

# Vision System for Autonomous Robots: Road to Eurobot 2007

Roberto Aloï, Danilo Treffiletti  
University of Catania, Italy  
roberto.aloi@email.it, treffiletti@urbands.net

**Abstract**—Vision systems are a fundamental component in the process of designing and building a robot.

In this paper we are going to discuss the techniques used to implement the vision system for a “recycling robot”, built to collect bottles, cans and batteries and to separate them into different bins.

The robot, designed to participate to the International competition of robotics “Eurobot 2007”, is able to distinguish the objects according to their shape and color, in a way that is independent from the lighting conditions of the surrounding space.

The algorithm we propose is based on a simple idea: using the CIE  $L^*a^*b^*$  color space to make color information of objects independent from the lighting ones.

The code we are going to treat is entirely written in C language, using the OpenCV libraries.

Sample codes will be furnished while explaining the details of the algorithm used.

**Index Terms**—Computer Vision, Eurobot, OpenCV, CIE  $L^*a^*b^*$ , Color Space.

## I. INTRODUCTION

One of the most troublesome issues in the process of designing and building a robot from scratch is connected with the fact that computers, today, are limited in their ability to interact with the surrounding world because of the lack of the ability to “see”.

Big efforts have been made in the last decades by the scientific community to provide computers with the functions typical of the human vision.

The branch of artificial intelligence whose main purpose is to make an intelligent entity to *see* (read, to *understand* a scene or features in an image) is referred to as *Computer Vision*[1][2][3][4][5].

In other words, *Computer Vision* can be considered as the extreme attempt of reproducing on machines the cognitive path made by man in the process of interpretation of the real world. A concise survey on the recent advances in computer vision can be found in [6].

The component of a robot aimed at the extraction of *features* from a scene and their translation in high-level information is called *Vision System*.

In this paper we report our<sup>1</sup> experience about the design and the implementation of the Vision System for an autonomous robot built to participate to the International competition “Eurobot 2007”.

<sup>1</sup>Our stays for the *DIIT Team* of the University of Catania. More info at: <http://eurobot.diit.unict.it>.

The paper is structured as follows. Section II is a brief introduction to the *Eurobot World*, the environment our robot is going to live in. In this section we will introduce the tasks and the goals our robot was designed to achieve (essential data for the implementation of a well designed vision system). Section III introduces some important theoretical concepts that represent the basis for the next discussions. In section IV we present the vision system in detail. Section V reports our conclusions.

## II. THE EUROBOT WORLD

### A. What Eurobot means

Eurobot is an international robotics contest which involves students, researchers and amateurs from all over the world. Created in 1998 with the name of “French Cup of Robotics”, in 2006 the competition involved 26 Countries, represented by 350 teams.



Fig. 1. The Eurobot Robotics Contest.

Organized in two phases (national qualifications and the international final), the contest consists in a real “tournament” in which the competing robots are dueling in “1 vs 1 challenges” towards the final glory.

Every year, a different robotic game is chosen and a really refined set of playing rules is established. Of course, robots must be absolutely autonomous. Any kind of communication with the robots (both wired or wireless) during the matches is forbidden. Robots have spatial limits, in terms of perimeter, height and so on and they must implement an obstacle avoidance system.

### B. The Eurobot 2007 Edition

This year, the chosen robotic “game” was not properly a game (maybe, did the organizers finish their list of games?).

The robots of the current edition are indeed a sort of *recycling robots*, whose main purpose is *sorting waste* in a given battlefield.

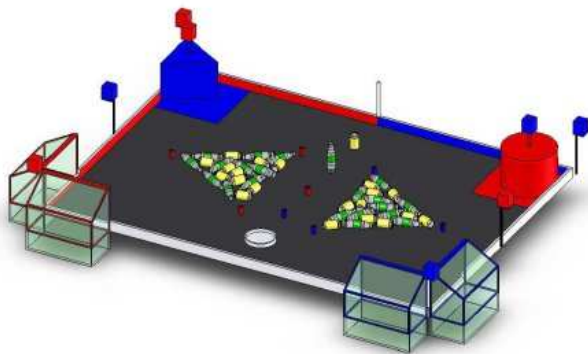


Fig. 2. The Battlefield for the Eurobot 2007 Competition.

