Examensarbete 20 poäng D-nivå

SEMI-AUTONOMOUS NAVIGATION OF A LEGGED ROBOT USING MONOCULAR VISION

Reg.kod: Oru-Te- EXA089- D111/05

Alessandro Rovetto och Francesco Scandelli

Örebro vårterminen 2005

Examiner: Lars Karlsson
Supervisor: Alessandro Saffiotti

Örebro universitet
Institutionen för teknik
701 82 Örebro

ÖREBRO UNIVERSITET

Örebro University
Department of technology
SE-701 82 Örebro, Sweden

Contents

Chapter 1

Introduction

1.1 Problem addressed: general

This thesis takes place in the general context of *rescue robotics*. In particular, it belongs to a larger project aimed at allowing Örebro University (Sweden) to join the future RoboCup Rescue competitions, see [37]. In this kind of competitions, the main goal is searching for simulated victims in unstructured environments that resemble disaster sites and are off-limits to human staff, see Fig. 1.2. The project concerns the development of a *mixed rescue team* composed of:

- a wheeled *mothership* robot;
- some Sony's *AIBOs* robots, referred as *scouts* (see Fig. 1.1);
- a *human operator*.

The motivation behind this configuration is that a heterogeneous team of a large robot, the *mothership*, and several smaller robots, the *AIBOs*, complement each other's strengths. In fact, a larger robot can carry more capable sensors, provide substantial computation, and, being wheeled, it is faster than the legged *scouts*. However, because of its size it is not able to travel everywhere in the environment. This is the case for the challenging environments one expects to find in the aftermath of a disaster, with fallen debris, collapsed walls, and cluttered floors. Smaller robots, while having less sensing, compensate by their ability to go in small and inaccessible places, small crevices etc.

In literature there is a work, [5], employing a mixed team with the same composition as above, but there are relevant differences concerning:

- *Operator interface.* One of the objectives of the general project to which thesis belong, is to provide the operator with an *"intelligent" interface*. This should contain, among other information like images

Figure 1.1: Sony's AIBO ERS-210

shot by the mothership, a series of windows showing, at the same time, stream of images related to what each AIBO is looking through its camera. These streams are supposed to be a sort of "background" processes that attract the attention of the operator, e.g., by highlighting a particular window, whenever a victim is recognized. Victim recognition is accomplished by image processing techniques that allow, for instance, to identify skin color, body motion, and other features related to possible victims in a rescue environment. In order to better evaluate the presence of victims, the operator can also examine a *panoramic image* obtained from different single images taken by a certain AIBO's camera. This permits to achieve a wider field of view than a simple image. In fact, an AIBO's picture presents a very low resolution and field of view, and it would be quite inadequate for victim localization purposes.

[5] only allows to consider one AIBO's simple image stream at the time, without any automatic victim identification.

- *Autonomy degree* of AIBOs. In [5], AIBO are simply *tele-operated*, whereas we deal with *semi-autonomous* (also called *tele-autonomous*) robots.

- *Overall coordination and supervision.* In [5], cooperation among robots is missing, whereas supervision is only provided by the operator. In our project robots are supposed to cooperate through a *distributed architecture* that allows them to cope with fundamental tasks like *self-localization.*

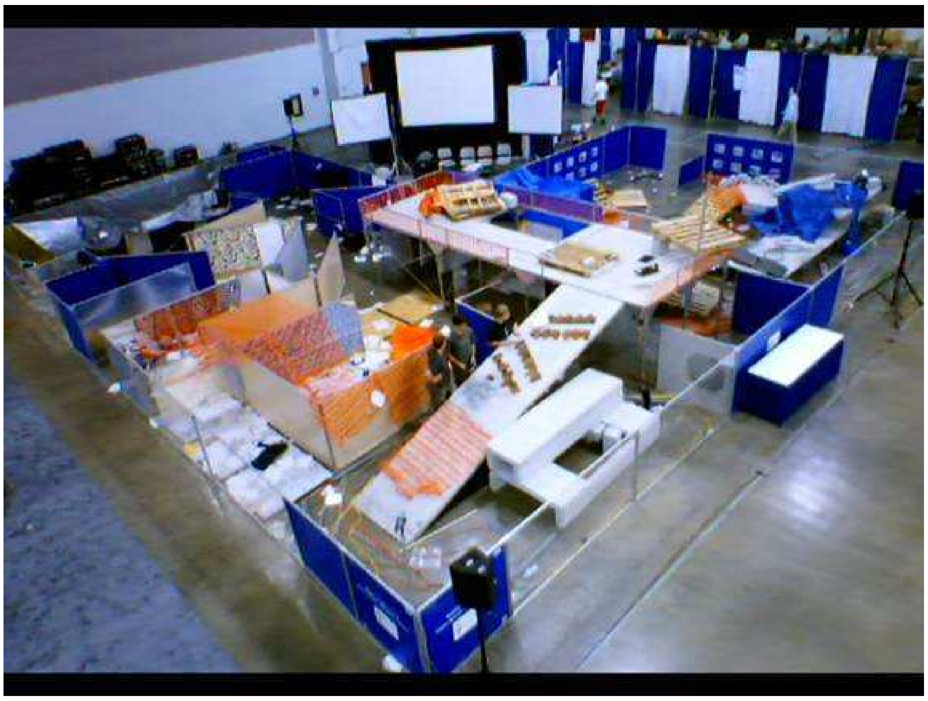


Figure 1.2: Rescue Robot League Arenas

1.2 Subproblem addressed: in this thesis

This thesis deals with *semi-autonomous* navigation of a *single* AIBO robot, moving on a *flat floor with several obstacles*. *Semi-autonomous* refers to the robot degree of autonomy. This means that the robot should implement cognitive functionalities that allow it to follow the general directions prescribed by the operator, by taking care of the low-level details like attitude control and *obstacle avoidance*. Moreover, the robot should provide *pre-interpreted sensor data* carrying rich information about the environment, thus simplifying the operator tasks of perceptual interpretation and situation assessment. In this way, the operator can focus on giving *abstract commands* to the robot, e.g., by clicking on a position in the environmental map where the robot should navigate.

The main mean through which our AIBO acquires information from the surrounding environment, is the *monocular vision*, accomplished by a camera placed in its head. Unfortunately, this camera provides poor images since *resolution and field of view are very limited*, thus restraining the robot perception of the environment. This is an important issue to be faced in order to cope with semi-autonomous navigation of an AIBO.

Another relevant topic concerns the own odometry information the robot exploits for *self-localization* purposes. In fact, AIBO, being a legged robot, does not provide a direct measure of its body displacement, as it happens, for instance, in the case of dead reckoning for wheeled robots. Moreover, computing the robot location starting from the measures provided by each of AIBO's twelve leg joint angle sensors, would be a very hard task since it would presuppose a precise knowledge about robot kinematics and dynamics. In addition, AIBO's sensors are not that accurate. Thus, achieving an effective method for AIBO self-localization is a challenging subject.

1.3 Outline of the approach

We frequently employ, throughout the thesis, the key concept of *mosaic image*. In fact, this is the principal input of all the vision-based tasks we implemented in our system. A *mosaic image* is a set of several simple images quickly taken by the AIBO's camera with the robot body standing still in a particular location of the environment. These images are shot from different pan/tilt orientations of the robot head so as to acquire a wider environment view than a simple image can permit.

The vision-based tasks using *mosaic images* constitute the solutions we propose to address the previous mentioned subproblem. In particular, we developed:

- An *interface* that allows the operator to access what the robot is "looking" at present, through a *panoramic image* obtained from the currently taken *mosaic image*.
- A *navigation system* based on an internal representation of the environment that, in our case, is a fuzzy occupancy grid. This map is constructed starting from obstacle range estimations that are obtained either applying an extended version of "visual sonar" algorithm (Cf. [16]) that makes use of *mosaic images*, or employing an obstacle detection algorithm designed by us. Whenever the operator indicates a goal environment location for the robot, a fuzzy path planner is invoked in order to find an efficient obstacle-free route. This planner exploits the well known A* algorithm. Actual execution of the path relies on a way-point technique and, at a lower level, on a proportional controller.
- A *self-localization* method based on the integration between a simple open-loop odometry already employed in [15], and a relative pose estimation obtained starting from different *mosaic images*. The latter method uses triangulation and landmark measurements.

1.4 Main contributions

The main original contributions of this thesis are:

1. *the overall framework*;
2. *visual sonar algorithm extended to mosaic images*;
3. *a new method for obstacle range estimation using mosaic images*;
4. *a new method for relative pose estimation based on mosaic images.*

Regarding the first contribution, we developed a stand-alone system for the wideranging task of semi-autonomous navigation of a robot in unknown environments.

Concerning the second contribution, this is the extension of an existing method for vision-based obstacle range estimation [16] to deal with mosaic images and unknown floor color.

The third contribution is an algorithm for obstacle range estimation which is alternative to the previous one.

The last contribution is a triangulation-based method to infer the robot ego-motion starting from corresponding features belonging to different mosaic images.

1.5 Outline of this thesis

The thesis is structured as follows:

Chapter 2 surveys different approaches suggested in literature to solve the problem addressed in this thesis.

Chapter 3 summarizes the principal components of our implemented system and the way they communicate. This chapter also provides some details about the construction of panoramic images.

Chapter 4 presents the "visual obstacle range finder" of our system. In particular, we discuss about the segmentation stage, the building of the calibration matrix, and the two methods employed for estimating obstacle positions.

Chapter 5 deals with the actual robot navigation in the environment. We describe the building of the fuzzy occupancy grid, the path planner, and the path execution stage.

Chapter 6 details the robot self-localization process, focusing on the exploitation of mosaic images. We first discuss a method for estimating the robot ego-motion starting from corresponding features belonging to two different mosaic images. Then, we present two implemented solutions for feature selection and matching within mosaic images: the first one is operator-based, whereas the second one is automatic.

Chapter 7 is devoted to the experimental verification of the calibration matrix we have obtained, the two methods for obstacle detection, and the visual self-localization technique. At the end, we also present a qualitative experiment concerning the whole navigation system.

Chapter 8 summarizes the principal achievements of this work, points out the most serious limitations, and suggests a few directions for future work.

Chapter 2

Related work

Related work can be characterized along four relevant dimensions: search and rescue robotics, visual sensors for monocular range estimation, collision-free navigation in unknown environments, and visual self-localization employing monocular sensor.

2.1 Search and rescue robotics

2.1.1 Overview

Robot platforms could potentially be used for a variety of tasks in search and rescue operations, including: delivery of food and medicals to buried survivors, rubble removal, victim transportation, and inspection of voids in the rubble pile. Until today, most robotic platforms have been used for the last task, exploring voids which are not accessible to humans because of their size or because of the extreme danger. [26] reports the result of a survey study about robotic platforms to be used in rescue and emergency response operations.

The first and most famous search and rescue operation to date has been the use of several robots to explore the rubble pile at ground zero at the World Trade Center after the September 11 event. The operation was carried out by the Center for Robot Assisted Search and Rescue (CRASAR) at the University of South Florida, FL, USA, under the direction of Prof. Robin Murphy (see http://crasar.csee.usf.edu).

According to [26], the critical aspect that distinguishes different robot platforms to be used for search and rescue operations is their size, since this determines the type of void that can be explored. The following classification has been proposed by Prof. Murphy:

Robot size	Type of void to explore
Man packable (micro)	sub-human (pipes, ventilation holes)
Man packable (mini)	confined space (void in muck pile)
Man portable	semi-structured (partially collapsed building)

Concerning the forms of locomotion for rescue platforms, there are six main categories: wheeled, tracked, legged, airborne, serpentine, and hybrid configurations. The large majority of existing platforms uses wheeled or tracked locomotion, including almost all of the commercially available platforms. The other kinds of locomotion are usually only employed in research laboratories. It should be noted that legged locomotion, to which Sony's AIBOs belong, is potentially the most versatile form of locomotion, e.g., it allows to negotiate steps and stairs, but it is slower, more fragile, and much harder to control than wheeled and tracked systems.

2.1.2 Modes of operation

Robotic platforms for rescue operations differ in the balance between need for tight operator control and ability of the platform to perform autonomously. The robot can operate under different "degrees of autonomy", ranging from pure tele-operation by part of the operator to full autonomous operation by part of the robot. [26] considers the following *modes of operation*:

- *Tele-operation*: the operator has full control. The robot can be seen only as a set of physical *actuators* and *sensors*;
- *Tele-autonomy*: the operator gives abstract-level commands and has access to pre-interpreted sensor data. This means the robot implements sensori-motor processes for *sensing and perception*, and for *motion control*.
- *Shared autonomy*: the robot and the operator are seen as on-pair partners in performing a joint task. In fact, both have *world modeling*, *planning and deliberation*, *sensing and perception*, and *motion control* capabilities.
- *Full autonomy*: the robot performs the task entirely by its own, without any human assistance.

Our navigation system can take place somewhere in between the extremes of the above classification. In particular, it belongs to the *tele-autonomy* class. In fact, our AIBO is able to autonomously reach environment locations indicated by the operator on the base of abstract and rich information about the environment, e.g., a map, provided by the robot itself. This kind of information simplifies the operator tasks of perceptual interpretation and situation assessment, thus releasing he/she from low-level navigation concerns, like obstacle avoidance, and allowing he/she to concentrate more on higher-level cognitive processes like the general exploration strategy of the environment.

A recent example of *tele-autonomy* is [12]. In this work various types of rescue robots, including multi-legged robots, are easily operated by the same interface, which is handled by a non-professional operator, i.e., without any prior training. This can be accomplished because the interface is the same as that of usual vehicles, e.g., cars, thus allowing the operator to operate the robots by employing the experience of daily life. The operator is provided with pre-interpreted sensor data which results, for instance, in force feedback, short informative texts, figures, and sounds. Abstract commands generated by the operator are then "converted", within each robot, in suitable control signals through a "semi-autonomous controller".

2.1.3 RoboCup Rescue

Overview

RoboCup Rescue and the related *Urban Search and Rescue (USAR) Robot Competitions* (see [36]) are among the most important academical activities in the research field of *search and rescue* robotics. The goal of USAR robot competitions is to increase awareness of the challenges involved in search and rescue applications, provide objective evaluation of robotic implementations in representative environments, and promote collaboration between researchers. It requires robots to demonstrate their capabilities in mobility, sensory perception, planning, mapping, and practical operator interfaces, while searching for simulated victims in unstructured environments, which are off-limits to human operators. As robot teams begin demonstrating repeated successes against the obstacles posed in the test-arenas, the level of difficulty will be increased accordingly so that the arenas provide a stepping-stone from the laboratory to the real world. Meanwhile, the yearly competitions will provide direct comparison of robotic approaches, objective performance evaluation, and a public proving ground for field-able robotic systems that will ultimately be used to save lives.

Examples

In the following, we briefly present the projects that were placed respectively in the first three positions at the last "RoboCupRescue - World Championship" (Lisbon, Portugal, 2004).

[13] describes a tracked robot with six crawlers whose configuration is variable depending on the terrain structure, thus allowing the robot to operate in a wide variety of situations. The robot is also endowed with a sensor head carried on a 5 D.O.F. manipulator. The sensor head provide binocular

vision and a set of rich specific sensors for victim identification, e.g., microphones and thermometers. The robot is *tele-operated* by two operators via two joysticks, one for the crawlers and one for the head manipulator. The operators evaluate, respectively, the images received from two cameras attached to the crawler structure, and from the head binocular camera. A 3D map building through laser scanner, is also performed for robot localization purposes and for allowing human rescue staff to find a route to the identified victims.

[33] makes use of a wheeled robot equipped with a laser scanner, two cameras and a microphone. All these sensor data are presented to the operator (the laser scanner allows a 3D-scene reconstruction) that *tele-operates* the robot via joystick. The operator is also capable to mark the location of victims, which he perceives on the screen, within the image.

[4] provides a *full-autonomous* robotic platform for victim identification. This is achieved through a strong interaction between visual cognitive inference (camera is the main sensor employed for victim recognition) and actions executed. Mapping, vision and navigation are all collaborative agents that work together sharing data to accomplish to complex identification and mapping tasks. The visual process reacts to interesting features in the arena trying to approach the focused area in order to achieve more successful observations from the environment. A reactive planner subsystem includes all the functionalities which are needed for accomplishing the role of coordinating and controlling the tasks mentioned above. The hardware architecture is constructed over a wheeled robot, endowed with 8 sonars and inertial sensors.

Finally, we discuss a work involving AIBOs.

Georgia Institute of Technology joined the USAR competition with a marsupial team consisting of a larger wheeled robot and several small legged AIBO robots, carried around by the larger robot (see [5]). The choice of an heterogeneous team was motivated, as in the case of our general problem (see Sec. 1.1), by the observation that a large robot and several smaller robots compliment each other's strengths.

AIBOs are *tele-operated* since an operator sends to each of them basic commands like "go left", "go forward", etc., on the base of environment images provided by the dogs. In this way, the human operator is in charge of: the perceptual interpretation of sensor data, the formation of plans and decisions as to the next actions to be performed by the robot. For these reasons, tele-operating a robot is a tough task, complicated by the limited view from the video camera. In fact, the human operator can incur loss of situational awareness, poor depth judgment, and failure to detect obstacles.

2.2 Visual sensors for monocular range estimation

In literature, there exists, among the others, two importants approaches for monocular range estimation of objects belonging to the environment: *visual looming* [23] and *visual sonar* [16].

Visual looming The looming algorithm is based on the simple fact that objects appear larger to a camera as they get closer, and smaller as they move away. Suppose a mobile robot endowed with a camera is located at distance d_0 from a stationary object; then the robot moves toward the object, reaching a distance d_1 from it. [23] shows that d_0 and d_1 can be calculated from $\Delta d = d_1 - d_0$, the robot displacement, and from p_0 and p_1, the size of the image projection of the same object at distances d_0 and d_1, respectively. This is referred as *looming equation.*

Note that it is irrelevant whether the displacement is the result of camera movements or object movement, and whether the motion is toward or away from the object. The looming equation can be shown to hold under such conditions that are reasonable for mobile robots, but, in order to find distances, a measure of the displacement is needed, e.g., through proprioceptive odometry.

Unfortunately, AIBO, being a legged robot, does not provide a direct measure of its body displacement, and computing the body displacement starting from each of its 12 leg joint angle sensors, would be a very hard task since it would presuppose a precise knowledge about dog kinematics and dynamics. Moreover, AIBO's joint sensors are not that accurate.

Visual sonar This vision algorithm behaves similarly to a sonar sensor. The vision processing consists of several discrete stages. The first stage takes the raw camera image ([16] employs Sony's AIBOs) and classifies, i.e., segments, each pixel into one of several color classes. Pixels corresponding to the floor are classified into the "floor" class. Pixels of other colors are classified into either one of the color classes for various a priori-known objects or into the unknown class for general obstacles. Scan lines are drawn in the segmented image that correspond to lines on the ground plane emanating from the reference point for the robot. [16] used scan lines spaced every 5 degrees around the robot. Objects are located along each scan line and identified if possible. Distance to objects is finally calculated by intersecting rays through the closest pixels of the object on the image onto the ground plane the robot is standing on.

2.3 Obstacle-free navigation in unknown environments

The problem of obstacle-free autonomous navigation in a completely unknown environment, has been addressed by several works. Most techniques can be classified into two major streams: *reactive* and *deliberative* navigation.

In reactive methods, there is a stimulus response relationship between sensors and actuators, with very limited or no *world modeling* at all (for instance, Cf. [2]). In deliberative techniques, a world model is used to formulate plans to which the robot is more or less committed;While reactive navigation proves to be flexible by virtue of its modular design approach (see Subsec. 2.3.3), it may fail when confronted with difficult tasks. On the other hand, deliberative navigation suffers from high computational requirements and performance degradation in dynamic environments. Based on the idea that dynamically acquired world models can be used to avoid certain pitfalls that representationless methods are subject to, a number of *mixed solutions* have been proposed, aimed at an efficient integration of world modeling and planning into reactive architectures Our system navigation draws inspiration from [21], which is somewhat in the line of mixed methods. In fact, it prescribes the incremental building of a dynamic world representation and the formulation of plans in accordance.

In the following subsections, we will first focus on navigation approaches based on an internal representation of robot's environment, and then we will shortly discuss reactive approaches.

2.3.1 World-model-based approaches

In this subsection we discuss some ways of *representing space* and *planning* obstacle-free paths over these representations. Finally, we briefly describe a *fuzzy* approach to navigation that deals with uncertainty management.

Representing space

In general spatial representation can be divided into two main groups: those that rely primarily on an underlying *metric representation*, e.g., spatial decomposition and geometric representations, and those that are *topological* (Cf. [7]). In our case, a metric representation is more suitable since we deal with distance measures of obstacle lying on the ground, hence in the following we will focus on the most common metric representation of a 2D world: *occupancy grid*.

Occupancy grid An occupancy grid is obtained by *sampling* discretely the two-dimensional environment to be described. The idea is to represent the *space itself* as opposed to representing individual objects within it. This precludes having to discriminate or identify individual objects.

This sampling can be performed in various ways using a number of different subdivision methods based on the shapes of the objects or, more commonly, by defining a sampling lattice embedded in space and sampling space at the nodes so defined. The simplest method is to sample space at the cells of a uniform grid. Samples taken at points in the lattice express the *degree of occupancy* at that sample point.

The main disadvantage of a *regular lattice representation* is that the grid resolution is limited by the cell size and the representation is storage intensive even if much of the environment is empty or occupied.

One alternative is to take advantage of the fact that many of the cells will have a similar state, especially those cells that correspond to spatially adjacent regions. From this point of view, two general approaches have been developed: one is to represent space using *cells of a nonuniform shape and size*, but more commonly a *recursive hierarchical representation* is used. The most common example belonging to the latter approach is the *quadtree.*

A *quadtree* [27] is a recursive data structure for representing a square two-dimensional region. It is a hierarchical representation that can potentially reduce storage. Begin with a large square region that encompasses all of the necessary space. Cells that are neither uniformly empty or full are subdivided into four equal subparts. Subparts are subdivided in turn until either they are uniformly empty or full until an a priori resolution limit is met.

Hierarchical representation systems based on a power of two decomposition, e.g., quadtree, are not suitable when the space is not well characterized by a power of two representation. Two alternative spatial decompositions that are not quite as restrictive as quadtree representations are *binary space partitioning trees* (BSP trees) and the *exact decomposition* method (Cf. [7] for more details).

2.3.2 Path planning

The general path-planning problem is to find a path t, which from some initial state A leads the robot to the goal position, trying to minimize the *cost* related to a route linking A to the goal, e.g., the length of the path. A significant literature on path planning exists ([14] provides a wide survey). Algorithm are constructed based on different theoretical assumptions and requirements concerning the following:

1. *Environment and robot.* The structure of the environment, the robot's capabilities, its shape, and so forth.

2. *Soundness.* Is the planned trajectory guaranteed to be collision free?

3. *Completeness.* Are the algorithms guaranteed to find a path, if one exists?

4. *Optimality.* The cost of the actual path obtained versus the optimal path.

5. *Space or time complexity.* The storage space or computer time taken to find solution.

To render the planning problem tractable it is often necessary to make a variety of *simplifications* with respect to the real environment. After an algorithm has been developed based on some set of assumptions, it must actually run in the real world. Idealized algorithms for path planning must be augmented to deal with many annoying realities, e.g., domain *uncertainty*, when applied in the field.

Searching a discrete state space

Search algorithms form a fundamental component of many robot path-planning algorithms. Given a discrete state space (that is a set of possible problem states obtained, for instance, through spatial decompositions discussed previously), and a state transition function to determine the states directly reachable from any given state, a *search method* is an algorithm to control the exploration of the state space in order to identify a path from some initial state to the goal.

Graph search is the most spread class of search methods. A large literature exists on this kind of searching (for instance, Cf. [22]) that is usually distinguished in *informed* and *uninformed* methods, depending on whether there is specific knowledge (encoded by an *heuristic function* $h(n)$, which estimates the cost of the cheapest path from node n to the goal) about the search space other then its simply definition, or not. The most widely-known form of informed search is called A^* search, which is optimal and complete, provided that h is *admissible*, i.e., it never overestimates the cost to reach the goal.

Considering unknown environments

One major flaw with many classical path planners is the assumption that the environment is known in advance. If the world model is inaccurate, then at some point during the executing of the plan the robot may encounter an event that makes the plan being executed invalid, and the robot must re-plan. Thus, the initial plan has been wasted.

Methods that deal with the need to *re-plan* and can reevaluate the path as it is being executed are known as *on-line algorithms*, and the trajectory they produce is sometimes referred as a *conditional plan.* A true on-line algorithm should be able to generate a preliminary trajectory even in the complete absence of any map. The *bug algorithm* [17] is an example of a simple on-line algorithm for path planning, but it suffers of a too strict assumption: it considers a robot with a perfect odometry.

A fuzzy navigation example

In selecting an appropriate world model, one must face the fact that the sensing process of a robotic system may behave in an inaccurate way sometimes. Therefore, rather than reconstruct a deterministic model of the environment, an uncertain map could be chosen. [21] proposes to define the map as a *fuzzy set*: a real number is associated to each point, quantifying the possibility that it belongs to an obstacle. The resulting representation is similar to an occupancy map. In [21] navigation is then accomplished by executing a path that has been computed on the fuzzy map through $A*$, which aims at minimizing the *risk* of collision along a path and its length.

2.3.3 Reactive approaches

The reactive paradigm is based on animal models of intelligence: the overall action of the robot is decomposed by *behaviour*, i.e., a mapping of sensory inputs to a pattern of motor actions, rather than by a deliberative reasoning process. Common to most reactive systems is that goals may not be represented explicitly, but rather they are encoded by the individual behaviors that operate within the controller, that is that module in charge of providing low-level control of the robot. Overall system behaviour emerges from the interactions that take place between the individual behaviors, sensor readings, and the world.

Subsumption control systems These are the best known reactive control architectures, first introduced by R. A. Brooks. They consist of a number of behavior modules arranged in a hierarchy of layers of competence. Different layers in the architecture take care of different behaviors. Lower layers control the most basic behaviors of the device, e.g., obstacle avoidance, whereas the higher behaviors modules control more advanced functions. Each level of competence includes all earlier levels. A subsumption architecture consists of a set of independent behaviors that directly map sensation to action. The behaviors are organized in a static hierarchy in which lower levels of competence have an implicit priority over higher ones.

The final emergent behavior of the system is the set of reactions that emerge from the various modules provided to achieve each of these levels of competencies. The incremental way in which subsumption architectures are constructed makes them quite straightforward to implement and allows for a great deal of experimentation and flexibility in terms of system design. However, given the random nature in which the conditions that trigger the various individual behaviors are met, the behavior that emerges from subsumption architecture can be very difficult to judge a priori. Moreover, debugging a subsumption architecture can be problematic.

2.4 Visual self-localization employing monocular sensor

We refer to *visual self-localization* as the problem of localizing the robot in the environment by exploiting the images taken by its camera. Robot visual positioning can be carried out either in an *absolute* way, i.e., w.r.t. a global framework attached to the environment, or in a *relative* way, i.e., w.r.t. to the previous robot pose estimation.

There are many approaches to the problem and we briefly present some of them in the following.

Landmark measurement This is based on the solution of geometric or trigonometric problems involving constraints on the position of environmental *landmarks* that are visible through the camera. A key issue in practice is whether the landmarks to be used are *artificial* or *natural*. Artificial landmarks emplaced specifically for the purposes of robot localization are typically much easier to detect, since they are chosen to be highly visible, and can be uniquely labeled (that is, their individual identities are known). Naturally occurring landmarks, on the other hand, preclude having to modify the environment, but their stable and robust detection can be a major issue, also because it can require highly complex image processing stages.

Within landmark measurement, *triangulation* refers to the solution of constraint equations relating the *absolute* pose of an observer to the positions of a set of landmarks. The simplest and most familiar case that gives this technique its name is that using *bearings* or *distance measurements* to two (or more) landmarks to solve a planar positioning task, thus solving for the parameters of a triangle given a combination of sides and angles.

Considering the domain of robotic soccer, [11] and [35] provide examples of triangulation-based self-localization methods that also handle *uncertainty* related to visual sensor measurements. [11] deals with natural landmarks and fuzzy logic whereas [35] employs both artificial and natural landmarks and makes use of a probabilistic Gaussian approach.

Camera ego-emotion estimation based on epipolar geometry Camera ego-motion estimation of a calibrated camera deals with the case of unknown movement of the robot camera, where rotation R and translation t need to be learned from point correspondences between two images (see [29]). The kind of geometry underlying this method is the same employed in *stereo vision*: epipolar geometry.

[9] proposes an algorithm for ego-motion estimation based on the computation of the *fundamental matrix*, that represents the mapping between correspondent points.

The main drawback of this class of methods is related to the *correspondence problem*, i.e., stating the correspondence between a set of points belonging to an image and the related set in a second image. The trouble is that the correspondence problem is inherently *ambiguous*. For instance, an unavoidable difficulty in searching for corresponding points is the *self-occlusion* problem, which occurs in images of non-convex objects: some points that are visible in the first image are not visible in the second one and vice versa.

Optical flow If we take a series of images in time, and the camera is moving, useful information about camera ego-motion can be obtained by analyzing and understanding the difference between images caused by the motion. From an image sequence, a function called *optical flow* is calculated: for every pixel, a velocity vector v is found which states how quickly that pixel is moving across the image, and the direction it is moving.

[1] shows that the problem of ego-motion determination, starting from optical flow, has been well addressed in literature.

Unfortunately, optical flow is hardly computed if significant change occurs between two consecutive images. This is the reason why optical flow is not suitable in our system. In fact, our visual self-localization process (see Ch. 6) employs a mosaic image made of six robot camera pictures as basic input for its computations, and this limits the rate at which the visual self-localization can be executed, because it takes a certain time for the dog to shoot the six pictures composing the mosaic and send them to the host (approximately, 10 seconds). In order to get small differences between consecutive mosaic images, we would need to stop the robot for grabbing pictures after each small step, but this would lead to an unfeasible system since it would take too much for the robot to move somewhere.

Nongeometric methods: perceptual structure These methods directly relate *appearance* or perceptual structure to pose. In this way, they do not need to invert visual sensor data to make geometric inferences, e.g., as in landmark measurement or in stereo vision-based techniques. The premise is that a mapping between sensor data and pose can be constructed *directly* without any intermediate representation based explicitly on 2D or 3D scene geometry.

