POLITECNICO DI TORINO

Master's Degree in
Mechatronic Engineering

Master's Degree Thesis

Dictionary of motion primitives for vision-based navigation using Optical Flow

Supervisors	Candidates
Prof. Gianluca Setti	Chiara Boretti
Prof. John Baillieul	Philippe Bich

April 2021

Abstract

In the last twenty years Autonomous Vehicles (AVs) have turned into reality but, despite the technology is becoming increasingly mature, AVs are still only able to reach relatively simple goals in structured environments with large energy consumption. A new generation of more energy efficient systems capable of pursuing complex goals in highly dynamic environments must be created. This is the goal of a MURI Project, sponsored by the U.S. Office of Naval Research (ONR), carried out by Boston University, Massachusetts Institute of Technology and several Australian universities and this is the context in which this thesis came about during the authors' visit to Boston University during the academic year 2020-21.

The objective of the work is to develop a dictionary of motion primitives that exploit visual cues coming from sequences of images acquired by a monocular camera in order to safely guide a mobile robot in unknown environments. From the computation of the optical flow field it is possible to retrieve the values of *time-to-transit*, a quantity probably computed in the animals' visual cortex, that is used in different steering control laws. In order to improve its estimation, negatively affected by rotational motions, a *Sense-Perceive-Act* cycle is introduced. After a filtering operation, an estimate of the environment's geometry is obtained thanks to the analysis of the spatial distribution of *time-to-transit* values and the suitable control law is applied.

The controller has to switch between two main motion primitives: the *Tau Balancing* control law and the *Single Wall* strategy. The former allows the navigation in different scenarios such as straight corridors and turns when the number of features is sufficiently high and uniformly distributed in the image. The latter is employed in situations characterized by feature sparsity.

The entire algorithm has been implemented in the Robotic Operative System (ROS) through nodes written in Python exploiting the OpenCV library. Everything has been tested in Gazebo on a ground vehicle and the simulation results show the ability of the robot to safely navigate in artificial environments (with fixed and a priori defined feature density) as well as in more realistic scenarios (with unknown feature density).

In order to understand the performances of the algorithm on a real platform, it has been deployed on a Jackal robot equipped with a MYNT EYE S1030 camera. Several experiments have been done remotely after sharing code with people from the Boston University Robotics Lab on the same UGV equipped with a Stereolabs Zed 2 camera. The results of this testing phase have been compared to the ones obtained through simulations highlighting the effectiveness of the control system developed.

Acknowledgements

We would like to thank our supervisor Professor Gianluca Setti for giving us the possibility to develop this work at Boston University.

We are highly indebted to Professor John Baillieul, his constant guidance and support during the last six months have been fundamental. His keen interest in our project and his willingness to share his vast knowledge made us complete the assigned tasks on time.

Our thanks and appreciations also go to the whole research group at Boston University with whom we had the chance to share ideas and information during the organized weekly meetings.

Moreover, we would like to express our gratitude to Professor Marcello Chiaberge and to the whole team working at PIC4SeR for giving us the opportunity to test our work on the Jackal robot despite the restrictions due to the pandemic. A special thank goes to Simone without whom we could not have completed our work in such a short time.

Last but not least we would like to thank our families for the love and the support that they have provided us during these challenging years.

Table of Contents

List of Tables

List of Figures

Acronyms

AV Autonomous Vehicle

BRIEF Binary Robust Independent Elementary Features

COM Center of Mass

DoG Difference of Gaussian

EMD Elementary Motion Detector

FAST Features from Accelerated Test

FOE Focus of Expansion

FOV Field of View

FReaK Fast Retina Keypoint

GPS Global Positioning System

GUI Graphical User Interface

hFOV Horizontal Field of View

ICR Instantaneous Center of Rotation

IMU Inertial Measurement Unit

LIDAR Light Detection and Ranging

MAV Micro Air Vehicle

MURI Multidisciplinary University Research Initiative

OF Optical Flow

ONR Office of Naval Research

ORB Oriented FAST and Rotated BRIEF

PIC4SeR PoliTO Interdepartmental Centre for Service Robotics

rBRIEF	Rotation-aware BRIEF
ROI	Region of Interest
ROS	Robot Operating System
SIFT	Scale-Invariant Feature Transform
SSMR	Skid-Steering Mobile Robot
SURF	Speeded-Up Robust Features
TTT	Time-to-transit
UGV	Unmanned Ground Vehicle
URDF	Unified Robot Description Format

Chapter 1

Introduction

In the last twenty years the rapid developments of radar technologies and microprocessor capacity has turned the idea of Autonomous Vehicles (AVs) into reality. The most popular application domain is the one of self-driving cars. These vehicles thanks to rich arrays of sensors such as GPS, LIDAR, cameras, radars and powerful processors can safely drive themselves from a starting point to a predefined destination. Although the technology is increasingly mature, AVs are only able to reach relatively simple goals in structured environments with large energy consumption.

In order to overcome these limitations, Boston University, Massachusetts Institute of Technology and several Australian universities have assembled a team of engineers, computers and neuroscientists to carry out a MURI Project[1] with the goal of developing a new generation of more energy efficient Autonomous Vehicles capable of pursuing complex goals in highly dynamic environments.

1.1 Motivation

Millions of years of evolution allowed animals to develop highly optimized and efficient solutions to survive in many dynamic environments, thus making the animal kingdom a long-standing source of inspiration for scientists and engineers. The main goal is to improve modern technologies thanks to the application of those efficient solutions already used by biological systems.

From an engineering perspective, it is of great interest to understand how animals perceive the environment where they live and what kind of processing algorithms

[1]"Neuro-autonomy: neuroscience-inspired perception, navigation, and spatial awareness for autonomous robots" sponsored by the U.S. Office of Naval Research (ONR). More info at: `http://sites.bu.edu/neuroautonomy/`

they use to move in it. Among the five senses, sight is the most important for many living creatures that use visual cues to navigate, hunt and survive. Different studies have been conducted to understand how animals make the most out of those signals. In [1] it is highlighted the fact that insects strongly rely on cues derived from optical flow and, thanks to them, they are able to estimate distances to obstacles and surfaces and safely explore the environment. In [2] J. Gibson starts to describe some basic optical information available for control but it is D. N. Lee in [3] who, years later, suggests that time-to-collision can play a central role in the perceptual guidance of actions. Y. Wang and B. Frost propose in [4] that this quantity is signalled by neurons in the nucleus rotundus of pigeons.

Despite a growing literature about time-to-transit as a key element for regulating motion behaviors, only in recent years TTT has gained importance in robotics. In favor of this increasing interest, the goal of this thesis is to exploit optical flow data coming from a single camera and to develop a dictionary of motion primitives, that use those data, to safely guide a mobile robot in an unknown environment.

1.2 State of the art

In the last decade, a lot of work has been done in order to bring the natural ability of animals to perceive and move in the environment to robotic systems. In [5], the three-dimensional movements of bats (*M. Velifer*) flying in their natural habitat in Texas have been observed, several steering control laws, based on time-to-transit, are proposed and the trajectories that have been synthesized thanks to these motion primitives are qualitatively bat-like.

Since optical flow seems to have a key role in guiding animal motions, in addition to already being present computer vision algorithms such as the Lucas-Kanade or Horn-Schunck method, some biologically plausible techniques to compute optical flow fields have been developed. In [6], two computational optical flow estimation algorithms, based on the correlational elementary motion detector (EMD), are presented. The correlational EMD is a theoretical model that predicts the interactions between two photoreceptors needed to perceive directional movement of the visual scene.

From Optical Flow (OF) it is possible to retrieve several data that can be used to create new motion primitives and TTT is one of those. Different studies propose the idea that this quantity is computed in the animals' visual cortex and that is why it could be reasonable to use time-to-transit in steering control laws. From bees' flight analysis discussed in [1], Srinivasan *et al.* highlight that these animals, when flying through a narrow passage, position themselves in order to experience the same image velocity in both eyes. This is the concept behind the Tau Balancing control law, introduced in [7] and further explained in [8], that can asymptotically guide a vehicle onto the center line between two corridor walls.

In 2015, Paul Seebacher implemented in [9] a TTT estimator together with a controller that exploited tau values as feedback signals on a Parrot ARDrone2, demonstrating the effectiveness of the Tau Balancing control law for real-time navigation. Three years later, Laura Corvese addressed in [10] the problem of switching between different navigation strategies depending on the environment characteristics. In her work, Corvese also introduced some limitations in the TTT computation for nonholonomic robots, with particular attention to the role of rotational motions. Taking inspiration from the fact that bees and flies separate the rotational and translational components of optical flow via behaviour ([11]), the idea of introducing a Sense-Perceive-Act cycle is proposed. This concept is then further developed in [12] where a possible solution to obtain more robust tau values from Lagrangian Optical Flow is presented.

Even if TTT seems to be an interesting quantity to be used in autonomous navigation, other strategies have been explored. In [13] a bio-inspired sensing and control scheme able to perform small-object detection and avoidance in unknown environments is proposed and experimentally tested on a quadrotor vehicle. The presented approach is based on the extraction of relative range and bearing of small-field obstacles from planar Optical Flow in an environment with small and wide-field obstacles.

Optical flow has two major limitations that are widely described in the literature on bio-inspired robotics. First of all OF only provides coupled information on distance and velocity and this makes more difficult for drones to adapt their reactions when landing, inducing a continuous oscillation of the Micro Air Vehicle (MAV). Another important drawback is that the optical flow field is very small in the direction in which a robot is moving, making the information about frontal obstacles coming from it quite poor. In a recently published article ([14]), these limitations are avoided by exploiting oscillations and by proposing a solution in which robots learn to estimate distances to objects by their visual appearance. The results of the learning process presented in this work form a hypothesis on how flying insects improve, over their lifetime, their navigational skills.

Since the above mentioned studies try to apply some bio-inspired techniques to compute Optical Flow in order to emulate the natural ability of animals to safely navigate in the environment on robotic systems, the need of having bio-inspired sensors arose. In 2013, starting from behavioral, anatomical and electrophysiological data from several species, G. Gremillion, J.S. Humbert and H.G. Krapp developed and characterized a model of the ocellar visual system of flying insects and they then fabricated a fully analog-printed circuit board sensor. Starting from the fact that ocelli and compound eyes can guarantee flight stabilization, the results of the work in [15] demonstrated stabilizing closed-loop feedback with analog ocellar circuitry together with Optical Flow egomotion sensing on a micro-air vehicle.

Bio-inspired robotics is a promising field and solutions for autonomous navigation and obstacles avoidance taking inspiration from the animal kingdom are receiving increasing attention. Despite several accomplishments, scientists and engineers have a lot of research to do before being able to reproduce the sinuosity of animal movements on robotic systems. The research activity can be conducted on different levels starting from the observation of the macroscopic behavior of living beings and the reproduction of these with already existing technologies, till the creation of new sensors, actuators and algorithms that attempt to precisely copy biological functions. It is unclear what level is better to focus on but certainly in the future robotic systems' abilities to replicate animal skills will largely exceed what is possible today.

1.3 Thesis overview

This thesis deals with different contents, from theoretical aspects that allow a better estimate of the time-to-transit and the development of a dictionary of motion primitives to the simulation and the implementation of the control strategy on a Clearpath Jackal UGV. The work is divided into 10 chapters organized as follows:

- **Chapter 2** covers the concept of Optical Flow, explaining the main characteristics of this computer vision technique by introducing also the different elements (detectors, descriptors and matchers) that allow to compute the optical flow field.
- **Chapter 3** introduces *Time-to-transit*, a quantity of paramount importance for the development of the motion primitives. By explaining the difference between Geometric TTT and Perceived TTT, the problems in computing the latter are introduced, together with a possible solution.
- **Chapter 4** is a general introduction of ROS and Gazebo, with particular attention to the aspects needed for the development of the thesis. An overview of the ROS architecture developed and used in this work is also provided.
- **Chapter 5** illustrates the strategy adopted to estimate the optical flow field in the Optical Flow node and it describes how the Tau Computation node produces TTT values and it divides them in five regions of interest (ROIs).
- In **Chapter 6** and **Chapter 7** the different motion primitives used for the control of the Jackal robot are presented, in particular a theoretical analysis and the results obtained from Matlab simulations are provided.
- **Chapter 8** discusses the different scenarios that the robot can face and how to recognize them from the characteristics of TTT signals in the image. Moreover,

different simulation worlds created in Gazebo are presented, starting from artificial environments to more realistic ones.

- **Chapter 9** covers the implementation of the control system on the physical Jackal robot and it provides the results of the tests and a comparison with the simulations is made.
- Conclusions and further work are introduced in **Chapter 10**.

Chapter 2

Optical Flow

What does the world look like when we move through it? Many have tried to find an answer to this question and such is the problem of representing the optical flow field. Optical Flow (OF), or sometimes Optic Flow, is defined as the pattern of apparent motion of image objects between two consecutive frames caused by movement. OF was discussed in the '40s by James Jerome Gibson ([16]), an American psychologist who made many contributions to the field of visual perception, and there followed many further developments in the late '70s and early 80's leading to major advances by Horn and Schunck in [17], Lukas and Kanade ([18]) and Shi and Tomasi ([19]).

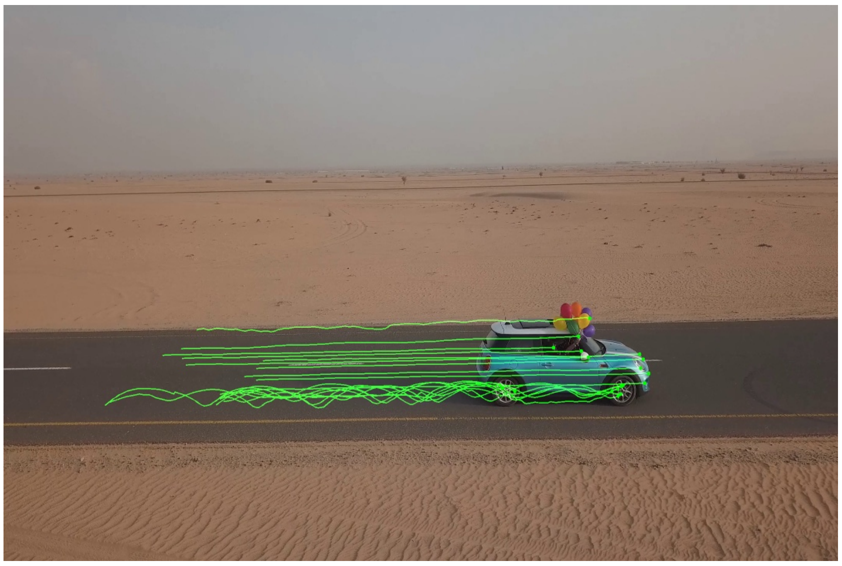


Figure 2.1: Sparse optical flow field for still sideways facing camera.

Optical Flow can be calculated in two forms: dense and sparse. Sparse OF provides the flow vector of some features in the image while dense OF allows to obtain the flow all over the image. To better understand how the Optical Flow relates to the real motion, the following equation is introduced:

$$\phi_{flow} = \frac{v(t)}{d(t)} \cdot sin(\theta(t)) - \omega(t) \tag{2.1}$$

where $v(t)$ represents the object-observer relative velocity, $d(t)$ the distance object-observer, $\theta(t)$ the angle between the image plane and the direction of motion and $\omega(t)$ the object-observer rotational relative velocity. It can be easily noted from (2.1) that flow is larger for objects close to the camera ($d(t) \to 0$) and very small for far-away objects as it is shown in Figure 2.2.

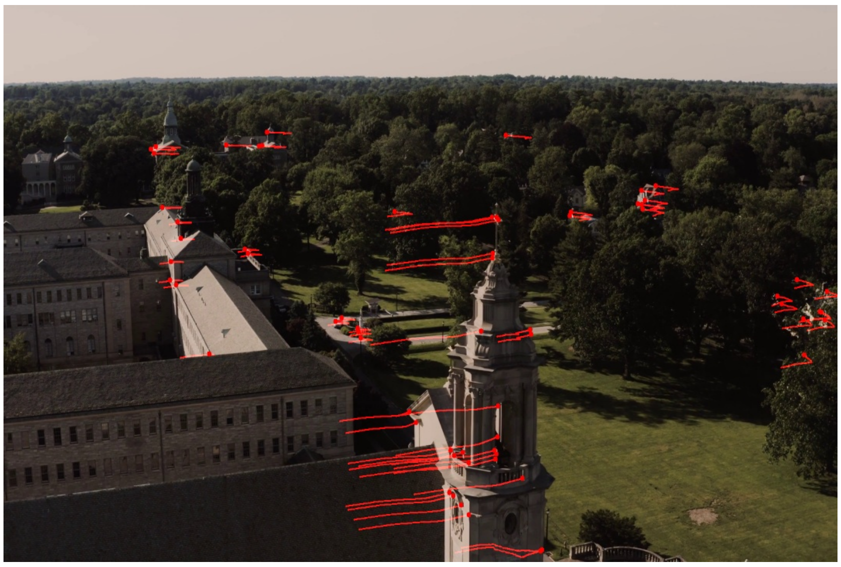


Figure 2.2: Sparse optical flow field from drone footage. Close objects have a larger OF field with respect to far-away ones.

Optical flow field, by its nature, is able to sense relative flow rates between frames but it cannot be used to compute the velocity of the observer nor the distance observer-objects in the scene without knowing some more information, e.g. the exact size of one of those objects. Moreover, as it can be noted in Figure 2.3, in a scenario with a moving observer and many moving elements, the resultant OF field is given by the addition, through superposition, of the flow of the objects and of the observer. That means that objects going towards the observer will have a large flow while a small OF value will be associated to the items moving in the observer's direction. Despite such confounding effects, it will be shown in the next chapter how Optical Flow provides a visual cue for navigation by means of a perceived quantity called *time-to-transit*.

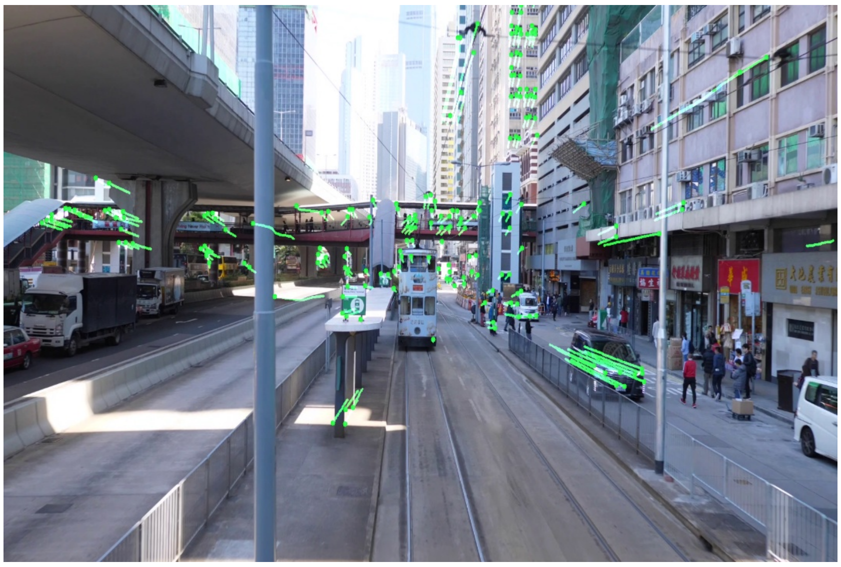


Figure 2.3: Sparse optical flow field in the case of a moving observer and many moving objects.

There are several techniques to compute Optical Flow and in the following sections two classical methods are presented.

2.1 Horn-Schunck method

An early method to calculate Optical Flow was proposed by Horn and Schunck in [17] which allows to produce a dense OF field. It can be applied considering images to be continuously differentiable. The algorithm tries to minimize distortions in the flow preferring the solutions that show more smoothness. As the pattern of brightness moves with time, the brightness of a particular pixel in the pattern remains constant, which means:

$$\frac{dI(x,y,t)}{dt} = 0$$

Applying the chain rule, it can be obtained that:

$$\frac{\partial I(x,y,t)}{\partial x}\frac{dx}{dt} + \frac{\partial I(x,y,t)}{\partial y}\frac{dy}{dt} + \frac{\partial I(x,y,t)}{\partial t} = 0 \tag{2.2}$$

With the small motion constraint, it is possible to represent the motion in its first order approximation:

$$\frac{dx}{dt} = v_x, \qquad \frac{dy}{dt} = v_y \tag{2.3}$$

The problem is formulated as a variational one where the desired vector field is defined as the minimizer of an energy function J given for two-dimensional image streams as:

$$J = \iint \left[(I_x v_x + I_y v_y + I_t)^2 + \alpha^2 (\|\nabla v_x\|^2 + \|\nabla v_y\|^2) \right] \mathrm{d}x \mathrm{d}y \tag{2.4}$$

where I_x, I_y and I_t represent the derivatives of the image intensity values along the x, y and time dimensions respectively. The parameter α is a regularization constant that controls the weight of the smoothness term while ∇ is the Laplace operator and v_x and v_y are components of the optical flow vector. In particular:

$$\nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \qquad \overrightarrow{V} = \begin{bmatrix} v_x(x,y) \\ v_y(x,y) \end{bmatrix}$$

Although this method allows the computation of a dense optical flow field with velocity vectors for each pixel in the image, it is susceptible to noise and it is computationally expensive. These characteristics make the Horn-Schunck method not ideal for real-time applications. Another algorithm to estimate the optical flow field in a more sparse manner is presented in the next section.

2.2 Lucas-Kanade method

An highly popular alternative method to estimate Optical Flow is the one developed by Bruce D. Lucas and Takeo Kanade in [20]. This technique allows the computation of a sparse optical flow field by optimizing an energy function at each given pixel using predetermined feature points in the image. Local methods do not offer information about the whole flow field but they are robust in the presence of noise and they are less computationally expensive so they can be used in real-time applications.

The Optical Flow equation can be written, after a first order Taylor expansion, in its most popular form as in (2.2) where $\frac{dx}{dt} = v_x$ and $\frac{dy}{dt} = v_y$ are the horizontal and vertical components of the OF velocities while I is the image intensity. The Lucas-Kanade method assumes a small and approximately constant displacement of the features in the image between two successive frames within a neighborhood (of size n) of the considered point p. So, given this assumption, it is possible to obtain the following set of equations:

$$\frac{\partial I}{\partial x}(p_i) v_x + \frac{\partial I}{\partial y}(p_i) v_y + \frac{\partial I}{\partial t}(p_i) = 0 \quad | \quad i = 1, ..., n$$

In order to try to determine the values of v_x and v_y deriving from this over defined system in two unknowns, a least squares optimization is performed after the definition of the following quantities:

$$A = \begin{bmatrix} \frac{\partial I}{\partial x}(p_1) & \frac{\partial I}{\partial y}(p_1) \\ \vdots & \vdots \\ \frac{\partial I}{\partial x}(p_n) & \frac{\partial I}{\partial y}(p_n) \end{bmatrix} \qquad v = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \qquad b = \begin{bmatrix} \frac{\partial I}{\partial t}(p_1) \\ \vdots \\ \frac{\partial I}{\partial t}(p_n) \end{bmatrix}$$

It can be written that:

$$v = A^+ b$$

where A^+ is the Moore-Penrose inverse of matrix A.

The Lucas-Kanade method, by its nature, works well for small movements but it fails when large pixel motion occurs. A possible solution to this problem is represented by the adoption of pyramids. Every level of the pyramid, starting from its base, has a lower resolution with respect to the previous one. The tracking of the features begins in the level with the lowest resolution where large pixel motion is reduced to small motion and it goes on until convergence. This technique enables the algorithm to perform well even when the displacement of features between two frames is larger then the neighborhood size. In Figure 2.4 a diagram shows the structure of a pyramid with three levels.

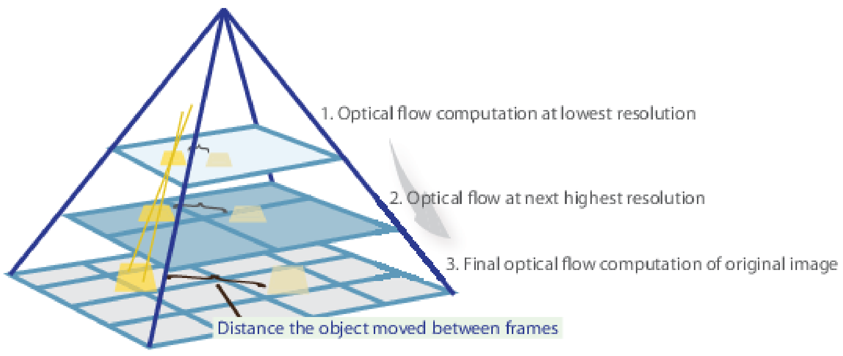

1. Optical flow computation at lowest resolution
2. Optical flow at next highest resolution
3. Final optical flow computation of original image
Distance the object moved between frames

Figure 2.4: The pyramid method enables Lucas-Kanade algorithm to handle large pixel motion. *Source: https://www.mathworks.com*

The comparison between the results obtained applying a sparse Optical Flow estimation method such as the one proposed by Lucas and Kanade and a dense OF estimation technique is presented in Figure 2.5.

2.3 Detectors, descriptors and matchers

In order to exploit Lucas-Kanade algorithm for the estimation of the optical flow field, it is important to rely on good and robust feature points which have to be

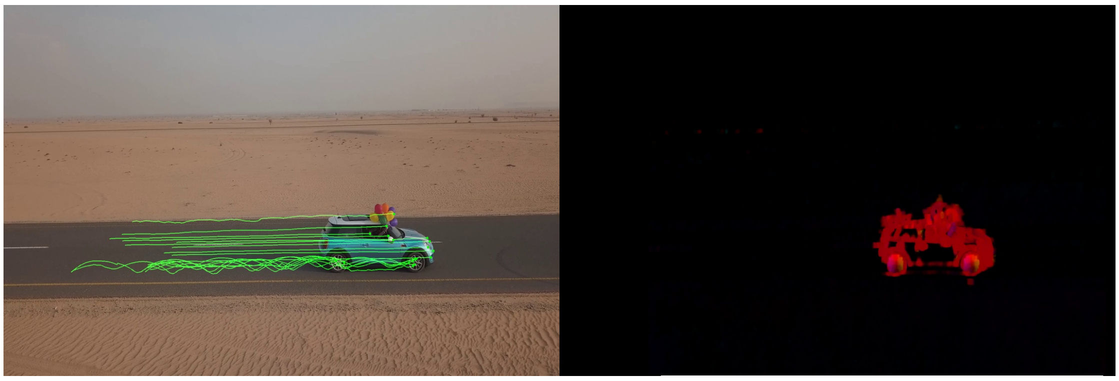


Figure 2.5: Comparison between a sparse optical flow field computed using the Lucas-Kanade algorithm on the left and a dense OF field on the right (Gunnar-Farneback).

invariant to pose changes, distinctive and detectable even if there are modifications in the viewing conditions.

The process used to obtain robust keypoints is composed of three steps: the first is called features detection, the second features description and the last features matching. Each phase can be implemented by different types of algorithms. Some of them are more computationally efficient but less accurate, others are slower but more precise and so the choice of which one is better really depends on the final goal to achieve.

2.3.1 Features Detection and Features Description

Feature detection is performed by using algorithms, known as detectors, which are typically able to detect only one type of feature. The final objective is to find notable points in the image that can be divided in three main groups:

- Edges: areas in the image in which the gradient is high.
- Corners: areas in the image in which the gradient and the curvature are high.
- Blobs: areas in the image in which some properties are considered almost constant.

Descriptors are necessary to identify feature points and to make the process of finding them in the subsequent frames easier. Descriptors store local information around a particular point and use them to uniquely identify that feature. These data are also useful to guarantee the scale or rotation invariance. The choice of the descriptor is crucial because they are computationally expensive and influence the duration of the matching phase where the descriptors' vectors are used to match the different keypoints from frame to frame.

Over the years, several algorithms have been implemented and they differ from each other for repeatability, computational performance and speed. An overview of the most common detectors and descriptors is now presented[1]:

1. **Harris Corner Detector**: an early attempt to detect corners in an image was performed by Harris and Stephens in 1988 ([18]). This method consists in finding the maximum difference in intensity for every pixel windows of a predefined size. It is possible to define a change function $E(\Delta x, \Delta y)$ that has to be maximized:

   $$E(\Delta x, \Delta y) = \sum_{x,y} w(x,y)[I(x+\Delta x, y+\Delta y) - I(x,y)]^2 \tag{2.5}$$

   where $w(x,y)$ is the window function, $I(x+\Delta x, y+\Delta y)$ is the shifted intensity and $I(x,y)$ is the current intensity. Applying a Taylor expansion to approximate $I(x+\Delta x, y+\Delta y)$ and considering I_x and I_y as image derivatives in x and y directions, it is possible to obtain an equation of the form:

   $$E(\Delta x, \Delta y) \approx \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} M \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \tag{2.6}$$

   where M is a matrix with components related to I_x and I_y. Harris and Stephens defined a score, that is an equation used to understand if a pixel window can contain a corner:

   $$R = det(M) - k(trace(M))^2 \tag{2.7}$$

   Since R depends on the determinant and the trace of the matrix M, its eigenvalues' magnitude (λ_1 and λ_2) determines if a particular area is a corner or a flat region, following these criteria:

   - $\mid R \mid$ small, when both λ_1 and λ_2 are small, means flat region
   - $\mid R \mid < 0$, when $\lambda_1 \gg \lambda_2$ or viceversa, means edge region
   - $\mid R \mid$ large, when both λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, means corner region

   The result of the Harris corner detection technique is a gray-scale image with different scores. The application of a suitable threshold to these values provides the corners in the image.

2. **The Scale-Invariant Feature Transform (SIFT) Detector and Descriptor** was proposed by Lowe ([21]) and it is an improvement of the Harris

[1]More complete information about detectors and descriptors can be found in the *OpenCV* documentation: `https://opencv.org`.

corner detector because it is scale and rotation invariant. The SIFT method consists in extracting the keypoints and in computing their descriptors by implementing four different steps. The first is represented by the scale space creation in which, starting from the original image, the algorithm generates several octaves (collection of images of the same size). Within each octave, images are progressively blurred using a Gaussian blur operator by a certain amount called scale (σ). The number of octaves and scales depends on the size of the original image. Once the scale space has been created, a new set of images has to be generated by using Difference of Gaussians (DoG) which is an approximation of the Laplacian of Gaussians. The DoG are obtained by subtracting two consecutive images in the same octave as shown in Figure 2.6 and they are scale-invariant.

