Advanced Visual Predictive Control Scheme For The Navigation Problem

Adrien Durand-Petiteville, Viviane Cadenat

Noname manuscript No.
(will be inserted by the editor)

Advanced Visual Predictive Control Scheme For The Navigation Problem

A. Durand-Petiteville · V. Cadenat

Received: date / Accepted: date

Abstract This work proposes a Visual Predictive Control (VPC) scheme adapted to the autonomous navigation problem among static obstacles. To do so, it is necessary to cope with several issues which by now limit the use of VPC in this context. Among them, we focus on the following ones: the need for precise prediction models to improve the task realization; the need for a long prediction horizon which is required to perform long range displacements and guarantee stability, but also results in a high computational burden and a more difficult implementation ; the possible optimization problem evolution at every iteration due to unexpected events (e.g., detection of new obstacles), which leads to non convex problems and therefore makes difficult its resolution.

The proposed VPC allows to tackle the above mentioned challenges. Based on a more accurate prediction model relying on an exact integration method, it integrates constraints to deal with actuator saturation, obstacle avoidance along the trajectory and stability. To deal with the two last mentioned challenges, the classical VPC scheme has been extended with two methods: the first one allowing to relax some constraints on the control inputs to reduce the computational burden; the second one for adequately refining the optimized trajectory to avoid local minima when the optimization problem evolves during the navigation. The proposed approach has been evaluated and compared to other VPC configurations. The obtained results show than it runs 60 times faster than classical configurations for similar performances.

A. Durand-Petiteville
Universidade Federal de Pernambuco UFPE, Departamento de Engenharia Mecânica
Av. da Arquitetura, 50740-550, Recife - PE, Brazil
E-mail: adrien.durandpetiteville@ufpe.br

V. Cadenat
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, UPS, LAAS, F-31400, Toulouse, France
E-mail: cadenat@laas.fr

1 Introduction

This paper deals with the autonomous navigation problem in an environment cluttered with obstacles. This problem, at the core of mobile robotics, has been tackled through a large variety of approaches which stretch on a wide spectrum depending on the existence (or not) of a map, the way it is used and the data it relies on. We thus first propose a brief review of the different classes of solutions existing in the literature and show the interest of using MPC (Model Predictive Control) and more specifically VPC (Visual Predictive Control) to address the navigation problem. Indeed, this latter technique allows to couple the computation of a feasible trajectory and the reactivity required to handle unexpected events during the mission. Next, we review the VPC approaches developed for different types of robotic platforms to list the addressed issues and the limitations of the proposed solutions. We finally present our VPC-based approach to the navigation problem dealing with challenges of both VPC and navigation. It allows us to exhibit the contributions at the core of the paper and to conclude with the paper outline.

1.1 The navigation methods

As previously mentioned, the literature proposes a large variety of approaches which range on a wide spectrum. On one side of the spectrum, a first class of methods, called map-based approaches relies on a map of the environment in order to plan a trajectory leading to the goal while dealing with constraints such as the presence of obstacles or the robot kinematics [1]. The challenges of these methods mainly lie in the construction of the map and in the robot localization. First, the map has to be accurate enough to allow a precise estimation of the robot pose. Second, it has to contain all the relevant information to lead to a trajectory dealing with the constraints, *e.g.,* the presence of obstacles. Thus, these methods appear relevant to perform long range navigation in the presence of constraints, but they seem to be less suitable to deal with unexpected obstacles, despite the existing extensions in the literature (see [2]).

On the other side of the spectrum, a second class of methods, called reactive or map-less approaches, does not use a map of the environment and benefits from local information. Most of these methods rely on sensor-based approaches, in the sense where the current pose and the goal are defined in terms of relevant measures in the sensor space [1]. Different sensors can be considered such as vision or laser rangefinders, leading to different spaces of expression [3]. The navigation is then performed by an output feedback controller to make the error between the current and reference sensory data vanish. Thus, no localization process with respect to a global frame or a map is required. The challenge in this class of methods lies in designing controllers able to guarantee the closed-loop stability while managing the different constraints required for the task execution (e.g., obstacle avoidance, actuator saturation, etc.). These methods appear relevant to navigate in the presence of unknown obstacles, but they seem to be less suitable to deal with numerous constraints.

Among the numerous methods lying in the middle of the spectrum, we focus on the Model Predictive Control (MPC) approach [4] [5]. With MPC schemes,

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 problem. Finally, at each iteration, a set of commands, and therefore a trajectory, is obtained via a solver minimizing the cost function while taking into account the constraints. It thus offers the possibility of dealing with numerous constraints in a natural way. MPC-based approaches usually strongly lean towards the map-based side of the spectrum. Indeed, the corresponding controllers are used to track a path or trajectory calculated before the navigation while dealing with some constraints [6] [7]. 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. However, if the task (and therefore the cost function) is defined in the sensor space, it becomes possible to couple some of the advantages of the two aforementioned classes: (i) the capacity to compute a trajectory dealing with constraints similarly to the map-based methods and (ii) the direct use of the desired and current measures in the closed-loop process, making unnecessary to rely on a map and improving the robot reactivity. This kind of approach appears to be relevant in our context where obstacles might unexpectedly appear on the robot trajectory during navigation. This idea is at the core of this paper.

1.2 The Visual Predictive Control (VPC)

In this work, it is proposed to use a VPC scheme [8], which is the fusion between Image-Based Visual Servoing (IBVS) [9] and Nonlinear Model Predictive Control (NMPC) [4] [5]. The choice of a camera as the main navigation sensor, and therefore of the image space as the sensor space, is motivated by the following reasons: (i) their sensing capacities are not limited to a plane, contrary to planar laser rangefinders; (ii) it is possible to rely on high-level features such as texture to identify a landmark; and (iii) their cost remains relatively low with respect to 3D LIDAR. It is thus possible to consider making the robot move through a complex environment where one or several embedded cameras identify a natural landmark used as a reference for the navigation task. Similarly to MPC, the VPC problem is formulated as the repeated solution of a finite horizon open-loop optimal control problem expressed in the image space, subject to system dynamics and input and state constraints. Thus, by defining the task in the sensor space, there is no need for an initial planning step nor a map. The trajectory described by the robot results from the optimization problem solved at each iteration and whose constraints are modified online based on the newly acquired data.

The interest for VPC-based controllers has grown over the last decade and the contributions are numerous. The following overview aims at analyzing and comparing some relevant works according to several key features to highlight their differences, shared characteristics and limitations. At first, it is worth mentioning that the VPC schemes have been used to control different robotic systems: a camera mounted on a robotic arm [10][11][12] [13][14], a flying camera [15] [16], a mobile robot [17] or a fixed-wing aerial vehicle [18]. In addition to this first difference, the mentioned works stand out on several aspects. Regarding the considered visual cues, VPC schemes generally rely on points to define the task (and therefore the cost function) in the image space, with the exception of [11] where image moments

are used to increase the robustness of the control algorithm. Concerning the prediction models, these latters are generally obtained by integrating a first order system based on the interaction matrix using the Euler's method, inducing a potential lack of accuracy. We may nonetheless mention the works by [12] and [14] where other leads have been followed. In the first one [12], the robotic arm is represented by a polytopic linear parameter-varying system in order to obtain a model independent of the visual feature depth. In the second one [14], the authors use a second order model allowing to integrate the visual features acceleration to obtain better and smoother 2D and 3D trajectories. Now, regarding the minimization problem resolution, most of the works rely on Quadratic Programming or Interior Point Algorithm. However, other solutions can be considered as proposed in [17] where a primal-dual neural network is used. Concerning the constraints, the majority of the mentioned works use them to bound the control inputs (actuator saturation) and the state variables (visual features visibility). However, more advanced uses can be considered. For example, [10] and [15] tighten the constraints to take into account the state uncertainties, allowing to obtain more robust controllers. In [18], the constraints are used to avoid obstacles in the vicinity of the robot, but their introduction makes the state and input sets of the minimization problem non-convex, thus increasing the difficulty of finding the global minima. The stability problem is also another example where the previously mentioned works may differ. Only one of them [15] partially addresses this problem by adding a zero terminal equality constraint, whereas it is a key issue when designing a controller. It is worth noting that the other works consider scenarii with relatively small camera displacements, short prediction horizons, and boundaries constraints leading to convex state and inputs sets. In such cases, VPC schemes without any explicit stability guarantee are usually sufficient to achieve the task. But they might be limited to perform a navigation task in a cluttered environment.

As shown by this overview, the coupling of IBVS with MPC requires to address several issues: (i) design of a sufficiently accurate and robust prediction model in the image space, (ii) design of constraints dealing efficiently with the system and environment specificities, (iii) design of an optimal problem guaranteeing the closed-loop stability, and (iv) selection or design of a numerical solver computing within an acceptable time a local or global solution for convex or non-convex problems. The above mentioned works take into account only one or two of these issues at a time, whereas the navigation problem in an environment cluttered with unexpected obstacles requires to address all of them. Our objective is to design a VPC scheme adapted to the navigation task.

1.3 Proposed solution and contributions

In this work, we consider a differential wheeled robot equipped with a camera used as the main sensor for navigation task, coupled with a laser rangefinder allowing to detect the obstacles in its vicinity. For this system, we propose: (i) an accurate prediction model benefiting from the particular mechanical structure of the considered robotic system that relies on the visual features depth; (ii) a set of constraints suitable to guarantee non collision along the trajectory, stability and actuator non saturation; (iii) a solution based on inputs constraints relaxation to

ease the terminal constraint satisfaction and guarantee the closed-loop stability while being computationally efficient and (iv) a solution to handle sub-optimality and local minima problems. The contributions lie in points (ii), (iii) and (iv). We now describe the proposed solution.

Let us start by the contribution stated in point (iii) from which follow the two other ones (ii) and (iv). We thus focus our efforts on the design of a VPC controller guaranteeing the closed-loop stability while being time-processing efficient. 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 [4]. For solutions relying on a finite prediction horizon, which is the case for all the above mentioned works, two main classes of approach are identified to guarantee stability. The first one enforces stability by adding a zero terminal equality constraint at the end of the prediction horizon [19, 20]. The second one relies on the quasi-infinite horizon method [21], 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. One more time, it is impossible to prove the existence of such a local control law without knowing the constraints in the terminal region. This is the reason why it is proposed here to guarantee the stability by adding a zero terminal equality constraint as in [15].

To this end, we propose to enlarge the size of the feasibility set, set for which there exists a trajectory reaching the goal while dealing with the constraints. This size depends on both the length of the prediction horizon and the boundaries of the control inputs. Large prediction horizons leading to large computational times, it seems more relevant to act on the control input boundaries than on the prediction horizon. To do so, one proposes to define two sets of constraints for the control inputs. The first ones correspond to the actual boundaries of the system and are applied on the first part of the prediction horizon, whereas the second ones are relaxed and applied to the rest of the prediction horizon. Thus, the command applied to the robot, which is in the first part of the prediction horizon, respects the actuators limits, while the feasible region is enlarged by the use of relaxed boundaries. It is worth mentioning that this solution makes sense for the navigation problem where the camera and laser ranges are limited. Indeed, it is important to consider actual constraints close to the robot where the environment is correctly perceived. On the contrary, far from it, it may be envisioned to consider relaxed ones as the image and the laser data are not available yet.

