Low cost smartcams design

Vincenzo Rana[#1], Matteo Matteucci[#2], Daniele Caltabiano[*3], Roberto Sannino[*4], and Andrea Bonarini[#5]

[#]*Dipartimento di Elettronica e Informazione − Politecnico di Milano*

{[1] rana, [2] matteuc, [5] bonarini}@elet.polimi.it

[*]*STMicroelectronics, Colleoni - La Dialettica*

{[3] daniele.caltabiano, [4] roberto.sannino}@st.com

***Abstract*— Nowadays, digital image processing is the most common form of image processing. Digital image processing makes it possible to enhance image features of interest while attenuating detail irrelevant to a given application, and then extract useful information about the scene from the enhanced image. For instance, digital cameras usually include dedicated digital image processing chips in order to improve at real-time the quality of images directly on-board.**

This paper proposes a very low cost vision system that is able to perform image processing tasks with a wide set of digital image processing algorithms, that have been specifically optimized for the proposed architecture. Some of these algorithms can be used to detect and to track a set of previously acquired targets. Finally, the proposed solution has been proved to be an effective solution even when applied to real-world case studies, such as the detection and the tracking of test-tubes in a medical environment.

I. Introduction

With the rapid evolution of computers and signal processor architectures, digital image processing ([3], [4]) has become the most common form of image processing. Digital image processing, in fact, is generally used not only because it is the most versatile method, but also because it is the cheapest. Digital cameras generally include dedicated digital image processing chips to convert the raw data from the image sensor into a color-corrected image organized in a standard image file format. Images from digital cameras often receive further processing to improve their quality, a distinct advantage digital cameras have over film cameras. Digital image processing is typically done by software programs that can manipulate the images in many ways and that implement methods which would be impossible by analog means.

Aim of this work is the study and the development of a very low cost (less then a half of the comparable solution that can be found in literature) embedded color vision system and the implementation of a set of optimized digital image processing algorithms, that will be used in order to track a target object and to collect information about it. Thus, in addition to the application of the classical digital processing techniques, the images flow will be also enriched with a collection of information that ranges from the position of recognized objects to their color, size or shape. The main contributions of the proposed work are the following:

- the study and the development of a very low cost architecture for digital image processing;
- the implementation of a set of optimized digital image processing algorithms to be used on the proposed architecture;
- the development of an embedded color vision system that has been successfully applied to real-world case studies.

The remaining of the paper is organized as follow. Section II presents similar works that can be found in literature. Section III introduces the prototyping platform, while Section IV presents some optimized digital image processing algorithms. Finally, Section V shows some experimental results and Section VI draws some concluding remarks.

II. State of the art

A lot of works have been performed on low cost embedded vision systems (such as [6], [5]), and the most interesting one is the CMUcam ([2], [1], [7]), originally developed at Carnegie Mellon University in 2002. This camera has been developed in three different versions: CMUcam1, CMUcam2 (in 2004) and CMUcam3 (in 2007).

CMUcam1 is a low-cost, low-power sensor for mobile robots. It is possible to use CMUcam1 vision system in order to perform many different kinds of on-board, real-time vision processing, with a maximum resolution of 80x143. Since CMUcam1 uses a serial port, it can be directly interfaced to other low-power processors such as *PIC* microprocessors. Using CMUcam1, it is easy to make a robot head that swivels around to track an object. It is also possible to build a wheeled robot that chases a ball around. In particular, at 17 frames per second, CMUcam1 is able to track the position of a colorful or bright object, to measure the RGB or YUV statistics of an image region, to automatically acquire and track the first object it sees, to physically track using a directly connected servo and to dump a complete image over the serial port or a bitmap showing the shape of the tracked object.

The CMUcam2 has been developed in order to overcome a few limits of the previous version (CMUcam1). In particular, the CMUcam2 is characterized by a maximum resolution of 176 x 255 and it is able to track user defined color blobs up to 50 frames per second (even if frame rate strictly depends on resolution and blob size settings). Furthermore, it is also able to track motion using frame differencing at 26 frames per second, to find the centroid of any tracked data, to transfer a real-time binary bitmap of the tracked pixels in an image, to automatically use servos to do two axis color tracking and to perform multiple pass image processing on a buffered image.

