

Proceedings of the 3rd International Conference
"from Scientific Computing to Computational Engineering"
(3rd IC-SCCE), 9 - 12 July 2008, Athens, Greece.

EMBEDDED VIDEO PROCESSING FOR DISTRIBUTED INTELLIGENT SENSOR NETWORKS

Jose E. Simo, Gines Benet, Gabriela Andreu
Instituto de Automática e Informática Industrial,
Universidad Politécnica de Valencia., Camino de Vera, s/n,
46022, Valencia, España
e-mail: {jsimo, gbenet, gandreu@disca.upv.es}

Abstract

In this paper the theoretical background of specific video processing algorithms is presented as well as the details of their implementation into SENSE(*) nodes of a distributed sensor network devoted to airport surveillance tasks. The video processing algorithms are implemented using embedded DSP boards, based on Blackfin DSP of Analog Devices vision system processing architecture. This architecture is formed by two main processing systems whose communication is realized via serial port. Each processor supports different and exclusive tasks that determine their process and peripheral features. The paper describes also the implementation details of the visual algorithms into the embedded architecture of the visual module and the preliminary results obtained.

Acknowledgments

This work has been partially supported by SENSE project (Specific Targeted Research Project within the Thematic priority IST 2.5.3 of the 6th Framework Program of the European Commission: IST Project 033279), by the Spanish Government by the complementary funding TIN2007-30367-E, and Conselleria d'Empresa, Universitat i Ciència of the Regional Government under the grant ACOMP/2007/205.

SENSE



EMBEDDED VIDEO PROCESSING FOR DISTRIBUTED INTELLIGENT SENSOR NETWORKS

Jose E. Simo¹, Gines Benet¹, Gabriela Andreu¹

¹ Instituto de Automática e Informática Industrial, Universidad Politécnica de Valencia., Camino de Vera, s/n,
46022, Valencia, España
e-mail: {jsimo, gbenet, gandreu@disca.upv.es}

Keywords: Video processing,, Intelligent Sensor Networks, Embedded Systems.

Abstract. *In this paper the theoretical background of specific video processing algorithms is presented as well as the details of their implementation into SENSE^(*) nodes of a distributed sensor network devoted to airport surveillance tasks. The video processing algorithms are implemented using embedded DSP boards, based on Blackfin DSP of Analog Devices vision system processing architecture. This architecture is formed by two main processing systems whose communication is realized via serial port. Each processor supports different and exclusive tasks that determine their process and peripheral features. The paper describes also the implementation details of the visual algorithms into the embedded architecture of the visual module and the preliminary results obtained.*

1 INTRODUCTION

This paper describes the current status of the implementation of the specific software that runs into the video module board of an intelligent sensor that is currently being developed under the SENSE project. The goal of the current application of the SENSE node is to detect and track people and objects standing or moving in determined areas of an airport. The specific objectives for low level visual processing in this SENSE application are:

- To represent the location of visual objects in image coordinates.
- To represent the contour and appearance of the visible regions of the visual objects as features.
- Low level classification of objects into three categories: person, person group and non-classified object .

Some considerations to be taken into account for the low level visual processing are:

- The scenes are dynamic
- Indoor scenes have illumination changes
- In the same scene, very illuminated and dark areas may appear.
- The background and foreground are dynamic.
- Soft real time processing

The video board software includes vision specific algorithms as well as the operating system code (including device drivers, interrupt handling code, hardware configuration code and kernel code). These programs must enable an adequate processing of the video images obtained from the camera, obtaining the video features required by the higher level processing board. JPEG video compression must be also carried out into the video board, as well as the XML stream generation and transmission to the reasoning board.

The document is divided into two main parts: First, the main vision algorithms developed are described in section 2 .Next, the details of the implementation of the above described algorithms and some preliminary performance results obtained with real boards are also explained in sections 3 and 4, respectively.