However, in order to preserve an efficient and safe navigation system while relaxing the control input boundaries, it is required to deal with two issues leading to contributions (ii) and (iv). First, the constraints applied to the system, *e.g.,* to avoid collisions with obstacles, have to be checked along the predicted trajectories and not only for the predicted states as it is usually done. Indeed, due to possible large commands, the predicted trajectory between two states can be sufficiently

large to pass through an obstacle without violating the state-centered collision constraints. Next, one has to deal with the sub-optimality of the computed solution. Indeed, predictive control schemes of complex systems usually rely on numerical solvers computing a local solution. It leads to a sub-optimal trajectory and in the worst case scenario, the first command might be null or very small. Since traditionally only the first command is applied, this means that the robot would be stopped, leading to a navigation failure. It will be shown in the following that the insertion of relaxed constraints increases the chance of navigation failures due to sub-optimal trajectories. Thus, in addition to the insertion of relaxed constraints, one presents in this work two other contributions: (i) a constraint to avoid collisions along the robot trajectory, and (ii) a method based on equivalent control vectors [22] to refine the obtained trajectory and reduce the chances of navigation failure.

The paper is organized as follows. First, the models of the different parts of the robotic system are presented. Next, the VPC scheme and the set of constraints to deal with obstacles and actuator boundaries are introduced. One then describes the equivalent control vector method developed for the considered robotic system. Next one introduces our two-steps approach to refine the computed trajectory and thus prevent the navigation task from failing. Finally, one presents results obtained with a 2D simulator where we consider ideal sensors (RGBD camera and laser rangefinder) in the presence of static obstacles. These results aim at illustrating our contributions and highlight the relevance of the proposed approach.

2 Preliminaries

This section is intended to introduce the elements that are required to build our VPC scheme, namely: the robot model, the visual features and their prediction model.

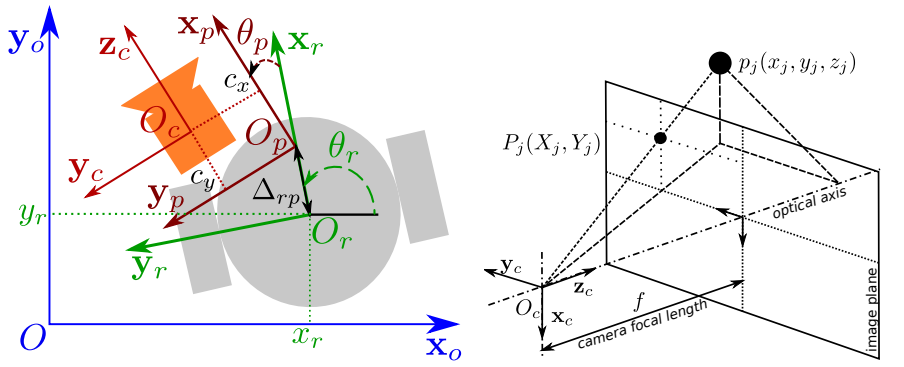
2.1 Robot Modeling

In this paper, we aim at controlling a camera embedded on the pan-platform of a differential robot using VPC. To model the robotic system, the following frames are introduced as shown in Figure 1(a): $\mathbf{F}_o(O, \mathbf{x}_o, \mathbf{y}_o, \mathbf{z}_o)$ as the world frame, $\mathbf{F}_r(O_r, \mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r)$ as the robot frame, $\mathbf{F}_p(O_p, \mathbf{x}_p, \mathbf{y}_p, \mathbf{z}_p)$ as the pan-platform frame, and $\mathbf{F}_c(O_c, \mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c)$ as the camera frame. The considered camera is modeled using the perspective camera model and its focal length is denoted by f (see Figure 1(b)). Finally, the camera state can be defined as follows:

$$\chi_c = [x_c, y_c, \theta_c]^T \tag{1}$$

where x_c and y_c are the coordinates of the point O_c in $\mathbf{F}_o$, while $\theta_c = \theta_r + \theta_p$, θ_r and θ_p being respectively the direction of the robot with respect to $\mathbf{x}_o$, and the orientation of the pan-platform with respect to $\mathbf{x}_r$.

As the camera is mounted on the pan-platform of a differential drive robot, its motion is controlled via the robotic system, and its number of degrees of freedom (DOF) is reduced to three (two translation motions along $\mathbf{y_c}$ and $\mathbf{z_c}$, one rotation



(a) Differential drive robot model (b) Perspective Camera Model

Fig. 1: System model

motion around $\mathbf{x_c}$) as shown by Figure 1(a)). The camera kinematic screw $\nu_\mathbf{c}$ can be written as follows [23]:

$$\nu_\mathbf{c} = \begin{bmatrix} -\sin(\theta_p) & \Delta_{rp}\cos(\theta_p) + c_x & c_x \\ \cos(\theta_p) & \Delta_{rp}\sin(\theta_p) - c_y & -c_y \\ 0 & -1 & -1 \end{bmatrix} \mathbf{Q} \tag{2}$$

where c_x and c_y are the coordinates of O_c along axes $\mathbf{x}_p$ and $\mathbf{y}_p$, Δ_{rp} being the distance between O_r and O_p (see Figure 1(a)). $\mathbf{Q} = [\upsilon, \omega_r, \omega_p]^T$ denotes the control input vector of the system 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 $\mathbf{F}_r$. Finally, using Equation (1), it is also possible to establish the camera kinematic model as follows:

$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta}_c \end{bmatrix} = \begin{bmatrix} \upsilon\cos(\theta_r) - \omega_r\Delta_{rp}\sin(\theta_r) \\ \upsilon\sin(\theta_r) + \omega_r\Delta_{rp}\cos(\theta_r) \\ \omega_r + \omega_p \end{bmatrix} \tag{3}$$

2.2 The Visual Features and their prediction model

As explained above, the navigation task consists in making the robot move with respect to a reference landmark tracked by the camera. In the case of visual servoing and thus in VPC, the task is defined in the image space [9]. In this work, as in most of the reviewed works in Section 1, we consider that the reference landmark can be characterized by N_v interest points, these latter being extracted by a dedicated image processing algorithm such as [24] when the landmark is made of an AprilTag. Let us denote by p_j one of the N_v aforementioned 3D interest points ($j \in [1, ..., N_v]$). Its coordinates in $\mathbf{F}_c$ are defined by (x_j, y_j, z_j). Following the perspective camera model, its projection is given by P_j whose coordinates are $S_j = (X_j, Y_j)$ in the image plane. As a consequence, the visual features vector is given by the following $2N_v$ dimensional vector $\mathbf{S}$:

$$\mathbf{S} = [X_1, Y_1, ..., X_{N_v}, Y_{N_v}]^T \tag{4}$$

For the given robotic system and for a sole point P_j, the time derivative of S_j is given by [9]:

$$\dot{\mathbf{S}}_j = \mathbf{L}_j \mathbf{J} \mathbf{Q} \tag{5}$$

where $\mathbf{J}$ is the robot Jacobian (see equation (2)) and $\mathbf{L}_j$ is the so-called interaction matrix associated to the considered visual features $\mathbf{S}_j$. For pointwise cues and considering that the camera has only three DOF, its expression reduces to [9]:

$$\mathbf{L}_j = \begin{bmatrix} \frac{X_j}{z_j} & \frac{X_j Y_j}{f} & -(f + \frac{X_j^2}{f}) \\ \frac{Y_j}{z_j} & (f + \frac{X_j^2}{f}) & \frac{-X_j Y_j}{f} \end{bmatrix} \tag{6}$$

Now, let us derive the visual features prediction model required to set up the VPC scheme. The robot is a sampled system whose inputs evolve at each instant $t_k = kT_s$, where T_s is the sampling time. Let us consider two different instants t_k and t_{k+1}. Assuming that the inputs $\mathbf{Q}(t_k)$ remain constant during this time interval, it is possible to solve Equation (5) between these two instants and obtain an analytical prediction of the visual feature $\mathbf{S}_j(t_{k+1})$ knowing $\mathbf{S}_j(t_k)$ and $\mathbf{Q}(t_k)$. We obtain (see [25] for more details):

$$X_j(t_{k+1}) = \frac{z_j(t_k) X_j(t_k)}{z_j(t_{k+1})} \tag{7}$$

$$\begin{aligned} Y_j(t_{k+1}) = \tfrac{f}{z_j(t_{k+1})} \Big\{ & C_1 \cos(A) - C_2 \sin(A) + \Delta_{rp} \sin(\theta_p(t_{k+1})) \\ & + \tfrac{\upsilon(t_k)}{\omega_r(t_k)} \cos(\theta_p(t_{k+1})) - c_y \Big\} \end{aligned} \tag{8}$$

$$z_j(t_{k+1}) = C_1 \sin(A) + C_2 \cos(A) - \Delta_{rp} \cos(\theta_p(t_{k+1})) + \frac{\upsilon(t_k)}{\omega_r(t_k)} \sin(\theta_p(t_{k+1})) - c_x \tag{9}$$

where:

$$A = \Big(\omega_r(t_k) + \omega_p(t_k)\Big) T_s$$

$$C_1 = \frac{Y_j(t_k) z_j(t_k)}{f} - \Delta_{rp} \sin(\theta_p(t_k)) - \frac{\upsilon(t_k)}{\omega_r(t_k)} \cos(\theta_p(t_k)) + c_y$$

$$C_2 = z_j(t_k) + \Delta_{rp} \cos(\theta_p(t_k)) - \frac{\upsilon(t_k)}{\omega_r(t_k)} \sin(\theta_p(t_k)) + c_x$$

Equations (7), (8) and (9) are analytic expressions that can be used to predict exactly the coordinates of the visual features. The proposed solution benefits from the robot particular mechanical structure to provide an exact integration of the differential equation (5). It thus offers a more accurate solution than solving this latter numerically using Euler's method as it is classically done [8] (see section 1). Moreover, it does not require any advanced/complex operation, maintaining the computational effort at a low cost. The first VPC challenge for navigation mentioned in the introduction is thus fulfilled.

3 Visual Predictive Control

In this section, we first recall the VPC framework and the main parameters impacting the controller behavior. Then, we present three constraints that have to be taken into account in the VPC problem in order to guarantee the convergence of the closed-loop system and the safety of the robot during a navigation task.

