture maps used. Row 5 shows synthetically generated images with simulated deformations. Row 6 shows deformations of the real objects.

4.1.2 Synthetic datasets

We generate 25000 RGB images, depth and registration maps for each of the 28 template texture maps, for a rectangular flat shape of 210×297 mm. The images are generated with random quasi-isometric deformations of the template under random camera viewpoints. We used *Blender* [60], that uses physics-based simulation engines to create synthetic deformations with different stiffness levels using position-based dynamics. We randomly defined anchor points and tensile and compressive forces in randomized 3D directions to generate the deformations. All the simulation parameters will be provided in the supplementary material. The camera pose is generated with a random rotation around the camera optical axis within the interval of $[-\frac{\pi}{4}, \frac{\pi}{4}]$ radians and random translations within the intervals $t_x \in [-150, 150]$ mm, $t_y \in [-150, 150]$ mm and $t_z \in [100, 600]$ mm. We use a distant light model whose illumination angles are parametrised with spherical coordinates drawn randomly in the interval $[-\frac{\pi}{18}, \frac{\pi}{18}]$ radians. We model the light reflectance components using Blender's Lambertian model for the diffuse component and the Cook-Torrance model for the specular component. In addition, the image brightness is varied in the range $[0.9, 1.1]$. We also randomly change the background using images from [61] and simulate occlusions by adding up to four synthetically generated discs with constant random colours and random diameters in the range of $[1, 10]$ pixels at random locations.

4.1.3 Real datasets

We include four real datasets, that correspond to the DS25, DS26, DS27 and DS28 texture maps. We thus have both real data and synthetic data for these templates and we mainly used them for testing our methods. DS25 and DS26 are extracted from [22], that uses a Kinect V2 for the recordings. The data for DS28 was provided by [62], commonly used for testing deformable methods. Finally, the data for DS27 was recorded using an Intel Real sense D435 depth camera. Depth ground-truth is available for these datasets, as all of them used depth sensors to capture data. All the recorded depth maps were aligned with the RGB images and resized to 135×240 pixels. Registration ground-truth is not available in these datasets. We show the intrinsic parameters of the depth cameras used in Table 3. All the real data were generated using videos with different viewpoints and deformations.

4.1.4 Training and testing data splits

We train our DNN models using mainly synthetic data due to the impossibility of obtaining registration labels in real data. Our synthetic dataset is composed of 25000 samples for each texture map (DS1S-DS28S). We save 80% of the samples for training on each texture map, leaving the remaining 20% for testing. When training template-generic approaches, like ours, we use the training set from the first 24 texture maps (*i.e.*, DS1S to DS24S) to train the model, and save templates DS25 to DS28 as unseen textures only for testing the systems and their generalization capacity. We then evaluate their results on synthetic data (*i.e.*, from DS1S to DS28S) and real data examples (*i.e.*, DS25R to DS28R). We train our template-specific method DeepSfT [22] individually for each texture map from DS25 to DS28. We train it with synthetic data and then fine-tune it with real data using a semi-supervised training approach.

Sequence	Samples	Train	Test
DS1S-DS28S	$25000 \cdot 28$	$20000 \cdot 24$	$5000 \cdot 28$
DS25R	3100	2727	373
DS26R	2116	1884	232
DS27R	1285	975	310
DS28R	193	143	50

Table 2: Train and test split for each image sequence. ‘S’ stands for synthetically generated sequences and ‘R’ stands for real sequences obtained with depth cameras.

4.2 Training Process Overview

We train the DTSNet model separately from RRNet and RRNet-DCT. In both cases we use synthetic data for training the DNNs and both synthetic and real data for testing.

4.2.1 Template Segmentation Loss

DTSnet is trained end-to-end using the binary cross-entropy loss for each pixel, as is common in semantic segmentation. The segmentation loss $\mathcal{L}_{seg}$ is defined as:

$$\begin{aligned}\mathcal{L}_{seg}(\theta_{DTS}) = -\sum_{i=1}^{B}\sum_{u,v}^{H\times W} \left(\gamma_i(u,v)\log(\tilde{\gamma}_i(u,v)) + \right.\\ \left.(1-\gamma_i(u,v))\log(1-\tilde{\gamma}_i(u,v))\right),\end{aligned} \tag{9}$$

where $\gamma_i(u,v)$ and $\tilde{\gamma}_i(u,v)$ represent respectively the ground-truth and estimated labels of pixel (u,v) in image i, and B is the number of synthetic images used in the training batch.

4.2.2 Registration and Reconstruction Loss

The registration and reconstruction models RRNet and RRNet-DCT use the following compound loss for training:

$$\mathcal{L}(\theta_{RR}) = \mathcal{L}_{reg}(\theta_{RR}) + \mathcal{L}_{rec}(\theta_{RR}), \tag{10}$$

where $\mathcal{L}_{reg}$ and $\mathcal{L}_{rec}$ are the registration and reconstruction losses respectively. The registration loss is:

$$\mathcal{L}_{reg}(\theta_{RR}) = \frac{1}{B\ H\ W}\sum_{i=1}^{B}\sum_{u,v}^{H\times W} \|\tilde{\eta}_i(u,v) - \eta_i(u,v)\|^2, \tag{11}$$

where $\hat{\eta}_i$ is the estimated registration map, η_i is the labeled registration map, and B is the number of synthetic images in the training batch. The reconstruction loss $\mathcal{L}_{rec}$ has three main terms:

$$\mathcal{L}_{rec}(\theta_{RR}) = \mathcal{L}_d(\theta_{RR}) + \mathcal{L}_n(\theta_{RR}) + \lambda\mathcal{L}_s(\theta_{RR}), \tag{12}$$

where $\mathcal{L}_d$ corresponds to the depth error, $\mathcal{L}_n$ is a normal error and $\mathcal{L}_s$ is the total-variation smoothing prior. The depth error $\mathcal{L}_d$ is the depth Mean Absolute Error (MAE):

$$\mathcal{L}_d(\theta_{RR}) = \frac{1}{B\ H\ W}\sum_{i=1}^{B}\sum_{u,v}^{H\times W} |\tilde{\rho}_i(u,v) - \rho_i(u,v)|, \tag{13}$$

where $\hat{\rho}_i$ is the estimated depth, ρ_i is the ground-truth depth and B is the batch number of image.

The term $\mathcal{L}_n$ computes the error between the estimated normal map and the ground-truth normal map:

$$\mathcal{L}_n(\theta_{RR}) = \frac{1}{B\ H\ W} \sum_{i=1}^{B} \sum_{u,v}^{H \times W} |\tilde{n}_i(u,v) - n_i(u,v)|, \tag{14}$$

where n and $\tilde{n}$ are the normals[1] at pixel (u,v) for ground-truth and estimation respectively. Given ρ (or alternatively $\tilde{\rho}$), the normal in equation (14) is:

$$n = \frac{1}{\sqrt{\rho_u^2 + \rho_v^2 + 1}} \begin{pmatrix} \rho_u & \rho_v & 1 \end{pmatrix}^\top, \tag{15}$$

where ρ_u and ρ_v are the first-order derivatives of ρ with respect to u, v. We use finite difference approximations to compute ρ_u and ρ_v with 3×3 Sobel filter masks S_u and S_v in directions u and v respectively:

$$\rho_u \approx S_u * \rho \qquad \rho_v \approx S_v * \rho, \tag{16}$$

where $*$ is 2D convolution. The approximation in (16) is necessary to make $\mathcal{L}_n$ fully differentiable with respect to θ_{RR}.

Finally, for the smoothing term $\mathcal{L}_s$, we use total variation [63] to reduce the high frequency noise [64] of the reconstruction map:

$$\mathcal{L}_s(\theta_{RR}) = \sum_{i=1}^{B} \|\nabla \tilde{\rho}_i\|_1^2. \tag{17}$$

The smoothing term influence is balanced by a hyper-parameter λ, which is empirically set to 10^{-9} in our experiments.

