2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

# Siame-se(3): regression in se(3) for end-to-end visual servoing

Samuel Felton, Élisa Fromont, Eric Marchand

*Abstract*— **In this paper we propose a deep architecture and the associated learning strategy for end-to-end direct visual servoing. The considered approach allows to sequentially predict, in se(3), the velocity of a camera mounted on the robot's end-effector for positioning tasks. Positioning is achieved with high precision despite large initial errors in both cartesian and image spaces. Training is fully done in simulation, alleviating the burden of data collection. We demonstrate the efficiency of our method in experiments in both simulated and real-world environments. We also show that the proposed approach is able to handle multiple scenes.**

## I. Introduction

Visual servoing (VS) is the task of controlling the motion of a robot in order to reach a desired goal or a desired pose using only visual information extracted from an image stream [4]. The camera can be mounted on the robot's end effector or directly observing the robot. Visual servoing usually requires the extraction and the tracking of visual information (usually geometric features) from the image in order to design the control law. While there has been progress in extracting and tracking relevant features, a new approach called direct visual servoing (DVS) emerged a decade ago [17], [6], [9], [8]. It has been demonstrated that the sole pixel intensities of the images can be used to control the robot's motion and that conventional tracking and matching processes can thus be avoided. Nevertheless, due to strong non-linearities in the cost function to be minimized, such direct approaches feature a small convergence domain compared to classical techniques.

To alleviate these issues, machine learning techniques have recently been investigated to learn the relation between the image error and the camera's motion. As for direct approaches, the goal is to avoid explicit feature extraction while increasing the convergence area. In recent years, the use of Deep Learning (DL) has soared, and is now the state-of-the-art on many robotics vision tasks. Lately, DL has been studied for the visual servoing use case. Works [1], [37], [33], [23], [24] already demonstrate interesting advances. Most of these works seek to learn, using reinforcement learning approaches, optimal motion for grasping tasks by directly regressing the joint velocities [23], [24], [13]. Other approaches, suited for positioning tasks, aim to estimate the pose difference between the current and desired camera poses [1], [37], [33].

The latter methods [1], [37], [33] amount to performing position-based visual servoing [36], [4] task (PBVS), where the error is regressed by a neural network. Indeed, PBVS is directly related to the pose estimation issue [27]. In [20], a CNN is used to estimate the camera pose with respect to a fixed scene. The network takes as input a single RGB image, and regresses the pose (camera translation and rotation). The authors expand their work in [18] and devise new losses to achieve better re-localization accuracy by taking into account geometric information and provide task uncertainty estimation. [1] proposes two networks based on standard CNN architectures. They are trained to regress the pose of the camera (using an approach similar to [20]). [33] is similar to [1] in that the two images are given together as different channels as input to a single network which regresses the pose difference. In this case, the chosen network is FlowNet [11], which is trained to perform optical flow estimation. They propose a 3D dataset, containing indoors and outdoors scenes. They show that their scheme is transferable to the real world and can be used to control the motion of a drone. In [12], a deep Siamese network is used to regress the relative pose between two cameras. Siamese networks are used to process multiple inputs in the same way. They have, for example, been successful to perform metric learning [16], [22] and tracking [3], [14]. [37] uses, as us, a siamese network for visual servoing but, they only regress the pose. They perform an early fusion of the two images with a much shallower siamese part and the network has a very high parameter count. They trained and tested their approach with real and very restricted data (related to plugging connectors).

In this paper, we propose a new learning-based framework to achieve visual servoing tasks. It is based on a siamese network that directly regresses the camera velocity without estimating the relative pose. Indeed we argue that relative pose estimation is error-prone mainly due to ill-defined loss functions. The contributions of this paper are:

- rather than estimating the positioning error and achieve a PBVS task, we directly regress the camera velocity and learn in an "end-to-end" manner. We use a loss function that is less sensitive to non-homogeneous scales between the regressed components;
- we propose a new network architecture, Siame-se(3), which is based on a siamese network, that applies a unique processing to multiple inputs (current and desired images);
- training is fully done in simulation and only requires a single image per scene, from which we generate multiple viewpoints to create the dataset;
- our approach is generic and allows us to consider visual servoing in multiple scenes with the same network.

Authors are with Univ Rennes, Inria, CNRS IRISA, Rennes, France Email: {samuel.felton, elisa.fromont, eric.marchand}@irisa.fr

The paper is structured as follows. First, we recall the principle of visual servoing approaches (section II). In Section III, we show how, starting from classical work dedicated to predicting poses, we can, with a relevant loss, directly regress the camera velocity. The architecture of our solution, Siamese(3) is then described in IV. Finally, we demonstrate the validity of our approach with experiments in Section V.

## II. Visual servoing overview

### A. Positioning task by visual servoing

The aim of a positioning task is to reach a desired pose of the camera $\mathbf{r}^*$, starting from an arbitrary initial pose. To achieve this goal, one needs to define a cost function that reflects, in the image space, this error. Most of the time this cost function is an error measure which needs to be minimized. Considering the actual pose of the camera $\mathbf{r}$ the problem can therefore be written as an optimization process:

$$\widehat{\mathbf{r}} = \arg\min_{\mathbf{r}} \mathbf{e}(\mathbf{r}). \tag{1}$$

which consists in minimizing an error $\mathbf{e}(\mathbf{r}) = \mathbf{s}(\mathbf{r}) - \mathbf{s}^*$ between what the camera sees (a set of features $\mathbf{s}(\mathbf{r})$) and what it wants to see (i.e., the desired configuration of the visual features $\mathbf{s}^*$).

