Politecnico di Milano

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Specialistica in Ingegneria delle Telecomunicazioni

3D Multi-Person Tracking in a multi camera environment using color features

Candidato:
Rossella Sblendido
Matricola 680558

Anno Accademico 2007–2008

Summary

The current work is concerned with the problem of improving the multi-person tracking in an indoor scenario in a multiple camera setup using color features. Redundancy among cameras is exploited to obtain a 3D representation of the scene by means of colored voxels, used as input for the tracking systems. A novel tracking technique based on the seminal particle filtering principle is applied: Sparse Sampling. This algorithm recursively estimate the state X_t, which in our case is the position of the centroid of a person, from an evolving set of samples placed on the surface of the tracked person. Weights are computed according to geometrical and color information contained in the voxel where the sample is placed. On one hand we consider the probability of a voxel to be a surface voxel and on the other we compute the probability of the voxel to belong to the volume of a target, using the color information stored in the Reference Histogram. To make the model resistent to illumination changes histograms are updated at every frame. Since the color histogram representation relies completely on the information extracted from the target color distribution, the choice of an appropriate color space is fundamental. We implemented a color system ad hoc, based on the YCrCb system. Multiple targets are tracked assigning a filter to every one and in order to achieve the most independent set of trackers, we consider a 3D blocking method to model interactions. Our system proved its effectiveness thought various tests and by means of objective metrics defined in the framework the CLEAR [16] multi-target tracking database. It guarantees performance similar to those of more complex algorithms (such as particle filtering) but with lower computational load.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The current work is concerned with the problem of improving the tracking of a group of people present in an indoor scenario in a multiple camera setup using color features. Robust, multi-person tracking systems are employed in a wide range of applications, including smart room environments, surveillance for security, health monitoring, as well as providing location and context features for human-computer interaction.

A number of methods for camera based multi-person 3D tracking has been proposed in the literature [19]. A common goal in these systems is robustness under occlusions created by multiple objects present in the scene. Single camera approaches [20] have been widely employed but are more vulnerable to occlusions, rotation and scale changes of the target. In order to avoid these drawbacks, multi-camera tracking techniques [13] exploit spatial redundancy among different views and provide 3D information as well. Integration of features extracted from multiple cameras has been proposed in terms of image correspondences [21], multi-view histograms [22] or voxel reconstructions [5].

Filtering techniques are employed to add temporal consistency to tracks. Kalman filtering approaches have been extensively used to track a single object under Gaussian uncertainty models and linear dynamics [20]. However, these methods do not perform accurately when facing noisy scenes or rapidly maneuvering targets. Particle filtering has been widely applied to cope with these situations since it can deal with multi-modal *pdf*s and is able to recover from lost tracks [10, 12].

Color is not an intrinsic property of an object. It is the perception of the energy emitted or reflected from the object, once processed by the human visual system and the brain, which makes us assign colors to this energy. Although the process followed by human brain in perceiving and interpreting color is a physio-psychological phenomenon that is not yet fully understood, it's clear that it is one of the most important characteristics used by our brain to identify objects or persons. Many methods have been proposed in literature to exploit the color information to implement accurate tracking system. Color histograms, in particular, are widely used because they are robust to partial occlusion, are rotation and scale invariant and are calculated efficiently. Color-based particle filters [17, 23] track the targets by comparing its histogram with the histograms of the sample positions using the Bhattacharyya distance. Illumination conditions, the visual angle, as well as the camera parameters can influence the quality of the recorded images with resulting appearance (and histogram) changes. Approaches to color constancy attempt to reconstruct the incident light and adjust the observed reflectance accordingly [25]. In practice, these methods are only applicable in highly constrained environment. To overcome this difficulty adaptive targets models are needed. Adaptivity is achieved by using Gaussian mixtures [26] and by updating regularly the histograms [24].

We propose a method for 3D tracking of multiple people in a multi-camera environment using color features. Redundancy among cameras is exploited to obtain a 3D representation of the scene by means of colored voxels, used as input for the tracking systems. A novel tracking technique based on the seminal particle filtering principle is applied: Sparse Sampling. This algorithm recursively estimate the state X_t, which in our case is the position of the centroid of a person, from an evolving set of samples placed on the surface of the tracked person. Weights are computed according to geometrical and color information contained in the voxel where the sample is placed. On one hand we consider the probability of a voxel to be a surface voxel and on the other we compute the probability of the voxel to belong to the volume of a target, using the color information stored in the Reference Histogram. To make the model resistent to illumination changes histograms are updated at every frame. Since the color histogram representation relies completely on the information extracted from the target color distribution, the choice of an appropriate color space is fundamental. We implemented a color system ad hoc, based on the YCrCb system. Multiple targets are tracked assigning a filter to every one and in order to achieve the most independent set of trackers, we consider a 3D blocking method to model interactions. Our system proved its effectiveness thought various tests and by means of objective metrics defined in the framework the CLEAR [16] multi-target tracking database. It guarantees performance similar to those of more complex algorithms (such as particle filtering) but with lower computational load. computational load.

1.1 Organization of the work

In chapter 2 we will introduce color in signal processing, describing its physical properties, how it is perceived by human beings and we will illustrate the various color systems. In chapter 3 an overview of various tracking algorithm is presented: Kalman Filter, Extended Kalman Filter, Particle Filter and multi-person tracking. In chapter 4 we will provide a detailed description of the our system: we will explain the background learning technique used, how voxels are generated and colored, the tracking algorithm implemented and how the color features are computed. In chapter 5 the results of the performed tests are presented, together with a description of the metrics used.

Chapter 2

Color in signal processing

The basic idea of our approach is that color can improve object tracking, providing better results than using just geometric information. In this chapter we will introduce color in signal processing, describing its physical properties, how it is perceived by human beings and finally we will illustrate the various color systems.

2.1 Physics of color

Color is not an intrinsic property of an object. It is the perception of the energy emitted or reflected from the object, once processed by the human visual system and the brain, which makes us assign colors to this energy. Although the process followed by human brain in perceiving and interpreting color is a physio-psychological phenomenon that is not yet fully understood, the physical nature of color can be expressed on a formal basis supported by experimental and theoretical results. In 1666, Sir Isaac Newton discovered that when a beam of sunlight passes through a glass prism, the emerging beam of light is not white but consists instead of continuous spectrum of colors ranging from violet at one end to red at the other. Figure 2.1 shows that the color spectrum may be divided into six broad regions: violet, blue, green yellow, and red. No color in the spectrum ends abruptly, but rather each color blends smoothly into the next.

Basically the color that humans perceive in an object are determined by the nature of the light reflected from the object. Visible light is composed of a relatively narrow band of frequencies in the electromagnetic spectrum, see fig. 2.2.

A body that reflects light that is balanced in all visible wavelengths appears white to the observer. However, a body that favors reflectance in a limited range of the visible spectrum exhibits some shades of color. For example green objects reflect light with wavelengths primarily in the 500 to 570 nm range while absorbing most of the energy at other wavelengths. Characterization of light is central to

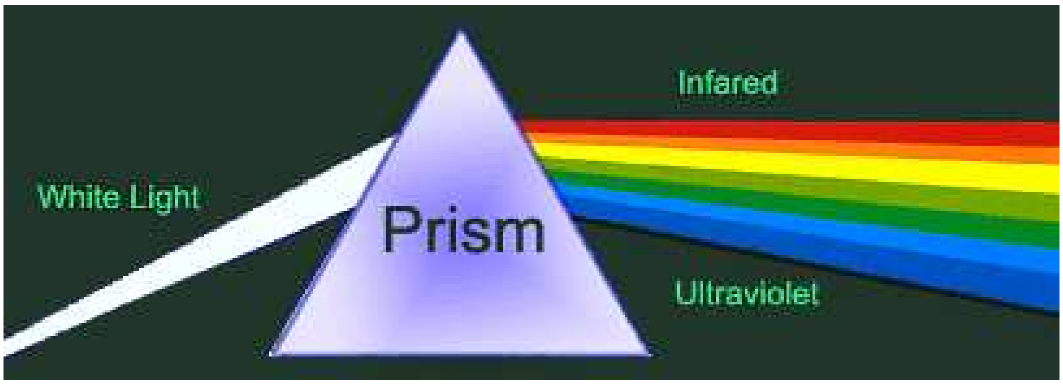


Figure 2.1. Color prism.

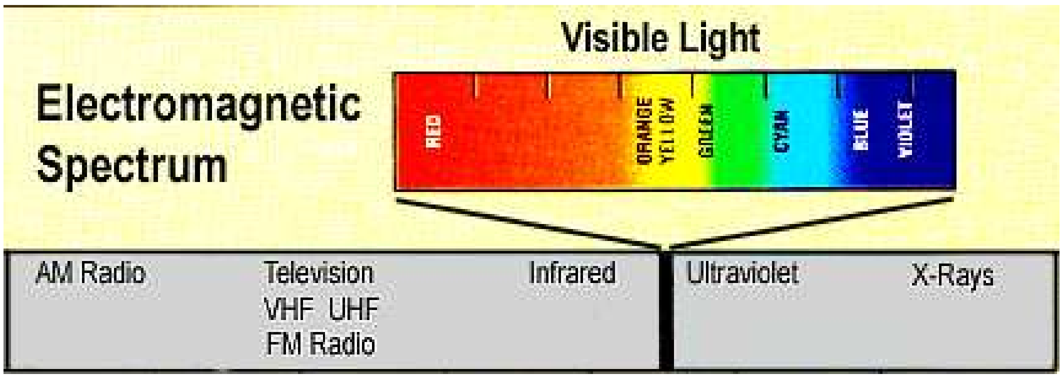


Figure 2.2. Electromagnetic spectrum.

the science of color. If the light is achromatic (void of color), its only attribute is its intensity. Achromatic light is what viewers see on a black and white television set. Chromatic light spans the electromagnetic spectrum from approximately 400 to 700 nm. Three basic quantities are used to describe the quality of a chromatic light source: radiance, luminance, and brightness. Radiance is the total amount of energy that flows from the light source, and it is usually measured in watts (W). Luminance, measured in lumens (lm), gives a measure of the amount of energy that an observer perceives from a light source. For example, light emitted from a source operating in the far infrared region of the spectrum could have significant energy (radiance), but an observer would hardly perceive it; its luminance would be almost zero. Finally, brightness is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of intensity and it's one of the key factors in describing color sensation.

2.2 Color perception

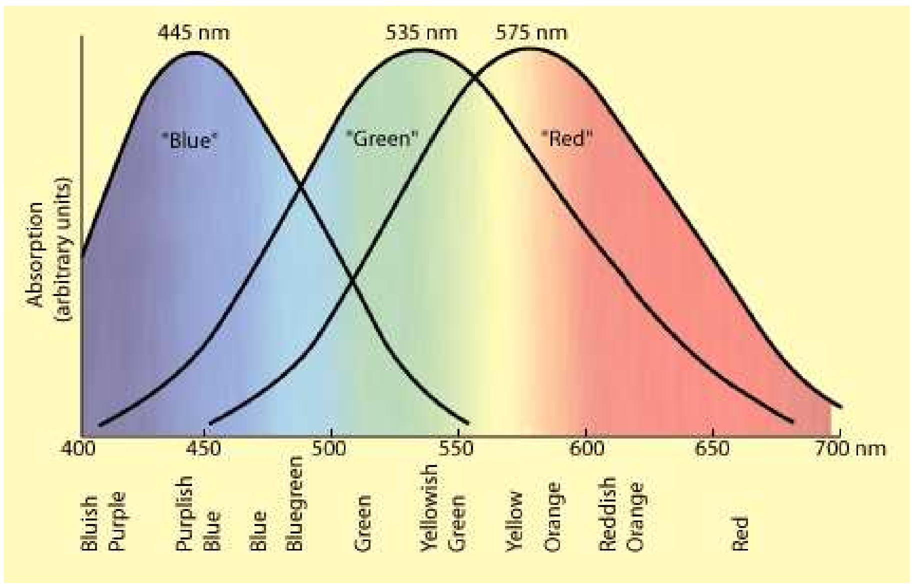


Figure 2.3. Absorption of light by red, green and blue cones in the human eye as a function of wavelength.

Cones are the sensors in the eye responsible for color vision. Detailed experimental evidence has established that the 6 to 7 million cones in the human eye can be divided into three principal sensing categories, corresponding roughly to red, green, and blue. Approximately 65% of all cones are sensitive to red light, 33% are sensitive to green light, and only about 2% are sensitive to blue (but blue cones are the most sensitive). Figure 2.3 shows average experimental curves detailing the absorption of light by red, green and blue cones in the eye.

Due to these absorption characteristics of the human eye, colors are seen as variable combinations of the so-called primary color red (R) , green (G), and blue (B). The amounts of red, green, and blue needed to form any particular color are called *tristimulus values*.

Any trichromatic theory cannot generate all the colors perceived by the brain, which is to say that, like most theories, there are some limitations. For example, using a modern laser-display system with monochromatic primaries at 635 nm (red), 532 nm (green), and 447 nm (blue), we try to simulate the perception of a monochromatic light at 580 nm (an orange color). Since the monochromatic orange stimulus excites the greenish and reddish cones, a contribution is required by both the green and red primaries, while no contribution is required from the blue primary. The problem is that the green primary also excites the bluish cones, making it impossible to exactly replicate the orange stimulus. This situation is not exclusive to the set of primaries in this example and is mostly perceptible when trying to reproduce very pure or monochromatic colors. Anyway the trichromatic theory works very well for most colors that are not monochromatic. Not to mention the fact that monochromatic colors, typically generated by lasers or far away cosmic phenomenon, are seldom seen in our daily lives. Nonetheless, it can be shown that, in some instances, some of the outofrange colors can be obtained by clever signal processing.

While the mechanisms of color vision at the level of the retina are well-described in terms of tristimulus values, color processing after that point is organized differently. A dominant theory of color vision proposes that color information is transmitted out of the eye by three opponent processes, or opponent channels, each constructed from the raw output of the cones: a red-green channel, a blue-yellow channel and a black-white ”luminance” channel. This theory has been supported by neurobiology, and accounts for the structure of our subjective color experience. Specifically, it explains why we cannot perceive a ”reddish green” or ”yellowish blue,” and it predicts the color wheel: it is the collection of colors for which at least one of the two color channels measures a value at one of its extremes. The exact nature of color perception beyond the processing already described, and indeed the status of color as a feature of the perceived world or rather as a feature of our perception of the world, is a matter of complex and continuing philosophical dispute.

The primary colors can be added to produce the secondary colors of light: magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). Mixing the three primaries, or secondary with its opposite primary color, in the right intensities produces white light. The characteristics generally used to distinguish one color from another are brightness, hue and saturation. As indicated earlier in section 2.1, brightness embodies the achromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Hue represents dominant color as perceived by an observer. Thus, when we call an object red, orange, or yellow, we specify its hue. The pure spectrum colors are fully saturated. Color such es pink (red and white) and lavender (violet and white) are less saturated, with the degree of saturation being inversely proportional to the amount of white light added. Hue and saturation taken together are called chromaticity, and, therefore, a color may be characterized by its brightness and chromaticity.

2.3 Color spaces

Color models, like all mathematical representations of physical phenomena, can be expressed in many different ways, each with its advantages and drawbacks. Some representation are formulated to help humans select colors and others are formulated to ease data processing in machines, with the various RGB spaces all falling in this last category. The goal is to minimize formulation complexity and the number of variables while maximizing “substance” and breath of coverage. One thing they have in common is the number of variables, or dimensions. Historically, whatever the meaning assigned to the variables, three of them were enough to describe all colors: RGB, Hue-Saturation-Intensity (HSI) and other HS based models, $L * a * b*$, xyY, etc. From this observation alone, one would be tempted to conclude that color is perceived with a three-signal-output mechanism to the brain since nature often uses a minimalist approach to do things. In many cases, more variables are added to complete a theory’s coverage or to supplement a physical limitation of the reproduction process. For example, black content (“K”) is added to cyan, magenta and yellow (CMY) inks to obtain better dark tones in traditional printing. Printing processes with more than four colors have been developed to extend the reproducible color range. These added variables are not additional dimensions per se since they are not totally independent of the primary coordinates. The first major distinction between color spaces is device dependency. Color coordinates from a device independent space are the same on all output media. For example, discounting surface reflection effects such as “shininess” or gloss, if the coordinates of a car color and the coordinates of that color in an image of that car are the same, then these colors are identically perceived by the human eye. The xyY space, and its equivalent XYZ representation, falls in this category. Expressing the previous example differently, a stimulus characterized by a given XYZ triad will be perceived as the same color as a stimulus from another source which has the same XYZ triad. The XYZ space is based on how human perceive light and is thus independent of the media on which the color is seen. On the other hand, a device dependent color space will have different coordinates for the same color for various output media. All RGB and CMYK spaces fall in the device dependent camp since they are defined in relation to very specific primary colors, either phosphors or ink. As many have found, an RGB triad from an Apple Macintosh computer does not represent the same color as the triad with identical values on a Windows machine. Another distinction is the ease with which the coordinates can be mentally associated with the codified color. The HSB space is being promoted as user-friendly since it is relatively easy to relate a HSB triad to the color represented by a given hue (the chromatic content, the presence of a color), its saturation (the ratio between chromatic and achromatic, i.e. white, gray, black, contents), and brightness (a relative-to-white lightness-darkness level). This ease of use is also true for other hue-saturation based models but the varieties of name and definitions for the third parameter, brightness, value, or intensity, can bring confusion. On the other hand, a set of RGB or CMYK coordinates can be difficult to visualize, and xyY values are almost impossible to deal with for the infrequent user.

2.3.1 Tristimulus and CIE RGB color space

One can picture this space as a region in three-dimensional Euclidean space if one identifies the x, y, and z axes with the stimuli for the long-wavelength (L) (red cones), medium-wavelength (M) (green cones), and short-wavelength (S) (blue cones) receptors. The origin, (S,M,L) = (0,0,0), corresponds to black. White has no definite position in this diagram; rather it is defined according to the color temperature or white balance as desired or as available from ambient lighting. The human color space is a horse-shoe-shaped cone such as shown in figure 2.4, extending from the origin to, in principle, infinity. In practice, the human color receptors will be saturated or even be damaged at extremely-high light intensities.

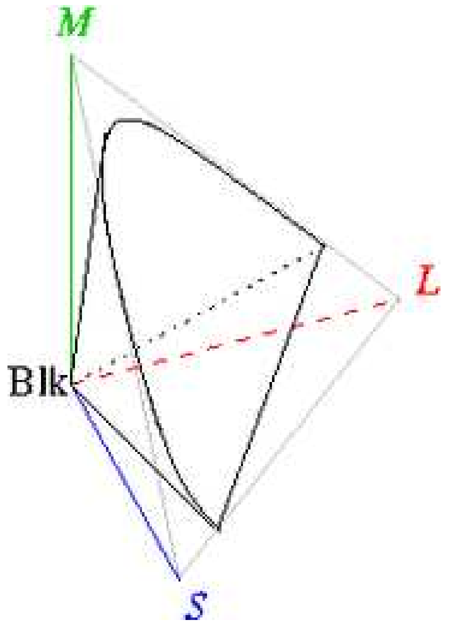


Figure 2.4. 3D representation of the human color space.

The most saturated colors are located at the outer rim of the region, with brighter colors farther removed from the origin. As far as the responses of the receptors in the eye are concerned, there is no such thing as "brown" or "gray" light. The latter color names refer to orange and white light respectively, with an intensity that is lower than the light from surrounding areas. One can observe this by watching the screen of an overhead projector during a meeting: one sees black lettering on a white background, even though the "black" has in fact not become darker than the white screen on which it is projected before the projector was turned on. The "black" areas have not actually become darker but appear "black" relative to the higher intensity "white" projected onto the screen around it.

The human tristimulus space has the property that additive mixing of colors corresponds to the adding of vectors in this space. This makes it easy to, for example, describe the possible colors (gamut) that can be constructed from the red, green, and blue primaries in a computer display.

In the 1920s, W. David Wright (Wright 1928) and John Guild (Guild 1931) independently conducted a series of experiments on human sight which laid the foundation for the specification of the CIE XYZ color space. The experiments were conducted by using a circular split screen 2 degrees in size, which is the angular size of the human fovea. On one side of the field a test color was projected and on the other side, an observer-adjustable color was projected. The adjustable color was a mixture of three primary colors, each with fixed chromaticity, but with adjustable brightness. The observer would alter the brightness of each of the three primary beams until a match to the test color was observed. Not all test colors could be matched using this technique. When this was the case, a variable amount of one of the primaries could be added to the test color, and a match with the remaining two primaries was carried out with the variable color spot. For these cases, the amount of the primary added to the test color was considered to be a negative value. In this way, the entire range of human color perception could be covered. When the test colors were monochromatic, a plot could be made of the amount of each primary used as a function of the wavelength of the test color. These three functions are called the color matching functions for that particular experiment. The color matching functions are the amounts of primaries needed to match the monochromatic test primary at the wavelength shown on the horizontal scale.

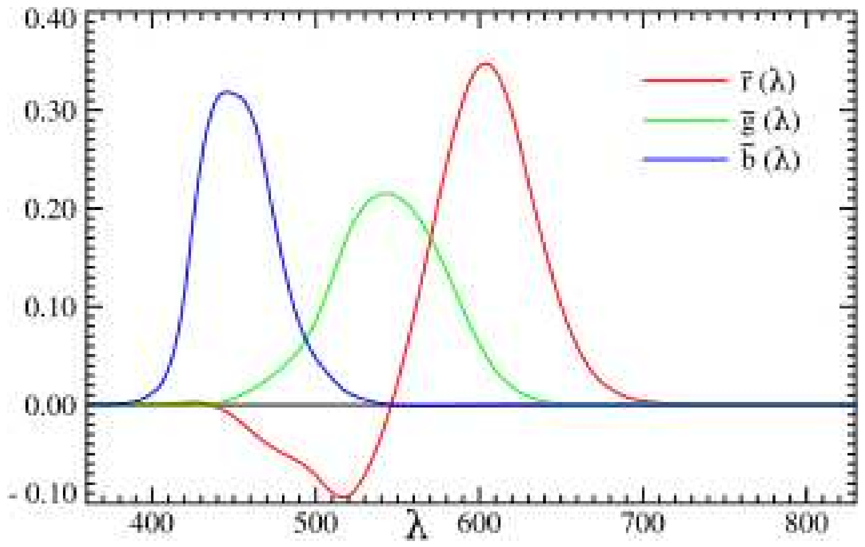


Figure 2.5. The CIE 1931 RGB Color matching functions.

Although Wright and Guild's experiments were carried out using various primaries at various intensities, and a number of different observers, all of their results were summarized by the standardized CIE RGB color matching functions $\overline{r}(\lambda)$, $\overline{g}(\lambda)$, and $\overline{b}(\lambda)$, obtained using three monochromatic primaries at standardized wavelengths of 700 nm (red), 546.1 nm (green) and 435.8 nm (blue). The color matching functions are the amounts of primaries needed to match the monochromatic test primary. These functions are shown in figure 2.5. Note that $\overline{r}(\lambda)$ and $\overline{g}(\lambda)$ are zero at 435.8, $\overline{r}(\lambda)$ and $\overline{b}(\lambda)$ are zero at 546.1 and $\overline{g}(\lambda)$ and $\overline{b}(\lambda)$ are zero at 700 nm, since in these cases the test color is one of the primaries. The primaries with wavelengths 546.1 nm and 435.8 nm were chosen because they are easily reproducible monochromatic lines of a mercury vapor discharge. The 700 nm wavelength, which in 1931 was difficult to reproduce as a monochromatic beam, was chosen because the eye's perception of color is rather unchanging at this wavelength, and therefore small errors in wavelength of this primary would have little effect on the results. The color matching functions and primaries were settled upon by a CIE special commission after considerable deliberation. The cut-offs at the short and longwavelength side of the diagram are chosen somewhat arbitrarily; the human eye can actually see light with wavelengths up to about 810 nm, but with a sensitivity that is many thousand times lower than for green light. These color matching functions define what is known as the "1931 CIE standard observer". Note that rather than specify the brightness of each primary, the curves are normalized to have constant area beneath them. This area is fixed to a particular value by specifying that

$$\int_0^\infty \overline{r}(\lambda)\,d\lambda = \int_0^\infty \overline{g}(\lambda)\,d\lambda = \int_0^\infty \overline{b}(\lambda)\,d\lambda \tag{2.1}$$

The resulting normalized color matching functions are then scaled in the r:g:b ratio of 1:4.5907:0.0601 for source luminance and 72.0962:1.3791:1 for source radiant power to reproduce the true color matching functions. By proposing that the primaries be standardized, the CIE established an international system of objective color notation.

