Visual predictive control for differential drive robots with parallel implementation on GPU

A. Durand-Petiteville, Viviane Cadenat

Visual Predictive Control for Differential Drive Robots with Parallel Implementation on GPU

A. Durand-petiteville[1] and V. Cadenat[2]

[1] *Federal University of Pernambuco UFPE, Department of Mechanical Engineering, Av. da Arquitetura, 50740-550, Recife - PE, Brazil*
adrien.durandpetiteville@ufpe.br
[2] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
Univ de Toulouse, UPS, LAAS, F-31400, Toulouse, France
cadenat@laas.fr

Abstract

This work focuses on the control of a camera mounted on a differential drive robot via a VPC (Visual Predictive Control) scheme. First, an exact model of the visual feature prediction is presented for this robotic system. Next, relying on the equivalent command vector concept, a parallel implementation on a GPU (Graphics Processing Unit) of the computation of the cost function and its gradient is presented. Finally, results show that the proposed approach is more accurate than the ones classically used and can be up to six times faster than CPU-based (Central Processing Unit) one for large prediction horizons and numerous visual features. It then becomes possible to implement a VPC controller running sufficiently fast to perform a navigation tasks, while guaranteeing the closed-loop stability by relying on large prediction horizons.

Keywords: Mobile Robot, Visual Servoing, Model Predictive Control, Parallel Implementation

1. Introduction

IBVS (Image-Based Visual Servoing), a sub-division of visual servoing, aims at controlling the motion of a camera mounted on a robotic system. To do so, the control law is designed in the image space and relies on the minimization of an error between the values of some visual features computed in the current image and their values of reference, corresponding to the task to achieve [1]. Thus, there is no need to estimate

the robot localization in a global frame and it is sufficient to track the visual features in the image. This method is usually chosen for its reactivity, the lack of localization, and its large stability margins [1]. However, the camera pose not being taken into account by the control law, the trajectory is entirely dependent of the number and types of visual features used to define the task (points, lines, moments, etc [1]). It is then difficult to provide any guarantee regarding the camera pose during the servoing in order to avoid collisions or occultations. Moreover, the evolution of the visual features in the image and the camera trajectory can be incompatible and induce a failure of IBVS. For example, it is well known that a classical IBVS controller cannot achieve a pure rotation around a set of point visual features [1]. Among the many advanced IBVS schemes developed over the last two decades [1], the VPC (Visual Predictive Control) approach [2] seems one of the most relevant to tackle these theoretical challenges.

VPC is the fusion between IBVS and NMPC (Nonlinear Model Predictive Control) [3]. It consists of a NMPC scheme where, similarly to IBVS, the camera pose is defined in the image space by a set of visual features. Thus, the task to achieve is classically defined by a cost function, which is the sum over a prediction horizon of the difference between the predicted and desired states. Next, a set of constraints dealing with the specific features of the task is added to the NMPC scheme. Finally, a numerical solver minimizing the cost function while taking into account the constraints is used to compute the command. The obtained control scheme thus combines the advantages of IBVS, *i.e.,* reactivity, absence of metric localization, and large stability margins [1], with the ones of NMPC, *i.e.,* ability to explicitly deal with constraints such as feature visibility, collision with obstacles, or control inputs boundaries. For this reason, the interest for VPC-based controllers has grown and several schemes were designed to control different robotic systems (see Table 1): a camera mounted on a robotic arm [4][5][6][7][8], a flying camera [9], a differential robot [10] a car-like robot[11], an underwater robotic vehicle [12] or a continuum robot [13]. In addition to this first difference, the mentioned works stand out on several points. For example, the choice of the visual features: VPC schemes generally rely on points to define the system state in the image space, with the exception of [4] where image moments are used to increase the robustness of the control algorithm. Another difference concerns the pre-

Table 1: Summaries of works related to VPC

	Robot	Visual features	Prediction model	Constraints	Stability
[4]	Robotic arm	Moments	IM-FOA	Inputs, Joints, Visibility	-
[5]	Robotic arm	Points	LPVS	Inputs, Joints, Visibility	Liapunov stability
[6]	Robotic arm	Points	SOM	Inputs, Joints, Visibility	-
[7]	Robotic arm	Points	IM-FOA	Inputs, Visibility	-
[8]	Robotic arm	Points	IM-FOA	Inputs	-
[9]	Flying camera	Points	IM-FOA	Inputs, Visibility	Terminal constrained set
[10]	Differential robot	Points	IM-FOA	Inputs, Visibility	-
[11]	Car-like robot	Points	IM-FOA	Inputs, Accelerations	-
[12]	Underwater robot	Points	IM-FOA	Inputs, Visibility	-
[13]	Continuum robot	Points	IM-FOA	Inputs, Visibility	-

IM-FOA: interaction matrix-based first-order approximation LPVS: linear parameter-varying system SOM: second-order model

diction models, which are generally obtained by integrating using the Euler's method a first order system based on the interaction matrix [7][12][13]. Two works propose a different approach: (i) in [5] the robotic arm is represented by a linear parameter-varying system in order to obtain a model independent of the visual feature depth and (ii) in [6] the authors use a second order model allowing to integrate the visual features acceleration to obtain better and smoother 2D and 3D trajectories. Regarding the minimization problem resolution, most of the works rely on Quadratic Programming or Interior Point Algorithm whereas the authors of [10] propose to use a primal-dual neural network. Concerning the constraints, the majority of the mentioned works use them to bound the control inputs (limits of the actuators) and the state variables (visual features visibility)[7] [12][13]. However, there exists a couple of more advanced uses. For example, [9] tightens the constraints to take into account the state uncertainties, allowing to obtain more robust controllers. The feasibility problem, and therefor the stability one, is a last example of such differences between the previously mentioned works. Among them, only [9] partially addresses this problem by adding a zero terminal equality constraint, whereas it is a key issue when designing a controller. As shown by this overview of some of the most recent works related to VPC, the coupling of IBVS with MPC (Model Predictive Control) requires to address several issues: design of a sufficiently accurate and robust prediction model in the image space, design of

constraints dealing efficiently with the system and environment specificity, design of an optimal problem guaranteeing the closed-loop stability, and selection or design of a numerical solver computing within an acceptable time a local or global solution for convex or non-convex problems. While the mentioned works take into account only one or two of these issues at a time, we propose to rely on a VPC scheme simultaneously dealing with obstacles, closed-loop stability and non-convex sets in the context of a navigation task. We then introduce an approach for a parallel implementation of the VPC scheme, leading to an efficient resolution of the optimization problem. It thus becomes possible to safely navigate in a cluttered environment, while guaranteeing the closed-loop stability.

The proposed work differs from both NMPC-based and VPC-based approaches. Indeed, NMPC-based approaches dedicated to navigation propose to track a path or trajectory calculated before the navigation [14]. Such approaches require a model of the environment prior to the start and are not suited to deal with unknown obstacles discovered along the navigation. In this work and similarly to [10], the navigation problem is setup such as the reference is defined in the image space and corresponds to a unique pose in the Cartesian space. The trajectory described by the robot thus results from the optimization problem solved at each iteration and whose constraints are modified online based on the newly acquired data. Concerning the VPC-based control strategies, in addition to tackling simultaneously several of the previously mentioned difficulties (stability, obstacle avoidance leading to non-convex sets and resolution time), we consider a navigation problem where the camera has to perform large displacements, which is not the case of the above cited works. These latter consider relatively small camera displacements with a short prediction horizon, and boundaries constraints leading to convex state and inputs sets. For such cases, VPC schemes without any explicit stability guarantees are usually sufficient to achieve the task. On the contrary, for the scenario considered in this paper, large prediction horizons and non-convex sets, it is mandatory to explicitly address the stability issue to guarantee the navigation success.

To do so, let us first recall that nominal NMPC schemes, and therefore VPC, only guarantee a stable closed-loop when considering an infinite prediction horizon [15]. For solutions relying on a finite prediction horizon, which is the case for all the men-

tioned works, two main classes of approach are identified to guarantee stability. The first one enforces stability by adding a zero terminal equality constraint which forces the last predicted state to be equal to the desired one [16, 17]. If the solution to the optimal problem respects this constraint, the problem is then feasible and it is guaranteed that the trajectory leads to the desired state [18]. The second one relies on the quasi-infinite horizon method [19], which consists in adding a terminal penalty term to the cost function and a terminal region constraint. Both are determined off-line such that the modified cost function gives an upper bound on the infinite horizon cost and guarantees a decrease in the cost function. This second class of solution does not seem to be appropriate to the navigation problem. Indeed, the obstacles are detected during the navigation and the constraints related to obstacle avoidance might be updated at every iteration. It is then impossible to determine a terminal region at the beginning of the navigation. Moreover, the quasi-infinite horizon method requires that there exists a known control law, local to the terminal region, that stabilizes the system and satisfies the constraints. Again, it is impossible to prove the existence of such a local control law without knowing the constraints in the terminal region.