The choice of the visual features $\mathbf{s}(\mathbf{r})$ is important since it will determine the camera motion and thus the robot behavior. Different approaches to VS which use various features $\mathbf{s}$ have been considered [4]. In image-based VS (IBVS), 2D primitives such as points or lines are retrieved from the images. In position-based VS (PBVS), the relative pose (position and orientation) between the current and desired camera position has to be estimated. In any case, it requires to extract and track visual information from the image in order to design the control law.

Recent works propose to directly use the information provided by the entire image [7], [6]: photometric visual servoing (PhVS). In [6], a control law was proposed that minimizes the error between the current image and the desired one: the vector of visual features in nothing but the image itself ($\mathbf{s}(\mathbf{r}) = \mathrm{vec}(\mathbf{I}(\mathbf{r}))$ where $\mathrm{vec}(\mathbf{I})$ denotes the vectorization of the image matrix $\mathbf{I}$) and the error to be regulated is the sum of squared differences (the SSD): $\mathbf{e}(\mathbf{r}) = \mathrm{vec}(\mathbf{I}(\mathbf{r})) - \mathrm{vec}(\mathbf{I_0})$. This leads to very precise end positioning, but suffers from a very limited convergence area due to a highly non linear cost function $\mathbf{e}(\mathbf{r})$.

In all cases, the visual servoing task is achieved by iteratively applying a velocity to the camera. This requires the knowledge of the interaction matrix $\mathbf{L_s}$ related to $\mathbf{s(r)}$ that links the variation of $\dot{\mathbf{s}}$ to the camera velocity $\mathbf{v}$ and which is defined as: $\dot{\mathbf{s}}(\mathbf{r}) = \mathbf{L_s}\mathbf{v}$. The control law is then given by [4]:

$$\mathbf{v} = -\lambda \mathbf{L}_{\mathbf{s}}^{+} \mathbf{e}(\mathbf{r}) \tag{2}$$

where $\mathbf{L}_{\mathbf{s}}^{+}$ is the pseudo inverse of $\mathbf{L_s}$ and $\lambda$ a positive scalar.

### B. Deep Learning-based Visual Servoing

DL has been studied for the visual servoing use case. Multiple works [1], [37], [33], [23], [24] already demonstrate satisfying results. These works seek to learn the optimal motion either by estimating the velocity of each of the robot's joints [23], [24], [13] or by estimating the pose difference between the two cameras [1], [37], [33]. These last methods estimate, from the current $\mathbf{I}(\mathbf{r})$ and desired $\mathbf{I}^*$ images, the displacement $\mathbf{\Delta r}$ that the camera has to achieve[1]. This is done considering a CNN using a PoseNet-like network [20]. Nevertheless, this estimation is very sensitive to non-homogeneous scales between translation and rotation. Once the displacement $\mathbf{\Delta r}$ to be achieved is computed from the CNN, it is immediate to compute the camera velocity using a classical PBVS control law [36], [4]:

$$\mathbf{v} = -\lambda \begin{pmatrix} {}^{c^*}\mathbf{R}_c\,{}^{c^*}\mathbf{t}_c \\ \theta\mathbf{u} \end{pmatrix} \tag{3}$$

In such approaches, the quality of the positioning task and camera trajectory is then dependent on the quality of the estimation of the relative pose. In the next section, we will see how to learn the camera velocity in an "end-to-end" manner. Instead of estimating $\mathbf{\Delta r}$, we will directly regress the camera velocity $\mathbf{v}$.

## III. Learning on $SE(3)$ and $se(3)$

Supervised training of neural networks requires the formulation of a loss function that is minimized while training. We explain in the following why it is difficult to define such a loss to successfully learn in $SE(3)$, the space of poses, and how to extend it to the space of $se(3)$.

### A. Deep learning for pose estimation

Poses (and pose differences) are elements of $SE(3) = \mathbb{R}^3 \times SO(3)$, with a real vector $\mathbf{t}$ representing the 3D pose position and a $3 \times 3$ rotation matrix $\mathbf{R}$ encoding the pose orientation. In machine learning, the loss function encodes the error between the model's prediction and the ground truth. The loss function of a network trained to predict a pose in $SE(3)$ would need to carefully balance the contributions of both components of the pose since they are not homogeneous (meters and radians). Previous works [1], [33], [20], [37] use a loss function of the following form:

$$L(\hat{\mathbf{t}}, \hat{\mathbf{q}}) = \|\mathbf{t} - \hat{\mathbf{t}}\|_2 + \beta\|\mathbf{q} - \hat{\mathbf{q}}\|_2 \tag{4}$$

where $\mathbf{q}$ is a minimal representation of $\mathbf{R}$, such as a quaternion or an axis/angle vector. The $\beta$ value is an hyperparameter to be chosen before training and acts as the weighting between the translation and the rotation components of the pose. $\beta$ is a sensitive parameter which can have a strong effect on the final performance of the learning system. Moreover, this parameter is scene-dependent as the impact

