Direct visual servoing in the non-linear scale space of camera pose

Guillaume Caron, Yusuke Yoshiyasu

Direct visual servoing in the non-linear scale space of camera pose

Guillaume Caron
CNRS-AIST JRL (Joint Robotics Laboratory), IRL, Japan
Université de Picardie Jules Verne, MIS laboratory, France
Email: guillaume.caron@u-picardie.fr

Yusuke Yoshiyasu
AIST AIRC, Japan
Email: yusuke-yoshiyasu@aist.go.jp

***Abstract*—This paper proposes to consider direct visual servoing (DVS) for object manipulation by a robot arm. The convergence domain limits of DVS are overcome by introducing the non-linear scale space related to camera pose. Its use in a new, yet little complex, direct cost can enlarge twice the convergence domain of one state-of-the-art DVS, as many experiments of symmetric object orientation control assess.**

I. Introduction

Visual Servoing (VS) controls a robot to actively align its camera stream to a desired view [6]. Image alignment is also performed by warping in Simultaneous Localization And Mapping (SLAM) [4] and Visual Odometry (VO) [16]. Usually, the alignment minimizes a cost function defined from the desired image (or geometric data) and the current acquired image. They either rely on matched features [11], [18], direct minimization of image brightness differences, *e.g.* Photometric VS (PVS) [7], or color-depth (RGBD) data [19], [2].

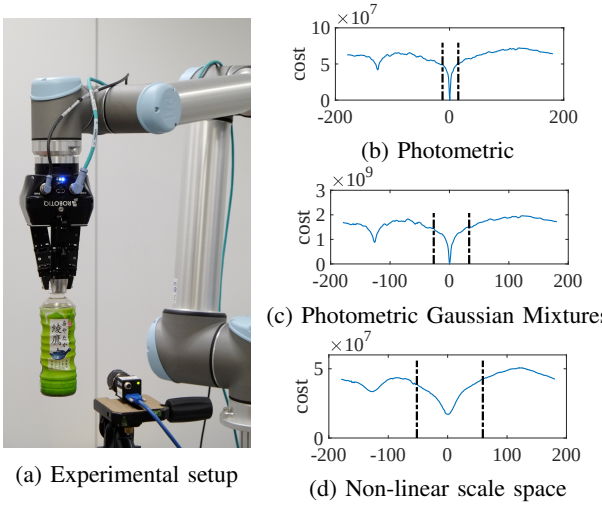
Most image alignment techniques have trouble balancing precision and width of convergence domain. Feature-based alignment can find its bound when matching features is impossible or imprecise [14]. On the other hand, direct alignment is very precise but within a tight convergence domain, as the seminal PVS work [7] showed (Fig. 1b). This fact limits direct alignment approaches to frame-by-frame tracking [10].

Recent progresses have made both Direct VS (DVS) [8], [9], [12], [5] and image alignment [1], [21] achieve large convergence domains. One of the key ideas was to control the Gaussian smoothing of images together with camera pose Degrees of Freedom (DoF) [8], [1]. Image smoothing removes local discontinuities of the image, hence those of the direct cost function [15], enlarging its convergence basin (Fig. 1c).

This work extends these lines of work by introducing a new Non-Linear Scale Space (NLSS) approach to DVS. To the best of our knowledge, NLSS has only been developed in the image denoising field for preserving natural edges [17], [3] or used as a consequence of optical properties, *e.g.* defocus [5]. The new NLSS considers anisotropic Gaussian filtering *depending on camera pose DoFs*. It provides larger convergence domains (Fig. 1d) than DVS with isotropic Gaussian filtering [8].

The main contributions of this work are summarized as:

- New NLSS model related to camera pose DoF, which shows generality over isotropic Gaussian smoothing
- A DVS control law based on the new NLSS capable of very large convergence domain (Fig. 1d)
- A method to build the non-linear scale space of the desired image offline, even with unknown depth



(a) Experimental setup (b) Photometric (c) Photometric Gaussian Mixtures (d) Non-linear scale space

Fig. 1: (a) Axially-symmetric object showing its main label. (b-d) Direct cost functions (abscissa: object angle in degrees) for (b,c) baseline and (d) new costs; dashed lines are convergence borders of width (degrees): (a) 25, (b) 54, (c) 107.

The contributions are evaluated for the automatic storage of goods on supermarket shelves, focusing on the difficult case of axially-symmetric objects: the robot grasps an object in unknown orientation around its axis of symmetry and must align the object to the desired orientation showing its main label (Fig. 1a). To our knowledge, this is the first time DVS is applied to object manipulation. Such a setup requires at least 3 camera DoFs that NLSS can formally deal with (see Sec. III-B). Experimental results with various types of objects in two setups show that NLSS can double the width of the convergence domain of [8].

In the rest of the paper, Section II recalls the state-of-the-art Photometric Gaussian Mixtures-based VS relying on isotropic Gaussians. Section III introduces the new non-linear scale space related to camera pose. It is then considered for DVS (Sec. IV). After that, the new NLSS alignment cost is evaluated with respect to state-of-the-art costs on a dozen of objects in Section V. Finally, DVS experiments with a robot arm assess the contributions (Sec. VI) before conclusion (Sec. VII).

II. Photometric Gaussian Mixtures-based DVS

A. PGM model and control law

We first review the Photometric Gaussian Mixtures-based DVS (PGM VS) [8], which relies on isotropic Gaussians.

Instead of considering image I with $N \times M$ pixels as input to the control law, PGM VS takes a Photometric Gaussian Mixture of the input image, G. Note that G mixes Photometric Gaussians (PG) of every pixel $I(\mathbf{u})$, *i.e.* $I : \mathbf{u} \in \mathcal{U} \longmapsto I(\mathbf{u}) \in [\![0, 255]\!]$, with $\mathcal{U} = [\![0, N-1]\!] \times [\![0, M-1]\!]$. A single PG is characterized by its *center* $\mathbf{u}$ and its *spread* $\lambda \in \mathbb{R}_+^*$, *i.e.* mean and standard deviation ($\mathbb{R}_+^* = \mathbb{R}_+ \setminus \{0\}$). Hence a PG is defined as $g : \mathbf{u}_g \in \mathcal{U}_g \longmapsto g(\mathbf{u}_g, I, \mathbf{u}, \lambda)$:

$$g(\mathbf{u}_g, I, \mathbf{u}, \lambda) = I(\mathbf{u}) \exp\left(-||\mathbf{u}_g - \mathbf{u}||^2/(2\lambda^2)\right). \quad (1)$$

Here g models the strength of attraction at pixel location $\mathbf{u}$ in image I [8]. If the definition domain of the PG is the same as the one of the image, then $\mathcal{U}_g = \mathcal{U}$.