Therefor, in this work, the stability is guaranteed by adding a zero terminal equality constraint. For such an approach, the size of the feasibility set, set for which there exists a trajectory reaching the goal while dealing with the constraints, depends on both the length of the prediction horizon and the boundaries of the control inputs. The latter depending on the physical limits of the actuators, it might be necessary to use a large prediction horizon to guarantee the stability. However, large prediction horizons increases the computational burden to obtain the cost function and its gradient. Indeed, they require to compute several thousand times the predicted visual features in a serial fashion, each predicted state depending on the previous one. To overcome this issue, we propose to rely on a GPU (Graphics Processing Unit) to execute some parts of the optimization process. The use of GPU in the field of trajectory optimization was investigated and has led to algorithm-level parallelism and instruction-level parallelism. With the first approach the underlying algorithm is modified to create more opportunities for simultaneous execution of instructions. Several multiple shooting methods were designed to parallelize an SQP (Sequential Quadratic Programming)

algorithm [20] or an iLQR (iterative Linear Quadratic Regulator) one [21] [22]. The second approach consists in implementing in a parallel fashion some of the operations performed by the algorithm, which does not change the theoretical properties of the algorithm. For example, in [23] the predicted states of a linear system are computed in a parallel fashion for a MPC navigation problem. In this paper the main contribution consists in investigating a parallel implementation of the predictive function for a VPC scheme. While it was straightforward in [23] to parallelize a linear kinematic system, the visual features case is more challenging due to the presence of non-linearities in the recursive prediction scheme. To overcome this issue, it is proposed to compute independently each predicted image directly from the last measured one. In other words, the predictions relying on kinematic models, we need a command directly linking any predicted camera state to the current one. To solve this problem, we present the equivalent control vectors allowing to compute such a command. It then becomes possible to implement on a GPU the computation of the cost function and its gradient and, thus to rely on large prediction horizons to guarantee the stability even for large displacements, such as the ones in the navigation tasks. Moreover, the proposed approach is compatible with the GPU implementation of the SQP or iLQR algorithms to further reduce the computation time.

The paper is organized as follows. In section II, the system model is described. Next, the visual features and several prediction models are detailed. Section IV focuses on the VPC scheme and different constraints that can be added to the problem. Section V is dedicated to the parallel implementation on GPU. Finally, section VI presents results comparing the proposed model and its implementation to classical approaches.

2. System modeling

In this work, a pinhole camera is controlled via a VPC scheme. To model the system, an orthonormal frame $\mathbf{F}_c(O_c, \mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c)$ represents the camera. The camera pose χ_c is expressed in the world frame $\mathbf{F}_o(O, \mathbf{x}_o, \mathbf{y}_o, \mathbf{z}_o)$. Finally, the camera kinematic screw $\mathbf{\Gamma}^{F_c}_{O_c/F_o}$, calculated at point O_c with respect to $\mathbf{F}_0$ and expressed in $\mathbf{F}_c$, can be

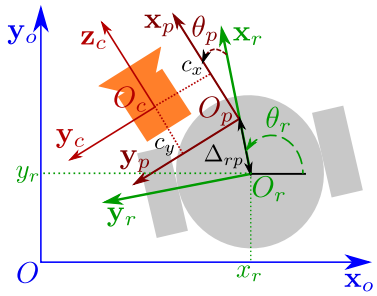


Figure 1: System model

decomposed as follows:

$$\mathbf{\Gamma}^{\mathbf{F}_c}_{O_c/\mathbf{F}_o} = \mathbf{\Gamma} = \begin{bmatrix} V^{\mathbf{F}_c}_{\mathbf{x}_c} & V^{\mathbf{F}_c}_{\mathbf{y}_c} & V^{\mathbf{F}_c}_{\mathbf{z}_c} & \Omega^{\mathbf{F}_c}_{\mathbf{x}_c} & \Omega^{\mathbf{F}_c}_{\mathbf{y}_c} & \Omega^{\mathbf{F}_c}_{\mathbf{z}_c} \end{bmatrix}^T \tag{1}$$

where $V^{\mathbf{F}_c}_{\mathbf{x}_c}$, $V^{\mathbf{F}_c}_{\mathbf{y}_c}$ and $V^{\mathbf{F}_c}_{\mathbf{z}_c}$ are respectively the linear velocities along $\mathbf{x}_c$, $\mathbf{y}_c$ and $\mathbf{z}_c$ expressed in $\mathbf{F}_c$. Following the same idea, $\Omega^{\mathbf{F}_c}_{\mathbf{x}_c}$, $\Omega^{\mathbf{F}_c}_{\mathbf{y}_c}$ and $\Omega^{\mathbf{F}_c}_{\mathbf{z}_c}$ are respectively the angular velocities along $\mathbf{x}_c$, $\mathbf{y}_c$ and $\mathbf{z}_c$ expressed in $\mathbf{F}_c$.

The camera is embedded on a differential robot equipped with a pan-platform. Let define $\mathbf{F}_r(O_r, \mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r)$ the robot frame and $\mathbf{F}_p(O_p, \mathbf{x}_p, \mathbf{y}_p, \mathbf{z}_p)$ the platform frame (see Figure 1). Let θ_r be the direction of the robot with respect to $\mathbf{x}_o$, θ_p the direction of the pan-platform with respect to $\mathbf{x}_r$, O_p the pan-platform center of rotation and Δ_{rp} the distance between the robot reference point O_r and O_p. Moreover, with x_r and y_r the coordinates of the point O_r in $\mathbf{F}_o$, we define the mobile base state as $\chi_r = [x_r, y_r, \theta_r]^T$. The control input is given by $\mathbf{Q} = [\upsilon, \omega_r, \omega_p]^T$, where υ and ω_r are the mobile base linear and angular velocities, and ω_p is the pan-platform angular velocity with respect to $\mathbf{F}_r$. Thus, it is possible to obtain the following kinematic model for the mobile base:

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \end{bmatrix} = \begin{bmatrix} \upsilon \cos(\theta_r) \\ \upsilon \sin(\theta_r) \\ \omega_r \end{bmatrix} \tag{2}$$

The camera being embedded on a differential robot equipped with a pan-platform, it only has three degrees of freedom. For this reason, with x_c and y_c the coordinates of

the point O_c in $\mathbf{F}_o$ and $\theta_c = \theta_r + \theta_p$, we define a reduced camera state such as:

$$\bar{\chi}_c = [x_c, y_c, \theta_c]^T \tag{3}$$

The corresponding reduced kinematic screw is then given by:

$$\begin{array}{rcccl} \overline{\mathbf{\Gamma}}^{\mathbf{F}_c}_{O_c/\mathbf{F}_o} & = & \overline{\mathbf{\Gamma}} & = & \left[V^{\mathbf{F}_c}_{\mathbf{y}_c}, V^{\mathbf{F}_c}_{\mathbf{z}_c}, \Omega^{\mathbf{F}_c}_{\mathbf{x}_c}\right]^T \\ & = & \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta}_c \end{bmatrix} & = & \begin{bmatrix} \upsilon\cos(\theta_r) - \omega_r \Delta_{rp}\sin(\theta_r) \\ \upsilon\sin(\theta_r) + \omega_r \Delta_{rp}\cos(\theta_r) \\ \omega_r + \omega_p \end{bmatrix} \end{array} \tag{4}$$

3. The visual features and the prediction models

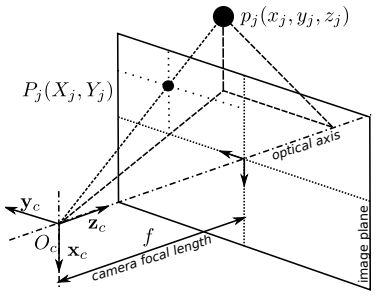


Figure 2: Perspective camera model

A VPC scheme relies on a landmark to control a camera with a focal length f (see Figure 2. We assume that this landmark can be characterized by N_v interest points which are extracted by an image processing. Therefore, the visual data are represented by a $2N_v$ dimensional vector $\mathbf{S}$. A point p_j, whose coordinates in the camera frame are given by (x_j, y_j, z_j), is represented by a point P_j whose coordinates are $S_j = (X_j, Y_j)$ in the image plane, with $j \in [1, ..., N_v]$ (see Figure 2).

