**2012 – 9th International Multi-Conference on Systems, Signals and Devices**

# A HW/SW implementation on FPGA of a robot localization algorithm

Agnès Ghorbel, Nader Ben Amor
and Mohamed Jallouli
Computer & Embedded System Laboratory,
Engineering School of Sfax (ENIS),
University of Sfax, BP 1169, 3029 Sfax, Tunisia.

Lobna Amouri
Control & Energy Management Laboratory,
Engineering School of Sfax (ENIS),
University of Sfax, BP 1169, 3029 Sfax, Tunisia.

***Abstract*—This paper presents an implementation of robot localization algorithm using FPGA technology. The adopted localization method uses webcam tracking images. This technique has been developed and implemented for the motion of the robot from an initial position towards another desired position. Firstly, we have validated our approach on PC platform using C language and OpenCV library. Secondly, for the autonomous navigation, a mixed HW/SW implementation was been developed using a high performance version of the NIOS processor coupled with a hardware accelerator. Experimental tests on Altera Cyclone III FPGA Starter Kit proved the effectiveness of the proposed architecture.**

***Index Terms*—HW/SW implementation, autonomous navigation, Altera FPGA, robot localization, webcam data, image processing.**

## I. Introduction

Robot Navigation is a very quickly developing field in the science of robotics. Mobile robots are especially being used as a substitute for humans or to do simple work that is either in or outside. It is becoming more and more important to be able to determine with exactitude the position of a robot in its environment, as well as to manage all the related mechanical, electronic and software issues.

The robot localization problem is a key problem in making truly autonomous robots. If a robot does not know where it is, it can be difficult to determine what to do next. In order to localize itself, a robot has access to relative and absolute measurements giving the robot feedback about its driving actions and the situation of the environment around the robot.

## II. State of the Art

Relative localization consists of evaluating the position and orientation using data of encoder and inertial sensor data. The integration is started from the initial position and orientation and is continuously updated. Though the technique is simple, it is prone to error due to imprecision in modeling, noise, drift and slip [1]. Since the position estimation are based on earlier positions, the error in the estimates increases over time.

Absolute localization provides position measurements based on observations made from the environment. This position information is independent of previous position estimates [2]. The location is not derived from integrating a sequence of successive measurements, but directly from one measurement. This has the advantage that the error in the position does not grow unbounded, as is the case with relative position techniques [2]. The major disadvantage of absolute measurements is their dependence on the characteristics of the environment.

In order to compensate the drawbacks of the two techniques, substantial improvement is provided by applying Kalman Filtering techniques [3]. These filters can estimate states of noisy systems in noisy environment. Another approach adapts the position and the orientation of a mobile robot through a weighted Extended Kalman Filter (EKF) [4] and [5]. These methods need much calculation for a mobile robot to perform a task. Other disadvantages are either the short range of used sensors or the necessity to know the initial position of the robot [6].
Other solution, in [7], uses a method which permits the vehicle to correct its drift by direct observation using a unique embedded CCD camera on a mobile robot. In [8], another localization technique is presented. It uses the correspondence between a current local map and the global map previously store in memory. In [9], a method calculates the position of the robot in order to intercept a moving target through visual feedback. The most important disadvantage of these methods is the necessity to know the initial position of the robot. In [10], a multi-DSP platform, based on TIs DSPs from the C2000/C5000 families is used for motion control and data processing on the mobile robot F.A.A.K. This technology is particularly efficient for implementation of complex localization technique. Their internal parallel structure is suitable for image processing algorithm. Azhar and Dimond in [11] use FPGAs for the implementation of control and sensor fusion algorithms in the inertial navigation system of a Mobile Inverted Pendulum (MIP) robot. The FPGA technology, despite its little use in robotics, has significant advantages especially important capacity that allows to use a single PFGA to control a robot. It can replace several DSPs and integrate specific HW components.

The whole proposed technique consists on combine sensors measurements with external absolute data to reduce the encoder position errors and provide the best estimate of the robot's position. In this paper, we are interested to implement the absolute localization algorithm (based on webcam data) on FPGA embedded system.

