Preprint for Springer Journal of Real-Time Image Processing
The final publication is available at link.springer.com

Roelof Berg · Lars König · Jan Rühaak · Ralph Lausen · Bernd Fischer †

Highly Efficient Image Registration for Embedded Systems Using a Distributed Multicore DSP Architecture

Preprint of August 28th 2014

Abstract We present a complete approach to highly efficient image registration for embedded systems, covering all steps from theory to practice. An optimization-based image registration algorithm using a least-squares data term is implemented on an embedded distributed multicore digital signal processor (DSP) architecture. All relevant parts are optimized, ranging from mathematics, algorithmics, and data transfer to hardware architecture and electronic components.

The optimization for the rigid alignment of two-dimensional images is performed in a multilevel Gauss-Newton minimization framework. We propose a reformulation of the necessary derivative computations, which eliminates all sparse matrix operations and allows for parallel, memory-efficient computation. The pixelwise parallellism forms an ideal starting point for our implementation on a multicore, multichip DSP architecture.

The reduction of data transfer to the particular DSP chips is key for an efficient calculation. By determining worst cases for the subimages needed on each DSP, we can substantially reduce data transfer and memory requirements. This is accompanied by a sophisticated padding mechanism that eliminates pipeline hazards and speeds up the generation of the multilevel pyramid.

Finally, we present a reference hardware architecture consisting of four TI C6678 DSPs with eight cores each. We show that it is possible to register high-resolution images within milliseconds on an embedded device. In our example, we register two images with 4096x4096 pixels within 93 ms while offloading the CPU by a factor of 20 and requiring 3.12 times less electrical energy.

R. Berg
Berg Solutions, Lübeck, Germany
E-mail: rberg@berg-solutions.de

L. König · J. Rühaak · B. Fischer
Fraunhofer MEVIS, Lübeck, Germany
E-mail: lars.koenig@mevis.fraunhofer.de

J. Rühaak
E-mail: jan.ruehaak@mevis.fraunhofer.de

Prof. Dr. R. Lausen
DHBW Karlsruhe, Karlsruhe, Germany
E-mail: lausen@dhbw-karlsruhe.de

Prof. Dr. B. Fischer †

1 Introduction

A classical general definition of image registration is given in [9]: “Image registration is the process of aligning two or more images of the same scene taken at different times, from different viewpoints and/or by different sensors.” Image registration is applied wherever information about the correspondence between several images is necessary. There are many examples of its application in different industries; in medical imaging, patient data acquired by different sensors, such as PET, CT, or ultrasonic devices, and possibly at different times can be combined in an enhanced overall image [2, 27]. In industrial applications, registration can help detect wrongly placed or incorrect assembly parts [32]. Furthermore, approaches to superresolution imaging rely on information about image correspondence [17].

Figure 1 shows a simple example of an image registration. When the two images of an apple need to be compared visually, a simple overlay of both images like in Figure 1(c) does not reveal the differences between them very clearly, because both apples are displaced from each other. Image registration can be used to find a transformation that removes the displacement of the template apple from the reference apple. By using this information, an overlay can be created (as in Figure 1(d)) that allows for a visual comparison of both apples for the human eye as well as for machine-vision algorithms.

The applications of image registration algorithms include a broad range of use cases in industrial and embedded setups [32, 33, 24, 15]. Devices in these applications are often restricted in space and power consumption, yet performance requirements are often very high (e.g., for driving assistance systems [10]) and demand specialized hardware and software development. Only by optimizing

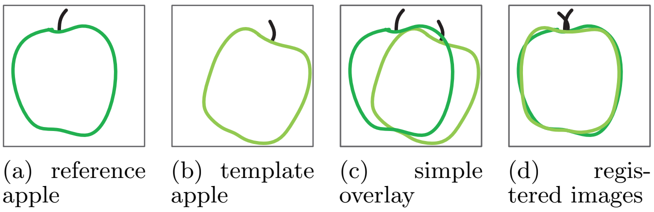

(a) reference apple (b) template apple (c) simple overlay (d) registered images

Fig. 1 Image registration explained: a simple overlay image of two similar apples (c) does not reveal the differences between the two fruits. By image registration the displacement between the two apples can be removed (d) and the differences between each other become clearly visible.

all areas involved can these high requirements be met and efficient solutions be developed.

In this paper, we present such a holistic approach to highly efficient image registration for embedded systems by describing the complete pipeline from theory to practice. All parts are optimized, from mathematics, algorithmics, and data transfer to hardware architecture and electronic components. We chose an established image registration algorithm based on a least-squares image intensity constancy assumption that performs a rigid registration of two images using derivative-based optimization. This algorithm was implemented on a distributed multicore digital signal processor (DSP) architecture, following the recent development of multicore DSPs [19]. Whereas the presented approach is tailored to two-dimensional images, the results can be directly extended to three-dimensional volumes and also higher-order distance measures [35, 21, 46].

The derivative calculations of the image registration objective function, which are needed for the employed Gauss-Newton optimization scheme, are reformulated to allow an efficient and full per-pixel parallel computation. This serves as an ideal base to utilize the computational power of the multicore DSP architecture. Additionally, an efficient way will be shown to organize and distribute the control and data streams in the proposed distributed computing environment. While significantly speeding up the overall calculation, this setup also enables offloading computations to the DSPs, leaving the CPU available for other tasks.

The contribution of this paper is a complete approach showing how optimization in all areas can result in a system capable of high-speed image registration, which is exemplified by using a high-performance, low-power embedded implementation. The optimized areas include:

- elimination of unnecessary sparse matrix operations (Chapter 3)
- design of efficient algorithm parallelization on two levels of parallelization: multichip and multicore (Sections 3.3 and 4.1)
- reduction of the amount of relevant image data by considering worst-case displacements (Section 4.2)
- elimination of processor pipeline hazards using image padding (Section 4.3)
- combination of padding and multi-resolution image generation (multilevel pyramid; Section 4.4)
- reference hardware architecture design tailored to the algorithm by using a cluster of multicore DSP coprocessors that calculate independently on dedicated RAM (Chapter 5).

These optimization techniques are versatile and not restricted to the presented registration approach. The optimizations are outlined for a specific amount of DSP coprocessors on one particular bus system, but it is straightforward to transfer the concept either to a system with a different number of DSPs or another bus system.

2 Related work

Image registration is generally well studied, and many different approaches have been developed on this subject [50, 3]. Many established state-of-the-art approaches [29, 28] use derivative-based numerical optimization. Among the variety of publications in this field, many deal with mathematical methods, but only a few papers deal with technical aspects focused on image registration on resource-limited embedded computer devices.

However, to the best of our knowledge, there are no approaches that use image registration with derivative-based optimization on embedded systems. The higher memory consumption and increased computational cost associated with the derivative computations [29] might cause this. Related research has implemented image registration on FPGA-based [38, 5], DSP-based [49], and SoC-based [7] hardware setups, but these approaches use derivative-free optimization schemes that exhibit inferior mathematical properties [31]. Our paper complements this research on image registration on embedded platforms by showing that even image registration using derivative-based optimization can be implemented efficiently on current embedded systems.