Given these scaled color matching functions, the RGB tristimulus values for a color with a spectral power distribution $I(\lambda)$ would then be given by:

$$R = \int_0^\infty I(\lambda)\,\overline{r}(\lambda)\,d\lambda \tag{2.2}$$

$$G = \int_0^\infty I(\lambda)\,\overline{g}(\lambda)\,d\lambda \tag{2.3}$$

$$B = \int_0^\infty I(\lambda)\,\overline{b}(\lambda)\,d\lambda \tag{2.4}$$

These are all inner products and can be thought of as a projection of an infinite-dimensional spectrum to a three-dimensional color.

2.3.2 CIE XYZ color space

One of the first mathematically defined color spaces was the CIE 1931 XYZ color space (also known as CIE 1931 color space), created by the International Commission on Illumination (CIE) in 1931. After developing the RGB model of human vision using the CIE RGB matching functions, the members of the special commission wished to develop another color space which related to the CIE RGB color space by a linear transformation.

In the CIE XYZ color space, the tristimulus values are not the S, M, and L responses of the human eye, but rather a set of tristimulus values called X, Y, and Z, which are roughly red, green and blue, respectively. Two light sources, made up of different mixtures of various wavelengths, may appear to be the same color; this effect is called metamerism. Two light sources have the same apparent color to an observer (such as the CIE 1931 standard observer) when they have the same tristimulus values, no matter what spectral distributions of light were used to produce them.

The CIE has defined a set of three color-matching functions, called $\overline{x}(\lambda)$, $\overline{y}(\lambda)$, and $\overline{z}(\lambda)$, which can be thought of as the spectral sensitivity curves of three linear light detectors that yield the CIEXYZ tristimulus values X, Y, and Z. The tabulated numerical values of these functions are known collectively as the CIE standard observer, see figure 2.6.

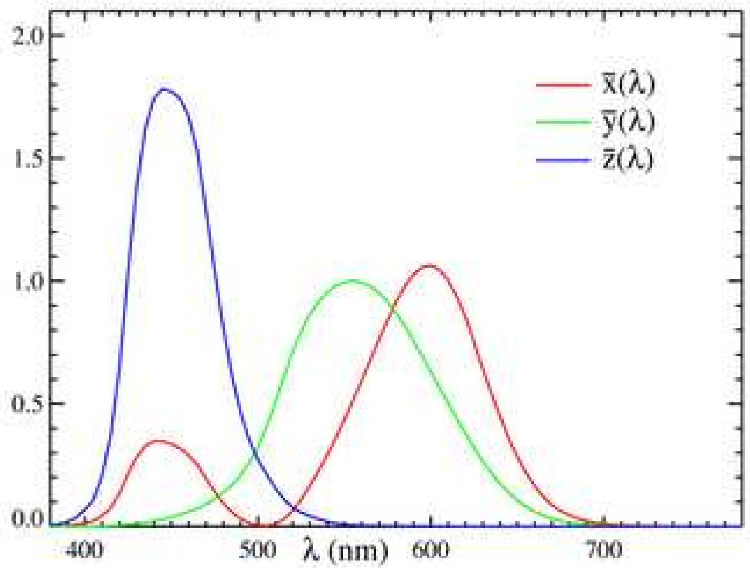


Figure 2.6. The CIE standard observer color-matching functions.

The tristimulus values for a color with a spectral power distribution $I(\lambda)$, are given in terms of the standard observer by:

$$X = \int_0^\infty I(\lambda)\,\overline{x}(\lambda)\,d\lambda \tag{2.5}$$

$$Y = \int_0^\infty I(\lambda)\,\overline{y}(\lambda)\,d\lambda \tag{2.6}$$

$$Z = \int_0^\infty I(\lambda)\,\overline{z}(\lambda)\,d\lambda \tag{2.7}$$

he CIE XYZ color space was deliberately designed so that the Y parameter was a measure of the brightness or luminance of a color. The chromaticity of a color was then specified by the two derived parameters x and y, two of the three normalized values which are functions of all three tristimulus values X, Y, and Z:

$$x = \frac{X}{X+Y+Z} \tag{2.8}$$

$$y = \frac{Y}{X+Y+Z} \tag{2.9}$$

$$z = \frac{Z}{X+Y+Z} \tag{2.10}$$

It is noted from these equations that

$$x + y + z = 1 \tag{2.11}$$

The derived color space specified by x, y, and Y is known as the **CIE xyY** color space and is widely used to specify colors in practice. For any value of x and y, the corresponding as a function of z (blue) is obtained from equations 2.8,2.9,2.10,2.11 by noting that $z = 1 - (x + y)$.

The X and Z tristimulus values can be calculated back from the chromaticity values x and y and the Y tristimulus value:

$$X = \frac{Y}{y}x \tag{2.12}$$

$$Z = \frac{Y}{y}(1 - x - y) \tag{2.13}$$

Figure 2.7 shows the related chromaticity diagram. The chromaticity diagram is a tool to specify how the human eye will experience light with a given spectrum. It cannot specify colors of objects (or printing inks), since the chromaticity observed while looking at an object depends on the light source as well.

The chromaticity diagram illustrates a number of interesting properties of the CIE XYZ color space:

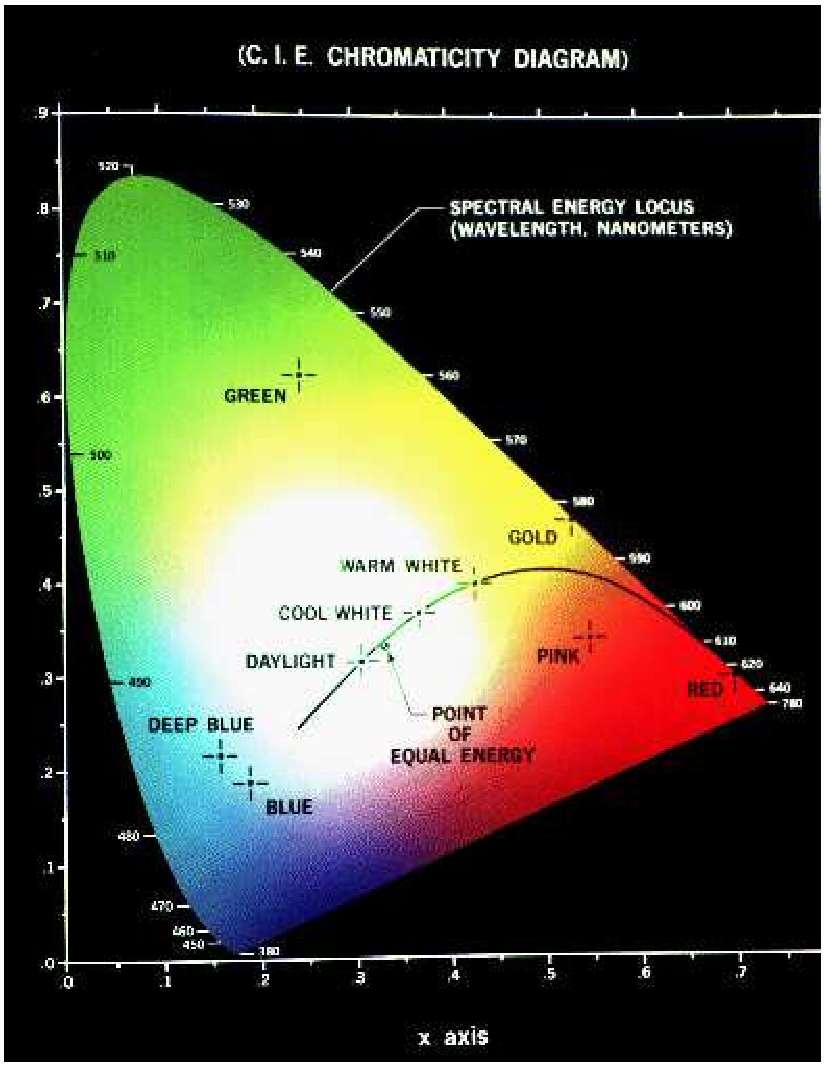


Figure 2.7. CIE chromaticity diagram.

- The positions of the various spectrum colors, from violet at 380 nm to red 780 nm, are indicated around the boundary of the tongue-shaped diagram. These are the pure colors shown in the spectrum of figure 2.2. Any point non actually on the boundary but within the diagram represents some mixture of spectrum colors.

- The point of equal energy correspond to equal fraction of the three primary colors; it represents the CIE standard for white light. Any point located on the boundary of the chromaticity chart is fully saturated. As a point leaves the boundary and approaches the point of equal energy, more white light is added to the color and it becomes less saturated. The saturation at the point of equal energy is zero.

- It is seen that all visible chromaticities correspond to non-negative values of x, y, and z (and therefore to non-negative values of X, Y, and Z).

- The chromaticity diagram is useful for color mixing because a straightline segment joining any two point in the diagram defines all the different color variations that can be obtained combining these two colors additively. Consider, for example, a straight line drawn from the red to the green points shown in figure 2.7. If there is more red light than green light, the exact point representing the new color will be on the line segment, but will be closer to the red point than to the green point. Similarly, a line drawn from the point of equal energy to any point on the boundary of the chart will define all the shades of that particular spectrum color. Extension of the procedure to the three colors is straightforward. To determine the range of colors that can be obtained from any three given colors in the chromaticity diagram, we simply draw connecting lines to each of the three color point. The result is a triangle, and any color inside the triangle can be produced by various combinations of the three initial colors.

- An equal mixture of two equally bright colors will not generally lie on the midpoint of that line segment. In more general terms, a distance on the xy chromaticity diagram does not correspond to the degree of difference between two colors. The idea of measuring color difference was developed by David MacAdam and summarized in the concept of a MacAdam ellipse. Based on the work of MacAdam, the CIE $L*u*v*$ (1960) and CIE $L*a*b*$ (1976) color spaces were developed, both of which were designed to be perceptually uniform (have an equal distance in the color space correspond to equal differences in color), as measured by MacAdam. Although they were a distinct improvement over the CIE 1931 system, they were not completely free of distortion.

- A triangle with vertices at any three fixed colors cannot enclose the entire color region in figure 2.7. This observation supports graphically the remark that not all colors can be obtained with three, single, fixed primaries.

2.3.3 RGB color space

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors, see figure

2.3.3. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

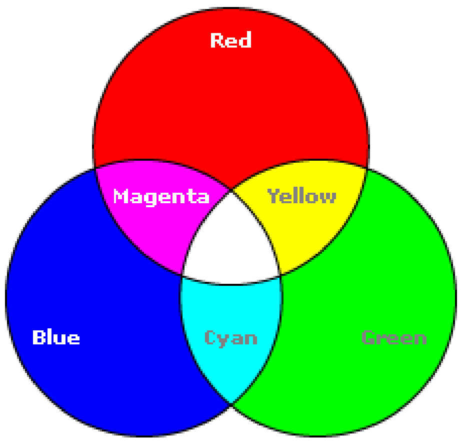


Figure 2.8. Additive color model.

The RGB color model itself does not define what is meant by red, green and blue colorimetrically, and so the results of mixing them are not specified as exact, but relative.

When the exact chromaticities of the red, green, and blue primaries are defined, the color model then becomes an absolute color space, such as sRGB or Adobe RGB.

The choice of 'primary' colors is related to the physiology of the human eye; good primaries are stimuli that maximize the difference between the responses of the cone cells of the human retina to light of different wavelengths, and that thereby make a large color triangle. The normal three kinds of light-sensitive photoreceptor cells in the human eye (cone cells) respond most to red (long wavelength or L), green (medium or M) and blue (short or S) light (peak wavelengths near 570 nm, 540 nm and 440 nm respectively). The difference in the signals received from the three kinds allows the brain to differentiate a wide gamut of different colors, while being most sensitive (overall) to yellowish-green light and to differences between hues in the green-to-orange region. Use of the three primary colors is not sufficient to reproduce all colors; only colors within the color triangle defined by the chromaticities of the primaries can be reproduced by additive mixing of non-negative amounts of those colors of light. A color in the RGB color model can be described by indicating how much of each of the red, green, and blue is included. Each can vary between the minimum (fully dark) and maximum (full intensity). If all the colors are at minimum the result is black. If all the colors at maximum, the result is a white.

These colors may be quantified in several different ways:

- Color scientists often place colors in the range 0 (minimum) through 1 (maximum). Many color formulae take these values. For instance, full intensity red using this convention is 1, 0, 0 for Red, Green, and Blue.
- The color values may be written as percentages, from 0% (minimum) to 100% (maximum). To convert from the range 0 to 1, see percentage. Full intensity red, using this notation, is 100%, 0%, 0%.
- The color values may be written as numbers in the range 0 to 255. This is commonly found in computer representations, where programmers have found it convenient to store each color value in one 8-bit byte. This convention has become so widespread that some writers now consider the range 0 to 255 an assumption and fail to give a context for their values. Full intensity red, using this scheme, is 255, 0, 0. This range of values is not proportional to the others, but rather a nonlinear gamma-encoded scale.
- That same range, 0 through 255, is sometimes written in hexadecimal, full intensity red becomes FF, 00, 00, which can be contracted to #FF0000 (a convention used by HTML).

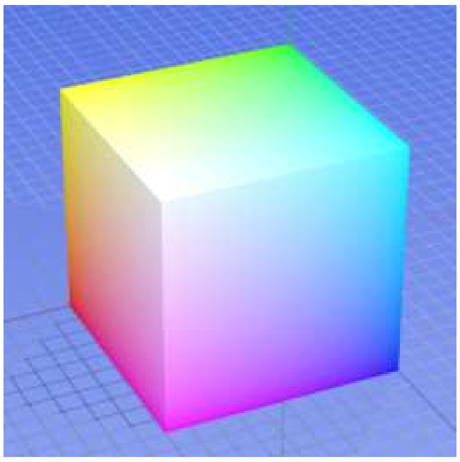


Figure 2.9. The RGB color model mapped to a cube. Values increase along the xaxis (red), yaxis (green) and zaxis (blue).

24-bit representation

RGB values encoded in 24 bits per pixel (bpp) are specified using three 8-bit unsigned integers (0 through 255) representing the intensities of red, green, and blue (usually in that order). For example, the following image shows the three ”fully saturated” faces of an RGB cube, unfolded into a plane. The above definition uses a convention known as full-range RGB. Color values are also often considered to be in the range 0.0 through 1.0, which may be mapped to other digital encodings.

Full-range RGB using eight bits per primary can represent up to 256 shades of white-grey-black, 255 shades of red, green, and blue (and equal mixtures of those), but fewer shades of other hues. The 256 levels do not represent equally spaced intensities, due to gamma correction.

Typically, RGB for digital video is not full range. Instead, video RGB uses a convention with scaling and offsets such that (16, 16, 16) is black, (235, 235, 235) is white, etc. For example, these scalings and offsets are used for the digital RGB definition in CCIR 601.

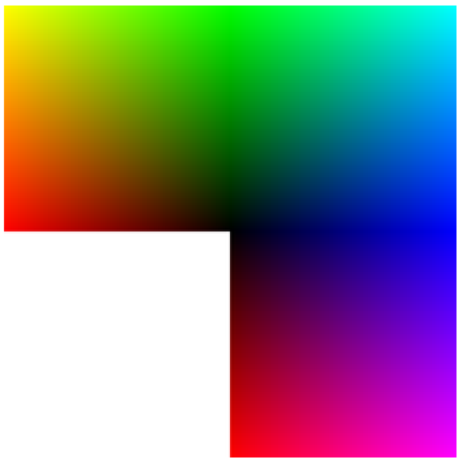


Figure 2.10. The RGB 24-bit representation

16-bit mode

There is also a 16 bpp mode sometimes known as Highcolor, in which there are either 5 bits per color, called 555 mode, or an extra bit for green (because the green component contributes most to the brightness of a color in the human eye), called 565 mode. In general, an RGB representation needs 1 bit more for red than blue and 1 more bit for green.

32-bit mode

The so-called 32 bpp mode is almost always identical in precision to the 24 bpp mode; there are still only eight bits per component, and the eight extra bits are often not used at all. The reason for the existence of the 32 bpp mode is the higher speed at which most modern hardware can access data that is aligned to byte addresses evenly divisible by a power of two, compared to data not so aligned. Some graphics hardware allows the unused byte to be used as an 8-bit paletted overlay. A certain palette entry (often 0 or 255) is designated as being transparent, i.e., where the overlay is this value the truecolor image is shown. Otherwise the overlay value is looked up in the palette and used. This allows for GUI elements (such as menus or the mouse cursor) or information to be overlayed over a truecolor image without modifying it. When the overlay needs to be removed, it is simply cleared to the transparent value and the truecolor image is displayed again.

48-bit mode (sometimes also called 16-bit mode)

16-bit mode can also refer to 16 bit per component, resulting in 48 bpp. This mode makes it possible to represent 65536 tones of each color component instead of 256. This is primarily used in professional image editing, like Adobe Photoshop for maintaining greater precision when a sequence of more than one image filtering algorithms is used on the image. With only 8 bit per component, rounding errors tend to accumulate with each filtering algorithm that is employed, distorting the end result.

RGBA

RGBA stands for **R**ed **G**reen **B**lue **A**lpha. While it is sometimes described as a color space, it is actually simply a use of the RGB color model, with extra information. The color is RGB, and may belong to any RGB color space, but an integral alpha value as invented by Catmull and Smith between 1971 and 1972 enables alpha blending and alpha compositing. The inventors named alpha after the Greek letter in the classic linear interpolation formula $\alpha A + (1 - \alpha)B$.

The alpha channel is the opacity channel. If a pixel has a value of 0% in its alpha channel, it is fully transparent (and, thus, invisible), whereas a value of 100% in the alpha channel gives a fully opaque pixel (traditional digital images). Values between 0% and 100% make it possible for pixels to show through a background like a glass (translucency), an effect not possible with simple binary (transparent or opaque) transparency. It allows easy image compositing. Alpha channel values can be expressed as a percentage, integer, or real number between 0 and 1 like RGB parameters.

Sometimes this is referred as ARGB (like RGBA, but first datum is alpha). For example, 0x80FFFF00 is 50%-transparent yellow, because all parameters are expressed on a 0-to-255 scale. 0x80 is 128, approximately half 255 (alpha 50%); 0xFF is 255, the greatest value a parameter can have (pure red); the second 0xFF is like the previous, but for green; and 0x00 is 0 in decimal (no blue). Red, green, and half transparency mixture are 50% transparent yellow.

2.3.4 HSI representation

Recognizing that the geometry of the RGB model is poorly aligned with the color-making attributes recognized by human vision, computer graphics researchers developed an alternate representation of RGB, **HSV** (**H**ue, **S**aturation, **I**ntensity), in the late 1970s, formally defined and described in Alvy Ray Smith [7].

When humans view a color object, we don't describe it by giving the percentage of each of the primaries but by its hue, saturation, and brightness. Hue is a color attribute that describes a pure color (pure yellow, orange o red for example), whereas saturation gives a measure of the degree to which pure color is diluted by white light. Brightness is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of intensity and is one of the key factors in describing color sensation. Intensity (grey level) is a most useful descriptor of monochromatic images. This quantity definitely is measurable and easily interpretable. The HSI color model decouples the intensity component from color-carrying information (hue and saturation) in a color image. As a result, the HSI model is an ideal tool for developing image-processing algorithms based on color descriptors that are natural and intuitive to humans. RGB is ideal for image color generation (as in image capture by a color camera or image display in a monitor screen), but its use for color description is much more limited. An RGB color image can be viewed as three monochrome intensity images (representing red, green and blue), so it should be possible to extract intensity from an RGB image. This become more clear if we take the RGB color cube, figure 2.3.3, and stand it on black (0,0,0) vertex, with the white vertex (1,1,1) directly above it, as shown in figure 2.3.4. The intensity (gray scale) is along the line joining this two vertices. Thus if we want to determine the intensity component of any color point we would simply pass a plane perpendicular to the intensity axis and containing the color point. The intersection of the plane with the intensity axis would give us a point with intensity value in the range [0.1]. We also note that the saturation (purity) of a color increases as a function of distance from the intensity axis. In fact, the saturation of points on the intensity axis is zero, as evidence by the fact that all points along the axis are gray.

In order to see how hue can be determined also from a given RGB point,consider figure 2.3.4, which shows a plane defined by three points (black, white and cyan). The fact that the black and white points are contained in the plane tells us that the intensity axis is also contained in the plane. Furthermore, we see that all points contained in the plane segment defined by the intensity axis and the boundaries of the cube have the *same* hue (cyan in this case). We would arrive at the same conclusion by recalling that all colors generated by three colors lie in the triangle defined by those colors. If two of those points are black and white and the third a

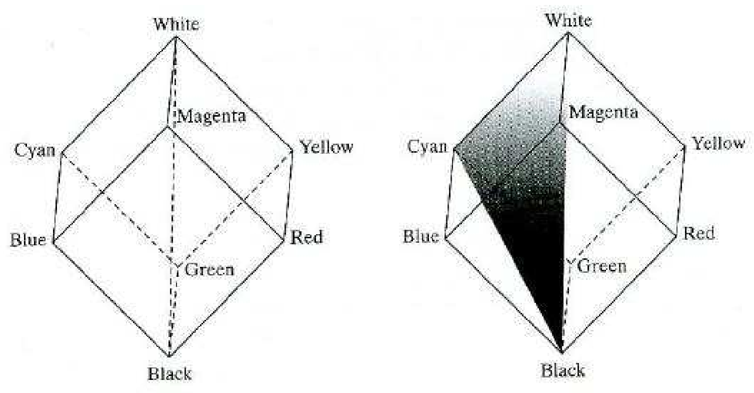


Figure 2.11. Conceptual relationships between the RGB and HSI color models

color point, all points on the triangle would have the same hue because the black and white components cannot change the hue (of course, the intensity and saturation of points in this triangle would be different). By rotating the shaded plane about the vertical intensity axis, we would obtain different hues. From these concepts we arrive at the conclusion that the hue, saturation and intensity values required from the HSI space can be obtained from the RGB color cube. That is, we can convert any RGB point to a corresponding point in the HSI color model by working out the geometrical formulas describing the reasoning outlined before. The key point is that the HSI space is represented by a vertical intensity axis and the locus of color points that lie perpendicular to this axis. HSI improves on the color cube representation of RGB by arranging colors of each hue in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top. As the planes moves up and down the intensity axis, the boundaries defined by the intersection of each plane with the faces of the cube have either a triangular or hexagonal shape. This can be visualized by looking at the cube down its greyscale axis, as shown in figure 2.3.4 at the top. In this plane we see that primary colors are separated by 120°. The secondary colors are 60° from the primaries, which means that the angle between the secondaries also is 120°. At the bottom figure 2.3.4 shows the same hexagonal shape and an arbitrary color point (shown as a dot). The hue of the point is determined by an angle from some reference point. Usually (but not always) an angle of 0° from the red axis designates 0 hue, and the hue increases counterclockwise from there. The saturation (distance from the vertical axis) is the length of the vector from the origin to the point. The origin is defined by the intersection of the color plane with the vertical intensity axis. The important components of the HSI color spaces are the vertical intensity axis, the length of the vector to a color point, and the angle this vector makes with the red axis. Therefore it is not unusual to see the HSI planes defined in term of the hexagon just discussed, a triangle, or even a circle, as figure 2.3.4 shows. The shape chosen really doesn't matter, since any one of these shapes can be warped into one of the other two by a geometric transformation. Figure 2.3.4 shows a 3D representation of the HSI model defined in terms of the hexagon.

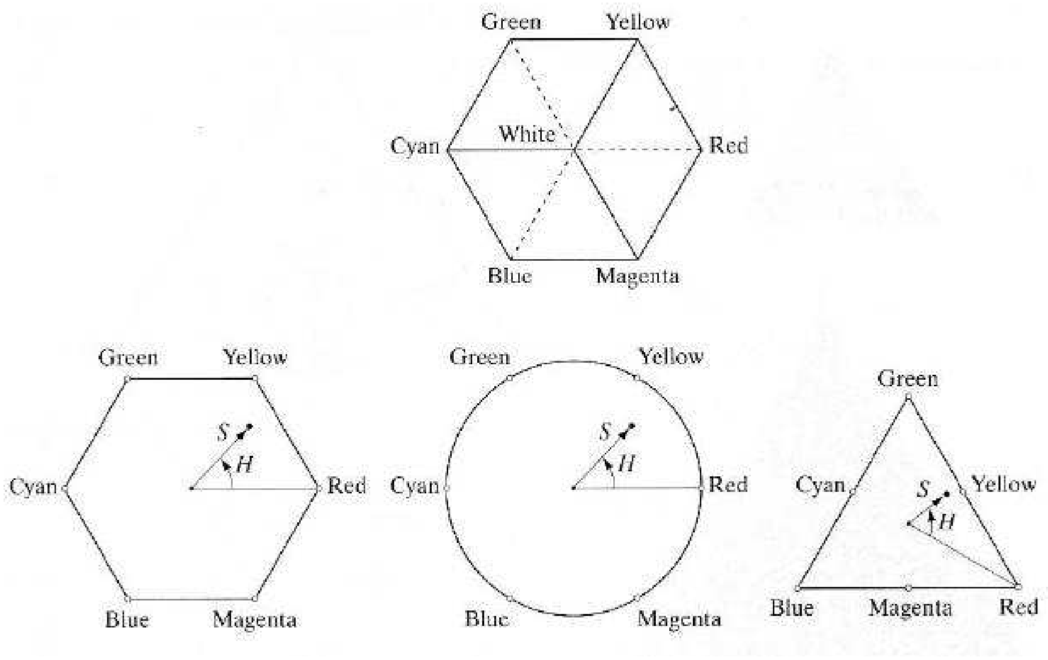


Figure 2.12. Hue and saturation in the HSI color model.