Since the goal of the CMUcam project is to provide simple vision capabilities to small embedded systems in the form of an intelligent sensor, the CMUcam3 extends upon this idea by providing a flexible and easy to use open source development environment that complements a low cost hardware platform. The CMUcam3 is an ARM7TDMI based fully programmable embedded computer vision sensor. The main processor is the NXP LPC2106 connected to an Omnivision CMOS camera sensor module. Custom C code can be developed for the CMUcam3 using a port of the GNU toolchain along with a set

of open source libraries and example programs. Executables can be flashed onto the board using the serial port with no external downloading hardware required. The CMUcam3 is characterized by a Common Intermediate Format (CIF) resolution (352x288) and is able to process images at a rate of 26 frames per second, to perform software JPEG compression, to detect user defined color blobs, to perform frame differencing and to dump raw images over the serial port.

III. Prototyping platform

The prototyping platform used in this work is the first of a set of modules designed by *STMicroelectronics* for robotics applications. A schema of this platform, called RoboVision1, can be found in Figure 1. It is a low cost vision system for acquisition and processing either at high frame-rates (up to 30fps) but with a low resolution (QQVGA) or, vice versa, at high resolutions (such as UXGA) but with a low frame-rate (0.5 fps).

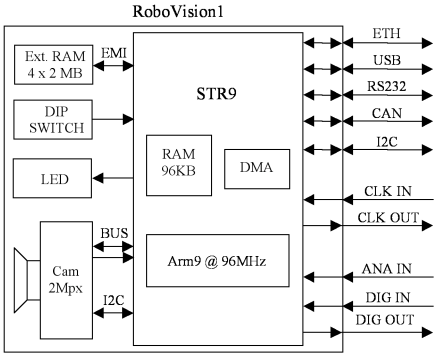

Fig. 1. Prototyping platform schema

The system, as shown in Figure 2, is mainly composed by a microcontroller, ST STR9, and a powerful camera, ST VS6724. The STR9 microcontroller has a 32bit ARM9 core (with a working frequency of 96MHz), with 96KB of embedded RAM, 512KB of flash and I/O peripherals, such as Ethernet 10/100, Full Speed USB, UARTs, I2Cs, SPIs, ADCs, CANs, GPIOs. The VS6724 is a UXGA resolution CMOS imaging device designed for low power systems, such as mobile phone applications. Manufactured using ST 0.18 μm CMOS Imaging process, the VS6724 integrates a high-sensitivity pixel array, a digital image processor and camera control functionalities. The VS6724 is capable of streaming UXGA video up to 30 fps JPEG and up to 15fps in uncompressed format. The maximum framerate is again 30 fps in uncompressed colored video streaming up to SVGA. The video data is output over an 8-bit parallel bus in JPEG (4:2:2 or 4:2:0), RGB, YCbCr or bayer formats while the VS6724 is controlled via an I^2C interface.

The parallel BUS of the Camera is acquired by the STR9 via a GPIO port and a DMA channel. The DMA is enabled for every frame in the Interrupt Service Routine (ISR) of the Fast IRQ connected to the camera VSYNC signal (rising edge). This signal, indeed, is high during the transmission of the frame and remains low when the bus is idle. The DMA is configured to read a specific amount of bytes from the DMA

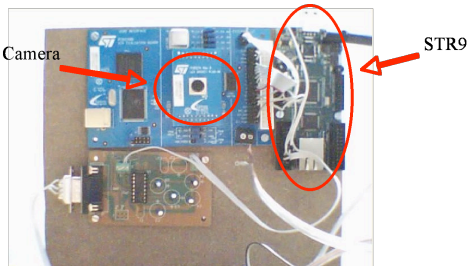

Fig. 2. Prototyping platform image

which should correspond to the image size, and copy them in a specific buffer, located in a shared variable. For each acquired row, the DMA generates an interrupt. This increases the counter indicating the actual number of rows read which can be managed by the processing algorithm. The maximum working frequency of the DMA acquisition system is 2.4 MHz which allows video formats presented in Table I to be streamed.

