Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria dell'Automazione

Motion planning for coverage with vision-inspired sensors

Laureando

Giorgio Corrà

Relatore

Prof. Ruggero Carli

Relatore esterno

Prof. Dimos Dimarogonas

Correlatore

Ph.D. Antonio Adaldo

Anno Accademico 2015/2016

Abstract

In this work, we address the problem of deploying a team of mobile sensing agents for monitoring a 3-D structure. A function for measuring the quality of the vision is defined and we use a line search method for optimizing the pose of single sensors. The algorithm is extended for collaborative coverage, exploiting intermittent communication between pairs of agents. The algorithm is enriched with a collision avoidance method for working in a constrained environment. All the proposed algorithms are tested in simulations and real-word aerial robots.

Acknowledgements

For the development of this thesis, I feel the need to be grateful to several persons. First, I want to thank Professors Ruggero Carli and Dimos Dimarogonas, for giving me the opportunity to work on such a fascinating topic, and in the amazing and stimulating environment of KTH. A heartfelt thanks to Dr. Antonio Adaldo, for all the suggestions you gave me and the constructive discussions we had during these months. Next, I want to thank all the Ph.D.'s and students with whom I shared this experience. Particularly, I cannot express in words my gratitude to Paul: without your help I would still be configuring ROS , crying in the SML because life is unfair; I will never forget your jokes, both the ones that I understood and the ones that I did not. Finally, all the friends that I met in Stockholm, among all Gustavo and Daniela. Thank you for all the time we spent together, and especially for pushing me in one of the best adventures I had in my life! I hope we will meet again.

Passando al versante italiano, c'è una pletora di persone che dovrei ringraziare per questi 5 (ma facciamo anche 23) anni. Sono già sicuro che dimenticherò qualcuno, perdonatemi.

- La compagnia di Zugliano e dintorni. Mi sono chiesto spesso se avessimo qualcosa in comune e cosa ci tenga uniti; dopo anni di studi la risposta è arrivata spontaneamente: siamo tutti cavalli.
- Paolo, meriti una menzione speciale. Sappiamo entrambi di essere anime gemelle, purtroppo la natura è stata crudele e ci ha voluto creare incompatibili. Dubito che si riuscirà mai a colmare questo gap, ma tanto vale provarci comunque.
- La Conca Bene, la Conca Male, ma soprattutto la Conca A. Non potevo crescere in un ambiente migliore, l'ostracismo è più che giustificato.
- Il Corty Shore e i suoi frequentatori abituali. Come tutte le cose belle, non poteva durare in eterno, ma è stata una bella manifestaziöne.
- Dal Lago, Broccardo e i commensali grossi di Via P. P. Cortese. Vi conosco, vi rispetto.
- Pippo, ovviamente. Sei stato un perfetto Gran Mogol, spero di essere stato una buona giovane marmotta.

Infine, il più grande ringraziamento va ai miei genitori e i miei fratelli, a cui questa tesi è dedicata. Grazie per avermi sempre sostenuto e appoggiato in ogni mia scelta.

Voglio considerare comunque, a questo punto, chiusa la polemica tra me e il sottoscritto.

Fabio Noaro

Contents

Chapter 1

Introduction

In this work, we develop an algorithm that allows to autonomously deploy a team of aerial robots equipped with vision-inspired sensors in order to monitor a 3-D structure. Such a problem can be treated as an instance of the classical *coverage problem*. In order to address this problem, we will first define a function that measures how good is the vision that the sensor (or the team of sensors) has of the object that we monitor. The function that we propose depends only on the reciprocal positions and orientations of the sensors and the body, and has an intuitive geometrical interpretation. Then, a gradient based algorithm is used and communications between agents are exploited to optimize the pose of the sensors. Moreover a collision avoidance technique is defined and implemented to allow safe movement of the agents in a constrained environment. Finally the algorithm is tested with simulations and implemented on real quadcopters for experimental validation.

In the following sections we give an overview in the state of art, as well as of the context in which this project was made.

1.1 Literature Review

Control of quadcopters and application

In the last years, great effort has been made in the research in the automatic control of unmanned aerial vehicles (UAVs). Their typical design is the quadcopter [1], which includes four propellers mounted in one plane, attached to independent electrical motors. The control of the this aerial vehicles is performed by adjusting the speeds of the motors. The reasons of the increasing popularity of quadcopters are multiple: their structure allows vertical take-off and landing, as well as stationary flight, so they are easily manoeuvrable even in small spaces; they can reach a high payload/weight ratio; the cost of components and spare parts have become very cheap. These characteristics make the UAVs ideal for research purposes, and also for hobbyists. For a complete survey of results in control theory, automation, robotics and bio-inspired engineering that involve quadcopters see [2, 3] and references therein.

One of the main challenges in which several works focus is controlling the motion of the quadrotor, for instance for tracking a given trajectory. In [4, 5], the problem is addressed applying classic non-linear control techniques with the aim of stabilizing the dynamics of the quadrotor. In [6], disturbances caused by wind are also taken into account. Sensor fusion and dynamic attitude estimation methods were also used in [7, 8, 9] for motion stabilization.

Another topic, inherently related to the simple trajectory tracking, is the generation of the trajectory that the quadrotor has to follow [10, 11]. The problem is often extended in a constrained version, where the trajectory also has to guarantee collision avoidance [12, 13].

One important branch of control theory which can be applied to aerial vehicles is *multiagent control*. Some examples are the so-called flocking and formation control of teams of quadcopters [14], the cooperative lifting and transportation of loads [15] or the problem of coverage in sensor networks.

Sensor networks and the coverage problem

A sensor network consists of a collection of sensing devices that can coordinate their actions through wireless communication and aim at performing tasks like reconnaissance, surveillance, target tracking and, generically, collection of information about the environment. Intuitively, in such tasks, the position of the sensors plays a crucial role. The *coverage problem* studies the deployment of the sensors in the space, in order to achieve the overall system objectives. The problem can be divided in several sub-cases, depending on the type of sensor that are used, on whether it is addressed from a centralized or a distributed point of view, and on the particular task considered.

Most of the existing results on coverage regard the use of sensors with symmetric, omnidirectional field of view [16], and only recently anisotropic [17] and vision based [18] sensors have been considered. The classical solution of the coverage problem considers *Voronoi tessellation* and the Lloyd algorithm [19] (see for instance [20]).

A complete survey of the literature goes beyond the scope of this thesis, therefore we relate to [21, 22] for a wider insight.

1.2 Related works

In this work we consider the problem of inspection of a 3D object with vision-based sensors, and we propose a novel footprint which is nor symmetric, nor omnidirectional. For movement of the sensors we exploit gradient search methods (see [23]) for the deployment. For the communication between agents we use a gossip-based interaction strategy, similar to the one proposed in [24] in which only asynchronous, unreliable communication are needed. Moreover, the same paper proposes to abstract the environment into a finite set of points, which can either be particular points of interest or represent a complete discretization of the environment. In this

Figure 1.1: Aeroworks 2020 logo.

work a similar idea is proposed, but adapted to the inspection task that we aim to perform.

1.3 Context

This work is a contribution to a European project called Aeroworks 2020. The goal of the project is to develop a team of collaborative aerial robotic workers, able to autonomously perform infrastructure inspection and maintenance tasks. Further details can be found at [25]. All the experiments took place in the Smart Mobility Lab [26], at KTH Royal Institute of Technology.

1.4 Thesis outline

In the rest of the thesis, the different stages of the formulation and implementation of the proposed coverage algorithm are presented. The work is organized as follows:

- In Chapter 2 some technical preliminaries are exposed.
- In Chapter 3 the footprint chosen for the sensors is explained and formalized, and the coverage problem is stated.
- In Chapter 4 the algorithm used for the solution of the coverage problem is described, and the convergence of the algorithm is proved.
- In Chapter 5 some simulations are shown and commented in depth.
- In Chapter 6 the experimental setup that was adopted is explained, and an experiment involving a real quadcopter is shown.
- In Chapter 7 the conclusions are given, as well as some insights for possible future developments.

Chapter 2

Technical preliminaries

2.1 Notation

A vector in $\mathbb{R}^n$ is denoted with a boldface latin letter, such as $\boldsymbol{a}$. For any $\boldsymbol{a} \in \mathbb{R}^n$, a_k denotes the k-th entry of $\boldsymbol{a}$, while $\|\boldsymbol{a}\|$ denotes the euclidean norm of $\boldsymbol{a}$. A unitary vector (i.e. a vector in $\boldsymbol{a} \in \mathbb{R}^n$, s.t. $\|\boldsymbol{a}\| = 1$) is denoted with a hat: $\hat{\boldsymbol{a}}$. The inner product between two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ is denoted with angle brackets: $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\top \boldsymbol{b}$.

A matrix in $\mathbb{R}^{n \times m}$ is denoted with a capital boldface latin letter, such as $\boldsymbol{A}$. In particular $\boldsymbol{I}$ indicates the identity matrix.

2.2 Coordinate frames and Euler angles

In the thesis, particularly in the experimental part (Chapter 6), we deal with two coordinate frames[1]: the world (or inertial) frame $W = \left(\boldsymbol{O}^W, \hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}\right)$, that is fixed, and the body frame $B = \left(\boldsymbol{O}^B, \hat{\boldsymbol{l}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{n}}\right)$, attached to the quadcopter, as in Figure 2.2b. If we consider a point $\boldsymbol{a}$ in the space we can write it with respect to the world frame, and we will denote it as $\boldsymbol{a}_W$, or with respect to the body frame, and we will denote it as $\boldsymbol{a}_B$. We can describe the transformation between the two forms using a rotation and a translation:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{a}_B \\ 1 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R} & \boldsymbol{O}_W^B \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_W \\ 1 \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{a}_W \\ 1 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R}^\top & \boldsymbol{O}_B^W \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_B \\ 1 \end{bmatrix} \end{aligned} \tag{2.1}$$

where $\boldsymbol{R} \in \mathrm{SO}(3)$ is the rotation matrix, whose entries are the direction cosines of $\hat{\boldsymbol{l}}, \hat{\boldsymbol{m}}, \hat{\boldsymbol{n}}$ with respect to $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}$, while $\boldsymbol{O}_W^B$ is the origin of the body frame written in the inertial coordinates, and $\boldsymbol{O}_B^W$ is the origin of the world frame written in the body coordinates.

[1] For a complete introduction to coordinate frames and rotation matrices, see [27].

Now we consider only the attitude, i.e. we assume that the origins of the two frames coincide. An intuitive way of expressing the orientation of the body frame is given by the Euler Angles, represented in figure 2.1b. The rotation is decomposed in three subsequent elementary rotations about one coordinate axis. To make the axis of the world frame coincide with the ones of the body frame we perform the following operations:

1. Yaw: rotate about axis $\hat{\boldsymbol{z}}$ of an angle γ, obtaining the frame $\left(\boldsymbol{O}, \hat{\boldsymbol{x}}', \hat{\boldsymbol{y}}', \hat{\boldsymbol{z}}'\right)$;

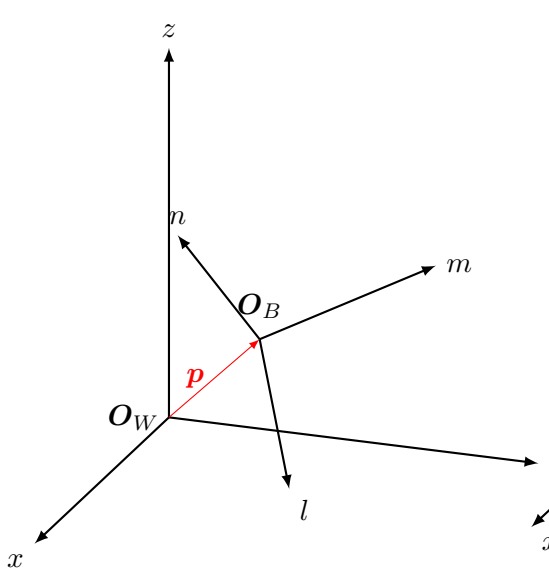
2. Pitch: rotate about axis $\hat{\boldsymbol{y}}'$ of an angle β, obtaining the frame $\left(\boldsymbol{O}, \hat{\boldsymbol{x}}'', \hat{\boldsymbol{y}}'', \hat{\boldsymbol{z}}''\right)$;

3. Roll: rotate about axis $\hat{\boldsymbol{x}}''$ of an angle α, obtaining the frame B.

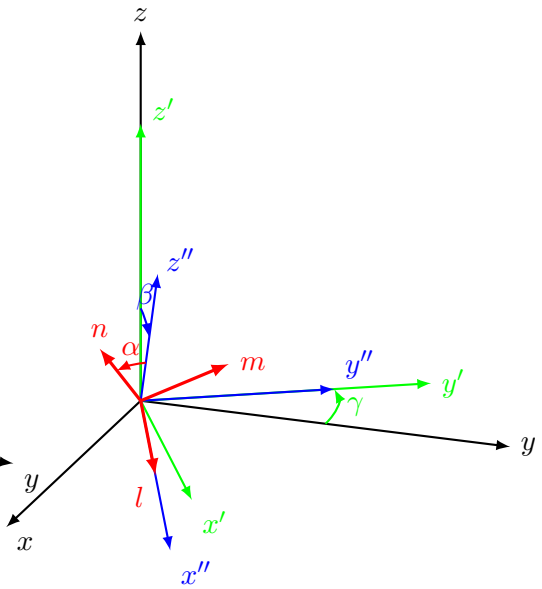
The correspondent rotation matrix can be obtained via multiplication of the rotation matrices associated with the single transformations:

$$\boldsymbol{R} = \boldsymbol{R}_z(\gamma)\,\boldsymbol{R}_y(\beta)\,\boldsymbol{R}_x(\alpha) = \begin{bmatrix} \mathrm{c}\,\gamma\,\mathrm{c}\,\beta & \mathrm{c}\,\gamma\,\mathrm{s}\,\beta\,\mathrm{s}\,\alpha - \mathrm{s}\,\gamma\,\mathrm{c}\,\alpha & \mathrm{c}\,\gamma\,\mathrm{s}\,\beta\,\mathrm{c}\,\alpha + \mathrm{s}\,\gamma\,\mathrm{s}\,\alpha \\ \mathrm{s}\,\gamma\,\mathrm{c}\,\beta & \mathrm{s}\,\gamma\,\mathrm{s}\,\beta\,\mathrm{s}\,\alpha + \mathrm{c}\,\gamma\,\mathrm{c}\,\alpha & \mathrm{s}\,\gamma\,\mathrm{s}\,\beta\,\mathrm{c}\,\alpha - \mathrm{c}\,\gamma\,\mathrm{s}\,\alpha \\ -\mathrm{s}\,\beta & \mathrm{c}\,\beta\,\mathrm{s}\,\alpha & \mathrm{c}\,\beta\,\mathrm{c}\,\alpha \end{bmatrix}, \tag{2.2}$$

where we used the post-moltiplication because the rotations are performed every time with respect to the current frame.

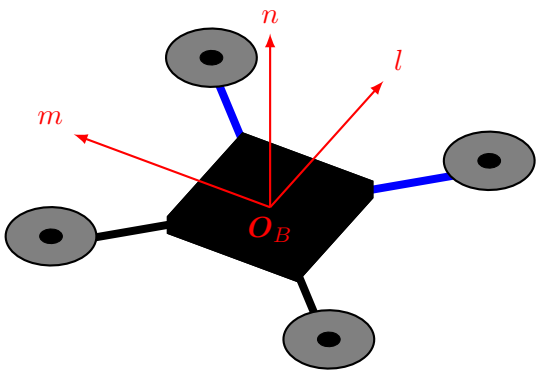


(a) World and body reference frames.



(b) Representation of the Euler angles.

(a) Iris quadcopter



(b) Sketch of the quadcopter with the body frame

2.3 Representation of the orientation

In the thesis, we represent the direction in which a sensor is pointing, as well as the direction normal to the surface of an object, as a 3-D unit vector (see Chapter 3). A unit vector $\hat{\boldsymbol{a}} = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}^\top$ can be equivalently expressed using only two independent parameters. Indeed it is defined with three scalars, but the constraint in the norm reduces to two the number of degrees of freedom. Therefore for defining $\hat{\boldsymbol{a}}$ we can use either its vectorial form or two angles (θ, ψ) in a latitude-longitude fashion, as in Figure 2.3a. It is easy to change from one form to the other with the formulas:

$$\hat{\boldsymbol{a}} = \begin{bmatrix} \cos(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) \\ \sin(\theta) \end{bmatrix}, \qquad \begin{cases} \psi = \arctan\left(\frac{a_2}{a_1}\right), \\ \theta = \arctan\left(\frac{a_3}{\sqrt{a_1^2+a_2^2}}\right). \end{cases} \tag{2.3}$$

Notice that using the function $arctan2$ we have $\psi \in (-\pi, \pi]$ and $\theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$ (since the squared root returns only positive values). The transformation from vector to angles is ambiguous only if $a_1 = a_2 = 0$, where the value of ψ is not defined. Anyway this will not cause any problems because in the proposed algorithm we only need the conversion from angles to vector.
We could also obtain the same result imagining to rotate the frame $\left(O, \hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \hat{\boldsymbol{b}}_3\right)$ about the third axis of an angle ψ and then of an angle $-\theta$ about the current second axis. The vector $\hat{\boldsymbol{a}}$ would result to be the first axis of the new frame, as represented in Figure 2.1a.

Now consider the case in which $\hat{\boldsymbol{a}}$ varies over time, so we should denote it as $\hat{\boldsymbol{a}}(t)$, but for simplicity we will drop the dependence on time. We can express the angular velocity of the rotation either in the original frame as $\boldsymbol{\omega} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^\top$

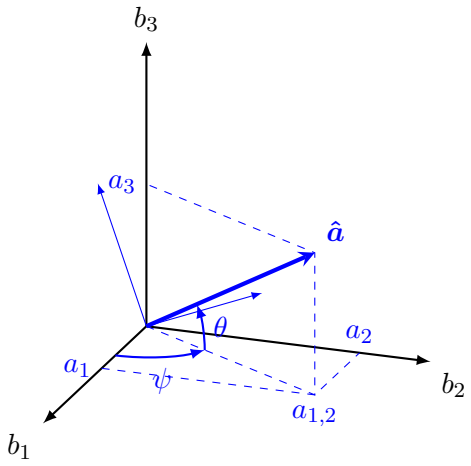
or with the latitude and longitude rates ($\dot{\theta}$, $\dot{\psi}$). It is easy to notice that:

$$\dot{\psi} = \omega_3, \tag{2.4}$$

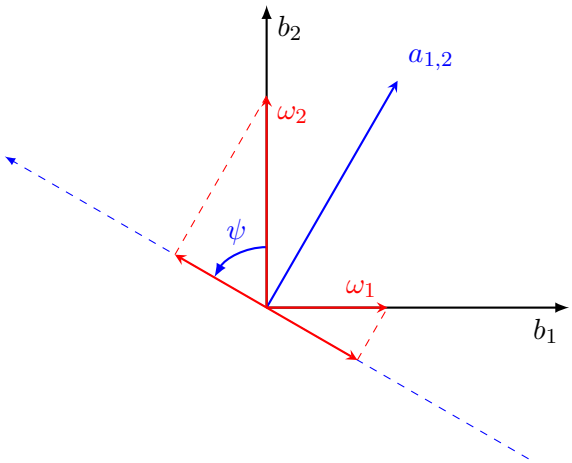
while the latitude rate can be computed by projecting ω_1 and ω_2 on the axis about which we perform the rotation of $-\theta$ (see Figure 2.3b):

$$\dot{\theta} = \sin(\psi)\,\omega_1 - cos(\psi)\,\omega_2, \tag{2.5}$$

where the sign is changed because we rotate of θ in clockwise direction.



(a) Representation of the two possible forms for expressing a unit vector $\boldsymbol{v}$



(b) Latitude rate and components of $\boldsymbol{\omega}$.

Chapter 3

Theoretical setup

In this chapter we define the footprint of the vision-based sensor, i.e. a function that measures the quality of the vision over the object that we are inspecting. As anticipated in the introduction, we do not consider the object as a whole, but just as a set of points on its surface, that we call landmarks. Depending on the case, the landmarks could be some particular points of interest of the object, or represent a complete discretization of it.

The definition of this function is proposed in Section 3.1 and its geometric interpretation is discussed. In Section 3.2 the coverage problem is formalized. In Section 3.3, we compute the gradient of the vision function, which will be used for the algorithm described in Chapter 4. In Section 3.4, it is shown an algorithm that allows to compute the optimal orientation for the sensor.
All the vectors in this chapter are expressed with respect to the world coordinate frame, thus we avoid to indicate the subscript to simplify the notation. Moreover we use the terms sensor and camera indistinctly.

3.1 Measure of the quality of vision

We represent a sensor as a pair $(\boldsymbol{p}, \hat{\boldsymbol{v}})$, where $\boldsymbol{p} \in \mathbb{R}^3$ is the position and $\hat{\boldsymbol{v}} \in \mathrm{SO}(2)$ is the orientation, expressed as the unit vector of the direction in which the camera is pointing. Now we consider a point on the object's surface, that we call *landmark*. We represent it as a pair $(\boldsymbol{q}, \hat{\boldsymbol{u}})$, where $\boldsymbol{q} \in \mathbb{R}^3$ is its position and $\hat{\boldsymbol{u}} \in \mathrm{SO}(2)$ is the unit vector of the normal direction with respect to the surface. A graphical representation (in $\mathbb{R}^2$ for simplicity) is proposed in Figure 3.1a.

We measure the *quality of the vision* that the camera has of the landmark $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ as:

$$\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, \boldsymbol{q}, \hat{\boldsymbol{u}}) = f(\|\boldsymbol{q} - \boldsymbol{p}\|) \left\langle \frac{\boldsymbol{q} - \boldsymbol{p}}{\|\boldsymbol{q} - \boldsymbol{p}\|}, \hat{\boldsymbol{v}} \right\rangle^+ \left\langle \frac{\boldsymbol{p} - \boldsymbol{q}}{\|\boldsymbol{q} - \boldsymbol{p}\|}, \hat{\boldsymbol{u}} \right\rangle^+, \tag{3.1}$$

where $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{y}$ is the scalar product and $x^+ = \max\{x, 0\}$ is the positive part. For now we can consider $f(\|\boldsymbol{q} - \boldsymbol{p}\|) = 1$, its role will be explained later.

