UNIVERSITÁ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in
Ingegneria Informatica

Implementazione di un sistema multi-camera per il pattugliamento perimetrale coordinato

Relatore
Prof. Schenato Luca

Laureando
Tamai Gianmario

Padova, 5 Aprile 2011

UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

Master's Degree in
COMPUTER ENGINEERING

Implementation of a coordinated multi-camera perimeter patrolling system

Supervisor
Prof. Schenato Luca

Examinee
Tamai Gianmario

Accademic Year
2010-2011

I was born in Tunisia,
grown in Egypt and
i'm fighting in Libya and Yemen.
I will rise in all Arab countries
until i reach Palestine.
My name is Freedom

Abstract

Questo lavoro di tesi nasce dalla attiva collaborazione tra l'Università di Padova e la Videotec S.p.A..
La tesi ha lo scopo di implementare l'algoritmo di pattugliamento perimetrale proposto dal gruppo di Sistemi di Controllo del Dipartimento di Ingegneria dell'Informazione sul testbed fornito dall'azienda. Il suo principale obiettivo è quello di adattare il problema teorico alle problematiche che sorgono quando si ci scontra con vincoli pratici. Il pattugliamento perimetrale rientra tra le caratteristiche che un sistema di videosorveglianza deve avere per essere completo ed interamente automatizzato. Nel nostro caso il sistema proposto ha un approccio distribuito che, a differenza degli strumenti che l'azienda ha già a disposizione, fornirebbe degli spunti interessanti per quanto riguarda la gestione delle risorse e delle telecamere in caso di guasti. Le principali problematiche che si sono trattate sono, in primo luogo lo studio dell'algoritmo proposto e le eventuali estensioni da adottare per estenderlo dal semplice pattugliamento 1-D in un ambiente 3-D. In particolare è stato suggerito un approccio per il controllo dei parametri della telecamera PTZ (come ad esempio il controllo dello zoom e della velocità angolare). Un altro importante contributo che è stato proposto è una possibile architettura distribuita per il patrolling perimetrale che consenta, in uno sviluppo futuro, di integrare attività di tracking. Dopo l'analisi svolta è stata sviluppata una demo che è stata testata del tesbed aziendale per il controllo di due telecamere in ambiente interno. Il software sviluppato, anche se in fase embrionale, ha subito risposto agli obiettivi proposti.

Contents

Acronym

PTZ	Pan Tilt Zoom
ACK	Acknowledgment
UML	Unified Model Language
SVD	Singular Value Decomposition
USB	Universal Serial Bus
CGS	Camera Ground System
CPGS	Camera Position Ground System
IPC	Inter Process Communications
CRF	Camera Reference Frame
WRF	World Reference Frame

Chapter 1

Introduction

Nowadays we can assist a grown in demand of security.
In every crossroad we can find a fixed camera that controls a small area or, in a more complex case, a group fixed camera that controls public places.
As we can see the number of cameras involved in a video-surveillance system depends by the size of monitored area. In a big place such as oil platforms, military bases, undergrounds and airports a camera fixed system is not a suitable solution.
For this reason a natural evolution of fixed camera is a PTZ camera that can moves itself through pan (horizontally) and tilt (vertically) movement with various levels of zoom.
Thanks to the new type of camera, large areas can be monitored with a limited number of terminals that can be moved by a user in order to tracking any events.
However new mechanism have to be implemented to aid human operators and for guarantee a fair coverage of areas.
In fact, thinking to large area that have to be controlled, an operator that manages overall security system has to monitorize a lot of videos and, consequently, one user cannot be sufficient to control the video-surveillance system.
From this stems the need of automated tools in order to patrol large areas and to track activities that detect and follow an event that occurred.

Our work is focused in the implementation of perimetral patrolling tool using PTZ cameras. The project is committed by Videotec S.p.A. in collaboration with the Department of Information Engineering of Padova's university.

1.1 Patrolling problem

In [1] patrol activity was defined as *the act of walking around an area in order to protect or supervise it.* Taking that definition, a good patrolling strategy is one that minimizes the time lag between two visits to the same location, ensuring that all locations are constantly monitored. There are some interesting varieties of patrolling problem that can facilitate the operator of video-surveillance system.
Indeed we can consider the coverage area problem, that consists in finding the optimal subdivision of the controlled area and in assigning that sub-area to every camera.
Another important kind of outdoor system scenario is the perimeter patrolling. Unlike previous approach, its surveillance is limited to one dimensional boundary of the area to be protected.
This kind of problem can have different architectures such as distributed or centralized.

Normally the patrolling activities are implemented in centralized structures where a central computing unit manages the information and controls the movement of each agent.
It is easy to understand that, in this architecture, the growth in number of agents raise up the computational complexity of the task.
This architecture evinces some leaks such as the difficulty in scheduling different tasks for each camera and a non scalable system, on the other hand it guarantees rapid fault detection and the agreement between for each camera tasks. Some more recent systems use distributed architectures. This improvement brings a normal PTZ to become a smart camera that has a processing unit and can take decisions in function of its local informations.
In other words the computational power of centralized architectures is distributed on overall system.

This approach has an important advantage that is scalability and it results more robust with respect to a centralized system, in managing complex events, in detecting fault and in adjusting patrolling bounds.

The system that we show, adopts a distributed architecture and treats the perimetral patrolling problem in his mono-dimensional definition.
In the next subsection we will try to take a brief review on literature of handled problems regarding distributed patrolling.

1.1.1 Previous work

In literature the patrolling problem shows analogies with the dynamic optimal coverage in sensor networks. As shown in [2] and [3] a team of mobile agents coordinates themselves to gain a distributed coverage of an area avoiding collision.
Indeed, in robotic system some important considerations are raised up in [4] where a multi-agent cooperative method is proposed to be robust and adaptive to perimeter change and a efficient communication is taken into account.
In [5] and [6] through graphs analysis an optimal strategies are studied for multi-agent patrolling.
Some interesting papers are [7] and [8] which talk about the concept of equitable partitioning in multi-agent robotic systems. In this scenario the mainly idea is to portion the operational space into balanced areas of influence considering also the physical constraints of any agents.

1.2 Videotec company

The Videotec S.p.A. works in the field of video-surveillance since 1986, year of its foundation.
It started its business being only an engineering industry, but then, reading the evolution of the market and the growing demand of new generation cameras, it proposed several types of camera. Nowadays the company submits many products that work in different scenario; from simple fixed cameras to explosion or vandal proof cameras.

Very interesting for our work is the Ulisse products line.
Ulisse products are PTZ cameras that integrate a high speed 360° rotating Pan and Tilt head with a camera housing. These products are ideal to be used in all kind of application for outdoor dynamic video surveillance.
Linked to its products the company has implemented a very interesting video agent called Albert. It's a distributed intelligence agent that cooperates with other units detecting events and patrols areas.
In our work we used the Ulisse series cameras to implement and test the algorithm proposed.

1.3 Contributions

In previous sections we saw the context of our work, we analyzed the problems and we found some instruments to solve them.
In the next pages we will explain the core-arguments of this thesis. Now we give to the reader the main improvements of our work.

- **Extension of the proposed algorithm: from 1-D line to PTZ line definition**: We propose an extension of the algorithm described in 2 to PTZ cameras. This algorithm is limited to pan movement. We expand it also to tilt movements and we propose a new improvements to yield more usable the patrolling system such as velocity and zoom controls.
- **Design and analysis of patrolling system**: we suggest a software architecture that fit our patrolling algorithm. In particular we describe the controllers involved in this system, their behaviour and characteristics.
- **Results in Videotec Testbed**: we give a briefing of our implemented architecture and we show our results in Testbet.

1.3.1 Thesis outline

We are close to the end of this introduction and we propose a view of the thesis structure chapter by chapter.

- Chapter 2, *A theoretical analysis of perimetral patrolling problem*: we will treat the mathematical definition of the problem proposed by the University of Padova in [9]. We will report the solution of distributed definition and a complete description of one of the problems about variants (Synchronous Gossipe-Type Protocol).

- Chapter 3, *Patrolling trajectories design: from 1-D to PTZ parameters*: in this chapter we will describe our mapping functions that bring our 1-D definition of the patrolling path to PTZ definition. We will also propose a suitable multi-camera calibration step for perimetral patrolling.

- Chapter 4, *Software architecture: design and implementation*: we will report the analisys of requirements with UML diagrams and an input-output analisys. Finally we will give a complete vision of our architecture describing the controller involved.

- Chapter 5, *Test results*: we will give to the lector our results, in particular the computation of angles using the algorithm proposed in [12], the convergence test of our program with a simulation of a group of cameras and the test made by using two cameras in Videotec Testbed.

- Chapter 6, *Conclusions and future developments*: we will report a brief review of our work, and an analysis of the results obtained. We will propose some future developments such as the distributed manage of velocity, the task assigment problem and the extension of the proposed algorithm for covering areas.

Now let's start with a theoretical analysis of the proposed algorithm in its variants in order to introduce the 3-D extension of the patrolling line.

Chapter 2

A theoretical analysis of perimetral patrolling problem

Starting to the previous considerations, arises the perimeter patrolling problem that was proposed in [9].
For a more widely vision, we start explaining the mathematical definition of the problem and the partitioning problem of the perimeter with its three approaches.

2.1 Definition of the problem

Given $\mathcal{L}$ as the perimeter to be patrolled, it is defined as $\mathcal{L} = [-L, L]$ where $L > 0$.
We call N, the cardinality of the cameras that have to patrol the line. We label the N cameras in crescent order from 1 to N.
Every camera has the following proprieties:

- it has 1-d.o.f. The field of view of each camera can change due to pan movements only (more ahead we explain how we extend this limitation also to tilt movement).

- it has fixed coverage range. During its movements the coverage range is unchanged.

- it has point f.o.v..

Now, under previous assumptions, for i-th camera we define that:

- $D_i = [D_{i,inf}, D_{i,sup}] \subset \mathcal{L}$ is the total coverage length of i-th camera due to scenario topology, agent configuration and its physical constaints.

- $v_i \in [-V_{i,max}, +V_{i,max}]$ is the (bounded) speed of the i-th camera during its movements.

- $A_i = [a_{i-1}, a_i]$ is the effective coverage range of i-th camera during patrolling activities. Obviously $A_i \subseteq D_i, \forall i \in 1, ..., N$;

- $z_i(t) : R^+ \to D_i$, is the continuous function that map the position of the f.o.v of the i-th camera as a function of the time variable t.

On our analysis we assume that the coverage ranges $D_i, i \in 1, ..., N$, satify the following interlacing constraints:

$$D_{i,inf} \leq D_{i+1,inf}, D_{i,sup} \leq D_{i+1,sup} \tag{2.1}$$

We introduce a proprely cost function J to define the patrolling problem. We can take J as a monotonic function of the time lag T_{lag} defined as the maximum elapsed time between two visits of the same location. More simply the minimization problem correspond to the computation of the smallest time lag T_o, constrained to the system dynamics.

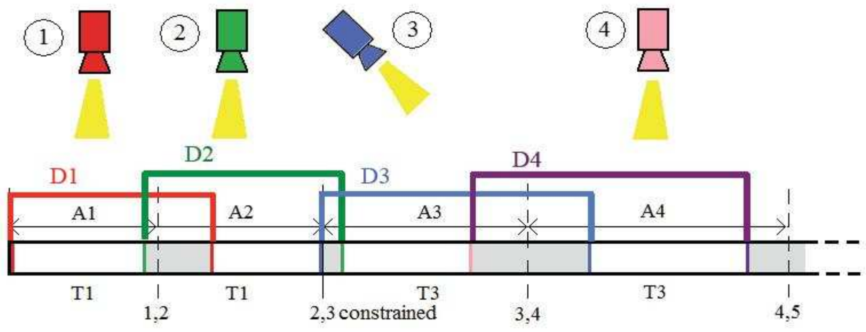


Figure 2.1: *Example of perimeter patrolled by a camera set. We can see the physical coverage D_i with the optimal partition domains A_i*

Now for a moment, we leave out the physical constraints of each camera. We gain the optimal coverage of the whole perimeter assuming that each camera patrol the path with its maximum speed $[V_{i,max}]$ with a periodical motion of period $\overline{T}$. The area length $|A_i|$ and the optimal period $\overline{T}$ are obtained in this way:

$$|A_i| = |V_{i,max}|\, T_o and T = 2T_o = \frac{2L}{\sum_{i=1}^{N} |V_{i,max}|} \tag{2.2}$$

Starting to this easy problem, we introduce the constrained solution.
In general a bounded solution is different to an unconstrained one; this solution could be the same only if the found solution with (1) is feasible ($A_i \subseteq D_i$). Called $T_{o,c}$ the optimal patrolling period with constraints, we have $T_{o,c} \geq T_o$. Appling a Divide and Conquer approach to this problem, the authors propose in [9] this solution:
If the uncostrained solution yields $A_i \nsubseteq D_i$, the optimal coverage is attained by splitting the domain into two different subproblems ($\mathcal{L}^l = [-L, D_{i,inf}]$ and $\mathcal{L}^r = [D_{i,sup}, L]$) and considering them separately.
Being ${T^l}_o$ and ${T^r}_o$ the optimal periods for the subproblem, the global coverage period is obtained as $T_{o,c} = max{T^l}_o, {T^r}_o$.

In the next subsection we are going to consider the distributed scenario. We assume that, at the begining, each camera is initialized with its partition $A_i(0)$ that in general does not coincide with the optimal solution.

In every algorithm's step each camera is allowed to update its bounds using only local information comming from neighboring cameras. The goal of the solution proposed in the next subsection is to lead the cameras to reach the optimal steady-state configuration for patrolling extremes.

2.1.1 Distributed optimal partitioning problem formulation

We assime that at time $t = 0$ each camera is initlialized with a dominance interval $A_i(0) = [a_{i,l}(0), a_{i,r}(0)]$ where $a_{i,l}(0)$ and $a_{i,r}(0)$ are respectively the left and the right extreme of A_i. We hire that the set $A_i(0), ..., A_N(0)$ statisfies three contraints.

- *physical constraint*: $A_i \subseteq D_i$ for $i \in 1, ..., N$
- *covering constraint*: $\bigcup_{i=\{1,...,N\}} A_i(0) = \mathcal{L}$
- *interlacing constraint*: $a_{i,l}(0) \leq a_{i+1,l}(0), a_{i,r}(0) \leq a_{i+1,r}(0)$

Observe that the interlacing and the covering constraints imply that $a_{i,l}(0) = -L$ and $a_{N,r}(0) = L$. The distributed algorithm has to allow for each camera to update its bounds using only local information coming from neighboring cameras. During its evolution the algorithm have to meet all the three constraints and the set of dominance intervals have to converge to the optimal partition.

Analyzing the type of communication between two neighboring cameras we can obtain different solutions of the same problem.

2.2 Different strategies for solving the partitioning problem

The authors in [9] propose different approach. In particular they propose the *synchronous solution* where, at each communication round, each camera transmits to its neighbors the information related to its current dominance interval. After that, they relaxed the synchronism and they proposed the *gossip-type*

communication protocol where at each iteration of the algorithm only a pair of neighboring cameras communicate with each other.
The authors suggest another subdivision of the same protocol: they propose the symmetric and asymmetric variants.
In the symmetric version, only one pair of neighboring cameras share their information and the communication occurs in both directions, while in the asymmetric one the exchange of information occurs in only one direction; this mean for example that one camera sends only its information and the adjacent camera reads the received data.

We can understand that the asymmetric gossip-type protocol proposed needs less resources than the other solution proposed. For this reason we treat only this protocol version in our work. In the following paragraph, we are going to explain in detail the asymmetric gossip-type protocol.

2.2.1 Asymmetric gossip-type algorithm

In [9] (section VI) the authors gave the description of the asymmetric gossip-type algorithm that doesn't take into account the physical bounds. With opportune changes, we report the description of this algorithm considering all the constraints.

Now we subdivide the algorithm in two steps: the **Transmission iteration** and the **Extremes' iteration**.

- **Transmission iteration**: At each time $t \in N$, there is only one camera that transmits its information to one of its neighbors camera. (Without loss of generality we assume that i-th camera sends its bounds to i+1 camera).

- **Extremes' iteration**: For $hdivi + 1$ camera h left unchange its bounds. From the new information received, i+1 camera updates its extreme as:

Called $a_{temp} = \frac{a_{i+1,r}(t)v_i + a_{i,l}(i)v_{i+1}}{v_i + v_{i+1}}$, then

$$a_{i+1,l}(t+1) = \begin{cases} D_{i+1,l} & if\ a_{temp} < D_{i+1,l} \\ a_{i,r}(t) & if\ a_{temp} > a_{i,r}(t) \\ a_{temp} & otherwise \end{cases} \tag{2.3}$$

Now we give the specular version of the extremes' iteration for completeness. We take into account the case in which i-th camera updates its bounds according to the information coming from i+1 camera.

Called $a_{temp} = \frac{a_{i,l}(t)v_{i+1} + a_{i+1,r}v_i}{v_i + v_{i+1}}$, then

$$a_{i,r}(t+1) = \begin{cases} D_{i,r} & if\ a_{temp} > D_{i,r} \\ a_{i+1,l}(t) & if\ a_{temp} < a_{i+1,l} \\ a_{temp} & otherwise \end{cases} \tag{2.4}$$

We have just seen the description of the algorithm that we will adopt for our implementation of patrolling system. We are going to introduce our extension on the original problem, in particular the introduction of PTZ cameras and other changes due to pratical needs.

Chapter 3

Patrolling trajectories design: from 1-D to PTZ parameters

To reach our targets, we have to explore the way to describe our patrolling trajectory in the PTZ (Pan Tilt Zoom) parameters.
First of all we have to gain an istrument that generate our path defined in a 3-D enviroment, in particular a mapping function that relates the distance of one point to the origin and the 3-D point that is linked to that distance. After this, we will show what are the reference systems involved, the parameter for calibrate a camera and our suggestion for calibrating a group of cameras that have to control a perimeter. Moreover, we describe the instrument that we have used to obtain a camera calibration using Matlab.
As we described in section 2, our model puts camera's f.o.v as a point. It is easy to imagine that, for an accurate patrolling activity, we want that a camera points a specific 3-D position given as input. From this arises the need of another mapping instrument that links a 3-D point P in the camera reference frame with *Pan* and *Tilt* angles that bring the camera to point P.
Finally we give two possible solutions to link image plane points to 3-D point.

3.1 Patrolling trajectory generation

As we said above, we give the way to obtain the patrolling path.
Using a PTZ camera we have 2 d.o.f; it implies a logical extension of the perimeter definition. Defined patrolled path as a 3-D line we can patrol a more complex perimeter. Given a set of 3-D points called $\mathcal{P}_{p2p}$, we suppose to obtain a patrolling path point-to-point definition in 3-D space. We need one black box function that accepts in entry the distance D from the origin of the line and returns the 3-D point P that has distance D from the origin.
In particular called $\mathcal{S}$ this function we have:

$$\mathcal{S} : \mathcal{L} \rightarrow \mathcal{P}$$

where

$$\mathcal{L} \in \mathbb{R}$$

$$\mathcal{P} \in \mathbb{R}^3$$

and $\mathcal{L}$ is the distance from the origin of the path of the 3-D point $\mathcal{P}$. To gain this function, we have to calculate the distance from the origin for each point in $\mathcal{P}_{p2p}$ and through the spline function we obtain a line that interpolate these points. The spline, as defined in [14], is a special function defined piecewise by polynomials. In this way we can obtain a function that could be evaluated in the domain of distances from origin. In other words we really stretch the 3-D patrolling path and we obtain 1-D line as shown in the following figure.