3.1 The VPC Scheme

As already mentioned, VPC is the result of coupling NMPC with IBVS. It thus shares characteristics from these two particular control techniques. As NMPC, it is the solution of a constrained optimal problem. More precisely, it consists in finding an optimal control sequence $\overline{\mathbf{Q}}^*(k)$ that minimizes at instant t_k a cost function J_{N_p} over a N_p steps prediction horizon under a set of user-defined constraints $\mathbf{C}(\overline{\mathbf{Q}}^*(k))$. The obtained optimal control sequence $\overline{\mathbf{Q}}^*(k) = [\mathbf{Q}^*(k|k), \mathbf{Q}^*(k+1|k), ..., \mathbf{Q}^*(k+N_p-1|k)]$ is a N_p dimensional vector where $\mathbf{Q}^*(k+1|k)$ is the $k+1^{th}$ optimal control input calculated at the k^{th} iteration. Moreover, we define N_c as the control horizon. It means that the N_c^{th} first predictions of the N_p long prediction horizon are computed using independent control inputs, while the remaining ones are all obtained using a unique control input equals to the N_c^{th} element of $\overline{\mathbf{Q}}^*(k)$ As IBVS, the cost function is defined in the image space instead of the state space as it is classically done in NMPC. It expresses as the sum of the quadratic error between the visual feature coordinates vector $\hat{\mathbf{S}}(k)$ predicted over the horizon N_p and the desired ones $\mathbf{S}^*$[1]. The optimal problem is then defined as follows:

$$\overline{\mathbf{Q}}^*(k) = \min_{\overline{\mathbf{Q}}(k)} \left(J_{N_p}(\mathbf{S}(k), \overline{\mathbf{Q}}(k)) \right) \tag{10}$$

with

$$J_{N_p}(\mathbf{S}(k), \overline{\mathbf{Q}}(k)) = \sum_{p=k+1}^{k+N_p} [\hat{\mathbf{S}}(p) - \mathbf{S}^*]^T [\hat{\mathbf{S}}(p) - \mathbf{S}^*] \tag{11}$$

subject to

$$\hat{\mathbf{S}}(l+1) = f(\hat{\mathbf{S}}(l), \mathbf{Q}(l)), \text{ with } l \in [k, ..., k+N_p-1] \tag{12a}$$

$$\hat{\mathbf{S}}(k) = \mathbf{S}(k) \tag{12b}$$

$$C(\overline{\mathbf{Q}}(k)) \leq 0 \tag{12c}$$

where $\overline{\mathbf{Q}}(k) = [\mathbf{Q}(k), ..., \mathbf{Q}(k+N_p-1)]$. Equation (12a) corresponds to the prediction model and is instanciated by Relations (7), (8) and (9). Equation (12b) guarantees that the predicted visual features at instant t_k are given by the last measure provided by the image processing algorithm. Finally, Equation (12c) gathers all the constraints which must be taken into account to successfully perform the navigation task. These latters are described in the following sections.

Remark 1: VPC generally works as follows. The minimization problem (10) is first solved, leading to the optimal sequence $\overline{\mathbf{Q}}^*(k)$. Usually, only its first term is sent to the system. Then, this process is repeated until the task is achieved.

Remark 2: Numerical solvers require to define an initial value for the vector to optimize, called initial guess. Its choice can have a strong impact on the obtained solution, especially for solvers providing local minima. Indeed, a smart initial guess allows not only to converge faster towards a solution but also to influence the local minimum which will be reached. In this work, the results of the previous optimization are used to build the initial values of the current one. To do so, the first command of $\overline{\mathbf{Q}}^*(k)$, *i.e.,* $\mathbf{Q}^*(k|k)$, is discarded, the remaining commands are

[1] They correspond to the visual features obtained when the task is performed, i.e., when the camera is correctly positioned with respect to the landmark of interest.

shifted by one index lower, and the last command is set up to zero. Thus, if the robot has perfectly achieved the motion due to the first command, the trajectory used as an initial guess is the same as the one computed at the previous step minus the last piece of motion of the robot. This solution increases the chances of finding a new solution close to the previous one, thus reducing possible oscillations.

3.2 The Zero Terminal Equality Constraint

As previously mentioned, the stability of our VPC scheme is achieved by introducing a zero terminal equality constraint. This constraint is expressed as the error between the value of the predicted visual features $\hat{\mathbf{S}}(k+N_p)$ obtained at the end of the prediction horizon, and the desired ones $\mathbf{S}^*$. We get:

$$||\hat{\mathbf{S}}(k+N_p) - \mathbf{S}^*|| = 0 \tag{13}$$

As this strict equality constraint is almost impossible to achieve, it is proposed to replace it by the following inequality constraint:

$$||\hat{\mathbf{S}}(k+N_p) - \mathbf{S}^*|| - \delta_{tc} \leq 0 \tag{14}$$

where δ_{tc} is a user defined threshold. It must be small enough to influence the optimization process as the equality would do, while offering an efficient implementation of the constraint. If constraint (14) is fulfilled at each iteration, there exists a trajectory leading from the current state to the desired one, thus guaranteeing the recursive feasibility [4]. On the contrary, if the solver cannot find a solution to this constrained problem, this means that the number of predictions is too small and/or the constraints on the control inputs are too restrictive to reach the goal [5]. This problem will be tackled in Section 4.

Remark 3: Although it is a well known solution to guarantee the stability of MPC controllers, the terminal constraint is not used in most of the works mentioned in the literature review proposed in Section 1. One of the reason could be the need for a large prediction horizon increasing the time required to minimize the optimal problem. Instead, authors usually weight the last predicted value based on the distance to the desired one. However, as shown in Section 1, this approach is not adapted to perform a navigation task, which is why we have tackled this problem in this work.

Remark 4: The terminal constraint guarantees the task achievement under the assumption that the predicted states are sufficiently accurate. If they are strongly erroneous, the camera might not converge towards its desired pose, leading to a task failure. In this context, the interest of using the above mentioned exact prediction model instead of a solution based on Euler's integration method appears clearly.

3.3 The Input Constraints

The control input constraints are defined by boundaries to avoid actuator saturation. They allow to ensure that the obtained optimal control sequence respects the

robot physical bounds. However, for a given number of prediction steps N_p, they also limit the size of the feasibility set. We propose to deal with this problem by splitting the set of control input constraints into two subsets. In the first one, the boundaries, called '*tight boundaries*', are defined by the actuators actual limits. In the second one, they lose their physical sense and are increased to enlarge the feasibility set. In this case, they are called '*relaxed or extended boundaries*'. Following this reasoning, we propose to define the input constraints as shown below:

$$\begin{aligned} &\begin{bmatrix} \mathbf{Q}(i) - \mathbf{Q}_{u|t} \\ \mathbf{Q}_{l|t} - \mathbf{Q}(i) \end{bmatrix} \leq 0, \text{ if } 1 \leq i \leq N_c - N_r \\ &\begin{bmatrix} \mathbf{Q}(i) - \mathbf{Q}_{u|r} \\ \mathbf{Q}_{l|r} - \mathbf{Q}(i) \end{bmatrix} \leq 0, \text{ if } N_c - N_r < i \leq N_c \end{aligned} \tag{15}$$

where $i \in [1, ..., N_c]$, N_r is the number of prediction steps with relaxed boundaries, $\mathbf{Q}_{l|t}$ and $\mathbf{Q}_{u|t}$ are respectively the lower and upper tight boundaries corresponding to the actuators limits, and $\mathbf{Q}_{l|r}$ and $\mathbf{Q}_{u|r}$ are respectively the lower and upper relaxed boundaries. Thus, the command applied to the robot, which belongs to the first part of the prediction horizon, respects the actuators boundaries, while the extended boundaries allow to enlarge the feasible set[2].

Remark 5: Coupling the terminal constraint with the extended boundaries allows guaranteeing the closed-loop stability by computing a trajectory leading to the desired pose while dealing with constraints. Even if the calculated trajectory cannot be followed by the robot, due to the control inputs outside of the actuators boundaries, it proves the existence of a path towards the goal. To make possible the tracking of the trajectory, it would be sufficient to break down the pieces of trajectory obtained with the relaxed constraints, into smaller pieces respecting the actuators boundaries. It would naturally require to increase the number of control steps.

3.4 The Obstacle Avoidance Constraints

To perform a safe navigation, it is necessary to ensure non collision. Constraints can be naturally used to fulfill this purpose. In most of applications, it is proposed to guarantee a minimal distance between one or several points of the obstacles and the centroid of the robot for each predicted pose. However, this solution is not sufficient in our case because the use of relaxed input boundaries may lead to large displacements. The risk of collision must then be checked not only for the predicted poses, but also all along the trajectories.

To this end, it is proposed to characterize the pieces of trajectory performed by the robot. First, let us recall that the control inputs are assumed to be constant during one sampling period. From this, it follows that, for the given robotic system, the pieces of trajectory realized between two consecutive poses are either a line segment when $\omega_r = 0$ or an arc of circle when $\omega_r \neq 0$. To verify whether a collision may occur or not, it thus suffices to check the distance between the laser points and the corresponding segment or arc of circle which respectively characterize the

[2] At first glance, it may appear interesting to set the relaxed constraints to infinity. However, this solution cannot be applied because most of the solvers require finite boundaries.

current obstacle and robot trajectory. Thus, for all N_o points C_m representing the obstacles, with $m \in [1, ..., N_o]$, the set of constraints can be written as:

$$\delta_c - \Delta(C_m, \hat{\chi}(n)|\hat{\chi}(n+1)) \leq 0 \tag{16}$$

where $n \in [1, ..., N_p - 1]$ and $\Delta(C_m, \hat{\chi}(n)|\hat{\chi}(n+1))$ is the shortest distance between C_m and the piece of trajectory between two consecutive predicted poses $\hat{\chi}(n)$ and $\hat{\chi}(n+1)$. Finally, δ_c is a user-defined distance preventing collisions. At this step, the second and third VPC challenges for navigation mentioned in the introduction are fulfilled, leading to contributions (ii) and (iii). In the next section, we will deal with the last one which is related to the solution sub-optimality. It will be the last contribution of this paper.

4 Refinement of the Sub-Optimal Solution

In this section, we first describe the sub-optimality problem that might occur when relying on a solver providing a local solution and we mention the issues that have to be taken into account in order to achieve the navigation task. Next, we present the equivalent control vector method which allows to merge sequences of commands. Finally, one introduces the Trajectory Refinement Algorithm, a two steps methods aiming at avoiding local minima.

4.1 The Sub-Optimal Solution Problem

After having stated the optimal control problem, it is necessary to compute a solution at each iteration, which is usually achieved by a numerical solver. For complex problems (nonlinear cost functions and constraints, non convex sets, large control input vectors), it is challenging to compute the global solution, and usually the solver only provides a local solution, *i.e.,* a sub-optimal solution.

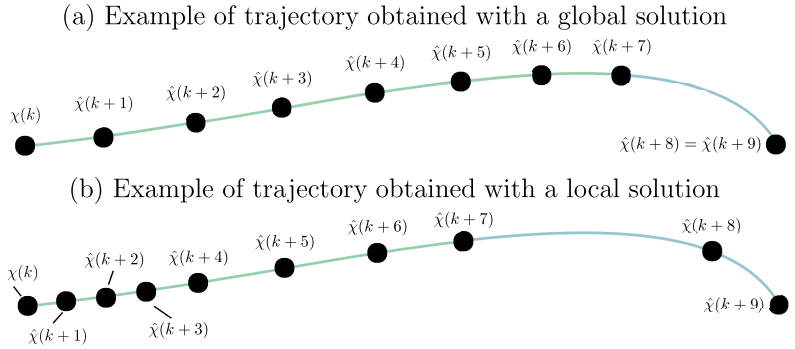


Fig. 2: Example of trajectories with $N_c = 9$ and $N_r = 2$. (a) Global solution: the first extended command allows reaching the goal and the second one is null. (b) Local solution: the two extended commands are used to reach the goal.

To illustrate the issues that might occur when relying on a sub-optimal solution, we use the regulation problem. When a global solution is computed, all the predicted states are as close as possible to the desired values while dealing with the constraints. Thus, in the case where there are more prediction steps than required to reach the goal, the unnecessary steps are the last ones and have null command values. For the proposed approach, it means that the global solution minimizes the use of the last steps with the relaxed constraints. They only allow completing the trajectory and are null when unnecessary (see Figure 2). When only a local solution can be computed, there is no more guarantee the pieces of trajectory obtained with the relaxed constraints are minimal (see Figure 2). In the worst case scenario, the first control inputs are null and the trajectory is only composed of the steps with the extended control inputs. In such a case, either the closely similar ones, the first command is null, or quasi null, and the robot stops navigating.