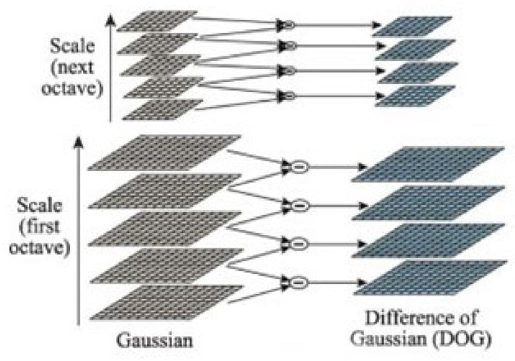


Figure 2.6: Difference of Gaussians. *Source: https://biomedpharmajournal.org*

In these images it is necessary to locate the maxima and the minima by iterating through each pixel and check its neighborhood. This check is performed within the current image, but also with the one above and the one below (notice that the lowermost and topmost scales are skipped because there are not enough neighbors to do the comparison). If at the end of this checking process the considered pixel is the smallest or the greatest of all its neighbors, it is marked as *keypoint*, otherwise it is discarded (this could occur also after few initial checks).

The second step of the SIFT algorithm has the goal of refining the location of the keypoints that come from the previous step. To do that a second order Taylor expansion of the image around the keypoints is used and the result is a more accurate position called the subpixel keypoint location or extremum. Since some of the found extrema lay on an edge or do not have enough contrast, two thresholding methods are applied to get rid of them. The keypoints in a subpixel location whose magnitude is lower than a certain threshold are rejected while for the ones on the edges a strategy similar to the one used by the Harris corner detector is adopted. A 2x2 Hessian matrix is used to compute the principal curvature and, if the ratio of the eingenvalues of the matrix is higher than a defined level, the keypoint is discarded. After this filtering process, the points at subpixel locations that remain are considered to be the strongest and the most interesting keypoints.

The third step is important for the rotation invariance and it consists in assigning to each keypoint an orientation. A neighborhood, that depends on the scale, is considered around a keypoint location and the gradient direction and magnitude is computed for every pixel in that neighborhood. With the obtained values a histogram is created and at the peak of the histogram corresponds the orientation to be assigned at the keypoint. In some cases the histogram could present more than one peak and this leads to the creation of a new keypoint that has the same location and scale of the original one but with the orientation specified by the second peak.

Finally, in the fourth step, the descriptor of each keypoint is defined. A 16x16 neighborhood around the point of interest is considered which is then divided into 16 4x4 sub-blocks. For each sub-block, an 8 bin orientation histogram is created. The keypoint descriptor is a vector representing the 128 normalized bin values available. In addition to this, several techniques are used to improve robustness against rotation and illumination changes. Although the SIFT detector and descriptor is really accurate, it is not suitable for real-time applications because of the large amount of computation necessary to determine the descriptor vector.

3. **Features from Accelerated Segment Test (FAST) Detector** was introduced in 2006 by E. Rosten, R. Porter and T. Drummond ([22]) and it is one of the most appropriate detection methods for real-time applications. The algorithm starts by selecting a pixel p with intensity I_p from an image and by choosing a threshold value t. In order to understand if the point p is a corner or not, a circle of 16 pixels around p is considered and if there exists a set of n (typically n=12) contiguous pixels (in the circle) that are all brighter than $I_p + t$ or all darker than $I_p - t$ then the point of interest p is a corner. In particular, this procedure is divided in two parts. The first one is called high-speed test and it considers the pixels located at the four cardinal points of the circle and if at least three of them are brighter or darker than p the point is considered, otherwise it is immediately discarded. Then, to the surviving candidates, the full segment test criterion in which all the pixels in the circle are examined is applied.

   This detection technique has some weaknesses that can be avoided by using a machine learning approach and a non-maximum suppression technique. The former selects a set of images for training and it applies to each of them the

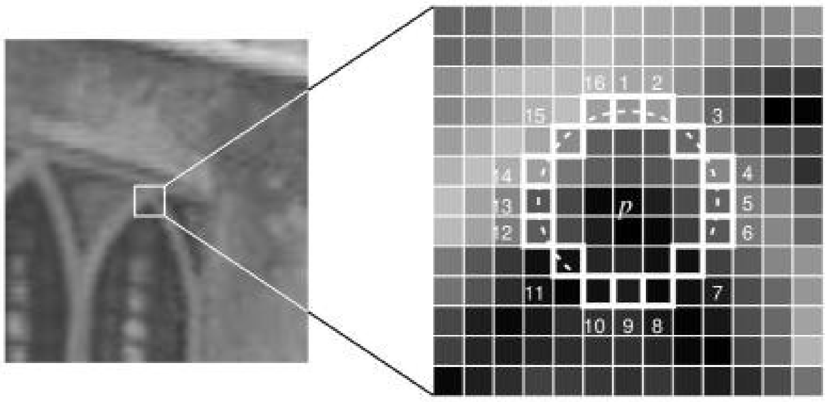


Figure 2.7: Pixels used by FAST detector. *Source: https://medium.com*

FAST algorithm. A vector P, that stores the values of the 16 pixels around the feature points found, is created. A state is then associated to every pixel x belonging to P in order to define three subsets called P_d, P_s, P_b. The three states are the following:

$$S = \begin{cases} d, & I_x \leq I_p - t \qquad (darker) \\ s, & I_p - t < I_x < I_p + t \qquad (similar) \\ b, & I_p + t \leq I_x \qquad (brighter) \end{cases} \tag{2.8}$$

This machine learning technique tries to extract the pixels that carry the most information and the decision tree that is created during the training is used for fast corner detection in other images. The non-maximum suppression strategy is a post processing step used to avoid the detection of multiple interest points in adjacent locations. It computes, for all the detected feature points, a score function V which is the sum of the absolute difference between p and 16 surrounding pixel values. When in presence of two adjacent keypoints, the point with the lowest V value is rejected. The FAST detector is suitable for real-time applications but it is sensitive to high levels of noise.

4. **Binary Robust Independent Elementary Features (BRIEF)** is a descriptor, introduced by M. Calonder *et al.* in [23], that converts image patches, containing all the pixels in the neighborhood of a keypoint, into binary feature vectors. Before computing these quantities, a Gaussian Kernel is applied to smooth the image and to make the feature descriptors stable and repeatable. From the smoothed patch, the algorithm defines a binary feature vector obtained from the response of binary tests that consist in comparing the intensities of the patches in the positions defined by two pixels of a random pair (x, y). The number of tests to be performed is n which could be 128, 256 or 512 depending on the length of the binary feature vector.

The advantage of using a BRIEF descriptor is the fast matching procedure (obtained if it is used the Hamming distance) with respect to other descriptors, but this technique does not perform well when invariance to large in-plane rotations is required.

5. **Oriented FAST and Rotated BRIEF (ORB) Detector and Descriptor**. This algorithm was introduced in 2011 ([24]) and it takes the advantages of two already known methods (FAST and BRIEF) but it improves them to reduce some of their weaknesses. The FAST detector is enhanced by introducing multiscale feature detection and orientation components to the keypoints. ORB uses a multiscale image pyramid, represented in Figure 2.8, in which there is a sequence of the same image but with different resolutions. Once the pyramid is created, the FAST algorithm runs on the images to detect keypoints at different levels making ORB partially scale invariant.

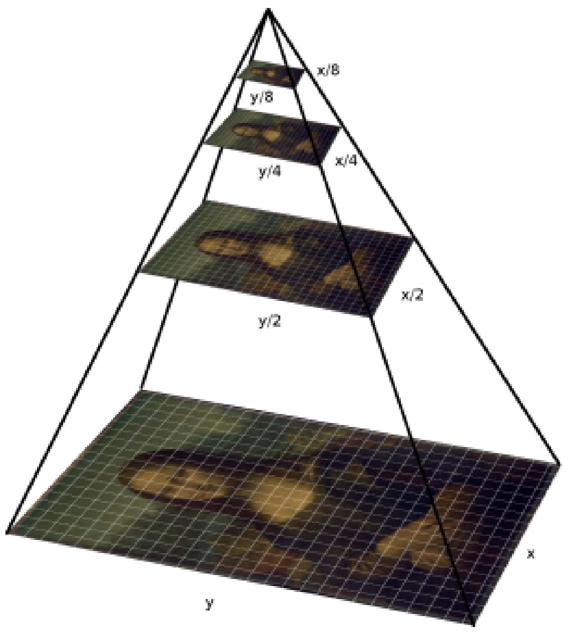


Figure 2.8: Multiscale image pyramid used by the ORB algorithm. *Source: https://medium.com*

After locating all the keypoints, the algorithm uses the intensity centroid to evaluate changes in the intensity level around a feature point and it assigns an orientation to the patch, thus obtaining some rotation-invariance. When the detection phase is completed, the BRIEF descriptor computes the binary feature vectors, but since it is not rotation invariant, an improvement has been introduced which transforms the BRIEF in the Rotation-Aware BRIEF (rBRIEF). For what regards the task of feature detection, the ORB algorithm performs as well as SIFT while it is faster by almost two order of magnitudes. Moreover, while SIFT and SURF are patented, ORB is open source and free to use.

6. **The Fast Retina Keypoint (FReaK) Descriptor** was introduced in 2012 by A. Alani *et al.* ([25]). This algorithm takes inspiration from the analysis of the human eye, in particular of the retina. The eye has a huge number of ganglion cells, responsible for observing the region of focus, whose density is not uniform, indeed in the neighborhood of the fovea it decays exponentially. However, also less dense cells are important for retrieving information about the orientation.

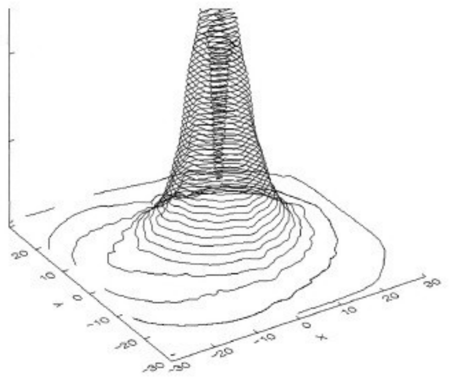


Figure 2.9: Density of the ganglion cells over the retina and retina areas. *Source: [25]*

The sampling pattern of the FReaK descriptor, shown in Figure 2.10, is composed of a number of individual circles, called receptor fields, whose density is higher around the keypoint and decreases in the neighborhood. A sort of redundancy is introduced by overlapping adjacent receptive fields and this leads to a better efficiency and to a reduction in the number of circles.
The resulting descriptors are binary strings formed by a sequence of single bits coming from the usage of Difference of Gaussians. These descriptors are constructed by thresholding the difference between the intensities of pairs of receptive fields, each one smoothed with a Gaussian kernel that has standard deviation equal to the radius of the circle. To improve the efficiency of the algorithm, a learning process is introduced to select the best pairs and this leads to a coarse-to-fine ordering of these couples. The results show that the first 512 pairs are sufficient to create a descriptor. Orientation is then determined by applying a distance scaled weighted average to the result of the Difference of Gaussians between two paired regions.

The FReaK algorithm is computationally-efficient and it requires a small

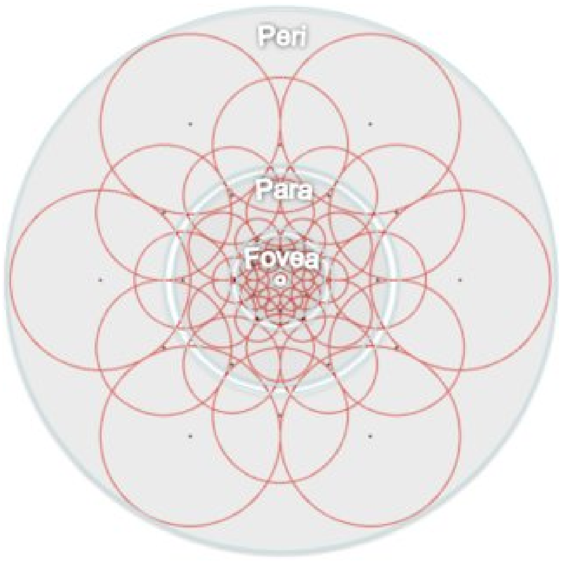


Figure 2.10: FReaK sampling pattern. *Source: [25]*

amount of memory. Moreover, since it is a binary descriptor, the matching phase takes less computation time (provided the usage of the Hamming distance). All these aspects make this description method highly suited for real time applications.

2.3.2 Features Matching

Once the feature points have been detected and described, the last stage, which consists in matching them between different frames, takes place. Two main matching techniques can be defined: library matching and cross image matching. The former considers keypoints from an image database while the latter considers descriptors coming from two images representing the same scene.

For what regards the second technique, the most common approach is Brute Force which consists in taking the descriptor of one feature in the first image and matching it with all the feature descriptors in the second one. This operation returns the closest feature point. The matching is performed through distance calculations based on L1 or L2 norms or the Hamming distance which is the favorite in case of BRIEF or FReaK descriptors since it is more computational efficient.

2.3.3 Implementation Choices

Since the goal of the work is to develop a controller capable of running in real-time on a robotic platform, advantages and disadvantages of the different detectors, descriptors and matchers and of the Optical Flow estimation techniques have been considered.

The Lucas-Kanade method in its pyramidal form has been chosen as Optical Flow estimation technique because it is fast and computational efficient since it has to compute only a sparse optical flow field. The features for which the OF vectors are estimated are selected using the ORB detector and descriptor because it is open source and it is suitable for real-time applications.

Chapter 3

Time-to-transit

Time-to-transit (TTT), identified by τ (tau), is a quantity that expresses the amount of time before which an object will cross the image plane of a moving observer. This term was introduced in 2012 by K. Sebesta and J. Baillieul ([7]) and it comes from the concept of *time-to-collision*, a name coined by scientists after several studies (e.g. [3], [4], [26]) that demonstrate its relevance in animals' avoidance and tracking behaviors.

3.1 Geometric and Perceived Time-to-transit

Time-to-transit is a quantity that can be used in different control laws but, in order to handle it properly, an important distinction between the geometric and the perceived TTT has to be made. To fix ideas, a simple unicycle vehicle is considered with kinematics:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}vcos\theta\\ vsin\theta\\ u\end{bmatrix} \tag{3.1}$$

where u is the turning rate, v is the forward speed in the direction of the body-frame's x-axis and θ is the heading of the vehicle.

The geometric time-to-transit is defined as the time it would take the vehicle with $v(t)$ and $\theta(t)$ held constant to cross a line intersecting the feature, located in (x_f, y_f), and perpendicular to the current heading. In mathematical terms, looking at Figure 3.1, it can be written as:

$$\tau_g(t) = \frac{cos\theta(t)(x_f - x(t)) + sin\theta(t)(y_f - y(t))}{v} \tag{3.2}$$

Geometric τ is considered as a proxy for the actual distance from an object and for this reason is taken as the ground-truth for visual depth estimation. However, as it can be noticed in (3.2), it is not possible to evaluate τ_g without some information on the geometry of the environment and on the features' position, making this quantity not suitable for navigation in unknown scenarios.

One of the most important data that can be retrieved from optical flow field estimation is the perceived time-to-transit that can be evaluated only through the pixels' motion in the image plane and without any knowledge on the size of the objects, the distance from them or the velocity of the observer. All these advantages, together with the fact that it is probably a quantity used also by animals, make perceived TTT a great candidate for exploitation in control laws. The calculation of TTT for a moving observer approaching an object can be done by solving a similar triangles problem. In Figure 3.1 point O is the feature, f is

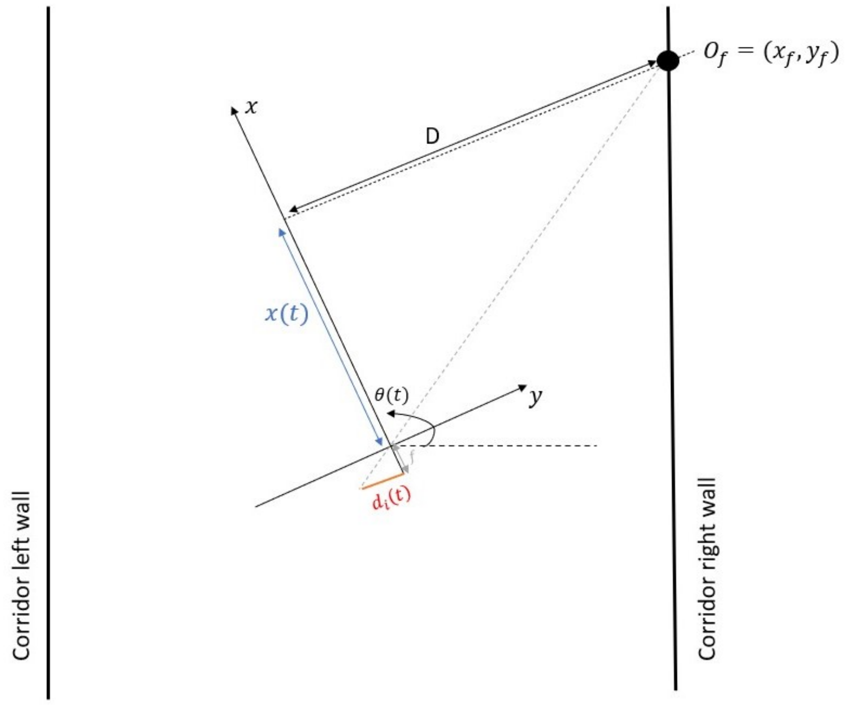


Figure 3.1: A single camera moving forward with constant linear velocity.

the focal length, D is the real distance between the point O and the observer in y-direction, $x(t)$ is the distance from the camera to the object in the x-direction and $d_i(t)$ is the distance to the feature projected on the image plane. Considering the similar triangles, this relation can be written as:

$$\frac{D}{x} = \frac{d_i}{f} \tag{3.3}$$

In order to manage the fact that the movement of the observer leads to changes in the scale of the object in the image plane, from (3.3) it is possible to develop the following differential equation:

$$\dot{d}_i(t) = \frac{v}{D \cdot f} d_i(t)^2 \tag{3.4}$$

and, provided that the velocity v is kept constant, it is easy to derive the solution:

$$d_i(t) = \frac{D \cdot f}{x_0 - vt} \tag{3.5}$$

where x_0 is the initial distance from the point O. To completely release the computation of the perceived TTT from the knowledge of D, x_0 or v it is possible to manipulate (3.3):

$$D \cdot f = d_i(t) \cdot x(t) \tag{3.6}$$

Then, under the assumption of constant velocity, by substituting $d_i(t)$ with the expression obtained in (3.5), *perceived time-to-transit* can be finally defined as:

$$\tau = \frac{x_0 - vt}{v} = \frac{d_i(t)}{\dot{d}_i(t)} \tag{3.7}$$

Equation (3.7) shows that τ can be estimated by considering only the motion of the pixels in the image plane (or on the retina if a moving animal). This quantity captures mixed information about real world distances and velocities and it can assume positive or negative values, if the feature associated to that τ is approaching or retreating respectively. The fact that time-to-transit values can be computed using only one sensor (a camera) and that they give an idea of how a moving observer can perceive the real world makes the usage of TTT perfect in feedback-control laws for vision-based systems.

However, it is of paramount importance to understand if perceived time-to-transit is always a reliable signal to be used for navigation purposes. There are several factors that can negatively affect the perception of the visual cues, making perceived TTT different from the one considered as the ground truth (the geometric TTT) and some of them are presented in the list below:

- Features that stay for a very short time in the field of view
- Too high sparsity of features that leads to flow uncertainty
- Discontinuities in the flow associated with boundaries that can be easily confused with noise
- Moving objects that produce localized flow vectors that could be inconsistent with the Optical Flow generated by self-motion
- Rotational motions of the observer

Regarding the last point, it is possible to notice how the rotational components of the motion affect the perceived τ by analyzing the following equations, presented by E. C. Hildreth *et al.* in [27]:

$$\begin{aligned} \dot{x} &= \frac{-T_x + xT_z}{Z} + R_x xy - R_y(x^2+1) + R_z y \\ \dot{y} &= \frac{-T_y + yT_z}{Z} + R_x(y^2+1) - R_y xy + R_z x \end{aligned} \tag{3.8}$$

where $(\dot{x}, \dot{y})$ is the projected velocity of an image feature located in (x, y) in the image plane, (T_x, T_y, T_z) and (R_x, R_y, R_z) are the components of translation and rotation matrices and $Z(x, y)$ is the depth of the surface point projecting to image location (x, y). In (3.8), the presence of rotational components influences the projected velocities by introducing additional terms to the one depending on Z and this leads to the impossibility of considering the perceived TTT as a proxy for depth.

In the following section, a possible solution is proposed with the goal of mitigating the effect of rotational motions in order to make the perceived quantities more similar to the geometric ones, giving them robustness and reliability for their usage in steering control laws. Different simulation results will be then presented and, in order to completely understand them, the difference between *Eulerian* and *Lagrangian Optical Flow sensing* must be discussed. The former assumes that each photoreceptor in the camera (or in the eye) can instantaneously detect in the image the feature points and their velocities. The latter is the form of visual sensing on which standard computer vision techniques are based and it consists in tracking the keypoints as they move in the retina.

3.2 Path segmentation

Starting from the results presented in [12], an algorithm to mitigate the effects of rotational components in the optical flow field on the estimation of time-to-transit values is developed. The idea behind this method is simple and it consists in the creation of a generic path by interleaving short straight lines with constant curvature segments. This leads to the introduction of a Sense-Perceive-Act cycle in which the sensing phase is performed when moving straight.

The duration of the sensing and of the acting must be chosen properly. Only a correct balancing of the two phases allows the possibility of navigating in the environment. An instantaneous sensing is possible only in an ideal scenario in which noise is not present. In a real environment the sense phase must be long enough to obtain a good estimation of time-to-transit values, but sensing for too much time means not being able to detect instantaneous variations of the optical flow field potentially losing important information coming from the environment.

Moreover, the sense-act interleaving must guarantee the stability of the control law applied in the acting phase. From a practical point of view, the duration of the two phases must be selected according to the specific application since it depends on the velocity that the robot should track and on how quickly it can turn. On the right in Figure 3.2, sensing (in black) and acting (in cyan) are correctly balanced and the resultant trajectory approximates the behavior that can be obtained with instantaneous sensing (on the left) while, in the center, a wrong balancing of the duration of the sense and of the act phases prevents from achieving the desired performance.

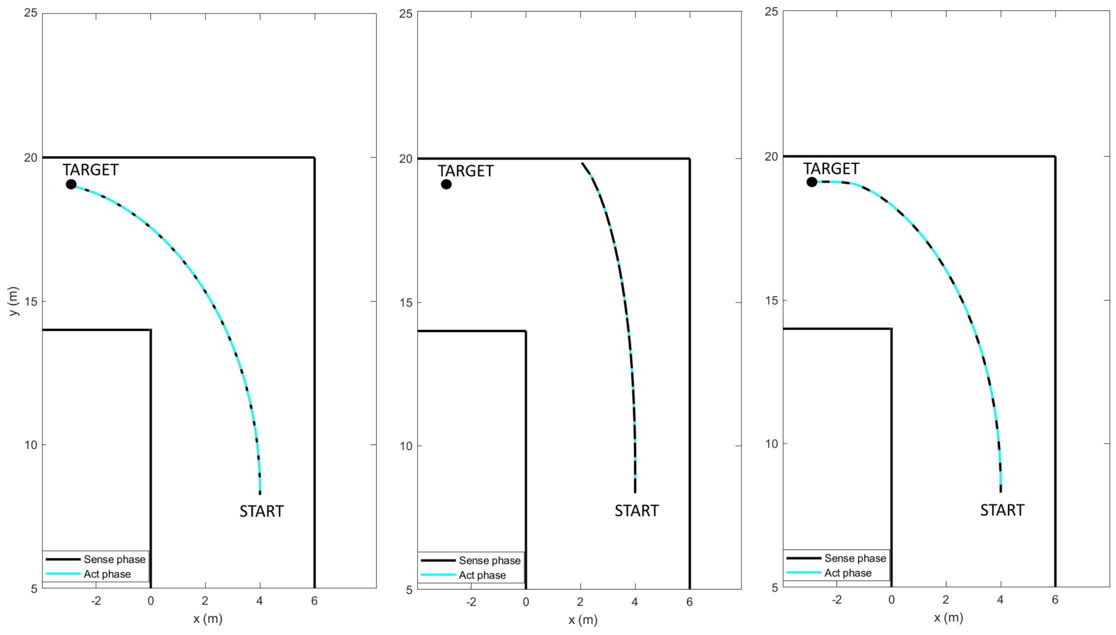


Figure 3.2: Resulting trajectories for act and sense phases with different duration. In particular, on the left, sensing is instantaneous (Eulerian sensing).

Algorithm 1 explains how the Sense-Perceive-Act cycle is implemented. Once all the relevant variables are set, a **while** cycle starts. The function RobotStatus() returns *False* only in the case the robot must be stopped. The computation of the TTT values is done by ComputeTTT() while the choice of the action to be performed, a topic discussed later in this work, is taken by ChooseAction(). The function GetTime() returns the time at which it has been called and *all_TTT* is a collection containing time-to-transit values and information about their distribution.

Algorithm 1 Sense-Perceive-Act cycle implementation

```
function MOVE($\alpha, \beta$)
    $sense \leftarrow True$
    $sense\_duration \leftarrow \alpha$
    $sense\_counter \leftarrow 0$
    $act \leftarrow False$
    $act\_duration \leftarrow \beta$
    $act\_counter \leftarrow 0$
    $all\_TTT.initialize()$
    ▷ Execution
    while ROBOTSTATUS() do
        if $sense = True$ then
            if $sense\_counter = 0$ then
                $sense\_start \leftarrow$ GETTIME()
            else if GETTIME() $-$ $sense\_start >= sense\_dur$ then
                $sense\_counter \leftarrow 0$
                $sense \leftarrow False$
                $act \leftarrow True$
                continue
            end if
            $action \leftarrow straight$
            COMMANDROBOT($action$)
            $new\_TTT \leftarrow$ COMPUTETTT()
            $all\_TTT.add(new\_TTT)$
            $sense\_counter \leftarrow sense\_counter + 1$
        else if $act = True$ then
            if $act\_counter = 0$ then
                $act\_start \leftarrow$ GETTIME()
                $action \leftarrow$ CHOOSEACTION($all\_TTT$)
            else if GETTIME() $-$ $act\_start >= act\_dur$ then
                $act\_counter \leftarrow 0$
                $act \leftarrow False$
                $all\_TTT.deleteAllItems()$
                $sense \leftarrow True$
                continue
            end if
            COMMANDROBOT($action$)
            $act\_counter \leftarrow act\_counter + 1$
        end if
    end while
end function
```

3.2.1 Simulation Results

In order to understand how the introduction of the Sense-Perceive-Act cycle affects the estimation of time-to-transit values, some tests have been run in Gazebo[1]. A sketch of the simulation environment is shown in Figure 3.3.

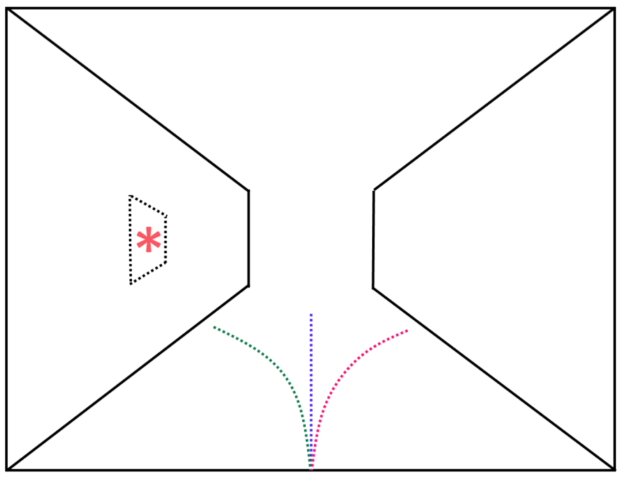


Figure 3.3: A representation of the simulation environment used.

A single feature, represented in the sketch by an asterisk, is inserted on the left of a corridor, three different trajectories are performed and the geometric and the perceived TTT values are computed in the case no Sense-Perceive-Act cycle is used. For all the simulations presented in this section tau-based Optical-Flow sensing is assumed to be Lagrangian and the forward linear velocity equal to 0.5m/s.

The results of the tests are summarized in Figure 3.4. When moving forward, geometric and perceived time-to-transit are quite similar. When turning away from the feature (rightward), it can be noted that geometric TTT tends to decrease faster with respect to when moving forward. In this case, perceived time-to-transit is very noisy and it differs a lot from the geometric one. An even worse scenario arises when turning towards the feature (leftward). In this case perceived TTT cannot be assumed as a proxy for the geometric τ anymore since the two curves have nothing in common. Figure 3.5 represents the results already discussed in a different way in order to facilitate the comparison between geometric and perceived TTT during the three tests.

With the data obtained from these simulations, it is clear that a dictionary of motion primitives based on perceived time-to-transit (as approximation of the

[1]More information about the Gazebo simulation environment can be found in Chapter 4.

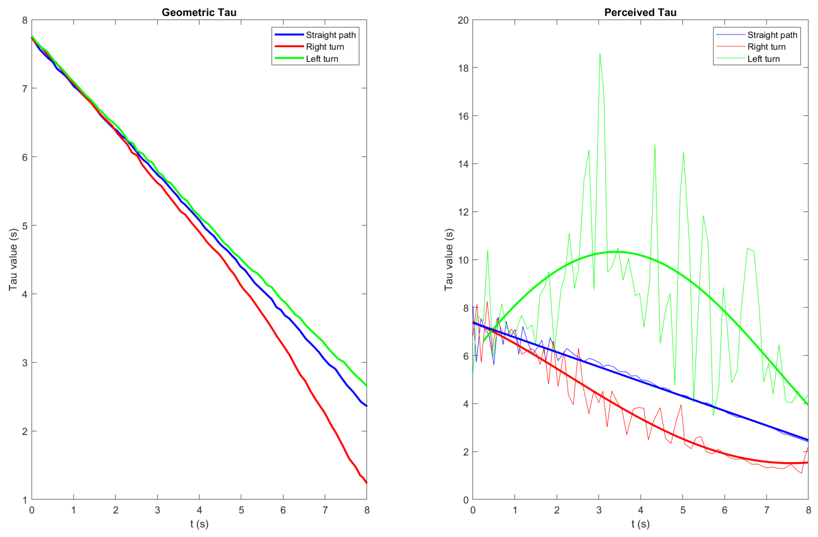


Figure 3.4: Geometric TTT values on the left and perceived time-to-transit data on the right (with interpolation functions).

