Visual Servoing in Autoencoder Latent Space

Samuel Felton, Pascal Brault, Elisa Fromont, Eric Marchand

***Abstract*—Visual servoing (VS) is a common way in robotics to control a robot motion using information acquired by a camera. This approach requires to extract visual information from the image to design the control law. The resulting servo loop is built in order to minimize an error expressed in the image space. We consider a direct visual servoing (DVS) from whole images. We propose a new framework to perform VS in the latent space learned by a convolutional autoencoder. We show that this latent space avoids explicit feature extraction and tracking issues and provides a good representation, smoothing the cost function of the VS process. Besides, our experiments show that this unsupervised learning approach allows us to obtain, without labelling cost, an accurate end-positioning, often on par with the best DVS methods in terms of accuracy but with a larger convergence area.**

***Index Terms*—Visual Servoing, Machine Learning for Robot Control**

I. Introduction

VISUAL servoing (VS) uses the information provided by a vision sensor to control the movements of a dynamic system [1]. VS can be used for tasks such as end-effector positioning, object picking or target tracking. It is especially useful in the cases where the current and target poses (positions and orientations) of the robot are not directly obtainable. This is the case when the scene may change in time and is not perfectly known. VS has applications in industrial settings for assembly operations, where the considered objects may not always be perfectly placed, making visual sensing a requirement for accurate operations. VS is framed as an optimisation process, where one seeks to minimise the difference between two sets of features that correspond to what is seen by the camera at the current pose and what is seen at the goal pose. The features that are used are often handcrafted. They can be 2D features (points, lines, ...) or 3D features (camera pose, 3D points). These features must be tracked over time and a matching process between features in the current and desired images is required. When the scene is not well structured (no tags, no well defined features), features tracking and matching is a difficult problem, which is often the cause of the failure of VS applications.

Photometric VS [2] or DVS (Direct Visual Servoing) has been formulated to work directly on the photometric information. By considering the whole image, these approaches do not require any tracking or matching process. They display very high positioning accuracy, but suffer from a small convergence area due to the fact that the optimisation problem is highly non-linear. To alleviate this issue, more compact image representations such as photometric moments [3], projections on an orthogonal basis (PCA [4] or DCT [5]) or Gaussian mixtures [6]) can be considered.

In the last decade, the effectiveness of learning-based methods on unstructured data has soared, in big part thanks to Deep Learning (DL). Neural networks, and especially Convolutional Neural Networks (CNNs) have achieved state of the art results in a variety of problems, encompassing classification [7], [8], optical flow regression [9] or pose estimation [10] among other tasks. This success comes in big part from the fact that deep networks learn a hierarchical representation of the input, each layer extracting the features relevant to the downstream layers and the final task to be solved. These models alleviate the need for hand-crafted features, as they are learned directly from the data. DL is often used in a supervised setting, where the labels are provided during training and the learned representation is optimised directly for the task. In the unsupervised setting, one can for example train a network to learn to reconstruct the original input with some constraints. This network is called an autoencoder (AE). AEs aim to learn a "good" representation of the data. This representation is often low dimensional, following the manifold hypothesis stating that the input data is generated from a much lower number of variables than its own dimensionality. AEs have been shown to be able to compress the information into a very small code, with better reconstruction than their linear counterparts, e.g. PCA [11].

Deep learning has previously been applied to VS. In some of these approaches [12], [13], [14], a deep neural network is used to estimate either the camera pose or the pose difference between two cameras, from which the control law can be derived. In [15], the camera velocity is directly regressed. This requires data labelling in the form of an image-pose correspondence and simulation is often used to train before transferring to the real world. These methods show a good convergence area, but the end positioning is not always accurate. A classical DVS method can be applied at the end of the process, to correct the remaining positioning error.

In this paper, we propose a novel visual servoing control law, based on photometric information in combination with a deep network. The network, an autoencoder, is trained in an *unsupervised* fashion, learning to compress the given input. Based on a convolutional architecture, it can extract richer features than other previous dimensionality reduction approaches. We establish the link between the learned representation and the camera motion, allowing for the control of robot in the latent space. We show through experiments, both in simulation and on a real robot, that servoing in the latent space is effective, with a very accurate positioning and a better convergence domain than other dimensionality reduction-based methods. This approach contrasts with other

Manuscript received: September, 7th, 2021; Revised November, 25th, 2021; Accepted December, 30th, 2021. This paper was recommended for publication by Editor Tamin Asfour upon evaluation of the Associate Editor and Reviewers' comments.

Authors are with Univ Rennes 1, Inria, CNRS IRISA, Rennes, France. Elisa Fromont is also with Institut Universitaire de France. Email: {samuel.felton, pascal.brault, elisa.fromont, eric.marchand}@irisa.fr.

Digital Object Identifier (DOI): see top of this page.

works combining DL and VS in that it does not require supervision. Moreover, it is less sensitive to the gap between simulation and real world, which is problematic in other DL works [15]. It also differs from reinforcement learning litterature [16], [17], [18], [19], where the control law (policy) is learned from the data and from a reward signal. This contrasts with our approach, in which we derive an analytical control law on a new, learned feature space. Moreover, it is complementary to visual planning approaches such as [20], [6] that use latent representations to generate subgoals in the form of images. In this case, the control/planning is still done w.r.t. images, while our method has no generation step, as we remain in the latent space.

The paper is structured as follows: we first give an overview of VS and how features extracted from images can be used to move in the cartesian space. We then detail how servoing can be done in the low dimension latent space of an autoencoder with our method, going from training a network to deriving the control law associated to a neural network. Finally, we validate the servoing scheme in experiments, both in simulation and deployed on a real robot.

II. Visual servoing

The goal of visual servoing is to control the motion of a robot with information extracted from images acquired by a camera, either looking at or mounted on the robot end-effector [1]. It is designed as a minimisation problem between the image features $\mathbf{s}(\mathbf{r})$, extracted at the current pose $\mathbf{r}$, and $\mathbf{s}^*$, the desired features at the desired pose $\mathbf{r}^*$. The error $\mathbf{e}$ to be minimized is then:

$$\mathbf{e} = \mathbf{s}(\mathbf{r}) - \mathbf{s}^*. \tag{1}$$

If the feature are correctly chosen, when the error is minimized in the image space, the robot has reached the desired position $\mathbf{r}^*$ in the 3D space.