2.3.5 CMY and CMYK color model

CMYK (short for **C**yan, **M**agenta, **Y**ellow, and **K**ey (black), and often referred to as **process color** is a subtractive color model, used in color printing, also used to describe the printing process itself. Cyan, magenta and yellow are the secondary colors of light or, alternatively the primary colors of pigments. According to figure 2.14 equal amounts of pigment primaries, cyan, magenta, and yellow should produce black. In practice, combining these colors for printing produces a muddy-looking black. So, in order to produce true black (which is the predominant color in printing), a fourth color, *black*,is added. That's why this system is often called "four color".

The CMYK model works by partially or entirely masking certain colors on the typically white background (that is, absorbing particular wavelengths of light). Such

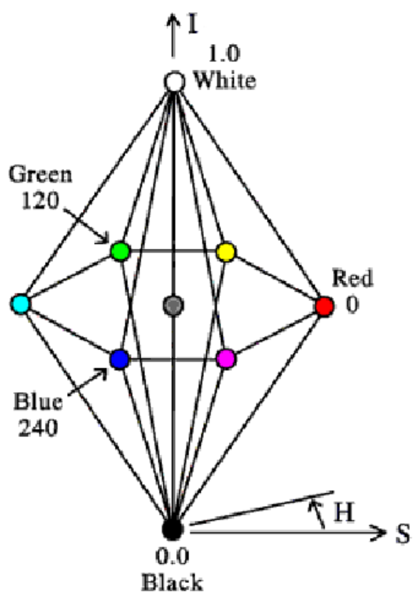


Figure 2.13. HSI color model representation.

a model is called subtractive because inks "subtract" brightness from white. For example, when a surface coated with cyan pigment is illuminated with white light, no red light is reflected from the surface. That is, cyan subtracts red light from reflected white light, which itself is composed of equal amounts of red, green and blue light.

In additive color models such as RGB, white is the "additive" combination of all primary colored lights, while black is the absence of light. In the CMYK model, it is just the opposite: white is the natural color of the paper or other background, while black results from a full combination of colored inks.

To convert from RGB to CMYK we use the following equation

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.14}$$

where the assuption is that all color values have been normalized to the range [0,1]. Equation 2.14 demonstrates that light reflected from a surface coated with pure cyan does not contain red (that is, $C = 1 - R$ in the equation). Similary, pure magenta does not reflect green, and pure yellow does not reflect blue.

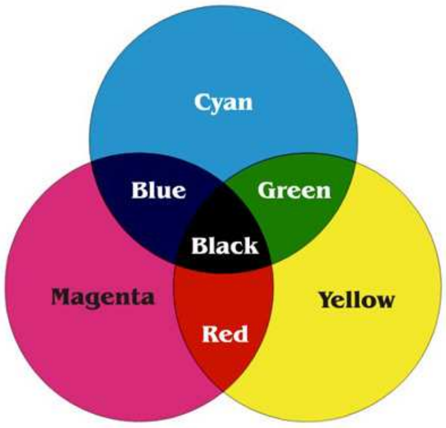


Figure 2.14. Subtractive color model.

2.3.6 YCrCb color model

YCbCr or **Y'CbCr** is a family of color spaces used in video and digital photography systems. Y' is the luma component and Cb and Cr are the blue and red chroma components. The prime on the Y is to distinguish the luma from luminance. YCbCr is sometimes abbreviated to YCC.

Y'CbCr is not an absolute color space. It is a way of encoding RGB information, and the actual color displayed depends on the actual RGB colorants used to display the signal. Therefore a value expressed as Y'CbCr is only predictable if standard RGB colorants are used, or if an ICC profile is attached (that is a profile that describes the colour attributes of a particular device) or implied which is used to translate value for the colorants in use. Video CRT displays are driven by red, green, and blue voltage signals, but these RGB color space signals are not efficient as a representation for storage and transmission, since they have a lot of mutual redundancy. Spaces such as Y'CbCr are used to separate out a luma signal (Y') that can be stored with high resolution or transmitted at high bandwidth, and two chroma components (Cb and Cr) that can be bandwidthreduced, subsampled, compressed, or otherwise treated separately for improved system efficiency. Y'CbCr signals (prior to scaling and offsets to place the signals into digital form) are called YPbPr, where

- Y carries luma (brightness) information.
- Pb carries the difference between blue and luma ($B - Y$).
- Pr carries the difference between red and luma ($R - Y$).

Luma is the weighted sum of gamma-compressed R'G'B' components of a color video. The word was proposed to prevent confusion between luma as implemented in video engineering and luminance as used in color science (i.e. as defined by CIE). Luminance is formed as a weighted sum of linear RGB components, not gamma-corrected ones.The formula used to calculate luminance used coefficients based on the CIE color matching functions and the relevant standard chromaticities of red, green, and blue.The definition of luminance (relative) is:

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{2.15}$$

The formula used to calculate luma arbitrarily also uses these same coefficients, but with gamma-compressed components:

$$Y' = 0.2126R' + 0.7152G' + 0.0722B' \tag{2.16}$$

where the prime symbol ' denotes gamma correction.

An example of gamma correction is the one used to define the sRBG color space, where $\gamma = 2.2$. Let C_{linear} be R, G, or B, and C_srgb be R_srgb,G_srgb or B_srgb, that is gamma-compressed:

$$C_{\text{srgb}} = \begin{cases} 12.92C_{\text{linear}}, & C_{\text{linear}} \leq 0.0031308 \\ (1+a)C_{\text{linear}}^{1/2.4} - a, & C_{\text{linear}} > 0.0031308 \end{cases} \tag{2.17}$$

where $a = 0.055$.

More in general YPbPr are created from the corresponding gamma-adjusted RGB (red, green and blue) source using two defined constants Kb and Kr as follows:

$$Y' = Kr \cdot R' + (1 - Kr - Kb) \cdot G' + Kb \cdot B', \tag{2.18}$$

$$Pb = \frac{1}{2} \cdot \frac{B' - Y'}{1 - Kb} \tag{2.19}$$

$$Pr = \frac{1}{2} \cdot \frac{R' - Y'}{1 - Kr} \tag{2.20}$$

where Kb and Kr are ordinarily derived from the definition of the corresponding RGB space.

Here, the prime (') symbols mean gamma correction is being used; thus R', G' and B' and to nominally range from 0 to 1, with 0 representing the minimum intensity (e.g., for display of the color black) and 1 the maximum (e.g., for display of the color white). The resulting luma (Y) value will then have a nominal range from 0 to 1, and the chroma (Cb and Cr) values will have a nominal range from -0.5 to +0.5. The reverse conversion process can be readily derived by inverting the above equations. When representing the signals in digital form, the results are scaled and rounded, and offsets are typically added. For example, the scaling and offset applied to the Y' component per specification (e.g. MPEG-2) results in the value of 16 for black and the value of 235 for white when using an 8-bit representation. The standard has 8-bit digitized versions of Cb and Cr scaled to a different range of 16 to 240. Consequently, rescaling by the fraction (235-16)/(240-16) = 219/224 is required when doing color matrixing or processing in YCbCr space, resulting in quantization distortions.

The scaling that results in the use of a smaller range of digital values than what might appear to be desirable for representation of the nominal range of the input data allows for some "overshoot" and "undershoot" during processing without necessitating undesirable clipping. Since the equations defining YCbCr are formed in a way that rotates the entire nominal RGB color cube and scales it to fit within a (larger) YCbCr color cube, there are some points within the YCbCr color cube that cannot be represented in the corresponding RGB domain (at least not within the nominal RGB range). This causes some difficulty in determining how to correctly interpret and display some YCbCr signals.

JPEG allows Y'CbCr where Y', Cb and Cr have the full 256 values, using $Kb = 0.114$ and $Kr = 0.299$:

$$Y' = +0.299 \cdot R'd + 0.587 \cdot G'd + 0.114 \cdot B'd \tag{2.21}$$

$$Cb = 128 - 0.168736 \cdot R'd - 0.331264 \cdot G'd + 0.5 \cdot B'd \tag{2.22}$$

$$Cr = 128 + 0.5 \cdot R'd - 0.418688 \cdot G'd - 0.081312 \cdot B'd \tag{2.23}$$

where $R'd$,$G'd$,$B'd$ is the gamma compressed RGB components in 0,1,2,...,255 and Y',Cb,Cr is in 0,1,2,...,255

2.3.7 Y*CrCb color model

Y*CrCb is the color model used in our system. In order to make color features stable we needed a color space which minimized the variations due to illumination changes. After some trials with the RGB model, which as expected proved quite instable, we decided to use the YCrCb model. YCrCb proved to be better than RGB: the chrominance channels are more resistent to illuminations changes than RGB values. Anyway the histograms computation (which is fundamental for implementing colors features) still stayed computationally expensive, 3D histograms were need to represent the three channels Y Cr Cb. That's why we implemented a new model: Y*CrCB. In Y*CrCb model only two dimensions are needed because the chroma components are multiplied by the luma component, thus keeping the chrominance and the luma information. The channel Y can therefore be neglected.

To convert RGB values we first scale the RGB components to a range of [0,1], than we compute Y, Cr, Cb using the following equations :

$$Y = 0.3 \cdot R + 0.59 \cdot G + 0.11B \tag{2.24}$$

$$Cr = R - Y \tag{2.25}$$

$$Cb = B - Y \tag{2.26}$$

Then we multiply Cr and Cb for Y:

$$YCr = 0.5 \cdot Y \cdot Cr \tag{2.27}$$

$$YCb = 0.5 \cdot Y \cdot Cb \tag{2.28}$$

Now Cr and Cb range from -0.5 to 0.5. We apply a zoom factor to emphasize the values of most interest, which are usually located at the center and we shift and scale to a range of [0,1]

$$Cr' = MIN[1, MAX(0, ZoomFactor \cdot YCr + 0.5)] \tag{2.29}$$

$$Cb' = MIN[1, MAX(0, ZoomFactor \cdot YCb + 0.5)] \tag{2.30}$$

Chapter 3

Tracking methods

3.1 Introduction

Video tracking is the process of locating a moving object (or several ones) in time. It represents a critical task in many application such as surveillance, perceptual user interfaces, augmented reality, smart rooms, object-based video compression and driver assistance. Two major components can be distinguished in a typical visual tracker:

- Target Representation and Localization
- Filtering and Data Association

Target Representation and Localization is mostly bottom-up process, which has also to cope with the changes in the appearance of the target. Usually the target's position is detected through some image processing techniques. Basically the image is segmented into objects which are then used for tracking. *Filtering and Data Association* is mostly a top-down process which is applied to predict and update the target's position. It deals with the dynamics of the tracked object, learning of scene priors, and evaluation of different hypotheses. The way the two components are combined and weighted is application dependent and plays a decisive role in the robustness and efficiency of the tracker.

The most abstract formulation of the filtering and data association process is through the state space approach for modeling discrete-time dynamic systems. At least two models are required: a model describing the evolution of the the state with time (the system model) and a model relating the noisy measurements to the state (the measurement model).

$$x_k = f_k(x_{k-1}, u_k) \tag{3.1}$$

$$z_k = g_k(x_k, v_k) \tag{3.2}$$

where z_k is a vector of observations, x_k is a state vector, g_k is a measurement function, f_k is a system transition function, u_k and v_k are noise vectors. The first equation is known as **state equation** (it represents the system model), and the second, as **measurement equation** (it represents the measurement model). The objective of tracking is to estimate the state x_k given all the measurements $z_{1:k}$ up that moment, or equivalently to construct the probability density function (pdf) $p(x_k|z_{1:k})$. The probabilistic state-space formulation and the requirement for the updating of information on receipt of new measurements are ideally suited for the Bayesian approach. This provides a rigorous general framework for dynamic state estimation problems. For many problems, an estimate is required every time that a measurement is received. In this case, a recursive filter is a convenient solution. A recursive filtering approach means that received data can be processed sequentially rather than as a batch so that it is not necessary to store the complete data set nor to reprocess existing data if a new measurement becomes available. This is achieved using Bayes theorem, which is the mechanism for updating knowledge about the target state in the light of extra information from new data.

In section 3.2 we will briefly describe the nonlinear tracking problem and its optimal Bayesian solution. When the noise vectors u_k and v_k are indipendent and Gaussian and f_k and g_k are linear functions, the optimal solution is provided by the Kalman filter. In this situation, the distributions of interest (prior and posterior) are also Gaussian and the Kalman filter can compute them exactly without approximations. We will discuss Kalman filter in section 3.3. When the function f_k and g_k are not linear, by linearization the Extended Kalman Filter is obtained, described in section 3.4. In section 3.5 we will describe Particle Filters, which are an important alternative to the extended Kalman filter.With particle filtering, continuous distributions are approximated by discrete random measures, which are composed of weighted particles, where the particles are samples of the unknown states from the state space, and the particle weights are "probability masses" computed by using Bayes theory. Finally in section 3.6 we will introduce the problem of multi-target tracking.

3.2 Non linear Bayesian Tracking

To define the problem of tracking, consider the evolution of the state sequence $x_k, k \in N$ of a given target given by [10]:

$$x_k = f_k(x_{k-1}, u_k) \tag{3.3}$$

The objective of tracking is to recursively estimate x_k from measurements

$$z_k = g_k(x_k, v_k) \tag{3.4}$$

From a Bayesian perspective, the tracking problem is to recursively calculate some degree of belief in the state x_k at time k, taking different values, given the data $z_{1:k}$ up to time. Thus, it is required to construct the pdf $p(x_k|z_{1:k})$. It is assumed that the initial pdf $p(x_0|z_0) \equiv p(x_0)$ of the state vector, which is also known as the prior, is available (z_0 being the set of no measurements). Then, in principle, the pdf may be obtained, recursively, in two stages: prediction and update. The prediction step uses the state equation and the already computed pdf of the state at time $t = k - 1$, $p(x_{k-1}|z_{1:k-1})$ to derive the prior pdf of the current state,$p(x_k|z_{1:k-1})$ via the Chapman-Kolmogorov equation:

$$p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1})dx_{k-1} \tag{3.5}$$

Note that in 3.5, use has been made of the fact that $p(x_k|x_{k-1},z_{1:k-1}) = p(x_k|x_{k-1})$ as 3.3 describes a Markov process of order one. Then, the update step employs the likelihood function $p(z_k|x_k)$ of the current measurement to compute the posterior pdf $p(x_k|z_{1:k})$ via Bayes'rule.

$$p(x_k|z_{1:k}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} \tag{3.6}$$

where the normalizing constant

$$p(z_k|z_{1:k}) = \int p(z_k|x_k)p(x_k|z_{1:k-1})dx_k \tag{3.7}$$

depends on the likelihood function $p(z_k|x_k)$ defined by the measurement model 3.4. In the update stage the measurement z_k is used to modify the prior density to obtain the required posterior density of the current state. This recursive propagation of the posterior density is only a conceptual solution in that in general, it cannot be determined analytically. Solutions do exist in a restrictive set of cases, including the Kalman filter, discussed in the next section.

3.3 Kalman Filter

The Kalman filter assumes that the posterior density at every time step is Gaussian and, hence, parameterized by a mean and covariance [10]. If $p(x_{k-1}|z_{1:k-1})$ is Gaussian, it can be proved that $p(x_k|z_{1:k})$ is also Gaussian, provided that certain assumptions hold [2]:

1. u_k and v_k are drawn from Gaussian distributions of known parameters.

2. f_k is a known and linear function of x_{k-1} and u_k.

3. g_k is a known and linear function of x_k and v_k.

That is, 3.3 and 3.4 can be rewritten as:

$$x_k = F_k x_{k-1} + u_k \tag{3.8}$$

$$z_k = G_k x_k + v_k \tag{3.9}$$

F_k and H_k are known matrices defining the linear functions. The covariances of u_k and v_k are, respectively, Q_k and R_k. Here, we consider the case when u_k and v_k have mean and are statistically independent. Note that the system measurement matrices F_k and H_k , as well as noise parameters Q_k and R_k, are allowed to be time variant.

The Kalman Filter algorithm can then be viewed as the following recursive relationship:

$$p(x_{k-1}|z_{1:k-1}) = N(x_{k-1}; m_{k-1|k-1}, Pk-1|k-1) \tag{3.10}$$

$$p(x_k|z_{1:k-1}) = N(x_k; m_{k|k-1}, Pk|k-1) \tag{3.11}$$

$$p(x_k|z_{1:k}) = N(x_k; m_{k|k}, Pk|k) \tag{3.12}$$

where

$$m_{k|k-1} = F_k m_{k-1|k-1} \tag{3.13}$$

$$P_{k|k-1} = Q_k + F_k P_{k-1|k-1} F_k^T \tag{3.14}$$

$$m_{k|k} = m_{k|k-1} + K_k(z_k - G_k m_{k|k-1}) \tag{3.15}$$

$$P_{k|k} = P_{k|k-1} - K_k G_k P_{k|k-1} \tag{3.16}$$

And where $N(x; m, P)$ is a Gaussian density with argument x, mean m, and covariance P, and

$$S_k = G_k P_{k|k-1} G_K^T + R_k \tag{3.17}$$

$$K_k = P_{k|k-1} G_K^T + S_k^{-1} \tag{3.18}$$

are the covariance of the innovation term $z_k - H_k m_{k|k-1}$, and the Kalman gain, respectively. In the above equations, the transpose of a matrix M is denoted by

M^T. This is the optimal solution to the tracking problem, if the (highly restrictive) assumptions hold. The implication is that no algorithm can ever do better than a Kalman filter in this linear Gaussian environment. All the distributions are then described by their means and covariances, and the algorithm remains unaltered, but are not constrained to be Gaussian. Assuming the means and covariances to be unbiased and consistent, the filter then optimally derives the mean and covariance of the posterior. However, this posterior is not necessarily Gaussian, and therefore, if optimality is the ability of an algorithm to calculate the posterior, the filter is then not certain to be optimal.

3.4 Extended Kalman Filter

If 3.3 and 3.4 cannot be rewritten in the form of 3.8 and 3.9 because the functions are nonlinear, then a local linearization of the equations may be a sufficient description of the nonlinearity. The Extended Kalman Filter (EKF) is based on this approximation. $P(x_k|z_{1:k})$ is approximated by a Gaussian

$$p(x_{k-1}|z_{1:k-1}) \approx N(x_{k-1}; m_{k-1|k-1}, Pk-1|k-1) \tag{3.19}$$

$$p(x_k|z_{1:k-1}) \approx N(x_k; m_{k|k-1}, Pk|k-1) \tag{3.20}$$

$$p(x_k|z_{1:k}) \approx N(x_k; m_{k|k}, Pk|k) \tag{3.21}$$

where

$$m_{k|k-1} = f_k(m_{k-1|k-1}) \tag{3.22}$$

$$P_{k|k-1} = Q_{k-1} + \tilde{F}_k P_{k-1|k-1} \tilde{F}_k^T \tag{3.23}$$

$$m_{k|k} = m_{k|k-1} + K_k(z_k - h_k(m_{k|k-1}) \tag{3.24}$$

$$P_{k|k} = P_{k|k-1} - K_k \tilde{G}_k P_{k|k-1} \tag{3.25}$$

and where now f_k and g_k are nonlinear function, and $\tilde{F}_k$ and $\tilde{G}_k$ are local linearizations of these nonlinear functions (i.e.,matrices)

$$\tilde{F}_k = \frac{df_k(x)}{dx} \tag{3.26}$$

where $x = m_{k-1|k-1}$

$$\tilde{G_k} = \frac{dg_k(x)}{dx} \tag{3.27}$$

where $x = m_{k|k-1}$

$$S_k = \tilde{G_k} P_{k|k-1} \tilde{G_k^T} + R_k \tag{3.28}$$

$$K_k = P_{k|k-1} \tilde{G_k^T} + S_k^{-1} \tag{3.29}$$

The EKF as described above utilizes the first term in a Taylor expansion of the nonlinear function. A higher order EKF that retains further terms in the Taylor expansion exists, but the additional complexity has prohibited its widespread use. However, the EKF always approximates $p(x_k|z_{1:k})$ to be Gaussian. If the true density is non-Gaussian (e.g., if it is bimodal or heavily skewed), then a Gaussian can never describe it well. In such cases particle filters will yield an improvement in performance in comparison to that of an EKF.

3.5 Particle Filtering Methods

3.5.1 SIS Particle Filter

The objective of tracking is to estimate the state x_k given all the measurements $z_{1:k}$ up that moment, or equivalently to construct the probability density function (pdf) $p(x_k|z_{1:k})$. In many practical situations, however, the optimal algorithms are impossible to implement, primarily because the distribution updates require integrations that cannot be performed analytically or summations that are impossible to carry out due to the number of terms in the summations. In particle filtering, the distributions are approximated by discrete random measures defined by particles and weights assigned to the particles [8]. If the distribution of interest is $p(x)$and its approximating random measure is:

$$\chi = \{x^{(m)}, w^{(m)}\}_{m=1}^{M} \tag{3.30}$$

where $x^{(m)}$ are the particles, $w^{(m)}$ are their weights and M is the number of particles used in the approximation, χ approximates the distribution $p(x)$ by

$$p(x) = \sum_{m=1}^{M} w^{(m)} \delta(x - x^{(m)}) \tag{3.31}$$

where $\delta(.)$ is the Dirac delta function. The next important concept used in particle filtering is the principle of importance sampling. Suppose we want to approximate a distribution $p(x)$ with a discrete random measure. If we can generate the particles from $p(x)$, each of them will be assigned a weight equal to $1/M$. When direct sampling from $p(x)$ is intractable, one can generate particles $x^{(m)}$ from a distribution $\pi(x)$, known also as importance function, and assign (nonnormalized) weights according to

$$w^{*(m)} = \frac{p(x)}{\pi(x)} \tag{3.32}$$

which upon normalization become

$$w^{(m)} = \frac{w^{*(m)}}{\sum_{i=1}^{M} w^{*(i)}} \tag{3.33}$$

Suppose now that the posterior distribution $p(x_{0:k-1}|z_{0:k-1})$ by the discrete random measure $\chi_{k-1} = \{x_{0:k-1}^{(m)}, w_{0:k-1}^{(m)}\}_{m-1}^{M}$. Note that the trajectories or streams of particles $x_{0:k-1}^{(m)}$ can be considered particles of $p(x_{0:k-1}|z_{0:k-1})$. Given the discrete random measure χ_{k-1} and the observation z_k , the objective is to exploit χ_{k-1} in obtaining χ_k . Sequential importance sampling methods achieve this by generating particles $x_k^{(m)}$ and appending them to $x_{0:k-1}^{(m)}$ to form $x_{0:k}^{(m)}$, and updating the weights $w_k^{(m)}$ so that χ_k allows for accurate estimates of the unknowns of interest at time k. If we use an importance function that can be factored as

$$\pi(x_{0:k}|z_{0:k}) = \pi(x_k|x_{0:k-1}, z_{0:k})\pi(x_{0:k-1}|z_{0:k-1}) \tag{3.34}$$

and if

$$x_{0:k}^{(m)} \sim \pi(x_{0:k-1}|z_{0:k-1}) \tag{3.35}$$

and

$$w_{k-1}^{(m)} \propto \frac{p(x_{0:k-1}^{(m)})|z_{0:k-1})}{\pi(x_{0:k-1}^{(m)}|z_{0:k-1})} \tag{3.36}$$

we can augment the trajectory $x_{0:t-1}^{(m)}$ with $x_t^{(m)}$, where

$$x_k^{(m)} \sim \pi(x_k|x_{0:k-1}^{(m)}, z_{0:k}) \tag{3.37}$$

and easily associate with it an updated weight $w_k^{(m)}$ obtained according to

$$w_k^{(m)} \propto \frac{p(z_k|x_k^{(m)})p(x_k^{(m)})|x_{k-1}^{(m)})}{\pi(x_k^{(m)}|x_{0:k-1}^{(m)}, z_{0:k})} w_{k-1}^{(m)} \tag{3.38}$$

The sequential importance sampling (SIS) algorithm can thus be implemented by performing the following two steps for every k:

- Draw particles $x_k^{(m)} \sim \pi(x_k|x_{k-1}^{(m)},z_{0:k})$, where $m = 1,2,...,M$.
- Compute the weights of $w_k^{(m)}$ according to 3.38.