[1]The following notations are used: $\mathbf{\Delta r} = ({}^{c^*}\mathbf{t}_c, \theta\mathbf{u})$, where $\mathbf{t}$ describes the translation part of the homogeneous matrix ${}^{c^*}\mathbf{T}_c$ related to the transformation from the current $\mathcal{F}_c$ to the desired frame $\mathcal{F}_{c^*}$, while its rotation part ${}^{c^*}\mathbf{R}_c$ is expressed under the form $\theta\mathbf{u}$, where $\mathbf{u}$ represents the unit rotation-axis vector and $\theta$ the rotation angle around this axis.

in the image space of translation and rotation errors will vary with respect to the depth of the scene. In [19], a new type of loss is introduced for multi-task learning which aims at automatically estimating the weighting of different tasks. When learning on $SE(3)$, one can view the estimation of the two components of the pose as two different prediction tasks. Following this idea, it is shown in [18] that superior result compared to training with the loss in (4) can be obtained. The multi-task loss works by introducing one weighting parameter $\sigma_{\mathbf{x}}$ for each task $\mathbf{x}$, to be optimised alongside the network via gradient descent. For the case of estimating $SE(3)$ poses, the loss is then given by:

$$\mathcal{L}(\hat{\mathbf{t}}, \hat{\mathbf{q}}) = \frac{1}{2\sigma_{\mathbf{t}}^2}\|\hat{\mathbf{t}} - \mathbf{t}\|_2 + \frac{1}{2\sigma_{\mathbf{q}}^2}\|\hat{\mathbf{q}} - \mathbf{q}\|_2 + \log \sigma_{\mathbf{t}}\sigma_{\mathbf{q}} \quad (5)$$

with $\sigma_{\mathbf{t}}$ (respectively $\sigma_{\mathbf{q}}$) the weighting associated to the translation (respectively rotation) error, and where the last term of the loss $\log \sigma_{\mathbf{t}}\sigma_{\mathbf{q}}$ is a regularization term. The added computational cost of this loss is negligible and does not impact inference, as the weights are discarded at inference.

## *B. Towards end-to-end servoing: from $SE(3)$ to $se(3)$*

Wrt. [1], [33], [37] our goal is to directly learn the velocity $\mathbf{v} = (\boldsymbol{\upsilon}, \boldsymbol{\omega})$ required to minimize ${}^{c^*}\mathbf{T}_c$. An advantage of this method is that, since we directly output $\mathbf{v}$, we are not restricted to a single control law, as is the case in works estimating the pose difference between the two cameras. It is thus possible to learn control laws based on IBVS, PBVS or incorporate more complex behaviour into the training data [5]. Using the multitask learning framework [19] presented above, the loss becomes:

$$\mathcal{L}(\hat{\boldsymbol{\upsilon}}, \hat{\boldsymbol{\omega}}) = \sum_{\mathbf{y} \in S}^{S = |\boldsymbol{\upsilon}_{\mathbf{xy}}, \boldsymbol{\upsilon}_{\mathbf{z}}, \boldsymbol{\omega}_{\mathbf{xy}}, \boldsymbol{\omega}_{\mathbf{z}}|} \frac{1}{2\sigma_{\mathbf{y}}^2}\|\hat{\mathbf{y}} - \mathbf{y}\|_2 + \log \prod_{\mathbf{y} \in S}^{S} \sigma_{\mathbf{y}} \quad (6)$$

where $S$ is the set of 4 losses to balance: the combined translation on the $x$ and $y$ axes ($\boldsymbol{\upsilon}_{\mathbf{xy}}$), the translation on the $z$ axis ($\boldsymbol{\upsilon}_{\mathbf{z}}$), as well as rotation on the $x$ and $y$ axis ($\boldsymbol{\omega}_{\mathbf{xy}}$) and on the $z$ axis ($\boldsymbol{\omega}_{\mathbf{z}}$). We argue that, while the translation (or rotation) is expressed in the same unit on all axes, their associated motion is not the same and as such, combining the $\sigma$ weights is not optimal, since estimating one type of translation/rotation may be easier than the other. For example, VS methods are traditionally more tolerant to large rotations around the z (depth) axis, while rotation on the $x, y$ are more constrained.

## IV. Siamese networks for velocity regression

In this section, we detail the architecture of our network, that is trained in an end-to-end manner with the loss presented in (6). We then describe the way that we use simulation to craft the dataset.

The proposed architecture is composed of two networks trained end-to-end. The first network extracts features from the desired and current images $\mathbf{I}$ and $\mathbf{I}^*$. The second network takes as input the difference between the features extracted from $\mathbf{I}$ and $\mathbf{I}^*$ and regresses $\mathbf{v}$ (given as ground truth during training).