Interpolation between sample images has been used for pose estimation and trajectory following, using a principal components encoding of the input data [19]. By creating a direct mapping for a low-dimensional subspace of the input images (obtained by eigenvector and eigenvalue analysis) to robot joint angles, the robot can use vision to track a prerecorded trajectory.

Other work has achieved accurate pose estimate by using sample images from either sonar or vision that sample the environment [6]. Images are first encoded as edge maps and these are then described by vectors of statistical descriptors. These input vectors are then used to train a neural network that maps between input images and output pose estimates.

These methods result to be quite unsuitable for our purposes mainly because they often are based on sample images that we cannot provide since our domain is initially unknown.

Chapter 3

System structure

The purpose of this chapter is providing a comprehensive idea of the semi-autonomous navigation system we have developed.

We start giving an brief description of the whole system (section 3.1), then we discuss about the software modules running on the robot (3.2), on the host computer (3.3, 3.4), and finally we analyze the mechanisms underlying the communication among these modules (3.5).

3.1 Overall framework

From an abstract point of view, our system can be seen as a set of coarse-grained *functionalities* organized in a two-layer architecture (see Fig. 3.1). In these architecture, the top layer implements higher cognitive processes for world *modeling* and for *planning*. The bottom layer implements sensori-motor processes for *sensing and perception*, and for *motion control*, which are connected to a set of robot *sensors* and *actuators*.

Moving to a more detailed description, our navigation system is actually a cyclic process whose main functional components are listed below:

- a *user interface* that allows the *operator* to:
  1. see what the dog is actually looking through its camera;
  2. decide which position of the environment the robot is supposed to go.
- A unit in charge of establishing an *obstacle-free path* from the robot current position to the one selected by the operator. This task relies on an *internal representation of the environment*;
- A module that actually make the robot *execute the above path*;
- An interface to the *sensori-motoric functionalities* of the robot. In particular, these functionalities concern:

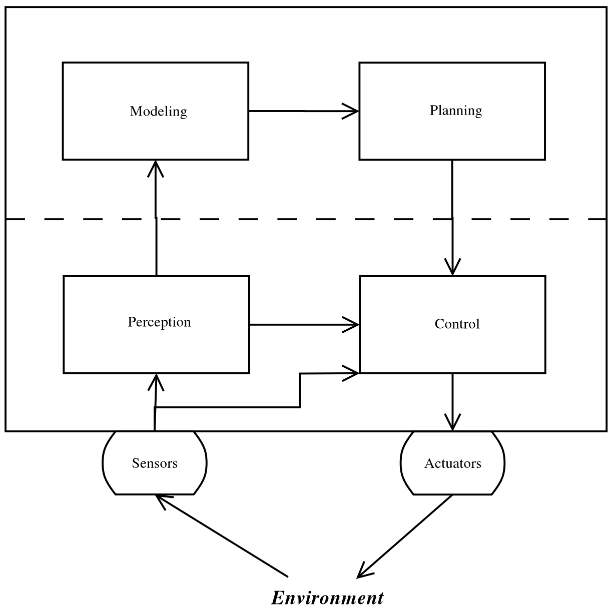


Figure 3.1: Abstract diagram of our navigation system. The top layer implements higher cognitive processes (world *modeling* and *planning*), whereas the bottom layer implements sensori-motor processes for *sensing and perception*, and for *motion control*, which are connected to a set of *sensors* and *actuators*. The arrows indicate the data flow. Note that there are two outgoing connections in "Sensors": one is directed into "Perception" and deals with the visual sensor, whereas the other one goes directly into "Control" and it deals with joint sensors.

1. translating velocity locomotion commands to an appropriate walking style;
2. stating the current robot velocities (odometric information);
3. making the robot take environment *mosaic pictures*. In this work, we refer to a *mosaic picture*, meaning a 2(V)×3(H) matrix of 6 robot camera pictures, taken from different head robot positions (note that the robot camera is located in its head).

- A *visual range finder* operating on the mosaic images taken by the robot. The aim of this component is measuring the obstacle distances from the robot observation point. In order to do that, a previous "identification" of obstacles in the images, i.e. a *segmentation*, is needed.
- A *visual self-localization* procedure that estimates periodically the current robot position by working on the mosaic images.
- A module that *updates the internal environment representation* with

the information provided by the visual range finder and the visual self-localization procedure.

Switching to an "application deployment" point of view, the above functional components are actually split between the processing unit resident on the robot and the processing unit resident on a (Linux) host computer. This division results from two main criteria:

Different resources: tasks like user interface or sensori-motoric functionalities "belong" intrinsically to either the host computer or the robot, respectively. This is because the two platforms provides different resources, e.g., human-computer interaction devices like monitor, keyboard, mouse, etc. in the case of the host computer, and motor devices in the robot case.

Computing resources: some tasks, especially those concerning visual processing, take place on the host computer because of its better computing resources.

The separation of the system applications, generates a communication issue between the two platforms. This leads to the introduction of a new system component, called *Fusion Router*, that is in charge of managing the information exchange between host computer and robot. Fig. 3.2 provides a diagram representing the main functional components of our navigation system and their location inside either the host computer or the robot platform.

3.2 Robot modules

In this section, we briefly explain the technology underlying the robot programming and then we go in the details of each robot module.

3.2.1 OPEN-R objects

The robot is programmed in a C++ software environment using the Sony's OPEN-R software development kit [20]. OPEN-R application software consists of several OPEN-R *objects*. The concept of an object is similar to one of a process in the UNIX or Windows operating systems with regard to the following points of view:

- an object corresponds to one executable file;
- each object runs concurrently with other objects;

The following are characteristics specific to objects:

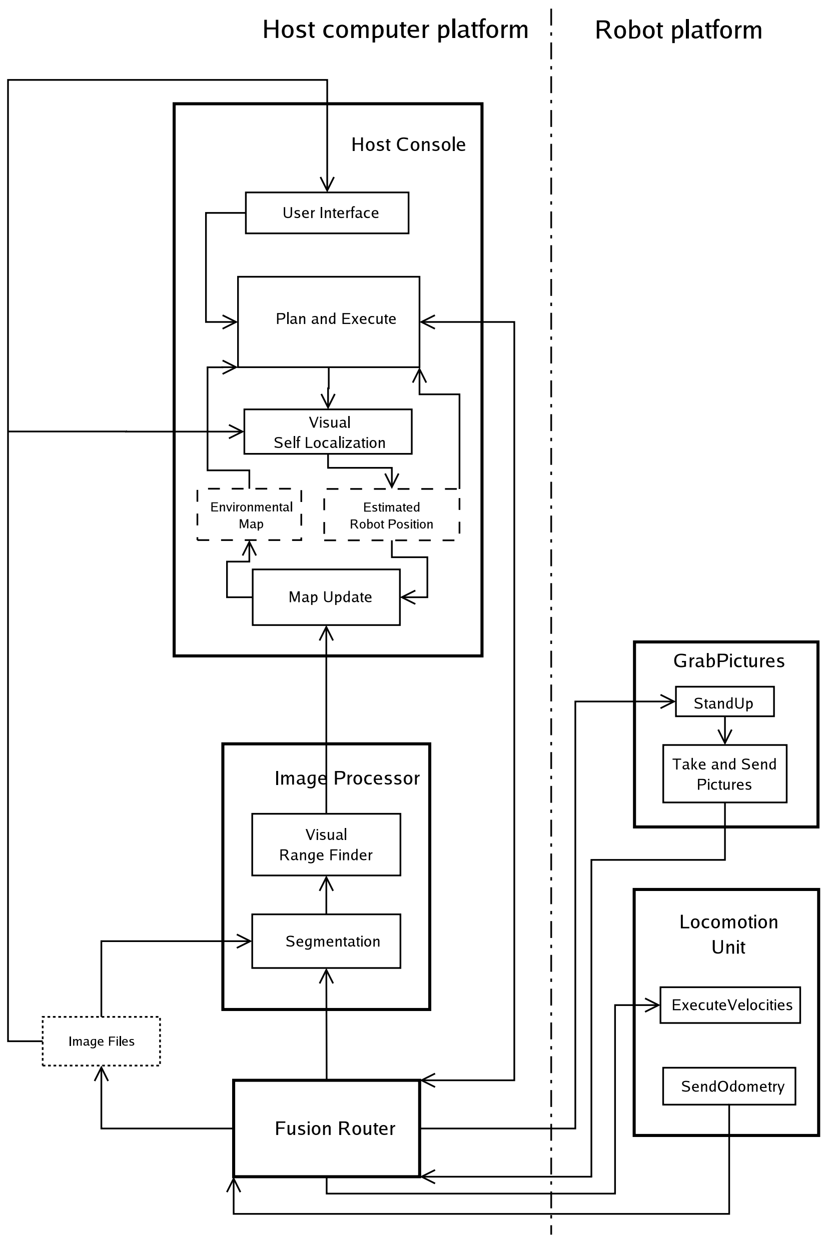


Figure 3.2: Navigation system structure divided in host computer and robot platform. The rectangles drawn with a continuous thick line, represent *processes* in the operating-system meaning; rectangles drawn with a continuous thin line, represent *functional components* of the system; the dashed rectangles represent *data structures* located in main memory, whereas the dotted rectangle, Image files, represents *files*. Note that the arrows linking the various components, indicate the direction of *information flow*. This can consists either in "actual" data, e.g., the pictures sent to FusionRouter by Take and Send Pictures, or in control signals, e.g., the "take pictures" command sent to Take and Send Pictures by StandUp.

- objects exchange information using message passing. An object can send messages to other objects. A message contains some data and a selector, which is an integer that identifies a task to be done by the receiver of the message. When an object receives a message, the function corresponding to the selector is invoked, with the data in the message as its argument.

- an object has multiple entry points. Unlike an ordinary programming environment in which a program has a single entry point "main()" , OPEN-R allows an object to have multiple entry points. Each entry point corresponds to a selector as explained above.

The OPEN-R system layer also provides a set of services (input of sound data, output of sound data, input of image data, output of control data to joints, and input of data from various sensors) as the interface to the objects. These services enable application objects to utilize the robot's underlying functionality, without requiring detailed knowledge of the hardware devices that comprise the robot.

3.2.2 Locomotion Unit

This part of code is strictly related to the *path execution* module resident in the host computer (see Subsection 3.3.2). In fact, the *path execution* module behaves like a supervisory process that exploits an abstract interface to the sensori-motoric robot functionalities, which are actually implemented by this part of robot code.

We employ a subset of the code written by the Swedish "Team Chaos 2004" (Cf. [15]) for the "Legged RoboCup 2004". In particular, our navigation system uses the following two services (see Figure 3.2):

- `ExecuteVelocities` accepts locomotion commands from the *path execution* host module in terms of *linear* and *rotational velocities*, and translates them to the actual robot walking;

- `SendOdometry` is in charge of sending continuously *odometric information* to the host computer. Odometry is simply obtained multiplying the last requested robot velocity by a constant slippery factor that roughly models the frictions between the dog legs and the ground, during the walk. This is an "open-loop" procedure that leads to quite imprecise odometrical information. For this reason, we try to get a more reliable robot position estimation integrating this kind of odometry with the estimation provided by the visual self-localization process (see Ch. 6).

3.2.3 Grab Pictures

This part of software concerns the robot response whenever it receives a command of taking an environment mosaic picture. This command comes from the host modules during the environment exploration performed by the robot. Moreover, this command only occurs when the dog is still, i.e., its current linear and rotational velocities are zero. In this case, the robot performs the following operations:

1. it reaches the *Sony AIBO's standard stand-up position* (Cf. [30]), starting from a kneeling position. In fact, the walking style employed in [15], makes the dog walk on its front knees and the robot still remains on its knees when its motion is stopped. Fig. 3.3 illustrates the stand-up position along with the robot leg joints involved.

   The aim of this positioning is bringing the robot to a priori-known posture before it starts taking pictures. Knowing the "grabbing picture pose" is extremely important because it allows to build the direct kinematics part of the calibration matrix (see Subsection 4.2), which, in turn, enables to state a correspondence between real world object shot by the robot camera and the related image pixels.

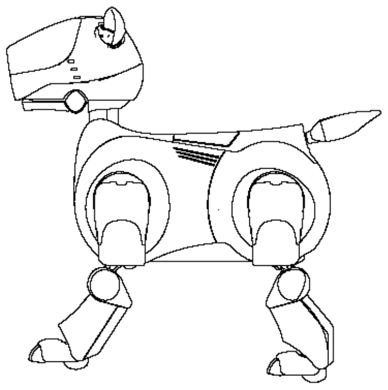
   Another reason for standing the dog up is raising the height of the robot camera, thus allowing for a larger field of view of the pictures.

   This pose is achieved through two intermediate postures. Each pose is specified by a set of angles, related to the joints belonging to the front and rear robot legs (note that all the other joints, e.g., the head ones, are not involved in this positioning). The procedure that actually makes the robot move, is based on OPEN-R primitives (see [32]).

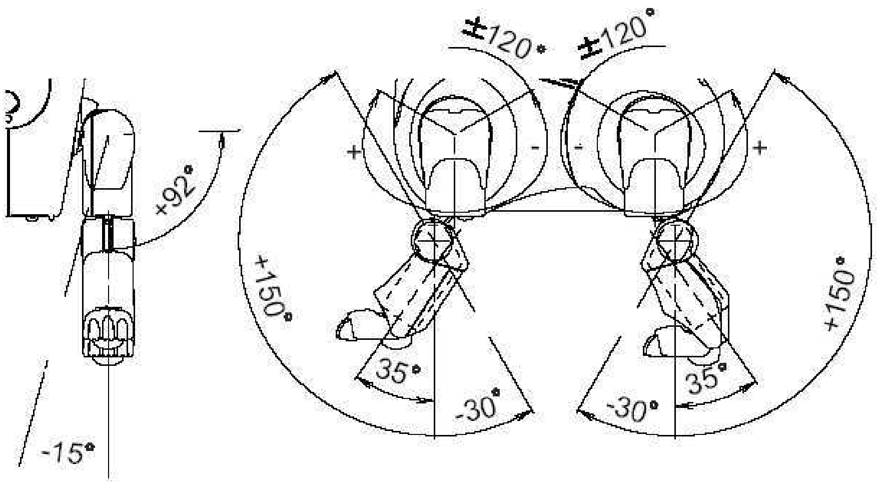
2. The robot moves its head (remember that the camera is located on the robot head), taking six different pictures that compose a mosaic. Each picture is shot with a particular combination of pan and tilt head angle. Fig. 3.4 shows the chronological sequence (within the mosaic) of the six images taken by the robot, whereas Fig. 3.5 illustrates the pan and tilt degrees of freedom of our robot neck.

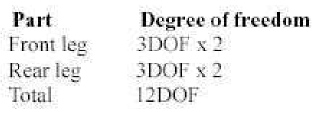
   Choosing the pan/tilt angle of each "tile" the composes the mosaic, is an important issue. The principal aim underlying this choice, is acquiring as much environment information as possible from a mosaic picture. This results in:

   - maximizing the whole mosaic environment field of view. This is achieved by trying to shoot non-overlapping environment regions, between any two contiguous mosaic tiles. This can be realized if:
     - the pan angle difference between two horizontal contiguous tiles (e.g. 1 and 2 in Fig. 3.4), is set to the horizontal field of



(a) Stand-up position.





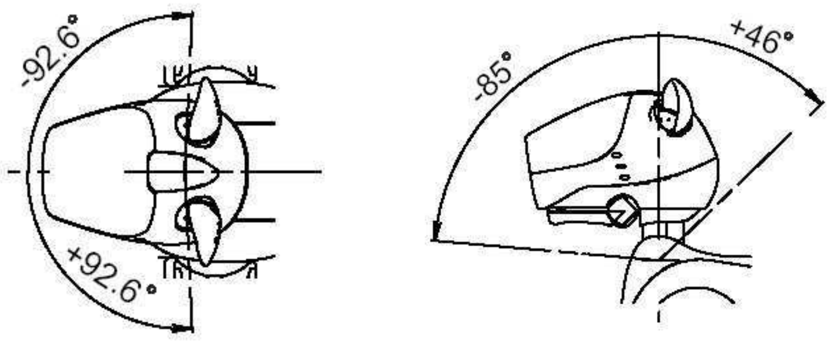
Part	Degree of freedom
Front leg	3DOF x 2
Rear leg	3DOF x 2
Total	12DOF

(b) Leg joints characterizing the stand-up position, along with their operational limits. Note that in the stand-up position, right and left legs are in the same configuration.

Figure 3.3: Stan-up position (a) and leg joints involved in it, along with a general description of the degrees of freedom of AIBO's legs (b).

3	2	1
4	5	6

Figure 3.4: Mosaic composition. The crescent numbers express the chronological order through which each tile has been acquired by the robot camera. Note that tiles belonging to the same row have been taken with the same tilt angle, whereas tiles belonging to the same column have been taken with the same pan angle.



(a) Pan (b) Tilt

Figure 3.5: Robot pan (a) and tilt (b) neck joints along with their operational limits and measuring conventions.

  view of a single tile image. Its value is $fov_H = 57.6°$, Cf. [30].
  - the tilt angle difference between two vertical contiguous tiles, is set to the vertical field of view of a single tile image. Its value is $fov_V = 47.8°$, Cf. [30].

- mosaic images focus on the part of environment belonging to the ground. In fact, that is the region we are interested in for navigation purposes. This is obtained by employing negative tilt values for the mosaic tiles (see Fig. 3.5(b)).

The above considerations lead to the following possible set of pan/tilt angles for the mosaic tiles (see also Fig. 3.4):

Tile	Pan angle	Tilt angle
1	$-fov_H$	$fov_V/2 + \mathit{TiltOffset}$
2	0	$fov_V/2 + \mathit{TiltOffset}$
3	fov_H	$fov_V/2 + \mathit{TiltOffset}$
4	fov_H	$-fov_V/2 + \mathit{TiltOffset}$
5	0	$-fov_V/2 + \mathit{TiltOffset}$
6	$-fov_H$	$-fov_V/2 + \mathit{TiltOffset}$

where *TiltOffset* is a negative angle that “tilts” the mosaic field of view toward the ground. We actually adopt $\mathit{TiltOffset} = -44°$.

The robot digitizing hardware provides images in YCrCb color format with a 176 × 144 resolution. Once an image has been acquired, it is immediately sent to the host computer via a TCP wireless connection (see Section 3.5).

3.3 Host modules

In this section, we explain, with more details, some elements that constitute the Host Console *process*, see Fig. 3.2. In particular, we speak about User Interface (subsec. 3.3.1) and Plan and Execute (3.3.2). The latter one, along with Map Update, Visual Self-Localization, and Image Processor, are the most substantial contributions of this work and they are described at length in chapters 4, 5 and 6.

Note that Fusion Router is treated in Section 3.5.1.

3.3.1 User interface

As already discussed, the main tasks of this unit are:

- providing a visual representation of what the robot is actually looking through the mosaic of pictures. This is a complex task since each mosaic tile shoots a different region of the environment with particular angles (see Section 3.2). Hence, a "fusion" procedure of the mosaic tiles is needed, in order to get a *panoramic picture*, representing a coherent vision of the environment to be presented to the operator. The main input to this procedure are the images taken by the robot and stored on the host computer file system, see Fig. 3.2. Refer to Section 3.4 for further details about the panoramic image building.

- Allowing the operator to decide which location of the environment the robot must reach. Provided that the operator's environment knowledge, relies only on the information acquired by the robot, the most logical way to indicate a goal position to the robot, is dealing directly with the internal environment representation of the robot, i.e., Environmental Map in Fig. 3.2.

  What we actually do, is showing to the operator the map that has been built by the robot up to now, and waiting until the operator clicks on a cell of the map (Fig. 3.6 shows the related window). After that, a

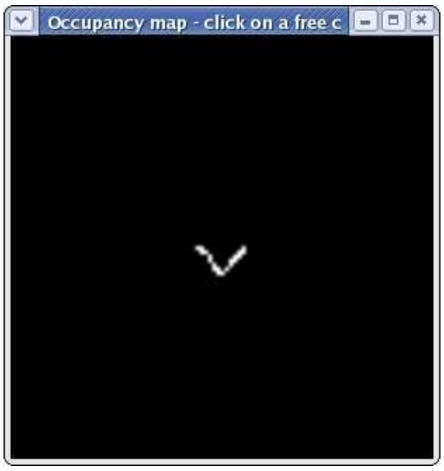


Figure 3.6: Environmental map window. This is shown every time the operator has to indicate a robot goal position by clicking on it. This map was obtained by positioning the robot in front of a sharp-cornered obstacle.

  path to the goal is planned and visualized to the operator. Then, the path execution procedure starts. Once the robot reaches the desired position, the operator can select another goal.

3.3.2 Plan and execute

Planning directly works on the environment map stored within Host Console (see Fig. 3.2), looking for an obstacle-free path from the Estimated Robot Position to the operator-selected goal (see Section 5.2 for further details).

Once a suitable path has been computed, the *path execution* procedure is invoked in order to make the dog reach the goal. During this phase, a strong cooperation with the robot modules is required. In fact, in order to follow the planned path, *path execution* needs to command proper velocities to the robot and check how it is actually executing them, through odometry information received by SendOdometry (see Fig. 3.2). Therefore, a bidirectional communication with the dog is demanded. This is realized through the Fusion Router component (see Section 3.5).

Following a path, includes reaching some intermediate environment locations, the so called *way-points* (see Subsec. 5.3.1). In order to make path execution as much precise as possible, i.e., to make the robot final position as much close to the goal as possible, a periodical robot self-localization is required, i.e., a "refresh" of Estimated Robot Position. In fact, information provided by SendOdometry are not enough for this aim (see Sec. 3.2). Therefore, at each way-point a Visual Self-Localization is performed (see ch. 6). Provided that Visual Self-Localization relies on what the robot is currently looking, *path execution* must command the dog, through Fusion Router, to take a mosaic picture, before invoking Visual Self-Localization.

3.4 Panoramic images

In this section, we illustrate the building of *panoramic images* to be presented to the operator, starting from mosaic images taken by the robot.

Panoramic images are an important mean to provide the operator with a wide view of the environment as it is "seen" from the robot.

In literature, panoramic images obtained from image sequences, are often carried out through a *registration* stage followed by a *blending* operation. The first one, which is much more relevant, starting from corresponding pixels belonging to a pair of *overlapping images*, computes the transformation from one image to the other, e.g., Cf. [34]. The last one is performed to "average" the two images in the overlapping region, and can be achieved, for instance, through a bilinear function. There should be noted that an automatic selection of corresponding pixels that allows to register images in an accurate way, is still a relevant problem.

Anyway, in our case the above techniques are hardly applicable since our mosaic tile pictures present small overlapping regions (see Subsec. 3.2.3). For this reason, we have developed three alternative ways to obtain panoramic images that do not make use of *registration*. They are listed here below in a complexity order.

3.4.1 Array of images

The first solution we have implemented is the simplest and obvious one: we just display the six tile images positioning them on the same flat plane in a

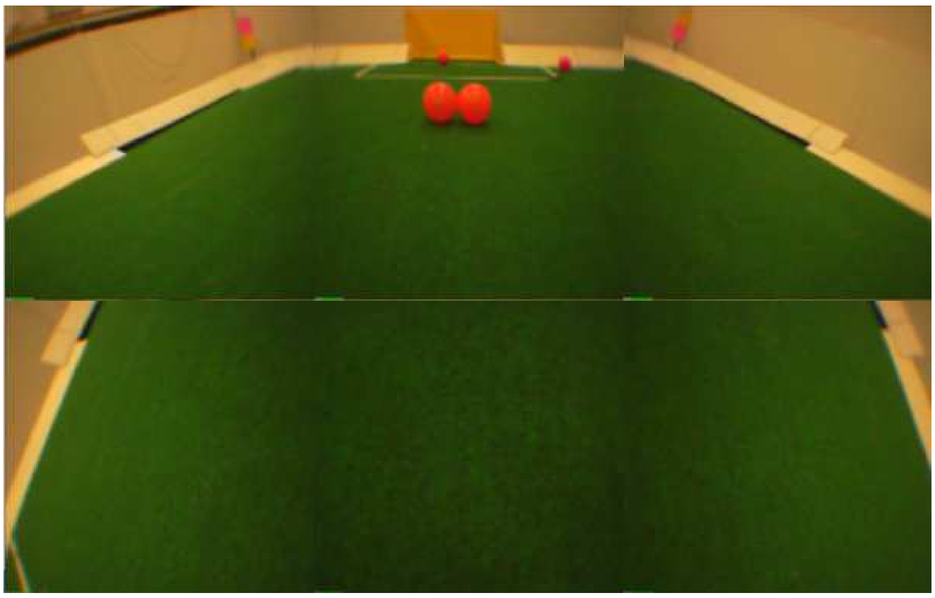


Figure 3.7: Tile images are placed on the same flat plane in a 2×3 grid. The *operator* does not have an homogeneous vision of the environment in front of the robot.

2×3 grid, with sides attached to each other just as tiles in a mosaic. The positioning process is performed assigning to each tile a grid cell, according to the pan and tilt angle the image has been taken with, see Sec. 3.2.3.

The resulting mosaic gives the *operator* a general idea of the environment surrounding the robot, but it suffers of some drawbacks. It does not seem a homogeneous image, mainly because image tiles appear on the same plane even they have been actually acquired on 6 different image planes, i.e., the planes corresponding to the particular combinations of pan and tilt angles related to the camera rotation.

3.4.2 Pan/tilt rotation

The main idea of this approach is to rotate the previous tile images in a virtual space, created using OpenGl, in order to resemble the actual position of the various tile image planes during the mosaic acquisition. In fact, we rotate tile images around according to their pan/tilt angles. To perform it, we need to know the distance f, i.e., the *focal length*, between the center of rotation, i.e., the *focal point*, and the image plane, expressed in the virtual 3D world coordinates. It can be computed knowing the horizontal field of view, fov_H, and the horizontal resolution, d_x, of the image as follows, (see Figure 3.8):

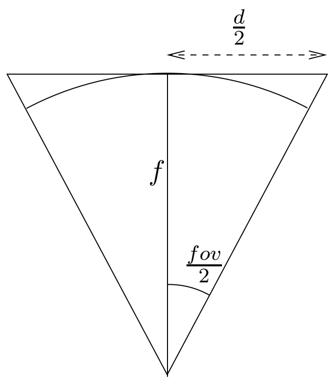


Figure 3.8: The focal length f is obtained considering the field of view of an image, fov, and the size of the image, d.

$$f = f_h = \frac{d_x}{2\tan\frac{fov_H}{2}}$$

where $fov_H = 57.6°$and $d_x = 176$ pixels. However, f can be computed also using the vertical field of view and vertical resolution of the image, yielding f_v. Since f_h and f_v result to be slightly different, due to rounding errors, we decide to use the average $f = \frac{f_h+f_v}{2}$.

The mosaic, resulting from the rotation of each tile, shows a more realistic view of the world than the procedure described in Sec. 3.4.1. In particular this method provides a more realistic perception of depth in the environment.

A drawback of this method is the presence of discontinuities due to the use of flat images, see Figure 3.9. To avoid this problem we have employed the following method.

3.4.3 Spherical projection

This method creates a "smooth" panoramic view of the environment, obtained projecting each of the six tile images onto the surface of a semi-sphere with radius equal to the focal length f. The observer point of view is ideally in the center of the sphere, i.e., in the focal point.

We use an approximated method to achieve this projection, exploiting, as in the previous subsection, OpenGl to create a virtual 3D space. The approximation concerns the division of each picture in $N \times N$ rectangle *elements*. Each of them is, then, reshaped and moved in order to wrap the semi-sphere surface. If N is too small, discontinuities in the final panoramic view will be visible. On the other hand, if N is too high, the computational load will slow down the execution. We find that $N = 24$ may be a good trade-off between quality and efficiency.

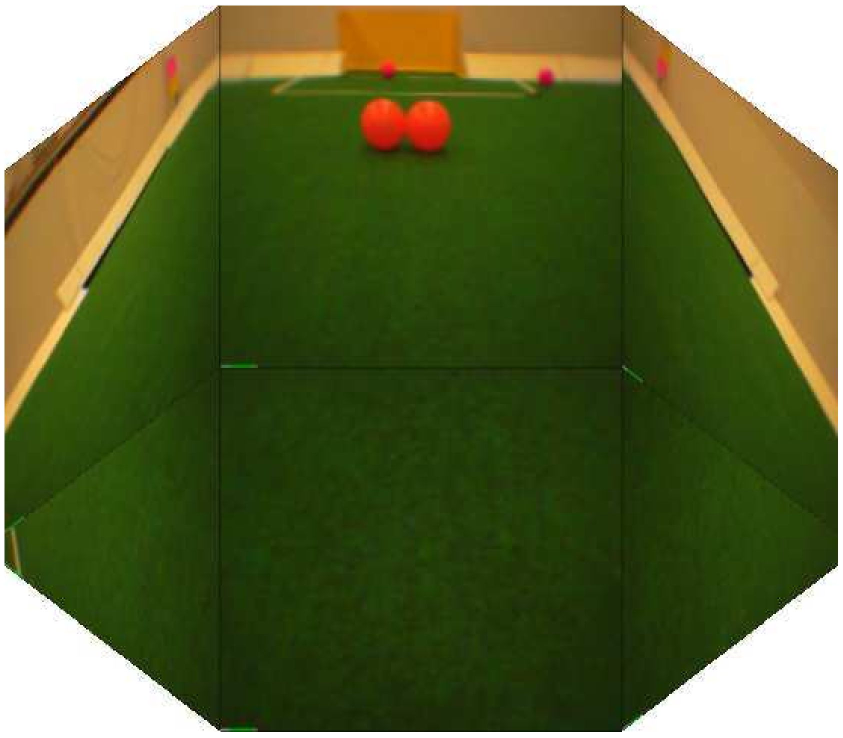


Figure 3.9: The six tile image planes are placed in a 3D virtual world in order to resemble their actual position during the mosaic acquisition performed by the robot camera. Note that the upper-lateral rotated tiles, i.e., tile 1 and 3, partially occlude the lower-lateral ones, i.e., tile 4 and 6.