Then, the PGM G is defined with a single λ as:

$$G(\mathbf{u}_g, I, \lambda) = \sum_{\mathbf{u} \in \mathcal{U}} g(\mathbf{u}_g, I, \mathbf{u}, \lambda). \quad (2)$$

As $\mathcal{U}$ is a discrete set, the exponential in the expression (1) of g is evaluated at discrete locations $\mathbf{u} \in \mathcal{U}$, thus it is sampled. Hence, in short, if λ reaches 0, then $G(\mathbf{u}_g)$ shrinks to $I(\mathbf{u})$ and PGM VS to PVS [7] (not recalled here for conciseness).

The control law of PGM VS is designed to minimize the Sum of Squared Differences (SSD) cost between current G, *i.e.* the PGM of I with λ, and G^*, *i.e.* the PGM of the desired image I^* with $\lambda^* \in \mathbb{R}_+^*$. PGM VS considers λ as an additional DoF [8] to those of camera pose $\mathbf{p} = [\mathbf{t}^\top, \theta, \mathbf{w}^\top]^\top$, representing the rigid transformation from the world frame $\mathcal{F}_w$ to the camera frame $\mathcal{F}_c$. In $\mathbf{p}$, $\mathbf{t} = [t_X, t_Y, t_Z]^\top \in \mathbb{R}^3$ is the 3D translation, and $\mathbf{w} = [w_X, w_Y, w_Z]^\top \in \mathbb{R}^3 : ||\mathbf{w}|| = 1$ with angle $\theta \in \mathbb{R}$ represent the 3D rotation as axis-angle.

To express the PGM SSD cost, elements of G are stacked as vector $\mathbf{G} \in \mathbb{R}^{|\mathcal{U}_g| \times 1}$ and those of G^* as $\mathbf{G}^* \in \mathbb{R}^{|\mathcal{U}_g| \times 1}$:

$$\mathscr{C}_{PGM}(\mathbf{p}, \lambda) = \frac{1}{2}||\mathbf{G}(\mathbf{p}, \lambda) - \mathbf{G}^*||^2. \quad (3)$$

The control law minimizing $\mathscr{C}_{PGM}$ is then expressed as:

$$\begin{bmatrix}\mathbf{v}^\top, & \dot{\lambda}\end{bmatrix}^\top = -\mu \mathbf{J_G}^+ (\mathbf{G}(\mathbf{p}, \lambda) - \mathbf{G}^*), \quad (4)$$

where $\mathbf{v} \approx \dot{\mathbf{p}}$, $\mu \in \mathbb{R}_+^*$ is a gain to tune the convergence pace, $\mathbf{J_G} \in \mathbb{R}^{|\mathcal{U}_g| \times D+1}$ juxtaposes $\mathbf{L_G} \in \mathbb{R}^{|\mathcal{U}_g| \times D}$, *i.e.* the interaction matrix related to a PGM sample (natural integer $D = 6$ is all 6 DoF are controlled, least otherwise), and $\mathbf{J}_\lambda \in \mathbb{R}^{|\mathcal{U}_g| \times 1}$, *i.e.* the Jacobian of the PGM sample with respect to λ. [8] details their expressions but one may note that $\mathbf{L_G}$ is function of geometric interaction matrices involving the pinhole camera model. It transforms the 3D point expressed in the camera frame $\mathbf{X} = [X, Y, Z]^\top \in \mathbb{R}^3$ into 2D image space. Let $\mathbf{x} \in \mathbb{R}^2$ be the projection of $\mathbf{X}$ into the intermediate normalized image plane:

$$\mathbf{x} = \begin{bmatrix}x, & y\end{bmatrix}^\top = \begin{bmatrix}X/Z, & Y/Z\end{bmatrix}^\top, \quad (5)$$

and $\tilde{\mathbf{x}} \in \mathbb{P}^2$ its homogeneous representation. Then, the 2D point $\tilde{\mathbf{u}} \in \mathbb{P}^2$ represented as the homogeneous coordinates of pixel location $\mathbf{u}$ is obtained by:

$$\tilde{\mathbf{u}} = \mathbf{K}\tilde{\mathbf{x}}, \quad (6)$$

where $\mathbf{K}$ is a camera matrix containing intrinsic parameters $\alpha_u \in \mathbb{R}_+$, $\alpha_v \in \mathbb{R}_+$, $\mathbf{u}_0 = (u_0, v_0) \in \mathbb{R}_+^2$:

$$\mathbf{K} = \begin{pmatrix}\alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1\end{pmatrix}. \quad (7)$$

The evolution of λ (λ^* is constant) while the camera is moving is one of the PGM VS's key to both wide convergence domain (with large λ) and precision (with small λ). [8] reports a sequence of PGM VS: Step 1 with λ^* large (exact value depends on experiments) and $\lambda = 2\lambda^*$ at initialization; Step 2 with constant $\lambda = \lambda^* = 1$.

B. Analysis on PGM VS

The contributions of PGM VS actually comes with two issues. The first one is theoretical, related to isotropic Gaussian smoothing. Indeed, as the expression of PGM (2) shows, the image is smoothed uniformly. This makes distant objects smoothed as much as near objects, despite their scale differences in the image. Hence, the information brought by distant objects, which is usually relevant to constrain rotations [20], can simply disappear. Another nested issue regards the expression of PG (1). Its Gaussian spread λ is expressed in the image domain. Hence, λ depends on image resolution [8] with currently no way to relate it to camera motion.

The second issue is practical. PGM VS smooths each acquired image for the spread value of current iteration. Although it may provide both large convergence domain and precision, its algorithmic complexity needs dedicated hardware and implementation to run about online (10 Hz with images of 100 $\times$ 100 pixels on the Graphics Processing Unit [8]).

