

Integrating Features Acceleration in Visual Predictive Control

Franco Fusco, Olivier Kermorgant, Philippe Martinet

▶ **To cite this version:**

Franco Fusco, Olivier Kermorgant, Philippe Martinet. Integrating Features Acceleration in Visual Predictive Control. IROS 2020 - IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct 2020, Las Vegas / Virtual, United States. hal-02909462

HAL Id: hal-02909462
https://hal.science/hal-02909462

Submitted on 30 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Integrating Features Acceleration in Visual Predictive Control

Franco Fusco[1], Olivier Kermorgant[1], and Philippe Martinet[2]

***Abstract*— This paper proposes new prediction models for Visual Predictive Control that can lead to both better motions in the feature space and shorter sensor trajectories in 3D. Contrarily to existing first-order models based only on the interaction matrix, it is proposed to integrate acceleration information provided by second-order models. This allows to better estimate the evolution of the image features, and consequently to evaluate control inputs that can properly steer the system to a desired configuration. By means of simulations, the performances of these new predictors are shown and compared to those of a classical model. Included experiments using both image point features and polar coordinates confirm the validity and generality of the approach, showing that the increased complexity of the predictors does not prevent real-time implementations.**

I. Introduction

Visual servoing is a well established control strategy allowing robust and precise positioning of a sensor in front of an object. Thanks to the direct sensory feedback, it is robust to modeling and calibration errors. Different setups can be considered [1], depending on whether the sensor is mounted on the end-effector of a robot or observing the system from a fixed location, or on the type of extracted features (image points, lines, reconstructed 3D pose, *etc*.).

Classical schemes generally rely on the use of a proportional controller to regulate the current features to a desired value. Such a control law is simple to implement, but has some drawbacks. The first one is that the motion of the sensor in the Cartesian space can be unsatisfactory when large displacements are involved. A classical example is the retraction that occurs when servoing from image points in presence of large camera rotations about its optical axis. Another issue is related to the time to convergence of these schemes, with the system slowing down when approaching the desired configuration due to the exponential decay imposed by the proportional controller. Finally, to deal with constraints such as visibility, joint limits and collision avoidance, some modifications to the basic approach are needed.

Several works can be found in the literature to address these issues. Since the 3D motion of the sensor depends also on the kind of selected features, some researches focused on finding those that can guarantee better decoupling in the camera motion, such as image moments [2]. Other well-known strategies exploit the so-called Efficient Second order Minimization, which relies on the use of the pseudo-inverse of the mean of the interaction matrix at the current iteration and at equilibrium, or the mean of the pseudo-inverses [3], [4]. To deal with constraints, a hierarchical stack of tasks was considered in [5], while a solution based on weighting matrices was used in [6].

Another approach is the one of Model Predictive Control (MPC). Such strategy is appealing in visual servoing as it provides a unified approach that can deal with most of the problems mentioned above. In fact, by means of a model of the system to be controlled, they can anticipate the states that will be traversed. This allows to select optimal control samples that will quickly guide the system towards a desired configuration, while taking into account physical limitations. Early approaches took low-level velocity controllers into account in order to speed up the convergence [7], while later works [8], [9] introduced different kind of constraints, such as features visibility, joint limits, maximum motor efforts. Finally, the use of the interaction matrix of the features to generate predictions directly in image space was investigated in different works [10], [11], [12]. The main drawback of using predictive strategies is that they come at the cost of an increased computational burden, especially when predictions are generated along a large horizon. Nonetheless, if a good trade-off between computational load and long horizons can be found, the final performances can be highly satisfactory.

With the objective of enhancing the performances of a MPC scheme, two directions are available: on one hand, the focus could be on improving the optimization routine used to obtain the control sequences to be sent to the actuators. On the other hand, attempts to enhance the model used for the predictions could be investigated. Intuitively, given an equal prediction horizon, models that can provide more accurate predictions should lead to better performances. In this paper, we investigate this second route and contribute to the study of Visual Predictive Control (VPC) by proposing two new local models to evaluate the features. More precisely, our proposal is to include acceleration information by considering second-order models [13], [14], which generally allows to better predict what will be the evolution of the features in the sensor space over the prediction horizon. As a direct consequence, this allows the control algorithm to produce better input signals, steering the system along nicer paths in the 3D space. Our work proposes a comparative study between existing predictive strategies from the literature and those relying on our predictors, while for a comparison of VPC against classical approaches the reader is referred to [9], [10].

This work was carried out in the framework of the PROMPT project, a project funded by RFI Atlanstic 2020. Parts of the equipment used here were funded by the project ROBOTEX, reference ANR-10-EQPX-44-01.

[1]Franco Fusco and Olivier Kermorgant are with Centrale Nantes, Laboratoire des Sciences du Numérique de Nantes LS2N, France franco.fusco@ls2n.fr ; olivier.kermorgant@ls2n.fr

[2]Philippe Martinet is with Université Cote d'Azur, Inria Sophia Antipolis, France philippe.martinet@inria.fr

The remainder of this paper is organized as follows: in Section II after recalling first and second-order models in visual servoing, we introduce the MPC formulation exploited by this paper. We present our predictors in Section II-C, altogether with a local model based on the interaction matrix that will be used for performance comparison. Simulations are presented in Section III-A, in which we consider a redundant serial manipulator with a camera mounted on its end-effector. Finally, real experiments are included in Section III-B to support our theoretical work. We consider two kinds of features for the servoing task: normalized image point and polar coordinates. In both cases, it is shown that including acceleration information in the predictive control algorithm enhances the quality of the motion of the manipulator in comparison with local predictive models based solely on the interaction matrix.

II. Visual Predictive Control

A. Visual Servoing Models

The classical model exploited to link the relative motion between the vision system and the observed object to the evolution of the features is based on the interaction matrix $\mathbf{L_s}$ of the feature set $\mathbf{s} \in \mathbb{R}^m$ [15]:

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbf{v} \tag{1}$$

In this equation, the kinematic screw $\mathbf{v} \in \mathbb{R}^6$ gathers the linear and angular velocity of the camera with respect to the object frame, expressed in the coordinate system of the sensor itself. The interaction matrix is a function of $\mathbf{s}$ and possibly of a set of parameters $\mathbf{z} \in \mathbb{R}^p$ (such as depth for image points or other 3D parameters not available in $\mathbf{s}$), but the dependencies are omitted above for brevity.