^(*) : SENSE stands for 'Smart Embedded Network of Sensing Entities' (Project 033279), currently developed under a grant from European Sixth Framework Programme Priority IST 2.5.3 Embedded Systems.

2 LOW LEVEL VISUAL PROCESSING IN SENSE NODES

Visual surveillance, either indoors or outdoors, is an active research topic in computer vision and various surveillance systems have been proposed in recent years: [8], [2], [3], [9]. The visual surveillance process may be divided into the following steps: environment modeling, motion detection, object classification, tracking, behavior understanding, human identification and data fusion. The general framework proposed for video processing in SENSE project is shown in Figure 1.

From the SENSE point of view, the video modality passes information to upper levels in the form Low Level Symbols (LLS). At low level, LLS are elaborated to include a draft classification into reduced pre-defined subsets matching the limited number of visual objects that can be detected. Therefore, the video processing is not focused to directly solve the problem of video surveillance in airport areas. Instead of that, the low level processing is focused to provide to high processing level with a simple object's description in the scene (including object's classification and features).

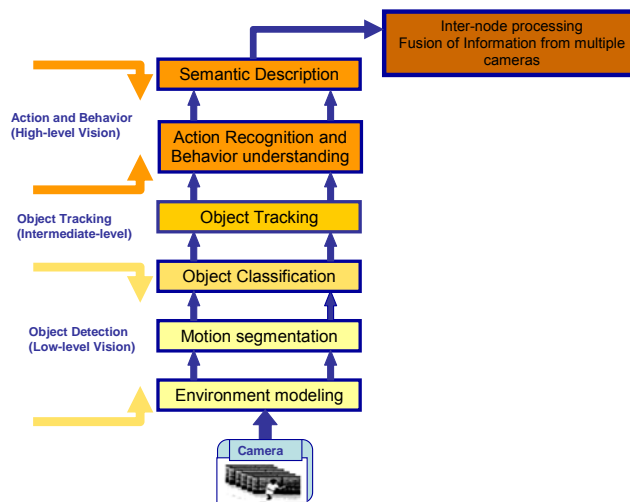


Figure 1. General Framework of low level visual modality.

In our application, (airport surveillance), attention must be paid to individuals standing alone, groups of people, and luggage. The system consists of a set of cameras with running low-level vision algorithms, covering the complete airport. One of their tasks is classifying objects in each frame by considering just the information gathered through the camera and a limited local history, if any. This classification is then fed to a higher level which controls several cameras and fuses all the incoming data. Low Level Visual Processing is responsible of object's identification and feature extraction while upper levels of semantic processing and multimodal integration are in charge of evaluating the behavior of detected objects. In the next subsections, the main video processing algorithms that undertake the low level visual processing will be described.

2.1 MAIN VIDEO PROCESSING ALGORITHMS

Some techniques and algorithms intended to low level visual processing have been used to perform the necessary steps to detect target objects in a scene, track them and classify them. The implemented algorithms have been chosen with the aim of being fast, but without forgoing the accuracy. Also, though most of them do not require parameters in order to allow them to be more flexible, some parameters are needed to constraint the working operation of the surveillance process. These algorithms are introduced in the next subsections.

Environment Model and Adaptive Background

The environment model involves tasks aimed to model the environment in which action will take place. Producing a precise image reference is a crucial task in computer vision applications such as video surveillance, especially when background subtraction is used as the first step for several moving object detection algorithms. The performance of other processes, as segmentation and tracking, included in the surveillance system depends on having a good model. The image model [1], [4] approach was the one chosen in our implementation of the background modeling; this decision was aimed to reduce memory requirements and to simplify the implementation. Also, it must be emphasized that most implementations found in the literature do usually use

this approach.