TABLE I
SUPPORTED VIDEO FORMATS

	Res1	Res2	Byte /Pixel	Img size (KBytes)	fps	Compr	PCLK (MHz)
Color JPG	1600	1200	2	3840	5	0.1	2.11
	1280	1024	2	2621	8	0.1	2.31
	800	600	2	960	22	0.1	2.32
	640	480	2	614	30	0.1	2.03
Color RAW	320	240	2	154	14	1	2.37
	240	180	2	86	25	1	2.38
	220	165	2	73	30	1	2.40

Since gray scale image dimensions are half the corresponding colored ones, their frame-rates are almost doubled. The Images are stored either in the internal SRAM (if smaller then 90KB) or in the External RAM via the External Memory Interface (EMI). If the image is compressed in JPEG format, it should be uncompressed prior to be processed, while, on the other hand, if the images are in raw format, they can be processed during the acquisition process row by rowFor this reason, the mentioned frame-rate can be respected only when small raw images are processed or when the stream has to be directly forwarder through the USB or Ethernet interfaces.

The camera integrates both a scaler and a cropper module which add Pan, Tilt and Zoom (PTZ) functionality to the overall system. The Horizontal Field Of View (H-FOV) of the camera is 50 degrees. The system has a specific input pin called CLK_IN which is used to synchronize the camera and other robotics modules with a common clock.

The system has been tested on a small robot, shown in Figure 3, in order to recognize and follow a red ball. The system was able to process a QQVGA (160x120) video stream at 25 fps, finding the centroid of the ball in the current image and moving the robot in order to maintain the ball in the center of the camera view.

IV. Digital image processing algorithms

In order to perform on-board digital image elaborations, a large set of digital image elaboration algorithms has been analyzed and implemented on the prototyping platform (they

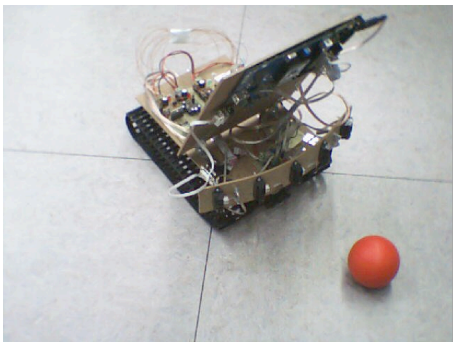

Fig. 3. Image of the robot

have been developed for the embedded STR9 microcontroller). This section aims at presenting some of these algorithms, in order to better explain how the target smart camera has been obtained. In particular, the following algorithms, that have been implemented in order to perform the on-board processing, will be described:

- **Background subtraction and foreground segmentation**, described in Section IV-A, that allows to achieve an image in which the background has been completely removed, in order to perform further elaborations only on the objects that are not part of the background. This elaboration can be performed both in a static scenario (in which the background that has to be removed from new images is always the same) and in a dynamic one (in which the background can vary, even if it has to be characterized by different colors with respect to the object that has to be tracked).
- **Blob detection**, described in Section IV-B, that allows to recognize the number, the shapes and the positions of the objects that are present in a particular image. In order to identify only target objects, it is possible to process the original image with the background subtraction algorithm, before performing the blob detection algorithm.
- **Object tracking**, described in Section IV-C, that can be performed after background subtraction and blob detection, in order to track a particular object between all the objects that are present in the images flow. This algorithm allows the detection of the position of a particular object even if another similar object is present in the same image.

A. Background subtraction and foreground segmentation

In the proposed approach, in order to cope both with the complexity of the task and with the limited resources of the prototyping platform, a couple of simple but effective algorithms (both based on color difference between the background and the objects that have to be tracked) have been developed, as shown in Figure 4.

The first algorithm can be used only in a scenario where the background can be considered static. In this case, after the initialization of the system (Figure 4 (A)), it is possible to acquire the static background (Figure 4 (C)) and to store it into the main memory (Figure 4 (E)) (the size of the image to be stored strictly depends on its resolution). The second step consists of a loop in which a new image is acquired by the cam