There are three kinds of waste the robot have to find: bottles, cans and batteries. Each *class of waste* is distinguished by specific colors (red or blue for the batteries, green for the bottles, yellow for the cans). The robots have to locate the garbage spread all over the battlefield and to transport it into different bins, identified by specific colors, too, and located in some prefixed locations.<sup>2</sup> A representation of the battlefield is given in figure 2.

As you can easily imagine, in a similar kind of world, the Vision System of the robot represents one of the most important components to perform the requested tasks.

### III. THEORETICAL BASIS

In this section we are going to introduce some important theoretical concepts, needed to better understand the following sections. Of course, we are not pretending of summarize the whole theory of vision systems, color spaces and image processing techniques in the following pages. For that, please refer to the relative references.

#### A. Vision Systems

We have already introduced in section I the concept of *Vision System* as “the component of a robot aimed at the extraction of some *features* from a scene and their translation in some kind of high-level information”.

Of course, a vision system and its internal organization are deeply application dependent. Anyway, it is possible to find some main features that are almost independent from the specific implementation and that are actually present in every CV system. In particular, it is possible to identify the following *stages*:

- Image acquisition
- Pre-processing
- Feature extraction
- Detection/Segmentation

<sup>2</sup>More details about the Eurobot contest and its 2007 edition can be found at the contest home page: <http://www.eurobot.org>.

- High-level processing

In the *Image acquisition* stage, a digital image is produced by one or more image sensors, such as cameras. Depending on the type of the image sensor, the image data can be two or three dimensional. The value of each pixel may represent the intensity of the light in one or several spectral bands (gray or color images) or can be related to other physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves, or nuclear magnetic resonance.

In the *Pre-processing* phase, the image grabbed during the previous stage needs to be pre-processed, in order to ensure it satisfies certain assumptions implied by the CV method adopted (such as re-sampling, noise reduction, contrast enhancement, etc).

With the *Feature extraction* step, features of the image like lines, edges, blobs or points are extracted from the image data. The *Detection* of relevant features of the image and the *Segmentation* of the image itself in *sub-images* represent the fourth step and allow the distinction in *relevant* and *not relevant* features for the next stage.

The *High-level processing* is the last step of the computer vision system. The input of this step usually is a small set of data that contains the detected objects of the image. The task of this stage is to extract some high level information about the real objects (like object type, position and size) from the input image.

We will analyze in detail the above stages in the following sections, while studying the algorithm used for the vision system in our robot.

#### B. The CIE $L^*a^*b^*$ Color Space

The algorithm we propose is strictly connected with the concepts of *color space*, *color model* and *human color perception*[7][8].

A lot of studies about color perception, in fact, showed that the human eye has some *photo receptors* to catch *short* (or *S*), *middle* (or *M*) and *Long* (or *L*) waves, better known as blue, green and red photo receptors (the notorious *RGB*). In other words, the color sensation of man is given by an appropriate use of these three parameters. These concepts have been translated for the first time in a more formal way by the *International Commission on Illumination* (or *CIE*) in 1931, with the mathematical definition of the first color space: the *CIE XYZ color space*[9]. One of the variants of the *CIE XYZ color space* is represented by the so called *CIELAB* (to be more precise, the *CIE  $L^*a^*b^*$* ) *color space*, actually considered the most complete *color model* for the description of the full set of colors visible to the human eye.

Let’s try, now, to better understand the structure of this color space and the reasons that moved us to adopt this color space in our vision system.

Unlike the “standard” (and probably better known) *RGB* color space, in the *CIELAB* the three parameters represent:

- Lightness ( $L^*$ )
- Position between magenta and green ( $a^*$ )
- Position between yellow and blue ( $b^*$ )

Going into details, lightness ranges between 0 and 100 ( $L^*=0$  yields *black*,  $L^*=100$  indicates *white*). Negative values for the  $a^*$  parameter represent a movement along the magenta-green axis towards the green, while positive values for  $a^*$  indicate a movement towards the red along the same axis. At the same way, negative values for the  $b^*$  parameter represent a movement along the yellow-blue axis towards the blue, while positive values for  $a^*$  indicate a movement towards the yellow along the same axis. For more clarity, please refer to figures 3 and 4, in which the graphs relative to the LAB color space for two different values of lightness have been reported.