In a VPC scheme, the state is given by the visual feature vector $\mathbf{S}$. With a predictive control scheme, it is mandatory to establish a model computing the future states. Here, we present three models to predict the evolution of the visual features. The two first ones are extracted from the literature [2] whereas the third one is specifically computed for a differential robot.

3.1. Global model

The global prediction model consists in de-projecting a point from the initial image to the initial camera frame ($\mathbb{R}^2 \rightarrow \mathbb{R}^3$), computes its coordinates in the prediction camera frame ($\mathbb{R}^3 \rightarrow \mathbb{R}^3$), and finally projects this point in the prediction image ($\mathbb{R}^3 \rightarrow \mathbb{R}^2$). By defining the projection matrix $\mathbf{H}_{i/c}$ and the homogeneous matrix $\mathbf{H}_{c(t_1)/c(t_2)}$ between two camera poses at instants t_1 and t_2 such as:

$$\begin{bmatrix} X_j \\ Y_j \\ z_j \\ 1 \end{bmatrix} = \begin{bmatrix} f/z_j & 0 & 0 & 0 \\ 0 & f/z_j & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} = \mathbf{H}_{i/c} \begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} \tag{5}$$

$$\mathbf{H}_{c(t_1)/c(t_2)} = \begin{bmatrix} \mathbf{R}_{c(t_1)/c(t_2)} & \mathbf{T}_{c(t_1)/c(t_2)} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{6}$$

where $\mathbf{R}_{c(t_1)/c(t_2)}$ and $\mathbf{T}_{c(t_1)/c(t_2)}$ are respectively a 3×3 rotation matrix and a 3×1 translation vector between $\mathbf{F}_c(t_1)$ and $\mathbf{F}_c(t_2)$, we obtain the global prediction model:

$$\mathbf{P_j}(t_2) = \mathbf{H}_{i/c(t_2)} \mathbf{H}^{-1}_{c(t_1)/c(t_2)} \mathbf{H}^{-1}_{i/c(t_1)} \mathbf{P_j}(t_1) \tag{7}$$

3.2. Local model

It relies on the mapping between the visual features evolution and the camera kinematic screw via the interaction matrix, classically given for a point P_j by [1]:

$$\mathbf{J}_i = \begin{bmatrix} \frac{-f}{z_j} & 0 & \frac{X_j}{z_j} & \frac{X_j Y_j}{f} & -(f + \frac{X_j^2}{f}) & Y_j \\ 0 & \frac{-f}{z_j} & \frac{Y_j}{z_j} & (f + \frac{X_j^2}{f}) & \frac{-X_j Y_j}{f} & X_j \end{bmatrix} \tag{8}$$

The local model is thus given by the integration between the current instant and the prediction one of the following equation:

$$\dot{\mathbf{S}}_j = \mathbf{J}_i \mathbf{\Gamma} \tag{9}$$

3.3. 3 DOF local model

In this paper, it is proposed to predict the visual features using a local model in the case of a 3 DOF camera. In other words, Equation 9 is modified to include the robotic

system carrying the camera. Let define the robot Jacobian by [24]:

$$\mathbf{J}_r = \begin{bmatrix} -\sin(\theta_p) & \Delta_{rp}\cos(\theta_p) + c_x & c_x \\ \cos(\theta_p) & \Delta_{rp}\sin(\theta_p) - c_y & -c_y \\ 0 & -1 & -1 \end{bmatrix} \tag{10}$$

where c_x and c_y are the coordinates of O_c along axes $\mathbf{x}_p$ and $\mathbf{y}_p$ (see Figure 2), and the reduced interaction matrix, *i.e.,* for a 3 DOF camera, by:

$$\overline{\mathbf{J}}_i = \begin{bmatrix} \frac{X_j}{z_j} & \frac{X_j Y_j}{f} & -(f + \frac{X_j^2}{f}) \\ \frac{Y_j}{z_j} & (f + \frac{X_j^2}{f}) & \frac{-X_j Y_j}{f} \end{bmatrix} \tag{11}$$

It is then possible to rewrite Equation 9 such as:

$$\dot{\mathbf{S}}_j = \overline{\mathbf{J}}_i \overline{\mathbf{\Gamma}} = \overline{\mathbf{J}}_i \mathbf{J}_r \mathbf{Q} \tag{12}$$

The robot is a sampled system whose inputs evolve at each instant $t = kT_s$, where T_s is the sampling time. By assuming that the inputs $\mathbf{Q}(t_1)$ are constant during the two instants t_1 and $t_2 = t_1 + T_s$, it is then possible to solve (12) between t_1 and t_2. After some computations (see [25]), we obtain:

$$\begin{aligned} X_j(t_2) &= \tfrac{z_j(t_1) X_j(t_1)}{z_j(t_2)} \\ Y_j(t_2) &= \tfrac{f}{z_j(t_2)} \Big\{ C_1 \cos(A) - C_2 \sin(A) + \Delta_{rp} \sin(\theta_p(t_2)) + \tfrac{v(t_1)}{\omega_r(t_1)} \cos(\theta_p(t_2)) - c_y \Big\} \\ z_j(t_2) &= C_1 \sin(A) + C_2 \cos(A) - \Delta_{rp} \cos(\theta_p(t_2)) + \tfrac{v(t_1)}{\omega_r(t_1)} \sin(\theta_p(t_2)) - c_x \end{aligned} \tag{13}$$

where:

$$\begin{aligned} A = &\quad \Big(\omega_r(t_1) + \omega_p(t_1)\Big) T_s \\ C_1 = &\quad \tfrac{Y_j(t_1) z_j(t_1)}{f} - \Delta_{rp} \sin(\theta_p(t_1)) - \tfrac{v(t_1)}{\omega_r(t_1)} \cos(\theta_p(t_1)) + c_y \\ C_2 = &\quad z_j(t_1) + \Delta_{rp} \cos(\theta_p(t_1)) - \tfrac{v(t_1)}{\omega_r(t_1)} \sin(\theta_p(t_1)) + c_x \end{aligned}$$

Unlike the two other ones, this model is a closed-form expression. It can be used to predict exactly the coordinates of the visual features. Moreover, it does not require any advanced/complex operation, offering a low computational cost.

Remark: *The three models rely on the z coordinate of the visual features. Thus, to accurately predict the values of the visual features, it is mandatory to estimate it or to measure it with a sensor such as a 3D camera. Using a constant value, which is widely used in classical IBVS [1], might lead to large prediction errors, especially for the navigation problem where the value of z has large variations.*

4. Visual Predictive Control

In this section, we first recall the VPC framework and the main parameters impacting the controller behavior. Then, we present constraints that could be taken into account either to guarantee the stability of the closed-loop system or the safety of the robot.

4.1. The VPC Scheme

A VPC scheme consists in coupling NMPC with IBVS. On the one hand, similarly to NMPC, it consists of computing an optimal control sequence $\overline{\mathbf{Q}}^*(.)$ that minimizes a cost function J_{N_p} over a prediction horizon of N_p steps while taking into account a set of user-defined constraints $\mathbf{C}(\overline{\mathbf{Q}}^*(.))$. The optimal control sequence is of length N_c, which represents the control horizon. In other words, the N_c^{th} first predictions are computed using independent control inputs, while the remaining ones are all obtained using a unique control input equals to $\overline{\mathbf{Q}}^*(N_c)$. On the other hand, similarly to IBVS, the task to achieve is defined as an error in the image space. To do so, we define $\mathbf{S}$ as the state vector containing the coordinates of N_v visual features and $\mathbf{S}^*$ as the desired state. In this work, we use points as visual features, and in this particular case $\mathbf{S} = [X_1, Y_1, ..., X_j, Y_j, .., X_{N_v}, Y_{N_v}]^T$. Finally, the cost function to minimize is defined as the sum of the quadratic error between the visual feature coordinates vector $\hat{\mathbf{S}}(.)$ predicted over the horizon N_p and the desired ones $\mathbf{S}^*$. Note that the proposed cost function is the one traditionally used for VPC schemes, but it does not represent the only choice. The optimal problem is then defined as follows:

$$\overline{\mathbf{Q}}^*(.) = \underset{\overline{\mathbf{Q}}(.)}{\min} \left(J_{N_p}(\mathbf{S}(k), \overline{\mathbf{Q}}(.)) \right) \tag{14}$$

with

$$J_{N_p}(\mathbf{S}(k), \overline{\mathbf{Q}}(.)) = \sum_{p=k+1}^{k+N_p} [\hat{\mathbf{S}}(p) - \mathbf{S}^*]^T [\hat{\mathbf{S}}(p) - \mathbf{S}^*] \tag{15}$$

subject to

$$\hat{\mathbf{S}}(k+1) = f(\hat{\mathbf{S}}(k), \mathbf{Q}(k)) \tag{16a}$$

$$\hat{\mathbf{S}}(k) = \mathbf{S}(k) \tag{16b}$$

$$C(\overline{\mathbf{Q}}^*(.)) \leq 0 \tag{16c}$$

where $\overline{\mathbf{Q}}(.) = [\mathbf{Q}(k), ..., \mathbf{Q}(k + N_p - 1)]$. The prediction function $f(\hat{\mathbf{S}}(k), \mathbf{Q}(k))$ in Equation 16a corresponds to the prediction model given in Equation 13. Moreover, the predicted visual features rely on the last measured ones, as stated by Equation 16b. Finally, Equation 16c is used to include the constraints in the optimization problem. Some of the constraints that could be included are presented in the following sections.

Remark 1: Solving Equation 14 leads to the optimal sequence $\overline{\mathbf{Q}}^*(.)$. As it is usually done, only the first element $\overline{\mathbf{Q}}^*(1)$ is applied to the system. At the next iteration, the minimization problem is restarted, and a new $\overline{\mathbf{Q}}^*(.)$ is computed. This loop is repeated until the task is achieved.

Remark 2: Numerical solvers require an initial value for the vector to optimize. In this work, the results of the previous optimization are used as the initial guess of the current one.

4.2. The Zero Terminal Equality Constraint

In this work, the stability of the VPC scheme relying on a finite prediction horizon is achieved by adding a zero terminal equality constraint. It is defined as the error between the prediction of the visual features $\hat{\mathbf{S}}(k + N_p)$ obtained at the end of the prediction horizon, and the desired ones $\mathbf{S}^*$.

$$||\hat{\mathbf{S}}(k + N_p) - \mathbf{S}^*|| = 0 \tag{17}$$

An equality constraint being almost impossible to achieve, we use the following inequality constraint:

$$||\hat{\mathbf{S}}(k + N_p) - \mathbf{S}^*|| - \delta_{tc} \leq 0 \tag{18}$$

where δ_{tc} is a user defined threshold sufficiently small to impact the optimization process similarly to the equality while offering an efficient implementation of the constraint.

The respect of this constraint leads to a computed trajectory connecting the current state to the desired one. It is then possible to guarantee the recursive feasability of the problem and thus to obtain the closed-loop stability [18]. If the solver cannot compute a $\overline{\mathbf{Q}}^*(.)$ that fulfills this constraint, then the prediction horizon is too short and/or the

constraints on the control inputs are too restrictive to reach the goal [3]. This issue is addressed in the following section.

Remark 3: Despite being a classical solution to guarantee the closed-loop stability of NMPC schemes, the works related to VPC studied during the literature review do not propose to include a terminal constraint. Instead, authors usually weight the last predicted value based on the distance to the desired one. This method helps the solver to converge towards an optimal solution but it does not guarantee the closed-loop stability.

Remark 4: The terminal constraint does not provide an absolute guarantee of the task realization. Indeed, in the case the predictions are strongly erroneous, the system cannot converge towards the real values of the desired states.

4.3. The Input Constraints

The constraints applied to the control input vector are usually in the form of boundaries. They allow to take into account the physical limits of the actuators and are expressed as inequality constraints:

$$\begin{bmatrix} \mathbf{Q}(i) - \mathbf{Q}_u \\ \mathbf{Q}_l - \mathbf{Q}(i) \end{bmatrix} \leq 0 \tag{19}$$

where $i \in [1, ..., N_c]$, and $\mathbf{Q}_l$ and $\mathbf{Q}_u$ are respectively the lower and upper boundaries corresponding to the actuators limits. Thus, the command applied to the robot, respects the actuators boundaries.

4.4. The Obstacle Avoidance Constraints

To perform a safe navigation, constraints can be used to avoid collision with the obstacles in the vicinity of the robot. Traditionally, the avoidance is obtained by guaranteeing a minimal distance between one or several points of the obstacles and the centroid of the robot for each predicted state. When considering small displacements between two states, this approach is sufficient to avoid collisions. In this work, it is proposed to only consider static, non-occulting[1], and circle-shaped obstacles. Each of

[1] Small enough for the camera to perceive the target from any configuration.

the N_o obstacles present in the scene is defined by $x_{o_m}, y_{o_m}, r_{o_m}$, with x_{o_m}, y_{o_m} the coordinates of the center, r_{o_m} the radius, and $m \in [1, ..., N_o]$. The collision risk is managed thanks to the following constraint:

$$\begin{bmatrix} \delta_s - \sqrt{\left(\hat{x}(k) - x_{o_1}\right)^2 + \left(\hat{y}(k) - y_{o_1}\right)^2} \\ \vdots \\ \delta_s - \sqrt{\left(\hat{x}(k+N_p) - x_{o_1}\right)^2 + \left(\hat{y}(k+N_p) - y_{o_1}\right)^2} \\ \delta_s - \sqrt{\left(\hat{x}(k) - x_{o_{N_o}}\right)^2 + \left(\hat{y}(k) - y_{o_{N_o}}\right)^2} \\ \vdots \\ \delta_s - \sqrt{\left(\hat{x}(k+N_p) - x_{o_{N_o}}\right)^2 + \left(\hat{y}(k+N_p) - y_{o_{N_o}}\right)^2} \end{bmatrix} \leq 0 \tag{20}$$

with $\delta_s = d_s + r_{o_m} + r_r$, where r_r represents the mobile base radius and d_s the safety distance around an obstacle. The predicted positions of the mobile base $\hat{x}(.)$ and $\hat{y}(.)$ are obtained by integrating Equation 2.

Remark 5: In this work, we consider one of the simplest form of the obstacle avoidance problem. Indeed, we use circle-shaped obstacles and the challenges related to their detection are not taken into account. However, adding obstacle avoidance constraints has a large impact on the problem. Indeed, unlike constraints such as command boundaries or camera field of view limits, it transforms the initial convex feasible set into a non-convex feasible set. In such a case, it becomes then necessary to use large values for N_c, which increases the complexity of the problem and the computation time necessary to solve it.

5. Parallel implementation

5.1. Parallelization of the problem

The implementation of a VPC scheme relies on a solver, which computes $\overline{\mathbf{Q}}^*(.)$ in an iterative fashion. To do so, at each iteration it is necessary to compute a large number of times the value of the cost function given in Equation 15 and its gradient (depending on the used solver). Rewriting Equation 15 such as:

$$J_{N_p}(\mathbf{S}(k), \overline{\mathbf{Q}}(.)) = \sum_{j=1}^{N_v} \sum_{p=k+1}^{k+N_p} J_{j|p} \tag{21}$$

where $J_{j|p} = [\hat{\mathbf{S}}_j(p) - \mathbf{S}_j^*]^T[\hat{\mathbf{S}}_j(p) - \mathbf{S}_j^*]$, it clearly appears that the computation of $J_{N_p}(\mathbf{S}(k), \overline{\mathbf{Q}}(.))$ contains two `for` loops. The length of the first one depends on the number N_v of visual features whereas the second one depends on the number N_p of predictive steps. Thus, the number of iterations required to compute the cost function can be large for a mobile robot. Indeed, we usually use more features than the minimum required number to obtain a controller robust to occultations and to deal with tracking issues. Moreover, to guarantee the feasibility of the predictive scheme and respect the terminal constraint, the number of prediction steps has to be sufficiently large. Thus, one of the challenges to address in order to control a robot with a VPC scheme consists in efficiently implementing the computation of the cost function for large values of N_v and N_p. To deal with this challenge, we propose to design a parallel algorithm computing the cost function and its gradient. The challenge consists then in breaking the two `for` loops to make possible the computation of each element $J_{j|p}$ independently.