Second-order models can be considered as well, relating the spatial velocity and acceleration of the sensor to the second derivative of the feature vector. These models can be briefly written in the form:

$$\ddot{\mathbf{s}} = \mathbf{L_s}\mathbf{a} + \mathbf{h_s} \tag{2}$$

with $\mathbf{a}$ representing the relative acceleration of the sensor expressed on the camera frame and $\mathbf{h_s}$ being a function of $\mathbf{s}$, $\mathbf{z}$ and $\mathbf{v}$. In particular, this last component can be written as a collection of quadratic forms [16]:

$$\mathbf{h_s} = \begin{bmatrix} \mathbf{v}^T\mathbf{G}_1\mathbf{v} \\ \cdots \\ \mathbf{v}^T\mathbf{G}_m\mathbf{v} \end{bmatrix} \tag{3}$$

wherein the (symmetric) matrices $\mathbf{G}_i$ are functions of $\mathbf{s}$ and $\mathbf{z}$ only. The analytical expressions of these matrices can be found in [13] for image point features, while those of polar coordinates have been derived in (4a) and (4b).

Assuming a *eye-in-hand* configuration and denoting with $\mathbf{q} \in \mathbb{R}^n$ the joint vector of the robot, the two models presented above can be rewritten in terms of joint velocities and accelerations, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ respectively. In particular, using the geometric Jacobian $\mathbf{J}$ expressed in the base frame of the robot and its derivative, the kinematics of the sensor is described by:

$$\mathbf{v} = \mathbb{T}\,{}^o\mathbf{v} = \mathbb{T}\mathbf{J}\dot{\mathbf{q}} \tag{5a}$$

$$\mathbf{a} = \mathbb{T}\,{}^o\dot{\mathbf{v}} = \mathbb{T}\left(\mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}}\right) \tag{5b}$$

where ${}^o\mathbf{v}$ represents the velocity of the camera expressed in the base frame o of the robot, while $\mathbb{T}$ expresses a change of basis from the base frame of the robot to the one of the camera. It is a block-diagonal matrix in the form $\mathbb{T} = \mathrm{diag}({}^c\mathbf{R}_o, {}^c\mathbf{R}_o)$, ${}^c\mathbf{R}_o$ being the rotation matrix expressing the orientation of the camera frame c with respect to o. It depends on the current joint configuration of the robot, and can be evaluated using the geometric model of the manipulator. Also note that $\mathbf{a}$ is not the derivative of $\mathbf{v}$ (which corresponds to the end-effector spatial velocity projected on the camera frame). It is instead the projection on the sensor frame of the spatial acceleration expressed in the base frame, *i.e.*, $\mathbf{a} = \mathbb{T}\frac{d}{dt}(\mathbf{J}\dot{\mathbf{q}})$. Injecting these expressions into (1) and (2) leads to:

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbb{T}\mathbf{J}\dot{\mathbf{q}} = \mathbf{J_s}\dot{\mathbf{q}} \tag{6a}$$

$$\ddot{\mathbf{s}} = \mathbf{J_s}\ddot{\mathbf{q}} + \boldsymbol{\mu}_\mathbf{s} \quad (\boldsymbol{\mu}_\mathbf{s} \doteq \mathbf{L_s}\mathbb{T}\dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{h_s}) \tag{6b}$$

with $\mathbf{J_s}$ sometimes referred to as *feature Jacobian*.

Using first and second-order models, the positioning task is generally achieved via linearizing feedback control laws, in which the aim is to enforce an exponential decay of the error $\mathbf{e_s} = \mathbf{s} - \mathbf{s}^\star$ to zero, so that the features converge to the desired value $\mathbf{s}^\star$. One problem with such controllers is that the convergence towards the target features slows down when the error decreases, resulting in longer execution times. To achieve faster responses, the control gains can be increased, but only up to a given limit to avoid large noise amplification.

Another issue is related to the 3D motion of the sensor. Features move along almost straight lines in $\mathbb{R}^m$ when using linearizing strategies, corresponding to shortest paths in the feature space. Due to the highly non-linear map between the relative sensor/object pose and the features, such motion can lead to sub-optimal camera trajectories in $\mathrm{SE}(3)$. A well known example is the retraction problem which can be observed when using image coordinates as features and with the object being purely rotated around the optical axis of the camera [17].

To overcome these problems, Predictive Controllers can be exploited. They generally perform faster since they tend to saturate control inputs to the allowed limits, thus reducing the time to convergence. Furthermore, if properly tuned they can enhance the overall quality of the motion by selecting paths that, despite being locally sub-optimal in the feature space, correspond to shorter robot motions [12]. Before entering into the details of Visual Predictive Control, we will recall in the next section the basics of generic nonlinear MPCs.

B. Model Predictive Control

Model Predictive Control is an optimal control technique that aims at determining the best input signal to be applied to a system by taking into account the future evolution

of the state. In our case, the continuous-time dynamics of the system is approximated by a non-linear discrete-time equivalent having the generic state-space form:

$$\mathbf{x}_{k+1} = f\left(\mathbf{x}_k, \mathbf{u}_k\right) \tag{7}$$

$\mathbf{x}_k$ and $\mathbf{u}_k$ being respectively the state and the control samples at the discrete step k. A common formulation for the MPC problem is a finite-horizon open-loop optimization that takes into account the model above and a set of constraints. In this optimization, the decision variables correspond to the control sequence $\underline{\underline{\mathbf{u}}} = \{\mathbf{u}_\tau, \mathbf{u}_{\tau+1}, \cdots, \mathbf{u}_{\tau+n_c-1}\}$, τ denoting the current time sample and n_c being the number of control samples, or *control horizon*, considered for the optimization. Given the sequence $\underline{\underline{\mathbf{u}}}$, $n_p \geq n_c$ "future states" visited by the system are evaluated using (7), n_p being the *prediction horizon*. Note that the first n_c states are produced starting from $\mathbf{x}_\tau$ and applying the control samples in $\underline{\underline{\mathbf{u}}}$, while the remaining ones are obtained by repeatedly using $\mathbf{u}_{\tau+n_c-1}$ as input until a total of n_p predictions are obtained.

In order to evaluate the optimal control sequence, the following problem has to be solved at the discrete time-sample τ:

$$\min_{\underline{\underline{\mathbf{u}}}} \sum_{k=1}^{n_p} \mathbf{e}_{\tau+k}^T \mathbf{A}_k \mathbf{e}_{\tau+k} + \sum_{k=0}^{n_c-1} \mathbf{u}_{\tau+k}^T \mathbf{B}_k \mathbf{u}_{\tau+k} \tag{8}$$

subject to the constraints:

$$\mathbf{u}_{\tau+k} \in \mathcal{U} \subseteq \mathbb{R}^{n_u} \quad k = 0, \cdots, n_c - 1 \tag{9a}$$

$$\mathbf{x}_{\tau+k} \in \mathcal{X} \subseteq \mathbb{R}^{n_x} \quad k = 1, \cdots, n_p \tag{9b}$$

The objective function is a quadratic stage cost that tries to push the error $\mathbf{e}_i = \mathbf{x}_i - \mathbf{x}_i^\star$ to zero, $\mathbf{x}_i^\star$ being the desired value of the state at step i. At the same time, the second sum in (8) tries to minimize the norm of the control input. The terms $\mathbf{A}_k$ and $\mathbf{B}_k$ are (semi)positive definite weighting matrices that control the relative importance of the different components in the objective.