Background modeling starts by creating a first background model (B_f) which is adapted over time to cope with light changes. This first model (B_f) may be constructed by two different ways:

- a) The first model (B_f) may be created by taking a frame with no object (just those belonging to the background), and considering this first frame (F_1) of the sequence as the model.

$$B_f(p_{ij}) = F_1(p_{ij}) \quad \forall p_{ij} \in F_1$$

- b) The first model (B_f) may be obtained based on a sequence of initial and consecutive input frames ($F_1, F_2, F_3, \dots, F_t$) containing no moving objects of the scenario where surveillance is to be made, and process the mean or mode of these images.

$$B_f(p_{ij}) = \text{mean}[F_1(p_{ij}), F_2(p_{ij}), F_3(p_{ij}), \dots, F_t(p_{ij})]$$

$$B_f(p_{ij}) = \text{average}[F_1(p_{ij}), F_2(p_{ij}), F_3(p_{ij}), \dots, F_t(p_{ij})]$$

$$B_f(p_{ij}) = \text{mode}[F_1(p_{ij}), F_2(p_{ij}), F_3(p_{ij}), \dots, F_t(p_{ij})]$$

In order to optimize the resources and to obtain a representative model it can be used a sequence of frames periodic on time but not necessarily consecutive, as $F_1, F_{1+x}, F_{1+2x}, \dots, F_{1+tx}$.

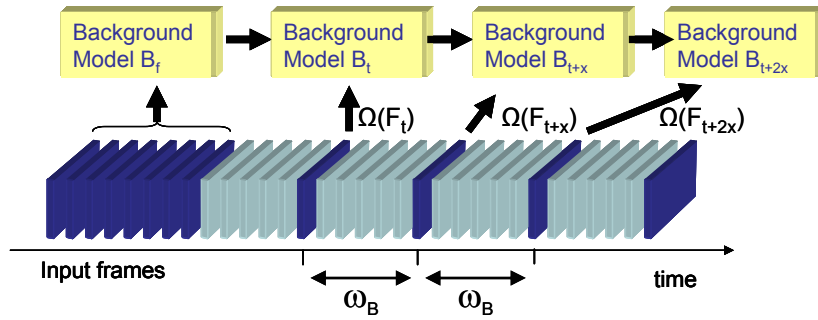


Figure 2. Background update model.

Once the first model B_f is created through any of the above mentioned methods, it is updated automatically after a certain time (or number of frames) to introduce the illumination changes in the scene into the model, see Figure-3. This update is performed according to a law which takes into account the current values of the model B_t and the background pixel of last frame acquired $\Omega(F_t)$. This can be expressed as:

$$B_{t+1}(p_{ij}) = \begin{cases} B_t(p_{ij}) & \forall p_{ij} \notin \Omega(F_t) \\ B_t(p_{ij}) + \alpha (F_t(p_{ij}) - B_t(p_{ij})) & \forall p_{ij} \in \Omega(F_t) \end{cases}$$

In previous formula, B_t and B_{t+1} are the previous and newly calculated background models; $\Omega(F_t)$ is an image which only contains the background pixels of the last frame acquired (the foreground pixels are not updated) and α is the background updating rate. Values of α can vary in the range $[0,1]$, typically, values between 0.02 and 0.05 provide good results. Also, laboratory experiments using $\omega_B = 5$ show good results. However, it is still an open line to determine whether these parameters, α and ω_B , should be predefined in the application or should adapt themselves to illumination evolution.

Motion segmentation

Motion segmentation in image sequences aims at detecting regions corresponding to moving objects such as vehicles and humans [2]. Detecting moving regions provides a focus of attention for later processes such as tracking and behavior analysis because only these regions need be considered in the later processes. At present, most segmentation methods use either temporal or spatial information in the image sequence.

We have implemented a hybrid algorithm [3], for detecting moving objects, by combining background subtraction and temporal differences using three frames. Figure 3 describes the implemented algorithm. If pixels detected by using subtraction and pixels detected by using motion detection are joined into an image, this image will contain all regions which are candidates to be tracked (these regions, from now on, will be called "blobs").