To understand the meaning of (3.1) it is useful to think in two dimensions. We can define $\hat{\boldsymbol{r}} = \frac{\boldsymbol{q}-\boldsymbol{p}}{\|\boldsymbol{q}-\boldsymbol{p}\|}$, which is the unit vector of the direction that joins the sensor's and the point's positions (see Figure 3.1b). Equation (3.1) becomes:

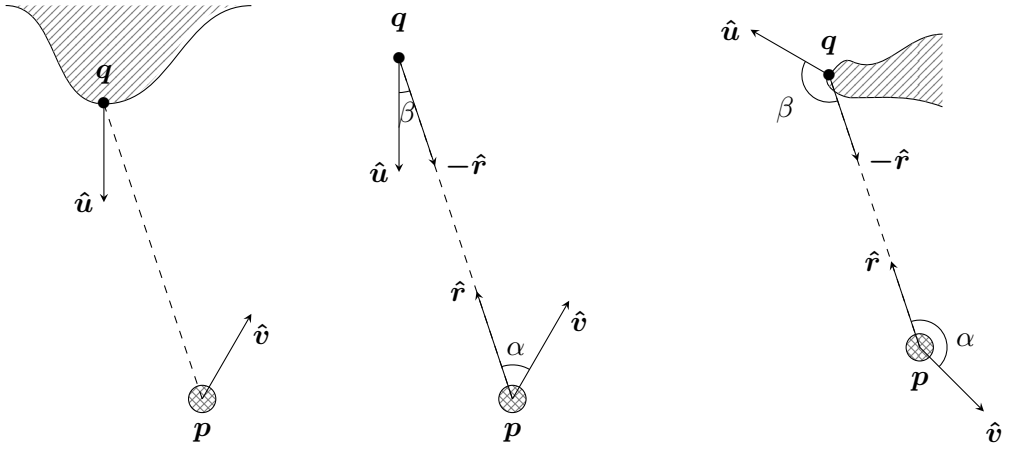
$$\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, \boldsymbol{q}, \hat{\boldsymbol{u}}) = \langle \hat{\boldsymbol{r}}, \hat{\boldsymbol{v}} \rangle^+ \langle -\hat{\boldsymbol{r}}, \hat{\boldsymbol{u}} \rangle^+ = \cos(\alpha)^+ \cos(\beta)^+, \tag{3.2}$$

where α is the angle between $\hat{\boldsymbol{r}}$ and $\hat{\boldsymbol{v}}$, and β is the angle between $-\hat{\boldsymbol{r}}$ and $\hat{\boldsymbol{u}}$. Notice that the quality of vision is higher if both α and β are small, and this is reasonable. Indeed a small value of α means that the camera is watching directly the point; a small value of β means that the camera is positioned almost orthogonally to the surface, so it has intuitively the best view of this part of the object.

Notice that in our definition we obtain $\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, \boldsymbol{q}, \hat{\boldsymbol{u}}) = 0$, as a consequence of the use of $(\cdot)^+$, in the following cases:

- $\langle \hat{\boldsymbol{r}}, \hat{\boldsymbol{v}} \rangle \leq 0$: this happens for all the points contained in the half space in the back of the camera.
- $\langle -\hat{\boldsymbol{r}}, \hat{\boldsymbol{u}} \rangle \leq 0$: this happens for all the points that are on the other side of the object, with respect to the camera.

Moreover, consider the situation described in Figure 3.1c, i.e. when both $\langle \hat{\boldsymbol{r}}, \hat{\boldsymbol{v}} \rangle \leq 0$ and $\langle -\hat{\boldsymbol{r}}, \hat{\boldsymbol{u}} \rangle \leq 0$. In this case if we didn't consider only the positive part, we would have a positive value of the vision even if clearly the point $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ is not visible from the camera because it is in the back of the camera and it is also covered by other parts of the object (given the orientation of $\hat{\boldsymbol{u}}$).

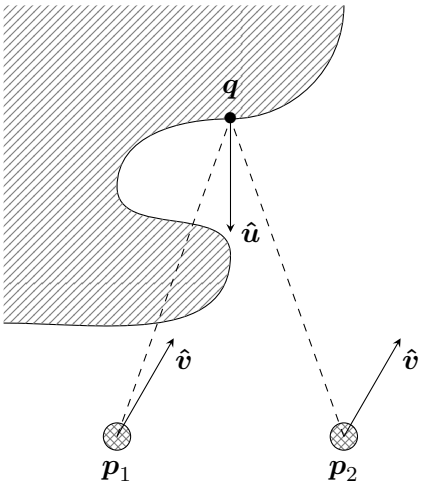


(a) Camera and object representation.

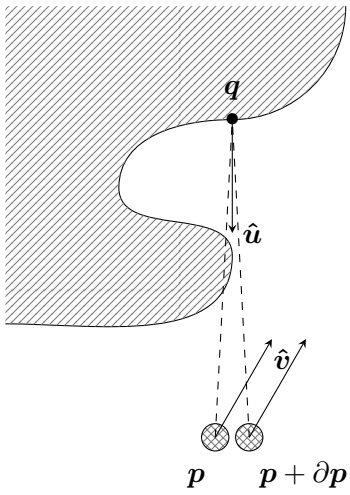
(b) Geometrical interpretation of vision.

(c) $\cos(\alpha) < 0,\ \cos(\beta) < 0$

Figure 3.1: Representation of the principal vectors and quantities used to define the vision function.



(a) Visibility of a landmark in presence of occlusions.



(b) Discontinuity in the vision function if we take occlusions into account.

Following these ideas, we can also write another equivalent definition of the vision function, less formal but more intuitive:

$$\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, \boldsymbol{q}, \hat{\boldsymbol{u}}) = \begin{cases} f(\|\boldsymbol{q}-\boldsymbol{p}\|) \left\langle \frac{\boldsymbol{q}-\boldsymbol{p}}{\|\boldsymbol{q}-\boldsymbol{p}\|}, \hat{\boldsymbol{v}} \right\rangle \left\langle \frac{\boldsymbol{p}-\boldsymbol{q}}{\|\boldsymbol{q}-\boldsymbol{p}\|}, \hat{\boldsymbol{u}} \right\rangle & \text{if } (\boldsymbol{q}, \hat{\boldsymbol{u}}) \text{ is visible,} \\ 0 & \text{if } (\boldsymbol{q}, \hat{\boldsymbol{u}}) \text{ is not visible,} \end{cases} \tag{3.3}$$

where the landmark $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ is considered *visible* if both the scalar products $\langle \hat{\boldsymbol{r}}, \hat{\boldsymbol{v}} \rangle$ and $\langle -\hat{\boldsymbol{r}}, \hat{\boldsymbol{u}} \rangle$ take positive values. This last definition may lead to wrong interpretations, so some remarks are necessary. Firstly, notice that even once the position of the sensor is fixed, the same landmark may be visible for certain orientations $\hat{\boldsymbol{v}}$ but not for others. Secondly, our definition of visibility does not take into account the presence of occlusions. Referring to Figure 3.2a and using our definition the landmark is visible from both the positions $\boldsymbol{p}_1$ and $\boldsymbol{p_2}$, while intuitively a camera positioned in $\boldsymbol{p}_1$ cannot see the point $\boldsymbol{q}$ because it is covered by another part of the object. Equation (3.3) could be still used if we changed the definition of visible landmark in a way that takes into consideration the possible occlusions (see [28] for examples of definitions). However using such a definition we would lose a property of smoothness of the vision function, while our definition ensures that the vision of a landmark is a continuous function of the position and orientation of the camera. Instead if we consider occlusions, the vision may jump from 0 to a positive value as a consequence of an infinitesimal change of the position (see Figure 3.2b).

Quality of the vision with multiple landmarks

Consider now the situation represented in Figure 3.3a, where one sensor has to monitor a set Q of m points:

$$Q = \{(\boldsymbol{q}_i, \hat{\boldsymbol{u}}_i),\ i = 1, \dots, m\}\,.$$

Then we measure quality of the vision of the camera over Q as:

$$\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \sum_{(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q} f(\|\boldsymbol{q} - \boldsymbol{p}\|) \left\langle \frac{\boldsymbol{q} - \boldsymbol{p}}{\|\boldsymbol{q} - \boldsymbol{p}\|}, \hat{\boldsymbol{v}} \right\rangle^{+} \left\langle \frac{\boldsymbol{p} - \boldsymbol{q}}{\|\boldsymbol{q} - \boldsymbol{p}\|}, \hat{\boldsymbol{u}} \right\rangle^{+}, \tag{3.4}$$

or equivalently, following the idea proposed in (3.3), as:

$$\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \sum_{(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q^V} f(\|\boldsymbol{q} - \boldsymbol{p}\|) \left\langle \frac{\boldsymbol{q} - \boldsymbol{p}}{\|\boldsymbol{q} - \boldsymbol{p}\|}, \hat{\boldsymbol{v}} \right\rangle \left\langle \frac{\boldsymbol{p} - \boldsymbol{q}}{\|\boldsymbol{q} - \boldsymbol{p}\|}, \hat{\boldsymbol{u}} \right\rangle, \tag{3.5}$$

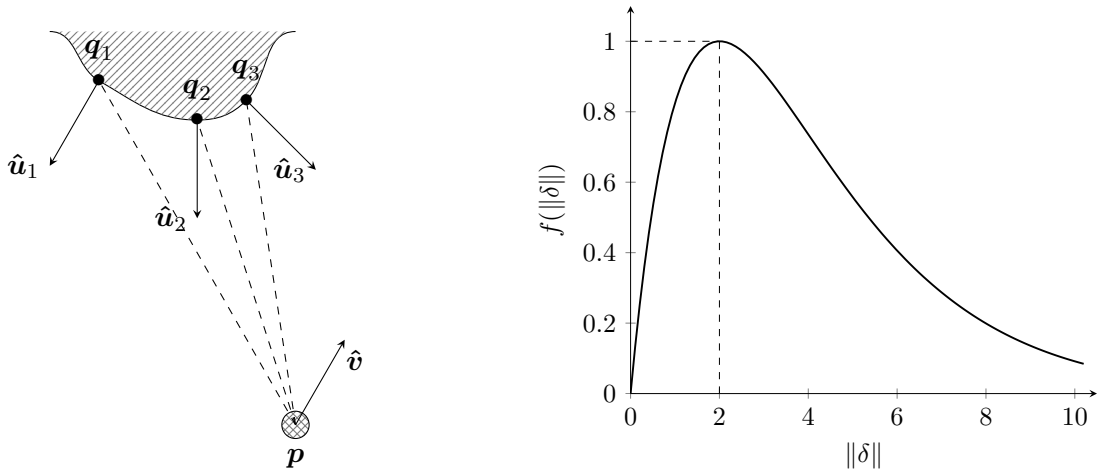
where $Q^V \subseteq Q$ contains only the points that are visible. Obviously, Q^V depends on Q but also on the position and orientation of the camera.

It can be proven that the value of the vision grows as the sensor goes farther from the object, if $f(\|\boldsymbol{q} - \boldsymbol{p}\|) = 1$. Indeed in this way all the angles α_i and β_i (defined previously) decrease, and thus the values of the cosines increase. A greater distance from the object turns out in a worse resolution of the image, so we want to avoid this situation. With this purpose we introduce a term which regulates the distance, that is $f(\|\boldsymbol{q} - \boldsymbol{p}\|)$. We choose a function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ with the following properties:

1. $f(\|\delta\|) \geq 0,\ \forall \delta \in \mathbb{R}_{\geq 0}$;
2. $f(\cdot)$ is continuously differentiable in all its domain;
3. $\lim_{\|\delta\| \to +\infty} f(\|\delta\|) = 0$;
4. $f(0) = 0$.

The first property ensures that the vision is still positive (or zero) in every configuration of camera and landmarks. The differentiability will be used in later computations in this chapter. The third property guarantees that the vision does not increase indefinitely, when the distance becomes greater (so it prevents the issue exposed previously). The fourth property ensures that the vision decreases when the sensor goes closer to the object (this is just a security measure). Moreover we add a fifth property:

5. $f(\|\delta\|) \in [0, 1]$ for $\|\delta\| \geq 0$.



(a) Camera and object representation

(b) Example of function $f(\|\delta\|)$.

Figure 3.3

This is not necessary for the algorithm to work, but it ensures that the value of the vision over a single point is still in $[0, 1]$, and over a set of m points is in $[0, m]$. Besides, a function that respects the first four properties has a global maximum. For ensuring that the same function respects also the fifth property it is sufficient to normalize it, dividing by the value of that maximum. We use:

$$f(\|\delta\|) = \frac{\|\delta\|}{d_{opt}} e^{1-\frac{\|\delta\|}{d_{opt}}}, \qquad d_{opt} > 0, \tag{3.6}$$

which has a maximum in d_{opt}. Figure 3.3b represents Function (3.6) with $d_{opt} = 2$. In this way, for the same amplitude of the angles, the maximum of the vision is reached at a distance d_{opt} from the landmarks.
Anyway all the computations that follow are valid for any choice of the function $f(\|\boldsymbol{q} - \boldsymbol{p}\|)$. It is also simpler to incorporate all the terms that depend only on the distance in a unique function:

$$\bar{f}(\|\boldsymbol{q} - \boldsymbol{p}\|) = \frac{f(\|\boldsymbol{q} - \boldsymbol{p}\|)}{\|\boldsymbol{q} - \boldsymbol{p}\|^2}. \tag{3.7}$$

Thus the vision function can be rewritten as:

$$\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \sum_{(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q^V} \bar{f}(\|\boldsymbol{q} - \boldsymbol{p}\|) \, \langle \boldsymbol{q} - \boldsymbol{p}, \hat{\boldsymbol{v}} \rangle \, \langle \boldsymbol{p} - \boldsymbol{q}, \hat{\boldsymbol{u}} \rangle \,. \tag{3.8}$$

Multiagent case and coverage score

Now we consider the most general case, in which there are n sensors for monitoring a set of landmarks Q. In our setup we partition Q in n subsets Q_i:

$$\mathbf{Q} = \{Q_i, \ i = 1, \ldots, n\}, \tag{3.9}$$

such that

$$\begin{cases} \bigcup_{i=1}^{n} Q_i & = Q \\ Q_i \cap Q_j & = \emptyset \quad \forall i \neq j, \end{cases}$$

and we associate every subset to an agent i that is responsible of monitoring them. Therefore every camera has a vision value $\text{vis}(\boldsymbol{p}_i, \hat{\boldsymbol{v}}_i, Q_i)$ that can be computed using equation (3.8). We define the overall vision by summing the single values of every agent:

$$\begin{aligned} \text{cov}(\mathbf{P}, \mathbf{Q}) &= \sum_{i=1,\dots,n} \text{vis}(\boldsymbol{p}_i, \hat{\boldsymbol{v}}_i, Q_i) \\ &= \sum_{i=1,\dots,n} \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}}) \in Q_i^V} \bar{f}(\|\boldsymbol{q} - \boldsymbol{p}_i\|) \langle \boldsymbol{q} - \boldsymbol{p}_i, \hat{\boldsymbol{v}}_i \rangle \langle \boldsymbol{p}_i - \boldsymbol{q}, \hat{\boldsymbol{u}} \rangle , \end{aligned} \tag{3.10}$$

and we call it *coverage score*. In the formula we denoted as $\mathbf{P}$ the set of the poses of all the cameras:

$$\mathbf{P} = \{(\boldsymbol{p}_i, \hat{\boldsymbol{v}}_i),\ i = 1, \dots, n\} .$$

3.2 Problem statement

The aim of our algorithm is to find the configuration of sensors and partition of Q that ensure the best overall vision of the landmarks, so we can express our problem as:

$$\begin{aligned} \underset{\mathbf{P},\mathbf{Q}}{\text{maximize}} \quad & \text{cov}(\mathbf{P}, \mathbf{Q}) \\ \text{subject to} \quad & \mathbf{P} = \{(\boldsymbol{p}_i, \hat{\boldsymbol{v}}_i),\ i = 1, \dots, n\} , \\ & \mathbf{Q} = \{Q_i,\ i = 1, \dots, n\} , \\ & \quad \begin{cases} \bigcup_{i=1}^{n} Q_i & = Q, \\ Q_i \cap Q_j & = \emptyset \quad \forall i \neq j. \end{cases} \end{aligned} \tag{3.11}$$

To do this we will optimize the pose of the single sensors and we will allow the trade of landmarks between couples of agents. In Chapter 4 the algorithm will be explained in detail.

3.3 Gradient computation

The computation of the gradient of the vision function will be useful for the optimization of the camera pose, because we will use a line search algorithm.

We consider each agent singularly, so we omit the index i. Moreover we consider the set of landmarks Q fixed. First we rewrite the vision function defined in (3.5):

$$\begin{aligned}\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) &= \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{q}-\boldsymbol{p}, \hat{\boldsymbol{v}}\rangle\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle \\ &= \left\langle \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big), \hat{\boldsymbol{v}} \right\rangle. \end{aligned} \tag{3.12}$$

Then, we compute the derivative of the vision with respect to time:

$$\begin{aligned}\frac{\partial}{\partial t}\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = &\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big)^\top \frac{\partial \hat{\boldsymbol{v}}}{\partial t} \\ &+ \hat{\boldsymbol{v}}^\top \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \frac{\partial}{\partial t}\Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big). \end{aligned} \tag{3.13}$$

The derivative of $\hat{\boldsymbol{v}}$ can be computed as:

$$\frac{\partial \hat{\boldsymbol{v}}}{\partial t} = \boldsymbol{\omega} \times \hat{\boldsymbol{v}} = \boldsymbol{S}(\hat{\boldsymbol{v}})^\top \boldsymbol{\omega},$$

where $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity of the agent and $\boldsymbol{S}(\hat{\boldsymbol{v}})$ is the skew-symmetric matrix associated to $\hat{\boldsymbol{v}}$ (see Appendix A.2).
Instead the other derivative in (3.13) can be computed as:

$$\begin{aligned}\frac{\partial}{\partial t}\Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big) = &\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big(\frac{\partial}{\partial t}\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\Big) \\ &+ \bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\Big(\frac{\partial}{\partial t}\Big(\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big)\Big), \end{aligned}$$

where:

$$\frac{\partial}{\partial t}\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|) = \bar{f}'(\|\boldsymbol{q}-\boldsymbol{p}\|)\frac{\partial}{\partial t}\Big(\,\|\boldsymbol{q}-\boldsymbol{p}\|\,\Big) = \bar{f}'(\|\boldsymbol{q}-\boldsymbol{p}\|)\frac{(\boldsymbol{p}-\boldsymbol{q})^\top}{\|\boldsymbol{q}-\boldsymbol{p}\|}\dot{\boldsymbol{p}},$$

and

$$\begin{aligned}\frac{\partial}{\partial t}\Big(\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big) &= \frac{\partial}{\partial t}\Big(\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,\Big)(\boldsymbol{q}-\boldsymbol{p}) + \langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,\frac{\partial}{\partial t}\Big(\boldsymbol{q}-\boldsymbol{p}\Big) \\ &= \dot{\boldsymbol{p}}^\top \hat{\boldsymbol{u}}(\boldsymbol{q}-\boldsymbol{p}) + (\boldsymbol{p}-\boldsymbol{q})^\top \hat{\boldsymbol{u}}(-\dot{\boldsymbol{p}}) \\ &= (\boldsymbol{q}-\boldsymbol{p})\hat{\boldsymbol{u}}^\top \dot{\boldsymbol{p}} + (\boldsymbol{q}-\boldsymbol{p})^\top \hat{\boldsymbol{u}}\dot{\boldsymbol{p}} \\ &= \Big((\boldsymbol{q}-\boldsymbol{p})\hat{\boldsymbol{u}}^\top + (\boldsymbol{q}-\boldsymbol{p})^\top \hat{\boldsymbol{u}}\,\boldsymbol{I}\Big)\dot{\boldsymbol{p}}, \end{aligned}$$

in which $\dot{\boldsymbol{p}} \in \mathbb{R}^3$ denotes the linear velocity of the agent and $\bar{f}'(\cdot)$ is the derivative of $\bar{f}(\cdot)$. Finally, joining all the terms and substituting in (3.13) we obtain:

$$\begin{aligned}\frac{\partial}{\partial t}\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = &\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big)^\top \boldsymbol{S}(\hat{\boldsymbol{v}})^\top \boldsymbol{\omega} \\ &+\ \hat{\boldsymbol{v}}^\top \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \left(\frac{\bar{f}'(\|\boldsymbol{q}-\boldsymbol{p}\|)}{\|\boldsymbol{q}-\boldsymbol{p}\|}\,\langle \boldsymbol{q}-\boldsymbol{p}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})(\boldsymbol{q}-\boldsymbol{p})^\top \right. \\ &\left. + \bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\Big((\boldsymbol{q}-\boldsymbol{p})\hat{\boldsymbol{u}}^\top + (\boldsymbol{q}-\boldsymbol{p})^\top \hat{\boldsymbol{u}}\,\boldsymbol{I}\Big)\right)\dot{\boldsymbol{p}} \\ = &\ \nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^\top \boldsymbol{\omega} + \nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^\top \dot{\boldsymbol{p}}, \end{aligned} \tag{3.14}$$

where we have put in evidence the gradient of the vision function with respect to the orientation and the position:

$$\nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)) = \boldsymbol{S}(\hat{\boldsymbol{v}}) \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big), \tag{3.15}$$

$$\begin{aligned}\nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)) = \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} &\left(\frac{\bar{f}'(\|\boldsymbol{q}-\boldsymbol{p}\|)}{\|\boldsymbol{q}-\boldsymbol{p}\|}\,\langle \boldsymbol{q}-\boldsymbol{p}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})(\boldsymbol{q}-\boldsymbol{p})^\top \right. \\ &\left. + \bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\Big(\hat{\boldsymbol{u}}(\boldsymbol{q}-\boldsymbol{p})^\top + (\boldsymbol{q}-\boldsymbol{p})^\top \hat{\boldsymbol{u}}\,\boldsymbol{I}\Big)\right)\hat{\boldsymbol{v}}. \end{aligned} \tag{3.16}$$

Notice that both the gradients depend only on the pose of the camera $(\boldsymbol{p}, \hat{\boldsymbol{v}})$ and the considered set of points Q, and that they are continuous if Q^V is constant. Finally notice that the gradient with respect to $\hat{\boldsymbol{v}}$ is always orthogonal to $\hat{\boldsymbol{v}}$.