To design the control law, the relationship between the motion of the features $\mathbf{s}$ in the image and the camera velocity $\mathbf{v}$ must be known. This is done thanks to the *interaction matrix* $\mathbf{L_s}$ (also known as image jacobian), which contains the partial derivatives of the features w.r.t. to the pose $\frac{\partial \mathbf{s}}{\partial \mathbf{r}}$. The link between the time variation (motion) $\dot{\mathbf{s}}$ of $\mathbf{s}$ and the camera velocity $\mathbf{v}$ is given by:

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbf{v}. \tag{2}$$

When the camera is mounted on the end-effector of the robot, the control law is directly deduced from (Eq. 1) and (Eq. 2). The camera velocity $\mathbf{v}$ which allows the error $\mathbf{e}$ to be minimised is obtained with:

$$\mathbf{v} = -\lambda \mathbf{L_s^+}\mathbf{e} \tag{3}$$

where λ is a gain parameter, which ensures an exponential decrease of the error, and $\mathbf{L_s^+}$ is the pseudo inverse of $\mathbf{L_s}$. VS is realised in a closed loop scheme, with $\mathbf{v}$ being computed for each new image acquired by the camera.

One interaction matrix of interest to us is that of a 2D point $\mathbf{x} = (x, y)$. It is given as

$$\mathbf{L_x} = \begin{pmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{pmatrix} \tag{4}$$

where Z is the depth of the point. Many types of features can be considered for VS, as long as their interaction matrix is available [1].

Recently, some approaches have tried to move away from handcrafted features by considering the pixel intensities [2]. It is known as Direct (or photometric) VS. In this setting, the feature is the image itself $\mathbf{s}(\mathbf{r}) = \text{vec}(\mathbf{I}(\mathbf{r}))$. The interaction matrix of the intensity $\mathbf{I_x}$ of a given pixel at the point $\mathbf{x}$ is [21]:

$$\mathbf{L_{I_x}} = -\nabla \mathbf{I_x}^\top \mathbf{L_x} \tag{5}$$

where $\nabla \mathbf{I_x}$ is the spatial gradient at $\mathbf{x}$. To consider an image $\mathbf{I}$ of dimensions $N \times N$ as the servoing features, the interaction matrix at every point of the image is computed and then are stacked to form the $N^2 \times 6$ Jacobian $\mathbf{L_I}$.

This leads to a very precise positioning, but suffers from a very small convergence domain and an unpredictable trajectory. To overcome the former problem, multiple approaches have been developed that aim at reducing the nonlinearity of the cost function by reducing the dimensionality of the input image while keeping the majority of the information. A possible solution is to project the image on an orthogonal basis to obtain a feature set of lower dimension. This basis can be learnt (eg, using Principal Component Analysis (PCA) [4]) or predetermined (eg, using a DCT [5]). For example, as far as a PCA is considered, a set of images is used to compute the basis (training). At run-time, the images are projected onto this pre-computed basis. The projection is a linear transformation, expressed as a matrix $\mathbf{U}$ of size $N^2 \times D$, with D a chosen low dimension. With $\mathbf{U}$, one can project an image onto the eigenspace as $\mathbf{w} = \mathbf{U}^T(\mathbf{I} - \bar{\mathbf{I}})$. Then, to perform servoing, the associated interaction matrix can be defined as:

$$\mathbf{L_w} = \mathbf{U}^\top \mathbf{L_I}. \tag{6}$$

In a similar way, the Discrete Cosine Transform (DCT) [5] can be considered to define the basis. The DCT allows for the image to be represented in the frequency domain by a sum of cosines of different frequencies and amplitudes. In both cases, PCA or DCT, reducing the dimensionality of the features allows to focus on the low frequencies of the images and discard the high frequencies which are shown to be unhelpful for the task.

[4] is our main source of inspiration. We also use learning as a way to compress the image, but we do so with a non-linear function, unlike PCA which is linear and projects on an orthogonal basis. Here, we use a more powerful learning algorithm: an autoencoder, capable of learning a non-linear representation of the data, leading to a better compression of the information.

III. AEVS: SERVOING ON LEARNED FEATURES

In this section we detail the inner workings of AEVS, an *Auto-Encoder Visual Servoing* method, and describe how to perform visual servoing on the features extracted by a neural network. We start by giving some background on autoencoders (AE). AE typically achieve better compression and reconstruction than other linear learning-based methods. They also have the ability to better approximate the non-linear manifold from which the data was generated. For these reasons, we argue that they are an ideal fit for reducing the dimensionality of Photometric VS, improving the convergence rate while retaining an accurate end positioning. AEVS, detailed afterwards, is based on the analytical computation of the interaction matrix of a neural network.

A. Background on autoencoders

Autoencoders (AE) [22] are neural networks composed of two distinct parts: an encoder h, which aims at projecting the learning data into a latent space (or bottleneck) with interesting properties and a decoding part d which decodes samples from the latent space into the original data space. Similarly to other neural networks, AE are trained by gradient descent and the parameters are updated in order to minimise a loss function, which is specific to the task at hand. Since gradient descent requires computing the gradients w.r.t. to the parameters and inputs of each layers, all the operations are differentiable. As for other neural networks, the most common layers of the AE apply a linear processing (convolutional or not), followed by a non-linear activation function. These non-linearities give its discriminatory and learning capabilities to neural networks, allowing them to form a non-linear mapping between the input and the output.