In [26], it is proposed to deal with the sub-optimality problem by adding a constraint forcing the cost function to decrease. However, this approach cannot be used for the navigation problem. Indeed, in this context, the optimization problem can be modified over the navigation. For example, when a new piece of obstacle or an obstacle is discovered, the obstacle constraints are updated, modifying the optimization problem. It might then be necessary to let the cost function increase in order to compute a trajectory leading to the goal while dealing with the constraints. Thus, to take full advantage of the introduction of the relaxed constraints while dealing with such issues, one proposes to refine the obtained trajectory. The objective is to conserve the overall shape of the trajectory while modifying the sequence of control inputs to avoid local minima. To do so, we rely on the equivalent control vector method presented in the following section.

4.2 Equivalent Control Vector

The equivalent control vector method aims at computing the smallest sequence of control inputs $\tilde{\mathbf{Q}}_{t_1|t_2}$ connecting two states of a system at instants t_1 and t_2, where $t_1 < t_2$. It can be used to substitute a sequence of commands $\overline{\mathbf{Q}} = [\mathbf{Q}(t_1), \mathbf{Q}(t_1 + T_s), ..., \mathbf{Q}(t_1 + N * T_s)]$, with $N \in \mathbb{N}^{*+}$, as it is shown in Figure 3. In the case of a VPC scheme, it is necessary to calculate the smallest sequence of commands connecting two images. To do so, a first solution presented in [23] calculates the equivalent control vector between two states of the whole robotic system, *i.e.,* the mobile base and the pan-platform. Because of the non-holonomic constraint on the mobile base, a two steps solution is obtained. In order to obtain a simpler solution, we consider in this work the camera state. Indeed, in the following, we first prove that the camera can be controlled with a unique command by studying its controllability. Next, we present the steps to obtain the camera equivalent control vector.

4.2.1 Controllability

To determine the smallest number of commands necessary to connect two camera states, we need to calculate the controllability of the following nonlinear discrete

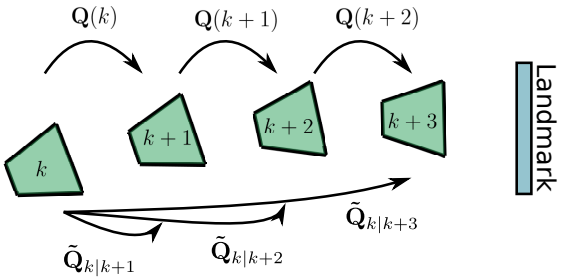


Fig. 3: Example of equivalent control vectors for a camera

system:

$$\chi_c(k) = g(\chi_c(k-1), \mathbf{Q}(k-1)) \tag{17}$$

where $g(\chi_c(k-1), \mathbf{Q}(k-1))$ is obtained by analytically solving Equation ((3)) between instants t_{k-1} and t_k. Its expression is given by:

$$x_c(k) = x_c(k-1) + \frac{2v(k-1)}{\omega_r(k-1)} \sin(\eta_1)\cos(\eta_2) + 2\Delta_{rp}\sin(\eta_1)\sin(\eta_2) \tag{18}$$

$$y_c(k) = y_c(k-1) - \frac{2v(k-1)}{\omega_r(k-1)} \sin(\eta_1)\sin(\eta_2) + 2\Delta_{rp}\sin(\eta_1)\cos(\eta_2) \tag{19}$$

$$\theta_c(k) = \theta_c(k-1) + (\omega_r(k-1) + \omega_p(k-1))T_s \tag{20}$$

with $\eta_1 = \frac{\omega_r(k-1)T_s}{2}$ and $\eta_2 = \frac{2\theta(k-1)+\omega_r(k-1)T_s}{2}$ and when $\omega_r \neq 0$ (the problem is straightforward if $\omega_r = 0$). According to [27], such a system is controllable in p steps if the following matrix P is full rank.

$$P = \begin{bmatrix} \frac{\partial g(\chi_c(p-1),\mathbf{Q}(p-1))}{\partial \mathbf{Q}(p-1)} & & \\ \frac{\partial g(\chi_c(p-1),\mathbf{Q}(p-1))}{\partial \chi_c(p-1)} & \frac{\partial g(\chi_c(p-2),\mathbf{Q}(p-2))}{\partial \mathbf{Q}(p-2)} & \\ & \dots & \\ \frac{\partial g(\chi_c(p-1),\mathbf{Q}(p-1))}{\partial \chi_c(p-1)} & \dots & \frac{\partial g(\chi_c(1),\mathbf{Q}(1))}{\partial \chi_c(1)} \frac{\partial g(\chi_c(0),\mathbf{Q}(0))}{\partial \mathbf{Q}(0)} \end{bmatrix}^T \tag{21}$$

For a controllability in one step, *i.e.*, $p = 1$, the P matrix becomes:

$$P = \frac{\partial g(\chi_c(0), \mathbf{Q}(0))}{\partial \mathbf{Q}(0)} \tag{22}$$

Using Equations (18), (19) and (20) in Equation (22), one obtains:

$$P = \begin{bmatrix} \frac{2}{\omega(0)}\sin(\eta_1)\cos(\eta_2) & \zeta_1 & 0 \\ \frac{2}{\omega(0)}\sin(\eta_1)\sin(\eta_2) & \zeta_2 & 0 \\ 0 & T_s & T_s \end{bmatrix} \tag{23}$$

where

$$\zeta_1 = -\frac{2v(0)}{\omega_r^2(0)}\sin(\eta_1)\cos(\eta_2) + \frac{v(0)}{\omega_r(0)}T_s\cos(\eta_1+\eta_2) - \Delta_{rp}T_s\sin(\eta_1+\eta_2)$$

and

$$\zeta_2 = -\frac{2\upsilon(0)}{\omega_r^2(0)}\sin(\eta_1)\sin(\eta_2) + \frac{\upsilon(0)}{\omega_r(0)}T_s\sin(\eta_1+\eta_2) + \Delta_{rp}T_s\cos(\eta_1+\eta_2)$$

The camera is controllable in one step, if the matrix given in Equation (23) is full rank. To determine if the P matrix is full rank, one computes its determinant:

$$det(P) = \frac{\upsilon(0)}{\omega_r(0)}\sin(\eta_1)T_s + \Delta_{rp}\cos(\eta_1)T_s \tag{24}$$

The determinant is non-null, and thus the P matrix is full rank, for $\tan(\eta_1) \neq -\frac{\Delta_{rp}\omega(0)}{\upsilon(0)}$[3]. Thus, it exists a constant control input vector $\tilde{\mathbf{Q}}_{t_1|t_2}$, named equivalent control vector, allowing to reach in one step any camera state $\chi_c(t_2)$ from $\chi_c(t_1)$, with $t_2 > t_1$. This equivalent control vector allows to link two images without the need of intermediate ones as it is illustrated in Fig. 3.

4.2.2 Computation of the equivalent speeds

Now that it has been determined that the camera is controllable with one step, one focuses on the computation of the equivalent control vector $\tilde{\mathbf{Q}} = [\tilde{\upsilon}, \tilde{\omega}_r, \tilde{\omega}_p]^T$.

– Computation of $\tilde{\omega}_r$:
In order to compute $\tilde{\omega}_r$, one first defines:

$$\Delta_{x_c} = x_c(t_2) - x_c(t_1) \qquad\qquad \Delta_{y_c} = y_c(t_2) - y_c(t_1)$$

It is then possible to re-write Equations (18) and (19) such as:

$$\Delta_{x_c} + 2\Delta_{rp}\sin(\eta_1)\sin(\eta_2) = 2\frac{\tilde{\upsilon}}{\tilde{\omega}_r}\sin(\eta_1)\cos(\eta_2) \tag{25}$$

$$\Delta_{y_c} - 2\Delta_{rp}\sin(\eta_1)\cos(\eta_2) = 2\frac{\tilde{\upsilon}}{\tilde{\omega}_r}\sin(\eta_1)\sin(\eta_2) \tag{26}$$

Multiplying Equations (25) and (26) by $\sin(\eta_2)$ and $\cos(\eta_2)$ respectively, and subtracting the results, one obtains :

$$\Delta_{x_c}\sin(\eta_2) - \Delta_{y_c}\cos(\eta_2) + 2\Delta_{rp}\sin(\eta_1) = 0 \tag{27}$$

$\tilde{\omega}_r$ can be deduced from Equation (27) using classical relationships of trigonometry. One finally obtains:

$$\tilde{\omega}_r = \frac{2}{T_s}\arctan\left(\frac{-\Delta_{x_c}\sin(\theta(t_1)) + \Delta_{y_c}\cos(\theta(t_1))}{2\Delta_{rp} + \Delta_{x_c}\cos(\theta(t_1)) + \Delta_{y_c}\sin(\theta(t_1))}\right) \tag{28}$$

[3] The determinant is null when $\upsilon(0)$ and $\Delta_{rp} = 0$. However, this case is not considered as $\Delta_{rp} \neq 0$ is a necessary condition to perform visual servoing [22]

– Computation of $\tilde{v}$:

One now computes the equivalent linear velocity of the mobile base $\tilde{v}$ depending on its equivalent angular velocity. To do so, Equations (18) and (19) are squared and then summed. One obtains:

$$\begin{aligned}\Delta_{x_c}^2 + \Delta_{y_c}^2 = \frac{\tilde{v}^2}{\tilde{\omega}_r^2}\Big(1 - 2\sin(\theta(t_1) + \tilde{\omega}_r T_s)\sin(\theta(t_1)) \\ -2\cos(\theta(t_1) + \tilde{\omega}_r T_s)\cos(\theta(t_1)) + 1\Big) \\ +\Delta_{rp}^2(1 - 2\cos(\theta(t_1) + \tilde{\omega}_r T_s)\cos(\theta(t_1)) - 2\sin(\theta(t_1) + \tilde{\omega}_r T_s)\sin(\theta(t_1)) + 1)\end{aligned} \tag{29}$$

$$\Delta_{x_c}^2 + \Delta_{y_c}^2 = 4\sin^2(\eta_1)\left(\frac{\tilde{v}^2}{\tilde{\omega}_r^2} + \Delta_{rp}^2\right) \tag{30}$$

Finally, one obtains the following equation for the equivalent linear velocity:

$$\tilde{v} = \sqrt{\tilde{\omega}_r^2\left(\frac{\Delta_{x_c}^2 + \Delta_{y_c}^2}{4\sin^2(\eta_1)} - \Delta_{rp}^2\right)} \tag{31}$$

– Computation of $\tilde{\omega}_p$:

Using Equation (20), one directly obtains the angular velocity of the pan-platform such as:

$$\tilde{\omega}_p = \frac{1}{T_s}(\theta_p(t_2) - \theta_p(t_1) + \theta_r(t_2) - \theta_r(t_1) + \tilde{\omega}_r T_s) \tag{32}$$

4.3 Trajectory Refinement

In this work, it is proposed to refine at each iteration the solution $\overline{\mathbf{Q}}^*(k)$ of the optimization problem to prevent the robot from stopping before the goal. To do so, we propose a two steps method relying on the equivalent control vector method. First, one merges the commands of the computed sequence that are too small. Thus, it is guaranteed that the first command is non-null, preventing the robot from stopping before achieving the navigation task. Next, in order to prevent null-commands in the middle of the trajectory, one extracts new commands from the large pieces of trajectory obtained with the relaxed commands. Thus we provide better initial conditions for the next optimization process. One now presents these two steps with greater details.