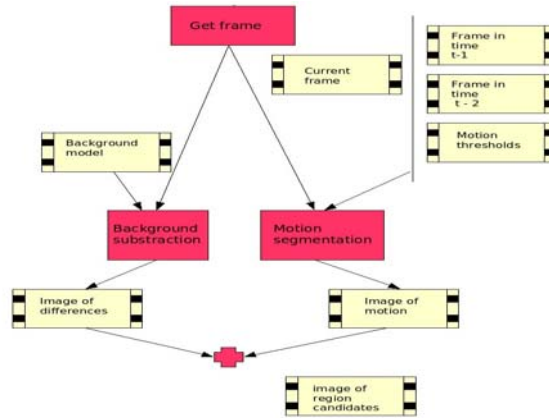


Figure 3. Proposed algorithm for detecting moving objects.

Local object tracking

After motion detection, the next step is to track moving objects from one frame to another in an image sequence. Tracking algorithms usually have considerable intersection with motion detection during processing. Tracking over time typically involves matching objects in consecutive frames using features such as points, lines or blobs. The local tracking goal is to determine which objects in the new frame correspond to those which were present in the previous one and to provide information about objects that could be joined, split, entered or disappeared from two consecutive frames. For each blob, its bounding box (BB) is calculated. The BB is the minimal rectangle that contains the blob. Tracking is performed by checking which BB's in the current frame overlap with those in the previous one. This way, we can detect easily and with a good confidence degree which blob has moved and where, without any further test [3], [4]. In Figure 4, a sample picture taken from a recorded video illustrates the results of the proposed tracking algorithm.

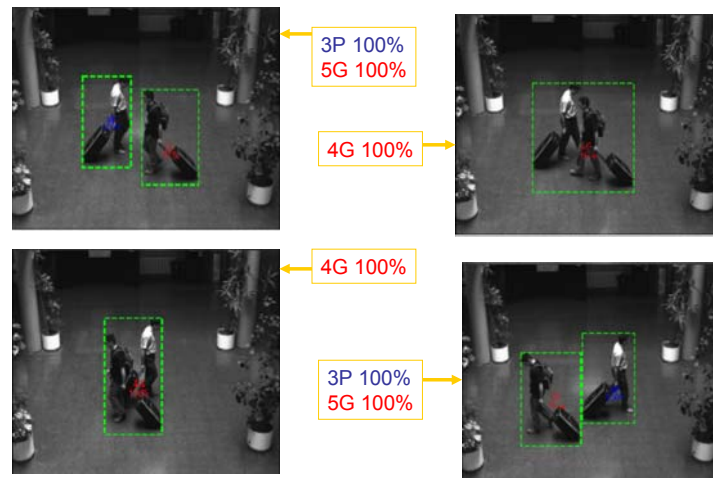


Figure 4. Sample picture from one of the recorded movies and the tracking process when two objects cross and keep the same direction after crossing.

Once blobs are assigned with a label (either new or coming from the previous scene) they are considered tracked objects, (or objects, for brevity).

Object features.

The result of the local tracking step is a number of objects which inherit some properties from those of the previous scene which had a relationship with them, or objects which are new in the scene. Objects in the current scene must be classified first, to give a class label to all objects which came into the scene (new objects) and to confirm the class label assigned to each object which remained inside the scene. In order to achieve this classification, some features must be calculated for each object in each frame.

It must be emphasized that features used to give the rough low-level classification must be quickly computed, and is also crucial to have a high local rate of successful classification; despite of higher level computing layer may correct local classification errors. We used two different feature sets: geometric features, which are mentioned in many papers discussing surveillance systems [8], [9], and another set of features based on the average density of foreground pixels in blob areas [10]. Both sets of features attempt to extract the essence of object silhouettes. Low level provides parameters to describe the video frame activity and characterization of foreground objects. For each detected region in a frame, a set of properties are measured in order to classify the objects at low level, and to deliver a list of attributes to the semantic level.