4.3 Training parameters and implementation details

We choose Adaptive Moment Estimation (ADAM) [65] as the training optimizer. To train the RRNet and RRNet-DCT models, we choose the learning rate $l_r = 20^{-4}$, $\beta_1 = \beta_2 = 0.9$, 60 epochs and a batch size of 15 images. For DTSNet, we configure the training with a learning rate of $l_r = 10^{-6}$, $\beta_1 = \beta_2 = 0.9$, 30 epochs and a batch size of 30. In both cases all the weights are initialized with random uniform sampling [66]. We use Tensorflow [67] for the implementation on an Nvidia GTX1080 GPU. The training of RRNet takes approximately 27 hours, while RRNet-DCT's takes 13 hours.

4.4 Camera intrinsics standardisation

Our proposed system can cope with different cameras at testing, without the need of fine tuning the network weights. This is achieved through the standardisation of camera parameters, as we previously showed in [22]. It is important to highlight that RRNet (or RRNet-DCT) is trained with synthetically generated images with fixed camera intrinsics. After training the network, the test images are adapted before being introduced in the network if the test camera intrinsic parameters differ from those used during training. Otherwise, the network cannot recover the correct depth. We can safely assume that the intrinsic parameters of the input image are known and reliable. This is a very common assumption in state-of-the-art SfT methods. For some sensors, such as those on smartphones or industrial cameras, *e.g.*, Kinect v1, v2 and Intel Real Sense D435, the intrinsic parameters are provided by an application programming interface (API), and this is also accurate. For all other cameras, intrinsic parameter estimation, also called camera calibration, is a very stable, streamlined and reliable process, widely used in SfT [2, 68]. It is important to highlight that if the true intrinsics and the calibration estimated ones are very different, it could produce some degradation in the system's behaviour, but in general, calibrations are sufficiently accurate to not cause any degradation. We emphasize that all prior DNN-based SfT methods require a calibrated camera. Not only that, they also require the same intrinsics for training their networks and testing their networks, which is strongly limiting. We have eliminated this requirement because our method handles different camera intrinsics for training and inference. Our proposal to cope with this problem is to apply an affine transform to the test images to adapt them to the intrinsics in the training camera. The affine transform is represented by the homogeneous affine matrix $A = K_{train} K_{test}^{-1}$, where K_{train} and K_{test} are the intrinsic matrices of the training and test cameras respectively. Once we obtain the converted test images, we clip them about their principal point and use zero padding if needed. This allows us to obtain the input image resolution of the DTSNet and RRNet, which is 135×240. Tables 3 and 4 show examples of intrinsics adaptation with three different cameras and their camera intrinsic parameters.

[1]We compute the normal assuming the camera is weak-perspective and thus the 3D surface is recovered as $(u, v, \rho(u,v))$. The exact normal in the perspective case involves a more complex expression. However, the weak-perspective approximation produces similar results in terms of training the DNN.

Camera	Resolution	f_u	f_v	c_u	c_v
Kinect V2	1920×1080	1057.8	1064.0	947.6	530.4
Kinect V1	640×480	589.3	589.8	321.1	235.5
Intel Real sense D435	1270×720	915.5	915.5	645.5	366.3

Table 3: Camera intrinsics of the different real cameras used in our experiments. We use Kinect V2 for training and all three cameras for testing.

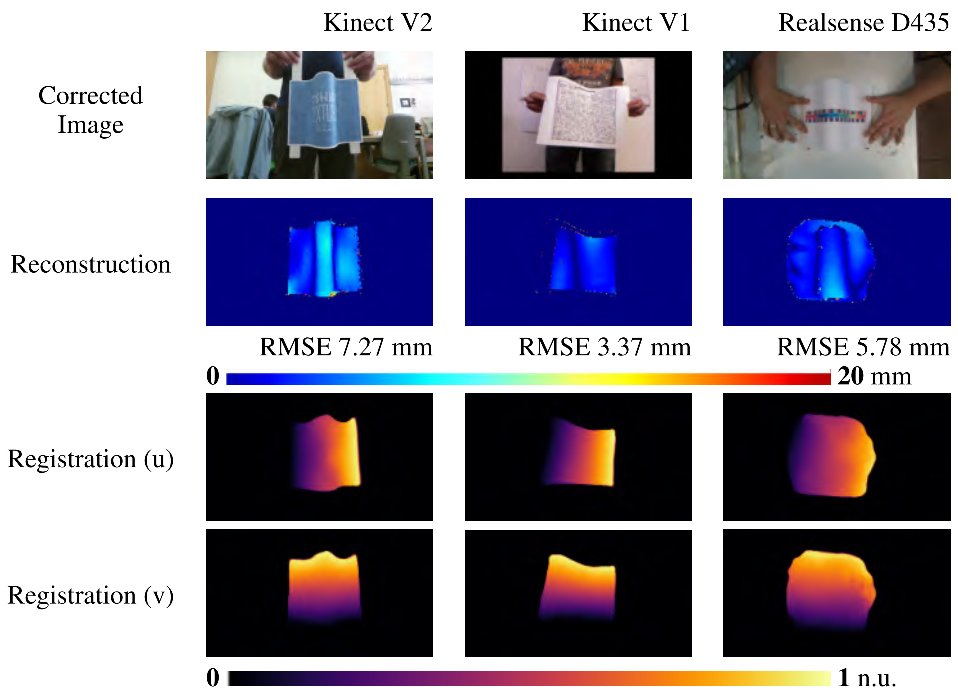


Table 4: Experimental results with different camera models. n.u. stands for normalised units in the template texture map.

5 Experimental Results

5.1 Compared methods and evaluation metrics

We compare our method with both classical and DNN-based state-of-the-art SfT methods.

5.1.1 Classical methods

We compare our SfT solution with three state-of-the-art classical SfT techniques. The first method [8] is referred to as NGO15. It requires to be initialized close to the solution, and was mainly conceived for short-baseline scenarios. NGO15's code is provided by [8]. The second and third methods [4], CH17 and CH17R, have publicly available code. CH17 is a classical SfT method that requires a known texture map to compute the registration between the template and the input image using classical image features. CH17 is based on the analytical local SfT solution and CH17R includes a refinement step based on non-convex iterative optimisation. These methods have important limitations, because they are feature-based, thus requiring a high level of texture detail and requiring a method to match the texture, in this case using SIFT [36] and KAZE [69] features. Additionally, they do not run in real-time and could only reconstruct the visible textured region of the objects. Hence, they could not reconstruct the whole object if it has textureless or occluded regions.

5.1.2 DNN-based methods

We test three state-of-the-art DNN-based SfT methods for comparison. The first method [20] is referred to as HDM-net. This approach is a DNN-based method that does not require the object's specific texture map to be known a priori.

However, it also cannot exploit the object's specific texture for more accurate registration and reconstruction. It is also limited by requiring the template's shape model to be flat and represented by a regular mesh. We have reimplemened this method as its code is not publicly available. We have adapted the input image size and the output mesh size to match the size and complexity of our templates to train with each one of them. The second method [21] is an evolution of [20] and is referred to as IsMo-GAN. It includes an adversarial approach for training. IsMo-GAN's code is provided by [21] and is trained with synthetic examples. The third method is our previous work, DeepSfT [22], which is a template-specific SfT method that requires to be retrained for each template. We trained DeepSfT using both synthetic and real training sets from Table 2, for the templates DS25, DS26, DS27 and DS28. We use the Root Mean Square Error (RMSE) metric to evaluate reconstruction (in mm) and registration (in px).

5.2 Template segmentation test

We compare DTSNet with a classic semantic segmentation method based on U-Net [48], where the template texture map is not included in the inputs. We train both DTSNet and U-Net with the training examples of our synthetic datasets (DS1S-DS24S) of the seen template texture maps. Table 5 and Table 6 show the segmentation results with unseen texture maps during training (DS25S-DS28S). We report the Intersection-Over-Union (IOU) for both models. Table 7 also shows qualitative segmentation results with real data (DS25R-DS28R), since we do not have segmentation labels in these datasets.

