Vincke *et al. EURASIP Journal on Embedded Systems* 2012, **2012**:5
http://jes.eurasipjournals.com/content/2012/1/5

EURASIP Journal on Embedded Systems
a SpringerOpen Journal

RESEARCH Open Access

Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems

Bastien Vincke[1*], Abdelhafid Elouardi[1] and Alain Lambert[2]

Abstract

Simultaneous localization and mapping (SLAM) is widely used by autonomous robots operating in unknown environments. Research community has developed numerous SLAM algorithms in the last 10 years. Several works have presented many algorithms' optimizations. However, they have not explored a system optimization from the system hardware architecture to the algorithmic development level. New computing technologies (SIMD coprocessors, DSP, multi-cores) can greatly accelerate the system processing but require rethinking the algorithm implementation. This article presents an efficient implementation of the EKF-SLAM algorithm on a multi-processor architecture. The algorithm-architecture adequacy aims to optimize the implementation of the SLAM algorithm on a low-cost and heterogeneous architecture (implementing an ARM processor with SIMD coprocessor and a DSP core). Experiments were conducted with an instrumented platform. Results aim to demonstrate that an optimized implementation of the algorithm, resulting from an optimization methodology, can help to design embedded systems implementing low-cost multiprocessor architecture operating under real-time constraints.

Introduction

Autonomous robots must be able to localize themselves. Simultaneous localization and mapping (SLAM) algorithms aim to build an environment map while estimating the robot pose. Many researches were conducted to develop SLAM algorithms like extended Kalman filter for SLAM (EKF-SLAM) [1,2], FAST SLAM [3], GRAPH SLAM [4], DP-SLAM [5] which aim to improve consistency, accuracy or robustness. Other algorithms derivate from the EKF-SLAM, such as algorithms using unscented Kalman filter (UKF) [6] which increases the localization accuracy against the classical EKF algorithm based on a linearized model. Only few works deal with the implementation of low-cost SLAM embedded systems.

Most of SLAM implementations rely on the use of accurate and dense measurements provided by expensive sensors like laser rangefinder sensors [7] or time of flight cameras [8]. High-priced smart sensors are not suitable to be integrated in most of embedded systems in commercial objectives or industrial applications.

Simultaneous localization and mapping systems using low-cost sensors have been recently designed. Abrate et al. [9] provide an implementation of the EKF-SLAM algorithm on a Khepera robot. The robot hosts limited range, sparse and noisy IR sensors. Experimental results have shown the importance of the sensor characteristics, the primitives (lines) extraction and data association. Yap and Shelton [10] use cheap, noisy and sparse sonar sensors embedded in a P3-DX robot. To cope with these low-cost sensors, the implemented SLAM algorithm uses a multi-scan approach and an orthogonality assumption to map indoor environments.

Classical SLAM algorithms are too computationally intensive to run on an embedded computing unit. They require at least laptop-level performances. Gifford et al. [11] present a low-cost approach to autonomous multi-robot mapping and exploration for unstructured environments. The robot hosts a Gumstix computing unit (600 Mhz), 6 IR scanning range arrays, a 3-axis gyroscope and odometers. Running DP-SLAM alone on the Gumstix with 15 particles takes on average 3 s per update. While

*Correspondence: bastien.vincke@u-psud.fr
[1] Univ Paris-Sud, CNRS, Institut d'Electronique Fondamentale, F-91405 Orsay, France
Full list of author information is available at the end of the article

using 25 particles, it takes more than 10 s per update. Authors have underlined the difficulty to find the right SLAM parameters to fit within the available computing power and the real-time processing. Magnenat et al. [12] present a system based on the co-design of a low-cost sensor (a slim rotating scanner), a SLAM algorithm, a computing unit, and an optimization methodology. The computing unit is based on an ARM processor (533 Mhz) running a FASTSLAM 2.0 algorithm [13]. Magnenat et al. [12] use an evolution strategy to find the best configuration of the algorithm and setting of the parameters.

As pointed out by [11,12], the first improvement of a SLAM algorithm is an efficient setting of the various parameters of the algorithm. Other modifications were investigated to reach real-time constraints. These modifications are necessary due to the low computing power and limited memory resources available on embedded systems. Features restriction for EKF-SLAM algorithm has been implemented to decrease the processing time [14]. Schroter et al. [15] focused on reducing the memory footprint of particle-based gridmap SLAM by sharing the map between several particles.

Robust laser-based SLAM navigation has long existed in robot applications, but systems implement sensors that, in some cases, are more expensive than the final product. Neato Robotics has developed a vacuum cleaner that implements a navigation system using a SLAM algorithm. The approach is based on a low-cost system implementing a designed laser rangefinder [16].

This article presents an efficient implementation of the EKF-SLAM algorithm on a multi-processor architecture. The approach is based on an algorithm implementation adequate to a defined architecture. The aim is to optimize the implementation of the SLAM algorithm on a low-cost and heterogeneous architecture implementing an SIMD coprocessor (NEON) and a DSP core. The hardware includes several low-cost sensors. As [17], we chose to use a low-cost camera (exteroceptive sensor) and odometers (proprioceptive sensors). Following [12], we efficiently tune the parameters of the SLAM algorithm. We improve on previous works by proposing an adequate implementation of the EKF-SLAM algorithm on a multiprocessing architecture (ARM processor, SIMD NEON coprocessor, DSP core). The specifications related to the NEON coprocessor and the DSP core improve the processing time and the system performance. Results aim to demonstrate that an optimized implementation of the algorithm, resulting from an evaluation methodology, can help to design embedded systems implementing low-cost multiprocessor architecture operating under real-time constraints.

Section "EKF-SLAM algorithm" introduces the EKF-SLAM algorithm. Section "Multiprocessor architecture and system configuration" presents the embedded multiprocessor architecture and the system configuration. Section "Evaluation methodology and algorithm implementation" details the evaluation methodology, provides a first algorithm implementation and analyzes this implementation in terms of processing time. A Hardware–software optimization is proposed and analyzed in Section "Hardware–software optimization and improvements". It presents SIMD optimizations and DSP parallelization. A performance comparison is then performed between the optimized and non-optimized instances. Finally, Section "Conclusion" concludes this article.

EKF-SLAM algorithm

Overview

Extended Kalman filter for SLAM estimates a state vector containing both the robot pose and the landmark locations. We consider that the robot is moving on a plane. The algorithm uses 3D points as landmarks. It uses proprioceptive sensors to compute a predicted vector and then corrects this state using exteroceptive sensors. In this article, we consider a wheeled robot embedding two odometers (attached to each rear wheel) and a camera.

State vector and covariance matrix

With N landmarks, the state vector is defined as:

$$\mathbf{x} = (x, z, \theta, x_{a_1}, y_{a_1}, z_{a_1}, \ldots, x_{a_N}, y_{a_N}, z_{a_N})^T \tag{1}$$

where:

- x,z are the ground coordinates (x-axis, z-axis) of the robot rear axle center. We suppose that the robot is always moving on the ground, so $y = 0$ (no elevation) and y does not appear in Equation (1).
- θ is the orientation of a local frame attached to the robot with respect to the global frame.
- $x_{a_1}, y_{a_1}, z_{a_1}, \ldots, x_{a_N}, y_{a_N}, z_{a_N}$ are the 3D coordinates of the N landmarks in the global frame.