The paper is organized as follows. In section 3, we describe the proposed approach for absolute localization. Then, we present the different algorithms used to determine absolute position of the robot in the test platform by exposing progressively validation's results. In section 4, we detail the HW/SW implementation on the Altera FPGA embedded system.

## III. Description of the absolute localization algorithm

The proposed absolute localization algorithm is based on webcam data. The provided images are treated with various image processing techniques. The aim of treatment is to obtain the position of the mobile robot on the platform.

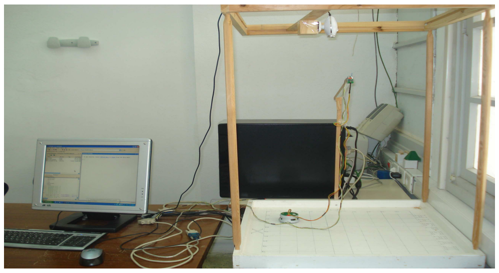


Fig. 1. The robot environment

As we shown in Fig. 1, the external absolute data are obtained from a camera mounted on the ceiling of the test environment and ensured the absolute localization. The webcam provides a color image which will subsequently be handled by an image processing program to determine firstly, the reference system and secondly the robot position. The robot used in this work is the Mini Khpera II. It is equipped with a 68331 Motorola processor with 25 MHz of frequency, 8 Infra-red proximity and ambient light sensors and a serial port providing communication with the PC.

This approach is based on three steps as shown in figure 2.

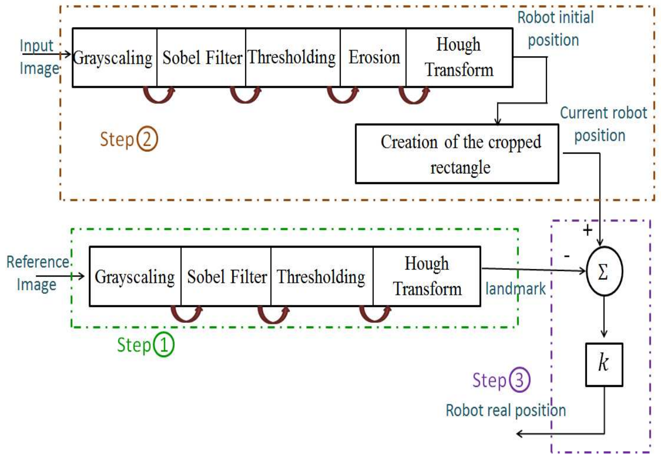


Fig. 2. The adopted approach

The step 1 consists on the localization of the four landmarks and definition of the robot cartesian coordinate system set up against which the position of the robot will be calculated. In the step 2, we calculate the current position of our robot in a sequence of images. In step 3 and last step, we estimate the position of the robot on the platform.

This technique has been implemented and validated as a first step, under Visual Studio with C language using OpenCV Library. OpenCV is an open source and free computer vision library. The only OpenCV features used in application are loading and displaying images in graphical windows (The images are captured from a camera and saved in the hard disk of PC). No predefined image processing function was used. This is in order to be able to embed quickly the application and facilitate the hardware accelerator's design.
The three steps shown previously will be detailed in this section.

### A. The robot Cartesian coordinate system

The aim of this part is to identify the four landmarks used for the cartesian coordinates system set up against which the position of the robot will be calculated. We extract the pixel coordinates of the reference system from a reference image, which is captured and stored in advance. This image, as shown in figure 3, describes the workspace of the mobile robot limited by four landmarks placed on the corners.