3.4 Optimal orientation

From equation (3.12) an optimal value can be derived for $\hat{\boldsymbol{v}}$, while keeping $\boldsymbol{p}$ fixed:

$$\begin{aligned}\hat{\boldsymbol{v}}_{opt}(\boldsymbol{p}, Q) &= \arg\max_{\hat{\boldsymbol{v}}:\|\hat{\boldsymbol{v}}\|=1} \text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) \\ &= \arg\max_{\hat{\boldsymbol{v}}:\|\hat{\boldsymbol{v}}\|=1} \left\langle \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big), \hat{\boldsymbol{v}} \right\rangle \\ &= \frac{\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V_{opt}} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big)}{\left\|\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^V_{opt}} \Big(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\,\langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,(\boldsymbol{q}-\boldsymbol{p})\Big)\right\|}, \end{aligned} \tag{3.17}$$

where we exploited the fact that the scalar product of two vectors is maximum if they are parallel. The major problem that prevents to use this equation is the dependence of Q^V on the orientation. Indeed finding $\hat{\boldsymbol{v}}_{opt}$ requires to know Q^V_{opt},

Algorithm 1: Finding $\hat{\boldsymbol{v}}_{opt}$ trying all the possible combinations of landmarks.

Data: $\boldsymbol{p}$, Q
Result: Optimal orientation $\hat{\boldsymbol{v}}_{opt}(\boldsymbol{p}, Q)$.
$M = 0$;
$\mathcal{P}(Q)$ = set of all possible subsets of Q;
foreach $Q^\star \in \mathcal{P}(Q)$ **do**
  $\hat{\boldsymbol{v}}^\star = \dfrac{\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^\star}\left(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\langle \boldsymbol{p}-\boldsymbol{q},\hat{\boldsymbol{u}}\rangle(\boldsymbol{q}-\boldsymbol{p})\right)}{\left\|\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q^\star}\left(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|)\langle \boldsymbol{p}-\boldsymbol{q},\hat{\boldsymbol{u}}\rangle(\boldsymbol{q}-\boldsymbol{p})\right)\right\|}$;
  $Q^V_{\hat{\boldsymbol{v}}^\star}$ = set of visible points from $(\boldsymbol{p}, \hat{\boldsymbol{v}}^\star)$;
  if $\left(Q^V_{\hat{\boldsymbol{v}}^\star} == Q^\star\right)$ *AND* $(vis(\boldsymbol{p}, \hat{\boldsymbol{v}}^\star, Q^\star) > M)$ **then**
    $\hat{\boldsymbol{v}}_{\boldsymbol{opt}} = \hat{\boldsymbol{v}}^\star$;
    $M = \mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}^\star, Q^\star)$;
  end
end

that is the set of visible landmarks when the camera is in $(\boldsymbol{p}, \hat{\boldsymbol{v}}_{opt})$, which in turn depends on $\hat{\boldsymbol{v}}_{opt}$. Therefore the formula cannot be applied directly.

One possible solution is given in Algorithm 1. The idea is to pretend to know already the set Q^V_{opt}, we call it $Q^\star$. Then we can compute the optimal orientation $\hat{\boldsymbol{v}}^\star$ using equation (3.17), where we substitute Q^V_{opt} with $Q^\star$. Then we check if our guess was correct: if $Q^\star$ coincides with the set of points that are visible from $(\boldsymbol{p}, \hat{\boldsymbol{v}}^\star)$ then $\hat{\boldsymbol{v}}^\star$ is a candidate to be the optimal orientation. If we do this procedure for all the possible combination of landmarks $Q^\star$ and choose the candidate with the maximum value of vision, we can find $\hat{\boldsymbol{v}}_{opt}(\boldsymbol{p}, Q)$.

The issue of this solution is that, if we are dealing with N landmarks, there are 2^N possible combinations that we have to try. Thus the number of iterations of our algorithm grows exponentially with the number of points that we are monitoring, so it is highly inefficient. This is why in our algorithm we will change our orientation following an angular velocity that guarantees the increase of the vision, instead of computing at every step the optimal orientation and rotating with the objective of aligning $\hat{\boldsymbol{v}}$ with $\hat{\boldsymbol{v}}_{opt}(\boldsymbol{p}, Q)$. Anyway, as we would expect, $(\boldsymbol{p}, \hat{\boldsymbol{v}}_{opt})$ is a stationary

point for the rotation, in the sense that:

$$\begin{aligned}\nabla_{\hat{\boldsymbol{v}}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}_{\boldsymbol{opt}}, Q^V_{opt})) &= \hat{\boldsymbol{v}}_{opt} \times \left(\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}}) \in Q^V_{opt}} \left(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|) \langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}} \rangle (\boldsymbol{q}-\boldsymbol{p}) \right) \right) \\ &= \left(\frac{\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}}) \in Q^V_{opt}} \left(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|) \langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}} \rangle (\boldsymbol{q}-\boldsymbol{p}) \right)}{\left\| \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}}) \in Q^V_{opt}} \left(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|) \langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}} \rangle (\boldsymbol{q}-\boldsymbol{p}) \right) \right\|} \right) \\ &\quad \times \left(\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}}) \in Q^V_{opt}} \left(\bar{f}(\|\boldsymbol{q}-\boldsymbol{p}\|) \langle \boldsymbol{p}-\boldsymbol{q}, \hat{\boldsymbol{u}} \rangle (\boldsymbol{q}-\boldsymbol{p}) \right) \right) = 0,\end{aligned} \tag{3.18}$$

because it is the cross product of two parallel vectors. As a consequence the time derivative of the vision is:

$$\frac{\partial}{\partial t} \mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}_{\boldsymbol{opt}}, Q^V_{opt}) = \nabla_{\boldsymbol{p}} (\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}_{\boldsymbol{opt}}, Q^V_{opt}))^\top \dot{\boldsymbol{p}},$$

which means that there is no angular velocity that leads to an increase of the vision.

Chapter 4

Coverage algorithm

In this section we describe the algorithm that we use to solve the optimization problem defined in Section 3.2:

$$\underset{\mathbf{P},\mathbf{Q}}{\text{maximize}} \ \text{cov}(\mathbf{P}, \mathbf{Q}), \tag{4.1}$$

subject to:

$$\mathbf{P} = \{(\boldsymbol{p}_i, \hat{\boldsymbol{v}}_i),\ i = 1, \ldots, n\},$$
$$\mathbf{Q} = \{Q_i,\ i = 1, \ldots, n\},$$

with

$$\begin{cases} \bigcup_{i=1}^{n} Q_i & = Q, \\ Q_i \cap Q_j & = \emptyset \quad \forall i \neq j. \end{cases}$$

Moreover, since we will work in a bounded environment Ω (that we call *mission space*) with multiple agents, we also add some constraints for the position in order to avoid collisions:

$$\begin{cases} \boldsymbol{p}_i(t) \in \Omega & \forall i,\ \forall t, \\ \left\| \boldsymbol{p}_i(t) - \boldsymbol{p}_j(t) \right\| >= R_s & \forall i \neq j,\ \forall t, \end{cases}$$

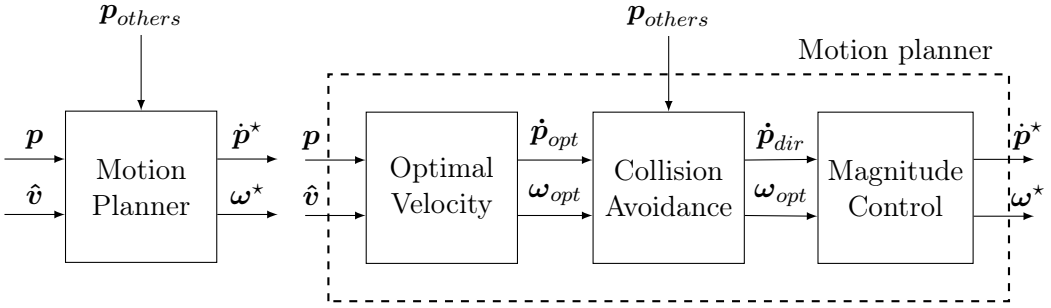
where t is the time. The first condition states that all the agents must remain inside the mission space, while the second states that the reciprocal distance between two agents cannot be lower than a *safety radius* R_s.

The algorithm that we propose consists of three main parts:

1. Initialization;
2. Pose optimization;
3. Trading of landmarks.

The first phase consists in the selection of the initial poses for the cameras and the set of landmarks associated to each agent (Section 4.1).

Then for every agent the pose is optimized, meaning that the value of the vision of the camera on its landmark set is maximized. This is done through a line search algorithm based on the gradient and implemented in a block that we call *motion planner*. It is basically a function that takes as input the position and the orientation of the camera and returns the linear and angular velocities that we want the camera to follow, as represented in Figure 4.1a. We can divide the motion planner into simpler components: the computation of the optimal velocity (explained in Section 4.2) simply computes the directions of linear and angular velocity that guarantee the maximum increase of the vision, and that will be followed if there is no obstacle in the path; the collision avoidance (explained in Section 4.3) takes into account the presence of other agents or boundaries of the mission space and chooses a direction for the linear velocity that guarantees safety and also an increase of the vision; the magnitude control (explained in Section 4.4) chooses a suitable magnitude for the velocity considering the limits of the physical system and ensuring the convergence. Once an agent has reached the optimal position, it becomes available for trading. This means that it will try to communicate with other agents and to exchange its landmarks. The communications are one-to-one, and the algorithm used is described in section 4.5. If a trade is performed successfully, then a new pose optimization is started, but now considering the new set of landmarks.



(a) Motion planner block. (b) Different parts in which we divide the motion planner.

Figure 4.1

4.1 Initialization

We consider a team of n agents with initial poses:

$$\mathbf{P}_0 = \left\{(\boldsymbol{p}_{0,i}, \hat{\boldsymbol{v}}_{0,i}),\ i = 1, \ldots, n\right\},$$

and a set of landmarks:

$$Q = \left\{(\boldsymbol{q}_j, \boldsymbol{u}_j),\ j = 1, \ldots, m\right\}.$$

partitioned initially as:

$$\mathbf{Q}_0 = \{Q_{0,i},\ i = 1, \ldots, n\},$$

where each $Q_{0,i}$ is associated to the corresponding agent. The initial conditions must respect the constraints:

$$\begin{cases} \bigcup_{i=1}^{n} Q_{0,i} = Q, & \\ Q_{0,i} \cap Q_{0,j} = \emptyset & \forall i \neq j, \\ \boldsymbol{p}_{0,i} \in \Omega & \forall i, \\ \left\| \boldsymbol{p}_{0,i} - \boldsymbol{p}_{0,j} \right\| >= R_s & \forall i \neq j. \end{cases}$$

This phase is very important because in general different initial conditions can lead to very different results, as will be clearer in simulations (see Chapter 5). Once assigned the initial sets of landmarks, each agent performs the pose optimization using a gradient-based algorithm.

4.2 Optimal velocity computation

This part is executed by every agent independently, therefore we can drop the notation relative to the identity of the agent. The first step is to compute the set of visible landmarks:

$$Q^V = \{(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q \mid \langle \boldsymbol{q} - \boldsymbol{p}, \hat{\boldsymbol{v}} \rangle > 0, \langle \boldsymbol{p} - \boldsymbol{q}, \hat{\boldsymbol{u}} \rangle > 0\}.$$

Then the gradient of the vision with respect to the position and the orientation is computed using the formulas (3.15) and (3.16), that are reported here:

$$\nabla_{\hat{\boldsymbol{v}}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)) = \boldsymbol{S}(\hat{\boldsymbol{v}}) \sum_{(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q^V} \Big(\bar{f}(\|\boldsymbol{q} - \boldsymbol{p}\|) \langle \boldsymbol{p} - \boldsymbol{q}, \hat{\boldsymbol{u}} \rangle (\boldsymbol{q} - \boldsymbol{p}) \Big),$$

$$\nabla_{\boldsymbol{p}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)) = \sum_{(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q^V} \Bigg(\frac{\bar{f}'(\|\boldsymbol{q} - \boldsymbol{p}\|)}{\|\boldsymbol{q} - \boldsymbol{p}\|} \langle \boldsymbol{q} - \boldsymbol{p}, \hat{\boldsymbol{u}} \rangle (\boldsymbol{q} - \boldsymbol{p})(\boldsymbol{q} - \boldsymbol{p})^\top + \bar{f}(\|\boldsymbol{q} - \boldsymbol{p}\|) \Big(\hat{\boldsymbol{u}} (\boldsymbol{q} - \boldsymbol{p})^\top + (\boldsymbol{q} - \boldsymbol{p})^\top \hat{\boldsymbol{u}}\, \boldsymbol{I} \Big) \Bigg) \hat{\boldsymbol{v}}.$$

We define the following sets of velocities:

$$ND(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \Big\{ (\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{R}^3 \times \mathbb{R}^3 \mid \nabla_{\boldsymbol{p}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^\top \boldsymbol{a} + \nabla_{\hat{\boldsymbol{v}}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^\top \boldsymbol{b} \geq 0 \Big\},$$

$$ND_{\dot{\boldsymbol{p}}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \Big\{ \boldsymbol{a} \in \mathbb{R}^3 \mid \nabla_{\boldsymbol{p}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^\top \boldsymbol{a} \geq 0 \Big\},$$

$$ND_{\boldsymbol{\omega}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \Big\{ \boldsymbol{b} \in \mathbb{R}^3 \mid \nabla_{\hat{\boldsymbol{v}}}(\mathrm{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^\top \boldsymbol{b} \geq 0 \Big\}.$$

The first is the set of *non-decreasing velocities*: if $(\dot{\boldsymbol{p}}, \boldsymbol{\omega}) \in ND(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)$, then moving according to this velocities will lead to an increase (or a maintenance) of the value

of the vision function. The second set is the one of *non-decreasing linear velocities*: if $\dot{\boldsymbol{p}} \in ND_{\dot{\boldsymbol{p}}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)$, then moving according to this $\dot{\boldsymbol{p}}$ while keeping $\boldsymbol{\omega} = \mathbf{0}$ will lead to an increase (or a maintenance) of the value of the vision function. Finally, $ND_{\boldsymbol{\omega}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)$ is the set of *non-decreasing angular velocities* and is analogous to $ND_{\dot{\boldsymbol{p}}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q)$, switching the role of $\dot{\boldsymbol{p}}$ and $\boldsymbol{\omega}$. It is easy to notice that:

$$ND_{\dot{\boldsymbol{p}}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) \times ND_{\boldsymbol{\omega}}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) \subseteq ND(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q). \tag{4.2}$$

Therefore, assuming that $\dot{\boldsymbol{p}}$ and $\boldsymbol{\omega}$ can be controlled independently, it is sufficient to choose them in the sets of non-decreasing linear and angular velocities respectively. So we impose:

$$\begin{cases} \nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^{\top} \dot{\boldsymbol{p}} \geq 0, \\ \nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^{\top} \boldsymbol{\omega} \geq 0, \end{cases} \tag{4.3}$$

which means that $\dot{\boldsymbol{p}}$ has to be in the half-space defined by the vector the gradient with respect to the position, and analogously for $\boldsymbol{\omega}$, if we work in $\mathbb{R}^3$ (see Figure 4.2). Moreover, we can always choose $\boldsymbol{\omega} \perp \hat{\boldsymbol{v}}$, indeed:

$$\frac{\partial \hat{\boldsymbol{v}}}{\partial t} = \boldsymbol{\omega} \times \hat{\boldsymbol{v}} = \left(\boldsymbol{\omega}_{\perp} + \boldsymbol{\omega}_{\parallel}\right) \times \hat{\boldsymbol{v}} = \boldsymbol{\omega}_{\perp} \times \hat{\boldsymbol{v}}, \tag{4.4}$$

where $\boldsymbol{\omega}_{\parallel}$ and $\boldsymbol{\omega}_{\perp}$ are the components of $\boldsymbol{\omega}$ respectively in the direction parallel and perpendicular to $\hat{\boldsymbol{v}}$. For the properties of the cross product the parallel component does not affect the result, thus can be always chosen equal to zero. Notice that $\nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))$ is given by the cross product of $\hat{\boldsymbol{v}}$ and a sum of vectors and therefore is always already perpendicular to $\hat{\boldsymbol{v}}$. Obviously this reasoning holds only in $\mathbb{R}^3$, since in $\mathbb{R}^2$ the angular velocity is a scalar quantity.

Finally, we define the directions of $\dot{\boldsymbol{p}}_{opt}$ and $\boldsymbol{\omega}_{opt}$, that are the ones that give the maximum increase of the vision function:

$$\begin{aligned} \dot{\boldsymbol{p}}_{opt} &= \underset{\dot{\boldsymbol{p}}:\|\dot{\boldsymbol{p}}\|=1}{\arg\max}\ \nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^{\top} \dot{\boldsymbol{p}} = \frac{\nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))}{\|\nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))\|}, \\ \boldsymbol{\omega}_{opt} &= \underset{\boldsymbol{\omega}:\|\boldsymbol{\omega}\|=1}{\arg\max}\ \nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))^{\top} \boldsymbol{\omega} = \frac{\nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))}{\|\nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))\|}, \\ &\underset{(\dot{\boldsymbol{p}},\boldsymbol{\omega}):\|\dot{\boldsymbol{p}}\|=\|\boldsymbol{\omega}\|=1}{\arg\max} \frac{\partial}{\partial t}\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q) = \left(\dot{\boldsymbol{p}}_{opt}, \boldsymbol{\omega}_{opt}\right), \end{aligned} \tag{4.5}$$

where we considered velocities with unitary norm because we are interested only in the direction of the movement, regardless its intensity.
Moreover the maximum instantaneous increase of the vision function, still considering unitary normed velocities, is:

$$\begin{aligned} &\max_{\|\dot{\boldsymbol{p}}\|=\|\boldsymbol{\omega}\|=1} \text{vis}\left(\boldsymbol{p}(t+\partial t), \hat{\boldsymbol{v}}(t+\partial t), Q\right) = \\ &\qquad \text{vis}(\boldsymbol{p}(t), \hat{\boldsymbol{v}}(t), Q) + \|\nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}(t), \hat{\boldsymbol{v}}(t), Q))\|\, \partial t + \|\nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}(t), \hat{\boldsymbol{v}}(t), Q))\|\, \partial t. \end{aligned}$$

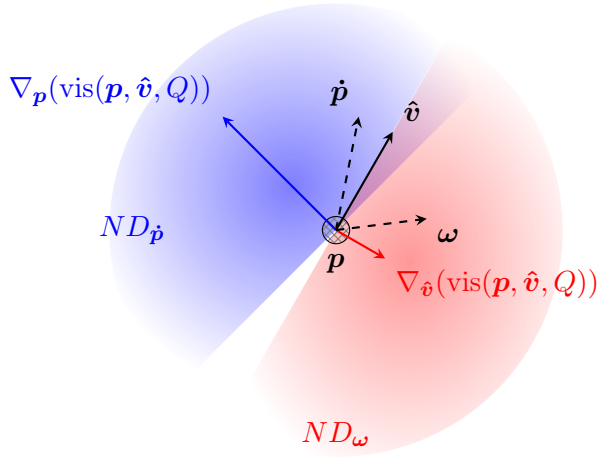


Figure 4.2: Example of choice of non decreasing velocities.

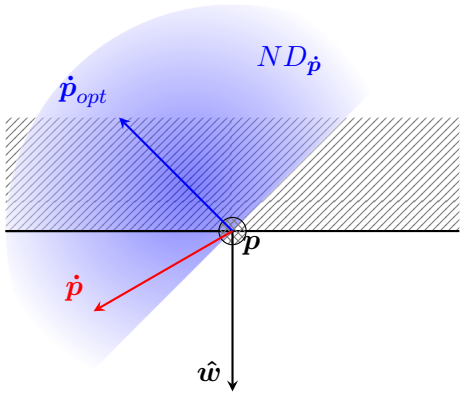
4.3 Collision avoidance

Since we work in a finite environment, potentially with multiple agents, we cannot always move freely, because we would risk to exit the mission space or to collide with the others. Therefore there are situations in which we cannot choose to follow the optimal direction, defined by $\dot{\boldsymbol{p}}_{opt}$. Instead, for the angular velocity, we will always choose the direction defined by $\boldsymbol{\omega}_{opt}$, because we assume that the rotation is independent from the position.

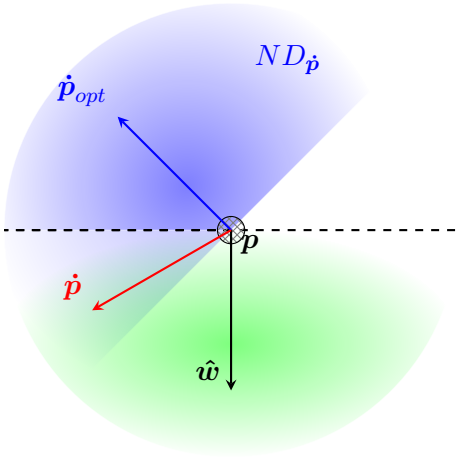
In this section we will propose two possible techniques for ensuring the collision avoidance. Both of them consider only the direction of the velocity, regardless its magnitude, and return a direction that, at least in short term, does not lead to a collision. The two methods differ for the concept of *safe velocity* from which they start, but are both expressed as maximization problems, in the sense that they search a velocity that does not lead to a collision but also ensures the maximum increment of the vision function. We will refer at the first technique as *abrupt*, because it leads to sharp changes of the velocity direction, and at the second as *smooth*, because it extends the first method to have a more continuous change of direction.

