# Workload Partitioning of a Bio-inspired Simultaneous Localization and Mapping Algorithm on an Embedded Architecture

Amraoui Mounir[1], Latif Rachid[2]
LISTI, ENSA
Ibn Zohr University
Agadir,80000, Morocco

Abdelhafid El Ouardi[3]
SATIE, Digiteo Labs
Paris-Saclay University
Orsay, France

Abdelouahed Tajer[4]
LISA, ENSA
Cadi Ayyad University
Marrakech, 40140, Morocco

***Abstract*—Many algorithms were developed to perform visual localization and mapping (SLAM) for robotic applications. These algorithms used monocular or stereovision systems to solve constraints related to the navigation in unknown or dynamic environment. The requirement of SLAM systems in terms of processing time and precision is a factor that limits their use in many embedded applications like UAVs or autonomous vehicles. Meanwhile, trends towards low-cost and low-power processing require massive parallelism on hardware architectures. The emergence of recent heterogeneous embedded architectures should help design embedded systems dedicated to Visual SLAM applications. It was demonstrated in a previous work that bio-inspired algorithms are competitive compared to classical methods based on image processing and environment perception. This paper is a study of a bio-inspired SLAM algorithm with the aim of making it suitable for an implementation on a heterogeneous architecture dedicated for embedded applications. An algorithm-architecture adequation approach is used to achieve a workload partitioning on CPU-GPU architecture and hence speeding up processing tasks.**

***Keywords—Simultaneous localization and mapping (SLAM); Bio-inspired algorithms; CPU-GPU workload partitioning; embedded systems; visual acuity (VA); hardware/software codesign***

## I. Introduction

The robot navigation is not always possible in some special circumstances, due to the unavailability of a map or because the environment keeps changing. Hence, for its localization, the robot needs to know its pose accurately which is not possible without a map, which brings us back to the initial issue. The dilemma of what should come first the map, or the pose is complex to solve, because it needs to calculate the pose and construct the map at the same time. A solution to this navigation problem is a Visual Simultaneous Localization and Mapping system also called V-SLAM [1]. The implemented algorithms use data inputs from different sensors mounted on the mobile robot, like cameras since they provide much more information about the environment.

To run a Visual SLAM algorithm, the minimum hardware requirement is a CPU based architecture and a monocular camera. Usually, as depth cannot be observed with one camera, most of systems use multi-view techniques to allow map reconstruction. Camera data is used to perform the scene photogrammetry and rebuild the trajectory map. This sensor presents several advantages (price, availability of a high amount of information) compared to other sensors it also has constraints such as the need for calibration, and sensitivity to light changes and intensity.

Bio-inspired approaches are based on learning concepts from nature and applying them to design an enhanced real time SLAM system. Hence most of these algorithms are aiming to simulate the biological retina and brain-based methods for features detection and description, which make the model complex and also its parallelization a real challenge.

Eyes represented by cameras are used to provide inputs data for front-end operations, but images are processed in a different way compared to classic methods. It can be categorized into simple eyes with one concave photoreceptor lens like for humans, and compound eyes like for some insects with multiple lenses [2]. This study is limited to simple eyes because it is the most feasible to be simulated in a mobile robot. Binocular and stereo vision systems can be then considered as a reference.

Simple eyes are grouped into two known categories based on their photoreceptor's cellular construction. Therefore, human and rodent were selected because they are both on the top of each group, since they are representing the best visual acuity (VA) [3].

In previous studies [4 - 5], bio-inspired approaches were proved to be very competitive methods, in term of accuracy and execution time compared to classic ones. This paper is sharping these results by studding two different models: the retina model HOOFR-SLAM [6] and hippocampal model for rodent RAT-SLAM [7].

The contribution is:

- an efficient partitioning model of a bio-inspired algorithm on a heterogeneous CPU-GPU architecture to improve the processing time performance.
- the evaluation of this model on a dedicated architecture based embedded application.

The aim is to be able to determine if bio-inspired V-SLAM methods can be used on a real-time application, despite their algorithmic complexity.

The reported resulting values are given based on the mean of 20 run results for at least 1000 timestamps. Datasets used are from the well-known KITTI benchmark [8] and Oxford New college opensource dataset [9]. Calibration and other algorithm parameters were adapted accordingly to each dataset to always have the maximum performance.

## II. Background

One of the advantages of Rat-SLAM navigation system is its ability to run in dynamic environment using cheap camera sensor, due to the nature of Rat retina that has a low Visual acuity (VA) compared to the human one [3]. From another hand the use of odometrical data in this algorithm is mandatory in order to combine visual scene matching and a semi-metric topological map representation. The used image model is a simulation of a biological rat retina perception. HOOFR-SLAM doesn't need any odometrical data and can rely only on inputs provided by a well calibrated camera.

Human vision is trichromatic because it has three types of color cones: long-wavelength "red", middle-wavelength "green", and short-wavelength "blue". Rat's vision is dichromatic because, they have only two wavelengths: a shorter blue wavelength than human but shifted toward and middle "green". Therefore, rats can see into the ultraviolet, this doesn't mean that they are color blind, they just have different color perception compared to human.

It is very important to understand how images are seen by each bio-inspired system. So, based on above, RAT-SLAM have to spend less time to convert images to gray scale than HOOFR-SLAM, with better adaptation to lower light environment, but accuracy is also lower, since human Visual Acuity (VA) is higher, this is why the use of an additional sensor (odometer) is needed.