Classification.

Classification is performed for each object in each frame; new objects are classified in order to give them a first classification. Objects having exact matches with objects in the previous scene are classified again to give more confidence to the assigned class. Classification is performed in three phases (see Figure 5). First of all, objects are classified as static or dynamic. In the second phase, each object in the scene is assigned to a class taking into account only the recently calculated local features. On a subsequent phase, each object has its own history (if any) to do a more accurate classification.

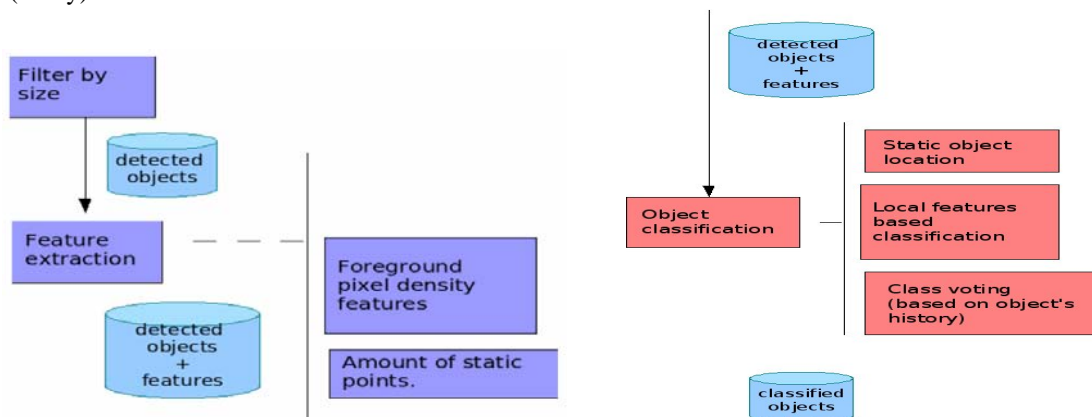


Figure 5. General procedures for Features Extraction (left) and Classification (right).

The classification based on current features (that means, those calculated for the current frame) is performed by invoking a K-NN classifier. This classifier gives for each object a class label (person, group of people and luggage) and a total amount of neighbours which were the nearest to the sample the object represented. Having only results from the classifier may lead to inaccurate labelling, which can be improved if additional information (other features) is available. In order to achieve this, the result given by the K-NN classifier is refined by using the dynamic state of the object. The confidence of each classification takes into account all previous classifications of the object. The sum of all votes of the most voted class is computed and divided by the total amount of votes the object has received. In this way the classification relies on the complete history of the object and not only on punctual values. Other methods for confidence calculation, could be the sum of distances from the most voted neighbours divided by the sum of all distances, which is similar to the one used.

3 TEST BED SOFTWARE IMPLEMENTATION

In the architecture, whose outline is depicted in Figure 6, the vision board must make the tasks corresponding to the compression of live images and the extraction of characteristics of the visual detected objects, mainly:

- Video compression (e.g. MJPEG)
- Real-Time processing the images to identify objects, XML encode features and transmit it to the main board (RDU).

The video processing algorithms described in previous sections of this paper have been implemented on a BF561 processor. This implies that the source code generation cannot be done in a single software project, but in some different projects.

Ethernet is a masterpiece on the communications center. The Ethernet Media Access Control (MAC) device of the ADSP-BF537 processor provides a 10/100-Mbits/s Ethernet. We have developed an embedded program, taking advantage of the Ethernet MAC and using the TCP/IP stack support provided by VisualDSP++. We use the Ethernet driver provided by analog, the lwIP TCP/IP stack libraries and the BSD Socket API to construct our

net services. Hardware optimizations related to memory (code and data distribution) and DMA transfers must be done between memory modules. The following optimizations have been carried out in both processors, BF537 and BF561:

- Cache optimization. With the aim of obtain best performance, cache banks must be used.
- Spatial blocks formed by code, for optimal memory RAM Banks access, must be distributed.
- Searching of core code processes (most used code functions) and local optimizations via pragmas or other methods (float by integer substitution, etc...).
- Maximize and control DMA flow between memory and peripherals, in order to minimize processing activity.