NLSS (Sec. III) is designed to address the theoretical aspect while NLSS VS (Sec. IV) targets the practical issue.

III. Non-linear scale-space related to pose

This section introduces a new type of NLSS based on anisotropic filtering with Gaussian kernels depending on the camera pose DoF, the camera projection model and the pixel position in the image plane. NLSS considers the relative pose $\boldsymbol{\delta}\mathbf{p} \in \mathbb{R}^6$ (defined similarly to $\mathbf{p}$, Sec. II-A) between two camera frames $\mathcal{F}_{c^*}$ and $\mathcal{F}_c$. Rodrigues' formulae express ${}^c\mathbf{M}_{c^*} \in \mathrm{SE}(3)$ from $\boldsymbol{\delta}\mathbf{p}$, transforming coordinates from frame $\mathcal{F}_{c^*}$ to $\mathcal{F}_c$. Thus, by writing the homogeneous representation of 3D point $\mathbf{X}$ as $\tilde{\mathbf{X}} \in \mathbb{P}^3$ and expressing it in frames $\mathcal{F}_c$ and $\mathcal{F}_{c^*}$, we have ${}^c\tilde{\mathbf{X}} = {}^c\mathbf{M}_{c^*}\, {}^{c^*}\tilde{\mathbf{X}}$.

In the rest of this section, we express monodimensional Gaussians in the image plane along the curve defined by projections $\mathbf{u}$ of $\mathbf{X}$, function of each element of $\boldsymbol{\delta}\mathbf{p}$.

A. Single DoF anisotropic kernels

For each pixel $I(\mathbf{u})$, $\mathbf{X}$ projects as $\mathbf{u}$ for $\boldsymbol{\delta}\mathbf{p} = \mathbf{0}$. Then, we define $\mathbf{u}_n = [u_n, v_n]^\top \in \mathbb{R}^2$, the coordinates of filtered image I_n, and $\tilde{\mathbf{u}}_n \in \mathbb{P}^2$ their homogeneous representation.

1) Anisotropic kernel related to t_X: To express the monodimensional Gaussian curve of the anisotropic kernel, we write $\mathbf{u}_n$ function of $\mathbf{X}$ and $\boldsymbol{\delta p} = [t_X, \mathbf{0_{1\times 5}}]^\top$, $t_X \neq 0$:

$$\tilde{\mathbf{u}}_n = \mathbf{K}\left[\left({}^{c^*}X + t_X\right)/{}^{c^*}Z, \quad {}^{c^*}Y/{}^{c^*}Z, \quad 1\right]^\top, \tag{8}$$

leading to:

$$\mathbf{u}_n = \mathbf{u} + \left[\alpha_u t_X/{}^{c^*}Z, \quad 0\right]^\top = \mathbf{u} + \boldsymbol{\Delta}_\mathbf{u}(t_X). \tag{9}$$

$\mathbf{u}_n$ hence belongs to the line of equation $v_n = v$.

The anisotropic kernel related to t_X is then expressed as:

$$f_n(\mathbf{u}_n, \mathbf{u}, t_X) = \frac{1}{\sqrt{2\pi}\lambda_n(t_X)} \exp\left(-\frac{(u_n - u)^2}{2\lambda_n(t_X)^2}\right), \tag{10}$$

$\forall \mathbf{u}_n \ : \ v_n = v$, with:

$$\lambda_n(t_X) = ||\boldsymbol{\Delta}_\mathbf{u}(\hat{t}_X)||, \tag{11}$$

such that $\hat{t}_X$ is the magnitude of camera horizontal translation parameterizing the scale ($\sim$*spread* in Sec. II).

The above defined kernel is anisotropic since it depends on the Z coordinate of the 3D point projected as $\mathbf{u}$ (Fig. 2). Considering the same Z for every pixel leads back to the isotropic Gaussian kernel, but for the *scale parameter* $\hat{t}_X$.

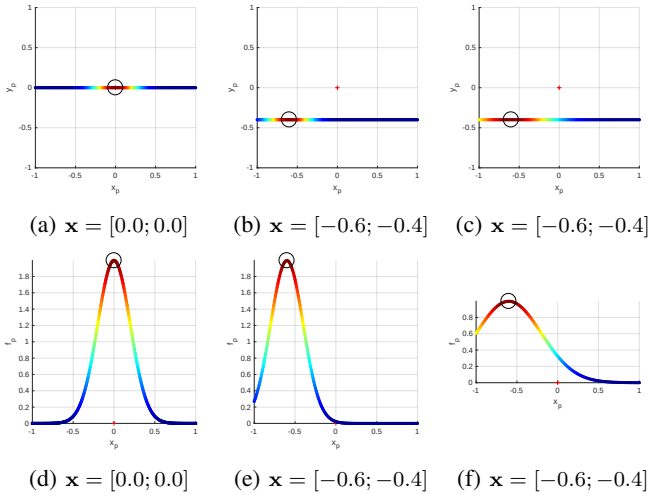
2) Anisotropic kernel related to t_Z: Generally speaking, t_Z leads to a Gaussian kernel of 2 dimensions. Indeed, from $\boldsymbol{\delta p} = [\mathbf{0_{1\times 2}}, t_Z, \mathbf{0_{1\times 3}}]^\top$, $t_Z \neq 0$, we get:

$$\tilde{\mathbf{u}}_n = \mathbf{K}\begin{bmatrix} \frac{{}^{c^*}X}{{}^{c^*}Z + t_Z} \\ \frac{{}^{c^*}Y}{{}^{c^*}Z + t_Z} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\alpha_u x\ {}^{c^*}Z}{{}^{c^*}Z + t_Z} + u_0 \\ \frac{\alpha_v y\ {}^{c^*}Z}{{}^{c^*}Z + t_Z} + v_0 \\ 1 \end{bmatrix}, \tag{12}$$

rewritten for u_n and v_n as:

$$\mathbf{u}_n = \mathbf{u} - \frac{t_Z}{({}^{c^*}Z + t_Z)}\begin{bmatrix}(u - u_0) \\ (v - v_0)\end{bmatrix} = \mathbf{u} + \boldsymbol{\Delta}_\mathbf{u}(t_Z). \tag{13}$$

Hence, $\mathbf{u}_n$ belongs to the half-line from $\mathbf{u}_0$ toward $\boldsymbol{\Delta}_\mathbf{u}(t_Z)$.