Since Rats have eyes on both sides of the head, the vision is binocular, the field is large and panoramic. So, a representation in a simple flat screen will distort the image, since computer screens are flat and adapted to human forward-facing eyes which have a smaller visual field and more binocular vision. Due to these facts, running the same dataset on HOOFR-SLAM and RAT-SLAM will be a real challenge due to the difference in image perception and cameras position, and must be converted and adapted to each bio-inspired system before use, also the dataset will never have the same length, making a direct comparison simply not possible.

Now, after standing up on all the differences related to visual performance and image perception, the next sections are describing the Rat-SLAM and HOOFR-SLAM algorithm concept and the details of each model front end, back-end, kernels and evaluate their algorithmic complexity.

Finally, a parallelization and an implementation on a single CPU, multi-cores CPU and CPU-GPU on laptop, then a Jetson TX1 system on ship (SoC) architecture.

## III. The RAT-SLAM

### A. Algorithm Inputs

The Rat-SLAM system as defined by [7] use a self-motion sensor data to create a representation known as experience map, also used to facilitate the exploration. This technic is used in robotic when the human intervention is not or low needed. This navigation system has two goals, short term goals using a local obstacle map, and long-term goals relaying on experience map, with the aim to reach the desired destination. These two steps can be resumed to Matching [10] and mapping processes, if compared to HOOFR-SLAM as per Fig. 1.

The RAT-SLAM algorithm uses two different sensors, as a main data input source; a camera and an odometer. If a robot operating system (ROS) is used, both inputs will be then received as ROS messages. Also, for simulations reasons, both inputs camera and odometrical messages derived from the mobile robot wheels encoders are provided together in a dataset file (*.bag file), therefore in this case a visual odometry functional block will not be used, this data is included in the opensource dataset benchmark from [9].

Cameras are placed on both sides of the robot to simulate rat eyes position, the vison is binocular, so the depth is calculated using information on head direction, calibration, functional regions of the hippocampus and surrounding areas. The camera motion is limited to the translational and rotational speed and it is estimated via an image comparison process, more details on how it is done are provided by [11].

The rotational velocity estimation is done by minimizing horizontal offset of two consecutive scanline profiles that are generated by summing the images in the vertical direction represented by *f* as follows:

$$f(S, I^j, I^k) = \frac{1}{w-|s|}\left(\sum_{n=1}^{w-|s|}\left|I^j_{n+\max(s,0)} - I^K_{n+\min(s,0)}\right|\right) \quad (1)$$

where $S$ is the profile shift, $I^j$ and $I^k$ are the scanline intensity profiles to be compared, and $w$ is the image width. The pixel shift $S_m$ in consecutive images $I^j$ and $I^k$ is the value of s that minimizes *f* for those two profiles.

$$S_m = \underset{s\,\epsilon|\rho-w,w-\rho|}{\operatorname{argmin}}\ f(S, I^j, I^k) \quad (2)$$

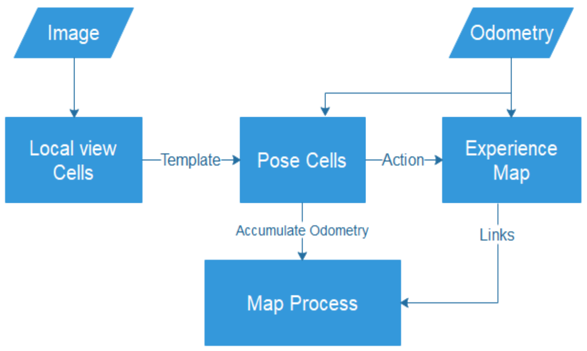


Fig. 1. Rat-SLAM Vision System Diagram with Inputs and Outputs Related to each Functional Block (FB), Local view Cells and Pose Cells are the Font-end and can be Considered as the Tracking Process, using Camera Image Data Plus Odometry as Inputs to Extract Features (Neural Approach) and Provide Actions as Output to the Experience Map. The Mapping Process uses Both Pose Cells Cumulated Odometry Outputs and Links to Avoid Infinity and to Close the Loop.

where ρ is the offset that ensures that there is enough overlap between the profiles. The translational velocity estimation is limited to a maximum value to prevent large changes in illumination and done by multiplying the minimum difference by a scaling factor. Two ROS message are then provided to the system: sensor_msgs::CompressedImage and nav_msgs:Odometry.

## B. Main Functional Blocks

RAT-SLAM is a robust bio-inspired navigation algorithm, used when dependence on human assistance is not or low needed, based on sensor inputs message data described above. The algorithm has three main functional blocks and sub-functional blocks as per Fig. 2. They are identified as follows:

Local view cells: This node uses image data from camera, to make a matching process, by using a comparison technique in order to determine if the visualized scene is familiar or novel. The block algorithm sequence is the following:

**Local view cells algorithm:**

```
Image Conversion, Cropping, sizing and Matching (FB1)
{
 Convert image to gray scale ()

/* For Rat-SLAM the input image is already converted Remove visually bland features like roads */

 Cropping image ()
 Convert_view_to_view_template ()

/* cropping to specified region of the original camera image */

 Do patch normalization ()
 Create_template ()

/* Calculate the sum of absolute difference between current view template that represents the current camera image, and all previously learnt templates*/

 Compare ()
}
```