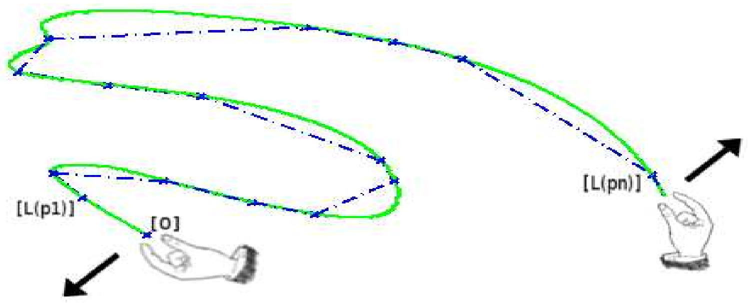


Figure 3.1: *Spline Function. The continue line is a path obtained by spline function, the doted line is the straight line that passes for each points*

Now we explain the steps to obtain this tool:

Algorithm 1 Algorithm that shows the use of spline function

```
{P is a vector composed by the points that define our trajectory}
$P \leftarrow def\_points()$
{cicle that compute a vector D of distances from origin. The i-th cell correspond to the distance from the origin of the i-th point}
$D_0 \leftarrow 0$
for $i = 1$ to N do
    $D_i \leftarrow distance(P_i)$
end for
{we compute a spline function}
$S \leftarrow spline(D, P)$
{now we can evaluate a spline S given as ingoing parameter the distance d and it returns the point P that is distant d from the origin}
$Point \leftarrow S(d)$
```

It is easy to understand the importance of this change. In fact we can apply our patrolling algorithm without minding the 3-D point managing, but we only work with the indexes of the path's array gained with spline function.

3.2 Definition of coordinate systems

In this section we are going to show what are the reference frames involved in a PTZ camera.

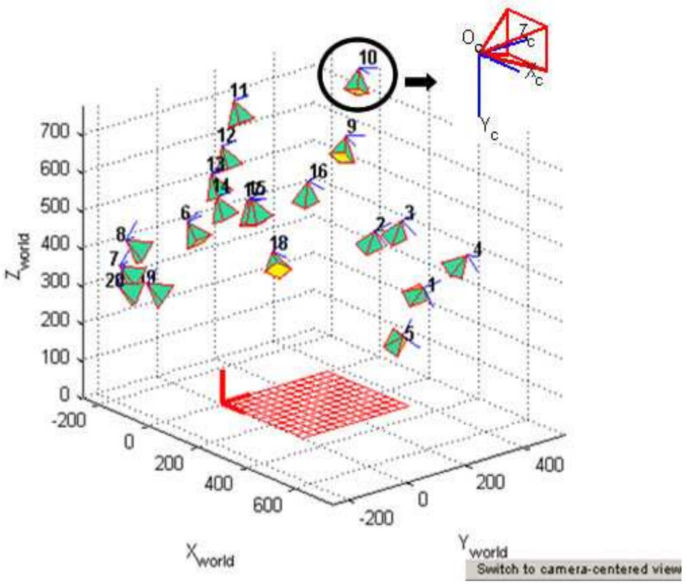


Figure 3.2: *Reference system involved in PTZ camera*

As we can see in Figure 3.2 there are two main reference systems:

- Camera reference frame (CRF): it is the system indicated in the image as Z_c, Y_c and X_c
- World reference frame (WRF): it is the system indicated as Z_w, Y_w and X_w in the image above

With this, we can represent the same point P in CRF and WRF.

There is a relation between CRF and WRF. In fact as we can see, a point expressed in CRF could be translate in the WRF with a rototraslation; now we are going to show how: Called,

- R the rotation matrix
- T the translaction vector

- P_c the point expressed in the CRF
- P_w the point expressed in the WRF

we have:

$$P_c = RP_w + T \tag{3.1}$$

and the inverse relation is:

$$P_w = R^T(P_c - T) \tag{3.2}$$

We underline that the rotation matrix R and the translation vector T are unique for each camera, more correctly we have to define R as R_i and T as T_i where i is the index of the camera.

We will show in next section how to gain these parameters.

3.3 Camera modeling

In this section we are going to introduce the model that we adopt to manage a camera. In particular we are going to explain what are the principal parameters that allow a conversion from image plane, CRF and WRF First of all we will introduce the intrinsic parameters that link image plane points to points expressed in CRF and then we will show the estrinsic parameters that relate CRF points to WRF points.

3.3.1 Intrinsic parameters

The intrinsic parameters are:

- Focal length $\boldsymbol{f_c}$: The focal length in pixels.
- Principal point $\boldsymbol{c_c}$: The principal point coordinates.
- Skew coefficient $\boldsymbol{alpha_c}$: The skew coefficient defining the angle between the x and y pixel axes.
- Distortions $\boldsymbol{k_c}$: The image distortion coefficients (radial and tangential distortions).

Now we are going to describe in a more explicit way the relation between image points and 3-D points.
Let P be a point in space of coordinate vector $P_c = \begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}$ in the camera reference frame.

$$p_n = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

After including lens distortion, the new normalized point coordinate p_d is defined as follows:

$$p_d = \begin{bmatrix} p_d(1) \\ p_d(2) \end{bmatrix} = (1 + k/c(1)r^2 + k_c(2)r^4 + k_c(5)r^6)p_n + d_x$$

Where $r = x_n^2 + y_n^2$ and d_x is the tangential distortion vector:

$$d_x = \begin{bmatrix} 2k_c(3)x_n y_n + k_c(4)(r^2 + 2x^2) \\ k_c(3)(r^2 + 2y_n^2) + 2k_c(4)x_n y_n \end{bmatrix}$$

Once distortion is applied, the final pixel coordinates $P_p = [x_p; y_p]$ of the projection of P on the image plane is:

$$\begin{cases} x_p = f_c(1)(p_d(1) + alpha_c * p_d(2)) + cc(1) \\ y_p = f_c(2)p_d(2) + c_c(2) \end{cases}$$

Therefore, in matrix notation:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = KK \begin{bmatrix} p_d(1) \\ p_d(2) \\ 1 \end{bmatrix}$$

where KK is the camera matrix defined as follows:

$$KK = \begin{bmatrix} f_c(1) & alpha_c * f_c(1) & c_c(1) \\ 0 & f_c(2) & c_c(2) \\ 0 & 0 & 1 \end{bmatrix}$$

3.3.2 Extrinsic parameters

Another important feature that this tool provides is the possibility to get a corrispondence with the 3-D world centered coordinate and the 3-D camera centered coordinate.

This mapping is obtained with the *extrinsic parameters*. In fact given P_w as a point space of coordinate vector $P_w = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}$ in the grid reference frame (take as world reference frame shown in Figure 3.2).

Let $P_c = \begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}$ the coordinate vector of the point P_w in the camera reference frame.

Then P_w and P_c are related by a rigid motion equation:

$$P_c = R_c P_w + T_c$$

where R_c is the rotation matrix and T_c is the translation vector, the last one indicates the distance between the camera center and the grid(world) center. As we can see the extrinsic parameters aided to compute the mapping function between CRF and WRF as we showed in equation 3.1.

Now we describe our tool that we used to compute these parameters underlining its propriety and function.

3.4 Cameras calibration and parameters estimation

To obtain the parameters that we have described, we used a Bouguet's Camera Calibration Toolbox for Matlab [15].
The toolbox we have used is based on [10]. In this paper, the authors describe four steps to calibrate a camera in order to obtain a mapping between 3-D reference coordinates and 2-D image coordinates. With this instrument it is possible to calibrate a camera on a grid as we will show in Figure 3.3. Through a relation of image pixel point of this grid and the dimension of grid's square that are known it can supplies the intrinsic parameter. Moreover, by the relation of points expressed in the grid reference frame and points expressed in the CRF, we can obtain the extrinsic parameters (this tool take the grid system as the WRF).
This tool provides two principal tools:

- **Single camera calibration**: this tool provides the intrinsic and extrinsic parameters of a fixed camera.
- **Dual camera calibration**: this tool provides the intrinsic and extrinsic parameters of both cameras to the same calibration board.

Let us define this tools starting from the single camera calibration.

3.4.1 Single camera calibration

This tool supplies our calibration parameters through these steps:

- **Take some snapshot of the calibration board in different position**: As we show in Figure 3.3, we can see different images of the same board placed in different positions.

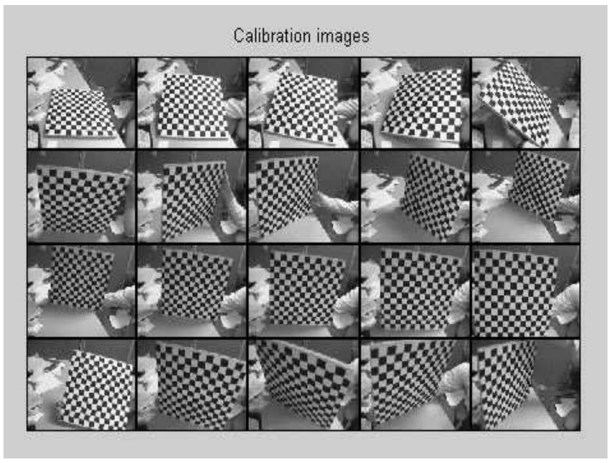


Figure 3.3: *Bouguet's toolbox – Picture to calibrate a camera*

- **For each image, we indicate the WRF in the grid**: As we can see in Figure 3.4 we have to indicate the placement of WRF for each photo.

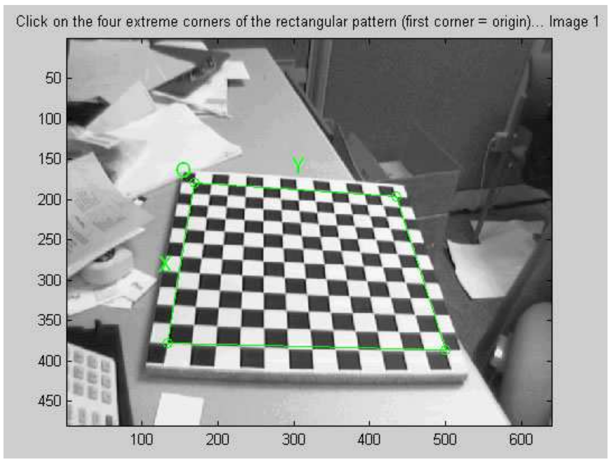

Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1

Figure 3.4: *Bouguet's toolbox – WRF in a grid*

- **Extraction of grid corner** : Through this command the tool provide the recognition of the point of the WRF that we have indicated in the previous step and after that, we can correct the wrong place corners. (Figure 3.5)

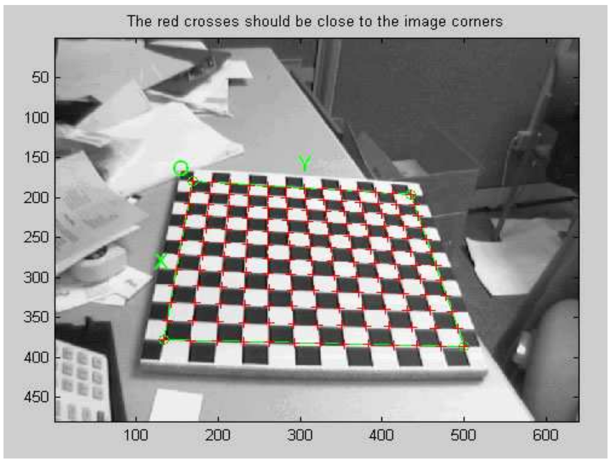
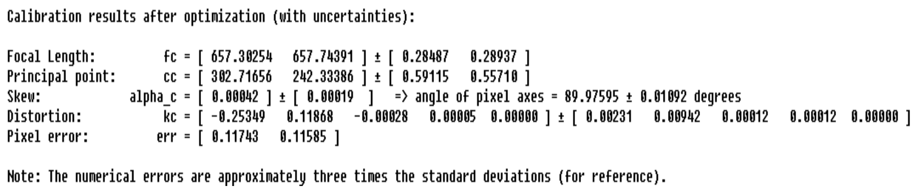


Figure 3.5: *Bouguet's toolbox – Corner extraction*

- **Extract intrinsic parameters**: in this step it provides the intrinsic parameters as we shown in Figure 3.6.



```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.30254   657.74391 ] ± [ 0.28487   0.28937 ]
Principal point:       cc = [ 302.71656   242.33386 ] ± [ 0.59115   0.55710 ]
Skew:             alpha_c = [ 0.00042 ] ± [ 0.00019  ]   => angle of pixel axes = 89.97595 ± 0.01092 degrees
Distortion:            kc = [ -0.25349   0.11868   -0.00028   0.00005  0.00000 ] ± [ 0.00231   0.00942   0.00012   0.00012  0.00000 ]
Pixel error:          err = [ 0.11743   0.11585 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figure 3.6: *Bouguet's toolbox – Intrinsic Parameters*

- **Extract extrinsic parameters**: After we have chosen a picture that indicates the final position of the grid and thus our WRF, through an extraction of grid corner for this image we gain the extrinsic parameters for WRF and CRF. (Figure 3.8 and Figure 3.7)

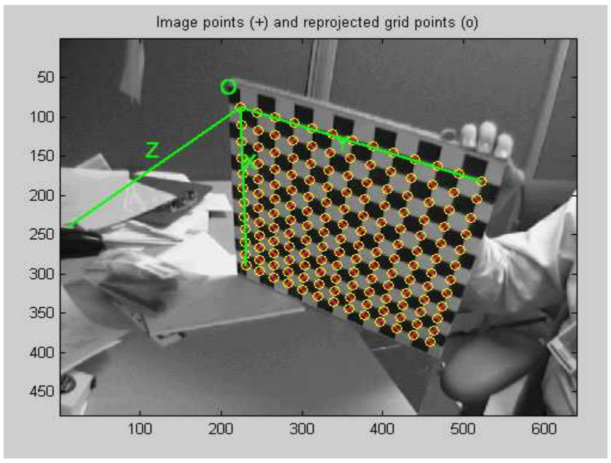
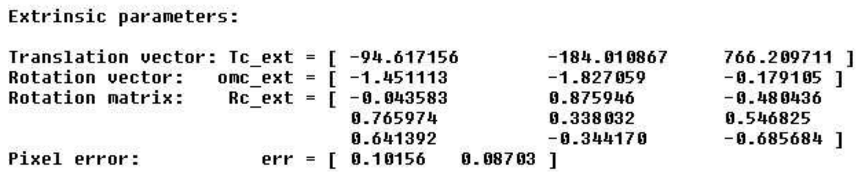


Figure 3.7: *Bouguet's toolbox – Image of WRF in the grid*



```
Extrinsic parameters:

Translation vector: Tc_ext = [ -94.617156        -184.010867      766.209711 ]
Rotation vector:   omc_ext = [ -1.451113         -1.827059        -0.179105 ]
Rotation matrix:    Rc_ext = [ -0.043583         0.875946         -0.480436
                              0.765974          0.338032         0.546825
                              0.641392          -0.344170        -0.685684 ]
Pixel error:           err = [ 0.10156   0.08703 ]
```

Figure 3.8: *Bouguet's toolbox – Extrinsic Parameters*

3.4.2 Dual camera calibration

As we said before, this tool provides a calibration of two cameras to the same grid, in particular we have to operate these steps to calibrate them:

- Place the grid in different positions and for each placement take for the right and the left camera an image.(as we have seen in Figure 3.3)
- Calibrate separately the two cameras with their picture and gain the intrinsic and extrinsic parameters as we have shown in subsection 3.4.1.
- Use the stereo calibrate tool to obtain the extrinsic parameter for each camera: as we show in Figure 3.9 we gain the graphical representation of our calibrate steps and the parameters that we have obtained.

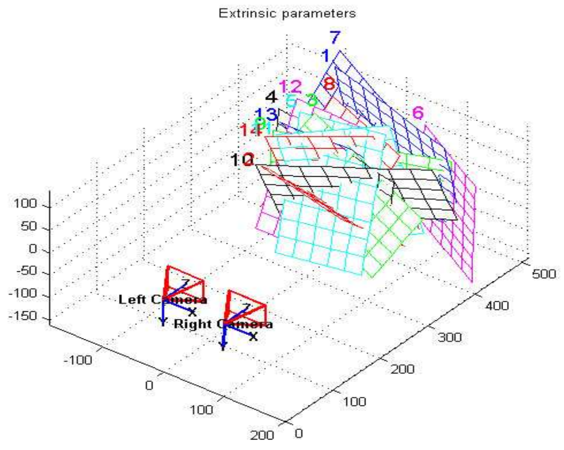


Figure 3.9: *Bouguet's toolbox – Intrinsic Parameters*

3.4.3 Mapping from 3D CRF point to image plane point

We have just described the mapping between image and 3-D world. Now, given a 3-D point P, we can compute the normalized point and finally we gain the pixel coordinates of a point P_p in the image.
It is also possible to produce the inverse mapping. In fact, given a pixel point P_p in the image we can produce the normalized point p_n with the function provided by the Bouguet's toolbox called:

$$\boldsymbol{normalize(P_p, f_c, c_c, k_c, alpha_c)} \tag{3.3}$$

Similarly to the intrinsic parameters, the uncertainties attached to the estimates of the extrinsic parameters omc, T_c are also computed by the toolbox. Those uncertainties are stored in the vectors omc_error, Tc_error.

We will see in the next section how to use The Camera Calibration Toolbox for Matlab to reach our scope, in particular how to gain a suitable calibration steps for our cameras system. In the next section we are going to introduce our proposal for calibrating a group of camera that have to do patrolling activity along a trajectory.

3.5 Multi-camera calibration

Given a set of N 3-D points expressed in world reference frame, for translating that points in a camera system we have to store for each camera the extrinsic parameters obtained calibrating them in the same world/grid frame. But it is an awful constraint because in a real situation we cannot have all cameras directed to the same point.

Among the constraints of the perimetral patrolling problem we can find one particular feature. The visual fields of consecutive cameras have to be overlapped, then we can calibrate the cameras two by two and take the grid frame of the first camera as the world frame. As we have shown in Figure 3.10,

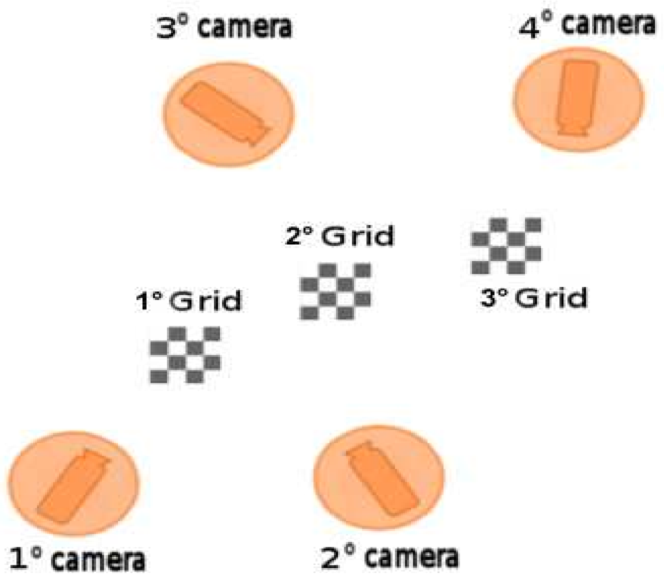


Figure 3.10: *Camera calibration two by two.*

if we take a random camera it borders with at most two cameras, then it has two distinct extrinsic parameters that it uses for translating a point received by a neighboring camera in his camera reference frame. Now we show how we have to operate.

Take the firs board as the WRF of the overall patrolling system and taking C_i as the i-th cameras with $1 < i < N$, we define:

- R^i_{gLeft} and T^i_{gLeft} : are the rotational matrix and the translaction vector that allow to translate a point from WRF of the left grids of the camera to the CRF of the i-th cameras.
- R^i_{gRight} and T^i_{gRight} are the rotational matrix and the translaction vector that allow to translate a point from WRF of the right grids of the camera to the CRF of the i-th cameras.

We note that these parameters are given by the extrinsic parameters obtained by a calibration to the grid. Now we show a diagram (Figure 3.11) that explains what are the definition of variables. Now called G_j as the j-th grid, where

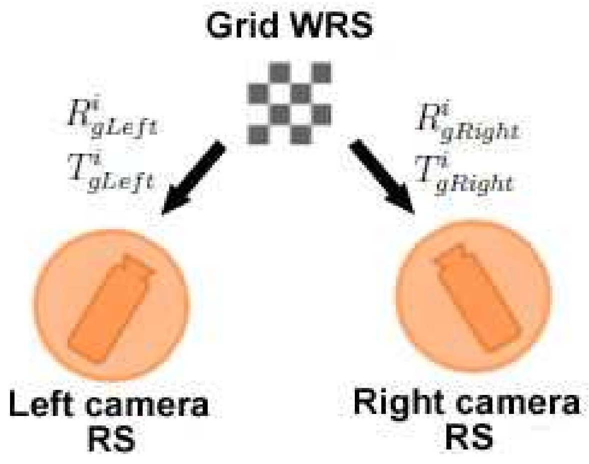


Figure 3.11: *Explanation of parameters for cameras calibration two by two.*

$1 < j \leq N$ (N is the cardinality of the grids); for each G_j we define a function $\mathcal{G}^j_{left}$ that translates a point from j-th grid to its left neighbor grid. We have:

$$\mathcal{G}^j_{left} : P_{G_j} \to P_{G_{j-1}} \tag{3.4}$$

where

$$P_{G_j} \in \mathbb{R}^3; P_{G_{j-1}} \in \mathbb{R}^3$$

are respectively the point of the starting grid and the point of the left grid of the P_G Let us define this function:

$$\mathcal{G}^{j}_{left}(P_{G_j}) = (R^{j}_{gleft})^{T}\left((R^{j}_{gRight}P_{G_j} + T^{j}_{gRight}) - T^{j}_{gleft}\right) \tag{3.5}$$

After this function definition we obtain a chain of functions that allows us to compute a mapping function from a point in anyone grids to the first grid. In fact, starting to k-th grid to obtain a function that translates a point from that grid to the firts grid we have:

$$\mathcal{G}^{2}_{left}(....\mathcal{G}^{k-1}_{left}(\mathcal{G}^{k}_{left}(P_{G_k}))) \tag{3.6}$$

Whereas the calibration between two consecutive cameras is provided by Camera Calibration Toolbox for Matlab as we have seen in 3.4.2, then our setting system is a suitable solution.

Until now, we have spoken about the camera calibration and we have proposed a suitable system to obtain the trajectory definition. Starting from this issues, in the next subsection we are going to treat about the computation of pan and tilt angle necessary to move the camera pointing a 3-D point given in the entrance with the aim to describe the patrolling path with Pan, Tilt and Zoom parameters.

3.6 Patrolling trajectory in PTZ parameters

As we said above, we have to gain a mapping from 3-D point to pan and tilt angles. First of all, to gain these angles that bring a camera to point a specific point P we have to introduce a mathematical model for PTZ camera.
The camera's reference frame spoted a point in the world as a triplet of parameter as shown in Figure 3.2.
Given a point in the space called P, if this point is in the center of the camera image plane, it has the normalized point:

$$p_n = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Indeed, the coordinates x and y of P are 0 while z-coordinate is a constant. We present two models to compute pan and tilt angles, the first one is an intuitive solution while the second one is more complex and design the cameras that we use.

3.6.1 Simple PTZ camera model

Now we give the following theoretical model for stylizing a camera that is proposed in [11].

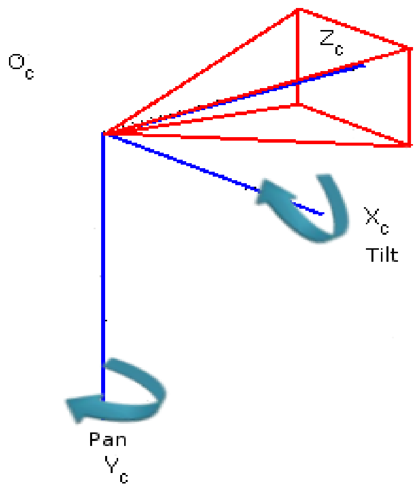


Figure 3.12: *Camera reference frame. The red pyramid is the camera and in blu we denote the reference system.*

As evidenced in the reduced model in Figure 3.12 using a rotation around x and y axes we can gain the pan and tilt rotation respectively.
Now, given in entrance a 3-D point P, we have to compute the pan and tilt angle that bring the camera to point P. In an analog mode we can compute a pair of angles that, through rotations around x and y axes, brought a point on z axis. Those angles are the inverse pan and tilt angle. We show how to find these angle.

Given a point $P = \begin{bmatrix} x & y & z \end{bmatrix}$ and we call the above angles θ and ϕ where θ is the inverse pan angle and ϕ the inverse tilt angle. Now we have:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sqrt{x^2+y^2+z^2} \end{bmatrix}$$

Given this matrix equation, it is easy to obtain Pan and Tilt angle through the solution of trigonometric equations with cramer's rule.

This model unfortunately is not realistic. In fact we have supposed that the rotation axes are centered to the same point (the origin of the system).
In a lot of PTZ cameras (such as our Ulisse cameras) this model is not suitable because the camera has axes that are non-centered.
Now we offer an offset based model that is suggested in [12] by ETH control group.

3.6.2 Offset based PTZ camera model

A realistic PTZ camera do not only has pan and tilt axes that are intersected. In fact taking our Ulisse PTZ camera shown in Figure 3.13 we can see how the

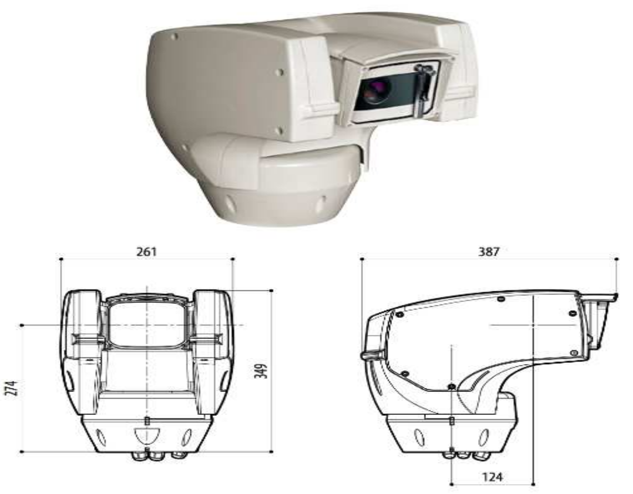


Figure 3.13: *Technical detail of Ulisse Compatc camera.*

movements engine are off-axes.
The model proposed by ETH is presented in the next figure (N.B. : Pay attention, camera reference system is not the classical system that we have show in Figure 3.2).

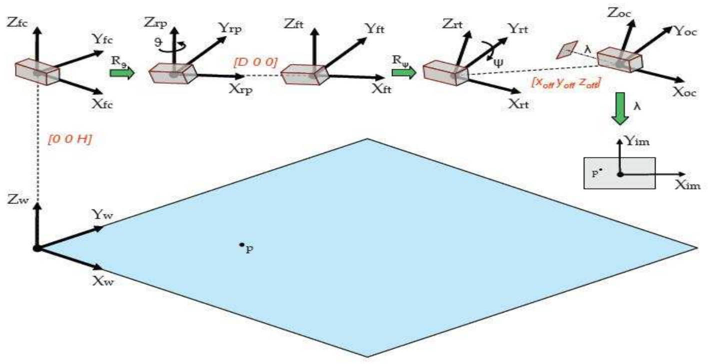


Figure 3.14: *Offset camera model .*

As we can see, given a combination $C(pan, tilt)$ of pan and tilt rotation and a point P obtained by a calibration of a camera starting to C, we can translate $P = [x_{oc}; y_{oc}; z_{oc}]$ in the original camera system $(XYZ)_{fc}$ through the series of matricians operations proposed in 3.7. We call original camera system, the camera reference system obtained where pan and tilt angles are both 0.

$$P_w = \begin{bmatrix} 0 \\ 0 \\ H \end{bmatrix} + R_\theta \left(\begin{bmatrix} D \\ 0 \\ 0 \end{bmatrix} + R_\phi \left(\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \begin{bmatrix} x_{oc} \\ y_{oc} \\ z_{oc} \end{bmatrix} \right) \right) \tag{3.7}$$

Where:

- $D, x_o, y_o and z_o$ are the offset shown in Figure 3.14. Respectively D is the offset between pan and tilt axes, where x_o, y_o and z_o are the offset between tilt axes and the camera's hole.
- R_ϕ and R_θ are rotation matrices. R_ϕ is the rotation matrix along tilt axis (Y-axis) and R_θ is the rotation matrix along pan axis (Z-axis).
- H is the height of camera from ground.

3.6.3 Computation of Pan and Tilt angles

Given this model, it is easy to obtain a corrispondence between points in the ground and pan and tilt angles. In fact given a point $P_w = [x_w, y_w, 0]$ we have to compute θ and ϕ to obtain a $P_{oc} = [d, 0, 0]$, where d is the distance between the point and the camera (under the assumption that the target is centered we have $y_{oc} = z_{oc} = 0$). Let us to explain in detail how to gain pan and tilt angles.

$$\Delta = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix} = \begin{bmatrix} D \\ 0 \\ 0 \end{bmatrix} + R_\phi \left(\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \begin{bmatrix} x_{oc} \\ 0 \\ 0 \end{bmatrix} \right)$$

we have

$$\Delta = \begin{bmatrix} D + (x_o + x_{oc}) \cos\phi + z_o \sin\phi \\ y_o \\ z_o \cos\phi - (x_o + x_o c) \sin\phi \end{bmatrix}$$

substituting Δ in 3.7

$$\Delta = \begin{cases} \delta_1 \cos\theta - \delta_2 \sin\theta = x_w \\ \delta_1 \sin\theta - \delta_2 \cos\theta = y_w \\ H + \delta_3 = 0 \end{cases}$$

with appropriate changes we can obtain:

$$\Delta = \begin{cases} \delta_1 \cos\theta \sin\theta - \delta_2 \sin^2\theta = x_w \sin\theta \\ \delta_1 \cos\theta \sin\theta - \delta_2 \cos^2\theta = y_w \cos\theta \\ H + \delta_3 = 0 \end{cases}$$

Now, from the first two equations we can derive

$$\cos\theta = \frac{\delta_2 y_w \pm \sqrt{x_w^4 - x_w^2 \delta_2^2 + x_w^2 y_w^2}}{x_w^2 + y_w^2} \tag{3.8}$$

this yelds two solutions, but since we know x_w and y_w we can see what is the correct root.

For gaining ϕ we use the fact that $\delta_3 = -H$, from this stems:

$$\sin\phi = \frac{H(x_o + x_{oc}) \pm \sqrt{z_o^4 - H^2 z_o^2 + (x_o + x_{oc})^2 z_o^2}}{(x_o + x_{oc})^2 + z_o^2} \tag{3.9}$$

As we can see x_{oc} is unknown since we have only x_w and y_w. Considering the offset much smaller than x_{oc} we can approximate x_{oc} as:

$$\sqrt{(\sqrt{x_w^2 + y_w^2} - D)^2 + H^2}$$

We gain two solutions for ϕ from our equation, thus we have to choose the suitable solution starting from the values of x_w and y_w and H.

3.6.4 Zoom control

It is also important, in order to aid the human operator, to obtain for each camera's video an image that has the same rate between real and digital dimension. In other words when we frame an object and we know its sizes, we want to have similar images independently from its distance from the camera. For this reason we have to set a new zoom function $\mathcal{Z}$ defined as:

$$\mathcal{Z} : \mathcal{D} -> [z_{min}, z_{max}]$$

where, given a specific model of camera, z_{min}, z_{max} are respectively the minimum and maximum values of zoom supported. For simplicity we can set $\mathcal{D}_i$ as the distance between an object and i-th camera. Given $\mathcal{D}_i$, we can link a level of zoom z_{ok} and obtain a ratio between them that we call $\mathcal{R}_z = \frac{\mathcal{D}_i}{z_{ok}}$. Using that ratio we set:

$$\mathcal{Z}_i = \frac{\mathcal{D}_i}{\mathcal{R}_z} \tag{3.10}$$

3.6.5 Velocity control

When an operator manages the video-surveillance system in his monitor flows the video stream from each camera that, we suppose, is doing patrolling activities along a perimeter. To aid the operator's work we have to patrol our perimeter with an acceptable velocity (for example 5m/s) and each camera have to do its movement keeping the patrol velocity along the path constant. This restriction simplifies the computation of the camera's bounds because the velocity data are no longer necessary, but it implies an additional problem: keeping constant the patrolling velocity along the perimeter, we obtain variable velocities in pan ($Vpan_i$) and tilt ($Vtilt_i$) movements. Analyzing more in deep this problem we can find new constraints for our cameras. Indeed, taking one camera, it has two velocity constraints that we define in this way:

- $V_{a_{pan_i}}$ that is $V_{a_{pan_{i_{min}}}} \leq V_{a_{pan_i}} \leq V_{a_{pan_{i_{max}}}}$
- $V_{a_{tilt_i}}$ that is $V_{a_{tilt_{i_{min}}}} \leq V_{a_{tilt_i}} \leq V_{a_{tilt_{i_{max}}}}$

where $V_{a_{tilt_{i_{max}}}}, V_{a_{tilt_{i_{min}}}}, V_{a_{pan_{i_{max}}}}, V_{a_{pan_{i_{min}}}}$ are values depending on the type of camera that we are analyzing.
Picking up the definition of our problem given in precedence, we can see how the velocity constraint in section 2 change. In fact we have to find the max value for the linear velocity that satisfies the two constraints reported above during patrolling activity along the path; but it isn't easy because the path that one camera have to control changes during patrolling, thus a camera has different values of linear velocity during the evolution of the algorithm. Moreover our algorithm is a distributed one and it implies that one camera can have only local information; but to find a global common value for velocity is a very complex activity that involves our algorithm in all its parts.
In order to solve this problem we can make an off-line search for this value that let us set a correct value for the patrolling velocity along the path.

Another important issue that we propose is how to compute the angular velocities to keep constant the linear velocity along the trajectory. For each movement of our camera from two points we have:

- V_i the constant velocity along the patrolling path.

- $d_{P_i \to P_{i+1}}$ the distance between the starting point (P_i) of this movement and its next point (P_{i+1}).
- $\Delta\theta_i$ the relative pan angle to move a camera from P_i to P_{i+1}.
- $\Delta\phi_i$ the relative tilt angle to move a camera from P_i to P_{i+1}.

frist we compute:

$$\Delta\theta_i = |\theta_{P_i} - \theta_{P_{i+1}}|$$

$$\Delta\phi_i = |\phi_{P_i} - \phi_{P_{i+1}}|$$

$$d_{P_i \to P_{i+1}} = |P_i - P_{i+1}|$$

and thus we can simply gain the time elapsed for traveling from P_i to P_{i+1} as

$$T_i = \frac{d_{P_i \to P_{i+1}}}{V_i}$$

finally, the angular velocities for this movement

$$V_{a_{pan_i}} = \frac{\Delta\theta_i}{T_i} \tag{3.11}$$

$$V_{a_{tilt_i}} = \frac{\Delta\phi_i}{T_i} \tag{3.12}$$

From these values we can obtain another mapping function that relate points to angular velocities.

3.7 3-D patrolling trajectry from image plane points

In order to get out a 3-D patrolling path to apply the distributed algorithm described in chapter number 2 we must find a simply mechanism for a human to set up the principal path's points. We call principal path point a corner point that describes a broken line that approximates an ideal patrolling path. In fact we could not be satisfied to gain a simply list of some 3-D points, but we have to find a relation between image points, that are given by a human, and 3-D points of the world. We are going to present two principal solutions to obtain this relation. The first method impose a restriction for which all the 3-D points must lies in the ground; the second, using two cameras, shows how to compute a 3-D point that is placed everywhere in the space starting from two pixel points.

3.7.1 Single camera with planar trajectory

In our targets we have to move a camera through pan, tilt and zoom movements. For reducing the number of freedom degrees we choose to mind only pan and tilt. With this simplification we only need a list of 3-D points that lie in the same plane; it implies that z coordinate is the same for every points that belong to the path.the

For gaining a 3-D point that lie in a plane starts to image's 2-D points we use the method of Bouguet's software called *normalize* (that we have show in 3.4.3). In fact, after a calibration step, from a 2-D pixel point and camera calibration parameters a function produces a normalized point p_n that is:

$$p_n = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$

where $P_c = \begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}$ is a point expressed in a camera reference frame and $P_w = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}$ is a point expressed in world reference frame. But points lies in the same plane and in order to allow a simply conversion from 3-D point expressed in one camera reference system to another camera

reference system we set the z-coordinate of a point expressed in the world reference system to a constant k, $(Z_w = k)$. We set $k = 0$ for simplicity.
In this way through *extrinsic parameters* of every camera we can translate a point from the world system to the camera system. In fact, as it was explained in the previous sections there is a relation between 3-D points expressed in the world reference system and points expressed in the camera reference system.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{c_{11}} & r_{c_{12}} & r_{c_{13}} \\ r_{c_{21}} & r_{c_{22}} & r_{c_{23}} \\ r_{c_{31}} & r_{c_{32}} & r_{c_{33}} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} t_{c_1} \\ t_{c_2} \\ t_{c_3} \end{bmatrix}$$

Thus,

$$X_c = r_{c_{11}} X_w + r_{c_{12}} Y_w + r_{c_{13}} Z_w + t_{c_1}$$
$$Y_c = r_{c_{21}} X_w + r_{c_{22}} Y_w + r_{c_{23}} Z_w + t_{c_2}$$
$$Z_c = r_{c_{31}} X_w + r_{c_{32}} Y_w + r_{c_{33}} Z_w + t_{c_3}$$

But we have posed $Z_w = 0$ thus,

$$X_c = r_{c_{11}} X_w + r_{c_{12}} Y_w + t_{c_1}$$
$$Y_c = r_{c_{21}} X_w + r_{c_{22}} Y_w + t_{c_2}$$
$$Z_c = r_{c_{31}} X_w + r_{c_{32}} Y_w + t_{c_3}$$

Now if we want to find the world centered 3-D coordinate given a normalized point p_n we have to resolve the next system of equation where X_w and Y_w are unknowns.

$$\begin{cases} p_n(1) = \frac{r_{c_{11}} X_w + r_{c_{12}} Y_w + t_{c_1}}{r_{c_{31}} X_w + r_{c_{32}} Y_w + t_{c_3}} \\ \\ p_n(2) = \frac{r_{c_{21}} X_w + r_{c_{22}} Y_w + t_{c_2}}{r_{c_{31}} X_w + r_{c_{32}} Y_w + t_{c_3}} \end{cases}$$

extracting X_w and Y_w we gain the follower parameter:

$$J = r_{c_{12}} - p_n(1) r_{c_{32}}$$
$$H = r_{c_{21}} - p_n(2) r_{c_{31}}$$
$$L = r_{c_{22}} - p_n(2) r_{c_{32}}$$
$$K = r_{c_{11}} - p_n(1) r_{c_{13}}$$
$$T_1 = -t_{c_3} p_n(1) + t_{c_1}$$
$$T_2 = -t_{c_2} p_n(2) + t_{c_3}$$

And,

$$A = \frac{-JH}{K+L}$$

$$B = \frac{HT_1}{K+T_2}$$

Thus,

$$X_w = \frac{B}{A}$$

$$Y_w = \frac{-T_1 - X_w J}{K} \tag{3.13}$$

$$Z_w = 0$$

With this method we have found a mapping from 2D to 3-D world-centered points, but this system has an important weakness. In fact, given an image recovered by a camera with a defined zoom, we have a bounded vision of the world and for gaining a complete definition of patrolling path we have to implement a complex mechanism. This tool must gather all the information coming from each camera and adds them (according to common points that must be defined) in order to describe the path in a suitable world reference system. This changes are not treated in this thesis because we want to focus the implementation of the algorithm proposed.
Now we are going to propose a possible solution to multi-camera calibration in a perimeter.

3.7.2 Dual camera with 3-D trajectory

As we can see the previous method has a limitation that constrains our work. Now we introduce a method that links a 3-D point with two pixel points. As we show in Figure 3.15 we can see a schema that describes our enviroment.

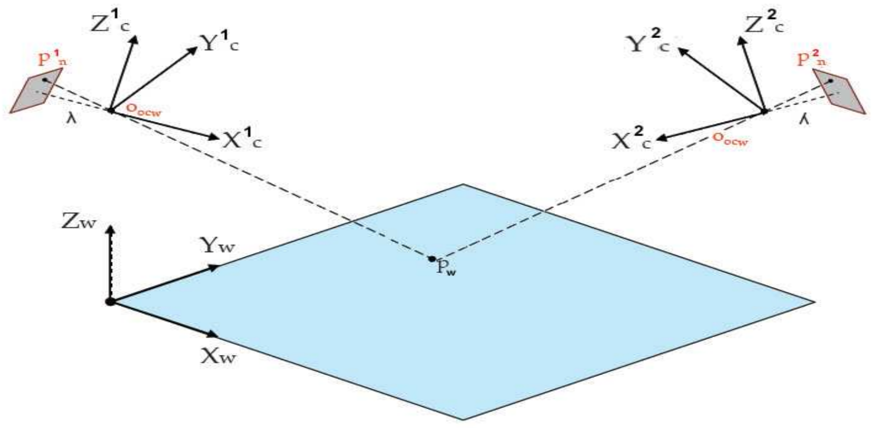


Figure 3.15: *A suitable model of distributed system.*

Now take a point P_w defined as:

$$P_w = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

It can be translated in the CRS of our cameras, we call P_c^1 and P_c^2 where the first is expressed in the first camera system and the second is expressed in the other camera. We have:

$$P_c^1 = R^1 P_w + T^1 \begin{bmatrix} X_c^1 \\ Y_c^1 \\ Z_c^1 \end{bmatrix}$$

$$P_c^2 = R^2 P_w + T^2 \begin{bmatrix} X_c^2 \\ Y_c^2 \\ Z_c^2 \end{bmatrix}$$

As we mentioned in section 3.3 we have a relation between point expressed in the CRF and pixel point and, in order to do it, we used the normalized tool provided by Camera Calibration Tool for Matlab in 3.4.3. In fact we have:

$$p_n^1 = \begin{bmatrix} X_c^1/Z_c^1 \\ Y_c^1/Z_c^1 \end{bmatrix} = \begin{bmatrix} x_n^1 \\ y_n^1 \end{bmatrix}$$

$$p_n^2 = \begin{bmatrix} X_c^2/Z_c^2 \\ Y_c^2/Z_c^2 \end{bmatrix} = \begin{bmatrix} x_n^2 \\ y_n^2 \end{bmatrix}$$

applying the method that we have used in the previous subsection we have

$$\begin{cases} x_n^1 = \frac{r^1_{c_{11}}X_w + r^1_{c_{12}}Y_w + r^1_{c_{13}}Z_w + t^1_{c_1}}{r^1_{c_{31}}X_w + r^1_{c_{32}}Y_w + r^1_{c_{33}}Z_w + t^1_{c_3}} \\ \\ y_n^1 = \frac{r^1_{c_{21}}X_w + r^1 c_{22}Y_w + r^1_{c_{23}}Z_w + t^1_{c_2}}{r^1_{c_{31}}X_w + r^1_{c_{32}}Y_w + r^1_{c_{33}}Z_w + t^1_{c_3}} \end{cases} \begin{cases} x_n^2 = \frac{r^2_{c_{11}}X_w + r^2_{c_{12}}Y_w + r^2_{c_{13}}Z_w + t^2_{c_1}}{r^2_{c_{31}}X_w + r^2_{c_{32}}Y_w + r^2_{c_{33}}Z_w + t^2_{c_3}} \\ \\ y_n^2 = \frac{r^2_{c_{21}}X_w + r^2 c_{22}Y_w + r^2_{c_{23}}Z_w + t^2_{c_2}}{r^2_{c_{31}}X_w + r^2_{c_{32}}Y_w + r^2_{c_{33}}Z_w + t^2_{c_3}} \end{cases} \tag{3.14}$$

Now we have three unknown parameters (X_w, Y_w and Z_w) and four equations and we are able to extract the coordinates of point P_w

The issues that we propose in this section are not important for our work because it is focused on the definition of patrolling trajectory in the pan, tilt and zoom parameters. To clarify our work we are going to introduce a review of our proposed mapping functions.

3.8 Review of mapping function: from image plane point to PTZ parameters

To summarize all the features we discovered, we propose the following schema:

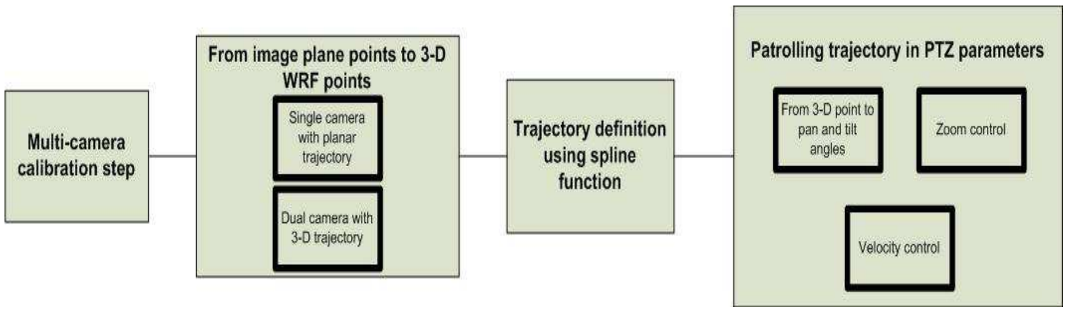

Multi-camera calibration step
From image plane points to 3-D WRF points
Single camera with planar trajectory
Dual camera with 3-D trajectory
Trajectory definition using spline function
Patrolling trajectory in PTZ parameters
From 3-D point to pan and tilt angles
Zoom control
Velocity control

Figure 3.16: *Review of mapping function.*

Let us describing the block diagram above:

- **Multi-camera calibration step**: As we described in section 3.5 we calibrate our set of cameras. This allows us to obtain the extrinsic and intrinsic parameters for each camera and a function that translate a point from anyone grid to the first grid take as the world reference system.

- **From image plane points to 3-D WRF points**: this step is obtained by the procedures described in section 3.7. The procedures are two. In the first we impose that the patrolling path lies on the ground, while the second method does not forces any conditions.

- **Trajectory definition using spline function**: as we have shown in section 3.1, we set up a continuously path using spline function. Starting from the 3-D points that are supply from the previous step we gain a corrispondence from distance from the origin of the Trajectory and 3-D point and vice versa.

- **Patrolling trajectory in PTZ parameters**: To translate a point from 3-D definition to PTZ definition we have to:

- Compute pan and tilt angles to bring a 3-D point in the center of the image. (subsection 3.6.3)
- Compute zoom parameters that keep the object observed to the same dimension during the patrolling activity. (subsection 3.6.4)
- Compute the angular velocity in tilt and pan movements for each point. (subsection 3.6.5)

3.9 Simulation

In this section we are going to test all the mapping function that bring a 1-D trajectory definition to PTZ definition with pan and tilt velocities.

First off all, we describe our simulation environment. It is composed by:

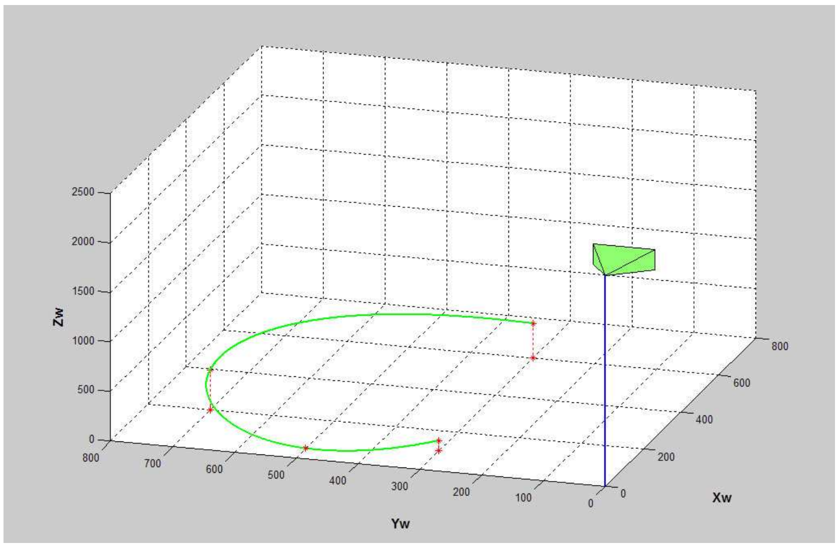


Figure 3.17: *Simulation environment*

- One PTZ camera modelled as a Ulisse Compact Camera
- Trajectory definition per point (see red crosses in Figure 3.17) and its spline approximation (see green line in Figure 3.17)

We note that in this figure, the camera is oriented as pan and tilt equals to zero. Running our simulation code written in Matlab, we gain the following diagram:

- **Camera position in 1-D trajectory definition**: Starting from a patrolling path described in our problem definition in section 2, we obtain a chart in which we analyze camera position along the time.

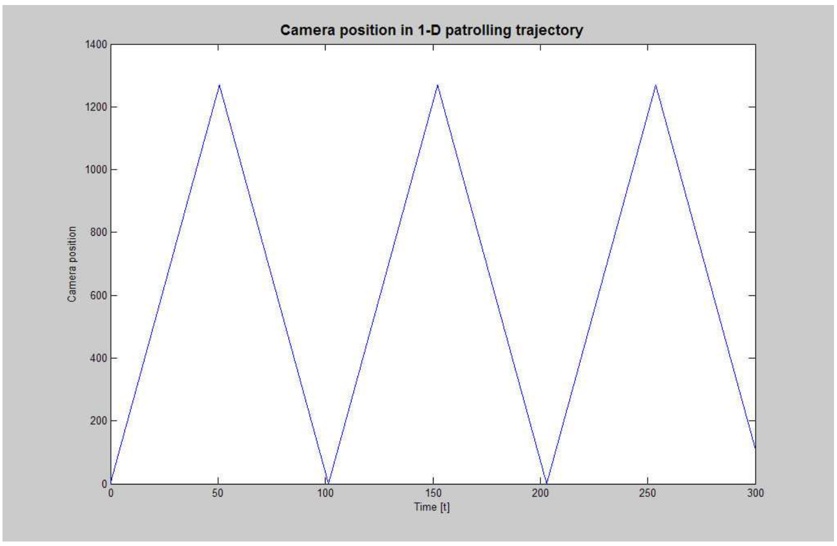


Figure 3.18: *1-D camera position along the time*

- **Pan angle**: We describe the movements of our camera in its Pan angle along the time.

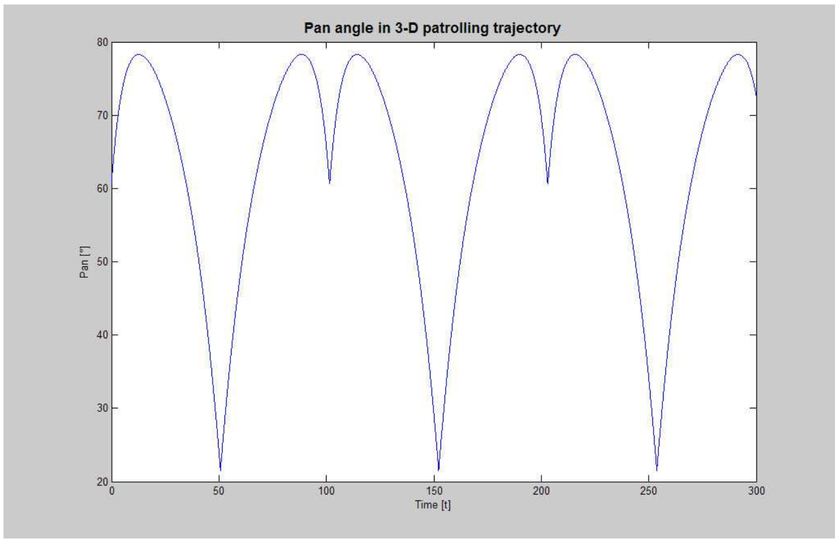


Figure 3.19: *Pan diagram*

- **Tilt angle**: We describe the movements of our camera in its Tilt angle along the time.

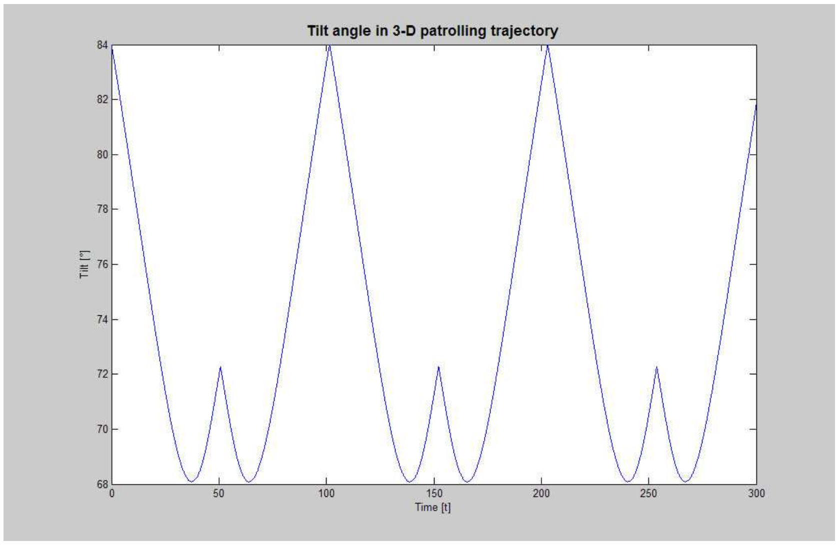


Figure 3.20: *Tilt diagram*

- **Zoom parameter**: We describe the variation of Zoom parameters along the time.

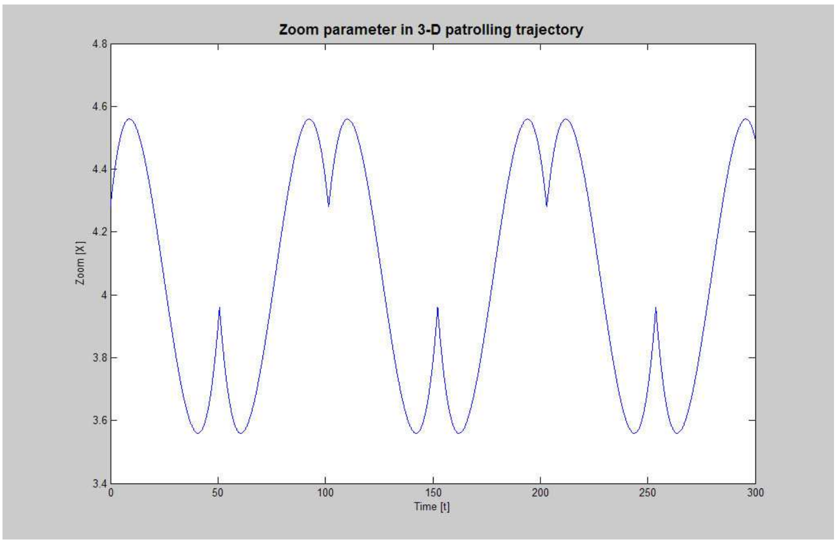


Figure 3.21: *Zoom diagram*

- **Pan angular velocity parameter**: We describe the variation of Pan's velocity during the time.

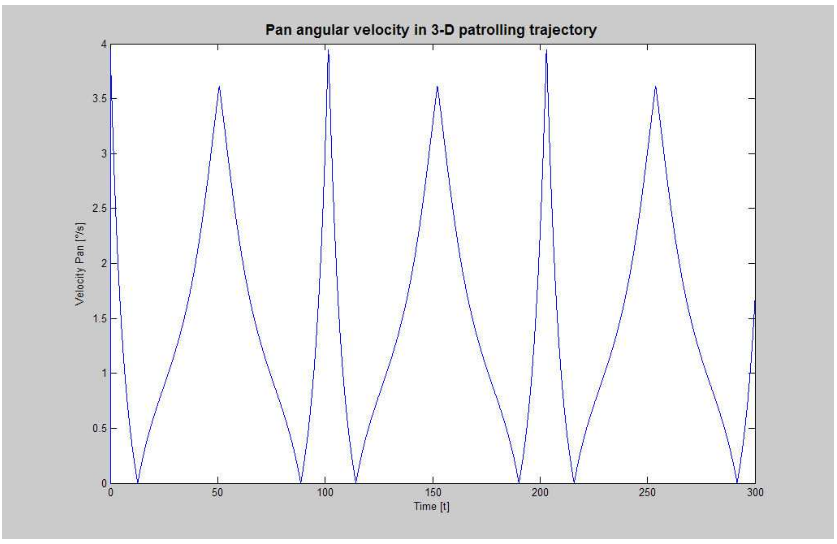


Figure 3.22: *Pan angular velocity diagram*

- **Tilt angular velocity parameter**: We describe the variation of Tilt's velocity during the time.

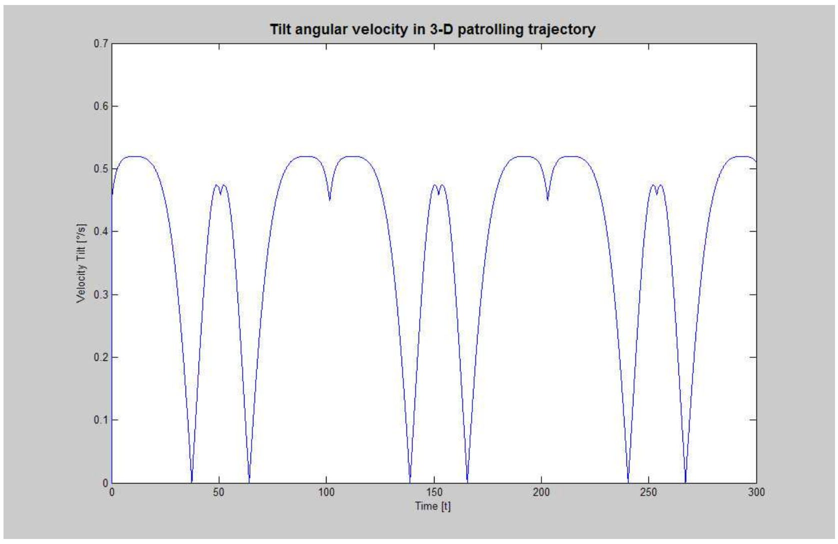


Figure 3.23: *Tilt angular velocity diagram*

These are the diagrams, obtained with our simulation program. We can see a few proprieties of them in particular:

- figurename 3.19 evidence how Pan angle depends more on trajectory definition then the height of camera and the distance of patrolling path to the camera.
- figurename 3.20 and figurename 3.21 show how Tilt and Zoom parameters depend largely on camera height and the distance between camera and trajectory.

Starting from this proposed analysis we are going to introduce a possible software architecture for the Videotec Testbed.
The software architecture that we are going to explain does not depend on camera calibration step, in particular we impose that we have the 3-D definition of our path with a spline function and we compute offline all the PTZ parameters that describe our trajectory.

Chapter 4

Software architecture: design and implementation

In next sections we will go in the detail of our patrolling software describing its component. Let us start with the software's analisis of requirements.

4.1 Requirements analysis of software

In light of the above consideration, we summarize the requirements of the system and define how it works. To describe as well the requiremens we choose to use UML to design our components. Our patrolling system is made by two main components: an user and a camera. The next figure describes a distributed system.

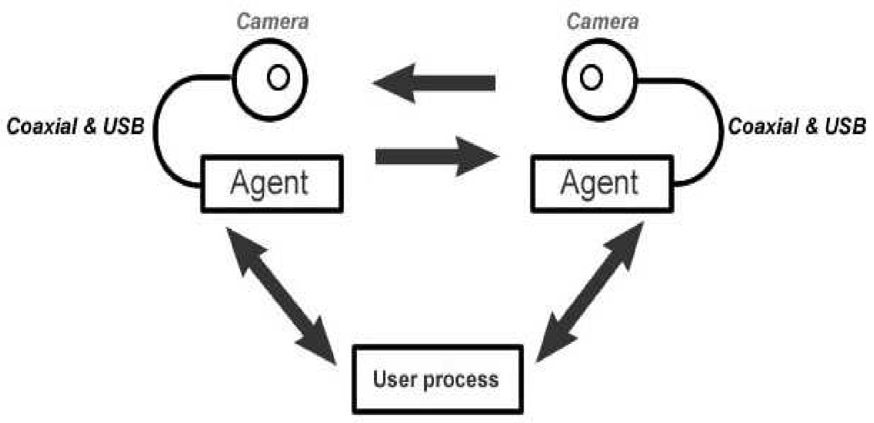


Figure 4.1: *A suitable model of distributed system.*