Abrupt collision avoidance

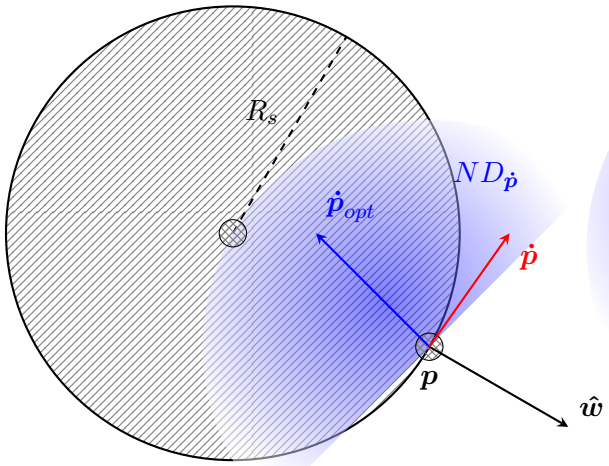
The obstacles that we consider are of two types: the limits of the mission space or other agents. Figure 4.3a describes the case where the agent is at one of the boundaries of the mission space (that we will call *walls*), it is not possible to move in the optimal direction. On the other hand, there are still non decreasing velocities that are allowed, for instance the velocity $\dot{\boldsymbol{p}}$ drawn in the figure can be considered safe because it does not bring the agent outside the mission space. The same holds in the case represented in Figure 4.3c, where the agent is dangerously close to another



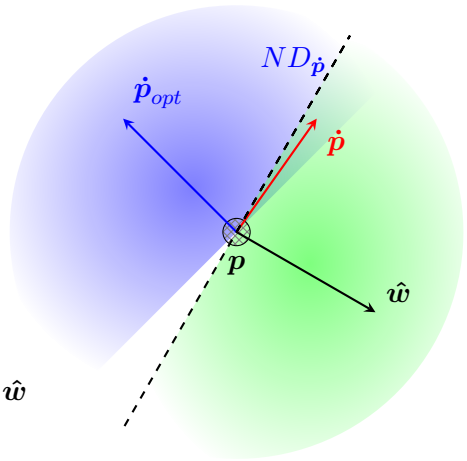
(a) Agent at the boundary of the mission space.



(b) Safe velocities in green.



(c) Agent at distance R_s from another agent.



(d) Safe velocities in green.

Figure 4.3: Safe velocities in case of imminent collisions.

agent, i.e. their distance is lower than the safety radius. Therefore in this case we consider *safe* all the velocities that do not lead to a decrease of the distance between the agent and the obstacle. Moreover we consider an obstacle only when the agent is at its boundary.

In order to translate into formulas the concept of safe velocity, we define a unit vector $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ associated to every obstacle. In the case of a wall $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ will be the normal vector to the surface and in the case of another agent it will be directed as the line

joining the two agents and oriented repulsively. A velocity will be acceptable if it is both in $ND_{\dot{\boldsymbol{p}}}$ and safe:

$$\begin{cases} \langle \dot{\boldsymbol{p}}_{opt}, \dot{\boldsymbol{p}} \rangle \geq 0, \\ \langle \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \dot{\boldsymbol{p}} \rangle \geq 0 \quad i = 1, \dots, n_o, \end{cases} \tag{4.6}$$

where n_o is the number of obstacles. Moreover, in the case there are multiple acceptable velocities we want to choose the one that is maximizing the increase of the vision function, that is the one that gives the maximum scalar product with $\dot{\boldsymbol{p}}_{opt}$. Notice that if we find an acceptable velocity, then all the others that are parallel but with different magnitude are acceptable as well. For this reason we can consider only unitary velocities, and we will deal later with the magnitude. Given these premises we can express our problem as an optimization problem:

$$\begin{aligned} \underset{\dot{\boldsymbol{p}}}{\text{maximize}} \quad & \langle \dot{\boldsymbol{p}}_{opt}, \dot{\boldsymbol{p}} \rangle, \\ \text{subject to} \quad & \langle \dot{\boldsymbol{p}}_{opt}, \dot{\boldsymbol{p}} \rangle \geq 0, \\ & \langle \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \dot{\boldsymbol{p}} \rangle \geq 0 \qquad i = 1, \dots, n_o, \\ & \|\dot{\boldsymbol{p}}\| = 1. \end{aligned} \tag{4.7}$$

Notice that considering only unitary velocities makes the problem nonlinear, so it is not possible to use well known techniques of linear programming, like the simplex algorithm or similar. On the other hand removing this constraint would lead to an unbounded problem, because any solution could be improved just by increasing the magnitude.

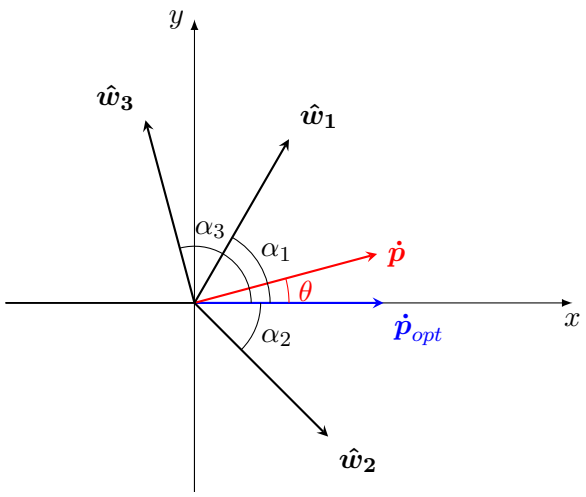
Note that all the inequalities of the problem (4.7) can be thought geometrically as requiring that $\dot{\boldsymbol{p}}$ must be in a half plan (if we are in $\mathbb{R}^2$) or in a half space (if we are in $\mathbb{R}^3$). In particular, the first inequality states that $\dot{\boldsymbol{p}}$ has to be in $ND_{\dot{\boldsymbol{p}}}$, and the other n_o inequalities state that it has to be in the half plan or half space that does not contain the obstacle. For defining a half plan in $\mathbb{R}^2$ we will us the line that establishes its boundary, which is the one orthogonal to $\hat{\boldsymbol{w}}$. Analogously in $\mathbb{R}^3$ for defining a half space we will use the plan orthogonal to $\hat{\boldsymbol{w}}$.

Proposition 1. *The solution of the problem* (4.7) *in* $\mathbb{R}^2$ *is either:*

- $\dot{\boldsymbol{p}}_{opt}$*;*
- *the projection (normalized) of* $\dot{\boldsymbol{p}}_{opt}$ *on the line orthogonal to* $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ *for some* $i = 1, \dots, n_o$*.*

Proof. Without loss of generality, we consider $\dot{\boldsymbol{p}}_{opt}$ aligned with the x axis of our coordinate frame. We can define the angles formed by each vector with the x axis: θ is the angle associated with the generic $\dot{\boldsymbol{p}}$ and α_i is associated with $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$, as in Figure 4.4a. The optimization problem (4.7) can be rewritten in terms of the angles as:

$$\arg\max_{\theta} \ \cos(\theta) = \arg\min_{\theta} \ \|\theta\|$$



(a) Representation of the angles α_i and θ.

where

$$\begin{cases} \cos(\theta) \geq 0 \\ \cos(\alpha_i - \theta) \geq 0 \end{cases}$$

The constraints are equivalent to:

$$\begin{cases} -\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2} \\ \alpha_i - \frac{\pi}{2} \leq \theta \leq \alpha_i + \frac{\pi}{2} \end{cases}$$

Now we can split the problem into two sub-problems:

$$\arg\min_{\theta} \ \theta \qquad \arg\min_{\theta} \ -\theta$$
$$\begin{cases} 0 \leq \theta \leq \frac{\pi}{2} \\ \alpha_i - \frac{\pi}{2} \leq \theta \leq \alpha_i + \frac{\pi}{2} \end{cases} \qquad \begin{cases} -\frac{\pi}{2} \leq \theta \leq 0 \\ \alpha_i - \frac{\pi}{2} \leq \theta \leq \alpha_i + \frac{\pi}{2} \end{cases}$$

and the solution of the initial problem will be the minimum of the two single solutions. Since both the sub-problems are problems of linear optimization we can exploit a general fact, valid for this kind of problems: the optimum value is always attained on the boundary of the feasible set. This means that at least one of the constraints must be tight, i.e. an equality. Therefore the possible solutions are:

$$\begin{aligned} \theta = 0 \quad &\Rightarrow \quad \dot{\boldsymbol{p}} = \dot{\boldsymbol{p}}_{opt}, \\ \theta = \pm\frac{\pi}{2} \quad &\Rightarrow \quad \langle \dot{\boldsymbol{p}}_{opt}, \dot{\boldsymbol{p}} \rangle = 0, \\ \theta = \alpha_i \pm \frac{\pi}{2} \quad &\Rightarrow \quad \langle \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \dot{\boldsymbol{p}} \rangle = 0. \end{aligned} \tag{4.8}$$

The last case means that $\dot{\boldsymbol{p}}$ must be on the line orthogonal to $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$, so in principle there could be two chances (the two directions of the line), one with positive scalar product

with $\dot{\boldsymbol{p}}_{opt}$ (the one that we are searching) and the other with negative product. To find the first one it is sufficient to consider the projection of $\dot{\boldsymbol{p}}_{opt}$ on the line, and the normalize it. The only case in which we have to consider both the directions of the line is when $\hat{\boldsymbol{w}}_{\boldsymbol{i}} = -\dot{\boldsymbol{p}}_{opt}$, because they both have null scalar product with $\dot{\boldsymbol{p}}_{opt}$, and this is exactly equivalent to the second case of (4.8). □

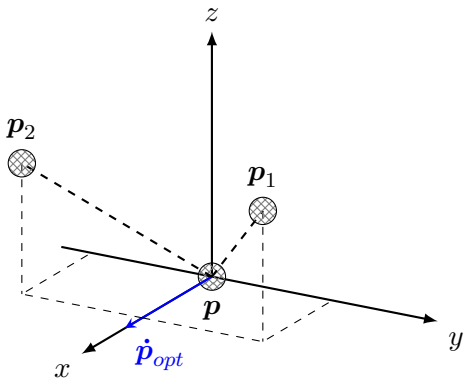
The Proposition 1 implies that the optimal solution can be found in a finite set whose dimension is linear with the number of obstacles n_o. Algorithm 2 implements the solution searching the element of this set that gives the maximum value: if $\dot{\boldsymbol{p}}_{opt}$ is feasible than it is the optimal solution, otherwise we try all the possible projections. Although there may exist more efficient algorithms to solve 4.7, note that the complexity of the proposed algorithm is polynomial (namely, $O(n_o)$) because every iteration of the loop can be done in $O(1)$. In the algorithm we use the function projection $(\dot{\boldsymbol{p}}_{opt}, \hat{\boldsymbol{w}}_{\boldsymbol{i}})$ that returns the normalized projection of the vector $\dot{\boldsymbol{p}}_{opt}$ on the line orthogonal to $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$.

Algorithm 2: Finding the optimal solution of problem (4.7) in $\mathbb{R}^2$, exploiting Proposition 1.

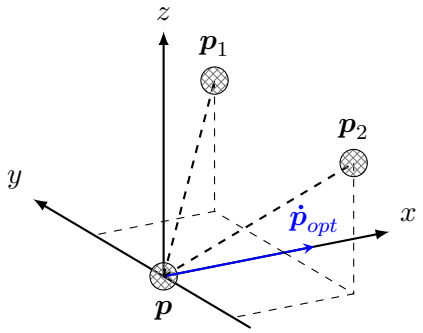
Data: $\dot{\boldsymbol{p}}_{opt}$, $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ $i = 1, \ldots, n_o$
Result: Optimal acceptable velocity $\dot{\boldsymbol{p}}$.
if $\left(\left\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \dot{\boldsymbol{p}}_{opt} \right\rangle \geq 0 \quad \forall k\right)$ **then**
  $\dot{\boldsymbol{p}} = \dot{\boldsymbol{p}}_{opt}$;
  return;
end
$max = 0$;
$\dot{\boldsymbol{p}} = 0$;
for $i = 1, \ldots, n_o$ **do**
  $\boldsymbol{proj}$ = projection$(\dot{\boldsymbol{p}}_{opt}, \hat{\boldsymbol{w}}_{\boldsymbol{i}})$;
  if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{proj} \rangle \geq 0 \quad \forall k)$ *AND* $\left(\left\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj} \right\rangle \geq max\right)$ **then**
    $\dot{\boldsymbol{p}} = \boldsymbol{proj}$;
    $max = \left\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj} \right\rangle$;
  end
end
return;

Note that Proposition 1 does not hold for all the possible configurations in $\mathbb{R}^3$. Consider for instance the case proposed in Figure 4.5, where there are two obstacles. In this case that each of the projection of $\dot{\boldsymbol{p}}_{opt}$ on one of the planes is excluded by the other constraint. Nevertheless it is easy to see that $\begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^\top$ is an acceptable velocity, even if it is not the optimal one (it is orthogonal to $\dot{\boldsymbol{p}}_{opt}$ so it does not lead to any instantaneous increase of the vision, but neither a decrease).

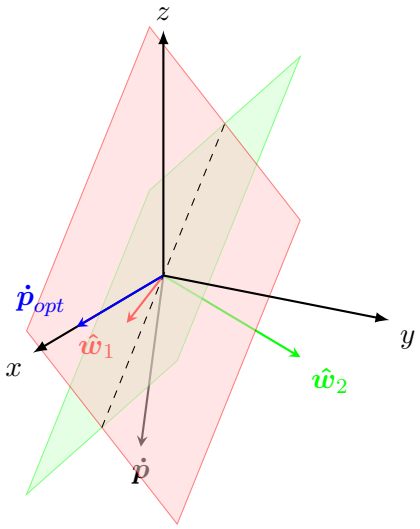
Conjecture 1. *The solution of the problem* (4.7) *in* $\mathbb{R}^3$ *is either:*



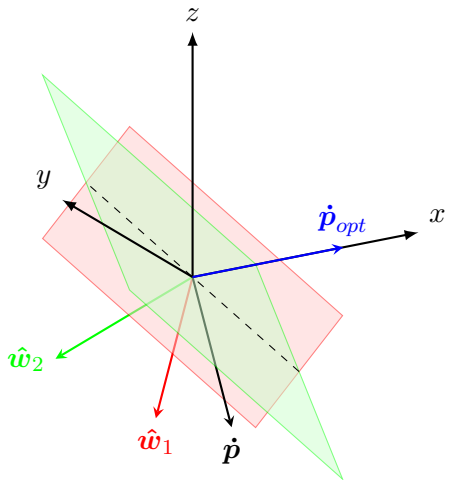
(a) Obstacles in $\mathbb{R}^3$, first view.



(b) Obstacles in $\mathbb{R}^3$, second view.



(c) Planes associated to obstacles, first view.



(d) Planes associated to obstacles, second view.

Figure 4.5

- $\dot{\boldsymbol{p}}_{opt}$;
- *the projection (normalized) of* $\dot{\boldsymbol{p}}_{opt}$ *on the plane orthogonal to* $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ *for some* $i = 1, \ldots, n_o$;
- *the unit vector that belongs to the intersection of the two planes orthogonal to* $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ *and* $\hat{\boldsymbol{w}}_{\boldsymbol{j}}$ *for some* $i, j = 1, \ldots, n_o$, $i \neq j$, *and has positive scalar product with* $\dot{\boldsymbol{p}}_{opt}$.

To corroborate Conjecture 1, we tried to find contradictions by solving problems with randomly generated data, both exploiting the conjecture (Algorithm 3 shows the pseudocode) and using the optimization tool *fmincon* provided by MATLAB, and

we did not find any. The algorithm that we propose is efficient, since its complexity is $O(n_o^2)$, and can never return an infeasible solution. Therefore, even if Fact 1 was false in some particular conditions that we did not find in testing, it could only happen that a solution exists but we do not find it with our algorithm, or we find a solution that is feasible but not optimal. The algorithm will never return a velocity that is decreasing the vision, or is not safe (according to the criterion that we use).

Algorithm 3: Finding the optimal solution of (4.7) in $\mathbb{R}^3$ exploiting Conjecture 1.

Data: $\dot{\boldsymbol{p}}_{opt}$, $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ $i = 1, \ldots, n_o$
Result: Optimal acceptable velocity $\dot{\boldsymbol{p}}$.
if $\left(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \dot{\boldsymbol{p}}_{opt} \rangle \geq 0 \quad \forall k\right)$ **then**
  $\dot{\boldsymbol{p}} = \dot{\boldsymbol{p}}_{opt}$;
  return;
end
$max = 0$;
$\dot{\boldsymbol{p}} = 0$;
for $i = 1, \ldots, n_o$ **do**
  $\boldsymbol{proj}$ = projection$(\dot{\boldsymbol{p}}_{opt}, \hat{\boldsymbol{w}}_{\boldsymbol{i}})$;
  if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{proj} \rangle \geq 0 \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj} \rangle \geq max\right)$ **then**
    $\dot{\boldsymbol{p}} = \boldsymbol{proj}$;
    $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj} \rangle$;
  end
  for $j = i + 1, \ldots, n_o$ **do**
    $\boldsymbol{inters}$ = intersection$(\hat{\boldsymbol{w}}_{\boldsymbol{i}}, \hat{\boldsymbol{w}}_{\boldsymbol{j}})$;
    if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{inters} \rangle \geq 0 \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{inters} \rangle \geq max\right)$ **then**
      $\dot{\boldsymbol{p}} = \boldsymbol{inters}$;
      $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{inters} \rangle$;
    end
  end
end
return;

Smooth collision avoidance

The problem with the method used till now is that it starts worrying of a collision when the agent is already at the boundary of the safe area. So it may happen that at a certain moment it is completely free to move in any direction and immediately after it has to change completely the velocity to avoid an obstacle. For preventing this condition, we set a worrying distance D_w. If the agent is closer than D_w to an obstacle then it starts to consider it. For all these obstacles we add one constraint

of the type:

$$\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \dot{\boldsymbol{p}} \rangle \geq \lambda_i, \tag{4.9}$$

where λ_i is a value in $[-1, 0]$ computed as:

$$\lambda_i = -\frac{d_i}{D_w}, \tag{4.10}$$

where d_i is the distance from the i-th obstacle. In Figure 4.6 it is shown the meaning of the constraint. Figure 4.6a represent the case in which the agent is close (meaning $d < D_w$) to a wall. The distance is computed orthogonally to the bound. As can be seen in Figure 4.6b, the geometrical meaning of the constraint is that the velocity $\dot{\boldsymbol{p}}$ must form with $\hat{\boldsymbol{w}}$ an angle of amplitude smaller than $\arccos(\lambda) \in [\pi/2, \pi]$. The farther the agent is from the obstacle, and the wider will be the angle, reaching the value π when $d = D_w$, which means that the $\dot{\boldsymbol{p}}$ can be any vector (as long as it does not turn in a decrease of the vision). The closer the agent gets to the obstacle and the narrower the angle will be, until it reaches $\pi/2$ when $d = 0$. In this case the angle becomes a half plan, and thus we are back in the situation presented previously. Figures 4.6c and 4.6d present the case of near collision between two agents. The only difference in this case is the fact that the distance is measured on the line joining the position of the two agents.

The mechanism can also be seen, more intuitively, in the opposite way: as the agent approaches the obstacle, the angle of unsafe velocities becomes wider, starting from an amplitude of zero and reaching $\pi/2$, which means that the only velocities that are allowed are the ones that make the agent draw apart from the obstacle.

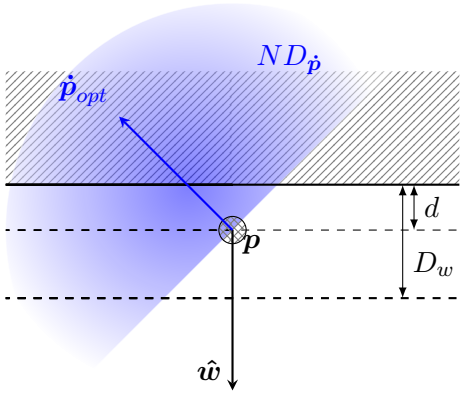
We can express the problem of finding the best direction for the velocity as a constrained optimization problem similar to (4.7) but with the new constraints:

$$\begin{aligned} \underset{\dot{\boldsymbol{p}}}{\text{maximize}} \quad & \left\langle \dot{\boldsymbol{p}}_{opt}, \dot{\boldsymbol{p}} \right\rangle, \\ \text{subject to} \quad & \left\langle \dot{\boldsymbol{p}}_{opt}, \dot{\boldsymbol{p}} \right\rangle \geq 0, \\ & \langle \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \dot{\boldsymbol{p}} \rangle \geq \lambda_i \qquad i = 1, \ldots, n_o, \\ & \|\dot{\boldsymbol{p}}\| = 1. \end{aligned} \tag{4.11}$$

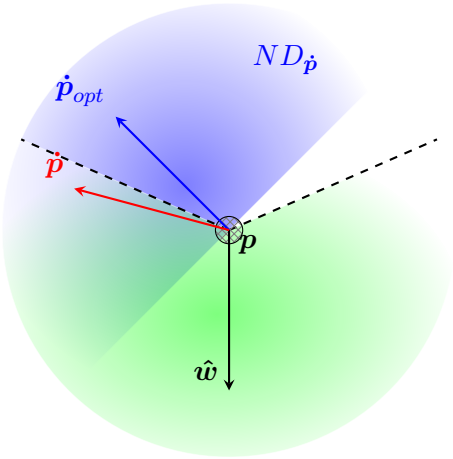
Proposition 2. *The solution of the problem* (4.7) *in* $\mathbb{R}^2$ *is either:*

- $\dot{\boldsymbol{p}}_{opt}$*;*
- *the projection (normalized) of* $\dot{\boldsymbol{p}}_{opt}$ *on one of the two rays that delimit the angle associated with one of the constraints* $\langle \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \dot{\boldsymbol{p}} \rangle \geq \lambda_i$*.*

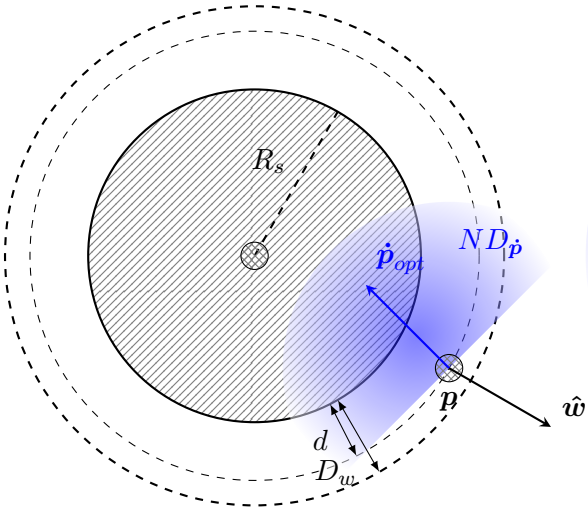
The proof of the Proposition 2 is analogous to the one of Proposition 1. Notice that generally we cannot exclude one of the projections of $\dot{\boldsymbol{p}}_{opt}$ on the two rays of the angle, so we will always consider both of them and we will check if they are feasible. The Algorithm 4 is analogous to Algorithm 2: we consider all the possible solutions and we search the one that guarantees the best increment of the vision.