**Pose Cells**: as per Fig. 1, it uses the odometric data, visual template from local view cell, to provide actions to make decisions used by experience map node. This node has two inputs and one output that are processed as follows:

- Template: is the output of the local view cell image comparison process, if the templates already exist it is directly injected into associated location in the pose cells, if not a new a local ID is associated to the pose cell network. Also, in case of a successive detection of the same template for a long period of time, it could be understood as the robot is not moving.
- Odometric: inputs are processed in Pose Cells; the functional block algorithm sequence is described as follow:

**Pose Cells Algorithm:**

```
Check templates and update template database (FB2)
{
 if (New Template)
 create_view_template ()
 else (Template exist)
 give a local template ID ()
 }
```

```
Local and global excitation, path integration, action and energy management (FB3)
{
 Local excitation ()
/*Add energy around each active pose cell */
 Local inhibition ()
 /*Remove energy from around each inactive pose cell */
 Global inhibition ()
 /*Remove energy from all active pose cells above zero */
 Network energy normalization ()
 Path integration using odometrical information ()
 Get_current_exp_id ()

/* Identify the centroid of the dominant activity packet in the network */
 Get_action ()
 Update_scene ()
 }
```

Experience Map: Manage based on both local view and pose Cells outputs, the graph building, relaxation and path planning. This block has three main outputs:

*1)* A path message with information on the trajectory to the goal.

*2)* A list of nodes and edges that makes a representation of the experience map.

*3)* The robot poses in the experience map.

*4)* Get the goal way point

The experience map algorithm is the following:

**Experience Map algorithm:**

```
Update experience map get new target (FB4)
{
 Update Experience map ()
/* update the current position of the experience map since the last experience*/

 Calculate_path_to_goal ()
 Get_goal_waypoint ()
 }

 Map Converging, add new goal (FB5)
{
 CREATE_NODE
 CREATE_EDGE
 SET_NODE
 iterate ()
 /* iterate the experience map. Perform a graph relaxing algorithm to allow the map to partially converge */
Set_goal_pose_callback ()
Add_goal ()
/* Setting and handling goals */
 }
```

According above description, RAT-SLAM software architecture allows a multithread processing; therefore it can be parallelized in an heterogenous System on chip (Soc). However, the proposal is to proceed with the code profiling first, to stand on every function block use rate, calculate accurately the number of times each function is called and get an overview on the memory usage for each block.

## C. Code Profiling and Workload Identification

In order to have better understanding about how the algorithm is behaving once run on a CPU, the source code has been profiled using a profiling tool to measure the processing time of the functional blocks, the parameters dependencies and hence the functional blocks that could be parallelized.

Profilers are designed tools for analyzing and improving performance of code execution. They allow analyzing the algorithm, to measure, while the code is running, how long a routine takes to execute, how often it is called, where it is called from and how much of total time at some spot is spent executing that routine.

By using the opensource Callgrind profiling tool [12], which is considered as a binary instrumentation profiler, the call history could be recorded among functions for every main functional block. Data collected and represented in Table I, correspond to the CPU and memory workloads using an Intel® core i7 @1.8 GHZ CPU with 8 GB RAM. Also, the cache memory usage for reading and writing operations is given. This information is important because it reveals about congestion in different memory buses.

A fast information is then given about the algorithm behavior, to target where the code parallelization can be more efficient, of course an adequation architecture algorithm analysis is mandatory to have a deeper understanding and make the maximum optimizations that can be done.

TABLE I. RAT-SLAM C++ WORKLOAD ON A CPU

| FB | CPU workload % | Cache memory reads % | % Cache memory writes % |
|---|---|---|---|
| Pose cells | 28.98 | 28.30 | 29.99 |
| Local View Cells | 41.61 | 42.77 | 39.70 |
| Experience Map | 29.41 | 28.93 | 30.31 |

## D. CPU-GPU Parallelization

Computing architectures have known the generalization of the concept of parallelism thanks to their significant technological evolution in recent years. Parallelization has spread to the level of processor architectures, notably with multicore processors and superscalar computers. In this context, founders and manufacturers of graphics processors have developed their architectures to be able to use them in applications with generic processing by designing GPGPU (General Purpose Graphic Processing Unit). Development languages like CUDA, OpenCL or OpenGL have been designed to use the potential of these processors for massively parallel computing purposes.

To our knowledge, in the state of the art, there is no study on the performance evaluation of a bio-inspired SLAM algorithm on GPU-based architectures.

The interest here is to explore hybrid architectures based on CPU-GPU for the implementation and parallelization of a bio-inspired SLAM algorithm. A first challenge lays in the complex data structure which results in the nature of the studied algorithm. Another interest lies in the specificity of the parallelization and programming of CPU-GPU architectures, which differs from conventional methods of parallel programming requiring mapping optimization and processing synchronizing with different coprocessors and shared memories.

Hence, the RAT-SLAM algorithm was divided into blocks with well-defined inputs and outputs (in number, type and size of data). An algorithm consists of a processing description, so it is the same for the blocks composing it.

The modeling in this work is based on a graph representation that illustrates the dependencies between the different blocks of the algorithm.

In the case of a heterogeneous architecture, the execution time of each functional block and therefore of the algorithm strongly depends on the mapping, otherwise on the way in which the functional blocks are distributed between the different processing units. The goal of the mapping optimization is to find one or more implementations allowing to reach defined constraints of real-time.