Now we are going to present the camera component. In a first instance we are going to present a statechart diagram that models the behavior of the component and its state along the time; after that, we will propose an activity diagram that shows what are its steps and controllers. Finally we will present an usecase diagram for the user component.

4.1.1 Camera component analysis

The camera component must manages only one camera and move it along patrolling path which was defined before. To describe its behavior over time, we can take it as a finite states machine. In fact this type of diagram emphasizes state's definitions and how the process changes its state over time.

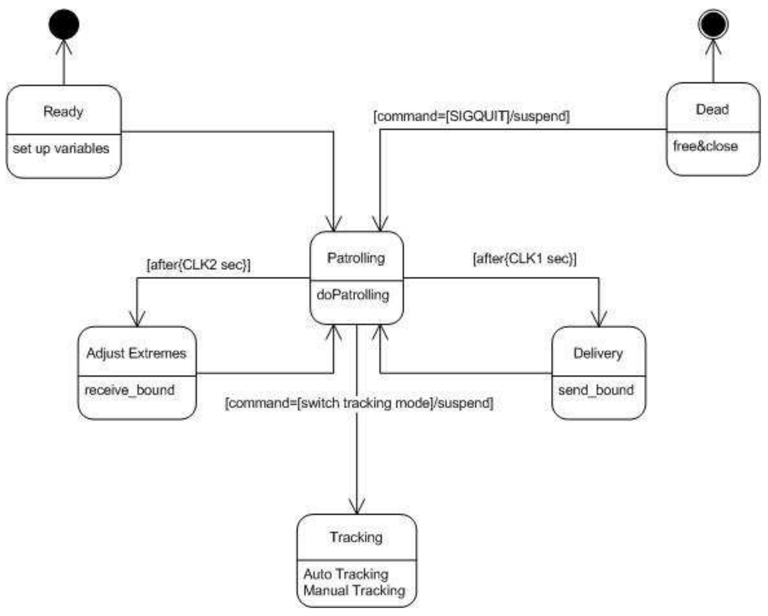


Figure 4.2: *UML Statechart diagram for Patrolling component.*

In Figure 4.2 we propose our ***statechart diagram*** that describes camera process.
First of all, we can see six main states that are activated by switching conditions. Now we give a description of camera component for each state:

- **Ready**: it is lunched and set up the initial enviroment variables.
- **Dead**: it is aborted after a user's command.
- **Patrolling**: it does the patrolling activity, such as move camera and controls it zoom and velocity.
- **Adjust Extremes**: it receives and computes the new bound of patrolling path according to its red data. This state stems by the Extremes' iteration of our patrolling algorithm proposed in section 2.2.1.
- **Delivery**: it sends its bound to neighboring cameras.
- **Tracking**: it does tracking activities. This state includes the Auto tracking and the Manual tracking state.

We summarize the description of the diagram in the next table.

Start state	**Final state**	**Event**	**Guard**	**Procedure**
Ready	Patrolling			inizialize enviroment variabiles
Patrolling	Adjust Extremes	after CLK_2 sec		
Patrolling	Delivery	after CLK_1 sec		
Patrolling	Auto-tracking	command received	command=auto-tracking	suspend
Patrolling	Manual-tracking	command received	command=manual-tracking	suspend
Patrolling	Dead	command received	command=SIGQUIT	suspend

Table 4.1: *Statechart table for diagram in Figure 4.2*

The table proposed above shows which are the state transitions, the events and the conditions that make the changes possible and the procedure that the process has to execute.

This diagram is focused in the description of process state but it doesn't explain very well the activity that it must do during the evolution over time. In order to do that we could be aided by the ***activity diagram*** in Figure 4.3.

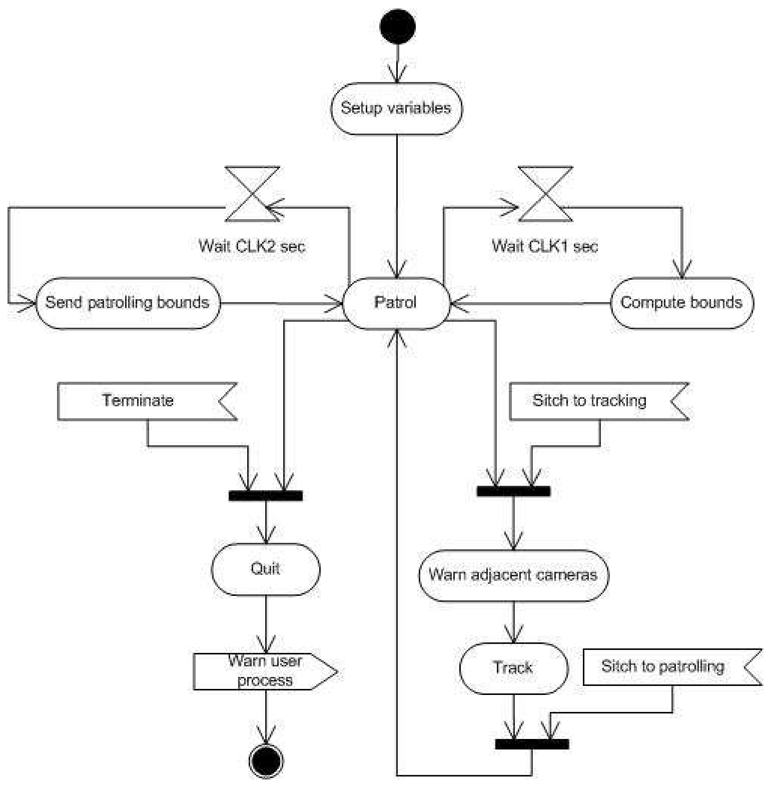


Figure 4.3: *UML Activity diagram for Patrolling component.*

For an easy lecture of the diagram we have to think how the processor's time needs to be associated to the patrolling activities. Moreover, this type of diagram shapes very well the communications signals between camera process and user process. In this diagram we can see six activities of camera process:

- **Setup variables**: it sets up the enviroment variables.
- **Send patrolling bounds**: it sends its bound to neighboring cameras.
- **Compute bound**: it receives the neighboring's bound and compute its new bound.
- **Warn adjacent camera**: it warns its neighboring that it has to leave the patrolling activity for entering in the tracking activity.
- **Track**: it does the tracking activity.
- **Patrol**: it does the patrolling activity. It moves the camera from the left to the right bounds, computed in *compute bound* activity.

- **Quit**: it deallocates the memory and closes the process.

There are also four interrupt signals that are modeled: three are ingoing and one is outgoing. Let us analyse them in the next table:

Signal	**Description**	**Type**
Switch to tracking	warns the camera component that must switch to tracking mode	ingoing
Switch to patrolling	warns the camera process that must set up the environment variables and goes in patroling mode	ingoing
Terminate	warns the camera component that must empty the variables that were used and quit	ingoing
Warn user process	warns user process that, after the *quit* activity, it is off	outgoing

Table 4.2: *Description of the signals of Figure 4.3*

We have just concluded the analysis of camera process. Now we are going to do a fast review of user's functions to control the camera process.

4.1.2 User process analysis

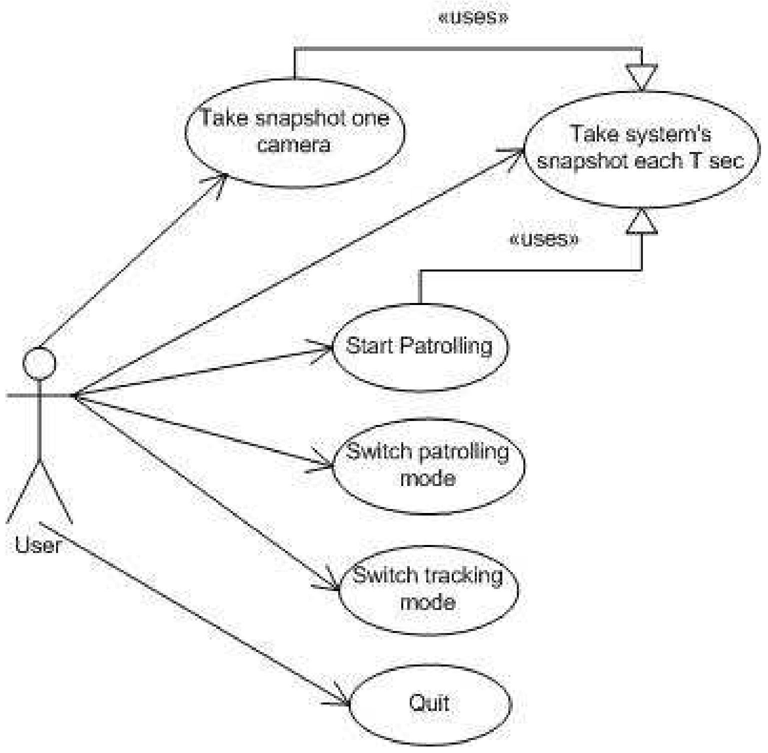


Figure 4.4: *UML Usecase diagram for User component.*

The diagram reported in Figure 4.4 shows what are the principal instruments of user for managing the camera through the communication between user and camera process. In the next table we clarify the usecase diagram describing the features for each state.

Usecase: Start Patrolling

Features	**Description**
Actors	User
Description	User starts the patrolling system
Preconditions	-
Main flow	user executes the program and start the patrolling system
Other flow	-
Postconditions	-

Usecase: Switch Tracking Mode

Features	**Description**
Actors	User
Description	User communicates to one camera to switch to tracking.
Preconditions	The patrolling system has to be started
Main flow	User chooses a camera C_i that he wants to controll and switch it to tracking mode. He can choose between auto-tracking and manual-tracking modalities
Other flow	If the choosen camera doesn't exist or it is already in tracking mode, it displaies a warning message
Postconditions	the choosen camera are suspend from patrolling activities

Usecase: Switch Patrolling Mode

Features	**Description**
Actors	User
Description	User communicates to one camera to switch to patrolling mode.
Preconditions	The patrolling system has to be started
Main flow	User choose a camera C_i that he wants to controll and switch it to patrolling mode.
Other flow	If the choosen camera doesn't exist or it is already in patrolling mode display a warning message
Postconditions	the choosen camera are suspend from his previous activities (eg: tracking activity)

Usecase: Take snapshot one camera

Features	**Description**
Actors	User
Description	User queries one camera to know its state.
Preconditions	The patrolling system has to be started
Main flow	User chooses a camera C_i and query it.
Other flow	If the choosen camera doesn't exists display a warning message
Postconditions	the choosen camera returns to user its state

Usecase: Take system's snapshot each T sec

Features	**Description**
Actors	User
Description	User launches this command and a the system start to query each camera every T sec, for each query each process respond with its state.
Preconditions	The patrolling system has to be started, if not this command start it.
Main flow	User launches this this command at the beginning
Other flow	User can start first the patrolling system than launches this command.
Postconditions	A continuously polling is made by the query component. It monitors processes evolution.

Usecase: Quit

Features	**Description**
Actors	User
Description	User closes the application and stops patrolling system.
Preconditions	The patrolling system has to be started
Main flow	User stops the overall system
Other flow	-
Postconditions	all patrolling process are killed

Table 4.3: *Description case by case of the usecase diagram*

We will see in the section 4.4.5 that this analysis is more important, in fact this step aided us and allows us to saved a lot of time, especially in the debug step of the system.

4.2 Technical consideration about software development

In previous sections we have described the UML diagram that describes the system requirement and the testbed architecture that we used. Submitted of our consideration and to the constraint that we have to use the C language, in order to use the serial library with the aim of moving the cameras, steams the following considerations. Starting to Figure 4.1 we have to implement a software in a centralized structure underline in the following figure.
To adjust our analysis to our testbed architecture we have to take into consid-

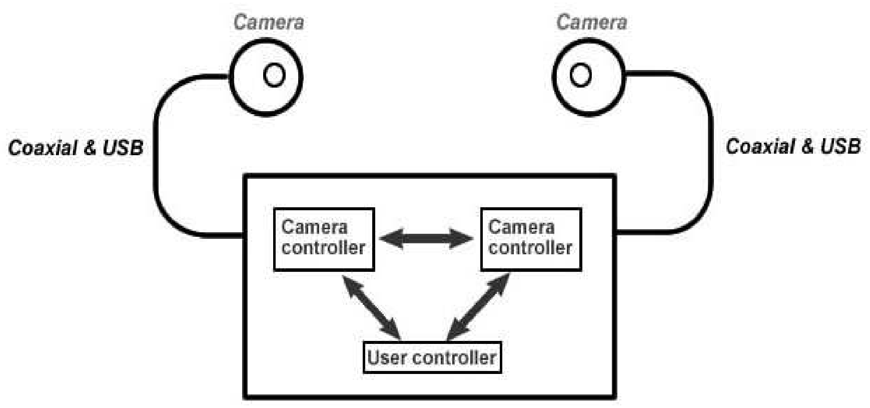


Figure 4.5: *Testbed architecture.*

eration the interface of communication that is possible to use with C language. There are two communication interfaces in order to obtain our scope. The first one is the communication interface that is used to communicate between two neighbouring cameras and it has to be a publish/subscribe protocol. The second one is the interface between the user and the camera component and it uses an interrupt protocol. For our project we chose the IPC libraries [16], in our particular case we used the libraries:

- *types.h and ipc.h*: They provide the definition of type variables and main functions.
- *signal.h*: it provides the interrupt communication signals

- *msg.h*:it provides the send/receive method to communicate throug component.

Finally, to complete our analogy, we treat all component as process that live in the same machine.

To explane more precisely the architetcure that we implemented, we present two component diagrams that clarify the structure of our process.

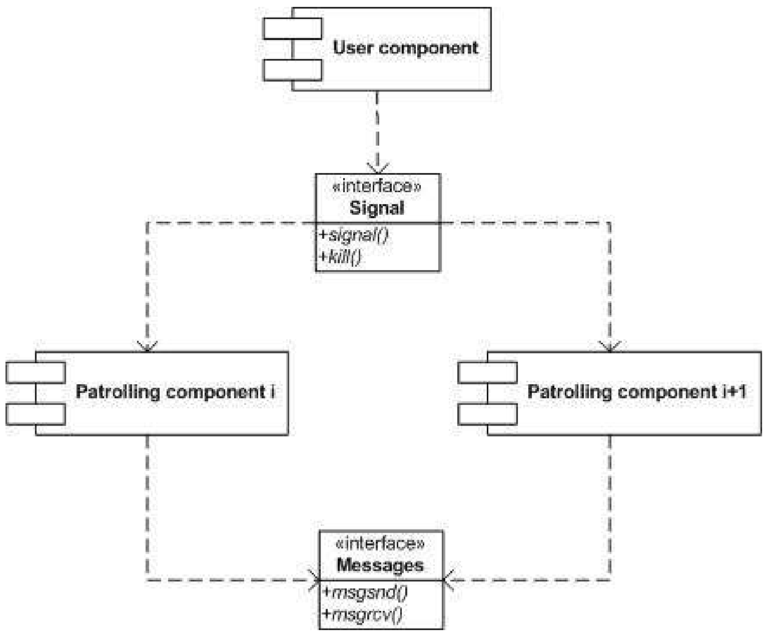


Figure 4.6: *UML Patrolling Component diagram of the communication interfaces.*

As we can see in Figure 4.6 we propose an UML component diagram that clarify the communication interface between processes. In particular there are:

- **signal interface**: it is the interface between user process and all the patrolling process. It provides two functions, such as:
  - *signal(INT,fnc)*: signal handler for interrupt *INT*. When an *INT* interrupt is lunched, the program catches the signal and executes the function *fnc*.
  - *kill(IDP,INT)*: it sends an interrupt signal of *INT* type to a process which has *IDP* id.
- **snd/rcv interface**: it is the interface between two processes. Each one of them controls only one camera. It provides two function, such as:
  - *msgsnd()*: after establish an one-directional communication channel between processes, the sender can sand a message through the channel using this function.
  - *msgrcv()*: similar to the previous function, but it provides, a receive function for the receiver process.

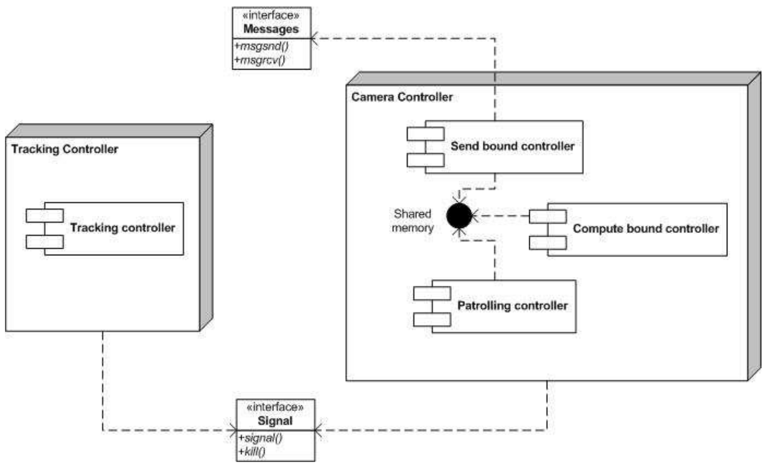


Figure 4.7: *UML Component diagram of the communication interfaces between Patrolling and Tracking component.*

The second component diagram (Figure 4.7) shows the structure of patrolling process, in particular it proposes a subdivision in components that cooperate each other. Looking at the analisys of activity diagram that we proposed in section 4.1.1 we can see four principal activities: *Send patrolling bound, Compute bound, Patrol* and *Track.* Analizing more deeply these activities we find three propriety that we have to take into account:

- *Patrol* activity only has to control the camera: when we control a camera, using the serial function provided by Videotec, we have to manage the signals using USB interface. In particular when we communicate a movement command to the camera we must wait a rotational time before forwarding another command. During this time our application must not idle, but it can do another available activity. For this reason and to keep separately the camera control from the other activity, we have to decide to implement a unique component that is dedicated only to camera managment. It is the *Patrolling controller* shown in Figure 4.7.

- *Send patrolling bound* and *Compute bound*: this two activities are indipendent from the other camera controller activities. They could be executed in concurrency to the Patrol activity and moreover they could be executed in parallel. For this reason we decided to implement two distinct component: The *Send bound controller* and *Compute bound controller* shown in section 4.4.3.

- *Track* activities are external to our scope: for this reason we implemented another process that is the tracking process that is launched when the patrolling process is suspended.

We have isolated the three components that compose the patrolling process. We decided to implement them by threads and they can communicate each other through a shared memory because they belong to the same process.

From these techical considerations stems the need of an analysis that spots the parameter that we must give as input of our process. In order to obtain this, we are going to introduce an input-output analysis of our process.

4.3 Input-Output analysis

We take into account the user process and we present the following table that describes its features in order to spot the scope, the ingoing and the outgoing parameters.

Scope:	Controls the cameras connected to the workstation and initializes the sequence of patrolling processes. Through *interrupt signals* queries a patrolling process P_i and controls the evolutions of the overall system.
In parameters:	-
Out parameters:	N processes where N is the number of camera connected via usb to the workstation.

Table 4.4: *Input-output analysis for user process*

From table 4.4 we can see how the user process creates one controller for each camera. This feature is created only for our testbed. In a future development will not be necessary to create processes, because they are launched, everyone of them, in its agent since beginning.

Very interesting is the analysis of the camera process that is reviewed in table 4.5.

Scope:	To move the camera C_i, given as input, along a patrolling path P delimited by $a_{i,l}$ and $a_{i,r}$ with a costant patrolling velocity v_i. At first instance $a_{i,l} = D_{i,inf}$ and $a_{i,r} = D_{i,sup}$. Through the comunication channels, the camera C_i comunicates its patrolling bounds ($a_{i,l}$ and $a_{i,r}$) to the camera C_{i+1} and C_{i-1}. C_{i+1} updates its patrolling bounds according to recieved parameters from C_i. N.B. we set the constant velocity along patrolling path as **always** feasible with the camera angular velocity constraints during the evolution of the algorithm.
In parameters:	• C_i: id of the camera connected via usb. • $a_{i,l} = D_{i,inf}$ and $a_{i,r} = D_{i,sup}$: at the first instance we set the patrolling physical constraints of the camera as input for $a_{i,l}$ and $a_{i,r}$. • v_i: constant velocity along the patrolling path. • $SNDch_{c_i \to c_{i-1}}$: communication channel between c_i and c_{i-1} cameras. Camera c_i send its bound to c_{i-1}. • $RCVch_{c_i \to c_{i-1}}$: communication channel between c_i and c_{i-1} cameras. Camera c_i receive the messages comming from c_{i-1}. • $SNDch_{c_i \to c_{i+1}}$: communication channel between c_i and c_{i+1} cameras. Camera c_i send its bound to c_{i+1}. • $RCVch_{c_i \to c_{i+1}}$: communication channel between c_i and c_{i+1} cameras. Camera c_i receive the messages comming from c_{i+1}. • $CLK1$: time between two consecutive transmission steps. • $CLK2$: time between two consecutive computing bound steps.
Out parameters:	• At every trasmission step, the proccess comunicate through message channels ($ch_{c_i \to c_{i-1}}$ and $ch_{c_i \to c_{i+1}}$) its new computed bound to processes (P_{i-1} and P_{i+1}). • P_{state_i}: process state. It could be [*patrolling*, *manual tracking*, *stop*, *ready*, *auto-tracking*]. It is returned when user process query patrolling process. • $a_{i,l}$ and $a_{i,r}$: it have to return it's patrolling bounds when it receives an interrupt by the user process.

Table 4.5: *Input-output analysis for camera process*

A very important specification, reported above, is the choose of velocity's parameter. This parameter has to be chosen offline. In fact it must be a feasible parameter during the entire evolution of our algorithm. Other notable parameters are the clocks (for the trasmission and the computing bound step) and the communication channels that are four for each camera due to the one-directional feature of our send/receive protocol.

4.4 Review of overall system architecture

In order to give to the lector a complete vision of the architecture, we are going to describe in detail each component of our system. In particular we will see the *Compute bound controller*, the *Transmission Bound controller* (for both controllers we will describe their way of communication), the *Camera controller* and the *User controller*. We underline that both controller (Trasmission and Compute bound controller) communicate via shared memory. In other words they save its sensible data (such as bounds variable) in the same memory location. We will see this feature in the *Camera controller* diagram and in the section 4.4.4.

4.4.1 Compute Bound controller

In order to describe the *Compute Bound controller* we take into account the diagram in Figure 4.8.

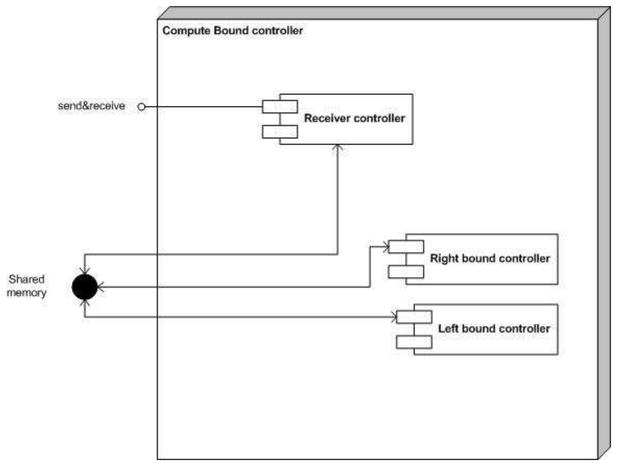


Figure 4.8: *UML Component diagram of the Compute Bound Controller.*

As we can see the main controller is composed by two sub-controller:

- **Receiver controller**: it controls the two ingoing channels for messages that are comming from its neighbouring cameras and receive the messages and saive it into the enviroment variable.
- **Left bound controller**: it computes the new left bound of the patrolling portion of path.
- **Right bound controller**: it computes the new right bound of the patrolling portion of path.

4.4.2 Transmission Bound controller

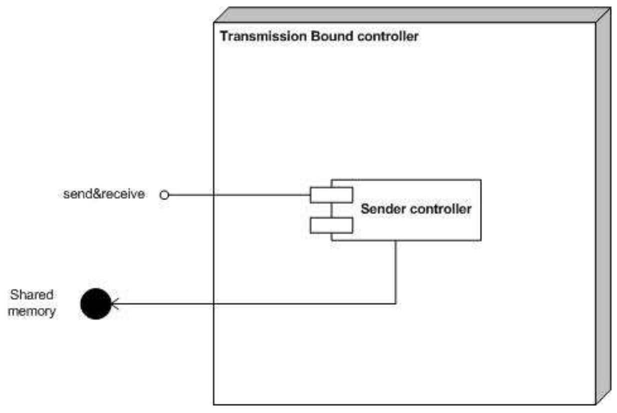


Figure 4.9: *UML Component diagram of the Trasmission Bound Controller.*

In light of the diagram in Figure 4.9 we have only one sub-controller:

- **Sender controller**: it sends to the neighbouring cameras its patrolling bounds using the send command improved by *msg.h* interface.

4.4.3 Patrolling controller

The target of this controller is simply to control the movement of one camera. Its features are displayed in Figure 4.10.

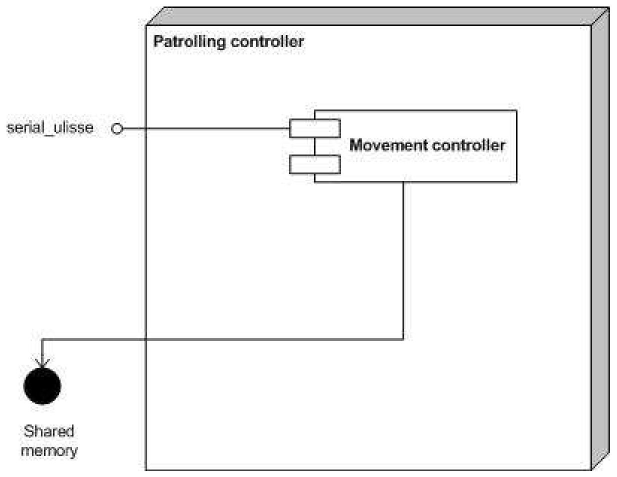


Figure 4.10: *UML Component diagram of the Patrolling Controller.*

The main sub-component (Movement controller) use the serial library to move its assigned camera. It moves its camera with a position command function and realizes the velocity control function proposed in section 3.6.5. In section 3.6.4 we have proposed a zoom control for our patrolling system. This improvement, unfortunately, cannot be implemented in our testbed because the available space isn't sufficient to make a review of our proposed changes.

4.4.4 Camera controller

As we mentioned, the *Camera controller* diagram Figure 4.7 has the target to clarify the communication interface among camera components. As we can see each component of the camera component communicates each other via shared memory. Let us see what are the main variables that camera controller has to save in the shared memory. In order to describe it we propose the following table:

As we can see in the table above, there is only one controller, that accesses in write mode, for each variable type. This implies that it is not neccessary to do some mutex variables.

Variables type	Compute Bound C.		Trasmission Bound C.		Patrolling C.	
	R	W	R	W	R	W
Bounds Variables		×	×		×	
Ingoing channels	×					
Outgoing channels				×		

Table 4.6: *Analysis of the type of access of the variables, in Camera controller*

4.4.5 User controller

Starting from the proposed diagram Figure 4.4 we can see the main functions that we need to implement. Now we propose the command that we implemented:

- ***start***: it starts the patrolling system
- ***snap [i]***: it queries the i-th patrolling process through a SIGHUP interrupt.
- ***sys_ snap [s]***: it creates a thread that poll each patrolling process using the function *snap*.
- ***switch [i]***: it changes the target of the i-th camera. If the i-th camera is doing patrolling, it switches the camera into tracking activity and vice versa. It warns the camera using SIGCHLD interrupt.
- ***quit***: it closes the system. It has to warn all the processes using SIGQUIT interrupt. The warned process has to execute the exit function and frees all the memories.

4.4.6 Tracking controller

The last one controller that we analize is the Tracking controller. It is outside our scope, but for completeness we implemented it. In particular we have to implement an instrument that is able to manage the user command (in particular *switch [i]*, *sys_snap [s]* and *quit*). In order to obtain this, we reuse the interrupt manage structure that we have implemented for patrolling controller and adjust it to the tracking controller requirements.

Chapter 5

Test results

In the previous chapter we have explaned the theoretical analysis for our problem. In particular we have spoken about the mathematical definition of the problem in chapter 2. Then we have proposed the key proposals for PTZ management in chapter 3 (in particular in section 3.6.3 we have analized the way to compute the pan and tilt angles to direct the camera to a given point). In chapter 4 we have suggested a suitable architecture for our patrolling software. After this complex analysis, in this chapter we are going to show what are the results gained using the proposed methods. We will see three main tests of the proposal that we have given:

- Test of the angles computation proposed in section 3.6.3.
- Test of the convergence of our perimetral patrolling software proposed in chapter 4.
- Test of the entire system in the Videotec testbed.

Now, Before explaining our tests we are going to introduce our testbed.

5.1 Testbed architecture

The Videotec testbed that we used is composed by two cameras of different type: we used an Ulisse Compact camera and an Ulisse standard camera. They have different characteristics that we have to take into account. In the next table we can see their features:

Camera type	Angular Velocity		Tilt		Offset		
	Max	Min	Max	Min	pan-tilt	tilt-hole	pan-hole
Ulisse Compact	120°/s	0.1°/s	90	-90	124mm	60mm	20mm
Ulisse standard	30°/s	0.1°/s	90	-40	na	na	na

Table 5.1: *Camera features table. For angular velocity we mean the velocity of PTZ camera in position control and the offset for stardard Ulisse are not aviable because they depend by camera installation*

Their position are shown in the following figure:

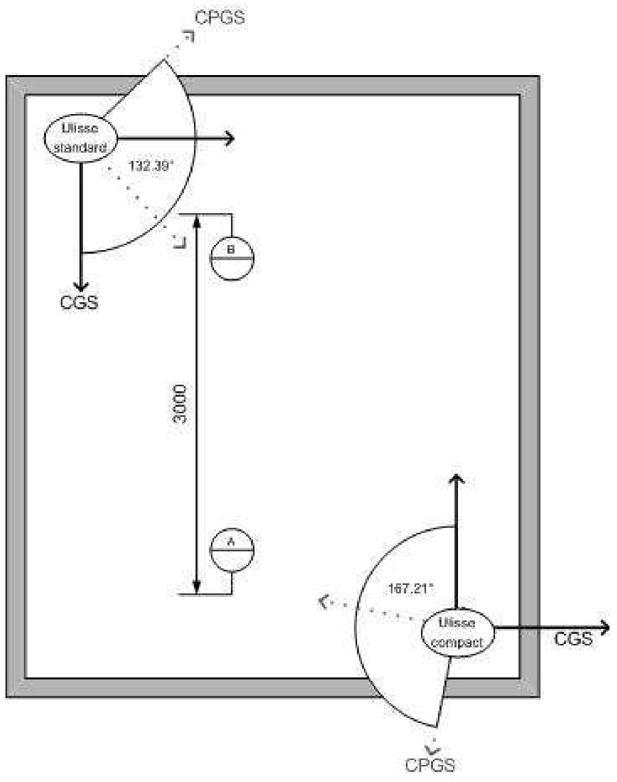


Figure 5.1: *Testbed pan. Layout of cameras and their reference systems*

As we can see the Testbet that we have at our disposal is limited and we must take the necessary measures for operating with it in order to realize a demo of our patrolling software. The cameras could be controlled through a serial interface (USB) and for the implementation of patrolling software we used a centralized control interface.

After this presentation of our testbed, we are going to explain the procedure that we have taken to compute the patrolling path and the angles associated to each point.

5.1.1 Computation of path and angles in testbed

Due to limitated available space and to the testbed's features exposed in the previous section we decided to do a semplification of our enviroment of work. In fact, as we show in Figure 5.1, we took, as the global patrolling path, a straight line of 3 meters from point A to point B. In order to take in consideration our mapping functions (described in section 3.8) in our testbed we do not mind the first two steps but we suppose to start from a 3-D definition of our trajectory poin by point and, after that we apply the step describes in section 3.1 and the mapping function that we have seen in section 3.6.

Now we introduce the steps that we made to setting up the enviroment of our testbed.

- **Setting up the ground system for each camera**: Every camera must have a ground system (CGS), against which we compute the coordinate of two starting point A and B. For each CGS we defined we have to gain the phase shift angle (ψ) between the position ground system of one camera (CPGS), due to its poistion in the space, and the CGS that we use for convenience.

- **Compute the coordinate of the points that describe our path**: For each CGS that we have defined we compute the coordinate of our points. This measures are affected by error because we used a meter for gaining them.

- **Compute a spline function to each CGS's point measured**: For each path definition against to one CGS, we computed a spline function to gain the complete definition of our patrolling path with respect to each camera.
- **Offline angle computation**: For each spline that we obtained, we computed the angles that corrispond to each point in the spline that we evaluated.

All these steps are implemented using Matlab. The output that this procedure yields is a vector where each cell is composed by three values: the pan and the tilt angles and the distance from the next point in the ground. Now we propose our tests starting to the computation of angles test.

5.2 Computation angles test

In section 3.6.3 we have described the process of angles computation; in particular we have seen that given a 2-D point $P = [x_w; y_w]$ through our method we gain Pan and $Tilt$. Our test was done in the Videotec testbed (section 5.1) and we used the method described in section 5.1.1. First of all we give the starting image of our test. As we can see in Figure 5.2 the red cross indicates

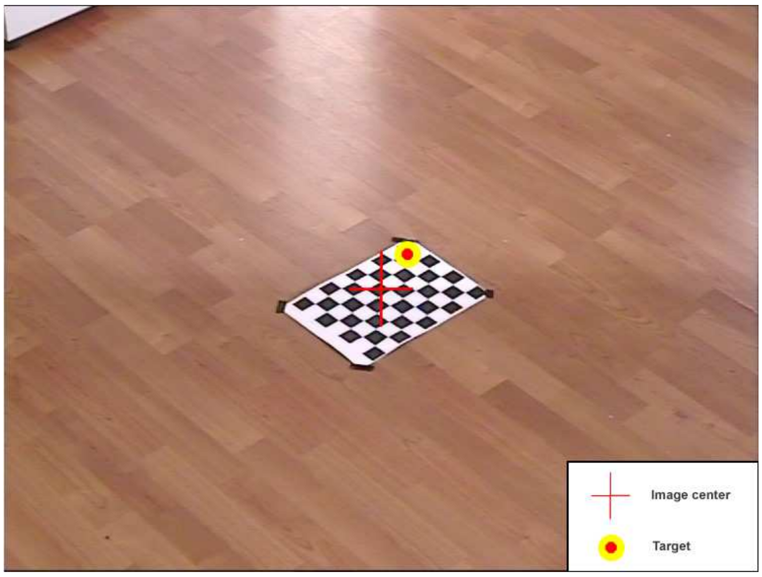


Figure 5.2: *Starting image of our angles test.*

the image center and the red-yellow circle shows the target point that we have to reach. This image is taken with the Ulisse Compact camera, it has:

$$Pan : 119°$$

$$Tilt : 39°$$

$$Zoom : 2.36\times$$

After applying our angle computation function we gain the new angle that bring our camera to point the red-yellow circle shown before. We gain:

$$Pan : 122°$$

$$Tilt : 37°$$

The image recovered after an absolute movement of the computed angles is:

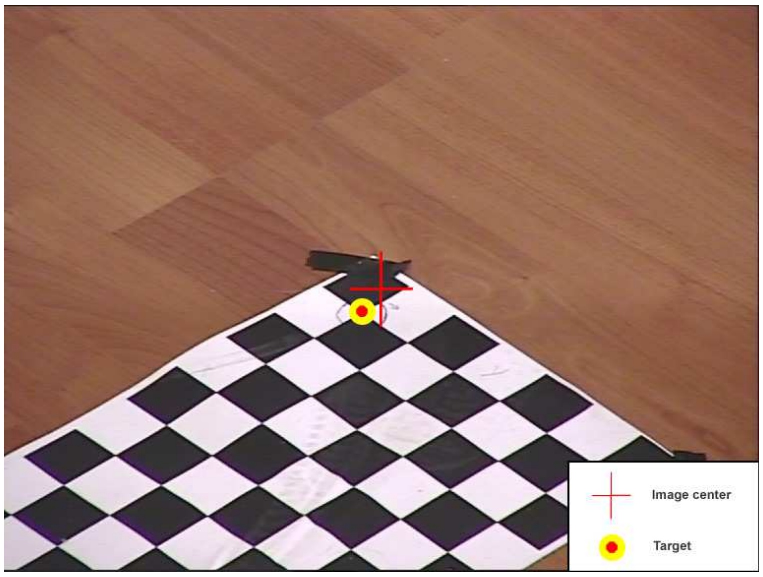


Figure 5.3: *Centered image of our angles test.*

In this image comes evident that the calculated angles do not bring the camera to point the target indicated. We underline that the Figure 5.3 is taken with a different level of zoom with respect to the starting image (Figure 5.2), its zoom is 10×. Now we analize the possible errors (see Figure 5.3)

- **Error due to measure of the target "by hand"**: As we said in section 5.1, due to the limit of our testbed we have to measure by hand the ground coordinate of the target point. This modus operandi introduce a measurement error that drugs our starting data. To give an idea of the error that we introduced with the Bouguet's toolbox, in 3-D space a point has the sensitivity of the millimeter, while when we measure a 3-D point in the space we have to consider a large space (more or less 3 square meters as shown in Figure 5.1).

- **Error due to measure of rotational angle of the camera**: As we show in Figure 5.1 we measured the angle between the CPGS and the

CGS that we chose. This measure is taken "by hand" and this procedure introduce a possible error.

These are the possible sources of error in our angles computation. Moreover, working in a limited environment, the computational errors are more evident; in fact in order to be able to work in this environment, we have to use an high level of zoom and it underlines the possible mismatch between computed angle and the target point.
If we work in outdoor environments these error are less evident. It is also possible to reduce the errors using the corrispondence between focal lenght and pixel dimension in order to find the relation between one pixel point and the corrispondent point in the CGS, but this improvement is not treated in this thesis.

Now we are going to prove the convergence of our implemented software with a simulation.

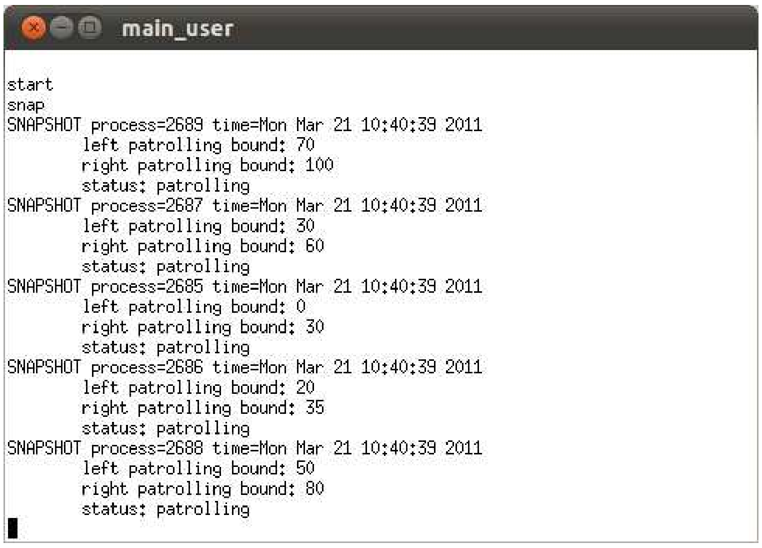
5.3 Convergence test

For this test we simulate a patrolling system composed by **five cameras** with these specifics: The patrolled perimeter can be assumed as a line that is defined

Camera	Left bound	Right bound	Trasmission B. Clock	Compute B. Clock
1	0	30	4 sec	4 sec
2	20	35	4 sec	4 sec
3	30	60	4 sec	4 sec
4	50	80	4 sec	4 sec
5	70	100	4 sec	4 sec

Table 5.2: *Specification of cameras for convergence test*

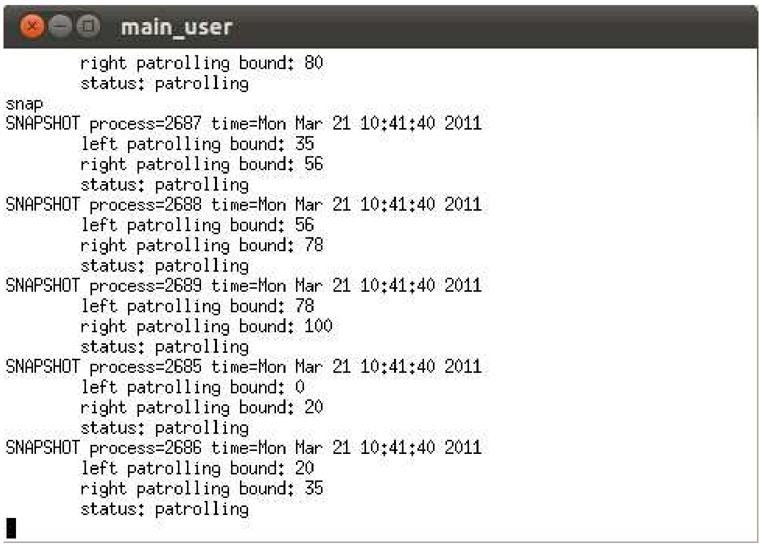
as a vector $[0; 100]$ and all the cameras have the same velocity.