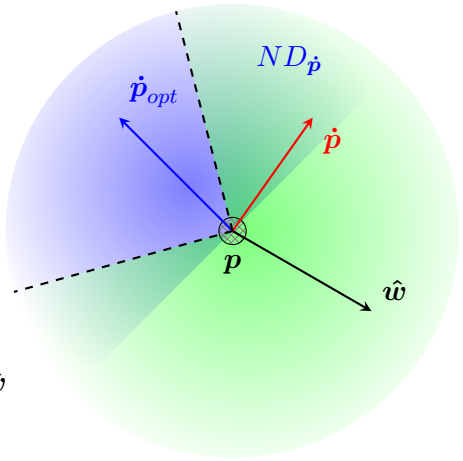
(a) Agent at the boundary of the mission space.



(b) Safe velocities in green.



(c) Agent at distance R_s from another agent.



(d) Safe velocities in green.

Figure 4.6: Safe velocities in case of near collisions.

The extension of Proposition 2 to $\mathbb{R}^3$ is similar to the one explained before, the only difference is that instead of dealing with angles, we deal with cones. So in this case when we approach an obstacle there will be a growing cone of velocities that cannot be selected. The axis of the cone will be orthogonal to the wall if the obstacle is a wall, and the line joining the two agents if the obstacle is another agent. The opening angle of the cone can be computed as $\arccos(-\lambda)$ (the minus sign is due to the fact that we are considering the velocities that we exclude). Analogously to

Algorithm 4: Finding the optimal solution of problem (4.11) in $\mathbb{R}^2$, exploiting Proposition 2.

Data: $\dot{\boldsymbol{p}}_{opt}$, $\hat{\boldsymbol{w}}_{\boldsymbol{i}}, \lambda_i \; i = 1, \ldots, n_o$
Result: Optimal acceptable velocity $\dot{\boldsymbol{p}}$.
if $\left(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \dot{\boldsymbol{p}}_{opt} \rangle \geq \lambda_k \quad \forall k\right)$ **then**
  $\dot{\boldsymbol{p}} = \dot{\boldsymbol{p}}_{opt}$;
  return;
end
$max = 0$;
$\dot{\boldsymbol{p}} = 0$;
for $i = 1, \ldots, n_o$ **do**
  $\boldsymbol{proj}_1, \boldsymbol{proj}_2 = \text{projection}(\dot{\boldsymbol{p}}_{opt}, \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \lambda_i)$;
  if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{proj}_1 \rangle \geq \lambda_k \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_1 \rangle \geq max\right)$ **then**
    $\dot{\boldsymbol{p}} = \boldsymbol{proj}_1$;
    $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_1 \rangle$;
  end
  if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{proj}_2 \rangle \geq \lambda_k \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_2 \rangle \geq max\right)$ **then**
    $\dot{\boldsymbol{p}} = \boldsymbol{proj}_2$;
    $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_2 \rangle$;
  end
end
return;

what happens in two dimensions, when $d = D_w$ the cone has a null opening angle, so there is actually no constraint, whilst when $d = 0$ the opening angle is $\pi/2$ which means that the cone degenerates into a plane, so we are in the situation presented previously.

Conjecture 2. *The solution of the problem* (4.11) *in* $\mathbb{R}^3$ *is either:*

- $\dot{\boldsymbol{p}}_{opt}$*;*
- *one of the projections (normalized) of* $\dot{\boldsymbol{p}}_{opt}$ *on one of the cone associated with one of the constraints* $\langle \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \dot{\boldsymbol{p}} \rangle \geq \lambda_i$*;*
- *one of the unit vector that belongs to the intersections of the two cones to* $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ *and* $\hat{\boldsymbol{w}}_{\boldsymbol{j}}$ *for some* $i, j = 1, \ldots, n_o, \; i \neq j$*.*

Algorithm 5 shows the pseudocode for the solution of the problem in $\mathbb{R}^3$. Also for Conjecture 2 we have no formal proof, but we did not find any contradiction when comparing the result with the one computed with the optimization tool *fmincon* of MATLAB. Moreover the algorithm's complexity is still polynomial ($O(n_o^2)$), since the projection and intersection with cones can be solved efficiently exploiting some geometry observations.

Algorithm 5: Finding the optimal solution of problem (4.11) in $\mathbb{R}^3$, exploiting Conjecture 2.

Data: $\dot{\boldsymbol{p}}_{opt}$, $\hat{\boldsymbol{w}}_{\boldsymbol{i}}, \lambda_i \; i = 1, \ldots, n_o$
Result: Optimal acceptable velocity $\dot{\boldsymbol{p}}$.
if $\left(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \dot{\boldsymbol{p}}_{opt} \rangle \geq \lambda_k \quad \forall k\right)$ **then**
  $\dot{\boldsymbol{p}} = \dot{\boldsymbol{p}}_{opt}$;
  return;
end
$max = 0$;
$\dot{\boldsymbol{p}} = 0$;
for $i = 1, \ldots, n_o$ **do**
  $\boldsymbol{proj}_1, \boldsymbol{proj}_2 = \text{projection}(\dot{\boldsymbol{p}}_{opt}, \hat{\boldsymbol{w}}_{\boldsymbol{i}}, \lambda_i)$;
  if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{proj}_1 \rangle \geq \lambda_k \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_1 \rangle \geq max\right)$ **then**
    $\dot{\boldsymbol{p}} = \boldsymbol{proj}_1$;
    $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_1 \rangle$;
  end
  if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{proj}_2 \rangle \geq \lambda_k \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_2 \rangle \geq max\right)$ **then**
    $\dot{\boldsymbol{p}} = \boldsymbol{proj}_2$;
    $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{proj}_2 \rangle$;
  end
  for $j = i + 1, \ldots, n_o$ **do**
    $\boldsymbol{inters}_1, \boldsymbol{inters}_2 = \text{intersection}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}, \lambda_i, \hat{\boldsymbol{w}}_{\boldsymbol{j}}, \lambda_j)$;
    if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{inters}_1 \rangle \geq 0 \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{inters}_1 \rangle \geq max\right)$ **then**
      $\dot{\boldsymbol{p}} = \boldsymbol{inters}_1$;
      $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{inters}_1 \rangle$;
    end
    if $(\langle \hat{\boldsymbol{w}}_{\boldsymbol{k}}, \boldsymbol{inters}_2 \rangle \geq 0 \quad \forall k)$ *AND* $\left(\langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{inters}_2 \rangle \geq max\right)$ **then**
      $\dot{\boldsymbol{p}} = \boldsymbol{inters}_2$;
      $max = \langle \dot{\boldsymbol{p}}_{opt}, \boldsymbol{inters}_2 \rangle$;
    end
  end
end
return;

4.4 Magnitude control

At this point we have fixed the search direction $\dot{\boldsymbol{p}}_{dir}$ in which the agent will move and we need to set the intensity of the velocity. For doing that we will use the method of *Backtracking Line Search* (BLS), which is based on the Armijo–Goldstein condition. If we want to maximize a generic function $f(\boldsymbol{x})$, we are in position $\boldsymbol{x}(t)$ and we move in the direction $\Delta_{\boldsymbol{x}}$ (that we suppose normalized), we have to choose a step-size α

that respects the condition:

$$f(\boldsymbol{x}(t) + \alpha\,\Delta_{\boldsymbol{x}}) \geq f(\boldsymbol{x}) + \alpha\,\beta\,\nabla f(\boldsymbol{x}(t))\,\Delta_{\boldsymbol{x}}, \tag{4.12}$$

where $\nabla f(\cdot)$ is the gradient of $f(\cdot)$ and β is a design parameter ($\beta \in (0,1)$). On the left hand side $\boldsymbol{x}(t) + \alpha\,\Delta_{\boldsymbol{x}}$ represents the next position $\boldsymbol{x}(t+1)$ that will be reached moving of a quantity α in the fixed direction. On the right hand side $\alpha\,\nabla f(\boldsymbol{x}(t))\,\Delta_{\boldsymbol{x}}$ represents the increase of the value of the function that we would have if the gradient was constantly equal to $\nabla f(\boldsymbol{x}(t))$ in all the path from $\boldsymbol{x}(t)$ and $\boldsymbol{x}(t+1)$. In other terms it is the expected increase of $f(\cdot)$. Since the gradient approximation is valid only locally it is generally not true that:

$$f(\boldsymbol{x}(t) + \alpha\,\Delta_{\boldsymbol{x}}) = f(\boldsymbol{x}) + \alpha\,\nabla^{\top} f(\boldsymbol{x}(t))\,\Delta_{\boldsymbol{x}},$$

unless we choose an infinitesimal value for α. Therefore the Armijo–Goldstein condition requires that the increase of the function in the next position must be greater than a fraction β of the increase promised by the gradient approximation.
The condition is surely verified for a sufficiently small value of α, even if this would mean that the movement has been insignificant. Therefore the BLS technique follows an iterative procedure for finding a value for α that is both big enough, and also able to guarantee a sufficient increase of the function. We set an initial value α_0 and at every step we decrease the step-size by multiplying it for a value $\tau \in (0,1)$ until the condition is satisfied.

Algorithm 6 shows the pseudocode that we use. We set the initial value as:

$$\alpha_0 = \dot{p}_{lim}\Delta t,$$

where Δt is the sampling time and $\dot{p}_{lim}$ is the limit velocity, i.e. the maximum magnitude of the linear velocity that we allow. Moreover we add a condition for stopping the loop, that is if $\alpha < \alpha_{min}$, with:

$$\alpha_{min} = \dot{p}_{min}\Delta t,$$

where $\dot{p}_{min}$ is the minimum magnitude of the velocity that still makes sense to command. Notice that, since we are working with a velocity instead of a difference of position, α has the dimension of a time. Besides we adopted a simplification: we did not t compute at every step the visible set in the next position. This is justified by the fact that the contribution of a new landmark in the field of view of the camera would be negligible, because we do not admit occlusions as we already discussed in Chapter 3. An analogous algorithm is used for computing the magnitude of the angular velocity, again setting two values ω_{lim} and ω_{min} that are the maximum and minimum value for the angular velocity respectively.

Notice that applying this method we obtain a discretization in the possible magnitudes of the velocities. Choosing a small value of τ will result in a rougher discretization, which could lead to big discontinuities in the magnitude of the velocity.

On the other hand setting $\tau \approx 1$ would lead to smoother velocities, but the computation time could increase because the number of iterations necessary to find the right step-size grows if the latter is small. The maximum number of iterations indeed is $\log_\tau (\dot{p}_{min}/\dot{p}_{lim})$. Anyway we will never work with a very small sampling time, so the time necessary to perform the algorithm is not limiting, and we will use $\tau = 0.9$. For β we choose the value 0.5, that is a standard value for the BLS.

Algorithm 6: Finding the magnitude of the linear velocity with the BLS method

Data: $\dot{\boldsymbol{p}}_{dir},\ \dot{\boldsymbol{p}}_{opt},\ Q^V,\ \boldsymbol{p},\ \hat{\boldsymbol{v}},\ \dot{p}_{lim},\ \dot{p}_{min},\ \Delta t,\ \beta,\ \tau$
Result: Velocity $\dot{\boldsymbol{p}}^\star$.
$\alpha = \dot{p}_{lim}\Delta t/\tau$;
$\alpha_{min} = \dot{p}_{min}\Delta t$;
$vis_t = \text{vis}\left(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q^V\right)$;
repeat
  $\alpha = \tau\alpha$;
  $vis_{t+1} = \text{vis}\left(\boldsymbol{p} + \alpha\dot{\boldsymbol{p}}_{dir}, \hat{\boldsymbol{v}}, Q^V\right)$;
until $\left(vis_{t+1} > vis_t + \alpha\,\beta\,\dot{\boldsymbol{p}}_{opt}^\top\,\dot{\boldsymbol{p}}_{dir}\right) OR\,(\alpha < \alpha_{min})$;
if $\alpha \geq \alpha_{min}$ **then**
  $\dot{\boldsymbol{p}}^\star = \alpha\dot{\boldsymbol{p}}_{dir}/\Delta t$;
else
  $\dot{\boldsymbol{p}}^\star = \boldsymbol{0}$;
end

Moreover the algorithm inherently specifies the termination condition of the pose optimization procedure, i.e. $\|\dot{\boldsymbol{p}}^\star\| < \dot{p}_{min}$ and $\|\boldsymbol{\omega}^\star\| < \omega_{min}$.

Finally notice that in the algorithm the Armijo-Goldstein condition is slightly modified, using the strict inequality. The reason of this choice is clarified in Section 4.6.

4.5 Trading of landmarks

What we described till now is the pose optimization procedure that every single agent will perform considering its own set of landmarks. Once an agent has finished this phase, it becomes available for trading its landmarks, so it tries to communicate with other agents. We consider only trades between two agents, and therefore only one-to-one communications. Every agent has a list containing the identities of the possible trading partners, so once it reaches its final pose it picks the first name of the list and sends a request of trade. If the partner is available (meaning that it finished its pose optimization phase) then a trading procedure is started, otherwise it deletes the name from the list and sends a request to the next agent. This goes on until its list is empty, and in that case the agent will remain passive, waiting for a request coming from someone else.

The trading procedure is described in Algorithm 7. For understanding it, we must recall that our target is to increase the coverage score defined in (3.10). We call the two agents involved in the trade A and B. Their contribution to the coverage score is:

$$\begin{aligned}\mathrm{vis}(\boldsymbol{p}_A, \hat{\boldsymbol{v}}_A, Q_A) + \mathrm{vis}(\boldsymbol{p}_B, \hat{\boldsymbol{v}}_B, Q_B) = &\sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q_A^V} \bar{f}(\|\boldsymbol{q}-\boldsymbol{p}_A\|)\,\langle \boldsymbol{q}-\boldsymbol{p}_A, \hat{\boldsymbol{v}}_A\rangle\,\langle \boldsymbol{p}_A-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle \\ &+ \sum_{(\boldsymbol{q},\hat{\boldsymbol{u}})\in Q_B^V} \bar{f}(\|\boldsymbol{q}-\boldsymbol{p}_B\|)\,\langle \boldsymbol{q}-\boldsymbol{p}_B, \hat{\boldsymbol{v}}_B\rangle\,\langle \boldsymbol{p}_B-\boldsymbol{q}, \hat{\boldsymbol{u}}\rangle\,.\end{aligned}$$

If the value of this sum increases, then also the total coverage score will increase, and this is exactly what the procedure achieves. Indeed the results of the trade are two sets $Q_A^\star$ and $Q_B^\star$ such that:

$$\begin{cases} Q_A^\star \cup Q_B^\star = Q_A \cup Q_B \\ \mathrm{vis}(\boldsymbol{p}_A, \hat{\boldsymbol{v}}_A, \boldsymbol{q}, \hat{\boldsymbol{u}}) \geq \mathrm{vis}(\boldsymbol{p}_B, \hat{\boldsymbol{v}}_B, \boldsymbol{q}, \hat{\boldsymbol{u}}) \quad \forall (\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q_A^\star \\ \mathrm{vis}(\boldsymbol{p}_B, \hat{\boldsymbol{v}}_B, \boldsymbol{q}, \hat{\boldsymbol{u}}) \geq \mathrm{vis}(\boldsymbol{p}_A, \hat{\boldsymbol{v}}_A, \boldsymbol{q}, \hat{\boldsymbol{u}}) \quad \forall (\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q_B^\star. \end{cases} \tag{4.13}$$

These conditions ensure that the value of the sum of the visions did not decrease. Algorithm 7 scrolls through both the lists and transfers to $Q_B^\star$ all the landmarks of Q_A that are seen better from B than from A and vice-versa. The trade is considered successful if at least one landmark was transferred. In this case the agents will re-initialize their partner lists and they will start a new pose optimization phase. Notice that it is possible that after a successful trade the contribution of one of the two agents to the coverage score decreased, but anyway the sum of the two vision values increased, and so did the total coverage score. If the trade is not successful then both the agents involved will delete the other from their lists of possible partners. It is important to highlight that when two agents have the same vision of one landmark they do not exchange it. Trading it in some cases could bring a benefit to the coverage score (after the pose optimization), but on the other hand it could also lead to infinite loops so we decide to avoid it.

Finally, when there is no possible trade (meaning that all the partner lists are empty) we consider the coverage algorithm concluded.

4.6 Convergence analysis

The convergence of the algorithm is ensured by the following propositions.

Proposition 3. *After every iteration of the algorithm (for any agent):*

$$\mathit{vis}(\boldsymbol{p}(t+\Delta t), \hat{\boldsymbol{v}}(t+\Delta t), Q) \geq \mathit{vis}(\boldsymbol{p}(t), \hat{\boldsymbol{v}}(t), Q), \tag{4.14}$$

and the equality holds if and only if $\dot{\boldsymbol{p}}(t) = \boldsymbol{0}$, $\boldsymbol{\omega}(t) = \boldsymbol{0}$.

Algorithm 7: Trading of landmarks between two agents.

Data: $\boldsymbol{p}_A$, $\hat{\boldsymbol{v}}_A$, Q_A, $\boldsymbol{p}_B$, $\hat{\boldsymbol{v}}_B$, Q_B
Result: Landmark sets $Q_A^\star$, $Q_B^\star$ that respect conditions (4.13).
$Q_A^\star = \emptyset$;
$Q_B^\star = \emptyset$;
foreach $(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q_A$ **do**
  if $vis(\boldsymbol{p}_B, \hat{\boldsymbol{v}}_B, \boldsymbol{q}, \hat{\boldsymbol{u}}) > vis(\boldsymbol{p}_A, \hat{\boldsymbol{v}}_A, \boldsymbol{q}, \hat{\boldsymbol{u}})$ **then**
    Add $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ to $Q_B^\star$;
  else
    Add $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ to $Q_A^\star$;
  end
end
foreach $(\boldsymbol{q}, \hat{\boldsymbol{u}}) \in Q_B$ **do**
  if $vis(\boldsymbol{p}_A, \hat{\boldsymbol{v}}_A, \boldsymbol{q}, \hat{\boldsymbol{u}}) > vis(\boldsymbol{p}_B, \hat{\boldsymbol{v}}_B, \boldsymbol{q}, \hat{\boldsymbol{u}})$ **then**
    Add $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ to $Q_A^\star$;
  else
    Add $(\boldsymbol{q}, \hat{\boldsymbol{u}})$ to $Q_B^\star$;
  end
end

Proof. Assume $\dot{\boldsymbol{p}}(t) \neq \mathbf{0}$. Then, from the magnitude control:

$$\mathrm{vis}(\boldsymbol{p}(t+\Delta t), \hat{\boldsymbol{v}}(t+\Delta t), Q) > \mathrm{vis}(\boldsymbol{p}(t), \hat{\boldsymbol{v}}(t), Q) + \alpha\, \beta\, \dot{\boldsymbol{p}}_{opt}(t)^\top\, \dot{\boldsymbol{p}}_{dir}(t).$$

where:

$$\alpha\, \beta\, \dot{\boldsymbol{p}}_{opt}(t)^\top\, \dot{\boldsymbol{p}}_{dir}(t) \geq 0.$$

Indeed $\alpha \geq \alpha_{min} > 0$ because $\dot{\boldsymbol{p}}(t) \neq \mathbf{0}$, $\beta \in (0,1)$ by hypothesis. Moreover $\dot{\boldsymbol{p}}_{opt}(t)^\top\, \dot{\boldsymbol{p}}_{dir}(t) \geq 0$ because $\dot{\boldsymbol{p}}_{dir}(t)$ must be a non decreasing linear velocity (by the constraints of the collision avoidance optimization problems).

Analogous proof could be given if $\boldsymbol{\omega}(t) \neq \mathbf{0}$. □

Proposition 3 states that during the pose optimization phase, the vision function is monotonically increasing for the single agents. Notice that in the proof we exploited the choice of considering the Armijo-Goldstein condition with the strict inequality. Otherwise, there would be cases in which the proposition does not hold: if $\dot{\boldsymbol{p}}_{opt}(t)^\top\, \dot{\boldsymbol{p}}_{dir}(t) = 0$, it would be possible to have:

$$\mathrm{vis}(\boldsymbol{p}(t+\Delta t), \hat{\boldsymbol{v}}(t+\Delta t), Q) = \mathrm{vis}(\boldsymbol{p}(t), \hat{\boldsymbol{v}}(t), Q)$$

for every value of α. Therefore the optimization phase could continue indefinitely, because there could be velocities not equal to zero that guarantee the maintenance of the value of the vision.

Proposition 4. *After every iteration of the algorithm (for any agent):*

$$cov(\boldsymbol{P}(t + \Delta t), \boldsymbol{Q}(t + \Delta t)) > cov(\boldsymbol{P}(t), \boldsymbol{Q}(t)). \tag{4.15}$$

Proof. During the pose optimization phase, for every agent the vision (i.e. its contribution to the coverage score) increases, as expressed by Proposition 3. During the trading phase, the contribution of the two agents involved increases, as explained in section 4.5. □

Proposition 4 ensures the convergence of the algorithm. Indeed the coverage score is monotonically increasing, but has also an upper bound given by the total number of landmarks.