Given the results about the workload of each functional block, knowing the cache memory number of read/write accesses, measuring the execution time of each block and sub-functions, and taking into consideration the algorithmic complexity analysis, the nature of interaction between functional blocks can be understood. So, functions that needs to be sent to GPGPU for a parallel processing are separated from the ones that need to remain on the host, because processing them on CPU will be adequate due to the sequential processing and to make benefit from the higher frequency of CPU.

So, based on the above, this study propose an optimized parallelization targeting a CPU-GPGPU heterogenous architecture, where all time-consuming functional blocks related to image processing, visual template generation, intermediate map creation, pose cell experience and matching, have been transferred to the GPGPU device as explained in Fig. 2. The CPU is then managing the sequential processes and acting as a host.

As per Fig. 2, the selected functions for a parallel computing were converted to CUDA. Also, for evaluation purposes, the algorithm is executed first on a laptop (see Table II for hardware specifications), then on a Jetson TX1 which is often used in automotive applications due its better performance compared to its predecessor TK1 (see Table III for Jetson Tegra X1 hardware specifications).

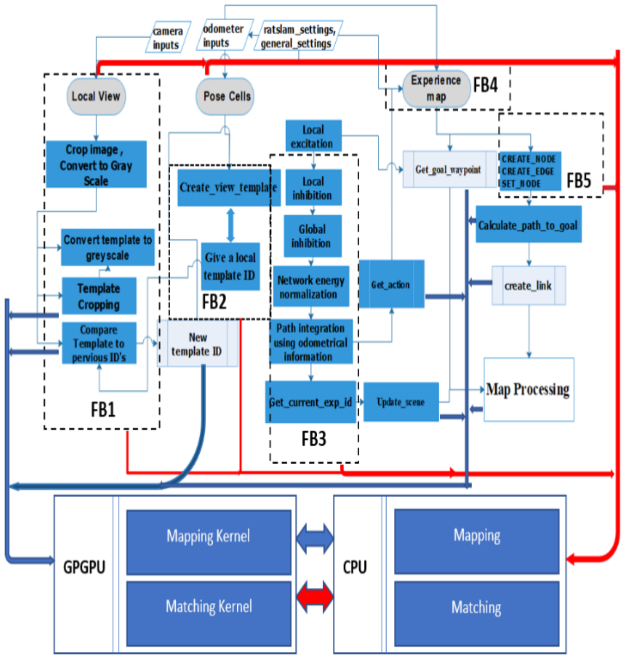


Fig. 2. Rat SLAM Algorithm flow Diagram: Interactions between main Functional Blocks and Sub-functions. Odometric Data and Image Inputs are Consecutively given by an IMU and a Camera from [9]. Buses in Red Represents the Bottleneck in Terms of Data Transfer. Bidirectional Arrows Represent Data Transferred between CPU and GPU.

TABLE II. HARDWARE SPECIFICATIONS OF THE CPU-GPU LAPTOP

| |
|---|
| CPU: Intel® core i7 @1.8 GHZ |
| GPU: NVIDIA GeForce MX110 |
| RAM: 8035640 KB |

TABLE III. HARDWARE SPECIFICATIONS OF JETSONTX1

| |
|---|
| CPU: Quad-Core ARM® Cortex®-A57 MP Core |
| GPU: 256-core NVIDIA Maxwell™ GPU |
| Memory: 4GB 64-bit LPDDR4 Memory |

## IV. HOOFR-SLAM

HOOFR is a bio-inspired binary detector, inspired from human retina. As introduced by Nguyan et al. [6], it is a novel model inspired from Hessian ORB - Overlapped FREAK (HOOFR) and based on the combination of the ORB detector [13-14] and the FREAK bio-inspired descriptor [15]. HOOFR use a similar method as ORB except that for filtering features it uses, instead of Harris filter, a Hessian Matrix represented by equation 3, together with Gaussian represented by equation 4 for smoothing.

$$H = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \tag{3}$$

Where $I$ is the pixel intensity and $\partial x, \partial y$ are the derivative in $x$ and $y$ direction, respectively.

$$G(x,y) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{4}$$

Where G(x,y) is a Gaussian 2D distribution and $\sigma$ is the standard deviation of the distribution.

Because the human retina has a better visual acuity, the rotation estimation is done based on the sum of gradients over a selected pair, by using long pairs to compute the global orientation, whereas select pairs are mainly with symmetric receptive fields with respect to the center, as per Fig. 3.

The set of selected pairs is done according to the following equation 5:

$$O = \frac{1}{M} \sum_{P_0 \in G} \left(I\left(P_0^{r_1}\right) - I\left(P_0^{r_2}\right)\right) \frac{P_0^{r_1} - P_0^{r_2}}{\left\| P_0^{r_1} - P_0^{r_2} \right\|} \tag{5}$$

where M is the number of pairs in G and P is the 2D vector of the spatial coordinates of the center of receptive field. However, compared to classical binary descriptor (like BRISK), HOOFR retinal pattern has more error in the orientation due to a larger receptive field allowing more estimation error, but this issue is solved by discretizing the space of orientations in much bigger steps, leading to more than 5 times smaller memory load (about 7 MB against 40 MB), and therefore a smaller execution time.