The constraints (9a) generally correspond to a series of inequalities that limit the control input within given bounds:

$$-\mathbf{u}_{lim} \leq \mathbf{u}_{\tau+k} \leq \mathbf{u}_{lim} \quad k = 0, \cdots, n_c - 1 \tag{10}$$

Similarly, the set of constraints (9b) ensures that predicted states remain inside the valid region $\mathcal{X}$. They can represent, *e.g.*, visibility constraints or joint limits.

In principle, it would be possible to select very long prediction and control horizons, solve the optimization once, and send the optimal control inputs one after the other to move the robot. However, this solution is not feasible for different reasons, the main one being that the model used to predict future states is usually just an approximation of the real evolution. It is thus necessary to repeat the optimization at each control iteration, in order to reduce the impact of modeling and identification errors. Having to repeat as many times as possible the optimization, it is thus desirable to also find a trade-off between long predictions and short computation times.

C. Visual Predictors using Acceleration

We propose in this section two new prediction models that take into account the acceleration of the features in the sensor space. As a starting point, we consider in the following a formulation based on the interaction matrix evaluated at each prediction sample, similarly to what has been proposed in [10] and [12] (LM_c predictor). In our case, we also consider the joint configuration as part of the state vector, and the joint velocity of the robot to be the control input of the system. This allows the controller to internally handle the redundancy and also to explicitly consider joint limits in the optimization. We also assume that the set of parameters $\mathbf{z}$ appearing in $\mathbf{L_s}$ can be described by models which are similar to (1) and (2), *i.e.*, such that $\dot{\mathbf{z}} = \mathbf{L_z}\mathbf{v}$ ($\mathbf{L_z}$ depending on $\mathbf{s}$ and/or $\mathbf{z}$). These parameters are predicted at each iteration as well, and therefore the state vector $\mathbf{x}$ corresponds to the concatenation of $\mathbf{q}$, $\mathbf{s}$ and $\mathbf{z}$. Given these choices, a velocity-based local prediction model is written as:

$$\begin{bmatrix} \mathbf{q}_{k+1} \\ \mathbf{s}_{k+1} \\ \mathbf{z}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_k \\ \mathbf{s}_k \\ \mathbf{z}_k \end{bmatrix} + \Delta t \begin{bmatrix} \mathbf{I} \\ \mathbf{J}_{\mathbf{s},k} \\ \mathbf{J}_{\mathbf{z},k} \end{bmatrix} \mathbf{u}_k \tag{11}$$

with $\mathbf{J_z} = \mathbf{L_z}\mathbb{T}\mathbf{J}$ and Δt representing the discretization period for the model. This model, namely $\mathcal{M}_1$, will be used later as a reference for comparison with our custom predictors.

$$\mathbf{G}_\rho = \begin{bmatrix} \frac{\sin^2\theta}{Z^2\rho} & -\frac{\sin 2\theta}{2Z^2\rho} & -\frac{\cos\theta}{Z^2} & -\frac{\left(\rho^2-1\right)\sin 2\theta}{2Z\rho} & \frac{-\rho^2\sin^2\theta+2\rho^2+\sin^2\theta}{Z\rho} & 0 \\ -\frac{\sin 2\theta}{2Z^2\rho} & \frac{\cos^2\theta}{Z^2\rho} & -\frac{\sin\theta}{Z^2} & \frac{\rho^2\cos^2\theta-2\rho^2-\cos^2\theta}{Z\rho} & \frac{\left(\rho^2-1\right)\sin 2\theta}{2Z\rho} & 0 \\ -\frac{\cos\theta}{Z^2} & -\frac{\sin\theta}{Z^2} & \frac{2\rho}{Z^2} & \frac{2\rho^2\sin\theta}{Z} & -\frac{2\rho^2\cos\theta}{Z} & 0 \\ -\frac{\left(\rho^2-1\right)\sin 2\theta}{2Z\rho} & \frac{\rho^2\cos^2\theta-2\rho^2-\cos^2\theta}{Z\rho} & \frac{2\rho^2\sin\theta}{Z} & \frac{\left(\rho^2+1\right)\left(2\rho^2\sin^2\theta+\cos^2\theta\right)}{\rho} & -\frac{\left(2\rho^4+\rho^2-1\right)\sin 2\theta}{2\rho} & -\frac{\left(\rho^2+1\right)\cos\theta}{2} \\ \frac{-\rho^2\sin^2\theta+2\rho^2+\sin^2\theta}{Z\rho} & \frac{\left(\rho^2-1\right)\sin 2\theta}{2Z\rho} & -\frac{2\rho^2\cos\theta}{Z} & -\frac{\left(2\rho^4+\rho^2-1\right)\sin 2\theta}{2\rho} & \frac{\left(\rho^2+1\right)\left(2\rho^2\cos^2\theta+\sin^2\theta\right)}{\rho} & -\frac{\left(\rho^2+1\right)\sin\theta}{2} \\ 0 & 0 & 0 & -\frac{\left(\rho^2+1\right)\cos\theta}{2} & -\frac{\left(\rho^2+1\right)\sin\theta}{2} & 0 \end{bmatrix} \tag{4a}$$

$$\mathbf{G}_\theta = \begin{bmatrix} \frac{\sin 2\theta}{Z^2\rho^2} & -\frac{\cos 2\theta}{Z^2\rho^2} & 0 & \frac{\cos 2\theta}{Z\rho^2} & \frac{\sin 2\theta}{Z\rho^2} & 0 \\ -\frac{\cos 2\theta}{Z^2\rho^2} & -\frac{\sin 2\theta}{Z^2\rho^2} & 0 & \frac{\sin 2\theta}{Z\rho^2} & -\frac{\cos 2\theta}{Z\rho^2} & 0 \\ 0 & 0 & 0 & -\frac{\cos\theta}{Z\rho} & -\frac{\sin\theta}{Z\rho} & 0 \\ \frac{\cos 2\theta}{Z\rho^2} & \frac{\sin 2\theta}{Z\rho^2} & -\frac{\cos\theta}{Z\rho} & -\frac{\left(\rho^2+2\right)\sin 2\theta}{2\rho^2} & \frac{\left(\rho^2+2\right)\cos 2\theta}{2\rho^2} & \frac{\sin\theta}{2\rho} \\ \frac{\sin 2\theta}{Z\rho^2} & -\frac{\cos 2\theta}{Z\rho^2} & -\frac{\sin\theta}{Z\rho} & \frac{\left(\rho^2+2\right)\cos 2\theta}{2\rho^2} & \frac{\left(\rho^2+2\right)\sin 2\theta}{2\rho^2} & -\frac{\cos\theta}{2\rho} \\ 0 & 0 & 0 & \frac{\sin\theta}{2\rho} & -\frac{\cos\theta}{2\rho} & 0 \end{bmatrix} \tag{4b}$$