4.3.1 Commands Merging

In order to merge commands, one first computes the N_p equivalent control vectors $\tilde{\mathbf{Q}}_{k|k+i}$ between the initial camera pose at instant t_k and the N_p predicted ones at the predicted instants $t_k + iT_s$, with $i \in [1, ..., N_p]$. Next, one needs to find among the $\tilde{\mathbf{Q}}_{k|k+i}$ respecting the boundaries $\mathbf{Q}_{l|t}$ and $\mathbf{Q}_{u|t}$ and the collision constraints, the one providing the largest piece of trajectory. It corresponds to the $\tilde{\mathbf{Q}}_{k|k+i}$ with the highest value for i among the ones dealing with the constraints. One denotes

the highest value of i as N_m and one defines $\mathbf{Q}_M = \tilde{\mathbf{Q}}_{k|k+N_m}$. When $N_m > 1$, it means that $\mathbf{Q}_M$ merges the N_m first control inputs $\mathbf{Q}(k), ..., \mathbf{Q}(k+N_m-1)$.

Now that a merging command dealing with the constraints has been computed, it has to be included in the sequence $\overline{\mathbf{Q}}^*(k)$. Let first define N_s as the number of tight commands that are conserved in the merging process and N_z as the number of commands that disappear and need to be replaced.

$$\begin{aligned} N_s &= N_c - N_r - N_m \\ N_z &= N_m - 1 \end{aligned} \tag{33}$$

To include the merging command $\mathbf{Q}_M$, the control sequence $\overline{\mathbf{Q}}^*(k)$ is modified as follows (see figure 4.b for an example):

- The first element $\overline{\mathbf{Q}}^*(1)$ is equal to $\mathbf{Q}_M$.
- The N_s following elements $[\overline{\mathbf{Q}}^*(2), ..., \overline{\mathbf{Q}}^*(2+N_s-1)]$ are equal to $[\mathbf{Q}(k+N_m), ..., \mathbf{Q}(k+N_m+N_s-1)]$.
- The N_z following elements $[\overline{\mathbf{Q}}^*(2+N_s), ..., \overline{\mathbf{Q}}^*(1+N_s+N_z)]$ are null.
- The last N_r elements are not modified.

4.3.2 Extraction of New Commands

The merging command being calculated and included, it is now proposed to extract tight commands from the relaxed ones to replace the null ones introduced in the previous step. The approach consists in extracting a piece of the trajectory obtained with the relaxed commands. One first defines two indices $u_n \in [N_c - N_r - N_z, ..., N_c - N_r]$ and $u_e = N_c - N_r + 1$ to respectively iterate over the null commands and the extended ones. Next, one defines a gain λ to extract from the extended command the longest piece of trajectory lying within the tight bounds. It is computed as follows:

$$\lambda = max\left(\frac{\tau_\upsilon}{|\upsilon(u_e)|}, \frac{\tau_{\omega_r}}{|\omega_r(u_e)|}, \frac{\tau_{\omega_p}}{|\omega_p(u_e)|}\right) \tag{34}$$

where τ_υ, τ_{ω_r}, and τ_{ω_p} are the upper boundaries on υ, ω_r, and ω_p. If $\upsilon(u_e)$, $\omega_r(u_e)$ and $\omega_p(u_e)$ are null or within the tight boundaries, then the index u_e is incremented by one to consider the next extended control inputs. λ being calculated, one obtains the new commands as follows:

$$\mathbf{Q}_E(u_n) = \lambda \mathbf{Q}(u_e) \tag{35}$$

Finally, after extracting a new tight command from an extended one, one updates the extended command to conserve the original trajectory. To do so, one computes the new state $\hat{\chi}_c(k+u_n)$ obtained with $\mathbf{Q}_E(u_n)$. It is then possible to compute the equivalent control vector $\tilde{\mathbf{Q}}_{k+u_n|k+u_e}$ between this new state and the end of the trajectory piece obtained with the extended command. This equivalent control vector is used as the updated extended control input (see figure 4.c).

The use of this method guarantees that the first command is non-null, which prevents the robot from stopping before reaching the goal. Moreover, the two steps are repeated to process the whole control sequence. At each new passage the control sequence is updated by removing its first element from the merging process

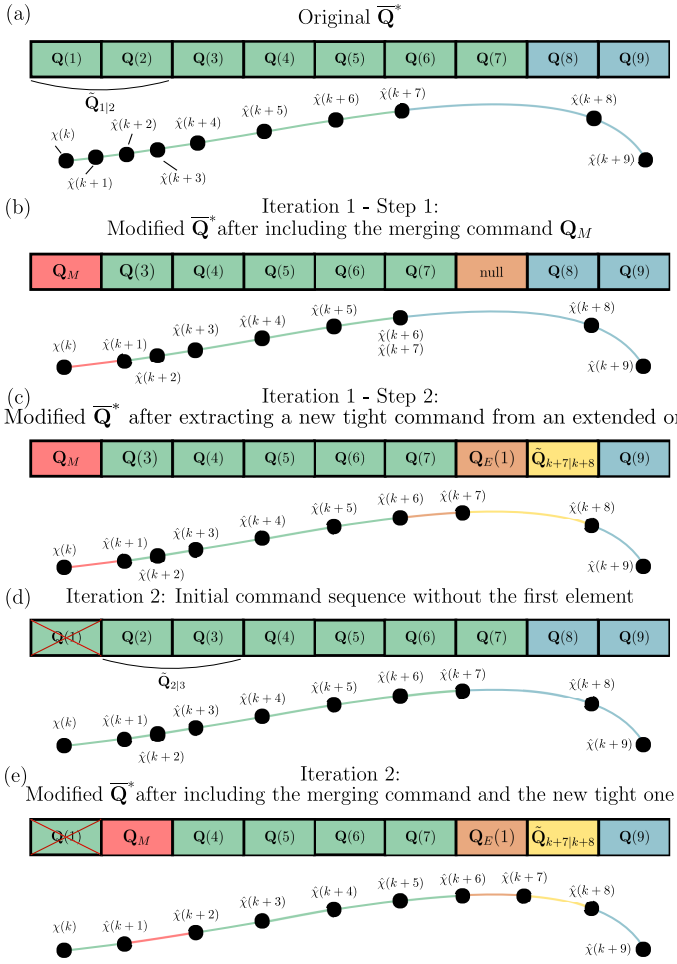


Fig. 4: Example of trajectory refinement. (a) Initial trajectory with $N_c = 9$ and $N_r = 2$. (b) The two first commands are merged, $N_m = 2$, $N_s = 5$, and $N_z = 1$. The merging command is included as the first element, the five remaining tight commands are copied and the seventh one is null. (c) The null command is replaced by a tight one (brown) extracted from the extended one. The extended command is updated (yellow). (d) Next iteration: the first element is removed from the merging process. (e) The second element is now the command merging the second and third commands. A tight command is included in the seventh element to compensate the merging one and the first extended command is updated. At this point the trajectory cannot be improved anymore.

(see figure 4.d). Thus, the whole control sequence is improved. Although only the first command is applied, providing a modified control sequence as initial values to the next optimization process allows improving the next calculated trajectory. Finally, it should be noted that even if our two-steps method is applied at each

Table 1: Configuration description - TC: terminal constraint - OC: obstacle constraint (Number of obstacles) - TR: trajectory refinement

	N_p	N_c	N_r	TC	OC	TR
Ω_1	15	15	0	No	Yes (1)	No
Ω_2	60	60	0	Yes	Yes (1)	No
Ω_3	15	15	5	Yes	Yes (1)	No
Ω_4	15	15	5	Yes	Yes (1)	Yes
Ω_5	15	15	3	Yes	Yes (2)	Yes
Ω_6	15	15	5	Yes	Yes (2)	Yes
Ω_7	15	15	7	Yes	Yes (2)	Yes
Ω_8	25	25	15	Yes	Yes (3)	Yes

iteration to the whole control sequence to improve the sub-optimal solution, it does not always modify it.

5 Results

In this section, we present the results obtained simulating a VPC servoing for a differential drive robot equipped with a camera. The program was implemented using the C++ language and the cost function minimization was done with the SQP solver from the NLopt package [28]. The tests were performed on an Intel Core i7-10700 running at 2.90Ghz coupled with 16 GB of RAM.

In this work, we consider the depth of the visual features as known, as it would be the case with a stereo camera like an Intel RealSense[4] one. Moreover, at the first step, the minimization problem is solved with a control vector equal to zero. For the next navigation steps, it is initialized with the results of the previous minimization. Finally, the tight boundaries are setup such as $0 \leq v \leq 0.4m/s$, $-0.1rad/s \leq \omega_r \leq 0.1rad/s$, and $-0.1rad/s \leq \omega_p \leq 0.1rad/s$, and the extended ones are ten times larger. In the figures representing the robot driving in the scene, the robot and the camera are represented in dark blue, the path of the mobile base by a plain orange line, and the predicted path of the camera by a dashed orange line. The desired camera pose is symbolized by a red triangle and the landmark is represented by red points. The obstacles are represented by plain green circles and the safety boundaries by pointed green circles. In the figures representing the visual features evolution, blue dots are the values for the initial robot pose, green dots are the values for current pose, and red dots the values for the last predicted pose. The dark blue circles represent the area corresponding to the terminal constraints and their center are the desired visual features values. Finally, plain and dashed lines correspond to the evolution of past and predicted visual features respectively.

We use eight configurations described in Table 1 to highlight the different ideas developed in this paper. Each configuration is denoted by Ω_i, with $i \in [1...8]$, and is characterized by the prediction horizon N_p, the control horizon N_c, the number of relaxed control inputs N_r, the use of the terminal constraint, the number of

[4] https://www.intelrealsense.com/

obstacles and the use of the introduced refinement trajectory method. The eight configurations are organized within three different sets. The first one (Ω_1, Ω_2, Ω_3 and Ω_4) allows to illustrate the need of a large prediction to guarantee the system stability and the improvements offered by the presented solution over the classical ones. Moreover, configuration (Ω_4) is used to provide some insight about the performances of the proposed VPC scheme and about the way it works. The second one (Ω_5, Ω_6 and Ω_7) allows to show the performances of the proposed VPC scheme for different parameters, illustrating their respective effect. Finally, the third one (Ω_8) is used to compare the proposed VPC scheme with a classical IBVS controller coupled with an obstacle avoidance one [29] in a more challenging environment.

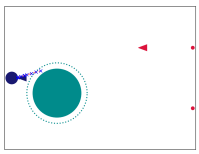
5.1 Impact of the prediction horizon

In this first set of simulations, one considers the first four configurations described in table 1. For the four simulations the navigation task is identical: the initial robot configuration is $[x_r = 0, y_r = 0, \theta_r = 0, \theta_p = 0]$ and the desired camera pose is $[x_c = 2, y_c = 0.5, \theta_c = 0]$. Moreover, the robot has to avoid one circle-shaped obstacle positioned at $[x_{o1} = 0.75, y_{o1} = -0.25]$, with a radius of $r_{o1} = 0.4$ meter and safety distance $\delta_c = 0.1$ meter.