The main difference between Rat-SLAM and HOOFR-SLAM matching method, is that the first algorithm is comparing the template image against the source image by sliding it, then a sum of square is calculated to give the movement direction, so this method is operating directly on the pixel value, while HOOFR, from the other hand, use a feature matching method. In general, for this kind of techniques, the accuracy and efficiency are strongly dependent on the used detector and descriptor. An evaluation has been done in [4-5] show the impact of execution time and accuracy by using different feature detectors/descriptors combination. Fig. 4 is schematizing the flow of each algorithm front-end process.

Unlike RAT-SLAM, HOOFR-SLAM has been parallelized in a previous work [16] and implemented in different heterogenous architectures, so it is re-evaluated on a laptop, in order to determine the acceleration rate in a the same hardware architectures, and to have a fear comparison with results obtained for Rat-SLAM.

Since both bio-inspired algorithms has different images perception and since the camera position in HOOFR is frontal but lateral for RAT, any comparison using the same dataset will not be equitable, even after a modification, because the number of evaluated templates will not be the same. Therefore, a corresponding dataset is used for each algorithm in order to get time spent per iteration, images examples are given by Fig. 5.

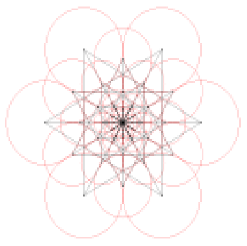


Fig. 3. Illustration of the Pairs Selected to Compute the Orientation.

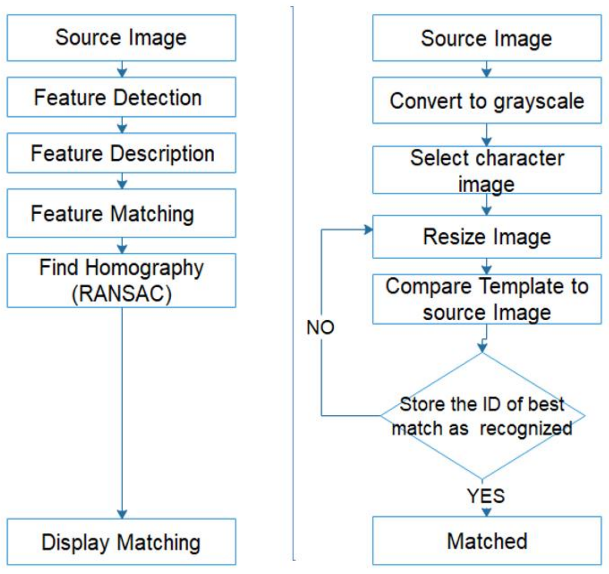


Fig. 4. Front End Process flow Diagram Showing differences between HOOFR-SLAM Feature Matching (Right) and RAT-SALM Template based Matching Method (Left).

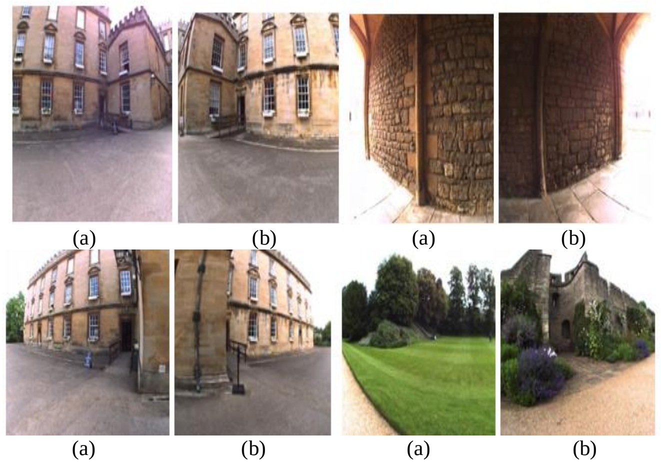


Fig. 5. Left and Right Images taken at different Lighting Conditions, by a RealSense Stereo Camera System, Simulating Rat Eyes and Placed in Lateral Position to have Similar Visual Perception as for a Rat. Right (a) and Left (b) Images used for RAT-SLAM, Images are Panoramic and May Appear Distorted to Human Eyes.

The evaluation was based on outdoor sequences where different lighting conditions are applied for better performances check.

## V. Experimental Results

A first evaluation for both selected bio-inspired algorithms is done based on a Multi-core Intel CPU and an NVIDIA GPGPU with a high-end 64-bit implementation. This configuration shown previously in Table II, allows the evaluation of different run modes on a CPU, multi-cores CPU and CPU-GPGPU combined. All presented results are the mean values of 20 runs.

Tables IV and V gives the processing times per iteration when the functional blocks are transferred to GPGPU as shown in Fig. 6.