However, AE are particularly appealing since they do not require labelled data to be trained. This comes from the fact that they seek to learn the identity function $f(\mathbf{x}) = d(h(\mathbf{x})) = \mathbf{x}$. While this might be trivial to learn, constraints are added in order for the AE to learn a meaningful representation (latent space). The simplest one is to impose an undercomplete representation that has a low dimensionality [11] and cannot conserve the full information of the data. AE compare favourably to their linear counterparts, e.g. PCA, in that they can achieve a lower error on the reconstruction task and encode into fewer variables the main factors of variations of the data, making their representation ideal for clustering and other downstream tasks. The reconstructed input is then $\tilde{\mathbf{x}} = d(h(\mathbf{x}))$. The latent, compressed information stemming from applying h is denoted as $\mathbf{z} = h(\mathbf{x})$. To minimise the reconstruction error, a loss function must be introduced. Two common choices are the mean squared error

$$\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 \tag{7}$$

in the case of real unbounded values or the binary cross-entropy (BCE)

$$\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = -\frac{1}{N} \sum_{i}^{N} \mathbf{x}_i \log(\tilde{\mathbf{x}}_i) + (1 - \mathbf{x}_i) \log(1 - \tilde{\mathbf{x}}_i) \tag{8}$$

when the inputs are binary or in the range $[0, 1]$. Both losses seek to maximise $p(\mathbf{x}|\mathbf{z})$, the probability of $\mathbf{x}$ being recovered from $\mathbf{z}$.

AEs can also learn a representation that is robust to noise and small perturbations. This can be done by an explicit regularisation term as done in [23], or by corrupting the input [24] while encouraging the reconstruction to be close to the uncorrupted input. These approaches enforce a small degree of invariance to small changes in the data, making the representation more robust.

AE have also previously been used for control. In reinforcement learning, they are useful to reduce the dimensionality of the input data, making easier to learn a policy afterwards [17], [19]. Given an appropriate architecture, they can extract spatial features such as points that can be used for control [25], [26].

In the following, we will consider an AE that is trained to reconstruct an input image $\mathbf{I}$. This AE, based on a convolutional architecture, must learn the parameters during training, so that the error between the reconstructed image $\tilde{\mathbf{I}}$ and $\mathbf{I}$ is minimised. Using a convolutional encoder allows us to learn a powerful and high level representation, which can be used for control purposes. Inspired by [5], we propose to guide our AE to learn a representation that is robust to changes in high frequencies and small photometric variations.

B. Learning a representation for visual servoing

In previous works approaching DVS through the prism of dimensionality reduction [4], [5], it has been shown that considering the main factors of variation in the data or their low frequency components improves the servoing convergence. In these approaches, this knowledge is explicit, as the explained variance/frequency of each of the extracted features is known. However, in the case of an AE, the latent space must be guided towards the goal of prioritising lower frequencies. Indeed, the traditional losses presented in Eq. (7) and Eq. (8) consider all pixels equally and independently and as such cannot incorporate such constraints. With these objective functions, the AE will try to fit both low and high frequencies and may learn a latent representation that is in part tied to the higher frequencies.

To remedy this, we propose to reformulate our reconstruction loss in the frequential domain. Applying the DCT to the original image $\mathbf{I}$ and its AE reconstruction $\tilde{\mathbf{I}}$, we obtain their frequential representations $\mathbf{F}$ and $\tilde{\mathbf{F}}$. $\mathbf{F}$ is an $N \times N$ matrix, in which the energy of the different frequencies of the signal $\mathbf{x}$ is stored. The frequencies are ordered and go from low ($\mathbf{F}_{0,0}$) to high ($\mathbf{F}_{N-1,N-1}$). Since there is an explicit ordering, we can choose which frequencies to prioritise in the loss. This is done by adding a weighting in the loss like so:

$$\mathcal{L}_{DCT}(\mathbf{F}, \tilde{\mathbf{F}}) = \frac{1}{N^2} \sum_{i=0}^{N} \sum_{j=0}^{N} \mathbf{M}_{i,j} |\mathbf{F}_{i,j} - \tilde{\mathbf{F}}_{i,j}| \tag{9}$$

The elements of the weighting mask $\mathbf{M}$ are defined as $\mathbf{M}_{i,j} = \frac{1}{i+j+1}$, placing more emphasis on having a correct reconstruction of the lower frequencies and mostly discarding the higher ones. Although the zigzag ordering [5] is usually

used to have a vectorized representation of the frequencies, we found that considering the elements of a diagonal of $\mathbf{F} - \tilde{\mathbf{F}}$ works equally well. We evaluate the impact of the loss function of the latent space and the performance improvements in Section IV.

C. Computing the interaction matrix of a neural network

In our method, we propose to perform VS in the latent space and reformulate the error (Eq. (1)) as a function of the encoding of an AE:

$$\mathbf{e} = \mathbf{z}(\mathbf{r}) - \mathbf{z}^* \tag{10}$$

For the robot to be controllable (by applying Eq. (3)), the variation of $\mathbf{z}$ w.r.t. to the camera velocity must be known, i.e the interaction matrix $\mathbf{L_z}$ must be computed. We choose as our servoing features the information bottleneck $\mathbf{z}$. Because the network will process an input image $\mathbf{I}$, its learned representation is dependent on $\mathbf{I}$ and $\mathbf{L_I}$ (see Eq. (5)) is required to compute $\mathbf{L_z}$. Computing $\mathbf{L_z}$ from $\mathbf{L_I}$ is a matter of chain rule derivations of the Jacobians:

$$\mathbf{L_z} = \frac{\partial \mathbf{z}}{\partial \mathbf{r}} = \frac{\partial \mathbf{z}}{\partial \mathbf{I}} \frac{\partial \mathbf{I}}{\partial \mathbf{r}} = \frac{\partial \mathbf{z}}{\partial \mathbf{I}} \mathbf{L_I} \tag{11}$$

where $\mathbf{L_I}$ is derived from equation (5). In the case of a simple one-layer linear autoencoder where $\mathbf{z} = \mathbf{WI} + \mathbf{b}$, with $\mathbf{W}, \mathbf{b}$ the learned encoder parameters, The interaction matrix is equivalent to the one presented in [4] and detailed in Eq. (6):

$$\mathbf{L_z} = \frac{\partial \mathbf{WI} + \mathbf{b}}{\partial \mathbf{r}} = \mathbf{W} \frac{\partial \mathbf{I}}{\partial \mathbf{r}} + \frac{\partial \mathbf{W}}{\partial \mathbf{r}} \mathbf{I} = \mathbf{W} \mathbf{L_I} \tag{12}$$

Linear autoencoders and PCA are tightly linked [27], [28]: they optimise the same objective and project into the same subspace. The projection $\mathbf{W}$ learned with a linear AE does not have the orthogonality constraint. However, the principal directions can be recovered from $\mathbf{W}$ by applying a singular value decomposition.