(a) $\mathbf{x} = [0.0; 0.0]$ (b) $\mathbf{x} = [-0.6; -0.4]$ (c) $\mathbf{x} = [-0.6; -0.4]$

(d) $\mathbf{x} = [0.0; 0.0]$ (e) $\mathbf{x} = [-0.6; -0.4]$ (f) $\mathbf{x} = [-0.6; -0.4]$

Fig. 2: Anisotropic kernel function of t_X on two $\mathbf{x}$ (black circles) for $\hat{t}_X = 0.2$ m: (a-d) $Z = 1.0$ m; (e-f) $Z = 0.5$ m.

Then, let us express the kernel function $f_n(\mathbf{u}_n, \mathbf{u}, t_Z)$, such that it represents a Gaussian centered at $\mathbf{u}$, of spread:

$$\lambda_n(t_Z) = ||\boldsymbol{\Delta}_\mathbf{u}(\hat{t}_Z)||, \tag{14}$$

similar to the anisotropic kernel of t_X (Sec. III-A1), but only defined for points $\mathbf{u}_n$ verifying (13):

$$f_n(\mathbf{u}_n, \mathbf{u}, t_Z) = \frac{\exp\left(-\frac{1}{2}(\mathbf{u}_n - \mathbf{u})^\top \Sigma_n^{-1}(\mathbf{u}_n - \mathbf{u})\right)}{\sqrt{2\pi}\lambda_n(t_Z)}, \tag{15}$$

after posing:

$$\Sigma_n = \boldsymbol{\Delta}_\mathbf{u}(\hat{t}_Z)\boldsymbol{\Delta}_\mathbf{u}(\hat{t}_Z)^\top. \tag{16}$$

The anisotropic kernel of (15) is *even more anisotropic than* the one of t_X of (10), as its spread depends on $\mathbf{u}$, in addition to Z, as Figures 3a and 3b show.

3) Anisotropic kernel related to θ_Y: First, express the curve in the image along which the monodimensional Gaussian related to θ_Y (shorten "$\theta\mathbf{w}$ with $\mathbf{w} = [0, 1, 0]^\top$") is defined. Considering $\boldsymbol{\delta p} = [\mathbf{0_{1\times 4}}, \theta_Y, 0]^\top$, $\theta_Y \neq 0$, we get:

$$\tilde{\mathbf{u}}_n = \mathbf{K}\left[\frac{\left[\left({}^{c^*}X\cos\theta_Y + {}^{c^*}Z\sin\theta_Y\right), \quad {}^{c^*}Y\right]}{-{}^{c^*}X\sin\theta_Y + {}^{c^*}Z\cos\theta_Y}, \quad 1\right]^\top. \tag{17}$$

Multiplying the right vector of (17) by ${}^{c^*}Z/{}^{c^*}Z$ leads to:

$$\tilde{\mathbf{u}}_n = \mathbf{K}\left[\frac{\left[\left(x\cos\theta_Y + \sin\theta_Y\right), \quad y\right]}{-x\sin\theta_Y + \cos\theta_Y}, \quad 1\right]^\top. \tag{18}$$

The above vector defines a curve in the image plane (Fig. 3d), function of θ_Y, $\forall x$, being the perspective projection of a circle perpendicular to the image plane, *i.e.* a hyperbola. To get a simpler expression, we express the same curve with respect to $x' = 0$. To do so, we write y' from $\tilde{\mathbf{x}}$ (right vector of (18)), seen as an orientation vector colinear to the line of sight of $\mathbf{x}$, and $\tilde{\mathbf{x}}' \in \mathbb{P}^2$, an orientation vector of $(x' = 0, y')$:

$$\tilde{\mathbf{x}} = [x, y, 1]^\top \quad \text{and} \quad \tilde{\mathbf{x}}' \propto [0, y, \sqrt{x^2 + 1}]^\top, \tag{19}$$

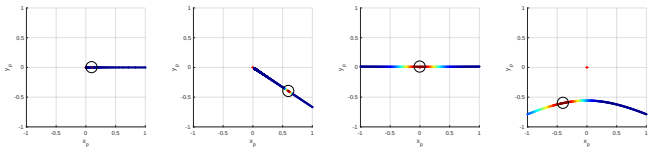
since $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ are the same, up to a pure rotation around camera axis $\mathbf{Y}_c$. We then express:

$$y' = y/\sqrt{x^2 + 1}. \tag{20}$$

Substituting x and y with x' and y' in (18) and (17) we get:

$$\mathbf{u}_n = \begin{bmatrix} \alpha_u \tan\theta_Y + u_0 \\ (v - v_0)\,/\left(\cos\theta_Y\sqrt{x^2 + 1}\right) + v_0 \end{bmatrix}, \tag{21}$$

with $x = (u - u_0)/\alpha_u$.



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

Fig. 3: Kernels of: t_Z on (circles) (a) $\mathbf{x} = [0.1; 0]$, (b) $\mathbf{x} = [0.6; -0.4]$, for $\hat{t}_Z = 0.2$ m, $Z = 0.5$ m; θ_Y on (c) $\mathbf{x} = [0; 0.01]$, (d) $\mathbf{x} = -[0.4; 0.6]$ for $\hat{\theta}_Y = \pi/14$, *i.e.* 13 degrees.

We then express the anisotropic kernel $f_n(\mathbf{u}_n, \mathbf{u}, \theta_Y)$ of hyperbolic support, for $\mathbf{u}_n$ of (21). Posing:

$$d_{\theta_Y} = [u_n, 1][u_0, 1]^\top \;/\; \left(|||[u_n, 1]|| \; ||[u_0, 1]||\right), \tag{22}$$

we get:

$$f_n(\mathbf{u}_n, \mathbf{u}, \theta_Y) = \frac{1}{\sqrt{2\pi}\lambda_n(\theta_Y)} \exp\left(-\frac{\arccos\left(d_{\theta_Y}\right)^2}{2\lambda_n(\theta_Y)^2}\right), \tag{23}$$

where:

$$\lambda_n(\theta_Y) = |\hat{\theta}_Y|. \tag{24}$$

Figures 3c and 3d show examples of kernels of θ_Y.