*1) Feature extraction network:* To extract the features from $\mathbf{I}$ and $\mathbf{I}^*$, we use a siamese network $\mathcal{S}$. The term *siamese* is employed because the same processing is applied to multiple inputs (i.e. one body – the shared weights – with multiple heads – the outputs [22]). The siamese network $\mathcal{S}$ extracts the features of image $\mathbf{I}$, denoted as $\mathcal{S}(\mathbf{I})$ which takes the form of a vector of fixed size that best describes the image for the task at hand. Since it is the processing for both images, the two feature vectors lie in the same latent space and can be "compared". The general method is independent of the extractor backbone, which can be freely chosen to take into account parameters such as inference speed, accuracy or the transfer learning weights available for a specific architecture (see Section V).

*2) Velocity regression network:* After the feature extraction step, the velocity $\mathbf{v}$ can be regressed. Drawing inspiration from the formulation of the error in visual servoing, we give as an input to the regression network $\mathcal{S}(\mathbf{I}) - \mathcal{S}(\mathbf{I}^*)$. Using a subtraction is natural here since the embeddings coming from $\mathcal{S}$ lie in the same space and subtraction expresses a notion of signed distance between the vectors' components. Concatenating $\mathcal{S}(\mathbf{I})$ and $\mathcal{S}(\mathbf{I}^*)$ is another possible way of fusing information, but experiments showed that results were inferior, with the added drawback that the number of trainable parameters of the regression network is higher.

When considering Equation (2), the computation of $\mathbf{v}$ is dependent on an interaction matrix $\mathbf{L_s}$ specific to the control law, the formulation of the features $\mathbf{s}$ and their estimated values. Here, we do not know the interaction matrix expressing $\frac{\partial \mathcal{S}(\mathbf{I})}{\partial \mathbf{r}}$. This is why we regress $\mathbf{v}$ with a nonlinear function of the error $f(\mathbf{e})$. This function $f$ is built from multiple Fully Connected (FC) layers (see Fig. 1), with ReLU [28] activations to make $f$ non-linear. The last FC layer has no activation function, as the values of $\mathbf{v}$ are unbounded. We choose this simple architecture because we do not make any assumption about the shape of the extracted features and their relations.

## *A. Training*

As deep learning requires large amounts of data and collecting such data on a real robot can be expensive, we resort to simulation to train our network. As in [1], we simulate the projection of a planar scene $\mathbf{I}^*, \mathbf{I}$ for two different poses. We simulates the desired control law from these poses using equation (3) and feed the network using the two images and the corresponding computed camera velocity. A potentially infinite number of viewpoints can then be generated to constitute the training dataset. Data are generated on the fly so as to avoid potential overfitting to specific viewpoints.

To help transfering from simulation to real-world, we use lighting and occlusion augmentations. Indeed, it has been shown that networks trained in simulated environments tend to transfer poorly to real world situations [35], [31], and that augmentation can bridge the gap between simulation and real world sample distributions. To ensure that the samples are relevant, they are further filtered so that there is a minimal

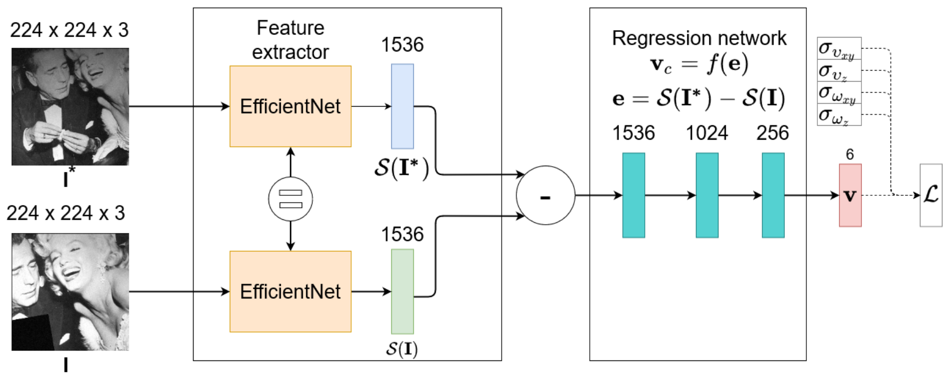

Fig. 1: Diagram of the proposed Siame-se(3) architecture. The images are first processed by a siamese feature extractor to compute $\mathcal{S}(\mathbf{I})$, $\mathcal{S}(\mathbf{I}^*)$. These features are then subtracted to form $\mathbf{e}$, from which we regress the velocity $\mathbf{v}$. To compute the loss $\mathcal{L}$ during training, we add free weights $\boldsymbol{\sigma}$ as done in [18], [19] so as to balance the optimized losses.

amount of overlap between $\mathbf{I}$ and $\mathbf{I}^*$ (i.e. the cameras are looking at the same part of the scene).

We argue that the proposed approach has multiple benefits. First, regressing the velocity directly allows us to have a generic feature representation, with the architecture being control law-independent. Indeed, although our network was trained with a PBVS control law, other control laws (2D, 2 1/2D,...) could have been considered. One needs only change the features used when generating $\mathbf{v}$, as the network architecture does not require any modification. Because the features are learned, the time usually spent engineering the feature extraction is spent training a network. Compared to other learning approaches, we rely on a siamese network to regress a compact image representation that can then be stored.

## V. Experiments and Results

This section details the dataset design, the training process and the experiments performed in simulation and on an actual 6-DOF robot. We consider a positioning task, with the robot being moved to a different starting pose, and the goal being to move back to the desired pose.