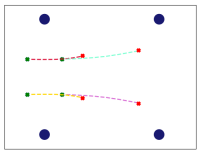
For the first one Ω_1 ($N_p = 15$ and $N_r = 0$), the range covered by the prediction horizon does not allow reaching the desired state from the initial one as it can be seen in the Cartesian space (Figure 5(a)) or in the image space (Figure 5(b)). With such a configuration, the stability is not guaranteed and the navigation might fail. This is what happens in this example where the robot reaches a local minima in the neighborhood the obstacle while driving towards the goal (Figure 5(c)). Thus, the VPC scheme fails to make the visual features converge towards the desired ones (Figure 5(d)).

With the second configuration Ω_2 ($N_p = 60$ and $N_r = 0$), the prediction horizon is sufficiently large to guarantee the navigation stability when the global solution of the optimization problem is calculated. However, the complexity of the optimization problem only allows the use of local solvers. Thus, from the initial pose, the solver only manages to compute a local optimum which does not respect the terminal constraint (Figures 5(e) and 5(f)). At each iteration, the solution optimality improves, and the solver eventually computes a trajectory dealing with the terminal constraint (Figure 5(h)) and leading to the desired pose (Figure 5(g)). From now on, the terminal constraint is respected and the VPC scheme allows the robotic system to reach the goal in the different spaces (Figures 5(i) and 5(j)) despite the computation of a non-optimal solution at each iteration. Using a large number of prediction steps indeed manages to safely achieve the navigation task. However, the large value of N_p leads to large computational times (see Table 2), limiting the use of this approach for a real-time application.

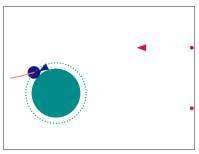
The third configuration Ω_3 ($N_p = 15$ and $N_r = 5$) is a first attempt to offer a large prediction horizon while limiting the number of prediction steps, and thus the processing time, by including five relaxed control input constraints. Similarly to the previous example, the prediction range is large enough, and it takes the local solver a couple of iterations to eventually compute a trajectory reaching the desired pose (Figure 5(k)) and dealing with the terminal constraint (Figure 5(l)).



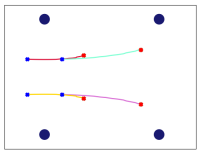
(a) Ω_1 - Initial pose



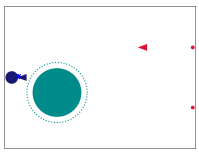
(b) Ω_1 - Initial visual features



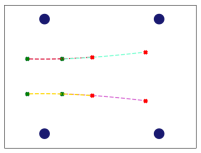
(c) Ω_1 - Final pose



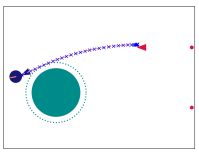
(d) Ω_1 - Final visual features



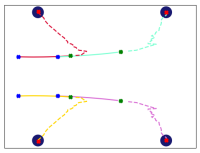
(e) Ω_2 - Initial pose



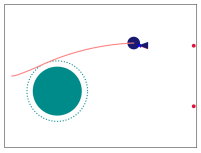
(f) Ω_2 - Initial visual features



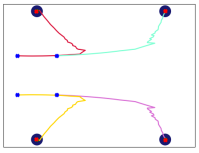
(g) Ω_2 - Intermediate pose



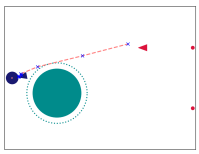
(h) Ω_2 - Intermediate visual features



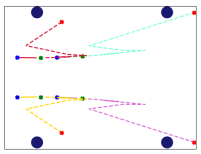
(i) Ω_2 - Final pose



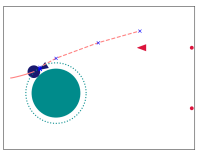
(j) Ω_2 - Final visual features



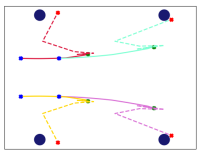
(k) Ω_3 - Intermediate pose



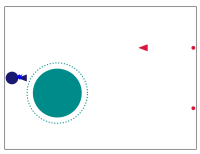
(l) Ω_3 - Intermediate visual features



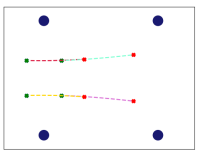
(m) Ω_3 - Final pose



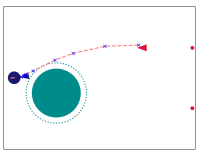
(n) Ω_3 - Final visual features



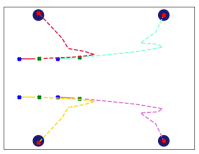
(o) Ω_4 - Initial pose



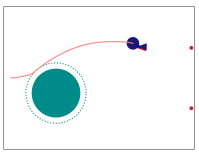
(p) Ω_4 - Initial visual features



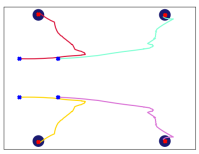
(q) Ω_4 - Internediate pose



(r) Ω_4 - Internmediate visual features



(s) Ω_4 - Final pose



(t) Ω_4 - Final visual features

Fig. 5: Simulated results for different configurations (1 obstacle)

Table 2: Processing performances for Ω_2 and Ω_4

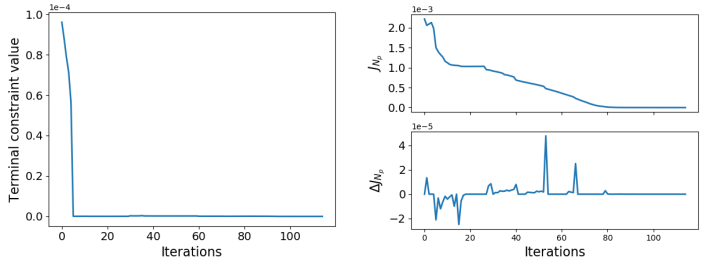
	Average Optimization Time	Number of Optimization Iterations	Average Time Per Iteration	Average Refinement Time
Ω_2	592 ms	51	17 ms	/
Ω_4	9.5 ms	59	0.17 ms	0.11 ms

But unlike the previous example, the robot stops before reaching the goal, due to the use of a local solver, and the navigation fails. At the corresponding state, the non-optimal solution offers a trajectory long enough to reach the desired pose but mostly relying on the steps with extended constraints. The steps with tight constraints are under-used, with null or quasi-null values (Figure 5(m)). With such a trajectory, the first predicted command, *i.e.,* the one actually used to control the system, is null, the robot stops, and the task fails (Figure 5(n)).

Finally, the example with configuration Ω_4 shows the relevance and efficiency of the proposed approach, *i.e.,* large prediction horizon obtained by composing with tight and relaxed constraints ($N_p = 15$, $N_r = 5$) and refinement of trajectory to improve a sub-optimal solution. First, similarly to the previous examples, the use of a local solver initially requires few iterations to compute a solution dealing with the terminal constraint and reaching the desired pose (Figures 5(o), 5(p), 5(q) and 5(r)). Then, once the terminal constraint is respected, *i.e.,* the trajectory leads to the desired pose, the refinement of the trajectory guarantees that the predicted steps with tight constraints, and especially the first one, are non-null. With such a configuration, the VPC scheme thus manages making the visual features converge towards their reference values (Figure 5(t)), and the camera reaches the desired pose. This example highlights the necessity to include both the terminal constraint and the method refining the trajectory in the VPC scheme in order to guarantee its stability when using relaxed constraints. Finally, the main advantage of this approach with respect to configuration Ω_2 is the significant decrease of the processing time. As it is shown in Table 2, the average time to solve the optimization problem is around 10 ms, *i.e.,* 60 times faster than for configuration Ω_2. It can be seen that the average time to solve one optimization iteration is around 100 times faster than for Ω_2 but in this case it is also necessary to take into account the refinement time which is about 0.11 ms. Finally we can notice that the use of the relaxed constraints does not make the problem significantly more challenging to solve. Indeed, the number of iterations required to solve the optimization problem are similar for both configurations Ω_2 and Ω_4. Thus, with the proposed approach, the stability is guaranteed and the processing time is compatible with a real-time application.

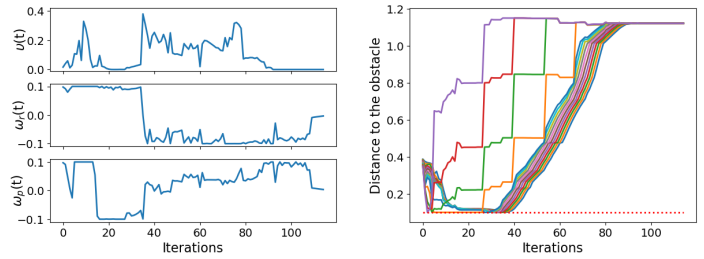
5.2 Close-up on the proposed solution

In this part we provide complementary data regarding the example using configuration Ω_4. First, we focus on the terminal constraint value plotted in Figure 6(a). As mentioned in the previous section, the constraint is initially not respected (positive value) due to the use of a local solver computing a too short trajectory.



(a) Terminal constraint

(b) Up: cost function value — Down: Variation of the cost function due to refinement



(c) System velocities

(d) Obstacle constraint - Each curve represents the distance between the obstacle and one piece of the predicted trajectory

Fig. 6: Complementary data for Ω_4

However, as it can be seen in the upper part of Figure 6(b), the solver manages to improve the sub-optimal solution and the cost function value decreases at each new iteration, leading to trajectories which are longer and closer to the desired pose. Thus, the solver eventually computes a trajectory respecting the final constraint, *i.e.,* the terminal constraint value is equal to zero or negative. In the lower part of Figure 6(b) it can be seen the evolution of ΔJ_{N_p} which corresponds to the difference between the cost function value obtained by the solver and the new cost function value after the refinement step. While the predicted trajectory has to deal with the obstacle, *i.e.,* up to the 25^{th} iteration, the computed trajectory is modified by the refinement method to avoid the navigation failure seen with configuration Ω_3. This results in an increase of the cost function: ΔJ_{N_p} is mostly negative or null before the 25^{th} iteration. After avoiding the obstacle, the refinement step merges sub-optimal pieces of trajectory and has an opposite impact on the cost function. Indeed, the cost value decreases: ΔJ_{N_p} is positive or null after the 25^{th} iteration. Thus, we can see that refining the trajectory to guarantee the success of the navigation task might imply either an increase or a decrease of the cost function if required.