First, the visual features being independent from each other, it is straightforward to split the calculation of the sequence of predicted values for each feature into N_v threads. Regarding the second loop, the prediction of the visual features being an iterative process (see Equation 13), it is currently impossible to compute each prediction individually.

To design a parallel implementation of the predictions, it is proposed to determine how an image at t_k can be directly linked to an image at $t_p > t_k$, without relying on the intermediate images at $t_{k+1}, ..., t_{p-1}$. It consists in determining the smallest sequence of control inputs allowing to reach the camera state $\chi_c(p)$ starting from the state $\chi_c(k)$. To do so, it is first required to study the controllability of the corresponding nonlinear discrete system $\chi_c(k+1) = g(\chi_c(k), \mathbf{Q}(k))$ where $g(\chi_c(k), \mathbf{Q}(k))$ is obtained by analytically solving equation (4) between t_k and t_{k+1}. Its expression is given by:

$$g : \begin{pmatrix} x_c(k+1) = x_c(k) + \frac{2v(k)}{\omega_r(k)} \sin(\eta_1)\cos(\eta_2) + 2\Delta_{rp}\sin(\eta_1)\sin(\eta_2) \\ y_c(k+1) = y_c(k) - \frac{2v(k)}{\omega_r(k)} \sin(\eta_1)\sin(\eta_2) + 2\Delta_{rp}\sin(\eta_1)\cos(\eta_2) \\ \theta_c(k+1) = \theta_c(k) + (\omega_r(k) + \omega_p(k))T_s \end{pmatrix} \quad (22)$$

with $\eta_1 = \frac{\omega_r(k)T_s}{2}$ and $\eta_2 = \frac{2\theta_r(k)+\omega_r(k)T_s}{2}$ and when $\omega_r \neq 0$ (the problem is straight-

forward if $\omega = 0$). According to [26], such a system is controllable in n_c steps if the following matrix $\mathbf{M}_c$ is full rank.

$$\mathbf{M}_c = \begin{bmatrix} \frac{\partial g(\chi_c(n_c-1),\mathbf{Q}(n_c-1))}{\partial \mathbf{Q}(n_c-1)} \\ \frac{\partial g(\chi_c(n_c-1),\mathbf{Q}(n_c-1))}{\partial \chi_c(n_c-1)} \frac{\partial g(\chi_c(n_c-2),\mathbf{Q}(n_c-2))}{\partial \mathbf{Q}(n_c-2)} \\ \dots \\ \frac{\partial g(\chi_c(n_c-1),\mathbf{Q}(n_c-1))}{\partial \chi_c(n_c-1)} \quad \dots \quad \frac{\partial g(\chi_c(1),\mathbf{Q}(1))}{\partial \chi_c(1)} \frac{\partial g(\chi_c(0),\mathbf{Q}(0))}{\partial \mathbf{Q}(0)} \end{bmatrix}^T \tag{23}$$

For $n_c = 1$, *i.e.,* the camera is controllable in one step, we obtain:

$$\mathbf{M}_c = \frac{\partial g(\chi_c(0), \mathbf{Q}(0))}{\partial \mathbf{Q}(0)} \tag{24}$$

After some computation, we obtain:

$$\mathbf{M}_c = \begin{bmatrix} \frac{2}{\omega_r(0)} \sin(\eta_1)\cos(\eta_2) & \xi_1 & 0 \\ \frac{2}{\omega_r(0)} \sin(\eta_1)\sin(\eta_2) & \xi_2 & 0 \\ 0 & T_s & T_s \end{bmatrix} \tag{25}$$

where

$$\xi_1 = -\frac{2\upsilon(0)}{\omega_r^2(0)} \sin(\eta_1)\cos(\eta_2) + \frac{\upsilon(0)}{\omega_r(0)} T_s \cos(\eta_1+\eta_2) - \Delta_{rp} T_s \sin(\eta_1+\eta_2)$$

$$\xi_2 = -\frac{2\upsilon(0)}{\omega_r^2(0)} \sin(\eta_1)\sin(\eta_2) + \frac{\upsilon(0)}{\omega_r(0)} T_s \sin(\eta_1+\eta_2) + \Delta_{rp} T_s \cos(\eta_1+\eta_2)$$

In order to evaluate the rank of the matrix, we compute the determinant:

$$det(M_c) = \frac{\upsilon(0)}{\omega_r(0)} T_s \sin(\eta_1) + \Delta_{rp} T_s \cos(\eta_1) \tag{26}$$

The determinant in Equation 26 being non-null for $\tan(\eta_1) \neq \frac{\Delta_{rp}\omega_r(0)}{\upsilon(0)}$, M_c is full rank for $n_c = 1$. Thus, it exists a constant control input vector $\tilde{\mathbf{Q}}_{t_k|t_p}$, named equivalent control vector, allowing to reach the camera state at t_p from the one at t_k in one step. This equivalent control vector allows to link two images without the need of intermediate ones as it is illustrated in Figure 3.

Finally, it remains to establish the equations allowing to compute each element of the equivalent control vector $\tilde{\mathbf{Q}}_{t_k|t_p}$. The system being controllable in one step, $\tilde{\mathbf{Q}}_{t_k|t_p}$ can be obtained by re-arranging the system given in Equation 22. Thus, defining

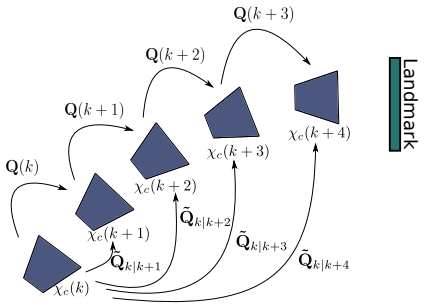


Figure 3: Example of equivalent control vectors

$\Delta_{x_c} = x_c(p) - x_c(k)$ and $\Delta_{y_c} = y_c(p) - y_c(k)$, we obtain:

$$\begin{aligned} \tilde{\upsilon} &= \sqrt{\tilde{\omega}_r^2 \left(\frac{\Delta_{x_c}^2 + \Delta_{y_c}^2}{4 \sin^2(\eta_{C1})} - \Delta_{rp}^2 \right)} \\ \tilde{\omega}_r &= \frac{2}{T_s} \arctan(\gamma) \\ \tilde{\omega}_p &= \frac{1}{T_s} \left[\theta_p(p) - \theta_p(k) + \theta_r(p) - \theta_r(k) + \tilde{\omega}_r T_s \right] \end{aligned} \tag{27}$$

with $\gamma = -\frac{\Delta_{x_c} \sin(\theta_r(k)) + \Delta_{y_c} \cos(\theta_r(k))}{2\Delta_{rp} + \Delta_{x_c} \cos(\theta_r(k)) + \Delta_{y_c} \sin(\theta_r(k))}$. By using the formulas given in Equation 27, it is now possible to obtain the equivalent control vector $\tilde{\mathbf{Q}}_{t_k|t_p} = [\tilde{\upsilon}, \tilde{\omega}_r, \tilde{\omega}_p]^T$ linking the image at instant t_p from the image at t_k. [2]

5.2. Implementation on GPU

The computation of each element $J_{j|p}$ of Equation 21 being independent, we now focus on the parallel implementation allowing to calculate a cost function for a given sequence of control inputs $\overline{\mathbf{Q}}(.) = [\mathbf{Q}(k), \ldots, \mathbf{Q}(k+N_p-1)]$ over a prediction horizon of N_p steps (see Figure 4). The computation of the cost function on a GPU is divided into three steps.

- First, the sequence of equivalent control vectors $[\tilde{\mathbf{Q}}_{t_k|t_k+1}, \ldots, \tilde{\mathbf{Q}}_{t_k|t_k+N_p}]$ are computed on the CPU using the formulas given in Equation 27.
- Next, $N_v \times N_p$ threads are launched on the GPU to calculate each $J_{j|p}$. Each

[2] It should be noted that the robot trajectory in the world frame computed with $\tilde{\mathbf{Q}}_{t_k|t_p}$ is not the same as the one calculated with the sequence $[\mathbf{Q}(k), ..., \mathbf{Q}(k+N_p-1)]$.

thread performs the prediction of the visual features using Equation 13 and calculate the difference with the reference values.