To extend the process to the case of a nonlinear layer, one must take into account the activation function that is commonly applied after the linear transformation of the layer. A traditional fully connected layer is of the form $\mathbf{z} = a(\mathbf{WI} + \mathbf{b})$, with a the activation function, typically applied element-wise. To find the interaction matrix associated to this nonlinear operation, we must derive w.r.t. to the pose $\mathbf{r}$:

$$\mathbf{L_z} = \frac{\partial \mathbf{z}}{\partial \mathbf{r}} = \frac{\partial a(\mathbf{WI} + \mathbf{b})}{\partial \mathbf{r}} \tag{13}$$

Applying the chain rule, we get:

$$\mathbf{L_z} = \frac{\partial a(\mathbf{WI} + \mathbf{b})}{\partial \mathbf{WI} + \mathbf{b}} \frac{\partial \mathbf{WI} + \mathbf{b}}{\partial \mathbf{r}} \tag{14}$$

And using the same process than for the PCA-based interaction matrix computation, which relies on the fact that $\mathbf{W}$ and $\mathbf{b}$ are independent of the pose $\mathbf{r}$ (they are learned offline), the final result is:

$$\mathbf{L_z} = \frac{\partial a(\mathbf{WI} + \mathbf{b})}{\partial \mathbf{WI} + \mathbf{b}} \mathbf{W} \mathbf{L_I} \tag{15}$$

The projection operation $\mathbf{WL_I}$ performs a linear combination of the gradient of the pixels with respect to pose and does not influence their direction. The derivative of the activation $\frac{\partial a(\mathbf{WI}+\mathbf{b})}{\partial \mathbf{WI}+\mathbf{b}}$ impacts the modelling of the interaction matrix. In the case of the ReLU activation, defined as $a(x) = max(x, 0)$, $\frac{\partial a(x)}{\partial x} = \mathbb{1}_{x>0}$. Thus, when computing $\mathbf{L_z}$, this activation will act as a binary selection, keeping only the information of the filters that were activated.

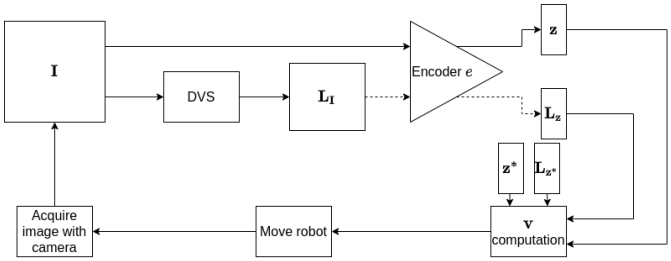


Fig. 1: Overview of our VS process. Given an image $\mathbf{I}$ and its associated interaction matrix $\mathbf{L_I}$, we project the information onto a low dimensional embedding $\mathbf{z}$ and compute its interaction matrix $\mathbf{L_z}$, with the dashed lines representing forward differentiation.

Moreover, a deep network is a stack of layers. To compute the interaction matrix of the final representation, we apply the chain rule recursively, from the input to the output. Considering a stack of n layer outputs $\mathbf{l}^0, ..., \mathbf{l}^n$ and the associated weights $\mathbf{W}^1, ..., \mathbf{W}^n$ and biases $\mathbf{b}^1 ..., \mathbf{b}^n$:

$$\begin{aligned} \mathbf{L_{l^0}} &= \mathbf{L_I} \\ \mathbf{L_{l^i}} &= \frac{\partial \mathbf{a}^i(\mathbf{W}^i \mathbf{l}^{i-1} + \mathbf{b}^i)}{\partial \mathbf{W}^i \mathbf{l}^{i-1} + \mathbf{b}^i} \mathbf{W}^i \mathbf{L_{l^{i-1}}} \end{aligned} \tag{16}$$

This process is a direct application of the forward automatic differentiation [29]. Computing the gradient in this manner is not recommended when the dimension of the output is less than that of the input, as it is less efficient than reverse differentiation. However, we are deriving a vector $\mathbf{z}$ in $\mathbb{R}^D$ w.r.t the pose $\mathbf{r}$, where $D \geq 6$. D cannot be inferior to 6, or it will incur the loss of a degree of freedom of the camera when servoing. Thus, using forward differentiation makes sense in our case and allows us to compute $\mathbf{L_z}$ and $\mathbf{z}$ in parallel. The integration of the network into the servoing loop is detailed in Figure 1.

Note that the transformation of the interaction matrix is linear, as seen in Eq. (11). However, unlike PCA, it is sample-dependent because of the nonlinearities.

While we have developed the case for a multilayer perceptron, it is straightforward to extend it to other types of networks, such as CNNs. One must simply compute the Jacobian of the involved transformations, such as the convolution operation $*$. Considering a convolution in an intermediate layer i, $\mathbf{C}^i = \mathbf{C}^{i-1} * \mathbf{W}$, where $\mathbf{C}^{i-1}$ is a feature map tensor of dimensions $C_{i-1} \times H \times W$ with C_{i-1} the number of features, $\mathbf{W}$ a set of learned filters, given the interaction matrix associated to $\mathbf{C}^{i-1}$, $\mathbf{L_{C^{i-1}}}$, as a tensor of dimension $6 \times C_{i-1} \times H \times W$ then, computing the interaction matrix $\mathbf{L_{C^i}}$ amounts to convolving each tensor associated to a pose component with the learned filter:

$$j : 0 \leq j < 6, \mathbf{L_{C^i,j}} = \mathbf{L_{C^{i-1},j}} * \mathbf{W} \tag{17}$$

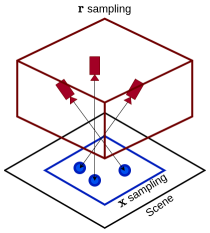


Fig. 2: Simulated data generation process. Focus points, $\mathbf{x}$ are sampled from the blue zone. A pose $\mathbf{r}$ looking at the point $\mathbf{x}$ is then generated in the red zone, above the scene.