4) Anisotropic kernels related to t_Y, θ_X and θ_Z: These kernels are not detailed for conciseness but the anisotropic kernel related to t_Y, resp. θ_X, is very similar to the one related to t_X (Sec. III-A1), resp. θ_Y (Sec. III-A3), up to a transpose.

The anisotropic kernel related to θ_Z is expressed following the same method than the other DoFs. Its shape is circular.

B. Combining anisotropic kernels

The most obvious combination of monodimensional kernels considers those related to t_X and t_Y (resp. Sec. III-A1 and III-A4) giving the bidimensional kernel $F_n()$:

$$F_n(\mathbf{u}_n, \mathbf{u}, t_X, t_Y) = f_n(\mathbf{u}_n, \mathbf{u}, t_X) f_n(\mathbf{u}_n, \mathbf{u}, t_Y). \tag{25}$$

Assuming a fronto-parallel planar scene (Z constant for all $\mathbf{u}$) and $t_X = t_Y = t_{XY}$, F_n of (25) becomes the conventional expression of the isotropic 2D Gaussian (homeoscedastic):

$$\begin{aligned} F_n(\mathbf{u}_n, \mathbf{u}, t_X, t_Y) &= \frac{\exp\left(-\frac{(u_n-u)^2}{2\lambda_n(t_X)^2}\right)}{\sqrt{2\pi}\lambda_n(t_X)} \frac{\exp\left(-\frac{(v_n-v)^2}{2\lambda_n(t_Y)^2}\right)}{\sqrt{2\pi}\lambda_n(t_Y)} \\ &= \frac{1}{2\pi\lambda_n(t_{XY})^2} \exp\left(-\frac{(u_n-u)^2 + (v_n-v)^2}{2\lambda_n(t_{XY})^2}\right). \end{aligned} \tag{26}$$

F_n of (26) is a particular case of (25) equivalent to the Gaussian kernel used in every work of image alignment exploiting, explicitly or not, the linear scale space [19], [10], [8], [1]. Ignoring the factor in front of the exponential, (26) is the same exponential as for a Photometric Gaussian (1) of PGM VS. Figures 4a to 4d show several examples of 2D kernels of (26), for various $\mathbf{u}$ and Z. By extension, mixtures of above mentioned 2D kernels are isotropic iff Z is constant (Fig. 4e and 4f) but anisotropic in general (Fig. 4g and 4h).

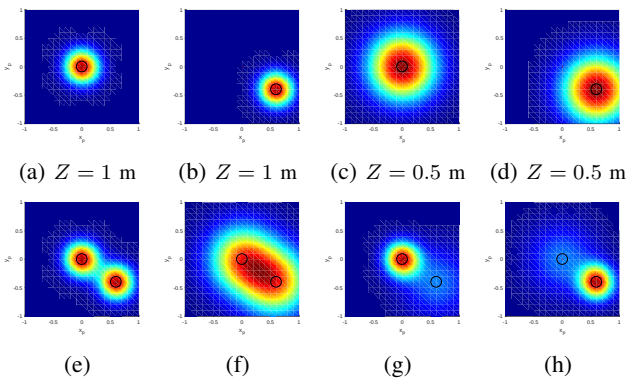
IV. NLSS Visual Servoing

This Section describes the new VS control law exploiting the NLSS filtered *desired image* (Sec. IV-A) and the computation of the *desired image* for NLSS-VS (Sec. IV-B).

A. Control law

The new NLSS VS control law is similar to (4) but only filters the desired image I^* anisotropically to obtain I_n^*. By stacking pixels of the current and desired images as vectors $\mathbf{I}(\mathbf{p})$ and $\mathbf{I_n}^*$, the new NLSS SSD cost is defined as:

$$\mathscr{C}_{NLSS}(\mathbf{p}) = \tfrac{1}{2}||\mathbf{I}(\mathbf{p}) - \mathbf{I_n}^*||^2. \tag{27}$$



(a) $Z = 1$ m (b) $Z = 1$ m (c) $Z = 0.5$ m (d) $Z = 0.5$ m

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

Fig. 4: (a-d) isotropic kernels for $\hat{t}_X = \hat{t}_Y = 0.2$ m on two $\mathbf{x}$: (a,c) $x = 0.0$, $y = 0.0$; (b,d) $x = 0.6$, $y = -0.4$. (e-f) mixtures of isotropic kernels (a, b), resp. (c, d). (g-h) mixtures of anisotropic kernels (b, c), resp. (a, d).

Thus, as I_n^* is constant over time, the same interaction matrix $\mathbf{L_I}$ as for PVS [7] is used in the NLSS VS control law:

$$\mathbf{v} = -\mu \mathbf{L_I}^+ \left(\mathbf{I}(\mathbf{p}) - \mathbf{I_n}^*\right). \tag{28}$$

In (28), $\mathbf{L_I}^+$ is the pseudo-inverse of the photometric interaction matrix $\mathbf{L_I}$ and μ is a gain (see (4)). $\mathbf{L_I}$ involves image gradients, camera intrinsic parameters and scene depth at each pixel (see [7] for the details). When the scene depth is unknown, setting the same constant depth for every pixel has shown the robustness of (4) to such approximation [8], [7]. Thus, the same behavior is expected for (28).

B. Generating the desired image for the control law

In our context, a static camera looks at a symmetric object rotated by a robot arm around its axis of symmetry. This is dual to the camera rotating around the object at constant depth. Hence, three camera degrees of freedom are sufficient: one rotation around the object axis of symmetry and two translations in the plane orthogonal to that axis. We consider the object axis $\mathbf{Z_o}$ fits the axis of symmetry. Assuming $\mathbf{Z_o}$ is perpendicular to the plane formed by camera axes $\mathbf{X_c}$ and $\mathbf{Z_c}$, the three camera DoFs are: t_X, t_Z, θ_Y. These DoFs are those to consider when generating the NLSS desired image I_n^*. Thus, the NLSS combines kernels (dropping $\mathbf{u}_n$ and $\mathbf{u}$ for compactness) $f_n(t_X)$, $f_n(\theta_Y)$ and $f_n(t_Z)$ (resp. (10), (23), (15)) as $F_n(t_X, \theta_Y, t_Z)$ in a same fashion as $F_n(t_X, t_Y)$ does (25).