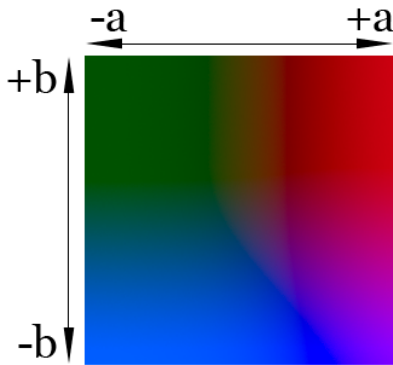


Fig. 3. LAB color space for a 25% lightness.

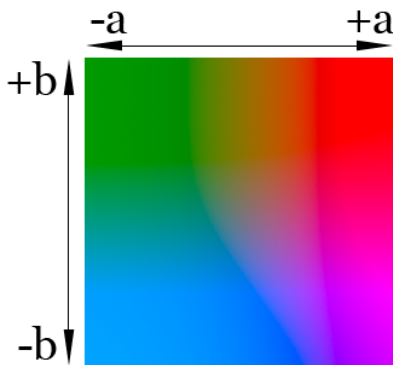


Fig. 4. LAB color space for a 50% lightness.

Unlike the other color spaces, like the RGB or the CMYK ones, the CIE  $L^*a^*b^*$  color space is an *absolute color space*. It has been developed to serve as a device independent model to be used as a reference for the other color spaces. Notice that the LAB model is a three dimensional model and it can only be properly represented in a three dimensional space. Of course, converting images from a color space to another one provokes the raising of some *error*. However, according to the results of the tests performed by Dan Margulis[10], the loss can be considered completely negligible. But why we're going to use such a color space in our algorithm?

The answer to this question is embedded into the nature of the LAB model itself. To better understand this concept, let's have a look to figure 5.

If someone would ask you how many colors are present

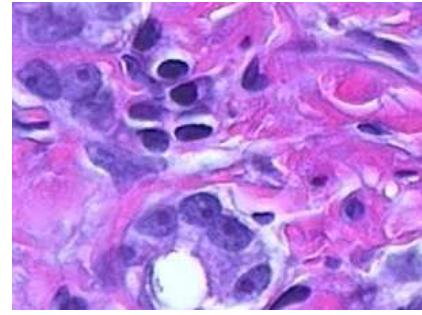


Fig. 5. How many colors can you distinguish in the picture?

within the previous image, in a way that is almost independent from the lighting information, it should be easy for you to say that there are essentially three colors present: pink, blue and white. The CIE  $L^*a^*b^*$  color space essentially has this capability. It can separate the *color information* from the *lighting information*, reducing the color parameters from three (R, G and B) to two ( $a^*$  and  $b^*$ ). Not that bad!

The reason that moved us towards the adoption of this color space in our project are related to the Eurobot 2006 experience[11]. In the previous edition of the competition, in fact, the designed vision system based is color analysis simply on the analysis of the R, G and B channels. The system worked correctly, but it was necessary to set the correct thresholds for the analysis operation for different conditions of lighting. With the new approach we totally overcome the problem.

### C. The OpenCV Libraries

OpenCV is a set of open source computer vision libraries originally developed by Intel<sup>3</sup>. The libraries are *cross-platform* and mainly aimed at real-time image processing. They showed great results in a very huge amount of application areas, ranging from the Human-Computer Interface field to the robotics one, passing through the areas of the biometrics and of the information security. For all of these reasons we decided to use the set of OpenCV libraries in our application. We will comment in section IV the specific OpenCV functions used, while discussing the proposed algorithm.

## IV. THE PROPOSED ALGORITHM

In this section we show the algorithm used to implement the vision system in our robot. The algorithm is entirely written in C language, using the OpenCV libraries. We will furnish some simplified code fragments to better understand the exposed concepts. We decided to structure this section in five subsections, corresponding to the stages we identified in section III, while speaking about the generic vision system.