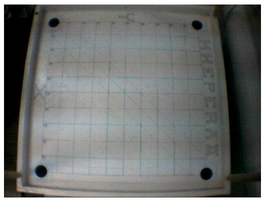


Fig. 3. Reference image

After extracting coordinates of the landmarks, we can define our reference system shown in figure 4:

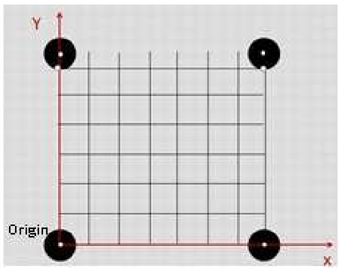


Fig. 4. Cartesian coordinates system

## *B. Determination of robot position*

We determinate the robot position in a sequence of images: we extract from each sequence's frame the position of robot Mini Khepera II.
In order to decrease the processing time, we apply a cropping rectangle, embracing the robot, to the binary input image in order to reduce the number of the treated pixels. The proposed algorithm [6] is presented as follow. In the first step, we have the first binary image with the default webcam image size 640*480 pixels. On this image we calculate the robot position in pixels coordinates. In the second step, we define a crop rectangle (Fig. 5) that depend on previous robot coordinates.

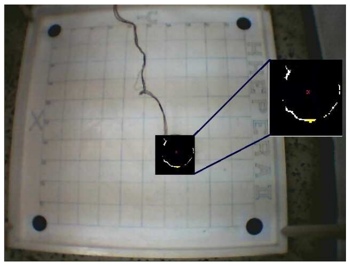


Fig. 5. Cropped rectangle: 70*70 pixels

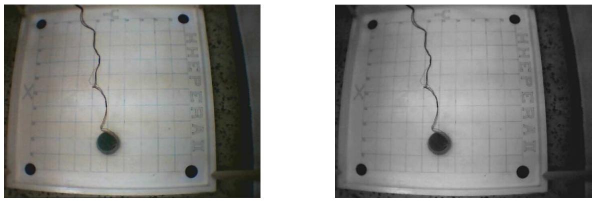
## *C. Robot and landmark localization technique*

The different algorithms used are the grayscaling, the Sobel filter, the thresholding, erosion and Hough Circle Transform.

*1) Grayscaling:* A grayscale (or graylevel) image is simply one in which the only colors are shades of gray. In fact a gray color is one in which the red, green and blue components all have equal intensity in RGB space [12].
A simple way to convert the image color to a graylevel is to calculate the pixels luminance using (1):

$$\mathrm{Gray} = 0.299 * \mathrm{Red} + 0.587 * \mathrm{Green} + 0.114 * \mathrm{Blue} \qquad (1)$$

The result after applying the grayscale on the reference image and on one of robot position is shown in (Fig.6).



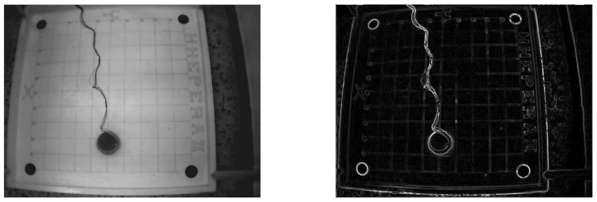
(a) Color image (b) Grayscale image

Fig. 6. Grayscale robot position image

*2) Edge Detection (Sobel Filter):* We use the Sobel edge detector to find the approximate absolute gradient magnitude at each point of an input grayscale image in order to detect the four landmarks and the robot.
We use a pair of 3 x 3 convolution masks, one estimating gradient in the x-direction and the other estimating gradient in y-direction [13]. These two masks are convolved with the incoming image data to measure the differences in intensity along the horizontal, vertical directions. These two measurements $E_h$ and $E_v$ are then combined to estimate edge magnitude and direction.
The gradient magnitude is estimated as (2):

$$|Mag| = \sqrt{(E_h)^2 + (E_v)^2} \qquad (2)$$

Figure 7 illustrates the image resulted from Sobel filter.



(a) Input image (b) Output edges

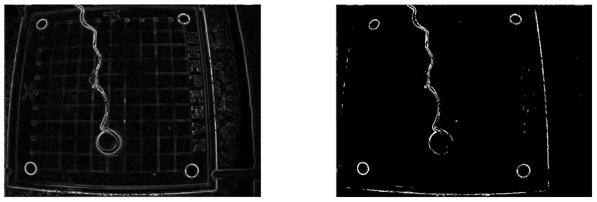
Fig. 7. Sobel Filter on robot position image

With this filter we can transform the input image (Fig.7(a)) into a black image unless at the points where a contour is detected that is marked in white (Fig.7(b)).