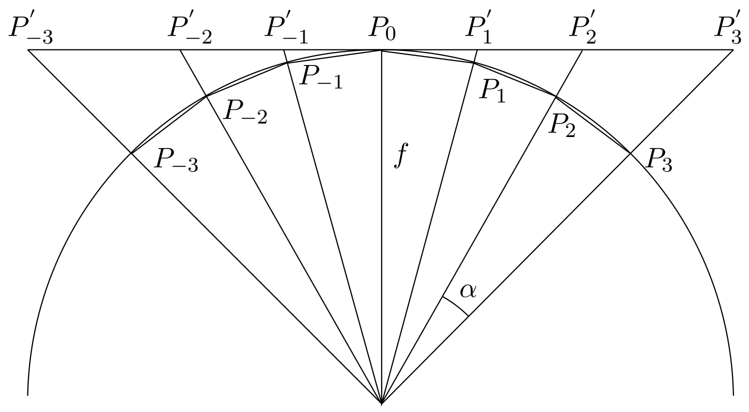


Figure 3.10: Here is shown the subdivision along one dimension of a tile image. This is divided into *elements* of equal field of view, α. The projected *elements* on the sphere surface have equal size, i.e., $\overline{P_{-3}P_{-2}} = \overline{P_{-2}P_{-1}} = \overline{P_{-1}P_0} = \overline{P_0P_1} = \overline{P_1P_2} = \overline{P_2P_3}$. By contrast, on the image plane, the lateral *elements* are bigger than the central ones, e.g. $\overline{P'_2P'_3} > \overline{P'_0P'_1}$

The building of such panoramic image can be decomposed into these subtasks to be performed for each *element*:

1. *creation* of a rectangle *element*, through the division of the image;
2. *reshaping* of the *element*;
3. *rotation* of the *element*, in order to place it on the sphere surface.

In the following paragraphs we explain how we have solved these subtasks.

Creation of the *element* We divide each image in rectangle *elements* of equal field of view, w.r.t. the focal point, see Figure 3.10. Note that *elements* close to the image borders, e.g., $\overline{P'_2P'_3}$ or $\overline{P'_{-2}P'_{-3}}$ in the Figure, have a larger dimension than the central ones. By contrast the projected *elements* have equal length, i.e., $\overline{P_{-3}P_{-2}} = \overline{P_{-2}P_{-1}} = \overline{P_{-1}P_0} = \overline{P_0P_1} = \overline{P_1P_2} = \overline{P_2P_3}$.

Reshape of the *element* The next step is to cover the semi-sphere surface with *elements*. In order to do that, first, we divide the sphere in a grid and then we fit each *element* in the corresponding grid cell.

There are different ways to draw a grid around a sphere and we used the meridian/parallel scheme, see Figure 3.11. *Latitude* is the angle corresponding to a parallel whereas *longitude* that one corresponding to a meridian. Each cell of the resulting grid has a constant height, because the arc length between two parallels is constant. On the contrary, the width depends on

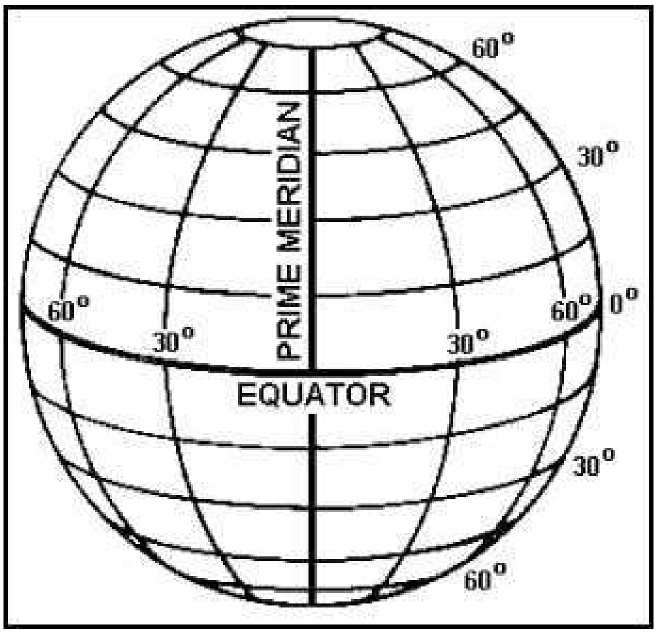


Figure 3.11: Division of the sphere in meridians and parallels. Each cell of this grid has a constant height and a variable width depending on the *latitude.*

the *latitude*, i.e., cells near to the "poles" of the globe have smaller width those close to the "equator".

In order to approximately fit the grid cell in a proper way, we choose to reshape the original rectangle *element* into a trapeze, because it resembles more the cell of the grid than a rectangle and fits better the slice between two meridians, see Figure 3.12.

Rotation of the *element* Each element is associated with a particular grid cell, depending on the image it belongs to, and on its location within the tile image itself. As last step, each *element* is rotated around the center of the sphere according to the *latitude* and the *longitude* of the cell to be filled. Once all the *elements* have been created, reshaped into a trapeze and rotated in the correspondent cell, the panoramic image is completed and wraps the surface of the sphere, see the result in Figure 3.13.

The panoramic image, obtained using the method so far described, offers better results than that ones described in the previous sections, because it does not suffer of the disadvantages of the others.

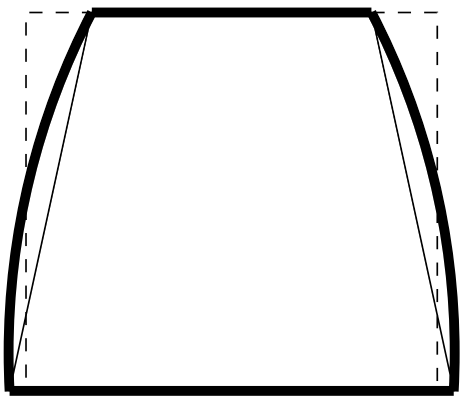


Figure 3.12: A grid cell of the sphere, drawn with a thick line, is fitted better by a trapeze than a rectangle, because of the different width of the bases.

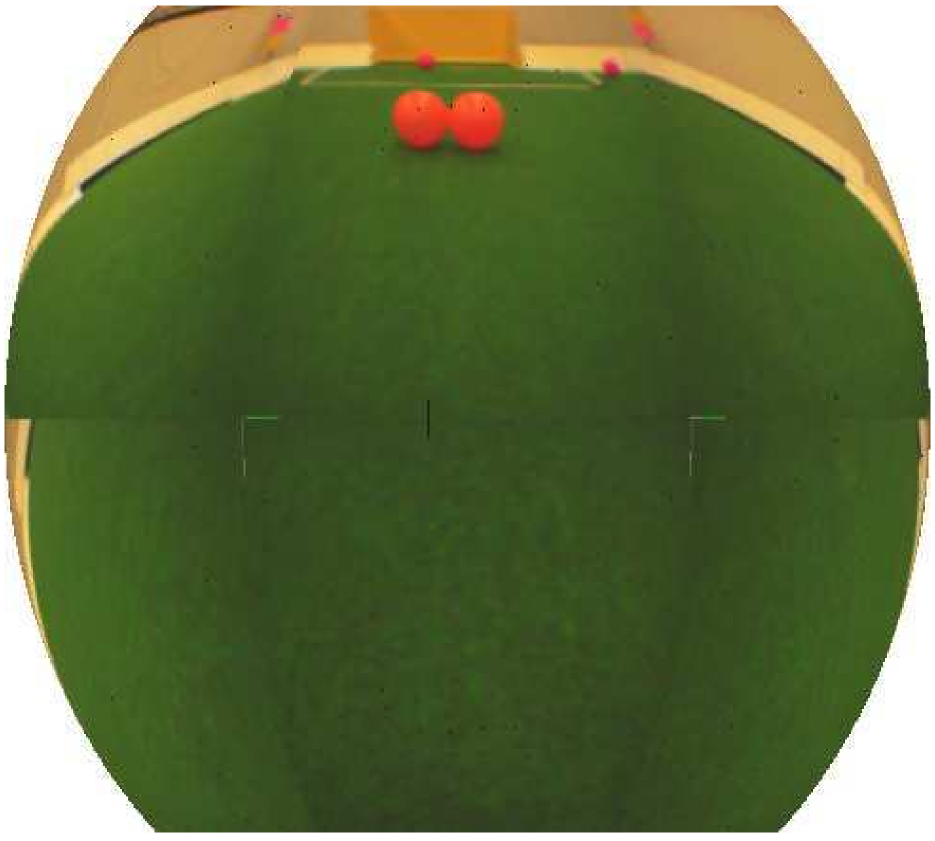


Figure 3.13: *Panoramic image*: projection on a sphere surface of the set of images. The point of view of the observer is set in the center of the sphere. The field of view of the whole image is 172.8° horizontally and 95.6° vertically.

3.5 Middleware

As we already explained, the separation of system applications between two platforms, produces a communication issue. Also the partition of host computer modules in different operating system *processes*, gives rise to inter-communication issues within the host platform. In the following subsections, we discuss both the topics.

3.5.1 Fusion Router

Fusion Router embodies the connection between host platform and robot. It exploits the *Remote Processing OPEN-R* (RP OPEN-R) technology, developed by Sony, Cf. [31]. RP OPEN-R is a remote processing environment where it is possible to execute an OPEN-R based program on a remote host which is not an AIBO. By using RP OPEN-R, some OPEN-R objects can be executed on the remote host (connected to AIBO via wireless LAN), and other objects can be executed directly on AIBO. All objects will be executed as one program, distributed between the two machines. In this way, Fusion Router, Locomotion Unit and GrabImage (see Fig. 3.2) run on the same "virtual" platform and their physical separation is almost hidden to the programmer that does not have to care about middleware routines. In fact, the communication between AIBO's objects and the remote host object (Fusion Router), is done by extending the ordinary message-passing protocol of the OPEN-R inter-object communication (Cf. [31]) over the wireless LAN. Note that RP OPEN-R implementation actually relies on the TCP.

3.5.2 Communication among host processes

All the communications among host processes are done via *UDP*. This is a straightforward solution that allows to connect easily two processes. Note that UDP is not a reliable protocol, but, since we deal with processes on the same machine, we do not need that characteristic since the physical connection is intrinsically robust.

An issue related to the use of UDP, is the serialization (marshalling) of complex data structure, e.g., `struct`, that must be performed by the sender process before it sends data to the receiver process. On the other way, the receiver must unmarshall the received byte stream in order to recover the original data structure.

Here is the list of UDP connections employed in our system (refer to Fig. 3.2):

- Fusion Router $\longrightarrow$ Host Console : used for transmitting odometry information originated in SendOdometry;
- Host Console $\longrightarrow$ Fusion Router : used for transmitting velocity commands that will be received by ExecuteVelocities;

- Fusion Router $\longrightarrow$ Segmentation : control signal used to notify the presence of new available image files. In fact, whenever the robot finishes to acquire a mosaic picture, it sends it to Fusion Router that, in turn, saves it to files.
- Visual Range Finder $\longrightarrow$ Map Update : used for transmitting obstacle ranges;

Chapter 4

Obstacle detection

In this chapter, we provide the details of the *"visual sensory" part* of our navigation system, which includes the segmentation of robot input images (section 4.1) and two possible methods employed for finding obstacle ranges (4.3, 4.4). The last ones work on the segmented images with the aim of stating properties of real world points, i.e., the obstacles distances from the robot. In order to do that, a previous understanding of how the environment is "mapped" onto the images shot by the dog, is needed (4.2).

4.1 Segmentation

Once the robot has taken some pictures of the surrounding environment segmentation is the first step to be done in order to extract useful information from the "raw" image data.

The general purpose of segmentation is partitioning an image into meaningful regions.

Provided that avoiding collision is essential during the navigation, our particular aim is to separate what is free ground from what is not, i.e., obstacles. A priori knowledge of the environment is needed to achieve this classification and it can consist either in information about the "free ground" or about the "obstacle".

Since we deal with a general rescue environment the definition of "obstacle" should be as less constrained as possible, for instance think about a collapsed building scenario with any kind of debris and rubble on the ground. Therefore it is better to specify the "free ground" and we have done this in two possible ways: either prompting the operator to indicate a "free ground" area in the images or making some assumptions regarding the location of "free ground" in the pictures. From a procedural point of view, the main difference between these solutions is that the second one is automatic whereas the first one is not. We shall discuss both solutions and compare them in subsections 4.1.2, 4.1.3, and 4.1.4, respectively.

Color space thresholding (see Subsec. 4.1.1) is the approach we use for segmenting because, even if it is a simple solution, it is more efficient than edge-based and region-based techniques and it is usually quite effective in background-objects classification problems like ours.

The threshold selection method is treated in 4.1.5, whereas image-enhancing filters to be applied just before or after the thresholding stage, are described in 4.1.6.

4.1.1 Thresholding

The following paragraphs give a brief explanation of gray-level and multi-spectral thresholding and describe the reasons we have chosen the latter one.

Monochromatic input images

Gray-level thresholding is the simplest segmentation process. Many objects or image regions are characterized by costant reflectivity or light absorption of their surfaces; a brightness costant or *threshold* can be determined to segment objects and background.

Thresholding is the transformation of an input image f to an output (segmented) binary image g as follows:

$$g(i,j) = 1 \text{ for } f(i,j) \geq T$$

$$g(i,j) = 0 \text{ for } f(i,j) < T$$

where T is the threshold, $g(i,j) = 1$ for image elements of the background, and $g(i,j) = 0$ for image elements of the objects (or vice versa), Cf. [29].

Color input images

Many practical segmentation problems need more information than is contained in one spectral band because this usually results in a more effective classification.

Color Space Transformation Our approach involves the use of thresholds in a three dimensional color space. Several color spaces are in wide use, including Hue Saturation Intensity (HSI), YUV (or YCrCb) and Red Green Blue (RGB). The choice depends on several factors including which is provided by the the digitizing hardware and utility for the particular application.

RGB is a familiar color space often used in image processing, but it suffers from an important drawback. In fact, we would like our classification process to be robust in the face of variations in the brightness of illumination, so it would be useful to define a particular color (for instance the ground) in terms of a ratio of the intensities of Red Green and Blue in the pixel. This can be done in an RGB color space, but the volume implied by such a relation is conical and cannot be represented with simple thresholds.

In contrast, HSI and YCrCb have the advantage that chrominance is coded in two of the dimensions (H and S for HSI or Cr and Cb for YCrCb) while intensity is coded in the third. Thus a particular color can be described as "column" spanning all intensities, Cf. [3].

We have chosen to work with YCrCb because our robot provides these colors in hardware. This combines the power of a robust color space without the performance penalty of a software color space transformation.

Classification criterion Our segmentation approach determines thresholds independently (see Subsec. 4.1.5) in each spectral band of chrominance (Cr, Cb), and combines them into a single segmented image.

The ground color class is specified as a set of four thresholds values: two for each dimension in the chrominance space. We exclude the Y component since we intend to reach a more robust classification, i.e., a classification that is possibly independent of the environment light conditions.

A pixel with values `Cr`, `Cb` is classified through the following comparisons:

```
if ( (Cr ≥ Crlowerthresh)
     AND (Cr ≤ Crupperthresh)
     AND (Cb ≥ Cblowerthresh)
     AND (Cb ≤ Cbupperthresh))
     pixel_color = 1; //free ground
else
     pixel_color = 0; //obstacle
```

4.1.2 Semiautomatic solution

In this strategy, the operator has to select (via mouse) a rectangular area in each picture composing the mosaic image provided by the robot (see Subsec. 3.2.3), whenever it grabs pictures. The selected picture portion, which should contain almost free ground, is used to compute the four thresholds of the ground color and the resulting segmented picture is immediately shown to the operator so that he can evaluate the outcome of his previous choice and either decide to select again over the same picture or the next one in the mosaic. See Fig. 4.1.

4.1.3 Automatic solution

This procedure is based on the assumption that the lower half of the mosaic image lower row, i.e., the lower half of each of the three mosaic tiles acquired with lower tilt angle (see Subsec. 3.2.3), contains almost only free ground (see Fig. 4.2). This hypothesis makes possible to avoid the operator intervention

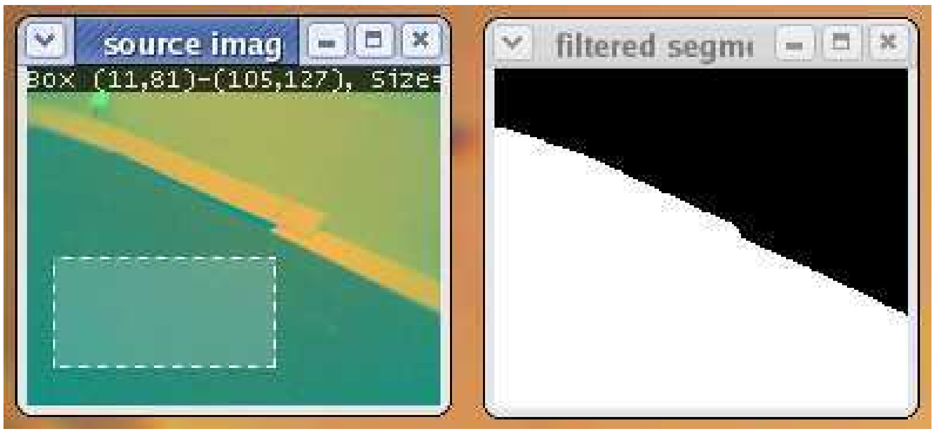


Figure 4.1: Snapshot related to the free-ground area selection task: the left window shows one of the mosaic "tile" YCrCb image provided by the dog together with the current operator selection rectangular area; the right window shows the result of segmentation applied to the left window image

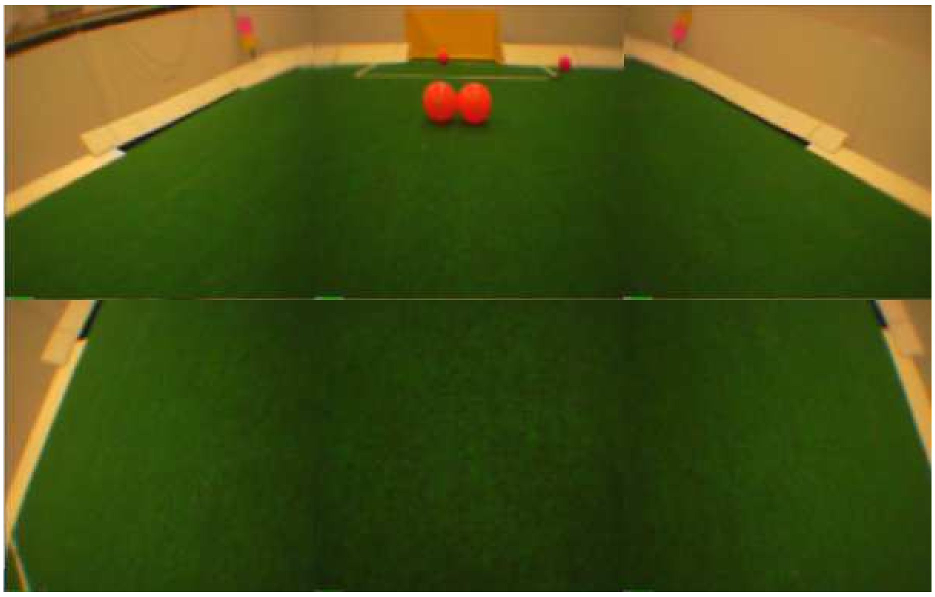


Figure 4.2: RGB mosaic image (see Subsec. 3.4.1 to understand how it has been obtained). Note that the lower half of the mosaic tiles belonging to the lower row, contains almost only free ground.

and develop an automatic algorithm.

Our approach consists in extracting four thresholds from each lower half of the above discussed three pictures, averaging them with the same weight and applying the derived thresholds to the whole the mosaic image.

4.1.4 Comparison

Of course the biggest advantage of the automatic solution is freeing the operator of the segmentation burden. The assumptions are not so restrictive because the three low-tilt mosaic tiles represent an environment zone which is very close to the robot, hence supposing the lower half of them are without obstacles, is reasonable to the extent that the whole navigation process does not bring the dog in the immediate obstacle vicinity.

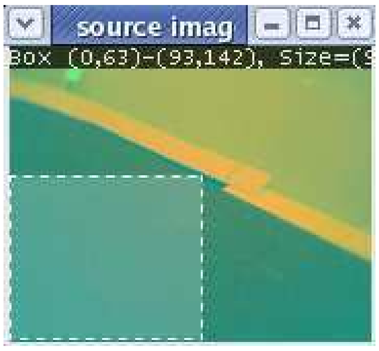
The major drawback is that the same thresholds are used in all the mosaic tiles and this leads to a more noisy classification than in the semi-automatic system where it is always possible determining thresholds from a free ground area in each tile. This is especially evident for the high-tilt tiles because the final thresholds do not take in account them in the average whereas the ground color could slightly vary in the mosaic tiles due to its eventual inhomogeneity.

4.1.5 Threshold selection

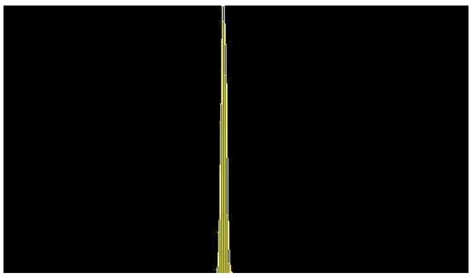
The way thresholds are computed is the same in both the above solutions although input to the threshold selection task is respectively an operator defined cropped image or a low-half tile picture.

The *input image* is decomposed in two channels (Cr,Cb) and the histogram is calculated from each one. Every histogram is then used independently to compute two thresholds representing the ground intensity range in a particular band. In fact, as we assume that the input image contains almost only free ground, each histogram has a main peak related to the principal intensity characterizing the ground in a specific channel and a series of smaller peaks related mainly to secondary ground intensities, and also to obstacles that are eventually present (See Fig. 4.3). Actually the thresholds are two local minima respectively on the right and left side of the histogram absolute maximum corresponding to the above discussed main peak. Being local minimum, a threshold coincides with the "valley" between two adjacent ground intensity peaks. This allows to consider selectively the ground "shades", depending on the way these local minima are chosen. We choose, on both the above referred sides, the closest local minimum to the absolute maximum that satisfies the condition

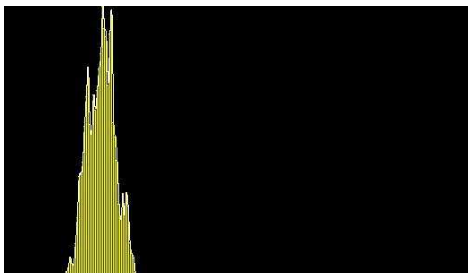
$$\frac{abs\_max}{local\_min} > RATIO$$



(a) Free-ground cropped image



(b) Histogram of the Cr channel of the cropped image



(c) Histogram of the Cb channel of the cropped image

Figure 4.3: Histograms (b-c) of the two chrominance components of the YCrCb free-ground selected image (a). You can clearly see the main peak in each histogram. Note that the shown histogram x-axes (horizontal ones) range from 0 to 255 whereas the shown y-axes range from 0 to the maximum value in each histogram function.

where $RATIO$ is a parameter characterizing the selection rule: the bigger is $RATIO$ the wider is the range defining the ground in each color component. It is usually better considering an high value of $RATIO$ in order to cover all ground shades. On the other hand, taking an extremely high value of $RATIO$ implies including almost everything in the class "ground" making segmentation a useless process. We usually consider $RATIO = 100$.

If such a minimum is not found we search for the last bin satisfying $\frac{abs\_max}{freq\_value} > RATIO$, where $freq\_value$ is the frequency value related to the current considered bin. Searching starts either from bin 255 or bin 0 (the histogram is a frequency function whose domain has 256 (0..255) values called bins), depending on whether the not-found minimum is the right-side one or the left-side one. In both cases, searching goes toward the absolute maximum.

Here is the algorithm in brief (for sake of clarity it does not contain the above mentioned particular case):

```
int histogram[256]; //the histogram is a frequency function
                    //whose domain has 256 (0..255) values
                    //called 'bins'

int abs_max;
int abs_max_bin;

int right_local_min;
int right_local_min_bin;

int left_local_min;
int left_local_min_bin;

int search_start_bin;
find_abs_max(); //initializes abs_max and abs_max_bin

//searching for the right local minimum
search_start_bin = abs_max_bin + 1;
right_local_min = MAX_INT;
while (abs_max/right_local_min < RATIO){
      //updating right_local_min and right_local_min_bin
      find_first_right_minimum(search_start_bin);

      search_start_bin = right_local_min_bin + 1;
}

//searching for the left local minimum
search_start_bin = abs_max_bin - 1;
```

```
left_local_min = MAX_INT;
while (abs_max/left_local_min < RATIO){
      //updating left_local_min and left_local_min_bin
      find_first_left_minimum(search_start_bin);

      search_start_bin = left_local_min_bin - 1;
}
```

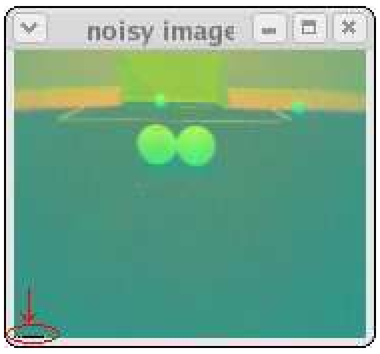
where MAX_INT is a very high integer value used for initializing a minimum search.

4.1.6 Filtering

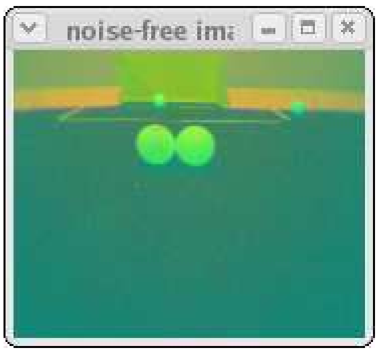
We use some filters before and after the thresholding procedure explained so far in order to improve the quality of the whole segmentation process.

Pre-filters

1. Provided that the robot camera pictures contain a systematic noise in the lower-left corner, i.e., sixteen pixels almost of the same color (green), we substitute each of these pixels with an *average* of its neighbouring pixels. See Fig. 4.4.



(a) Noisy image. The noisy pixels are in the circle.



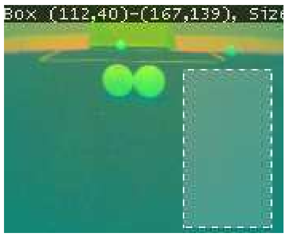
(b) Filtered image

Figure 4.4: Filtered image (b) resulting from having substituted each of the sexteen noisy pixels in (a) with an *average* of its neighbouring pixels.

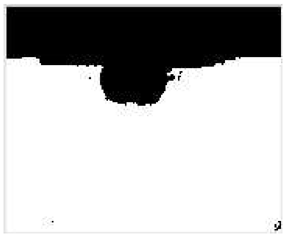
2. Then we use a *Gaussian filter* with $\sigma = 1$ to smooth the images. This is a general image pre-processing technique for noise filtering. (PICTURES)

Post-filters

Once an image has been thresholded, we apply a 4-connectivity *majority filter* which sets each pixel to either 1 or zero depending on the majority value of its 4-connected neighbors. Then, the resulting image undergoes a double 4-connectivity *erosion.* The aim of the series of these two filters, is to eliminate isolated small groups of noisy pixels, i.e, wrongly classified "obstacle" pixels. See Fig. 4.5.

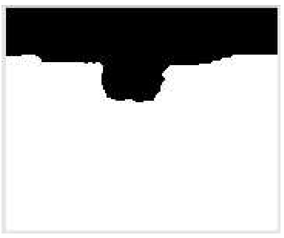


(a) Source YCrCb "tile" image with the active free-ground selection area used for thresholding.



(b) Thresholded image. Look at the misclassified pixels, e.g., isolated black pixels and small isolated groups of object pixels located in the bottom-right corner and close to the right ball.

(c) Image resulting from applying a majority-vote filter to the thresholded image. Isolated pixels from (b) are removed but the small groups still remain.



(d) Final segmented image obtained by applying two erosions to the majority-vote filtered. The small groups of wrongly classified object pixels are finally filtered out. image

Figure 4.5: Post filtering chain: source image (a) is thresholded in (b) that is filtered with a majority-vote filter. This results in (c) that is finally filtered through two erosions giving the final image (d)

4.2 Mapping between real world and image plane

In this section, we focus on the relation between real world points and pixels belonging to the image plane of a particular mosaic image tile.

This mapping can be expressed, using homogeneous coordinates, with a matrix K, called *calibration matrix.*

This approach is based on the decomposition of K in two parts:

1. A matrix that expresses the position and orientation of a frame attached to the robot camera with respect to a dog's body framework. It depends both on extrinsic parameters related to the robot specification, as joint lengths and location of rotation centers, and on the particular *pan* and *tilt* characterizing a particular image taken by the dog (see Section 3.2.3). It is called *direct kinematics transformation* T_b^c. The subscript b stands for the base framework which stands on the ground and whose origin origin coincides with the projection of the head/tilt center on the ground. The superscript c, instead, stands for the framework with origin coinciding with the center of the camera, located in the robot head (see Figure 4.6 and Figure 4.8).

2. A matrix that expresses the perspective projection of the world points into the image plane. It depends on intrinsic parameters that are related to the camera itself, e.g., focal length and pixel dimension. It is called *perspective transformation* Ω.

The *calibration matrix* is expressed as follows:

$$K = \Omega T_b^c \tag{4.1}$$

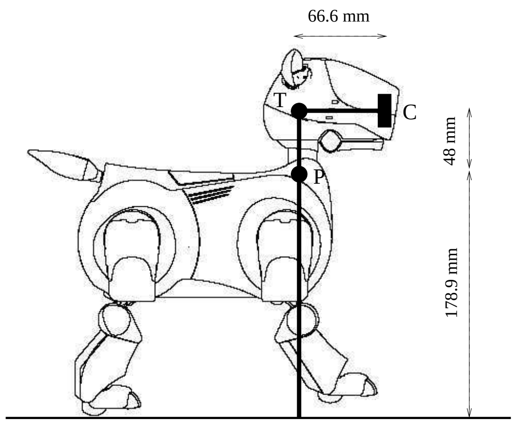
and the relation between the image pixel P_I^c, expressed in c framework, and the 3D world point P^b is:

$$P_I^c = KP^b \tag{4.2}$$

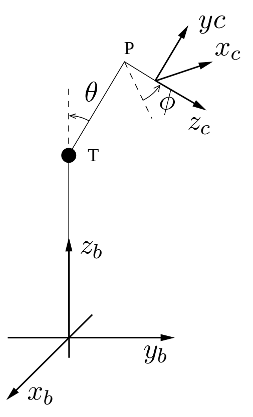
In the remainder of this section, we first discuss the direct kinematics transformation (subsec. 4.2.1) and the perspective transformation (4.2.2). After that, we speak about the points belonging to the real world, and more specifically to the ground (our region of interest), that are actually *visible* from the image plane (4.2.3). Finally, we describe the inverse transformation that allows to infer the ground position of a point that is projected into a given pixel (4.2.4).