- Finally, the cost function J_{N_p} is obtained by summing the terms $J_{j|p}$ using $(N_v \times N_p)/2$ threads and a classical parallel sum algorithm.

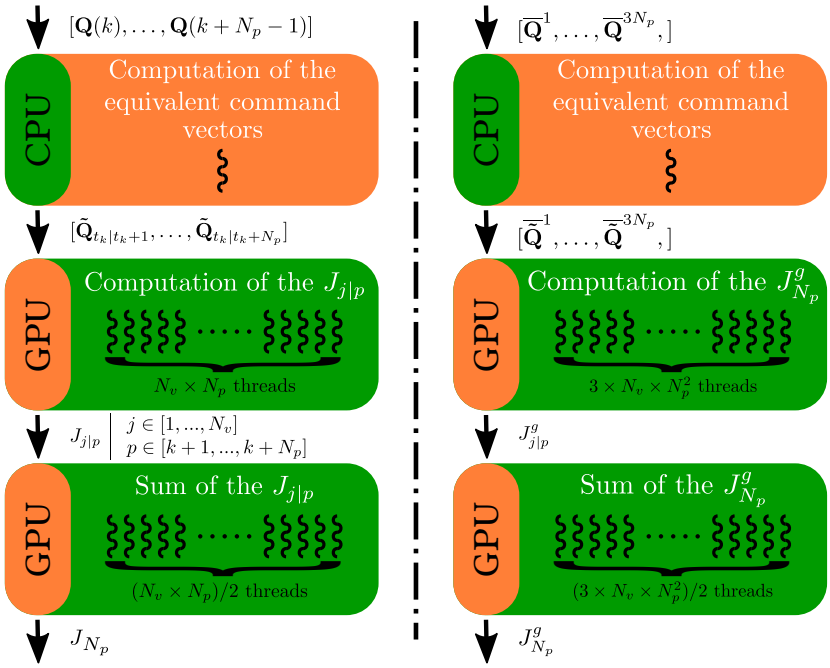


Figure 4: Left: Parallel computation of the cost function - Right: Parallel computation of the gradient

Among the solvers that can be used to minimize the cost function given in Equation 15, some require to provide the gradient of the cost function with respect to the sequence of control inputs $[\mathbf{Q}(k), \ldots, \mathbf{Q}(k+N_p-1)]$. Due to the iterative nature of the prediction process, it is challenging to provide an analytic expression of the gradient. It is then proposed to numerically compute the gradient. To do so, it is possible to rely on the parallel implementation designed for the cost function.

The control input sequence $\overline{\mathbf{Q}}(.)$ being of dimension $3N_p$, the gradient dimension is also $3N_p$. To compute each ∇_g, the g^{th} component of the gradient where $g \in [1, ..., 3N_p]$, a small variation δ_Q is added to the g^{th} component of the control input sequence $\overline{\mathbf{Q}}$. We obtain $3N_p$ modified control input sequences denoted $\overline{\mathbf{Q}}^g(.)$. They

are the input parameters of the parallel algorithm presented on the right side of Figure 4, and which is a slightly modified version of the previous algorithm.

- First, $3N_p$ equivalent command vectors sequences are calculated using formulas given in Equation 27.
- Next, $3N_p^2N_v$ parameters $J^g_{j|p}$ are computed thanks to Equation 13.
- The $J^g_{j|p}$ are summed to obtain the $3N_p$ elements $J^g_{N_p}$.
- Finally, each ∇_g is calculated using:

$$\nabla_g = \frac{J^g_{N_p} - J_{N_p}}{\delta_g} \tag{28}$$

Thus, thanks to the equivalent control vectors, the cost function and its gradient can be calculated using a parallel implementation. It should then be possible to consider a large number of visual features and prediction steps while maintaining a low computational cost.

6. Results

In this section, we present results obtained in simulation showing the relevance and the efficiency of the proposed approach. To do so, a program simulating the hardware and software flowcharts presented respectively in Figures 5 and 6 was developed using the C++ and CUDA languages. In Figure 6, the `Initialize Command`, `Cost Function`, `Terminal Constraint`, and `Obstacle Constraint` blocks were implemented relying on the methods presented in Sections 4 and 5. The `Move Robot` block simulates the robot displacement using the equations presented in Section 2. The `Image Processing` block simulates an image processing algorithm computing the coordinates in the image frame of four points characterizing the landmark of interest. To do so, we rely on the perspective camera model and on the global model presented in Section 3. The last block, the `Point Cloud Processing` one, simulates the process of data provided by a 2D laser rangefinder. It provides the coordinates in the robot frame of a set of points belonging to the obstacle's surface visible to the sensor. Finally, the solvers used in this work are part of the NLopt nonlinear-optimization

package. This implementation was tested on an Intel Core i7-8750H CPU running at 2.20GHz and a NVidia GeForce GTX 1060 GPU.

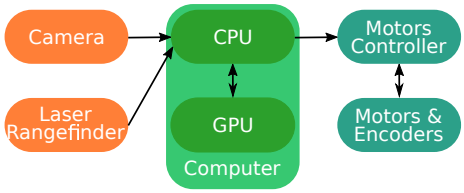


Figure 5: Hardware flowchart

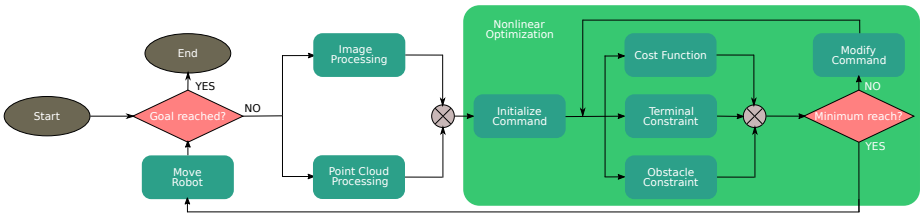


Figure 6: Software flowchart

6.1. Comparison of the models

In section 3, it was shown that we can rely on five models to predict the visual features for a 3 DOF camera[3]: the local model solved with the Euler method, the local model solved with Runge-Kutta (RK) method, the global model solved with the Euler method, the global model solved with RK method, and finally the local model for a 3 DOF camera presented in section 3.3. In this first part, we investigate the accuracy of the prediction models and their sensitivity to the sampling time and visual feature depth without taking into account the computation time. To do so, the following scenario is considered: the robot initial pose is $x_r = 0$, $y_r = 0.5$, $\theta_r = 0$ and $\theta_p = 0$, the desired camera pose is $x_c = 5.1$, $y_c = 0$, $\theta_c = 0$, and finally the landmark of interest is made of four points whose xyz coordinates are $[6, 0.5, 1.5]$, $[6, -0.5, 1.5]$, $[6, 0.5, 0.5]$, and $[6, -0.5, 0.5]$. Moreover, three cases are considered: case $\#1$ where $T_s = 0.2s$

[3]When a 6 DOF model is used, the non-available velocities are considered null.

and $|v| < 1m/s$, case $\#2$ where $T_s = 0.2s$ and $|v| < 0.5m/s$, and case $\#3$ where $T_s = 0.1s$ and $|v| < 1m/s$. The robot is controlled via a VPC scheme relying on the exact measures and data, and this represents the ground truth for the comparison. At each iteration, $\overline{\mathbf{Q}}^*(.)$ is used to predict the visual feature for each model. The quadratic error between the predictions and the ground truth over the prediction horizon is saved for each model. Finally, in Figure 7, we provide the sum of the errors over the whole servoing.

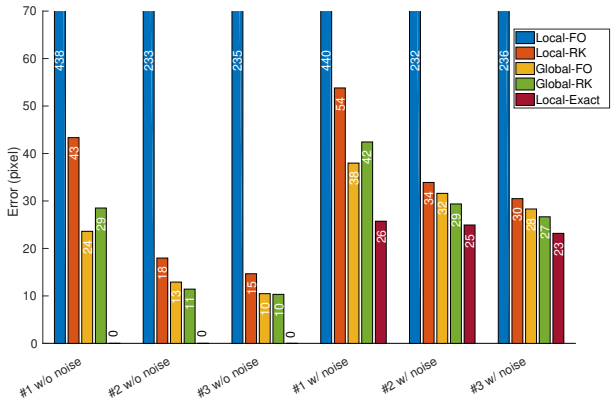


Figure 7: Total error of prediction for five models

In the first three groups of bars in Figure 7, the depth of the visual features is perfectly known. For such a case, the 3 DOF local model presented in this work perfectly predicts the visual features. The other model predictions are within a reasonable error margin, except the local model solved with the Euler method whose predictions are strongly erroneous. The 3 DOF local model being the exact closed-loop equation of the feature prediction, this result was predictable. Moreover, it is neither affected by the value of the sampling time T_s nor the linear velocity v. On the contrary, the other models are sensitive to these values. Indeed, when T_s or v are getting larger, the camera displacement between two sampling instants is larger, increasing the errors introduced by the linearizations included in these methods.