*3) Image Thresholding:* Image thresholding which extracts the object from the background in an input image is one of the most common applications in image analysis. Among the image thresholding methods, bi-level thresholding separates the pixels of an image into two regions (i.e. the object and the background); one region contains pixels with gray values smaller than the threshold value and the other contains pixels with gray values larger than the threshold value [14]. If $\mathrm{f}(i,j)$ is the gray level of point $(i,j)$ and T the threshold value, the thresholded image $\mathrm{g}(i,j)$ is then defined as:

$$\mathrm{g}(i,j) = \begin{cases} white & if\ f(i,j) \geq \mathrm{T} \\ black & otherwise \end{cases} \qquad (3)$$

We can show thresholding result image in figure 8.



(a) Input image (b) Output thresholded

Fig. 8. Thresholded robot position image

As we can see, we can reduce in Fig. 8(a) a large quantity of informations (the white lines) while conserving in Fig. 8(a) nearly all pertinent informations (essential to its comprehension) to separate objects from background.

*4) Morphological operator (Erosion):* The erosion was been applied only to the robot position image to isolate robot from four landmarks.
The erosion operator takes two pieces of data as inputs. The first is the image which is to be eroded. The second is a (usually small) set of coordinate points known as a structuring element (also known as a kernel). It is this structuring element that determines the precise effect of the erosion on the input image [12].

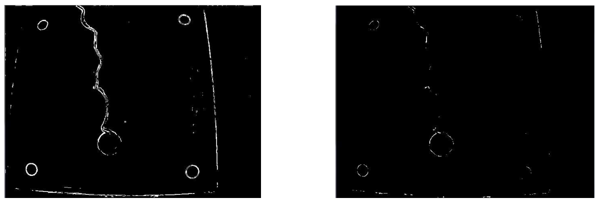

(a) Input binary image (b) Output eroded

Fig. 9. The erode robot position image

Figure 9 shows the result after applying erosion on the binary image. The structuring element (SE) chosen in this part, $SE = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ is a disk with radius 5 in order to not change the circular form of the robot.

*5) Hough Circle Transform:* To locate the four landmarks in the reference image and our robot, an algorithm for detecting circles was required. This algorithm will allow us to extract, first, center coordinates of the landmarks to define the reference system, second and on each acquired image, the coordinates of the robot in the environment in which it is operating. The adopted algorithm is the Hough Circle Transform.

The circle is actually simpler to represent in parameter space since his parameters can be directly transfer to the parameter space. The equation of a circle is:

$$\mathrm{r}^2 = (\mathrm{x}-\mathrm{a})^2 + (\mathrm{y}-\mathrm{b})^2 \tag{4}$$

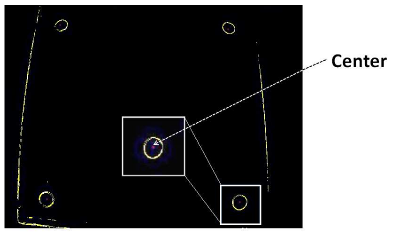
As it can be seen, the circle have three parameters, a, b and r. Where a and b are the center of the circle respectively in the x and y direction and where r is the radius. The parametric representation of the circle is represented as follows:

$$\begin{cases} \mathrm{x} = \mathrm{a} + \mathrm{r}\cos\theta \\ \mathrm{y} = \mathrm{b} + \mathrm{r}\sin\theta \end{cases} \tag{5}$$