4.2.1 Direct kinematics transformation

Direct kinematics transformation is the matrix transformation that allows to express a generic point p from b-coordinates (p^b) to c-coordinates. This relation is expressed by:



(a) Schema of the robot model where P and T are pan and tilt rotation centers and C is the camera. In this figure, the robot has tilt $\theta = 0$ and pan $\phi = 0$.



(b) Position and orientation of {b} and {c} frames

Figure 4.6: *Direct kinematics transformation* is the matrix transformation that allows to express a generic point p from {b}-coordinates (p^b) to {c}-coordinates (p^c)

$$p^c = T_b^c p^b \tag{4.3}$$

In order to compute T_b^c, we need to know the chain of rotations and translations that leads the {b} framework to the {c} framework, see Figure 4.6. Using the AIBO model specifications, we achieved the following intermediate transformations, expressed using homogeneous matrices:

- A_0^b, translation of h along z_b-axis, with $h = (64.9{+}64{+}50) = 178.9$ mm that is the height of the head tilt center from the ground plane, see Figure 4.7(a);
- A_1^0, rotation of $-\frac{\pi}{2}$ around x_0-axis, see Figure 4.7(b);
- A_2^1, rotation of π around z_1-axis, see Figure 4.7(c);
- $A_3^2(\theta)$, rotation around x_2-axis of the *tilt* angle θ[1], see Figure 4.7(d);
- A_4^3, translation of p along y_3-axis, with $p = 48$ mm that is the difference between heights of the head pan center and the head tilt center from the ground plane, see Figure 4.7(e);

[1] θ is measured in the opposite way with respect to the convention assumed in Fig. 3.5(b)

- $A_5^4(\phi)$, rotation around y_4-axis of *pan* angle ϕ, see Figure 4.7(f);
- A_c^5, translation along z_5-axis of $l = 66.6$ mm that is the distance between the camera plane and pan rotation center, see Figure 4.7(g).

The multiplication of the previous matrices, expresses the complete transformation that leads {c} to {b} framework.

$$T_c^b = A_0^b A_1^0 \dots A_5^4 A_c^5 \tag{4.4}$$

Since (4.4) expresses the coordinate transformation from {c} to {b} frame, according to the relation

$$p^b = T_c^b p^c, \tag{4.5}$$

we have to invert it to achieve the coordinate transformation from {b} to {c} frame, that is what we actually need.

Inversion of T_c^b is:

$$\begin{aligned} T_b^c &= (T_c^b)^{-1} = \\ &= (A_c^5)^{-1}(A_5^4)^{-1} \dots (A_1^0)^{-1}(A_0^b)^{-1} = \\ &= A_5^c A_4^5 \dots A_0^1 A_b^0 \end{aligned} \tag{4.6}$$

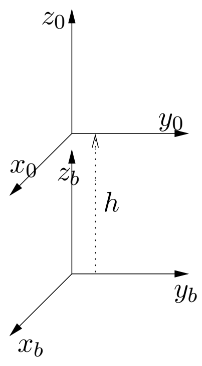
Since each transformation is elementary, i.e., composed by only a rotation or a translation around/along an axis, the inverses are easily computed considering the opposite angle for rotations and the opposite direction for translations.

4.2.2 Perspective transformation

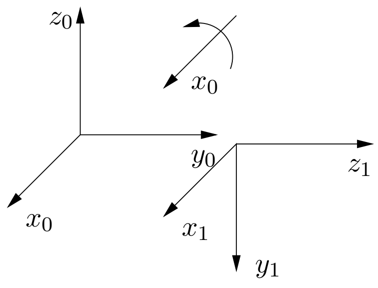
The second component of the *calibration matrix* is the *perspective matrix*, which models the process of projection of a world point, expressed in a framework attached to the image plane (this can be done through the direct kinematics transformation), into the image plane. In order to explicit this matrix, we have used the pinhole camera model that is one of the most widely used in computer vision and robotics (Cf. [7] and [28]).

A point of an object is projected through a pinhole or optical center of a camera onto a unique location on the planar image surface (see Figure 4.9). The equations that describe this projection are:

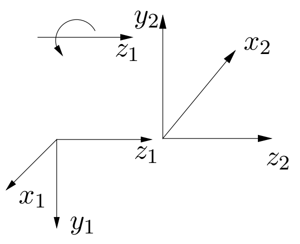
$$\begin{cases} X = \frac{f p_x^c}{f - p_z^c} \\ \\ Y = \frac{f p_y^c}{f - p_z^c} \end{cases} \tag{4.7}$$



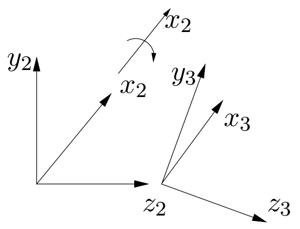
(a) Translation of $h = 178.9$ mm along z_b



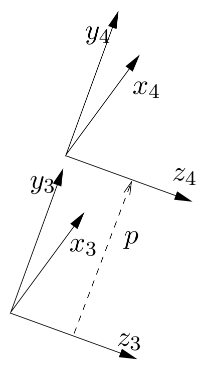
(b) Rotation of $-\frac{\pi}{2}$ around x_0



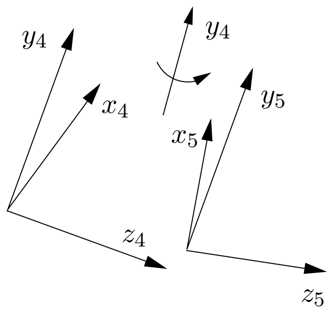
(c) Rotation of π around z_1



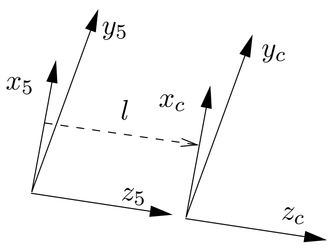
(d) rotation of tilt angle around x_2



(e) Translation of p along y_3

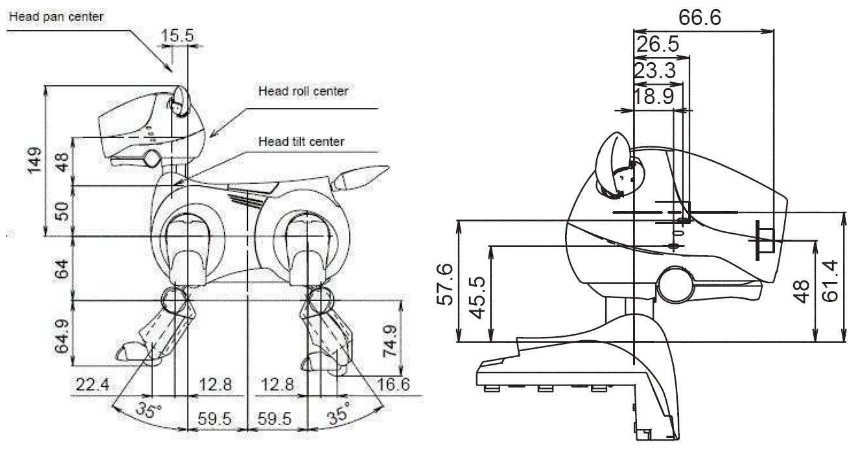


(f) Rotation of *pan* angle around y_4



(g) Translation of $l = 66.6$ mm along z_5

Figure 4.7: Chain of rotations and translations that leads the {b} framework to the {c} framework. Note that each rotation subfigure shows the rotation axis and an arrow indicating the positive rotation direction.



(a) Body model (b) Head model

Figure 4.8: Model specification with measures expressed in mm

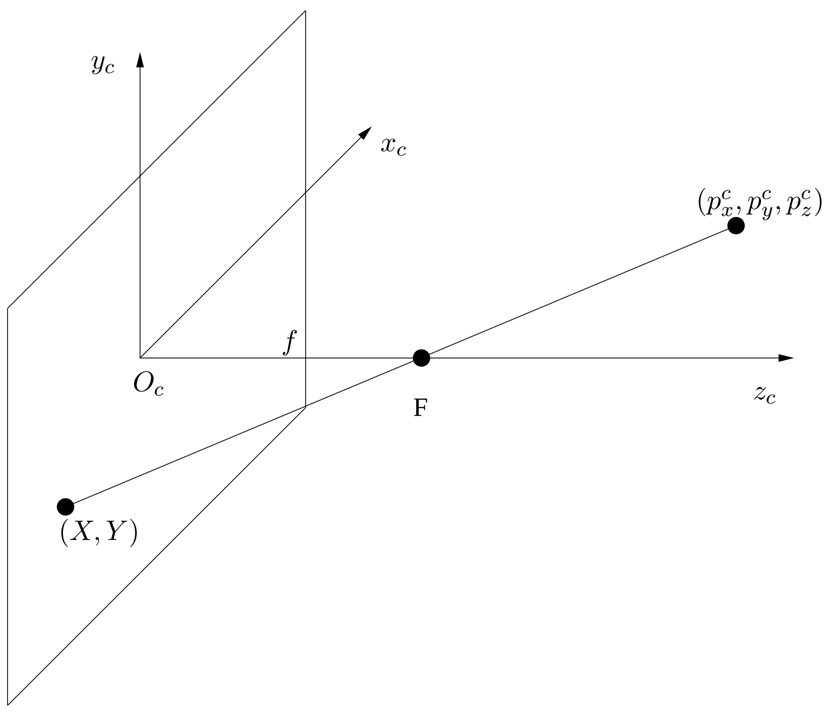


Figure 4.9: Perspective transformation

where (X, Y) are the projected point coordinates and f is the focal length of the camera (intrinsic parameter). These coordinates are expressed in mm, whereas the following equations yield coordinates in pixels:

$$\begin{cases} X_I &= \frac{\alpha_x f p_x^c}{f - p_z^c} + X_0 \\ \\ Y_I &= \frac{\alpha_y f p_y^c}{f - p_z^c} + Y_0 \end{cases} \tag{4.8}$$

X_0 Y_0 are values, expressed in pixel, that allow to move framework {c} center. We decide to move it in the left down corner of the image.

α_x and α_y are scale factors that realizes the conversion between mm and pixels. These values are parameters dependent on the camera sensor used. The camera installed on AIBO is a 1/6 inch format CMOS sensor. This format has an actual sensor diagonal of 2.7 mm (the actual diagonal is not simply 1/6 inch long), e.g., Cf. [18]. The images are gathered at a resolution of 352(H)×288(V) but robot middleware restricts the available resolution to a maximum of 176(H)×144(V), which is the resolution employed in our system.

Thus, knowing the sensor diagonal and the pixel numbers, the square pixel has a side

$$dim = \frac{2.7}{\sqrt{176^2 * 144^2}} = 11.87\ \mu m$$

Scale factors are computed as follows:

$$\begin{cases} \alpha_x &= \frac{1}{dim} \\ \alpha_y &= \frac{1}{dim} \end{cases}$$

This transformation is not linear but it can be written in a linear form exploiting the homogeneous coordinates of a point (x_i, y_i, z_i):

$$\begin{cases} X_I &= \frac{x_i}{z_i} \\ \\ Y_I &= \frac{y_i}{z_i} \end{cases} \tag{4.9}$$

Using homogeneous coordinates for both (X_I, Y_I) and (p_x^c, p_y^c, p_z^c), Eq. (4.7) is reformulated as following:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \Omega \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix} \tag{4.10}$$

with

$$\Omega = \begin{bmatrix} \alpha_x & 0 & X_0 & 0 \\ 0 & \alpha_y & Y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

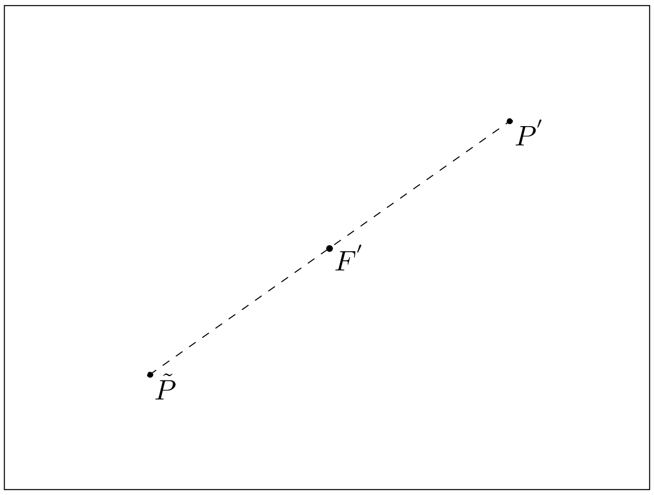


Figure 4.10: Symmetry about focus

However, (X_I, Y_I) found with Eq. 4.9 is not the right result because the pinhole camera model as you can see in Figure 4.9, inverts the projected image. We discuss this problem in the following paragraph.

Y-coordinate inversion

The perspective projection has a "side effect" that is similar to a symmetric transformation about a point: it inverts the position of a point P with respect to the focus (see Figure 4.9).

If $P^{'}$ is the orthogonal projection on the image plane of P and $F^{'}$ the one of F, you can observe in Figure 4.10, that $\tilde{P}$, the perspective projection of P with respect to F, is symmetric to P' with respect to F'. Thus, in order to map the real world in the proper way (not upside down) it is necessary to apply another transformation.

Y-coordinate must be inverted to see the image downside up, whereas the X-coordinate must not be changed for the following reason:

image to be in the focus center and looking at the 3D world, in order to see the same scene projected onto the image plane we need to turn back. In this way what it was for instance on the left in the 3D world, is projected again on our left onto the image plane. On the contrary, turning back, what was up in the 3D world is projected down onto the image plane and that is the reason we need a Y-inversion.

According to these considerations, inversion around the center of the image, (X_0, Y_0), is needed only for the y-coordinate.

Eq. (4.8) is then substituted by the following:

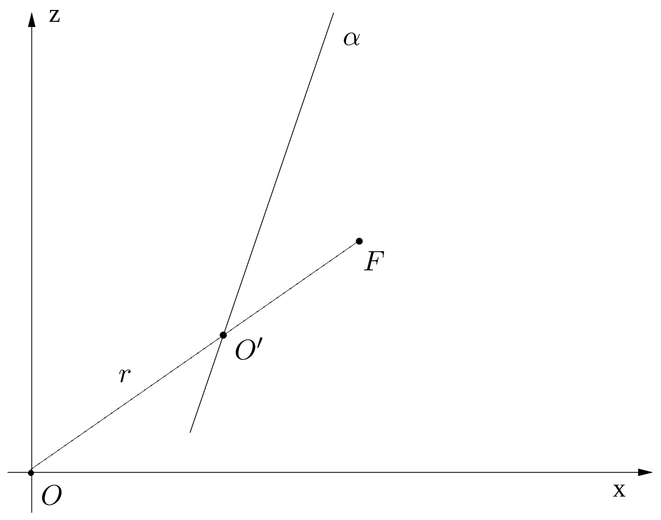


Figure 4.11: 2D lateral section of the world, where α is the image plane, F the focus point of the pinhole camera model and O is the origin of {b} frame. This is a counter example that shows that O projection, O', is not correctly projected because F is not between O and O' along the projection line r.

$$\begin{cases} X_I &= \frac{\alpha_x f p_x^c}{f - p_z^c} + X_0 \\ Y_I &= \frac{-\alpha_y f p_y^c}{f - p_z^c} + Y_0 \end{cases} \tag{4.12}$$

and the perspective matrix becomes:

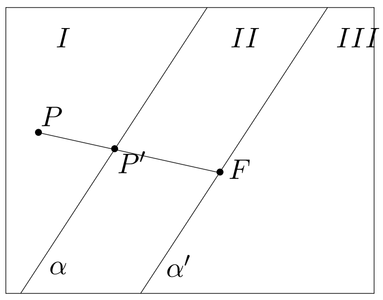
$$\Omega = \begin{bmatrix} \alpha_x & 0 & X_0 & 0 \\ 0 & -\alpha_y & Y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

4.2.3 Visible field

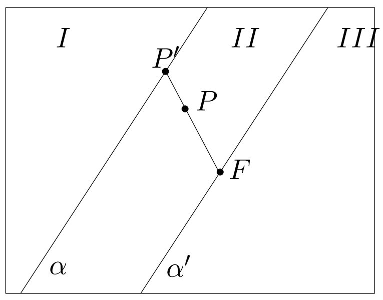
Now that we have found the calibration matrix, it is easy to map a point of the world into the correspondent pixel into the image. However, the domain of the transformation is not the whole $\Re^3$ space.

The following is a counter example, see Figure 4.11: point $O = [0\ 0\ 0]^T$ is not correctly projected (accordingly with the pinhole camera model), because line r that contains O and F, intersects the image plane α in a point O' that is between O and F. O' could be accepted only if F were between the point to project O and the projected point O', along line r.

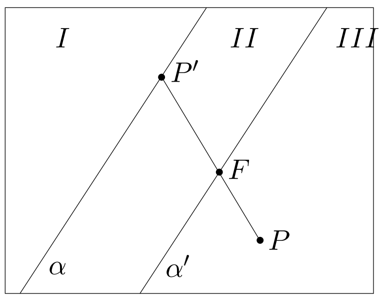
It is useful to formulate the following:



(a) Point P in region I is not *visible* because F is not in PP'



(b) Point P in region II is not *visible* because F is not in PP'



(c) Point P in region III is *visible* because F is in PP'

Figure 4.12: Lateral sections of the 3D world, with the image plane α, the focus F, the plane α' parallel to α passing through F and a point P with its projection P'. P is *visible* only if it is located in region III.

Definition 1. *A point P is called* ***visible*** *from a plane α with respect to a focus F, if the line r that contains P and F intersects α in a point P' such that F belongs to the segment PP'.*

In Figure 4.12(c) point P is visible because F belongs to segment PP'.

The locus of visible points Γ, according to Definition 1, is deduced by the following theorem:

Theorem 1. *The subset of $\Re^3$ that is the locus of* ***visible*** *points from the image plane α with respect to a point F, is the half-space of $\Re^3$ that is bounded by the plane α' parallel to α passing through F and that does not contain α.*

Proof. The possible regions of analysis are I, II, III, planes α and α' (see Figure 4.12):

- all points of III are *visible* by definition;
- a generic point P in II yields to a segment $P'PF$, therefore P is not visible by definition;
- a generic point P in I leads to a segment $PP'F$, thus P is not visible by definition;
- P in α leads to $P \equiv P'$ and then to $PP'F$ or $P'PF$, so P is not visible by definition;
- with P in α', P' cannot be determined because line r connecting P and F is parallel to α.

□

According to Theorem 1, if a point P is not in the region Γ of visibility, its projection is not determinable.

So far we have analyzed the *visibility* of points in $\Re^3$, in the following paragraph we will focus on the subset of Γ that includes only points laying onto the ground plane, which is the world region of interest of our system. We call this subset *ground visible field*, Λ.

Ground visible field

The purpose of this paragraph is to find a sufficient condition that guarantees that a ground point $P(x, y, 0)$ is visible from the image plane characterized by a pan ϕ and a tilt θ angle. This is achieved, for instance, defining a subset of the ground visible field that depends only on the tilt angle θ of the image plane, whereas the pan angle ϕ can have any value of its range.

Tilt and pan angles of interest for this project are θ[2] $\in [-\frac{\pi}{2}, 0]$, $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, (the angle ranges actually employed in the implementation are slightly different, see subsec. 3.2.3).

Let $\alpha_{\phi,\theta}$ be the image plane with pan ϕ and tilt θ, we call $\Sigma^F_{\phi,\theta}$ the ground visible field from the image plane $\alpha_{\phi,\theta}$ w.r.t. the focus point F:

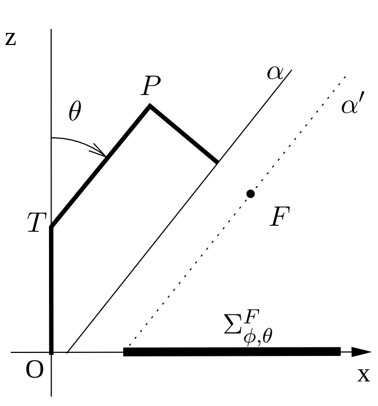
$$\Sigma^F_{\phi,\theta} = \{P(x, y, z) | z = 0 \land P \textit{ is visible from } \alpha_{\phi,\theta} \text{ w.r.t. } F\}$$

The region we are interested in, is:

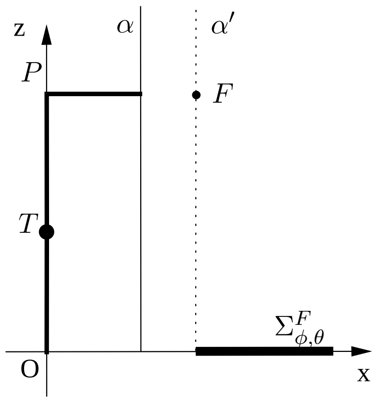
$$\Lambda = \Sigma^F_{\phi} = \bigcap_{\theta \in [\frac{-\pi}{2}, 0]} \Sigma^F_{\phi,\theta}$$

and as you can see in Figure 4.13

[2] Here θ is measured accordingly with the tilt-angle AIBO's convention, see Fig. 3.5(b). From now on, we will always adopt this convention.



(a) Ground visible field $\Sigma^F_{\phi,\theta}$ when $\theta < 0$



(b) Ground visible field $\Sigma^F_{\phi,\theta}$ when $\theta = 0$

Figure 4.13: Lateral section of the 3D world with the robot model. T and P are respectively the pan and tilt rotation center. α is the image plane and α' is its parallel through the focus F. The visible ground field r is minimum when $\theta = 0$

$$\Sigma^F_{\phi} = \Sigma^F_{\phi,\theta=0},$$

Hence, Σ^F_{ϕ} is more easily computed studying $\Sigma^F_{\phi,\theta=0}$ with $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

Consider now a frame {XYZ} obtained by rotating {xyz} (the {b} frame) around z of the current ϕ pan angle. From Figure 4.14 and Figure 4.15 it is clear that the ground visible field is the half-plane $Y > g$ with $g = l + f = 68.16$.

Changing framework from $\{XOY\}$ to $\{xOy\}$ with a rotation of $-\phi$, the above relation is expressed as follows:

$$Y = -x \sin \phi + y \cos \phi > g \tag{4.14}$$

Thus, (4.14) is the requested sufficient condition that guarantees that a ground point P(x, y, 0) is *visible* from the image plane $\alpha_{\phi,\theta}$ with respect to focus point F for every admissible value of θ and ϕ.

4.2.4 Inverse-transformation: from pixels to real world ground points

In the previous section we have analyzed how to convert a point from the 3D real world to the correspondent pixel into the image plane using the *calibration matrix*. In this section we call $\mathcal{F} : \Gamma \subset \Re^3 \longmapsto \Re^2$ the function that executes this transformation, where Γ is the visibility region.

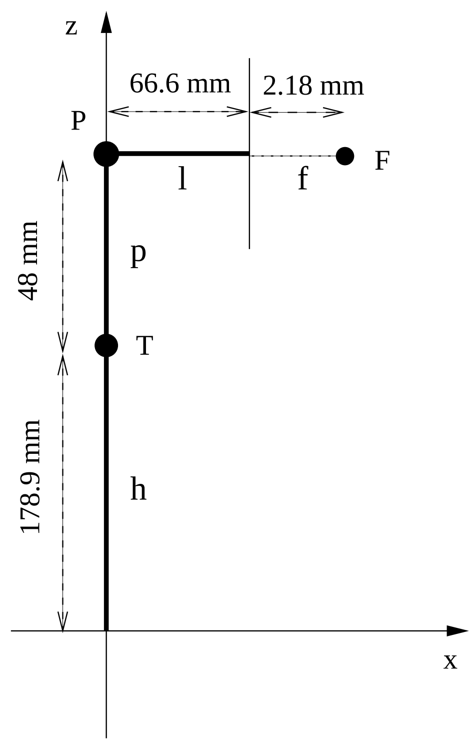


Figure 4.14: Robot model with joint lengths and pan and tilt rotation centers, P and T respectively. Note that in this picture $\phi = 0$

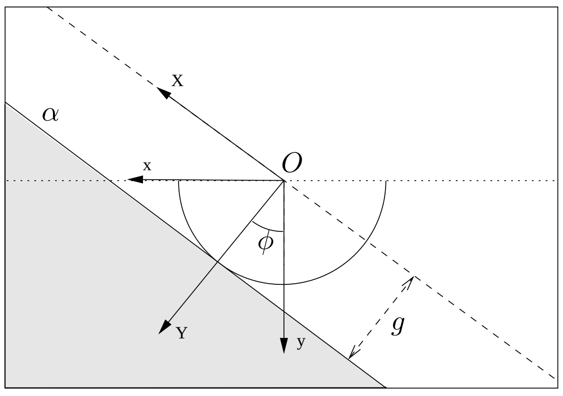


Figure 4.15: Top view of the ground. The shadowy region identifies the ground visible points from the image plane α with pan ϕ and tilt $\theta \in [-\pi, 0]$. {xOy} is aligned with {b} and {XOY} is aligned in such a way that X-axis is parallel to α

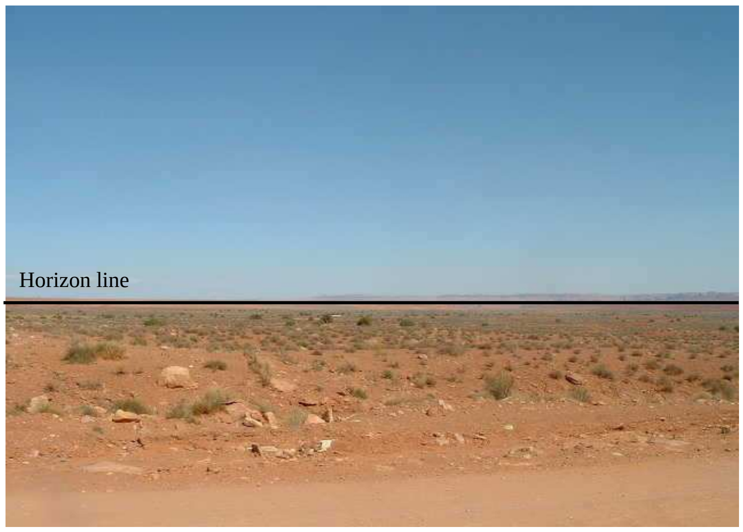


Figure 4.16: Horizon is the imaginary line that splits the image in two parts: the lower with points that are projections of the ground and the upper with points that are not.

We are also interested in knowing the inverse transformation of $\mathcal{F}$, that allows to convert a pixel of the image plane to the correspondent point in the 3D world.

However, $\mathcal{F}$ is not an injective function, since an image plane point is the projection of an infinite number of points in the 3D space. Thus, to be able to compute $\mathcal{F}^{-1}$, we need to consider only ground points for reducing the domain of $\mathcal{F}$ (we shall discuss this in the next subsection).

In the last two subsections, we first speak about image pixels that are not projections of ground points, and finally, we describe the mathematical procedure that allows to achieve the inverse transformation.

Injective property: ground plane assumption

The projection of a line l passing through the focus F is a single point P', i.e., P' is the projection of all the points that belongs to l. The only way to make $\mathcal{F}$ injective is to set a rule to choose unequivocally a point among all the infinite points P of l.

In our case, the employed rule deals with the "ground plane assumption", that means that we consider only points that lie on the ground plane. This implies that the domain of $\mathcal{F}$ is reduced from the visibility region $\Gamma \subset \Re^3$ to Λ.

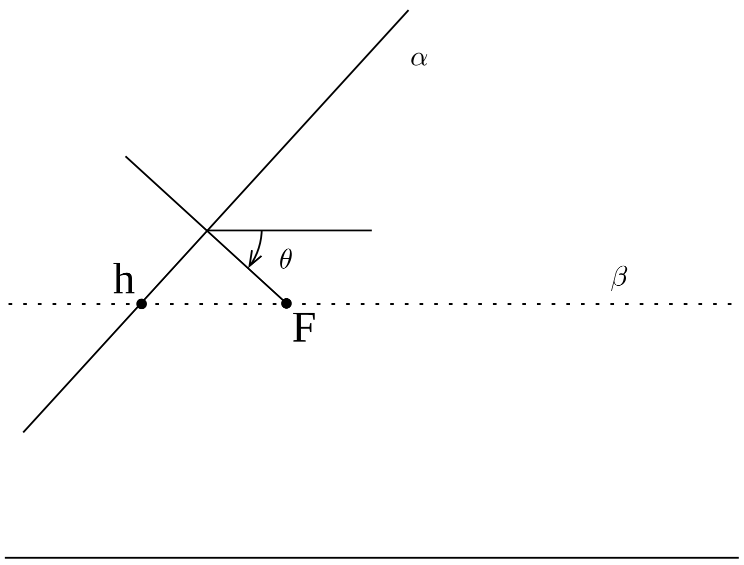


Figure 4.17: Horizon line is the projection into the image plane α of a plane β parallel to the ground and passing through the focus F

Horizon line

The "ground plane assumption" that we have used to make our transformation injective has a drawback: it makes $\mathcal{F}$ also non-surjective. Indeed, there are points on the image plane that are not projection of any points on the ground. See Figure 4.16, it shows a view of a landscape split in two portions: the ground and the sky. We call in this work *horizon* the imaginary separation line.

In the picture all the pixels above the horizon cannot be the projection of a ground point, therefore in order to compute the inverse transformation, the codomain of $\mathcal{F}$ must be reduced considering only points under the horizon line.

The horizon line can be seen as the projection into the image plane of the plane parallel to the ground passing through the focus point. In case the image plane has pan angle $\phi = 0$, the horizon is an horizontal line and is computed easily. On the contrary, with a non-null pan angle, the way to achieve the horizon equation requires further considerations.

Null pan angle The projection of an horizontal plane is an horizontal line with equation in the image framework expressed as:

$$y = y_0 + f \tan\theta \tag{4.15}$$

where y_0 is the y-coordinate of the image center, f is the focal length and $\theta \in [-\pi, 0]$, see Figure 4.17.

Non-null pan angle For a non-null pan angle we decide to achieve the horizon line equation by mapping two ground points $P_1(x_1, y_1, 0)$ and $P_2(x_2, y_2, 0)$ in two distinct image pixels.