To this end, the approach from [35] is extended by adapting the computations to state-of-the-art embedded processor technology. The benefits of this new approach, which originally were not published with a focus on embedded systems, are ideal for an embedded environment: the approach results in the elimination of sparse matrix calculations, minimizing CPU operations, and reduces the amount of necessary RAM as well as the utilization of the memory bus and processor caches. Moreover, the approach is fully parallel to a per-pixel extent, which helps parallelize the algorithm for distributed computing on independent calculation cores. It will be shown that the speed of image registration on embedded systems can be raised to an extent far beyond those in [49], for example.

3 Mathematical framework

The registration methodology used in this work is built upon a sound mathematical foundation. Following the ideas presented in [29], the registration task is formulated as a continuous, yet finite-dimensional optimization problem. Computing derivatives of the objective function associated with the registration task allow efficient Newton-type minimization methods. Also, the presented approach is based on the methods provided in [29]. These, however, were designed for a powerful PC environment and are therefore too memory-consuming and demand too much computational resources for an embedded setup.

This section shows how the internal structure of the proposed registration method can be exploited to drastically reduce computational costs, thus enabling the usage of sophisticated image registration methods on an embedded device.

Section 3.1 introduces the required mathematical framework to formulate the registration task. Section 3.2 is concerned with the objective function to be minimized for registration and the computation of its derivatives. In Section 3.3, the novel problem-specific formulation of the mathematical model and its computational benefits are presented.

3.1 Registration scheme

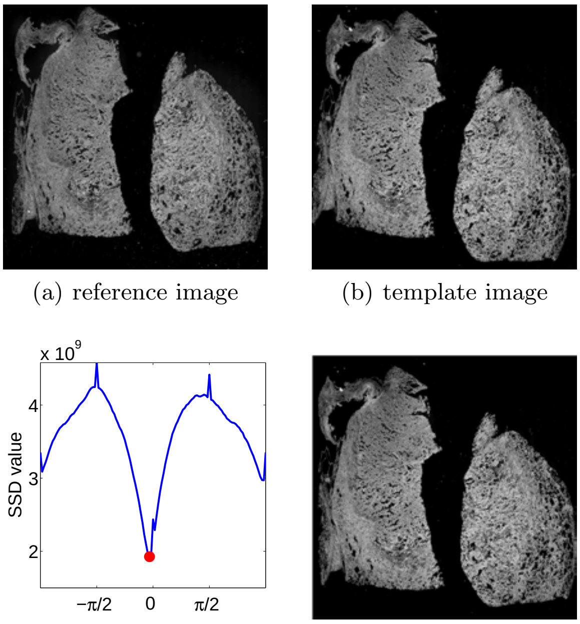
The goal of image registration is to establish a correspondence between two images [28], a fixed reference image $\mathcal{R}$, and a deformable template image $\mathcal{T}$. The images are modeled as continuous functions $\mathcal{R} : \mathbb{R}^2 \to \mathbb{R}$ and $\mathcal{T} : \mathbb{R}^2 \to \mathbb{R}$ with compact support in domains $\Omega_\mathcal{R} \subseteq \mathbb{R}^2$ and $\Omega_\mathcal{T} \subseteq \mathbb{R}^2$, respectively. The correspondence is expressed by an a-priori unknown function $\varphi : \Omega_\mathcal{R} \to \mathbb{R}^2$, often called *transformation*, between $\mathcal{R}$ and $\mathcal{T}$. In a variational setting, the function φ is characterized as a minimizer of an optimization problem.

The key idea is to describe a correspondence by the notion of image distance [29]. Similar images, such as those showing similar structures at the same positions, are assumed to exhibit a high correspondence. The notion of image similarity is formalized by a distance measure depending on the images $\mathcal{T}, \mathcal{R}$ and the transformation φ. Various choices focusing on different image characteristics have been proposed, such as *Mutual Information* [48, 25], *Normalized Gradient Fields* [14], or *Normalized Cross Correlation* [12] for images acquired from different devices or the classical *Sum of Squared Differences* [3, 4].

The Sum of Squared Differences is a simple, yet powerful distance measure. It yields good results, especially in mono-modal registration settings with comparable intensities [11, 18]. Moreover, the Sum of Squared Differences can be implemented in a very fast and efficient way, as will be shown later. In our approach, a discretized version of the continuous functional

$$\mathcal{D}_{\mathrm{SSD}}(w) = \frac{1}{2} \int_{\Omega_\mathcal{R}} \left(\mathcal{T}(\varphi_w(\mathbf{x})) - \mathcal{R}(\mathbf{x})\right)^2 \mathrm{d}\mathbf{x} \tag{1}$$

is derived and then implemented on the embedded target.



(a) reference image

(b) template image

(c) SSD versus rotation angle

(d) deformed template image

Fig. 2 Distance measure computation of reference and rotated template. A detailed description of the image data can be found in Section 6.3. The template is rotated around the image center and the respective SSD value is computed for each step. The deformed template image of the optimal rotation angle marked as a red dot in (c) is shown in (d). The minimum value is obtained at a rotation angle of $-5.6°$.

The transformation $\varphi_w : \Omega_\mathcal{R} \to \mathbb{R}^2$ maps the reference image domain to the template domain depending on the transformation parameters w. It then allows comparison of the fixed reference image $\mathcal{R}$ and the deformed template $\mathcal{T}(\varphi_w) := \mathcal{T} \circ \varphi_w$. The distance measure $\mathcal{D}(\mathcal{T}(\varphi_w), \mathcal{R})$ depends on the fixed $\mathcal{R}$ and the transformed template $\mathcal{T}(\varphi_w)$. Thus, to find a plausible alignment of the images,

$$\mathcal{D}(\mathcal{T}(\varphi_w), \mathcal{R}) =: \mathcal{D}(w) \xrightarrow{w} \min, \tag{2}$$

i.e., $\mathcal{D}$ has to be minimized by varying the transformation parameters. Here, φ_w with $w = (\alpha, t_1, t_2)$ allows for rigid (i.e., rotation, translation) transformations that map a single point $\mathbf{x} = (x_1, x_2)^\top \in \mathbb{R}^2$ with $\varphi_w : \mathbf{x} \mapsto A\mathbf{x} + t, t = (t_1, t_2)$, and $A := A(\alpha)$ is a two-dimensional rotation matrix

$$A(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix},$$

where α denotes the rotation angle and t describes the translation in x and y directions. The value of $\mathcal{D}_{\mathrm{SSD}}$ for different rotations is illustrated in Figure 2. Note that this framework has three degrees of freedom, which are expressed by the parameters w. By choosing an arbitrary transformation matrix A, this transformation model can easily be extended to general affine-linear transformations, additionally allowing for scaling and shearing.