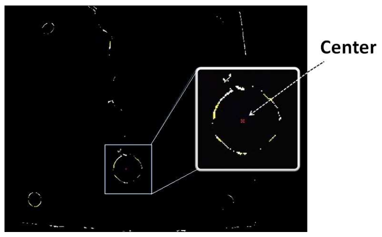
The algorithm of Circular Hough Transformation can be summarized [15] to:

```
1) Find edges
2) // HOUGH BEGIN
3) For each edge point:
   Draw a circle with center in the edge point with
   radius r and increment all coordinates that the
   perimeter of the circle passes through in the ac-
   cumulator matrix which essentially has the same
   size as the parameter space.
4) Find one or several maxima in the accumulator:
   The highest numbers (selected in an intelligent way,
   in relation to the radius) correspond to the center
   of the circles in the image.
5) // HOUGH END
6) Map the found parameters (r, a, b) corresponding
   to the maxima back to the original image.
```

Applying this transform in our images, we have managed to locate in the one hand, four landmarks (defining the reference system of the robot) (see Fig. 10(a)) and in the other hand, the position of the robot (Fig. 10(b)).



(a) Center of landmarks



(b) Center of robot

Fig. 10. Centers with HCT

## *D. Transformation from image coordinates to real coordinates*

We define the left bottom landmarks as the origin of our system as shown in figure 11. We obtain the real image coordinates by a simple difference between the pixel coordinates either of the robot or of the reference system. The real image coordinates (in pixels) is given by:

$$\begin{cases} x_{robot} = j_r - j_1 \\ y_{robot} = i_r - i_1 \end{cases} \tag{6}$$

Where $i_r$ and $j_r$ are the position (i,j) of the robot on the image and $i_1$ and $j_1$ are the position (i,j) of the origin.
The resulting coordinates are multiplied by a constant coefficient $k$. This coefficient is calculated on the basis of both the

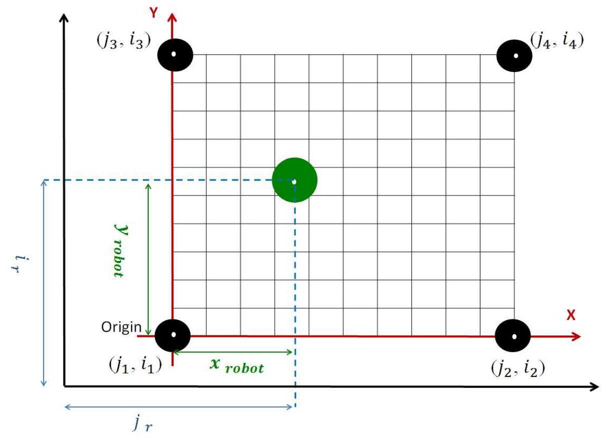


Fig. 11. The positioning architecture adopted

real and the pixels distance between the landmarks along the x-axis and the y-axis. The coefficient $k$ is given by:

$$\begin{cases} K_x = \dfrac{D_x}{j_2 - j_1} \\ K_y = \dfrac{D_y}{i_4 - i_1} \end{cases} \tag{7}$$

where $D_x$ and $D_y$ are respectively the real distance in centimeters between the landmarks along the x-axis and the y-axis.

By applying these method described above on the hardware environment described by a Intel(R) Core 2 duo PC with a frequency 2GHz, the average processing time is about 0.009 seconds which is more than enough to provide continuous robot navigation. In order to ensure autonomous navigation, we decide to implement this technique of localization on an ALTERA FPGA embedded system.

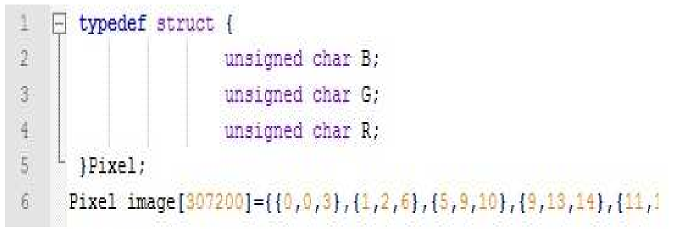
## IV. Implementation on Altera FPGA Platform

Once validated on a PC, the adopted approach to detect the robot center will be implemented on an Altera Cyclone III FPGA Starter Kit for autonomous navigation. In this section, we present the steps needed in order to implement this technique on the NIOS-II processor. Then we detail the various stages to measure time processing. Finally, we expose different techniques used to optimize and reduce the overall execution time.