P_1 and P_2 are projected into the horizon line only if their distances from the robot reference is infinite. Provided that, under a perspective projection, parallel lines converge to the same point on the horizon, we need to carry P_1 and P_2 to infinity along lines with different direction.

A line laying on the ground and passing through the point $(x_0, y_0, 0)$ with direction $v = (v_x, v_y, 0)$ can be described as the set of points

$$P_\lambda = \begin{pmatrix} x_0 + \lambda v_x \\ y_0 + \lambda v_y \\ 0 \\ 1 \end{pmatrix},$$

with λ between $-\infty$ and ∞. The generic projection $I_\lambda(u, v, w)$ of P_λ, expressed in homogeneous coordinates, is given by (Cf. Eq. 4.2):

$$I_\lambda = KP_\lambda \tag{4.16}$$

where K is the calibration matrix of the image plane.

Making (4.16) explicit, we obtain:

$$\begin{cases} u &= (x_0 + \lambda v_x)K_{11} + (y_0 + \lambda v_y)K_{12} + K_{14} \\ v &= (x_0 + \lambda v_x)K_{21} + (y_0 + \lambda v_y)K_{22} + K_{24} \\ w &= (x_0 + \lambda v_x)K_{31} + (y_0 + \lambda v_y)K_{32} + K_{34} \end{cases} \tag{4.17}$$

where K_{ij} is the element at i^{th}-row and j^{th}-column of K.

Expressing $I_\lambda(U, V)$ in non-homogeneous coordinate:

$$\begin{cases} U &= \frac{(x_0+\lambda v_x)K_{11}+(y_0+\lambda v_y)K_{12}+K_{14}}{(x_0+\lambda v_x)K_{31}+(y_0+\lambda v_y)K_{32}+K_{34}} \\ \\ V &= \frac{(x_0+\lambda v_x)K_{21}+(y_0+\lambda v_y)K_{22}+K_{24}}{(x_0+\lambda v_x)K_{31}+(y_0+\lambda v_y)K_{32}+K_{34}} \end{cases} \tag{4.18}$$

As $\lambda \to +\infty$, Eq. (4.18) becomes:

$$\begin{cases} U &= \frac{v_x K_{11}+v_y K_{12}}{v_x K_{31}+v_y K_{32}} \\ V &= \frac{v_x K_{21}+v_y K_{22}}{v_x K_{31}+v_y K_{32}} \end{cases}$$

$I^v_\infty(U, V)$ is called *vanishing point* associated with the family of straight lines with direction $v(v_x, v_y, 0)$ (Cf. [22]).

Now that we have a method to find out two different *vanishing points*,computing the horizon line is a straightforward task.

Here there are two possible directions that are symmetric with respect to the image plane normal:

$$v_1(\sin(\phi + \tfrac{\pi}{4}), \cos(\phi + \tfrac{\pi}{4}), 0)$$
$$v_2(\sin(\phi - \tfrac{\pi}{4}), \cos(\phi - \tfrac{\pi}{4}), 0)$$

where ϕ is the pan angle of the image plane. We compute the corresponding *vanishing points* I_∞^{v1}, I_∞^{v2} and the requested horizon passes through them.

Inverse transformation computation

The *ground plane assumption* and the *horizon line* introduced in the previous paragraphs have made the function $\mathcal{F}$ both injective and surjective, hence we are able now to compute $\mathcal{F}^{-1}$.

Inverse transformation can be formulated starting from the intersection between the Eq. (4.2) and the plane equation (Cf. [7]).

(4.2) can be rewritten as:

$$(u, v) = \left(\frac{(KX)_1}{(KX)_3}, \frac{(KX)_2}{(KX)_3} \right) \tag{4.19}$$

where (u,v) is the projected point on the image plane expressed in pixels, K is the calibration matrix and $(KX)_i$ is the i-element of the KX vector.

The 3D-point (x, y, z) that gives rise to the projection (u, v) can be found by solving

$$\begin{bmatrix} (uK)_3 - (K)_1 \\ (vK)_3 - (K)_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.20}$$

where $(K)_i$ is the i^{th} row of the K matrix.

Intersecting (4.20) with the ground plane equation $A\mathbf{x} = 0$ with $A = [0\ 0\ 1\ 0]$, we obtain:

$$\begin{bmatrix} K_1 - uK_3 \\ K_2 - vK_3 \\ A \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.21}$$

In a scalar form (4.21) becomes:

$$\begin{cases} (K_{11} - uK_{31})x + (K_{12} - uK_{32})y + K_{14} - uK_{34} & = & 0 \\ (K_{21} - vK_{31})x + (K_{22} - vK_{32})y + K_{24} - vK_{24} & = & 0 \end{cases} \tag{4.22}$$

where K_{ij} is the element at row i and column j of K.

(4.22) is a system of two equations with two unknowns x, y (u and v are known).

In matrix form:

$$M \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{q} = 0 \tag{4.23}$$

solvable inverting M matrix:

$$\begin{bmatrix} x \\ y \end{bmatrix} = -M^{-1}\mathbf{q} \tag{4.24}$$

with

$$M = \begin{bmatrix} K_{11} - uK_{31} & K_{12} - uK_{32} \\ K_{21} - vK_{31} & K_{22} - vK_{32} \end{bmatrix}$$

and

$$\mathbf{q} = \begin{bmatrix} uK_{34} - K_{14} \\ vK_{24} - K_{24} \end{bmatrix}$$

Note that M is invertible because we have assumed so far that $\mathcal{F}$ is bijective.

Concluding, eq. 4.24 gives the (x, y) position on the ground of the point that corresponds to pixel (u, v) in the image.

4.3 Obstacle range computation: first method

In order to implement a robust system for the robot navigation in an unknown space, we need to know where obstacles are located before being able to plan and execute a path.

[16] has developed a technique, called *visual sonar*, to identify obstacle position on the ground using a single image as input. We have adopted this technique, extending it to manage a set of 6 images as input. This set of images is taken with different pan/tilt angles such that approximately a field of view of 180° horizontally and 90° vertically is covered (see Subsec. 3.2.3).

The main idea of *visual sonar* is to draw a set of scanlines in the image that correspond to lines on the ground plane emanating from the origin of an egocentric robot framework, i.e., the $\{b\}$ frame. We use scanlines that are spaced by 5° each other around the robot, because we consider it a reasonable trade off between accuracy of obstacle locating and computational load. We are also supported in our choice by the work of [16].

Each scan line is analyzed onto the segmented image (see Subsec. 4.1) for searching a non-ground pixel, i.e., obstacle pixel. Once an obstacle is found, its distance is computed through the *calibration matrix* of the camera (see Sec. 4.2) and its angle, w.r.t. the local reference framework, is extrapolated from the analyzed scan line.

In the next subsections, we explain, in a logical order, the procedure that allows to find an obstacle distance starting from the related scan line on the ground. In fact, we discuss how we map a ground scan line into a line located in the image plane (4.3.1); how we extract the segment belonging to the image rectangle from the related line lying onto the image plane (4.3.2); how we actually represent this segment in pixels (4.3.3); how we find a pixel representing an obstacle and how we compute its distance on the ground (4.3.4); finally, we provide a motivation for the fact that we consider only the first obstacle pixel found on a scan line into the image (4.3.5).

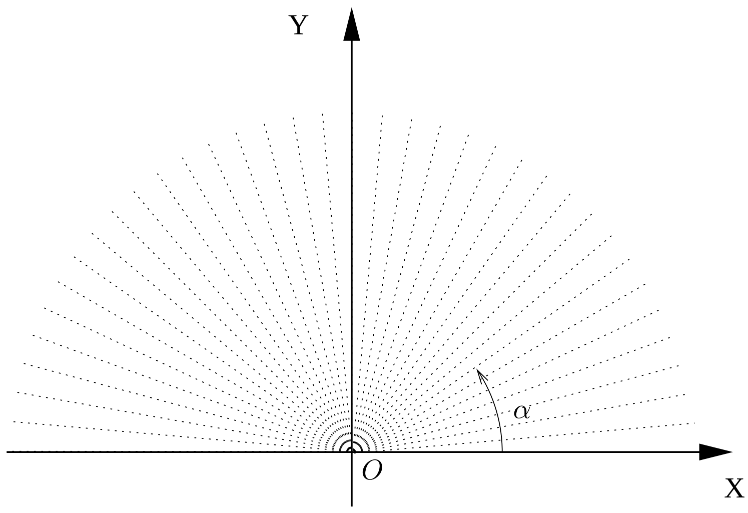


Figure 4.18: Origin O emanates a pencil of concentric scan lines spaced with 5 degrees. α angle is measured counter clockwise starting from x-axis.

4.3.1 Mapping a ground line to the image

We consider scanlines as a pencil of lines on the ground that must be mapped onto the image plane. The center of the pencil is the reference point for the dog, that in egocentric coordinate can be simply set to $O = [0\ 0\ 0]^T$, the $\{b\}$ origin, (see Figure 4.18).

A line is uniquely identified by a vector and an application point. The easiest application point that can be chosen is O because it is common to all the scan lines, but it is not always *visible* for all pan and tilt angle of the image plane (see 4.2.3). For this reason we need to choose a point enough far from O such that it satisfies relation (4.14).

Once we know the first point $P_1(x_1, y_1)$ and the vector indicating the line direction of the i^{th} scan line:

$$v_i = \begin{bmatrix} \cos(i\Delta\alpha) \\ \sin(i\Delta\alpha) \\ 0 \end{bmatrix}$$

with $\Delta\alpha = 5°$, the second point is computed as $P_2 = P_1 + \gamma v_1$ (γ is a scale factor big enough to avoid rounding errors with the transformation when P_1 and P_2 are too close to each other). Both P_1 and P_2 are converted in pixels on the image plane to P_1^I and P_2^I. It is not necessary that P_1^I and P_2^I belong to the image rectangle, because at the moment we are only interested in tracing a line on the image plane that corresponds to a direction on the ground.

The scan line vector is obtained as follows:

$$v^I = P_2^I - P_1^I$$

4.3.2 Clipping the line into the image rectangle

The line found in the previous paragraph must be clipped to fit the image rectangle. The aim of this section is to find the two endpoints of the clipped line and to decide whether such segment is still suitable for our purpose or we can discard it.
We have two possible cases:

- line intersects the rectangle image, then the found endpoints are used to draw and track the scan line
- line does not intersect the rectangle image, hence it is useless to the visual sonar technique and it can be discarded for the considered image.

The clipping process finds for each side of the rectangle the intersection with the line and it stops whenever it reaches 2 intersection points.

The algorithm needs to manage also singular cases, i.e., line passing through vertexes as shown in Figure 4.19:

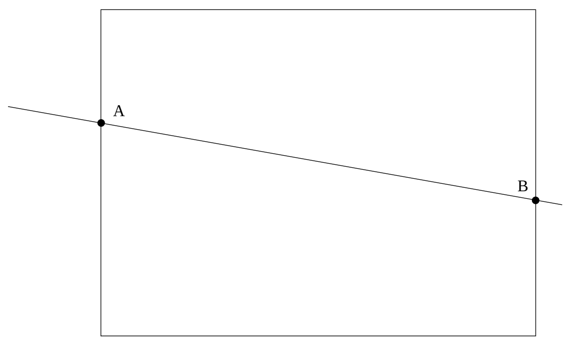
- only two coincident intersections, see Figure 4.19(b): the line is discarded because it is outside the image;
- two coincident intersections and one isolated, see Figure 4.19(c): the line is inside the image and it must be kept.

The found segment must be intersected again with the horizon line, because we are interested only in points on the ground.
If the intersection with the horizon line is inside the image, the scan line does not have to go beyond it. Figure 4.20(a) shows an example with the intersection C between A and B. In this case AC is the segment of scan line. On the contrary, if C does not belong to AB, we keep AB as scan line.

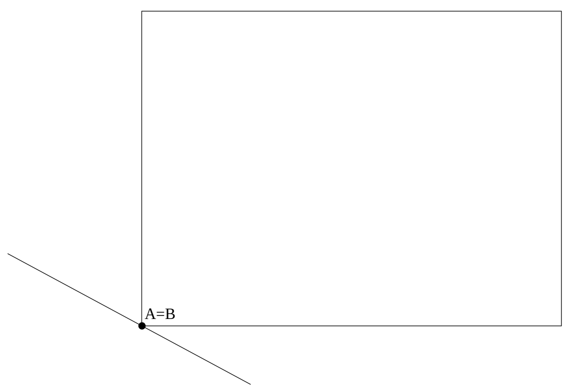
4.3.3 Representation of a segment into an image: DDA Algorithm

Once we know the first and the last pixel in the image plane of the scan line, we use the DDA (Digital Differential Analyzer) algorithm to find out all the points of the line connecting them (Cf.[10]).

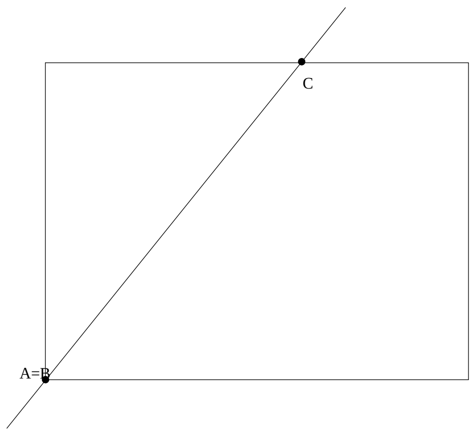
Let (x_a, y_a) and (x_b, y_b) be the endpoints of the line segment to be drawn. Figure 4.21 shows that depending on the slope of a line, only one pixel must



(a) 2 distinct intersections

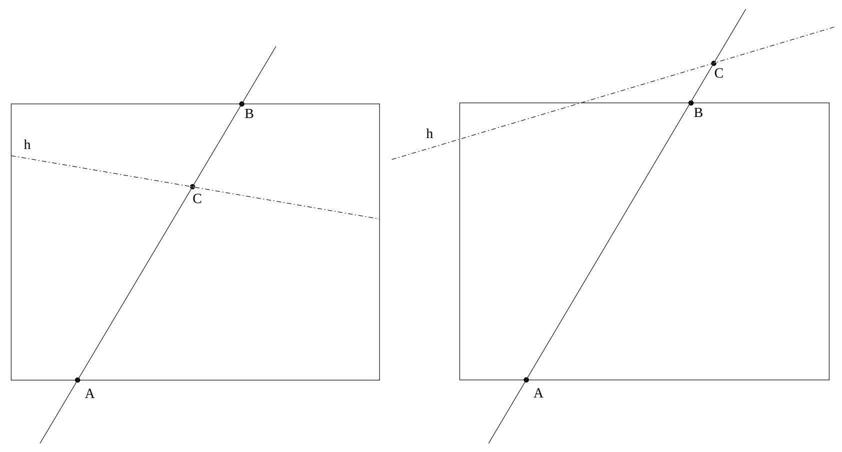


(b) 2 coincident intersections



(c) 2 coincident and one isolated intersections

Figure 4.19: The clipping window algorithm must manage the three kinds of intersection of the line with the window



(a) Intersection inside the image (b) Intersection outside the image

Figure 4.20: The intersection between the scan line and the horizon h may happen in two ways: inside the image (a) when C belongs to AB or outside (b) when it does not. The resulting segment is AC in (a) and AB in (b).

be set per row or per column. If the absolute value of the line angular coefficient is less than or equal to 1, then one pixel per column is set, otherwise one pixel per row.

The first point to be set is (x_a, y_a), current x and y assume x_a and y_a value. A new position of the line is found by adding a Δx to x and Δy to y. The next point to be set is determined by rounding x and y to the nearest integer. This process continues, using the values of x and y before rounding, until the end point (x_b, y_b) is reached.

The increments are determined by first calculating the absolute value of $x_b - x_a$ and $y_b - y_a$. If $|x_b - x_a| > |y_b - ya|$, then steps equal to 1 or -1 must be taken in x-direction along the line, and steps equal to the slope of the line, which is less than 1 in absolute value, must be taken in y-direction. In case of a steeper line, steps equals to 1 are taken in y-direction and steps in x-direction are small than 1 in absolute value.
An implementation of the algorithm follows:

```
void lineDDA(xa,ya,xb,yb){
int dx=xb-xa,dy=yb-ya,steps,k;
float x_increment,y_increment,x=xa,y=ya;

if (ABS(dx)>ABS(dy))
steps=ABS(dx);
else
```

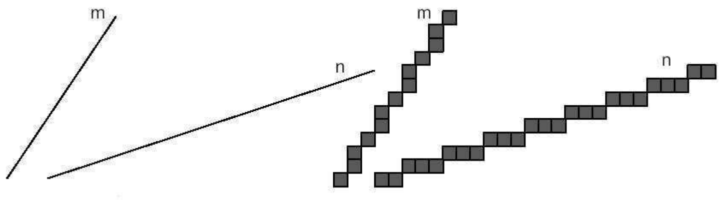


Figure 4.21: Line m has a slope > 1 and it is converted with 1 pixel set per row, line n instead has a slope < 1 and then is converted with a pixel set per column.

5	4	6
2	1	3

Figure 4.22: Sorting of the images that permits to satisfy relation 4.25

```
steps=ABS(dy);

x_increment=dx/step;
y_Increment=dy/step;

x[0]=ROUND(x);
y[0]=ROUND(y);

for(int k=1;k<steps;k++){
x ++= x_increments;
y ++= y_increments;

x[k]=ROUND(x);
y[y]=ROUND(y);
}
}
```

4.3.4 Finding the first obstacle distance along each scan line

Each scan line onto the segmented image is inspected for searching the *first* pixel that has not the same color of the ground (ground pixel are white and non-ground pixel are black in the segmented image).

Before introducing the algorithm used for the extraction of the first obstacle, is useful to formalize a notation for scanlines in the image plane.

A scan line is indicated with S_α, where α is the angle of the corresponding ground line (see Fig. 4.18).

The portion of S_α that is included in image $i \in [1, 6]$ is indicate with S^i_α.

Each scan line S^i_α is made of a sorted set of pixels indicated with $S^i_\alpha(k)$ and $S^i_\alpha(0)$ is the first pixel of the scan line α mapped into image i.

The number of pixel $N(S^i_\alpha)$ for each portion of scan line S^i_α is not constant. $N(S^i_\alpha) = 0$ means that S_α is not visible in image i.

The transformation of a pixel into the related ground point is performed by function $\mathcal{G}^i$, where i indicates that it uses the calibration matrix of the i-image to perform the transformation, i.e., a particular combination of pan, tilt angles.

Using the previous notation $\mathcal{G}^i(S^i_\alpha(k))$ is the ground point corresponding to the k-th pixel of the scan line portion with angle α belonging to $i^{th} - image$.

The set of images are sorted in such a way that, for each scan line S_α, the following statement is true:

$$\begin{gathered} \forall h \exists k \left(k \in [0, N(S^i_\alpha) - 1] \wedge h \in [0, N(S^j_\alpha) - 1] \wedge i < j \right) \\ \longrightarrow \\ \|\mathcal{G}^i(S^i_\alpha(k))\| < \|\mathcal{G}^j(S^j_\alpha(h))\| \end{gathered} \tag{4.25}$$

This means that if $\mathcal{G}^i(S^i_\alpha(k))$ represents the distance of the closest obstacle (w.r.t. the scan line origin) found on scan line α in image i, it is not necessary to search for closer obstacles along α in any image j, with $j > i$.

The order of images that satisfies this statement is illustrated in Figure 4.22. It is easy to validate this order w.r.t. eq. 4.25 if you think at the following example (see also Fig. 4.23). The scan line characterized by $\alpha = 10°$, is always visible only in the lower and upper right tiles of the mosaic image. If an obstacle is found along the portion of this scan line contained in the lower tile, we can say that we will not find any other closer obstacles in the scan line projected in upper tile. In fact, the upper tile has been shot with an higher tilt angle and, for this reason, it cannot show, along a scan line, an obstacle which has not been already detected along the same scan line in the lower tile. Concluding, scan line α must be searched for obstacle first in the lower right tile, and then in the upper right one.

In this way, in order to find the closest obstacle to the reference, the vector $\left[S^1_\alpha S^2_\alpha \dots S^6_\alpha\right]$ is scanned until an obstacle is found. This order allows

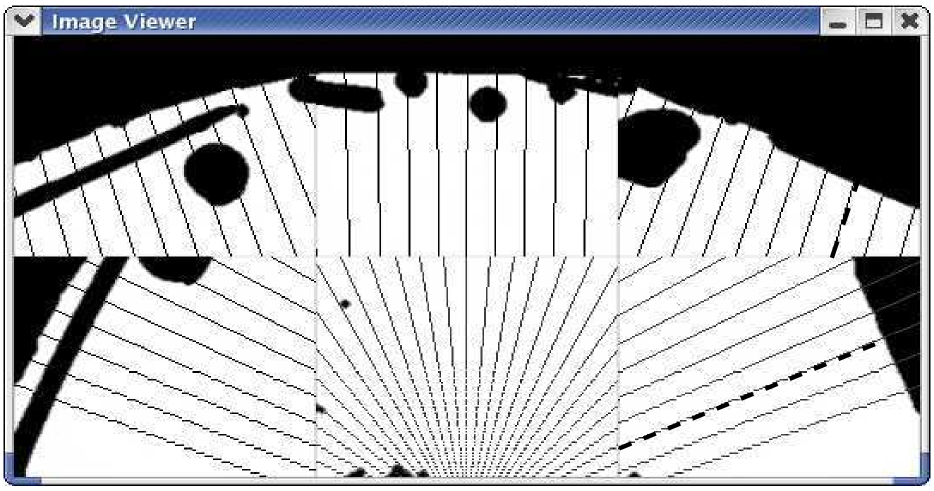


Figure 4.23: Segmented mosaic image showing the projection of ground scan lines that are 5° spaced. The thicker dashed scan line corresponds to $\alpha = 10°$.

to optimize the searching of the closest obstacle along a scan line, avoiding to consider all the images.

Once the first obstacle is found on the scan line S^i_α, the pixel is anti-converted into the ground point $P_\alpha = (x, y, 0)$,using the corresponding calibration matrix related to the i^{th} image. The distance of P_α to the reference is $\sqrt{x^2 + y^2}$ and the angle is α.

On the other hand, if S_α is scanned until the last pixel of S^6_α and there are not found obstacles so far, we can state that there is not any obstacle (visible to the robot) along α direction on the ground.

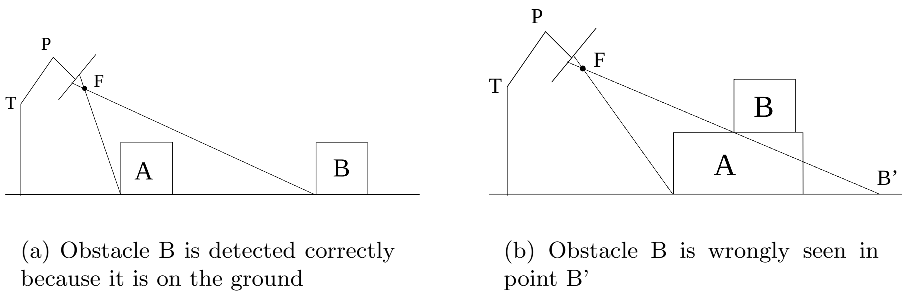
4.3.5 Scanning till the first obstacle pixel

We could find for each scan line, if eventually present, further obstacle pixels beyond the first one, but a range computation for those pixels might lead to errors.

In fact the range computation works correctly only for pixels representing points on the ground plane. An elevated obstacle point would be seen further than its actual position and the correspondent object to which the point belongs, bigger than its actual size, see Figure 4.24(b).

We can state that the first obstacle pixel of a scan line is the projection of a point that lies on the ground because of the continuity with adjacent points.

Hence, in order to avoid committing range errors, we stop scanning the line at the first obstacle pixel because it is the only one we are sure about the correspondent ground point location.



(a) Obstacle B is detected correctly because it is on the ground

(b) Obstacle B is wrongly seen in point B'

Figure 4.24: The visual sonar algorithm could find, if eventually present, a further obstacle beyond the first one, but it would be an error to consider it because we do not have any warranty that pixel is located on the ground.

4.4 Obstacle range computation: second method

Even if [16] refers to its obstacle range detection algorithm as *visual sonar*, actually it is slightly different from the way how a sonar works. In fact, a sonar, after having emitted a radiation cone, measures the distance of the closest obstacle encountered within this cone. Whereas, *visual sonar* only measures the closest obstacle range along a particular scan line. From this point of view, it resembles more a laser range finder than a sonar.

In this section, we describe an alternative method to *visual sonar* that draws inspiration from the *real functioning of sonar*. Subsec. 4.4.1 depicts the algorithm, whereas Subsec. 4.4.2 makes a qualitative comparison between this method and previous one (see Sec. 4.3). An experimental comparison will be given in Sec. 7.3.

4.4.1 Algorithm description

As in the previous method, we ideally decompose the ground half-plane visible to the robot into adjacent $5°$ wide angles. Each of these angles is centered along one of the previous scan lines. The common vertex of all the angles is still the origin of the egocentric frame, $\{b\}$. See Fig. 4.25.

We sequentially scan the obstacle pixels of each tile of the current mosaic image in order to compute their related position on the ground. This is achieved applying to each pixel the inverse transformation $\mathcal{F}^{-1}$ (see Sec. 4.2.4). Of course, we consider only "invertable" pixels, i.e., those lying below the horizon line present in each tile.

Once an obstacle pixel has been "anti-projected" onto a ground point, we are able to state to which angle this point belongs and we temporary store the distance, w.r.t. the origin of $\{b\}$, of such a point only if it has not been found another obstacle pixel whose associated ground point, in the currently considered angle, has a shorter distance. In order to keep track of

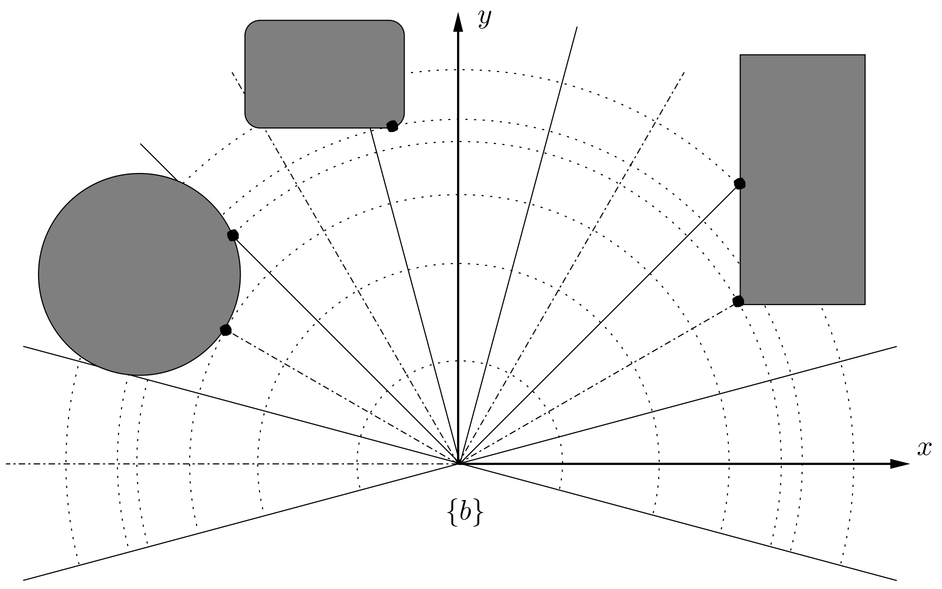


Figure 4.25: *Obstacle range computation: second method.* This figure shows a top view of the ground with some obstacles (filled objects). The continuous lines represent the sides of the *angles*, whereas the dash-dotted lines represent the related bisectors that coincide with the *scan lines* of the visual sonar method. Note that, for clarity of the picture, angles are drawn wider than they actually are in the method. The small filled circles represent the closest object points computed with this method, within each angle. The dotted arcs represent ground points that are equidistant from the $\{b\}$ origin. These arcs can be considered as a sort of *circular wave* propagating from the robot current position, and hitting the closest obstacles.

the minimum distances, we need a number of variables that is equal to the number of angles on the ground.

In this way, at the end of the sequential mosaic search, we obtain, for each angle, the closest found obstacle, as it happens in the case of a ring of sonar.

4.4.2 Comparison

This method is simpler than the previous one since it does not have to deal with mapping of scan lines in the image plane, e.g., DDA algorithm is not needed anymore. Furthermore, starting from a mosaic image, it allows to "extract" more information about ground obstacles since it analyzes all the pixels and not only those lying on the projected scan lines.

There is also a price to be paid: the *computational complexity.* In fact, if you assume that a scan line is projected into a mosaic tile with a number of pixels that is proportional to the square root of the tile image size, i.e., width$\times$height, (for instance, Fig. 4.23 shows that a scan line contained in a tile can be either compared to the width or length of the tile, depending on the scan line angle), the complexity of the previous method is proportional both to such a square root and to the number of scan lines. Whereas, the complexity of the second method is clearly proportional to the size of the tile image. However, it should be noted that if the number of scan lines increases, i.e., if they are less spaced, the complexity of the first method tends to that of the second one.

Chapter 5

Navigation

In this chapter, we speak about *representing* (5.1), *reasoning* (5.2), and *acting* in the environment space, for our navigation purposes. In fact, we discuss the internal environmental representation of our robot (Section 5.1), the way how it plans an obstacle-free path on this representation (5.2), and, finally, we describe how this path is actually mapped into commands for the robot actuator unit (5.3).

5.1 Occupancy Grid

Once we are able to compute obstacle distances, next step is organizing these information such that we can obtain a robot's environment representation that is required for establishing which parts of the environment are free for navigation.

The most natural representation of a robot's environment is a map. In addition to representing places in an environment, a map may include other information, including regions that are unsafe or difficult to traverse, or information of prior experiences. In general spatial representation can be divided into two main groups: those that rely primarily on an underlying metric representation and those that are topological (Cf. [7]). In our case, a metric representation is more suitable since we deal with obstacle distance measures.

Perhaps the most straightforward representation of space is to sample discretely the two-dimensional environment, that consists of a planar ground in this thesis. A method to do this is sampling space at the cells of a uniform grid. Samples taken at points in the lattice express the degree of occupancy at that sample point: is space empty, full, or partially full? These kind of 2D grids are known as *occupancy grids* (Cf. [7]) and this is what we actually employ.