The state covariance matrix is defined as:

$$\mathbf{P} = \begin{bmatrix} P_{xx} & P_{xz} & P_{x\theta} & P_{xx_{a_1}} & .. & P_{xz_{a_N}} \\ P_{zx} & P_{zz} & P_{z\theta} & P_{zx_{a_1}} & .. & P_{zz_{a_N}} \\ P_{\theta x} & P_{\theta z} & P_{\theta\theta} & P_{\theta x_{a_1}} & .. & P_{\theta z_{a_N}} \\ P_{x_{a_1}x} & P_{x_{a_1}z} & P_{x_{a_1}\theta} & P_{x_{a_1}x_{a_1}} & .. & P_{x_{a_1}z_{a_N}} \\ .. & .. & .. & .. & .. & .. \\ P_{z_{a_N}x} & P_{z_{a_N}z} & P_{z_{a_N}\theta} & P_{z_{a_N}x_{a_1}} & .. & P_{z_{a_N}z_{a_N}} \end{bmatrix} \tag{2}$$

Prediction

The prediction step relies on the measurements of the proprioceptive sensors, the odometers, embedded on our experimental platform. A non linear discrete-time state-space model is considered to describe the evolution of the robot configuration $\mathbf{x}$:

$$\mathbf{x}_{k|k-1} = \mathbf{f}(\mathbf{x}_{k-1|k-1}, \mathbf{u}_k) + \mathbf{v}_k \tag{3}$$

where $\mathbf{u}_k$ is a known two-dimensional control vector, assumed constant between the times indexed by $k-1$ and k, and $\mathbf{v}_k$ is an unknown state perturbation vector that accounts for the model uncertainties. $\mathbf{x}_{k-1|k-1}$ represents the state vector at time k-1, $\mathbf{x}_{k|k-1}$ represented the state vector after the prediction step, $\mathbf{x}_{k|k}$ represents the state vector after the estimation step. The classical evolution model, described in [18], is considered:

$$\mathbf{f}(\mathbf{x}_{k-1|k-1}, \delta s, \delta\theta) = \begin{pmatrix} x_{k-1} + \delta s \cos\left(\theta_{k-1} + \frac{\delta\theta}{2}\right) \\ z_{k-1} + \delta s \sin\left(\theta_{k-1} + \frac{\delta\theta}{2}\right) \\ \theta_{k-1} + \delta\theta \\ x_{a_1,k-1} \\ y_{a_1,k-1} \\ z_{a_1,k-1} \\ .. \\ .. \\ x_{a_N,k-1} \\ y_{a_N,k-1} \\ z_{a_N,k-1} \end{pmatrix} \tag{4}$$

where $\mathbf{u}_k = (\delta s, \delta\theta)$; δs is the longitudinal motion and $\delta\theta$ is the rotational motion [19]:

$$\begin{pmatrix} \delta s \\ \delta\theta \end{pmatrix} = \mathbf{g}(\varphi_l, \varphi_r) = \begin{pmatrix} \frac{w_r\delta\varphi_r + w_l\delta\varphi_l}{2} \\ \frac{w_r\delta\varphi_r - w_l\delta\varphi_l}{e} \end{pmatrix} \tag{5}$$

where:

- w_r and w_l are respectively the radius of the right and left wheel.
- e is the length of the rear axle.
- $\delta\varphi_i = \delta p_i \frac{2\pi}{\rho}$ with $i \in \{r, l\}$ (r=right, l=left), δp_i: number of steps, ρ: odometer resolution. $\delta\varphi_i$ is the angular movement of the right/left wheel.

The state covariance matrix is defined as:

$$\mathbf{P}_{k|k-1} = \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{P}_{k-1|k-1}\frac{\partial\mathbf{f}}{\partial\mathbf{x}}^T + \mathbf{Q}_k \tag{6}$$

where

- $\frac{\partial\mathbf{f}}{\partial\mathbf{x}} = \begin{bmatrix} 1 & 0 & -\delta s \sin\left(\theta_{k-1|k-1} + \frac{\delta\theta}{2}\right) & 0 & .. & 0 \\ 0 & 1 & \delta s \cos\left(\theta_{k-1|k-1} + \frac{\delta\theta}{2}\right) & 0 & .. & 0 \\ 0 & 0 & 1 & 0 & .. & 0 \\ 0 & 0 & 0 & 1 & .. & 0 \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & .. & 1 \end{bmatrix}$
- $\mathbf{Q}_k$ is the covariance matrix of the process noise.

Estimation

The estimation of the state is made using the camera which returns the position in the image (u_i, v_i) of the i-th landmark.

The innovation and its covariance matrix: The pinhole model is used to project a known landmark position into the image:

$$\begin{aligned} \begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} &= \text{pinhole}(x_{a_i}^{\text{cam}}, y_{a_i}^{\text{cam}}, z_{a_i}^{\text{cam}}) \\ &= \begin{bmatrix} fk_u & s_{uv} & c_u \\ 0 & fk_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \frac{x_{a_i}^{\text{cam}}}{z_{a_i}^{\text{cam}}} \\ \frac{y_{a_i}^{\text{cam}}}{z_{a_i}^{\text{cam}}} \\ 1 \end{pmatrix} \end{aligned} \tag{7}$$

where:

- (u_i, v_i) is the position of the i-th landmark in the image.
- $(x_{a_i}^{\text{cam}}, y_{a_i}^{\text{cam}}, z_{a_i}^{\text{cam}})$ is the position of the i-th landmark in the camera frame.
- f is the focal length.
- (k_u, k_v) is the number of pixels per unit length.
- s_{uv} is a factor accounting for the skew due to non-rectangular pixels. In our case, we take s_{uv}=0.

Equation (7) can be written as the predicted observation equation for a single landmark:

$$\mathbf{h}^i(\mathbf{x}_{k|k-1}) = \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} c_u + fk_u \frac{x_{a_i}^{\text{cam}}}{z_{a_i}^{\text{cam}}} \\ c_v + fk_v \frac{y_{a_i}^{\text{cam}}}{z_{a_i}^{\text{cam}}} \end{pmatrix} \tag{8}$$

The pose of a landmark in the camera frame is defined from its pose $(x_{a_i}, y_{a_i}, z_{a_i})$ in the global frame:

$$\begin{pmatrix} x_{a_i}^{\text{cam}} \\ y_{a_i}^{\text{cam}} \\ z_{a_i}^{\text{cam}} \end{pmatrix} = \left(\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{pmatrix} x_{a_i} - x \\ y_{a_i} \\ z_{a_i} - z \end{pmatrix} \right) - \begin{pmatrix} 0 \\ 0 \\ D \end{pmatrix} \tag{9}$$

Where D is the length between the camera and the robot rear axle center.

During the observation step, the algorithm matches M landmarks ($M <= N$) whose observations are added in $\mathbf{h}_k = \begin{pmatrix} \mathbf{h}^0 \\ .. \\ \mathbf{h}^{M-1} \end{pmatrix}$.

Thus, the innovation is:

$$\mathbf{Y}_k = \hat{\mathbf{z}}_k - \mathbf{h}_k(\mathbf{x}_{k|k-1}) \tag{10}$$

where $\hat{\mathbf{z}}_k$ is the measurement for all the M predicted observations.

The innovation covariance $\mathbf{S}_k$ is:

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{11}$$

where $\mathbf{H}_k$ is the Jacobian of $\mathbf{h}_k$ and $\mathbf{R}_k$ is the observation noise covariance.

State estimation: The state is updated using the classical EKF equations:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k\mathbf{S}_k^{-1} \\ \mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + \mathbf{K}_k\mathbf{Y}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \end{aligned} \tag{12}$$

Visual landmarks

The landmarks used in the observation equation are extracted from images. Landmark initialization defines the initial coordinates and the initial covariance of landmarks localization (also called interest points or features). In [20], we have evaluated the processing time of corner detectors like Harris, Shi-Tomasi or FAST. Harris and Shi-Tomasi detectors were more time consuming than the FAST detector and do not provide significantly better localization results than FAST. Consequently, there is no need to implement more sophisticated algorithms such as Harris or Shi and Tomasi. FAST [21] (Features from Accelerated Segment Test) corner detector relies on a simple test performed for a pixel p by examining a circle of 16 pixels (a Bresenham circle of radius 3) centered on p. A feature is detected at p if the intensities of at least 12 contiguous pixels are all above or all below the intensity of p with a threshold t. Even if this detector is not highly robust to noises and depends on a threshold it produces stable landmarks and is computationally very efficient [21].