3.2 Function value and derivatives

To compute a solution to the registration problem (2), the continuous formulation is first discretized. Hence, the use of well-established methods from numerical optimization is made possible. The domain $\Omega_{\mathcal{R}}$ is discretized into $M \times N$ equisized cells along x and y direction, with the center points $\mathbf{x}_i,\ i = 1, \dots, MN$. Using the midpoint quadrature rule, a discretized version D of the distance measure (1) can then be written as

$$D_{\mathrm{SSD}}(w) = \frac{\bar{h}}{2} \sum_{i=1}^{MN} \left(\mathcal{T}(\varphi_w(\mathbf{x}_i)) - \mathcal{R}(\mathbf{x}_i)\right)^2 , \tag{3}$$

where $\bar{h} = h_x h_y$ is the area of each cell.

The template image is typically only known at a discrete set of data points, but the transformed coordinates $\varphi_w(\mathbf{x}_i)$ will generally not coincide with these points. To compute $\mathcal{T}(\varphi_w(\mathbf{x}_i))$, interpolation is used to evaluate the discrete template image at arbitrary coordinates. Bilinear interpolation with Dirichlet zero boundary conditions is applied.

For discrete image data $\mathcal{I} \in \mathbb{R}^{M \times N}$ given on an axis parallel grid with unit spacing and a point $\mathbf{x} \in \mathbb{R}^2$, let $(p_1, p_2), (q_1, p_2), (q_1, q_2), (p_1, q_2)$ denote the coordinates of the four adjacent data points. To interpolate data $\mathcal{I}$ at coordinates $\mathbf{x} = (x_1, x_2)$, the bilinear interpolation function $\mathcal{T}(I, \mathbf{x}) =: \mathcal{T}(\mathbf{x})$ is given by

$$\begin{aligned}\mathcal{T}(\mathbf{x}) = (q_2 - x_2)\left((q_1 - x_1)\mathcal{I}_{p_1,p_2} + (x_1 - p_1)\mathcal{I}_{q_1,p_2}\right) \\ + (x_2 - p_2)\left((q_1 - x_1)\mathcal{I}_{q_1,q_2} + (x_1 - p_1)\mathcal{I}_{p_1,q_2}\right).\end{aligned} \tag{4}$$

As the goal is a fast convergence of the optimization scheme, derivative-based optimization techniques featuring super-linear convergence [31] are used. This implies that the computation of distance-measure derivatives is also needed, thus being more involved both from a mathematical and a software engineering point of view.

In our setting, the problem of image registration is phrased as the minimization of a function $D_{\mathrm{SSD}} : \mathbb{R}^3 \to \mathbb{R}$, i.e., a mapping from the parameter space (rotation, translation) to one real-valued number representing the image distance. To this end, the function D_{SSD} is decomposed into a concatenation of vector-valued functions involving all MN discrete sampling points at once. These are defined as

$$y : \mathbb{R}^3 \to \mathbb{R}^{2MN},\ w \mapsto \begin{pmatrix} (A\mathbf{x}_1 + t)_1 \\ (A\mathbf{x}_1 + t)_2 \\ \vdots \\ (A\mathbf{x}_{MN} + t)_1 \\ (A\mathbf{x}_{MN} + t)_2 \end{pmatrix} \in \mathbb{R}^{2MN},$$

with A, b as defined in Section 3.1, i.e., y maps the three parameters w to a vector of MN deformed sampling points. Additionally, using $\mathbf{y} = (y_1, y_2)^\top$, the definition

$$T : \mathbb{R}^{2MN} \to \mathbb{R}^{MN},\ \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{MN} \end{pmatrix} \mapsto \begin{pmatrix} \mathcal{T}(\mathbf{y}_1) \\ \vdots \\ \mathcal{T}(\mathbf{y}_{MN}) \end{pmatrix}$$

applies, which evaluates the image at all of the MN deformed points, thus creating a vector of the deformed template image intensities. Setting $R_i := \mathcal{R}(\mathbf{x}_i)$, we continue by formulating

$$r : \mathbb{R}^{MN} \to \mathbb{R}^{MN},\ \begin{pmatrix} T_1 \\ \vdots \\ T_{MN} \end{pmatrix} \mapsto \begin{pmatrix} T_1 - R_1 \\ \vdots \\ T_{MN} - R_{MN} \end{pmatrix}$$

as a vector-valued residual function, and finally

$$\psi : \mathbb{R}^{MN} \to \mathbb{R},\ \begin{pmatrix} r_1 \\ \vdots \\ r_{MN} \end{pmatrix} \mapsto \frac{\bar{h}}{2} \sum_{i=1}^{MN} r_i^2$$

as the sum of all squared residual elements. Hence, the discrete objective function can be written as a concatenation of the functions

$$D_{\mathrm{SSD}} : \mathbb{R}^3 \xrightarrow{y} \mathbb{R}^{2MN} \xrightarrow{T} \mathbb{R}^{MN} \xrightarrow{r} \mathbb{R}^{MN} \xrightarrow{\psi} \mathbb{R}.$$

The above formulation allows straightforward computation of the *analytical* gradient using the chain rule as

$$\begin{aligned}\nabla D_{\mathrm{SSD}}(w) = &\frac{\partial \psi}{\partial r}[r(T(y(w)))] \cdot \frac{\partial r}{\partial T}[T(y(w))] \\ &\cdot \frac{\partial T}{\partial y}[y(w)] \cdot \frac{\partial y}{\partial w}[w].\end{aligned} \tag{5}$$

Having derived the analytical structure of the derivatives on a coarse component-wise level, we continue by describing the derivatives of all individual components. The first two individual derivatives are given by

$$\frac{\partial \psi}{\partial r}[r] = \bar{h}(r_1, \dots, r_{MN}) \text{ and}$$
$$\frac{\partial r}{\partial T}[T] = I_{MN},$$

with $I_{MN} \in \mathbb{R}^{MN \times MN}$ as the identity matrix, i.e., the derivative of the residual function r can be omitted from the actual computations. For higher-order distance measures such as the Normalized Gradient Fields, the residual derivative turns out to be significantly more complex, see e.g. [35] for an extensive analysis.

Following (4) and using the notation ∂_i for the partial derivative with respect to the i-th component, the derivative of the template image at a point $\mathbf{x} = (x_1, x_2)$ is given by

$$\begin{aligned}\partial_1 \mathcal{T}(\mathbf{x}) = &-\left((q_2 - x_2)\mathcal{I}_{p_1,p_2} + (x_2 - p_2)\mathcal{I}_{q_1,p_2}\right)\\ &+\left((q_2 - x_2)\mathcal{I}_{q_1,q_2} + (x_2 - p_1)\mathcal{I}_{p_1,q_2}\right) \text{ and}\\ \partial_2 \mathcal{T}(\mathbf{x}) = &-\left((q_1 - x_1)\mathcal{I}_{p_1,p_2} + (x_1 - p_1)\mathcal{I}_{q_1,p_2}\right)\\ &+\left((q_1 - x_1)\mathcal{I}_{q_1,q_2} + (x_1 - p_1)\mathcal{I}_{p_1,q_2}\right).\end{aligned}$$