Finally, in Figures 6(c) and 6(d), the evolution of the system velocities and of the obstacle constraints are presented. Regarding the control inputs, it can be noticed that they stay within the tight boundaries despite the use of relaxed bound-

Table 3: Example of trajectory refinement

	Initial sequence			After merging			First extraction			Second extraction		
Prediction	v	ω_r	ω_p	v	ω_r	ω_p	v	ω_r	ω_p	v	ω_r	ω_p
1	0.002	0.097	0.1	0.002	0.097	0.1	0.002	0.097	0.1	0.002	0.097	0.1
2	0.036	0.1	-0.061	0.036	0.1	-0.061	0.036	0.1	-0.061	0.036	0.1	-0.061
3	0	0.1	0.092	0	0.1	0.092	0	0.1	0.092	0	0.1	0.092
4	0	0.097	0.1	0	0.097	0.1	0	0.097	0.1	0	0.097	0.1
5	0	0.098	0.1	0	0.098	0.1	0	0.098	0.1	0	0.098	0.1
6	0.149	0.099	0.090	0.149	0.099	0.090	0.149	0.099	0.090	0.149	0.099	0.090
7	0.399	0.1	0.098	0.399	0.1	0.098	0.399	0.1	0.098	0.399	0.1	0.098
8	0	-0.094	0.1	0.002	-0.094	-0.097	0.002	-0.094	-0.097	0.002	-0.094	-0.097
9	0	0.1	-0.097	0	0	0	0.007	0.1	-0.079	0.007	0.1	-0.079
10	0.002	-0.1	-0.1	0	0	0	0	0	0	0.007	0.1	-0.078
11	0.078	1	-0.795	0.078	1	-0.795	0.070	0.910	-0.716	0.062	0.821	-0.638
12	0.251	0.523	-0.963	0.251	0.523	-0.963	0.251	0.523	-0.963	0.251	0.523	-0.963
13	3.866	-1	0.997	3.866	-1	0.997	3.866	-1	0.997	3.866	-1	0.997
14	3.990	-0.992	-0.541	3.990	-0.992	-0.541	3.990	-0.992	-0.541	3.990	-0.992	-0.541
15	0	0.028	1	0	0.028	1	0	0.028	1	0	0.028	1

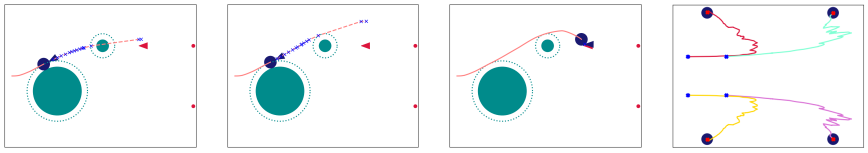
aries to enlarge the prediction horizon. Concerning, the obstacles constraints, they are respected all along the navigation for each piece of the predicted trajectory.

Finally, we provide an example of a trajectory refinement in Table 3. It presents the evolution of the calculated 15 steps (the first 10 ones with tight constraints and the last 5 ones with relaxed constraints) during this particular refinement phase. The commands computed by the solver are given in the first set of three columns, while the ones obtained after merging are given in the second set of three. First, the algorithm detects that the commands highlighted with blue are sufficiently small to be merged in a unique command. It results in the commands highlighted with orange, where the one in the 8^{th} line is the result of the merging step and the ones on the two following ones are substituted with null values. It is then necessary to extract twice a tight command from the first relaxed one, on the 11^{th} line, in order to replace the null commands of the 9^{th} and 10^{th} lines. The results are presented in the last two sets of three columns. First, the commands highlighted with green correspond to the first extraction with the new non-null tight command on the 9^{th} line and the updated relaxed one on the 11^{th} line. Similarly, the commands highlighted with red correspond to the new non-null tight command and to the updated relaxed one. For both extractions, the limiting component was the mobile base angular velocity. This is why they are equal to the tight bounds while the other ones, mobile base linear velocity and pan-platform angular velocity, were scaled accordingly. The sequence of control inputs presented in the last set of three columns corresponds to the result of the optimality refinement algorithm and is now the new solution to the optimization problem.

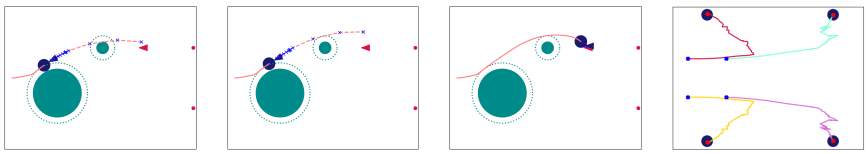
5.3 More advanced examples

One now proposes a set of three simulations exploring different scenarii with the proposed approach. To do so, the configurations Ω_5, Ω_6 and Ω_7 presented in Table 1 are used. For the three of them, a second circle-shaped obstacle is positioned

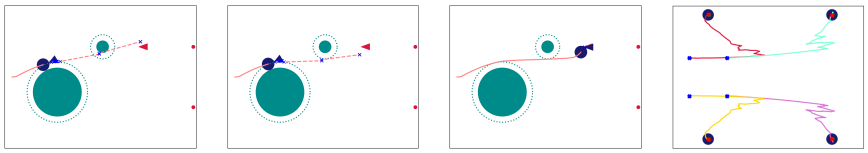
at $[x_{o2} = 1.5, y_{o2} = 0.5]$, with a radius of $r_{o1} = 0.1$ meter. Unlike the previous obstacle, this one cannot be initially detected by the laser rangefinder as one considers its range limited to one meter. Thus, the second obstacle is detected during the navigation, modifying the optimization problem along the servoing. It thus allows to propose a simulation closer to a real experiment where the optimization problem is constantly modified via the updates of the obstacle constraints.



(a) Ω_5 - Before detection of the 2^{nd} obstacle (b) Ω_5 - After detection of the 2^{nd} obstacle (c) Ω_5 - Final pose (d) Ω_5 - Final visual features



(e) Ω_6 - Before detection of the 2^{nd} obstacle (f) Ω_6 - After detection of the 2^{nd} obstacle (g) Ω_6 - Final pose (h) Ω_6 - Final visual features



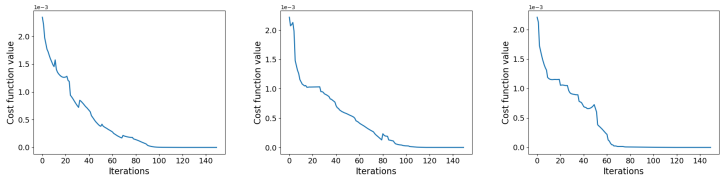
(i) Ω_7 - Before detection of the 2^{nd} obstacle (j) Ω_7 - After detection of the 2^{nd} obstacle (k) Ω_7 - Final pose (l) Ω_7 - Final visual features

Fig. 7: Simulated results for different configurations (2 obstacles)

The results of the simulation performed with configuration Ω_5 ($N_p = 15$, $N_r = 3$ and optimality refinement algorithm) are presented in Figures 7(a), 7(b), 7(c) and 7(d). As it is shown in Figure 7(a), the predicted trajectory initially passes through the second obstacle which has not been detected yet and therefore is not taken into account in the constraints. As soon as it is detected, the constraints are modified accordingly and the resulting trajectory avoids the newly discovered obstacle (Figure 7(b)). The insertion of a new obstacle constraint significantly modifies the trajectory and the solver does not manage to respect the terminal constraint. Similarly to the beginning of the servoing, the optimality of the solution is improved at each new iteration and the terminal constraint is eventually respected. Thus, the visual features converge towards their reference values 7(d)) and the robot reaches the desired pose (Figure 7(c)).

The last set of simulations obtained with configurations Ω_6 and Ω_7 are shown in Figures 7(e), 7(f), 7(g), 7(h), 7(i), 7(j), 7(k) and 7(l). The configurations are similar to Ω_5 except for the number of relaxed constraints increased to $N_r = 5$ for Ω_6 and to $N_r = 7$ for Ω_7. For the configuration Ω_6, the robot trajectory and the visual features evolution are strongly similar despite a different number of relaxed constraints. For the configuration Ω_7, the robot trajectory is different as the robot manages to pass between the two obstacles. Indeed, due to the larger value of N_r, the part of the trajectory made of the relaxed constraints has more degrees of freedom and is less rigid. Thus, the solver can compute at an early stage a trajectory passing between the obstacles. Despite these small differences, this highlights the ease of selection of the number of relaxed constraints. Indeed, it is sufficient to offer a prediction horizon long enough to guarantee the stability, while the approach is not over sensitive to the value of N_r.

Finally, in Figure 8, the evolution of the cost function is plotted for configurations Ω_5, Ω_6 and Ω_7. For the three configurations, we notice a raise of the cost function value when the constraint related to the second obstacle is introduced. This raise, which is more or less important depending on the considered configuration, prevents the use of the cost function constraint to deal with sub-optimality as done in [26]. When relying on such a constraint, it is mandatory to be able to compute a trajectory whom cost is smaller than the one at the previous iteration. This seems relevant when the optimization problem is constant over the whole servoing. However, as it has been shown with these examples, the approach presented in this paper is more appropriate when the constraints vary over time, *e.g.,* when using obstacle constraints.

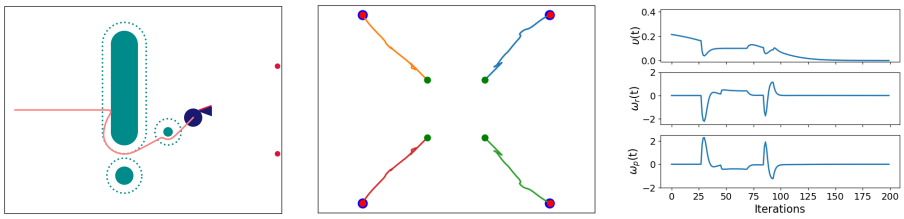


(a) Ω_5 (b) Ω_6 (c) Ω_7

Fig. 8: Evolution of the cost functions

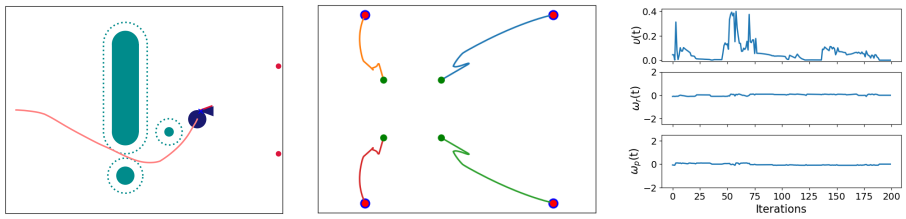
5.4 Comparison with classical IBVS coupled with obstacle avoidance

To conclude this section, one proposes to compare the results obtained when performing a navigation task with the proposed VPC scheme, with the ones obtained by coupling a classical IBVS controller [9] with an obstacle avoidance one [30] (IBVS-OA). With the IBVS-OA approach, the robot is initially controlled relying a classical IBVS controller. When the robot is too close from an obstacle, *i.e.,* when the current distance to one obstacle is smaller than a defined threshold d_0, one switches to the obstacle avoidance controller. This latter allows the mobile-base to follow an envelope at a distance d_0 from the contour of the obstacle. During



(a) Final pose (b) Final visual features (c) System velocities

Fig. 9: Simulated results obtained (IBVS-OA)



(a) Final pose (b) Final visual features (c) System velocities

Fig. 10: Simulated results obtained (VPC)

the obstacle avoidance phase, the camera is controlled using a modified IBVS controller to keep the landmark in the center of the camera field of view. One switches back to the IBVS controller when the obstacle is avoided, *i.e.,* when the camera is aligned with the mobile base ($\theta_p = 0$). To guarantee the continuity of the control inputs, one does not switch directly from one controller to the other. One relies on dynamical sequencing which guarantees that the values of two successive controllers are identical at the switching time [29]. The parameters to be set in this approach are the controllers gains (λ_{vs} for the visual servoing controller and λ_{oa} for the obstacle avoidance one), the safety distance triggering the obstacle avoidance (d_0) and the time constant of the dynamical sequencing (τ).