In the last three groups of bars in Figure 7, a zero-centered Gaussian noise with an amplitude of 10% of the real value is added to the depth to simulate a value measured

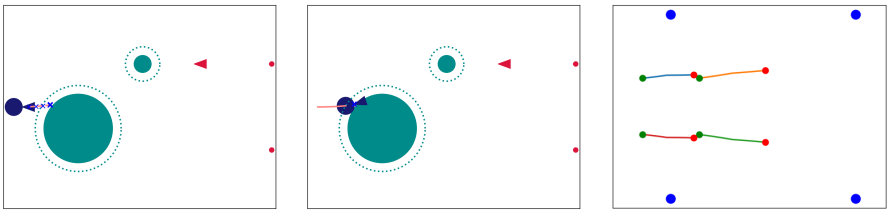
by a 3D camera or estimated via a non-linear observer. For the three cases, the 3 DOF local model is the most accurate. The other models offer similar performances for the cases $\#2$ and $\#3$, but are significantly less accurate for the first case. Once again, it is shown that the performances downgrade when the camera displacement between two sampling times increases.

Thus, for a 3 DOF camera mounted on a differential robot, it seems relevant to rely on the model presented in this work. Indeed, it is the most accurate model and the less sensitive one to the sampling time parameter and to the error on the visual features depth. Moreover, it is a closed-form model not requiring any advanced computation, and it allows a parallel implementation as presented in section 5.

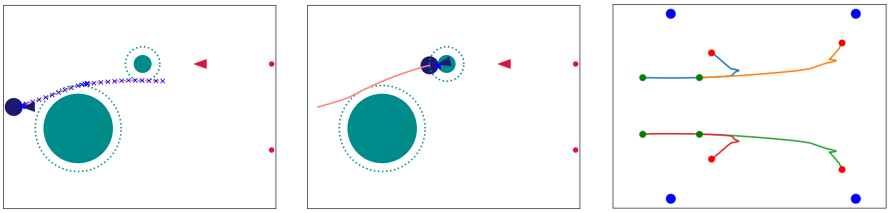
6.2. Examples of navigation via VPC

In this second set of results, we present a scenario highlighting the need for large values of prediction horizon in order to achieve navigation tasks. In this example, the robot has to drive the camera from its initial pose $\bar{\chi}_c(0) = [0, 0, 0]^T$ to the desired one $\bar{\chi}_c^* = [2, 0.5, 0]^T$ while dealing with two non-occulting obstacles and the following command boundaries: $0 \leq v \leq 0.4m/s$, $-0.1rad/s \leq \omega_r \leq 0.1rad/s$, and $-0.1rad/s \leq \omega_p \leq 0.1rad/s$. To do so, three different lengths for the prediction and control horizons are considered: $N_p = N_c = 10$, $N_p = N_c = 40$, and $N_p = N_c = 50$. The results are presented in Figure 8, where the robot and the camera are represented in dark blue, while the path executed by the mobile base is a plain orange line. The predicted path of the camera is represented by a dashed orange line and the predicted position of the camera by blue crosses. The desired camera pose is symbolized by a red triangle and the landmark is represented by red points. The obstacles are represented by a plain green circle and the safety boundary by a pointed green circle. In the figures representing the visual features evolution, green dots are the initial values, red dots the final values, and blue ones are the desired values.

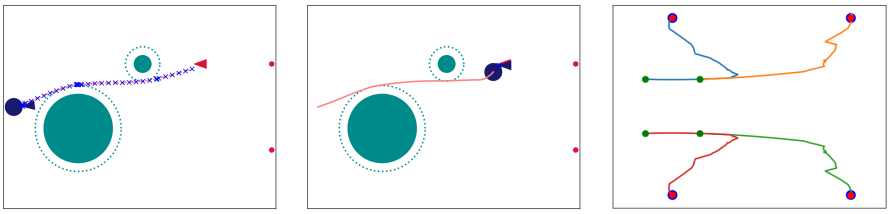
For the first test with $N_p = N_c = 10$, the predicted trajectory at the initial state does not reach the desired camera pose (Figure 8(a)). Indeed, the number of prediction steps is too small for the given command boundaries. The last predicted state not being in the vicinity of the desired one, the stability of the closed-loop controller is



(a) $N_p = 10$ - Initial state (b) $N_p = 10$ - Final state (c) $N_p = 10$ - Visual features



(d) $N_p = 40$ - Initial state (e) $N_p = 40$ - Final state (f) $N_p = 40$ - Visual features



(g) $N_p = 50$ - Initial state (h) $N_p = 50$ - Final state (i) $N_p = 50$ - Visual features

Figure 8: Three examples of navigation via VPC with $N_p = N_c = 10$ (a-c), $N_p = N_c = 40$ (d-f), and $N_p = N_c = 50$ (g-i)

thus not guaranteed and the navigation might fail. As it can be seen in Figure 8(b), this is what happens in this test. In the vicinity of the first obstacle, the robot reaches a local minima and stops without completing the task. Indeed, the visual features do not reach their desired values (Figure 8(c)). In the second test, we significantly increase the length of the prediction and control horizons by setting up $N_p = N_c = 40$. Despite a longer predicted trajectory, the final predicted state still does not lie in the vicinity of the camera desired state (Figure 8(d)). The stability is not guaranteed and the robot one more time does not manage to complete the task. Indeed, it reaches a local minima next to the second obstacle (Figure 8(e)) and the visual features do not converge towards their reference values (Figure 8(f)). Finally, we use $N_p = N_c = 50$. For this configuration, the number of prediction steps is large enough to compute a trajectory whose last state lies in the vicinity of the desired camera pose (Figure 8(g). The problem is thus feasible and the closed-loop stability is guaranteed. As it can be seen in Figures 8(h) and 8(i), the robot manages to drive the camera to its desired state, making thus the visual features converging towards their reference values.

In these examples, the robot has to drive less than 3 meters while dealing with obstacles, and it is already necessary to use at least 50 prediction steps to guarantee the system stability. It highlights then the need for large values of the prediction horizon in order to navigate using VPC schemes. However, large prediction horizons lead to large computation times, making this approach not eligible for an implementation on a robot. This is why it was mandatory to develop a new approach reducing the impact of the prediction horizon on the solving of the optimal problem.

6.3. Parallel implementation

In this last part, we measure the time necessary to solve the optimal problem. We only consider the 3 DOF local model and we use a scenario similar to the one described in 6.1. The computation time is measured for several configurations: N_p is equal to 10, 20, 30 or 40 and N_v is equal to 4, 8 or 16. Finally, five solvers are used to solve the optimal problem: i) the Nelder-Mead simplex algorithm, ii) the Sequential Quadratic Programming (SQP) algorithm with CPU implementation, iii) the Conservative Convex Separable Approximation (CCSA) algorithm using with CPU implementation, iv)

the SQP algorithm with GPU implementation, and v) the CCSA with GPU implementation. The Nelder-Mead simplex algorithm is the only solver among the selected ones that does not require to provide a value for the gradient. These nonlinear solvers were selected for their abilities to deal with nonlinear (in)equality constraints. Also, using different solvers can show the approach efficiency despite the selected solver.

In Figure 9, the processing times for the different solvers and configurations are given. First, it can be seen that providing an implementation of the gradient speeds up the minimization. Indeed, the processing times obtained with the Nelder-Mead simplex algorithm are always significantly larger than the ones obtained with the SQP and CCSA algorithms implemented on a CPU. Second, when comparing the CPU and GPU implementations for the SQP and CCSA algorithms, we note that for small values of N_p and N_v the processing times are similar. On the other hand, for larger values, the processing times obtained with a CPU implementation are significantly larger than the ones obtained with a GPU implementation. Thus, the computation on GPU seems relevant mostly for large values of N_p and N_v, such as when performing a navigation task. Computing one $J_{j|p}$ on a GPU being slower than on a CPU, it is usually relevant to rely on the GPU implementation for large numbers of parallel threads. In the present case, $N_p \times N_v = 120$ seems to be the threshold to switch from CPU to GPU. Indeed, in Figure 9, for $N_p = 30$ and $N_v = 4$, the processing times of the last four solvers are similar.