For the vector-valued evaluation function T, the Jacobian is therefore given by

$$\frac{\partial T}{\partial y} = \begin{pmatrix} \partial_1 \mathcal{T}(\mathbf{y}_1)\ \partial_2 \mathcal{T}(\mathbf{y}_1) & & \\ & \ddots \quad \ddots & \\ & & \partial_1 \mathcal{T}(\mathbf{y}_{MN})\ \partial_2 \mathcal{T}(\mathbf{y}_{MN}) \end{pmatrix},$$

see also Figure 3. Finally, the derivative of the function y, which maps the rigid parameters to a transformed grid, can be written as

$$\frac{\partial y}{\partial w} = \begin{pmatrix} -\sin(\alpha)(\mathbf{x}_1)_1 - \cos(\alpha)(\mathbf{x}_1)_2 & 1 & 0 \\ \cos(\alpha)(\mathbf{x}_1)_1 - \sin(\alpha)(\mathbf{x}_1)_2 & 0 & 1 \\ \vdots & \vdots & \vdots \\ -\sin(\alpha)(\mathbf{x}_{MN})_1 - \cos(\alpha)(\mathbf{x}_{MN})_2 & 1 & 0 \\ \cos(\alpha)(\mathbf{x}_{MN})_1 - \sin(\alpha)(\mathbf{x}_{MN})_2 & 0 & 1 \end{pmatrix},$$

thus completing the analysis of the gradient components from (5).

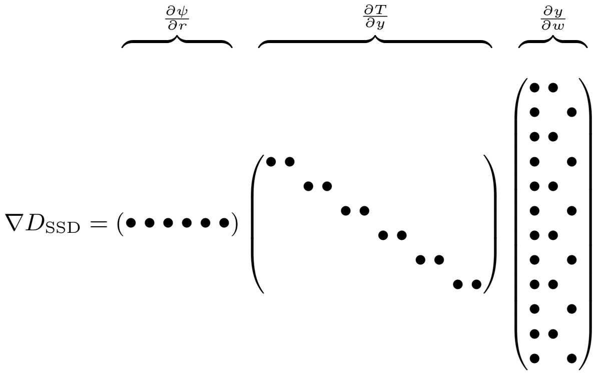
A quasi-Newton method is chosen as a tradeoff between fast convergence and fast calculation for the optimization scheme. With this method, the exact Hessian is replaced by a quadratic approximation. Because of the least-squares structure of the distance measure D_{SSD} the so-called Gauss-Newton approximation [31, 6] was chosen to avoid calculating second-order image derivatives, which are highly sensitive to noise. This scheme has been used in image registration frameworks with great success [29, 47, 11]. Defining $dr := \frac{\partial r}{\partial T}[T(y(w))] \cdot \frac{\partial T}{\partial y}[y(w)] \cdot \frac{\partial y}{\partial w}[w]$, the Gauss-Newton approximation $H(w)$ of the Hessian is given by

$$H(w) := \bar{h}\ dr^\top dr \approx \nabla^2 D_{\text{SSD}}(w). \tag{6}$$

To obtain a descent direction s_k in each step k with these components, the equation

$$H(w_k)s_k = -\nabla D_{\text{SSD}}(w_k)^\top \tag{7}$$

is solved, and the current iterate w_k is updated to $w_{k+1} = w_k + \tau s_k$. To ensure a sufficient decrease of the distance measure, the Armijo line search method [31] is used to determine the parameter τ.



$\frac{\partial \psi}{\partial r}$
$\frac{\partial T}{\partial y}$
$\frac{\partial y}{\partial w}$
$\nabla D_{\text{SSD}} =$

Fig. 3 Schematic view of the sparse matrix structure in the computation of ∇D_{SSD}.

3.3 Problem specific derivative calculation

Implementing the straightforward matrix-based approach on a computer or an embedded system is complicated, especially when focusing on fast execution time. The most critical resource is the main memory of the target platform. Both the amount of available memory and the bus bandwidth are bottlenecks which must be considered. The aforementioned methodology using matrix calculations to obtain the gradient and the Hessian would consume a lot of memory and bandwidth, even if efficient algorithms for sparse matrix arithmetic were used.

Fortunately, the sparse structure of the matrices discussed in Section 3.2 (see Figure 3) can be exploited to drastically reduce computational costs and derive a very efficient, problem-specific algorithm. Instead of assembling the particular derivative components as single matrices, as, for example, in [29, 34], we derive a closed-form formula only involving the very few computations required.

With the information about the sparse matrix structure and the corresponding values, the complete analytical gradient

$$\nabla D_{\text{SSD}} = \left(\partial_\alpha D_{\text{SSD}}, \partial_{t_x} D_{\text{SSD}}, \partial_{t_y} D_{\text{SSD}}\right)$$

of the objective function associated with the registration task can now be formulated. Using (5) and setting $\mathcal{T}_w(\mathbf{x}_i) := \mathcal{T}(\varphi_w(\mathbf{x}_i))$ and

$$\begin{aligned}c_i := &\ \partial_1 \mathcal{T}_w(\mathbf{x}_i)(-\sin(\alpha)(\mathbf{x}_i)_1 - \cos(\alpha)(\mathbf{x}_i)_2)\\ &+ \partial_2 \mathcal{T}_w(\mathbf{x}_i)(\cos(\alpha)(\mathbf{x}_i)_1 - \sin(\alpha)(\mathbf{x}_i)_2)\end{aligned}$$

yields the final result

$$\partial_\alpha D_{\mathrm{SSD}}(w) = \bar{h} \sum_{i=1}^{MN} (\mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i)) \cdot c_i \qquad (8)$$

$$\partial_{t_x} D_{\mathrm{SSD}}(w) = \bar{h} \sum_{i=1}^{MN} (\mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i))\, \partial_1 \mathcal{T}(\varphi_w(\mathbf{x}_i))$$

$$\partial_{t_y} D_{\mathrm{SSD}}(w) = \bar{h} \sum_{i=1}^{MN} (\mathcal{T}_w(\mathbf{x}_i) - \mathcal{R}(\mathbf{x}_i))\, \partial_2 \mathcal{T}(\varphi_w(\mathbf{x}_i)).$$

Using the same approach, the approximation H of the Hessian $\nabla^2 D_{\mathrm{SSD}}$ in (6) can be phrased as a sum of rank-one matrices l_i

$$H(w) = \bar{h} \sum_{i=1}^{MN} l_i, \qquad (9)$$

with

$$l_i := \begin{pmatrix} c_i^2 & c_i \partial_1 \mathcal{T}_w(\mathbf{x}_i) & c_i \partial_2 \mathcal{T}_w(\mathbf{x}_i) \\ c_i \partial_1 \mathcal{T}_w(\mathbf{x}_i) & (\partial_1 \mathcal{T}_w(\mathbf{x}_i))^2 & \partial_1 \mathcal{T}_w(\mathbf{x}_i) \partial_2 \mathcal{T}_w(\mathbf{x}_i) \\ c_i \partial_2 \mathcal{T}_w(\mathbf{x}_i) & \partial_1 \mathcal{T}_w(\mathbf{x}_i) \partial_2 \mathcal{T}_w(\mathbf{x}_i) & (\partial_2 \mathcal{T}_w(\mathbf{x}_i))^2 \end{pmatrix}$$

depending only on the i-th pixel.