The problem of building an occupancy map from distance measures is made difficult by the uncertainty introduced by the sensing process. Due to the inherent limitations of our visual sensor (see Subsection 5.1.1), it is not always possible to decide whether or not a given point in the workspace is occupied by an obstacle. Rather than *deciding*, a more reliable approach is to convey all the available knowledge into an uncertain representation. In literature there are some methods to deal with uncertainty, e.g., probability theory, Dempster-Shafer theory, but some studies (e.g., Cf. [21]) clearly indicate that, with respect to other methodologies, fuzzy logic provides a more robust and efficient tool for managing the uncertainty introduced by the ultrasonic and laser-based sensing processes that are "similar" to our "visual sensor" to some extent (See Subsec. 5.1.1). In this way, each grid cell can be labeled as *empty*, *occupied*, *indeterminate* (unexplored), or *ambiguous*(giving a discordant information over several measures). The representation of these concepts, their quantification and the construction of suitable uncertainty models are very natural and straightforward in a fuzzy logic framework whereas techniques based on probability theory essentially do not discriminate between indeterminate and ambiguous cells; moreover, their computational load is generally higher [8].

Our navigation system exploits an existing code (Cf. [25]), designed for sonar sensors, that implements a fuzzy occupancy grid.

5.1.1 Employed technique: Fuzzy Logic

The above cited code defines the empty and the occupied spaces by two fuzzy sets E and O over the universal set U (the environment), assumed to be a bitmap, i.e., a two-dimensional subset of $\Re^2$ discretized in square cells of a given size. The corresponding membership function $\mu_E(C)$ and $\mu_O(C)$ quantify the degree of belief that the cell $C \in U$ is empty or occupied, respectively, as computed on the basis of the available measures.

In the fuzzy logic context, the two sets E and O are not complementary. Thus, for a given cell C, $\mu_E(C)$ and $\mu_O(C)$ convey independent information. This situation is particularly convenient in view of the peculiar characteristics of the range sensing process. In fact, any distance measure refers to the closest obstacle along a particular scan line (see Sec. 4.3) or within an angle centered on a particular scan line (see Sec. 4.4), thereby indicating the presence of an empty space up to a certain distance. No information is provided about the state of the area beyond such distance: the available evidence does not suggest neither emptiness nor occupancy. Only incorporating measures taken at different viewpoints it will be possible to discriminate between the two possibilities. A further advantage of the use of fuzzy logic is that the two basic sets E and O can be combined in various ways to identify conflicting or insufficient data.

The map is built by calling the function `GmFuseRange` (Cf. [25]) whenever a distance measure is available. The function parameters are:

- a reference to the E and O sets;

- the coordinate (in mm) of the observation point (`ObsPoint`), i.e, the origin of the scan lines, expressed in the occupancy map framework;
- the range `r`, w.r.t. the scan lines origin, and bearing `theta`, w.r.t. to a local observation point frame oriented like the occupancy map framework, of the detected obstacle. In the case of *visual sonar*, `theta` refers to the particular scan line hitting the obstacle, as depicted in Fig. 5.1. Whereas, considering the second obstacle detection method (see Sec. 4.4), `theta` identifies the bisector of the angle within which the measure has been read.

See Fig. 5.1 for a representation of the above parameters.

Given the i-th distance measure ($i = 1, 2, ...$), `GmFuseRange` essentially performs the following operations:

1. generating a *local representation* (w.r.t. the current observation point) of the empty and the occupied space, i.e, two local fuzzy sets E_i and O_i. These take in account what is usually called *sensor model*, i.e., the uncertainty model of the visual sensor.
2. *fusing* the previous local information into the global representation of the empty and the occupied space, which is contained in E and O.

We are going to discuss the above phases in next two subsections.

Sensor model

The performance of the visual sensor may be affected by various phenomena:

1. the whole segmentation process may alterate the location of an obstacle inside an image. This is due, for instance, to misclassified pixels or pixels filtered out by the final erosion.
2. imprecise knowledge about the calibration matrix, e.g., imprecise movements of the robot head joints, lens distorsion;
3. impossibility to state the exact location of a ground point, starting from the related pixel in an image, because of the intrinsic finite resolution of the image itself.

Another strong uncertainty source arises from the imprecise knowledge of the current robot position inside the grid map, and consequently of the observation point of a certain distance measure. This kind of uncertainty cannot be strictly incorporated in the sensor model because it does not affect directly an obstacle distance observed from the current scan line origin, but it deeply affects the location of that obstacle in the occupancy grid since it influences the observation point provided to the above discussed `GmFuseRange`

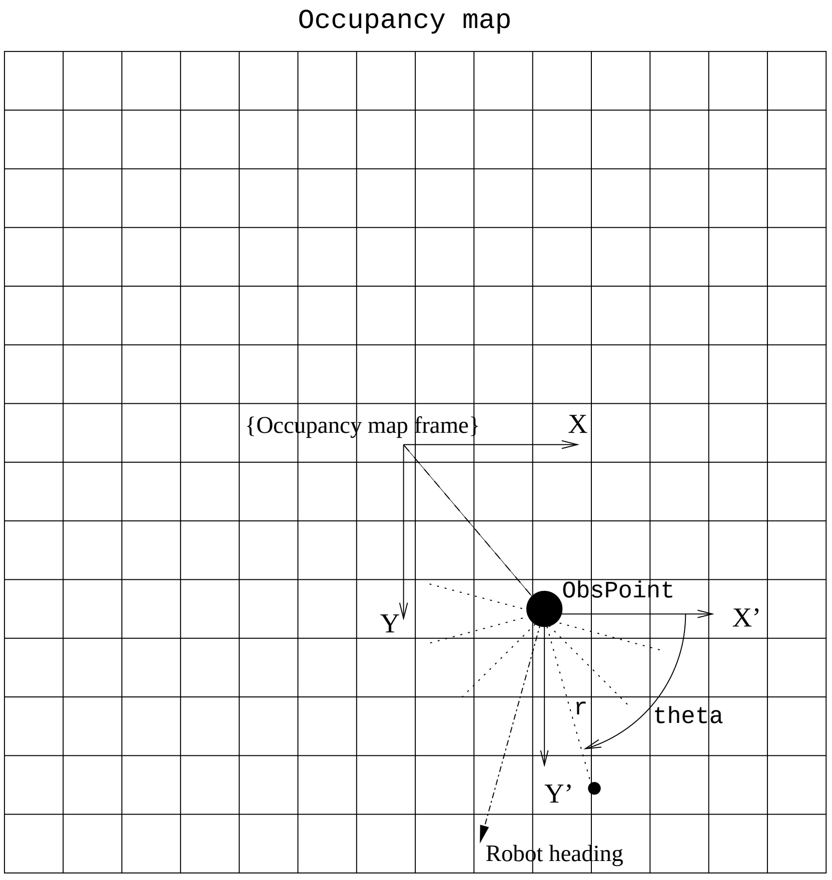


Figure 5.1: Occupancy map framework {X,Y} and parameters needed to call `GmFuseRange`. The dotted lines represent the scan lines "emanated" from `ObsPoint`. One of them is hitting an obstacle (small circle) at distance `r` and angle `theta`, w.r.t. to the local observation point frame {X',Y'}. `theta` is easily computed since we know the angle of the hitting scan line w.r.t. to current robot heading (drawn with a dash-dot line), which is also known through the self-localization process (see Ch. 6).

function. Moreover, this uncertainty has a similar nature to the one that characterizes the distance measures (excluding the segmentation uncertainty source) because the self-localization, which is responsible of providing the current observation point, is based on a visual process exploiting the calibration matrix as well (see Chapter 6). Finally, even if this self-localization uncertainty is not directly modelled, the general fuzzy framework we use, may take it in account somehow.

In the remainder we introduce a basic uncertainty model (implemented in [25]) that associates to the i-th range measure r_i a *prototypical* representation of the empty and occupied space in the "*radiation cone*", i.e., E_i and O_i respectively. This model is usually employed with ultrasonic and laser sensors and, in this situation, the term *radiation cone* has a clear physical meaning. In our case, the *radiation cone* embodies the *angular uncertainty* related to r_i and we consider it as 5° wide. In the case of the second method of obstacle detection, this choice is motivated by the fact the such method only provides, as output, a measure of the closest obstacle within a certain 5° wide angle. In the *visual sonar* case, this choice is motivated by the inaccuracy related to the difference between a real ground scan line, for instance characterized by angle α, and the "anti-projection" of the scan line α lying into the image plane. Experimental evidence (see Sec 7.3) has shown that such difference can be roughly modeled through an angular uncertainty of approximately 5° .

A range reading r_i indicates that an obstacle is located somewhere along the arc of circumference of width $w = 5°$ and radius r_i centered at the local origin of scan lines. Points located in the proximity of this arc are likely to be occupied, while there is evidence that points well inside the circular sector of radius r_i are empty.

To model this uncertain information, we introduce the two functions

$$\begin{aligned}
f_{E_i}(\rho; r_i) &= \begin{cases} k_E & 0 \le \rho < r_i - \Delta r \\ k_E \cdot \frac{r_i - \rho}{\Delta r} & r_i - \Delta r \le \rho < r_i \\ 0 & \rho \ge r_i \end{cases} \\
f_{O_i}(\rho; r_i) &= \begin{cases} 0 & 0 \le \rho < r_i - \Delta r \\ k_O \cdot \left(1 - \frac{|r_i - \rho|}{\Delta r}\right) & r_i - \Delta r \le \rho < r_i + \Delta r \\ 0 & \rho \ge r_i + \Delta r \end{cases}
\end{aligned}$$

that describe, respectively, how the degree of belief of the assertions "empty" and "occupied" vary with the distance ρ from the sensor, for a range reading r_i. Here, k_E and and k_O are two positive constants corresponding to the maximum values attained by the functions, $2 \cdot \Delta r$ is the stretch of the area considered "proximal" to the arc of radius r_i. The profile of f_{E_i} and f_{O_i} is displayed in Fig. 5.2.

The degree of belief of each assertion is assumed to be higher for points close to the scan line axis because the measure r_i has been read on that

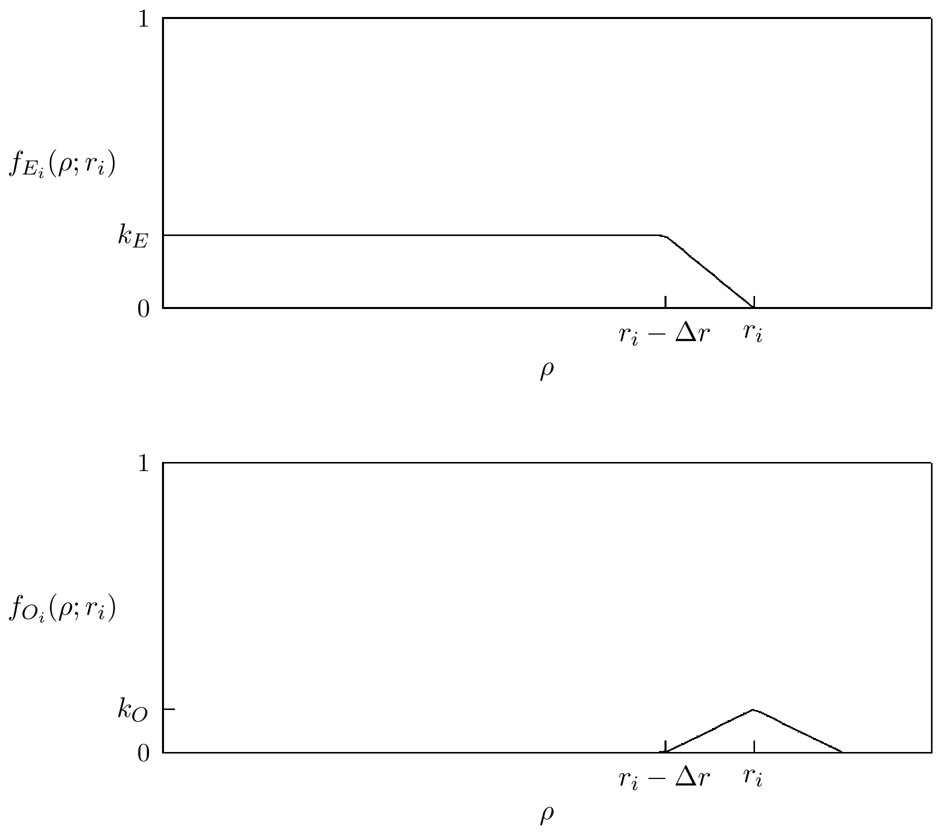


Figure 5.2: The two uncertainty functions f_{E_i} and f_{O_i} for a range reading r_i.

axis in the case of *visual sonar* method. Whereas, in the case of the second method, this results to be an empirical choice. As a consequence, points at the borders of the "5°-wide radiation cone", are expected to have a lower degree of belief. This is realized by defining a local *modulation* function

$$m(\theta) = \begin{cases} \frac{2.5^\circ - |\theta|}{2.5^\circ} & |\theta| \leq 2.5^\circ \\ 0 & |\theta| > 2.5^\circ \end{cases}$$

where $\theta = 0$ identifies the scan line related to r_i (see Fig. 5.3).

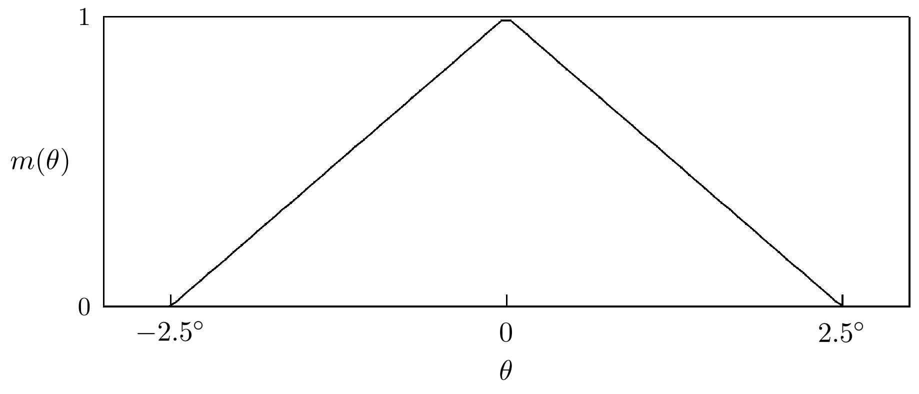


Figure 5.3: The modulation function.

Finally, we wish to limit the influence of the range reading r_i to an area close to scan line origin location. In particular, by defining the *visibility* function

$$v(\rho) = \begin{cases} 1 & \rho \leq \rho_v \\ 0 & \rho > \rho_v \end{cases}$$

the degree of belief of the assertions "empty" and "occupied" is nonzero only inside a circular sector of radius ρ_v centered at the scan line origin. We usually consider $\rho_v = 1.5m$ because observed distances longer than 1.5-2 m are often quite imprecise. This is due to the increasing loss of radial resolution of pixels along a scan line, i.e., the further (from the image projection of the scan line origin) are any two adjacent pixels belonging to the same scan line, the bigger is the distance between the two ground points corresponding to the previous pixels. This is an effect of the perspective transformation involved in the calibration matrix (see 4.2).

We can now define the two fuzzy sets E_i and O_i through their membership functions

$$\mu_{E_i}(\rho, \theta) = f_{E_i}(\rho; r_i) m(\theta) v(\rho)$$

$$\mu_{O_i}(\rho, \theta) = f_{O_i}(\rho; r_i) m(\theta) v(\rho)$$

i.e., by *and*-ing the previously introduced certainty functions. These sets represent, respectively, how the degrees of belief of the assertions "empty" and "occupied" vary inside the radiation cone. Note that the above membership functions are expressed in local polar coordinates with respect to the scan line origin position, and assume nonzero values only inside the subset of the "radiation cone" within the visibility radius.

Fusion

The task of the fusion phase is to integrate the local information contained in E_i and O_i into the global fuzzy sets E and O of empty and occupied points.

This can be performed using a fuzzy *aggregation* operator in order to update respectively E and O with E_i and O_i:

$$E := E \cup E_i$$

$$O := O \cup O_i.$$

[25] implements "$\cup$" through the Lukasievic T-conorm

$$\mu_{A \cup B}(x) = min(1, \mu_A(x) + \mu_B(x)). \tag{5.1}$$

Regarding the implementation efficiency, it should be noted that since $\mu_{O_i}(C)$ and $\mu_{E_i}(C)$ (where C is an occupancy grid cell) are nonzero only inside a circle of radius ρ_v centered at the current scan line origin, it is necessary to update E and O only in the same area.

5.2 Path Planning

Path planning is concerned with the problem of moving the robot from an initial position to a goal position, possibly avoiding obstacles and considering the minimum length path. Path planning algorithms can be classified as either global or local. Ours is global because it takes in account all the information in the environment when finding a route. In fact, it operates on the whole occupancy grid by first building a fuzzy occupancy map M_p suitable for planning purposes and finally applying the A* algorithm on M_p.

5.2.1 Building the planning map

A fuzzy logic framework presents the advantage of allowing the detection of conflicting or insufficient information. In fact, since E and O are not complementary (see 5.1.1), their intersection is the fuzzy set, A, of *ambiguous* cells, with the corresponding membership value representing the degree of contradiction. Here is the definition of A:

$$A = E \cap O.$$

Similarly, the fuzzy set of *indeterminate* cells can be defined as

$$I = \bar{E} \cap \bar{O}.$$

A conservative map S_p of the *safe-for-planning* cells is obtained by "subtracting" the *occupied*, the *ambiguous* cells from the *empty* ones and "adding" to this intermediate result the *indeterminate* cells

$$S_p = E \cap \bar{O} \cap \bar{A} \cup I \tag{5.2}$$

and the *planning map* M_p, i.e., the map that will be considered by the planning algorithm, is

$$M_p = \bar{S}_p.$$

The *indeterminate* cells are regarded as safe because, in order to reach a given goal, the robot will have to traverse regions that are indeterminate at the beginning of the motion (the environment is a priori unknown). Thus, the planner must be allowed to propose a path going through such regions.

For the above computations we use the following operator:

- $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$;
- $\mu_{A \cap B} = max(0, \mu_A(x) + \mu_B(x) - 1)$, that is the Lukasievic T-norm;
- for the "$\cup$" we employ the Lukasievic T-conorm, see equation 5.1.

In our implementation the sets of cells so far described (indeterminate, ambiguous, etc.) do not correspond to actual data structures that are stored during the occupancy grid building process. In fact, M_p is directly computed from the fuzzy sets E and O by developing the various terms in 5.2.

5.2.2 Planning algorithm

During the planning phase, a path P is produced from the current robot position (corresponding to the starting cell S) to the goal G by applying a graph search algorithm aimed at minimizing the risk along the path and its length. Note that a path P, in our case, is defined as a sequence of adjacent cells $S, ..., G$ and we use the notion of 4-connectivity to define the concept of *adjacency.* The fuzzy map M_p is used in this step.

A natural planning strategy is to avoid areas of M_p where the risk of collision is high, that are identified by cells with large values of μ_{M_p}. This may be achieved by defining a proper *cost function* for a path P, and then searching for minimum-cost paths. We first consider the case of a point robot; this assumption will be removed later.

We use an intuitive cost function, defined as

$$g(P) = \sum_{C_i \in P} \mu_{M_p}(C_i),$$

that is a measure of the integral risk along the path.

As a planning method, we have adopted the *A** algorithm, which allows to incorporate heuristic information when available, resulting in an efficient search. We shall not recall here the details of the algorithm, that are well known (Cf. [22]).

To apply *A**, we need as a basic tool a heuristic function $h(C)$ estimating the cost of the optimal path from the generic cell C to the goal G. *A** will be complete under the *admissibility* condition

$$0 \leq h(C) \leq h^*(C), \quad \forall C$$

where $h^*(C)$ is the *actual* cost of the minimum-cost path from C to G. Moreover, the heuristic function $h(\cdot)$ is said to be *locally consistent* if, for any pair of adjacent cells (C_i, C_j), we have

$$0 \leq h(C_i) \leq h(C_j) + w(C_i, C_j)$$

being $w(C_i, C_j)$ the cost of the arc between C_i and C_j. Under this assumption, whenever C_i is expanded during the algorithm visit, the current path from S to C_i is already optimal. The choice $h(\cdot) \equiv 0$ is trivially admissible and locally consistent, resulting however in a *non-informed* algorithm.

The use of *A** to generate paths minimizing g on the fuzzy map M_p is immediate. The cost of the arc joining two adjacent cells C_i and C_j is defined as

$$w(C_i, C_j) = \mu_{M_p}(C_j)$$

so that the cost of a path P coincides with $g(P)$, except for the additive constant $\mu_{M_p}(S)$. As for the heuristic function, we use

$$h(C_j) = d(C_j) \cdot \mu_{M_p}^{min} \tag{5.3}$$

in which $d(C_j)$ is the minimum number of cells that compose a subpath from C_j to G, and $\mu_{M_p}^{min}$ is the smallest value of μ_{M_p} over M_p. The heuristic function 5.3 is clearly admissible and locally consistent. There are two remarks:

- in our 4-connectivity map, $d(C_j) = |x_G - x_j| + |y_G - y_j|$, that is the so called Manhattan distance;

- to obtain an informed *A**, it must be $\mu_{M_p}^{min} > 0$. Hence, we initialize all values of μ_{M_p} with a small positive constant.

In order to obtain a safer path, we have chosen to perform an $\alpha - cut$ of μ_{M_p}. That is, only cells belonging to the (crisp) subset

$$M_p^\alpha = \{C \in U : \mu_{M_p} \leq \alpha\}$$

are considered admissible for planning. By choosing an appropriate value for α, we can obtain a reasonable trade-off between the integral and the maximum risk.

Last remark concerns removing the point robot assumption so far adopted. Assume that the robot can be approximated by a circle of radius γ (in our case $\gamma = 298/2$ since 298 mm is the biggest robot dimension, i.e., the length of the dog, Cf. [30]), whose center is located at (the center of) cell C. Since each occupancy grid has side δ ($\delta < \gamma$, we usually consider $\delta = 100$ mm), the robot body will be contained in a square of $\eta \times \eta$ centered at C, being $\eta = 2 \cdot round(\gamma/\delta) + 1$, with $round(x)$ the nearest integer to x (for instance, $\delta = 100$ mm implies $\eta = 3$ cells). Hence, we can build an *augmented* map M_p^a by defining $\mu_{M_p^a}$ as the maximum value of μ_{M_p} attained in the square of $\eta \times \eta$ cells centered at C. Planning for a point in M_p^a is equivalent to planning for the actual robot in M_p. To cope with this assumption removal, it is sufficient to modify $A*$ so as to compute $\mu_{M_p^a}(C)$ only when C is actually visited.

5.3 Path Following

Path following deals with the actual execution of the trajectory given by the path planning module. This results in managing the physical velocity commands to be sent to the robot, odometric information received from the robot and keeping track of robot position in the map. We assume that the environment does not change during the execution of a path, i.e., the obstacles are static, so we do not handle map variations once a route has been established.

5.3.1 Way-points technique

Path planning produces a 4-connected trajectory often consisting of several small δ-size cells (we often set $\delta = 100$ mm). Selecting some significant cells from this trajectory, can decompose and thus ease the path following problem, in fact it is reduced to a series of *go-to-position* problem where the *position* is one of the extracted cells. We chose to consider only the cells where a 90° trajectory change takes place (see Fig. 5.4). In this way we avoid to apply the *go-to-position* procedure to intermediate points of a segment belonging to the trajectory, in fact it would be useless since *go-to-position* stops temporary the robot when the current *position* is reached.

5.3.2 Feedback loop

The above mentioned *go-to-position* problem can be solved with a feedback control loop: see Fig. 5.5 and the following paragraphs for a detailed explanation of the diagram blocks.

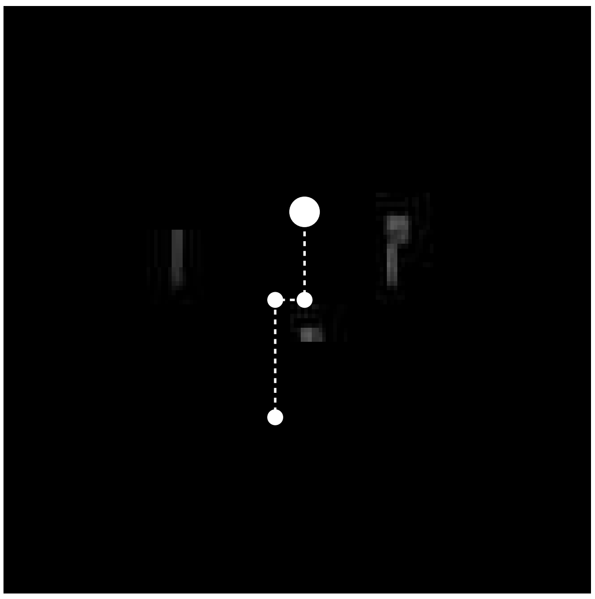


Figure 5.4: Fuzzy occupancy map showing a planned path (dotted line) starting from an initial position (big circle), and the related way-points (small circles). The latter ones corresponds to the cells where a 90° trajectory change takes place.

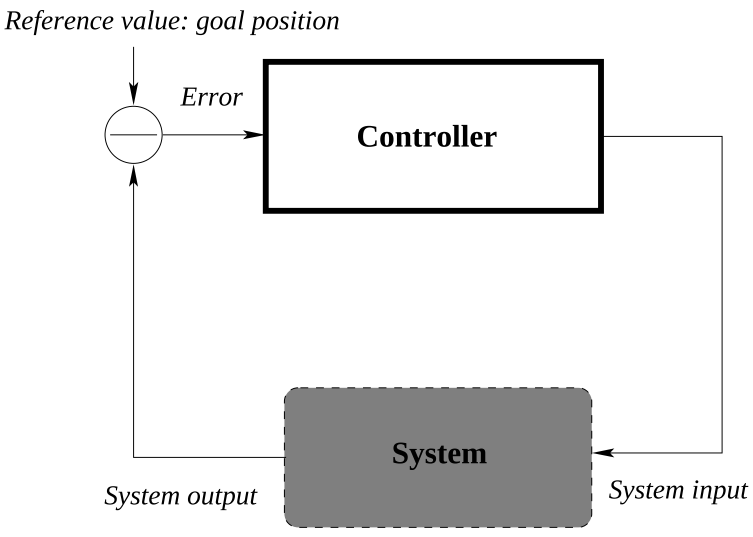


Figure 5.5: Feedback control loop

System input

We use only two of the three available velocity components (See 3.2.2) as *system input*:

- rotational velocity
- linear velocity

In fact, we do not need lateral velocity to carry out the *go-to-position* procedure (See 5.3.2). Moreover these two components are sent separately to the dog so that a motion command can exclusively be a linear movement along the current dog heading or a pure rotation. This is due to the nature of the controller algorithm (5.3.2) and to the fact that the Locomotion Unit (3.2.2) makes the dog walk more imprecisely when the speed components are mixed together in a single command.

System output

This is the current robot position and heading and their are obtained by integrating the velocities that are continuously provided by the robot (See 3.2.2). Integration starts from an initial position and heading that are contained in Estimated Robot Position (Fig. 3.2), i.e., the position and heading computed when Visual Self Localization has been invoked last time. Velocities received from the dog are not very accurate because they are simply obtained multiplying the previous velocity commands by a kind of constant slippery factor, see Sec. 3.2.2

Controller

The controller embodies the *go-to-position* routine, where *position* is a 2D point of the free ground. Thus, this is a two-dimensional problem but it can easily be divided in two one-dimensional problems:

- first the robot heading is corrected modifying rotational velocity in order to make the dog head the goal position;
- then the distance between the current position and goal is corrected through the linear velocity.

These sub-problems are achieved with two proportional controllers and the overall strategy is the following (See also fig.5.6):

```
global variables: x, y, theta
parameters       : ROT_GAIN, LIN_GAIN, DELTA_TH, DELTA_POS

procedure go-to-position(xg,yg)
```

```
repeat
   compute current robot position(x,y,theta)
   compute distances to goal(dx,dy)
   convert (dx,dy) to errors (theta_err, pos_err)
   if |theta_err > DELTA_TH|
      /* not heading the target: compensate theta */
      v_rot = ROT_GAIN*theta_err
   else
      /* heading to target: compensate position */
      v_lin = LIN_GAIN*pos_err
   send velocities (v_rot, v_lin) to robot
   pause some time
until |pos_err < DELTA_POS|
send velocities (0,0) to robot
```

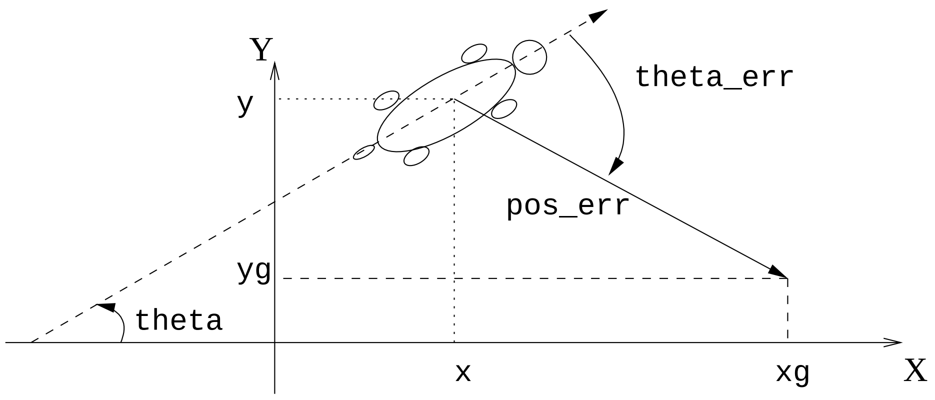


Figure 5.6: *go-to-position* problem

`compute current robot position` integrates the last velocities received by the robot over a period that is the time elapsed since `compute current robot position` has been called last time. This results in a cumulative position error (as previously said, velocity information considered in the "system output" block are imprecise).

Proportional control The heart of proportional control are the two gains: `ROT_GAIN` and `LIN_GAIN`. They affect the behaviour of the whole control system, therefore their tuning is crucial:

- if `LIN_GAIN` is too high the robot will not reach directly the goal but it will go back and forth around the goal;
- if `ROT_GAIN` is too high the robot heading will oscillate around the goal heading.

On the other hand, if the gains are too low the control system will be slow in following the reference value. We found that a reasonable trade-off can be obtained with:

- `LIN_GAIN` $= 0.2$;
- `ROT_GAIN` $= 2$.