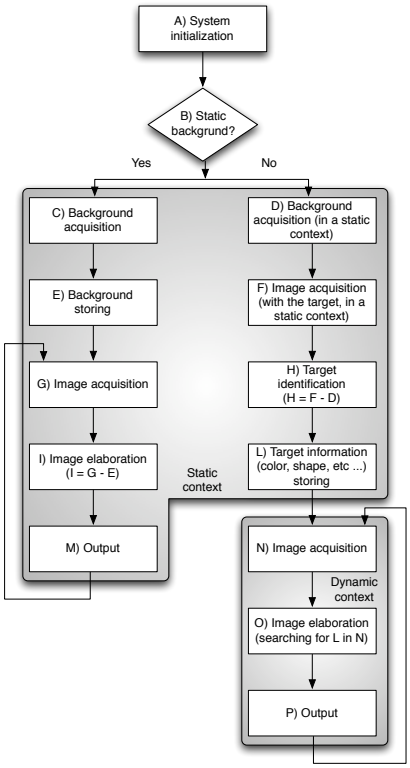

Fig. 4. Background subtraction and foreground segmentation algorithms

(Figure 4 (G)) and then elaborated (Figure 4 (I)) by subtracting the background stored in Figure 4 (E) from the image acquired in Figure 4 (G). Thus, the output can be generated and the loop can start again. When it is not possible to consider a static background, it is necessary to acquire information about the objects that have to be tracked, in order to remove from new images all the pixels that do not correspond to the desired targets. In this way it is possible to perform a foreground segmentation task while the background dynamically changes. As shown in Figure 4, the first step that has to be performed consists of the acquisition of the background (Figure 4 (D)) and of the image including the target object (Figure 4 (F)) in a static context. In this way, it is possible to identify the target object (Figure 4 (H)) by subtracting the background acquired in Figure 4 (D) from the image acquired in Figure 4 (F) and to store its information (such as size, shape, color, etc...) into the main memory (Figure 4 (L)). At this point, after the target profiling phase, it is possible to enter into a dynamic context, in which the background can dynamically change. In this new context it is possible to perform a loop in which a new image is acquired (Figure 4 (N)) and then elaborated (Figure 4 (O)). This last operation is performed by searching in the new image the previously profiled target object, exploiting information such as the color of the pixels of the target object. Finally, in the last step the image is transferred to the next image processing algorithm (Figure 4 (P)).

B. Blob identification and detection

The algorithm that has been used to implement the blob identification and detection phase is shown in Figure 5 and essentially consists of 2 phases.

In the first phase (Figure 5 (B)), an image elaborated, for instance by the background subtraction algorithm, can be processed (with a single scansion of the image) in order to assign to adjacent pixels belonging to the same objects the same identification number; after this phase, all the pixels identified by the same identification number belong to the same object, but not all the pixels belonging to the same object are identified by the same identification number (at least another scansion of the image is required in order to obtain this result). Anyway, in the first phase it is possible to compute a set of lists of identification numbers that refer to the same blob: this set of lists will be used in the next phase in order to assign the same blob identification letter to all the pixels belonging to the same object.

The second phase (Figure 5 (C)) performs a second scansion of the image, in order to map all the identification numbers that refer to the same object to the same blob identification letter. After this phase, the elaborated image consists of a set of objects identified by the same identification letter and can be transferred to the next image processing algorithm for further elaboration.

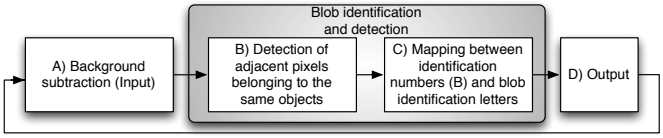

Fig. 5. Blob identification and detection algorithm

In order to better explain the proposed approach, let us consider a simple example of an image that consists of 144 pixels (a 16x9 image). A simple image consisting of 4 objects with different shapes is presented in Figure 6 (2 of them share the same color). This image can be the result of a background subtraction phase that has been performed in order to set all the pixels belonging to the background to a predefined value (blank in this case).

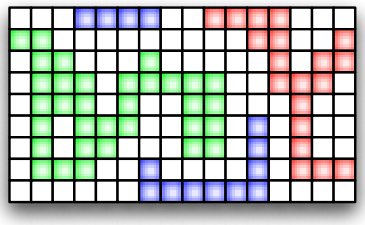

Fig. 6. A simple example of a 16x9 image

The first step of the blob detection algorithm presented in this work is the detection of adjacent pixels belonging to the same objects. This phase can be performed by assigning a specified value to each pixel in the following way:

- the pixel on the top left corner of the image will receive either a 0, if it belongs to the background, or a 1, if it belongs to an object;
- any other pixel of the first top row will receive:
  - a 0 if belonging to the background;
  - the same value of the adjacent pixel on its left if both the pixels belong to an object;
  - a new value (never used before on the same image) if it belongs to an object but the adjacent pixel on its left belongs to the background;
- any other pixel of the other rows will receive:
  - a 0 if belonging to the background;
  - the same value of the adjacent pixel on its left if both the pixels belong to an object;
  - the same value of the adjacent pixel on its top if both the pixels belong to an object, but the adjacent pixel on the left of the lower one belongs to the background;
  - a new value (never used before on the same image) if it belongs to an object but the adjacent pixel on its left belongs to the background.