TABLE IV. Average Processing Time (MS) Evaluation for RAT-SLAM with Parallel Implementation on a Multi-core CPU, and a CPU-GPGPU Laptop

| | | Intel® core i7 | NVIDIA GeForce MX110 | Acceleration % |
|---|---|---|---|---|
| | | Multi Core CPU | CPU-GPGPU | |
| Local View | FB1 | 183.17 | 31.93 | 82.57 |
| Pose cells | FB2 | 0.75 | 0.22 | 70.67 |
| | FB3 | 38.49 | 22.32 | 42.01 |
| Experience Map | FB4 | 133.93 | 84.81 | 36.68 |
| | FB5 | 0.16 | 0.08 | 50.00 |

TABLE V. Average Processing time (MS) Evaluation for RAT-SLAM with Parallel Implementation on an ARM CPU, and CPU--GPGPU using ARM-Nvidia TX1

| | | ARM®Cortex-A57 | NVIDIA Maxwell™ | Acceleration% |
|---|---|---|---|---|
| | | Multi Core CPU | CPU-GPGPU | |
| Local View | FB1 | 111.76 | 32.24 | 71.15 |
| Pose cells | FB2 | 0.75 | 0.29 | 61.33 |
| | FB3 | 26.59 | 20.94 | 21.25 |
| Experience Map | FB4 | 113.82 | 86.86 | 23.68 |
| | FB5 | 0.13 | 0.09 | 30.73 |

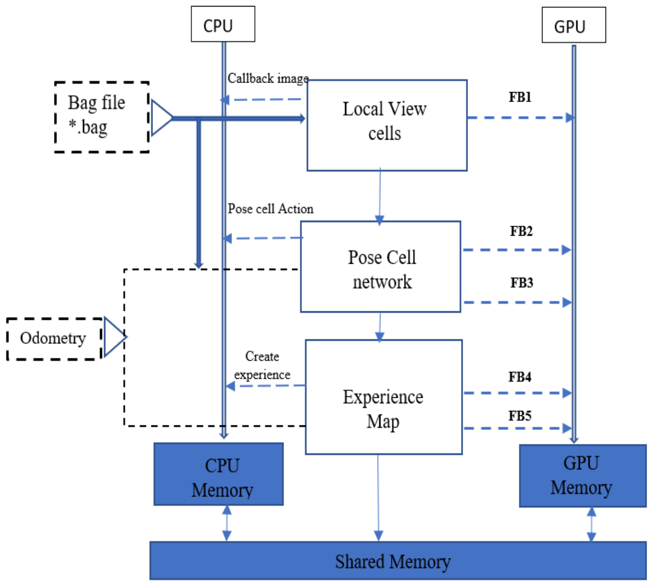
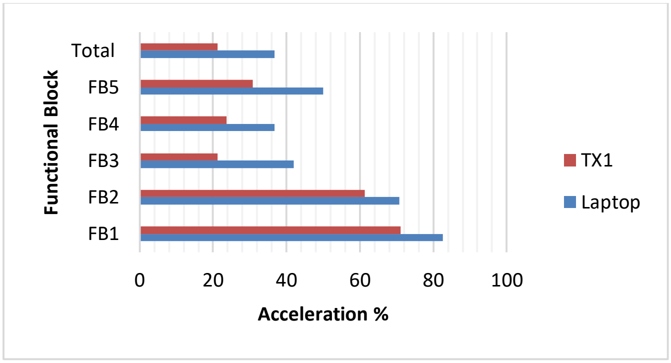


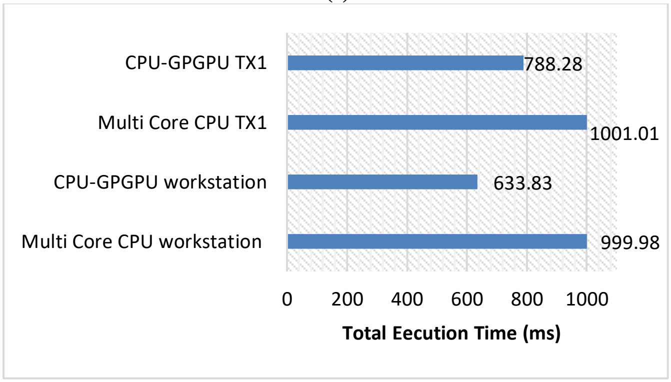
Fig. 6. Simplified Schematic Representation of Functional Blocks Sent to GPU and the One Remaining on CPU for Rat-SLAM.

The evaluation was achieved using the well-known open-source dataset Oxford New College, 2008 [9], settings and camera calibration parameters are shared by the same source. Images are panoramic and were converted to gray scale to match the most the rodent eyes perception to comply with the template matching process requirement.

Based on above results, the processing time of the most time-consuming functional blocks is drastically reduced using a parallel implementation by 36.68% on the laptop and 23.68% for TX1, this is due to the higher GPU performance and number of cores on the laptop compared to TX1, Fig. 7 is a graphical representation of the total execution time speed up for the algorithm on GPGPU.



(a)



(b)

Fig. 7. Comparison of Acceleration (A) and Total Execution time (B) for Oxford New College [9] on Laptop and on TX1 Hardware.

Below, graphs in Fig. 8, 9 and 10 represent the average execution timing measurement related to NVIDIA GeForce MX110 GPGPU and on ARM Cortex-A57 on TX1, also the power consumption in Watts for both Laptop CPU and GPU.

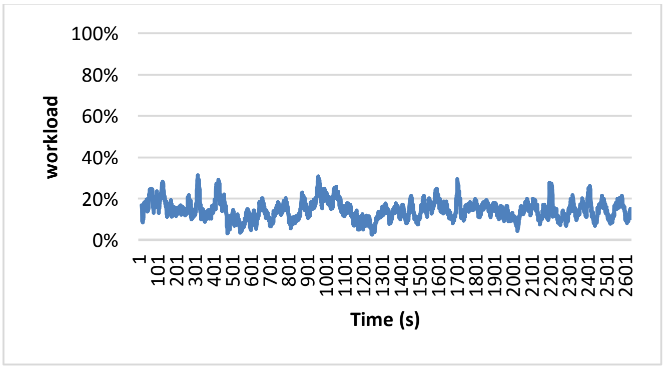


Fig. 8. Average Workload Evolution for RatSLAM on Laptop NVIDIA GeForce MX110 GPU, the Workload Everage is around 20% during the Total Execution Time of the Selected Dataset [9].