Chapter 5

Simulations

For simulating our coverage algorithm, we use the scheme in Figure 5.1, implemented in MATLAB. To simulate the movement of the agent we simply integrate the velocities, meaning that in the integrator block we use the following the equations:

$$\begin{cases} \boldsymbol{p}(t+\Delta t) = \boldsymbol{p}(t) + \dot{\boldsymbol{p}}^{\star}(t)\Delta t \\ \hat{\boldsymbol{v}}(t+\Delta t) = \hat{\boldsymbol{v}}(t) + \left(\boldsymbol{\omega}^{\star}(t) \times \hat{\boldsymbol{v}}(t)\right)\Delta t \end{cases} \tag{5.1}$$

where:

$$\boldsymbol{\omega}^{\star}(t) \times \hat{\boldsymbol{v}}(t) = \frac{\partial \hat{\boldsymbol{v}}}{\partial t}(t)$$

It is important to observe that in general the second equation in (5.1) does not return a unitary vector. Therefore we need to apply also a normalization:

$$\begin{aligned} \boldsymbol{v} &= \hat{\boldsymbol{v}}(t) + \left(\boldsymbol{\omega}^{\star}(t) \times \hat{\boldsymbol{v}}(t)\right)\Delta t \\ \hat{\boldsymbol{v}}(t+\Delta t) &= \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} \end{aligned}$$

In all the simulations presented, when the unit of measurement is left implicit, the times are measured in s, the positions in m, the velocities in m/s, the angles in rad and the angular velocities in rad/s.

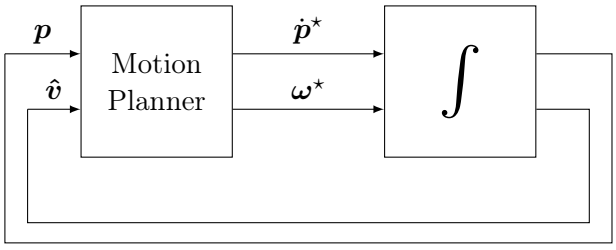


Figure 5.1: Block diagram for the simulation.

5.1 Unconstrained optimization in two dimensions

We consider the following set of landmarks:

$$Q = \{(\boldsymbol{q}_1, \hat{\boldsymbol{u}}_1), (\boldsymbol{q}_2, \hat{\boldsymbol{u}}_2), (\boldsymbol{q}_3, \hat{\boldsymbol{u}}_3)\} \tag{5.2}$$
$$= \left\{ \left(\begin{bmatrix} 1 \\ 4.5 \end{bmatrix}, \begin{bmatrix} \cos(-2\pi/3) \\ \sin(-2\pi/3) \end{bmatrix} \right), \left(\begin{bmatrix} 2 \\ 4.1 \end{bmatrix}, \begin{bmatrix} \cos(-\pi/2) \\ \sin(-\pi/2) \end{bmatrix} \right), \left(\begin{bmatrix} 2.5 \\ 4.3 \end{bmatrix}, \begin{bmatrix} \cos(-\pi/4) \\ \sin(-\pi/4) \end{bmatrix} \right) \right\},$$

and we set the initial pose to:

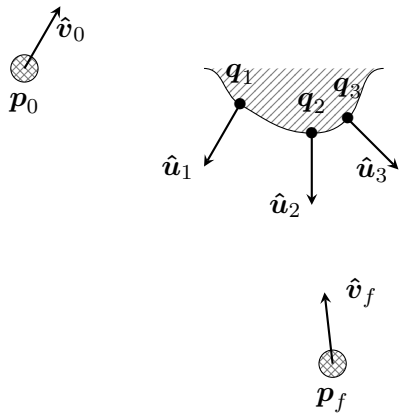
$$\boldsymbol{p}_0 = \begin{bmatrix} -1 \\ 5 \end{bmatrix}, \quad \hat{\boldsymbol{v}}_0 = \begin{bmatrix} \cos(\psi_0) \\ \sin(\psi_0) \end{bmatrix}, \quad \psi_0 = \pi/3,$$

as in Figure 5.2a. For this simulation we set the optimal distance from the landmarks to $d_{opt} = 3$ and we do not consider any boundary in the mission space. Figure 5.2b shows the optimal orientation vector computed in different positions, obtained through the algorithm explained in section 3.4. This allows us to compute the value of the vision in those points and build a sort of map which tells us where is the maximum that we are searching. Figure 5.2c shows the contour lines of the vision function and the gradient with respect to the position, i.e. $\nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))$. It can be noticed that the vision function is equal to zero for any orientation behind the landmarks, and that there is only one maximum.

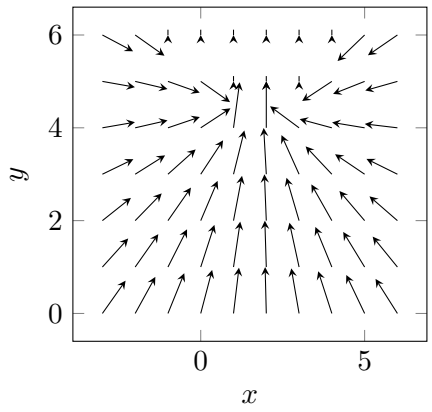
In this simulation, since we do not consider any spatial constraint, we do not need the collision avoidance block and we can always choose the optimal direction for the movement, which means that $\dot{\boldsymbol{p}}_{dir} = \dot{\boldsymbol{p}}_{opt}$. For the minimum and maximum velocities we set the values:

$$\dot{p}_{lim} = 0.3\tfrac{m}{s}, \qquad \dot{p}_{min} = 0.01\tfrac{m}{s},$$
$$\omega_{lim} = \frac{\pi}{4}\tfrac{rad}{s}, \qquad \omega_{min} = \frac{\pi}{180}\tfrac{rad}{s}.$$

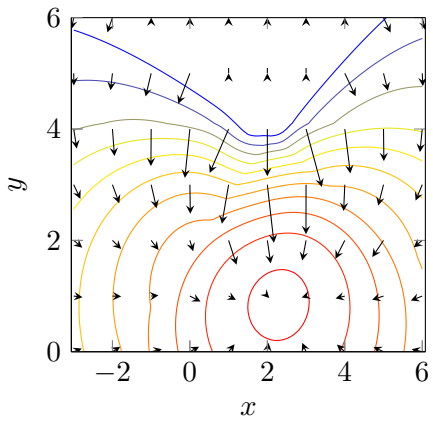
For the backtracking line search we used $\tau = 0.9$ and $\beta = 0.5$.
In Figure 5.2d is shown the trajectory followed by the camera, and in Figure 5.3 are reported the time evolution of the main variables. It can be noticed that, as we expected, the camera reaches the maximum point in the vision map, and that the value of the vision is continuous and monotonically increasing along the trajectory. Instead in the graph of the linear velocity there are two discontinuities at $t = 3.2\,s$ and $t = 14\,s$. These can be explained with the change of the visible set of landmarks: at the beginning only $(\boldsymbol{q}_1, \hat{\boldsymbol{u}}_1)$ is visible, then $(\boldsymbol{q}_2, \hat{\boldsymbol{u}}_2)$ becomes visible and finally also $(\boldsymbol{q}_3, \hat{\boldsymbol{u}}_3)$. Indeed, as we noticed when we computed the gradients with respect to position and orientation, both of them are continuous unless there is a change of the visible set. The magnitude of the linear velocity is constantly equal to the maximum value allowed $\dot{p}_{lim}$, until the final part of the simulation, when it decreases gradually because the camera is close to the maximum point. This happens because the value that we chose for $\dot{p}_{lim}$ is relatively small so it always verifies the Armijo condition for all the initial part of the movement. On the other hand, observing the



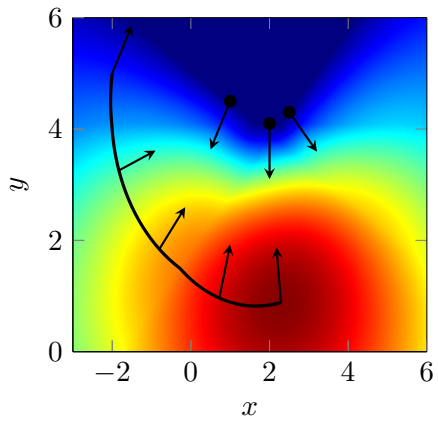
(a) Landmarks that we want to monitor, initial and final pose of the camera.



(b) Optimal orientation vectors computed in all the map.



(c) Contour lines of the vision function (computed in every point with the optimal orientation) and direction of the gradient with respect to the position.



(d) Trajectory followed for the camera placement.

Figure 5.2: Optimization of the pose of one camera in unbounded mission space.

evolution of the angular velocity we can notice that only at the beginning it reaches ω_{lim}, while in the whole central part of the simulation ω is almost constant, until the end when it reaches zero. This can be explained looking at Figure 5.4, where are represented the values of ψ and ψ_{opt} (dashed in the graph), which was computed for every position. When the simulation starts there is a big difference between the orientation of the camera and the optimal one, and this results in a big value of ω; after approximately 1.5 s this gap decreases and remains almost constant, due to the fact that the camera is still moving: at every step ω would bring $\hat{\boldsymbol{v}}$ to equal $\hat{\boldsymbol{v}}_{opt}$, but since the position keeps changing, also the optimal orientation changes; finally,

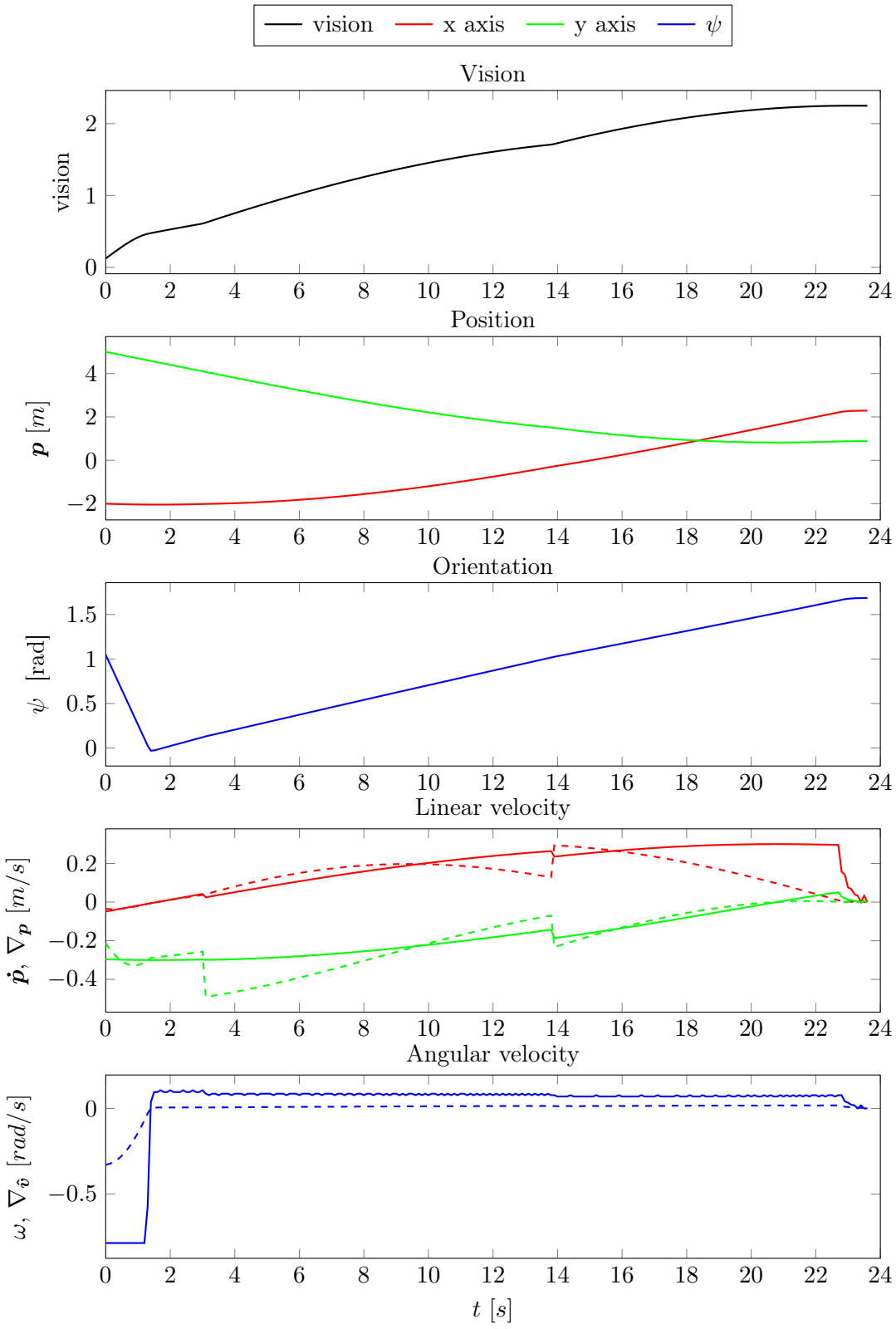


Figure 5.3: Data relative to the simulation of unconstrained optimization.

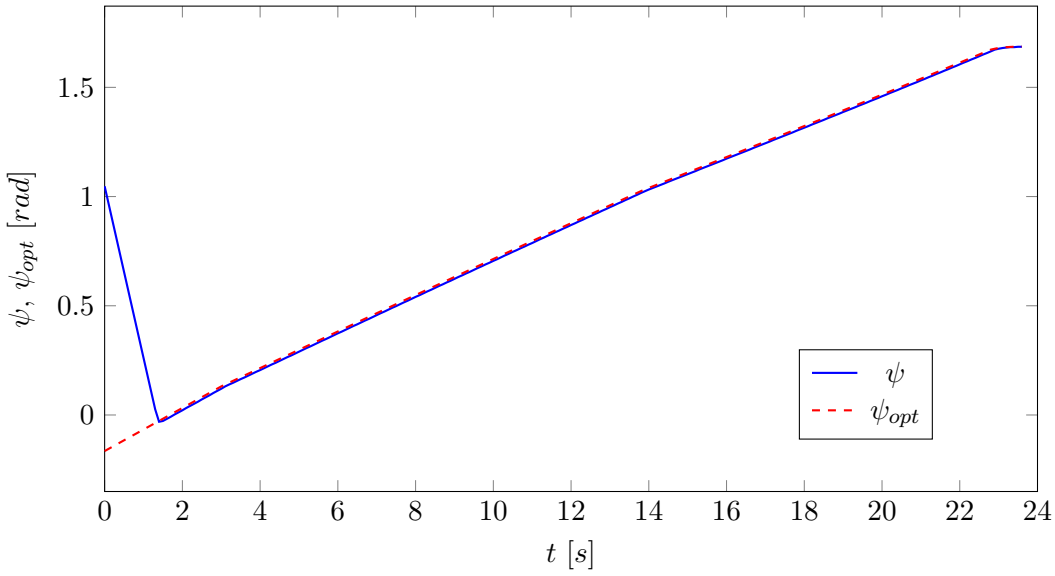


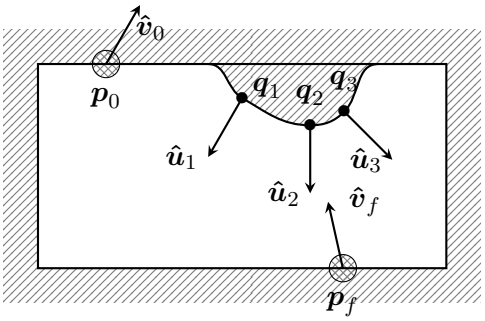
Figure 5.4: Camera orientation during the simulation and optimal orientation in the corresponding position.

when the camera reaches the final position and stops, the gap in the orientation is bridged and at the end $\psi = \psi_{opt}$. Notice that the value of ω in the central part keeps oscillating simply because the exact value that would be necessary to compensate the movement of the camera is not one of those obtainable with the Backtracking Line Search.

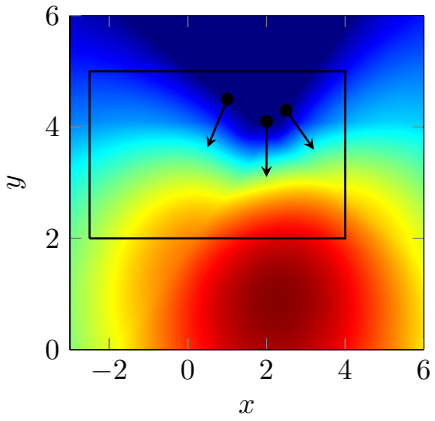
An important observation is necessary regarding the magnitude control. As can be seen from the velocity plots in the simulation, the magnitude of $\nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))$ and $\nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))$ decreases to zero when we approach the final point (see the dashed lines in the velocity plots in Figure 5.3). So in principle we could simply choose to use $\dot{\boldsymbol{p}} = \nabla_{\boldsymbol{p}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))$ and $\omega = \nabla_{\hat{\boldsymbol{v}}}(\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, Q))$, without using the backtracking line search, but simply saturating the velocities to $\dot{p}_{lim}$ and ω_{lim} and stopping when they are smaller than the minimum values. Anyway this would lead to a very slow convergence (nearly twice the time in the case of the previous simulation), because the velocities are smaller then the ones that we obtain with the BLS, since the gradient decreases when we approach the maximum.

5.2 Collision avoidance in two dimensions

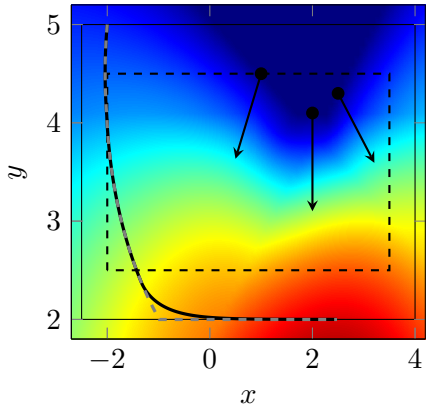
We will now study what happens in the same setup (same landmark set and initial position), but when the mission space is bounded or when there are multiple agents involved.



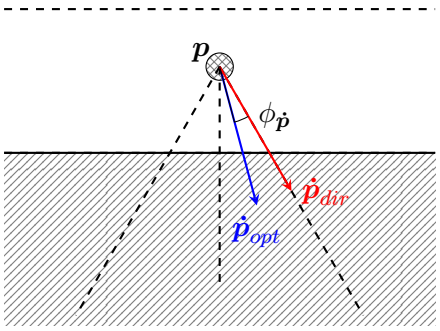
(a) Landmarks that we want to monitor and bounds of the mission space, initial and final pose of the camera.



(b) Vision map and walls. Notice that the maximum is outside of the mission space.



(c) Trajectory followed in the camera placement. The dashed grey trajectory is relative to the abrupt version of the collision avoidance while the black one is relative to the smooth version.



(d) Collision avoidance and explanation of the $\phi_{\dot{p}}$ angle.

Figure 5.5: Optimization of the camera pose in a rectangular mission space.

Bounded mission space

At first, we consider the rectangular environment represented in Figure 5.5a. As can be seen in Figure 5.5b, the maximum is outside of the mission space so it cannot be reached.

We perform the simulation with both the collision avoidance techniques proposed in 4.3, the abrupt one and the smooth one. The trajectories obtained are in Figure 5.5c, and they both reach the same final point, which is the local maximum inside the mission space. The dashed grey trajectory is the one relative to the abrupt version of the collision avoidance. It simply follows the same path of simulation in

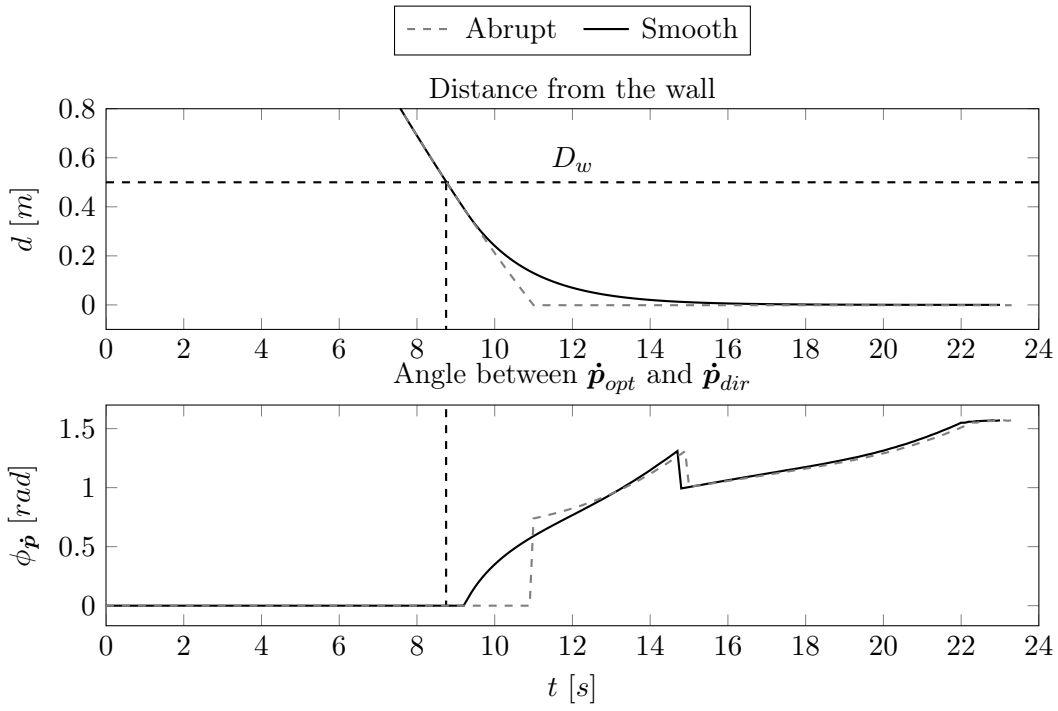
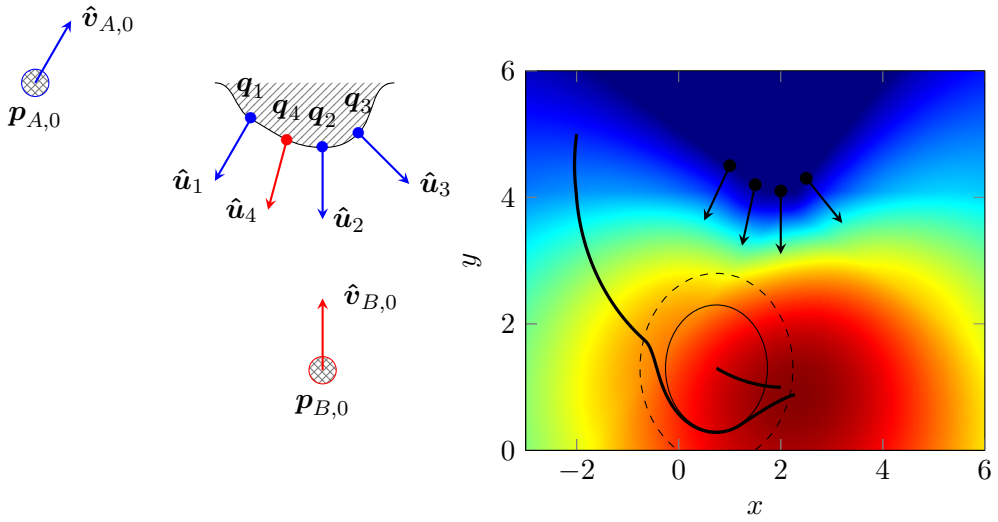


Figure 5.6: Evolution of $\phi_{\dot{\boldsymbol{p}}}$, whose meaning is explained in Figure 5.5d.

free space until it reaches the wall, and then it moves parallel to the wall till $\dot{\boldsymbol{p}}_{opt}$ is perpendicular to the wall. Instead with the smooth version (black trajectory) the camera follows the same path until it is at distance D_w from the wall, then it starts worrying about the collision. As it gets closer to the wall, the angle of unsafe velocities becomes wider (see Figure 5.5d). If $\dot{\boldsymbol{p}}_{opt}$ is safe then its direction is followed, otherwise it is chose its projection on the closest ray of the angle. Therefore, as the camera approaches the wall, the velocity smoothly turns until it becomes parallel. We denote with $\phi_{\dot{\boldsymbol{p}}}$ the angle between the optimal direction $\dot{\boldsymbol{p}}_{opt}$ and the one that is followed $\dot{\boldsymbol{p}}_{dir}$. In Figure 5.6 are compared the values of $\phi_{\dot{\boldsymbol{p}}}$ in the two cases. In the abrupt case, it is zero until the wall is reached, then there is a sudden change because the $\dot{\boldsymbol{p}}_{dir}$ becomes parallel to the wall. Instead in the smooth case the angle changes gradually while the distance with the wall decreases. Then the angles follow approximately the same evolution (there is a small delay in the abrupt case because the path followed is longer). The sharp change of angle at $t \approx 15\,s$ is due to a new visible landmark, therefore what changes is not $\dot{\boldsymbol{p}}_{dir}$ but $\dot{\boldsymbol{p}}_{opt}$. Notice that there is a short time ($t \in [8.8, 9]$) in which even if $d < D_w$ the camera keeps moving in the direction of $\dot{\boldsymbol{p}}_{opt}$ also in the smooth case. This happens because the angle of unsafe velocities is small and $\dot{\boldsymbol{p}}_{opt}$ is still considered a safe velocity, because it is not perpendicular to the wall.