Ideally, the *desired image* I_n^* in (28) would be the result of the NLSS filtering with kernel $F_n(t_X, \theta_Y, t_Z)$ of a single acquired image of the object at desired orientation ${}^o\theta_Z^*$ (here expressed in the object frame $\mathcal{F}_o$ for brevity). As the 2D anisotropic kernel $F_n(t_X, \theta_Y, t_Z)$ involves translations, it depends on scene depth, thus object shape. We propose a procedure to compute I_n^*, *even when the depth is not available*. Thus, only a passive monocular camera is required and 3D models or depth measurements are not necessary.

To compute I_n^*, equivalently to the explicit filtering of I^* with kernels of the NLSS (Sec. III), we acquire a set of images

with several object orientations around ${}^o\theta^*_Z$ and compute once the Gaussian weighted sum of this set of images. The interval of orientations and the Gaussian weights depend on the NLSS scale parameter ${}^o\hat{\theta}_Z$, equivalent to three scale parameters in the camera frame. So, for a given ${}^o\hat{\theta}_Z$, we define the range of orientations Ω around the object desired orientation ${}^o\theta^*_Z$ as:

$$\Omega = [{}^o\theta^*_Z - N_\lambda {}^o\hat{\theta}_Z, {}^o\theta^*_Z + N_\lambda {}^o\hat{\theta}_Z], \tag{29}$$

where N_λ is nothing but a well-known integer factor leading Ω to cover 68.3%, 95.5% or 99.7% of the Gaussian curve integral for N_λ equal, respectively, to 1, 2 or 3.

Then, each pixel value $I^*_n(\mathbf{u})$ of the *desired image* is computed as the bounded integral of pixel values acquired at the same location $\mathbf{u}$ but spanning the whole range Ω, *i.e.*:

$$I^*_n(\mathbf{u}) = \int_\Omega \frac{I(\mathbf{u}, {}^o\theta_Z)}{\sqrt{2\pi}|{}^o\hat{\theta}_Z|} \exp\left(-\frac{({}^o\theta_Z - {}^o\theta^*_Z)^2}{2|{}^o\hat{\theta}_Z|^2}\right) d\,{}^o\theta_Z. \tag{30}$$

Although a continuous acquisition procedure exposing the camera while rotating the object at a speed varying accordingly to the inverse of the Gaussian of (30) could be imagined, it would be very complex to implement. To make the computation of $I^*_n(\mathbf{u})$ easy, Ω is sampled with step $\delta_\theta \in \mathbb{R}^*_+$, and a discrete set of images is acquired for each element of Ω. These images are summed, each weighted with the Gaussian in (30). The cardinality of Ω gives a normalizing factor to ensure pixel values of the *desired image* I^*_n remain in the $[0, 255]$ interval.

V. Cost functions experimental comparison

In order to compare cost functions of NLSS VS (27) with respect to the state-of-the-art PGM VS (3) and the classical PVS [7], we consider 13 objects of various sizes (bottles and food boxes, Fig. 5a, 5b). Each object is axial-symmetric and its frame is set so that axis $\mathbf{Z_o}$ fits the axis of symmetry. Hence, we can simply note ${}^o\theta_Z$ as the object angle around its axis of symmetry. It should be noted that due to this symmetry, only the object appearance, not its geometry, allows us to detect orientation changes around $\mathbf{Z_o}$.

6850 images (http://mis.u-picardie.fr/~g-caron/pub/data/ConveJRL.zip, 4 GB) of 720×960 pixels were acquired by a webcam, while each object slowly rotated for 360 degrees around $\mathbf{Z_o}$ on the center of a turn table. The constant rotation speed was set so that the angle between neighboring acquired images is $\delta_\theta = 0.8$ degrees (Sec. IV-B) on average. Then, to compare the NLSS cost (27) (N-cost) with baseline PGM (4) (G-cost) and photometric (P-cost: SSD between I and I^*) costs, one image I^* per object is selected as the one facing the most the main label to the camera (Fig. 5a, 5b). After that, I^*_n is computed from images before and after I^*, in the acquired image sequence, among the range of angles spanning Ω (Sec. IV-B). As Ω, hence I^*_n, depends on the scale parameter ${}^o\hat{\theta}_Z$ of the NLSS related to the rotation around object axis of symmetry, several values of ${}^o\hat{\theta}_Z$ are considered for comparison but a single $N_\lambda = 3$ (examples of I^*_n filtered with ${}^o\hat{\theta}_Z = 6$ degrees, Fig. 5c, 5d).

The convergence domain width is experimentally evaluated. For an object, every acquired image while the object was rotating is used to evaluate the cost functions from -180 to +180 degrees (see Fig. 1b, 1c and 1d for the *ayataka* object). Then, the cost function is numerically differentiated to find left ${}^o\theta^{(l)}_Z$ and right ${}^o\theta^{(r)}_Z$ borders of the convergence domain of the global minimum. As these borders might not be symmetric, the convergence domain width is computed as twice the minimum absolute value among ${}^o\theta^{(l)}_Z$ and ${}^o\theta^{(r)}_Z$.

Table I shows the width of convergence domain of all three costs for various values of spread (Sec. II) and scale (Sec. III), for every object. The N-cost always outperforms the P-cost, sometimes with very large ratios, *e.g.* ×10 for *noodles*. Surprisingly, the G-cost does not outperform the P-cost for one object. For three objects, N-cost is on par with G-cost. Out of them, N-cost shows a larger domain than G-cost in 6 cases. Conversely, G-cost shows a larger domain than N-cost in 4 cases. Actually, local minima appear in the N-cost very close to the global optimum with large ${}^o\hat{\theta}_Z$. A slight modification of the N-cost, filtering isotropically the current image with a little spread (see the 8 columns on the right of Tab. I), overcomes the issue and makes the N-cost largely outperforming the G-cost for 11 over 13 objects.