To the best of our knowledge, prediction models used so far in visual control all exploited only first-order approximations based on (1). Our first proposal is a new model, denoted as $\mathcal{M}_2$, that considers a second-order Taylor series expansion for the state vector, *i.e.*, $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t\dot{\mathbf{x}}_k + \frac{1}{2}\Delta t^2\ddot{\mathbf{x}}_k$. In this case, the control input $\mathbf{u}$ is no longer the joint velocity of the manipulator, but its acceleration. Clearly, to properly update the state over several steps it is also necessary to predict the value of all first derivatives of $\mathbf{q}$, $\mathbf{s}$ and $\mathbf{z}$. Regarding the joint velocity, we can simply do that by numerical integration of the joint acceleration. This implies that $\dot{\mathbf{q}}$ will be added to the predicted state as well. Regarding the features and the related parameters, we instead propose to estimate their first derivative using their jacobians according to (6a), thus avoiding the addition of further elements in the state vector. As a result, the prediction model can be written as:

$$\begin{bmatrix}\mathbf{q}_{k+1}\\ \dot{\mathbf{q}}_{k+1}\\ \mathbf{s}_{k+1}\\ \mathbf{z}_{k+1}\end{bmatrix} = \begin{bmatrix}\mathbf{q}_k + \Delta t\dot{\mathbf{q}}_k\\ \dot{\mathbf{q}}_k\\ \mathbf{s}_k + \Delta t\mathbf{J}_{\mathbf{s},k}\dot{\mathbf{q}}_k + \frac{\Delta t^2}{2}\boldsymbol{\mu}_{\mathbf{s},k}\\ \mathbf{z}_k + \Delta t\mathbf{J}_{\mathbf{z},k}\dot{\mathbf{q}}_k + \frac{\Delta t^2}{2}\boldsymbol{\mu}_{\mathbf{z},k}\end{bmatrix} + \begin{bmatrix}\frac{\Delta t^2}{2}\mathbf{I}\\ \Delta t\mathbf{I}\\ \frac{\Delta t^2}{2}\mathbf{J}_{\mathbf{s},k}\\ \frac{\Delta t^2}{2}\mathbf{J}_{\mathbf{z},k}\end{bmatrix}\mathbf{u}_k \tag{12}$$

Our second proposal lies between classical approaches based solely on velocity information and the acceleration-controlled model presented above. Like in the first case, it assumes the control input $\mathbf{u}$ to be the velocity of the manipulator. However, it also exploits the fact that the second term in (2) depends only on the current features/parameters and the velocity of the sensor. As it is done in classical local models, we assume that the twist of the camera is piece-wise constant, and at the time-sample k it is thus possible to evaluate $\mathbf{v}_k$ as $\mathbb{T}\mathbf{J}_k\mathbf{u}_k$. Altogether with $\mathbf{s}_k$ and $\mathbf{z}_k$, this allows to approximate features acceleration as $\ddot{\mathbf{s}}_k \simeq \mathbf{h}_{\mathbf{s},k}$ (the acceleration of the sensor is therefore not taken into account). Using a similar procedure to evaluate $\ddot{\mathbf{z}}$, this provides the following update scheme:

$$\begin{bmatrix}\mathbf{q}_{k+1}\\ \mathbf{s}_{k+1}\\ \mathbf{z}_{k+1}\end{bmatrix} = \begin{bmatrix}\mathbf{q}_k\\ \mathbf{s}_k\\ \mathbf{z}_k\end{bmatrix} + \Delta t\begin{bmatrix}\mathbf{I}\\ \mathbf{J}_{\mathbf{s},k}\\ \mathbf{J}_{\mathbf{z},k}\end{bmatrix}\mathbf{u}_k + \begin{bmatrix}\mathbf{0}\\ \frac{\Delta t^2}{2}\mathbf{h}_{\mathbf{s},k}\\ \frac{\Delta t^2}{2}\mathbf{h}_{\mathbf{z},k}\end{bmatrix} \tag{13}$$

Since this model is a mix between those based on first and second order models, we like to refer to it as *hybrid* predictor, $\mathcal{M}_\mathcal{H}$. One advantage of this update scheme with respect to $\mathcal{M}_2$ is that it allows to integrate higher-order information while still keeping the same size for the state vector and introducing less elements into the model, reducing the computational burden for the optimization.

To illustrate why the proposed models can lead to better results, Fig. 1 shows a simple example of predictions made using the three schemes above. This case considers a free-flying camera that observes four image points while retracting and rotating about its optical axis. The velocity and acceleration of the sensor, which are both continuous and non-constant, are sampled with a period of $50\,\mathrm{ms}$, and fed to each model to perform predictions for $\mathbf{s}$ and $\mathbf{z}$ in open-loop. $\mathcal{M}_1$ slightly drifts from the actual state during time, whereas $\mathcal{M}_2$ provides the most reliable predictions. Finally, the hybrid predictor $\mathcal{M}_\mathcal{H}$, despite being less accurate than (12) due to the truncated approximation of $\dddot{\mathbf{s}}$, provides a rather good approximation of the real evolution.

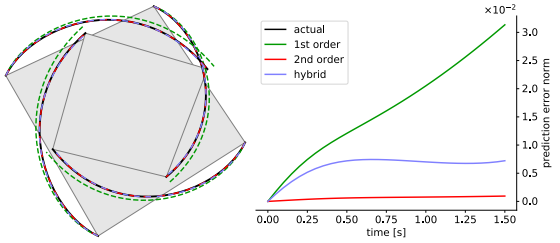


Fig. 1. On the left: motion of four image points (in black) altogether with predictions obtained using $\mathcal{M}_1$ (green), $\mathcal{M}_2$ (red) and $\mathcal{M}_\mathcal{H}$ (blue). On the right: norm of the prediction error during time, for each strategy.