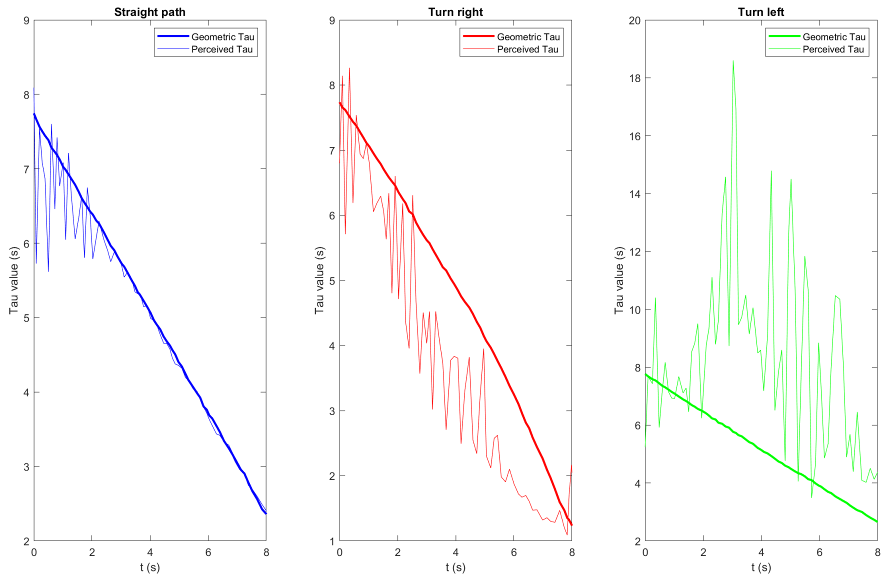


Figure 3.5: Direct comparison between geometric and perceived time-to-transit values during the three tests without the implementation of Algorithm 1.

geometric one) with the goal of guiding a robot in an unknown environment cannot be developed if no Sense-Perceive-Act cycle is introduced. The results of the tests run when the strategy summarized by Algorithm 1 was implemented are represented in Figure 3.6.

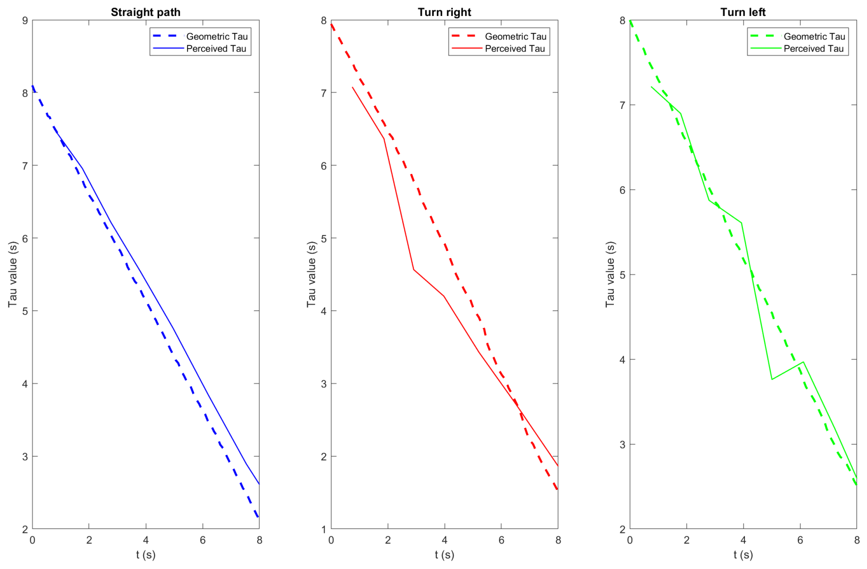


Figure 3.6: Direct comparison between geometric and perceived time-to-transit values during the three tests with the implementation of Algorithm 1. The straight path segments are the results of the algorithm's implementation of the Sense-Perceive-Act cycle.

It can be easily noted that a great improvement in the estimation of time-to-transit has been made. Experimentally, the best results are obtained when the ratio $\frac{\alpha}{\beta}$ is close to 1.5 (where α and β are the duration of the Sense and of the Act phase respectively). In this case α=0.4s and β=0.25s and the forward linear velocity v is constant and equals to 0.5m/s. Even if the two kinds of TTT are still not coincident, the usage of perceived time-to-transit data as a proxy for the geometric ones is now justified.

Chapter 4

ROS and Gazebo

In this work, two important tools have been exploited in order to achieve the final objective: ROS and Gazebo. In ROS the software components of the navigation system have been developed and, thanks to Gazebo, their performances have been tested in different customized simulation worlds. In the following section, a summary of the most important characteristics of the used tools is provided focusing on the functionalities that have been exploited in this work.

4.1 Robot Operating System

The Robot Operating System (ROS) is an open-source, not real-time, meta-operating system for writing robot software. It provides almost all the services that a general operating system can offer and also a collection of tools, libraries and conventions for obtaining, building, writing, and running code across multiple computers in order to simplify the task of creating complex and robust behavior across a wide variety of robotic platforms, as stated on the ROS official website (`https://www.ros.org/about-ros/`). One of the main advantages of ROS is represented by the fact of being a distributed framework of processes (*Nodes*) that can be designed independently and loosely coupled at runtime and this is in line with the main goal of ROS that is to support code reuse in robotics research and development (`http://wiki.ros.org/ROS/Introduction`).

The Robotic Operating System can be divided in three levels of concepts: ROS Filesystem Level, ROS Computation Graph Level and ROS Community Level.

4.1.1 ROS Filesystem Level

This level includes all the ROS resources that are saved on the hard-disk, in the list below a brief description of them is provided (for more details refer to `http://wiki.ros.org/ROS/Concepts`):

- *Packages*: they are used to organize software in ROS. A package usually contains *nodes*, ROS-dependent libraries, datasets and configuration files.
- *Metapackages*: they are a particular type of package commonly used to represent a group of related packages.
- *Package Manifests*: they contain the important data of a package, e.g name, description, dependencies and other metadata.
- *Repositories*: a repository contains a collection of packages which share the same version and can be released together.
- *Message (msg) type*: it defines the data structure of a particular message.
- *Service (srv) type*: considering a particular service, the service type defines the data structure of the request and response related to that service.

4.1.2 ROS Computation Graph Level

The Computation Graph is the ROS computation network of the peer-to-peer type that is used to exchange and process the data between the nodes that compose the graph. The concepts in the Computation Graph are explained in `http://wiki.ros.org/ROS/Concepts` and a summarized description of some of them is now presented:

- *Nodes*: they are the processes in which the data are computed, a node can be written in different programming languages but it is important to use the ROS client library in order to make it usable in the ROS framework.
- *Master*: it is the core of the software, thanks to the master the nodes are able to find each other, to exchange messages and invoke services. The ROS master provides name registration and lookup to the rest of the nodes, it can be run with the command *roscore* from the shell and, for a ROS network, the master must be unique.
- *Parameter server*: it is part of the master and it allows data to be stored by key in a central location.
- *Messages*: they are data structures with dedicated typed fields, the nodes use messages to communicate with each other.
- *Topics*: in ROS the nodes cannot exchange data directly between each other by using messages but they have to subscribe to or publish on a topic in order to receive or to give information. Topics are a sort of message bus which the processes can access and each topic has a specified name and a type defined by the message that it transports. A single node can access to multiple topics

and several processes can concur to publish or subscribe to the same topic, provided that the messages' type is the one supported by the topic. In this last case the publishers and the subscribers are independent from each other.

- *Services*: they are used when a request/response interaction is needed. A service has a name and it is defined by a pair of message structures, one for the request and one for the response.
- *Bags*: they are an important mechanism used by ROS to store data. In particular, when the tool *rosbag* is associated to a topic, it saves all the messages that pass through that topic on a *.bag* file.

4.1.3 ROS Community Level

The concepts of this level are composed by all the resources that allow software and knowledge exchange between different communities. Some examples of resources are *Repositories*, *ROS Wiki* or *Distributions*.

4.2 Customized ROS Framework

In this work, exploiting the advantages of ROS, a customized framework has been developed in order to achieve the final goal of autonomously guiding a mobile robot, equipped with a single monocular camera, in unknown environments. The most important parts of the ROS framework that have been created are *Nodes*, *Topics* and *Messages*.

The Nodes are the fundamental units in which the data are computed and then manipulated, in this work they are written in Python. The framework is composed of three main nodes that are useful to achieve the final goal. Moreover, other less important nodes have been implemented to record information from simulations, to obtain a map of the keypoints in the image and to compute the histogram of the optical flow field in different environments. The most important nodes are the *Optical flow*, the *Tau Computation* and the *Controller* nodes. An accurate description of the former two will be provided in Chapter 5 while the structure of the latter is summarized in Algorithm 1 of Chapter 3. The control laws used in the Controller are presented in Chapters 6 and 7 and the mechanism that it implements for deciding which is the best motion primitive to apply is described in Chapter 8.

Other important concepts in the ROS framework are topics and messages and, as explained in the previous section, they are necessary for the data exchange between different nodes. The messages used in this work are of the type "Image", "Twist", "TauValuesMsg" and "OpticalFlowMsg" (all described in Chapter 5). The first two are already existing types, while the latter have been created from scratch.

These messages are exchanged between nodes by using topics. In particular, the topics that have been exploited are:

- */front/image_raw* that is the topic to which the Optical Flow node subscribes to obtain the images from the monocular camera. It accepts messages of the type "Image".
- */optical_flow* that is the topic on which the Optical Flow node publishes the coordinates of the keypoints and to which the Tau Computation node subscribes to receive them. It accepts messages of the type "OpticalFlowMsg".
- */tau_values* that is the topic on which the Tau Computation node publishes the time-to-transit average values related to each image's ROIs and to which the Controller node subscribes to receive them. It accepts messages of the type "TauValuesMsg".
- */jackal_velocity_controller/cmd_vel* that is the topic on which the Controller node publishes the steering command for the Jackal robot. It accepts messages of the type "Twist".

In Figure 4.1, an outline of the ROS framework developed in this work is proposed with snapshots of the functionalities implemented in each node. In a

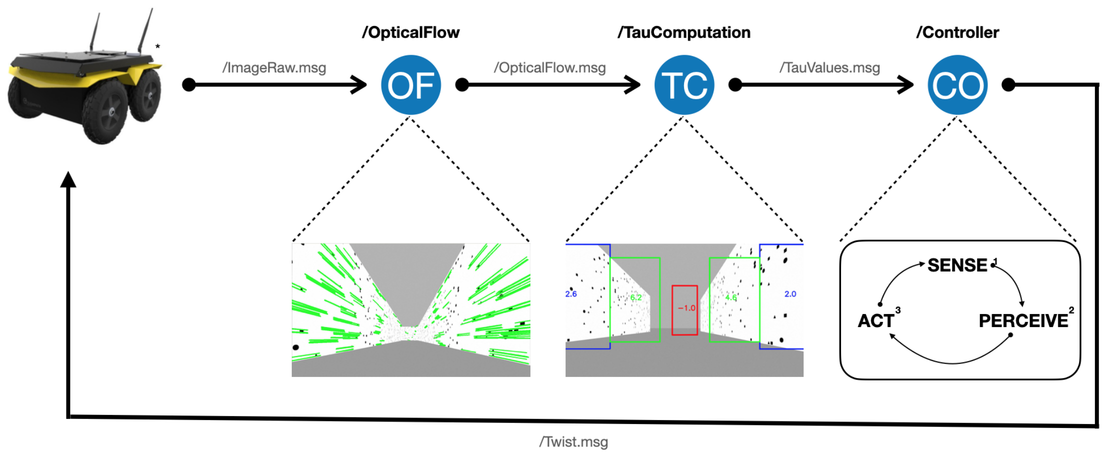


Figure 4.1: Graphical representation of the ROS framework developed in this work.

previous version the framework was a little bit different from the one presented in Figure 4.1. The operations performed in the Optical Flow and in the Tau

Computation nodes were done in a single node which had to estimate the optical flow field, to compute the time-to-transit values, to divide them in the different ROIs and then to compute the average time-to-transit values for each ROI. All these computations implemented in a single node lead to a code very difficult to be analyzed and hardly reusable. The solution of dividing the OF estimation and the TTT computation has been adopted in order to improve the reusability of the code on various robotic platforms and to make the debug process easier if errors occur.

4.3 Gazebo

Simulations are of paramount importance when new algorithms have to be tested. The creation of customized simulation environments allows us to verify the effectiveness of the software components that have been developed and to observe the behavior of the new software item without causing damages to objects or people.

Gazebo[1] is a 3D simulator widely used in the robotic field because it can be easily coupled with ROS, resulting in a powerful robot simulator. The Graphical User Interface (GUI) is intuitive and it allows the users to add, to update and to delete pre-existing or new models. Gazebo offers several options in order to make the simulation scenarios as realistic as possible. One of the main technical characteristics is the dynamic simulation of 3D robot models (performed through a high performance physics engine) that can be further improved by adding advanced 3D graphic options such as high-quality lighting, textures or shadows. Moreover, it is possible to simulate a large number of sensors (e.g. camera, GPS, radar, LIDAR) on an equally wide number of robot models.

In this work, the Jackal robot model equipped with a PointGrey Flea3 monocular camera has been used. The Unified Robot Description Format (URDF) file is provided by Clearpath Robotics. This *.xml* file is necessary to describe the robot model and its mechanical specifications in order to correctly simulate the Jackal UGV in Gazebo. Then, different simulation environments, with various difficulties and geometries, have been created to finally test the performances of the implemented controller.

4.3.1 Creation of virtual simulation environments

Gazebo offers the possibility of using already existing simulation environments but it also allows the creation of new scenarios from scratch by adding objects to build the desired world. These objects are defined by models that can be already

[1]Further information about the Gazebo simulation environment can be found at `http://gazebosim.org/`

present in the Gazebo models' database or they have to be created by using the *Model editor*. In particular, in this work, two different types of environments can be identified: realistic environments and artificial environments.

The artificial environments are simulation scenarios in which the walls have a customized texture with a specific feature density fixed a priori. In Figure 4.2, some examples of these textures which consist of Bernoulli distributions of a set of shapes such as triangles, circles and squares are presented. The artificial environments have

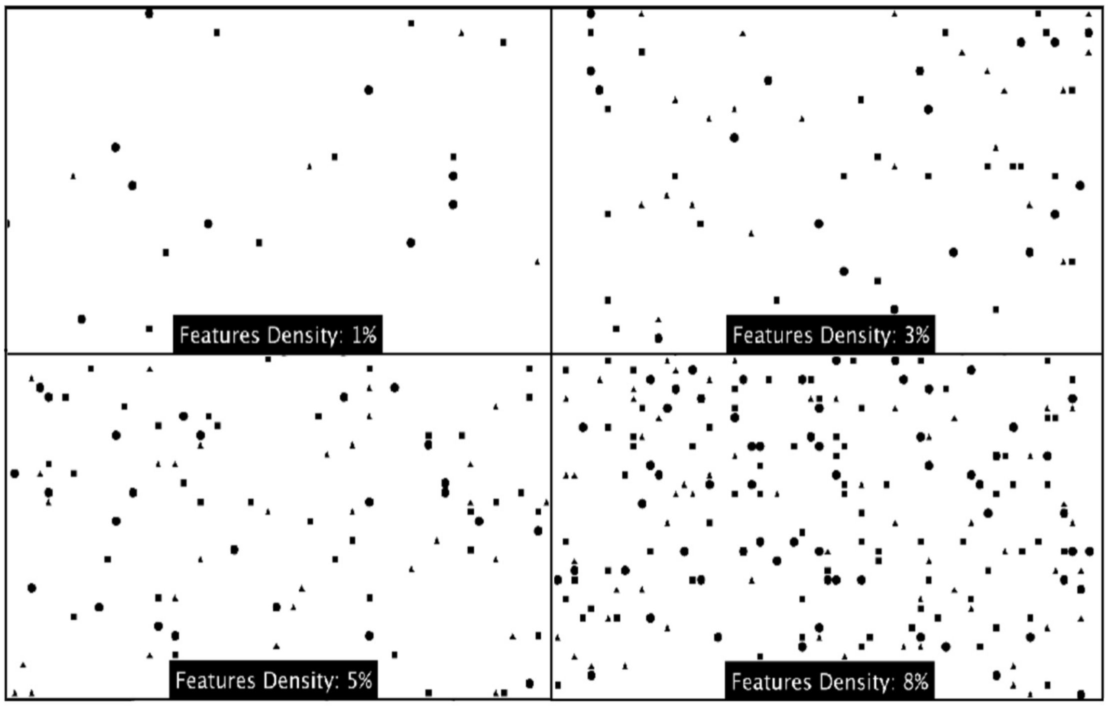


Figure 4.2: Bernoulli distributions used as a texture for the walls in artificial simulation environments.

been used in the first part of the simulation tests because they allow us to verify the effectiveness of the software components in scenarios with a precise feature density and so to test which are the situations in which the number of features negatively affects the algorithm's behavior. In Appendix A, a deeper analysis of the effect of feature density on the different control laws is discussed.

However, the robot has to be able to navigate in the real world in which it is not possible to define a fixed feature density because it can vary depending on the scenario that the vehicle has to face. This is the reason that leads to the creation of more realistic environments on Gazebo. These types of simulation scenarios have been created through the usage of already existing models, e.g. trees, cars and houses. Then, the performance of the implemented algorithm running on the

Jackal robot has been observed. The results of these tests and some considerations are presented in Chapter 8.

An example of an artificial and a real simulation environment is shown in Figure 4.3, more scenarios will be presented in Chapter 8.

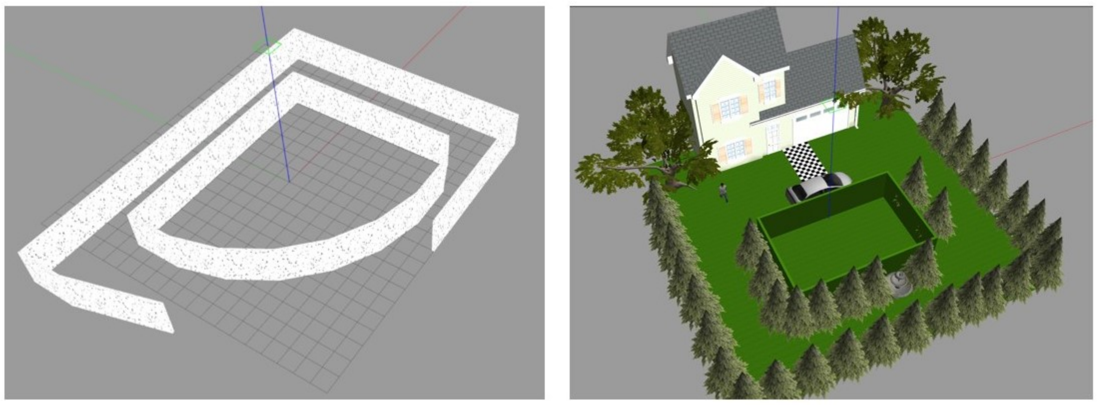


Figure 4.3: On the left an artificial environment while on the right a realistic environment. Both are implemented using the Gazebo simulation environment.

Chapter 5

Tau subdivision in Regions of Interest

In Chapter 4 an introduction of the ROS framework developed in this work is proposed and in the following sections a deeper analysis of the first two nodes implemented is provided. The Optical Flow node is the one devoted to the estimation of the optical flow field while the Tau Computation node receives the pixel coordinates of the keypoints and their velocities and it computes the associated time-to-transit values. Then, after some filtering process, the visual cues are divided into five regions of interest (ROIs) to obtain a better understanding of the environment.

5.1 The Optical Flow Node

The Optical Flow node is responsible for the optical flow field estimation. It acquires a sequence of images from the camera mounted on the Jackal robot and it extracts the relevant features to finally compute the optical flow vectors. The node subscribes to the topic **/camera_used/image_raw** to get a camera frame and, thanks to the *CVBridge* library, the ROS image is converted into a format manageable by *OpenCV*. Then, on the gray-scale image, the keypoints have to be found by using an ORB detector and descriptor. An important parameter to be provided to the ORB algorithm is the maximum number of features to retain (*maxFeatures*) and this has to be set properly. A first attempt was made with 600 features but, as can be noticed in the top image of Figure 5.2, all the points are located in the peripheral parts of the image because they have a better definition and so their edges are sharper.

However, for the scope of this work, it is important to have some information also from the center area. In order to reach this objective, the original image is divided into three equal parts and on each sub-image ORB is applied with different values of *maxFeatures (b,c)*. The division is shown in Figure 5.1.

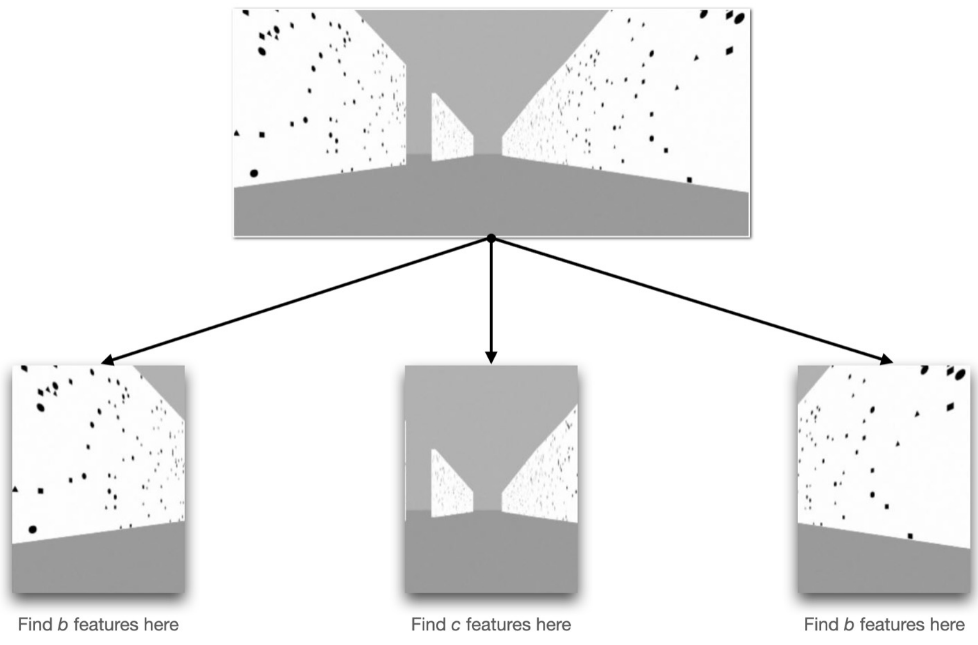


Figure 5.1: The image from the camera is split in three sub-images, where $b = 250$ and $c = 150$.

The results, presented in Figure 5.2, reveal that the implemented strategy successfully increases the presence of feature points in the center part of the image. This approach tries to mimic the behavior of human eyes which, differently from cameras, have an higher density of photoreceptors in the *fovea centralis*, characteristic that leads to a great central vision.

The ORB detector and descriptor finds all the points of interest in the image, but then it is important to compare two subsequent frame in order to find the same points and draw the optical flow vectors. This operation is performed by the pyramidal Lucas-Kanade method through the OpenCV function **cv2.calcOpticalFlowPyrLK** which receives as inputs the previous frame with the related keypoints, the current frame and some parameters useful to set limits in the pyramidal research[1] such as the number of levels in the pyramid, the termination criteria of the iterative search algorithm and the quality level below which a feature point cannot be considered. It is crucial to choose the right values for these

[1]More information about the pyramidal Lucas-Kanade method can be retrieved in Chapter 2

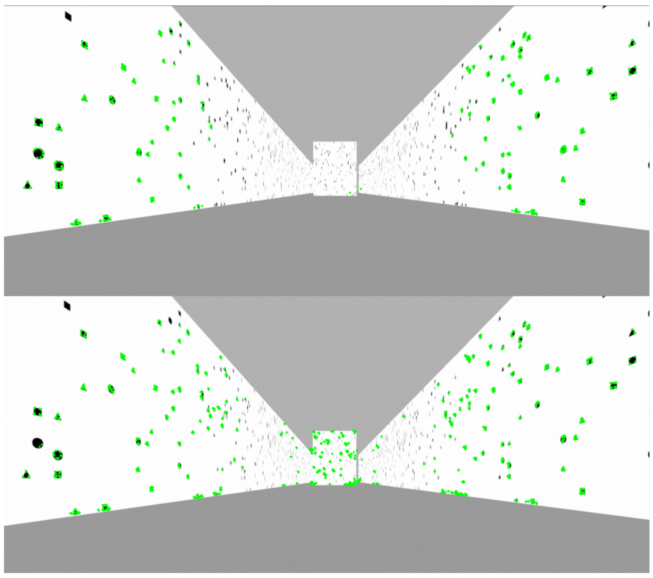


Figure 5.2: Comparison between the features found before (on top) and after (on the bottom) dividing the original image in sub-images.

parameters in order to reach an acceptable trade-off between speed and accuracy. The function **cv2.calcOpticalFlowPyrLK** returns the position of the keypoints, provided as input, in the current frame along with an attribute called *status* that is 1 if the matching of a keypoint went well or *status=0* otherwise. When all the good points (*status=1*) are selected, the same points in the previous keypoints' array have to be considered in order to draw the flow vectors and to obtain the optical flow field, as shown in Figure 5.3.

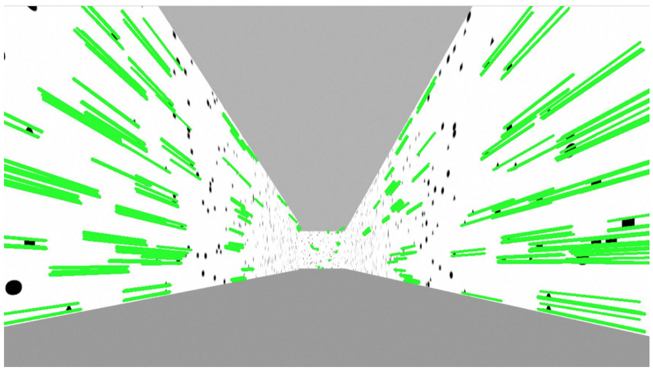
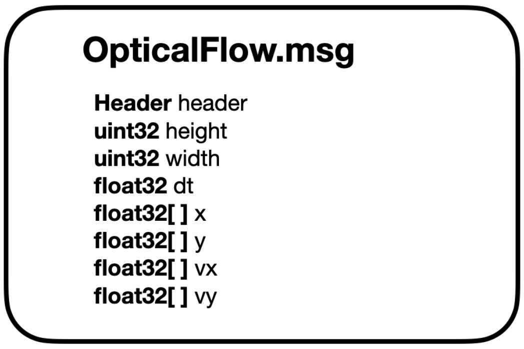


Figure 5.3: Optical Flow Field.

Finally, the previous frame and the previous keypoints' array are updated with the current frame and the keypoints' array found by applying the ORB algorithm.

In order to pass the data collected in this node to the Tau Computation node, a ROS message, called */OpticalFlowMsg.msg*, has been created with the structure presented in Figure 5.4 where height and width are the frame's dimensions, dt is the time interval between the previous and the current frame, (x, y) are the good points' coordinates and (v_x, v_y) are their velocity components along the x and y directions.



```
OpticalFlow.msg

Header header
uint32 height
uint32 width
float32 dt
float32[ ] x
float32[ ] y
float32[ ] vx
float32[ ] vy
```

Figure 5.4: Structure of the */OpticalFlowMsg.msg.*

5.2 The Tau Computation Node

The goal of the Tau Computation node is to analyze the array of keypoints with their velocities packed in the Optical Flow message, to compute time-to-transit values and to create the input signals to be provided to the controller.

The first operation that has to be done to achieve this goal is to transform the data coming from the Optical Flow node in order to make them coherent with the reference frame used to compute time-to-transit values which is fixed at the center of the image as shown in Figure 5.5. In this position it should be located at the Focus of Expansion (FOE), the point from where the optical flow field expands. This is in principle not true, but the estimation of the FOE is generally a computational-expensive operation, very sensitive to noise and, once its location is known, no significant improvements are produced in the computation of the TTT.

The standard coordinate systems used in OpenCV place the origin of coordinates in the upper lefthand corner of the image rectangle. If a forward-looking camera is going to be used to generate a steering signal based on balancing Optical Flow on left and right halves of the image, the most natural coordinate system is centered at the point where the optical axis intersects the image plain. For rotation-free motions aligned with the direction of the camera axis, this origin of coordinates will be at the focus of expansion (FOE). We choose our coordinates in this way so as to have a natural balance between flow in the left and right halves of the image.

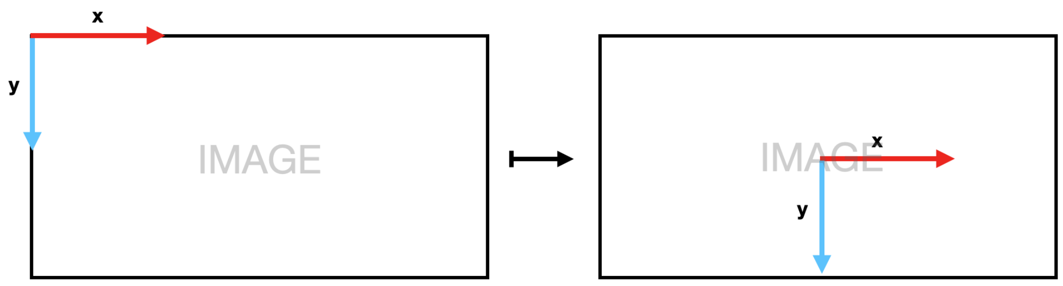


Figure 5.5: On the left, the reference frame used in the Optical Flow node. On the right, the one utilized to compute the time-to-transit values in the Tau Computation node.

Once all the data are expressed in the right reference frame, time-to-transit is computed as follows:

$$|\tau_i| = \frac{\sqrt{x[i]^2 + y[i]^2}}{\sqrt{v_x[i]^2 + v_y[i]^2}}$$

where x, y, v_x and v_y are the arrays packed in the Optical Flow message. The next operation to be performed is crucial for the development of this work. The fixed-size set of inputs that are provided to the controller must be defined and it should be:

- as small as possible
- rich enough to correctly represent the environment

These two characteristics are in conflict and a trade-off must be found. In order to take a decision, the distribution of the tau values in the image space and of the optical flow field have been analyzed in different environments thanks to a ROS node specifically built for this scope. In Figure 5.6 a graphical representation of the distribution of TTT values in the image is shown.