```
start
snap
SNAPSHOT process=2689 time=Mon Mar 21 10:40:39 2011
        left patrolling bound: 70
        right patrolling bound: 100
        status: patrolling
SNAPSHOT process=2687 time=Mon Mar 21 10:40:39 2011
        left patrolling bound: 30
        right patrolling bound: 60
        status: patrolling
SNAPSHOT process=2685 time=Mon Mar 21 10:40:39 2011
        left patrolling bound: 0
        right patrolling bound: 30
        status: patrolling
SNAPSHOT process=2686 time=Mon Mar 21 10:40:39 2011
        left patrolling bound: 20
        right patrolling bound: 35
        status: patrolling
SNAPSHOT process=2688 time=Mon Mar 21 10:40:39 2011
        left patrolling bound: 50
        right patrolling bound: 80
        status: patrolling
```

Figure 5.4: *Image test – Start of our patrolling software.*

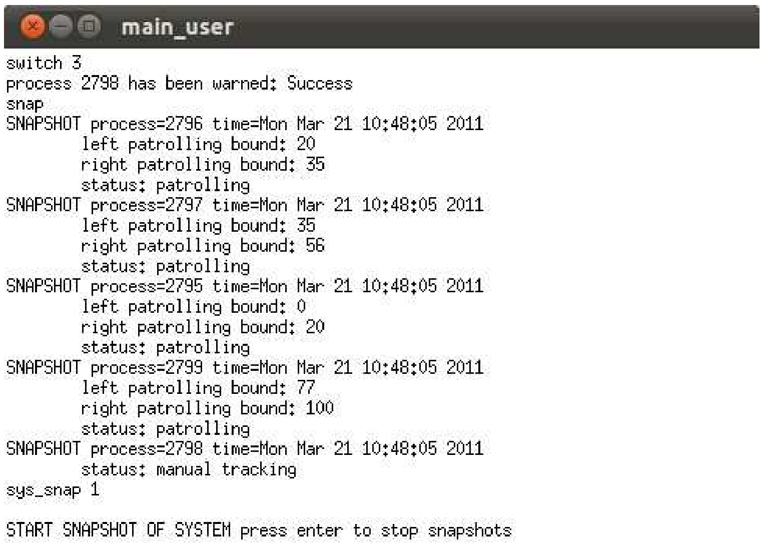
As we can see in the Figure 5.4 we start the system and we gain the configuration described in the above table.
After a few seconds, the system reaches the convergence as shown in Figure 5.5.