### A. Purely Software Implementation

The first step we have to do is the adaptation of the code which has been executed under Visual Studio using OpenCV library. Yet, it's impossible to migrate to an embedded program for NIOS-II CPU with such library because the reading function used for loading and displaying image is not supported by the NIOS-II.

So, to overcome this problem, we have declared images in header file (.h) directly in the code (see Fig.12). This file contains an array of pixel whose the struct pixel defines the RGB components.



```
typedef struct {
            unsigned char B;
            unsigned char G;
            unsigned char R;
 }Pixel;
Pixel image[307200]={{0,0,3},{1,2,6},{5,9,10},{9,13,14},{11,
```

Fig. 12. Example of header file

After importing, configuring and executing the program on the FPGA, we need to verify the result. Since our FPGA is not equipped with a VGA port, we have displayed in the NIOS console the $RGB$ components values, we put each in a matrix and we concatenated them in order to obtain a 3 channels matrix, using MATLAB command:

$$\text{MyRGB} = \text{cat}(3, \text{Red}, \text{Green}, \text{Blue}) \tag{8}$$

The images obtained are identical to those shown in figure 10 on the PC version. This validates the purely software C version developed for the NIOS.

### B. Time processing measure

The implementation of the absolute localization algorithm on the FPGA leads to an assembly of time execution which are resumed in table I and table II.
We can notice that we obtained 166.912 seconds as time processing for the first image (640*480 pixels). However, the rest of images (70*70 pixels) needed only 12.169 seconds.

TABLE I
Measuring Result for the first image

| Section | % of total time | Time(sec) | Time(clocks) |
|---|---|---|---|
| Gris and Sobel | 81.6 | 136.13765 | 13613765110 |
| Binarisation | 0.802 | 1.33832 | 131831534 |
| Erosion | 4.77 | 7.95948 | 795947555 |
| Hough Transform | 12.9 | 21.47605 | 2147604584 |
| **Total Time :** | 166.912 seconds | | |

TABLE II
Measuring Result for the remaining images

| Section | % of total time | Time(sec) | Time(clocks) |
|---|---|---|---|
| Gris and Sobel | 18.61 | 2.26168 | 226168335 |
| Binarisation | 0.290 | 0.03528 | 3528226 |
| Erosion | 1.4 | 0.17022 | 17021901 |
| Hough Transform | 79.7 | 9.70168 | 970168486 |
| **Total Time :** | 12.1697 seconds | | |

To ensure fluid robot navigation, the processing time must be less than 0.1 seconds. Then, we decide to accelerate our system by various methods.

## C. Tools for accelerating time processing

In order to accelerate the system and reduce the execution time, we use different methods:

*1) Tools available on the Nios II processor:*

- The use of the fast version of the NIOS (NIOS-II/f core) that provides a performance over 300 MIPS and Six-stage pipeline to achieve maximum MIPS per MHz.
- The use of the floating-point custom instructions, that implements single-precision, floating-point arithmetic operations (addition, subtraction, multiplication and division). This IP has been used to compute the grayscale shown in (1).

The add of these two accelerated tools leads to an assembly of time execution which are resumed in table III and table IV.
We can notice that we obtained 30.485 seconds as time processing for the first image and the rest of images (70*70 pixels) needed only 4.10418 seconds.