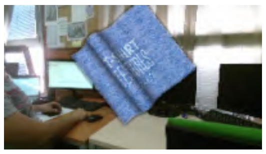
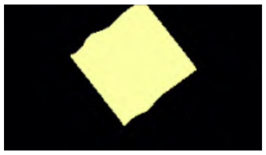
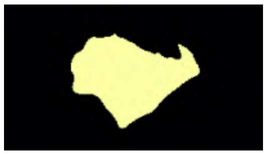
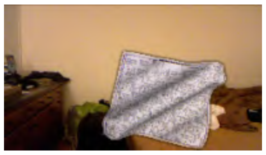
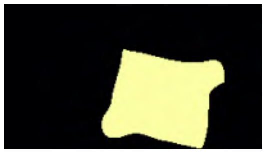
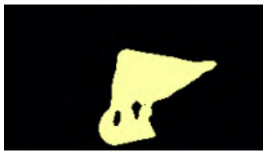
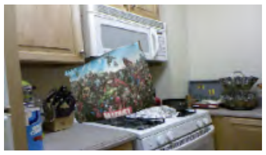
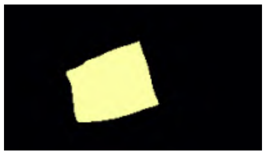
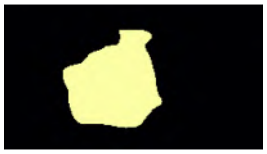
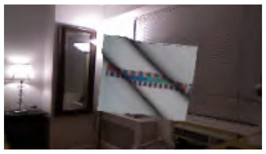
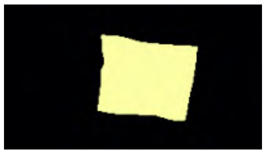
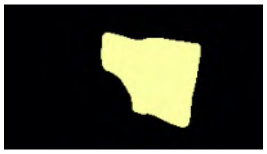
Dataset	Input Image	DTSNet Segmentation	UNET Segmentation	DTSNet IOU	U-Net IOU
DS26S				0.979	0.577
DS28S				0.961	0.635
DS25S				0.970	0.759
DS27S				0.976	0.793

Table 5: Synthetic data, unseen texture maps.

Sequence	Samples	DTSNet IOU	U-Net IOU
DS25S	5000	**0.931**	0.732
DS26S	5000	**0.946**	0.810
DS27S	5000	**0.925**	0.724
DS28S	5000	**0.938**	0.693
Total	20000	**0.935**	0.739

Table 6: Quantitative evaluation on synthetic test data with rectangular templates.

The results show that DTSNet performs significantly better than the classic U-Net model, both qualitatively and quantitatively. This indicates that the problem of segmenting the template is greatly simplified by allowing the segmentation model to adapt to the texture map, as in DTSNet. It can be observed in Table 7 that the DTSNet

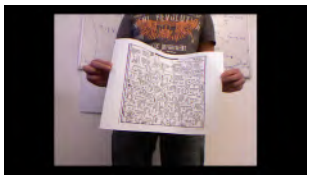
Dataset	Input Image	DTSNet Segmentation
DS25R		
DS26R		
DS27R		
DS28R		

Table 7: Qualitative evaluation of DTSNet using real data and unseen texture maps.

performance worsens with real data, possibly because of the render gap between our synthetic data and the real images, yet being reasonably good, in any case much better than U-Net on synthetic data. Templates DS26 and DS28 show a better segmentation, qualitatively very close to ground-truth. Templates DS25 and DS27 show poorer segmentations, especially DS27. This is explained by the template's poor texture, the occlusions produced by the hands, and the illumination conditions.

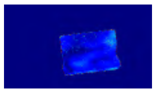
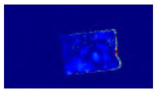
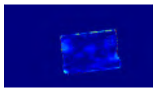
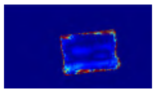
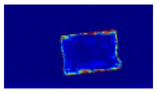
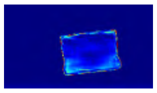
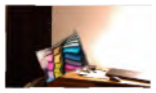
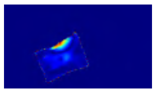
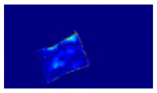
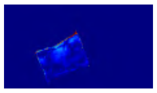
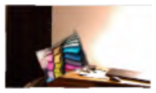
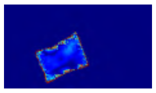
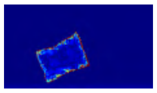
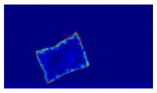
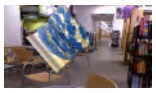
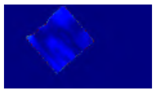
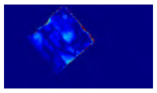
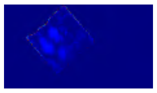
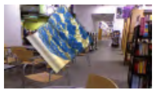
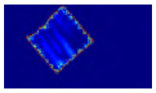
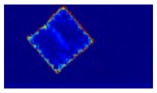
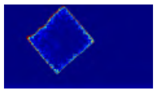
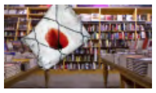
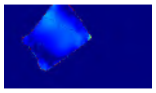
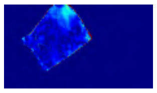
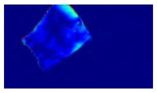
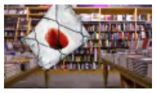
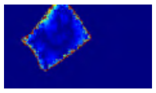
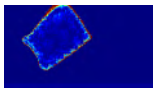
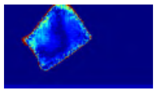
5.3 Testing on trained textures

We show in Tables 8 and 9 the quantitative and qualitative results obtained with four randomly selected templates (DS1, DS6, DS15, DS19) and the average accuracy for all the datasets. It is important to highlight that the methods CH17 and CH17R use registration as an input to solve SfT, and in this case the registration comes from labelled data (GTR), giving an advantage to these methods over the others. After analyzing Table 8, we can see that without using registration labelled data, as in the case of CH17+GTR and CH17R+GTR, RRNet is the best method both in registration and reconstruction error, with a mean error of 3.11 mm in reconstruction and 2.26 pixels in registration. It is followed by its DCT approximation (RRNet-DCT), that logically reduces its performance because of the DCT compression and the smaller model. The third method is the IsMo-GAN with a mean error of 7.25 mm. In this case NGO15, obtains the worst results with a mean error of 19.82 mm, which is expected because it is here tested in a wide-baseline scenario for which it was not originally designed.

In Table 9 we can observe the qualitative results obtained with RRNet and RRNet-DCT over the four randomly selected textures DS1, DS6, DS15 and DS19. In the third, four and fifth columns we visualize the error maps both in registration and reconstruction, decoupled in their u and v components. As can be seen, RRNet performs better than the RRNet-DCT in all cases, which is logical because of the DCT approximation. In particular, it shows larger errors towards the edges of the template, where more high frequencies are filtered with the DCT transformation. In spite of this, RRNet-DCT obtains better results than many state-of-the-art methods and is several orders faster than RRNet and the other methods, as shown in the experiments of section 5.7.

		Registration RMSE (px)		Reconstruction RMSE (mm)						
Sequence	Samples	RRNet-DCT	RRNet	CH17+GTR	CH17R+GTR	NGO15	HDM-net	IsMo-GAN	RRNet-DCT	RRNet
DS1S	5000	2.96	**2.15**	2.14	**1.82**	21.43	8.67	6.34	4.23	3.41
DS6S	5000	3.24	**2.56**	2.14	**1.82**	15.78	7.45	7.03	4.46	2.74
DS15S	5000	3.41	**1.93**	2.14	**1.82**	18.65	9.89	7.21	3.86	3.32
DS19S	5000	3.74	**2.41**	2.14	**1.82**	23.45	11.72	8.43	4.36	2.98
Total	120000	3.33	**2.26**	2.14	**1.82**	19.82	9.43	7.25	4.22	3.11

Table 8: Quantitative evaluation on synthetic data with trained rectangular templates. The results of CH17+GTR and CH17R+GTR have the same error because they use the labelled registration, which is the same for all the databases, as only texture changes.