The importance function plays a very important role in the performance of the particle filter. This function must have the same support as the probability distribution that is being approximated. In general, the closer the importance function to that distribution, the better the approximation is. In the literature, the two most frequently used importance functions are the prior and the optimal importance function. The prior importance function is given by $p(x_k|x_{k-1}^{(m)})$, and it implies particle weight updates by

$$w_k^{(m)} \propto w_{k-1}^{(m)} p(z_k|x_k^{(m)}) \tag{3.39}$$

The optimal importance function minimizes the variance of the importance weights conditional on the trajectory $x_{0:t-1}^{(m)}$ and the observations $z_{0:t}$ and is given by $p(x_t|x_{0,t-1}^{(m)},z_{o:t})$. When the optimal function is used, the update of the weights is carried out according to

$$w_k^{(m)} \propto w_{k-1}^{(m)} p(z_k|x_{k-1}^{(m)}) \tag{3.40}$$

Note that implementations of particle filters with prior importance functions are much easier than those with optimal importance functions. The reason is that the computation of $p(z_t|x_{t-1}^{(m)})$ requires integration.
A common problem with particle filters is the degeneracy phenomenon, where after a few iterations, all but one particle will have negligible weight. It has been shown [11] that the variance of the importance weights can only increase over time, and thus, it is impossible to avoid the degeneracy phenomenon. This degeneracy implies that a large computational effort is devoted to updating particles whose contribution to the approximation to $p(x_k|z_{1:k})$is almost zero. A suitable measure of degeneracy of the algorithm is the effective sample size N_{eff}, defined as

$$N_{eff} = \frac{N_s}{1 + Var(w_k^{*(m)})} \tag{3.41}$$

where , $w_k^{(*m)} = p(x_k^{(m)}|z_{1:k})/\pi(x_k^{(m)}|x_{k-1}^{(m)},z_k)$ is referred to as the "true weight" This cannot be evaluated exactly, but an estimate $\widehat{N_{eff}}$ of N_{eff} can be obtained by

$$\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_s}(w_k^{(m)})^2} \tag{3.42}$$

Notice that $N_{eff} < N_s$, and small N_{eff} indicates severe degeneracy. Clearly, the degeneracy problem is an undesirable effect in particle filters. The brute force approach to reducing its effect is to use a very large N_s. This is often impractical;

therefore, we rely on two other methods: a) good choice of importance density and b) use of resampling.

Good Choice of Importance Density: The first method involves choosing the importance density $\pi(x_k|x_{k-1}^{(m)}, z_k)$ to minimize $Var(w_k^{*(m)})$ so that N_{eff} is maximized.

This optimal importance density suffers from two major drawbacks. It requires the ability to sample from $\pi(x_k|x_{k-1}^{(m)}, z_k)_{opt}$ and to evaluate the integral over the new state. In the general case, it may not be straightforward to do either of these things. There are two cases when use of the optimal importance density is possible.

Resampling is a scheme that eliminates particles with small weights and replicates particles with large weights. In principle, it is implemented as follows:

- Draw M particles, $x_k^{*(m)}$ from the discrete distribution χ_k.
- Let $x_k^{(m)} = x_k^{*(m)}$, and assign equal weights $(1/N_s)$ to the particles.

The resampling step involves generating a new set $\{x_k^{(m)*}\}_{i=1}^{N_s}$ by resampling (with replacement) N_s times from an approximate discrete representation of given by

$$p(x_k|z_{1:k}) \approx \sum_{m=1}^{N_s} w_k^{(m)} \delta(x_k - x_k^{(m)}) \tag{3.43}$$

so that $Pr(x_k^{(m)*} = x_k^{(j)}) = w_k^{(j)}$. The resulting sample is in fact an i.i.d. sample from the discrete density 3.43; therefore, the weights are now reset to $w_k^{(m)} = 1/N_s$.

The idea of resampling is depicted in Figure 3.1 [8] with $M = 10$ particles. There, the left column of circles represents particles before resampling, where the diameters of the circles are proportional to the weights of the particles. The right column of circles are the particles after resampling. In general the large particles are replicated and the small particles are removed. For example, the ”blue” particle with the largest weight is replicated three times and the ”yellow” particle, two times, whereas the green particles, which have small weights, are removed. Also, after resampling all the circles have equal diameters, that is, all the weights are set to $1/M$.

In Figure 3.2 it's represent pictorially the random measures and the actual probability distributions of interest as well as the three steps of particle filtering: particle generation, weight update, and resampling. In the figure, the solid curves represent the distributions of interest, which are approximated by the discrete measures. The sizes of the particles reflect the weights that are assigned to them. Finally in Figure 3.3, it's displayed a flowchart that summarizes the particle filtering algorithm. At time k, a new set of particles is generated, and their weights are computed. Thereby we obtain the random measure χ_k , which can be used for estimation of the desired unknowns. Before proceeding with the generation of the set of particles for time

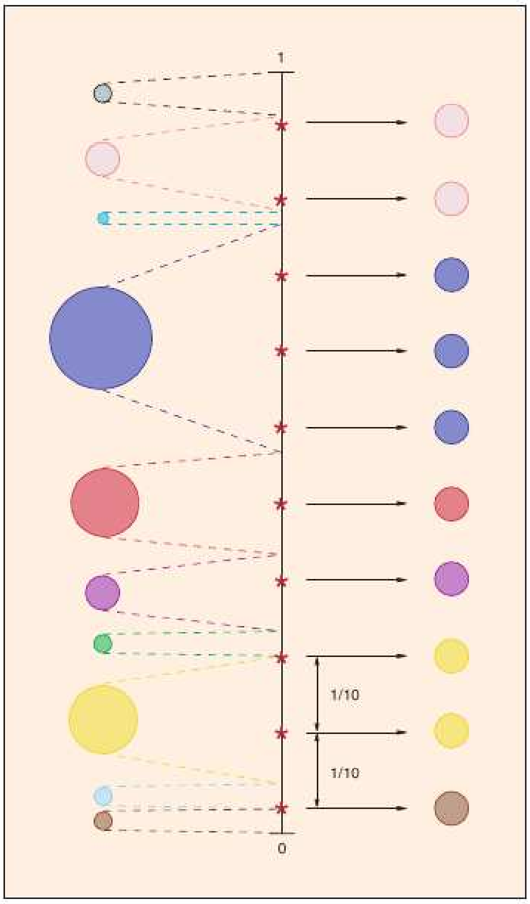


Figure 3.1. A schematic description of resampling.

instant $k+1$, we estimate the effective sample size N_{eff}. If the effective particle size is below a predefined threshold, resampling takes place; otherwise we proceed with the regular steps of new particle generation and weight computation.

3.5.2 SIR Particle Filter

The SIR filter proposed in is an MC method that can be applied to recursive Bayesian filtering problems. The assumptions required to use the SIR filter are very weak. The state dynamics and measurement functions need to be known, and it is required to be able to sample realizations from the process noise distribution of v_{k-1} and from the prior. Finally, the likelihood function $p(z_k|x_k)$ needs to be available for pointwise evaluation (at least up to proportionality). The SIR algorithm can be easily derived from the SIS algorithm by an appropriate choice of i) the importance density, where

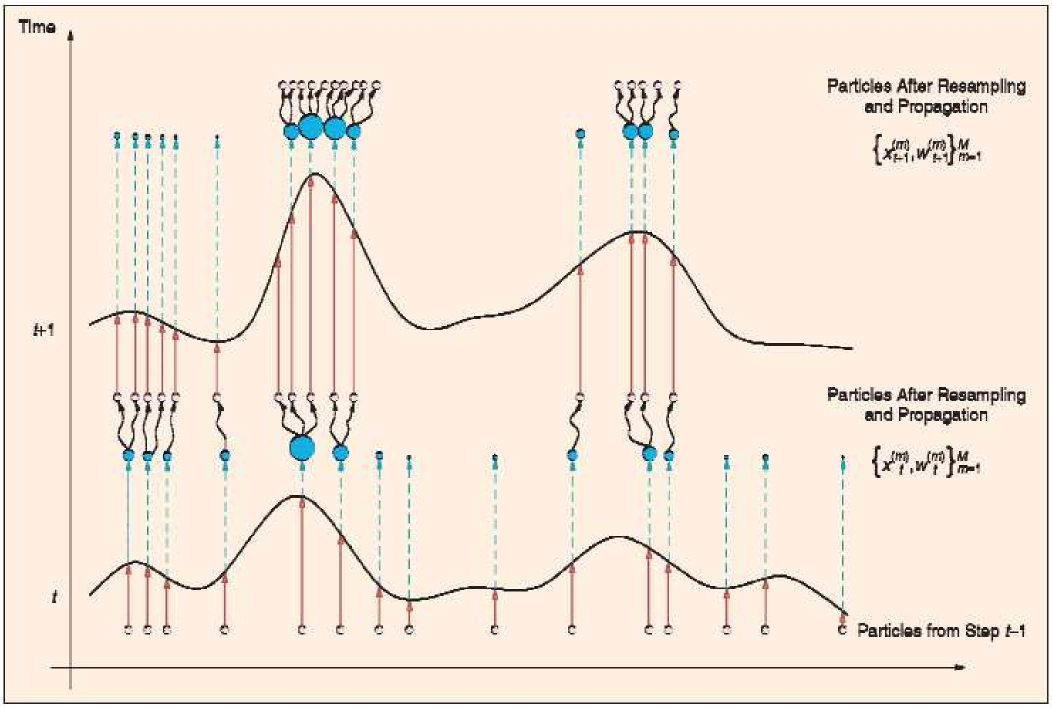

Time
t+1
t
Particles After Resampling and Propagation
$\{x_{t+1}^{(m)}, w_{t+1}^{(m)}\}_{m=1}^{M}$
Particles After Resampling and Propagation
$\{x_{t}^{(m)}, w_{t}^{(m)}\}_{m=1}^{M}$
Particles from Step t-1

Figure 3.2. A pictorial description of particle filtering.

$\pi(x_k|x_{k-1}^{(m)},z_{k-1})$ is chosen to be the prior density $p(x_k|x_{k-1}^{(m)})$, and ii) the resampling step, which is to be applied at every time index. The above choice of importance density implies that we need samples from $p(x_k|x_{k-1}^{(m)})$. A sample $x_k^{(m)} \sim p(x_k|x_{k-1}^{(m)})$ can be generated by first generating a process noise sample $v_{k-1}^{(m)} \sim p_v(v_{k-1})$ and setting , $x_k^{(m)} = f_k(x_{k-1}^{(m)},v_{k-1}^{(m)})$, where $p_v(\cdot)$ is the pdf of v_{k-1}. For this particular choice of importance density, it is evident that the weights are given by

$$w_k^{(m)} \propto w_{k-1}^{(m)} p(z_k|x_k^{(m)}) \tag{3.44}$$

However, noting that resampling is applied at every time index, we have $w_{k-1}^{(m)} = 1/N$ $\forall\ m$ therefore:

$$w_k^{(m)} \propto p(z_k|x_k^{(m)}) \tag{3.45}$$

The weights given by the proportionality are normalized before the resampling stage. As the importance sampling density for the SIR filter is independent of measurement z_k , the state space is explored without any knowledge of the observations. Therefore, this filter can be inefficient and is sensitive to outliers. Furthermore, as resampling is applied at every iteration, this can result in rapid loss of diversity in

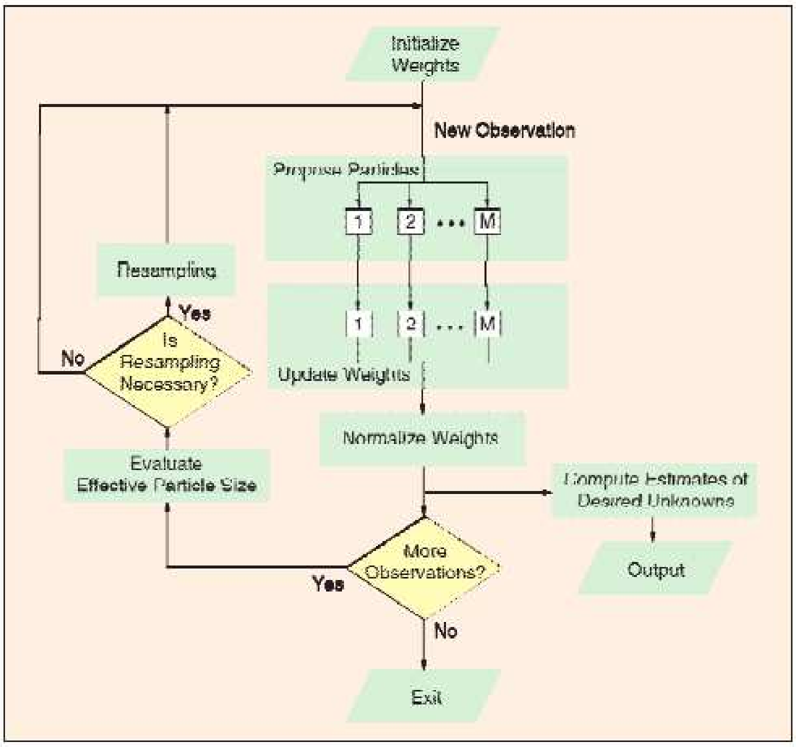


Figure 3.3. A block diagram of particle filtering.

particles. However, the SIR method does have the advantage that the importance weights are easily evaluated and that the importance density can be easily sampled.

3.6 Multi - Target Tracking: The Joint Particle Filter

This section we describe how multiple targets can be modeled using a joint particle tracker. We then show how to approximate this correct but intractable joint filter with multiple nearly independent trackers.

3.6.1 The Joint Particle Filter

A joint particle filter is quite similar to a simple particle filter. The primary difference is that each "particle" estimates the locations of all the targets being tracked [12]. An important consequence of the joint representation is that each particle is d^n - dimensional, where d is the dimensionality of an individual filter and n the number of targets. As a consequence, if N particles are necessary for reliable tracking of a single target, N^n are typically required for tracking n target. Tracking multiple targets with joint hypotheses is an exponentially complex problem both in terms of space and computational time. First, we assume that the joint appearance likelihood $P(Z_t|X_t)$ factors over the different targets X_{it} , with $i \in 1,...,n$:

$$p(Z_t|X_t) = \prod_{i=1}^{n} P(Z_{it}|X_{it}) \tag{3.46}$$

This assumes targets will not occlude each other. This assumption is in fact not needed for the joint tracker, but is crucial in approximating the joint tracker with n independent trackers.

The benefit of using a joint tracker is that we can model non trivial interactions between targets at the motion model stage, i.e. intelligent targets that take account of each other's position in order to avoid collisions. If the targets do not interact at all, the joint motion model $P(X_t|X_{t-1})$ also factors over all targets as $\prod_j P(X_{jt}|X_{j(t-1)})$, and one could just use n independent particle filters. Let's model the interaction between targets by a Markov random field (MRF) constructed on the fly for the current time-slice. An MRF is a graph (V,E) with undirected edges between nodes where the joint probability is factored as a product of local potential functions at each node, and interactions are defined on neighborhood cliques. The most commonly used form is a pairwise MRF, where the cliques are pairs of nodes that are connected in the undirected graph. We will assume the following pairwise MRF form:

$$P(X_t|X_{t-1}) \propto \prod_i P(X_{it}|X_{i(t-1)}) \prod_{i,j \in E} \psi(X_{it},X_{jt}) \tag{3.47}$$

where the $\psi(X_{it},X_{jt})$ are pairwise *interaction potentials.* Since it is easier to specify the interaction potential in the log- domain, we express $\psi(X_{it},X_{jt})$ by means of the Gibbs distribution:

$$\psi(X_{it},X_{jt}) \propto exp(-g(X_{it},X_{jt})) \tag{3.48}$$

where $g(X_{it},X_{jt})$ is a penalty function. Joint particles where some targets overlap the location of another target are penalized. The penalty function $g(X_{it},X_{jt})$ depends on the distance $d(X_{it},X_{jt})$ between two targets, is maximal when two targets coincide, and gradually falls off as targets move apart. This models the fact that targets will not overlap, as is appropriate in a tracking application. Note that the interaction potential 3.48 does not depend on the previous target state X_{t-1} , and hence the target posterior distribution for the joint MRF filter factors is

$$P(X_t|Z_t) = kP(Z_t|X_t) \prod_{i,j \in E} \psi(X_{it},X_{jt}) \sum_r q_{t-1}^{(r)} \prod_i P(X_{it}|X_{i(t-1)}^{(r)}) \tag{3.49}$$

where $q_t^{(r)} = P(Z_t|X_t^{(r)})$.

We note that the interaction term moves out of the mixture distribution. This means that we can simply treat the interaction term as an additional factor in the importance weight. In other words, we sample from the joint proposal distribution function

$$\pi(X_t) = \sum_r q_{t-1}^{(r)} \prod_i P(X_{it}|X_{i(t-1)}^{(r)}) \tag{3.50}$$

by drawing a mixture component at random, and then moving each individual target independently. Then, we weight each of the particles $X_t^{(s)}$ so obtained by

$$q_t^{(s)} = \prod_{i=1}^{n} P(Z_{it}|X_{it}^{(s)}) \prod_{i,j\in E} \psi(X_{it}^{(s)},X_{jt}^{(s)}) \tag{3.51}$$

Importance sampling is notoriously inefficient in high- dimensional state spaces: if not enough particles are used, all but a few particles will have a near zero-weight. As a result, the Monte Carlo approximation for the posterior, while asymptotically unbiased, will have high variance. N nearly independent trackers, where n is the number of targets, may provide trajectories nearly as well as the optimal joint tracker, while requiring substantially less computation. The trackers are not fully independent, because they consider the locations of other targets when scoring particles. In particular, we penalize particles that overlap the locations of other targets (as described above for the joint tracker). The scoring in the independent trackers is a simplification of the scoring in the joint tracker, as follows:

$$q_{it}^{(r)} = P(Z_{it}|X_{it}^{(r)}) \prod_{i,j\in E} \psi(X_{it}^{(r)},\bar{X}_{j(t-1)}) \tag{3.52}$$

where $\bar{X}_{j(t-1)}$ is the estimated state for target j at time $t-1$, i.e. the mean of the particles of the j^{th} tracker.

Chapter 4

System overview

In this chapter we will describe the method for 3D tracking of multiple people in a multi-camera environment using color features we implemented at the UPC. For a given frame in the video sequence, a set of N images are obtained from the N cameras. Foreground regions from input images are labelled using a segmentation algorithm based on Stauffer-Grimson's background learning and substraction technique, illustrated in section 4.1. Redundancy among cameras is exploited by means of a Shape-from-Silhouette (SfS) technique to generate a 3D reconstruction of the scene as illustrated in section 4.2. After generating the 3D data (voxels) a coloring algorithm is applied to assign them a color (section 4.2.1). Finally we'll describe two tracking algorithms: one based on particle filtering (just for comparison reasons) and one based on sparse sampling, with and without using color features, that is the main topic of this work.

4.1 Background learning: Stauffer-Grimson's technique

The video sequences we used to implement and test our system are recorded in smart rooms. Smart rooms are rooms provided with cameras, microphones,and other sensors which use these inputs to try to interpret what people are doing. An example can be seen in figure 4.1, where the smart room of the UPC is shown. For a given frame in the video sequence a set of N images are obtained from the N cameras (see Figure 4.2).

Then a segmentation algorithm based on Stauffer-Grimson's background learning technique is performed, in order to detect the foreground regions [4]. The Stauffer-Grimson's technique rather than explicitly modeling the values of all the pixels as one particular type of distribution, simply model the values of a particular pixel as a mixture of Gaussians. Based on the persistence and the variance of each of

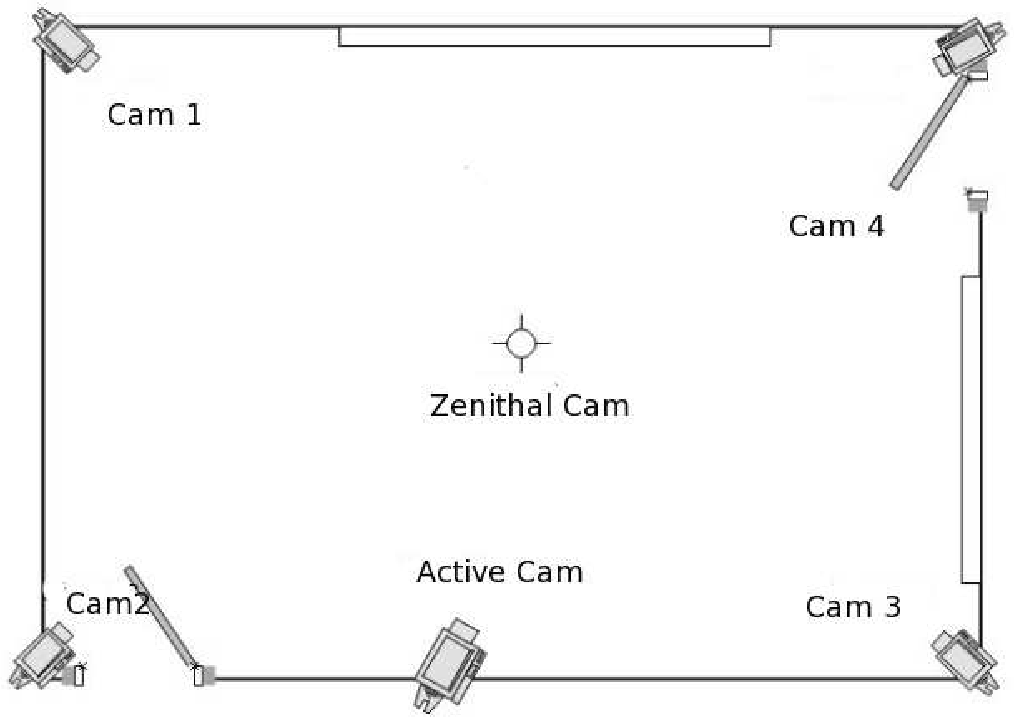


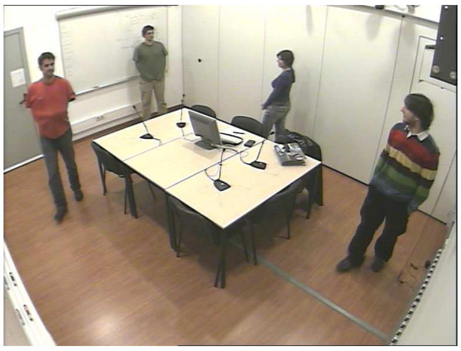
Figure 4.1. UPC Smart Room.

the Gaussians of the mixture, it determines which Gaussians may correspond to background colors. Pixel values that do not fit the background distributions are considered foreground until there is a Gaussian that includes them with sufficient, consistent evidence supporting it. The values of a particular pixel over time as considered as "pixel process". The "pixel process" is a time series of pixel values, e.g. scalars for grayvalues or vectors for color images. At any time, t, what is known about a particular pixel, $\{x_0,y_0\}$, is its history:

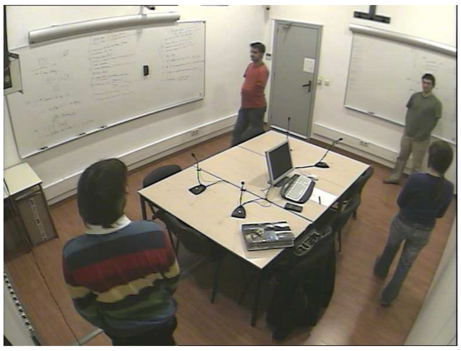
$$\{X_1,...,X_t\} = \{I(x_0,y_0,i) : 1 \leq i \leq t\} \tag{4.1}$$

where I is the image sequence.