TABLE III
MEASURING RESULT FOR THE FIRST IMAGE

| Section | % of total time | Time(sec) | Time(clocks) |
|---|---|---|---|
| Sobel | 67.61 | 20.60082 | 2060081908 |
| Binarisation | 0.786 | 0.23956 | 23956105 |
| Erosion | 6.34 | 1.93425 | 193425323 |
| Hough Transform | 25.3 | 7.71108 | 771107747 |
| **Total Time :** | 30.486 seconds | | |

TABLE IV
MEASURING RESULT FOR THE REMAINING IMAGES

| Section | % of total time | Time(sec) | Time(clocks) |
|---|---|---|---|
| Sobel | 6.66 | 0.27339 | 27339231 |
| Binarisation | 0.147 | 0.00602 | 602300 |
| Erosion | 0.905 | 0.03716 | 3715793 |
| Hough Transform | 92.3 | 3.78730 | 378730007 |
| **Total Time :** | 4.10418 seconds | | |

*2) Creation of hardware accelerator:* We notice in Table III that the Sobel function monopolizes 67.6% of total time since it contains complex operations (equation (2)).

So, we design a Hardware Accelerator that computes equation (2) to HW in order to accelerate execution time for the first image We realize a system composed of a NIOS II Processor that communicates with HW accelerator through the Avalon bus (see Fig. 13):

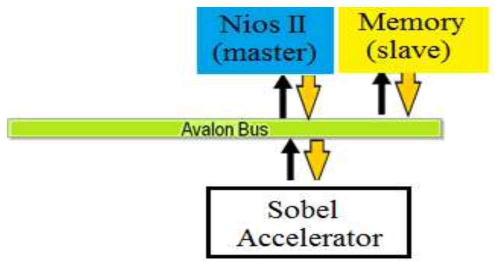


Fig. 13. The Sobel Hardware Accelerator

The schematic block of the Sobel HW accelerator is shown in figure 14.

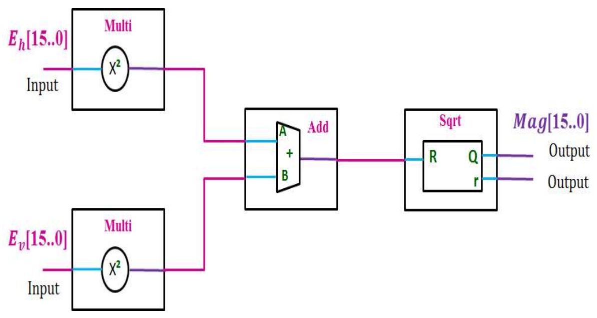


Fig. 14. A schematic block of sobel accelerator

The visualizing images after adding the Sobel Accelerator are identical to those shown in figure 10.

These results validate the design of the figure 13.

With all these accelerating tools (NIOS-II/f, Floating Point coprocessor and the Sobel accelerator), we reached a 16.617 seconds of execution time for the first image (Tab. V) which corresponds to 90% of time decrease against previous time.

TABLE V
MEASURING RESULT FOR THE FIRST IMAGE AFTER ACCELERATION

| Section | % | Time(sec) | Time(clocks) |
|---|---|---|---|
| Sobel | 38.6 | 6.40658 | 640657845 |
| Binarisation | 1.440 | 0.23955 | 23955376 |
| Erosion | 10.04 | 1.72875 | 172875110 |
| Hough Transform | 49.6 | 8.24190 | 824190103 |
| **Total Time :** | 16.6171 seconds | | |

For the rest of the images, we reached 3.889 seconds that corresponds to 75% of time decrease (Tab. VI):

TABLE VI
MEASURING RESULT FOR THE REMAINING IMAGES AFTER ACCELERATION

| Section | % | Time(sec) | Time(clocks) |
|---|---|---|---|
| Sobel | 3.35 | 0.13031 | 13030674 |
| Binarisation | 0.158 | 0.00615 | 615323 |
| Erosion | 0.958 | 0.03728 | 3727687 |
| Hough Transform | 95.5 | 3.71578 | 371577659 |
| **Total Time :** | 3.88982 seconds | | |

## V. CONCLUSION

In this work, we have realized an application that ensures the absolute localization of the robot Khepera II in its workspace using webcam data. We developed a C version of the localization algorithm using OpenCV library. Then we implemented it on an ALTERA FPGA embedded system using the NIOS processor. We used specific computation hardware (floating point coprocessor) and a custom Hardware accelerator (Sobel) to reduce the overall execution time.
The obtained results proved the effectiveness of the proposed HW/SW architecture in accelerating the processing time. However, it still enough to ensure continuous and real time robot navigation. The further improved the obtained values, an implementation of a parallel architecture with a multi-processor FPGA system will be held in future work. Due to the limited capacity of the used FPGA (Cyclone III FPGA Starter kit), other optimizations techniques cannot be used especially multiprocessor techniques. These techniques are particularly efficient especially for application with high inherent parallelism like our navigation application. Direct navigation of the robot using the Cyclone III FPGA Starter kit is not possible due to the lack of DVI input nor VGA output.