### A. Network training

As a feature extractor siamese network, we choose a recent state of the art neural network architecture, EfficientNets [34], and apply transfer learning [29] to benefit from the training of EfficientNets on ImageNet [10] to initialise the weights of our architecture. EfficientNets is chosen because its ratio accuracy/parameters is higher than other architectures and this alleviates the risk of overfitting. Besides, the number of Floating Point Operations Per Seconds is also lower (1.8B for the EfficientNetB3 vs 4.1B for the nowadays very popular ResNet-50 [15]), making inference during servoing faster and more efficient. Finally, It has been shown in [34] that the transfer learning capabilities of the EfficientNet are better than other standard architectures, albeit on other classification datasets. There are multiple versions of EfficientNet, going from B0 to B7, with an increasing width, depth and resolution scaling. As a trade-off between accuracy and computation, we settled on the B3 version, which has 10.5M parameters. A ResNet-50 was also evaluated, but the results were inferior.

The network is initialized with the pretrained ImageNet weights, which are not frozen during training to allow potentially large readaptation of the network for our regression task. While the siamese backbone is an EfficientNetB3, which works on images of dimensions $300\times300$, we directly feed images of size $224 \times 224$.

The network is trained on a single RTX 2080Ti for 50 epochs with 100k samples per epoch and batch size 25, for a total of 5M image pairs seen during training. We use the Adam optimizer [21] with a learning rate of 1e-4 for the model weights and 5e-4 for the loss weights. We initialise these to $\sigma^2_{\boldsymbol{v}_{\mathbf{xy}}} = \sigma^2_{\boldsymbol{v}_{\mathbf{z}}} = \sigma^2_{\boldsymbol{\omega}_{\mathbf{xy}}} = \sigma^2_{\boldsymbol{\omega}_{\mathbf{z}}} = e^0 = 1.0$. While these values are initialized higher than the starting errors, we note that this acts as a learning rate warm-up and helps the convergence.

At inference, since $\mathbf{I}^*$ is captured at the start, the feature extraction to get $\mathcal{S}(\mathbf{I}^*)$ is only ran once and $\mathcal{S}(\mathbf{I}^*)$ is stored, heavily reducing the computational cost. On a RTX 2080Ti, the network runs in ≈20ms per iteration. Common neural network tricks, such as folding batch normalization into convolutional layers or using mixed precision, may be used to deploy Siame-se(3) onto embedded devices. The network is trained with PyTorch [30] and the data is generated with ViSP [26]. The training takes approximately 18 hours on a single GPU, making it easy to deploy rapidly. The storage requirements are weak, with the model taking around 50MB of memory vs 550Mb for the VGG.

### B. Simulation experiments

We first compare the impact of different settings on the servoing results by running experiments in simulation. The approach is also compared with photometric VS [7] and a another recent direct VS approach where the images are transformed in the frequency domain [25]. The considered scene is the one used in the experiments of Fig. 4. We run 1000 trajectories, with the mean and standard deviations of the starting translation and rotation errors of 12.5cm ± 7.5cm/17.5°± 12°. Results reported in Table I show

| Method | Translation error in cm | Rotation error in degrees | Convergence, % | Trajectory absolute position error in cm | Trajectory absolute rotation error in degrees |
|---|---|---|---|---|---|
| DVS [7] | $0.2 \pm 1.5$ | $0.1 \pm 0.9$ | 34.8 | $2.19 \pm 3.77$ | $1.97 \pm 3.47$ |
| DCT-VS, $k = 20$ [25] | $0.01 \pm 0.001$ | $0.004 \pm 0.003$ | 38.6 | $1.41 \pm 3.5$ | $1.12 \pm 3.07$ |
| Point-based IBVS, ORB features [32] | $2.3 \pm 4.6$ | $2.7 \pm 7.1$ | 43.4 | $2.49 \pm 4.51$ | $2.8 \pm 7$ |
| [1]: VGG backbone, learned loss [19], regressing $\mathbf{\Delta r}$, Eq (5) | $5.4 \pm 2.0$ | $3.43 \pm 1.65$ | 76.9 | $5.08 \pm 1.91$ | $3.28 \pm 1.44$ |
| Siame-SE(3), Eq (5) | $1.12 \pm 0.43$ | $0.79 \pm 0.3$ | 98 | $1.05 \pm 0.33$ | $0.74 \pm 0.23$ |
| Siame-se(3), Eq (6) with two task weights | $1.27 \pm 0.44$ | $0.94 \pm 0.3$ | 97.8 | $1.15 \pm 0.36$ | $0.85 \pm 0.25$ |
| Siame-se(3), Eq (6) | $1.0 \pm 0.48$ | $0.75 \pm 0.3$ | 98.9 | $1.08 \pm 0.48$ | $0.77 \pm 0.3$ |

TABLE I: Results of different methods on a set of 1000 simulated trajectories in a single scene setting. The end positioning errors of the servoing, as well as the mean trajectory errors with respect to a PBVS control law are reported in centimetres and degrees for the cases that converge.