### A. Image acquisition

The *Image acquisition* phase is devoted to the grabbing process of digital images from the camera. We initially used an *OpenGL Eurobot Simulator*<sup>4</sup> we wrote in Java to generate

<sup>3</sup>More info at: <http://sourceforge.net/projects/opencvlibrary/>.

<sup>4</sup><http://prof3ta.netsons.org>

the needed frames. Then we moved to a real 160 x 120 pixels resolution camera (the one that is actually embedded in the robot).

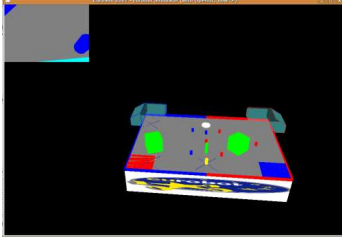


Fig. 6. Our OpenGL Eurobot Simulator.



Fig. 7. The first prototype to test the vision system.

The Image acquisition phase is made up two steps: an initialization and the real grabbing stage. The initialization is performed just once when the system starts. During initialization we set the frame size and rate and other important parameters needed by the OpenCV framework. Then we periodically grab some frames from the camera. Each frame, dealt as a “snapshot” of the external world, is saved into the main memory of the embedded system for further elaborations.

To interact with the camera we used some of the functions from the *setpwc* tool that employ the well known *ioctl* system calls. The acquisition of the image in the OpenCV framework is initialized through the `cvCaptureFromCAM()` function from the libraries. The code used to set the frame size and rate is reported below.

After the initialization phase, the camera is ready to acquire images. Then, two simple OpenCV functions are used for the real image acquisition: the `cvGrabFrame()` and the `cvRetrieveFrame()` functions. The former takes a picture from the camera and saves it into the main memory, while the latter simply returns a pointer to the memory area that contains the image. An example of an acquired image is shown in figure 8.

### B. Pre-processing

In this step, our system essentially converts the *RGB composite image* captured from the camera into a new *LAB composite image*. The camera present in our vision system adopts, like most of the other cameras, displays, printers and scanners, the *absolute color space sRGB*. The conversion from *sRGB* to *CIE L\*a\*b\** is performed in two steps:

```
void set_dimensions_and_framerate
(int fd, int w, int h, int framerate) {
    struct video_window vwin;

    /* get resolution/framerate */
    if (ioctl(fd, VIDIOCGWIN, &vwin) == -1)
        error_exit("VIDIOCGWIN");

    if (w > 0 && h > 0) {
        vwin.width = w;
        vwin.height = h;
    }

    if (vwin.flags & PWC_FPS_FRMASK) {
        /* set new framerate */
        vwin.flags &= ~PWC_FPS_FRMASK;
        vwin.flags |= (framerate<<PWC_FPS_SHIFT);
    }

    if (ioctl(fd, VIDIOCSWIN, &vwin) == -1)
        error_exit("VIDIOCSWIN");
} else {
    fprintf(stderr, "This device doesn't
        support setting the framerate.\n");
    exit(1);
}
}
```

The *set\_dimensions\_and\_framerate* function



Fig. 8. An example shot taken with the embedded camera

- From *sRGB* to *CIE XYZ*
- From *CIE XYZ* to *CIE L\*a\*b\**

Notice that, in the first conversion, the intensity of each *sRGB* channel has to be expressed with a floating point value, in a range between 0 and 1. The value of the intensity in the *CIE XYZ* channels are evaluated with the following formula:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41245 & 0.35758 & 0.18042 \\ 0.21267 & 0.71516 & 0.07217 \\ 0.01933 & 0.11919 & 0.95023 \end{bmatrix} \cdot \begin{bmatrix} f(R) \\ f(G) \\ f(B) \end{bmatrix}$$

where the function  $f(K)$  is defined as follows:

$$f(K) = \begin{cases} \frac{K}{12.92} & K \leq 0.04045 \\ \left(\frac{K+0.055}{1.055}\right)^\gamma & K > 0.04045 \end{cases} \quad \text{for}$$

The  $f(K)$  function is needed to approximate the non linear behavior of the gamma value in the *sRGB* color space. The value we used for  $\gamma$  in the above formula is  $\gamma = 2.2$  and represents the average value for a real display.