Since the robot should be able to recognize straight corridors, corners and the presence of an obstacle (also in the direction of motion), five inputs, representing the average tau in five regions of the image, can be enough. A detailed description of how these inputs can be used to understand the general geometry of the environment in which the robot is moving is provided in Chapter 8. In Figure 5.7, the division of the image in ROIs used in this work is presented. Clearly, the image can be divided in many different ways and also the number of regions can be changed. More

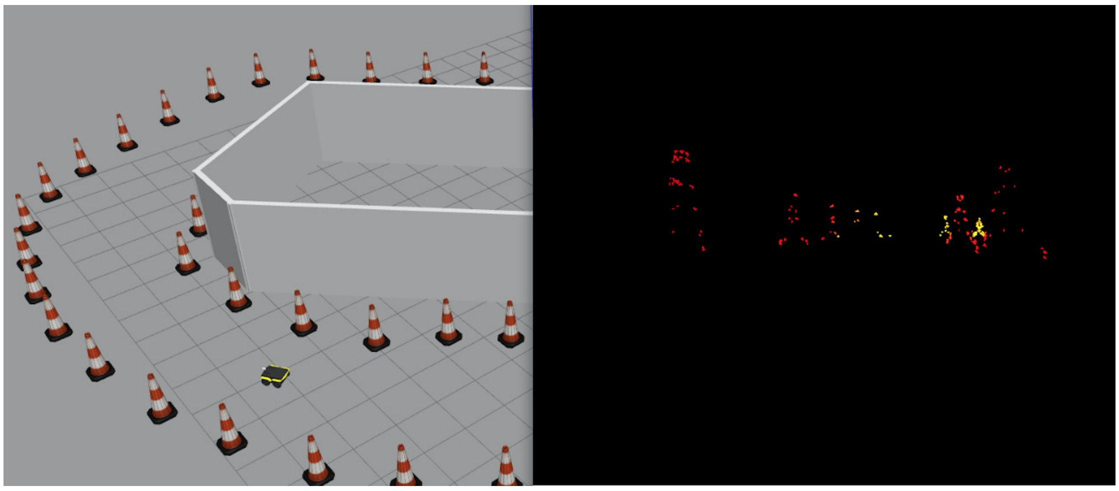


Figure 5.6: On the left, an artificial environment created in Gazebo. On the right what a moving robot perceives. Points, which represent the features, are colored (from red to white) according to the magnitude of their associated TTT that can be used as a proxy for distance.

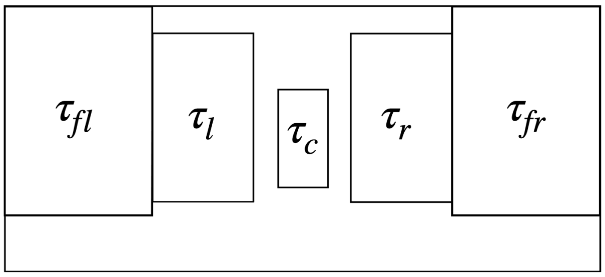


Figure 5.7: The division of the image in five ROIs.

ROIs means additional complexity that has to be motivated by an improvement of performance.

In order to obtain the five time-to-transit values, the average TTT starting from all the values belonging to each region must be computed. A filtering operation then is mandatory for many reasons. First of all, at the beginning of the sense phase some residual rotational components can still affect the computation of TTT. Moreover, glitches and noise are always present during the entire sensing. In this work a simple but efficient filtering strategy is employed which consists in sorting the array of time-to-transit values coming from a region and then discarding its smallest and highest values as shown in the Code Listing 5.1 (Python language). TTT values coming from a single region should be similar to each other. Since from simulations it seems that at the beginning of the sense phase taus normally assume values that are quite different from the ones computed at the end of the sensing, the objective of future work could be to set to an higher value the percentage of discarded values during the first part of the sense phase. Moreover, noise and wrong matches are other important sources of errors that persist and lead to time-to-transit values very different from the average so even at the end of the sensing phase the array of TTT values cannot be considered totally reliable.

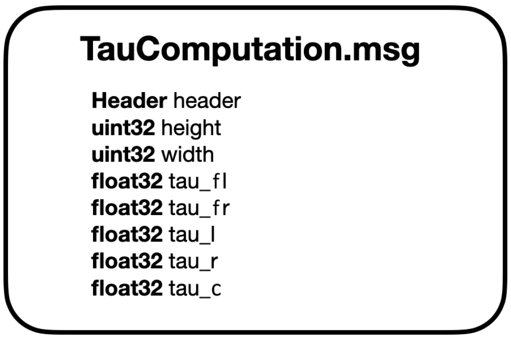
```python
import numpy as np

def tau_filtering(array, percentage):
    dim = np.size(array)
    jump = int(percentage * dim)
    array = np.sort(array)
    array = np.delete(array, range(jump))
    array = np.delete(array, range(dim - jump, dim))
    return array
```

Listing 5.1: Filtering function in the Tau Computation ROS node.

When the filtering process comes to an end, for every region that has a residual number of time-to-transit values higher than a predefined threshold, the average τ is computed. If the quantity of the remaining TTT data is not high enough, a -1 is attributed to the specific ROI.

A ROS message, called */TauValues.msg*, is responsible for transmitting the five inputs to the controller. The message is composed by an header, the height and the width of the image used to collect the visual cues and the five τ inputs coming from the different ROIs (depicted in Figure 5.7) as it is summarized in Figure 5.8.



TauComputation.msg

Header header
uint32 height
uint32 width
float32 tau_fl
float32 tau_fr
float32 tau_l
float32 tau_r
float32 tau_c

Figure 5.8: Structure of the */TauValues.msg*.

A visual representation of the TTT data coming from each Region of Interest of the image is represented in Figure 5.9.

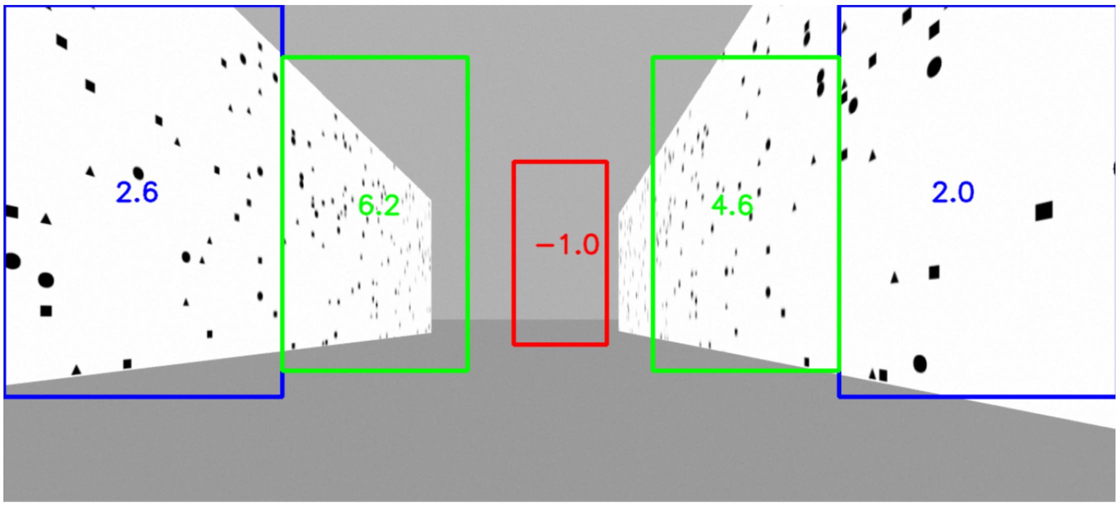


Figure 5.9: Visual representation of the inputs coming from the different ROIs of the image.

Chapter 6

Tau Balancing control law

When flying through a narrow passage, bees position themselves in the center of it in order to experience the same image velocity in both eyes. This characteristic of bee's flight has been highlighted by Srinivasan *et al.* in [1] and it represents the idea behind the *Tau Balancing* control law discussed in this Chapter. Time-to-transit is a concept that extends *time-to-contact* which is an idea that has been around the perceptual psychology literature for a while. The idea of using it as a steering signal is new and has been introduced in [7] and further explained in [8]. The goal of *Tau Balancing* is to asymptotically guide a vehicle onto the center line between two corridor walls. In this work, a slightly different version of this control law is presented and used.

6.1 Stability Analysis

The simple kinematic model of planar motion introduced in Chapter 3 is considered:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} vcos\theta \\ vsin\theta \\ u \end{bmatrix}$$

where u is the turning rate, v is the constant forward speed in the direction of motion ($v = 1$) and θ is the heading of the vehicle. Referring to Figure 6.1, the geometric time-to-transit can be written as:

$$\tau(t) = \frac{cos\theta(t)(x_r - x(t)) + sin\theta(t)(y_r - y(t))}{v}$$

In this section the steps presented in [8] to prove the effectiveness of τ as a steering signal are summarized. An idealized visual model based on two photoreceptors one on each side of the center of focus will be considered. Values of $\tau(t)$ can be determined since instantaneous detection of perfect features at both $d(t) = \pm 1$ and their corresponding derivatives are available. A global reference frame located in the center of an infinitely long corridor with the x-axis being perpendicular and pointing to the right wall will be considered. Every point on the walls is a clear and perfectly detectable feature.

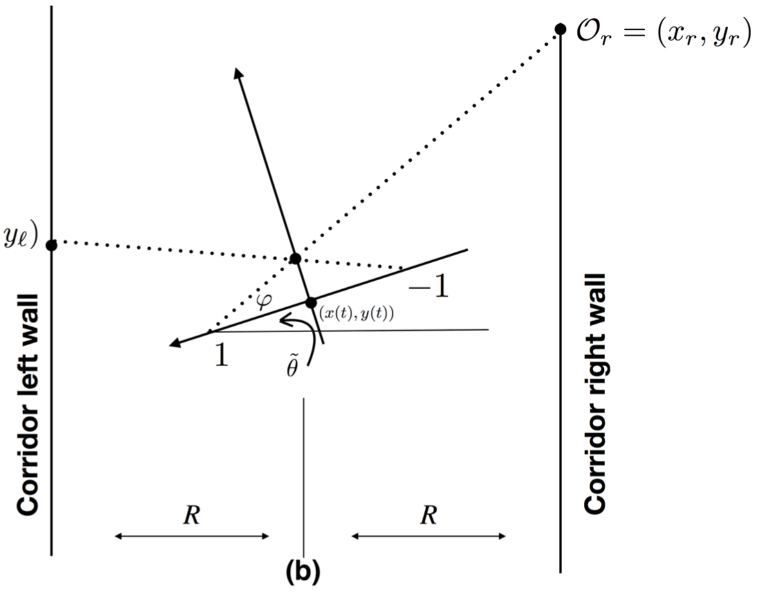


Figure 6.1: A moving vehicle in an infinitely long corridor. $O_l(x_l, y_l)$ and $O_r(x_r, y_r)$ are the two features detected on the walls and $tan(\varphi) = f$ which is the pinhole camera focal length and $\theta = \tilde{\theta} + \frac{\pi}{2}$. *Source: [8].*

The *Theorem 1* in [8] states that for any gain $k > 0$ there is an open neighborhood U of $(x, \theta) = (0, \frac{\pi}{2})$, $U \subset \{(x, \theta) : -R < x < R; \varphi < \theta < \pi - \varphi\}$ such that for all initial conditions (x_0, y_0, θ_0) with $(x_0, \theta_0) \in U$, the *Tau Balancing* control law:

$$u(t) = k(\tau_l - \tau_r) \tag{6.1}$$

asymptotically guides the vehicle, with the simple kinematic model of planar motion (3.1), onto the center line between two walls.

Tedious calculations allow to compute the position in the global reference frame of the detected left and the right feature as:

$$O_l = \begin{pmatrix} -R \\ y + fsin(\theta) + \frac{(R+x+fcos(\theta))(cos(\theta)+fsin(\theta))}{sin(\theta)-fcos(\theta)} \end{pmatrix}$$

$$O_r = \begin{pmatrix} R \\ y + fsin(\theta) + \frac{(R-x-fcos(\theta))(fsin(\theta)-cos(\theta))}{fcos(\theta)+sin(\theta)} \end{pmatrix}$$

and from these coordinates, the time-to-transit values related to the left and right feature can be retrieved (considering $v = 1$):

$$\tau_l = cos(\theta)(-R - x) + sin(\theta)\left(fsin(\theta) + \frac{(fsin(\theta) + cos(\theta))(fcos(\theta) + R + x)}{sin(\theta) - fcos(\theta)}\right)$$

$$\tau_r = cos(\theta)(R - x) + sin(\theta)\left(fsin(\theta) + \frac{(fsin(\theta) - cos(\theta))(-fcos(\theta) + R - x)}{sin(\theta) + fcos(\theta)}\right)$$

With some additional computations u can be rewritten as:

$$u = k(\tau_l - \tau_r) = -\frac{2fk(fcos(\theta)(sin(\theta) + R) + xsin(\theta))}{f^2cos^2(\theta) - sin^2(\theta)} \tag{6.2}$$

From (6.2) it can be noticed that the following subsystem can be isolated:

$$\begin{bmatrix}\dot{x}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos\theta\\ k(\tau_l - \tau_r)\end{bmatrix} \tag{6.3}$$

The system can be linearized about $(x, \theta) = (0, \frac{\pi}{2})$, which is also the only rest point in the domain $\{(x, \theta) : -R < x < R; \varphi < \theta < \pi - \varphi\}$, obtaining:

$$\begin{bmatrix}\delta\dot{x}\\ \delta\dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & -1\\ 2fk & -2(kf^2 + kRf^2)\end{bmatrix}\begin{bmatrix}\delta x\\ \delta\theta\end{bmatrix}$$

The eigenvalues of the matrix representing the controlled system are:

$$\lambda_{1,2} = -f^2k(1 + R) \pm \sqrt{fk[f^3k(1 + R)^2 - 2]}$$

Since they are always in the left plane, the system is asymptotically stable. Moreover, if $k > \frac{2}{f^3(1+R)^2}$, the eingenvalues are real and negative meaning the vehicle will not undergo oscillations aligning itself onto the center line. The requirement that the initial condition must lie in the interval $\varphi < \theta_0 < \pi - \varphi$ is necessary since otherwise the right or the left feature will not be detected anymore and this problem is illustrated in Figure 6.2.

Since in this work the time-to-transit values used as input for the controller are five (as presented in Chapter 5) and four of them can be used to align the robot onto the center line of a corridor, a slightly different version of the *Tau Balancing* control law can be written:

$$u(t) = k_f(\tau_{fl} - \tau_{fr}) + k_m(\tau_l - \tau_r) \tag{6.4}$$

where τ_{fl}, τ_{fr}, τ_r and τ_l are the TTT values of the different ROIs of the image (the central region of interest is not included since it is not useful to balance the robot's heading). The subsystem in (6.3) becomes then:

$$\begin{bmatrix}\dot{x}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos\theta\\ k_f(\tau_{fl} - \tau_{fr}) + k_m(\tau_l - \tau_r)\end{bmatrix} \tag{6.5}$$

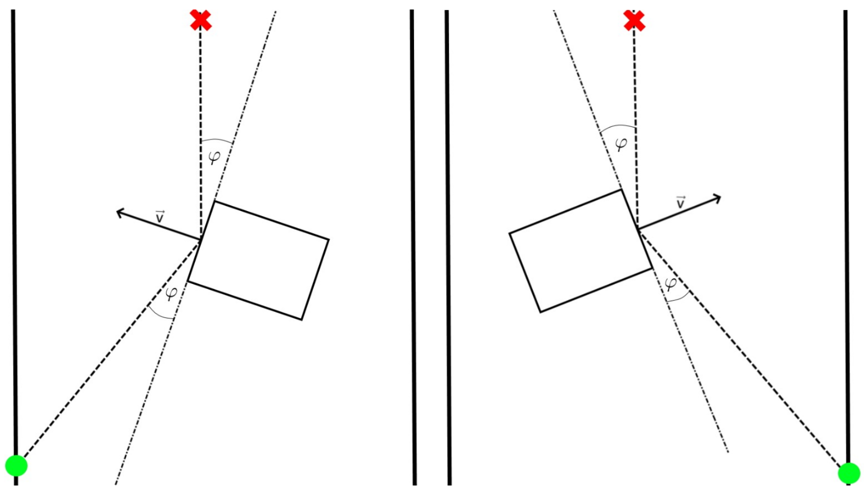


Figure 6.2: The position of the robot when $\theta_0 = \pi - \varphi$ on the left and when $\theta_0 = \varphi$ on the right. These angles are known as *critical angles* since one of the two features cannot be detected anymore.

A modified version of *Theorem 1* in [8] can be written since in this case for any gain $k_f > 0$ and $k_m > 0$ there is an open neighborhood U' of $(x, \theta) = (0, \frac{\pi}{2})$, $U' \subset \{(x, \theta) : -R < x < R; \varphi_2 < \theta < \pi - \varphi_2\}$ such that for all initial conditions (x_0, y_0, θ_0) with $(x_0, \theta_0) \in U'$, the *Tau Balancing* in (6.4) aligns the vehicle to the center of the corridor. The angle φ_2 is the one presented in Figure 6.3. It is clear

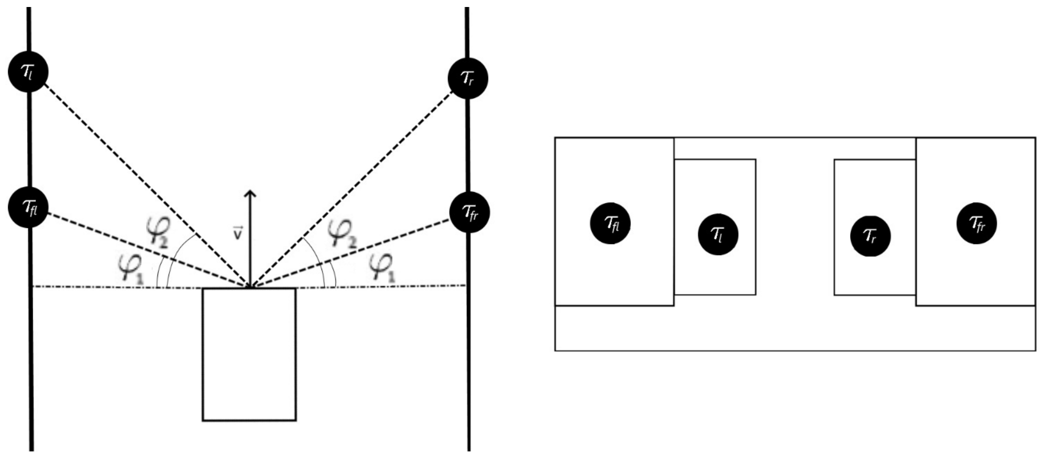


Figure 6.3: The *Tau Balancing* control law presented in (6.4) allows to take as input the time-to-transit values coming from four different ROIs.

that the open neighborhood U' has the same structure of U but since $\varphi_2 > \varphi_1 = \varphi$ (and in particular in this work $\varphi_2 \approx 2 \cdot \varphi_1$), $U' \subset U$ meaning that this new version of the control law allows to use more information coming from the environment (4 input are used instead of 2) but the initial condition (x_0, θ_0) must lie in a more restricted area. When this is not true, the *Tau Balancing* control law as presented in [8] is used considering $\varphi = \varphi_1$.

It is possible to linearize the subsystem in (6.5) about the rest point $(x, \theta) = (0, \frac{\pi}{2})$ obtaining:

$$\begin{bmatrix} \delta\dot{x} \\ \delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 2(f_f k_f + f_m k_m) & -2(k_f f_f^2 + k_m f_m^2 + k_f R f_f^2 + k_m R f_m^2) \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\theta \end{bmatrix}$$

with $f_f = tan(\varphi_1)$ and $f_m = tan(\varphi_2)$. In this case it is like using two pinhole cameras with focal length f_f and f_m. The eigenvalues of the matrix representing the controlled system are:

$$\lambda_{1,2} = -(f_f^2 k_f + f_m^2 k_f)(1+R) \pm \sqrt{(f_f k_f + f_m k_m)[(f_f^3 k_f + f_m^3 k_m)(1+R)^2 - 2]}$$

It can be noticed that they are always in the left half plane, proving the coefficient matrix is Hurvitz. Moreover, if the condition $(f_f^3 k_f + f_m^3 k_m) > \frac{2}{(1+R)^2}$ is met, the eigenvalues are real and negative meaning that the robot will not experience oscillations in aligning onto the center line.

6.2 Eulerian Simulation Results

In this section some Matlab simulations showing the behavior of the *Tau Balancing* control law with Eulerian sensing are presented and discussed. Eulerian sensing, as introduced in Chapter 3, means that each photoreceptor in the camera (or in the eye) can instantaneously detect in the image the feature points and their velocities while Lagrangian sensing is the form of visual sensing on which standard computer vision techniques are based and it consists in tracking the keypoints as they move in the retina.

In Figure 6.4, the motion primitive is used in its original form (6.1) and the vehicle undergoes oscillations reaching the center line since the condition $k > \frac{2}{f^3(1+R)^2}$ is not respected. In Figure 6.5 the results of a simulation when $k > \frac{2}{f^3(1+R)^2}$ allow to see the effect of the real and negative eigenvalues of the controlled system's matrix. Moreover, it is possible to notice that, even starting quite far from the rest point $(x, \theta) = (0, \frac{\pi}{2})$, the control law can align the vehicle onto the center line of the corridor showing good robustness. A comparison between the behavior of the *Tau Balancing* presented by Baillieul in [8] and its slightly modified version (6.4) discussed in this work is made in Figure 6.6. It can be noticed that from the theoretical point of view, no better performances are achieved but the advantage brought by this modified version of *Tau Balancing* comes into play when dealing

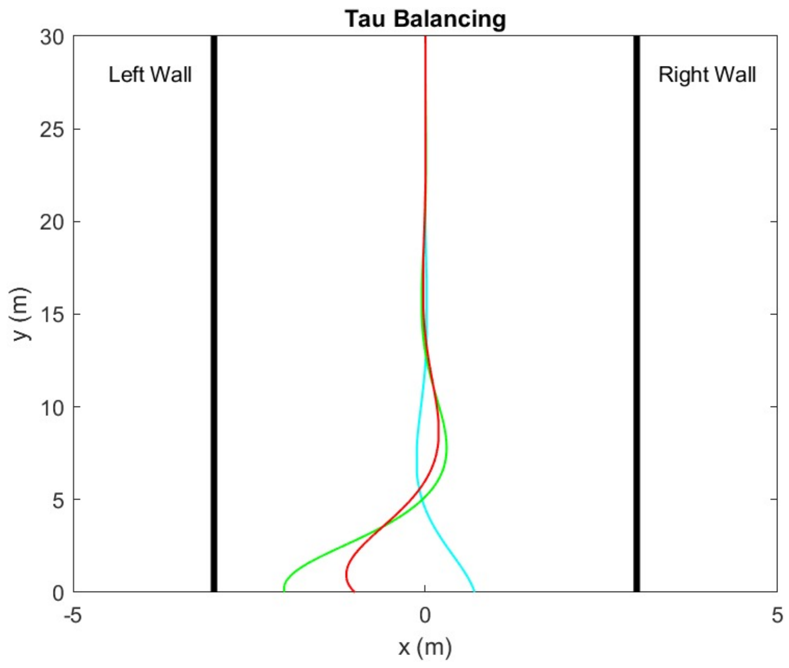


Figure 6.4: Matlab simulation showing the behavior of the *Tau Balancing* control law in (6.4) when $k < \frac{2}{f^3(1+R)^2}$.

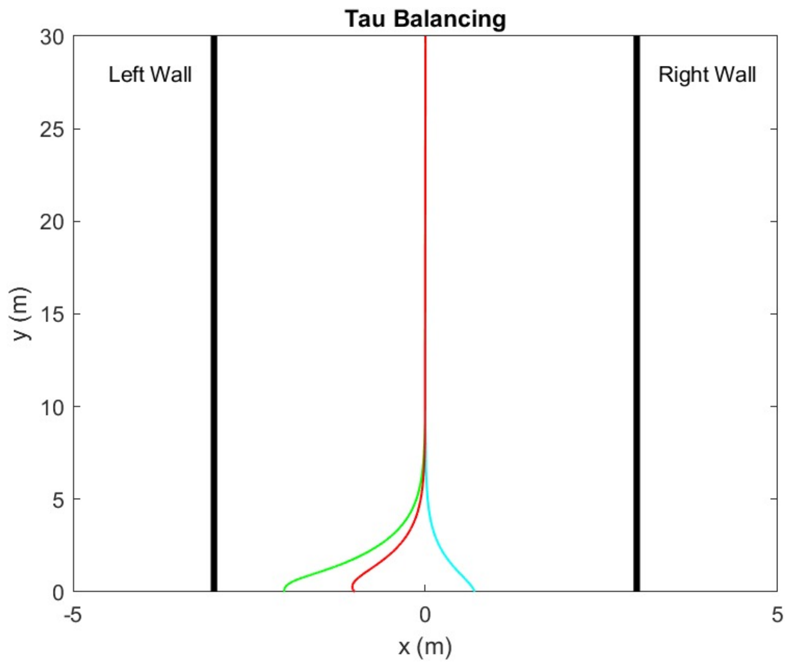


Figure 6.5: Matlab simulation showing the behavior of the *Tau Balancing* control law in (6.4) when $k > \frac{2}{f^3(1+R)^2}$.

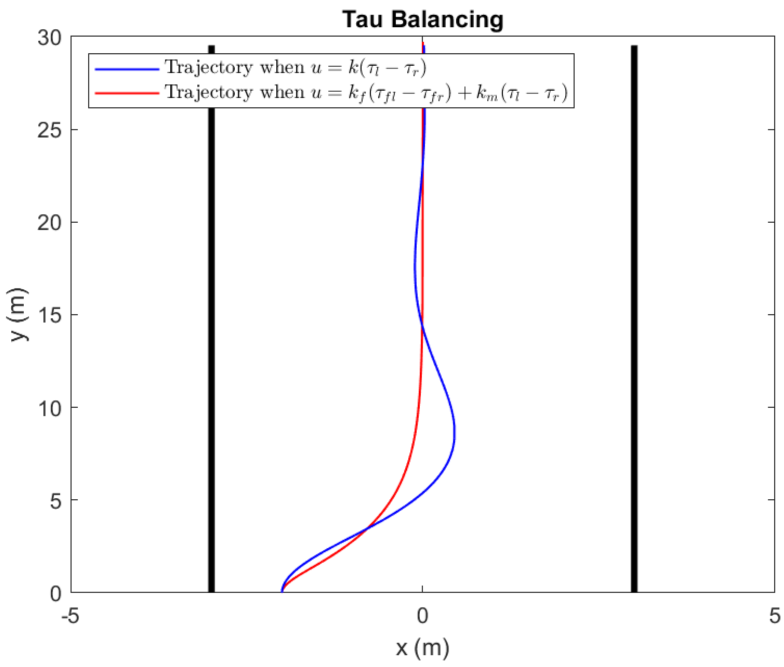


Figure 6.6: Matlab simulation showing that the vehicle undergoes oscillations when using $u = k(\tau_l - \tau_r)$ with $k = 0.14$ while a smoother behavior is obtained when $u = k_f(\tau_{fl} - \tau_{fr}) + k_m(\tau_l - \tau_r)$ with $k_f < k$ and $k_m < k$ (in particular $k_f = k_m = 0.12$).

with its implementation on a real platform and when Lagrangian sensing is used. In order to avoid oscillations, the condition $(f_f^3 k_f + f_m^3 k_m) > \frac{2}{(1+R)^2}$ must be met instead of the one presented in [8] which can be rewritten (considering $f = f_f$) as $f_f^3 k > \frac{2}{(1+R)^2}$. This means that $|k_f|$ and $|k_m|$ can satisfy the condition and be smaller than $|k|$ so it is possible to give less weight to each ROI to achieve the same goal.

This can be important since the time-to-transit values coming from the different areas of the image are always affected by an error. Having the possibility to rely on four ROIs instead of two can lead to better performances and to increased robustness.

Chapter 7

Single Wall control law

The *Tau Balancing* control law presented in Chapter 6 is a simple and useful steering law, but it requires the presence of feature points in both right and left image sides. This is a weakness in environments which present sparsity of features. In order to cope with this problem, a new control law has been introduced called *Single Wall* strategy that exploits visual cues coming only from the left or the right side of the environment.

In the following sections the control law is presented and theoretical considerations about its stability are made in the case in which the *Sense-Perceive-Act* cycle is not applied, then Eulerian simulations are performed to show the effectiveness of this motion primitive. Finally, some considerations on the effects of introducing the Sense-Perceive-Act cycle on the stability of the controlled system are discussed.

7.1 The problem of distance maintenance using Optical Flow

When a mobile robot equipped with a monocular camera moves in an unknown environment, it has to face the situation in which the feature density on the right or on the left side of the image it acquires is not sufficiently high to produce an acceptable estimate of the TTT. In this case the Tau Balancing control law cannot be used to move in the environment. Considering Figure 7.1, the problem of navigating using visual cues coming only from one side of the environment can be managed by aligning the vehicle to the line that connects features O_1 and O_2.

A possible solution has been proposed by Kong *et al.* in [5] observing the behavior of a species of bats, the *Myotis Velifer*, in their natural habitat near Johnson City in Texas. The trajectories of these animals follow the edge of a wooded area thanks to the maximization of the difference in time-to-transit of features along the boundaries. This is the idea behind the formulation of the

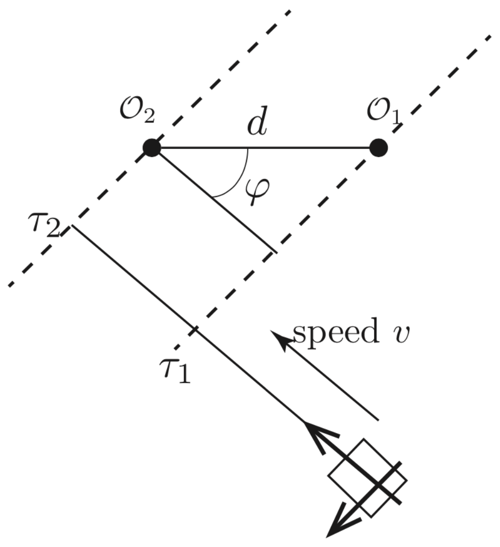


Figure 7.1: A moving vehicle has to align its line of travel to the line that intersects features O_1 and O_2. *Source: [5].*

following motion primitive:

$$u(t) = k[\tau_2'(\theta(t)) - \tau_1'(\theta(t))] \tag{7.1}$$

introduced in *Theorem 4.1* in [5] which states that for any $k > 0$, the steering control law, where $\tau_j'(\theta) = \frac{\partial \tau_j}{\partial \theta}$, asymptotically aligns the vehicle with the semi-infinite line directed from O_1 to O_2 provided that the initial orientation θ_0 is such that $\tau_2 > \tau_1$ and $v = 1$.