D. Visual Predictive Control Scheme

Our VPC scheme exploits the formulation reported in Section II-B, with some adaptations specific to our studied case. First of all, the objective of the controller is to steer only the features from the initial configuration to a fixed value $\mathbf{s}^\star$. There is instead no need to ensure the convergence of neither $\mathbf{q}$ nor $\mathbf{z}$ to any specific value. For this reason, the matrices $\mathbf{A}_k$ are selected as diagonal, with the elements corresponding to the joint configuration and to the features parameters equal to zero.

Regarding the weight associated to the features, we adopt the diagonal weighting matrix $\mathbf{Q}_k = \alpha^k\mathbf{I}$, α being a tunable parameter, similarly to what was done in related works. In particular, in [18], [19] it was proposed to use $\alpha = 1/e$, corresponding to a decreasing weight of features samples. This was justified by the fact that in these works, the authors were also integrating an on-line trajectory generator, and the decreasing factor ensured better tracking of the reference. An increasing weighting policy was instead considered in [10], where a factor $\alpha = 2$ was used. They showed that this can bear to higher decoupling in the camera motion, since the optimization algorithm is "encouraged" to look for solutions that are possibly sub-optimal in the short term, but that ensure shorter overall motions and faster convergence.

The secondary objective of our VPC scheme is to evaluate a motion that minimizes the velocity of the manipulator. This is justified mainly by the fact that we are interested in controlling a redundant robot, for which the visual task is not sufficient to fully stabilize the joint position. By minimizing the velocity, once the robot has completed the visual task the optimal solution becomes to stop the robot, thus solving the problem. To do that, we use the (constant) diagonal matrix $\mathbf{R}_k = \beta\mathbf{I}$, $\beta > 0$ controlling the relative importance with respect to the visual task. Note that the objective is to minimize the velocity independently from the model used for predictions. Since $\mathcal{M}_1$ and $\mathcal{M}_\mathcal{H}$ assume a velocity control, we have $\mathbf{B}_k = \mathbf{R}_k$ and $\mathbf{A}_k = \mathrm{diag}\,(\mathbf{0}_n, \mathbf{Q}_k, \mathbf{0}_p)$. Instead, as the velocity is part of the state vector in the case of $\mathcal{M}_2$, the weighting matrix is inserted as part of $\mathbf{A}_k$, *i.e.*, $\mathbf{A}_k = \mathrm{diag}\,(\mathbf{0}_n, \mathbf{R}_k, \mathbf{Q}_k, \mathbf{0}_p)$, with the desired state $\dot{\mathbf{q}}_k^\star$ simply being a zero-vector. In this case, the minimization

of the control input (the acceleration) is not included as an objective ($\mathbf{B}_k = \mathbf{0}_n$). Indeed, minimizing the acceleration would lead to a conservative velocity behavior, resulting in potential instability or longer time to convergence.

Finally, two kind of state constraints are considered during the optimization, in addition to control input bounds. The first one is introduced to keep the joint configuration of the robot within the allowed limits, *i.e.*, $\mathbf{q}_{min} \leq \mathbf{q}_{\tau+k} \leq \mathbf{q}_{max}$, $\forall k = 1, \cdots, n_p$. The second set of state constraints is introduced to prevent the object to leave the field of view of the camera. When servoing from image points, the constraint simply writes as $\mathbf{s}_{min} \leq \mathbf{s}_{\tau+k} \leq \mathbf{s}_{max}$, with $\mathbf{s}$ containing the x and y coordinates of the observed points. When polar coordinates are used, the constraints above are expressed for each point i in terms of the features ρ_i and θ_i as:

$$\begin{aligned} x_{min} \leq \rho_{i,k} \cos\theta_{i,k} \leq x_{max} \\ y_{min} \leq \rho_{i,k} \sin\theta_{i,k} \leq y_{max} \end{aligned} \tag{14}$$

III. Simulations and Experimental Results

To investigate the performances of the proposed models, we conducted several simulations and experiments. Our setup involves a 7-dof Kuka LWR4+ arm, having a camera mounted on top of its end-effector, as shown in Fig. 2. The sensor observes a planar object with four black circles. Depending on the experiment, either the image coordinates of these points or their polar representation are used as features.

Several libraries have been used to implement the VPC scheme. Pinocchio [20] has been used for the geometric model of the robot, while the optimization algorithm SLSQP [21], [22] from NLopt [23] was chosen to evaluate the control sequence at each iteration. We also exploited the software package CppADCodeGen [24] in order to generate efficient code for the prediction models and to speedup computations.

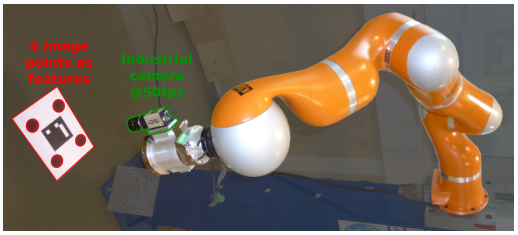


Fig. 2. The Kuka arm used in our experiments.

We detail in the next section two sets of simulations and discuss few aspects related to parameters tuning. Afterwards, we report results from real experiments which support the feasibility and effectiveness of our approach. Note that no comparison is performed between predictive approaches and classical controls based on feedback linearization, which can be found already in other works, *e.g.*, [9], [10]. Instead, our analysis focuses on the comparison between existing local predictors based on the interaction matrix and our proposed models.

A. Simulations

Simulations were run in a simple environment that updates the state of the robot by integrating velocity/acceleration commands, therefore not taking into account the dynamic model of the manipulator. To simulate the visual feedback gathered by a camera, the geometric model of the Kuka arm was used, since all transformations are perfectly known in simulation. Features are sent at a rate of $50\,\mathrm{Hz}$, to match the one used in real experiments, and the control period Δt_c is thus set to $20\,\mathrm{ms}$.

Depending on the difficulty of the problem, the optimization might require more time than the control period. We thus set, in NLopt, a time limit $0.9 \cdot \Delta t_c = 18\,\mathrm{ms}$ as termination condition in addition to other stopping criteria.

We compared the three models considering various setups, and we report here two set of tests. In all shown cases, the selected features are the image coordinates of four points. In the first set, the camera has to perform a rotation of $180°$ about its optical axis, and a short translation. In the second case, the motion requires to rotate also about the X and Y axes of the camera and a larger translation is required as well. These cases are known to be particularly challenging for standard feedback control strategies, as they cause large retraction of the sensor.