In the second conversion, the components of the *reference white point* are defined as:  $X_n = 0.950456$ ,  $Y_n = 1.0$  and  $Z_n = 1.088754$ . The values for the intensities in the *CIE L\*a\*b\** color space are calculated with the following

formulas:

$$L^* = 116 \cdot g(Y/Y_n) - 16$$

$$a^* = 500 \cdot [g(X/X_n) - g(Y/Y_n)]$$

$$b^* = 200 \cdot [g(Y/Y_n) - g(Z/Z_n)]$$

where the function  $g(t)$  is defined in the following way, to prevent an infinite slope at  $t = 0$ :

$$g(t) = \begin{cases} t^{\frac{1}{3}} & t > 0.008856 \\ 7.787 \cdot t + \frac{16}{116} & t \leq 0.008856 \end{cases} \text{ for}$$

The whole transformation from sRGB to CIELAB is achieved through the OpenCV `cvCvtColor()` function. The three parameters of this function represent, in order: the source image, the destination image and a *selector* for the conversion to be applied. The last parameter is set to the `CV_BGR2Lab` OpenCV constant value. The `cvCvtPixToPlane()` function is used to extract three *gray scaled images* from the converted image. These images represent the intensity values for channels  $L^*$ ,  $a^*$  and  $b^*$ . An example of the three extracted images is shown in figure 9.

### C. Feature extraction

Within this stage we identify the sections of the image that contain the target colors. This is achieved through the definition of some *thresholds* applied to the  $L^*$ ,  $a^*$  and  $b^*$  planes obtained in the previous stage. We apply the procedure listed below to each pixel of the original image, creating a new *binary image* for each color we are looking for. The procedure results in a binary image, in which each pixel is white if consistent with the selected thresholds, black otherwise.

The showed code fragment represents only the main structure of the procedure. We actually use an optimized version of that procedure to perform the *feature extraction*. With reference to the code, the `img` object represents the captured image; the `cie_plane_L`, `cie_plane_a` and `cie_plane_b` objects are the three color planes of the image in the **CIE  $L^*a^*b^*$**  color space; the `yellow`, `green`, `blue`, `red` and `white` objects are initially empty images, filled “step by step” during the execution of the algorithm. The properties of each *color object* can be summarized as follows:

- Yellow has a low intensity value on the  $a^*$  plane and an high intensity on the  $b^*$  plane. The value of the  $L^*$  parameter is not significant.
- Green has a low intensity value on the  $a^*$  plane and an high intensity on the  $b^*$  plane, but threshold values are different from the yellow ones. The value of the intensity on the  $L^*$  plane is not significant.
- Blue has a low intensity value on both the  $a^*$  and  $b^*$  planes. The value of the  $L^*$  parameter is not significant.
- Red has a high intensity value on both the  $a^*$  and  $b^*$  planes. The value of the  $L^*$  parameter is not significant.
- White has a mean intensity value on both the  $a^*$  and  $b^*$  planes. The value of the intensity on the  $L^*$  plane is high.

```
for ( x = 0; x < img->width; x++ ) {
    for ( y = 0; y < img->height; y++ ) {
        uint yellow_value;
        uint green_value;
        uint blue_value;
        uint red_value;
        uint white_value;
        uint cieL_value;
        uint ciea_value;
        uint cie_b_value;

        cieL_value = (uint) cvGetReal2D(cie_plane_L, y, x);
        ciea_value = (uint) cvGetReal2D(cie_plane_a, y, x);
        cie_b_value = (uint) cvGetReal2D(cie_plane_b, y, x);

        if( ciea_value < yellow_threshold_a &&
            cie_b_value > yellow_threshold_b ) {
            yellow_value = 255;
        } else {
            yellow_value = 0;
        }
        cvSetReal2D( yellow, y, x, yellow_value );

        if( ciea_value < green_threshold_a &&
            cie_b_value > green_threshold_b ) {
            green_value = 255;
        } else {
            green_value = 0;
        }
        cvSetReal2D( green, y, x, green_value);

        if( ciea_value < blue_threshold_a &&
            cie_b_value < blue_threshold_b ) {
            blue_value = 255;
        } else {
            blue_value = 0;
        }
        cvSetReal2D( blue, y, x, blue_value);

        if( ciea_value > red_threshold_a &&
            cie_b_value > red_threshold_b ) {
            red_value = 255;
        } else {
            red_value = 0;
        }
        cvSetReal2D( red, y, x, red_value);

        if( ( val_ciea > white_threshold_min &&
            val_ciea < white_threshold_max ) && (
            val_cieb > white_threshold_min &&
            val_cieb < white_threshold_max ) &&
            val_cieL > white_threshold_L ) {
            white_value = 255;
        } else {
            white_value = 0;
        }
        cvSetReal2D( white, y, x, white_value);
    }
}
```