IV. Experiments

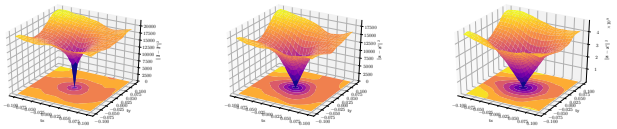
In this section, we evaluate the performance of our approach for VS. We use a well known convolutional architecture and train in simulation. In the same environment, we compare our results with other photometric-based VS approaches and explore the impact of different hyperparameters on the final results. Finally, experiments on a real robot are conducted to demonstrate the capabilities of AEVS in a real world setting, both on a scene used for training and on a novel one.

A. Implementation details

To deploy our method, we train an autoencoder tasked with minimising the reconstruction error. We use ResNet-18 [8] for both our encoding and decoding networks. We found that this relatively small and robust architecture (using residual connections) is sufficient for our task, although more recent ones such as EfficientNets [30] could have also been used. In the decoding part, we "flip" the standard ResNet, replacing the downsampling operations with upsampling ones. The weights between the two networks are not shared. We observed that Batch Normalization (BN) [31] had a detrimental impact on servoing. We therefore replace it with another normalization scheme, Weight Normalization (WN) [32]. WN encourages the norm of the parameters to grow, which is not the case for BN. When computing $\mathbf{L_z}$, the values coming from $\mathbf{L_I}$ will typically be orders of magnitude smaller than the intensities in $\mathbf{I}$: using BN may cause precision issues as we iterate through the layers. Moreover, we replace the classically used global average pooling at the end of the network with a grouped convolution, since the invariance induced by the pooling is not helpful for our task, and leads to poorer control at test time. After the last convolution, we project the features onto an embedding of the desired size with a linear fully connected layer.

Finally, the last requirement for training is a dataset : we use a simple simulation, similar to [12], [15], where the considered scene is planar. Training then requires a single image of the scene. To get a varied sampling of the scene, we randomly draw a point $\mathbf{x}$ near the center of the scene (so as to avoid seeing the edges, which are not of interest). We then sample a camera pose $\mathbf{r}$ above the scene, directly looking at $\mathbf{x}$ and with a random rotation around the focal axis (see Figure 2).

The network is trained for 50 epochs, on a dataset of 20k images for training and validated on 10K images, with a batch size of 50. We use the Adam optimiser, with a learning rate of 10^{-3}. Training on a RTX 2080Ti takes around three hours with deterministic algorithms (used in all our experiments), and an hour without, making deployment fast. At inference, computing $\mathbf{z}$ and $\mathbf{L_z}$ takes about 7ms on the GPU.



(a) DVS (b) DCT-VS (c) AEVS

Fig. 3: The servoing loss landscape for (a) DVS (b) DCT-VS (c) AEVS, on an x/y translation motion around the desired pose.

When performing servoing, we choose a Levenberg-Marquardt-based control law and replace the traditional velocity computation (Eq. (3)) with

$$\mathbf{v} = -\lambda(\mathbf{H} + \mu diag(\mathbf{H}))^{-1}\mathbf{L_z}^{\top}(\mathbf{z} - \mathbf{z}^*) \tag{18}$$

with $\mathbf{H} = \mathbf{L_z}^T\mathbf{L_z}$ and $\mu = 0.01$. This control law has been shown to vastly improve the results when considering photometric information for VS [2], [4], [5].

B. Experiments on simulated data

The first validation of our method is shown in Fig. 3. We study the loss landscape of the error function $\mathbf{e} = \mathbf{z} - \mathbf{z}^*$. To visualise this, we compute the euclidean norm of the error at different offsets from the desired pose. Our nonlinear representation effectively smoothes the servoing cost function, leading to a better convergence area at test time.

To study our method and compare it with other approaches, we develop two test sets, corresponding to different scenarios. In the first one, we generate poses $\mathbf{r}, \mathbf{r}^*$, so that they look around the same point of the scene. We add noise (up to 10cm displacements) to the points observed by $\mathbf{r}$ to add difficulty to the test cases. This scenario will create large displacements (with some far from the training set) in translation (on all axes) and in rotation on the x/y axes to compensate for the translation. The translation is sampled uniformly in a 3D box above the scene of dimensions $[1.2m, 1.2m, 0.3m]$. The rotation around the z axis is kept small (few degrees). The average starting error is of 47cm$\pm$15cm and 37°$\pm$11°. In the second scenario, we study the convergence on screwing motions, where the displacement is on the z axis. For $\mathbf{r}$, we create displacements with translations in $[-30cm, 30cm]$ and rotations in $[-70°, 70°]$. The desired poses $\mathbf{r}^*$ are located 60cm above the scene, with a random rotation around the z axis. For the two scenarios, we generate 500 test cases $\mathbf{r}, \mathbf{r}^*$. We chose those test cases because they feature large overlap between the images, and are the cases where using direct VS schemes makes the most sense. When the overlap is small, our method has a low convergence rate, as do the other direct VS methods. We first look at the performance of the loss function proposed in Section III-B and compare it with a standard reconstruction loss, the pixel-wise binary cross entropy (BCE, Eq. (8)). The results, displayed in Figure 4, show that our proposed loss greatly improves the results on the screw motion test case, allowing a convergence rate of

up to 91% and overall improves the results when compared to PCA (detailed below). When using BCE, the convergence rate is 82.52% $\pm$ 0.4%, while it is 88.84% $\pm$ 1.5% with the frequential loss. In the look at test case, the results are similar (92.76% $\pm$ 1.75% for BCE, 92.56% $\pm$ 0.8% for the frequential loss), with a slight advantage for the BCE loss in terms of median and maximum convergence but with a higher variance overall.