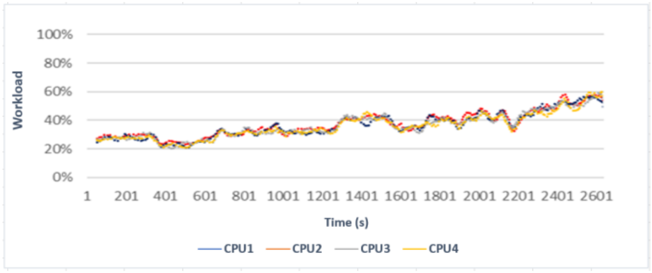


Fig. 9. Average Workload Evolution for RatSLAM on Laptop Intel Core i7 CPU, the Workload Everage is Variating between 20% and 60% during Execution of the Selected Dataset [9].

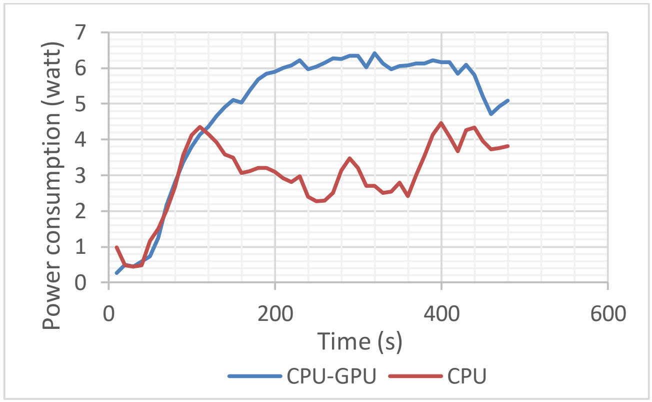


Fig. 10. The CPU-GPGPU Power Comsumption is Calcuted only for RATSLAM Algorithm when the Power Resulting form Operating System is not Considered.

As per above graphs , the CPU workload has been reduced by an average of 20% when running the RatSLAM algorithm, by running parallel functional blocks on GPGPU, but due to the nature of the algorithm some blocks cannot run on GPU because they are sequential and therefore are kept on CPU.

As expected the temperature will also increase on the CPU in the same way as per below Fig. 11, which can be considered as a week point for implementation of Rat SLAM in an embedded architecture where the use of a cooling system is not always possible.

An evaluation of HOOFR-SALM using KITTI-07 open source dataset, gives the results shown in Tables VI and VII.

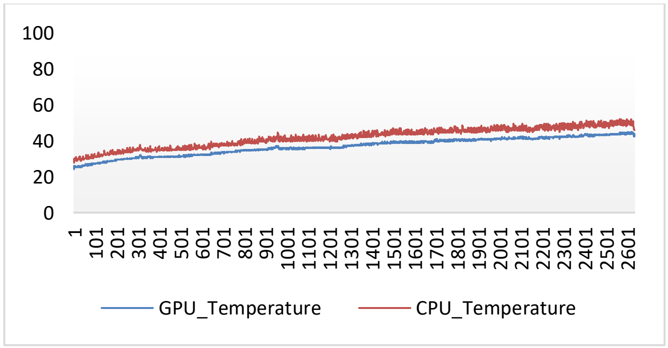


Fig. 11. Temperature Profile for Laptop CPU and GPU, it Increases Over Time Due to the Amount of Input Data.

TABLE VI. AVERAGE PROCESSING TIME (MS) EVALUATION FOR HOOFR-SLAM WITH PARALLEL IMPLEMENTATION ON MULTI-CORE CPU, CPU-GPGPU ON INTEL® CORE I7 / NVIDIA GEFORCE MX110

| | Intel® core i7 / NVIDIA GeForce MX110 | | | | |
|---|---|---|---|---|---|
| | CPU (ms) | Multi Core CPU (ms) | Acceleration % | CPU-GPU (ms) | Acceleration % |
| Extraction | 8.6 | 8.6 | 0.00 | 8.6 | 0.01 |
| Mapping | 52.79 | 52.76 | 0.06 | 27.88 | 47.15 |
| Loop detection | 15.38 | 15.34 | 0.26 | 8.05 | 47.52 |
| Map Processing | 0.48 | 0.45 | 6.25 | 0.19 | 58.85 |

TABLE VII. AVERAGE PROCESSING TIME (MS) EVALUATION FOR HOOFR-SLAM FOR PARALLEL IMPLEMENTATION ON MULTI-CORE CPU, CPU-GPGPU ON TX1 ARM®CORTEX-A57 NVIDIA MAXWELL™

| | ARM®Cortex-A57 NVIDIA Maxwell™ | | |
|---|---|---|---|
| | Multi Core CPU | CPU-GPU | Acceleration % |
| Extraction | 16.78 | 16.73 | 0.31 |
| Mapping | 21.92 | 16.47 | 25 |
| Loop detection | 21.92 | 16.47 | 25 |
| Map Processing | 0.58 | 0.4 | 31 |

## VI. CONCLUSION

This paper presented an algorithmic complexity study for two bio-inspired algorithms. It proposed an optimized parallel implementation on a CPU-GPU by studying, in a practical way, optimization possibilities for workload partitioning. It also presented understanding of bio-inspired algorithms with necessary techniques to accelerate there processing times for real time SLAM applications.