Dataset	Network	Input Image	Depth error	Registration error (u)	Registration error (v)	Depth RMSE (mm)	Warp u RMSE (px)	Warp v RMSE (px)
DS1S	RRNet					3.13	2.40	1.75
DS1S	RRNet-DCT					5.75	3.24	3.08
DS6S	RRNet					2.91	2.69	2.23
DS6S	RRNet-DCT					5.78	4.52	2.32
DS15S	RRNet					2.07	2.95	1.32
DS15S	RRNet-DCT					4.07	4.32	2.36
DS19S	RRNet					3.54	3.60	2.53
DS19S	RRNet-DCT					5.02	3.96	5.53

Depth error colormap(mm) 0 **20** mm
Registration error colormap(mm) 0 **20** px

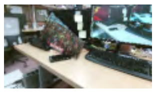
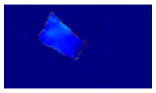
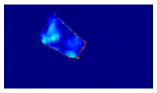
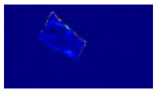
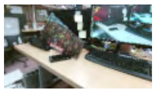
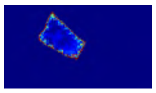
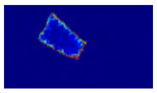
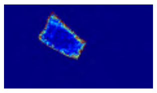
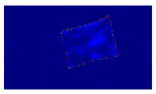
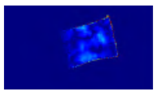
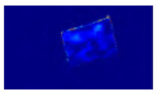
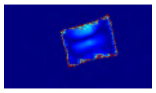
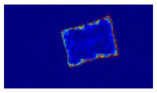
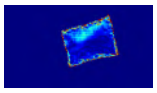
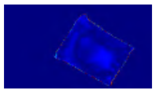
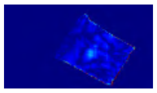
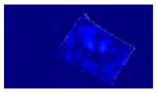
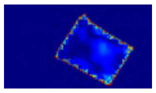
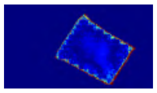
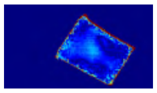
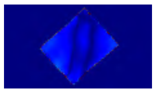
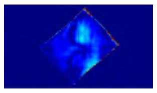
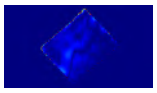
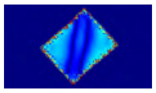
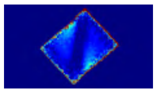
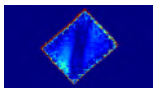
Table 9: Qualitative and quantitative evaluation examples on synthetic seen texture maps.

5.4 Testing on unseen texture maps

In this experiment we test and compare the proposed systems on the datasets using texture maps not seen during training. These are DS25, DS26, DS27 and DS28, which correspond to different types of textures with a high variety in texture content and colour. DS25, DS26 and DS28 are known from previous SfT works [22, 62]. We show in Tables 11, 10, 13, and 14 the quantitative and qualitative results with real and synthetic data in the four proposed datasets.

Excluding CH17+GTR and CH17R+GTR, which are tested using registration ground-truth, Tables 11 and 10 show that DeepSfT obtains the best results. We recall that this method is trained separately on each dataset and is thus expected to perform well on them. Among the rest of DNN-based SfT methods, RRNet and RRNet-DCT obtain the best results with registration and reconstruction errors around 3-4 px and 4-5 mm respectively. This is an excellent result, taking into account that they have not seen the texture maps during training and they have to adapt to them at run-time. The other methods are at least 30% less accurate than our models in this experiment. Again, NGO15 obtains the worst results in this experiment. Table 12 shows the registration and reconstruction results for a related representative input image.

We show the registration and reconstruction results with real data in Tables 13 and 14. Since registration ground-truth is not available, we do not provide registration error with real data. In terms of reconstruction, DeepSfT is the best method

Dataset	Network	Input Image	Depth error	Registration error (u)	Registration error (v)	Depth RMSE (mm)	Warp u RMSE (px)	Warp v RMSE (px)
DS25S	RRNet					4.61	4.82	2.21
DS25S	RRNet-DCT					5.93	4.69	6.07
DS26S	RRNet					1.96	3.14	1.98
DS26S	RRNet-DCT					4.77	4.02	4.71
DS27S	RRNet					1.71	2.51	1.68
DS27S	RRNet-DCT					5.87	3.99	5.79
DS28S	RRNet					2.63	3.72	1.61
DS28S	RRNet-DCT					5.92	4.80	5.39

Depth error colormap(mm) 0 — **20** mm
Registration error colormap(mm) 0 — **20** px

Table 10: Qualitative and quantitative evaluation examples on synthetic unseen texture maps.

		Registration RMSE (px)			Reconstruction RMSE (mm)							
Sequence	Samples	RRNet	RRNet-DCT	DeepSfT	CH17+GTR	CH17R+GTR	NGO15	HDM-net	IsMo-GAN	DeepSfT	RRNet	RRNet-DCT
DS25S	5000	3.27	3.76	**1.60**	2.14	**1.82**	17.23	9.42	8.66	1.93	3.93	4.33
DS26S	5000	2.89	3.52	**1.92**	2.14	1.82	15.45	10.21	8.98	**1.67**	4.11	4.63
DS27S	5000	2.45	3.81	**2.31**	2.14	**1.82**	21.11	8.54	7.21	2.76	4.53	4.82
DS28S	5000	3.87	4.43	**1.54**	2.14	**1.82**	14.54	7.61	7.43	2.10	3.68	4.71
Total	20000	3.12	3.88	**1.84**	2.14	**1.82**	17.08	8.94	8.07	2.12	4.06	4.62

Table 11: Quantitative evaluation on synthetic test data with rectangular templates. The results of CH17+GTR and CH17R+GTR have the same error because they use the labelled registration, which is the same for all the databases, as only texture changes.

with a mean error of 7.32 mm, taking into account that it has been trained on each of the templates, including fine-tuning with real data. In the second and third places we find RRNet and RRNet-DCT, with a mean error of 9.47 mm for RRNet. The next places are occupied by DNN-based methods, the next best being IsMo-GAN followed by HDM-net. The last places are occupied by the classical methods CH17+DOF, CH17R+DOF and NGO15, which obtains errors larger than 15 mm, that is a high error in terms of reconstruction. Although the reconstruction errors have increased with respect to the ones in synthetic data, our methods achieve accurate reconstructions, similar to DeepSfT, in unseen templates. We can see that the DNN-based SfT methods that do not exploit the texture map as an input (*i.e.* IsMo-GAN and HDM-net) obtain rougher reconstructions compared to the proposed methods RRNet and RRNet-DCT, which use the texture maps. By not exploiting the texture map, these methods can only use visual cues that are not specific to the object's texture (in particular shading effects and geometric contours). Those cues are certainly important, but they are not sufficient to precisely register and reconstruct strongly deforming objects.

In terms of segmentation results, we can observe qualitatively that the segmentation masks obtained by DTSNet fit both in shape and location with the deformed template of the input image. In terms of registration both RRNet and RRNet-DCT obtain registration maps in the u and v axes that are qualitatively correct. In the case of RRNet-DCT, higher variations are observed near the edges of the template, indicating a higher error. These variations are probably due to the existence of high frequencies near the edges of the template. We highlight two remarkable cases, the first one