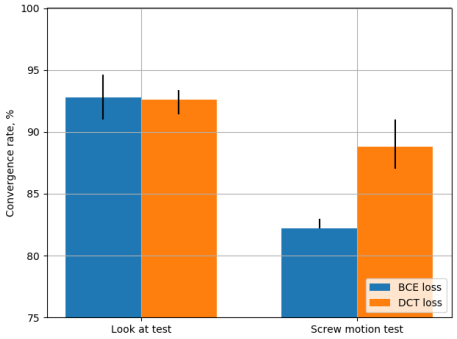


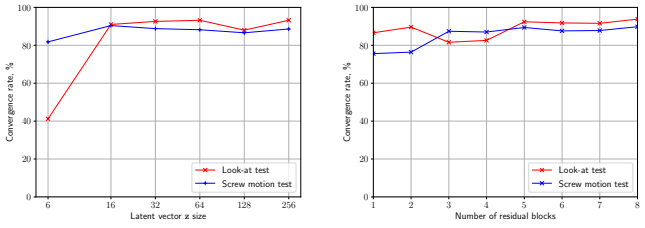
Fig. 4: Convergence rate of AEVS for two loss functions on the two test cases. We report median, minimum and maximum convergence rate across 5 different seeds.

Next, we evaluate the results of our method in comparison to other direct VS approaches, namely DVS [2], DCT-based VS [5] and PCA-based VS [4], as well as an end-to-end deep learning approach, Siame-se(3) [15]. The latter demonstrates much higher convergence areas (beyond the ones tested here) but a much lower precision. The results are reported in Table I. For DCT-VS, PCA-VS and the two AEVS variants, we project the input to a latent space of size 32. For AEVS, we select our best-performing networks from the experiment of Figure 4 (in terms of convergence rate), trained with the two losses (BCE and the frequential domain loss presented in Section III-B) and show their results. We study the convergence rate, as well as the end error for the cases that converge to the desired pose. In all cases, the impact of dimensionality reduction is evident (second line against all four of the last lines in the table) and leads to a better convergence. In the first test case ("look at"), our method achieves better convergence than other methods, while maintaining a precise positioning. The best results are achieved when trained with the common pixel-wise BCE loss. This may indicate that in this case, learning a representation that conserves more high frequency information is helpful. For DVS, convergence is far slower and servoing must be run for more iterations than other methods, due to the shape of the cost function. Indeed, when there are compensations between translations and rotations on the x/y axes, servoing drives the process towards flat valleys of the cost function, leading to a slow convergence. Siame-se(3) manages to converge close to the desired pose in every case, but exhibits lower accuracy. For the screw motion test, our method sensibly improves convergence when compared to other methods if it is trained with the frequential loss presented in Section III-B. When using a standard pixel-wise loss, the performance falls behind other methods. Siame-se(3) produces a high error, but even in cases that did not converge, it reduces the pose error so that it reaches around 2.5 cms/°in average. For AEVS, All samples that reach a near zero velocity (they settle in a minimum of the cost function) find the global minimum, i.e the desired pose. This illustrates that the learned latent space is discriminative and the cost function smooth, leading to a correct optimization.

	Look-at test		Screw motion test	
	Convergence %	End error mm, °	Convergence %	End error mm, °
Siame-se(3) [15]	100	9.2, 0.67	87	14.9, 1.5
DVS [2]	59.0	0.0004, 0.0	79.4	0.0001, 0.0
DCT-VS [5]	82.4	0.04, 0.004	86.2	0.04, 0.003
PCA-VS [4]	90.0	0.03, 0.003	87.4	0.02, 0.002
AEVS, pixel loss (Eq. 8)	94.6	0.03, 0.003	83	0.1, 0.01
AEVS, frequential loss (Eq. 9)	93.2	0.02, 0.002	91	0.06, 0.006

TABLE I: Comparative results of different VS methods on two test cases. The convergence rate, as well as the end positioning error for the cases that converge are reported.

We then study the impact of the network hyper-parameters on the results for AEVS, trained with the frequential loss (Eq. (9)). We first evaluate the impact of the latent size dimension (Figure 5a). It can be seen that VS works slightly better with lower dimensional representations, although using larger ones is also effective. When using a bottleneck of size 6, which is the true dimensionality of the manifold (images only depend on the pose), servoing works especially poorly in the look-at test. This may indicate that the retained information is not enough to disentangle the ambiguities between the x/y translations and y/x rotations that are present in those cases. Finally we evaluate the impact of the encoder depth on servoing (Figure 5b). To do this, we train a series of 8 ResNets. For Resnet i, only the i first residual blocks of the encoder are active and the remaining blocks only retain their skip projections. We also keep the end projection so that we reach a latent space of the same dimension. A similar operation is applied to the decoder, where the i last blocks are kept as is. It can be seen that while VS works for a linear projection of the low level features, mid/high level non-linear features acquired in the later residual blocks are impactful in learning a good representation for servoing.



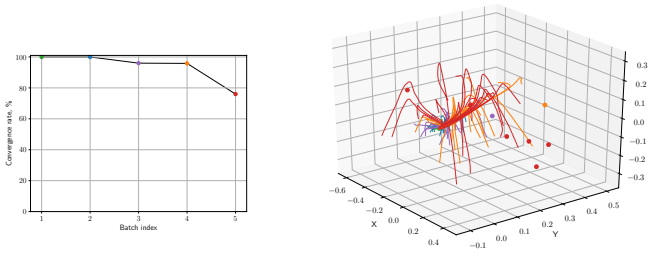
(a) (b)

Fig. 5: Servoing results, depending on (a) latent space size and (b) network depth.

More experiments can be found in our supplementary material [33], where we study the impact of additional learning parameters, as well as perform experiments on different objects.