Figure 7 shows the result (0 values have been omitted and substituted with blank pixels) that can be obtained by applying the first step of the blob detection algorithm to the image presented in Figure 6.

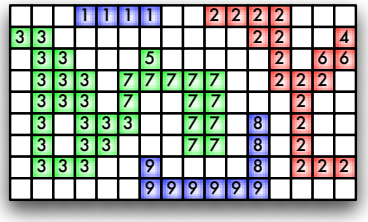

Fig. 7. First step of the blob detection algorithm on a 16x9 image

The second step of the algorithm is the mapping between identification numbers (values assigned in the previous phase) and blob identification letters.

As shown in Figure 8, the object identified by the value 1 can be directly mapped to the object A, while the object B includes all the pixels with value 2, 4 and 6. Object C can be identified by putting together all the pixels with number 3, 5, and 7, while object D consists of all the pixel with value 8 and 9. Figure 9 shows the result that can be obtained by applying

```
A -> 1
B -> 2, 4, 6
C -> 3, 5, 7
D -> 8, 9
```

Fig. 8. Mapping between identification numbers and blob identification letters

the second phase of the blob detection algorithm to the image presented in Figure 7. As it is possible to see in Figure 9, all the objects have been correctly detected and identified.

C. Object tracking

The first step of the proposed object tracking algorithm, as shown in Figure 10, is the evaluation (Figure 10 (C)) of the list of blobs obtained in the blob identification and detection phase (Figure 10 (B)). If this list is empty (either no blobs have been detected or all the detected blobs have been already processed), then it is possible to generate the output (Figure 10 (D)), otherwise a blob of the list has to be selected and evaluated (Figure 10 (E)), with respect to its color,

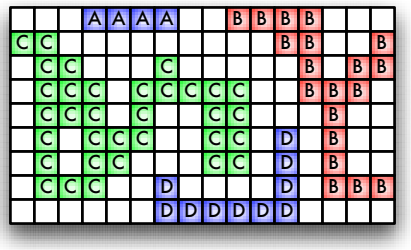

Fig. 9. Second step of the blob detection algorithm on a 16x9 image

shape, position, etc... This evaluation is necessary in order to understand if the selected blob represents the previously acquired primary target (Figure 10 (F)) (it can be either the target acquired during the background subtraction phase with a dynamic background or the first blob introduced in the scene - in this case the blobs list will contain only a single blob, that will be the primary target). If the selected blob has been detected to be the primary target (for instance, its color, shape and position match with a certain percentage the color, shape and position of the previously acquired primary target), then it is possible to update the currently selected primary target and to remove the blob from the blobs list (Figure 10 (G)). Otherwise, if the target has been detected to be a secondary target, it is possible to update the secondary targets list and to remove the blob from the blobs list (Figure 10 (H)).

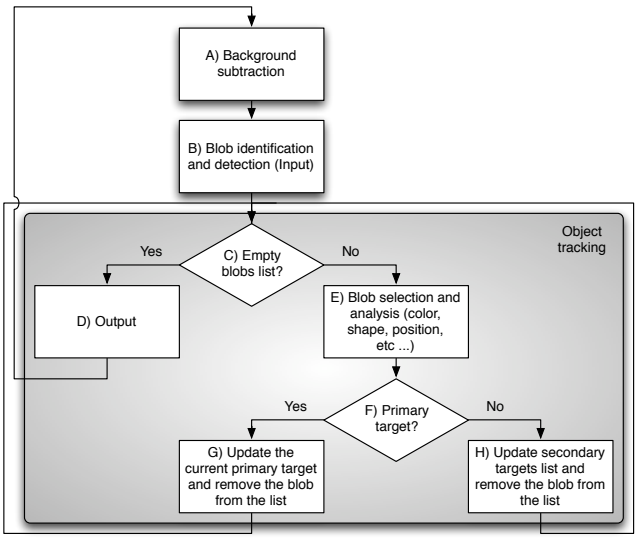

Fig. 10. Object tracking algorithm

V. Experimental results

One of the real scenarios in which the proposed low cost smartcam has been exploited in order to perform a recognition task is the test-tubes detection task. Each test-tube, that differs from another one for the shape, the size or the color of the cap, has to be treated in a different way with respect to its characteristics; this convention is very useful both to avoid to perform medical tests (that can also be very expensive) on wrong test-tubes and to avoid the contamination of the content of a test-tube by performing a wrong medical test.