VI. Visual servoing experiments

We use a Universal Robot 10 (UR10) with a Robotiq 3-finger gripper (Fig. 1a). They are connected via a wired network to a laptop (Intel Core i7, Ubuntu 16.04) that runs VS control laws. Images with 512×640 pixels are acquired by a Flir 30 FPS camera (Computar lens, focal length 8 mm). Each object area in the image is manually selected to avoid perturbations and ensure a fair comparison of VS methods.

The camera intrinsic parameters (7) were computed from device datasheets. The relative transformation between the end-effector to the object frame was set to the identity matrix as the gripper could grasp objects from their cap or top cover (its Z-axis is aligned with each object axis of symmetry). The camera to robot calibration was done from a single pose computation of a 4 dots marker held by the gripper using the ViSP library [13] and the kinematics obtained from UR10.

PVS, PGM VS and the new NLSS VS were applied on the objects of Section V. Initial poses ${}^o\theta^{(0)}_Z$ were set by incrementally increasing the angular error with respect to the desired orientation ${}^o\theta^*_Z$ with a step of 10 degrees, reduced to 5 to help ranking methods. Gains of the three control laws were manually set to allow the maximum speed without

(a) *ayataka* (b) *noodles* (c) (d)

Fig. 5: (a-b) Axial-symmetric objects (2 among 13). Ayataka is a tea bottle. (c-d) Corresponding *desired* NLSS images I^*_n.

TABLE I: Convergence domain width [degrees] comparison between P-cost, G-cost and N-cost, for various spreads (sp) λ and λ^* (pixel unit), and scales (sc) ${}^o\hat{\theta}_Z$ (degrees). **Bold**† shows the largest domain per object. If no tie, **Bold**§ shows the second largest domain with another method. A negative sign indicates that the method changes the global optimum.

	P	G-cost																N-cost								N-cost with filtered I							
sc/sp*		1	1	1	1	2	2	2	2	5	5	5	5	10	10	10	10	0.2	1.7	3.2	4.7	6.2	7.7	9.2	10.7	0.2	1.7	3.2	4.7	6.2	7.7	9.2	10.7
sp		1	1.2	1.3	2	2	2.3	2.6	4	5	5.8	6.5	10	10	11.5	13	20									1	1	1	1	1	1	1	1
kitsune	17	19	19	7	-1	44	22	7	-1	**62**§	10	0	-0	58	0	-6	-1	17	**62**§	58	55	55	3	3	3	19	62	62	65	55	**70**†	2	2
ayataka	25	56	9	-2	-2	43	9	-2	-2	43	-2	-2	-2	8	-2	-2	-2	25	61	61	61	106	106	**107**†	**107**†	55	66	69	104	104	**106**§	**106**§	**106**§
chips	17	17	-7	-3	-1	19	-6	-3	-1	22	-0	-3	-0	**127**§	-1	-1	-0	17	17	20	60	89	100	120	6	17	19	20	87	90	130	**133**†	**133**†
corn	34	34	7	-1	-0	100	6	-1	-0	99	-1	-0	-0	3	-0	-0	-0	34	100	100	100	100	**130**§	**130**§	1	34	100	100	100	100	**169**†	**169**†	168
noodles	6	10	10	10	9	10	10	13	-3	32	0	0	-3	0	0	-3	-3	6	12	**61**§	**61**§	**61**§	**61**§	1	0	10	12	**87**†	69	69	71	71	71
cheese	6	13	13	13	22	13	13	16	3	**77**†	3	3	-0	3	0	0	-0	6	13	18	36	16	13	10	10	6	13	27	**73**§	65	13	13	2
curry	5	14	14	14	2	14	14	11	-3	**123**†	0	-0	-2	0	-5	-3	-3	5	14	37	74	72	74	74	3	14	14	63	84	84	84	84	**98**§
plastic	7	11	11	9	5	11	12	12	2	45	2	2	4	2	4	4	4	7	11	16	**47**§	**47**§	14	14	14	11	11	16	**56**†	**56**†	**56**†	14	14
gum	10	10	10	0	-2	13	-0	-2	-0	**86**§	-2	-2	-2	-2	-0	-2	-2	10	32	48	52	0	0	0	-0	10	32	84	84	86	**87**†	**87**†	2
jagabee	7	7	7	7	-10	9	10	-10	-0	**34**§	-0	-0	-0	0	-0	-0	-0	7	10	**34**§	**34**§	**34**§	-0	1	1	7	23	42	44	47	**49**†	47	47
koala	33	34	33	31	-1	37	33	34	-1	**48**§	37	-1	-1	45	-1	-1	-3	33	35	44	44	44	**48**§	21	21	34	37	44	44	48	48	62	**65**†
oolong	88	114	48	-4	-2	114	48	-6	-2	118	-13	-2	-2	120	-11	-2	-0	88	114	114	122	122	**123**†	**123**†	0	114	114	118	120	122	122	**123**†	**123**†
suntory	7	7	7	5	-2	7	5	5	-2	7	3	3	3	3	3	3	0	7	**19**†	2	3	-0	-0	-0	-0	7	**19**†	2	3	-0	-0	-0	-0

oscillations at the optimum. NLSS VS runs with the tradeoff value ${}^o\hat{\theta}_Z = 6$ degrees suggested by Table I. On its side, PGM VS runs with two steps but $\lambda = \lambda^*$ constant as Table I shows different values can change the global optimum. It also saves time at runtime, even if images had to be resized to 105×86 pixels to meet the same 10 Hz control loop rate as [8]. Then, PGM VS runs with $\lambda = 1$. If it fails to converge, then λ is successively set to 2, then 5, then 10.

NLSS VS clearly outperforms PVS, always extending the convergence domain, up to 22 times. For instance, NLSS VS converges for -40 degrees $\leq {}^o\theta_Z^{(0)} - {}^o\theta_Z^* \leq 180$ degrees with *cheese* whereas PVS diverges if $|{}^o\theta_Z^{(0)} - {}^o\theta_Z^*| > 5$ degrees. PGM VS outperforms PVS as well but not always. For instance, PVS converges between -35 and 30 degrees with *corn* but PGM VS only converges between -10 and 5 degrees (-35 and 35 degrees for NLSS VS). Hence, on this criterion, NLSS VS performs better than PGM VS too.