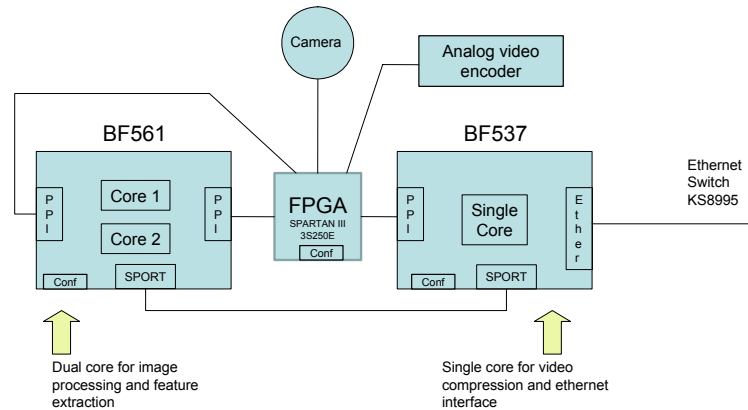


Figure 6. Video Board Architecture Outline

3.1 SYSTEM CODE

System Code is the code that makes the hardware run. System code includes device drivers, interrupt code, configuration code and kernel code. Although the application has been completely written in C language, there are some assembler code functions. The main parts of the system code are the following:

- Device Drivers
 - Omnivision OV7660 Camera driver
 - PPI0 driver
 - SPORT
- Interrupt handlers

3.2 SOFTWARE FOR FEATURES EXTRACTION UNIT (BF561)

The implemented code for the BF561 processor has been obtained by translating into C code the vision algorithms previously written in Matlab language during the debugging of these algorithms. This visual processing code consists of several phases; each of these phases gets some info from the previous phase, treats it, and passes the data to the next phase. Thus, this algorithm can be implemented as a software pipeline.

Each of the phases has been created as a thread. We have four different threads for the algorithm. The threads have been distributed between the two cores of the processor according to its computational weight. Figure 7 depicts the main algorithm threads and their distribution.

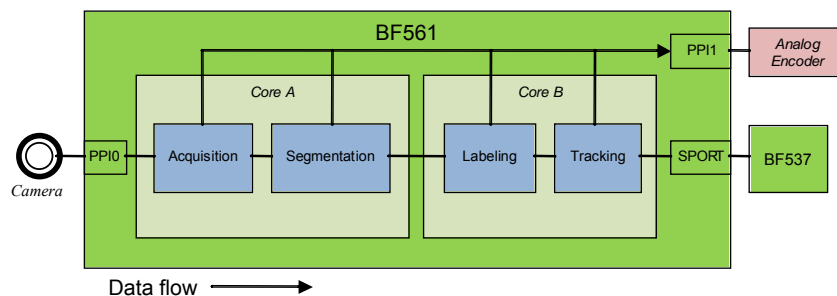


Figure 7: Implemented vision algorithms structure overview

Between threads there are shared circular buffers. So each thread reads data from the previous buffer, treats

it, and stores the results into the next buffer. Figure 8 illustrates the evolution of a frame through the different phases of the algorithm. First, the YUV frame is converted into a gray frame. Then, this gray frame is compared with the background model to obtain a binary image. With this information, we proceed to label the image, in order to find the properties of the connected components. The last image of the sequence shows the output of the labeling algorithm. In this frame, some noise has appeared. This noise will be detected and discarded in the tracking phase.