The FAST detector [21] is related to the wedge-model style of detector evaluated using a circle surrounding a candidate pixel. To optimize the detector processing-time, this model is used to made a decision classifier which is applied to the image (Figure 1).

Matching based on zero-mean sum of squared differences

The EKF-SLAM matches the previously detected feature with a new one using zero-mean sum of squared differences (ZMSSD).

The covariance of the projected feature localization defines a searching area τ. This area includes the robot localization uncertainty and the landmarks localization uncertainty. We use the ZMSSD to find the best candidate point inside τ. For each candidate point $p : (p_x, p_y)$, the N_p value of the weighted ZMSSD is:

$$N_p = w(p_x, p_y) \times \text{ZMSSD} \tag{13}$$

and

$$\begin{aligned} \text{ZMSSD} = \sum_{i,j} \Bigg(& (d(i,j) - m_d) \\ & - \left(\text{im}\left(p_x + i - \frac{\text{des}}{2}, p_y + j - \frac{\text{des}}{2} \right) - m_i \right) \Bigg)^2 \end{aligned} \tag{14}$$

where:

- $w(p_x, p_y)$ is the Gaussian weights defined by the landmark covariance.
- $i \in [\,0; \text{des} - 1]$ and $j \in [\,0; \text{des} - 1]$ and des is the descriptor size.
- d is the feature descriptor.
- m_d and m_i are respectively the means of the pixel values in the descriptor and in the image window.
- im is the image.

The observation p_{obs} will be selected using $p = (p \in \tau | N_p = \min(N_{p_j}), \forall p_j \in \tau)$.

The descriptor, used to identify the landmark during the matching, is classically a small image window of 9×9 pixels to 16×16 pixels around the interest point. Davison [22] claims that this sort of descriptor is able to serve as long-term landmark feature.

Landmark initialization based on davison method

Landmark initialization consists of defining the initial coordinates and the initial covariance of landmarks (interest points). Various methods exist and can be classified as an undelayed or delayed method. Undelayed method adds landmarks with only one measurement whereas the delayed method needs two or more frames. We chose to use the widely spread delayed method proposed by [2] which is both efficient and adequate to implement.

Figure 1 Image (320 × 240 Pixels) of the embedded camera and result of the FAST detector.

Furthermore the work of Munguia and Grau [23] shows that the delayed method have the same performance as the undelayed method.

In order to compute the 3D depth of a newly detected landmark, as [2], we initialize a 3D line into the map along which the landmark must lie. This line starts at the estimated camera position and heads to infinity along the feature viewing direction. The line is composed of 100 particles which represent depth hypothesis. The prior probability used is uniform and the range is 0.5 to 15 m. At subsequent time, each particle (a feature depth hypothesis) is projected into the image, matched and its probability is re-weighted.

When the ratio of the standard deviation of depth to the expected value is below a threshold, the distribution is approximated as a Gaussian and the landmark is initialized. The landmark pose $A_i = (x_{a_i}, y_{a_i}, z_{a_i})$ is added to $\mathbf{x}$ and the A_i covariance is added into $\mathbf{P}$.

Multiprocessor architecture and system configuration

In order to test and validate the EKF-SLAM algorithm, experiments were conducted with an instrumented mobile robot called Minitruck [24]. The platform was teleoperated during the experiments. For our first evaluation, the experiment consists to operate inside a large corridor of our research lab (see Figure 2).

We have developed a system architecture on the top of a multi-processor board (Gumstix Overo) based on the OMAP3530 chip (see Figure 3). The OMAP chip integrate a RISC processors (ARM Cortex A8 500 Mhz) with an SIMD NEON coprocessor, a DSP (TMS320C64x + 430 Mhz) and a graphical processor unit (POWERVR SGX). This board communicates with an additional processor for control and data acquisition (Atmega168 16 Mhz).

Multiple sensors (odometers and a camera) are interfaced to this architecture (Figure 2). The variety of sensors enables us to evaluate the SLAM algorithms with different types of sensor data and take advantage of the information complementary of these sensors. Our objective is to evaluate the implementation of SLAM algorithms using land vehicles and sensors, like steering encoders and a camera.

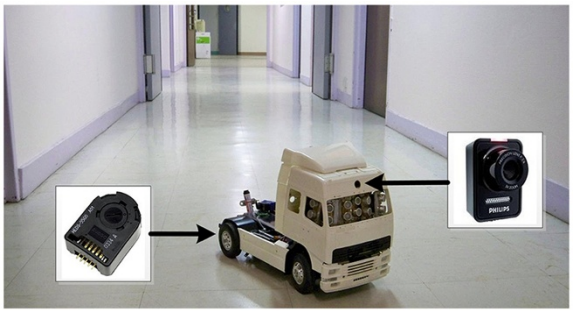

Figure 2 Minitruck in action embedding a multi-sensor system.

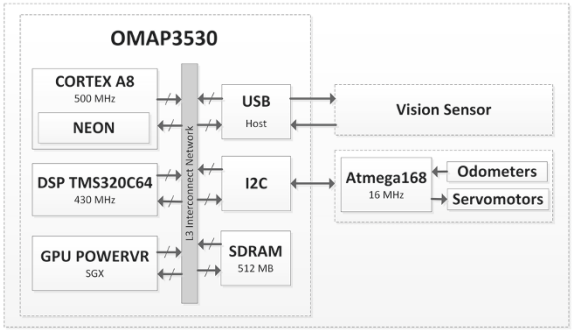

Figure 3 System architecture.

The use of wheel and steer encoders is obvious in robotics and navigation. Simple kinematic motion models can be used to integrate velocity and heading measurements from wheel and steer encoders to provide an estimation of the mobile robot location and orientation. Estimations are regularly subject to considerable errors due to misalignment, offsets and wheels slippage. It is possible to implement basic models to approximate and correct offset and slippage errors on-line leading to significant improvement of performances. We chose two HEDS 5540 odometers for our experimental vehicle.

The feature detection in SLAM application relies on the embedded sensors. We chose to achieve this extraction using a vision sensor (a cheap USB webcam, Philips SPC530NC, delivering 30 fps). We chose to use all possible images (30 fps) because it is much easier to perform point matching if the movement is small. Conventional approaches for vision systems design are usually based on general purpose computers interfaced with cameras. The new computing technologies (SIMD, DSP, multi-cores) can greatly accelerate algorithm processing, but require rethinking these algorithms by optimizing the parallelism. This parallel processing is pushed to integrate near the sensors parallel computing units [25]. We have used a Gumstix processing module based on OMAP3530 architecture. It is an heterogeneous architecture (ARM Cortex-A8 500 Mhz processor with power consumption less than 300 mW, SIMD NEON integrated coprocessor, DSP C64x processor and a 3D graphics accelerator) that communicates via a WLAN connection (802.11 g).

The WLAN connection is used only to control speed and direction of the vehicle. In the future, a dedicated algorithm to autonomous navigation will be implemented

and thus the WLAN connection will be used to achieve only the system monitoring. A coprocessor (ATMega168) takes care of data acquisition. It controls the robot speed and its direction using two pulse-width modulation (PWM) signals. It decodes signals coming from the odometers embedded in the rear wheels. It communicates with the main board using an I2C interface. This interface allows the main processor to retrieve odometers data and to send instructions corresponding to speed and direction.