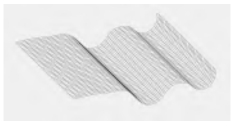
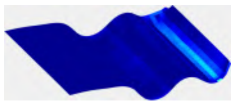
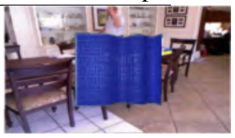
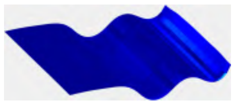
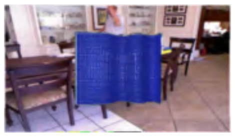
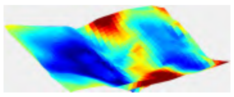
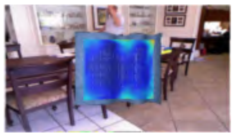
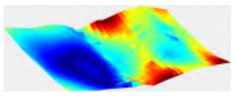
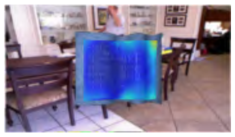
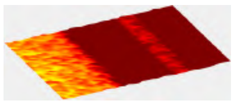
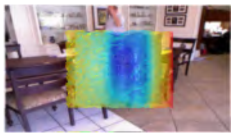
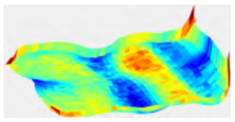
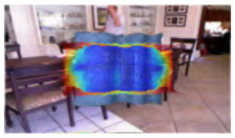
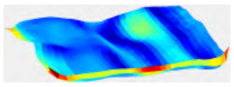
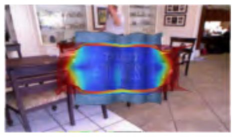
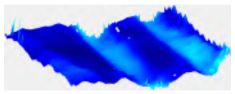
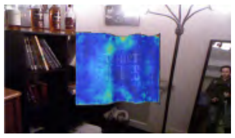
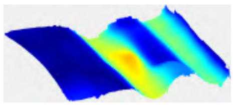
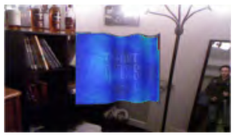
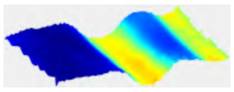
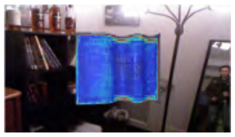
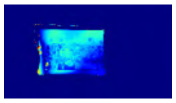
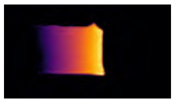
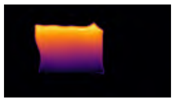
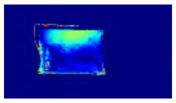
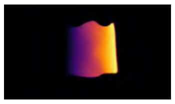
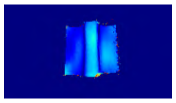
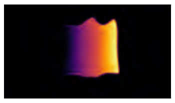
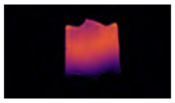
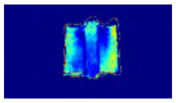
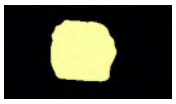
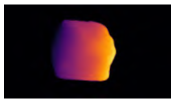
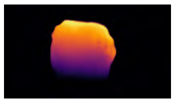
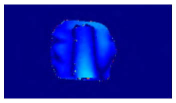
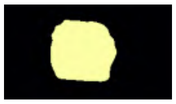
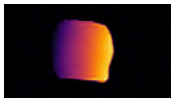
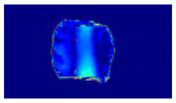
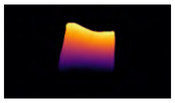
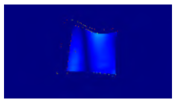
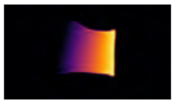
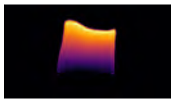
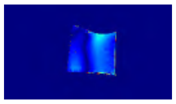
	Ground-truth 3D surface	DS1 Input Image
		
Method	3D Reconstruction & RMSE colormap	Registration ROI & RMSE colormap
CH17+GTR		
CH17R+GTR		
CH17+DOF		
CH17R+DOF		
NGO15		
HDM-net		
IsMo-GAN		
DeepSft		
RRNet		
DCT-RRNet		
RMSE colormap	**0** **30** mm	**0** **1** n.u.

Table 12: Visual comparison of results computed from RRNet and other classical and DNN-based SfT methods. The reconstructions are colored according to RMSE with heat maps (middle column). The registration results are visualised with an overlay of the predicted template shape projected onto the input image. Registration errors are visualised with heat maps (right column). n.u. stands for normalised texture map units.

Dataset	Network	Input Image	Segmentation	Registration map (u)	Registration map (v)	Depth error	Depth RMSE (mm)
DS25R	RRNet						7.61
DS25R	RRNet-DCT						8.09
DS26R	RRNet						7.27
DS26R	RRNet-DCT						8.79
DS27R	RRNet						5.78
DS27R	RRNet-DCT						6.14
DS28R	RRNet						3.37
DS28R	RRNet-DCT						5.21

Error colormap mm 0 20 mm

Table 13: Qualitative and quantitative evaluation examples on real unseen texture maps.

		Reconstruction RMSE (mm)							
Sequence	Samples	CH17+DOF	CH17R+DOF	NGO15	HDM-net	IsMo-GAN	DeepSfT	RRNet	RRNet-DCT
DS25R	373	16.17	15.45	17.43	16.44	13.15	**7.37**	10.13	12.34
DS26R	232	20.34	19.24	23.54	15.87	11.72	**9.51**	11.20	13.01
DS27R	310	27.25	23.17	21.56	13.46	14.53	**5.46**	7.94	10.31
DS28R	50	17.21	16.94	27.43	17.92	15.91	**6.97**	8.63	9.52
Total	965	20.26	18.70	22.49	15.92	13.82	**7.32**	9.47	11.29

Table 14: Quantitative evaluation on real test data with rectangular templates.

is DS27R segmentation, which is worst than DS25, DS26 and DS28, possibly due to the template poor texture and the lighting conditions. The second case is about DS26R with RRNet-DCT, where the registration map in the v axis looks smooth but is incorrect.

5.5 Testing template specific methods on unseen templates

This experiment test non-generic methods like [22] on unseen templates, checking the response of this type of system to different templates than the trained one. We show in Table 15, the results of [22] trained on the template DS26 and evaluated on 2 examples of templates DS25 and DS27.

As can be seen, the system does not respond correctly to the provided inputs, obtaining noise maps with activations. These maps show the raw activations of the network, with values that usually are filtered out thanks to their smaller values, being lower than zero or part of the background.

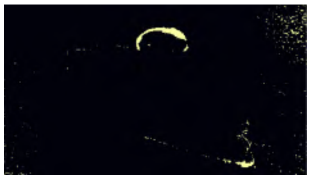
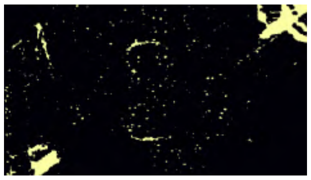
Dataset	Input Image	Depth map	Depth error (mm)
DS25R			437.25
DS25R			349.32

Table 15: DeepSfT evalued on non-trained templates.

5.6 Shape completion evaluation

Table 16 shows results of the DNN reconstruction and the ARAP shape completion post-processing in the templates DS26, DS27 and DS28. We can see in the table from the left to right columns, the input image, the ground-truth, the reconstruction provided by the DNN, the ground-truth and the DNN solution in the same space and finally the shape completion results. The ground-truth and DNN results are provided by coloured point clouds, with the ground-truth in red and the DNN prediction in blue. It is important to highlight that the ground-truth is provided in DS27 by a synthetic simulation and in cases DS26 and DS28 by the ground-truth obtained using Kinect V2 and Kinect V1. The RMSE errors are evaluated over the visible surface regions of the ground-truth after and before using shape completion and denoted by DNN RMSE and ARAP RMSE respectively. The obtained errors are very similar, which means that the shape completion is not hardly affected by the DNN predictions and is recovering the occluded areas according to the visible reconstruction correctly. We highlight the DS27S case which is a very difficult case in which the shape completion provides an approximate representation of the hard occluded regions near the edges, which makes the error rise more than in the DS26R and DS28R cases. We see qualitatively in the last column that the provided 3D shapes are representative of the ground-truth deformations of the object.

5.7 Timing experiments

In these experiments, we compare the frame rate and the number of trainable parameters among all the compared methods. All the compared methods are benchmarked using a desktop PC with a single NVIDIA GTX-1080 GPU. Tables 17 and 18 show the results in terms of frame rate and number of trainable parameters. The frame rate results are evaluated in CPU and GPU, which makes an important difference for the classical methods and shows the level of implementability of the system on low-cost platforms. In terms of speed, it is important to recall that none of the classical methods evaluated (*i.e.* CH17, CH17R, and NGO15) achieve real-time, while in the case of the DNN-based SfT methods DeepSfT, HDM-net, RRNet and RRNet-DCT achieve real-time in GPU, and only RRNet-DCT in CPU. In terms of trainable parameters we only show the DNN-based methods, where we can show that the methods that use less parameters are IsMo-GAN and HDM-net, followed in third place by RRNet-DCT. The most parameter intensive methods are DeepSfT, R50F and RRNet. We recall that RRNet-DCT uses only 21.16% of the RRNet parameters, which in terms of memory is a great improvement.