This is a suitable scenario in which a low cost smartcam can be employed. In this work, the following couple of experiments with test-tubes are presented:

- detection of test-tubes presence or absence, that will be described in Section V-.1;
- tracking of the desired test-tube, that will be described in Section V-.2.

1) Test-tubes detection: As an example, let us consider Figure 11 (A), in which some red test-tubes have been put into a small container. If the desired behaviour is the recognition of the location of red test-tubes in the small container, it is possible to use the proposed smartcam to perform the algorithms presented in the previous sections, in order to obtain the desired information (that corresponds to the detection of test-tubes presence or absence).

Figure 11 (B) shows an image generated by the smartcam as an output of the elaboration of Figure 11. The image elaborated by the smartcam shows that the position of red test-tubes have been correctly identified.

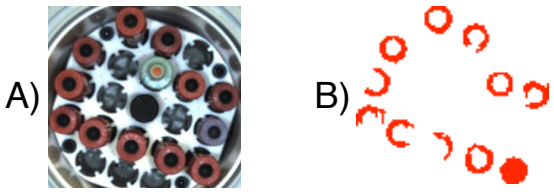

Fig. 11. The small container filled with some red test-tubes (A) and the image elaborated by the smartcam in order to locate the red test-tubes (B)

A second test has been performed on the same small container after that the container has been subjected to a rotation, as shown in Figure 12 (A). In Figure 12 (B) it is possible to see how the smartcam is still able to correctly detect the position of all the red test-tubes, simply processing the new image, acquired after the rotation process.

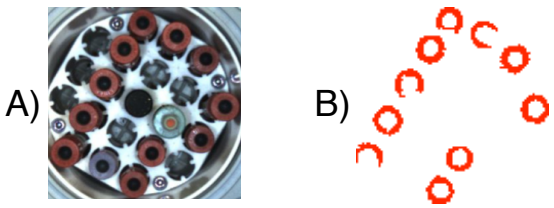

Fig. 12. The small container filled with some red test-tubes, after the rotation (A) and the new image elaborated by the smartcam in order to locate the red test-tubes (B)

2) Test-tube tracking: The second experiment has been performed by moving the smartcam over a rack of test-tubes, in order to detect and to track the violet-purple ones. The smartcam has been moved from bottom to up, passing through the following positions, as shown in Figure 13: V, W, X, Y and Z.

In position V and W, the smartcam is not able to detect any violet-purple test-tube (since no violet-purple test tube is present in this portion of the rack), so the image elaborated by the smartcam in this position is a completely blank (that means empty) image.

When passing through position X, as shown in Figure 14 (A), the smartcam detects one violet-purple test-tube and

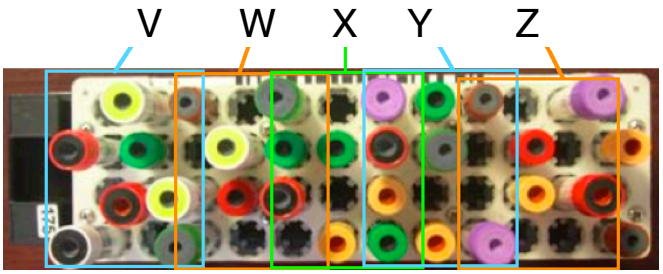

Fig. 13. Image of the rack of test-tubes

acquires it as the primary target that has to be tracked. After the detection, the smartcam starts to track the violet purple test-tube, as shown in Figure 14 (B), in which the test-tube is represented by a red (that means it is the primary target) ring.