This simulation clarifies why the smooth version of the collision avoidance is prefer-



(a) Initial poses of the two agents and respective landmarks sets. The mission space is unbounded.

(b) Trajectories followed by the two agents. The inner circle has radius R_s, the outer one as radius $R_s + D_w$.

Figure 5.7: Optimization of the pose of two cameras (the trade is not considered) for testing the collision avoidance between agents.

able: in a real situation the velocity cannot change instantaneously, so it is reasonable to expect that the trajectory followed in the abrupt case would lead to a collision.

Avoidance between agents

Similar conclusions can be drawn in the case presented in Figure 5.7a, where there are two cameras, called A and B. The difference with the previous simulations is that we add a fourth landmark that is assigned to B. The initial conditions are chosen so that agent B will interfere with the movement of agent A. The initial poses are:

$$\boldsymbol{p}_{A,0} = \begin{bmatrix} -1 \\ 5 \end{bmatrix}, \quad \hat{\boldsymbol{v}}_{A,0} = \begin{bmatrix} \cos(\psi_{A,0}) \\ \sin(\psi_{A,0}) \end{bmatrix}, \quad \psi_{A,0} = \pi/3,$$
$$\boldsymbol{p}_{B,0} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \hat{\boldsymbol{v}}_{B,0} = \begin{bmatrix} \cos(\psi_{B,0}) \\ \sin(\psi_{B,0}) \end{bmatrix}, \quad \psi_{B,0} = \pi/2,$$

and the landmarks sets are:

$$Q_A = \{(\boldsymbol{q}_1, \hat{\boldsymbol{u}}_1), (\boldsymbol{q}_2, \hat{\boldsymbol{u}}_2), (\boldsymbol{q}_3, \hat{\boldsymbol{u}}_3)\},$$
$$Q_B = \{(\boldsymbol{q}_4, \hat{\boldsymbol{u}}_4)\},$$

where the first three landmarks are the same as in the previous simulations (see Equation (5.2)), while the fourth is:

$$(\boldsymbol{q}_4, \hat{\boldsymbol{u}}_4) = \left(\begin{bmatrix} 1.5 \\ 4.2 \end{bmatrix}, \begin{bmatrix} \cos(-7\pi/12) \\ \sin(-7\pi/12) \end{bmatrix} \right).$$

The vision map represented in Figure 5.7b is relative to agent A, and it is the same as before. Moreover are shown the trajectories followed by the two cameras. Agent B arrives to its final position in less than five seconds and then it stops. Therefore A has to change its path and circumnavigate B, but it finally reaches its maximum. The inner circle represented in figure represents the minimum distance that can be kept safely between two agents (in this case $1\,m$), the outer circle shows the area where A needs to worry about the collision ($D_w = 0.5\,m$). Notice that A if possible keeps the minimum distance from B so in the choice of radius R_s we need to keep a safety margin.

Velocity oscillation due to collision avoidance

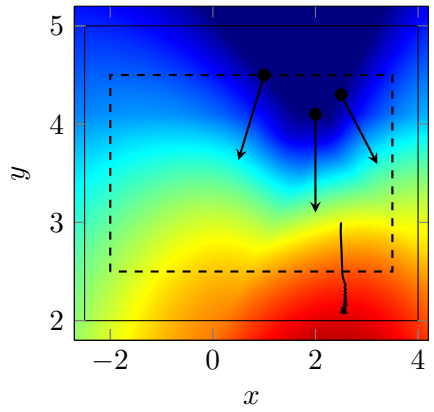
The collision avoidance algorithm that we proposed in particular situations can lead to undesired oscillations in the velocity. Consider for instance the case proposed in Figure 5.8a: the setup is equivalent to the one described previously in the case of bounded mission space, but with a different initial position. In this example:

$$\boldsymbol{p}_0 = \begin{bmatrix} 2.5 \\ 5 \end{bmatrix}, \qquad \hat{\boldsymbol{v}}_0 = \begin{bmatrix} \cos(\psi_0) \\ \sin(\psi_0) \end{bmatrix}, \qquad \psi_0 = \pi/2.$$

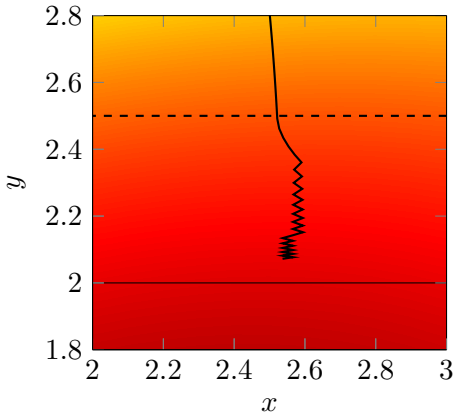
As shown in the Figure 5.8b the trajectory is not smooth but presents oscillations in the direction parallel to the wall. Figure 5.8c explains the cause of this behaviour: $\dot{\boldsymbol{p}}_{opt}$ is almost perpendicular to the wall and a slight modification of the position can make it pass from one side to the other of the direction orthogonal to the wall. But since we always choose the projection of $\dot{\boldsymbol{p}}_{opt}$ on the closest ray of the angle, this small change of $\dot{\boldsymbol{p}}_{opt}$ turns into a big change of $\dot{\boldsymbol{p}}_{dir}$. Moreover the oscillation becomes even bigger when we approach the wall, because the angle becomes wider. Figure 5.8d shows $\dot{\boldsymbol{p}}$ how the velocity changes during the simulation: the y component decreases during when the distance with the wall becomes smaller, but the x component is alternatively positive and negative.

Clearly such a behaviour can cause instability in a real system, and in any case this input cannot be followed by a system with inertia (notice that we are simulating using a simple integrator, so any velocity can be followed), so we need to solve this issue. The problem derives from the fact that we decide to choose always the best direction for the velocity but in this case there is not a big difference between the two possible choices, because $\dot{\boldsymbol{p}}_{opt}$ is almost vertical. Therefore what we will do is adding another constraint to the optimization problem of the collision avoidance:

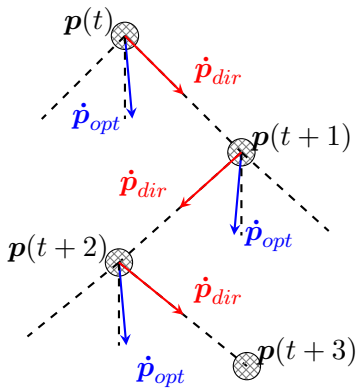
$$\langle \dot{\boldsymbol{p}}_{dir}(t), \dot{\boldsymbol{p}}_{dir}(t-1) \rangle > \lambda_{dir},$$



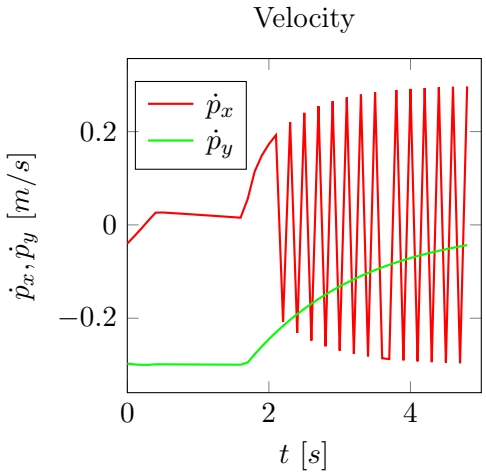
(a) Vision map and walls. The camera trajectory shows oscillations.



(b) Zoom of the trajectory: oscillations in the direction parallel to the wall.



(c) Explanation of the oscillating behaviour.



(d) Evolution of the velocity that show that the oscillation is only relative to the x component, i.e. the one parallel to the obstacle.

Figure 5.8: Simulation showing an issue of the collision avoidance method, which happens when the optimal direction is almost orthogonal to an obstacle.

where $\lambda_{dir} \in [0, 1]$ is close to one. The meaning of the constraint is that, in addition to being a safe velocity, $\dot{\boldsymbol{p}}_{dir}(t)$ must also be close enough to the direction of the last iteration. The result of adding this constraint is shown in Figure 5.9: there are lateral movements, but they are wider then before because the camera moves in one of the two directions until the velocity is put to zero for the magnitude control. We used $\lambda_{dir} = 0.95$, which means that the angle between the past and the new direction must be greater than 20°, but with any value between 0.3 and 0.99 we obtain the

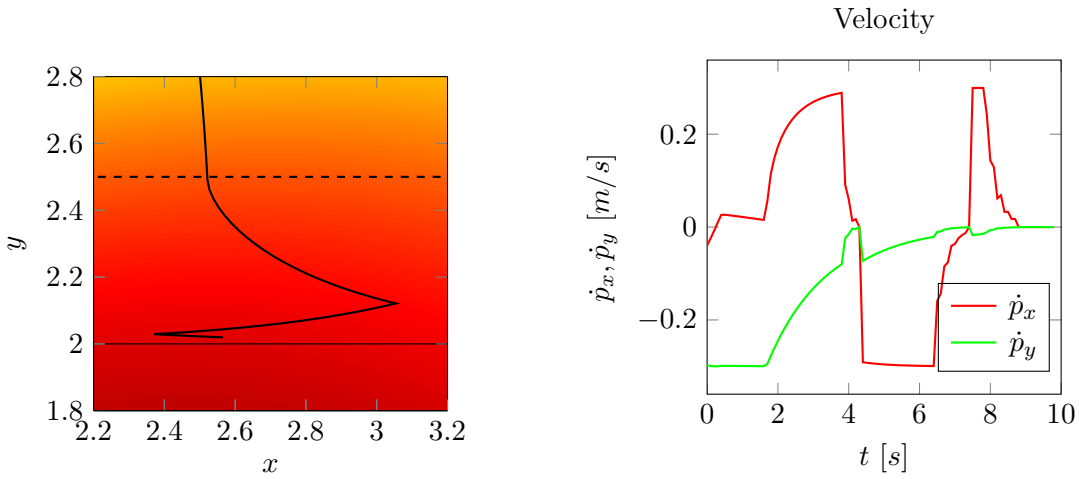


Figure 5.9: Simulation with the limit in the change of direction for preventing oscillations.

same result, because its role is just excluding a direction that is very different from the past one. However, for obtaining this result we had to set a very small value for ω_{min}, otherwise once finished the first sweep (at $t \approx 4\,s$) the camera would stop because both the linear and angular velocity would be lower than the minimum values. Hence, in this case the camera will not stop in the optimal position, but close to it.

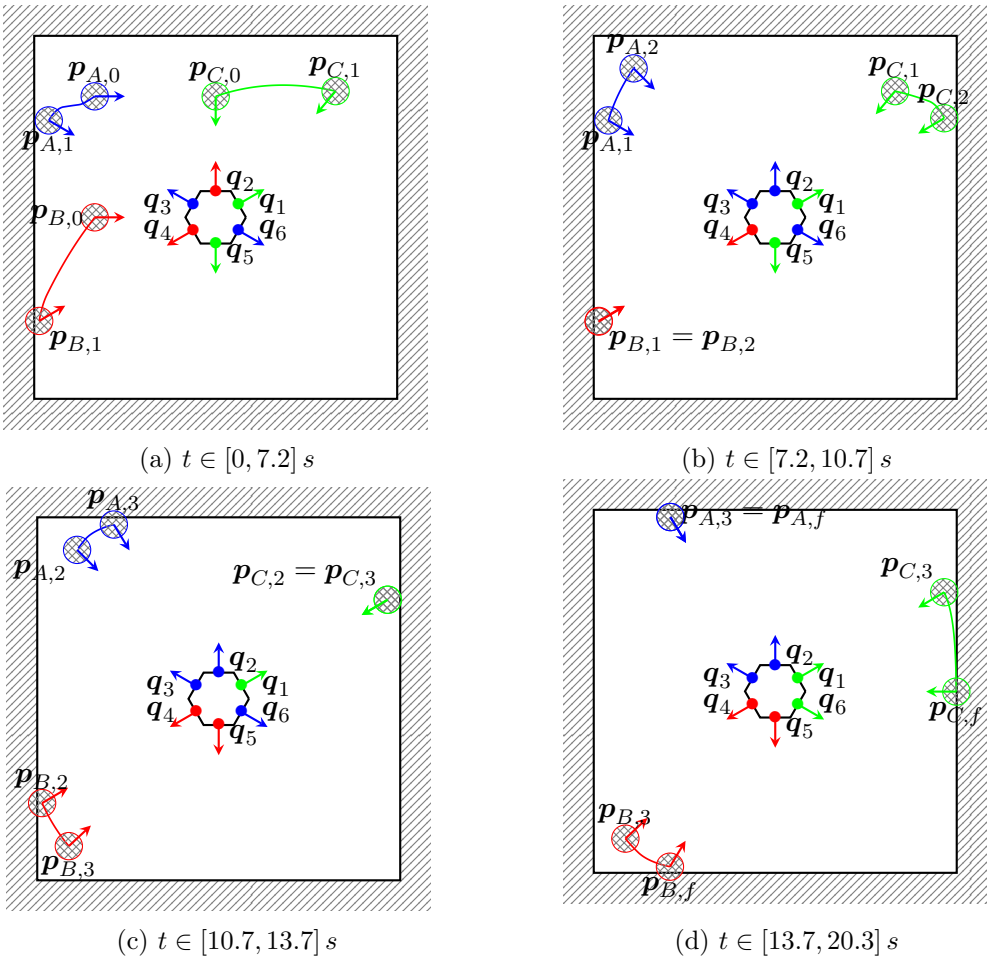
5.3 Multiple agents in two dimensions

We now present a complete simulation of our algorithm. We consider an hexagonal structure of side $L = 0.5m$ with one landmark at the centre of each edge, as in Figure 5.10:

$$Q = \left\{ (\boldsymbol{q}_k, \hat{\boldsymbol{u}}_k) = \left(L\frac{\sqrt{3}}{2} \begin{bmatrix} \cos(\phi_k) \\ \sin(\phi_k) \end{bmatrix}, \begin{bmatrix} \cos(\phi_k) \\ \sin(\phi_k) \end{bmatrix} \right),\ k = 1, \ldots, 6 \right\},$$
$$\phi_k = \frac{\pi\, k}{3} - \frac{\pi}{6}.$$

The mission space is $[-3, 3] \times [-3, 3]$. We run the simulation with three agents A,B and C, initialized with conditions:

$$\begin{aligned}
&\boldsymbol{p}_{A,0} = \begin{bmatrix} -2 & 2 \end{bmatrix}^\top, && \psi_{A,0} = 0, && Q_{A,0} = \{(\boldsymbol{q}_k, \hat{\boldsymbol{u}}_k),\ k = 3, 6\}, \\
&\boldsymbol{p}_{B,0} = \begin{bmatrix} -2 & 0 \end{bmatrix}^\top, && \psi_{B,0} = 0, && Q_{B,0} = \{(\boldsymbol{q}_k, \hat{\boldsymbol{u}}_k),\ k = 2, 4\}, \\
&\boldsymbol{p}_{C,0} = \begin{bmatrix} 0 & 2 \end{bmatrix}^\top, && \psi_{C,0} = -\tfrac{\pi}{2}, && Q_{C,0} = \{(\boldsymbol{q}_k, \hat{\boldsymbol{u}}_k),\ k = 1, 5\},
\end{aligned}$$



(a) $t \in [0, 7.2]\, s$

(b) $t \in [7.2, 10.7]\, s$

(c) $t \in [10.7, 13.7]\, s$

(d) $t \in [13.7, 20.3]\, s$

Figure 5.10: Simulation of monitoring of an hexagonal structure with multiple agents.

as in Figure 5.10a. This is how the simulation evolves:

1. Figure 5.10a: each agent can see only one of its landmarks so they start optimizing the pose to have the best vision of that landmark. At $t = 3.7\, s$ A reaches the optimal pose $\boldsymbol{p}_{A,1}$ and it stops. It tries to communicate with B and C for trading the landmarks but they are still busy, so it waits for someone else to ask him to trade. At $t = 7.2\, s$ B finishes the optimization in $\boldsymbol{p}_{B,1}$, so it communicates with A and the trade is successful: the landmark 2 can be seen from A but not from B, so it becomes responsibility of A. In the mean time C reaches position $\boldsymbol{p}_{C,1}$ but it is still optimizing the pose.

2. Figure 5.10b: A starts the optimization, while B waits because it is already at its optimum and the other two agents are busy. At $t = 10.7\, s$ C reaches the maximum, so it can trade with B and the landmark 5 passes from Q_C to Q_B.

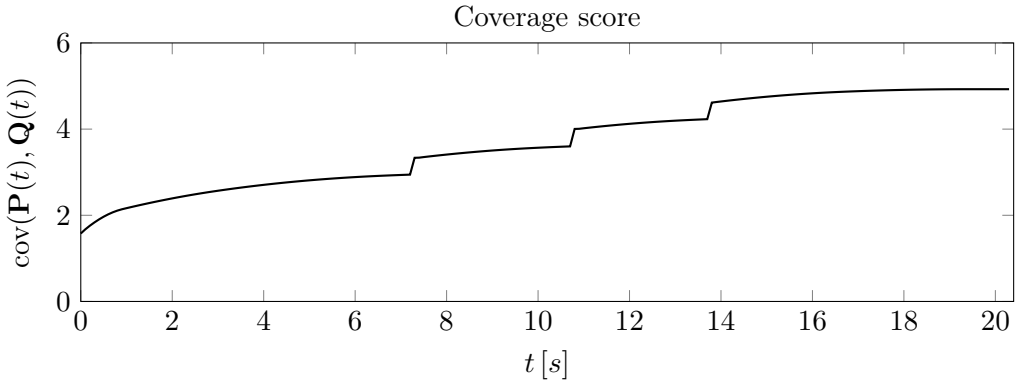


Figure 5.11: Coverage score over time during the multiagent simulation.

3. Figure 5.10c: B starts optimizing, while C waits until A reaches the maximum at $t = 13.7\,s$. Then A communicates with C and landmark 6 is traded.

4. Figure 5.10d: A waits, while B and C optimize. First B and then also C reach their maximum but no trade is successful, so the final configuration has been reached.

During the whole process, the coverage score is always increasing, as can be seen in Figure 5.11, and there are three jumps corresponding to the three successful trades.

Chapter 6

Experimental results

In this chapter, we present the experimental set-up that we used and the results that were obtained. The algorithm proposed in chapter 4 was implemented in ROS (Robot Operative System [29]) environment using the PYTHON language [30], and then used to command both simulated and real quadcopters (IRIS+ of 3D ROBOTICS [31]).
In Section 6.1 is described the control structure that was used to command the quadcopters. In Section 6.2 are shown some of the experiments that were performed. During the chapter we deal with two different coordinate frames (the world frame and the body frame), and we use the Euler angles for expressing the orientation of the quadcopter. An introduction to these concepts is given in 2.2. Moreover we use two alternative ways for expressing the orientation of the camera: one is through the vector $\hat{\boldsymbol{v}}$ used till now, and the other one is through two angles in a latitude-longitude fashion. Further details can be found in 2.3.

6.1 Control set-up in ROS

As was explained in Chapter 4 the algorithm works as a state machine with two states, that we called *pose optimization* and *trade*. While the algorithm is running every agent switches multiple times between one state and the other, but it is easier to treat them independently, since the control loop that is needed in the two cases is different.

Control loop for pose optimization

The control loop can be divided in several simpler blocks, as represented in the diagram in Figure 6.1. We adopted a modular approach: each block was coded as a ROS node or a PYTHON function. This means that all the different components are implemented independently and, in future modifications, can be easily substituted.