We conclude that the proposed methods RRNet and RRNet-DCT are faster than the classical and DNN-based methods, without using a high quantity of trainable parameters, especially in RRNet-DCT. These speed monitoring results and lower parameter count show the ability of RRNet-DCT to run on low-cost or embedded devices.

5.8 Monocular depth estimation comparison

We carried out experiments that tries to compare monocular reconstruction methods with RRNet. We propose two state-of-the-art DNN-based monocular reconstruction methods, DenseDepth [30] and BTS [70]. We tested their accuracy in recovering the object's depth map with two types of experiments:

- The first experiment tested the two state-of-the-art methods trained previously in NYUDepth dataset [61], which contains indoor scenes RGB-D images with different types of objects. The depth error is computed

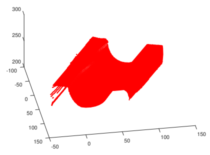
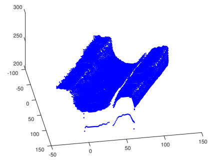
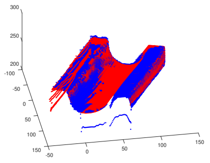
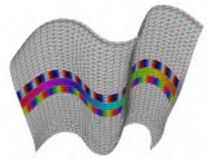
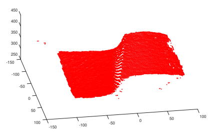
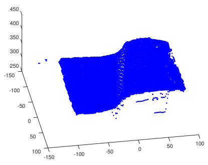
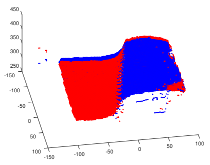
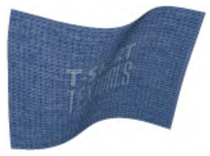
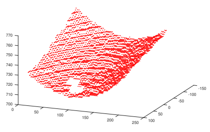
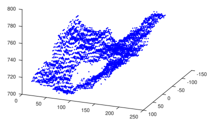
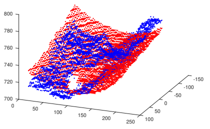
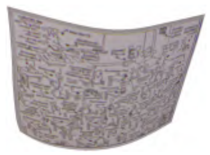
Dataset	Input Image	Ground-truth	DNN reconstruction output	DNN reconstruction output vs GT	Shape completion
DS27S				 DNN RMSE (mm) 4.12	 ARAP RMSE (mm) 5.43
DS26R				 DNN RMSE (mm) 6.23	 ARAP RMSE (mm) 6.41
DS28R				 DNN RMSE (mm) 8.51	 ARAP RMSE 9.12(mm) -

Table 16: Examples of RRNet results before and after shape completion.

	DeepSfT	CH17	CH17R	DOF	NGO15	HDM-net	IsMo-GAN	RRNet	RRNet-DCT
GPU Time (fps)	20.40	-	-	8.84	-	25.12	10.47	62.12	**133.34**
CPU Time (fps)	0.33	0.75	0.19	0.42	0.03	4.89	1.26	17.34	**32.67**

Table 17: Average framerate of the evaluated methods.

	DeepSfT	RRNet	RRNet-DCT	HDM-net	IsMo-GAN
Training parameters	95,282,765	56,907,651	12,291,510	10,079,171	**9,009,022**

Table 18: Number of parameters of the DNN methods.

exclusively on the visible regions of the images and our images are adapted to match the intrinsics of [61], making a fairer comparison. We can observe that the two monocular reconstruction methods have higher errors than RRNet.

- The second experiment is to fine-tune the monocular reconstruction methods with all the real training data, constraining these methods to detect only the real objects of the real dataset.

All the obtained results can be observed in Table 19 where the state-of-the-art fine-tuned versions are named as BTS+FT and DenseDepth+FT. This second case shows an evident reduction of the error, but despite this, it is still several orders higher than RRNet. With these two experiments, we show that BTS and Densedepth achieve a fairly accurate average shape, but are not able to obtain results comparable to RRNet, despite training with only the objects of interest.

6 Conclusions

We have proposed RRNet, the first real-time (CPU and GPU), dense, wide-baseline and texture generic DNN-based SfT solution. No previous DNN-based SfT method adapts to new textures without retraining, which makes our method a step closer to a generic DNN-based SfT method, enabling it for real applications. Future work will aim to generalize RRNet to multiple template shapes, volumetric and thin-shell, by feeding a shape representation as a new DNN input.

Sequence	DenseDepth RMSE	DenseDepth+FT RMSE	BTS RMSE	BTS+FT RMSE	RRNet RMSE	RRNet-DCT RMSE
DS25	78.12	26.72	69.94	22.61	**10.13**	12.34
DS26	93.87	51.25	89.22	14.76	**11.20**	13.01
DS27	101.65	74.38	121.48	65.40	**7.94**	10.31
DS28	395.53	22.47	84.45	12.34	**8.63**	9.52

Table 19: Monocular depth reconstruction methods comparison with RRNet and RRNet-DCT. All the errors are measured in mm.

We will also research non-supervised approaches that do not require labelled data. We will use priors on the temporal information, the spatial smoothness, the deformation model and the texture information of the deformed object. We will study how to deal with uncalibrated cameras and how to self-calibrate at run-time.

References

[1] M. Salzmann, F. Moreno-Noguer, V. Lepetit, and P. Fua, “Closed-form solution to non-rigid 3d surface registration,” *European Conference on Computer Vision*, pp. 581–594, 2008.

[2] A. Bartoli, Y. Gérard, F. Chadebecq, T. Collins, and D. Pizarro, “Shape-from-template,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2099–2118, 2015.

[3] D. T. Ngo, J. Östlund, and P. Fua, “Template-based monocular 3d shape recovery using laplacian meshes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 172–187, 2016.

[4] A. Chhatkuli, D. Pizarro, A. Bartoli, and T. Collins, “A stable analytical framework for isometric shape-from-template by surface integration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 5, pp. 833–850, 2017.

[5] D. Casillas-Perez, D. Pizarro, D. Fuentes-Jimenez, M. Mazo, and A. Bartoli, “The isowarp: The template-based visual geometry of isometric surfaces,” *International Journal of Computer Vision*, May 2021.

[6] F. Brunet, A. Bartoli, and R. I. Hartley, “Monocular template-based 3d surface reconstruction: Convex inextensible and nonconvex isometric methods,” *Computer Vision and Image Understanding*, vol. 125, pp. 138–154, 2014.

[7] T. Collins and A. Bartoli, “Realtime shape-from-template: System and applications.” in *International Symposium on Mixed and Augmented Reality*, 2015, pp. 116–119.

[8] D. T. Ngo, S. Park, A. Jorstad, A. Crivellaro, C. D. Yoo, and P. Fua, “Dense image registration and deformable surface reconstruction in presence of occlusions and minimal texture,” in *IEEE International Conference on Computer Vision*, 2015, pp. 2273–2281.

[9] A. Malti, A. Bartoli, and T. Collins, “A pixel-based approach to template-based monocular 3d reconstruction of deformable surfaces,” in *Proceedings of the IEEE International Conference on Computer Vision*, 11 2011, pp. 1650–1657.

[10] D. Casillas-Perez, D. Pizarro, D. Fuentes-Jimenez, M. Mazo, and A. Bartoli, “Equiareal shape-from-template,” *Journal of Mathematical Imaging and Vision*, vol. 61, no. 5, pp. 607–626, 2019.

[11] A. Malti, R. Hartley, A. Bartoli, and J.-H. Kim, “Monocular template-based 3d reconstruction of extensible surfaces with local linear elasticity,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1522–1529.

[12] A. Malti, A. Bartoli, and R. Hartley, “A linear least-squares solution to elastic shape-from-template,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1629–1637.

[13] B. Koo, E. Ozgur, B. L. Roy, E. Buc, and A. Bartoli, “Deformable registration of a preoperative 3d liver volume to a laparoscopy image using contour and shading cues,” in *Medical Image Computing and Computer Assisted Intervention*. Springer International Publishing, 2017, pp. 326–334.