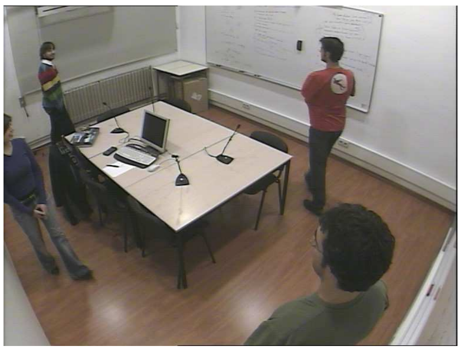
The value of each pixel represents a measurement of the radiance in the direction of the sensor of the first object intersected by the pixel's optical ray. With a static background and static lighting, that value would be relatively constant. If we assume that independent, Gaussian noise is incurred in the sampling process, its density could be described by a single Gaussian distribution centered at the mean pixel value. Unfortunately, the most interesting video sequences involve lighting changes, scene changes, and moving objects. If lighting changes occurred in a static



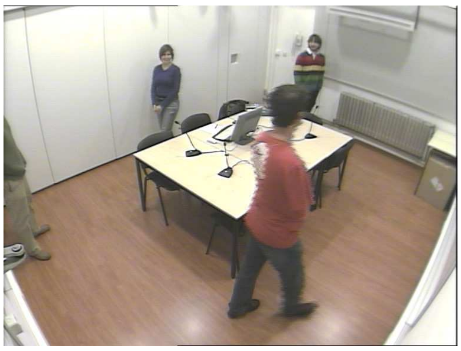
(a) *Camera 1*



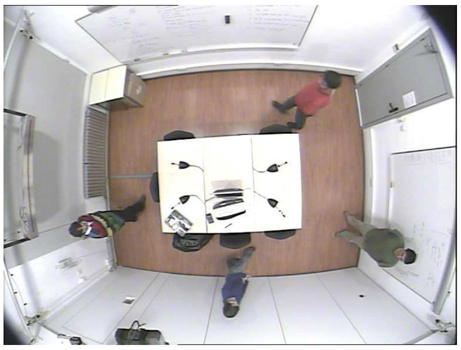
(b) *Camera 2*



(c) *Camera 3*



(d) *Camera 4*



(e) *Camera 5*

Figure 4.2. Images recorded by the 5 cameras at the UPC.

scene, it would be necessary for the Gaussian to track those changes. An additional aspect of variation occurs if moving objects are present in the scene. Even a relatively consistently colored moving object is generally expected to produce more variance than a static object. Also, in general, there should be more data supporting the background distributions because they are repeated, whereas pixel values for different objects are often not the same color. These are the guiding factors in our choice of model and update procedure. The recent history of each pixel, $\{X1,...,Xt\}$, is modeled by a mixture of K Gaussian distributions. The probability of observing the current pixel value is:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(X_t,\mu_{i,t},\Sigma_{i,t}) \tag{4.2}$$

where K is the number of distributions, $\omega_{i,t}$ is an estimate of the weight (what portion of the data is accounted for by this Gaussian) of the i^{th} Gaussian in the mixture at time t, $\mu_{i,t}$ is the mean value of the i^{th} Gaussian in the mixture at time t,$\Sigma_{i,t}$ is the covariance matrix of the i^{th} Gaussian in the mixture at time t, and where $\eta(X_t,\mu_{i,t},\Sigma_{i,t})$ is a Gaussian probability density function. K is determined by the available memory and computational power. Every new pixel value, X_t, is checked against the existing K Gaussian distributions, until a match is found. A match is defined as a pixel value within 2.5 standard deviations of a distribution. If none of the K distributions match the current pixel value, the least probable distribution is replaced with a distribution with the current value as its mean value, an initially high variance, and low prior weight.
One of the significant advantages of this method is that when something is allowed to become part of the background, it doesn't destroy the existing model of the background. The original background color remains in the mixture until it becomes the K^{th} most probable and a new color is observed. Therefore, if an object is stationary just long enough to become part of the background and then it moves, the distribution describing the previous background still exists with the same μ and σ^2, but a lower ω and will be quickly re-incorporated into the background. As the parameters of the mixture model of each pixel change, it's important to determine which of the Gaussians of the mixture are most likely produced by background processes. Heuristically, these are the Gaussian distributions which have the most supporting evidence and the least variance. To understand that, consider the accumulation of supporting evidence and the relatively low variance for the background distributions when a static, persistent object is visible. In contrast, when a new object occludes the background object, it will not, in general, match one of the existing distributions which will result in either the creation of a new distribution or the increase in the variance of an existing distribution. Also, the variance of the moving object is expected to remain larger than a background pixel until the moving object stops.

To model this, a method is needed for deciding what portion of the mixture model best represents background processes. First, the Gaussians are ordered by the value of $\omega\sigma$. This value increases both as a distribution gains more evidence and as the variance decreases. After re-estimating the parameters of the mixture, it is sufficient to sort from the matched distribution towards the most probable background distribution, because only the matched models relative value will have changed. This ordering of the model is effectively an ordered, open-ended list, where the most likely background distributions remain on top and the less probable transient background distributions gravitate towards the bottom and are eventually replaced by new distributions. Then the first B distributions are chosen as the background model, where

$$B = argmin_b \sum_{k=1}^{b} \left(\omega_k > T \right) \tag{4.3}$$

where T is a measure of the minimum portion of the data that should be accounted for by the background.

4.2 Voxel generation

After labelling the foreground region in the N images using the Stauffer Grimson's technique, the redundancy among cameras is exploited to generate voxels thought the Shape from Silhouette method [5]. The whole volume of interest, that is the whole room, is divided into equal sized 3D voxels. Each voxel v is projected onto the N images and the algorithm tests if these projections belong to pixel labelled as foreground during the background learning. Assuming that the camera performs a exact perspective projection the image formation process can be expressed as a projective mapping from a 3D system to a 2D system (see figure 4.3). The projection equations can be written as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \frac{fX}{Z} \\ \frac{fY}{Z} \end{pmatrix} \tag{4.4}$$

Every point in an image represents a possible line of sight of an incoming light ray: any 3D point along the ray projects to the same image point, so only the direction of the ray is relevant, not the distance of the point along it. To represent the directions of the rays we could use the coordinates (x',y') of the pixel of the projection. Another way is by arbitrarily choosing some 3D point along each ray to represent the ray's direction. In this case we need three homogeneous coordinates instead of two inhomogeneous ones to represent each ray. This seems inefficient, but it has the significant advantage of making the image projection process much easier

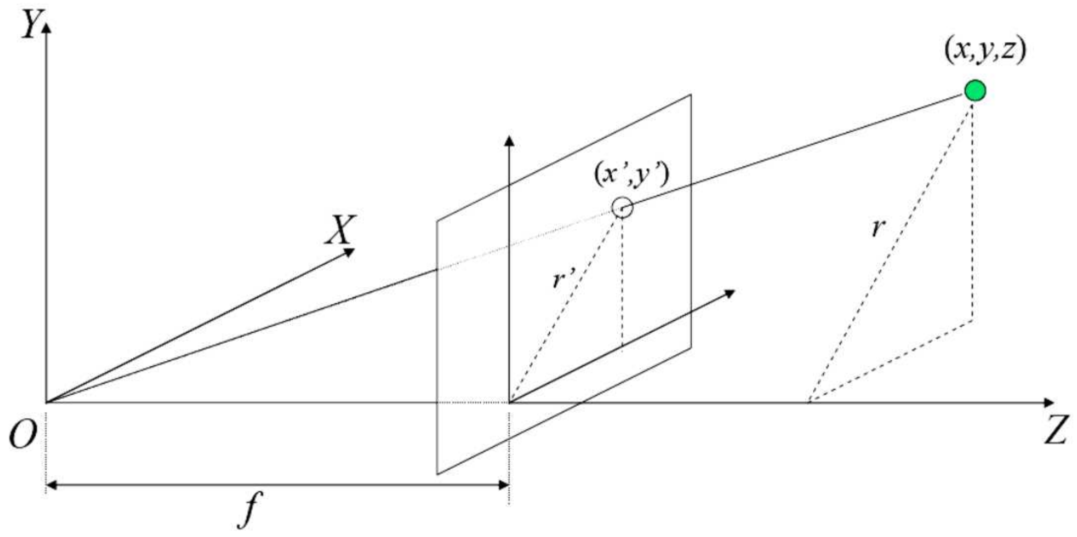


Figure 4.3.

to deal with. If we suppose that the image plane of the camera is $z' = 1$, the ray through pixel (x',y') can be represented homogeneously:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \frac{fX}{Z} \\ \frac{fY}{Z} \\ 1 \end{pmatrix} \tag{4.5}$$

Using equation 4.5 all points in the 3D world have been represented in a *camera centered* coordinate system, that is, a coordinate system which has its origin at the camera focal point. In practice however, the 3D points may be represented in terms of coordinates relative to an arbitrary coordinate system $\hat{X} = [X',Y',Z']$

$$X = R\hat{X} + T \tag{4.6}$$

where X is the coordinates vector referred to the camera centered system, R is a rotation matrix and T is a translation matrix.

The voxels represent a 3D discrete grid, that's why the projection operation needs an algorithm to approssimate a line in a 3D discrete grid, as showed in Figure 4.4.

We use an adaptation of the Bresenham Line-Drawing Algorithm, see Appendix A for more details.

Using equation 4.6 each voxel v is projected onto the N images. Denoting with $i_k = Proj(v)_k$ the projection of voxel v on the k^{th} image. Also let $Is\_Foreground(i_k)$

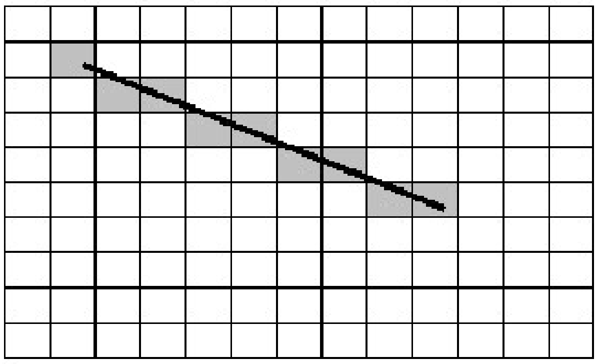


Figure 4.4. Line drawn on a 3D grid.

be the function that returns TRUE if i_k was labelled as foreground pixel and FALSE otherwise. Moreover, the set of all voxels which belong to the foreground objects is denoted by INSIDE, while OUTSIDE represents the set of voxels which are not occupied by the foreground objects. A voxel v is classified by using the algorithm 1.

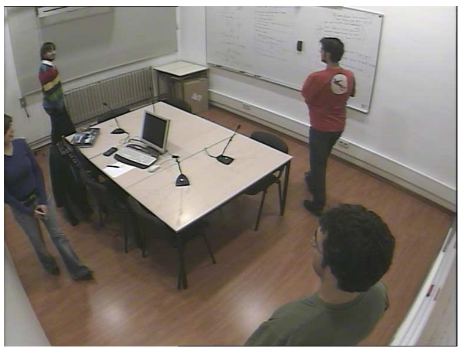
Algorithm 1 Voxel Extraction

```
for $v = 1$ to $NVoxel$ do
  for $k = 1$ to $N$ do
    $i_k = Proj(v)_k$
    if Is_Foreground($i_k$) is true then
      count++
    end if
  end for
  if count$\geq N - 1$ then
    $v \in$ INSIDE
  else
    $v \in$ OUTSIDE
  end if
end for
where $NVoxel$ is the number of voxel of the room.
```

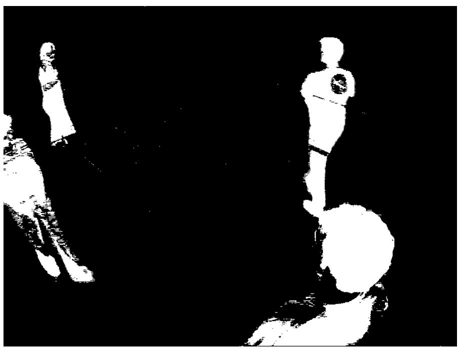
In figure 4.5 the whole process can be seen.

4.2.1 Voxel coloring

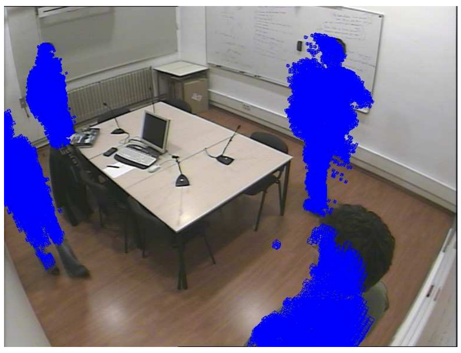
After the volume reconstruction it is fundamental to add the color information, which is not so trivial. We define photorealistic a 3D reconstructions of real scenes



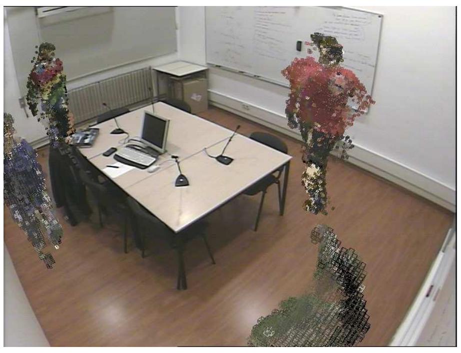
(a)



(b)



(c)



(d)

Figure 4.5. Voxel extraction: (a) frame processed, (b) background learning, (c) voxel generation, (d) voxel coloring

whose reprojections contain sufficient color and texture information to accurately reproduce images of the scene from a broad range of target viewpoints. To ensure accurate reprojections, the input images should be representative, i.e., sparsely distributed throughout the target range of viewpoints. Seitz and Dyer proposed two criteria that a photorealistic reconstruction technique should satisfy [6]:

1. **Photo Integrity**: The reprojected model should accurately reproduce the input images, preserving color, texture and pixel resolution
2. **Broad Viewpoint Coverage**: Reprojections should be accurate over a large range of target viewpoints. This requires that the input images are widely distributed about the environment

The voxel coloring problem is to assign colors (radiances) to voxels (points) in a 3D volume so as to maximize *photo integrity* with a set of input images. That is, rendering the colored voxels from each input viewpoint should reproduce the original image as closely as possible. In order to solve this coloring problem we must consider the following two issues:

1. Uniqueness: Multiple voxel colorings can be consistent with a given set of images. How can the problem be well-defined?
2. Computation: How can a voxel coloring be computed from a set of input images?

Assuming that both the scene and lighting are stationary and that surfaces are approximately Lambertian. Under these conditions, the radiance at each point is isotropic and can therefore be described by a scalar value which we call color. A 3D scene S is represented as a finite set of opaque voxels (volume elements), each of which occupies a finite homogeneous scene volume and has a fixed color. We denote the set of all voxels with the symbol V. An image is specified by the set I of all its pixels. For now, assume that pixels are infinitesimally small. Given an image pixel p and scene S, we refer to the voxel $V \in S$ that is visible and projects to p by $V = S(p)$. The color of an image pixel $p \in I$ is given by $color(p,I)$ and of a voxel V by $color(V,S)$. A scene S is said to be complete with respect to a set of images if, for every image I and every pixel $p \in I$, there exists a voxel $V \in S$ such that $V = S(p)$. A complete scene is said to be *consistent* with a set of images if, for every image I and every pixel $p \in I$,

$$color(p,I) = color(S(p),S) \tag{4.7}$$

Assuming a pinhole perspective projection model, let P and Q be scene points and I be an image from a camera centered at C. We say P occludes Q if P lies on the line segment $\bar{CQ}$. We require that the input cameras are positioned so as to satisfy the following constraint:

1. **Ordinal visibility constraint**: There exists a norm $\| \cdot \|$ such that for all scene points P and Q, and input images I, P occludes Q in I only if $\| P \| < \| Q \|$.

We call such a norm occlusion-compatible. For some camera configurations, it is not possible to define an occlusion-compatible norm. However, a norm does exist for a broad range of practical configurations. For instance, suppose the cameras are distributed on a plane and the scene is entirely below that plane. For every such viewpoint, the relative visibility of any two scene points depends entirely on which point is closer to the plane, so we may define $\| \cdot \|$ to be distance to the plane. More generally, the ordinal visibility constraint is satisfied whenever no scene point is contained within the convex hull C of the camera centers. Here we use the occlusion-compatible norm $\| P \|_C$, defined to be the Euclidean distance from P to C. For convenience, C is referred to as the camera volume.

The ordinal visibility constraint provides a depthordering of points in the scene. We now describe how this ordering can be used in scene reconstruction. Scene reconstruction is complicated by the fact that a set of images can be consistent with more than one rigid scene. Determining a scene's spatial occupancy is therefore an illposed task because a voxel contained in one consistent scene may not be contained in another. (see Fig. 4.6). Alternatively, a voxel may be part of two consistent scenes, but have different colors in each (Fig. 4.7).

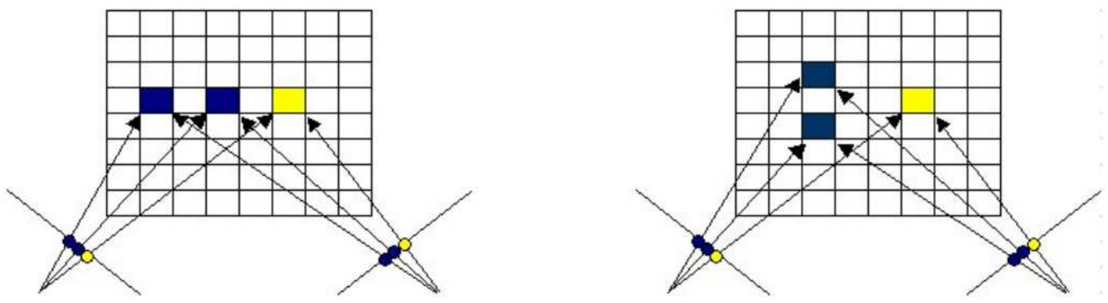


Figure 4.6. Shape ambiguity: the two scene have no point in common.

A voxel V is a color invariant with respect to a set of images if, for every pair of scenes S and S' consistent with the images, $V \in S \cap S'$ implies $color(V,S) = color(V,S')$. Let $I_1,...,I_n$ be a set of images. For a given image point $p \in I_j$ define V_p to be the voxel in $\{S(p)|Sconsistent\}$ that is closest to the camera volume. V_p

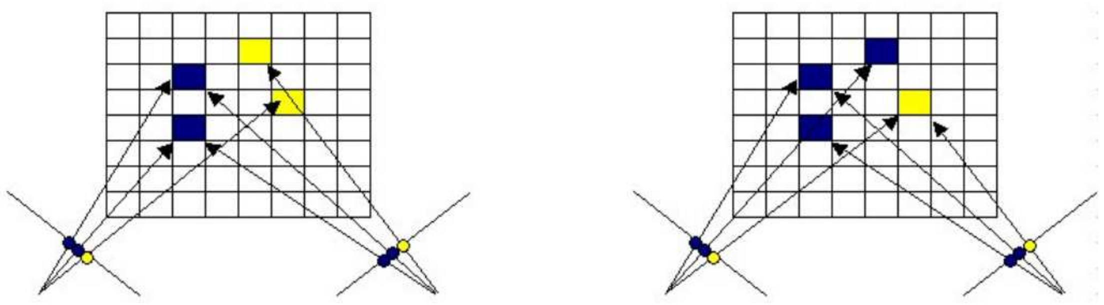


Figure 4.7. Color ambiguity: the two scene share a point that has different color assignment.

is a color invariant. To establish this, observe that $Vp \in S$ implies $Vp = S(p)$, for if $V_p \neq S(p)$, S(p) must be closer to the camera volume, which is impossible by the definition of V_p. It follows from Eq. 4.7 that V_p has the same color in every consistent scene; V_p is a color invariant. The **voxel coloring** of an image set $I_1, ..., I_n$ is defined to be:

$$\bar{S} = \{V_p | p \in I_i, i \leq i \leq m\} \tag{4.8}$$

The voxel coloring is computed one voxel at a time in an order that ensures agreement with the images at each step, guaranteeing that all reconstructed voxels satisfy Eq. 4.7.
To demonstrate that voxel coloring form consistent scenes, it's needed to show that they are complete, i.e. they account for every image pixel. In order to make sure that the construction is incrementally consistent, i.e., agrees with the images at each step, a weaker form of consistency that applies to incomplete voxel sets is needed. Accordingly, we say that a set of voxels with color assignments is *voxel-consistent* if its projection agrees fully with the subset of every input image that it overlaps. More formally, a set S is said to be voxel-consistent with images $I_1, ..., I_n$ if for every voxel $V \in S$ and image pixels $p \in I_i$ and $q \in I_j$, $V = S(p) = S(q)$ implies $color(p, I_i) = color(q, I_j) = color(V, S)$. For notational convenience, define S_V to be the set of all voxels in S that are closer than V to the camera volume. Scene consistency and voxel consistency are related by the following properties:

1. If S is a consistent scene then $\{V\} \bigcup S_V$ is a voxel-consistent set for every $V \in S$.

2. Suppose S is complete and, for each point $V \in S$, $V \bigcup S_V$ is voxelconsistent. Then S is a consistent scene.

Defining the following partition of 3D space into voxel layers of uniform distance from the camera volume:

$$V_C^d = \{V|\, \|V\|_C = d\} \tag{4.9}$$

$$V = \bigcup_{i=1}^{r} V_c^{d_i} \tag{4.10}$$

where $d_1; ...; d_r$ is an increasing sequence of numbers. The voxel coloring is computed inductively as follows:

$$SP_1 = \{V|V \in V_{d1}, \{V\} voxel - consistent\} \tag{4.11}$$

$$SP_k = \{V|V \in V_{d1} \bigcup SP_{k-1} voxel - consistent\} \tag{4.12}$$

$$SP = \{V|V = SP_r(p) for some pixel p\} \tag{4.13}$$

To prove that $SP = \bar{S}$ first we define $\bar{S}_i = \{V|V \in \bar{S}, \|V\|_C \leq d\ \}$. $\bar{S}_1 \subseteq SP_1$‘ by the first consistency propriety. Inductively assume that $\bar{S}_{k-1} \subseteq SP_{k-1}$ and let $V \in \bar{S}_k$. By the first consistency property, $\{V\} \cup \bar{S}_{k-1}$ is voxel-consistent, because the second set includes the first and SP_{k-1} is itself voxel-consistent. It follows that $\bar{S} \subseteq SP_r$. SP_r is complete, since one of its subsets is complete, and hence consistent by the second consistency property. SP contains all the voxel in SP_r that are visible in any image, and is therefore consistent as well. Therefore SP ia s consistent scene such that for each pixel p, $SP(p)$ is at least as close to C as $\bar{S}(p)$. Hence $SP = \bar{S}$. In summary the following properties of voxel coloring have been shown:

- $\bar{S}$ is a consistent scene
- Every voxel in $\bar{S}$ is a color invariant
- $\bar{S}$ is computable from any set of images satisfying the ordinal visibility constraint

If a voxel V is not fully occluded in image I_j, its projection will overlap a nonempty set of image colored pixels, π_j. We denote with $\pi_j = Proj(V)_j$ the projection of voxel V on the I_j image and define the function $is\_Foreground(\pi_j)$, that returns TRUE if π_j was labelled as foreground and FALSE otherwise, the function $Saturation()$ that returns the saturation value of pixel o or a voxel and the function $assign\_color()$ that assign a color to a voxel. Without noise or quantization effects, a consistent voxel should project to a set of pixels with equal color values. In presence of these effects we should measure somehow the likelihood of voxel consistency. This is done using algorithm 2. Basically we project the voxel into the N images, we count in how many images the voxel is projected in a pixel labelled as foreground, if this number is $\geq N-1$ we take the voxel into account and we assign it the color of the most saturated pixel. This choice is motivated by the fact that we need colors different from each other and quite brilliant to perform a good tracking and taking the most saturated pixel assures that.

Algorithm 2 Voxel Coloring Algorithm

```
for $V = 1$ to $NVoxel$ do
  for $j = 1$ to $N$ do
    $\pi_j = Proj(V)_j$
    if $Is\_Foreground(\pi_j)$ is true then
      count++
    end if
  end for
  if count$\geq N-1$ then
    for $j = 1$ to $N$ do
      if $Saturation(\pi_j) > Saturation(V)$ then
        $assign\_color(V) = \pi_j$
      end if
    end for
  else
    $V \in$ NOCOLORED
  end if
end for
where $NVoxel$ is the number of voxel of the room and $N$ the number of the cameras.
```

4.3 Tracking

The objective of tracking is to estimate the state x_k given all the measurements $z_{1:k}$ up that moment, or equivalently to construct the probability density function (pdf) $p(x_k|z_{1:k})$. One of the best known tracking method is particle filtering, which approximate the distributions of interest by means of discrete random measures defined by particles and weights assigned to the particles. We will illustrate the tracking system implemented at the UPC using particle filtering in section 4.3.1, just to compare it with the tracking system we used, based on sparse sampling. Sparse sampling is a new tracking algorithm, developed at the UPC to reduce the computational effort of the tracking system and will be described in detail in section 4.3.3.

4.3.1 Particle Filtering

Particle Filtering (PF) is an approximation technique for estimation problems where the variables involved do not hold Gaussianity uncertainty models and linear dynamics. The tracking scenario can be tackled by means of this algorithm to estimate the 3D position of a person $x_t = (x,y,z)_t$ at time t, taking as observation a set of colored voxels representing the 3D scene up to time t denoted as $z_{1:t}$. Multiple people might be tracked assigning a PF to each target and defining an interaction model to ensure track coherence. For a given target x_t, PF approximates the posterior density $p(x_t|z_{1:t})$ with a sum of N_s Dirac functions:

$$p\left(x_t|z_{1:t}\right) \approx \sum_{j=1}^{N_s} w_t^j \delta(x_t - x_t^j), \tag{4.14}$$

where w_t^j are the weights associated to the particles and x_t^j their positions. For this type of tracking problem, a Sampling Importance Re-sampling (SIR) PF is applied to drive particles across time [10]. Assuming importance density to be equal to the prior density, weight update is recursively computed as:

$$w_t^j \propto w_{t-1}^j p(z_t|x_t^j). \tag{4.15}$$

PF avoids the particle degeneracy problem by re-sampling at every time step. In this case, weights are set to $w_{t-1}^j = 1/N_s$, $\forall j$, therefore

$$w_t^j \propto p(z_t|x_t^j). \tag{4.16}$$

Hence, the weights are proportional to the likelihood function that will be computed over the incoming volume z_t. The re-sampling step derives the particles depending on the weights of the previous step, then all the new particles receive a starting weight equal to $1/N_s$ which will be updated by the next volume likelihood function.