In addition to its simplicity, our formulation offers a number of significant computational advantages over traditional matrix-based description. First, there is no need for storing large chunks of data for deformed templates or image derivatives, for example, as everything needed can easily be computed on the fly. No sparse matrix arithmetic is required. Second, each summand can be computed independently, as there are no dependencies between the terms. This allows direct parallelization with up to pixelwise granularity, requiring only standard reduction techniques to obtain the final result. Finally, the closed formulation leads to a very compact and efficient code.

Our matrix-free calculation scheme benefits from the fact that the residual derivative $\frac{\partial r}{\partial T}$ is the identity for the SSD distance measure. For more general distance measures, the approach is still viable, as long as the residual value at a pixel i does not depend on too many other pixels, i.e., the residual matrix exhibits a reasonable sparsity. In the case of the Normalized Gradient Fields distance measure, the same approach has been successfully applied to three-dimensional images [35] and has further been extended to non-linear deformable registration [21].

The optimized mathematical reformulation forms the backbone of our method. Only by eliminating all matrix-related overhead and all parallelization obstacles we can successfully bring the algorithm to the embedded target device. The following section presents the extensions and modifications to our method which allow us to harvest the full computational power of contemporary high-end DSP coprocessors.

4 Hardware-specific algorithm design

The matrix-free mathematical framework is an ideal starting point for designing a distributed algorithm. Therefore, the focus is on the use of an architecture of several multicore DSPs connected to a main processor as co-processors. DSPs are an attractive choice because they are both integrated circuits optimized for algorithmic calculations and designed for a high energy efficiency measured in FLOPs per Watt [36]. Furthermore, they offer special capabilities such as VLIW (very long instruction word), SIMD (single instruction multiple data), or special commands (e.g., for multiply-accumulate operations) [8, 20].

4.1 Distributed calculation

To effectuate the advantages of DSP coprocessors, it is necessary to design a parallel algorithm for distributed calculation. In particular, this applies to the computation of function value (3), Jacobian (8), and the Gauss-Newton approximation of the Hessian (9). Hence, the computations can be schematically described as

$$F(\text{image}) = \sum_{i=1}^{MN} f(\mathbf{x}_i)$$

with $\mathbf{x}_i$ being a discretization of the reference image domain $\Omega_{\mathcal{R}}$, see Section 3.2. These computations involve many floating point operations. Compared to this computational workload, the resource demand of the rest of the algorithm – the Gauss-Newton update step (7) – is negligible. Our approach utilizes the discrepancy between the low computational demand of the Gauss-Newton update step and the expensive operations for iterating over pixel data. This is done by executing the Gauss-Newton update steps on the main processor first and then offloading the computationally expensive calculation of function value, Jacobian, and Hessian to the DSP coprocessors. Furthermore, only the function-value computation is needed for the Armijo line search, whereas the Gauss-Newton update also requires the Jacobian and the Hessian approximation.

The offloaded calculations can be performed on different independent areas of the images for each DSP. The reference image in Figure 4, for example, is divided into four subareas S_i enabling a distributed calculation

$$F(\text{image}) = \sum_{i=1}^{\#\text{DSPs}} \sum_{j \in S_i} f(\mathbf{x}_j)$$

with each DSP i being responsible for its particular subarea S_i.

In this way, the central processor can control the whole calculation. In the beginning, the central processor

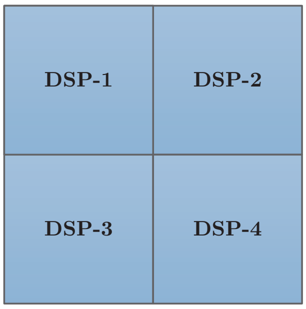


Fig. 4 Dividing the reference image into equal-sized responsibility areas for each DSP.

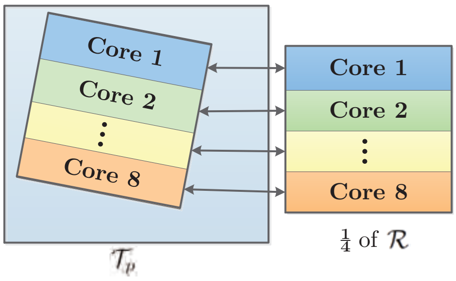


Fig. 5 Image distribution to the cores of one DSP. The subarea of $\mathcal{R}$ for which this particular DSP is responsible is shown on the right. On the left, the corresponding template image subarea $\mathcal{T}_p$ – which is a part of $\mathcal{T}$ – is depicted. Each partial image is split up into eight horizontal slices for the eight DSP cores.

transmits the image data to each DSP; afterwards it controls the iterations of the Gauss-Newton scheme. When function value, Jacobian, and/or Hessian are needed, the main processor utilizes the DSPs as coprocessors with each DSP responsible for one (and always the same) particular subarea of the reference image. Furthermore, the central processor collects the results of all DSPs and combines them to a single result for the next iteration. Because each of the n DSPs will operate on only $1/n$ of the reference image's data, a theoretical speedup of up to n can be achieved for the offloaded operations.

When using a cluster of multicore DSPs, one further step of parallelization has to be considered. Both the number of DSP chips and the amount of calculation cores inside each DSP chip contribute to the total result

$$F(\text{image}) = \sum_{i=1}^{\#\text{DSPs}} \sum_{k=1}^{\#\text{cores}} \sum_{j \in S_{i,k}} f(\mathbf{x}_j)$$

where $S_{i,k}$ is the subset for which core k of DSP i is responsible. Therefore, a method has to be defined with regard to how a particular DSP distributes the workload to its calculation cores.

As our method is based on a per-pixel independent formula, virtually any partitioning scheme can be taken. For optimal cache usage, the image data can best be used in a memory-aligned manner leading to a partitioning into several horizontal slices, as Figure 5 shows. There, an example of a DSP with eight calculation cores is given. In this case, each calculation core will calculate 1/8 of the partial result for which the DSP is responsible, which is, again, an (up to) linear speedup according to the amount of DSP cores.

The calculation result of each pixel is a scalar for the function value, a vector for the Jacobian, and a matrix for the Hessian.[1] Therefore, each DSP calculation core first sums up the per-pixel results of the particular slice it is responsible for, then the DSP sums up the results of its calculation cores, and finally, the main processor sums up the results of each DSP to obtain the final and combined result for all pixels.

4.2 Memory architecture

An important topic for a hardware-specific algorithm in embedded computing is the memory layout. A parallel memory architecture was chosen for the calculation on several DSP coprocessors, as described above. The advantage is that each DSP has its own private RAM assigned. The bus bandwidth between the memory and the DSPs therefore does not have to be shared or divided. This is important because it is seldom the processing power that is the bottleneck in modern systems but rather the memory bandwidth [26], as the processor's instruction rate is significantly higher than the memory throughput.