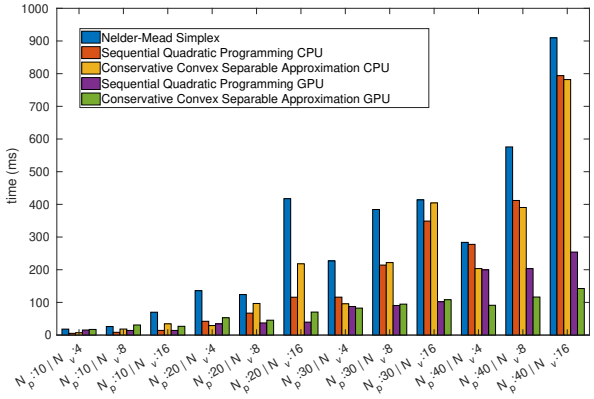


Figure 9: Processing time for five solver configurations

7. Conclusion

In this paper, the control of a differential robot equipped with a camera via a VPC scheme has been tackled. We first have presented a prediction model of the visual features for the specific robotic system (3 DOF camera). This model is the analytic solution to the prediction problem. As such, it offers an exact and efficient way to predict the visual features. Next, a parallel implementation of the calculation of the cost function and its gradient has been presented. It relies on the equivalent command vectors that allow to break a `for` loop computation into independent threads. Thus, it is possible to minimize the cost function in less than $150\ ms$ when using a prediction horizon of 40 steps and 16 visual features (worst case in our tests). However, when the number of prediction steps is small, the GPU implementation does not seem relevant to decrease the optimization time.

This represents a first step towards the use of VPC controllers to perform navigation tasks, while guaranteeing the closed-loop stability by relying on large prediction horizons. However, the results were obtained for the specific case of a 3 DOF camera. It would then be interesting to generalize the approach to a 6 DOF camera in order to be able to use for any kind of robotic system. Moreover, other processes could benefit from the equivalent vectors in order to reduce their computation time. For example, the constraints to avoid collisions with obstacles or to avoid that the visual features leave the field of view require a large amount of computation. Implementing these processes in a parallel fashion would allow to rely on a VPC controller in a complex and challenging environment. Finally, in order to further reduce the total optimization time, it would interesting to couple this instruction-level approach with algorithm-level ones, such as the iLQR algorithm [22] or the Model Predictive Path Integral controller [7]

References

[1] F. Chaumette, S. Hutchinson, P. Corke, Visual servoing, in: Springer Handbook of Robotics, Springer, 2016, pp. 841–866.

[2] G. Allibert, E. Courtial, F. Chaumette, Predictive control for constrained image-based visual servoing, IEEE Trans. on Robotics 26 (5) (2010) 933–939.

[3] L. Grüne, J. Pannek, Nonlinear model predictive control, in: Nonlinear Model Predictive Control, Springer, 2017, pp. 45–69.

[4] A. Burlacu, C. Copot, C. Lazar, Predictive control architecture for real-time image moments based servoing of robot manipulators, Journal of Intelligent Manufacturing 25 (5) (2014) 1125–1134.

[5] S. Liu, J. Dong, Robust online model predictive control for image-based visual servoing in polar coordinates, Transactions of the Institute of Measurement and Control 42 (4) (2020) 890–903.

[6] F. Fusco, O. Kermorgant, P. Martinet, Integrating features acceleration in visual predictive control, IEEE Robotics and Automation Letters 5 (4) (2020) 5197–5204.

[7] I. Mohamed, G. Allibert, P. Martinet, Sampling-based mpc for constrained vision based control, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2021), 2021.

[8] S. Sajjadi, M. M. Fallah, M. Mehrandezh, F. Janabi-Sharifi, Stochastic image-based visual predictive control, in: 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), IEEE, 2021, pp. 551–556.

[9] S. Heshmati-alamdari, G. K. Karavas, A. Eqtami, M. Drossakis, K. J. Kyriakopoulos, Robustness analysis of model predictive control for constrained image-based visual servoing, in: 2014 IEEE Int. Conf. on Robotics and Automation, 2014, pp. 4469–4474.

[10] F. Ke, Z. Li, H. Xiao, X. Zhang, Visual servoing of constrained mobile robots based on model predictive control, IEEE Transactions on Systems, Man, and Cybernetics: Systems 47 (7) (2017) 1428–1438.

[11] D. Pérez-Morales, O. Kermorgant, S. Domínguez-Quijada, P. Martinet, Multisensor-based predictive control for autonomous parking, IEEE Transactions on Robotics (2021) 1–17.

[12] S. Heshmati-alamdari, A. Eqtami, G. C. Karras, D. V. Dimarogonas, K. J. Kyriakopoulos, A self-triggered position based visual servoing model predictive control scheme for underwater robotic vehicles, Machines 8 (2) (2020) 33.

[13] S. Norouzi-Ghazbi, A. Mehrkish, M. M. Fallah, F. Janabi-Sharifi, Constrained visual predictive control of tendon-driven continuum robots, Robotics and Autonomous Systems 145 (2021) 103856.

[14] T. P. Nascimento, C. E. T. Dórea, L. M. G. Gonçalves, Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach, International Journal of Advanced Robotic Systems 15 (1) (2018) 1729881418760461.

[15] F. Allgower, R. Findeisen, Z. K. Nagy, et al., Nonlinear model predictive control: From theory to application, Journal-Chinese Institute Of Chemical Engineers 35 (3) (2004) 299–316.

[16] D. Q. Mayne, H. Michalska, Receding horizon control of nonlinear systems, in: Proceedings of the 27th IEEE Conference on Decision and Control, IEEE, 1988, pp. 464–465.

[17] S. a. Keerthi, E. G. Gilbert, Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations, Journal of optimization theory and applications 57 (2) (1988) 265–293.

[18] P. O. Scokaert, D. Q. Mayne, J. B. Rawlings, Suboptimal model predictive control (feasibility implies stability), IEEE Transactions on Automatic Control 44 (3) (1999) 648–654.

[19] H. Chen, F. Allgöwer, A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability, Automatica 34 (10) (1998) 1205–1217.

[20] D. Kouzoupis, R. Quirynen, B. Houska, M. Diehl, A block based aladin scheme for highly parallelizable direct optimal control, in: 2016 American Control Conference (ACC), IEEE, 2016, pp. 1124–1129.

[21] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, M. Diehl, A family of iterative gauss-newton shooting methods for nonlinear optimal control, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018, pp. 1–9.

[22] B. Plancher, S. Kuindersma, A performance analysis of parallel differential dynamic programming on a gpu., in: WAFR, 2018, pp. 656–672.

[23] D.-K. Phung, B. Hérissé, J. Marzat, S. Bertrand, Model predictive control for autonomous navigation using embedded graphics processing unit, IFAC-PapersOnLine 50 (1) (2017) 11883–11888.

[24] A. Durand-Petiteville, Navigation référencée multi-capteurs d'un robot mobile en environnement encombré, Ph.D. thesis, Université Paul Sabatier-Toulouse III (2012).

[25] D. Folio, V. Cadenat, Treating Image Loss by using the Vision/Motion Link: A Generic Framework, IN-TECH, 2008, Ch. 4.

[26] Djeridane, Sur la commandabilité des systèmes non linéaires à temps discret, Ph.D. thesis, Université Paul Sabatier (2004).

Biography

Dr. Adrien Durand-Petiteville is an assistant professor at UFPE, Recife, Brazil. He received his Ph.D. in control theory and robotics in 2012, from LAAS-CNRS, France. In 2013 he was a lecturer at QUT, Brisbane, Australia. From 2014 to 2018, he was a research fellow at the University of California Davis, USA. His research interests focus on mobile robotics, visual servoing, predictive control, embedded manipulation, and general-purpose GPU.

Dr. Viviane Cadenat is an associate professor at Paul Sabatier University (UPS) of Toulouse (France) since 2000. She received her PhD degree in robotics from UPS

in 1999 and her HDR in 2011. She now belongs to LAAS-CNRS. Her research interests are focused on the design of sensor-based control strategies to perform complex navigation or collaborative manipulation tasks in dynamic poorly known environments.