To evaluate the designed system, an experiment was achieved in a corridor of our lab. Frames have been grabbed at 30 fps with 320×240 resolution. Odometer data were sampled at 30 Hz. During the experiment, references are periodically drawn on the ground by an embedded marker.

Evaluation methodology and algorithm implementation

Our evaluation methodology is based on the identification of the processing tasks requiring a significant computing time. It is based on several steps: we analyze first the execution time of tasks and their dependencies on the algorithm's parameters. A threshold is fixed for each parameter. The algorithm is then partitioned in order to have functional blocks (FBs) performing defined calculations. Each block is then evaluated to determine its processing time. Function blocks that require the most important execution time are then optimized to reduce the global processing time.

Algorithm 1 summarizes the main tasks of EKF-SLAM. The algorithm is composed of two process: Prediction and Correction. The correction process implements three tasks: matching, estimation and initialization.

Algorithm 1 EKF-SLAM

$\chi \leftarrow \emptyset$ ▷List of Landmarks for initialization
Robot pose initialization
while localization is required **do**
  DATA ← Sensors Data acquisition
  if DATA = (φ_l, φ_r) **then** ▷Odometer's data
    PREDICTION
    $(\delta s, \delta\theta) \leftarrow \mathbf{g}(\varphi_l, \varphi_r)$ (see Eq (5))
    $\mathbf{x}_{k|k-1} \leftarrow \mathbf{f}(\mathbf{x}_{k-1|k-1}, \delta s, \delta\theta)$ (see Eq (4))
    $\mathbf{P}_{k|k-1} \leftarrow \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{P}_{k-1|k-1} \frac{\partial \mathbf{f}}{\partial \mathbf{x}}^T + \mathbf{Q}_k$ (see Eq (6))
  else if DATA = Camera **then**
    FAST detector applied on the image
    MATCHING:
    for Each Landmark $\mathbf{N_i} \in \mathbf{x}_{k|k-1}$ **do**
      $u_i, v_i, \tau_i \leftarrow$ pinhole$(\mathbf{x}_{k|k-1}, \mathbf{N_i})$ (see Eq (8))
      if $(u_i, v_i) \in$ Camera Frame **then**
        $\hat{\mathbf{z}}_k \leftarrow$ ZMSSD$(\tau_i, \mathbf{N_i})$ (see Eq (13))
        $\mathbf{h}_k \leftarrow (u_i, v_i)$
        $\mathbf{Y}_k \leftarrow \hat{\mathbf{z}}_k - \mathbf{h}_k$
        $\mathbf{H}_k \leftarrow \frac{\partial \mathbf{h}_k}{\partial \mathbf{x}}\Big|_{\mathbf{x}_{k|k-1}}$
      end if
    end for
    ESTIMATION:
    $\mathbf{S}_k \leftarrow \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$ (see Eq (11))
    $\mathbf{K}_k \leftarrow \mathbf{P}_{k|k-1} \mathbf{H}_k \mathbf{S}_k^{-1}$ (see Eq (11))
    $\mathbf{x}_{k|k} \leftarrow \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{Y}_k$ (see Eq (12))
    $\mathbf{P}_{k|k} \leftarrow (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$ (see Eq (12))
    INITIALIZATION:
    for Each $\mathbf{L} \in \chi$ **do** ▷$\mathbf{L}$: Aspiring new Landmark
      $\mathbf{L}_{\text{obs}} \leftarrow ZMSSD(\mathbf{L})$ (see Eq (13))
      Update the particles weight according $\mathbf{L}_{\text{obs}}$ (see [2])
      Compute σ_{depth}, *depth*
      if $\frac{\sigma_{\text{depth}}}{\text{depth}} < \epsilon$ **then**
        Compute $\mathbf{L}$, $\mathbf{P_L}$
        append$(\mathbf{x_{k|k-1}}, \mathbf{L})$; append$(\mathbf{P_{k|k-1}}, \mathbf{P_L})$
        remove$(\chi, \mathbf{L})$
      end if
    end for
    if Lack of Landmark **then** ▷see [8]
      append$(\chi$, New_Landmarks)
    end if
  end if
end while

Prediction process

This phase updates the mobile robot position ($\mathbf{x}_{k|k-1}$) according to its proprioceptive data acquired from odometers (φ_l, φ_r). The processing time of the prediction process is constant. It just updates the 3D vector containing the robot pose and its 3×3 covariance matrix. During the prediction step, the landmarks localization and uncertainties do not change: landmarks are defined in the global frame.

Correction process

The processing time of the correction process is not constant. The following of this section studies the processing time of each task of the process and their dependencies.

Matching task Each landmark in the state vector must be projected in the camera frame using the pinhole model (see L. 2). The computing time of these projections depends only on the number of landmarks in the state vector (L. 2). For each projected landmark on the focal plane, ZMSSD matches an observation. Both the size of the descriptor and the size of the searching area τ will affect the computing time (see Equation (13)).

The processing time of the matching task depends on several parameters:

- The number of landmarks in the state vector.
- The number of visible landmarks on the focal plane.
- The size of the descriptor.
- Both the localization uncertainty of the mobile robot and the landmarks.

In practice, all the previously defined parameters should be set in order to bound the computing time. The first three parameters can be set by the users. The uncertainty depends on the followed path and cannot be bounded.

Estimation task The estimation task uses the classical Kalman equations to update both the robot and landmarks uncertainties. The processing time of the estimation task is time-consuming and depends on:

- The number of landmarks in the state vector.
- The number of landmarks observed.

The size of the matrix and thus the computing cost of the matrix multiplication in the Equations (11) and (12) depend on the number of landmarks in the state vector. Moreover, Equation (11) depends on the number of landmarks observed. As for the matching process, these parameters (size of the state vector and number of observations) should be bounded in order to achieve this estimation task in a constant computing time.

Initialization task For each landmark under initialization, each particle (a feature depth hypothesis) is projected into the image, matched and its probability is re-weighted. If there is a lack of landmarks under initialization, we add aspiring new landmarks. The processing time of the initialization task depends on:

- The number of landmarks being initialized.
- The size of the descriptor.
- Both the localization uncertainty of the mobile robot and the landmark.

The number of landmarks being initialized and the size of the descriptors can be bounded. For each landmark being initialized, we have to update the probability of each localization hypothesis using a matching process. As for the matching task, the computing time depends on the localization uncertainty of the mobile robot and the landmarks.

Thresholds definition

Previous section shows that the computation time of each task of the EKF-SLAM algorithm depends on many variables. For real-time implementation, it is important to get a constant, or at least a bounded computation time. To solve this constraint we have to:

- set the maximum number of landmarks in the state vector. The size of the state vector will be fixed. Therefore, no dynamic memory allocation will be needed.
- set the maximum number of landmarks observed. This keeps the computation time of the estimation task constant using a fixed size matrix multiplication.
- set the maximum number of landmarks being initialized in order to bound the computation time of the initialization task. Unfortunately, it will not be sufficient to keep the computation time of the initialization task constant due to its internal matching step.
- bound the computing time induced by the uncertainties. The only solution to get a bounded global-processing-time is to set a maximum execution time for the matching task. Due to the constant processing time of the prediction and the estimation task, the execution time of both the matching and initialization task can be bounded (33 ms - ($t_{\text{prediction}}$ + $t_{\text{estimation}}$)). We chose to use all possible images (30 fps).We set a maximum execution time for the matching task. The algorithm proceeds in a way to match a maximum of landmarks in a bounded time. The initialization task has a dynamic execution time depending on the real processing time of the matching task and the number of landmarks being initialized. The lower bound of this dynamic execution allows at least a minimum number of landmarks to be initialized.