Finally, the best state at time t of target m, X_t^m, is derived based on the discrete approximation of Eq.4.14. The most common solution is the Monte Carlo approximation of the expectation as

$$X_t^m = \mathbb{E}\left[x_t|z_{1:t}\right] \approx \frac{1}{N_s} \sum_{j=1}^{N_s} w_t^j x_t^i. \tag{4.17}$$

4.3.2 Filter Implementation

Two crucial factors are to be taken into account when implementing a PF: the likelihood evaluation and the propagation model. For a given target m, an adaptive reference histogram H_t^m of the colored surface voxels is available. CbCr color space is chosen due to its robustness against light variations and 21 bins for every channel are employed in the calculations. Let $\mathcal{E}_t^j$ be the ellipsoid centered at x_t^j with a fixed size roughly modelling the human body. Function $p(z_t|x_t^j)$ can be defined as the likelihood of the ellipsoid $\mathcal{E}_t^j$ overlapping the volume corresponding to the tracked person and matching its color histogram. Information obtained at time t from the binary, V_t^b, and color, V_t^c, 3D reconstructions (see Figs. 1c and 1d) are used to define the likelihood function as:

$$p(z_t|x_t^j) = \alpha \frac{|V_t^b \cap \mathcal{E}_t^j|}{|\mathcal{E}_t^j|} + (1-\alpha)B(H_t^m, H(V_t^c \cap \mathcal{E}_t^j)), \tag{4.18}$$

where $|\cdot|$ is the number of occupied voxels of the enclosed volume, $B(\cdot)$ is the Bhattacharya distance, $H(\cdot)$ stands for the color histogram extraction operation. Factor α controls the influence of each term (foreground and color information) in the overall likelihood function. Propagation model has been chosen to be a Gaussian noise added to the state of the particles after the re-sampling step. More sophisticated schemes employ previously learned motion priors to drive the particles more efficiently [12]. However, this would penalize the efficiency of the system when tracking unmodelled motions patterns and, since our algorithm is intended for any motion tracking, no dynamical model is adopted.

4.3.3 Sparse Sampling

PF approach to tracking defines a set of instances of the position of the tracked person, the particles, and a formulation to measure the fitness of these hypothesis with relation to the observable data. However, the evaluation of this likelihood function may be computationally expensive. Sparse sampling is a valid alternative to PF which reduces the computational effort.

The estimation of the state X_t in Eq.4.17, that is in our case the position of the centroid of a person, is extracted by computing the expectation over all the surface voxel positions. By randomly selecting a given number of voxels on this surface, it is still possible to obtain an enough accurate estimation of X_t. We define the *Sparse Sampling* (SS) algorithm as a method to recursively estimate X_t from an evolving set of samples placed on the surface of the tracked person. Since we are no longer exploring the state space, we will talk about *samples* instead of *particles*.

Essentially, the proposed algorithm follows the PF analysis loop (re-sampling, propagation, evaluation and estimation). Being our volume a discrete representation, samples are constrained to occupy a single voxel and move with displacements on the 3D discrete orthogonal grid. By defining the appropriate likelihood function, samples attain high weights when placed on the surface while the re-sampling block is constrained to place the newly created samples on the foreground voxels. With this process, we define a recursive way to obtain a sparsely sampled version of the surface of the target and, therefore, its centroid.

Propagation

The propagation step of the Sparse Sampling algorithm is quite similar to those described for particle filtering (see chapter 3). We need to generate samples from a important density $\pi(x_k|x_{k-1},z_k)$. In our case we chose $\pi(x_k|x_{k-1},z_k) = p(x_k|x_{k-1})$ that is the prior density as for the SIR particle filter (see 3.5.2). We place the samples using the estimation done in the previous frame and according to a pdf associated with a propagation model. Anyway tracking people in small spaces, such as a smart room, doesn't require a complex propagation model because movements are limited. We assume that the target at time k we'll be near the position estimated at frame $k-1$. That's a really simple movement model but works well in our conditions. The only constraint to satisfy in order to obtain a good performance is that the resampling, that occurs at every iterations, must assure that the samples cover more or less uniformly all the space near a target.
To assure better performance with the same number of samples and to reduce the computational effort we require that there must be just one sample per voxel. A pictorial description of this constraint in figure 4.8.

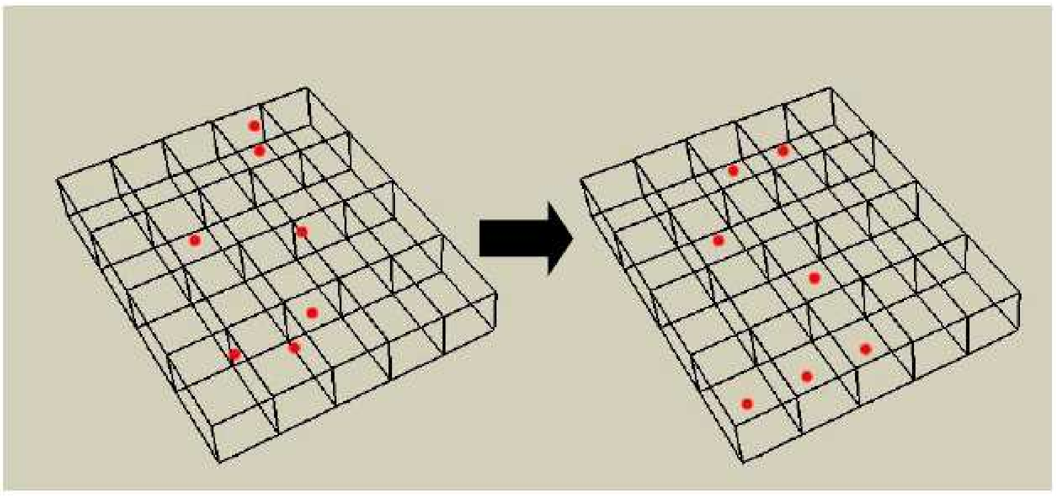


Figure 4.8. A voxel must be occupied by just one sample (red points).

It's not always possible to establish an unambiguous relationship between samples and voxels above all during the initialization of the filter and if the samples in use are more then 500. However this happens only for a limited amount of time, after some frames the diversification of the sample is quite effective. We can conclude that, after the filter initialization, there is a transient state during which the available information is not totally exploited because of redundancy in samples positioning.

Weights computation

After the propagation the next step of the algorithm is the weights computation, that consist of assigning to every sample a weight which estimate the likelihood of the voxel occupied by the sample to be part of a volume of a person and more exactly of the person tracked in the previous frame. The function $p(z_k|x_k)$ is obtained exploiting the geometrical information of the region near the sample and analyzing the possible interactions between different target in the current frame.

The likelihood of a sample $x_k^{(m)}$, which is associated to one voxel, is estimated considering if the voxel belongs to the volume corresponding to a person or not. For example a very simple approximation to obtain $p(z_k|x_k^{(m)})$ could be done assigning a positive value to $w_k^{(m)}$ if the voxel considered is a *foreground* voxel or 0 if it's a *background* voxel.

$$w_k^{(m)} \propto \begin{cases} \frac{1}{N_f}; & v \in \psi \\ 0, & v \in \beta \end{cases}$$

where N_f is the number of the samples which are positioned in a voxel labelled as foreground and is needed to normalize the weights, ψ is the set of foreground voxels and β the set of background voxels. It's easy to notice that this approximation is strongly affected by noise in the 3D reconstruction of the scene. A sample which falls in a voxel that is wrongly labelled as foreground will have a weight different from zero and, on the contrary, a sample positioned in a voxel wrongly labelled as background will have a weight equal to zero and will not be taken into account in the resampling step. This can be avoided using an estimation based not only on one voxel but on a set of neighbor voxels. This is the new rule for the weight computation:

$$w_k^{(m)} \propto \frac{1}{|C(x_k^{(m)},q)|} \sum_{v \in C(x_k^{(m)},q)} r(v) \tag{4.19}$$

where

$$r(v) = \begin{cases} 1; & v \in \psi \\ 0, & v \in \beta \end{cases}$$

and $C(\cdot)$ stands for the neighborhood over a connectivity q domain on the 3D orthogonal grid and $|C(\cdot)|$ represents its cardinality. This way of computing the likelihood is more precise and robust to noise. To improve this estimation we could add a measure of the distance between the sample and the foreground voxel:

$$w_k^{(m)} \propto \frac{1}{|C(x_k^{(m)},q)|} \sum_{v\in C(x_k^{(m)},q)} r(v)d(x_k^{(m)},v); \tag{4.20}$$

Anyway in this way we tend to concentrate the samples in some points of attraction, penalizing the uniformity of the distribution of the samples. These points of attraction are located in the region where the reconstruction is more precise. If the reconstruction is uniformly good this measure provides good results. Anyway since a good reconstruction is not always possible, another method is needed to assure the dispersion of the samples.

The method used at the UPC is to concentrate the weights on the superficies of the volume. The weights computation is modified to estimate the probability of a sample to be on a surface voxel. A sample located on a surface voxel will have in its neighborhood more or less an equal number of foreground and background voxels. We should establish in which proportion the two kind of voxels should appear in the neighborhood of a sample to deduce that it's on the surface. In figure 4.9 are showed some example of possible configuration of a sample located on the surface. In the first and third case the sample is on a foreground voxel, but the number of foreground and background voxel in its neighborhood is different. In the second case the sample falls in a background voxel but anyway could be considered on the surface. In the forth case an example of bad reconstruction is showed. The proportions of foreground and background voxels in these examples is the following: 3-6, 6-3, 5-4 e 4-5. We could deduce that approximately the number of foreground and background voxels should be the same to consider a sample aa located on the surface.

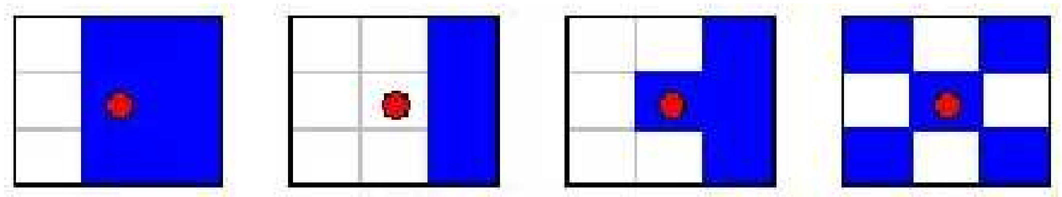


Figure 4.9. Four possible configuration for a sample located on the surface.

The new expression used for weights computation is:

$$w_k^{(m)} \propto f\big(|C(x_k^{(m)},q))|\big)\Big[1-\Big|\Big[\frac{1}{|C(x_k^{(m)},q)|}\Big(\sum_{v\in C(x_k^{(m)},q)} r(v)-\Big(|C(x_k^{(m)},q))|-\sum_{v\in C(x_k^{(m)},q)} r(v)\Big)\Big)\Big]\Big|\Big] \tag{4.21}$$

where f is a function whose independent variable is the number of foreground voxel in the analyzed neighborhood and it's used to modify the weights computation, changing the proportion of foreground and background considered optimal to assume that a sample is on the surface.

Thanks to the above explained weights computation, to the propagation (section 4.3.3) and the resampling, discussed in the next section, it's possible to obtain a good individual tracker, resistent to noise and partial occlusion. Anyway tracking multiple persons presents an higher level of complexity. People interact between them in many ways and it's difficult to track them when they touch each other or are really close. The 3D representation of two persons next to each other presents often just one big volume instead of two volume almost overlapping. In this case it's impossible to keep a good accuracy in the tracking and there is no way of avoiding it modifying the weights computation, because it's a problem of the representation.

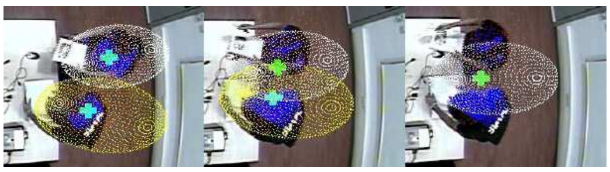
Another frequent problem is mismatching : when two people get near and then separate sometimes happens that tracker i change target and follow target j and tracker j start following target i. In this case it's possible to avoid this problem including in the weight computations the color features, as explained in section 4.4.

Multi-Person Tracking

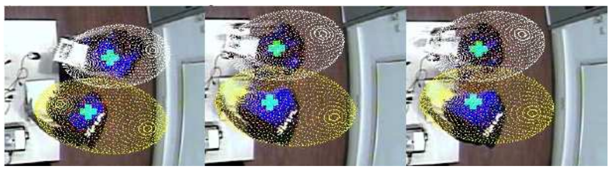
Challenges in 3D multi-person tracking are basically twofold. First, finding an interaction model in order to avoid mismatches and target merging. The second is filtering spurious objects that appear in scene reconstruction and discarding non-relevant objects such as chairs or furniture. Several approaches have been proposed [[13], [22]] but the joint PF [12] presented in section 3.6.1 is the optimal solution to multi-target tracking. However, its computational load increases dramatically with the number of targets to track since every particle estimates the location of all targets in the scene simultaneously. That's why in the tracker implemented at UPC we used a split filter per person, which requires less computational load at the cost of not being able to solve some complex cross-overs [14]. However, this situation is alleviated by the fact that cross-overs are restricted to the horizontal plane in our scenario. Let us assume that there are N independent trackers, being N the number of humans in the room. Nevertheless, they are not fully independent since each filter can consider voxels from other tracked targets in both the likelihood evaluation or the 3D re-sampling step resulting in target merging or identity mismatches. In order to achieve the most independent set of trackers, we consider a blocking method to model interactions. Blocking methods penalize particles that overlap zones with other targets. Hence, blocking information can be also considered when computing the particle weights as:

$$w_k^{(m)} = \frac{1}{N_f} p(z_t|x_k^{(m)}) \prod_{k=1,k\neq n}^{N} \beta(X_{k-1}^{n},X_{k-1}^{N}) \tag{4.22}$$

where N is the total number of trackers, n the index of the evaluated tracker and X is the estimated state. Term $\beta(\cdot)$ is the blocking function defining exclusion zones that penalize particles that fall into them. For our particular case, considering that people in the room are always sitting or standing up (this is a meeting room so we assume that they never lay down), a way to define an exclusion region modelling the human body is by using an ellipsoid with fixed x and y axis. Axis in z is a function of the estimated centroid height. An example of this exclusion technique is depicted in figure 4.10 . Tracked objects that come very close can be successfully tracked even though their volumes have partially merged.



(a)



(b)

Figure 4.10. Frame without (a) and with (b) blocking condition. Samples from the tracker A (white ellipsoid) falling into the exclusion zone of tracker B (yellow ellipsoid) will be penalized by a multiplicative factor $\alpha \in [0,1]$.

Resampling

Resampling has usually been defined as a process where some noise is added to the position of the re-sampled samples according to their weights [10]. In our current tracking scenario, re-sampling adds some discrete noise to samples only allowing motion within the 3D discrete positions of adjacent foreground voxels as depicted in 4.11. Since in the propagation phase no movement model is applied, the resampling should guaranty that the samples will be positioned in the area where is more probable to find a target, that is the higher the weight, the more replicas will be created. First of all the algorithm calculate how many replicas will be generated for every sample:

$$|x_k^{(m)}| = w_k^{(m)} N_s \tag{4.23}$$

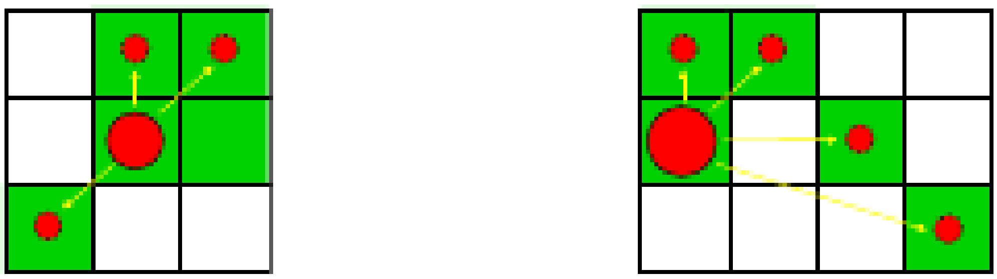


Figure 4.11. Discrete re-sampling example (in 2D).

where $|x_k^{(m)}|$ is the number of replicas associated to the sample $x_k^{(m)}$,$w_k^{(m)}$ is the weight of the sample and N_s is the total number of samples used.

The new samples are generated according to the following equation:

$$x_k^{(m)} = x_k^{(m)} + U\big[-\frac{\delta x}{2},\frac{\delta x}{2}\big]\overrightarrow{i} + U\big[-\frac{\delta y}{2},\frac{\delta y}{2}\big]\overrightarrow{j} + \big[-\frac{\delta z}{2},\frac{\delta z}{2}\big]\overrightarrow{k} \tag{4.24}$$

Every replica is generated displacing $x_k^{(m)}$ using an aleatoric distance, obtained from uniform distributions in X, Y and Z. Let r be a coordinate, N_s the number of samples used and h_v the size of the voxel in cm. The displacement (in cm) is computed as follows:

$$\delta r = 3.9\sqrt{h_v}\sqrt{N_s} exp(\frac{-N_s h_v}{20000}) \tag{4.25}$$

In equation 4.25 the number of samples is variable. We proved that there's a maximum numbers of samples that can be used to solve the tracking problem, above that number the performance gets worse. In our system the volumes are composed from 500 up to 20000 voxels (rarely there are more than 10000 voxels), in general using more than 2000 samples implies a certain redundancy.

4.4 Sparse sampling with color features

The purpose of our system is to track people in a smart room environment. In the previous section we described the tracking system implemented using sparse sampling, in which the weights were computed considering just the geometric information, that is the probability of a sample to be located on the surface. This approach worked quite good but still suffered some problems: mismatches were impossible to prevent, bad segmentation leaded to track spurious objects instead of the right target, the filters were sometimes instable. Our purpose was to solve these difficulties and to improve the performance of the tracker using color features. Color is a mark really useful to distinguish a person from another or a person from an object and exploiting this information for tracking lead to better performance.

Adding the color information to the weights computation means to modifying the weights update rule, remembering that:

$$w_t^j \propto p(z_t|x_t^j) \tag{4.26}$$

where $p(z_t|x_t^j)$ is the likelihood of a particle belonging to the volume corresponding to a person. The new rule takes into account both the geometric information and the color information:

$$p(z_t|x_t^j) = \alpha G + (1 - \alpha)B \tag{4.27}$$

where G is the coefficient computing using the geometric information, basically that computed by 4.21 and C that obtained exploiting color, as we will explain in the next sections.

4.4.1 Computing 3D Histograms

In order to track we need to define the color properties of every target (in our case human people) and to assign a likelihood $p(z_t|x_t^j)$ to a region of space to belong to a target. The most common approach is to use color histograms, [17, 23]. Color histograms are robust to partial occlusion, are rotation and scale invariant and are calculated efficiently. Using 3D data and having no a-priori knowledge about the position of the target, the computation of the histogram is not trivial. We formulated a two step procedure:

1. Histogram initialization
2. Histogram update

Histogram initialization: First of all an estimation of the position of target is needed, this is achieved using the coordinates of the centroid computed by the filter using only the geometrical information ($\alpha = 1$). Obviously this estimation should be as accurate as possible, that is the filter must became stable, this happens usually after 8-10 frames. If the filter last more than 10 frames the *Reference Histogram* is computed and this will be the color mark for the target. The computation is done as follows: once obtained the coordinates of the centroid only the voxels which fall inside the ellipsoid centered in the centroid estimated are taken into account for the histogram computation. The ellipsoid is sized based on typical human body measures, see figure 4.12.

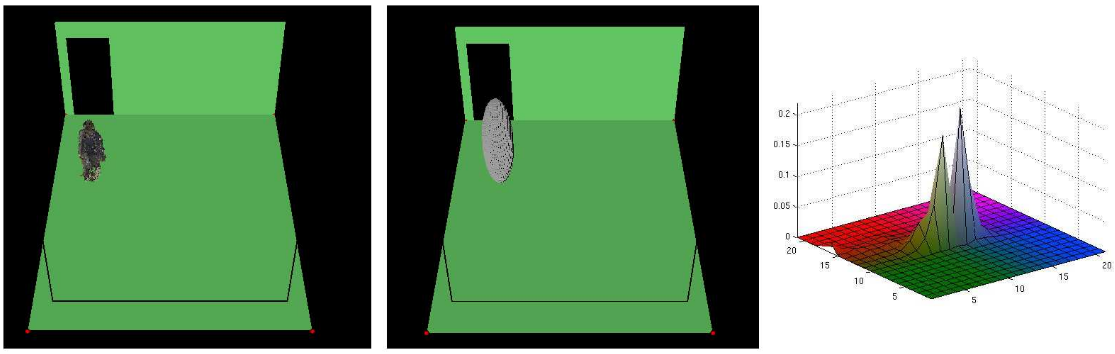


Figure 4.12. Histogram extraction

For the tracking purpose is fundamental to have a system robust to illumination changes: if the histograms vary too much from a frame to another just because the light has changes (which happens quite often) the tracker will fail to identify its proper target. RGB was discarded, because it proved quite instable, as expected. YCrCb worked well but the histograms computation was still computational espensive: 3D histograms were needed to process the information properly. That why we implemented a new color system Y*CbCb, see section 2.3.7 for details. Basically the chroma components are multiplied by the luma component, thus keeping the chrominance and the luma information and allowing the use of only two dimensions. We computed histograms with different numbers of bins to analyze if this choice could affect the performance. The results obtained didn't vary much if the bins are between 15 and 25. We chose to use 21 bins.

Histogram update: The Reference Histograms need to be updated in order to be resistent to illumination changes and noise. This is done using a linear combination of the new computed histogram and the Reference Histogram stored in the previous frame, for each target i:

$$Ref\_Hist^i_{t+1} = R \cdot Ref\_Hist^i_t + (R-1)Hist^i_t \tag{4.28}$$

where R is the memory factor, which controls the update speed, $Ref\_Hist_t^i$ is the Reference Histogram of target i at time t, that is the Reference Histogram stored at time $t = t-1$ and $Hist_t^i$ is the histogram of target i computed at time t, basically the histogram computed using the position of the centroid estimated at time t. We used different values for the memory factor to analyze if this could affect the performance. Values between 0.1 and 0.3 proved quite ineffective. To give more importance in the updating to the new computed histogram means adjusting faster to changing conditions. Unfortunately if there are imprecisions in the estimation of the position of the centroid, the histogram computed won't mirror the color characteristics of the target and having an high weight will affect much the precision of the Reference Histogram. Values between $0.7 - 0.9$ didn't prove better because the updating resulted too slow. The system was unable to cope with illumination changes and the number of mismatches increased. Values between $0.4 - 0.6$ are a good compromise between a rapid update and carefulness and provided good results. We use in our implementation $R = 0.5$.

4.4.2 Likelihood evaluation through color

Our main goal is to estimate the likelihood of a particle belonging to the volume corresponding to a person using color information, that is determining the coefficient B in equation 4.27. Using the information stored in the Reference Histogram for every target i:

$$B = \frac{1}{|C(x_t^j,q)|} \sum_{p \in C(x_t^j,q)} m(x_t^j,p,Ref\_Hist_t^i) \tag{4.29}$$

where $C(\cdot)$ stands for the neighborhood over a connectivity q domain (the same used to compute the G coefficient) and m is a function which assign a likelihood to a colored voxel p in the neighborhood of the sample j.

The design of the function m is fundamental to assure good performances of the tracker.At the beginning we use a simplified approch, trying to define a fixed threshold based on the analysis of the color histograms. We tried different values to check if it was possible to identify some positive trends. Anyway the histograms vary from a target to another and from frame to frame, a fixed threshold can't cope with that. The system lacked of adaptativity.

We developed a new function m, which uses an adaptative threshold:

$$m = \begin{cases} 0, & Ref\_Hist_t^i(p) < threshold \\ 1, & Ref\_Hist_t^i(p) > threshold \end{cases}$$

Threshold is a value which is computed at every frame and for every Reference Histogram. In our approach the threshold is determined so that the 80% of the values of the Reference Histograms are taken into account. This is reasonable because the remaining 20% is made up of noise or of really small values which aren't distinguishing of the target.