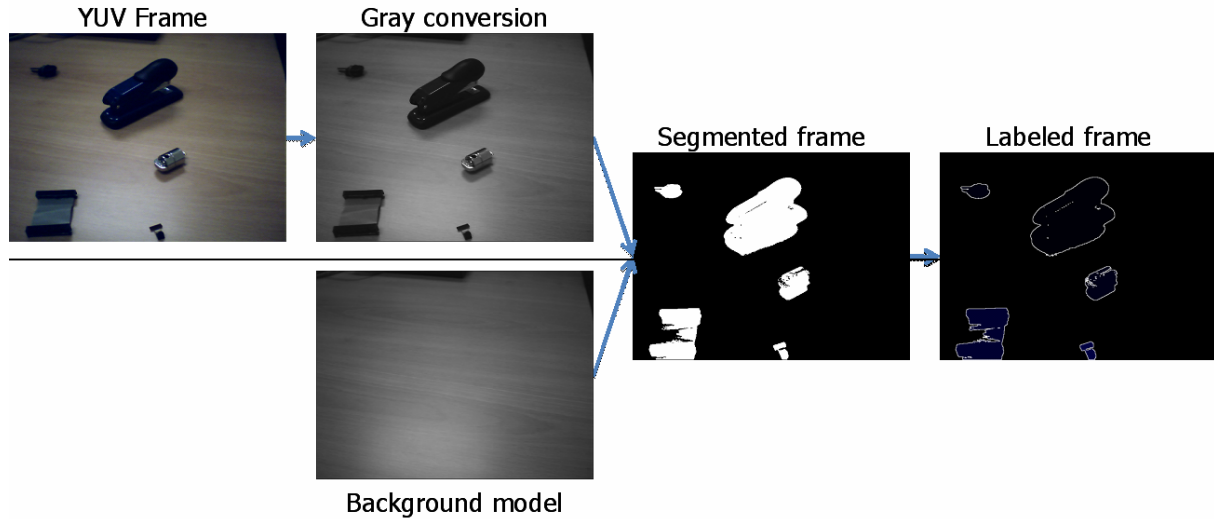


Figure 8. Phases of a frame during its processing.

Buffer Control

As previously indicated, the threads share the algorithm data using circular buffers. These buffers need to be controlled in some way. This arbitration can be done with the semaphores provided by VDK.

The state of each buffer is indicated by two semaphores. The first one is used to know how many buffer positions are empty. When a thread wants to write some information to this buffer, “pends” on the semaphore. If the buffer is full, the thread gets blocked. The second semaphore indicates how many positions of the buffer are full. The reading thread “pends” on this semaphore. Thus, if the buffer is empty, it gets blocked until new data arrives. So, in general terms, for a particular buffer, the thread of the phase n “pends” on the “is full” semaphore and posts the “is empty” one and the thread of the phase $n+1$ “pends” on the “is empty” semaphore and posts the “is full” one. This buffer control method allows involved threads to simultaneously write and read from the buffer. This increases considerably the system performance. It also ensures exclusive access to the data present into the buffer.

3.3 SOFTWARE FOR COMPRESSION UNIT (BF537)

The BF537 module functionalities have been divided into port services, such as a client-server model. These are the services where the external applications could be connected to. The implemented list of services is summarized in the following Table I.

PORT	SERVICE	COMMANDS
JPEG Image Stream Service Port 30000	Provides video streaming in MJPEG format on demand.	Input : Request for one frame Output: JPEG frame
Configuration Service Port 30003	Configures different low level video processing parameters.	Input: XML file
XML data frame Service Port 30001	Provides the XML information if a JPEG is requested from Image Stream Service (no requests via Ethernet port).	Output: XML with image features.

Table I: BF537 module services

4 PRELIMINARY RESULTS AND CONCLUSIONS

The implemented code has been tested using real video sequences from onboard camera, using 8 frames per

second (125ms per frame). This includes execution of all video processing algorithms in both cores of BF561 as well as the JPEG compression and XML generation in BF537 processor, using the LLS received from the other processor via the SPORT. This XML stream is sent through the Ethernet interface and received in a remote computer with a client image program developed in Java.