Map management

To keep the size of the state vector constant, we need to delete some landmarks when inserting new ones. The new state vector includes new landmarks (whose initialization has just been performed) and previously used landmarks. Auat Cheein and Carelli [26] proposes an efficient method to select landmarks for the estimation task. It is based on the evaluation of the influence of a given feature on the convergence of the state covariance matrix. The method matches all possible landmarks and computes $(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)$ from Equation (12). Unfortunately, we cannot implement it exactly as proposed by [26] due to the high computing time. We chose to add the landmarks, based on the previous estimation step, by selecting the previous landmarks which have the best previous influence on the convergence of the state covariance matrix. At time k, we select the landmark which had the smallest $(\mathbf{I} - \mathbf{K}_{k-1}\mathbf{H}_{k-1})$.

Table 1 Functional block partitioning

	Functional block (FB)	Description	Line
1	Prediction	The entire prediction process	7, 8, 9
2	FAST	The FAST corner detector application	11
3	Landmark projection	The projection of one landmark on the camera plane	14
4	ZMSSD-M	The correlation computation between one candidate point of the image and one descriptor during the Matching Task	16
5	$\mathbf{H}_i$	$\mathbf{H}_i$ computation for one observation	19
6	Estimation	The entire estimation task	23 to 26
7	ZMSSD-I	The correlation computation between one candidate point of the image and one descriptor during the Initialization Task	29
8	Weight updating	The update of the particle weight for the initialization step	30, 31
9	Addition of a new landmark	The insertion of a new landmark under initialization	39

Functional block partitioning

All the previously defined tasks do not have a fixed computing time, their computing time depends on the experiment. We have defined FBs which have a fixed computing times to optimize the implementation. The computing time of the FB do not depend on the experiment. Experiments will only affect the number of iterations of some FBs(3,4,5,6,7,8 and 9). From the previous algorithm, we have defined 9 FBs and their runtimes are studied in below Table 1.

Each FB has a fixed computing time and some FB can occur more than one time (Landmark projection, ZMSSD, $\mathbf{H}_i$, Weight updating, Addition of a new landmark).

Processing time evaluation

As an application scenario, the robot moves over a square of 6 m side. At the end of the trajectory, it joined the initial starting position. Using only odometers, the final localization has an error of 1.6 m. With the EKF-SLAM algorithm, the localization has been significantly improved. The final error is approximately 0.4 m. EKF-SLAM includes all viewed landmarks in the state vector. Indeed, the localization result depends on the number of landmarks but the size of the state vector and the number of observations must be bounded to achieve a bounded computing time. The overall accuracy of the EKF-SLAM depends on the number of the landmarks in the state vector and the matched observations. The accuracy of the localization depends monotonically on the number of processed landmarks.

The given EKF-SLAM (Algorithm 1) is processed sequentially on the embedded ARM processor operating at 500 MHz (no coprocessor is implemented). In the following, all times given correspond to times evaluated on the embedded system using the ARM processor. The data acquisition time is constant:

- The odometer data acquisition is achieved in 0.7 ms (this processing time is due to the I2C communication with the Atmega168 processor).
- Each image acquisition takes 1.8 ms (due to USB data transfer).

The prediction step does not require significant processing time, it takes only 0.093 ms per iteration. As for the matching task, the estimation task cannot be achieved in a constant processing time. Estimation task processing time depends on the total number of landmarks and the number of matched landmarks. Figure 4 shows the processing time of the estimation task according to the number of landmarks in the state vector. The estimation task is entirely processed on the ARM processor (no use of coprocessor). Obviously it will be impossible to take into account all the landmarks detected when the algorithm is processed: the computation time will be

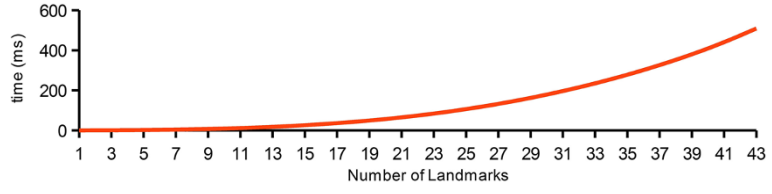

Figure 4 Processing time of the estimation task.

higher than the 33 ms allowed. It is necessary to find a compromise between the number of landmarks and the processing time.

Experimental results

An experiment was conducted to evaluate the processing time of the different blocks of the algorithm (including tasks with unboundable processing time). For this experiment, we set the size of the descriptor to 16×16 pixels and we set the thresholds as follows:

- Maximum number of landmarks in the state vector: 25.
- Maximum number of observed landmarks: 25.
- Maximum number of landmarks being initialized: 20.

First, we can analyze the runtime of the 8 previously defined FBs of the algorithm. We have used the integrated cycle counter register (CCNT) of the ARM processor to compute the processing time of each FB. The prediction process (FB1) occurs in 0.093 ms. Table 2 summarizes, for the other FBs, the processing time per iteration, the mean of the number of iterations and the mean of the processing time per correction process. The estimation task could not be processed in some iterations of the correction process, especially when there is no matched landmark.

The mean processing time by frame is approximately 80.8 ms which corresponds to the sum of all processing times: prediction process (FB1) and correction process (FB2 to FB9). The processing time of the estimation task (FB6) is approximately 70.5 ms and it represents about 87% of the global processing time. The FAST detector (FB2) represents 3.4 ms. The ZMSSD-M task (FB4) takes 2.63 ms per correction process. Finally, the initialization task (FB7, FB8 and FB9) takes 3.9 ms. These six FBs represent 99.6% of the global processing time. We focused on an efficient implementation of these FBs to enhance the global processing time.

Hardware–software optimization and improvements

OMAP3530 architecture description

The OMAP3530 is an heterogeneous architecture designed by TI (Texas Instruments) and implements an ARM Cortex-A8 500 MHz processor, a NEON coprocessor with SIMD instructions, a DSP C64x processor and a 3D graphics accelerator.

The NEON unit is similar to the MMX and SSE extensions existing on an X86 processor. It is optimized for Single Instruction Multiple Data (SIMD) operations. The NEON unit has two floating point pipelines, an integer pipeline and a 128 bits load/store/permute pipeline. An efficient implementation on the SIMD NEON architecture improves the processing time. NEON instructions perform "Packed SIMD" processing as follows:

- Registers are considered as vectors of the same data type elements
- Data types can be: signed/unsigned 8, 16, 32, 64-bits or single precision floating point
- Instructions perform the same operation on multiple data simultaneously as shown in Figure 5. The number of simultaneous operations depends on the data type: NEON supports up to 16 operations at the same time using 8-bits data.

SIMD optimization results

In the Algorithm 1, the time-consuming FBs are: the estimation block (FB6), the initialization blocks (FB7, FB8 and FB9), the FAST detector block (FB2) and the ZMSSD-M block (FB4). FAST detector is already an optimized instance using machine learning [21]. Moreover, FAST has been already implemented on an FPGA based architecture [27]. We chose to optimize the other FBs. The matching task computes ZMSSD which computes the image correlation. It performs the same operation (addition, subtraction, multiplication and comparison) on

Table 2 Processing time of the correction process FBs on the main processor (ARM)

Functional block (FB)	Processing time per iteration (μs)	Mean of the number of iterations per correction process	Mean of the processing time per correction process (μs)
2. FAST	3400	1	3400
3. Landmark projection	9	19	180
4. ZMSSD-M	11.29	233	2630
5. $\mathbf{H}_i$	14.5	4.5	66
6. Estimation	88845	0.8	70568
7. ZMSSD-I	11.29	123	1388
8. Weight updating	638	4.0	2586
9. Addition of a new landmark	103	0.18	18

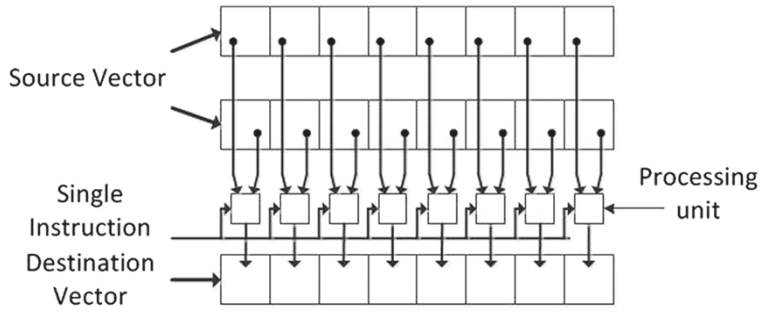

Figure 5 Data processing in a NEON architecture.