Output saturation The control we implement is not purely proportional because a output value saturation is also applied before sending the velocities to the dog. The main reasons for this further step are:

1. linear and rotational velocities obey a constraint concerning their maximum values after those the robot cannot accomplish the requested speeds anymore because of mechanical limitations;

2. linear velocity should be higher than a certain threshold because otherwise the robot could not execute any movements due to the friction with the ground. This would make the robot more imprecise in reaching a certain point, e.g. the odometry could state that the dog is moving even by a small speed whereas it is stuck on the ground because of the friction. This minimum value of course depends on the particular ground features.

Chapter 6

Self-localization

In this chapter, we describe the *visual self-localization* process that allows to infer the current *robot position and orientation* in the environment, starting from different mosaic images taken by the robot.

As already said in the previous chapters, the robot itself provides continuously a measure of its current linear and rotational velocity. If these measures were enough accurate we would estimate the current robot position and orientation only through a simple mathematical integration, but, the fact that they are very imprecise (see Subsec. 3.2.2), leads to the need of a further procedure for better determining the robot location. Since the principal robot sensor employed in this work is the visual one, we decide to exploit the mosaic images captured from the environment to achieve this task.

Our visual self-localization is a differential procedure, that is, it provides the current robot location by computing the *change of robot position and orientation* since the visual self-localization has been performed last time. For this reason, it needs the current mosaic picture and the second-last one that has been taken. This change is carried out on the base of the correspondence among *image features* extracted in the *old* and *recent* mosaic images, which have been taken, respectively, at time t_o and t_r, with $t_o < t_r$. Fig. 6.1 shows a diagram describing this procedure.

Actually, our visual self-localization process cannot cope with the location estimation problem without the help of the velocity information received from the robot, even if these are quite inaccurate. In fact, our visual self-localization employs the mosaic image as basic input for its computations, and this limits the rate at which the visual self-localization can be executed, because it takes a certain time for the robot to shoot the six pictures composing the mosaic and send them to the host (approximately, 10 seconds). For this reason, our visual self-localization can be only a time-discrete procedure and we decide to perform it at each way-point of the planned path (see Subsec. 5.3.1). However, in order to follow a given planned path (see

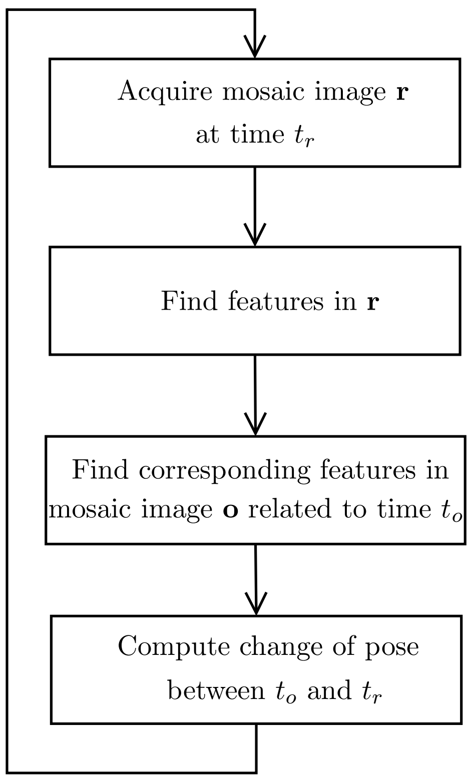

Acquire mosaic image **r** at time t_r
Find features in **r**
Find corresponding features in mosaic image **o** related to time t_o
Compute change of pose between t_o and t_r

Figure 6.1: Diagram describing the main steps of our visual self-localization procedure.

Sec. 5.3), there must be a kind of feedback concerning the current robot locomotion, even between any two consequent "calls" of visual self-localization. To achieve this, we employ the velocity information received from the robot for estimating its location between any two adjacent way-points (see Subsec. 5.3.2).

In section 6.1, we describe the computation of the difference of robot position and orientation, given two pairs of mosaic image features. In the last sections, we discuss two solutions for selecting and finding corresponding features in the two mosaics: one method is manual (Sec. 6.2), and the other one is automatic (Sec. 6.3).

6.1 Relative pose estimation

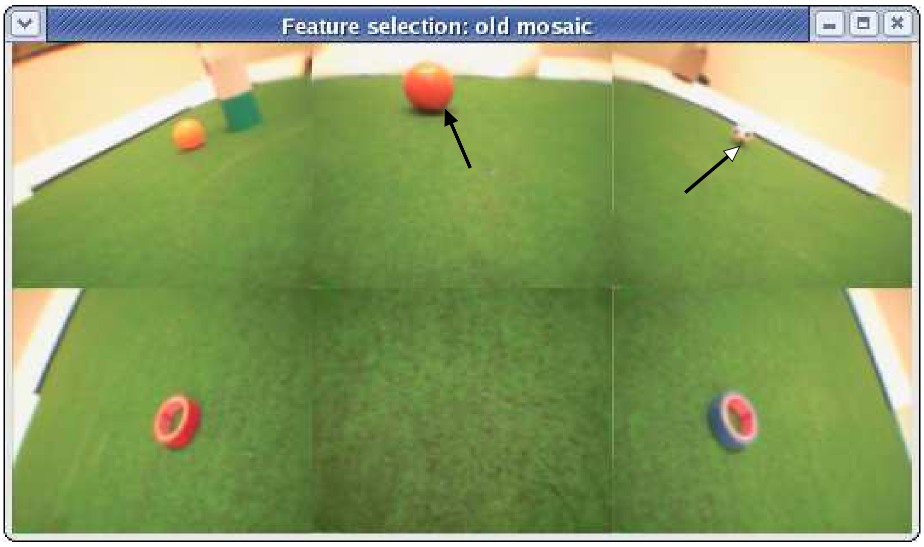
In this section, we explain how it is possible to estimate the change in position (also called displacement) and orientation of the robot, starting from *two* pairs of mosaic image pixels. The first pair represents two features belonging to *objects lying on the ground* and projected into a mosaic image taken at time $t = t_o$. Whereas, the second pair represents the same features, i.e., the same ground objects, projected in another more recent mosaic image, shot at $t = t_r$, with $t_o < t_r$.

We require that these features correspond to non-elevated points of objects lying on the ground, e.g., corners of a paper lying on the ground. We refer to those points as *landmarks* (see Sec. 2.4). This assumption allows us to determine the locations of the landmarks w.r.t. the egocentric frame related either to the mosaic image at $t = t_o$ or $t = t_r$ (see Subsec. 4.2.4), and, in turn, to apply *triangulation* techniques (see Sec. 2.4) for estimating the change of the robot pose. Fig. 6.2 gives an example of two pairs of corresponding features in the old and recent mosaic.

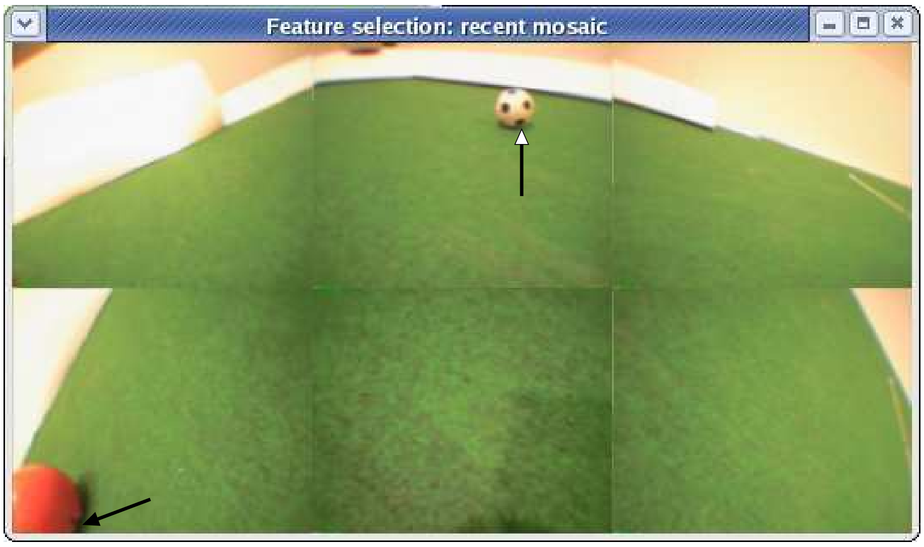
Let $(u_1, v_1)_o^i$ and $(u_2, v_2)_o^j$ be the coordinates of the pixels representing the two landmarks L_1 and L_2. These pixels belong, respectively, to tile i and j ($i, j \in [1..6]$, see Fig. 3.4) of the old mosaic o. Similarly, we have $(u_1, v_1)_r^m$ and $(u_2, v_2)_r^n$ for the same landmarks in tiles m and n of the recent mosaic r.

6.1.1 Displacement computation

Starting from $(u_1, v_1)_r^m$ and $(u_2, v_2)_r^n$, Eq. 4.24 allows to find the distances d_1 and d_2 of the two landmarks on the ground from the current location of the robot, i.e., from the origin of the egocentric frame $\{b_r\}$ related to mosaic r (for the egocentric frame we use the same conventions as in Sec. 4.2). In fact, m and n identify two particular combinations of *pan* and *tilt* angles (see Subsec. 3.2.3) that determine the two calibration matrices K_m and K_n to be used in Eq. 4.24 for getting d_1 and d_2, respectively. Using a notation



(a) A pair of features related to points be loging to objects on the ground. The arrow with the filled head refers to the first feature, which belongs to the big ball on the ground. Whereas, the arrow with the empty head refers to the second feature which is associated with the soccer ball.



(b) The corresponding pair of features in the recent mosaic. Note that, in the meanwhile, the robot has approached the soccer ball.

Figure 6.2: *Example of corresponding features.* (a) is the old mosaic image, whereas (b) is the recent one.

similar to the one adopted in Subsec. 4.2.4, we have

$$d_1 = \|\mathcal{F}^{-1}_{K_m}(u_1, v_1)\|$$

$$d_2 = \|\mathcal{F}^{-1}_{K_n}(u_1, v_1)\|.$$

With the same procedure, we can obtain $L_1^{\{b_o\}}$ and $L_2^{\{b_o\}}$, that are, respectively, the positions on the ground of landmarks L_1 and L_2, w.r.t. the old location of the robot, i.e., w.r.t. the egocentric frame $\{b_o\}$ related to mosaic o. More precisely, starting from $(u_1, v_1)^i_o$ and $(u_2, v_2)^j_o$, we have:

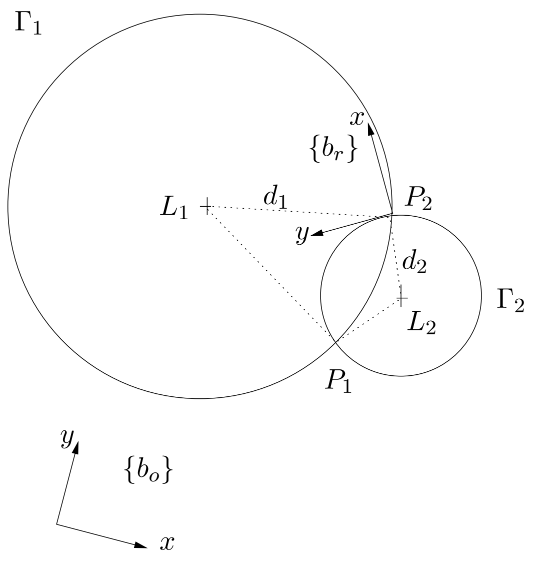
$$L_1^{\{b_o\}} = \mathcal{F}^{-1}_{K_i}(u_1, v_1)$$

$$L_2^{\{b_o\}} = \mathcal{F}^{-1}_{K_j}(u_2, v_2).$$

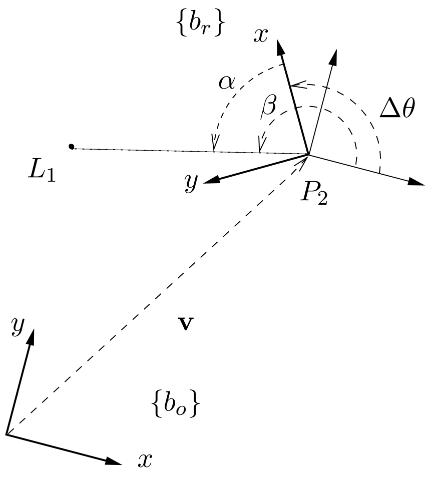
Now we can individuate two circles, Γ_1 and Γ_2 (see Fig. 6.3(a)), which are centered, respectively, in $L_1^{\{b_o\}}$ and $L_2^{\{b_o\}}$, and with radius d_1 and d_2. The intersection points, $P_1^{\{b_o\}}$ and $P_2^{\{b_o\}}$, between these circles represent the possible estimations of the robot position at time t_r, w.r.t. $\{b_o\}$. In fact, provided that the robot has observed, at time t_r, a distance d_1 to landmark L_1 and d_2 to landmark L_2, it must be on points belonging to both the above circles, i.e., on one of the two intersections. More formally, the two intersections represent the possible origins of $\{b_r\}$ w.r.t. $\{b_o\}$.

Once we have found the two possible relative robot locations, we must discriminate between them. Look at Fig. 6.3(a): provided that both landmarks L_1 and L_2 are visible in mosaic r (note that "L_1" and "L_2" are simple labels that we "attach", respectively, to the first and second feature provided in input to the *relative pose estimation* procedure) and that the mosaic has an horizontal field of view of approximately 180° (see Subsec. 3.2.3), if the robot were in P_1 it would "see" L_1 on the *left* w.r.t. L_2 (note that the robot heading coincides with the y-axis); viceversa, if it were in P_2 it would "see" L_1 on the *right* w.r.t. L_2. Thus, if we encode somehow the *angular information* "from P, L_1 is seen either on the left or on the right w.r.t. L_2", with $P \in \{P_1, P_2\}$, we can then discriminate between P_1 and P_2, since we can observe this information from the *real* location of the robot. The previous *angular information* is encoded in the following way: starting from $L_1^{\{b_o\}}$, $L_2^{\{b_o\}}$, and $P^{\{b_o\}}$ we compute the angle $\widehat{L_1PL_2}$ which is measured clockwise from L_1 to L_2 and then normalized in $[-180°, 180°]$. If $\widehat{L_1PL_2}$ is positive then L_1 is seen, from P, on the left w.r.t. L_2 (this case happens in Fig. 6.3(a) when $P = P_1$), whereas if $\widehat{L_1PL_2}$ is negative then L_1 is seen, from P, on the right w.r.t. L_2 (this case happens in Fig. 6.3(a) when $P = P_2$). Moreover, it can be easily demonstrated, just looking at Fig. 6.3(a), that the following relation is always true:

$$\widehat{L_1P_1L_2} = -\widehat{L_1P_2L_2}.$$



(a) *Triangulation problem framework.* Landmarks L_1 and L_2 are observed from the egocentric frames $\{b_o\}$ and $\{b_r\}$. These observations allow to build Γ_1 and Γ_2, and to estimate the current robot position in frame $\{b_o\}$, i.e., the origin of $\{b_r\}$, that could be either P_1 or P_2. The last two points are discriminated using the angle $\widehat{L_1L_2}$ observed in $\{b_r\}$.



(b) *Rotation computation.* $\mathbf{v}$ is the displacement vector (in this case P_2 is the robot position), whereas the change in rotation, $\Delta\theta$, between $\{b_r\}$ and $\{b_o\}$, is computed as the difference between β and α. Note that $\{b_r\}$ and $\{b_o\}$ are drawn with thicker lines.

Figure 6.3: *Relative pose estimation.* (a) shows the framework of an instance of our triangulation problem, whereas (b) focuses on the computation of robot rotation.

In order to discriminate, we finally measure the real observed angle between L$_1$ and L$_2$, $\widehat{L_1 L_2}$, as the $[-180°, 180°]$ normalization of

$$\phi\left(L_1^{\{b_r\}}\right) - \phi\left(L_2^{\{b_r\}}\right),$$

where $\phi\left(Q^{\{f\}}\right)$ indicates the angle related to vector Q expressed in frame $\{f\}$, measured in the usual counter-clockwise convention. From what has been explained so far, the *real* robot location is P_1 if

$$sgn\left(\widehat{L_1 L_2}\right) = sgn\left(\widehat{L_1 P_1 L_2}\right),$$

otherwise it is P_2. Note that $sgn\left(\cdot\right)$ is the function sign.

Finally, there is an important case to consider: when the two circles Γ_1 and Γ_2 do not intersect. This could happen when $P_1^{\{b_o\}}$, $P_2^{\{b_o\}}$, d_1, and d_2 are estimated in such a wrong way that leads to one of the following situations characterized by $\Gamma_1 \cap \Gamma_2 = \emptyset$:

- Γ_1 contains Γ_2 or viceversa;
- Γ_1 is outside Γ_2.

These errors are due to the impossibility of selecting and finding exactly the same corresponding features in both the mosaics, and inaccuracies in the calibration matrix. Anyway, when $\Gamma_1 \cap \Gamma_2 = \emptyset$ we cannot apply the above triangulation technique and we decide to estimate the robot displacement and rotation by simply integrating the current velocities provided by the robot.

6.1.2 Rotation computation

Once the robot displacement, $\mathbf{v}$, between t_o and t_r, has been found, we still have to find out the *change in rotation* $\Delta\theta$, i.e., the rotation that allows $\{b_o\}$ to have the same orientation as $\{b_r\}$. This can be easily computed for instance in the following way (refer to Fig. 6.3(b)):

$$\Delta\theta = \beta - \alpha$$

where $\beta = \phi\left((L_1 - \mathbf{v})^{\{b_o\}}\right)$ and $\alpha = \phi\left(L_1^{\{b_r\}}\right)$. Note that $\mathbf{v} = P_2$ in the figure.

6.2 Operator-based feature selection and matching

In order to select corresponding features to be "sent" as input to the *relative pose estimation* procedure (see Sec. 6.1), we implemented, as first approach to the problem, a trivial solution that requires the operator to select (via mouse) two pairs of corresponding features in the old and recent mosaic images, which are displayed at each way-point. Fig. 6.2 is actually a screedump of our system that show the windows presented to the operator (the arrows has been added for highlighting a possible choice of corresponding features).

6.3 Artificial-landmark-based automatic solution

In this section we present an automatic approach for selecting and finding corresponding features that will be used by the procedure explained in Sec. 6.1 for estimating the robot egomotion.

This approach is based on the following assumptions:

- in the environment there are N a-priori known convex objects, i.e., N artificial landmarks, but their locations is unknown;
- each of these landmarks is univocally determined by a different known color, which is coded through four thresholds (see Subsec.4.1.1);
- landmarks lie on the ground and they are flat. Moreover, they have such a size that allows to state that they will be made of at least k pixels, when projected in a mosaic image.

The method is divided in two parts: in the first one (Subsec. 6.3.1), we find the artificial landmarks present in mosaic o and r; in the second one (Subsec. 6.3.2), we look for corresponding landmarks found in mosaic o and r.

6.3.1 Landmark selection

In this subsection, we describe the procedure that allows to find the artificial landmarks in a mosaic image. This procedure must be applied once for mosaic o, and another time for mosaic r. The output of this procedure is a list l containing, for each of the N landmark color classes, the related "candidate" features eventually found in the mosaic. We speak about "candidates" because there can be more than one feature associated with a given landmark color class. This can be due, for instance, to false positives present in the environment. Each "candidate" is specified through the coordinates of a representative pixel (a sort of *centroid* which always corresponds to a landmark ground point, since landmarks are flat and convex), and the ID of the tile it has been found in (see Fig. 3.4 for the numeration of tile IDs).

We look for "candidates" in every tile of a given mosaic in the following way. The tile picture is *sequentially* scanned (bottom→up / left→right) for finding a pixel belonging to one of N color classes. Whenever a pixel is analyzed in the whole landmark selection stage, it is marked as "visited" so that it will not be examined anymore. If the current pixel belongs to one of N color classes, for instance c, a *region growing* starting from this pixel is performed. This operation aims to find an 8-connected region of pixels of the same color class c. In this way, we try to isolate a whole landmark. At the end of the region growing, we obtain a pixel "blob" characterized by the color class c, the number of pixels m it is made of, and the coordinates of its "centroid", which are obtained by averaging the coordinates of all the composing pixels. If $m \geq k$, then the current blob is considered as a candidate for the landmark c, and it is added to the occurrences of color c in list l.

Once the current region growing finishes, the sequential "scanning" starts again from the pixel which is next to the previous reached pixel, and it stops at the first non-visited pixel belonging to one the N color classes. At this point another region growing is executed.

The whole procedure finishes when all the pixels of all the tiles has been analyzed.

6.3.2 Landmark matching

In this subsection, we discuss how to find the matching between two candidates of the same landmark c that have been found, respectively, in mosaic o and r. Actually, we need two matchings, since the *relative pose estimation* needs information about two different landmarks. In the following, we explain the procedure that allows to find these two matchings, given two lists of candidates, l_o and l_r (see previous subsection).

The best situation happens when there are at least two landmarks that are both present in the two lists with only one candidate, i.e., there exist two *univocal* matchings. In that case, we take the pixel coordinates of two corresponding centroids, along with their tile IDs, and we use these information as parameters for *relative pose estimation*.

Unfortunately, the univocal matching is not the only type of matching that occurs. In fact, due to false positives, it could happen to have two or more candidates of a certain landmark in l_o and one or more candidates of the same landmark in l_r, or viceversa. We refer to this case as *multiple* matching.

When there is *at most one matching between the lists*, i.e., there is at most one landmark common to both the mosaics, we cannot infer the robot relative pose through triangulation, thus the *relative pose estimation* is not invoked. In this case, we decide to estimate the ego-motion using the *current velocities provided by the robot*, even if these constitute inaccurate odometric information (see Subsec. 3.2.2).

Instead, when there are at least two matchings and the univocal ones are at most one, we must deal with one ore more multiple matchings. In the following we explain how choosing among the possible combinations of candidate correspondences, related to a multiple matching, making use of the velocities provided by the robot.

Suppose we have, for landmark L_1 of color class c, n and m candidate centroids, ${}^{p}C^{o}_{i}$ and ${}^{q}C^{r}_{j}$ with $i \in [1..n]$, $j \in [1..m]$, respectively in list l_o and l_r. The back superscripts p and q, indicate that C^{o}_{i} and C^{r}_{j} have been found, respectively, in tile p of mosaic o, and in tile q of mosaic r. Among the possible pairs of corresponding centroids, (C^{o}_{i}, C^{r}_{j}), we choose the pair, $({}^{\overline{p}}C^{o}_{\overline{i}}, {}^{\overline{q}}C^{r}_{\overline{j}})$, associated with the closest distance between, C^{o}_{i} and C^{r}_{j}. That is, we choose the corresponding candidates whose centroids, when "anti-projected" on the ground through Eq. 4.24, have the closest distance. In fact, the distance, measured on the ground, of two corresponding landmark points should be ideally zero, but, due to the impossibility of selecting exactly the same features on both the mosaics, inaccuracies of the calibration matrix, and defective knowledge about the transform between $\{b_o\}$ and $\{b_r\}$ (see after for further details), such distance is never null. Note that the anti-transform $\mathcal{F}^{-1}$ brings C^{o}_{i} and C^{r}_{j} to vectors lying on the ground and expressed, respectively, in $\{b_o\}$ and $\{b_r\}$. In order to compute their distance, we need to express them w.r.t. the same frame and we choose $\{b_o\}$. This means that we need to estimate the change in translation and rotation between $\{b_o\}$ and $\{b_r\}$, i.e., the robot egomotion. For this purpose, we employ the odometry provided by the robot. In fact, integrating the linear and rotational velocities provided by the robot during the time interval $\Delta = t_r - t_o$, we can easily build the homogeneous matrix, A^{o}_{r}, that allows to express the anti-transform of ${}^{q}C^{r}_{j}$, $\mathcal{F}^{-1}_{K_q}(C^{r}_{j})$ (see Subsec. 6.1 for the notation), into frame $\{b_o\}$:

$$\left(\mathcal{F}^{-1}_{K_q}(C^{r}_{j})\right)^{\{b_o\}} = A^{o}_{r}\mathcal{F}^{-1}_{K_q}(C^{r}_{j}).$$

Finally, the centroids to be chosen, ${}^{\overline{p}}C^{o}_{\overline{i}}$ and ${}^{\overline{q}}C^{r}_{\overline{j}}$, are such that

$$\forall i \in [1..n]\ \forall j \in [1..m]$$

$$\| \left(\mathcal{F}^{-1}_{K_{\overline{q}}}(C^{r}_{\overline{j}})\right)^{\{b_o\}} - \mathcal{F}^{-1}_{K_{\overline{p}}}(C^{o}_{\overline{i}})\| \leq \| \left(\mathcal{F}^{-1}_{K_q}(C^{r}_{j})\right)^{\{b_o\}} - \mathcal{F}^{-1}_{K_p}(C^{o}_{i})\|.$$

Now that we have found the "best" corresponding centroids for L_1, we must repeat this procedure for L_2 only if it is characterized by a multiple matching as well. Finally, we are able to invoke the *relative pose estimation* procedure in order to try to achieve a better egomotion estimation than the one employed for determining $C^{o}_{\overline{i}}$ and $C^{r}_{\overline{j}}$.

Chapter 7

Experiments

In this chapter, after a brief explanation of the common experimental set-up, we test in a systematic way three modules of our system: *calibration matrix*, *obstacle detection* and *visual self-localization*. For each of them, we report the *experimental set-up*, the *performance indexes* used for evaluating the experiment, a statistic of the collected *results* and a brief *data analysis*.

At the end of the chapter we illustrate a run of the robot in a simple unknown environment, employing the overall framework we developed for the whole task of obstacle-free navigation.

7.1 Common experimental set-up

All the experiments described in this chapter were performed using a Sony's AIBO ERS-210A. This is a 4 legged robot, with 3 degrees of freedom for each leg and 3 degrees of freedom for the head (pan, tilt and roll). It is equipped with a CMOS sensor camera, installed in its head, with a maximum resolution of 176×144 pixels and a field of view of approximately 57.6° horizontally and 47.8° vertically. Note that these are the fields of view of each tile composing a mosaic image.

The experimental location was a room with only artificial illumination consisting in both neon and halogen lamps placed on the ceiling.

We placed the obstacles and made the robot move on a synthetic carpet which is usually employed in RoboCup Soccer. This carpet allows the robot to walk with a low slippery factor. Moreover, it has a uniform color.

7.2 Calibration matrix

The first component we want to test in this chapter is the *calibration matrix*. Its correctness is important to be evaluated, since we want to validate the procedure through which this matrix has been obtained. Moreover, the calibration matrix accuracy affects other relevant vision-based tasks like obstacle range estimation and self-localization.

In particular, with this experiment, we want to test $\mathcal{F}^{-1}$, i.e., the inverse transform, built starting from the *calibration matrix*, that maps pixels below the horizon line in the mosaic image to the corresponding ground points.

7.2.1 Experimental set-up

We placed in front of the dog a set of objects at a constant distance d from the origin of the robot framework, i.e., $\{b\}$, and spaced of α degrees, such that to cover a semicircumference. The considered values for d were 13 cm, 30 cm, 60 cm, 100 cm, 160 cm, 230 cm, and the obstacles were spaced of 5° for the first two ranges and every 10° for the others, see Fig. 7.1. Objects were then selected by clicking via mouse the related pixels in the mosaic image, and the results obtained by applying $\mathcal{F}^{-1}$ to those pixels were compared with the real object ground locations . The choice of using less spaced objects, i.e., a lower number of objects, at higher distances, is due to the fact that far objects that are too close each other result in corresponding groups of pixels that are almost indistinguishable, thus making inaccurate the mouse selection. The experiment was carried out using a set of about 150 measures.

7.2.2 Performance indexes

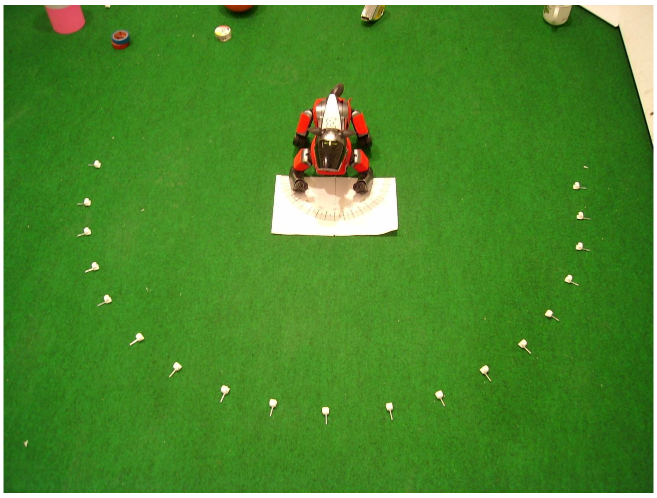
In order to evaluate the accuracy of the computed object positions, we use two performance indexes: the *distance error* and the *error on the angle*, w.r.t. the robot framework. The first is expressed as a percentage error, the second one as an absolute error in degrees.

7.2.3 Results

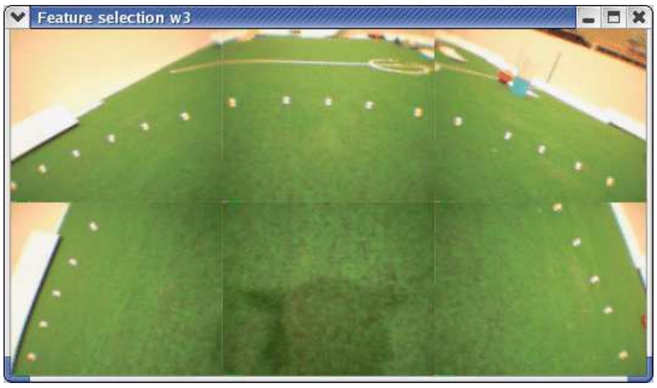
Since the *calibration matrix* is different for each tile of the mosaic, we grouped the collected data by the tile in which the pixels were selected. Note that it is not possible to evaluate all the ranges in each tile. For instance, objects 30 cm far are not visible in tile 1 and 3, and, in order to test the calibration matrix of tile 5, we introduced a specific obstacle set at 13 cm, which is visible only in that tile. See Figure 3.4 for the tile numeration.