Code fragment for the creation of the *binary images*

In figure 10, we report an example of the three *binary images*, representing the Green, Blue and Red colors of the original *composite image* according to our thresholds. Notice that the thresholds, in the robot, are saved in a `.dat` files, accessible from both the vision system (written in C language) and the rest of the software running on the embedded (totally written in Erlang language<sup>5</sup>).

### D. Detection/Segmentation

This step is related to the detection of the **connected components**, present in the binary images we created in the previous step and to the selection of the connected components characterized by a “suitable” size for the system. The

<sup>5</sup><http://www.erlang.org/>

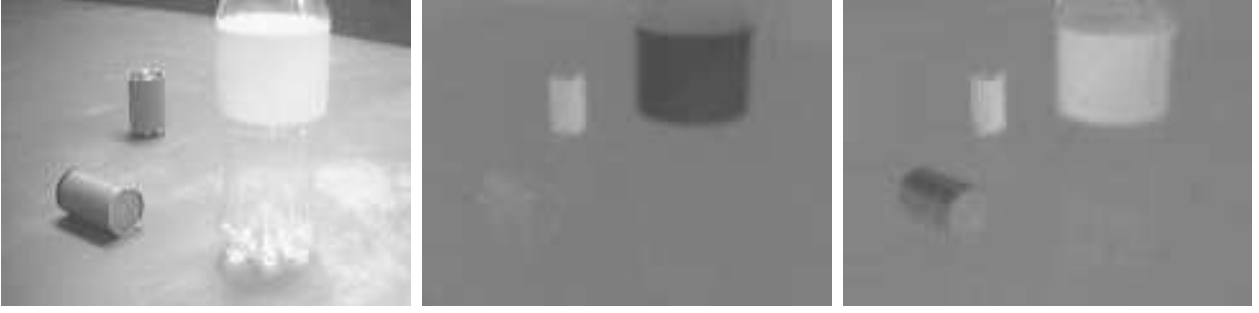


Fig. 9. The  $L^*$ ,  $a^*$  and  $b^*$  planes.



Fig. 10. The Green, the Blue and the Red binary images.

research of connected components is made once for each color binary image using the `cvFindContours()` function. The function parameters are the binary image, some setting parameters and a list of the connected components to be filled. The areas of the detected connected components are evaluated by the `cvContourArea()` function, that computes the area delimited by the borders of the connected component itself. The obtained value is compared with an *area threshold* specific for each class of object. If smaller, the connected component is discarded. Otherwise, we move to the last part of the algorithm.

#### E. High-level processing

This step receives as input the connected components selected during the previous step. The components are classified according to their color. Then, position, size and orientation of the corresponding objects are estimated thanks to the *CAM Shift* algorithm. The *Continuously Adaptive Mean Shift* algorithm is based on the *Mean Shift* algorithm, a robust, non-parametric technique that climbs the gradient of a probability distribution to find the mode of the distribution, used in a repetitive way until one of the exit criteria is reached (maximum number of iterations reached or difference between two subsequent iterations “small enough”). The *CAM Shift* algorithm finds an ellipse surrounding the connected component. The angle between the major axis of the ellipse and the bottom border of the frame can be used to get important information about the actual orientation of the object. Finally, the system retrieves the center of mass of the object using the *moments calculus*, providing us the required position information. The coordinates of the center of mass are calculated with the

following formulas:

$$X_{mass} = \frac{M_{10}}{M_{00}} \text{ and } Y_{mass} = \frac{M_{01}}{M_{00}}$$

To apply the *CAM Shift* algorithm, our system applies a mask on the binary images to select only one object per image, passing the masked binary image to the OpenCV `cvCamShift()` function, together with the bounding box of the connected component and the needed exit criteria. The bounding box is evaluated through the `cvContourBoundingRect()` function, while the moments are found with the `cvMoments()` function. The size of the objects is retrieved from the area (calculated in the previous step), while the size of the track box is evaluated through the *CAM Shift* algorithm.