From one hand, based on above temporal evaluation results, a first conclusion is that the use of multiple CPU's cores cannot accelerate much the algorithm compared to one CPU core. This is due to the congestion of data in buses at the on-chip memory level since it is a shared resource for all CPU's cores. The memory is considered as a bottleneck in this case and using a higher CPU frequency or more memory will not help much.

From another hand, considering a real time sequence where the frequency is higher than 30fps, and based on the experimental results when executing both algorithms, it is clearly seen that despite the considerable acceleration, Rat SLAM still cannot fulfill the real time expectation as it should be executed in less than 33ms per frame, due to the matching sequence that is dependent on the number of images perceived by the camera sensor. Furthermore, the algorithm needs to keep previously seen templates in the memory for localization and to fine tune the map (loop closing), which has an impact on the final execution time.

In the case of HOOFR-SLAM, which is a feature-based approach that doesn't depend on the dataset size, the matching process time is not increased by increasing the number of input images. This is very important because the processing time will remain practically the same for all iterations and therefore the parallelization is more efficient.

These two studies covering the exploration of hybrid architectures based on GPU-CPU for the implementation and parallelization of bio-inspired SLAM applications, allowed to draw conclusions about the challenges to be met related to the complexity, the structure of data and the nature of the algorithms studied. The results presented in this paper confirm that future heterogeneous architectures will represent potential candidates to embed complex algorithms such as those of bio-inspired SLAM applications.

Future work will focus on the implementation of selected functional blocks on FPGA architectures in order to bring defined processing closer to the sensor and hence allow image processing on the fly and reserve the GPU for massively parallel processing.

## AKNOWLEDGMENT

This research is supported by Information Systems and Technologies Engineering Laboratory, LISTI. ENSA Ibn Zohr University, Agadir, Morocco and SATIE Paris-Saclay University, Orsay, France.

## REFERENCES

[1] T. Savaria and R. Balasubramanian, "V-SLAM: Vision-based simultaneous localization and map building for an autonomous mobile robot," 2010 IEEE Conference on Multisensor Fusion and Integration, Salt Lake City, UT, USA, 2010, pp. 1-6.

[2] Land, M.F.; Fernald, R.D. (1992). "The evolution of eyes". Annual Review of Neuroscience. 15: 1–29. doi:10.1146/annurev.ne.15.030192.000245. PMID 1575438

[3] Caves EM, Brandley NC, Johnsen S. Visual Acuity and the Evolution of Signals. Trends Ecol Evol. 2018;33(5):358-372. doi: 10.1016/j.tree.2018.03.001.

[4] M. Amraoui, R. Latif, A. Elouardi and A. Tajer, "Features Extractors Evaluation Based V-SLAM Applications," 2019 4th World Conference on Complex Systems (WCCS), Ouarzazate.

[5] M. Amraoui, R. Latif, A.E. Ouardi, A. Tajer "Feature Extractors Evaluation Based V-SLAM for Autonomous Vehicles", Advances in Science, Technology and Engineering Systems Journal, vol. 5, no. 5, pp. 1137-1146 (2020).

[6] D. Nguyen, A. El Ouardi, E. Aldea and S. Bouaziz, "HOOFR: An enhanced bio-inspired feature extractor," *2016 23rd International Conference on Pattern Recognition (ICPR)*, Cancun, 2016, pp. 2977-2982.

[7] Milford, M.J., Wyeth, G.F., Prasser, D.: Ratslam: a hippocampal model for simultaneous localization and mapping. In: 2004 IEEE International Conference on Robotics and Automation, Proceedings, ICRA 2004, vol. 1, pp. 403–408. IEEE (2004).

[8] Andreas Geiger , Philip Lenz , Christoph Stiller , Raquel Urtasun; "Vision meets Robotics: The KITTI Dataset"; International Journal of Robotics Research, IJRR,2013.

[9] M. Smith, I. Baldwin, Churchill, R. Paul, Newman, "The new college vision and laser data set", The International Journal of Robotics Research, vol.28, issn.0278-3649, May.2009.

[10] U.Muhammad, M.Tanvir, K.Khurshid, "Feature Based Correspondence: A Comparative Study on Image Matching Algorithms" International Journal of Advanced Computer Science and Applications(IJACSA), 7(3), 2016.

[11] D.Ball, S.Heath, , J.Wiles, et al. OpenRatSLAM: an open source brain-based SLAM system. Auton Robot 34, 149–176 (2013).

[12] Antonio J. Peña, Pavan Balaji, "A data-oriented profiler to assist in data partitioning and distribution for heterogeneous memory

in HPC Parallel Computing", Volume 51,2016, Pages 46-55, SSN 0167-8191.

[13] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, 2017.

[14] E.Adel, M.Elmogy,H.Elbakry, "Image Stitching System Based on ORB Feature-Based Technique and Compensation Blending" International Journal of Advanced Computer Science and Applications(IJACSA),6(9),2015.

[15] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint", In Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 510-517.

[16] Nguyen, Dai-Duong. "A vision system based real-time SLAM applications. (Un système de vision pour la localisation et cartographie temps-réel)." (2018).