In the case in which the vehicle has the possibility to retrieve its orientation in space and the coordinates (x_1, y_1) and (x_2, y_2) of the features in a world-frame, the control law in (7.1) could be written as:

$$u(t) = k[-sin(\theta)(x_2 - x_1) + cos(\theta)(y_2 - y_1)] \tag{7.2}$$

Since the goal of this work is to use only visual cues coming from a monocular camera and there is no possibility to access the global configuration information, the control law in (7.2) cannot be exploited. On the contrary, the motion primitive in (7.1), also known as *Tau Difference Maximizing*, could be of interest. However, taking into account the fact that time-to-transit estimation is already in itself quite susceptible to noise, the computation of its derivative is generally quite poor and this makes it difficult to obtain the desired control action when the steering control law is used on a real platform. An alternative approach to align a mobile robot to the line that connects O_1 to O_2 using only data coming from Optical Flow is presented in the following section.

7.2 Stability Analysis

Starting from the assumptions made in Section 6.1 about the kinematic model of the unicycle and the definition of the geometric time-to-transit, also for the Single Wall strategy it is possible to follow the reasoning presented in [8], but with an important difference that deals with the sparsity of the features. In [8] it is considered an idealized visual model based on two photoreceptors and the $\tau(t)$ values are computed by considering an instantaneous detection of features at $d(t) = \pm 1$. In situations in which sparsity of features occurs, the visual cues coming from one of the two sides (with respect to the focus of expansion) are not available and it is important to continue the navigation with only one feature detected at $d(t) = 1$ or $d(t) = -1$ and its derivative.

The situation considered is shown in Figure 7.2, it represents a vehicle which is moving forward with constant velocity $v = 1$ and, on the right, an infinitely long wall in which each point is a perfectly detectable feature. The global reference frame is located at a distance R from the wall with the x-axis pointing on the right and perpendicular to the wall. The coordinates of the detected feature point O_r are expressed in the global reference frame and they are:

$$O_r = \begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} R \\ y + fsin(\theta) + \frac{(R-x-fcos(\theta))(fsin(\theta)-cos(\theta))}{fcos(\theta)+sin(\theta)} \end{pmatrix}$$

By substituting the coordinates' values in the definition of the geometric time-to-transit, the τ_r expression becomes:

$$\tau_r = cos(\theta)(R-x) + sin(\theta)\left(fsin(\theta) + \frac{(fsin(\theta)-cos(\theta))(-fcos(\theta)+R-x)}{sin(\theta)+fcos(\theta)}\right)$$

The Single Wall control law can then be written as:

$$u(t) = -k(\tau_r - c) \tag{7.3}$$

The objective is to reset the difference between τ_r and c, if τ_r is bigger than c the control law has to provide a steering action that leads to reduce the τ_r and so the difference $\tau_r - c$. If $\tau_r > c$ the vehicle has to turn right, but since the turning rate is considered positive counter clock-wise, in order to provide the correct control action a negative $u(t)$ is necessary. Moreover, if $\tau_r < c$ the difference becomes negative and the steering action provided is positive and turns the robot on the left.

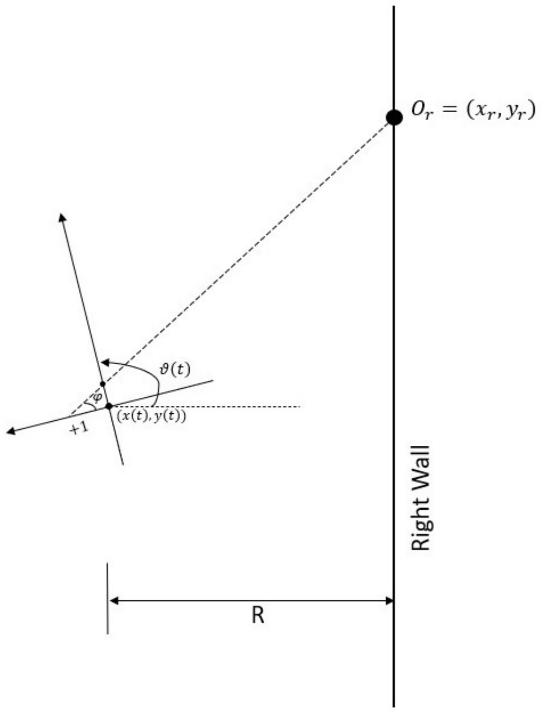


Figure 7.2: A moving vehicle and an infinitely long wall on the right. $O_r(x_r, y_r)$ is the feature detected on the wall and $tan(\varphi) = f$ which is the pinhole camera focal length.

Considering (7.3), by substituting the τ_r equation, the control law can be written as:

$$u = -k\left(\left(cos\theta(R-x) + sin\theta\left(fsin\theta + \frac{(fsin\theta - cos\theta)(R-x-fcos\theta)}{fcos\theta + sin\theta}\right)\right) - c\right)$$

and with some further algebra:

$$u = -k\left(\frac{f(R-x+sin(\theta))}{sin(\theta)+fcos(\theta)} - c\right)$$

Observing the equation already computed, the following subsystem can be isolated:

$$\begin{bmatrix}\dot{x}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos\theta\\ -k(\tau_r - c)\end{bmatrix} \tag{7.4}$$

In the range of variables of interest, the only equilibrium point for the subsystem is $(x, \theta) = (\frac{f+fR-c}{f}, \frac{\pi}{2})$ and it is possible to demonstrate that it exists a neighborhood U

of this point such that for all initial conditions (x_0, y_0, θ_0) with $(x_0, \theta_0) \in U$ the Single Wall strategy asymptotically guides the vehicle (with kinematic model 3.1) onto the line with x-coordinate constant and equals to the one of the rest point. The interval U, for any gain $k > 0$, can be chosen as $U \subset \{(x, \theta) : x < R; \frac{\pi}{2} - \varphi < \theta < \pi - \varphi\}$. The constraint $\theta < \pi - \varphi$ is necessary because otherwise the camera is not able to detect the feature anymore. In Figure 7.3 all the possible scenarios in which the Single Wall strategy is used in this work are considered. In order to study

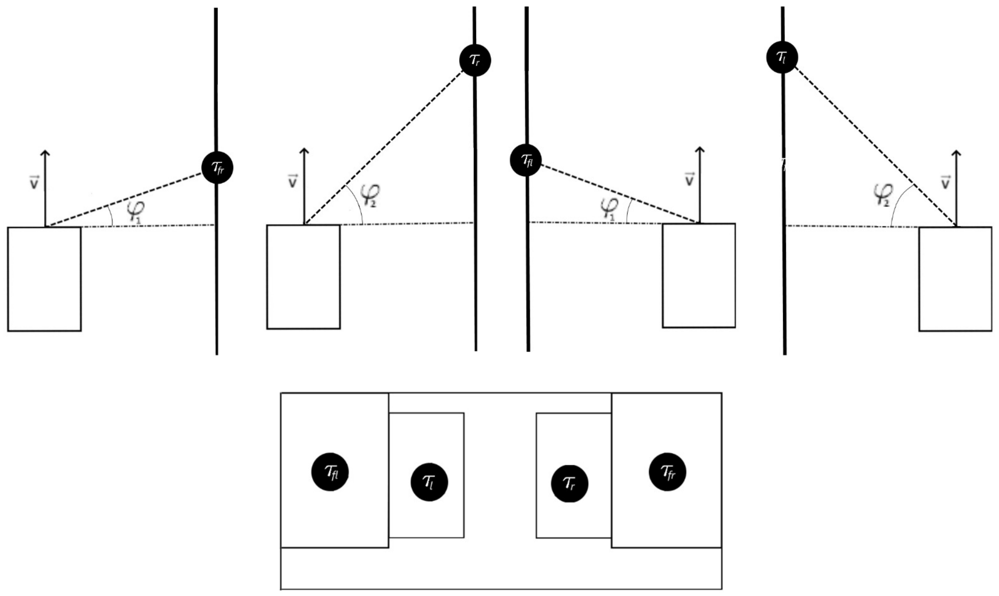


Figure 7.3: A moving vehicle and an infinitely long wall. In the first two scenarios, the wall is on the right, $O_r(x_r, y_r)$ is the detected feature, $tan(\varphi_1) = f_1$ and $tan(\varphi_2) = f_2$. In the last two scenarios, the wall is on the left, $O_l(x_l, y_l)$ is the detected feature, $tan(\varphi_1) = f_1$ and $tan(\varphi_2) = f_2$. Depending on the position of the available feature, f is chosen to be $f = f_1$ or $f = f_2$.

the stability of this motion primitive, the subsystem (7.4) is linearized around the equilibrium point $(x, \theta) = (\frac{f+fR-c}{f}, \frac{\pi}{2})$, which is also the only rest point in the interval U, obtaining:

$$\begin{bmatrix} \delta\dot{x} \\ \delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ fk & -kfc \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\theta \end{bmatrix}$$

The eigenvalues of the matrix representing the controlled system are:

$$\lambda_{1,2} = -\frac{kfc}{2} \pm \frac{\sqrt{kf(kfc^2 - 4)}}{2} \tag{7.5}$$

Both of them are always in the left plane and for this reason the system is asymptotically stable. It is important to notice that if $k < \frac{4}{fc^2}$ the eigenvalues become complex numbers and this introduces oscillations in the system.

The same considerations apply to the study of the equilibrium point and of the stability of the control law with the feature point detected only on the left. The Single Wall control law becomes:

$$u(t) = k(\tau_l - c) \tag{7.6}$$

In this case, if $\tau_l > c$ the difference is positive and the robot has to turn on the left to minimize it, while if $\tau_l < c$ the difference is negative and the vehicle has to turn on the right. Moreover, by substituting the coordinates' value of the left feature (presented in 6.1) in the geometric definition of the TTT the following expression is obtained:

$$\tau_l = cos(\theta)(-R - x) + sin(\theta)\left(f sin(\theta) + \frac{(f sin(\theta) + cos(\theta))(f cos(\theta) + R + x)}{sin(\theta) - f cos(\theta)}\right)$$

and, by using it in (7.6), it can be defined:

$$u = k\left(\frac{f(R + x + sin(\theta))}{sin(\theta) - f cos(\theta)} - c\right)$$

It is then possible to isolate the following subsystem:

$$\begin{bmatrix} \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta \\ k(\tau_r - c) \end{bmatrix}$$

and compute the rest point that, in this case, is $(x, \theta) = (\frac{c-fR-f}{f}, \frac{\pi}{2})$. The open neighborhood of the rest point is $U \subset \{(x, \theta) : -R < x; \varphi < \theta < \frac{\pi}{2} + \varphi\}$. To compute the eigenvalues, the system is linearized around the equilibrium point resulting again in:

$$\begin{bmatrix} \delta\dot{x} \\ \delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ fk & -kfc \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\theta \end{bmatrix}$$

The eigenvalues of the matrix representing the controlled system are the same presented in (7.5), this means that the conclusions about the stability remain the ones expressed above.

The two versions of the Single Wall control law presented in (7.3) and (7.6) depend on the time-to-transit but also on a parameter called c that has to be chosen properly because it defines the equilibrium point and so it determines where the vehicle is asymptotically guided for every initial conditions (x_0, y_0, θ_0) with $(x_0, \theta_0) \in U$. The c is a constant value that can be selected according to two different procedures which are presented in the following section and Eulerian simulations (with instantaneous sensing), performed on Matlab, are shown for both cases.

7.3 Eulerian Simulation Results

1. c **as constant value fixed a priori:** in this situation the term c is defined as a constant value fixed a priori and it remains the same for all the time in which the robot moves through the environment. The simulations presented in Figure 7.4 show possible trajectories generated by the Single Wall control law with different initial conditions (x_0, y_0, θ_0). As it can be noticed, in the

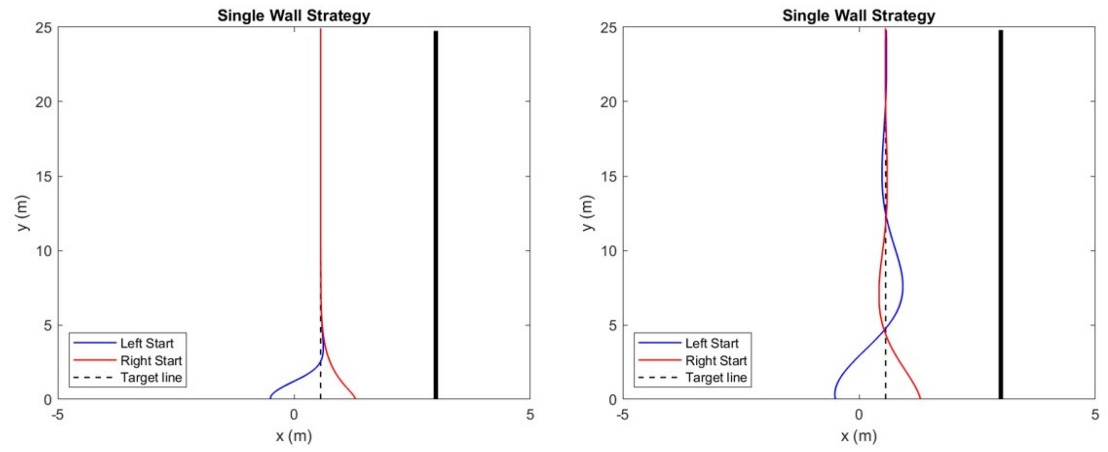


Figure 7.4: Possible trajectories obtained with the Single Wall control law $u(t) = -k(\tau_r - c)$ with $c = 1.8$. On the left the proportional gain has been set to respect $k > \frac{4}{fc^2}$ and so there are no oscillations, on the right k has been chosen such that $k < \frac{4}{fc^2}$ and the system undergoes oscillations.

left image there are no oscillations in the trajectories because the proportional gain k has been chosen in order to avoid complex eigenvalues, while in the right image what happens if the k value does not respect the condition $k > \frac{4}{fc^2}$ is represented. However also in this situation the linearized system is asymptotically stable, and the simulation shows asymptotic approach of the vehicle to a straight path parallel to the wall.

2. c **as constant value changed dynamically:** in this situation the term c is still defined as a constant but it is equal to $\tau(t_0)$, where t_0 is the time at which the Single Wall strategy starts to be applied. Since this motion primitive aims to support the navigation when the Tau Balancing control law cannot be used because of feature sparsity, every time a switch between the Tau Balancing and the Single Wall strategy occurs the c value has to be redefined. Figure 7.5 shows a possible trajectory generated by the Single Wall control law with initial condition $(x_0, y_0, \theta_0) = (1, 0, \frac{2\pi}{3})$ and on the left image there are real, negative eigenvalues since the proportional gain is $k > \frac{4}{fc^2}$. On the right image

$k < \frac{4}{fc^2}$ and the eigenvalues are negative but complex.

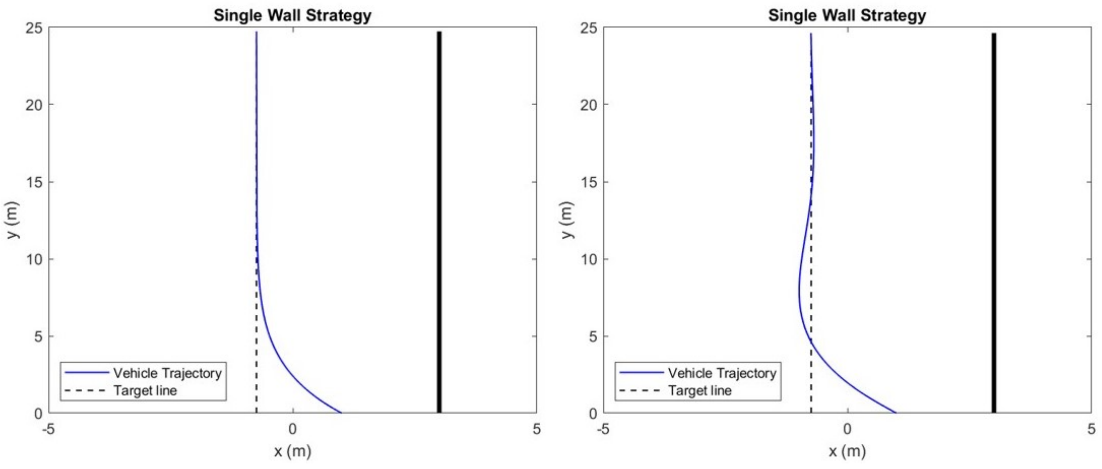


Figure 7.5: Possible trajectories obtained with the Single Wall control law $u(t) = -k(\tau_r - c)$ where $c = \tau(t_0)$ and $(x_0, y_0, \theta_0) = (1, 0, \frac{2\pi}{3})$. On the left there are no oscillations because $k > \frac{4}{fc^2}$, on the right there are oscillations because $k < \frac{4}{fc^2}$.

The methods presented above for the choice of the c value in the Single wall strategy are both valid and they lead to the desired control action. However, even if from a theoretical point of view there is no difference in choosing one method with respect to the other, from tests on the real Jackal robot (refer to Chapter 9 for more details) it has been noticed that the second approach is not always reliable when used on a real platform. Since this method selects $c = \tau(t_0)$, it is important to have a good estimate of the time-to-transit values. The introduction of the Sense-Perceive-Act cycle leads to a generally improved estimate of TTT values, but this does not guarantee that at the end of every sense phase the perceived time-to-transit is equal to the geometric one also because rotational motions are not the only source of error in the TTT estimation. A wrong time-to-transit value at t_0 means a bad choice of c and so an unwanted behavior of the control strategy. This is the reason why, in this work, it has been decided to use the first method and to select the c term as a constant, fixed a priori, for all the time in which the robot is moving through the environment.

7.4 Effects of the Sense-Perceive-Act cycle

In order to understand which are the effects of introducing the Sense-Perceive-Act cycle on the stability of the controlled system, a simplified situation in which Eulerian sensing is used will be analyzed. According to the approach presented in

Theorem 2 by Baillieul and Kang in [12], a sample-and-hold version of the Single Wall steering law (with detected feature on the right) can be defined as follows:

$$u(t) = -k[\tau_r(x(t_i), \theta(t_i)) - c], \quad t_i \leq t < t_{i+1} \tag{7.7}$$

where the sampling instants are spaced with $t_{i+1} - t_i = h > 0$. It is possible to prove that, also in this case, there exists a range of gain $k > 0$ that asymptotically guides the vehicle onto the line $x = \frac{f+fR-c}{f}$ for any sufficiently small sampling interval h.

Considering $v = 1$, $f = 1$ and taking $\phi = \theta - \frac{\pi}{2}$, since $\dot{\theta} = u$, the angular velocity becomes:

$$\dot{\phi} = -k\left[\tau_r\left(x(t_i), \phi(t_i) + \frac{\pi}{2}\right) - c\right], \quad t_i \leq t < t_{i+1}$$

Exploiting the equation describing τ_r, the discrete time evolution of ϕ can be then expressed as:

$$\phi(t_{i+1}) = \phi(t_i) - hk\left(\frac{R - x + cos\phi(t_i)}{cos\phi(t_i) - sin\phi(t_i)} - c\right)$$

which means that the discrete time evolution of the heading can be retrieved by iterating the following x-dependent mapping:

$$g(\phi) = \phi - hk\left(\frac{R - x + cos\phi}{cos\phi - sin\phi} - c\right)$$

In particular:

$$g'(\phi) = 1 - hk\left(\frac{1 + (R - x)(cos\phi + sin\phi)}{(cos\phi - sin\phi)^2}\right)$$

It can be noticed that the denominator is always positive, while the numerator is positive in the parameter range of interest $x < R$ and $-\frac{\pi}{4} < \phi < \frac{3\pi}{4}$. This leads to the possibility of choosing a k sufficiently small to make g a contraction on these intervals, as a consequence ϕ converges to 0. Since $\dot{x} = -sin\phi$, it is true that the sample-and-hold version of the Single Wall strategy (with detected feature on the right), presented in (7.7), can asymptotically guide the vehicle onto the line $x = \frac{f+fR-c}{f}$.

The same reasoning can be applied to the Single Wall strategy with detected feature on the left, obtaining the same results. In this case:

$$g(\phi) = \phi + hk\left(\frac{R + x + cos\phi}{cos\phi + sin\phi} - c\right)$$

$$g'(\phi) = 1 - hk\left(\frac{1 + (R + x)(cos\phi - sin\phi)}{(cos\phi + sin\phi)^2}\right)$$

In the parameter range of interest $x > -R$ and $-\frac{3\pi}{4} < \phi < \frac{\pi}{4}$, the sample-and-hold version of the steering control law aligns the vehicle to the line $x = \frac{c-f-fR}{f}$.

Chapter 8

Spatial Awareness and Simulation Results

In this chapter a detailed description of how a mobile robot can understand its position and the general geometry of the environment in which it is moving is provided. In particular, in the following sections, the characteristics of the visual cues coming from four different scenarios are analyzed and some simulations, run in Gazebo using Lagrangian sensing, are used to show the behavior of a mobile robot safely navigating in these environments.

8.1 Straight Corridors

Straight corridors represent the easiest environment in which the mobile robot used in this work can navigate. In this scenario the classical distribution of the OF field in the image is the one shown in Figure 8.1. When moving in a straight corridor the τ values generally come from every region of interest but the central ROI (whose time-to-transit is not available or very high) and they are continuous in time. These are clear indicators that the environment is a straight corridor. Moreover, since in this case the control law used by the robot to navigate in the environment is the following[1]:

$$u(t) = k_f(\tau_{fl} - \tau_{fr}) + k_m(\tau_l - \tau_r) \tag{8.1}$$

a balanced (symmetrical) distribution of the OF field is expected because this is a straightforward consequence of the application of the steering law which keeps the differences $\tau_{fl} - \tau_{fr}$ and $\tau_f - \tau_r$ as close as possible to 0. Smaller values in

[1]Chapter 6 provides more information about this version of the *Tau Balancing* control law.

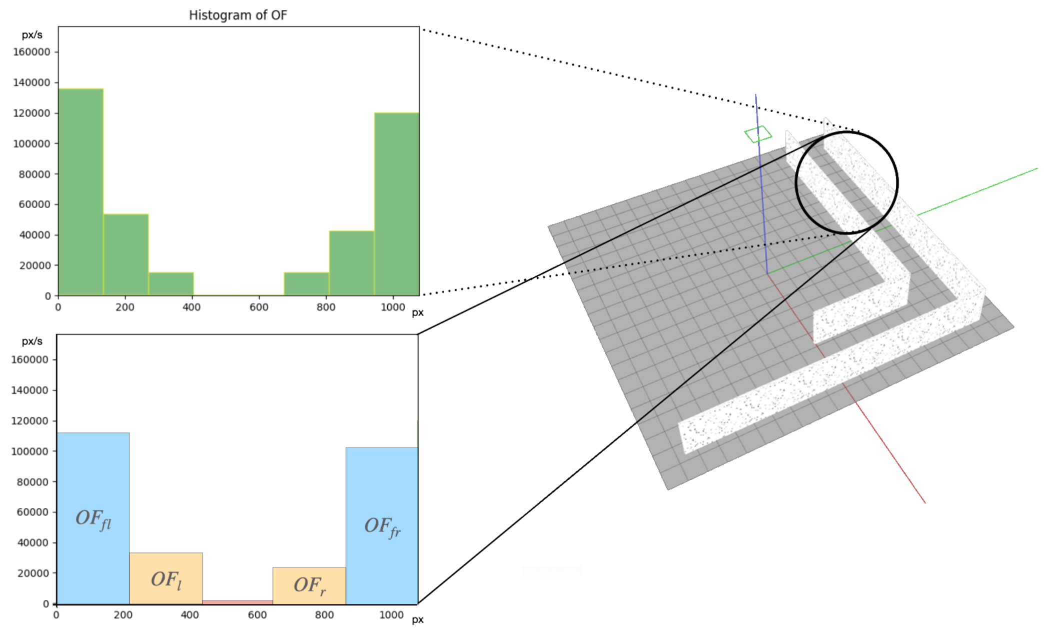


Figure 8.1: Averaged optical flow field distribution in the image typical of a straight corridor when 8 (at the top) or 5 (at the bottom) ROIs are considered.

the optical flow field on the right of the image mean the robot is located on the left with respect to the center of the corridor and vice versa. Another important thing to notice in Figure 8.1 is the fact that eight ROIs can be useful to obtain a more refined idea of the geometry of the environment, but five are enough to distinguish a straight corridor from alternative scenarios taken into account in this work and this will be more and more evident going through the chapter. In the case only time-to-transit values coming from the far left and far right or from the left and the right regions of the image are available, the classical version of the Tau Balancing control law is used in one of this two forms:

$$u(t) = k_f(\tau_{fl} - \tau_{fr})$$

$$u(t) = k_m(\tau_l - \tau_r)$$

8.1.1 Lagrangian simulations results

In this section, results obtained from simulations run in the Gazebo simulation environment[2] are shown and discussed. With respect to the ones obtained using

[2]More information about ROS and Gazebo are provided in Chapter 4.

Matlab, in this case Lagrangian sensing and a real model of the Jackal robot (the platform on which the control strategy implemented in this work will be tested) are used. The behavior of the UGV in artificial environments with fixed a priori feature density and in real scenarios (in which feature density is not constant) will be analyzed. On the left of Figure 8.2, an artificial straight corridor in which feature density on the walls is a Bernoulli distribution with $p = 0.05$ is represented. On the right, it is instead possible to observe the trajectory of the Jackal robot

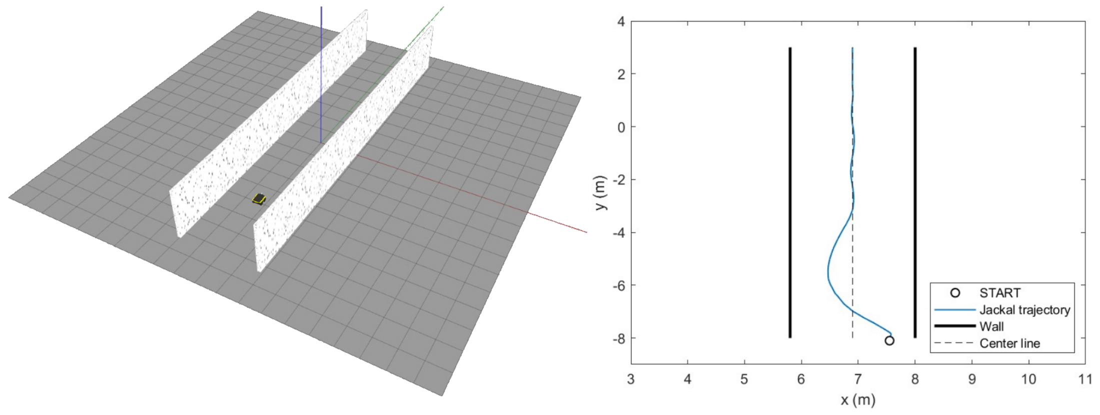


Figure 8.2: On the left, an artificial straight corridor while, on the right, the trajectory of the Jackal robot moving in it is represented.

which is using the Tau Balancing control law in (8.1) to move to the center of the corridor. An initial overshoot is present, but then the center line is stably reached and the robot undergoes only minor oscillations, probably due to residual noise in the estimation of time-to-transit. Clearly, if we lower the feature density, more oscillations can be observed and a discussion on the relation between feature density and control error is made in Appendix A. Looking at what happens when using Eulerian sensing (see Figure 6.5), it can be noticed that, even if the eigenvalues of the system are real and negative, when using Lagrangian sensing in combination with the Sense-Perceive-Act cycle, overshoot and oscillations are present. The main reason behind the presence of a so evident overshoot can be hidden in the delays that the introduction of sense and act phases entails but a theoretical analysis of why this happens is the goal of further work.

Since the final objective is to test the control strategy developed on a real platform, simulations in environments that present no a priori defined density of features must be run. In Figure 8.3, the result of one of these tests is shown. In this case, the performance is even better (in terms of overshoot) with respect to when testing in the artificial corridor. This can be the consequence of the fact that the tree-lined avenue contains a lot of detectable features making the estimation of time-to-transit values more accurate. In any case, this result shows the difficulty of

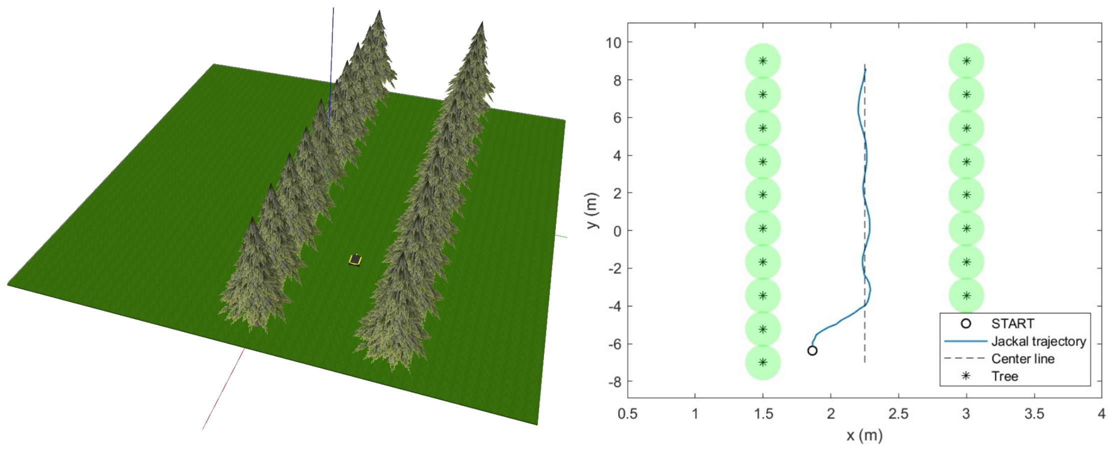


Figure 8.3: On the left, a tree-lined avenue is represented while, on the right, it is shown the trajectory of the Jackal robot moving in the environment.

studying, from a theoretical point of view, what are the effects of using Lagrangian sensing in combination with the Sense-Perceive-Act cycle.