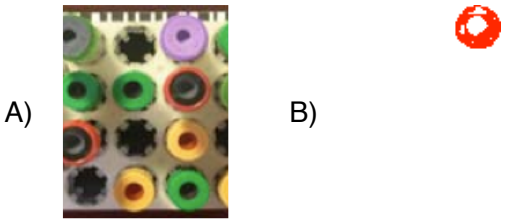

Fig. 14. Image of position X (A) and image elaborated by the smartcam (B)

When passing through position Y, as shown in Figure 15 (A), the smartcam detects a second violet-purple test-tube on the top right corner of the image (while the first violet-purple test-tube is located on the bottom left corner of the image). As shown in Figure 15 (B), the smartcam is able both to track the previously acquired violet-purple test-tube (the primary target represented by a red ring) and to detect the new violet-purple test-tube as a secondary target (represented by a green ring).

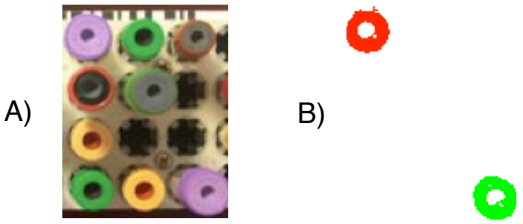

Fig. 15. Image of position Y (A) and image elaborated by the smartcam (B)

A. State of the art comparison

Table II summarizes the comparison between CMUcam1, CMUcam2, CMUcam3 and the proposed low cost smartcam. The target price of the proposed smartcam is definitely below 100$, that means less than the cheapest CMUcam. The first version of the CMUcam, in fact, costs approximately 109$, while the price of the second version, the CMUcam2, is around 179$. Even if the price of the proposed smartcam is smaller than the price of CMUcam1 and CMUcam2, with respect to the technological aspect it is comparable with the CMUcam3. The third version of CMUcam, in fact, is provided with 1 MegaBytes of internal memory (instead of the 90 KiloBytes of the proposed smartcam), but presents an ARM 7 processor (while the proposed smartcam is provided with an ARM 9 processor) and a camera with a lower resolution, with respect to the one proposed in this work (352x288 versus 1600x1200). Finally, the proposed smartcam is able to operate at a higher frame-rate w.r.t. the CMUcam3 and also supports JPEG compression in hardware.

TABLE II
SMARTCAMS COMPARISON

	CMUcam1	CMUcam2	CMUcam3	Proposed smartcam
Frame differencing	No	Yes	Yes	Yes
Maximum resolution	80 x 143	176 x 255	352 x 288	1600 x 1200
Image windowing	No	Yes	Yes	Yes
JPEG support	No	No	Yes (SW)	Yes (HW)
Blobs detection	No	Yes	Yes	Yes
Maximum frame-rate (FPS)	17	50	26	30
Price	109$	179$	239$	$\leq$ 100$

VI. CONCLUDING REMARKS

As shown in Section V, the proposed smartcam has been proved to be a suitable solution for real-world case studies, since it is able to correctly detect, recognize and track the desired object, both in a static and in a dynamic background context.

Furthermore, the proposed smartcam is comparable to the CMUcam3 both from the technological point of view and from the functional point of view, as previously discussed in Section V-A, but it will cost less then the half of the price of the CMUcam3 (which cost is around 239$).

Finally, as it is possible to see in Table II, and for the previously discussed reasons, the proposed smartcam has been proved to be an effective and firmly low cost solution for the detection and the tracking of objects, based on the elaboration on their color, position, shape and size.

ACKNOWLEDGMENTS

This work was partially supported by the HiPEAC network of excellence (www.hipeac.net).

REFERENCES

[1] Illah Nourbakhsh Anthony Rowe, Charles Rosenberg. A simple low cost color vision system. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2001.

[2] Illah Nourbakhsh Anthony Rowe, Charles Rosenberg. A low cost embedded color vision system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[3] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach. Distributed embedded smart cameras for surveillance applications. *Computer*, 39(2):68–75, Feb. 2006.

[4] K.I. Kiy and E.D. Dickmanns. A color vision system for real-time analysis of road scenes. *Intelligent Vehicles Symposium, 2004 IEEE*, pages 54–59, June 2004.

[5] M. Kontitsis, K.P. Valavanis, and R. Garcia. A simple low cost vision system for small unmanned vtol vehicles. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3480–3486, Aug. 2005.

[6] Dong Li, Yunsheng Jiang, and Guiyou Chen. A low cost embedded color vision system based on sx52. *Information Acquisition, 2006 IEEE International Conference on*, pages 883–887, Aug. 2006.

[7] A. Rowe, C. Rosenberg, and I. Nourbakhsh. A second generation low cost embedded color vision system. *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, 3:136–136, June 2005.