This formulation brings good results, anyway the function m is quite sharp.The possible weights are 1 or 0, no other value is possible. If a sample occupies a voxel whose color value is beyond the threshold in the Reference Histogram of the target, this sample is assigned weight = 1, if the value is just a little below that threshold, its weight will be zero. In this way the information stored in the Reference Histogram is not fully exploited, we consider just the value of the threshold and according to that the weights are computed, thus neglecting other important characteristics. For example we don't take into account the shape of the Reference Histogram, if it is sparse or if it has a peak or two. Moreover if there are imprecisions in the 3D reconstruction or the estimated position of the centroid, which is used for the histogram computation, is not accurate, using a sharp rule for the weights assignment could affect the performance of the tracker. Trying to solve these drawbacks we formulated a new weights assignment rule:

$$m = \begin{cases} 0, & Ref\_Hist_t^i(p) < threshold_{90} \\ ]0,...,1[, & threshold_{70} < Ref\_Hist_t^i(p) < threshold_{90} \\ 1, & Ref\_Hist_t^i(p) > threshold_{70} \end{cases}$$

In this case if the color value of the voxel occupied by a sample is below $threshold_{90}$ (that is the threshold computed so that the 90% of the values of the Reference Histograms are taken into account) the weight will be 0, if the value is beyond the $threshold_{70}$ (that is the threshold computed so that the 70% of the values of the Reference Histograms are taken into account) the weight will be 1. If the value is between these thresholds the weight is assigned linearly according to the position of the value in this interval. The width of this interval mirrors the shape of the Reference Histogram, if it's wide it means that the histogram is quite sparse, if it's small the histogram probably has a peak.

We tried to exploit color features to reduce the number of mismatches using the Bhattacharyya distance between the Reference Histograms of two targets. A mismatch occurs when the tracker j instead of tracking target j, tracks target k. The solution we implemented is to compute at every frame the Bhattacharya distance between the Reference Histogram of target j at frame $k-1$ and the histogram centered in the centroid estimated by tracker j at frame k:

$$B = 1 - \sum_{b=0}^{N_b} \sqrt{h_b^{RefHist_j} h_b^{Hist_j}} \tag{4.30}$$

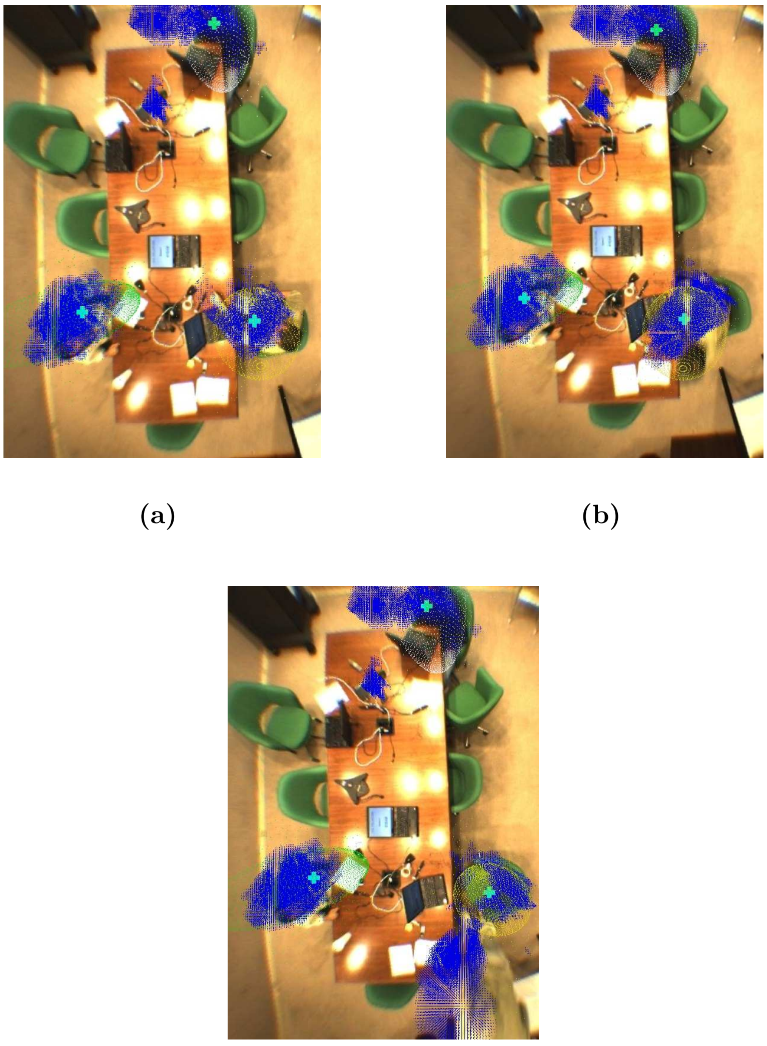
where h_j are the bins and N_b the maximum numbers of bins. If the Bhattacharyya distance is $= 0$ the histograms match perfectly, if it's $= 1$ they are totally different. In our case we posed the following condition to try to avoid mismatches: if the Bhattacharyya distance was bigger than 0.5 the filter was deleted. Anyway after few trial we realized that the numbers of mismatches didn't diminish. There are two main reasons for that: (1) the histograms of two different targets could be similar, thus having a small Bhattacharyya distance and eluding the above condition, the mismatch occurs anyway; (2) due to illumination changes the histogram of target j at frame k could be different from its Reference Histogram at frame $k-1$ thus leading to deleting a filter which indeed worked well and unfortunately increasing the instability of the filters. Eventually we conclude that this condition was not improving the tracking at all.

To reduce the number of false positives (that is volumes wrongly labelled as humans) we decided to exploit the color information checking if in a volume classified as a person there were a certain number of voxels, whose color were probably the color of the skin. According to [18], we built a probabilistic model of the color of skin to assign a voxel a probability to be a skin voxel. The volume was labelled as human if had a certain number of voxels (50 in our case which is a reasonable number according to the body dimensions) whose probability to be a skin voxel was high. The numbers of false positives didn't diminish much, instead the number of persons that the tracker didn't detect (misses) increased. This is due probably to an imprecise model of the color of the skin. The model should be adjusted to the system. Moreover the centroid position is sometime imprecise, that's why not all the skin voxels fall into the ellipsoid used for the histogram computation, resulting fewer than they actually are.

Anyway on the whole the color information proved to be really useful and improved the performance of the tracker. This is due manly to three reasons:

- Color proved to solve problems related to wrong background learning
- Color helps avoiding mismatches
- Color makes the filters more stable

Wrong background learning: In figure 4.13 and figure 4.14 a sequence is show where, because of wrong background learning, voxels belonging to a chair are labelled as foreground. In figure 4.13 the tracker without using color information instead of following the person track the chair, see (c),below on the right. In figure 4.14 the same sequence is show, but using color the tracker performs well and track the proper target. Without using color the tracker has no memory of the target appearance but only of its previous position. The samples near the chair gain high weights because they are pretty near to the target position in the previous frame. The color features has the effect of moving the samples toward the target. Exploiting the information stored in the Reference Histogram the samples near the chair, which is green and quite different from the colors of the person, are assigned weights smaller than those near the target, whose colors have a greater value in the Reference Histogram.



(a) **(b)**

(c)

Figure 4.13. Sequence of frames where the tracker doesn't follow the person standing up (below on the right).

Mismatches: Color is fundamental to avoid mismatches: if two persons cross,

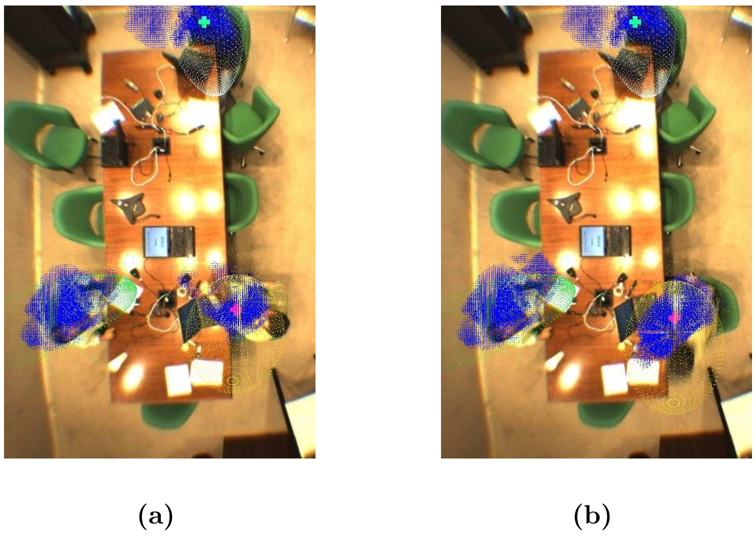
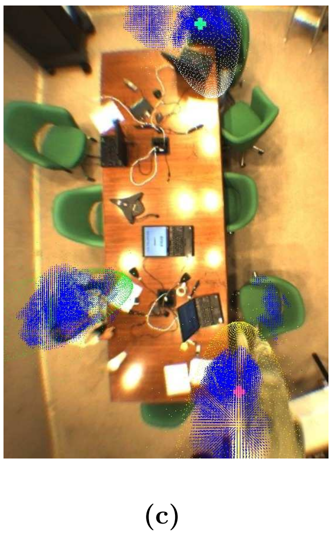



Figure 4.14. Same sequence but using color the tracker doesn't fail.

the geometrical information is not enough to identify them after they have separated. Using the Reference Histogram of both this is possible because the samples of a target are attracted by its color, thus tracking well. For example in figure 4.15 without using the color information the tracker isn't able to follow the person which stands up and cross the other target (c), at the top of the picture. In (d) a new filter is created to track it, causing a mismatch. In figure 4.16 this doesn't happen: the person is tracked perfectly, the filter follows the target from (a) to (d).

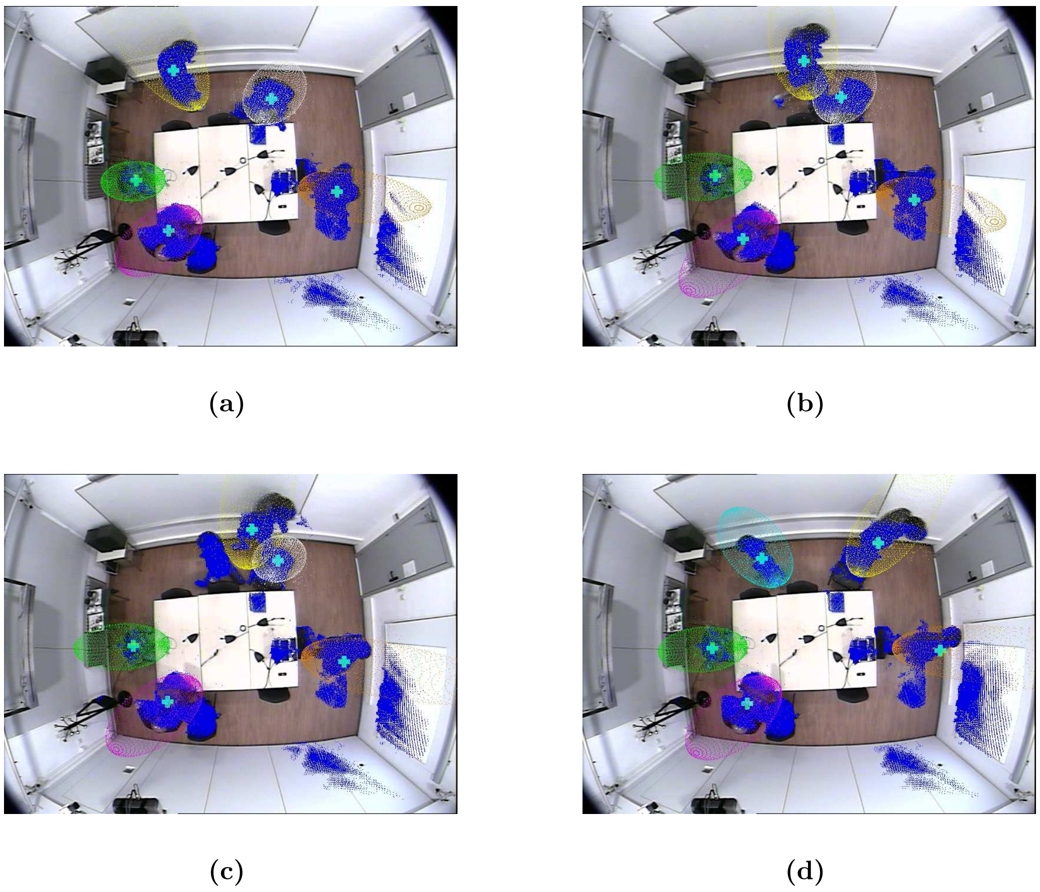
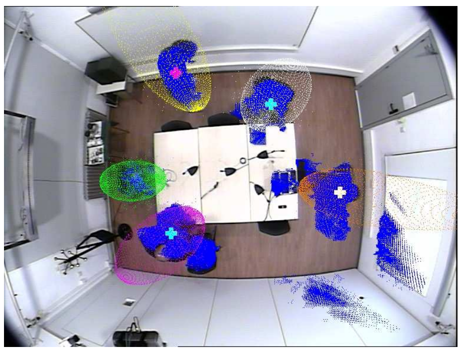


Figure 4.15. Sequence of frames showing a mismatch.

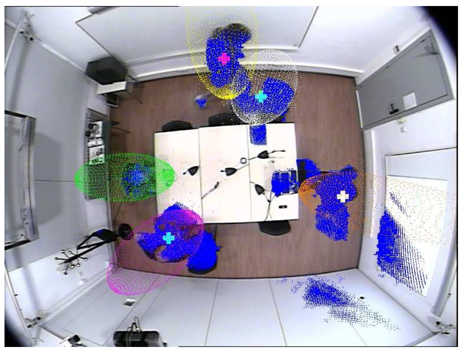
Stability of the filters: In our experiments color proved useful to make the filters more stable. This is related to the degeneracy phenomenon according to which after a few iterations, all but only few samples will have negligible weight. To avoid that in our system we implemented a resampling algorithm, anyway this is sometimes not enough to avoid degeneracy. As a countermeasure we decided that if the N_{eff} of a filter goes under a fixed threshold, the filter is deleted. The N_{eff} can be approximated by:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2} \tag{4.31}$$

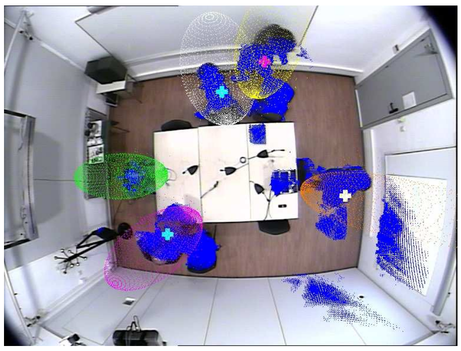
We noticed that using color information the N_{eff} is higher because the weights are higher. This means that the filters aren't deleted as easily as before and they become more stable. In figure 4.17 a sequence of frames is shown where the filter



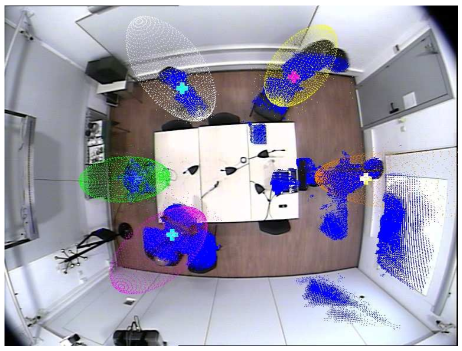
(a)



(b)



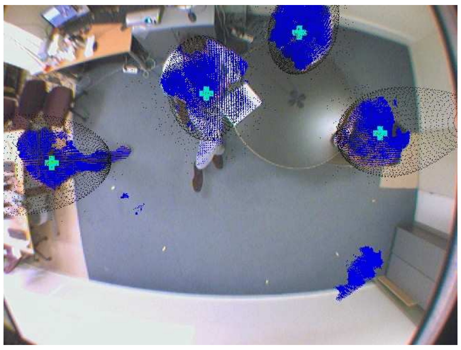
(c)



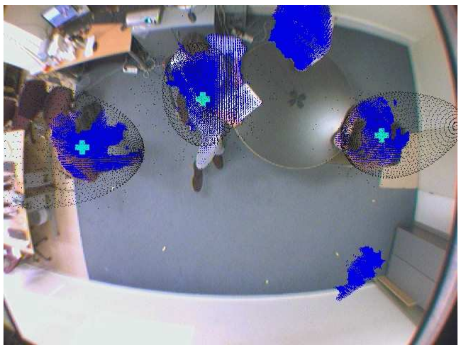
(d)

Figure 4.16. Same sequence but using color the mismatch doesn't happen.

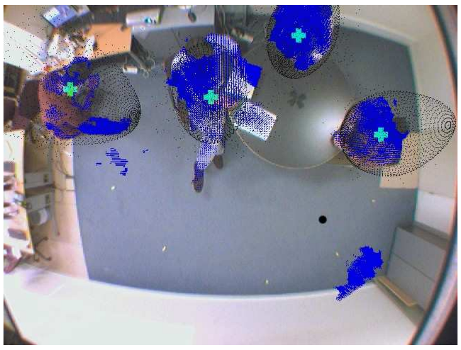
tracking the person at the top of the pictures is constantly created and deleted. In figure 4.18 this doesn't happen because the color features make the filter more stable.



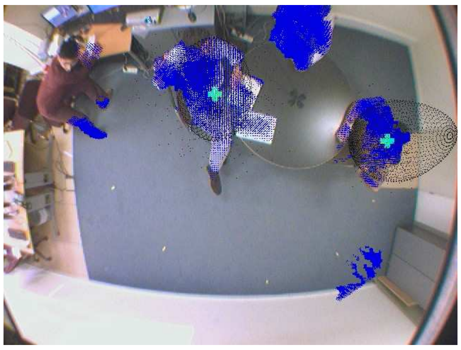
(a) *Frame 3200*



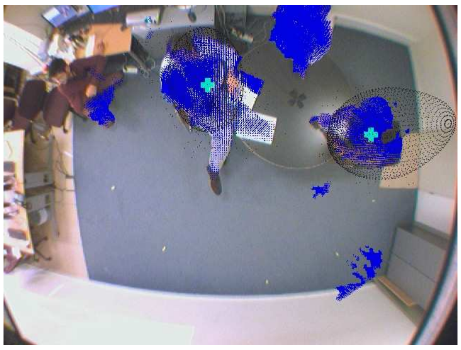
(b) *Frame 3240*



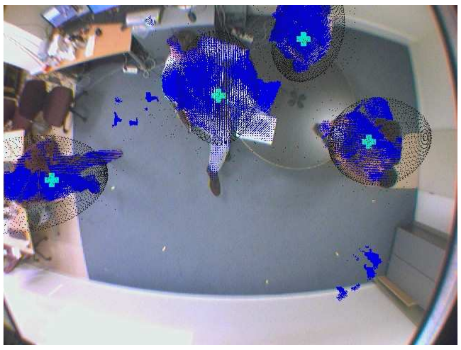
(c) *Frame 3350*



(d) *Frame 3860*

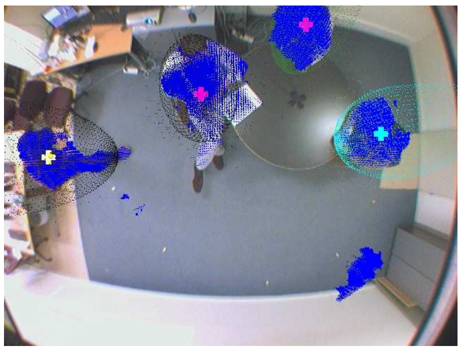


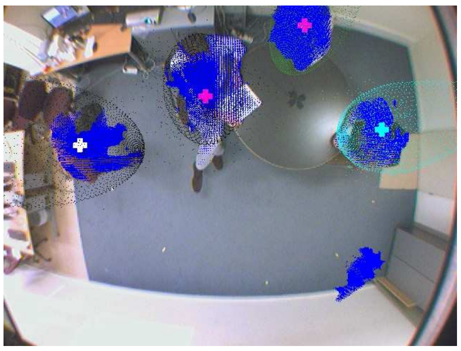
(e) *Frame 3990*

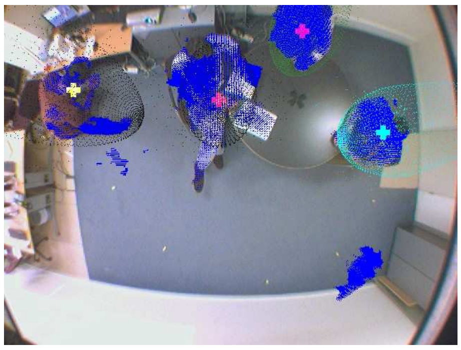


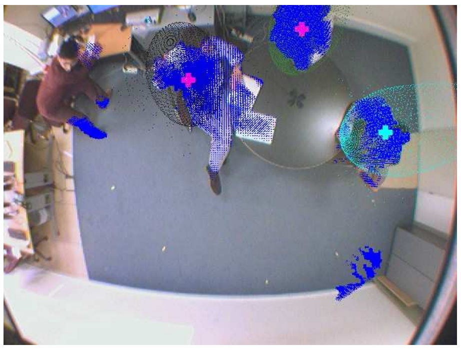
(f) *Frame 4970*

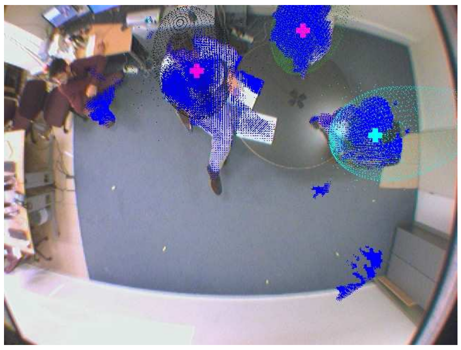
Figure 4.17. Sequence of frames where a filter (at the top of the picture) is constantly created and destroyed.


(a) *Frame 3200*


(b) *Frame 3240*


(c) *Frame 3350*


(d) *Frame 3860*


(e) *Frame 3990*

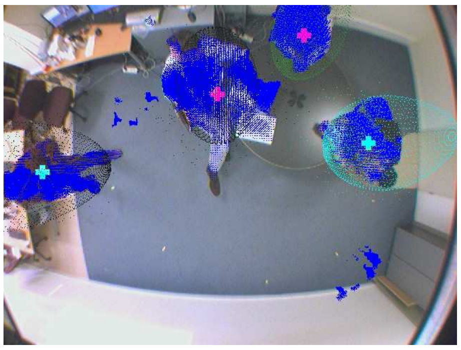

(f) *Frame 4970*

Figure 4.18. Same sequence but using color features the filter has become stable.

Chapter 5

Experimental results

In this chapter we will present the results of the tests performed to prove the effectiveness of our system. SS and SS with color features are compared between each other and with another system based on particle filtering. We will briefly describe the metrics used for the comparison too.

5.1 Metrics

Defining measures to express all of the important characteristics of a system for continuous tracking of multiple objects is not a straightforward task. Various measures, all with strengths and weaknesses, currently exist and there is no consensus in the tracking community on the best set to use. We chose to adopt the metrics proposed in [15] for multi-person tracking evaluation. These metrics, being used in international evaluation contests [16] and adopted by several research projects such as the European CHIL or the U.S. VACE allow objective and fair comparisons. In the following, these metrics are briefly introduced and a systematic procedure for their calculation is shown.

5.1.1 The MOT Precision and Accuracy Metrics

For the design of the CLEAR multiple object (person) tracking metrics, the following criteria were followed:

- They should allow to judge a tracker's precision in determining exact object locations.

- They should reflect its ability to consistently track object configurations through time, i.e. to correctly trace object trajectories, producing exactly one trajectory per object.

Additionally:

- to have as few free parameters, adjustable thresholds, etc, as possible to help make evaluations straightforward and keep results comparable.
- to be clear, easily understandable and behave according to human intuition, especially in the occurence of multiple errors of different types or of uneven repartition of errors throughout the sequence.
- to be general enough to allow comparison of most types of trackers (2D, 3D trackers, acoustic or visual trackers, etc).
- to be few in number and yet expressive, so they may be used e.g. in large evaluations where many systems are being compared.

Based on the above criteria and assuming that for every time frame t a multiple object tracker outputs a set of hypotheses $h_1,...,h_m$ for a set of visible objects $o_1,...,o_n$, the procedure to evaluate its performance is defined as follows:

Let the correspondence between an object o_i and a hypothesis h_j be valid only if their distance $dist_{i,j}$ does not exceed a certain threshold T (in our case 500mm), and let $M_t = (o_i,h_j)$ be a dynamic mapping of objecthypothesis pairs.