Also, to test the performance of these algorithms, some stressing tests have been made. We understand stress when the camera detects a lot of objects in the scene (for instance, shaking the camera or moving it). This causes the labeling algorithm to run slowly as well as the tracking. As a quality indicator, the total execution time per frame has been obtained and plotted in the following Figures 9 and 10, corresponding to a video sequence over 2.000 frames (around 33 minutes of video sequence). As it can be seen in these figures, the time per frame remains almost constant with reduced fluctuations due to the excessive stress induced in the stress test, and is almost independent of the number of detected blobs.

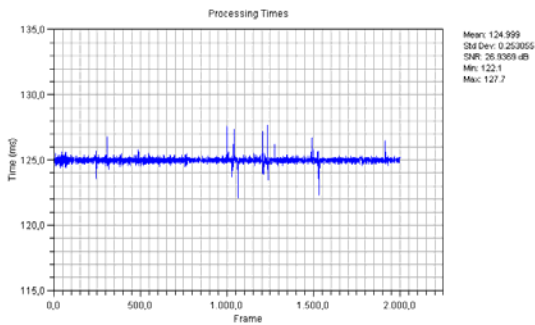


Figure 9: Time between frames over a 2000 frames execution.

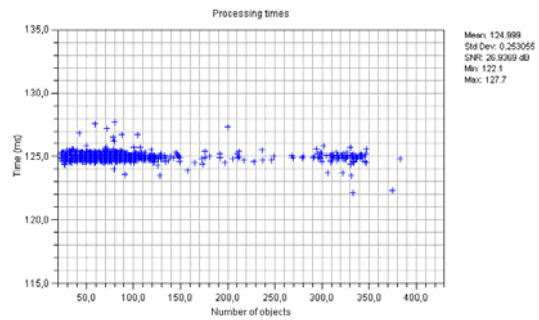


Figure 10: Time between frames as a function of the number of objects per image.

5 REFERENCES

- [1] B. Shoushtarian, H.E. Bez, "A practical adaptive approach for dynamic background subtraction using an invariant colour model and object tracking", *Pattern Recognition Letters* 26 (2005) 5–26
- [2] Hu Weiming, Tan Tieniu, Liang Wang, and Steve Maybank, "A Survey on Visual Surveillance of Object Motion and Behaviors", *IEEE Transactions on Systems, man, and Cybernetics, Vol. 34, NO. 3, August 2004.*
- [3] R. T. Collins et al., "A System for Video Surveillance and Monitoring" Robotics Institute, CMU-RI-TR-00-12 VSAM. Final Report.
- [4] L. Fuentes and S. Velastin, "From tracking to advanced surveillance," in *Proceedings of IEEE International Conference on Image Processing, (Barcelona, Spain), Sept 2003.*
- [5] JPEG library documentation. http://www.analog.com/processors/platforms/JPEG_Motion-JPEG.html
- [6] Aho, Ullman, "Understanding the compiler application and about grammars", in *Compilers, Principles, Techniques and Tools, Syntax analysis. Pearson Education.*
- [7] XML. <http://www.w3.org/TR/xml/> XML 1.0 technical specification.
- [8] Haritaoglu I., Harwood D., and Davis L.S. "W4: Real time surveillance of people and their activities". *IEEE Transactions on PAMI*, pages 809 – 830, 2000.
- [9] Wren C. R., Azarbayejani A., Darrell T., and Pentland A. P., "Pfinder: Real-Time Tracking of the Human Body". *IEEE Transactions on PAMI IEEE*, Vol. 19, No. 7, July 1997, pp 780-785.
- [10] Rosell J., Andreu G., Rodas A., Atienza V., and Valiente J., "Feature sets for people and luggage recognition in airport surveillance under real-time constraints". *VISIGRAPP08*, pages 662 – 665, 2008.