In the two tests, both the control and prediction horizon are set to $n_c = n_p = 10$. In addition, the discretization time used in the predictors, Δt, is chosen as $\Delta t = 5\Delta t_c = 0.1\,\mathrm{s}$. This allows the optimization to consider future states that are sufficiently distant in the future, while keeping the dimension of the prediction vector small enough to allow a real time implementation. It must be noted, however, that the optimization halts due to the time limit criterion for all models. Nonetheless, we checked that a premature ending of the optimization does not significantly alter the performances. In particular, even when the optimization is allowed to run for as long as $200\,\mathrm{ms}$, the features error norm reduces only by a small amount, while acceptable results can still be obtained even if the limit is reduced up to $13\,\mathrm{ms}$.

Other parameters were tuned by trial and error, and kept constant across simulations. In particular, the value of α was chosen as 1.1, in order to disfavor "greedy" motions that try to move the features straight towards the goal configuration, as discussed in Section II-D. We noticed that this is a suitable choice especially when the camera has to perform large rotations, like in reported simulations. The parameter β was set to a small value compared to the one of α to allow motions to be rather fast in the beginning, achieving quicker converge to the desired feature configuration. We noticed that depending on the kind of control input, the value needs to be adjusted differently. In particular, $\beta = 10^{-3}$ was used for velocity-controlled models ($\mathcal{M}_1$ and $\mathcal{M}_\mathcal{H}$). Regarding $\mathcal{M}_2$, a smaller factor proved to work better in general, and thus β was set to 10^{-4} in that case. To obtain these values, we followed a simple procedure, firstly fixing α to 1 and focusing on testing different values for β. We noticed that it was sufficient to identify a suitable order of magnitude for it, while precisely adjusting the value bears no major differences in the performances. Afterwards, we considered the weight associated to the last prediction sample, $\alpha' \doteq \alpha^{n_p} = \alpha^{10}$. By considering few different values for it, we could coarsely

tune α. Our final choice was $\alpha' = 2.5$, leading to $\alpha \simeq 1.1$ as mentioned above.

1) First Simulation Test: In this first set of tests the camera has to rotate of $180°$ around its optical axis, while also performing a short translation (less than $7\,\mathrm{cm}$) in order to position in front of the four points.

Results are shown in Fig. 3. Each row corresponds to a specific prediction model, and the columns report respectively the features path in the image plane, the feature error and a 3D view of the robot. In the first column, starting points are represented with a circle, while a square is used for the desired configuration. Green lines in the 3D view help to visualize the motion of the sensor.

As visible in middle column of Fig. 3, all strategies are able to quickly converge to the desired configuration: within $1\,\mathrm{s}$ when using $\mathcal{M}_1$ and $\mathcal{M}_\mathcal{H}$, and around $1.5\,\mathrm{s}$ with $\mathcal{M}_2$. Nonetheless, the performances of $\mathcal{M}_1$ are otherwise worse, showing some sudden changes of direction in the feature space but more importantly retraction in 3D. We think that this can be justified by the higher drifts associated to a first-order linearization and consequently to less reliable predictions. On the other hand, $\mathcal{M}_2$, despite converging less quickly than the other models, presents smooth motions and reduced retraction. $\mathcal{M}_\mathcal{H}$ also features nice evolution in image space, fast convergence and very little retraction, but presents few sudden changes of direction as visible in the 3D view.

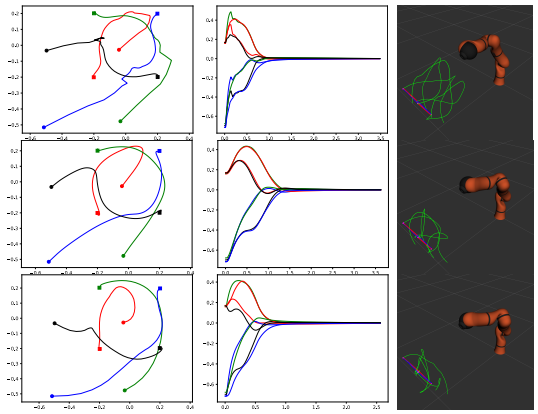


Fig. 3. First simulation test. The rows correspond to $\mathcal{M}_1$ (top), $\mathcal{M}_2$ (middle) and $\mathcal{M}_\mathcal{H}$ (bottom), while the columns to the features trajectories in the image (left), the feature error (center) and a 3D view of the manipulator (right).

2) Second Simulation Test: In this second set of simulations, a case presenting larger initial feature errors is shown. With respect to the previous case, the sensor needs to perform a larger translation ($20\,\mathrm{cm}$) and now features rotations of $15°$ and $30°$ around the X and Y axes respectively in addition to a rotation around Z of $165°$.

Fig. 4 shows the obtained results in terms of features evolution and 3D motion of the robot. Compared to $\mathcal{M}_1$, the proposed approaches lead again to better performances, with the features following rather smooth paths in the image. It is also interesting to notice that these controls seem to steer the features firstly along a translational path, subsequently performing a mainly rotational motion. Considering the 3D views, it can be seen that $\mathcal{M}_1$ performs a rather long motion, leading to a slightly higher time to convergence. $\mathcal{M}_2$ provides instead nice results, with regular motions and reduced retraction. Finally, $\mathcal{M}_\mathcal{H}$, despite presenting few sudden changes in its motion like in the previous test, features an overall good behavior similar to that of $\mathcal{M}_2$.

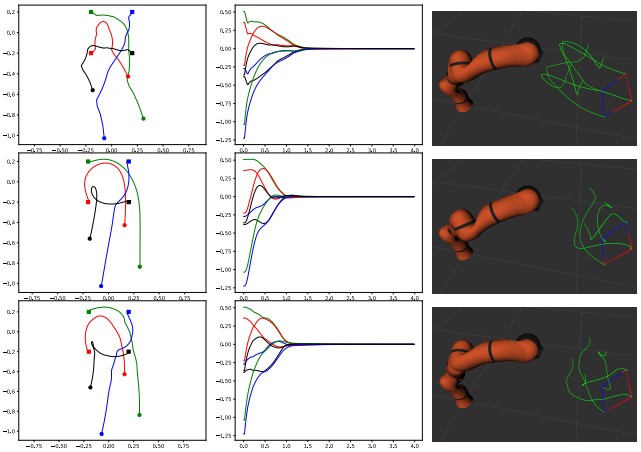


Fig. 4. Second simulation test. The rows correspond to $\mathcal{M}_1$ (top), $\mathcal{M}_2$ (middle) and $\mathcal{M}_\mathcal{H}$ (bottom), while the columns to the features trajectories in the image (left), the feature error (center) and a 3D view of the manipulator (right).