The two navigation modes are tested using the following scenario. The initial robot configuration is $[x_r = 0, y_r = 0, \theta_r = 0, \theta_p = 0]$ and the desired camera pose is $[x_c = 2, y_c = 0, \theta_c = 0]$. Moreover, the robot has to avoid 3 obstacles: a wall ($[x_w = 1.25, y_w = 0.25]$, 1 meter of height and 0.4 m of width) and two circle-shaped obstacles ($[x_{o1} = 1.25, y_{o1} = -0.75]$ with a radius of $r_{o1} = 0.1$ meter, and $[x_{o2} = 1.75, y_{o2} = -0.25]$ with a radius of $r_{o2} = 0.05$ meter). For both modes, one uses a 0.1 meter safety distance, *i.e.,* $\delta_c = 0.1$ for VPC and $d_0 = 0.1$ for IBVS-OA. The VPC scheme is setup with the Ω_8 configuration while the IBVS-OA parameters are $\lambda_{vs} = 0.35$, $\lambda_{oa} = 2$ and $\tau = 2$ seconds.

In Figures 9 and 10, the results obtained with both navigation modes are shown. In both cases the navigation task is accomplished: the visual servoing is achieved (Figures 9(b) and 10(b)) while collisions are avoided (Figures 9(a) and 10(a)). However the obtained trajectories are significantly different. With the IBVS-OA controllers, the robot first moves straight up encountering the wall. At this instant, it switches to the obstacle avoidance controller which allows to

follow the envelope defined by d_0. Once the obstacle is avoided, it switches back to the IBVS controller. Once again the robot is too close from an obstacle and the obstacle avoidance controller is re-activated. Finally, after bypassing the obstacle, it switches back to the IBVS controller to reach the desired pose. On the other side, the VPC controller allows to take into account the presence of an obstacle since the beginning of the navigation. Thus, the robot drives towards the goal while anticipating the wall and the first round-shaped obstacle. It manages to pass between these two obstacles while respecting the safety distance constraint. Finally, it detects the second round-shaped obstacle, avoids it and reaches the desired pose. Thanks to the predictive nature of the VPC controller, the robot manages to drive towards the goal and avoid obstacles while traveling a shortest distance than with the first method: 2.43 meters with VPC versus 2.67 meters with IBVS-OA. Moreover, it should be noted that with the IBVS-OA mode the robot is close to crashing the wall and does not manage to properly follow the envelope around the second round-shaped obstacle. Two solutions could be used to solve these problems. First, one can increase the safety distance, which triggers the obstacle avoidance earlier. Thus the robot has more time and space to move towards the envelope. However, a too large value of d_0 would close the gap between the wall and the bottom obstacle. It would not then be possible for the robot to drive between the two obstacles. The second solution would consist in increasing λ_{oa} to make the robot converge faster towards the envelope. This solution would significantly increase the values of the control inputs, and they might overpass the boundaries tolerated by the robotic system. Thus, one can see that it can be challenging to adequately setup the parameters of the IBVS-OA while it is sufficient to provide a long enough prediction horizon for the VPC.

Finally, one focuses on the commands for both modes (Figures 9(c) and 10(c)). One can notice that the IBVS-OA mode generates peaks for the angular velocities ω_r and ω_p. They happen when the robot switches to the obstacle avoidance controller in order to follow the given envelope. The computed commands can be outside of the admissible bounds depending on the values of the parameters λ_{oa} and τ. One more time, the setup of the parameters is a key process to achieve the navigation. On the other side, the command constraint used with a VPC controller allows to keep the computed commands within the given range of values as seen in Figure 10(c). This constraint is sufficient and there is no need for extra tuning.

This comparison shows the interest of the VPC approach over the coupling of an IBVS and obstacle avoidance controllers: the method tuning is easier and the obtained trajectory is shorter due to the predictive nature of the controller. However, the VPC approach is more expensive in terms of calculation and might not be successfully applied if this problem is not properly taken into account.

6 Conclusion

This work has proposed an advanced visual predictive control scheme adapted to the autonomous navigation problem. Indeed, VPC appears to be an interesting approach to deal with both local and global aspects of this problem. However, despite numerous works, it is still difficult to apply this technique to this particular context because of several issues including stability management, high computational burden, optimization problem non convexity, etc. The proposed VPC scheme ef-

ficiently deals with these issues thanks to several contributions: (i) a method for relaxing the input constraints in order to be able to consider large prediction horizons, thus guaranteeing stability while reducing the computational burden; (ii) a two-steps approach based on the equivalent control vectors for refining the optimized trajectory, thus preventing the robot from stopping and avoiding task failures. The approach has been deeply tested and evaluated with numerous simulations of navigation among obstacles. It has also been compared to classical VPC approaches (without input constraints relaxation and trajectory refinement). The obtained results show that the proposed method solves the optimization problem 60 times faster than these latter, thus outperforming them.

Based on these promising results, future works will extend this approach to take into account further constraints and to deal with other important navigation issues such as the visibility of the visual features, the avoidance of dynamical obstacles, etc. It is also planned to experiment it on our robots. Regarding the optimization problem itself, it seems interesting to model the navigation as a multi-objective optimization problem and/or to consider different classes of solvers. By exploring these methods, we might be able to obtain better results in terms of navigation and computational performances.

7 Statements

Funding: The authors did not receive support from any organization for the submitted work.

Conflicts of interest: The authors declare they have no conflict of interest.

Code or data availability: This work does not content any code or data publicly available.

Authors' contributions: Conceptualization and Methodology: A. Durand-Petiteville (ADP) and V. Cadenat (VC); Formal analysis and investigation: ADP; Writing - original draft preparation: ADP; Writing - review and editing: ADP and VC.

Ethics approval: Not applicable.

Consent to participate: Not applicable.

Consent for publication: Not applicable.

References

1. R. Siegwart and I.R. Nourbakhsh. *Introduction to autonomous mobile robots*. A bradford book, Intelligent robotics and autonomous agents series. The MIT Press, 2004.
2. Javier Minguez, Florant Lamiraux, and Jean-Paul Laumond. Motion planning and obstacle avoidance. In *Springer handbook of robotics*, pages 1177–1202. Springer, 2016.

3. F. Bonin-Font, F. Ortiz, and G. Oliver. Visual navigation for mobile robots : a survey. *Journal of intelligent and robotic systems*, 53(3):263, 2008.
4. Frank Allgower, Rolf Findeisen, Zoltan K Nagy, et al. Nonlinear model predictive control: From theory to application. *Journal-Chinese Institute Of Chemical Engineers*, 35(3):299–316, 2004.
5. Lars Grüne and Jürgen Pannek. Nonlinear model predictive control. In *Nonlinear Model Predictive Control*, pages 45–69. Springer, 2017.
6. Tiago P Nascimento, Carlos Eduardo Trabuco Dórea, and Luiz Marcos G Gonçalves. Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach. *International Journal of Advanced Robotic Systems*, 15(1):1729881418760461, 2018.
7. Tiago T Ribeiro and André GS Conceição. Nonlinear model predictive visual path following control to autonomous mobile robots. *Journal of Intelligent & Robotic Systems*, 95(2):731–743, 2019.
8. G. Allibert, E. Courtial, and F. Chaumette. Predictive control for constrained image-based visual servoing. *IEEE Trans. on Robotics*, 26(5):933–939, October 2010.
9. F. Chaumette and S. Hutchinson. Visual servo control, part 1 : Basic approaches. *Robotics and Automation Mag.*, 13(4), 2006.
10. A. Assa and F. Janabi-Sharifi. Robust model predictive control for visual servoing. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2715–2720, Sep. 2014.
11. Adrian Burlacu, Cosmin Copot, and Corneliu Lazar. Predictive control architecture for real-time image moments based servoing of robot manipulators. *Journal of Intelligent Manufacturing*, 25(5):1125–1134, 2014.
12. A. Hajiloo, M. Keshmiri, W. Xie, and T. Wang. Robust online model predictive control for a constrained image-based visual servoing. *IEEE Transactions on Industrial Electronics*, 63(4):2242–2250, April 2016.
13. Antonio Paolillo, Teguh Santoso Lembono, and Sylvain Calinon. A memory of motion for visual predictive control tasks. In *International Conference on Robotics and Automation*, number CONF, 2020.
14. Franco Fusco, Olivier Kermorgant, and Philippe Martinet. Integrating features acceleration in visual predictive control. *IEEE Robotics and Automation Letters*, 2020.
15. S. Heshmati-alamdari, G. K. Karavas, A. Eqtami, M. Drossakis, and K. J. Kyriakopoulos. Robustness analysis of model predictive control for constrained image-based visual servoing. In *2014 IEEE Int. Conf. on Robotics and Automation*, pages 4469–4474, May 2014.
16. Aaron Mcfadyen, Peter Corke, and Luis Mejias. Visual predictive control of spiral motion. *IEEE Transactions on Robotics*, 30(6):1441–1454, 2014.
17. F. Ke, Z. Li, H. Xiao, and X. Zhang. Visual servoing of constrained mobile robots based on model predictive control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(7):1428–1438, July 2017.
18. Daewon Lee, Hyon Lim, and H Jin Kim. Obstacle avoidance using image-based visual servoing integrated with nonlinear model predictive control. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 5689–5694. IEEE, 2011.
19. David Q Mayne and Hannah Michalska. Receding horizon control of nonlinear systems. In *Proceedings of the 27th IEEE Conference on Decision and Control*,

pages 464–465. IEEE, 1988.
20. S. S. Keerthi and 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):265–293, 1988.
21. Hong Chen and Frank Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1217, 1998.
22. Adrien Durand-Petiteville. *Navigation référencée multi-capteurs d'un robot mobile en environnement encombré*. PhD thesis, Université Paul Sabatier-Toulouse III, 2012.
23. A. Durand-Petiteville, M. Courdesses, and V. Cadenat. A new predictor/corrector pair to estimate the visual features depth during a vision-based navigation task in an unknown environment. In *7th International Conference on Informatics in Control, Automation and Robotics*, Funchal, Portugal, June 2010.
24. John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198. IEEE, 2016.
25. D. Folio and V. Cadenat. *Treating Image Loss by using the Vision/Motion Link: A Generic Framework*, chapter 4. IN-TECH, 2008.
26. Pierre OM Scokaert, David Q Mayne, and James B Rawlings. Suboptimal model predictive control (feasibility implies stability). *IEEE Transactions on Automatic Control*, 44(3):648–654, 1999.
27. B. Djeridane. *Sur la commandabilité des systèmes non linéaires à temps discret*. PhD thesis, Université Paul Sabatier, 2004.
28. Steven G. Johnson. The nlopt nonlinear-optimization package, 2020.
29. V. Cadenat, D. Folio, and A. Durand-Petiteville. A comparison of two sequencing techniques to perform a vision-based navigation task in a cluttered environment. *Advanced Robotics*, 2012.
30. P. Souères, T. Hamel, and V. Cadenat. A path following controller for wheeled robots wich allows to avoid obstacles during the transition phase. In *IEEE, Int. Conf. on Robotics and Automation*, Leuven, Belgium, May 1998.