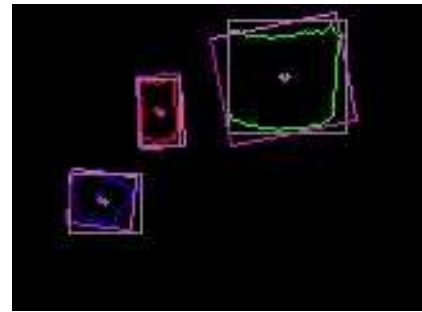


Fig. 11. The detected objects in the scene.

In figure 11 we can see the detected objects. The red, blue and green borders represent the edges of the corresponding detected objects (refer to figure 8 for an easy comparison). The colored points are the centers of mass of the relative objects. The bounding boxes for each connected component are marked

in grey. The magenta boxes represent the estimated orientation of the objects.

The final output of this stage is represented by the information contained in the following table:

Type of object	Area	Angle	X	Y
Bottle	1498	10.68°	7.2 cm	68.2 cm
Blue Battery	346	7.20°	-7.3 cm	42.4 cm
Red Battery	314	86.84°	-5.3 cm	58.7 cm

These data are passed to the embedded system (the real core of the robot) that will take the proper decisions about the strategy to be applied to achieve the requested tasks (i.e. moving towards a specific object, opening or closing the pliers, activating the brushes).

## V. CONCLUSIONS

It should be clear that *there is not an absolute "optimum" vision system*. A Vision System can be valued only within its context. In these terms, we can label a Vision System as "good" if it allows the robot to perform the task it was designed to. In a simplified world as the Eurobot one, we can successfully use an "easy" algorithm as the proposed one because of the assumptions we can do about the external world. If we are moving in a totally different context, we will probably need some "refined" methods.

One of the most delicate issues in the process of designing a vision system for a robot is related to the careful evaluation of the trade off between needs, costs and time. The tests we performed on the vision system of our robot showed great results (as you can notice from the screen-shots contained in this paper), but we need to wait the starting of the final competition of the Eurobot 2007 contest for the definitive proof.

## REFERENCES

- [1] D.H. Ballard and C.M. Brown, *Computer Vision*. Prentice-Hall, 1982.
- [2] H. Barlow, C. Blakemore and M. Weston-Smith (Eds), *Images and Understanding*. Cambridge University Press, 1990.
- [3] A. Basu and X. Li, *Computer Vision: Systems, Theory and Applications*. World Scientific, 1993.
- [4] M. Brady and H.G. Barrow, *Computer Vision*. North-Holland, 1981.
- [5] A. Low, *Introductory Computer Vision and Image Processing*. McGraw Hill, 1991.
- [6] M. Picardi and T. Jan, *Recent advances in computer vision*. The Industrial Physicist.
- [7] M. D. Fairchild, *Color Appearance Models*. Addison-Wesley, Reading, MA, 1998.
- [8] G. Sharma, H.J. Trussell, *Digital Color Imaging*. IEEE Transactions on Image Processing, Vol. 6, No. 7, July 1997.
- [9] CIE, *Commission Internationale de l'Eclairage Proceedings*. Cambridge University Press, 1931.
- [10] D. Margulis, *Photoshop Lab Color: The Canyon Conundrum and Other Adventures in the Most Powerful Colorspace*, Peachpit Press, 2005.
- [11] V. Nicosia, C. Spampinato and C. Santoro for the Eurobot DIIT Team, *Software Agents for Autonomous Robots: the Eurobot 2006 Experience*. CEUR Workshop Proceedings ISSN 1613-0073, Vol. 204, 2006.