8.2 Turns

Turns represent a challenge for vision-based navigation using Optical Flow and in this section some of their distinctive characteristics will be analyzed. In Figure 8.4, a typical distribution of the optical flow field captured by a mobile robot ready to turn is represented.

The presence of a turn can be detected by the control strategy implemented in this work by looking at the appearance of a suddenly unbalanced optical flow field and of a small time-to-transit value coming from the central ROI (of the order of some seconds). The reason behind the discontinuity of the τ signal is graphically represented in Figure 8.5. This sudden change in the time-to-transit signal has to be handled properly otherwise it can lead to severe stability issues.

First of all, some aspects of how to ideally handle a turn are taken into account and all the considerations in this section will be made assuming the linear velocity $v = 1$ that means the vehicle path is parameterized by arc length. Considering W as the corridor width, to successfully cover a 90° arc the vehicle turning rate $u(t)$ must be able to exceed a minimum value u_{min} defined as follows:

$$u_{min} = \frac{\sqrt{2}-1}{\sqrt{2}W} \tag{8.2}$$

This comes from considering an ideal vehicle (that can be represented by a point) starting with heading $\theta_0 = \frac{\pi}{2}$ from the far right part of the corridor. Timing is

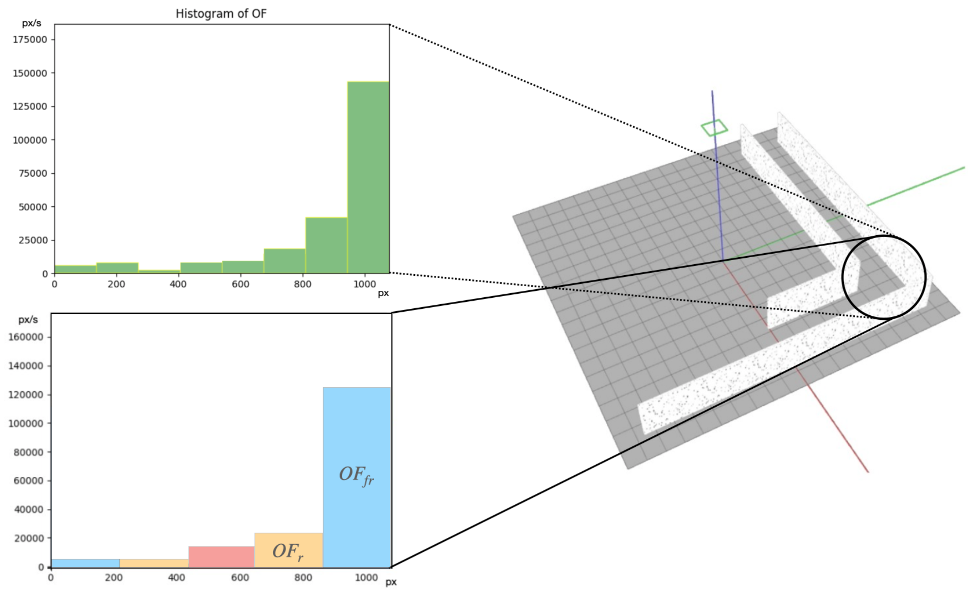


Figure 8.4: Averaged optical flow field distribution in the image typical of a turn when 8 (at the top) or 5 (at the bottom) ROIs are considered.

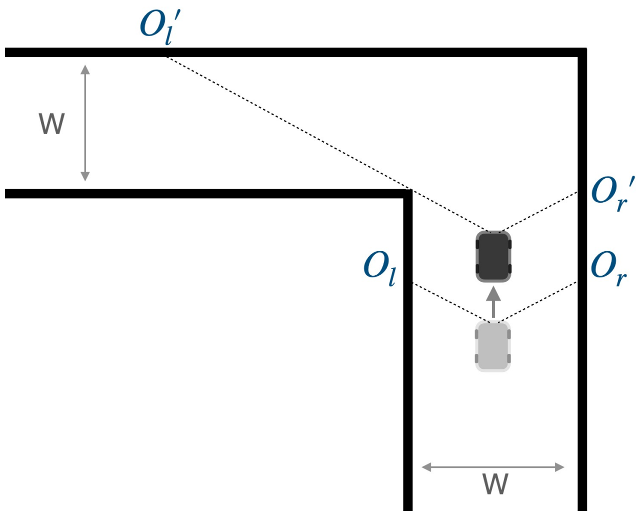


Figure 8.5: A mobile robot approaching a turn. The time-to-transit associated to the feature detected on the left changes suddenly when the feature O_l' is detected.

essential when dealing with a turn since turning too soon will end up hitting the left wall of the vertical segment while turning too late will make the mobile robot crash into the right wall of the horizontal segment. In order to understand when it is possible to start turning from a purely theoretical point of view, some Matlab simulations have been run for different initial conditions (x_0, θ_0).

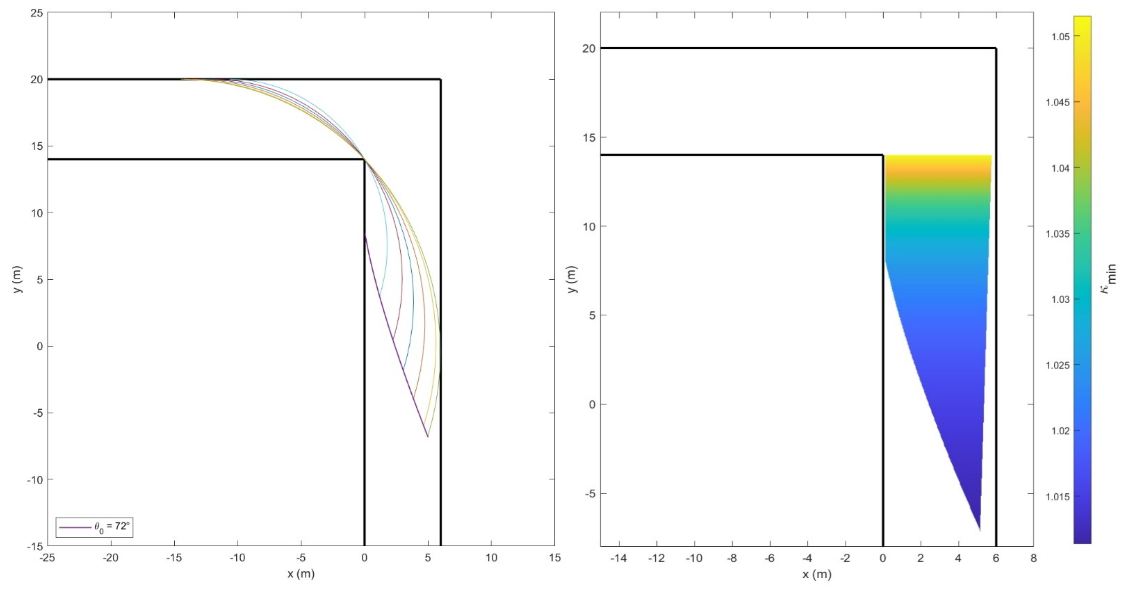


Figure 8.6: On the left, it is shown the line from where it is possible to start turning when the initial heading is $\theta_0 = 72°$ (with some possible trajectories). On the right the full area (in the case $\theta_0 = 72°$) from where a mobile robot can start turning and the relative minimum constant curvature that must be guaranteed to make the turn is represented.

In this work, the mobile robot can perform a turn combining the following two motion primitives:

$$u(t) = k'_f(\tau_{fl} - \tau_{fr})$$

$$u(t) = -k(\tau_r - c) \quad or \quad u(t) = k(\tau_l - c)$$

The Tau Balancing control law (where $k'_f > k_f$ in (8.1) to balance more rapidly the optical flow field) can exploit the discontinuity of the time-to-transit signal to make the vehicle start turning while the Single Wall strategy handles the situation, that typically occurs when aligning with the new corridor at the end of the turning procedure, in which no features in the left (when turning left) or in the right (when turning right) ROIs are detected. In this scenario the introduction of the Sense-Perceice-Act cycle is of paramount importance because the rotational effects discussed in Chapter 3 severely confounds the accuracy of tau estimation and the usage of averaged time-to-transit values allows to avoid (or at least to handle) the destabilizing effect due to τ discontinuity. In order to understand the behavior of the Jackal robot in different environments, some Gazebo simulations using Lagrangian sensing are introduced.

8.2.1 Lagrangian simulations results

A first simulation tries to highlight the behavior of the Tau Balancing control law when small right and left turns are present. In Figure 8.7, the artificial environment where the feature density is given by a Bernoulli distribution with $p = 0.03$ and the trajectory of the mobile robot moving in it are presented. Even if the robot starts

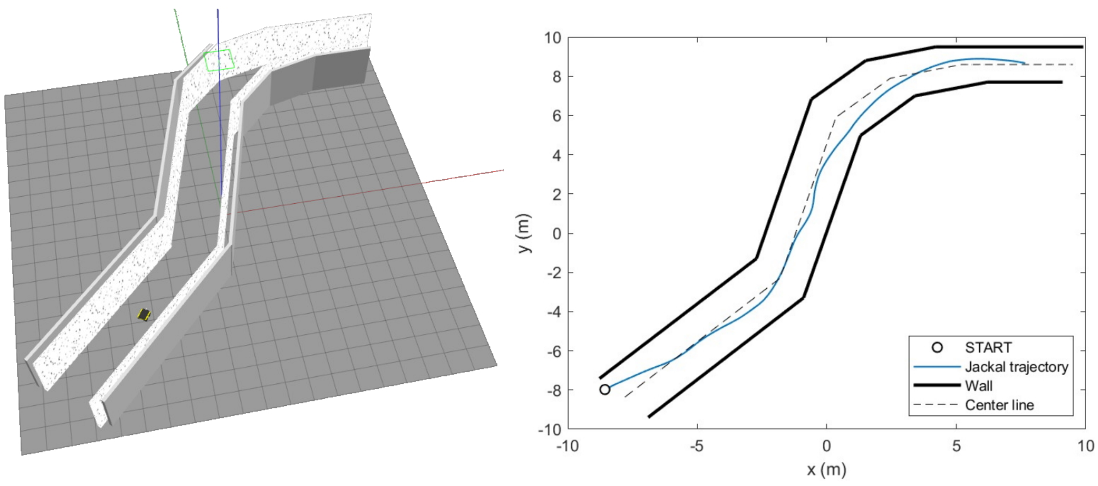


Figure 8.7: On the left, the artificial simulation environment created in Gazebo is shown while, on the right, the Jackal trajectory is represented.

on the left part of the corridor, it is able to stably reach its center. The timing chosen by the UGV to start turning is correct since it allows the vehicle to end the turning procedure without hitting or getting too close to the wall. In Figure 8.8, a more challenging scenario represented by a corridor with an L turn with feature density given by a Bernoulli distribution with $p = 0.05$ is shown. Also in this case, even if the environment is more challenging since it presents a 90° turn, the robot successfully moves from the vertical to the horizontal corridor. Some oscillations are present, but they are probably linked to residual noise in the optical flow field which leads to inaccuracies in the time-to-transit estimation. Moreover, when reaching the end of the horizontal corridor, the number of features that the Jackal robot can detect decreases and this means wider oscillations.

The results of a simulation in a real environment (with feature density not defined a priori) are presented in Figure 8.9. In this case the absence of features on the left ROIs obliges the mobile robot to completely trust the right regions of interest and a safe distance from trees is maintained. When the UGV perceives the presence of a corridor (a bridge, in this case), it properly aligns itself to the center of it.

Another important aspect that must be taken into account to obtain good results is the width of the horizontal field of view (hFOV) of the camera that the

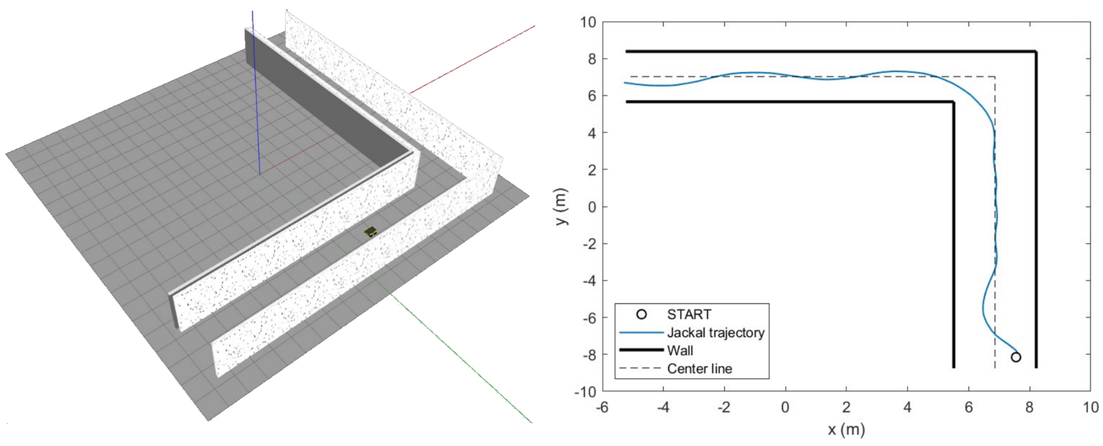


Figure 8.8: On the left an artificial L turn is shown while, on the right, the Jackal trajectory is represented.

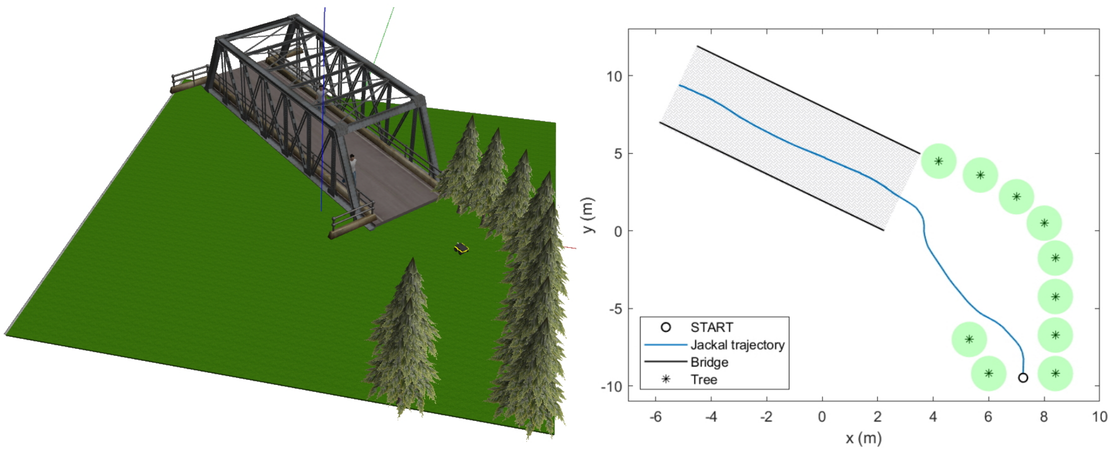


Figure 8.9: On the left a challenging realistic environment while, on the right, the trajectory followed by the Jackal is presented.

mobile robot uses to acquire data. Since, as it has already been discussed, timing is essential when dealing with turns, a small hFOV can let the robot perceive the turn too early and this can lead to a crash. A comparison between Figure 8.10 and Figure 8.5 can be made to better understand the issue.

In the next section a more detailed description, supported by simulations using Lagrangian sensing, of when and how the Single Wall strategy works is presented.

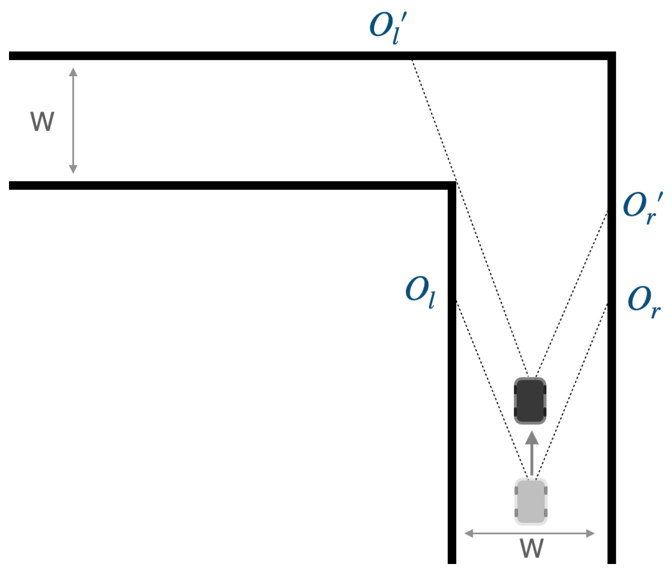


Figure 8.10: A too small hFOV can let the robot perceive the turn too early and it forces the vehicle to abandon the Tau Balancing control law used in straight corridors.

8.3 Single walls

Single walls is an expression that wants to describe all the scenarios in which features are detected only in one side of the environment (or, more generally, when only one side has enough features to permit the computation of acceptable TTT values). The typical distribution of the optical flow field of these kind of environments is graphically represented in Figure 8.11. To deal with feature sparsity the control law used to guide the mobile robot is the Single wall strategy presented in Chapter 7. In the case in which all the ROIs on the left or on the right allow the estimation of time-to-transit, only the TTT value coming from the far left (or far right) region is used. The reason behind this choice is that normally the optical flow field is more accurate in these areas since pixel motion is higher.

The results of a simulation in an artificial environment are summarized in Figure 8.12. Also in this case the feature density on the wall is given by a Bernoulli distribution with $p = 0.05$. The UGV can explore the environment even if no features are detected on the right side of the image it acquires. Some oscillations were expected since the motion primitive used relies just on a single time-to-transit value. It is also good to keep in mind that the Single wall strategy aims to support the navigation only when Tau Balancing cannot be exploited.

In order to show that the motion primitive used to deal with sparsity of features can be helpful even during turns, a more complex artificial environment, shown

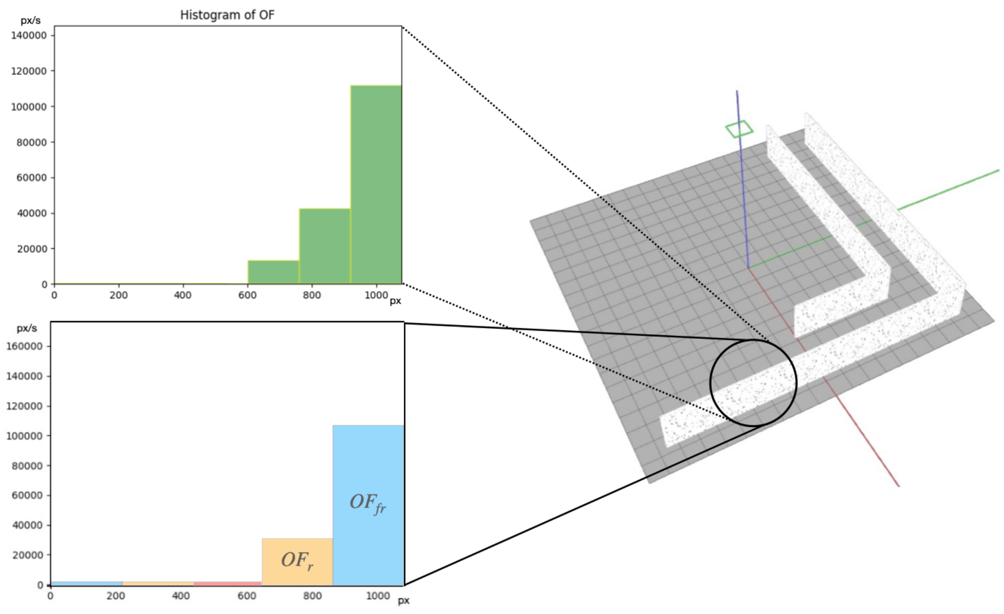


Figure 8.11: Averaged optical flow field distribution in the image with features detected only on the right side when 8 (at the top) or 5 (at the bottom) ROIs are considered.

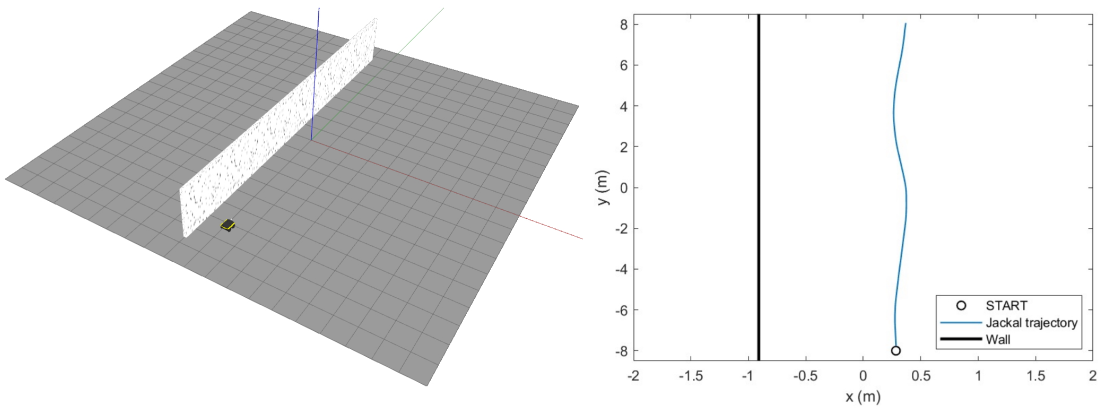


Figure 8.12: On the left an artificial wall while, on the right, the trajectory followed by the Jackal robot.

in Figure 8.13, has been created. The feature density of the walls is represented, also in this case, by a Bernoulli distribution with $p = 0.05$. In Figure 8.13 not only the Single Wall strategy, but all the motion primitives presented are used and the Jackal is able to correctly explore the environment keeping a safety distance from every wall. An even more promising result is shown in Figure 8.14, when the

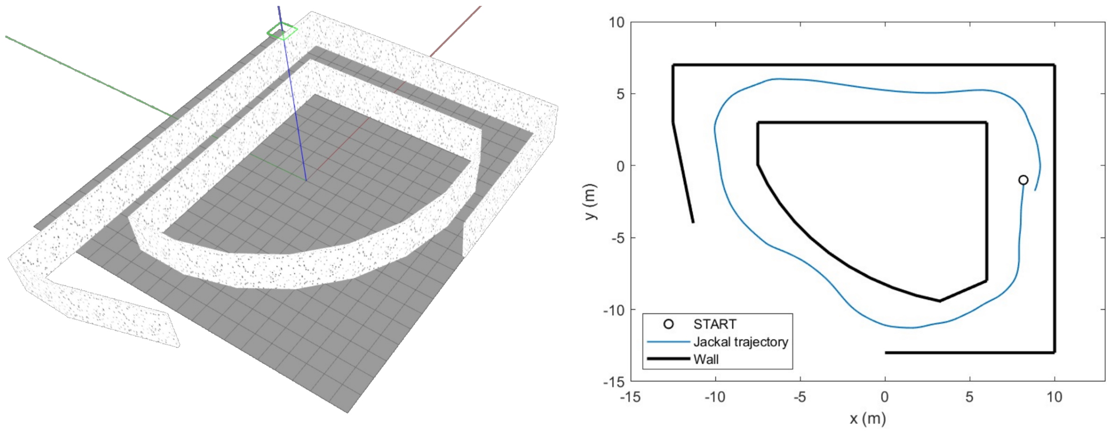


Figure 8.13: On the left a complex artificial environment to highlight the possible usage of the Single Wall strategy even when dealing with turns while, on the right, the trajectory of the Jackal is represented.

scenario is realistic and the feature density has not been chosen a priori.

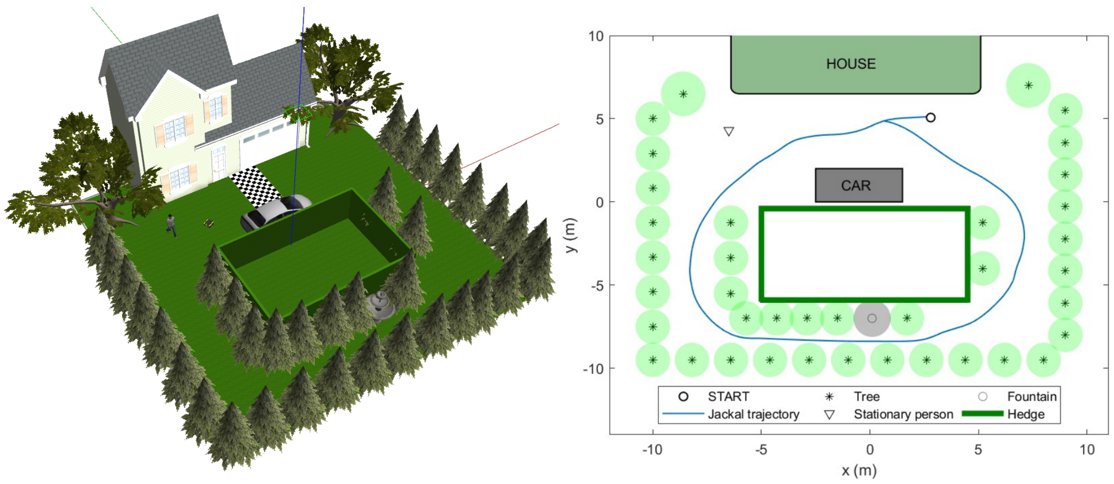


Figure 8.14: On the left a complex realistic environment in which all the motion primitives presented in this work are used while, on the right, the trajectory followed by the Jackal robot is shown.

8.4 Obstacles in the central ROI

One of the weaknesses of the estimation of time-to-transit from Optical Flow with a monocular camera is the fact that in the center of the image, near the focus of expansion, only small pixel motion is present and this leads to poor estimation precision. This can be a real problem when there is the need of detecting the presence of an object to be avoided in the central region of interest and in order to solve this issue a lot of work has still to be done. In Figure 8.15, the expected optical flow field that the robot should perceive when in such a scenario. The

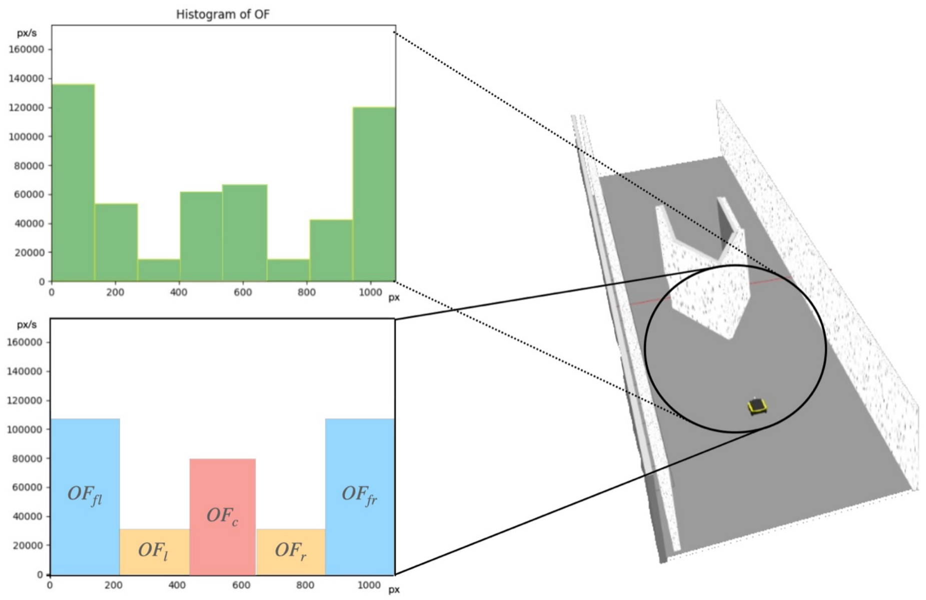


Figure 8.15: Averaged optical flow field distribution when a central object is present in the case 8 (at the top) or 5 (at the bottom) ROIs are considered.

central region can be an instrument to detect the presence of an obstacle, but the TTT value coming from it is not accurate and it cannot give a precise idea on the remaining time before which the robot would crash into the object if no steering action is performed. If the object is not in the center of the image, the Tau Balancing control law used in this work can easily avoid the collision while, in the case in which the obstacle is in the middle of a corridor and the robot is already perfectly aligned to the center of it, this motion primitive alone is not enough to quickly steer the vehicle and avoid the collision.

From a theoretical point of view, it could be possible to properly combine the Tau Balancing control law and the Single Wall strategy, as depicted in Figure 8.16, to solve the problem. Following this idea, the Single Wall strategy should be activated according to a specific timing based on the time-to-transit coming from the central ROI and its only goal would be to slightly move the robot from the center of the corridor to the left or to the right to create an unbalanced optical flow field that the Tau Balancing control law can then balance again driving the UGV around the obstacle. This is in practice not easily implementable for many reasons,

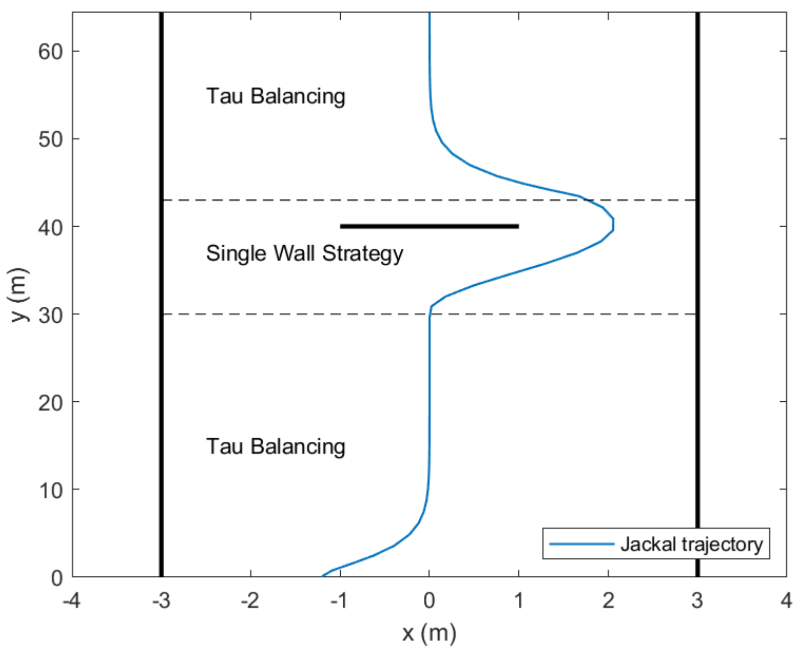


Figure 8.16: The combination of two motion primitives could help to avoid an obstacle in the center ROI.

some of which already discussed. The main problem of this approach is that the time-to-transit coming from the central region of interest is very noisy and so it can be used as an indicator of obstacle ahead of the vehicle but its value cannot be used as a reliable steering signal. In this work no solution to solve the problem will be presented since it will be subject of further work. What is important to notice is that, in general, it is theoretically possible for a looming obstacle to avoid being detected if the overall optical flow field of the environment is highly symmetric, but such symmetry is improbable in natural scenes.