the servoing results of methods [7], [25], as well as a re-implementation of the network trained in [1] and various versions of our siamese network. The IBVS method is based on ORB [32] keypoints. The network denoted as Siame-SE(3) has the same design as Siame-se(3), but is trained to regress the pose difference $\mathbf{\Delta r}$. "Siame-se(3) with two task weights" is the velocity regressing Siame-se(3), but only two weights are used during training: one for translation and one for rotation instead of the 4 used in Eq. 6. Direct servoing methods such as DVS and DCT-VS suffer from low convergence rates, but have the advantage of having a very precise end positioning. Compared to our Siamese networks, the VGG from [1] diverges more often and tends to remain far from the desired position. For our siamese architecture, the results show that regressing the velocity $\mathbf{v}$ works well and achieves similar positioning accuracy to when it is trained to output $\mathbf{\Delta r}$ with the advantages discussed before (genericity and independence to the control law). It can also be seen that our siamese network trained to imitate PBVS succeeds and provides a trajectory that is closer to the ground truth (straight line in the Cartesian space) than with other methods, which either do not have this objective or, in the case of the VGG, realize it poorly, as attested by the trajectory statistics in Table I. While photometric methods will often diverge, our method converges in almost all cases with a small remaining error that is fairly constant. Running photometric VS[7] after any of the network-based methods results in a positioning error below the millimetre/tenth of a degree on all samples, illustrating the complementarity of the methods.

Finally, Fig. 2 examines the convergence behaviour of different methods as a function of the overlap between the starting and desired images. While traditional methods benefit from higher overlap, Siame-se(3) manages to reliably converge even when the overlap is low. This suggests that in the monoscene case, Siame-se(3) learns a global representation akin to a pose.

## C. Multiscene experiments

To further test our method, we train on multiple scenes at the same time: this time the reference scene is drawn randomly. To build the dataset, we select multiple images from the ImageNet dataset. We restrain ourselves to a single class of ImageNet [10]: the car class. By doing this, we should have common information between the scenes, which is often the case when applying VS to the real world. Because

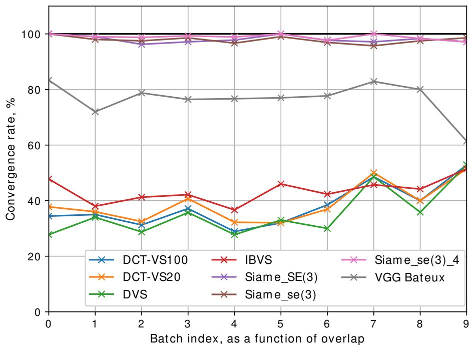

Fig. 2: Convergence rate vs overlap between starting and desired images. The first batch has very low overlap (0-10%), while the last one has a higher one (90-99%).

of the added difficulty, we train for 75 epochs. We train several networks, each on different numbers of scenes and with or without color, to demonstrate the scalability (to multiple scenes) of the method, as well as the potential benefit of incorporating chrominance information. We notice that as the number of scene grows, training becomes harder (the training loss does not converge to zero). To remedy this, we use curriculum learning [2]: we progressively add ramp up augmentation over the first 10 epochs. Fig. 3 highlights the obtained results, and reports the results of servoing on the scenes the networks were trained on. It can be seen that adding color information is indeed helpful. As the number of scene grows, the convergence rate decreases, especially for the grayscale version of the network. The end positioning error is also higher, but remains small enough that other servoing methods can be used afterwards to correct it.

## D. Experiments on a 6DOF robot

Fig. 4 demonstrates a real-world servoing example. We place the desired pose (see Fig. 4(b)) 80cm above the center of scene and move to another pose to get Fig. 4(a). The initial displacement is very large, with $\mathbf{\Delta r}(0)$ = (45cm, 19cm, 9cm, 34°, -20°, -47°), and $\mathbf{I}(0)$ contains large specularities. We run 500 iterations, with $\lambda = 0.1$. The final error is $\mathbf{\Delta r}$ = (0.5cm, 0.2cm, 0.06cm, 0.3°, -0.33°, 0.12°) and the end image difference is displayed in Fig. 4(c). While the end positioning is not perfect, it can be trivially corrected by applying photometric VS [7] in the last few iterations, since we are close to the desired pose and the image error is low. Fig. 4(d) shows the values of $\mathbf{v}$ regressed by the network,

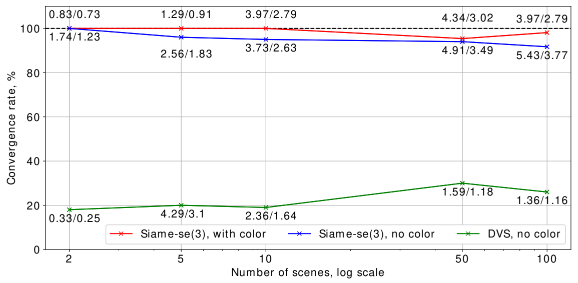

Fig. 3: Multiscene results: for different number of scenes, the convergence rate of our method is reported. We also report the convergence rate of photometric VS, a non learning-based approach. For each point we also report the end positioning error for samples that converged in the format cm/°.

which are smooth, as attested by the almost exponential decrease of the error in Fig. 4(e). The 3D trajectory in Fig. 4(f) highlights the straightness of the trajectory. Finally, we compare in Fig. 4(g) the norm of the values of our formulation of the error $\mathbf{e} = \mathcal{S}(\mathbf{I}) - \mathcal{S}(\mathbf{I}^*)$ with the SSD of $\mathbf{I}$ and $\mathbf{I}^*$ which puts in evidence a correlation between the behaviour of the two.