Let $M_0 =$. For every time frame t,

1. For every mapping (o_i,h_j) in M_{t-1}, verify if it is still valid. If object o_i is still visible and tracker hypothesis h_j still exists at time t, and if their distance does not exceed the threshold T , make the correspondence between o_i and h_j for frame t.
2. For all objects for which no correspondence was made yet, try to find a matching hypothesis. Allow only one to one matches. To find optimal correspondences that minimize the overall distance error, Munkre's algorithm is used. Only pairs for which the distance does not exceed the threshold T are valid. If a correspondence (o_i,h_k) is made that contradicts a mapping (o_i,h_j) in M_{t-1}, replace (o_i,h_j) with (o_i,h_k) in M_t. Count this as a mismatch error and let mme_t be the number of mismatch errors for frame t.
3. After the first two steps, a set of matching pairs for the current time frame is known. Let c_t be the number of matches found for time t. For each of theses matches, calculate the distance d_t^i between the object o_i and its corresponding hypothesis.
4. All remaining hypotheses are considered false positives. Similarly, all remaining objects are considered misses. Let fp_t and m_t be the number of false positives and misses respectively for frame t. Let also g_t be the number of objects present at time t.

5. Repeat the procedure from step 1 for the next time frame. Note that since for the initial frame, the set of mappings M_0 is empty, all correspondences made are initial and no mismatch errors occur.

Based on the matching strategy described above, two very intuitive metrics can be defined: The **M**ultiple **O**bject **T**racking **P**recision (**MOTP**), which shows the tracker's ability to estimate precise object positions, and the **M**ultiple **O**bject **T**racking **A**ccuracy (**MOTA**), which expresses its performance at estimating the number of objects, and at keeping consistent trajectories:

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t} \tag{5.1}$$

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} \tag{5.2}$$

The MOTA can be seen as composed of 3 error ratios:

$$\bar{m} = \frac{\sum_t m_t}{\sum_t g_t} \tag{5.3}$$

$$\bar{fp} = \frac{\sum_t fp_t}{\sum_t g_t} \tag{5.4}$$

$$m\bar{m}e = \frac{\sum_t mme_t}{\sum_t g_t} \tag{5.5}$$

the ratio of misses, false positives and mismatches in the sequence, computed over the total number of objects present in all frames.

The MOTP shows tracker's ability to estimate precise object positions, and the MOTA expresses its performance at estimating the number of objects, and at keeping consistent trajectories. MOTP scores the average metric error when estimating multiple target 3D centroids, while MOTA evaluates the percentage of frames where targets have been missed, wrongly detected or mismatched.

In figure 5.1 a pictorial description of misses, false positives and mismatches is presented. In figure 5.2 a case of a miss and consequent mismatch is showed.

5.2 Results

In order to evaluate the performance of the proposed algorithm, we used a set of multi-view scenes in an indoor scenario involving up to 6 people, for a total of approximately 25 min. The analysis sequences were recorded with 5 fully calibrated

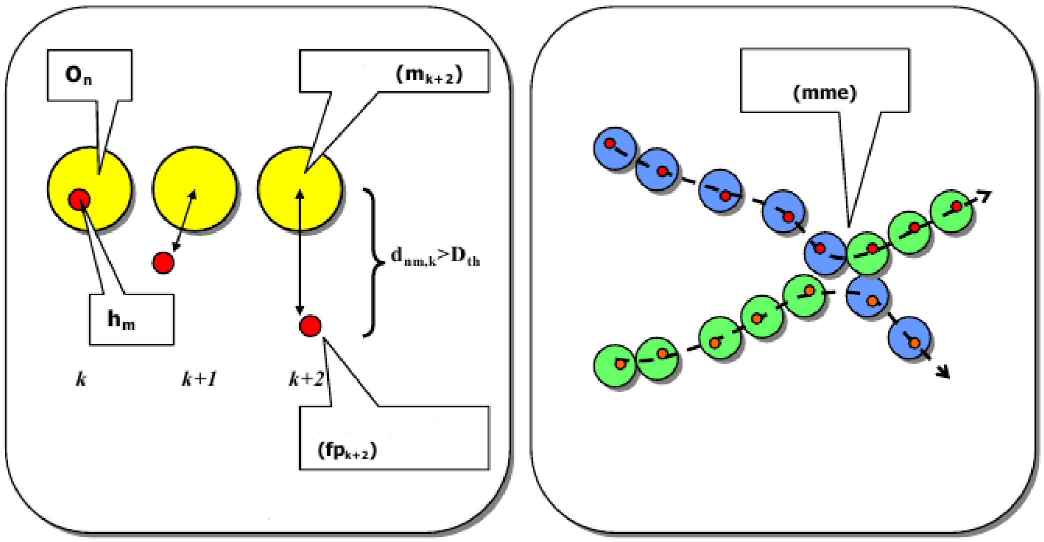


Figure 5.1. Misses, false positives and mismatches.

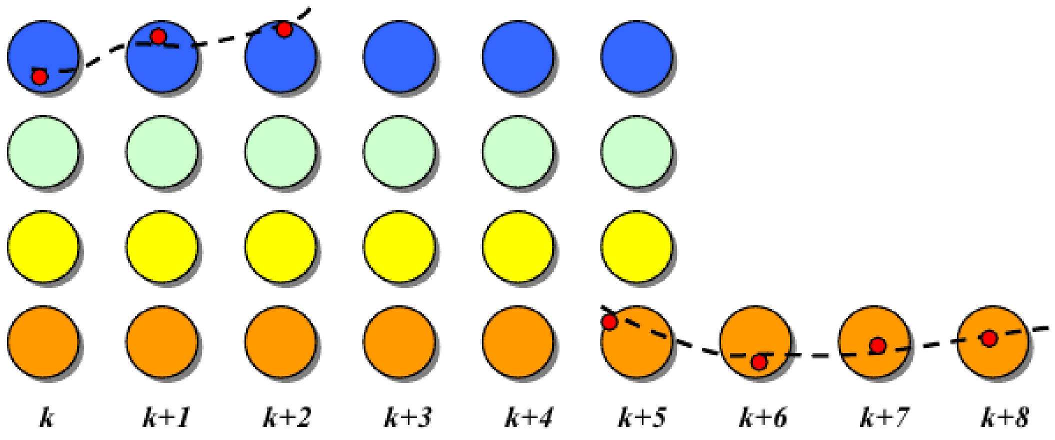


Figure 5.2. Miss and consequent mismatch.

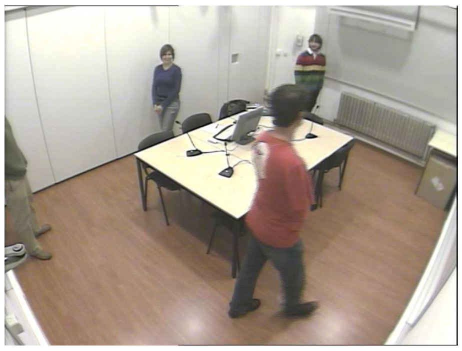
and synchronized cameras with a resolution of 720x576 pixels at 25 fps (see an example in Figure 5.3).

The test environment is a 5m by 4m room with occluding elements such as tables and chairs. The employed test database has been included in the CLEAR06 Evaluation [16]. Video annotations were realized using an in house developed tool. This tool allows to sequentially display video frames to be annotated, for the 4 corner cameras. On each displayed picture, the annotator was to click on the head

(a) *Camera 1*

(b) *Camera 2*

(c) *Camera 3*


(d) *Camera 4*

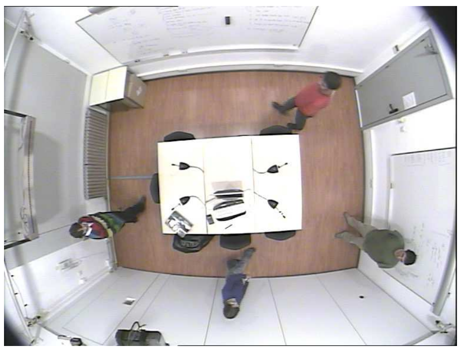

(e) *Camera 5*

Figure 5.3. Images recorded by the 5 cameras at the UPC.

centroid (the estimated center of the head), the left eye, right eye, and nose bridge of the annotated person. In addition to these four points, a face rectangle bounding box was used to delimit the person's face. The 2D coordinates within the camera planes were interpolated among all cameras in order to compute the real "ground truth" location of the person within the room.
In table 5.1 are summarized the results of our tests using Sparse Sampling (SS) and Sparsa Sampling with color features (SS Color). The system by us implemented (SS Color) proved better than Sparse Sampling without color features (SS). This shows that color improves the tracking, which is what we wanted to demonstrate. The computational time of our system is more than doubled respect to that of Sparse Sampling. This is understandable because handling color features and computing color histograms make the system more complex. The position of the centroid is tracked with more accuracy (3.8 mm improvement on the average). The samples are attracted by the color of the target, thus achieving a more precise placing. The MOTA is increased (it is 9.3% better on the average) for the reasons already explained in section 4.4, color (1) solve some problems related to wrong background learning; (2) helps avoiding mismatches; (3) makes the filters more stable. The better results are obtained using 600 samples. An inferior amount of samples is not enough to guarantee a good tracking accuracy and beyond this value the samples are too many, causing negative effects.

In table 5.1 are presented the results achieved using the particle filtering based algorithm, described in section 4.3.1 to compare them with those obtained by our system. PF and SS achieved similar performances but the main difference arose in the computational load measured as frames per second (fps). Since likelihood function is computed over a local neighborhood in the SS case, the overall complexity of the algorithm is reduced in comparison with the PF.

Furthermore, the impact of color information in $MOTP$ and $MOTA$ scores for the SS tracking system is depicted in Figure 5.4 and for the PF tracking system in Figure 5.5.

	SS			SS Color		
Samples	MOTA	MOTP	fps	MOTA	MOTP	fps
100	21.3	159	1.80	22.6	157	0.92
200	52.2	128	2.24	92.3	121	0.90
400	80.1	121	2.14	93.1	115	0.88
600	**88.8**	**119**	**1.47**	**94.4**	**116**	**0.72**
800	81.2	120	1.57	77.3	119	0.77
1000	74.3	126	1.42	75.2	122	0.70

	PF			PF Color		
Particles	MOTA	MOTP	fps	MOTA	MOTP	fps
100	2.1	167	0.95	9.3	166	0.96
200	15.4	160	0.75	42.7	159	0.69
400	77.2	142	0.38	87.3	125	0.36
600	**83.0**	**135**	**0.29**	**95.2**	**120**	**0.25**
800	81.9	131	0.27	94.0	119	0.20
1000	82.3	128	0.22	90.1	120	0.15

Table 5.1. Quantitative experiments for different number of particles/samples. Voxel size was set to be $\nu = 2$ cm.

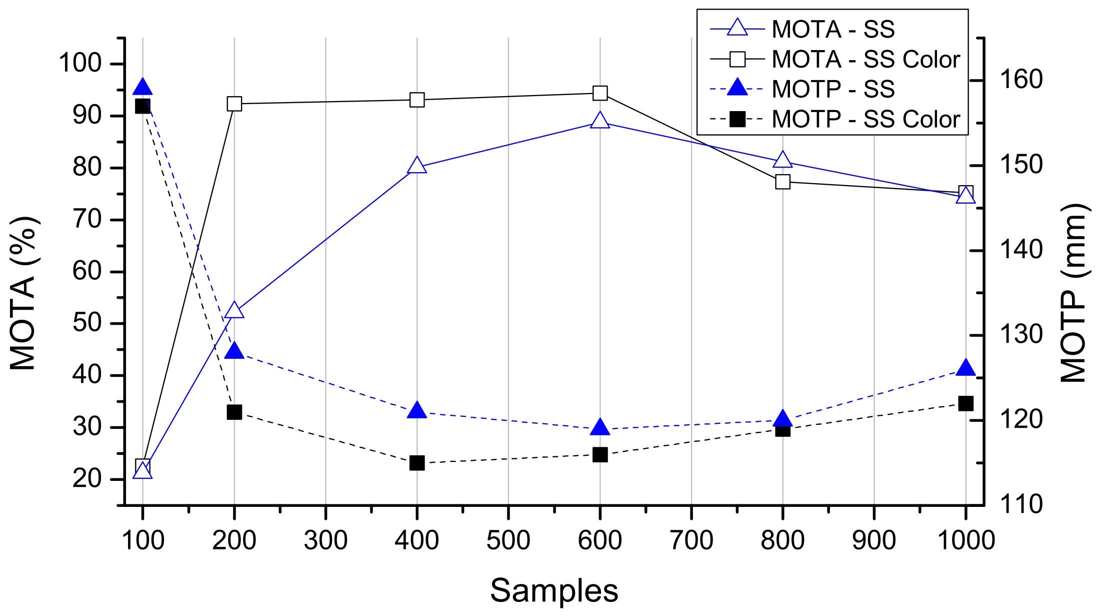


Figure 5.4. MOTP and MOTA scores for various numbers of samples using SS and SS Color algorithm. Low MOTP and high MOTA scores indicate low metric error in estimating multiple targets 3D positions and high tracking performance.

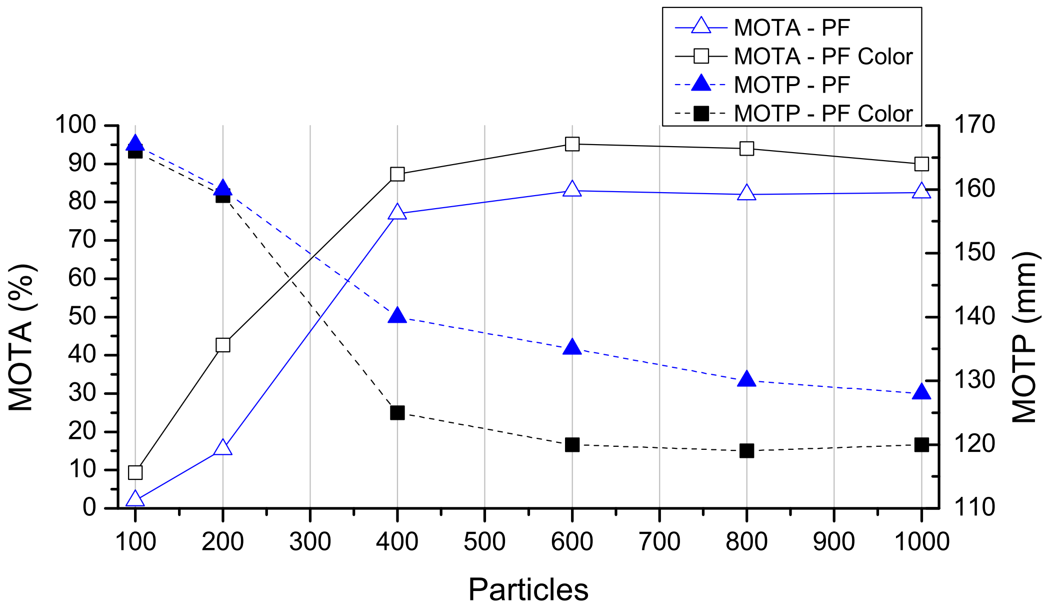


Figure 5.5. MOTP and MOTA scores for various numbers of samples using PF and PF Color algorithm.

Chapter 6

Conclusions and Future Work

The goal of our work was to improve an already existent multi-person tracking system exploiting color information. The previous tracking algorithm recursively estimated the state X_t, which in our case is the position of the centroid of a person, from an evolving set of samples placed on the surface of the tracked person and assigned the weights according to the probability of a voxel to be a surface voxels. To improve the performance we modified the weights update rule, including color features. In our system weights are computed including a coefficient which estimate the probability of the voxel to belong to the volume of a target, using the color information stored in the Reference Histogram. To make the model resistent to illumination changes histograms are update at every frame. Multiple targets are tracked assigning a filter to every one and in order to achieve the most independent set of trackers, we consider a 3D blocking method to model interactions.

The results obtained over a large test database proved the effectiveness of our strategy. Color feature improved the accuracy of the tracker of 3.8 mm on the average (the samples are attracted by the color of the target, thus achieving a more precise placing) and increased the MOTA, it is 9.3% better on the average. Color proved useful to (1) solve some problems related to wrong background learning; (2) avoid mismatches; (3) make the filters more stable.

Future research could implement a more precise human classifier (which is the algorithm that classifies the volumes as persons or as not-persons). For example an efficient model to detect the color of the skin could be developed and that feature could be used to diminish the number of false positive. Moreover an integration with audio technologies towards a multi-modal tracking system and a real-time implementations of the presented algorithms could be desirable.

Appendix A

The Bresenham Line-Drawing Algorithm

The Bresenham Line-Drawing Algorithm is used to approximate a line in a raster grid, which in our case is the grid determined by the voxels.

We restrict the allowable slopes of the line to the range $0 \leq m \leq 1$.

If we further restrict the line-drawing routine so that it always increments x as it moves, it becomes clear that, having computed a point at (x,y), the routine has a severely limited range of options as to where it may put the next point on the line:

- It may put the point $(x + 1,y)$, or:
- It may put the point $(x + 1,y + 1)$.

So, working in the first positive octant of the plane,line drawing becomes a matter of deciding between two possibilities at each step.

In drawing (x,y) the routine will, in general, be making a compromise between what it should draw and what the resolution of the grid actually allows it to draw. Usually the point (x,y) will be in error, the actual, mathematical point on the line will not be addressable on the pixel grid. So we associate an error, ϵ , with each y ordinate, the real value of y should be $y + \epsilon$. This error will range from -0.5 to just under +0.5.

In moving from x to $x + 1$ we increase the value of the true (mathematical) yordinate by an amount equal to the slope of the line, m. We will choose to draw $(x + 1,y)$ if the difference between this new value and y is less than 0.5.

$$y + \epsilon + m \leq y + 0.5 \tag{A.1}$$

Otherwise we will draw $(x + 1,y + 1)$.It should be clear that by so doing we minimize the total error between the mathematical line segment and what actually gets drawn.

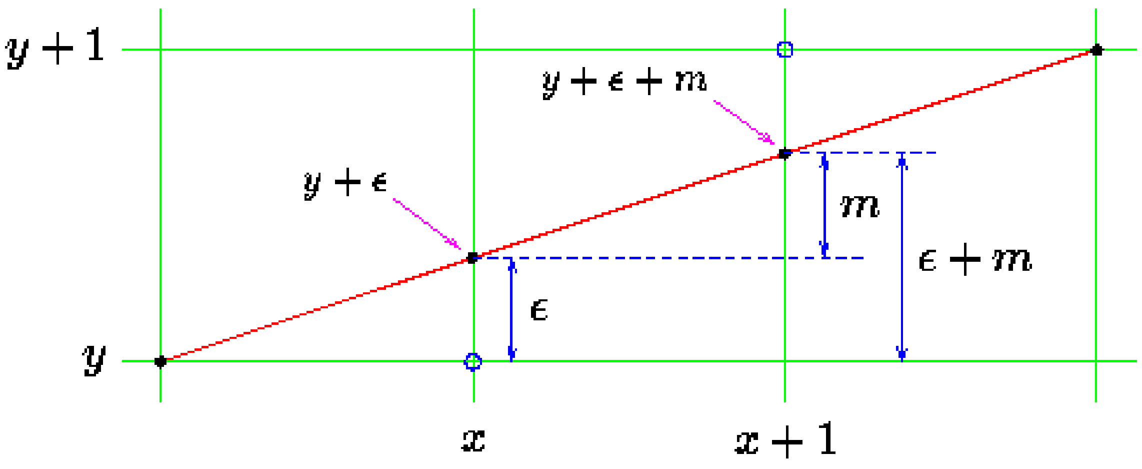


Figure A.1. Bresenham Line-Drawing Algorithm

The error resulting from this new point can now be written back into ϵ, this will allow us to repeat the whole process for the next point along the line, at $x+2$. The new value of error can adopt one of two possible values, depending on what new point is drawn. If $(x+1,y)$ is chosen, the new value of error is given by:

$$\epsilon_{new} \leftarrow (y+\epsilon+m) - y \tag{A.2}$$

Otherwise it is:

$$\epsilon_{new} \leftarrow (y+\epsilon+m) - (y+1) \tag{A.3}$$

In 3 the Bresenham Line-Drawing Algorithm is showed.

Algorithm 3 Bresenham Line-Drawing Algorithm

```
$\epsilon \leftarrow 0$,$y \leftarrow y_1$
**for** $x \leftarrow x_1$ to $x_2$ **do**
  draw point at $(x,y)$
  **if** $\epsilon + m \leq 0.5$ **then**
    $\epsilon \leftarrow \epsilon + m$
  **else**
    $y \leftarrow y + 1$ , $\epsilon \leftarrow \epsilon + m - 1$
  **end if**
**end for**
```

Bibliography

[1] D.Comaniciu, V.Ramesh and P. Meer, "Kernel-Based Object Tracking," in *IEEE Trans. On Pattern Analysis And Machine Intelligence*, vol. XXV, No 5, May 2003.

[2] Y. C. Ho and R. C. K. Lee, "A Bayesian approach to problems in stochastic estimation and control," in *IEEE Trans. Automat. Contr.*, vol. AC-9, pp. 333 339, 1964.

[3] T. Clapp and S. Godsill, "Improvement strategies for Monte Carlo particle filters," in *Sequential Monte Carlo Methods in Practice*, New York: Springer Verlag, 2001.

[4] C. Stauffer and W. Grimson, "Adaptive background mixture models for realtime tracking," in *Proc. of CVPR*, 333 339, 1998.

[5] Cheung, Kanade, Bouget and Holler, "A real time system for robust 3D voxel reconstruction of human motion," in *IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, pp. 714 720, 2000.

[6] Seitz and Dyer, "Photorealistic scene reconstruction by voxel coloring," in *Proc. Computer Vision and Pattern Recognition Conf.*, pp. 1067 1073, 1997.

[7] Alvy Ray Smith, "Color Gamut Transform Pairs," in *Computer Graphics 12*, Aug. 1978.

[8] Djuric, Kotecha, Zhang, Huang, Ghirmai, Bugallo and Miguez, "Particle Filtering," in *IEEE Signal Processing Magazine*, Sep. 2003.

[9] Ganzales and Woods, "Digital image processing," Prentice Hall, 2001.

[10] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," in *Signal Processing, IEEE Tran.*, vol. 50, no. 2, pp. 174 188, 2002.

[11] A. Doucet, "On sequential Monte Carlo methods for Bayesian filtering," *Dept. Eng., Univ. Cambridge, UK, Tech. Rep.,* 1998.

[12] Z. Khan, T. Balch, and F. Dellaert, "Efficient particle filter- based tracking of multiple interacting targets using an MRF- based motion model," in *Int. Conf. on Intelligent Robots and Systems*, 2003, vol. 1, pp. 254 - 259.

[13] K. Bernardin, T. Gehrig, and R. Stiefelhagen, "Multi and single view multiperson tracking for SmartRoom environments," in *CLEAR Evaluation Workshop*,

2006.

[14] A. Lopez, C. Canton-Ferrer, J. R. Casas. "Multi-Person 3D Tracking with Particle Filters on Voxels". in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Honolulu (USA), April 16 - 20, 2007.

[15] K. Bernardin, A. Elbs, and R. Stiefelhagen, "Multiple object tracking performance metrics and evaluation in a smart room environment," in *IEEE Int. Workshop on Vision Algorithms*, 2006, pp. 53 68.

[16] "CLEAR Evaluation," http://www.clear-evaluation.org.

[17] Nummiaro, Meier and Gool, "An adaptative color-based particle filter" in *Image and Vision Computing* 2002, 1-12.

[18] Storring and Andersen, "Skin colour detection under changing lighting condition." in *Araujo and J. Dias 7th Symposium on Intelligent Robotics Systems,* 187-195.

[19] N. Checka, K.W.Wilson,M.R. Siracusa, and T. Darrell, "Multiple person and speaker activity tracking with a particle filter," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 2004, vol. 5, pp. 881 - 884.

[20] D. Focken and R. Stiefelhagen, "Towards vision-based 3D people tracking in a smart room", in *IEEE Int. Conf. on Multimodal Interfaces*, 2002, pp. 400 - 405.

[21] C. Canton-Ferrer, J. R. Casas, and M. Pardás, "Towards a Bayesian approach to robust finding correspondences in multiple view geometry environments", in *Lecture Notes on Computer Science*, 2005, vol. 3515, pp. 281 - 289.

[22] O. Lanz, "Approximate Bayesian multibody tracking," in *Pattern Analysis and Machine Intelligence, IEEE Trans.* on, vol. 28, no. 9, pp. 1436 - 1449, 2006.

[23] P. Perez, C. Hue, J. Vermaak and M. Gangnet, "Color-Based Probabilistic Tracking", *European Conference on Computer Vision* (2002) 661-675.

[24] M. Isard and J. MacCormick, BraMBLe, "A Bayesian Multiple-Blob Tracker", *International Conference on Computer Vision (2001)* 34 - 41.

[25] D.A. Forsyth, "Colour constancy and its application in Machine Vision", *Oxford University Press*, 1995.

[26] S. McKenna, Y. Raja and S. Gong, "Tracking Colour Objects Using Adaptive Mixture Models", *Image and Vision Computing 17* (1999) 225 - 231.

[27] *http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html*