Chapter 9

Tests on a Real Platform

When new software components are developed, after the verification of the algorithms' effectiveness through simulations, it is important to test them on a real platform. Thanks to the availability of the PoliTo Interdepartmental Centre for Service Robotics (PIC4SeR) and to their collaboration, the software presented in this work have been deployed and tested on a Jackal UGV equipped with a MYNT EYE S1030 camera.

A brief description of the real platform and of the sensors that have been used is now provided, then the results of the tests are presented.

9.1 Jackal UGV

The Jackal UGV is a small, fast and easy to use robotic platform developed by Clearpath Robotics. It has an onboard computer, an IMU and a GPS fully integrated with ROS. It is possible to equip the Jackal robot with a wide number of accessories by simply connecting cables to the internal onboard computer. This UGV is waterproof and it can be used on different types of terrain. In Figure 9.1 the Jackal robot is shown.

9.1.1 Mathematical modeling

The mechanical configuration of the Jackal UGV is composed by two pairs of fixed wheels, a pair for each side, that can be differentially driven to perform curved trajectories. The Jackal robot, because of its configuration, belongs to the category of the skid-steering mobile robots (SSMR) in which a lateral skidding is necessary to follow curved paths. Obviously this type of mobile robot presents velocity constraints that differ from the ones of vehicles in which the skidding is not used. Models of both the kinematics and dynamics of SSMRs have been described in [28],

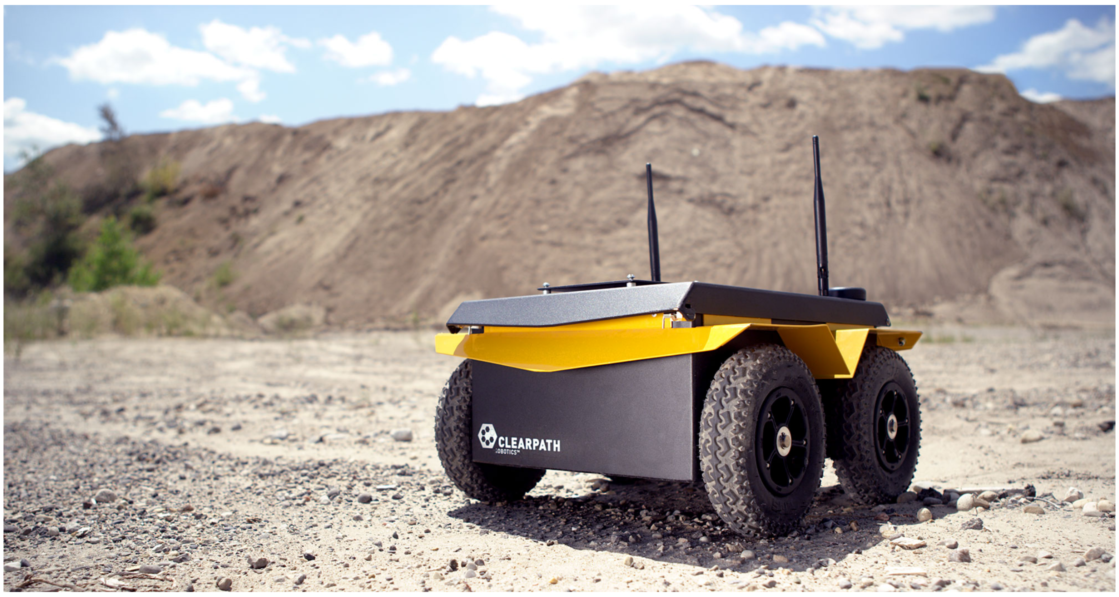


Figure 9.1: Jackal UGV. *Source: https://clearpathrobotics.com*

and key features of the models are presented in Appendix B.

Even if the kinematic model of a SSMR is different from the one of the unicycle, which has been used in this work to demonstrate the stability of the presented vision-based steering laws, the extent of the violation of the nonholonomic constraints is considered small enough to still safely use the Tau Balancing control law and the Single Wall strategy.

9.2 Sensors

The final objective of this work is to safely guide a mobile robot in unknown environments exploiting only the data coming from a single camera mounted on it. This goal has been achieved using only one sensor, a camera. The camera that has been used in this work is the MYNT EYE S1030 stereo camera, but before it another type of camera, the Intel RealSense D435i stereo camera, has been tested. A summary of the characteristics of both cameras is now presented, focusing on the reason why the MYNT EYE S1030 has been chosen.

9.2.1 Intel RealSense D435i

The Intel RealSense D435i is a depth camera with an integrated Inertial Measurement Unit (IMU) which is used for the detection of movement and rotations and it has 6 degrees of freedom. The camera is shown in Figure 9.2 and it is composed of a D430 module and an RGB camera. The former is used for depth estimation while the latter is used to acquire RGB images. Since, considering the scope of this

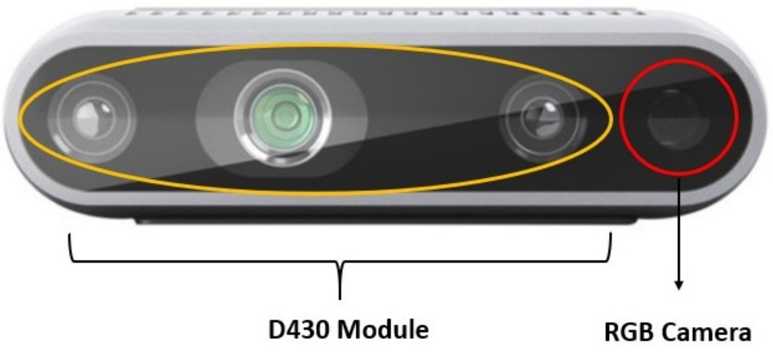


Figure 9.2: Intel RealSense D435i stereo camera. More information can be found at *https://www.intelrealsense.com*

work, the important part of the camera is the RGB monocular camera, in Table 9.1 a description of its main characteristics is provided.

RGB frame resolution:	1920x1080
RGB frame rate:	30 fps
RGB sensor technology:	Rolling Shutter
RGB sensor FOV (H $\times$ V $\times$ D):	$64° \times 41° \times 77°$ ($\pm 3°$)
RGB sensor resolution:	2 MP

Table 9.1: Main characteristics of the RGB camera of the Intel RealSense D435i.

Even if the resolution and the frame rate of the RGB camera are suitable for the purposes of this work, a relevant limitation is represented by the horizontal field of view (hFOV) of the sensor that has to be wide enough to recognize, with the proper timing (as discussed in Chapter 8), specific characteristics of the visual cues, e.g. discontinuities of the time-to-transit signal in environments containing turns.

9.2.2 MYNT EYE S1030

The need of a large horizontal FOV has led us to discard the Intel RealSense D435i in favor of the MYNT EYE S1030 stereo camera, presented in Figure 9.3. It is a depth and an infrared camera with an integrated IMU and it gives the possibility to retrieve the monochromatic images from both the single cameras.

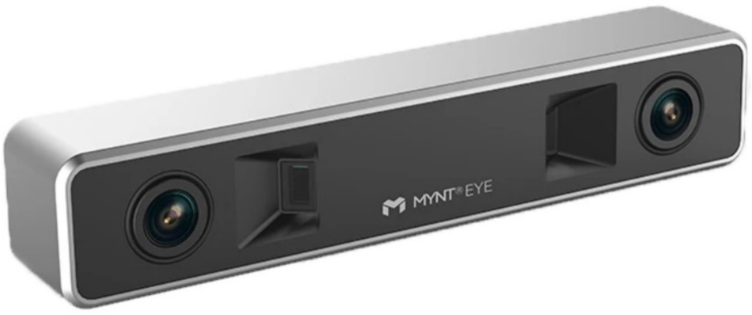


Figure 9.3: The MYNT EYE S1030 stereo camera mounted on the Jackal robot. *Source: https://www.mynteye.com/*

Some of the main characteristics of this stereo camera are summarized in Table 9.2 and, as it can be noticed, the horizontal FOV is more or less two times larger than the one of the Intel RealSense stereo camera. Since, as discussed before, a wide horizontal field of view is a strong requirement in this work, the MYNT EYE S1030 has been chosen as the suitable sensor for performing the tests.

Frame resolution:	725x480
Frame rate:	60 fps
Sensor FOV (H × V × D):	122° × 76° × 146°
Sensor Range:	0.5 ~18 meters +

Table 9.2: Main characteristics of the MYNT EYE S1030.

The stereo camera has been mounted on the Jackal UGV, as shown in Figure 9.4, with the right camera centered with respect to the vehicle. Then, during the tests, only the images coming from the right camera have been used to respect the constraint of using a monocular camera to guide the robot. The characteristics of the robot and of the camera exploited during experiments at PIC4SeR are very close to the ones of the material used at Boston University. People in Boston use a Jackal UGV equipped with various sensors such as a LIDAR, a PTZ camera and a ZED 2 stereo camera which has an hFOV very close to the one of the MYNT EYE S1030. In Figure 9.5, the Jackal configuration belonging to Boston University is shown. Since the platform used in this work and the one at Boston University are similar, it will be possible to exchange code and perform more tests in different environments but this will be objective of future work.

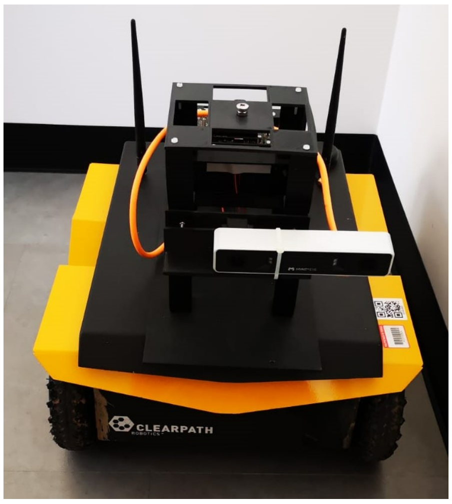


Figure 9.4: Jackal setup for tests in the PIC4SeR laboratory.

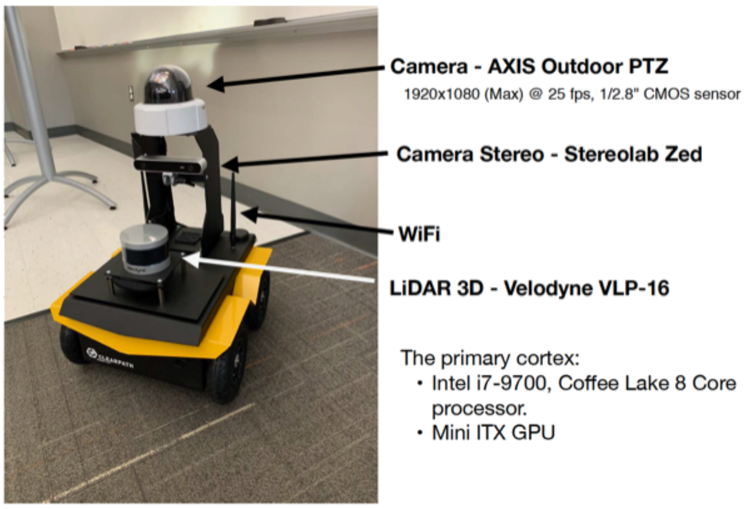
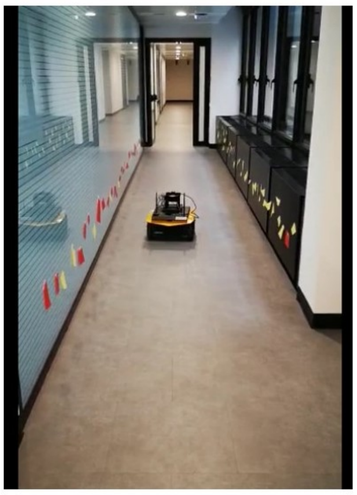

Camera - AXIS Outdoor PTZ
1920x1080 (Max) @ 25 fps, 1/2.8" CMOS sensor
Camera Stereo - Stereolab Zed
WiFi
LiDAR 3D - Velodyne VLP-16
The primary cortex:
• Intel i7-9700, Coffee Lake 8 Core processor.
• Mini ITX GPU

Figure 9.5: Jackal setup at Boston University.

9.3 Tests Results

The effectiveness of the entire algorithm has been tested by observing the behavior of the Jackal robot moving through different environments which have been set up at PIC4SeR and at the Boston University Robotics Lab. For each scenario the data referring to the position of the Jackal have been collected in order to plot the trajectories on Matlab. To retrieve this information a particular node, called *ekf*, has been used in which the Odometry and the IMU data are considered to compute the position of the robot with respect to a coordinates' frame that has the origin where the Jackal is powered on for the first time. The results of the tests are now presented and the boundaries in the Matlab simulations are an accurate representation of the geometry of the environments created in the laboratory.

The first test has been conducted on a straight corridor, the features are represented by the edges and the corners of the post-it notes on the right and left walls. In Figure 9.6 a picture of the scenario and the trajectories of the vehicle starting from different points are shown. As it can be noticed, in both situations



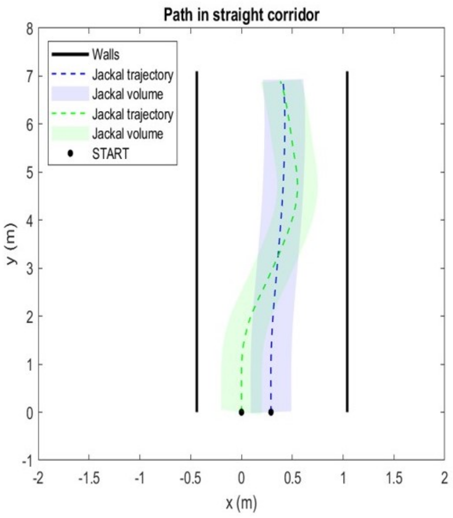


Figure 9.6: On the left the straight corridor created in the laboratory while, on the right, two trajectories followed by the Jackal robot for two different starting points.

the robot is able to reach the center of the corridor and the performances are very close to the ones performed in the Gazebo simulation environment presented in Chapter 8.

The second test has been run on a scenario characterized by a turn of almost 90°, here the features are represented by the edges and the corners of the post-it notes but also symbols on several boxes. The result is shown in Figure 9.7. The timing to perform the turn is correct and the results show a good behavior of the robot. The fact that at the end of the corridor the Jackal is not yet in the center of it is due to the short length of the environment after the turn. A longer corridor would allow the UGV to reach the center.

The third test concerns the single wall scenario in which the Single Wall control strategy $u(t) = -k(\tau_r - c)$ is used. In this case two tests are performed, one with the parameter c as a constant fixed a priori and the second with the c as the TTT value recorded the first instant in which the Single Wall control law is applied

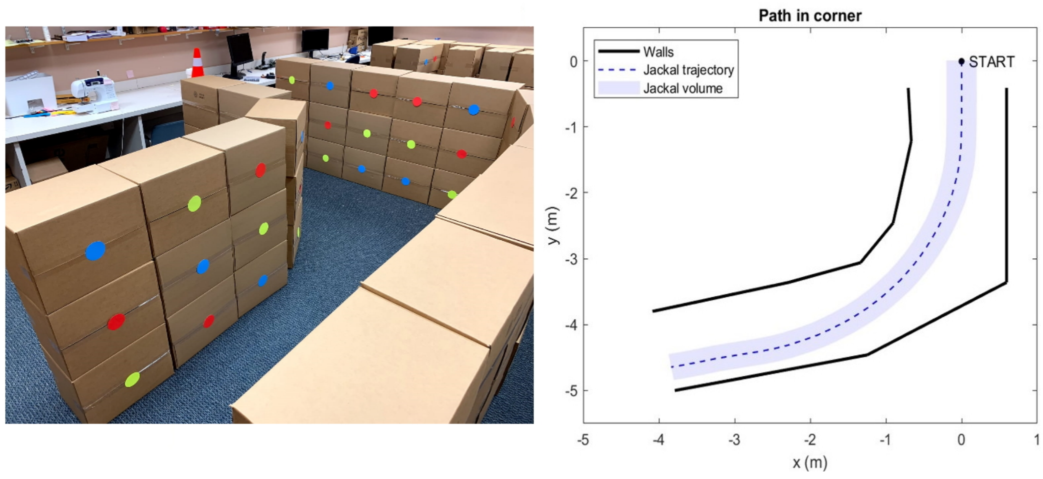


Figure 9.7: On the left the turn of almost 90° created in the laboratory while, on the right, the trajectory followed by the Jackal robot.

(refer to Chapter 7 for more details about this motion primitive). The resulting trajectories are shown in Figure 9.8. Both trajectories show the ability of the robot

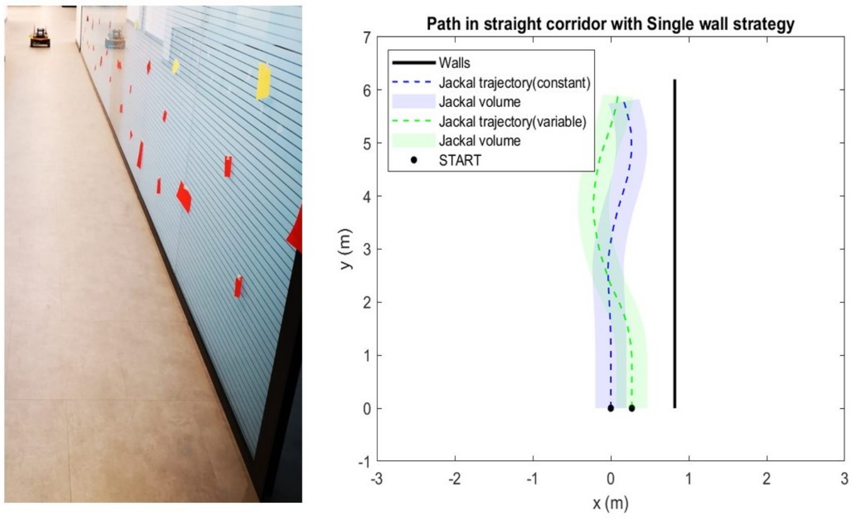
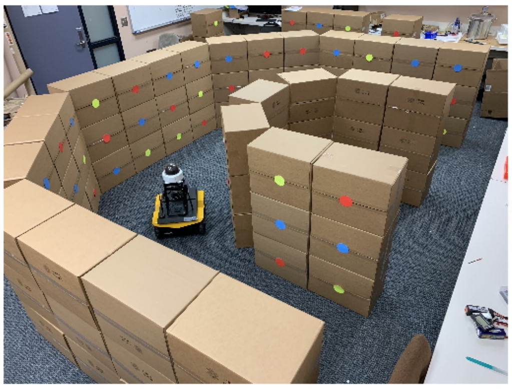


Figure 9.8: On the left the single wall environment created in the laboratory while, on the right, the trajectories followed by the Jackal robot: in blue the trajectory with $c = constant$ fixed a priori, in green the trajectory with $c = \tau(t_0)$ where t_0 is the first instant in which the single wall control law is applied.

to maintain a certain distance from the wall, as already highlighted in Gazebo simulations presented in Chapter 8. However, due to the sparsity of features in the real environment and to the fact that the single wall control law relies on a single TTT value, the results of the tests present some oscillations. Nevertheless, stability seems to be always guaranteed and zero crashes into the walls have been registered.

Finally, the most complicated test consists in performing a curve with a U shape, in this scenario both the control laws described in Chapter 6 and 7 have to be applied and this leads to test also the switching mechanism described in Chapter 8. Figure 9.9 shows the environment setup and the resulting trajectory of the mobile vehicle. The UGV is able to successfully perform the turn. Also in this scenario it



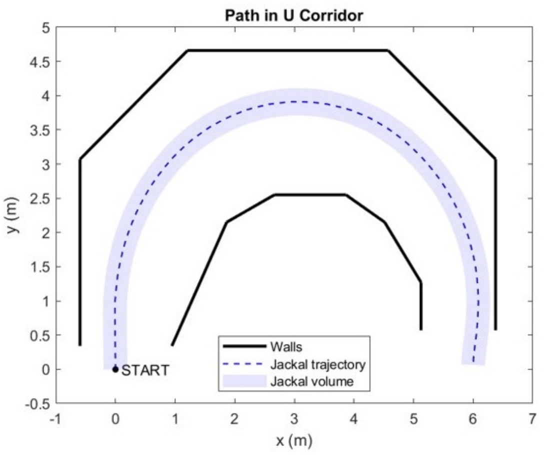


Figure 9.9: On the left the U turn created in the laboratory while, on the right, the trajectory followed by the Jackal robot.

can be noticed that at the end of the turning procedure, the path is too short to allow the vehicle to perfectly align to the center of the environment. Future work should focus on larger and more complex spaces.

In general, the similarity between real tests and the simulations presented in Chapter 8 confirms the validity of the choice of using Gazebo as a simulation platform.

Chapter 10

Conclusions and Future work

In this chapter, final considerations are made and an analysis of the advantages that the use of the developed algorithms can introduce is provided. Moreover, some remaining open questions are also presented together with possible improvements and related further work.

10.1 Conclusions

The algorithm developed in this work has been tested in simulation environments and on a real platform. Both the testing phases have produced good results, as discussed in Chapter 8 and Chapter 9, indeed the robot is able to safely navigate in unknown environments using only visual information coming from a monocular camera. Neither in simulations nor in real tests crashes were observed. The obtained results are a sign of the effectiveness of the control strategy implemented.

The developed algorithm presents several advantages, one of them is its simplicity that makes it easily implementable on a wide variety of robotic platforms for real-time applications. The control strategy can be useful to support navigation in different situations such as exploration of dangerous environments (e.g. tunnel inspections) or when GPS is not available or not sufficiently precise (e.g. in cluttered environments, underground). Moreover, it could be also employed when GPS signal is available but it is necessary to avoid unexpected obstacles during the navigation.

However, an open question for the implemented algorithm is represented by its possible behavior when moving objects are present in the environment since it has only been evaluated in static scenarios. This and others open questions will be addressed in future work.

10.2 Future work

The effectiveness of the control strategy implemented in this work has been shown and discussed in the previous chapters and in the previous section, however there are some additional considerations that can be made to improve the analysis of certain concepts. Possible improvements that can be done are now described.

From a theoretical point of view, it is necessary to perform a new stability analysis to find theoretical basis to justify the stability of the system when Sense-Perceive-Act cycles are introduced. Indeed, by observing simulation results it is possible to see that also in the case in which the cycles are present the system response remains stable. The only difference is that it is delayed and it requires more time to reach the steady-state value. For what regards the practical aspect, there will be the possibility of performing more tests in real scenarios thanks to the collaboration with the Boston University. The robotic platform that will be used is the Jackal UGV and this allows an exchange of code and the collection of more experimental results.

Moreover, one of the open question that the algorithm presents is related to a problem in the optical flow field because the optical flow vectors around the FOE are too small to obtain a good estimate of the time-to-transit. This leads to the impossibility of correctly detecting and avoiding obstacles right in front of the vehicle and, even if this is not a common situation, a strategy to manage it has to be implemented. A possible approach that could be investigated consists in recognizing changes in the optical flow field without estimating the time-to-transit coming from the central ROI and applying a little steering action (with sign that depends on the general optical flow field distribution) when a change in the central part of the image is detected. By doing so, the object to be avoided is no more centered in the image and the control strategy presented in this work should be able to avoid the crash.

In the control strategy developed the distribution of the optical flow field gives an idea of the geometry of the environment which is used to select the proper control action. In this case the optical flow field is computed at every sensing phase. The mobile robot does not have a memory of past experiences that would be useful to improve its navigation when moving in already visited scenarios and it is not able to predict future actions which is something that humans and animals always do. Predictive coding ([29]), which is a complex theory that suggests the brain is constantly generating and updating a model of the environment, can be considered to improve what is presented in this work. Deep learning ([30]), that is a powerful class of machine learning algorithms, could be considered to implement some predictive actions taking also in considerations past experiences to cope with the remaining open questions of the control strategy implemented.

The controller developed can be also adapted to other autonomous vehicles such as the aerial vehicles. In this case it is important to consider also the vertical displacement and a similar control strategy using TTT can be used by considering the values coming from the upper and lower part of the image.

Appendix A

Effect of feature density on control action

A brief discussion on how feature density can affect the control action will be provided in this Appendix. Let's start considering an environment like the one presented in Figure A.1. The geometric time-to-transit related to a single feature is

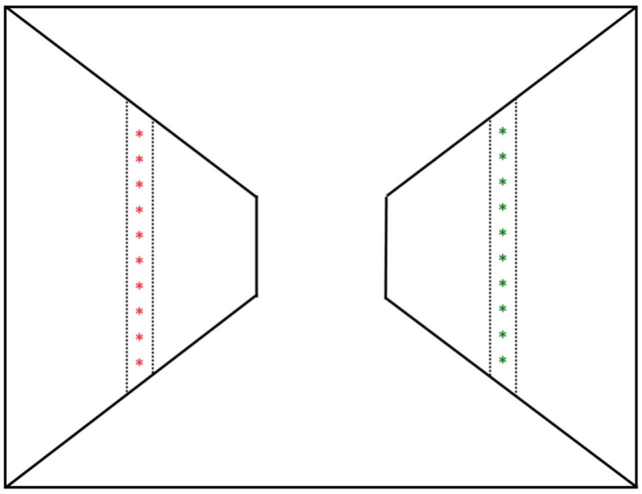


Figure A.1: A corridor composed of many strips each of them containing a fixed amount of features (identified by $*$).

estimated through the computation of the perceived TTT and this is possible once the sparse optical flow field of the image is retrieved. With respect to the geometric quantity, the perceive time-to-transit is negatively affected by many factors among which is how precisely the optical flow field is estimated. The τ associated to every feature will be considered as a random variable represented by a Normal Distribution with mean τ_0 which corresponds to the geometric time-to-transit value for the specific feature as depicted in Figure A.2. Since time-to-transit is a proxy

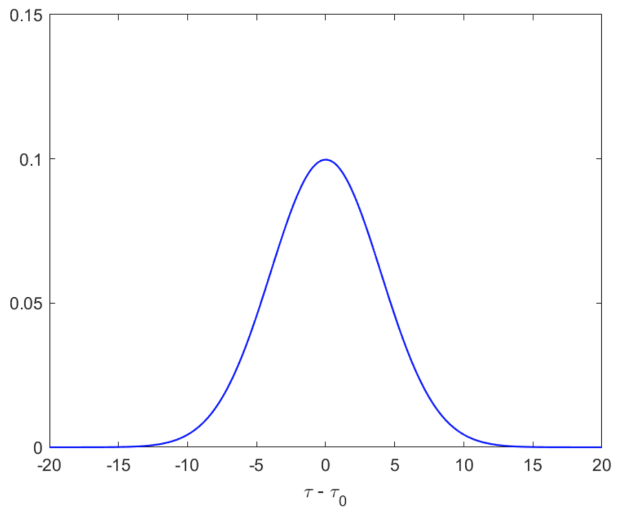


Figure A.2: Representation of the perceived time-to-transit as a Gaussian random variable with mean τ_0.

for distance, a vehicle equipped with a monocular camera to better estimate its distance from a wall can rely on more features computing an averaged TTT using all the features in a wall strip. To simplify the computations, the assumption that the left and the right strip in Figure A.1 contain both N features is considered. It is possible to define τ_l and τ_r as random variables with mean $\mu_0 = \tau_0$ but with variance proportional to N as depicted in Figure A.3. In mathematical therms, it

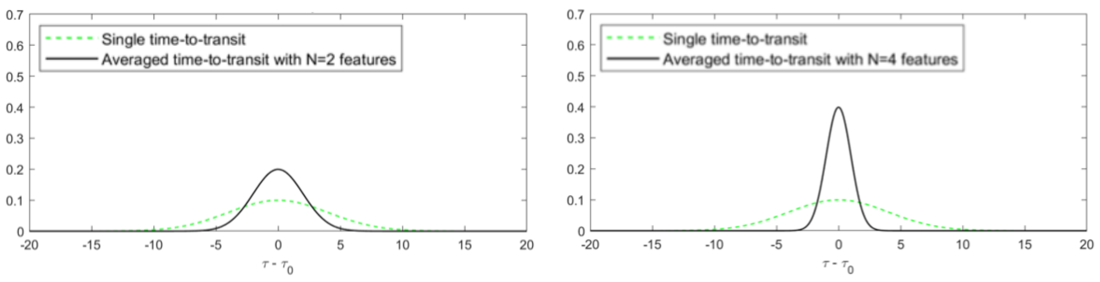


Figure A.3: Increasing the number of features in a wall strip will lead to a better estimation of τ_l and τ_r.

can be written that:

$$\tau_i \sim \mathcal{N}(\mu_0, \sigma_0^2) \qquad \tau_l \sim \mathcal{N}(\mu_0, \frac{1}{N}\sigma_0^2) \qquad \tau_r \sim \mathcal{N}(\mu_0, \frac{1}{N}\sigma_0^2)$$

At this point, it is important to take into account how the optical flow field is computed. In particular, features are matched frame by frame and the feature density can affect the ability to correctly track them. The characteristic of a matcher with respect to the feature density is not easy to determine but some considerations can be made. Suppose that similar features (with similar descriptors) are present in the same wall strip. It is reasonable to think that the bigger the number of features, the more difficult will be to distinguish one from another. Assuming $N_{max} = 100$ as the maximum number of features that can be contained in a wall strip, the probability to wrongly match a feature between two frames can be graphically represented as in Figure A.4[1]. No assumptions will be made on $\xi(N)$ in the development of the following computations except that $0 \leq \xi(N) \leq 1$ for every N.