However, dedicated RAM has a major drawback. In a naive approach, the image data has to be stored several times, i.e., once in each DSP's independent RAM. In this case, not only the demand of total RAM space is n ($n =$ amount of DSPs) times higher, but the whole image data has to be transferred n times from the main processor's RAM to the DSP's RAM. To mitigate this shortcoming, the image data was split up in such a way that each DSP is responsible only for a smaller part of the total image data.

Concerning the reference image, the situation is simple: If e.g. four DSPs are used, each DSP only needs to access one quarter of the reference image (as depicted in Figure 4) because each DSP only performs 25% of the total calculation. This reduces the RAM demand for the reference image on each particular DSP by 75%.

4.2.1 Worst-case displacement parameters

For the template image, the procedure is more difficult because the relevant field of view moves and rotates during the algorithm iterations. Therefore, the involved image data is usually bigger than 25% (in case of four

[1] The Jacobian is a derivative to the transformation parameters w and should not be confused with the image gradient obtained e.g. by the Sobel operator. The same applies to the (approximated) Hessian.

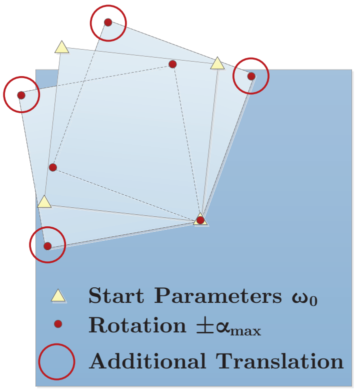


Fig. 6 Finding the dimensions of $\mathcal{T}_p$.

DSPs, see the left part of Figure 5). Where no predictions can be made with regard to the image's positions accessed on a particular DSP, the whole template image has to be stored in each DSP's RAM.

However, many embedded systems connected to a physical system have limitations that lead to an expectable range of registration results. The registration result will, for example, be limited by the size of the embedded system's image sensor. Furthermore, the space available for an object's displacement under investigation might be physically limited. The algorithm can utilize this information to reduce the amount of necessary image data for each DSP by considering a field-of-view size that is likely not to be exceeded during all iterations.

This worst case with regard to field-of-view size can be considered as the big, light blue square in the left of Figure 5 denoted as $\mathcal{T}_p$. The inner square is allowed to shift and rotate within $\mathcal{T}_p$ during the calculation iterations, but the edges of the inner square should not exceed the borders of $\mathcal{T}_p$, as no image data is defined outside $\mathcal{T}_p$. To identify the dimensions of $\mathcal{T}_p$, worst-case displacement parameters $w_{\max} = (\alpha_{\max}, t_{\max})$ are estimated, with $|\alpha| \leq \alpha_{\max}$ and $|t| \leq t_{\max}$, and α being the maximal rotation and $t_{\max}$ being the maximal length of the translation $t = (t_1, t_2)$, as defined in Chapter 3.

By identifying a $w_{\max}$ suiting the physical environment of the embedded system, the image part $\mathcal{T}_p$ of the template image can be determined that has to be transferred to a particular DSP's RAM. In order to avoid overhead by more complex polygonal shapes, we only consider axis-parallel, rectangular regions for $\mathcal{T}_p$. Fortunately, however, it is sufficient to only consider certain parameter sets instead of sampling over all possible parameter combinations. These sets are described in the following algorithm:

1. The angle $\alpha_{\max}$ is added to the start angle α_0. Then, the subarea for the particular DSP is rotated around the reference image center. Finally, all corner coordinates of the rotated subarea are identified (large red dots in Figure 6).

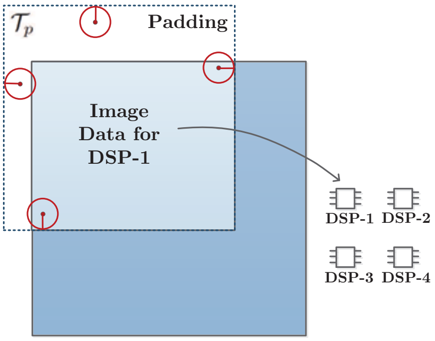


Fig. 7 Distribution of image parts to the DSPs.

2. The same operation, but with α_{max} being subtracted from α_0 (instead of being added to it).
3. If some of the absolute angles 45°, 135°, 225°, or 315° are located within the range of $[\alpha_0 - \alpha_{max}, \alpha_0 + \alpha_{max}]$, these angles are also used to identify corner coordinates, as mentioned in 1.
4. The four outermost left, right, top, and bottom coordinates are determined from the set of all coordinates identified above (dots surrounded by circles in Figure 6).
5. $t_{\max}$, the maximal length of the translation vector, is added/subtracted to these four coordinates to determine the outermost positions (circles around the dots in Figure 6, lines in the circles pointing to the outermost part of the circles in Figure 7).

This way, the outermost positions defining the boundaries of $\mathcal{T}_p$ are determined which can be reached by rotating and shifting by any admissible value. Figure 7 shows an example of a partial template image $\mathcal{T}_p$ identified by this algorithm. The intersection of the two squares defines the area of the template image that has to be transferred to the RAM of DSP-1.

The area of $\mathcal{T}_p$ in the left upper corner defines the positions that might be accessed by the algorithm, but no image data is available for this region. Typically, this case is handled by assuming Dirichlet zero boundary conditions [29]. For efficient data transfer, only the intersection pointing to defined image data mentioned in the previous paragraph will be transferred. With regard to the range of undefined data, only meta information defining its size will be used during the image data transfer.

4.2.2 Clipping Armijo line search

We observed during the first step of Armijo line search, which is a part of the Gauss-Newton optimization described in Section 3.2, that w_{k+1} will easily exceed the limitations given by w_{max} – even when the final registration result will be within the bounds of w_{max}. This jeopardizes the aforementioned procedure of identifying and transferring only partial image data as calculations for the function value (3), and the Jacobian (8) would need to access more image data than is available in $\mathcal{T}_p$.

Our experiments showed that the line search parameter τ can be restricted in such a way that the obtained parameters $w_{k+1} = w_k + \tau s_k$ lie within the limits of w_{max}. This even speeds up the Armijo line search itself when function value calculations for implausible w_{k+1} are skipped. Furthermore, it can be ensured in this way that w_{k+1} will not exceed the limitations given by w_{max} during the calculation.

One limitation to this technique is, of course, that images with a bigger displacement than allowed by w_{max} will lead to an implausible registration result. Therefore, it must be ensured that w_{max} is big enough for the given physical machine setup.

4.3 Padding to prevent pipeline hazards

It is necessary for a super-efficient calculation to use the processor's ALUs (Arithmetic Logical Units) to their utmost extent. Especially during nested loops with every pixel being accessed ($\sum_x \sum_y f(x, y)$), all available ALUs must run with as little idle time as possible. A major factor for achieving this is to avoid hazards in the processor's instruction pipeline that can occur when executing a branch command.

A pipeline hazard is a phenomenon in pipelined processors operating on several subsequent commands in different stages at the same time. In a three-stage pipeline, for example, the most recent instruction will be fetched while the instruction before is decoded, and the third instruction will be executed simultaneously. When the program counter is moved to another position by a branch command, the instruction pipeline has to be rebuilt. On a DSP using VLIW technology (Very Long Instruction Word), with several instructions being encoded into one pipelined instruction word, a pipeline hazard leads to even worse effects as more instructions are affected by one single pipeline hazard.