[14] D. Pizarro and A. Bartoli, “Feature-based deformable surface detection with self-occlusion reasoning,” *International Journal of Computer Vision*, vol. 97, no. 1, pp. 54–70, 2012.

[15] V. Gay-Bellile, A. Bartoli, and P. Sayd, “Direct estimation of nonrigid registrations with image-based self-occlusion reasoning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 87–104, Jan 2010.

[16] T. Collins, P. Mesejo, and A. Bartoli, “An analysis of errors in graph-based keypoint matching and proposed solutions,” in *European Conference on Computer Vision*. Springer, 2014, pp. 138–153.

[17] T. Collins, A. Bartoli, N. Bourdel, and 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*. Springer, 2016, pp. 404–412.

[18] A. Agudo, F. Moreno-Noguer, B. Calvo, and J. M. M. Montiel, "Sequential non-rigid structure from motion using physical priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 5, pp. 979–994, 2016.

[19] A. Pumarola, A. Agudo, L. Porzi, A. Sanfeliu, V. Lepetit, and F. Moreno-Noguer, "Geometry-aware network for non-rigid shape prediction from a single view," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4681–4690.

[20] V. Golyanik, S. Shimada, K. Varanasi, and D. Stricker, "Hdm-net: Monocular non-rigid 3d reconstruction with learned deformation model," *CoRR*, vol. abs/1803.10193, 2018. [Online]. Available: http://arxiv.org/abs/1803.10193

[21] S. Shimada, V. Golyanik, C. Theobalt, and D. Stricker, "IsMo-GAN: Adversarial learning for monocular non-rigid 3d reconstruction," in *Computer Vision and Pattern Recognition Workshops*, 2019.

[22] D. Fuentes-Jimenez, D. Casillas-Perez, D. Pizarro, T. Collins, and A. Bartoli, "Deep shape-from-template: Wide-baseline, dense and fast registration and deformable reconstruction from a single image," 2018.

[23] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[24] A. Dosovitskiy, P. Fischery, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*. IEEE Computer Society, 2015, pp. 2758–2766.

[25] X. Liu, C. Qi, and L. Guibas, "Flownet3d: Learning scene flow in 3d point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 06 2019, pp. 529–537.

[26] W. Wu, Z. Y. Wang, Z. Li, W. Liu, and L. Fuxin, "Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation," in *European Conference on Computer Vision*. Springer, 2020, pp. 88–107.

[27] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2650–2658.

[28] R. Garg, V. K. B.G., G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision*. Springer International Publishing, 2016, pp. 740–756.

[29] F. Liu, C. Shen, G. Lin, and I. D. Reid, "Learning depth from single monocular images using deep convolutional neural fields." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 2024–2039, 2016.

[30] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv e-prints*, vol. abs/1812.11941, 2018. [Online]. Available: https://arxiv.org/abs/1812.11941

[31] R. Alp Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7297–7306.

[32] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1653–1660.

[33] J. Bednarik, P. Fua, and M. Salzmann, "Learning to reconstruct texture-less deformable surfaces from a single view," in *International Conference on 3D Vision*, 2018, pp. 606–615.

[34] D. Yang and J. Deng, "Shape from shading through shape evolution," 2017.

[35] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European Conference on Computer Vision*, vol. 3951, 07 2006, pp. 404–417.

[36] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[37] J. Pilet, V. Lepetit, and P. Fua, "Fast non-rigid surface detection, registration and realistic augmentation," *International Journal on Computer Vision*, vol. 76, no. 2, pp. 109–122, February 2008.

[38] M. Salzmann and P. Fua, "Reconstructing sharply folding surfaces: A convex formulation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 1054–1061.

[39] M. Perriollat, R. Hartley, and A. Bartoli, “Monocular template-based reconstruction of inextensible surfaces,” *International Journal of Computer Vision*, vol. 95, no. 2, pp. 124–137, 2011.

[40] E. Özgür and A. Bartoli, “Particle-sft: A provably-convergent, fast shape-from-template algorithm,” *International Journal of Computer Vision*, vol. 123, no. 2, pp. 184–205, 2017.

[41] N. Haouchine and S. Cotin, “Template-based monocular 3D recovery of elastic shapes using lagrangian multipliers,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, July 2017.

[42] N. Haouchine, J. Dequidt, M.-O. Berger, and S. Cotin, “Single view augmentation of 3D elastic objects,” in *International Symposium on Mixed and Augmented Reality*. IEEE, 2014, pp. 229–236.

[43] A. Agudo and F. Moreno-Noguer, “Simultaneous pose and non-rigid shape with particle dynamics,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2179–2187.

[44] Q. Liu-Yin, R. Yu, L. Agapito, A. Fitzgibbon, and C. Russell, “Better together: Joint reasoning for non-rigid 3d reconstruction with specularities and shading,” *British Machine Vision Conference*, pp. 42.1–42.12, 2016.

[45] A. Tsoli and A. A. Argyros, “Patch-based reconstruction of a textureless deformable 3d surface from a single rgb image,” in *IEEE/CVF International Conference on Computer Vision Workshop*, 2019, pp. 4034–4043.

[46] J. Bednarik, P. Fua, and M. Salzmann, “Learning to reconstruct texture-less deformable surfaces from a single view,” in *2018 International Conference on 3D Vision*, 2018, pp. 606–615.

[47] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation.” *CoRR*, vol. abs/1511.00561, 2015.

[48] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention*. Cham: Springer International Publishing, 2015, pp. 234–241.

[49] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[50] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, 1974.

[51] N.Ahmed, T.Natarajan, and K.R.Rao, “Discrete cosine transform,” *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, 1974.

[52] I. Ito, “A new pseudo-spectral method using the discrete cosine transform,” *Journal of Imaging*, vol. 6, no. 4, p. 15, Mar. 2020. [Online]. Available: https://doi.org/10.3390/jimaging6040015

[53] L. George, “Audio compression based on discrete cosine transform, run length and high order shift encoding,” *International Journal of Engineering and Innovative Technology*, vol. 4, pp. 45–51, 07 2014.

[54] A. Raid, W. Khedr, M. El-dosuky, and W. Ahmed, “Jpeg image compression using discrete cosine transform - a survey,” *International Journal of Computer Science and Engineering Survey*, vol. 5, 05 2014.

[55] M. Servais and G. de Jager, “Video compression using the three dimensional discrete cosine transform (3d-dct),” *Proceedings of the South African Symposium on Communications and Signal Processing*, pp. 27–32, 1997.

[56] M. Alsayyh, D. Mohamad, and W. Abu-Ulbeh, “Image compression using discrete cosine transform and discrete wavelet transform,” *Journal of Information Engineering and Applications*, vol. 3, pp. 54–58, 01 2013.

[57] V. P. Vishwakarma, S. Pandey, and M. N. Gupta, “A novel approach for face recognition using dct coefficients re-scaling for illumination normalization,” in *International Conference on Advanced Computing and Communications*, 2007, pp. 535–539.

[58] O. Sorkine and M. Alexa, “As-rigid-as-possible surface modeling,” in *Proceedings of the fifth Eurographics symposium on Geometry processing*, 2007, pp. 109–116.

[59] S. Parashar, D. Pizarro, A. Bartoli, and T. Collins, “As-rigid-as-possible volumetric shape-from-template,” in *The IEEE International Conference on Computer Vision*, December 2015.

[60] Blender Online Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Blender Institute, Amsterdam. [Online]. Available: http://www.blender.org

[61] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, “Indoor segmentation and support inference from rgbd images,” in *European Conference on Computer Vision*, 2012.

[62] Computer Vision Laboratory, *Deformable Surface ReconstructionDatabase*, Computer Vision Laboratory, Ecole Polytechnique Federale de Lausanne–EPFL. [Online]. Available: https://cvlab.epfl.ch/data/data-dsr-index-php/

[63] L. I. Rudin, S. Osher, and E. Fatemi, “Nonlinear total variation based noise removal algorithms,” *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259 – 268, 1992.

[64] D. Strong and T. Chan, “Edge-preserving and scale-dependent properties of total variation regularization,” *Inverse Problems*, vol. 19, no. 6, pp. S165–S187, nov 2003.