B. Experimental Tests

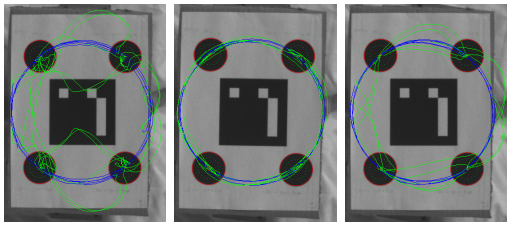
We now present the experimental results related to the comparison of the different models. Since our hardware does not yet allow to directly control the robot in acceleration, in order to test $\mathcal{M}_2$ we numerically integrated the acceleration signal produced by the VPC and sent the resulting signal to the low-level velocity controller running on our robot.

To support the effectiveness and generality of our approach, we tested our controllers with different features: we firstly considered standard image point coordinates (as in the simulations included above) and then the polar coordinates of the centers of the four circles printed on the object. In both cases, our predictors outperform classical local models, confirming the validity of our work.

1) Servoing from Image Points: Regarding parameters tuning, experiments were run with the same values as in the two sets of presented simulations, except for the value of β. In fact, it was noticed that a larger value better suits real experiments in which noise is also present. In practice, we changed its value from 10^{-3} to 10^{-2} for the two velocity-controlled schemes. In the case of $\mathcal{M}_2$, it was necessary to increase it slightly more in order to reduce oscillations and prevent the velocity to grow excessively. We decided to use in this case $\beta = 5 \cdot 10^{-3}$, which, although not sufficient to completely remove oscillations, was found to be a good trade-off in order to maintain acceptable time to convergence.

Tests were performed by placing the planar object in front of the robot and rotating it multiple times about its normal axis, of an angle of almost $90°$(see the video accompanying this paper). Note that the desired configuration $\mathbf{s}^\star$ was still

kept to a constant value, and therefore the rotations were assumed as pulses perturbing the equilibrium of the system.



(a) $\mathcal{M}_1$ (b) $\mathcal{M}_2$ (c) $\mathcal{M}_{\mathcal{H}}$

Fig. 5. Features trajectory in the image using respectively $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_{\mathcal{H}}$. Blue segments correspond to features displacements induced by the object rotation, while green ones are due to the control moving the camera.

The trajectories followed by the features in the image can be seen in Fig. 5. To help better interpret the results, we used two different colors to distinguish between paths crossed by the features mainly due to the motion of the object (in blue) and those caused by the controller moving the camera (green). The former are nearly perfect circles, since the controller does not react fast enough and the robot remains still for almost the entire time when the object moves. Considering the motion induced by the controller, the behavior is quite different depending on the involved predictor. $\mathcal{M}_1$ is not able to perform a nice rotation, with the features going either too near or too far from the ideal circumference. This also results in an undesirable motion, as it can be seen in the attached video. On the contrary, the trajectory obtained using $\mathcal{M}_2$ is much closer to a pure rotation about the optical center of the camera, and the motion in 3D is satisfactory as well. Finally, $\mathcal{M}_{\mathcal{H}}$ gives intermediate results, with features path that, despite not being as nice as in the case of the second-order predictor, are not deviating too much from a pure rotation. Furthermore, the motion of the sensor features almost no retraction.

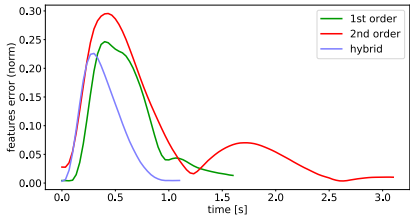
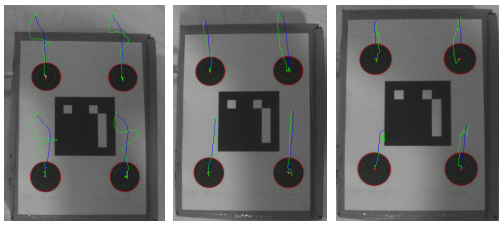


Fig. 6. Features error norm during the first rotation of the object.

The evolution of the feature error during time is reported for the first rotation of the object in Fig. 6. It can be seen that the first-order predictor is outperformed by $\mathcal{M}_{\mathcal{H}}$, taking almost half a second longer to reach the same error magnitude. It should also be noted that $\mathcal{M}_2$, besides being the slowest, presents a rather large overshoot. This is hard to observe from Fig. 5(b), but is clearly visible in the video attachment. Such oscillatory behavior had been observed sometimes in simulation as well, but after proper tuning of the parameters it had been removed. On the other hand, as mentioned in the beginning of this section, it was hard to completely remove them in the real implementation without further sacrificing performances. We believe that one explanation for these oscillations is to be found in the use of the integrator to obtain the velocity command from the acceleration. As neither this block nor the low-level velocity controller are taken into account in the prediction, model uncertainties become larger and predictions are thus less reliable, finally leading to lower performances in practice.

2) Servoing from Polar Coordinates: In this last set of experiments, we tested the performances of the predictors when polar coordinates of the point centers are used as features. Parameters have been kept as in previous experiments, with the exception for the weight associated to angles errors. In fact, the angle errors e_{θ_i} (which are expressed in radians) tend to vary more than the radii ρ_i for the same displacement. For this reason, their weight in the objective was reduced of a factor of 5, *i.e.*, $\mathbf{Q}_k = \alpha_k \text{diag}(1, 1/5, 1, 1/5, 1, 1/5, 1, 1/5)$.

It is well-known that, in classical servoing, polar coordinates perform well when the object is subject to a pure rotation around the optical axis of the camera. Hence, we focus on a case in which the object performs a relatively large translational motion. This is known to lead to less satisfactory trajectories for the system due to the non-linearities in the interaction matrix. As it can be seen in Fig. 7, the results obtained in this case with the three predictors detailed in this paper perform, in comparison, similarly to the previous case: $\mathcal{M}_1$ features the worst results, both in terms of points trajectories and in 3D – as shown in the video attachment. On the other hand, both $\mathcal{M}_2$ and $\mathcal{M}_{\mathcal{H}}$ perform better, with more satisfactory image and 3D trajectories.



(a) $\mathcal{M}_1$ (b) $\mathcal{M}_2$ (c) $\mathcal{M}_{\mathcal{H}}$

Fig. 7. Trajectory of the point centers in the image using respectively $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_{\mathcal{H}}$ when servoing from polar coordinates. Blue segments correspond to displacements induced by the object translation, while green ones are due to the control moving the camera.

The evolution of the features errors is finally given in Fig. 8. It can be seen that all models present a less smooth evolution compared to the case of servoing from image coordinates. We believe that this is mainly due to the optimization being harder to accomplish due to the rather high non-linearities corresponding to polar coordinates. Nonetheless, the benefits coming from considering the acceleration in the predictors are still evident, with $\mathcal{M}_2$ and $\mathcal{M}_{\mathcal{H}}$ being characterized by lower maximal errors with respect to $\mathcal{M}_1$.