C. Robot experiments

Finally, we deploy AEVS on an 6 DOF gantry robot, to test on real scenes. First, we study the convergence of our method. To do so, we generate samples with increasing difficulty, sampling from Gaussian distributions with 0 means and growing standard deviations. For the x/y translations, the standard deviations are $0.02/0.05/0.1/0.2/0.4$m and halve these for the z. The orientations of the cameras are such that the initial viewpoints have a large overlap with the desired one, but we also add noise $\in [-0.1m, -0.1m]$ to the focal point in the scene in order to increase the difficulty. We also add rotations around the focal axis, in the $[-30°, 30°]$ range. For each difficulty batch, we generate 25 samples. We ensure that the starting poses lie in the robot workspace. The results can be seen in Figure 6. Our method exhibits a strong convergence rate and, for batches 3 and 4, the only failing cases are due to large specular reflections appearing in the starting image. This is not unexpected, as specularities are particularly impactful on photometric methods, built on the hypothesis that the scene is lambertian. For the last batch, with standard deviation 0.4m on the x/y axes, 3 out of the 6 failure cases are also due to speculars, while in the other 3, AEVS either converges to a suboptimal pose far from the desired one or leads the robot out of its workspace. Figure 6b shows the trajectories accomplished by the robot during servoing. The samples tend to be generated in a half volume, with a positive y displacement, due to a joint limit of the robot that constrains the workspace. The trajectories show a distinct pattern: a large translation on the z axis is first produced. Then, it is corrected in the following iterations, along with the remaining displacement on the other axes. This, however, is not tied to our method but seems to be a common behaviour across photometric methods, as can be seen in our next experiment.



(a) (b)

Fig. 6: Results of the large scale robot experiment. (a) Convergence rate for batches with growing difficulties. (b) Servoing trajectories. Each color corresponds to a difficulty batch and points indicate diverging samples.

Our first detailed example (Figure 7) evaluates the method on the scene on which it was trained. We start with an initial displacement of $\Delta\mathbf{r}_0$ = (-32.78cm, 18.07cm, -6.16cm, 18.41°, 27.51°, 16.98°) and run our method for 1.5k iterations. Minimising the error in the latent space (Figure 7f) leads to a correct minimisation of the pose error and convergence is successful, with a final positioning error of $\mathbf{\Delta r}_{final}$ = (0.03cm, 0.05cm, 0.01cm, 0.05°, 0.03°, -0.14°). DVS, however, fails to converge. In a similar way as other photometric methods, AEVS first tends to correct the error on the z axis (although it does overestimate the translational motion in the first iterations), which accounts for the largest error in the image space. It finishes by correcting the displacement on the x/y axes. The trajectory (blue in Figure 7h) remains similar to the other methods.

In the second experiment, we study whether our method can handle novel scenes on which the AE is not explicitly trained. To do so, we train AEVS with the frequential loss on 100k images, from 10 different scenes. We train for 25 epochs, with a learning rate of 10^{-4}. We select our scenes from the "car" class of the ImageNet [34] dataset. They thus should have common semantic characteristics that we will try to exploit by servoing in the real world on a toy RC car. The starting displacement is (8.74cm, -23.67cm, -1.08cm, -18.55°, -17.92°, -32.62°), and the displacement in the image (Figure 8c) is fairly large. This scene, visible in Figure 8a and Figure 8b introduces 3D information that is not present in the first experiment. Moreover, it is quite challenging, as it contains specularities on the body and the windshield of the car, as well as high frequency patterns, such as the wheels, the checkered pattern of the decal and the added objects. Despite this, the method manages to reach a final displacement $\mathbf{\Delta r}_{final}$ of (0.31cm, -0.11cm, 0.07cm, -0.09°, -0.34°, 0.0°). There is a small remaining error, shown in Figure 8d, and is the minimum achievable error with AEVS. In this case, we still consider our method successful, considering the shift between the training set (planar images of modern cars) and the presented scene.

More experiments, including servoing on industrial objects, can be found in our supplementary material [33] (Section IV) to show the versatility of our method.

V. Conclusion

We have presented a novel way to perform visual servoing, controlling the camera motion in the latent space of an autoencoder. The method features a broad convergence area, as well an accurate positioning. The representation provided by the autoencoder is compact, and the camera motion required in order to minimise the latent error is computed analytically. The unsupervised method allows us to avoid using costly labelled data and our experiments also show that our method has a great generalization potential on unseen data. The ability to compute the interaction matrix for a generic neural network paves the way to other venues of research. The flexibility of neural networks, both in their architecture and their objective function could allow us to introduce new interesting constraints in the latent space to obtain smoother robot trajectories and improve the convergence of our method in complex cases.

References

[1] F. Chaumette and S. Hutchinson, "Visual servo control, Part I: Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[2] C. Collewet, E. Marchand, and F. Chaumette, "Visual servoing set free from image processing," in *IEEE ICRA'08*, Pasadena, May 2008, pp. 81–86.

[3] M. Bakthavatchalam, O. Tahri, and F. Chaumette, "A Direct Dense Visual Servoing Approach using Photometric Moments," *IEEE Trans. on Robotics*, vol. 34, no. 5, pp. 1226–1239, October 2018.