8 bits data. The computation of the ZMSSD can be optimized using the SIMD NEON architecture. The estimation task is based on floating point matrix multiplication, it could efficiently be optimized using the SIMD NEON coprocessor (the ARM Cortex A8 does not include any floating point unit (FPU)). The initialization FBs will be studied at Section "Parallel implementation on a DSP processor".

ZMSSD (FB4)

The EKF-SLAM matches features using ZMSSD. ZMSSD is computed for each landmark using Equation 2. We chose to use a descriptor with 16×16 8-bits pixels size due to the efficiency of SIMD NEON architecture to deal with 128/64 bits vectors.

Basic implementation The basic implementation of the ZMSSD function block computes the means of the pixel values in a window m_i (m_d can be precalculated when the landmark is detected). Then the ZMSSD (*ZMSSD*) is computed using loops (Algorithm 2).

Algorithm 2 Basic ZMSSD

1: $m_i \leftarrow 0$
2: ZMSSD $\leftarrow 0$
3: **for Each** $i \in [0; \text{des}-1], j \in [0; \text{des}-1]$ **do**
4: $m_i \leftarrow m_i + \text{im}(p_x + i - \frac{\text{des}}{2}, p_y + j - \frac{\text{des}}{2})$
5: **end for**
6: $m_i \leftarrow m_i/(\text{des} \times \text{des})$
7: **for Each** $i \in [0; \text{des}-1], j \in [0; \text{des}-1]$ **do**
8: ZMSSD $\leftarrow$ ZMSSD $+ \sum_{i,j}((d(i,j) - m_d) - (\text{im}(p_x + i - \frac{\text{des}}{2}, p_y + j - \frac{\text{des}}{2}) - m_i))^2$
9: **end for**

This implementation takes 12.60 μs on the ARM processor.

Efficient scalar implementation The second implementation aims to modify the calculation of ZMSSD in order to avoid the use of two loops. Formally the ZMSSD is written as:

$$\text{ZMSSD} = \sum_{i,j}((d - m_d) - (\text{im} - m_i))^2 \tag{15}$$

where $d = d(i,j)$ and $\text{im} = \text{im}(p_x + i - \frac{\text{des}}{2}, p_y + j - \frac{\text{des}}{2})$

By expanding the ZMSSD, we obtain:

$$\text{ZMSSD} = \sum_{i,j}((d-m_d)^2 - 2(d-m_d)(\text{im}-m_i) + (\text{im}-m_i)^2) \tag{16}$$

$$= \sum_{i,j}(d^2 - 2d.m_d + m_d^2 - 2d.\text{im} + 2d.m_i + 2m_d.\text{im} - 2m_d.m_i + \text{im}^2 - 2\text{im}.m_i + m_i^2) \tag{17}$$

Using $m_d = \sum_{kl} \frac{d(k,l)}{\text{des}\times\text{des}}$ and $m_i = \sum_{kl} \frac{\text{im}(p_x+k-\frac{\text{des}}{2}, p_y+l-\frac{\text{des}}{2})}{\text{des}\times\text{des}}$, we simplify the sum:

$$\sum_{i,j} m_d = \sum_{i,j}\sum_{kl} \frac{d(k,l)}{\text{des} \times \text{des}} \tag{18}$$

$$= \sum_{kl} d(k,l) \tag{19}$$

$$\sum_{i,j} m_i = \sum_{i,j}\sum_{kl} \frac{\text{im}(p_x + k - \frac{\text{des}}{2}, p_y + l - \frac{\text{des}}{2})}{\text{des} \times \text{des}} \tag{20}$$

$$= \sum_{kl} \text{im}\left(p_x + k - \frac{\text{des}}{2}, p_y + l - \frac{\text{des}}{2}\right) \tag{21}$$

The equation becomes:

$$\text{ZMSSD} = \left[2\sum_{i,j}\text{dim} - \left(\sum_{i,j} d\right)^2 - \left(\sum_{i,j}\text{im}\right)^2\right] /(\text{des} \times \text{des}) + \sum_{i,j} d^2 + \sum_{i,j}\text{im}^2 - 2\sum_{i,j}\text{dim} \tag{22}$$

Using the notation:

- $\mathrm{Sd} = \sum_{i,j} d(i,j)$ the sum of the descriptor pixels (this sum can be precalculated).
- $\mathrm{Si} = \sum_{i,j} \mathrm{im}(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2})$ the sum of the image pixels.
- $\mathrm{SSi} = \sum_{i,j} \mathrm{im}(p_x+i-\frac{\mathrm{des}}{2}, p_y+j-\frac{\mathrm{des}}{2}) \times \mathrm{im}(p_x+i-\frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2})$ the sum of squared image pixel values.
- $\mathrm{SSd} = \sum_{i,j} d(i,j) \times d(i,j)$ the sum of the squared descriptor pixel values (this sum can be precalculated).
- $\mathrm{Sdi} = \sum_{i,j} d(i,j) \mathrm{im}(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2})$ the sum of the product of the descriptor pixels and the image pixels.

The final equation is:

$$\begin{aligned}\mathrm{ZMSSD} =& [\,((2Sd \times Si) - Sd^2 - Si^2)/(\mathrm{des} \times \mathrm{des})] \\ &+ SSi + SSd - 2Sdi \end{aligned} \tag{23}$$

The implementation of Algorithm 2 becomes Algorithm 3:

Algorithm 3 Efficient scalar ZMSSD

1: $Si \leftarrow 0$
2: $SSi \leftarrow 0$
3: $Sdi \leftarrow 0$
4: **for Each** $i \in [\,0; \mathrm{des} - 1], j \in [\,0; \mathrm{des} - 1]$ **do**
5: $Si \leftarrow Si + \mathrm{im}(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2})$
6: $SSi \leftarrow SSi + \mathrm{im}(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2}) \times \mathrm{im}(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2})$
7: $Sdi \leftarrow Sdi + \mathrm{im}(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2}) \times d(i,j)$
8: **end for**
9: $\mathrm{ZMSSD} \leftarrow (((2Sd \times Si) - Sd^2 - Si^2)/(\mathrm{des} \times \mathrm{des})) + SSi + SSd - 2Sdi$

In this instance, we use only one loop. This reduces memory access. Using this implementation, the computing time decrease from 12.60μs to 11.29μs.