```
main_user
        right patrolling bound: 80
        status: patrolling
snap
SNAPSHOT process=2687 time=Mon Mar 21 10:41:40 2011
        left patrolling bound: 35
        right patrolling bound: 56
        status: patrolling
SNAPSHOT process=2688 time=Mon Mar 21 10:41:40 2011
        left patrolling bound: 56
        right patrolling bound: 78
        status: patrolling
SNAPSHOT process=2689 time=Mon Mar 21 10:41:40 2011
        left patrolling bound: 78
        right patrolling bound: 100
        status: patrolling
SNAPSHOT process=2685 time=Mon Mar 21 10:41:40 2011
        left patrolling bound: 0
        right patrolling bound: 20
        status: patrolling
SNAPSHOT process=2686 time=Mon Mar 21 10:41:40 2011
        left patrolling bound: 20
        right patrolling bound: 35
        status: patrolling
▮
```

Figure 5.5: *Image test – Convergence reached.*

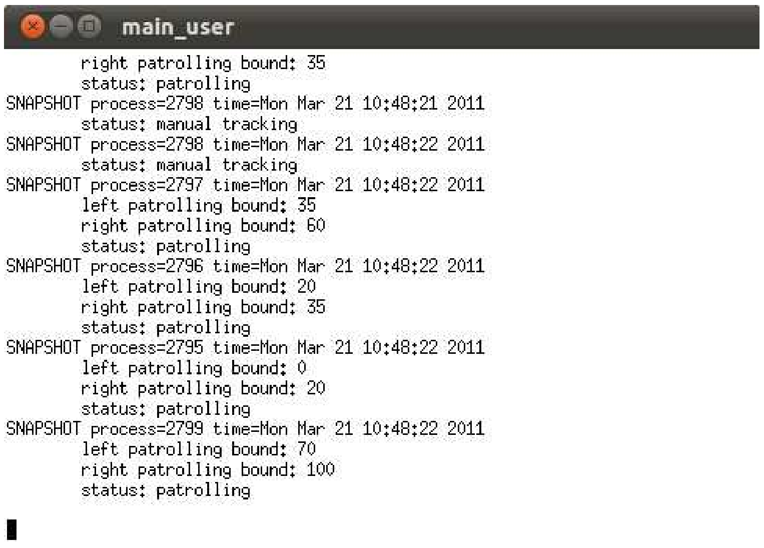
In this image we can see which is the optimal subdivision of patrolling spaces; in particular we observe that the second camera (process 2686) responds to its physical bounds. In fact, if we suppose that all the cameras do not have any physical bounds, each camera should patrol a portion of path that is long 20.
Now we show what happen when we unplug (e.g. switch to patrolling) one camera. With the command *switch* we warn the fourth camera as we report in Figure 5.6.