In the following table, a statistic of the obtained results is reported, indicating for each performance index, mean and standard deviation of the signed values, *mean(st.dev.)*, and mean of the absolute values, *abs mean*. We report also the mean of the absolute values, grouped by range (see column "mean").

		tile 1	tile 2	tile 3	tile 4	tile 5	tile 6	mean
13 cm	dist mean (st.dv)					-7.18 (1.98)		
	dist abs mean					7.18		7.18
	angle mean (st.dv)					1.07 (2.79)		
	angle abs mean					2.53		2.53
30 cm	dist mean (st.dv)		7.41 (1.47)		-5.15 (2.29)		11.21 (5.00)	
	dist abs mean		7.41		5.15		11.21	7.96
	angle mean (st.dv)		-1.88 (1.61)		5.91 (2.11)		-4.44 (2.05)	
	angle abs mean		2.1		5.91		4.44	4.28
60 cm	dist mean (st.dv)	14.17 (8.29)	8.33 (2.48)	5.21 (6.81)	-15.00 (6.06)		12.22 (4.16)	
	dist abs mean	14.50	8.33	2.88	15.00		12.22	11.32
	angle mean (st.dv)	-1.80 (3.94)	1.60 (2.72)	1.13 (4.09)	1.00 (2.76)		-4.98 (3.50)	
	angle abs mean	3.40	2.00	2.88	1.67		4.98	3.06
100 cm	dist mean (st.dv)	5.60 (8.44)	-0.2 (1.30)	6.00 (2.45)	-24.75 (2.63)		7.71 (4.49)	
	dist abs mean	7.2	1.00	6.00	24.75		7.71	8.93
	angle mean (st.dv)	-0.2 (1.48)	1.40 (3.05)	3.00 (1.22)	2.75 (1.71)		1.68 (1.73)	
	angle abs mean	1.00	2.60	3.00	2.75		2.04	2.60
160 cm	dist mean (st.dv)	0.56 (13.24)	-4.15 (10.27)	16.25 (7.49)				
	dist abs mean	10.94	9.26	16.25				11.11
	angle mean (st.dv)	2.60 (1.71)	1.73 (2.69)	3.00 (1.00)				
	angle abs mean	2.6	2.45	3.00				2.74
230 cm	dist mean (st.dv)	3.84 (27.30)	1.39 (12.09)	22.70 (11.18)				
	dist abs mean	22.24	8.69	23.21				18.85
	angle mean (st.dv)	2.17 (2.49)	0.00 (1.41)	1.28 (1.15)				
	angle abs mean	2.83	1.20	1.33				2.06



(a) *Top view of the experimental set-up.* The robot is in the center of a semi-circumference with radius of 30 cm, and made of small white objects spaced every 10°. Under the dog there is a paper over which a goniometer has been printed. This is used to set the objects.



(b) *Robot view of the experimental set-up.* This is what is visible from the mosaic image shot by the robot. The white objects are purposely small in order to make their mouse selection almost univocal. Note, in the middle-low tile, the shadow of the robot head and ears.

Figure 7.1: Two views of the experimental set-up used for testing the calibration matrix.

7.2.4 Data analysis

Analyzing the collected data, we can see that, in the most of the cases, the mean value of the distance error is not null. It reveals a systematic error due, for instance, to inaccurate pan/tilt rotation of the robot camera. This imprecision could be particularly evident in tile 4 since in this case the distances are always under-estimated.

Another cause may be imputed to the measuring procedure itself, e.g., a non-exact positioning of the robot in the center of the semicircumference. In fact, in the case of 13 cm range, an error of 1 cm leads to a systematic distance error of 8%.

We can notice also that for 160 and 230 cm ranges, the standard deviation is considerably higher than for shorter ranges, owing to a more sensitive dependence on the pixel selection and to the head-rotation error.

Figure 7.2(a) shows the *abs mean* of all the samples with respect to the range. It is evident a growing trend of the error as the range increases. The error peak in 60 cm could be explained by a not enough large set of samples.

On the contrary, the angle error does not seem to have a strong dependence on the object distance. Figure 7.2(b) shows the angular *abs mean* with respect to the range. It has not an evident trend, but it is bounded under 5°. Looking instead at the signed *mean*, only tile 6 and 4 reveal high systematic errors, that could be due, again, to the imprecise camera rotation.

7.3 Obstacle detection

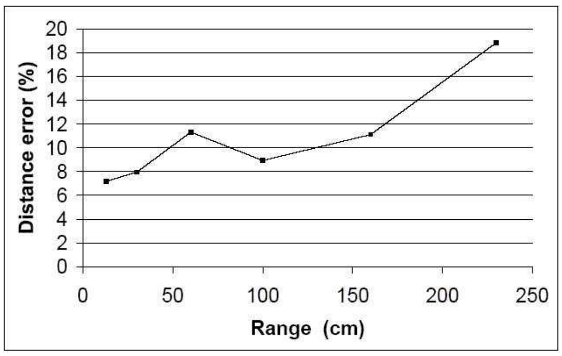
The obstacle detection module reveals the presence of obstacles placed on the ground around the robot, and estimates their distance from the robot framework. We have implemented two algorithms to realize it, see respectively Sec. 4.3 and Sec. 4.4. The former, *algorithm 1*, detects the first obstacle point along a scan line, whereas the latter, *algorithm 2*, detects the closest one within an angle. In this section we test both and we compare them by finding out the main differences in the results.

7.3.1 Experimental set-up

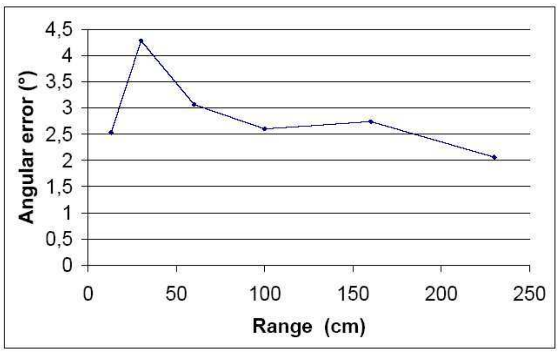
In these experiments, we employed two different environmental scenarios. Each of them was made of the robot surrounded by a set of obstacles with different shapes and dimensions, e.g., some small balls with 5-10 cm diameter and some bars 60-100 cm long, see Figure 7.3.

Concerning the tuning of the segmentation stage, we employed $RATIO = 400$, see Subsec. 4.1.5.

In order to evaluate the obstacle detection, we used different combinations of two parameters. The first one is the type of scenario illumination:



(a) Absolute percentage error of the distance as function of the range.



(b) Absolute angular error as function of the range.

Figure 7.2: Error functions of the distance and the angle of an object.

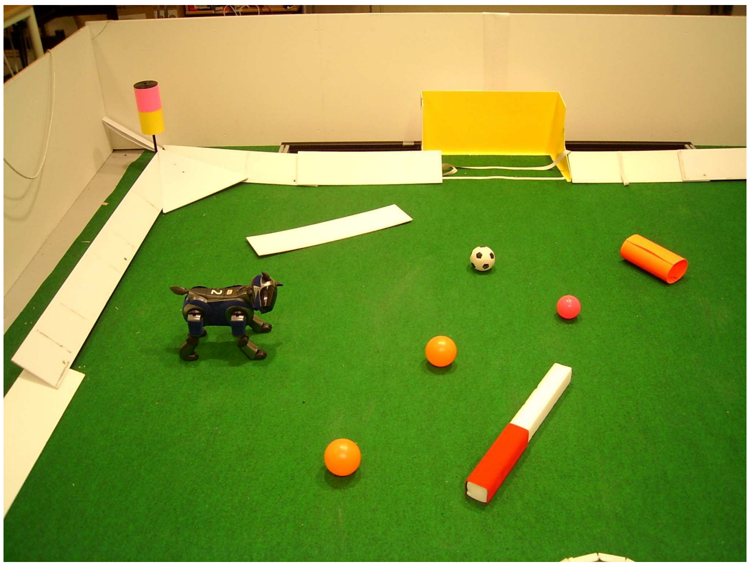


Figure 7.3: A scenario with balls and bars in front of the dog.

neon or halogen lamps. The second one determines manual or automatic segmentation (see Sec. 4.1).

7.3.2 Performance indexes

Our obstacle detection tries to find the closest obstacle either along a scan line or within an angle, depending on the considered algorithm. For each scan line or angle there exist four different cases:

- an obstacle is found and it corresponds to a real obstacle (*True Positive, TP*);
- no obstacles are found and there are not any obstacles, as well, in the real scenario (*True Negative, TN*);
- an obstacle is found, but it does not exist in the scenario (*False Positive*, FP);
- no obstacles are found, but a real obstacle exists (*False Negative*, FN).

The performance indexes used to evaluate the false detections are the number of FP and FN.

Note that in this section we do not want to test the accuracy of the detected obstacle distances because these strongly depend on the *calibration matrix*, already evaluated in Sec. 7.2.

An important remark deals with distinguishing a FP from a TP related to a high error on the distance. This is not always a trivial task and such situation occurs to coincide with an incorrect segmentation. We adopted the following criterion to discriminate between FP and TP: *if* the segmented image shows isolated groups of misclassified obstacle pixels that are closer to the projected $\{b\}$ origin than all the other obstacle pixels, and are detected by either a projected scan line or a projected angle, *then* such an obstacle is considered as FP; *otherwise*, if the segmentation enlarges or shrinks a true obstacle, this is classified as TP, although the distance error is high due to inaccurate segmentation.

Moreover, owing to the inaccuracy of the calibration matrix for high ranges, see Sec. 7.2, we do not consider obstacles with a real distance higher than 160 cm. Such a "threshold" is taken in account, within the sensor model, in the *visibility* function (see Subsec. 5.1.1).

7.3.3 Results

The table below summarizes the results obtained in this experiment. For each scenario, it is indicated the type of illumination, neon or halogen, and the type of segmentation, automatic or a manual, for both the algorithm adopted.

FP and FN express the number of scan lines or angles that detect a False Positive or a False Negative. Note that the sum of FP and FN has 36 as upper-bound, i.e., the total number of scan lines or angles analyzed for each mosaic.

		algorithm 1		algorithm 2	
Scenario	Parameters	FP	FN	FP	FN
1	Neon & manual	4	6	5	3
	Halogen & manual	4	6	5	3
	Neon & automatic	4	6	7	1
	Halogen & automatic	4	6	4	2
2	Neon & manual	1	7	1	5
	Halogen & manual	1	8	2	6
	Neon & automatic	4	3	3	4
	Halogen & automatic	4	4	3	2

7.3.4 Data Analysis

The previous table shows that the best results are reached when a manual segmentation is performed. In fact, this usually allows to reduce the FP number. The automatic segmentation, instead, seems to reduce the number of FN, but this is not a real improvement of performance, because there is a sort of transformation of FN in FP. The automatic segmentation, in fact, produces many noisy obstacle pixels at short distances from the robot. Thus, scan lines or angles related to a FN in the manual segmentation, might detect one of these noisy pixels and give arise to a FP in the automatic segmentation.

A comparison between the two algorithms shows that *algorithm 1* detects more FN and less FP than *algorithm 2*. This is explainable considering two main reasons:

- provided that *algorithm 1* samples the space only every 5 degrees, it is less sensitive to the segmentation noise described above;
- *algorithm 2* is less sensitive to the calibration matrix inaccuracy that causes the detection of FN. See Fig. 7.4 for a graphical example.

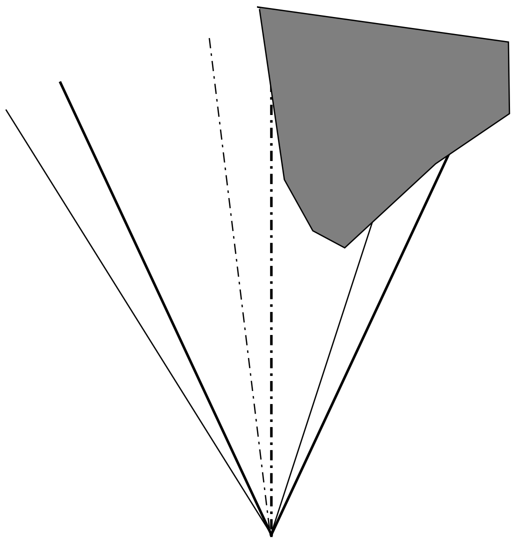


Figure 7.4: This figure illustrates why *algorithm 2* is less sensitive to detection of FN than *algorithm 1*. It is depicted a real scan line on the ground (thick dash-dotted line) and the related real angle considered by *algorithm 2* (the angle sides are the thick continuous lines). The thin dash-dotted line and the thin continuous lines represent respectively the scan line and the angle sides as they are "perceived", on the average, by the robot, because of the calibration matrix inaccuracy. Note that the figure refers to the case of a positive systematic angle error like one of those reported in the table of Subsec. 7.2.3. As you can see, an obstacle placed on the real scan line and within the related angle, might not be detected by *algorithm 1*, giving rise to a FN. Whereas, *algorithm 2* can correctly considers the obstacle as a TP.

The obtained results show a general high number of FN with respect to the FP. This is also because of the *RATIO* value used. Setting *RATIO* to a lower value, the number of FP increases, whereas the number of FN decreases. This is usually a preferred situation, because, in order to avoid obstacles, a FN is a more relevant error than a FP, i.e., a longer path generated to avoid a "fictitious" obstacle is preferred to a path that leads against an "unseen" obstacle.

7.4 Visual self-localization

In this section we want to evaluate the precision of the visual self-localization stage that we have implemented according to the technique discussed in chapter 6. During the execution of the overall navigation system, the self-localization exploits the integration of the velocity information supplied by the robot with the visual information derived from the landmark recognition. In this experiment, however, we employ only the "pure" visual component of the self-localization, in order to achieve an intrinsic evaluation of this part. This means that we consider only the cases in which the other component of self-localization, the open-loop odometry, is not needed for estimating the current robot position (see Ch. 6). Anyway, an example of self-localization with both the components will be given in Section 7.5, where we will show a run of the whole navigation task.

7.4.1 Experimental set-up

Our visual self-localization requires the presence in the environment of a set of landmarks with different known colors. We used four flat papers with sides of about 15 cm, lying on the ground. The employed colors, pink, blue, yellow and green, are uniform and easily distinguishable. This choice was made to avoid misdetection of landmarks.

The selection of corresponding landmarks is performed in two modalities: manually clicking via mouse directly on the mosaic images, or in an automatic way.

The robot was moved from the starting position to the goal manually and we verified the estimated robot position provided by the visual self-localization procedure. This experiment was repeated 30 times using always different displacements and robot headings, for covering many possible configurations.

7.4.2 Performance indexes

The performance indexes used in this section are:

- percentage error on the length of the displacement;
- absolute error on the angle of the displacement direction;
- absolute error on the angle of the robot heading.

7.4.3 Results

The following is a statistic of the experimental results. *Dist*, *Angle* and *heading* are the mean values of the previously described performance indexes.

	Dist	Angle	Heading
manual	15.44	7.53	9.99
automatic	20.30	13.67	16.37

7.4.4 Data analysis

The obtained results show that the automatic self-localization leads to a considerable worsening of the performance, in particular the heading angle reveals an increase of the error of 6.4°. This worsening is due to the automatic determination of landmark centroids. In fact, correspondences among centroid pixels are found in order to individuate the same landmark in different mosaic images, but usually such pixels do not represent exactly the same real point of the landmark since this is a quite large object.

7.5 Obstacle avoidance

In this section we want to show the functioning of the overall navigation system in an environment containing obstacles and landmarks with a-priori known color.

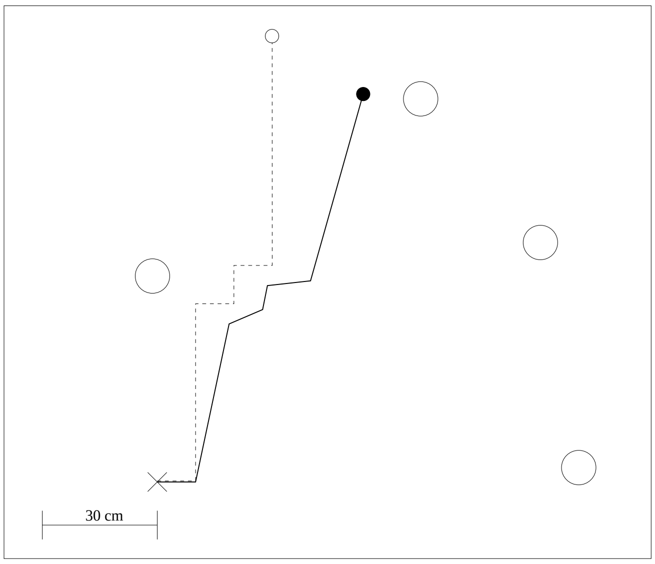
The robot, placed in a starting point, takes a mosaic picture of the environment in front of it and, using the information of obstacle distances obtained through the visual sensor (implemented, in this experiment, with the *visual sonar* algorithm), builds a map with cells of 10 cm size. Once the map is displayed in the host console, we select the goal cell that is supposed to be reached. A path connecting the start to the goal cell is planned and then executed, exploiting, at each path waypoint, the robot visual self-localization.

In this experiment we used a different set of landmarks from those described in Sec. 7.4, because, we needed more visible landmarks to be observed by the robot throughout its navigation. Therefore, we employed a set of four cylinders, see Figure 7.5(b), and the related centroids were computed such that they were always on the ground.

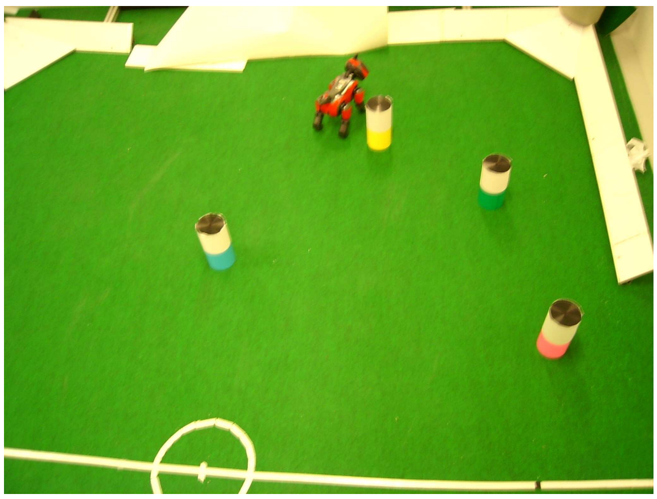
Figure 7.5(a) illustrates the run of the robot. The circles represents the four landmarks used. They are considered both as landmarks and as obstacles to be avoided.

The starting point is marked with a cross and the final one with a small circle. The planned path is drawn with a dashed line and the actual executed one with a continuous line.

The final reached position is not exactly the goal point, but it is quite close to it, about 30 cm far. This difference is due to the cumulative error of the estimate robot position that grows during the whole execution of the path. In fact, the visual self-localization computed at each waypoint, is a relative pose estimation that, unavoidably, brings the system to accumulate errors at each step. This error accumulation, sometimes, might lead the path execution to fail in avoiding obstacles, especially if this is made of a large number of way-points.



(a) The scenario is made of four landmarks, the big circles. The robot, initially at the cross point, is expected to reach the goal, the empty small circle, following the obstacle-free planned path, the dashed line. The continuous line represents the real executed path. The goal and the real final position, the filled small circle, are 30 cm distant and this is mainly due to the error of the self-localization, accumulated during the execution.



(b) A picture of the scenario, took when the AIBO reached the final position.

Figure 7.5: A run of the AIBO robot in a scenario with four landmarks, used for the visual self-localization and considered also as obstacles to be avoided.

Chapter 8

Conclusions

8.1 Summary

In this thesis we have described the problems and the related employed techniques underlying the implementation of a *semi-autonomous navigation* system in unstructured environments, using a Sony's AIBO robot.

The main original contributions of this thesis are:

1. the overall framework;
2. visual sonar algorithm extended to mosaic images;
3. a new method for obstacle range estimation using mosaic images;
4. a new method for relative pose estimation based on mosaic images.

Provided the breadth of the problem addressed in this thesis, we have coped with it sometimes exploiting existing solutions, as in the case of map building and planning [21, 25], other times extending existing ideas to our needs, as in the case of "visual sonar" [16], and sometimes designing new approaches, as in the case of the second method of obstacle detection (see Sec. 4.4) and the visual self-localization procedure (see Sec. 6.1). Apart from the previously mentioned contributions, a relevant part of this work has concerned the actual integration of the several components needed for the task of semi-autonomous navigation in the particular distributed robotic platform employed by us. These components range from those belonging to more abstract layers, like planning an obstacle-free path, those belonging to the perception level, like estimation of obstacle ranges, to those dealing with the "physical" interface between robot and environment, like execution of a path.

For what concerns the research issues, this work has focused mainly on some possible solutions to the AIBO's intrinsic weaknesses concerning lack of effective sensors for obstacle detection and range estimation, and for self-localization.

8.2 Critical evaluation

The original goal of this thesis was to develop a system for semi-autonomous navigation of an AIBO in unknown environments. Emphasis was put on the exploitation of the monocular vision that is the major mean AIBO has to acquire information from the environment, even though its camera provides images with a limited field of view. In particular, monocular vision was intended mainly to cope with detecting obstacles and with robot self-localization.

We have faced the problem related to the obstacle perception, designing two different ways for inferring obstacle distances. These methods are based on *mosaic images*, i.e., sets of images taken by the robot camera from different orientations, which supply a wider field of view than a simple image. Experimental evidence has shown that these methods provide acceptable results, at least comparable with those obtained in the research domain of the legged RoboCup, which also employs Sony's AIBOs.

Concerning the tough problem related to the self-localization of a legged robot, we have proposed a visual approach employing *mosaic images* as well, which is alternative to the only use of the defective open-loop odometry (See Subsec. 3.2.2).

Making use of the above solutions, we have developed a stand-alone system for the semi-autonomous navigation of the robot. In fact, throughout the environment exploration, the robot builds a map of the detected obstacles and this allows the operator to click on such map to provide the robot with a goal environment location to be reached. Then, the robot tries to reach autonomously that location by moving on a flat floor of unknown color and taking care of avoiding obstacles.

There are also some limitations to be overcome. In fact, considering the self-localization, the results are not completely satisfying, especially concerning the robot heading error resulting in the automatic version (see Subsec. 7.4.3). Instead, regarding the whole semi-autonomous navigation, we have obtained a system still quite far from its actual applicability in real unknown environments, since we have assumed static obstacles during the execution of a path, and the presence of artificial landmarks. Moreover, the error sensitivity of the visual self-localization method sometimes affects the effectiveness of the execution of an obstacle-free path. However, we have provided a modular framework where future enhancements can be applied.

8.3 Future work

Concerning the obstacle range detection, improvements might be achieved essentially in two directions:

- Calibration matrix. Other ways to calibrate the robot camera could be tried. In fact, we have adopted a "direct" method that builds the matrix starting from all the needed intrinsic and extrinsic parameters, but there exists other "indirect" methods to get such matrix that are based on sample camera images. These methods could model some phenomena like lens distorsions and systematic positioning error of the camera that we have not taken in account.

- Segmentation. This is a difficult operation to be performed in unknown environments since it is hard to distinguish general obstacles from the free ground, in every light condition. We have employed a thresholding technique, but other solutions could be tried like edge detection-based segmentation or clustering methods.

Regarding the visual self-localization, the first important enhancement to be done is avoiding the artificial landmarks. This could be reached, for instance, employing feature detection algorithms based on natural landmarks, e.g., those relying on edge detection. Moreover, a model of the uncertainty related to range and bearing landmark measurement could be introduced. For instance, knowledge of the *Geometric Dilution Of Precision (GDOP)* (see [7] for more details) can be used by the robot to select which landmarks are to be used if multiple landmark choices are available. However, in practical situations a robotic system may encounter difficulties in uniquely identifying landmarks, or the position estimation may be unstable owing to landmark geometry, as in the case of the flat, relatively large landmarks we have employed in Sec. 7.4. One approach to dealing with intermittently reliable landmark information could be to combine, in a tighter way than we have done in our work, landmark-based position information with position information from the open-loop odometry. This fusion of information from two sources could be accomplished using, for instance, a Kalman filter.

In order to provide a more robust navigation in presence of both static and dynamic obstacles, a behavior-based approach, exploiting the AIBO's infrared sensor, could be integrated with the deliberative architecture we have employed. In fact, the infrared sensor can detect obstacle within a range of approximately one meter, even thought it does not produce accurate range estimation.

Bibliography

[1] S. Beauchemin, and J. Barron. *The computation of optical flow.* ACM Computing Surveys, 27(3):433-467, 1995.

[2] R. A. Brooks. *A robust layered control system for a mobile robot.* IEEE J. Robot. Automat., vol. 2, no. 1, pp. 14 23, 1986.

[3] J. Bruce, T. Balch, and M. Veloso. *Fast and Cheap Color Image Segmentation for Interactive Robots.* Proceedings of IROS-2000, Japan, 2000.

[4] A. Carbone, G. Ugazio, A. Finzi, F. Pirri, M. Cialdea, M. Iarusso, and A. Orlandini. *RoboCupRescue - Robot League - Team Alcor-Dia, Italy* Awarded paper at RoboCupRescue - World Championship, Lisbon, Portugal, 2004.

[5] F. Dellaert, T. Balch, M. Kaess, R. Ravichandran, F. Alegre, M. Berhault, R. McGuire, E. Merrill, L. Moshkina, and D. Walker. *The Georgia Tech Yellow Jackets: A Marsupial Team for Urban Search and Rescue.* AAAI Mobile Robot Competition, Edmonton, Alberta, Canada, 2002.

[6] G. Dudek. *Environment mapping using multiple abstraction levels.* Proceedings IEEE, 84(11):375-397, 1996.

[7] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics.* Cambridge University Press, 2000.

[8] E. Fabrizi, G. Oriolo, and G. Ulivi. *Accurate Map Building via Fusion of Laser and Ultrasonic Range Measures.* in Fuzzy Logic Techniques for Autonomous Vehicle Navigation, A. Saffiotti, D. Driankov (Eds.), Springer, 2001.

[9] R.I. Hartley. *Estimation of relative camera positions for uncalibrated cameras.* In 2nd European Conference on Computer Vision, Santa Margherita Ligure, Italy, pages 579-587, Springer Verlag, Heidelberg, 1992.

[10] D.Hearn, M.P.Baker. *Computer Graphics with OpenGL*, third edition, Prentice Hall, 2004.

[11] D. Herrero-Perez, H. Martinez-Barbera, and A. Saffiotti. *Fuzzy Self-Localization using Natural Features in the Four-Legged League.* Proceedings of the International RoboCup Symposium, Lisbon, Portugal, 2004.

[12] K. Ito, Z. Yang, K. Saijo, K. Hirotsune, A. Gofuku, and F. Matsuno. *A rescue robot system for collecting information designed for ease of use.* To appear in IEEE International Workshop on Safety, Security, and Rescue Robotics, Kobe, Japan, 2005.

[13] E. Koyanagi, Y. Ooba, S. Yoshida, and Y. Hayashibara. *RoboCupRescue - Robot League - Team Toin Pelican, Japan* Awarded paper at RoboCupRescue - World Championship, Lisbon, Portugal, 2004.

[14] J.-C. Latombe. *Robot Motion Planning.* Kluwer, Norwell, MA, 1991.

[15] K. LeBlanc, S. Johansson, J. Malec, H. Martinez, and A. Saffiotti. *Team Chaos 2004.* RoboCup 2004, Lisbon, 2004.

[16] S.Lenser, M.Veloso. *Visual Sonar: Fast Obstacle Avoidance Using Monocular Vision.* In Proceedings of IROS'03, the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, Oct. 2003.

[17] V. Lumelsky and A. Stepanov. *Path-planning strategies for a point mobile automaton moving admist unknown obstacles of arbitrary shape.* Algorithmica, 2(4):403-440, 1987.

[18] Micron Technology, Inc.
http://www.micron.com/products/imaging/technology/lens.html

[19] S.K. Nayar, H. Murase, and S.A. Nene. *Learning, positioning, and tracking visual appearance.* In E. Straub and R.S. Sipple (eds.), Proceedings International Conference Robotics and Automation. Volume 4, pp. 3237-3244, Los Alamitos, CA, 1994.

[20] OPEN-R SDK. http://openr.aibo.com.

[21] G. Oriolo, G. Ulivi, and M. Venditelli. *Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments.* IEEE Trans. on Systems, Man, and Cybernetics, vol 28, no. 3, Part B, pp 316-333, 1998.

[22] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[23] E.Sahin and P.Gaudiano. *Visual Looming as a range sensor for mobile robots.* Fifth International Conference on Simulation of Adaptative Behavior, Zurich, Switzerland, 1998.

[24] A. Saffiotti. *Artificial Intelligence Techniques for Mobile Robots, lecture notes.* A. Saffiotti, Orebro University, Sweden, 2004.

[25] A. Saffiotti. *The GridMap Library.* A. Saffiotti, 2000.

[26] A. Saffiotti. *Platforms for Rescue Operations.* Study commissioned by NIROS, the Swedish National Center for Innovative Rescue and Safety Systems, 2004.

[27] H. Samet. *Region representation: Quadtrees from boundary codes.* Comm. ACM, 23(2):163-170, 1980.

[28] L. Sciavicco, B. Siciliano. *Robotica Industriale.* Modellistica e controllo di manipolatori (seconda edizione), McGraw-Hill, 2000.

[29] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision, Second Edition.* PWS Publishing, 1999.

[30] SONY. *OPEN-R SDK Model Information for ERS-210.* Sony Corporation, 2004.

[31] SONY. *OPEN-R SDK Programmer's Guide.* Sony Corporation, 2004.

[32] SONY. *OPEN-R SDK Level2 Reference Guide.* Sony Corporation, 2004.

[33] H. Surmann, R. Worst, M. Hennig, K. Lingemann, A. Nuechter, K. Pervoelz, K.R. Tiruchinapalli, T. Christaller, and J. Hertzberg. *RoboCupRescue - Robot League - Team KURT3D, Germany*

[34] R. Szeliski. *Video Mosaic for Virtual Environments* IEEE Computer Graphics and Applications, 16(2):22-30, March 1996.

[35] M. Veloso, S. Lenser, D. Vail, M. Roth, A. Stroupe, and S. Chernova. *CMPack-02: CMU's Legged Robot Soccer Team* Carnegie Mellon University, Pittsburgh, PA (USA), 2002.

[36] USAR Robot Competitions.
http://www.isd.mel.nist.gov/projects/USAR/competitions.htm

[37] http://www.aass.oru.se/Agora/RoboCup/Rescue/