Vector implementation SIMD NEON architecture allows vector processing and performs the same operation on all the vector processing-units. We have implemented a vectorized instance of the ZMSSD functional block as follows (Algorithm 4):

Algorithm 4 SIMD vectorized ZMSSD

1: $V8x8\ V\mathrm{image} \leftarrow 0$ ▷ V8x8: 8×8 bits vector
2: $V8x8\ V\mathrm{descriptor} \leftarrow 0$
3: $V16x8\ VSi \leftarrow 0$ ▷V16x8: 8×16 bits vector
4: $V32x4\ VSSi \leftarrow 0$ ▷V32x4: 4×32 bits vector
5: $V32x4\ VSdi \leftarrow 0$
6: **for Each** $i \in [\,0; \mathrm{des} - 1], j = 0, 8$ **do**
7: $V\mathrm{image} \leftarrow \mathrm{load}_8(im(p_x + i - \frac{\mathrm{des}}{2}, p_y + j - \frac{\mathrm{des}}{2}))$ ▷load 8 pixels
8: $V\mathrm{descriptor} \leftarrow \mathrm{load}_8(d(i,j))$
9: $VSi \leftarrow VSi + V\mathrm{image}$
10: $VSSi \leftarrow VSSi + V\mathrm{image} \times V\mathrm{image}$
11: $VSSi \leftarrow VSSi + V\mathrm{image} \times V\mathrm{image}$
12: **end for**
13: $Si \leftarrow \mathrm{sum}(VSi)$ ▷Sums the component of a vectors
14: $SSi \leftarrow \mathrm{sum}(VSSi)$
15: $Sdi \leftarrow \mathrm{sum}(VSdi)$
16: $\mathrm{ZMSSD} \leftarrow (((2Sd \times Si) - Sd^2 - Si^2)/256) + SSi + SSd - 2Sdi$

This instance uses 8 pixels at time. SIMD NEON architecture allows computing eight addition or eight multiplication simultaneously. The processing time of the vector implementation decreases to 1.27 μs.

Computation time results Table 3 summarizes the processing time of the three different implementations of the ZMSSD functional block. The SIMD implementation is approximately 10 times faster than a basic implementation.

Estimation (FB6)

ARM Cortex A8 do not integrate a FPU. That's why the processing time of the estimation FB is significant (Figure 4). To optimize the matrix multiplication, we have used the EIGEN3 library [28] which provides SIMD NEON optimized functions. Figure 6 presents the results of the processing-time of the estimation task implemented on the ARM processor (non-optimized task) and those using the SIMD NEON coprocessor (optimized task). The processing time of the optimized task is approximately eight times faster than those of the non-optimized one. This gain is due to the lack of the FPU in the Cortex A8 and to the efficiency of the NEON to evaluate a multiply and accumulate instruction in only one CPU cycle.

Parallel implementation on a DSP processor

Digital signal processors (DSP) are usually used in vision systems [29]. They integrate a number of resources that serve to enhance image processing versatility. The use of digital signal processing with data sharing ensures that image processing will be achieved in parallel. With a DSP based image processing, it is possible to parallelize the

Table 3 ZMSSD processing time

	Processing time	Percentage of the basic implementation
Basic implementation	12.60	100
Scalar implementation	11.29	89.6
SIMD implementation	1.27	10.8

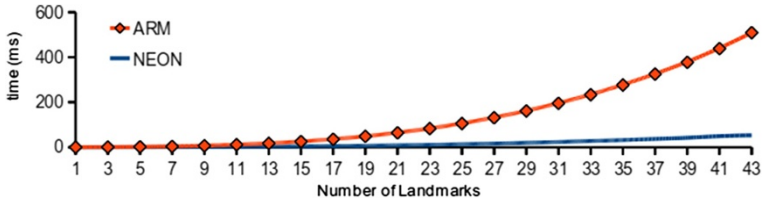

Figure 6 Processing time of the estimation task on the ARM and NEON coprocessor.

EKF-SLAM algorithm on the multiprocessor architecture (ARM, NEON and DSP processors). This allows enhancing the global processing time especially when we consider to operate in real-time constraints. The landmarks matching (FB3 to FB5) and the robot position estimation (FB6) tasks must be processed sequentially. Fortunately, the initialization tasks (FB7, FB8 and FB9) can run simultaneously with the matching and estimation tasks.

Rethinking the implementation to obtain a parallel implementation, the instance of Algorithm 1 with block partitioning leads to the Algorithm 5.

Algorithm 5 Multiprocessed EKF-SLAM

1: Robot pose initialization
2: **while** localization is required **do**
3: **if** DATA = Odometers **then**
4: PREDICTION
5: **else if** DATA = Camera **then**
6: FAST detector
7: **ARM Processor** MATCHING and ESTIMATION (FB 3, 4, 5 and 6)
8: **DSP Processor** INITIALIZATION (FB 7, 8 and 9)
9: **end if**
10: **end while**

The architecture of the OMAP3530 can interface the ARM and DSP processors using a shared memory. Figure 7 shows the data transfer mechanism using a shared DDR memory area. For each acquired image, the ARM processor writes the image (320×240 pixels), the robot position and its uncertainty on the shared memory. Data transfer between the ARM processor and DSP processor for a 320×240 gray image is done in one millisecond. When the initialization of a landmark is completed, the DSP processor returns the position and the uncertainty of possible new landmarks.

Global results

We have improved the EKF SLAM implementation using the SIMD NEON coprocessor and the DSP processor. We have implemented the matching and estimation tasks on a NEON coprocessor and the initialization tasks on a DSP processor. FAST corner detector is already an optimized algorithm using machine learning [21]. For the latest experiment, we set the same thresholds as Section "Experimental results".

Table 4 summarizes the processing time per iteration and the mean processing time per Frame of each FB. The computing time of the initialization task (blocks 7, 8 and 9) implemented on the DSP processor is approximately 4.0 ms. The DSP processor computes the initialization task while the ARM-NEON processors compute the prediction, FAST detection, matching and estimation tasks.

With this implementation and since the processing-time of the initialization task (4.0 ms) is smaller compared to the sum of the processing times of the matching and estimation tasks (13.0 ms for blocks 3, 4, 5 and 6), the overall computing time is reduced to the sum of the processing-times of the prediction process (0.093 ms),

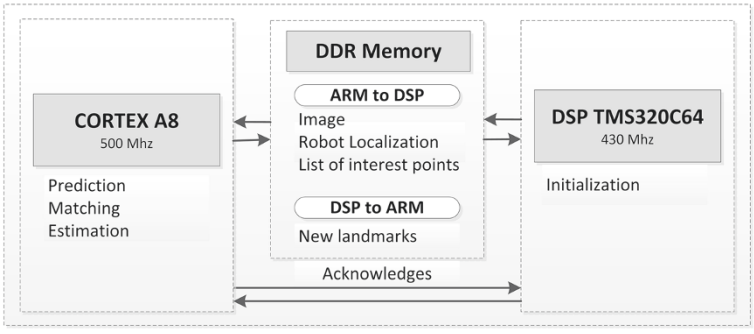

Figure 7 ARM-DSP interface with a shared memory.

Table 4 FBs processing times on ARM, NEON and DSP processors

Functional bloc (FB)	Nonoptimized implementation (μs) ARM only		Optimized implementation (μs)		
	Processing time per iteration	Mean processing time per frame	Processing time per iteration	Mean processing time per frame	Processing unit
1. Prediction	93	93	93	93	ARM
2. FAST	3400	3400	3400	3400	ARM
3. Landmark projection	9	180	9	180	ARM
4. ZMSSD-M	11.29	2630	1.27	295	NEON
5. $\mathbf{H}_i$	14.5	66	14.5	66	ARM
6. Estimation	88845	70568	15690	12552	NEON
Initialization task (FB7, 8 and 9)	3992	3922	4025	4025	DSP
Total	–	80859	–	16586	–

the FAST detector (3.4 ms), the matching and estimation tasks (13.0 ms). The mean processing time per frame with the optimized implementation is 17.6 ms (we add 1 ms for the ARM/DSP data transfer) whereas the nonoptimized implementation has a processing time of 80.85 ms. The optimized processing time represents 22% of the nonoptimized one. The processing time has been reduced by 78%.