```
switch 3
process 2798 has been warned: Success
snap
SNAPSHOT process=2796 time=Mon Mar 21 10:48:05 2011
        left patrolling bound: 20
        right patrolling bound: 35
        status: patrolling
SNAPSHOT process=2797 time=Mon Mar 21 10:48:05 2011
        left patrolling bound: 35
        right patrolling bound: 56
        status: patrolling
SNAPSHOT process=2795 time=Mon Mar 21 10:48:05 2011
        left patrolling bound: 0
        right patrolling bound: 20
        status: patrolling
SNAPSHOT process=2799 time=Mon Mar 21 10:48:05 2011
        left patrolling bound: 77
        right patrolling bound: 100
        status: patrolling
SNAPSHOT process=2798 time=Mon Mar 21 10:48:05 2011
        status: manual tracking
sys_snap 1

START SNAPSHOT OF SYSTEM press enter to stop snapshots
```

Figure 5.6: *Image test – Switching the fourth camera.*

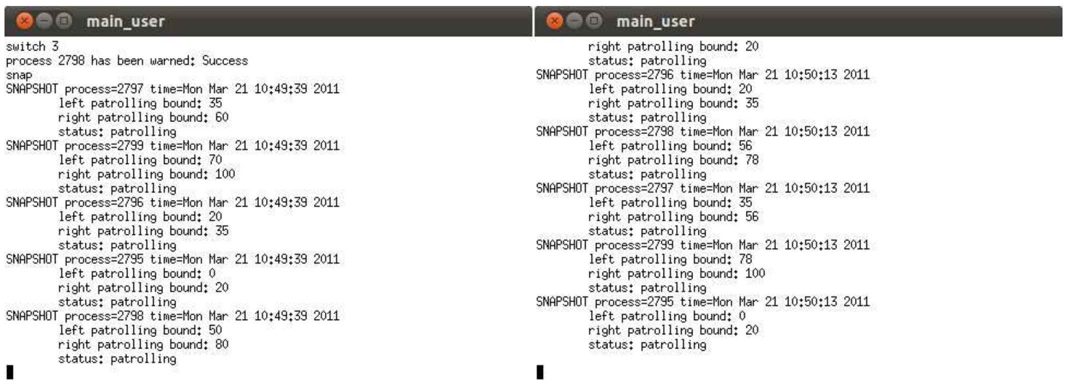
After a few seconds, the adjacent cameras become aware of the absence of the fourth camera and they resize their bounds in order to cover the area of the deficient camera, as we can see in Figure 5.7.



```
        right patrolling bound: 35
        status: patrolling
SNAPSHOT process=2798 time=Mon Mar 21 10:48:21 2011
        status: manual tracking
SNAPSHOT process=2798 time=Mon Mar 21 10:48:22 2011
        status: manual tracking
SNAPSHOT process=2797 time=Mon Mar 21 10:48:22 2011
        left patrolling bound: 35
        right patrolling bound: 60
        status: patrolling
SNAPSHOT process=2796 time=Mon Mar 21 10:48:22 2011
        left patrolling bound: 20
        right patrolling bound: 35
        status: patrolling
SNAPSHOT process=2795 time=Mon Mar 21 10:48:22 2011
        left patrolling bound: 0
        right patrolling bound: 20
        status: patrolling
SNAPSHOT process=2799 time=Mon Mar 21 10:48:22 2011
        left patrolling bound: 70
        right patrolling bound: 100
        status: patrolling

▮
```

Figure 5.7: *Image test – Convergence reached even the fourth camera is unplugged.*

We reconnect the camera and the system come back to the optimal coverage state (Figure 5.8).

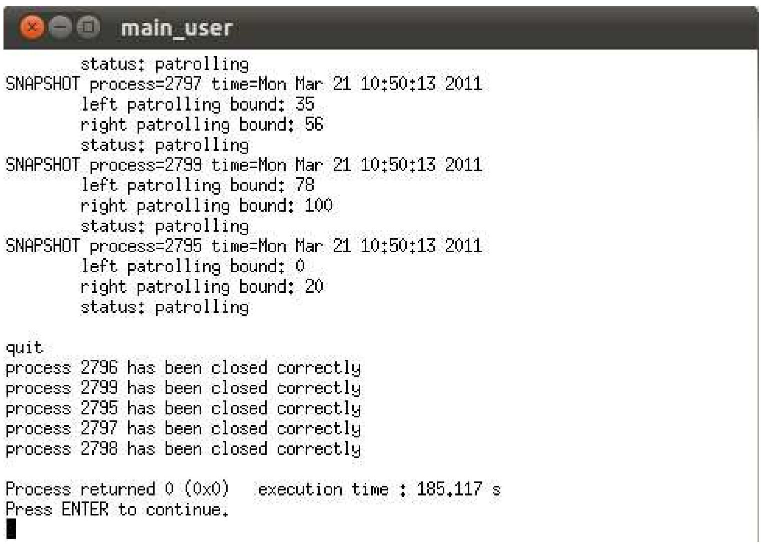


```
switch 3
process 2798 has been warned: Success
snap
SNAPSHOT process=2797 time=Mon Mar 21 10:49:39 2011
        left patrolling bound: 35
        right patrolling bound: 60
        status: patrolling
SNAPSHOT process=2799 time=Mon Mar 21 10:49:39 2011
        left patrolling bound: 70
        right patrolling bound: 100
        status: patrolling
SNAPSHOT process=2796 time=Mon Mar 21 10:49:39 2011
        left patrolling bound: 20
        right patrolling bound: 35
        status: patrolling
SNAPSHOT process=2795 time=Mon Mar 21 10:49:39 2011
        left patrolling bound: 0
        right patrolling bound: 20
        status: patrolling
SNAPSHOT process=2798 time=Mon Mar 21 10:49:39 2011
        left patrolling bound: 50
        right patrolling bound: 80
        status: patrolling
```

```
        right patrolling bound: 20
        status: patrolling
SNAPSHOT process=2796 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 20
        right patrolling bound: 35
        status: patrolling
SNAPSHOT process=2798 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 56
        right patrolling bound: 78
        status: patrolling
SNAPSHOT process=2797 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 35
        right patrolling bound: 56
        status: patrolling
SNAPSHOT process=2799 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 78
        right patrolling bound: 100
        status: patrolling
SNAPSHOT process=2795 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 0
        right patrolling bound: 20
        status: patrolling
```

Figure 5.8: *Image test – Come back to patrolling of the fourth camera and the reaching of steady optimal convergence state.*

Finally we close our system as we can see in the last figure (Figure 5.9).



```
        status: patrolling
SNAPSHOT process=2797 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 35
        right patrolling bound: 56
        status: patrolling
SNAPSHOT process=2799 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 78
        right patrolling bound: 100
        status: patrolling
SNAPSHOT process=2795 time=Mon Mar 21 10:50:13 2011
        left patrolling bound: 0
        right patrolling bound: 20
        status: patrolling

quit
process 2796 has been closed correctly
process 2799 has been closed correctly
process 2795 has been closed correctly
process 2797 has been closed correctly
process 2798 has been closed correctly

Process returned 0 (0x0)   execution time : 185.117 s
Press ENTER to continue.
```

Figure 5.9: *Image test – Closing of our patrolling system.*

5.4 Testbed test

In this section we analize the results of our test in the Videotec's testbed. As we mentioned in section 5.1, we have two cameras of different type and our test consists in patrolling a line of three meters, as we can see in Figure 5.1. We initialize the system with these parameters shown in the next table:

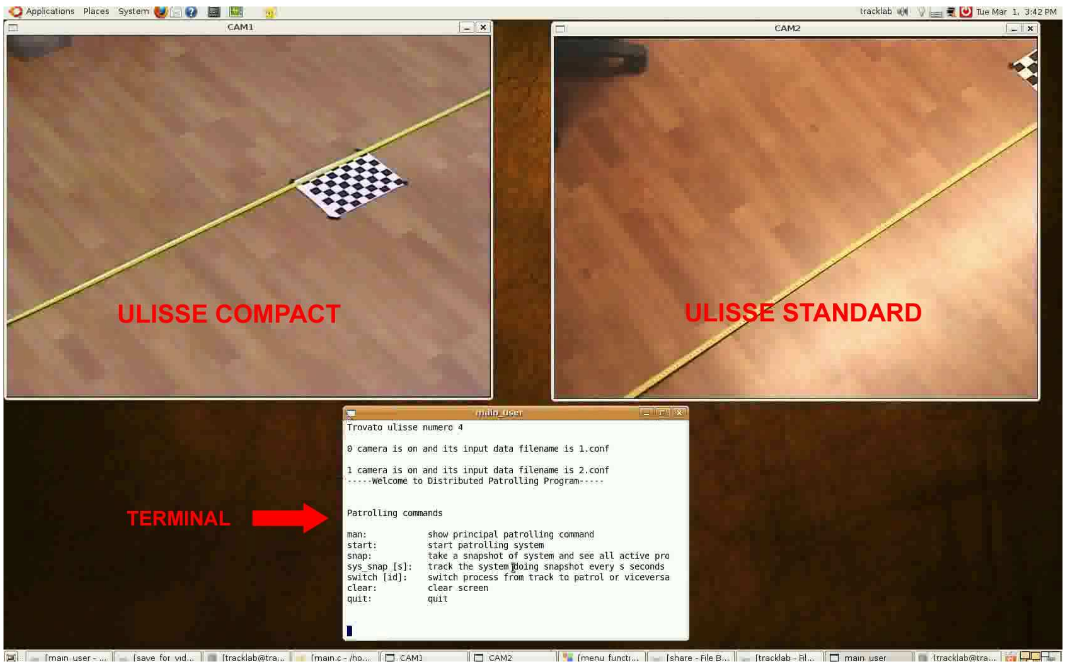
Camera	Left bound	Right bound	Trasmission Clock	Compute Clock	Velocity
1	0	15	5 sec	5 sec	5 m/s
2	0	15	5 sec	5 sec	5 m/s

Table 5.3: *Specification of cameras for the test on testbed*

Our test consists of four steps:

1. **Start the patrolling system**: we launch our system with the command *sys_snap 1* and we gain a continuously monitoring of the process until the system reaches the convergence state.
2. **Switch the camera from patrolling to tracking activity**: through the command *switch* we close the patrolling activity and we launch the tracking for the Ulisse standard camera.
3. **Wait the camera to detect what is happened and observe its behaviour**: we wait few seconds before the Ulisse Compact camera discovers that its neighbor camera are in tracking mode and we observe that it comes back to its original bounds.
4. **We reinsert the Ulisse standard camera** we plug this camera to the patrolling system and we wait the convergence state.

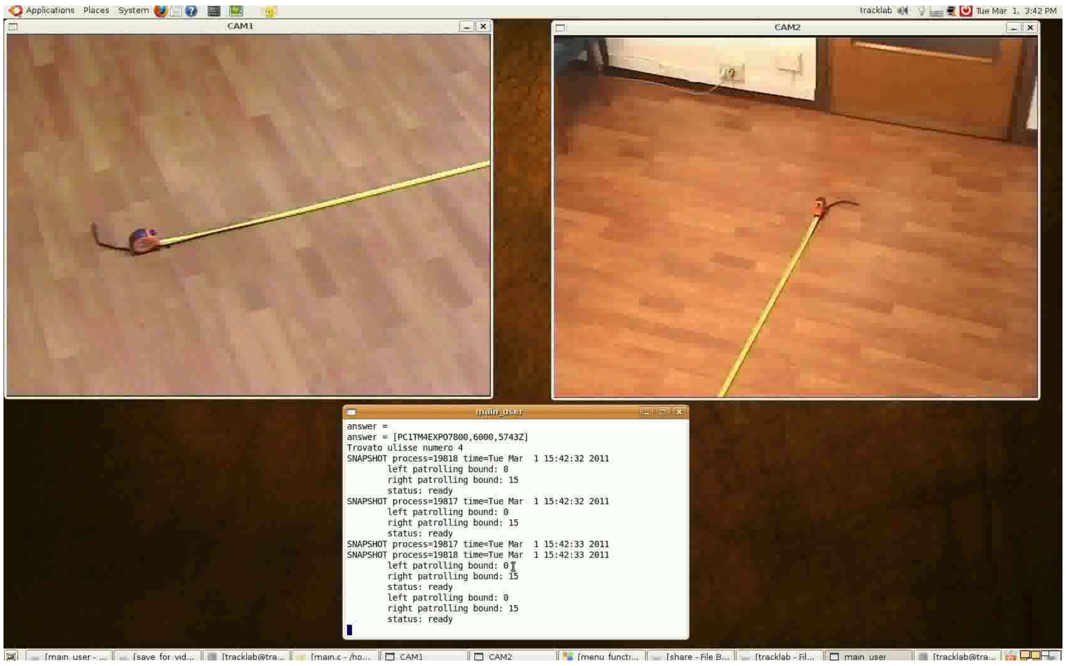
Now we are going to show our results:
The images that we have taken are composed by three principal elements shown in Figure 5.10.