[65] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[66] Y. B. Xavier Glorot, Antoine Bordes, “Understanding the difficulty of training deep feedforward neural networks,” *Procedings of Machine Learning Research*, 2010.

[67] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, and et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[68] A. Bartoli, D. Pizarro, and T. Collins, “A robust analytical solution to isometric shape-from-template with focal length calibration,” in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 961–968.

[69] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, “Kaze features,” in *Proceedings of European Conference on Computer Vision*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 214–227.

[70] J. H. Lee, M. Han, D. W. Ko, and I. H. Suh, “From big to small: Multi-scale local planar guidance for monocular depth estimation,” *CoRR*, vol. abs/1907.10326, 2019.

A Detailed architecture

Layer num	Type	Output size	Kernels/Activation
1	Input	(135,240,3)	–
2	Convolution 2D	(135,240,64)	(7,7)
3	Batch Normalisation	(135,240,64)	–
4	Activation	(135,240,64)	Leaky Relu(alpha=0.1)
5	Max Pooling 2D	(45,80,64)	(3,3)
6	Encoding Convolutional Block	(45,80,[64, 64, 256])	(3,3) stride=(1,1)
7	Encoding identity Block	(45,80,[64, 64, 256])	(3,3)
8	Encoding Convolutional Block	(23,40,[128, 128, 512])	(3,3)
9-10	Encoding identity Block x 2	(23,40,[128, 128, 512])	(3,3)
11	Encoding Convolutional Block	(12,20,[256, 256, 1024])	(3,3)
12-14	Encoding identity Block x 3	(12,20,[256, 256, 1024])	(3,3)
15	Decoding Convolutional Block	(24,40,[1024, 1024, 256])	(3,3)
16	Cropping 2D	(23,40,128)	(1,0)
17-18	Encoding identity Block x 2	(23,40,[1024, 1024, 256])	(3,3)
19	Concatenate Layer	(23,40,768)	
20	Decoding Convolutional Block	(46,80,[512, 512, 128])	(3,3)
21	Cropping 2D	(45,80,128)	(1,0)
22-23	Encoding identity Block x 2	(45,80,[512, 512, 128])	(3,3)
24	Concatenate Layer	(45,80,384)	
25	Decoding Convolutional Block	(90,160,[256, 256, 64])	(3,3)
26-27	Encoding identity Block x 2	(90,160,[256, 256, 64])	(3,3)
28	Bilinear resize Layer	(135,240,64)	
29	Concatenate Layer	(135,240,128)	
30	Convolution 2D	(135,240,64)	(3,3)
31	Batch Normalisation	(135,240,64)	–
32	Activation	(135,240,64)	Leaky Relu(alpha=0.1)
33	Convolution 2D	(135,240,2)	(3,3)
34	Activation	(135,240,2)	Linear
20	Decoding Convolutional Block	(46,80,[512, 512, 128])	(3,3)
21	Cropping 2D	(45,80,128)	(1,0)
22-23	Encoding identity Block x 2	(45,80,[512, 512, 128])	(3,3)
24	Concatenate Layer	(45,80,384)	
Number of parameters	56,907,651		

Table 20: Proposed segmentation architecture for DTSNet.

Layer num	Type	Output size	Kernels/Activation
1	Input	(135,240,3)	–
2	Convolution 2D	(135,240,64)	(7,7)
3	Batch Normalisation	(135,240,64)	–
4	Activation	(135,240,64)	Leaky Relu(alpha=0.1)
5	Max Pooling 2D	(45,80,64)	(3,3)
6	Encoding Convolutional Block	(45,80,[64, 64, 256])	(3,3) stride=(1,1)
7	Encoding identity Block	(45,80,[64, 64, 256])	(3,3)
8	Encoding Convolutional Block	(23,40,[128, 128, 512])	(3,3)
9-10	Encoding identity Block x 2	(23,40,[128, 128, 512])	(3,3)
11	Encoding Convolutional Block	(12,20,[256, 256, 1024])	(3,3)
12-14	Encoding identity Block x 3	(12,20,[256, 256, 1024])	(3,3)
15	Decoding Convolutional Block	(24,40,[1024, 1024, 256])	(3,3)
16	Cropping 2D	(23,40,128)	(1,0)
17-18	Encoding identity Block x 2	(23,40,[1024, 1024, 256])	(3,3)
19	Concatenate Layer	(23,40,768)	
20-A	Decoding Convolutional Block	(46,80,[512, 512, 128])	(3,3)
21-A	Cropping 2D	(45,80,128)	(1,0)
22-23-A	Encoding identity Block x 2	(45,80,[512, 512, 128])	(3,3)
24-A	Concatenate Layer	(45,80,384)	
25-A	Decoding Convolutional Block	(90,160,[256, 256, 64])	(3,3)
26-27-A	Encoding identity Block x 2	(90,160,[256, 256, 64])	(3,3)
28-A	Bilinear resize Layer	(135,240,64)	
29-A	Concatenate Layer	(135,240,128)	
30-A	Convolution 2D	(135,240,64)	(3,3)
31-A	Batch Normalisation	(135,240,64)	–
32-A	Activation	(135,240,64)	Leaky Relu(alpha=0.1)
33-A	Convolution 2D	(135,240,2)	(3,3)
34-A	Activation	(135,240,2)	Linear
20-A	Decoding Convolutional Block	(46,80,[512, 512, 128])	(3,3)
21-A	Cropping 2D	(45,80,128)	(1,0)
22-23-A	Encoding identity Block x 2	(45,80,[512, 512, 128])	(3,3)
24-A	Concatenate Layer	(45,80,384)	
25-B	Decoding Convolutional Block	(90,160,[256, 256, 64])	(3,3)
26-27-B	Encoding identity Block x 2	(90,160,[256, 256, 64])	(3,3)
28-B	Bilinear resize Layer	(135,240,64)	
29-B	Concatenate Layer	(135,240,128)	
30-B	Convolution 2D	(135,240,64)	(3,3)
31-B	Batch Normalisation	(135,240,64)	–
32-B	Activation	(135,240,64)	Leaky Relu(alpha=0.1)
33-B	Convolution 2D	(135,240,2)	(3,3)
34-B	Activation	(135,240,2)	Linear
Number of parameters	56,907,651		

Table 21: Proposed registration and reconstruction architecture RRNet.

Layer num	Type	Output size	Kernels/Activation
1	Input	(135,240,3)	–
2	Convolution 2D	(135,240,64)	(7,7)
3	Batch Normalisation	(135,240,64)	–
4	Activation	(135,240,64)	Leaky Relu(alpha=0.1)
5	Max Pooling 2D	(45,80,64)	(3,3)
6	Encoding Convolutional Block	(45,80,[64, 64, 256])	(3,3) stride=(1,1)
7	Zero Padding	(49,86,256)	(2,3) stride=(1,1)
8	Encoding identity Block	(49,86,[64, 64, 256])	(3,3)
9	Encoding Convolutional Block	(25,43,[128, 128, 512])	(3,3)
10	Zero Padding	(27,47,512)	(1,2) stride=(1,1)
11-12	Encoding identity Block x 2	(27,47,[128, 128, 512])	(3,3)
13	Encoding Convolutional Block	(14,24,[256, 256, 1024])	(3,3)
14	Zero Padding	(16,30,1024)	(1,3)
15-17	Encoding identity Block x 3	(16,30,[256, 256, 1024])	(3,3)
18	Convolution 2D	(16,30,512)	(3,3) Leaky Relu(alpha=0.1)
19	Convolution 2D	(16,30,256)	(3,3) Leaky Relu(alpha=0.1)
20	Convolution 2D	(16,30,128)	(3,3) Leaky Relu(alpha=0.1)
21	Convolution 2D	(16,30,64)	(3,3) Leaky Relu(alpha=0.1)
18	Convolution 2D	$(16,30,18 \cdot 3)$	(3,3) Linear
11	Inverse DCT Transform	(135,240,3)	
Number of parameters	12,291,510		

Table 22: Proposed registration and reconstruction architecture RRNet-DCT.