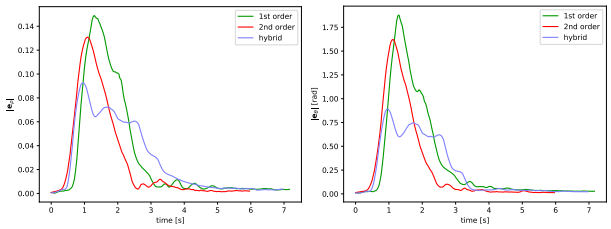


Fig. 8. Features error norm during the motion using polar coordinates. On the left, the error associated to ρ. On the right, the error associated to θ.

IV. Conclusions

We presented in this article new models for Visual Predictive Control that lead to better robot motions, thanks to the inclusion of features acceleration in the feature space. The performances of our models were compared against another local predictor based solely on the first-order interaction matrix. By means of simulations, it was shown that convergence can be achieved in a short time while avoiding undesirable behaviors even in presence of large displacements. The results were validated also thanks to real experiments with an industrial redundant robot, showing the benefits of integrating acceleration information into the predictors. In addition, by considering different features to describe the observed object, we demonstrated that the approach is general and that its effectiveness is not bound to a specific parameterization of the observed object.

We are willing to test the proposed approach using image moments as features. This case is more complex due to the structure of the interaction matrix, which depends also on higher-order moments, theoretically leading to a parameter vector $\mathbf{z}$ of infinite dimension. In addition, as we mentioned in the last section, the controllers are not able to react immediately to object motions. In order to achieve high speed performances in dynamic environments, we believe that a key improvement will be to actively estimate the kinematics of the observed object [25], [26], and to take it into account explicitly inside the prediction models. Finally, a further development might consider time-varying features references, rather than fixed configurations as it was done in simulations and experiments. This will likely require to adapt the formulation of the optimization problem, since the objective (8) currently contains a contribution that tries to minimize the joint velocity of the manipulator, which is in contrast with a task that requires the manipulator to continuously move.

References

[1] F. Chaumette and S. Hutchinson, "Visual servo control. II. Advanced approaches," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 109–118, mar 2007.

[2] O. Tahri and F. Chaumette, "Point-based and region-based image moments for visual servoing of planar objects," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1116–1127, dec 2005.

[3] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *IEEE International Conference on Robotics and Automation. Proceedings.*, 2004, pp. 1843–1848 Vol.2.

[4] O. Tahri and Y. Mezouar, "On visual servoing based on efficient second order minimization," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 712–719, may 2010.

[5] N. Mansard and F. Chaumette, "Task sequencing for high-level sensor-based control," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 60–72, feb 2007.

[6] O. Kermorgant and F. Chaumette, "Dealing with constraints in sensor-based robot control," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 244–257, feb 2014.

[7] J. A. Gangloff and M. F. De Mathelin, "High-speed visual servoing of a 6-d.o.f. manipulator using multivariable predictive control," *Advanced Robotics*, vol. 17, no. 10, pp. 993–1021, jan 2003.

[8] M. Sauvee, P. Poignet, E. Dombre, and E. Courtial, "Image Based Visual Servoing through Nonlinear Model Predictive Control," in *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego (CA), 2006, pp. 1776–1781.

[9] G. Allibert, E. Courtial, and Y. Touré, "Visual predictive control for manipulators with catadioptric camera," in *IEEE International Conference on Robotics and Automation*, Pasadena (CA), may 2008, pp. 510–515.

[10] G. Allibert and E. Courtial, "What can prediction bring to image-based visual servoing?" in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis (USA), oct 2009, pp. 5210–5215.

[11] C. Lazar, A. Burlacu, and C. Copot, "Predictive control architecture for visual servoing of robot manipulators," in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 44, no. 1 PART 1. Elsevier, jan 2011, pp. 9464–9469.

[12] G. Allibert, E. Courtial, and F. Chaumette, "Predictive control for constrained image-based visual servoing," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 933–939, oct 2010.

[13] M. Keshmiri, W. F. Xie, and A. Mohebbi, "Augmented image-based visual servoing of a manipulator using acceleration command," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 10, pp. 5444–5452, oct 2014.

[14] S. Vandernotte, A. Chriette, P. Martinet, and A. S. Roos, "Dynamic sensor-based control," in *14th International Conference on Control, Automation, Robotics and Vision*, Phuket (TH), nov 2016, pp. 1–6.

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

[16] F. Fusco, O. Kermorgant, and P. Martinet, "A Comparison of Visual Servoing from Features Velocity and Acceleration Interaction Models," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Macau, nov 2019, pp. 4447–4452.

[17] F. Chaumette, "Potential problems of stability and convergence in image-based and position-based visual servoing," in *The confluence of vision and control*. Springer, London, 1998, pp. 66–78.

[18] C. Copot, A. Burlacu, and C. Lazar, "Visual predictive control architecture based on image moments for manipulator robots," in *IEEE International Symposium on Industrial Electronics*, jun 2011, pp. 963–968.

[19] A. Burlacu, C. Copot, and C. Lazar, "Predictive control architecture for real-time image moments based servoing of robot manipulators," in *Journal of Intelligent Manufacturing*, vol. 25, no. 5. Elsevier, may 2014, pp. 1125–1134.

[20] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C ++ library A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, Paris, jan 2019.

[21] D. Kraft, "A software package for sequential quadratic programming," *DFVLR-FB 88-28*, 1988.

[22] ——, "Algorithm 733: TOMP–Fortran modules for optimal control calculations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 262–281, 1994.

[23] S. G. Johnson, "The NLopt nonlinear-optimization package," 2019. [Online]. Available: http://github.com/stevengj/nlopt

[24] J. R. Leal, "CppADCodeGen: C++ Algorithmic Differentiation with Source Code Generation," 2019. [Online]. Available: https://github.com/joaoleal/CppADCodeGen

[25] O. Ait-Aider, N. Andreff, J. M. Lavest, and P. Martinet, "Simultaneous object pose and velocity computation using a single view from a rolling shutter camera," in *Lecture Notes in Computer Science*, vol. 3952 LNCS. Springer, Berlin, Heidelberg, 2006, pp. 56–68.

[26] R. Dahmouche, N. Andreff, Y. Mezouar, O. Ait-Aider, and P. Martinet, "Dynamic visual servoing from sequential regions of interest acquisition," *International Journal of Robotics Research*, vol. 31, no. 4, pp. 520–537, apr 2012.