We plan in our ongoing work to realize all the application using the ML 507 platform with PowerPC processor which is more performant than NIOS and integrating the camera for images acquisition.

Furthermore, the use of the camera is restricted to an internal environment (indoor) to be able to fix it, while in an external environment, it will be harder to do it. So, to overcome this problem, a multi-sensors (GPS, cameras or DGPS) fusion approach is a way to improve environment perception in a real-life scenario would involve a much larger area.

## REFERENCES

[1] S.I. Roumeliotis, G.S. Sukhatme, and G.A. Bekey, “Smoother based 3D Attitude Estimation for Mobile Robot Localization“. In *Proc. IEEE International Conference in Robotics and Automation*, May 1999.
[2] R. Negenborn, “Robot Localization and Kalman Filters On finding your position in a noisy world“. September 1, 2003.
[3] M.S. Grewal and A.P. Andrews, “Kalman Filtering, Theory and Practice“. Prentice Hall, 1993.
[4] J.Z. Sasiadek, P. Hartana, “Sensor Data Fusion Using Kalman Filter“. *In: Proc. Conf ISIF*, 2000, pp. 19-25.
[5] F. Kobayashi, F. Arai, T. Fukuda, K. Shimojima, M. Onoda and N. Marui, “Sensor Fusion System using recurrent fuzzy inference“. Journal of Intelligent and Robotic Systems , vol. 23, 1998, 201-216.
[6] M. Jallouli, L. Amouri, N. Derbel “an effective localization method for robot navigation through combined encoders positioning and retiming visual control“ Journal of Automation, Mobile Robotics and Intelligent Systems, volume 3, n 2 2009.
[7] K. ACHOUR and A.O. DJEKOUNE “Localization and guidance with an embarked camera on a mobile robot“ Advanced Robotics, Vol. 16, No. 1, pp. 87- 102 VSP and Robotics Society of Japan 2002.
[8] A. Kak, K. Andress, C. Lopez-Abadia, M. Carroll, J. Lewis, “Hierarchical Evidence Accumulation in the Pseiki System and Experiments in Model-Driven Mobile Robot Navigation“ in *Annual Conference on Uncertainty in Artificial Intelligence, North-Holland* Vol 5, Elsevier Science Publishers B.V., pp. 353-369
[9] L. Freda, G. Oriolo, “Vision-based interception of a moving target with a non-holonomic mobile robot“. Robotics and Autonomous Systems Journal, vol. 55, February 2007, pp. 419-432.
[10] I. Masr and M. Gerke “DSP-Based Control of Mobile Robots“DSP Education and Research Symposium, Birmingham, U.K, November 2004.
[11] M. A. H. B. Azhar, K. R. Dimond “FPGA-based Design of an Evolutionary Controller for Collision-free Robot“ *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*.
[12] R. Fisher, S. Perkins, A. Walker and E. Wolfart. “Hypermedia Image Processing Reference“ 2003.
[13] O. R. Vincent, O. Folorunso ”A Descriptive Algorithm for Sobel Image Edge Detection ” *Proceedings of Informing Science and It Education Conference* (InSITE) 2009.
[14] L. Huang and Mao-Jiun J. Wangt, ”Image thresholding by minimizing the measures of fuzziness ” Journal of Pattern recognition vol. 28, no1, pp. 41-51 1995
[15] S. Just Kjeldgaard Pedersen ”Circular Hough Transform” Aalborg University, Vision, Graphics, and Interactive Systems, November 2007