The motion planner was already presented in chapter 4 and is used to command

the linear and angular velocities that the agent has to follow[1]. These velocities are then processed by low level controllers for obtaining the commands that are required as input by the quadcopter. The speed controller that we used is basically a PID controller which returns the desired 3d-force $\boldsymbol{F}^\star = [F_x \ F_y \ F_z]$.

As mentioned, with the real quadcopter we are able to control just one of the degrees of freedom of the camera orientation. This is why we adopted the latitude-longitude configuration for the orientation: in this way the longitude coincides with the yaw of the quadcopter (so it can be physically controlled), while the latitude would correspond with the tilt movement of a camera attached to the quad. In the experiments this last degree of freedom is just simulated, meaning that the latitude value θ is obtained by integration of the latitude rate $\dot{\theta}$. The conversion from angular velocity in the world frame and latitude and longitude rates is done with the Equations (2.4), (2.5):

$$\dot{\psi} = \omega_3, \dot{\theta} = \sin(\psi)\,\omega_1 - cos(\psi)\,\omega_2.$$

Moreover, since one of the inputs of the quadcopter is the yaw rate, which coincides with the longitude rate ($\dot{\gamma} = \dot{\psi}$), we do not need a yaw rate controller.
Finally $\boldsymbol{F}^\star$ and $\dot{\psi}$ are converted to the final inputs of the quadcopter: the desired thrust $T^\star$, roll angle $\alpha^\star$, pitch angle $\beta^\star$ and yaw rate $\dot{\gamma}^\star$. Then there is an on-board controller which transforms these inputs in the angular velocities of the four propellers.
In the real system, the feedback is given by a motion capture system (QUALISYS [32]): there is a network of 12 infrared cameras which are able to identify some markers attached to the quadcopters and provide an estimate of their position, orientation and velocity.
For returning to the vectorial convention for the orientation, it is sufficient to apply Equation (2.3):

$$\hat{\boldsymbol{v}} = \begin{bmatrix} \cos(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) \\ \sin(\theta) \end{bmatrix}.$$

Control loop for trading

In the trading phase the pose of the quadcopter has to remain still. Therefore we use a position controller (PID) and an angle controller (PI) as in Figure 6.2. The same configuration is also used for in the initialization phase, for the initial positioning of the agents.

[1] In practice the three sub-blocks in which we divided the motion planner (optimal velocity computation, collision avoidance and magnitude control) were in implemented in three separated ROS nodes for having a more modular structure.

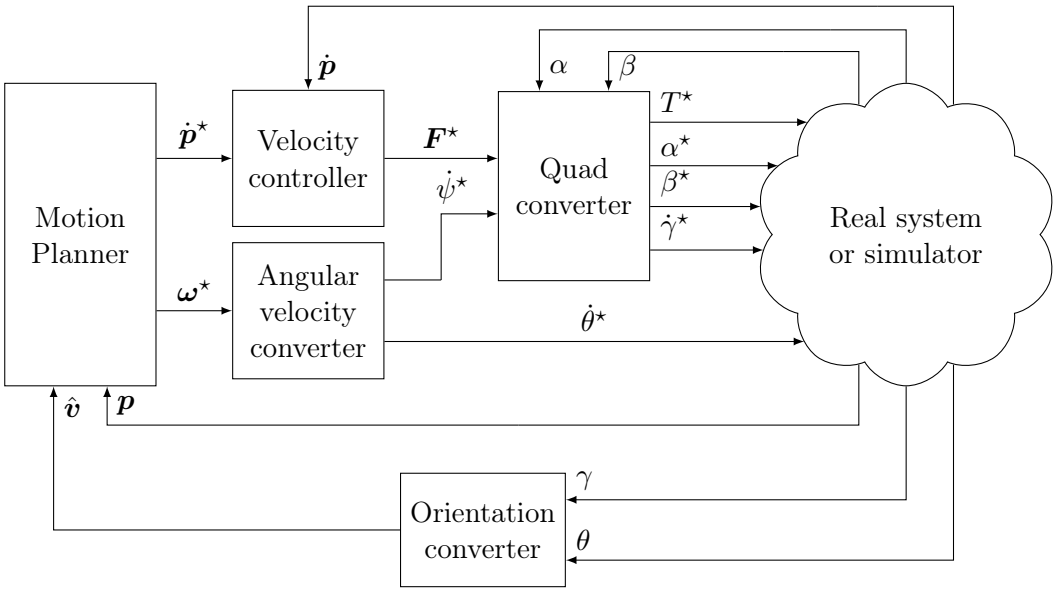


Figure 6.1: Block diagram of the control loop for the pose optimization of one quadcopter. The scheme is slightly simplified because it does not take into account the presence of the other agents.

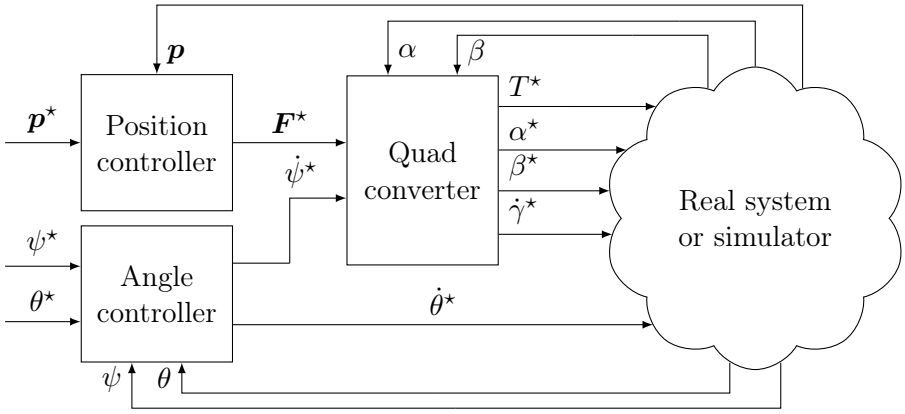


Figure 6.2: Block diagram of the control loop for the trading phase and the initial positioning.

6.2 Experiments

For safety reasons and to reduce the risk of damaging the quadcopters and other equipment, the experiments were made inside an "arena", consisting of a net on the top and the sides and mattresses on the bottom. Moreover a projector was used to dynamically display useful data on the floor of the arena. The arena's

dimensions are approximately $6\,m \times 6\,m \times 4\,m$, while the quadcopters have a diameter of approximately $0.8\,m$. In order to avoid collision with the arena, the mission space Ω was set as a cage of $4\,m \times 4\,m \times 3\,m$, while the safety radius was set to $1.5\,m$ and the worry-distance to $0.5\,m$.

The experiment presented in this section is relative to the inspection of a cube with edge of length $0.5\,m$ using three agents. One of the agent (A, in blue) is a real quadcopter while the other two (B, in red, and C, in green), are simulated. The cube is centred in $[0, 0, 1]^\top$ and has three landmarks on each of the sides and three on the top face. The initial poses of the agents are:

$$\begin{aligned}
\boldsymbol{p}_{A,0} &= \begin{bmatrix} -1 & -1 & 0.75 \end{bmatrix}^\top, & \psi_{A,0} &= 0, & \theta_{A,0} &= 0, \\
\boldsymbol{p}_{B,0} &= \begin{bmatrix} -1 & 1.5 & 2 \end{bmatrix}^\top, & \psi_{B,0} &= 0, & \theta_{B,0} &= 0, \\
\boldsymbol{p}_{C,0} &= \begin{bmatrix} 1 & 1.5 & 0.75 \end{bmatrix}^\top, & \psi_{C,0} &= pi, & \theta_{C,0} &= 0,
\end{aligned}$$

and to each agent was assigned one landmark per face, as in the first screenshot of Figure 6.3. During the experiments the projector was used to display the partition of the landmarks in real time, as well as the positions and orientations of the cameras. The other screenshots in the Figure 6.3 show the evolution of the algorithm over time. In the final configuration all the landmarks are visible by the responsible agent: A and C monitor the sides and B the top face (it is positioned above the cube with the camera facing down).

Finally, Figure 6.4 shows the coverage score over time during the experiment. As expected the value is monotonically increasing. The drop at $t \approx 25\,s$ is only due to a bad synchronization between two agents. In particular there is a trade between A and C, which is disadvantageous for C. After the trade the two agents update their landmarks set, but in this case C does so one step before A, causing an apparent decrease of the coverage score.

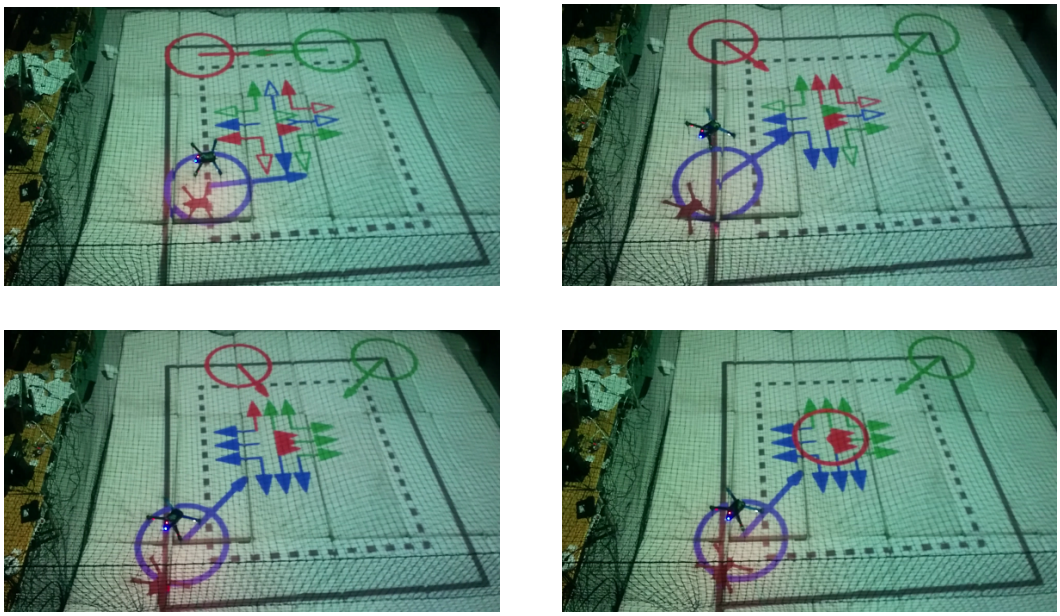


Figure 6.3: Screenshots showing the evolution of the algorithm for the coverage of a cube, in the experiment with one real quad.

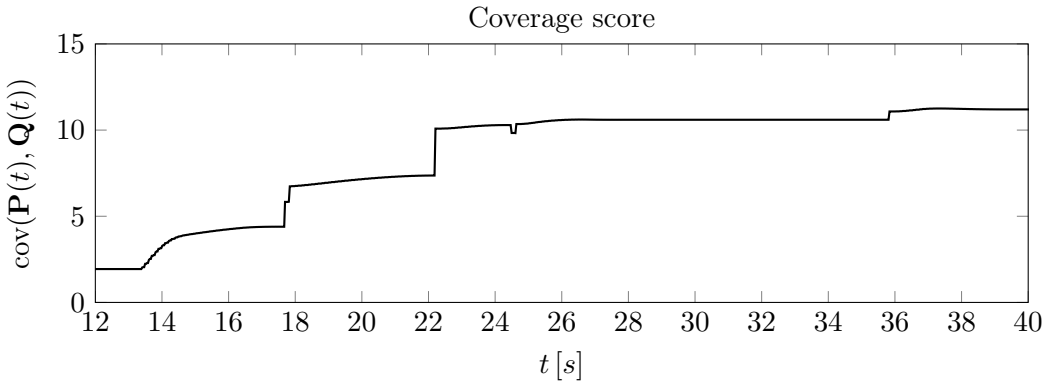


Figure 6.4: Coverage score over time during the experiment.

Chapter 7

Conclusion

In this thesis, the problem of the automatic deployment of vision-based sensors for the inspection of 3D objects was addressed. We started by proposing a vision function based on a discretization of the observed object. This approach is particularly suitable for applications in which some meaningful points of the object can be defined, but can also be applied in the general case by performing a complete discretization of the object's surface. Starting from this definition, an algorithm for the coverage optimization was developed. The algorithm is founded on the alternation of pose optimization of a single agent and trading between pairs of agents. The main idea that was followed was that the coverage score must always increase during the procedure. The implementation follows a modular approach, which allows to test the single components of the algorithm separately and also to replace them, if necessary. For the pose optimization, a gradient search method was followed, and it was integrated with a collision avoidance strategy. For the trading phase, a gossip protocol was followed, meaning that we only allow binary communications and we do not make assumptions on reliability or frequency of them. The algorithm was proved to converge to equilibrium both theoretically and experimentally, using real quadcopters.

In the following section some possible extensions of this thesis are proposed.

7.1 Future work

A generalization of this work could consider the presence of occlusions. As discussed in Chapter 3, this could be done simply by changing the definition of *visible landmark* adopted. The problem of distinguishing which points can be seen from a camera is deeply studied in the problems with camera networks (a wide survey can be found in [28]). However, this change leads to discontinuities in the vision function, which prevent the use of gradient-based methods. Instead, local evaluation of the vision could be exploited, but this could significantly increase the computational complexity of the algorithm.

A second extension could take into account moving landmarks, i.e. dealing with

$(\boldsymbol{q}(t), \hat{\boldsymbol{u}}(t))$. In this case the equation of the time derivative of the vision would be:

$$\frac{\partial}{\partial t}\text{vis}(\boldsymbol{p}, \hat{\boldsymbol{v}}, \boldsymbol{q}, \hat{\boldsymbol{u}}) = \langle \nabla_{\boldsymbol{p}}, \dot{\boldsymbol{p}} \rangle + \langle \nabla_{\hat{\boldsymbol{v}}}, \boldsymbol{\omega}_{cam} \rangle + \langle \nabla_{\boldsymbol{q}}, \dot{\boldsymbol{q}} \rangle + \langle \nabla_{\hat{\boldsymbol{u}}}, \boldsymbol{\omega}_{lmk} \rangle ,$$

where $\dot{\boldsymbol{p}}$, $\boldsymbol{\omega}_{cam}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{\omega}_{lmk}$ are respectively the linear and angular velocities of the camera and the landmark. In this case the conditions for ensuring the increase of the vision would depend on the movement of the landmark, therefore some a priori knowledge about the motion of the landmark could be used. A scenario in which such a problem could arise is the assistance of an agent that has limited sensing capabilities (due to limits in the payload): another quadcopter equipped with a camera could follow it and provide useful information about, for instance, the presence of obstacles.

Appendix A

Useful vector properties

A.1 Gradient of vectorial functions

Given $\boldsymbol{x} \in \mathbb{R}^n$ and $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$, we will denote the gradient of f as $\nabla_{\boldsymbol{x}} f(\cdot)$ and it will be a column vector:

$$\nabla_{\boldsymbol{x}} \left(f(\cdot) \right) = \begin{bmatrix} \frac{\partial f(\cdot)}{\partial x_1} \\ \vdots \\ \frac{\partial f(\cdot)}{\partial x_n} \end{bmatrix}.$$

If instead we consider a vectorial function $\boldsymbol{g}(\cdot) : \mathbb{R}^n \to \mathbb{R}^m$ then the gradient will be a matrix in $\mathbb{R}^{n \times m}$:

$$\nabla_{\boldsymbol{x}} \left(\boldsymbol{g}(\cdot) \right) = \begin{bmatrix} \frac{\partial g_1(\cdot)}{\partial x_1} & \cdots & \frac{\partial g_m(\cdot)}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial g_1(\cdot)}{\partial x_n} & \cdots & \frac{\partial g_m(\cdot)}{\partial x_n} \end{bmatrix}.$$

For any $\boldsymbol{a} \in \mathbb{R}^n$ the following properties for the derivation hold:

$$\nabla_{\boldsymbol{x}} \left(\boldsymbol{x}^\top \boldsymbol{a} \right) = \nabla_{\boldsymbol{x}} \left(\boldsymbol{a}^\top \boldsymbol{x} \right) = \boldsymbol{a}, \tag{A.1}$$

$$\nabla_{\boldsymbol{x}} \left(\|\boldsymbol{x} - \boldsymbol{a}\| \right) = \frac{\boldsymbol{x} - \boldsymbol{a}}{\|\boldsymbol{x} - \boldsymbol{a}\|}, \tag{A.2}$$

$$\nabla_{\boldsymbol{x}} \left(\frac{\boldsymbol{x} - \boldsymbol{a}}{\|\boldsymbol{x} - \boldsymbol{a}\|} \right) = \frac{\boldsymbol{I}}{\|\boldsymbol{x} - \boldsymbol{a}\|} - \frac{(\boldsymbol{x} - \boldsymbol{a})(\boldsymbol{x} - \boldsymbol{a})^\top}{\|\boldsymbol{x} - \boldsymbol{a}\|^3}. \tag{A.3}$$

These relations were taken from [33].

A.2 Time derivative of a vector in rotating coordinate frames

Consider a time varying rotation matrix $\boldsymbol{R}(t)$. Its time derivative can be computed as:

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{S}(\boldsymbol{\omega}(t)) \boldsymbol{R}(t), \tag{A.4}$$

where $\boldsymbol{S}(\boldsymbol{\omega}(t))$ is the skew symmetric matrix:

$$\boldsymbol{S}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{A.5}$$

Now consider a constant vector $\boldsymbol{a}^\star$ and its rotation $\boldsymbol{a}(t) = \dot{\boldsymbol{R}}(t)\boldsymbol{a}^\star$. Then:

$$\dot{\boldsymbol{a}}(t) = \boldsymbol{S}(\boldsymbol{\omega}(t))\boldsymbol{R}(t)\boldsymbol{a}^\star = \boldsymbol{\omega}(t) \times \boldsymbol{a}(t). \tag{A.6}$$

Further details can be found in [27].

Bibliography

[1] P. Pounds, M. Mahony, and P. Corke. Modelling and control of a quad-rotor robot. *Australasian Conference on Robotics and Automation*, 2006. 7

[2] S. Gupte, P.I.T. Mohandas, and J.M. Conrad. A survey of quadrotor Unmanned Aerial Vehicles. *Proceedings of IEEE Southeastcon*, 2012. 7

[3] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J.K. Hedrick. An overview of emerging results in cooperative UAV control. *IEEE Conference on Decision and Control (CDC)*, 2004. 7

[4] H. Voos. Nonlinear Control of a Quadrotor Micro-UAV using Feedback-Linearization. *IEEE International Conference on Mechatronics (ICM)*, 2009. 8

[5] A. Mokhtari, N. K. M'Sirdi, K. Meghriche, and A. Belaidi. Feedback linearization and linear observer for a quadrotor unmanned aerial vehicle. *Advanced Robotics*, 20, 2006. 8

[6] C. Eaton, E. Chong, and A. Maciejewski. Multiple-Scenario Unmanned Aerial System Control: A Systems Engineering Approach and Review of Existing Control Methods. *Aerospace*, 3, 2016. 8

[7] D. Campolo, L. Schenato, L. Pi, X. Deng, and E. Guglielmelli. Attitude Estimation of a Biologically Inspired Robotic Housefly via Multimodal Sensor Fusion. *Advanced Robotics*, 23, 2009. 8

[8] D. Campolo, G. Barbera, L. Schenato, L. Pi, X. Deng, and E. Guglielmelli. Attitude Stabilization of a Biologically Inspired Robotic Housefly via Dynamic Multimodal Attitude Estimation. *Advanced Robotics*, 23, 2009. 8

[9] M.G. Earl and R. D'Andrea. Real-time attitude estimation techniques applied to a four rotor helicopter. *IEEE Conference on Decision and Control (CDC)*, 2004. 8

[10] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. *IEEE International Conference on Robotics and Automation*, 2011. 8

[11] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31, 2012. 8

[12] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5, 1986. 8

[13] J. Van den Berg, S.J. Guy, M. Lin, and D. Manocha. *Reciprocal n-Body Collision Avoidance*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. 8

[14] M. Turpin, N. Michael, and V. Kumar. Decentralized formation control with variable shapes for aerial robots. *IEEE International Conference on Robotics and Automation*, 2012. 8

[15] N. Michael, J. Fink, and V. Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30, 2011. 8

[16] M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar. Swarm distribution and deployment for cooperative surveillance by micro-aerial vehicles. *Journal of Intelligent & Robotic Systems.*, 2016. 8

[17] A. Gusrialdi, S. Hirche, D. Asikin, T. Hatanaka, and M. Fujita. Voronoi-based coverage control with anisotropic sensors and experimental case study. *Intelligent Service Robotics*, 2, 2009. 8

[18] Y. Kantaros, M. Thanou, and A. Tzes. Visibility-oriented coverage control of mobile robotic networks on non-convex regions. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. 8

[19] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28, 1982. 8

[20] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE International Conference on Robotics and Automation*, 20, 2004. 8

[21] L. Mihaylova, T. Lefebvre, H. Bruyninckx, K. Gadeyne, and J. De Schutter. Active Sensing for Robotics - A Survey. *5th International Conference On Numerical Methods and Applications*, 2002. 8

[22] A. Sangwan and R. P. Singh. Survey on Coverage Problems in Wireless Sensor Networks. *Wireless Personal Communications*, 80, 2015. 8

[23] S.J. Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag New York, 1 edition, 1999. 8

[24] J. W. Durham, R. Carli, P. Frasca, and F. Bullo. Discrete Partitioning and Coverage Control for Gossiping Robots. *IEEE Transactions on Robotics*, 28, 2012. 8

[25] Aeroworks 2020, European project. http://www.aeroworks2020.eu/. 9

[26] Smart Mobility Lab at KTH. https://www.kth.se/en/ees/forskning/strategiska-forskningsomraden/intelligenta-transportsystem/smart-mobility-lab. 9

[27] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008. 11, 68

[28] A. Mavrinac and X. Chen. Modeling Coverage in Camera Networks: A Survey. *International Journal of Computer Vision*, 101, 2013. 17, 65

[29] ROS: Robot Operative System. http://www.ros.org/. 59

[30] Python. https://www.python.org/. 59

[31] 3D Robotics. https://3dr.com/. 59

[32] Qualisys: Motion Capture System. http://www.qualisys.com/. 60

[33] K.B. Petersen and M.S. Pedersen. The matrix cookbook. *Technical University of Denmark*, 2008. 67