When PGM VS performs better than PVS, NLSS VS can also largely outperform PGM VS, for example with *jagabee*: NLSS VS could converge for $-30 \leq {}^o\theta_Z^{(0)} - {}^o\theta_Z^* \leq 180$ degrees, whereas for PGM VS $-50 \leq {}^o\theta_Z^{(0)} - {}^o\theta_Z^* \leq 20$ degrees.

However, there are objects, such as *suntory*, for which NLSS VS' converge domain is only 20 % of PGM VS' (25 versus 120 degrees). Actually, on the 13 considered objects, the average convergence domain width is almost the same for NLSS VS (132.7 degrees) than for PGM VS (131.9 degrees), while PVS' being 34.2 degrees. But as VS results with *corn*, *jagabee* and *suntory* let think, the deviation of the convergence domain width is larger for PGM VS (75.5 degrees) than for NLSS VS (69.6 degrees), making PGM VS' performances harder to predict than NLSS VS ones.

To finalize the comparison, we compute the balance of the convergence domain around ${}^o\theta_Z^*$ by averaging the absolute difference of absolute values of the domain limits (the lower, the better balanced): PGM VS reaches 84.7 degrees, whereas NLSS VS stands at 50.4 degrees, *i.e.* 1.7 times better balanced.

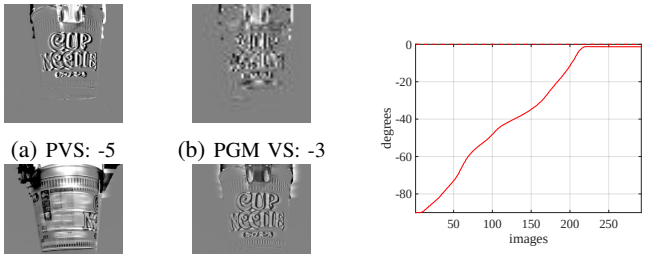
The experiments are gathered in the accompanying video.

Figure 6 shows VS results for the *noodles* box. NLSS VS converges smoothly from the farthest initial angle 30 times the one of PGM VS, 18 times the one of PVS. The error at convergence (Fig. 6d) cannot be zero as I_n^* is filtered but I not (recall the N-cost at the optimum, Fig. 1d). Despite that fact, precision at convergence is 1.2 degrees (0.2 degrees for PVS and 2.3 degrees for PGM VS, both for much smaller maximum initial error than for NLSS VS, see Fig. 6).

Execution times are exactly the same for PVS and NLSS VS: 5 ms per image on average. PGM VS takes 83 ms per image when $\lambda = 10$ and 36 ms when $\lambda = 1$, despite a careful implementation on precomputations and processing images with 36 times less pixels than PVS and NLSS VS.

VII. Conclusion

This paper introduced for the first time the non-linear scale space related to all camera DoF. It enlarges significantly the convergence domain of direct visual servoing for a much lower computational complexity than previous reference works on photometric Gaussian mixtures (similar to a linear scale space). Future works will adapt online the non-linear scale space parameters to improve the convergence pace.



(a) PVS: -5 (b) PGM VS: -3 (c) NLSS VS: -90 (d) NLSS VS: -1.2 (e) angle error over images

Fig. 6: *Noodles* box servoing on main label. (a-c) Cropped initial errors (images or PGMs) at the maximum angular error (degrees) allowing to converge. (d) NLSS VS final error.

REFERENCES

[1] Y. Ahmine, G. Caron, E. Mouaddib, and F. Chouireb. Adaptive lucas-kanade tracking. *Image and Vision Computing*, 88:1 – 8, 2019. 1, 4

[2] M. Argus, L. Hermann, J. Long, and T. Brox. FlowControl: Optical flow based visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2020. 1

[3] D. Boscaini, J. Masci, E. Rodolà, M. M. Bronstein, and D. Cremers. Anisotropic diffusion descriptors. In *Annual Conf. of the European Association for Computer Graphics, Eurographics*, page 431–441, 2016. 1

[4] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser. Simultaneous Localization And Mapping: A Survey of Current Trends in Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, 2017. 1

[5] G. Caron. Defocus-based direct visual servoing. *IEEE Robotics and Automation Letters*, 6(2):4056–4063, 2021. 1

[6] F. Chaumette and S. Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 13:82–90, 2006. 1

[7] C. Collewet and E. Marchand. Photometric visual servoing. *Trans. Rob.*, 27(4):828–834, Aug. 2011. 1, 2, 4, 5

[8] N. Crombez, E. Mouaddib, G. Caron, and F. Chaumette. Visual servoing with photometric gaussian mixtures as dense features. *IEEE Transactions on Robotics*, 35(1):49–63, 2019. 1, 2, 4, 6

[9] L.-A. Duflot, R. Reisenhofer, B. Tamadazte, N. Andreff, and A. Krupa. Wavelet and Shearlet-based Image Representations for Visual Servoing. *The Int. J. of Rob. Research*, 38(4):422–450, 2019. 1

[10] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018. 1, 4

[11] N. Krombach, D. Droeschel, S. Houben, and S. Behnke. Feature-based visual odometry prior for real-time semi-dense stereo slam. *Robotics and Autonomous Systems*, 109:38 – 58, 2018. 1

[12] E. Marchand. Direct visual servoing in the frequency domain. *IEEE Rob. and Autom. Letters*, 5(2):620–627, 2020. 1

[13] 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. 5

[14] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu. Relative camera pose estimation using convolutional neural networks. In *Advanced Concepts for Intelligent Vision Systems*, pages 675–687, 2017. 1

[15] H. Mobahi, C. L. Zitnick, and Y. Ma. Seeing through the blur. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1736–1743, 2012. 1

[16] S. A. S. Mohamed, M. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7:97466–97486, 2019. 1

[17] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans.actions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990. 1

[18] 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/RSJ Int. Conf. on Intelligent Robots and Systems*, 2020. 1

[19] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *IEEE Int. Conf. on Computer Vision Workshops (ICCV Workshops)*, pages 719–722, 2011. 1, 4

[20] J. Tardif, Y. Pavlidis, and K. Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2531–2538, 2008. 2

[21] K. Wang, K. Wang, and S. Shen. FlowNorm: A learning-based method for increasing convergence range of direct alignment. In *IEEE Int. Conf. on Robotics and Automation*, pages 2109–2115, 2020. 1