These branch instructions that can lead to pipeline hazards would normally be implemented in the given algorithm when handling Dirichlet boundary conditions in image interpolation. This means that a constant default pixel value will be used when a coordinate points outside the known image. In a simple approach, the code that fetches pixel values from the RAM will have a conditional branch deciding on the basis of the coordinate position whether the request for an image value is inside the boundaries of defined image data or whether to use the default value. Branch executions, however, lead to pipeline hazards.

For a branch free – and therefore pipeline hazard free – operation, the image data can be surrounded by padding borders of defined image data that are big enough to guarantee that each request to fetch an image value will point to defined data. The padding borders contain a copy of the constant default pixel value being used to implement the Dirichlet boundary conditions on every coordinate. This way, the code inside the nested image interpolation loops operates without any branches, yielding a significant speedup as shown in Section 7.3.

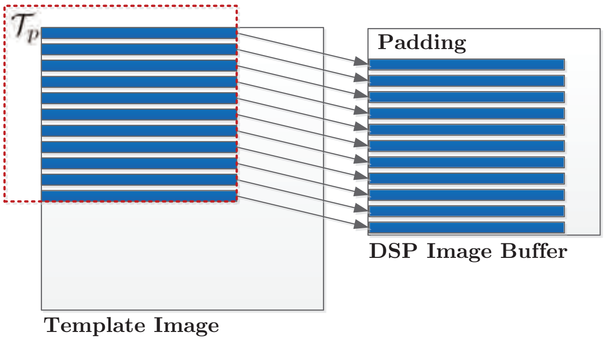


Fig. 8 Transferring $\mathcal{T}_p$ to the padded destination buffer.

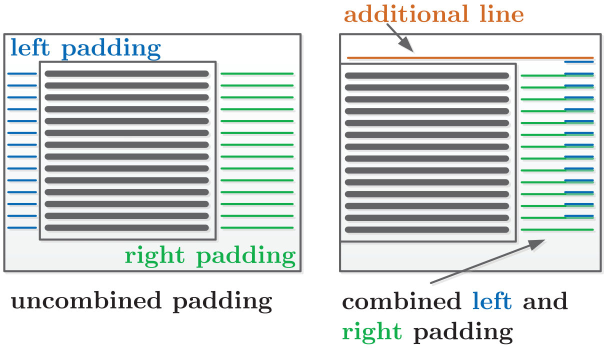


Fig. 9 Combining several padding borders.

The extents of the necessary padding borders define the amount of necessary padding bytes that are well known from the described procedure of calculating a $\mathcal{T}_p$ described in Section 4.2.1. In Figure 7, the padding areas are shown in the left upper area. It becomes clear that, when storing the image data row-wise, some bytes are needed for the padding as a trailer to the image buffer and some in between the rows of defined image data.

For a distributed calculation, the extent of the padding borders has to be transferred to the DSP system along with the relevant image data (which is an insignificant overhead as it is no additional image data but just a few size-values). With this information, the DSP system can assemble a data buffer consisting of initialized (usually zeroed) memory and then place the image data to the right places inside the bigger buffer row by row, as Figure 8 shows.

To save memory on the DSPs, the right and the left borders of one image can be combined. This is especially useful when the $w_{\max}$ parameters are so wide that padding borders appear not only at two but at all four image edges. Additionally, padding becomes necessary at all borders if only one DSP is used because in this case, the combination of positive and negative shift $t_{\max}$ (in $w_{\max}$) will always lead to padding borders around

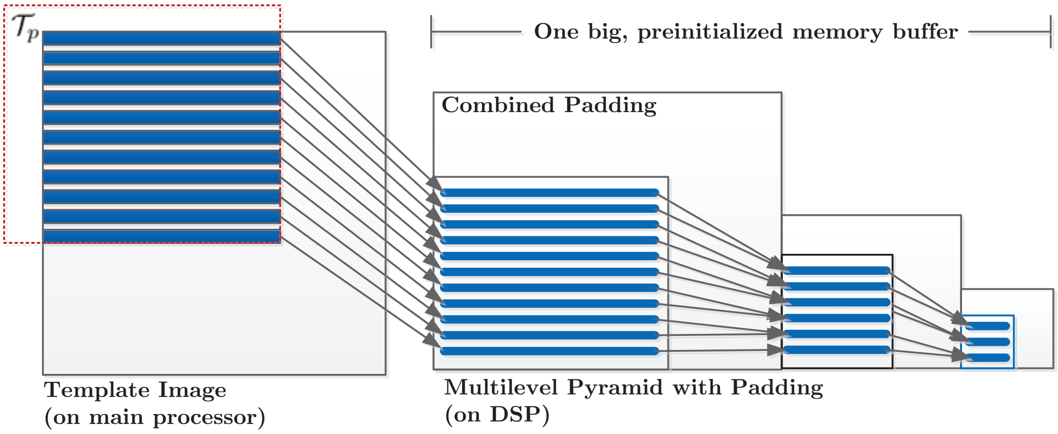


Fig. 10 Efficient multilevel pyramid generation for $\mathcal{T}_p$ with omitted padding areas.

the whole image. In Figure 9, four padding borders surround the image when the extraction of $\mathcal{T}_p$ is finished (left part of the figure), whereas an image on the DSP is surrounded by only three borders (right part of the figure). Here, the biggest of the borders is taken and used as the right border, and one additional row is added to the top border in case the utmost upper left pixel is accessed (which would otherwise fail, as there is only a right border).

4.4 Efficient multilevel pyramid generation

Our approach is based on a multilevel scheme [13], which means that the image registration starts on a coarse image at a low image resolution and then increases the resolution during the algorithm execution in certain steps up to the final full image size.

Therefore, it is necessary to access the image data in several coarser versions. As the algorithm operates down the pyramid (meaning that it accesses the coarsest image at first), the original high-resolution image has to be shrunk to the coarsest level at the beginning. As it is computationally efficient to execute the operations that shrink the image data as few times as possible, our implementation initially shrinks the image several times until the coarsest size is reached and keeps all intermediate images in the RAM. This is advantageous because the intermediate steps will be needed later when the multilevel scheme accesses image data on finer pyramid levels.

The data buffer for $\mathcal{R}_p$ and $\mathcal{T}_p$ must be big enough to hold the image data of all multilevel pyramid instances including the padding borders. On each coarser level of the pyramid, four neighboring pixels are averaged (half height and half width). Using this scheme, the total amount of memory m_{total} needed for the whole image pyramid can be calculated by using the convergence of the geometric series

$$m_{total} = \sum_{k=0}^{\infty} \left(\frac{1}{4}\right)^k \cdot m_o = \frac{1}{1-\frac{1}{4}} \cdot m_o = \frac{4}{3} \cdot m_o$$

with the memory amount of the input image (with padding)

$$m_o = (i_w + b_l + b_r) \cdot (i_h + b_t + b_b)$$

and i_w, i_h being the image width/height, b_l, b_r, b_t and b_b the padding border dimensions (left, right, top, bottom). The convergence value is used as a buffer size and will therefore contain enough space for any number of pyramid levels. In our experiments, however, the number of pyramid levels that were actually used was determined by the formula $\lceil \log_2(i_w/32)) \rceil$ because lower and higher values resulted in slower calculation (with square images, $i_w = i_h$).