Conclusion

This article proposed an efficient implementation of the EKF-SLAM algorithm on a multiprocessor architecture. The overall accuracy of the EKF-SLAM depends on the number of the landmarks in the state vector and the matched observations. Both are linked to the time allowed to the embedded architecture to compute the robot pose. Based on the application constraints (real-time localization) and an evaluation methodology, we have implemented the algorithm in consideration of the underlying hardware architecture. A runtime analyses shows that the FBs and the initialization task represents 99.6% of the global processing time. We have used an optimized instance of the FAST detector. Two FBs (in matching and estimation tasks) have been optimized on an SIMD NEON architecture. The initialization task has been parallelized on a DSP processor. This optimization required a modification of the algorithm implementation. Using the optimized implementation, the global processing time was reduced by a factor equal to 4.7. The results demonstrate that an embedded systems (with a low-cost multiprocessor architecture) can operate under real-time constraints, if the software implementation is designed carefully. To scale with larger environment, we are going to include an approach of local/global mapping as proposed by [30]. Using this approach, we will be able to map larger environment. The map joining system will be implemented on the GPU coprocessor integrated on the OMAP3530.

Other future developments will be centered around a Hardware–software co-design to improve the system performances implementing a system-on-chip with a field programmable gate array (FPGA). The use of a configurable architecture accelerates greatly the design and validation of a proof of real-time and system-on-chip concept.

Competing interests
The authors declare that they have no competing interests.

Author details
[1] Univ Paris-Sud, CNRS, Institut d'Electronique Fondamentale, F-91405 Orsay, France. [2] IFSTTAR, IM, LIVIC, F-78000 Versailles, France.

Received: 24 November 2011 Accepted: 16 June 2012
Published: 18 July 2012

References

1. M Dissanayake, P Newman, S Clark, H Durrant-Whyte, M Csorba, A solution to the simultaneous localization and map building (SLAM) problem. IEEE Trans. Robot. Autom. **17**, pp. 229–241 (2001)
2. A Davison, I Reid, N Molton, O Stasse, MonoSLAM: real-time single camera SLAM. IEEE Trans. Pattern Anal. Mach. Intell. **29**, pp. 1052–1067 (2007)
3. M Montemerlo, S Thrun, D Koller, B Wegbreit, in *National Conference on Artificial Intelligence,* FastSLAM: a factored solution to the simultaneous localization and mapping problem. Orlando, Florida, USA, 2002, pp. 593–598
4. J Folkesson, HI Christensen, in *IEEE International Conference on Robotics and Automation,* Graphical SLAM-a self-correcting map. LA, New Orleans, USA, 2004, pp. 383–390
5. A Eliazar, R Parr, in *International Joint Conference on Artificial Intelligence.* DP-SLAM: fast, robust simultaneous localization and mapping without predetermined landmarks. vol. 18. Acapulco, Mexico, 2003, pp. 1135–1142
6. S Thrun, Probabilistic robotics. Assoc. Comput. Mach. **45**(3), pp. 52–57 (2002)
7. C Brenneke, O Wulf, B Wagner, in *IEEE/RSJ International Conference on Intelligent Robots and Systems,* Using 3d laser range data for slam in outdoor environments. Las Vegas, Nevada, USA, 2003, pp. 188–193
8. A Prusak, O Melnychuk, H Roth, I Schiller, Pose estimation and map building with a time-of-flight-camera for robot navigation. Int. J. Intell. Syst. Technol. Appl. **5**(3), pp. 355–364 (2008)
9. F Abrate, B Bona, M Indri, in *European Conference on Mobile Robots,* Experimental EKF-based SLAM for mini-rovers with IR sensors only. Freiburg, Germany, 2007
10. T Yap, C Shelton, in *IEEE International Conference on Robotics and Automation,* SLAM in large indoor environments with low-cost, noisy, and sparse sonars. Kobe, Japan, 2009, pp. 1395–1401

11. C Gifford, R Webb, J Bley, D Leung, M Calnon, J Makarewicz, B Banz, A Agah, in *IEEE International Conference on Technologies for Practical Robot Applications,* Low-cost multi-robot exploration and mapping. Woburn, Massachusetts, USA, 2008, pp. 74–79
12. S Magnenat, V Longchamp, M Bonani, P Rétornaz, P Germano, H Bleuler, F Mondada, in *IEEE International Conference on Robotics and Automation,* Affordable SLAM through the co-design of hardware and methodology, Anchorage, Alaska, 2010, pp. 5395–5401
13. M Montemerlo, S Thrun, D Koller, B Wegbreit, in *International Joint Conference on Artificial Intelligence,* FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. Acapulco, Mexico, 2003, pp. 1151–1156
14. S Rezaei, J Guivant, E Nebot, in *IEEE/RSJ International Conference on Intelligent Robots and Systems,* Car-like robot path following in large unstructured environments. Las Vegas, Nevada, USA, 2003, pp. 2468–2473
15. C Schröter, H Böhme, H Gross, in *European Conference on Mobile Robots,* Memory-efficient gridmaps in Rao-Blackwellized particle filters for SLAM using sonar range sensors. Freiburg, Germany, 2007, pp. 138–143
16. K Konolige, J Augenbraun, N Donaldson, C Fiebig, P Shah, in *IEEE International Conference on Robotics and Automation,* A low-cost laser distance sensor. Pasadena, California, USA, 2008, pp. 3002–3008
17. P Pirjanian, N Karlsson, L Goncalves, E Di Bernardo, Low-cost visual localization and mapping for consumer robotics. Indust. Robot. **30**(2), pp. 139–144 (2003)
18. E Seignez, M Kieffer, A Lambert, E Walter, T Maurin, Real-time bounded-error state estimation for vehicle tracking. IEEE Int. J. Robot. Res. **28**, pp. 34–48 (2009)
19. R Siegwart, I Nourbakhsh, *Introduction to Autonomous Mobile Robots* (The MIT Press, London, 2004)
20. B Vincke, A Elouardi, A Lambert, in *IEEE/SICE International Symposium on System Integration,* Design and evaluation of an embedded system based SLAM applications. Sendai, Japan, 2010, pp. 224–229
21. E Rosten, R Porter, T Drummond, Faster and better: a machine learning approach to corner detection. IEEE Trans. Pattern Anal. Mach. Intell. **32**, pp. 105–119 (2009)
22. A Davison, in *IEEE International Conference on Computer Vision,* Real-time simultaneous localisation and mapping with a single camera. Nice, France, 2003, pp. 1403–1410
23. R Munguia, A Grau, in *European Conference on Mobile Robots,* Freiburg, Germany, 2007, pp. 1–6
24. E Seignez, A Lambert, T Maurin, in *IEEE International Conference On Information And Communication Technologies: From Theory To Application,* An experimental platform for testing localization algorithms. Damascus, Syria, 2006, pp. 748–753
25. A Elouardi, S Bouaziz, A Dupret, L Lacassagne, JO Klein, R Reynaud, in *International Journal on Computer Science and Applications,* A smart architecture for low-level image computing. 2008, pp. 1–19
26. F Auat Cheein, R Carelli, Analysis of different feature selection criteria based on a covariance convergence perspective for a SLAM algorithm. Sensors. **11**, pp. 62–89 (2010)
27. M Kraft, A Schmidt, A Kasinski, in *International Conference on Computer Vision Theory and Applications,* High-speed image feature detection using FPGA implementation of fast algorithm. Funchal, Madeira, Portugal, 2008, pp. 174–179
28. Eigen, (2012). http://eigen.tuxfamily.org/
29. K Gunnam, D Hughes, J Junkins, N Kehtarnavaz, A vision-based DSP embedded navigation sensor. IEEE Sens. J. **2**(5), pp. 428–442 (2002)
30. P Piniés, J Tardós, Large-scale slam building conditionally independent local maps: application to monocular vision. IEEE Trans. Robot. **24**(5), pp. 1094–1106 (2008)

doi:10.1186/1687-3963-2012-5
Cite this article as: Vincke *et al.*: **Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems.** *EURASIP Journal on Embedded Systems* 2012 **2012**:5.