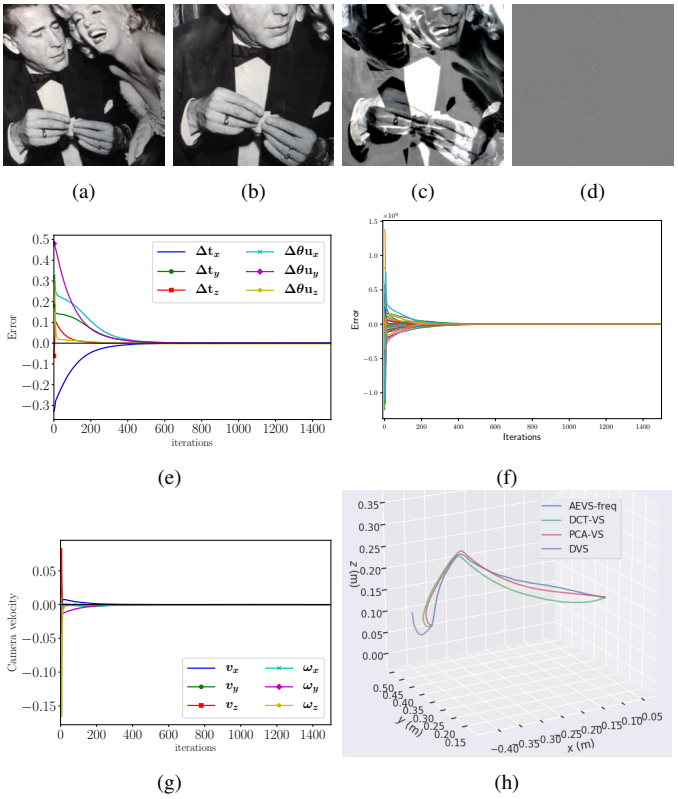


Fig. 7: First experiment: (a) Starting image $\mathbf{I}$. (b) Desired image $\mathbf{I}^*$. (c) Starting image difference $\mathbf{I}-\mathbf{I}^*$. (d) Final image difference. (e) Pose difference $\mathbf{r}-\mathbf{r}^*$. (f) Error in the latent space $\mathbf{z}-\mathbf{z}^*$. (g) Camera velocities $\mathbf{v}$. (h) 3D trajectories of the different methods.

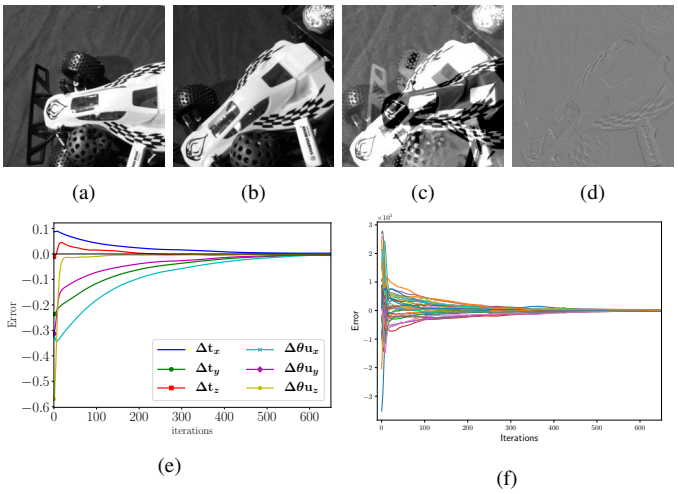


Fig. 8: Second experiment: (a) Starting image $\mathbf{I}$. (b) Desired image $\mathbf{I}^*$. (c) Starting image difference $\mathbf{I}-\mathbf{I}^*$ (d) Final image difference. (e) Pose difference $\mathbf{r}-\mathbf{r}^*$. (f) Error in the latent space $\mathbf{z}-\mathbf{z}^*$.

[4] E. Marchand, "Subspace-based visual servoing," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2699–2706, July 2019.

[5] ——, "Direct visual servoing in the frequency domain," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 620–627, Apr. 2020.

[6] N. Crombez, E. Mouaddib, G. Caron, and F. Chaumette, "Visual Servoing with Photometric Gaussian Mixtures as Dense Feature," *IEEE Trans. on Robotics*, vol. 35, no. 1, pp. 49–63, Jan. 2019.

[7] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.

[9] 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 ICCV*, 2015, pp. 2758–2766.

[10] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," *IEEE ICCV*, pp. 2938–2946, 2015.

[11] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–7, 08 2006.

[12] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke, "Training Deep Neural Networks for Visual Servoing," in *IEEE ICRA 2018*, May 2018, pp. 3307–3314.

[13] A. Saxena, H. Pandya, G. Kumar, A. Gaud, and K. M. Krishna, "Exploring convolutional networks for end-to-end visual servoing," in *IEEE ICRA 2017*, May 2017, pp. 3817–3823.

[14] C. Yu, Z. Cai, H. Pham, and Q. C. Pham, "Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing," in *IEEE/RSJ IROS*, 2019, pp. 935–941.

[15] S. Felton, E. Fromont, and E. Marchand, "Siame-se(3): regression in se(3) for end-to-end visual servoing," in *IEEE ICRA'21*, Xi'an, China, May 2021.

[16] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The J. of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[17] H. Van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," in *IEEE/RSJ IROS'16*, 2016, pp. 3928–3934.

[18] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz, "A practical approach to insertion with variable socket position using deep reinforcement learning," in *IEEE ICRA'19*, 2019, pp. 754–760.

[19] J. Mattner, S. Lange, and M. Riedmiller, "Learn to swing up and balance a real pole based on raw visual input data," in *International Conference on Neural Information Processing*, 2012, pp. 126–133.

[20] S. Nair and C. Finn, "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation," *arXiv preprint arXiv:1909.05829*, 2019.

[21] E. Marchand, "Control camera and light source positions using image gradient information," in *IEEE ICRA'07*, 2007, pp. 417–422.

[22] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE PAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[23] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *ICML'11*, Madison, 2011, p. 833–840.

[24] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *ICML'08*, 2008, pp. 1096–1103.

[25] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *IEEE ICRA'16*, 2016, pp. 512–519.

[26] E. Y. Puang, K. P. Tee, and W. Jing, "Kovis: Keypoint-based visual servoing with zero-shot sim-to-real transfer for robotics manipulation," in *IEEE IROS'20*, 2020, pp. 7527–7533.

[27] E. Plaut, "From principal subspaces to principal components with linear autoencoders," *arXiv preprint arXiv:1804.10253*, 2018.

[28] D. Kunin, J. Bloom, A. Goeva, and C. Seed, "Loss landscapes of regularized linear autoencoders," in *ICML'19*, vol. 97, Jun 2019, pp. 3560–3569.

[29] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018.

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

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML'15*, 2015, p. 448–456.

[32] T. Salimans and D. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Neural Information Processing Systems 2016*, 2016.

[33] S. Felton, P. Brault, E. Fromont, and E. Marchand, "Visual servoing in autoencoder latent space: supplementary material," https://hal.inria.fr/hal-03448667, 2021.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.