```
Trovato ulisse numero 4

0 camera is on and its input data filename is 1.conf

1 camera is on and its input data filename is 2.conf
-----Welcome to Distributed Patrolling Program-----


Patrolling commands

man:             show principal patrolling command
start:           start patrolling system
snap:            take a snapshot of system and see all active pro
sys_snap [s]:    track the system doing snapshot every s seconds
switch [id]:     switch process from track to patrol or viceversa
clear:           clear screen
quit:            quit
```

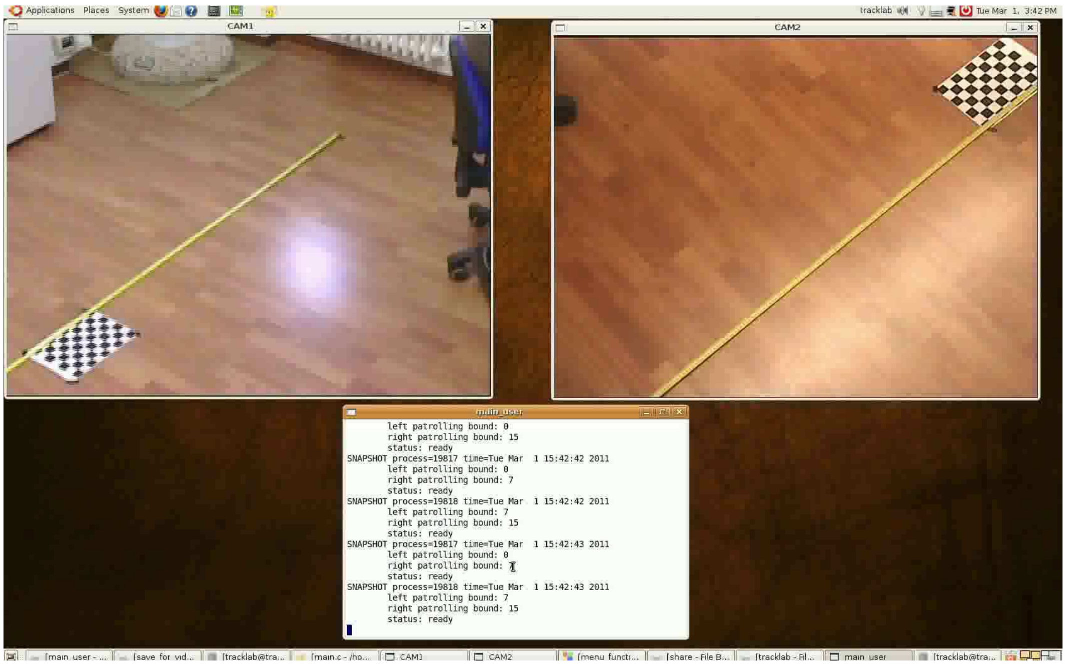
Figure 5.10: *Testbed's test – figures schema.*

The Figure 5.11 and Figure 5.12 show the point one of our test. As we can see the two cameras start from the same location and they do their activity. In the terminal of Figure 5.12 we can see the convergence state of our system.



```
answer =
answer = [PC1TM4EXPO7800,6000,5743Z]
Trovato ulisse numero 4
SNAPSHOT process=19818 time=Tue Mar  1 15:42:32 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19817 time=Tue Mar  1 15:42:32 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19817 time=Tue Mar  1 15:42:33 2011
SNAPSHOT process=19818 time=Tue Mar  1 15:42:33 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
```

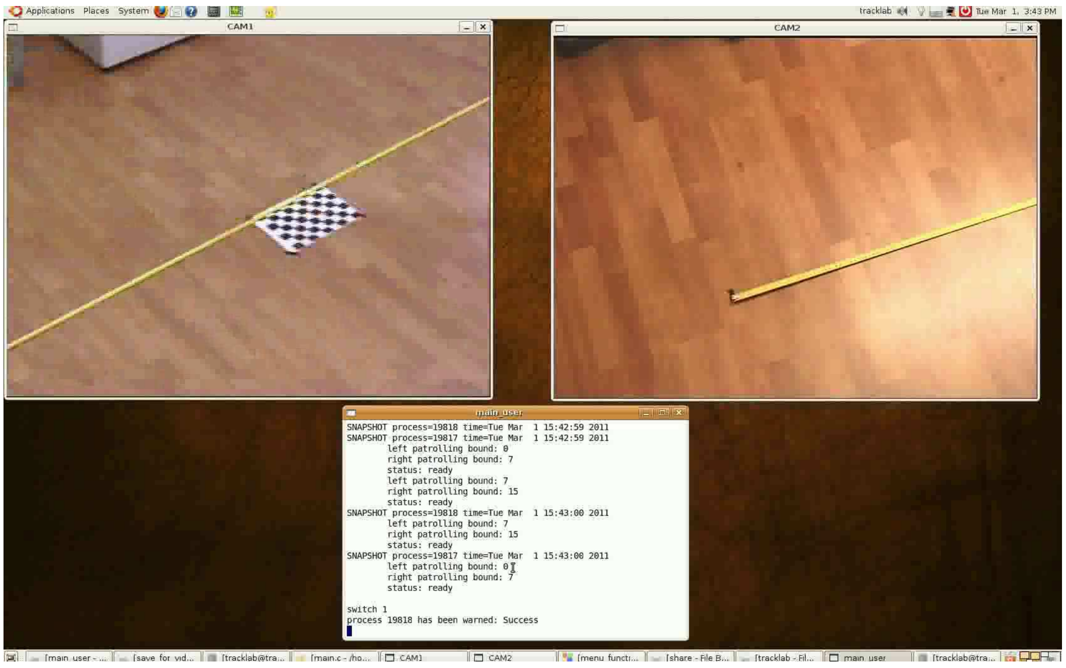
Figure 5.11: *Testbed's test – start of the system.*



```
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19817 time=Tue Mar  1 15:42:42 2011
        left patrolling bound: 0
        right patrolling bound: 7
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:42:42 2011
        left patrolling bound: 7
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19817 time=Tue Mar  1 15:42:43 2011
        left patrolling bound: 0
        right patrolling bound: 7
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:42:43 2011
        left patrolling bound: 7
        right patrolling bound: 15
        status: ready
```

Figure 5.12: *Testbed's test – state of convergence reached.*

In Figure 5.13 we rappresent the point two of the test and in Figure 5.14 we show how only one camera moves.

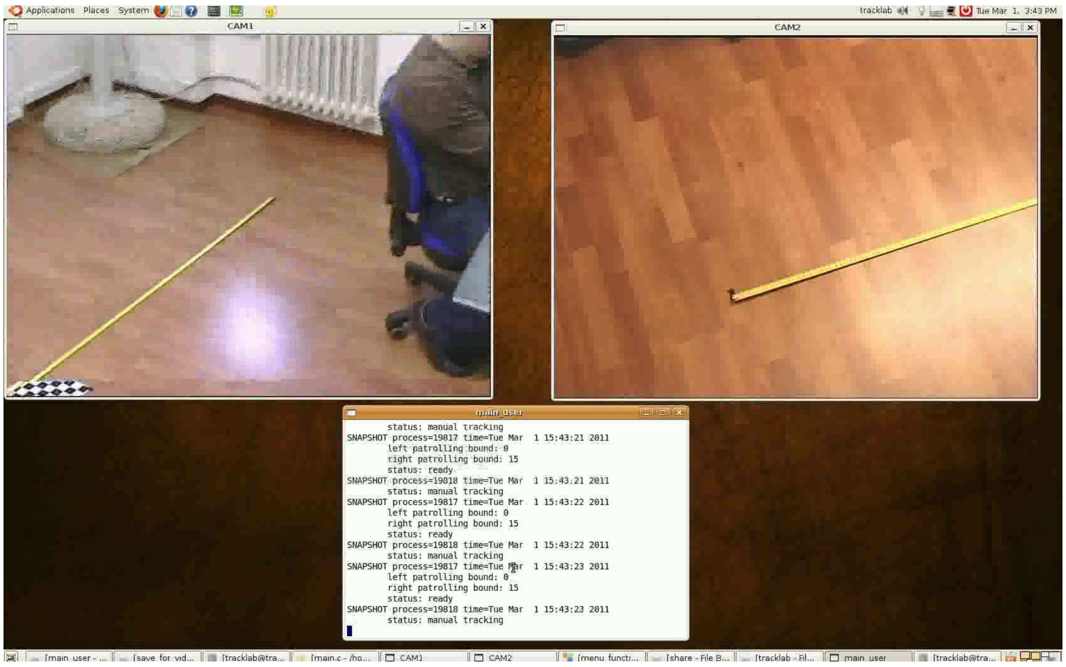

Applications Places System
tracklab Tue Mar 1, 3:43 PM
CAM1
CAM2
main_user
[main_user - ... [save for vid... [tracklab@tra... [main.c - /ho... CAM1 CAM2 [menu functi... [share - File B... [tracklab - Fil... main_user [tracklab@tra...

```
SNAPSHOT process=19818 time=Tue Mar  1 15:42:59 2011
SNAPSHOT process=19817 time=Tue Mar  1 15:42:59 2011
        left patrolling bound: 0
        right patrolling bound: 7
        status: ready
        left patrolling bound: 7
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:43:00 2011
        left patrolling bound: 7
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19817 time=Tue Mar  1 15:43:00 2011
        left patrolling bound: 0
        right patrolling bound: 7
        status: ready

switch 1
process 19818 has been warned: Success
```

Figure 5.13: *Testbed's test – we warn Ulisse Standard camera.*

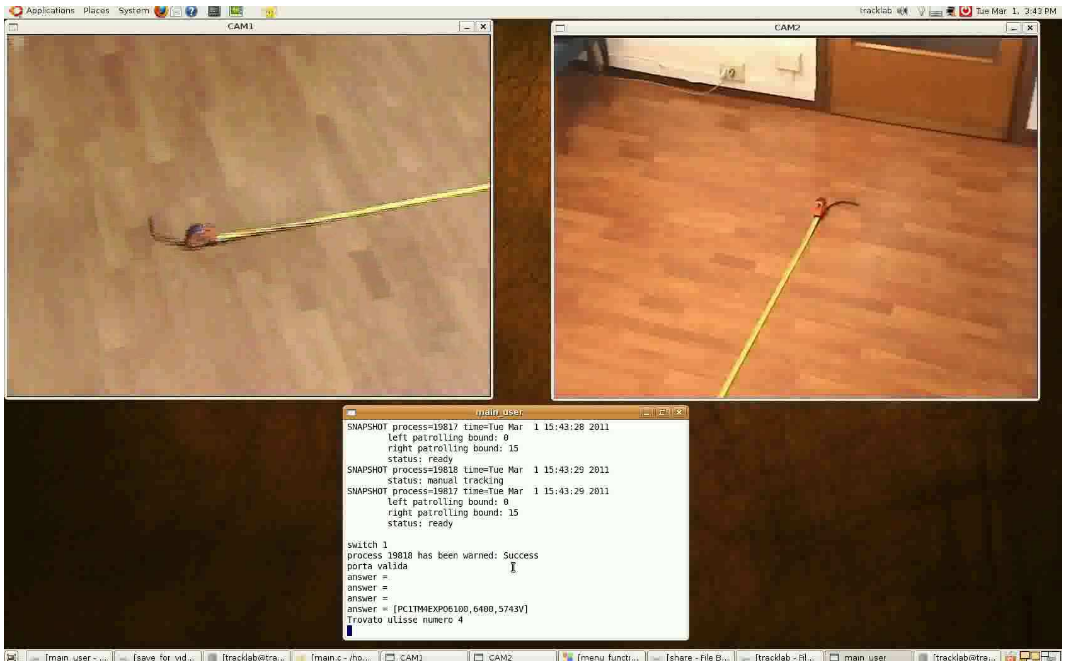
Finally, in the last picture (Figure 5.15), we replug the camera to the system.

As we can notice the images are not prefectly centered to the target line: only one camera (Ulisse Compact) is closed to the path. It is caused by the error introduced into the measures (as we have shown in 5.2).
We underline that the offsets are available only for the Ulisse Compact camera thus the angles computation for the Ulisse standard are affected by offset error.



```
        status: manual tracking
SNAPSHOT process=19817 time=Tue Mar  1 15:43:21 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:43:21 2011
        status: manual tracking
SNAPSHOT process=19817 time=Tue Mar  1 15:43:22 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:43:22 2011
        status: manual tracking
SNAPSHOT process=19817 time=Tue Mar  1 15:43:23 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:43:23 2011
        status: manual tracking
```

Figure 5.14: *Testbed's test – only Ulisse Compact camera moves itself.*



```
SNAPSHOT process=19817 time=Tue Mar  1 15:43:28 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready
SNAPSHOT process=19818 time=Tue Mar  1 15:43:29 2011
        status: manual tracking
SNAPSHOT process=19817 time=Tue Mar  1 15:43:29 2011
        left patrolling bound: 0
        right patrolling bound: 15
        status: ready

switch 1
process 19818 has been warned: Success
porta valida
answer =
answer =
answer =
answer = [PC1TM4EXPO6100,6400,5743V]
Trovato ulisse numero 4
```

Figure 5.15: *Testbed's test – reconnection of the Ulisse Standard camera.*

Chapter 6

Conclusions and future developments

Along this thesis we have treated all the features of perimetral patrolling problem with PTZ cameras, in particular the extension of the problem proposed in [9]. The changes we have proposed are one first solution for our problem; in fact in order to obtain a complete video surveillance system it is necessary to analyze other improvements that we will give in this chapter. Let us analyze our proposed improvements and their goodness:

- **Extension of the proposed algorithm. From 1-D line to PTZ definition**: we have treated this improvement in chapter 3. In the section 3.8 we have explained a diagram block in order to manage the entire system, from 2-D image plane point to PTZ parameters that describes anyone 3-D trajectory. It is evident that the calibration procedure depends from the calibration tool that we use. In a future context, we suggest to adopt another calibration toolbox to gain a more precisely step of calibration and for creating a multi-camera calibration step that we described in section 3.5.
  In our thesis we have proposed a possible controller for the zoom and the velocity. It is evident that they must be tested in a more complex testbed in order to measure their effective state and improve them. In particular, in a future development, Videotec propose some tests in its outdoor testbed in Schio. From these tests it could be possible to discover some improvements, for example we can implement and analyze the zoom control proposed or try to improve the velocity control with

the one provided by Videotec's cameras.

- **Software architecture**: in the chapter 4 we have proposed our architecture for patrolling software. In particular we made a distributed analysis but we implemented this software in a centralized structure. In future, we could change the interface of communication in order to adapt our software to the Videotec's agent that is called Albert.

From this analysis stems the need to obtain more results in future tests with different configurations of the testbed.

There are also some improvements that we can discover analyzing the problem in its completeness. As we can see in section 4.3 the choice of velocity was made offline. In fact we have to test different velocities along different path's segments and we have to take the minimum of these. In a future development we will try to formulate a new distributed algorithm that adjusts the velocity during the evolution of the algorithm.
Another important feature that we have to take into account is the development of a mechanism that chooses for each camera the correct activity to be assigned. This problem is called *Task assignment problem.* A solution to this problem is given in [13] by the Automated Control Group of the Padova's University. Starting from our work it is possible to extend and to implement new tools that cover also these requests.
In order to realize an extension of our algorithm, it is possible to change the target of our problem from patrolling along a perimeter to patrolling in an area. We could think about an extension of this algorithm such as a preprocessing step that consists in finding a path that, defined a zoom function, covers areas instead of a simple 1-D line. In fact, in our analysis, we have taken the f.o.v of one camera as a point, but in a real situation, it is an area that depends from the level of zoom that we adopt.
In conclusion, our work is one of the first steps to gain a complete distributed video surveillance system. We handled the first problems that arise adapting the theoretical problem to the PTZ cameras. We are sure that our progress could be helpful for the future problems, in particular when we are in presence of integration problems with PTZ cameras.

Chapter 7

Acknowledgements

- Ringrazio la mia famiglia che, nonostante tutte le difficoltà, mi ha sempre sostenuto. Ringrazio anche i parenti tutti, in particolare coloro che non ci sono più: sono sempre nei miei pensieri.

- I miei amici di infanzia, con loro ne ho passate tante e di tutti i tipi. Grazie di cuore.

- I miei colleghi di università per i continui scambi di opinione e per questi lunghi 5 anni passati insieme. Sopratutto per i momenti di grande impegno, i lavori di gruppo, le pause tra una lezione e l'altra e le pause pranzo in cui si iniziava a parlare del piu e del meno e si finiva a parlare dei grandi problemi che affliggono il mondo. I soliti discorsi noiosi da ingegneri.

- Ringrazio Piero Donaggio per tutto ciò che mi ha insegnato in questi sei mesi di tirocinio.

- Ringrazio il professore Luca Schenato che si è dimostrato, in ogni momento, disponibile e cordiale. Sono convinto che se tutti i professori fossero come lui i problemi dell'Università italiana si dimezzerebbero.

- Ed infine, ringrazio te. Tu che in questi anni sei sempre stata al mio fianco in ogni momento anche in quelli più bui, e sai che ce ne sono stati tanti. Questa laurea è anche un pò tua, se sono arrivato a questo importante traguardo è grazie a te. Da oggi 5 Aprile 2011 inizia una nuova avventura che spero mi porti a costruire un futuro assieme a te. Ti amo.

Bibliography

[1] F.R. Abate, Ed., *The Oxford Dictionary and Thesaurus: The Ultimate Language Reference for American Readers*, Oxford University Press, 1996.

[2] I. Hussein and D. Stipanovic, *Effective coverage control using dynamic sensor networks*, dec. 2006, pp. 2747 - 2752.

[3] I. Hussein and D. Stipanovic, *Effective coverage control using dynamic sensor networks with flocking and guaranteed collision avoidance*, jul. 2007, pp. 3420 - 3425.

[4] D. Kingston, R. Beard, and R. Holt, *Decentralized perimeter surveillance using a team of uavs*, Robotics, IEEE Transactions on, vol. 24, no. 6, pp. 1394 - 1404, dec. 2008.

[5] Y. Chevaleyre, *Theoretical analysis of the multi-agent patrolling problem*, sep. 2004, pp. 302 - 308.

[6] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and C. Y., *Recent advances on multi-agent patrolling*, Lecture Notes in Computer Science, vol. 3171, p. 474?483, 2004.

[7] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo, *Equitable partitioning policies for mobile robotic networks*, IEEE Transactions on Automatic Control, Provisionally Accepted, 2008.

[8] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo, *Equitable partitioning policies for robotic networks*, in ICRA'09: Proceedings of the 2009 IEEE

international conference on Robotics and Automation. Piscataway, NJ, USA: IEEE Press, 2009, pp. 3979 - 3984.

[9] R. Carli, A. Cenedese, L. Schenato, *Distributed Partitioning Strategies for Perimeter patrolling*, Proceedings of the ACC'11, 2011.

[10] J. Heikkila, O. Silven, *A Four-Step Camera Calibration Procedure with Implicit Image Correction*, In Proc. of IEEE Computer Vision and Pattern Recognition, pp. 1106-1112, 1997.

[11] P. Desai, K.S. Rattan, *Indoor Localization and Surveillance using Wireless Sensor Network and Pan/Tilt Camera*, Proceedings of the IEEE 2009 National, 2010.

[12] D. M. Raimondo, S. Gasparella, D. Sturzenegger, J. Lygeros, M. Morari, *A tracking algorithm for PTZ cameras*, 2010.

[13] A. Cenedese, F. Cerruti, M. Fabbro, C. Masiero, L. Schenato, *Decentralized task assignment in camera networks*, Conference on Decision and Control CDC10, 2010

Sitography

[14] http://en.wikipedia.org/wiki/Spline_(mathematics)

[15] http://www.vision.caltech.edu/bouguetj/calib_doc/

[16] http://en.wikipedia.org/wiki/Inter-process_communication