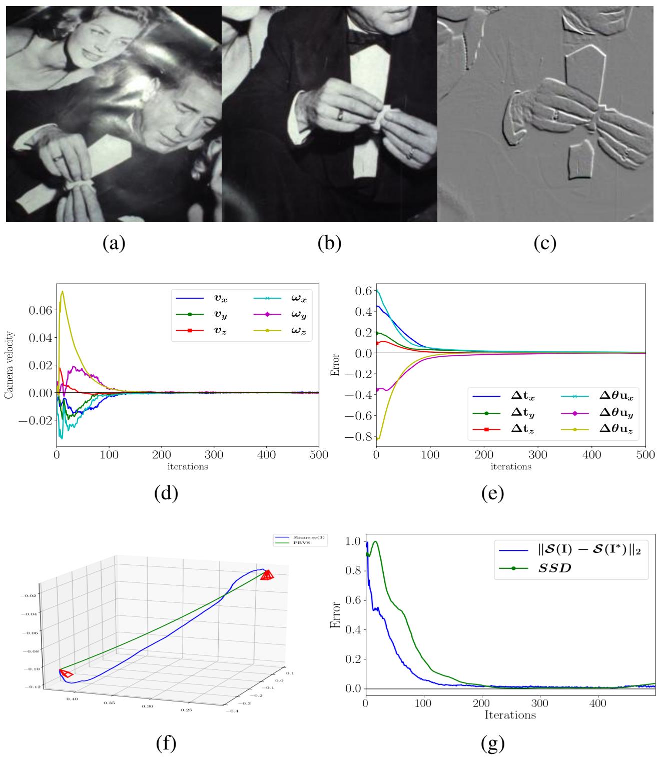

Fig. 4: (a) Starting image $\mathbf{I}$. (b) Desired image $\mathbf{I}^*$. (c) Final difference image. (d) Velocities computed by the network. (e) Pose difference $\mathbf{\Delta r}$. (f) 3D trajectory performed by Siame-se(3) in blue compared to a ground truth PBVS trajectory. (g) Normalised comparison of the Sum of Squared Differences (SSD) and the norm of $\mathcal{S}(\mathbf{I}) - \mathcal{S}(\mathbf{I}^*)$

In our second experiment, we test our multiscene network on (printed) A3-sized scenes. During training, we set the average distance from the poster to 30cm. As in our simulation setting, we train on 100 different images of cars. For this experiment, we picked one scene in the training set, and ran the servoing with the 6DOF robot on this printed scene. To illustrate the complementarity of our method and DVS, we first run Siame-se(3) for 500 iterations, then correct the remaining error with DVS. The starting and desired image are presented in 5(a) and (b). There is very low overlap between the two. Indeed, the initial error is $\mathbf{\Delta r}(0)$ = (-17.9cm, 1cm, 0.5cm, 4.3°, 18.4°, 44.6°). After our method is over, the remaining error is $\mathbf{\Delta r}(500)$ = (-1.6cm, -0.4cm, 0.3cm, -0.89°, 3.16°, -0.76°). The remaining error in image space is displayed in 5(c). In general, the remaining error after VS with Siame-se(3) is present because of compensations between the translations and rotations on the x and y axes, as the diminution of the error in the image is mostly the same between the two types of motion. From there, DVS is applied and the final distance from the desired pose is 0.06cm/0.13°. Our method accomplishes most of the motion, providing a fairly straight trajectory and is able to bring the end effector close enough to the desired pose so that it is in the convergence cone of DVS.

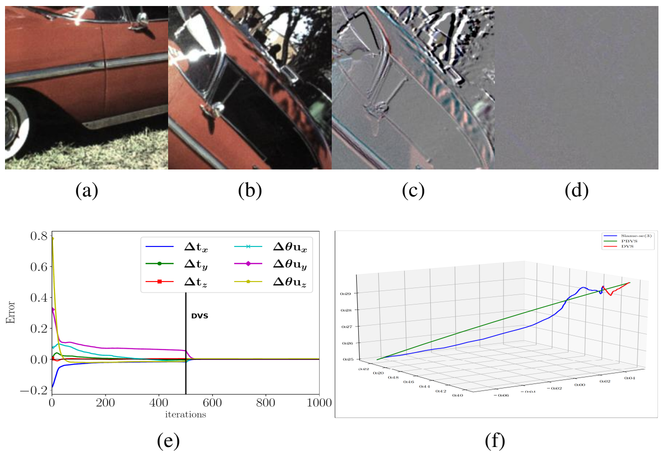

Fig. 5: (a) Starting image $\mathbf{I}$. (b) Desired image $\mathbf{I}^*$. (c) Image difference before using DVS. (d) Final image difference. (e) Pose difference $\mathbf{\Delta r}$. (f) 3d trajectories. Green: ground truth trajectory. Blue: Siame-se(3). Red: DVS.