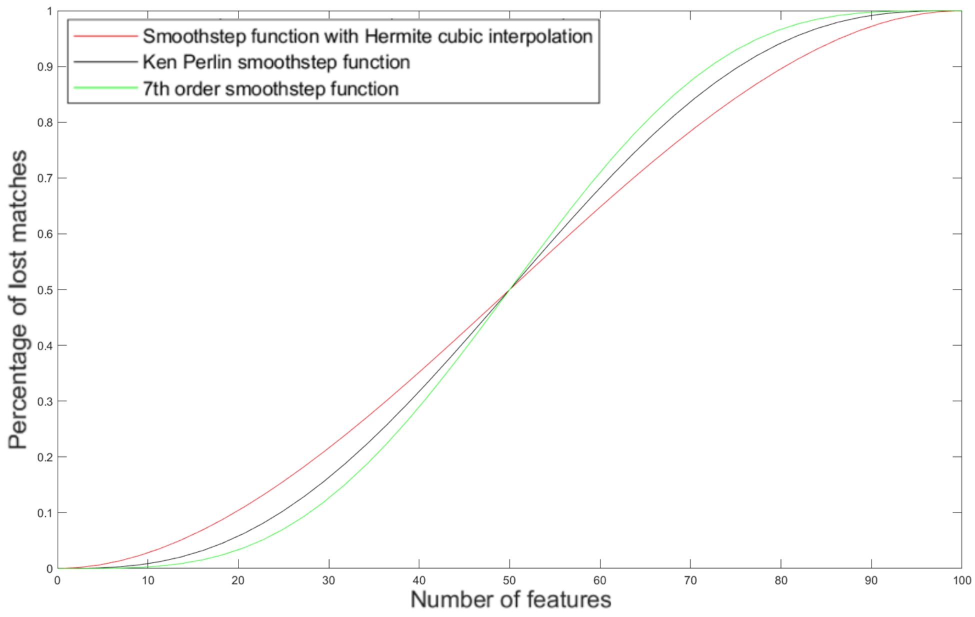


Figure A.4: Probability $\xi(N)$ of a wrong match with respect to the number of features in a wall strip described by different smooth-step functions.

Considering the *Tau Balancing* control law, it can be written that the effect of feature density on the control action when perceived time-to-transit instead of the

[1]More information about Ken Perlin smoothstep function can be found at `https://en.wikipedia.org/wiki/Smoothstep`

geometric TTT is used, is represented by the error $\epsilon_{Tb}(N)$:

$$\epsilon_{Tb}(N) = u^* - u = k(\tau_l^* - \tau_r^*) - k(\tau_l - \tau_r) = k(\tau_l^* - \tau_r^* - \tau_l + \tau_r)$$

and considering the worst case:

$$\epsilon_{Tb}(N) = k(\mu_l - \mu_r - (\mu_l \pm 3\sigma_l^2(N)) + (\mu_r \pm 3\sigma_r^2(N))) = k(3\sigma_l^2(N) + 3\sigma_r^2(N))$$

Since $\sigma_l^2 = \sigma_r^2 = \sigma$:

$$\epsilon_{Tb}(N) = 6k\sigma^2(N) = \frac{6k\sigma_0^2}{N}$$

This still does not take into account the effect of using a real matcher that cannot guarantee zero wrong matches. This can be modeled writing:

$$\epsilon_{Tb}(N) = 6k\sigma^2(N) = 6k\sigma_0^2 \frac{1}{N - N \cdot \xi(N)}$$

By knowing the probability of wrong matches with respect to the number of features in a single wall strip, it is possible to understand where the error is minimized:

$$\frac{d\epsilon_{Tb}(N)}{dN} = 6k\sigma_0^2 \frac{-1(1 - (\xi(N) + N \cdot \xi'(N))}{(N - N \cdot \xi(N))^2}$$

and since $0 < N < N_{max}$:

$$\frac{d\epsilon_{Tb}(N)}{dN} = 0 \quad \Rightarrow \quad \xi(N) + N \cdot \xi'(N) - 1 = 0$$

Similar results can be retrieved considering the *Single Wall* strategy. In particular:

$$\epsilon_{Sw}(N) = u^* - u = k(\tau^* - c) - k(\tau - c) = k(\tau^* - \tau)$$

and in the worst case:

$$\epsilon_{Sw}(N) = k(\mu - (\mu \pm 3\sigma^2(N))) = \frac{3k\sigma_0^2}{N}$$

Considering the probability of wrong matches, it is finally obtained that, in this case:

$$\epsilon_{Sw}(N) = 3k\sigma^2(N) = 3k\sigma_0^2 \frac{1}{N - N \cdot \xi(N)}$$

In Figure A.5, the behavior of $\epsilon(N)$ is plotted for different values of k and the same would happen changing the variance σ_0^2. The plot is obtained with $\xi(N)$ represented by a smooth-step function. As it can be noticed, the control error increases when k or σ_0^2 increases while the range of N that leads to an acceptable

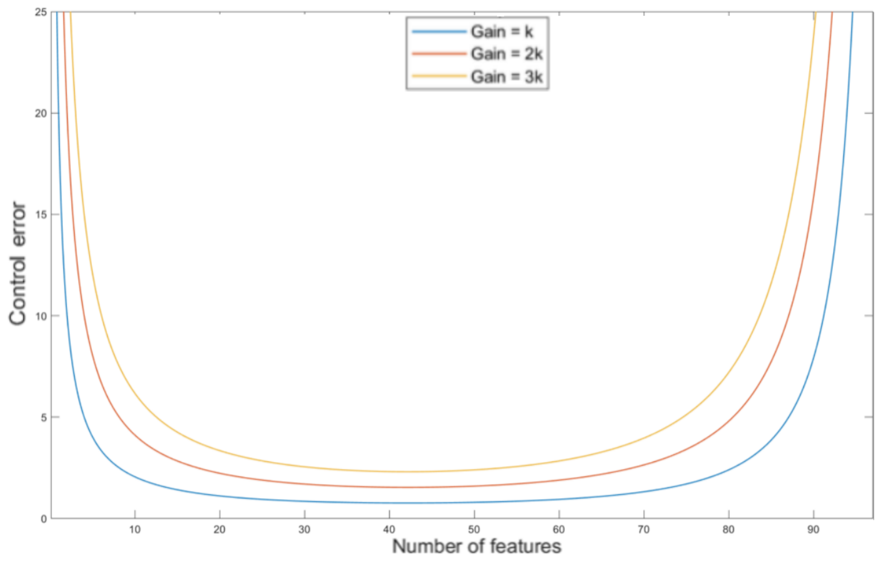


Figure A.5: The behavior of $\epsilon(N)$ for different values of k is shown.

control error tightens. It is clear that feature density is not something that can be chosen a priori in a real environment, but to understand how this can affect the control action can be useful. The ideal scenario would be the one in which the number of features does not worsen the effect of the control action. From a practical point of view this is not possible, but it is of paramount importance to know what can make visual navigation in unknown environments unfeasible. Surely it will be subject of a further work a detailed study about the performance of different types of matchers with respect to feature density. How $\epsilon(N)$ can vary when the function $\xi(N)$ changes is shown in Figure A.6. The control error remains the same for different $\xi(N)$ when N is sufficiently small while the range of N that leads to an acceptable control error changes.

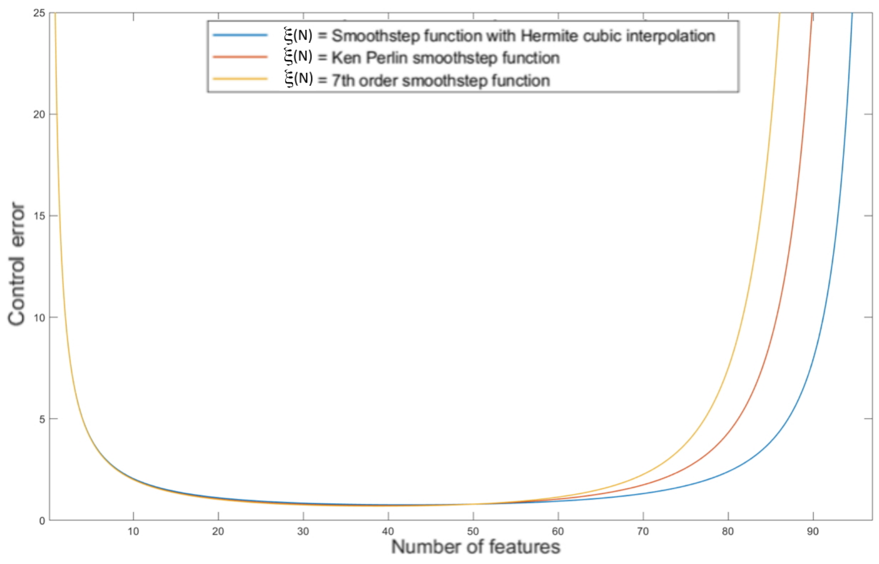


Figure A.6: The behavior of $\epsilon(N)$ for different $\xi(N)$ is shown.

Appendix B

Mathematical modeling of the Jackal UGV

In 2004, K. Kozłowski and D. Pazderski presented the kinematic and the dynamic model of a SSMR in [28]. A summary of what they found is now provided, considering the vehicle moving on a planar surface.

B.1 Kinematic model

Consider a robot on a plane surface, as depicted in Figure B.1, it is possible to define an inertial orthonormal basis (X_g, Y_g, Z_g) and a local coordinates frame (x_l, y_l, z_l) located at the center of mass (COM) of the robot. The coordinates of the COM in the inertial basis are (X, Y, Z) but, since only the plane motion is considered, $Z = constant$. The linear and the angular velocities in the local reference frame can be written as:

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix} \quad \boldsymbol{\omega} = \begin{bmatrix} 0 \\ 0 \\ \omega \end{bmatrix}$$

then, a state vector $\boldsymbol{q}$ which expresses the generalized coordinates of the robot can be defined as follows:

$$\boldsymbol{q} = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix}$$

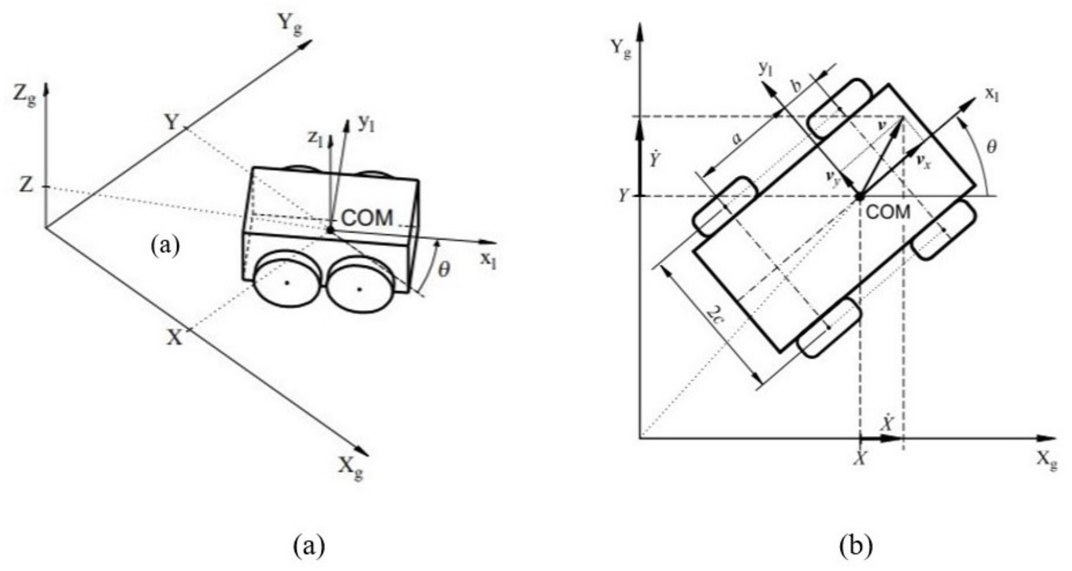


Figure B.1: The SSMR model in the Inertial basis. *Source: [28]*

where θ is the orientation of the local reference frame with respect to the inertial one. The $\boldsymbol{q}$ vector can be derived to obtain the vector of the generalized velocity:

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix}$$

By observing image (b) of Figure B.1, it can be noticed that the relation between the velocities of the COM in the inertial reference frame and its velocities in the local reference frame is expressed by:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{B.1}$$

and, because of planar motion, it possible to say that $\dot{\theta} = \omega$.

Equation B.1 describes the free-body kinematics of the Jackal, but it is necessary to analyze also the relationship between the wheels and the robot velocities to understand the restrictions to which the SSMR plane movement is subjected. Suppose that the wheel is in contact with the plane only in a point P_i, in this case $\omega_i(t)$ is the angular velocity of the i^{th} wheel. Moreover, the lateral velocity v_{iy} of the wheel has to be taken into account because it introduces the lateral skidding if the SSMR vehicle changes its orientation. The above considerations lead to the fact that the wheel i is tangent to the path only if the vehicle is moving on a straight line and so when $\omega = 0$.

In order to simplify the computations, a simple case in which the longitudinal slip between the wheels and the surface is neglected can be considered. In this situation the longitudinal component of the wheel velocity, expressed in the local reference frame, is the following:

$$v_{ix} = r_i \omega_i \tag{B.2}$$

where r_i is the effective rolling radius of the i^{th} wheel.

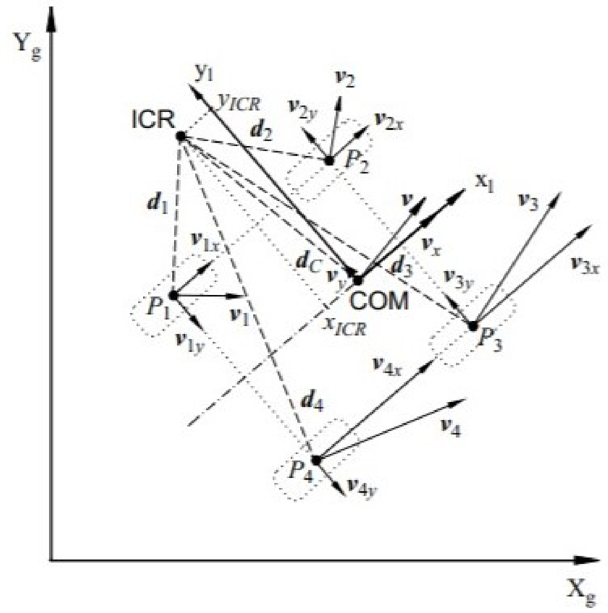


Figure B.2: Graphical representation of the Jackal's wheels velocities. *Source: [28]*

To develop a kinematic model, all the wheels have to be considered together. Looking at Figure B.2, two quantities can be defined: $\boldsymbol{d} = [d_{ix}, d_{iy}]^T$ and $\boldsymbol{d}_C = [d_{Cx}, d_{Cy}]^T$. These are the radius vectors of the i^{th} wheel and of the COM respectively, expressed with respect to the local reference frame from the instantaneous center of rotation (ICR). After some geometrical considerations, it is possible to state that:

$$\frac{v_{ix}}{-d_{iy}} = \frac{v_x}{-d_{Cy}} = \frac{v_{iy}}{d_{ix}} = \frac{v_y}{d_{Cx}} = \omega \tag{B.3}$$

Considering Figure B.2 and image (b) of Figure B.1, the following equations can be retrieved:

$$\begin{aligned}
d_{1y} &= d_{2y} = d_{Cy} + c \\
d_{3y} &= d_{4y} = d_{Cy} - c \\
d_{1x} &= d_{4x} = d_{Cx} - a \\
d_{2x} &= d_{3x} = d_{Cx} + b
\end{aligned} \tag{B.4}$$

and, by combining (B.3) and (B.4), the relationships between the wheels velocities are obtained:

$$\begin{aligned}
v_L &= v_{1x} = v_{2x} \\
v_R &= v_{3x} = v_{4x} \\
v_F &= v_{2y} = v_{3y} \\
v_B &= v_{1y} = v_{4y}
\end{aligned} \tag{B.5}$$

where v_L and v_R are the longitudinal velocities components of the left and right wheels, while v_F and v_B are the lateral coordinates of the velocities of the front and rear wheels.

An additional passage that can be made is to rewrite the equation (B.3) by introducing the definition of the coordinates of the ICR in the local frame ICR=$[x_{ICR}, y_{ICR}] = [-d_{xC}, -d_{yC}]$:

$$\frac{v_x}{y_{ICR}} = \frac{-v_y}{x_{ICR}} = \omega$$

It is possible to retrieve the relations between the velocities of the wheels and the robot velocity by looking at equations (B.2), (B.3) and (B.5):

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & -c \\ 1 & c \\ 0 & -x_{ICR} + b \\ 0 & -x_{ICR} - a \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}$$

$$\boldsymbol{\omega_w} = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \frac{1}{r} \begin{bmatrix} v_L \\ v_R \end{bmatrix}$$

In the second formula ω_L and ω_R are the angular velocities of the left and right wheels and $r = r_i$ for each wheel.

Finally, by combining the equations above, the approximated relations between the robot velocities and the wheel velocities can be defined as:

$$\boldsymbol{\eta} = \begin{bmatrix} v_x \\ \omega \end{bmatrix} = r \begin{bmatrix} \frac{w_L + w_R}{2} \\ \frac{-w_L + w_R}{2c} \end{bmatrix} \tag{B.6}$$

where $\boldsymbol{\eta}$ is a new control input introduced at kinematic level. The accuracy of equation (B.6) can be considered valid only if the longitudinal slip is not dominant.

Moreover, to ensure an high validity to the relation, the parameters c and r must be identified experimentally.

To complete the kinematic model of the SSMR, an additional constraint on the velocity has to be considered:

$$v_y + x_{ICR}\dot{\theta} = 0 \tag{B.7}$$

and, since this constraint is not integrable, it consists in a nonholonomic constraint that can be rewritten in Pfaffian form as follows:

$$\begin{bmatrix} -\sin\theta & \cos\theta & x_{ICR} \end{bmatrix} \begin{bmatrix} \dot{X} & \dot{Y} & \dot{\theta} \end{bmatrix}^T = \boldsymbol{A}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0}$$

by observing that $\dot{\boldsymbol{q}}$ is always in the null space of $\boldsymbol{A}$ it is possible to write the equation that describes the kinematic of the robot, which is a nonholonomic and underactuated system, as:

$$\dot{\boldsymbol{q}} = \boldsymbol{S}(\boldsymbol{q})\boldsymbol{\eta}$$

where,

$$\boldsymbol{S}(\boldsymbol{q})^T \boldsymbol{A}(\boldsymbol{q})^T = 0$$

and

$$\boldsymbol{S}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & x_{ICR}\sin\theta \\ \sin\theta & -x_{ICR}\cos\theta \\ 0 & 1 \end{bmatrix}$$

B.2 Dynamic model

For what regards the main theme of the work, the nonholonomic dynamics is of little importance but a brief introduction of it is provided for the sake of completeness. The dynamic model is computed by following the steps presented in [28]. To obtain the dynamic equation of a SSMR the Lagrange-Euler formula with Lagrange multipliers, to include the nonholonomic constraint (B.7), has to be considered.

Since only a planar motion is performed, the potential energy of the robot is zero and so it is possible to state that the Lagrangian of the system equals the kinetic energy:

$$L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\boldsymbol{q}, \dot{\boldsymbol{q}})$$

To simplify the calculations, the energy of the rotation wheels can be neglected and the mass distribution can be considered homogeneous. These assumptions allow to write the kinetic energy, after some mathematical computations, as follows:

$$E_k = T = \frac{1}{2}m(\dot{X}^2 + \dot{Y}^2) + \frac{1}{2}I\dot{\theta}^2$$

where m is the mass of the robot and I is its moment of inertia about the COM. The inertial forces can be obtained by calculating the partial derivative of the kinetic energy and then deriving again by the time:

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{\boldsymbol{q}}}\right) = \begin{bmatrix} m\ddot{X} \\ m\ddot{Y} \\ I\ddot{\theta} \end{bmatrix} = \boldsymbol{M}\ddot{\boldsymbol{q}} \tag{B.8}$$

where $\boldsymbol{M}$ is a diagonal matrix.

To define the generalized resistive forces the following vector can be introduced:

$$\boldsymbol{R}(\dot{\boldsymbol{q}}) = \begin{bmatrix} F_{rx}(\dot{\boldsymbol{q}}) & F_{ry}(\dot{\boldsymbol{q}}) & M_r(\dot{\boldsymbol{q}}) \end{bmatrix}^T \tag{B.9}$$

in which F_{rx} and F_{ry} are the forces that cause dissipation of energy expressed in the inertial frame, while M_r is the resistant moment around the COM. It is possible to define also a vector for the active forces:

$$\boldsymbol{F} = \begin{bmatrix} F_x & F_y & M \end{bmatrix}^T \tag{B.10}$$

in which F_x and F_y are the forces generated by the actuator, while M is the active torque around the COM. In Figure B.3 a graphical representation of the resistive and active forces is shown.

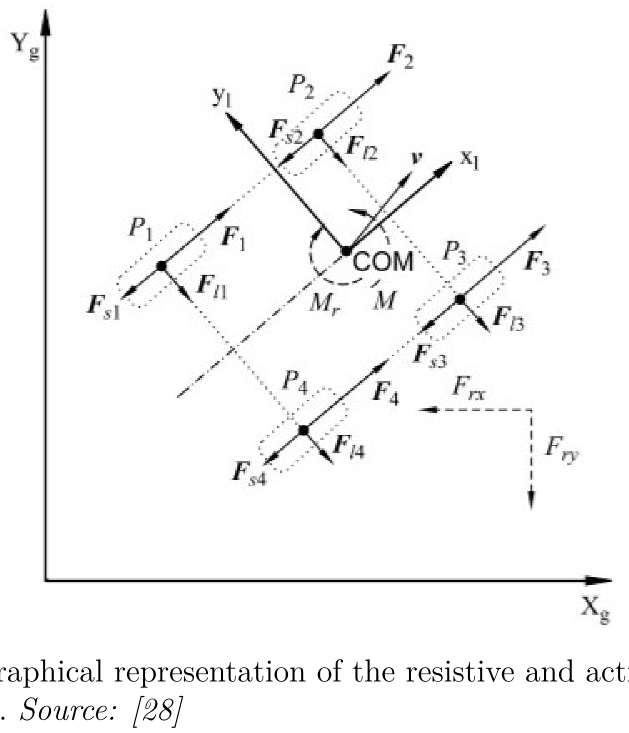


Figure B.3: Graphical representation of the resistive and active forces acting on the Jackal UGV. *Source: [28]*

After some mathematical computations and considering the explicit expressions of the vector components in (B.10), the generalize active forces can be rewritten as follows:

$$\boldsymbol{F} = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{\tau} \tag{B.11}$$

where,

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix} = \begin{bmatrix} \tau_1 + \tau_2 \\ \tau_3 + \tau_4 \end{bmatrix}$$

and

$$\boldsymbol{B}(\boldsymbol{q}) = \frac{1}{r} \begin{bmatrix} \cos\theta & \cos\theta \\ \sin\theta & \sin\theta \\ c & -c \end{bmatrix}$$

τ_L and τ_R are the torques produced by the wheels on the left and right sides of the robot, $\boldsymbol{B}$ is an input trasformation matrix, r is the radius of each wheel and c is a geometric parameter represented in Figure B.1.

The free-body dynamic equation can be found by combining equations (B.8), (B.9) and (B.11) and it can be expressed as follows:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{R}(\dot{\boldsymbol{q}}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{\tau} \tag{B.12}$$

and with the introduction of the Lagrange multiplier $\boldsymbol{\lambda}$ that takes into account the nonholonomic constraint, the complete dynamical model of a SSMR is the following:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{R}(\dot{\boldsymbol{q}}) = \boldsymbol{B}(\boldsymbol{q})\boldsymbol{\tau} + \boldsymbol{A}(\boldsymbol{q})^T\boldsymbol{\lambda}$$

Bibliography

[1] M. V. Srinivasan, R. J. D. Moore, S. Thurrowgood, D. Soccol, and D. Bland. «From biology to engineering : insect vision and applications to robotics». In: *Frontiers in Sensing: From Biology to Engineering.* Ed. by F. G. Barth, J. A. C. Humphrey, and M. V. Srinivasan. Vienna, AT: Springer, 2012 (cit. on pp. 2, 47).

[2] J. J. Gibson. «Visually controlled locomotion and visual orientation in animals». In: *British Journal of Psychology* 49 (Aug. 1958), pp. 182–194 (cit. on p. 2).

[3] D. N. Lee. «A theory of visual control of braking based on information about time-to-collision». In: *Perception* 5 (Dec. 1976), pp. 437–459 (cit. on pp. 2, 21).

[4] Y. Wang and B. J. Frost. «Time To Collision is Signalled by Neurons in the Nucleus Rotundus of Pigeons». In: *Nature* 356 (Mar. 1992), pp. 236–238 (cit. on pp. 2, 21).

[5] Z. Kong, K. Özcimder, N. Fuller, A. Greco, D. Theriault, Z. Wu, T. Kunz, M. Betke, and J. Baillieul. «Optical Flow Sensing and the Inverse Perception Problem for Flying Bats». In: *Proceeding of the 52nd IEEE Conference on Decision and Control (CDC).* Florence, Italy, Dec. 2013, pp. 1608–1615 (cit. on pp. 2, 55, 56).

[6] P. A. Shoemaker, A. M. Hyslop, and J. S. Humbert. «Optic Flow Estimation on Trajectories Generated by Bio-Inspired Closed-Loop Flight». In: *Biological Cybernetics* 104(4-5) (May 2011), pp. 339–350 (cit. on p. 2).

[7] K. Sebesta and J. Baillieul. «Animal-Inspired Agile Flight Using Optical Flow Sensing». In: *Proceeding of the 51st IEEE Conference on Decision and Control (CDC).* Maui, Hawaii, Dec. 2012, pp. 3727–3721 (cit. on pp. 2, 21, 47).

[8] J. Baillieul. «Perceptual Control with Large Feature and Actuator Networks». In: *Proceeding of the 58th IEEE Conference on Decision and Control (CDC).* Nice, France, Dec. 2019, pp. 3819–3826 (cit. on pp. 2, 47, 48, 50, 51, 53, 57).

[9] P. J. Seebacher. «Motion control using optical flow of sparse image features». Master Thesis. Boston: Boston University, 2015 (cit. on p. 3).

[10] L. Corvese. «Perceptual aliasing in vision-based robot navigation». Master Thesis. Boston: Boston University, 2018 (cit. on p. 3).

[11] D. Eckmeier, B. R. H. Geurten, D. Kress, M. Mertes, R. Kern, M. Egelhaaf, and et al. «Gaze Strategy in the Free Flying Zebra Finch (Taeniopygia guttata)». In: *PLOS One* 3(12) (Dec. 2008) (cit. on p. 3).

[12] J. Baillieul and F. Kang. «Visual Navigation with a 2-pixel Camera–Possibilities and Limitations». In: *IFAC 2020, Virtual World Congress.* Berlin, Germany. Extended version available at: `http://arxiv.org/abs/2103.00285`, July 2020 (cit. on pp. 3, 24, 63).

[13] H. D. Escobar-Alvarez, M. Ohradzansky, J. Keshavan, B. N. Ranganathan, and J. S. Humbert. «Bioinspired Approaches for Autonomous Small-Object Detection and Avoidance». In: *IEEE Transactions on Robotics* 35(5) (Oct. 2019), pp. 1220–1232 (cit. on p. 3).

[14] G. C. H. E. de Croon, C. De Wagter, and T. Seidl. «Enhancing optical-flow-based control by learning visual appearance cues for flying robots». In: *Nature Machine Intelligence* 3(1) (Jan. 2021) (cit. on p. 3).

[15] G. Gremillion, J. S. Humbert, and H. G. Krapp. «Bio-inspired modeling and implementation of the ocelli visual system of flying insects». In: *Biological Cybernetics* 108(6) (Sept. 2014), pp. 735–746 (cit. on p. 3).

[16] J. J. Gibson. *The perception of the visual world.* Boston: Hooughton Mifflin, 1950 (cit. on p. 7).

[17] B. K. Horn and B. G. Schunck. «Determining optical flow». In: *Technical Symposium East, International Society for Optics and Photonics* 181 (1981), pp. 319–331 (cit. on pp. 7, 9).

[18] C. G. Harris and M. Stephens. «A Combined Corner and Edge Detector». In: *Proceedings of the Alvey Vision Conference.* Manchester, UK, Sept. 1988, pp. 147–152 (cit. on pp. 7, 13).

[19] J. Shi and C. Tomasi. «Good features to track». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Manchester, UK, June 1994, pp. 593–600 (cit. on p. 7).

[20] B. D. Lucas and T. Kanade. «An iterative image registration technique with an application to stereo vision». In: *IJCAI'81: Proceedings of the 7th International Joint Conference on Artificial intelligence* 2 (Aug. 1981), pp. 674–679 (cit. on p. 10).

[21] D. Lowe. «Distinctive image features from scale-invariant keypoints». In: *International Journal of Computer Vision* 60(2) (2004), pp. 91–110 (cit. on p. 13).

[22] E. Rosten, R. Porter, and T. Drummond. «FASTER and better: A Machine Learning Approach to Corner Detection». In: *IEEE Transactions on Software Engineering* 32(1) (Jan. 2010), pp. 105–119 (cit. on p. 15).

[23] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. «BRIEF: Binary Robust Independent Elementary Features». In: *European Conference on Computer Vision-ECCV 2010*. Heraklion, Greece, Sept. 2010, pp. 778–792 (cit. on p. 16).

[24] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. «ORB: an efficient alternative to SIFT or SURF». In: *IEEE International Conference on Computer Vision (ICCV)*. Barcelona, Spain, Nov. 2011, pp. 2564–2571 (cit. on p. 17).

[25] A. Alani, R. Ortiz, and P. Vandergheynst. «FREAK:Fast Retina Keypoint». In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Providence, USA, June 2012, pp. 510–517 (cit. on pp. 18, 19).

[26] D. N. Lee and P. E. Reddish. «Plummeting gannets: a paradigm of ecological optics». In: *Nature* 293 (1981), pp. 293–294 (cit. on p. 21).

[27] E. C. Hildreth, H. B. Barlow, and H. C. Longuet-Higgins. «Recovering Heading for Visually Guided Navigation in the Presence of Self-Moving Objects». In: *Philosophical Transactions: Biological Sciences* 337(1281) (Sept. 1992), pp. 305–313 (cit. on p. 24).

[28] K. Kozłowski and D. Pazderski. «Modeling and control of a 4-wheel skid-steering mobile robot». In: *International Journal of Applied Mathematics and Computer Science* 14(4) (Jan. 2004), pp. 477–496 (cit. on pp. 79, 97–99, 101, 102).

[29] R. P. N. Rao and D. H. Ballard. «Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects». In: *Nature Neuroscience* 2(1) (Jan. 1999), pp. 79–87 (cit. on p. 88).

[30] Y. LeCun, Y. Bengio, and G. Hinton. «Deep learning». In: *Nature* 521(7553) (May 2015), pp. 436–444 (cit. on p. 88).