## VI. Conclusion

We have proposed a new siamese deep architecture for end-to-end visual servoing. Our network directly regresses, in real time, the velocity of a camera in se(3). Our method features an accurate positioning, a broad convergence cone and combines well with photometric VS. While we have focused on learning a PBVS control law, this method can be applied to other control laws with minimal effort. Using multi-task learning to regress the different components of the velocity reduces the time spent training and tweaking networks, helping deploy VS solutions faster. Our learning protocol effectively trains our network on simulated data and can be applied to real scenes. Simulated and real experiments demonstrated the efficiency of the proposed approach.

## References

[1] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke. Training deep neural networks for visual servoing. In *IEEE Int. Conf. on Robotics and Automation, ICRA'18*, pages 3307–3314, Brisbane, Australia, May 2018.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[3] L. Bertinetto, J. Valmadre, J. Henriques, A. Vedaldi, and P. Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865, 2016.

[4] F. Chaumette and S. Hutchinson. Visual servo control, Part I: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, December 2006.

[5] F. Chaumette and S. Hutchinson. Visual servo control, Part II: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, March 2007.

[6] C. Collewet and E. Marchand. Photometric visual servoing. *IEEE Trans. on Robotics*, 27(4):828–834, August 2011.

[7] C. Collewet, E. Marchand, and F. Chaumette. Visual servoing set free from image processing. In *IEEE Int. Conf. on Robotics and Automation, ICRA'08*, pages 81–86, Pasadena, CA, May 2008.

[8] N. Crombez, E.M. Mouaddib, and G. Caron. Photometric Gaussian mixtures based visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'15*, pages 5486–5491, Hamburg, Germany, September 2015.

[9] A. Dame and E. Marchand. Entropy-based visual servoing. In *IEEE Int. Conf. on Robotics and Automation, ICRA'09*, pages 707–713, Kobe, Japan, May 2009.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] A. Dosovitskiy, P. Fischer, 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 *IEEE Int. Conf. on Computer Vision*, pages 2758–2766, 2015.

[12] S. En, A. Lechervy, and F. Jurie. RPNet: an end-to-end network for relative camera pose estimation. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[13] C. Finn, X. Tan Yu, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *IEEE Int. Conf. on Robotics and Automation, ICRA'16*, pages 512–519, 2016.

[14] D. Gordon, A. Farhadi, and D. Fox. Re$^3$: Real-time recurrent regression networks for visual tracking of generic objects. *IEEE Robotics and Automation Letters*, 3(2):788–795, April 2018.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[16] E. Hoffer and N Ailon. Deep metric learning using triplet network. In Aasa Feragen, Marcello Pelillo, and Marco Loog, editors, *Similarity-Based Pattern Recognition*, pages 84–92. Springer International Publishing, 2015.

[17] V. Kallem, M. Dewan, J.P. Swensen, G.D. Hager, and N.J. Cowan. Kernel-based visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and System, IROS'07*, pages 1975–1980, San Diego, USA, October 2007.

[18] A. Kendall and R. Cipolla. Geometric loss functions for camera pose regression with deep learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6555–6564, 2017.

[19] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

[20] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. *IEEE International Conference on Computer Vision, ICCV*, pages 2938–2946, 2015.

[21] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *2015 Int. Conf. on Learning Representations (ICLR)*, 2015.

[22] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, July 2015.

[23] A. Lee, S. Levine, and P. Abbeel. Learning visual servoing with deep features and fitted q-iteration. In *Int. Conf. on Learning Representations, ICLR*, March 2017.

[24] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, January 2016.

[25] E. Marchand. Direct visual servoing in the frequency domain. *IEEE Robotics and Automation Letters*, 5(2):620–627, 2020.

[26] E. Marchand, F. Spindler, and F. Chaumette. ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12(4):40–52, December 2005. Special Issue on "Software Packages for Vision-Based Control of Motion", P. Oh, D. Burschka (Eds.).

[27] E. Marchand, H. Uchiyama, and F. Spindler. Pose estimation for augmented reality: a hands-on survey. *IEEE Trans. on Visualization and Computer Graphics*, 22(12):2633–2651, December 2016.

[28] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, 2010.

[29] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10), 2010.

[30] A. Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[31] X.B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *IEEE Int. Conf. on Robotics and Automation, ICRA'18*, 2018.

[32] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *Int. Conf. on Computer Vision*, pages 2564–2571, 2011.

[33] A. Saxena, H. Pandya, G. Kumar, A. Gaud, and K.M. Krishna. Exploring convolutional networks for end-to-end visual servoing. In *IEEE Int. Conf. on Robotics and Automation, ICRA'17*, pages 3817–3823, May 2017.

[34] M. Tan and Q.V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Int. Conf. on Machine Learning*, 2019.

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'17*, pages 23–30, 2017.

[36] W. Wilson, C. Hulls, and G. Bell. Relative end-effector control using cartesian position-based visual servoing. *IEEE Trans. on Robotics and Automation*, 12(5):684–696, October 1996.

[37] C. Yu, Z. Cai, H. Pham, and Q. Pham. Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 935–941, 11 2019.