The pyramid generation can be sped up to just a few milliseconds execution time – as shown in Section 7.2 – when the implementation accesses only pixels known to contain relevant image information. Therefore, our implementation preinitializes the whole image memory to the default pixel value. When executed while the system is idle (after the boot-up and after each calculation), this can be done without negative effects on the calculation speed. No time is wasted for setting the pixel values of the padding borders during the image transfer from CPU to DSP as well as during pyramid generation.

Figure 10 shows the image transfer and the pyramid generation. Here, only the areas marked with the horizontal lines are accessed row by row, as is obvious due to the facts mentioned in the previous paragraph. The other memory parts – being the padding borders introduced in Section 4.3 – remain untouched. On a multicore DSP, this operation can be parallelized by dividing the image into slices. In this case, every calculation core processes a set of subsequent lines.

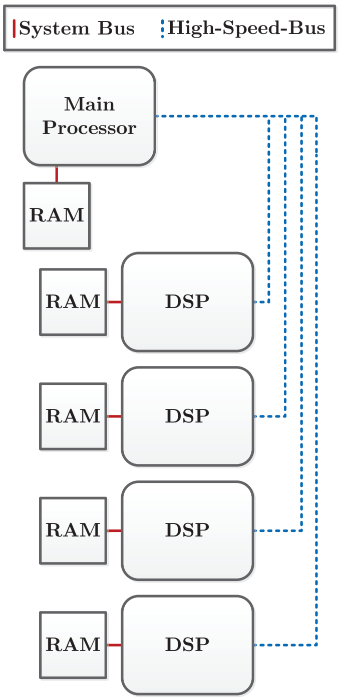


Fig. 11 Reference hardware architecture.

5 Embedded systems architecture

After the mathematical and algorithmical approach has been discussed, the implementation in an embedded system will now be described. First, an abstract overview of the reference hardware architecture is given. This is followed by an outline of the experimental hard- and software setups which will then be used for the analysis of the proposed method's consumption of computational resources.

5.1 Reference hardware architecture

A main aspect of this paper is to show how image registration using derivative-based optimization can be performed on an embedded system in a holistic approach. This means that mathematical and algorithmical problems are covered and a feasible hardware architecture will be developed. The algorithm presented above is tailored to a system consisting of a central processor (or microcontroller) and four DSP coprocessors with dedicated RAM for each DSP. As mentioned, the results of this paper can easily be extended to a different number of DSP coprocessors.

The abstract reference architecture is depicted in Figure 11. A central processor (CPU or microcontroller) is connected to four DSP coprocessors using a highspeed bus system. This could be a bus system like PCIe, SRIO, or Hyperlink. The central processor has its own system bus and its RAM is dedicatedly connected. This means that the DSP coprocessors cannot directly access the central processor's RAM, and vice versa. Furthermore, each DSP has its own system bus connected to its own dedicated RAM. The advantage of this architecture is that the RAM bandwidth does not need to be shared between several coprocessors.

Maximizing the available RAM bandwidth is crucial because the DSPs access each image pixel of $\mathcal{R}$ and its four counterparts in $\mathcal{T}$ once in every single offloaded calculation step. This applies all the more so, as the RAM for $\mathcal{T}$ is accessed in a semi-random order because of the image rotation, which can lead to cache misses when image resolutions are high.

5.2 Experimental hardware setups

To analyze the proposed methods, two experimental hardware setups were built according to the reference hardware architecture described in Section 5.1. We subdivide the mainboards of embedded systems into two basic categories:

- **PC-like, COTS embedded boards:** Commercial Off-The-Shelf (COTS) mainboards based on conventional PC technology for devices which have less space, power, and/or cost restrictions. Fields of use include POS/POI (Point of Sales / Information) terminals or IPCs (Industrial PCs) for industrial automation. These systems are often based on standard mainboards with Intel or AMD mainstream processors [22, 23].
- **Fully customized embedded boards:** systems based on a customized circuit board tailored to meet special restrictions for space, power, and/or cost. Examples are omnipresent, including cars, cellphones, or portable devices. These mainboards are often based on microcontrollers such as PowerPC or ARM, which are power efficient and, in addition to the CPU, also contain many peripherals on one single, cost-efficient, easy-to-integrate chip (e.g. [40]).

ID	**Field of Use**
Intel setup	PC-like, COTS embedded boards
ARM setup	Fully customized embedded boards

Table 1 Experimental hardware setups for two different categories of embedded system mainboards.

We built an experimental hardware setup for each of the two system types: one based on conventional PC technology and an Intel processor, which is a blueprint, for example, for POS/POI/IPC systems. The other one, reflecting a field of uses that need a custom circuit board design, is driven by an ARM processor. Both systems are described in detail below, while the identifiers 'Intel setup' and 'ARM setup' of Table 1 are used to distinguish between the two.

By using state-of-the-art electronic components that combine high effectiveness, low material cost, and wide availability for industrial mass products, the setups are suited for experimental performance measurements giving results with a high practical benefit.

5.2.1 DSP

For both the Intel and the ARM setups, the TI model C6678 was chosen as the DSP coprocessor. Being a superscalar high performance DSP running eight cores on each coprocessor with up to an 1.25 GHz clock rate, it is currently one of the fastest mainstream DSPs for industrial applications. In total, one coprocessor can reach a calculation speed of up to 160 billions of single precision floating point operations per second (FLOP). The specification of the C6678 DSP as taken from [44, 42] is summarized in Table 2.

Clock speed	1.25 GHz
Peak performance	320 GMAC / 160 GFLOP
Calculation cores	8
SIMD vector size	128 bit
Chip dimensions	24 mm × 24 mm
Power consumption	<10 W

Table 2 The Texas Instruments C6678 DSP. (GMAC: billions of multiply accumulates per second, GFLOP: billions of floating point operations per second.)

Programming these DSPs requires an individual system software layer, similar to programming a microcontroller. Our system is based on the realtime operating system SYS/BIOS [43] provided by the DSP vendor TI, which allows the most control over the DSP's hardware features. The C6678 DSP, for example, provides four cache levels with different memory speeds, and the DSP software can explicitly control the cache location of each memory block as well as the cache synchronisation.

5.3 Intel setup (PC-like, COTS embedded boards)

The Intel setup (Table 1) is based on an Intel Core i5 Ivy Bridge 3570K as the main processor. This model has four cores, fast PCIe and DDR3 connectivity [16]. The clock speed is 3.4 to 3.8 GHz while the TDP (Thermal Design Power) is 77W. Together with the DSP infrastructure power consumption of about 54W [1] and the power consumption of the PC infrastructure, this is a setup for high-performance devices with fewer constraints in power consumption and device size.

Four C6678 DSPs were connected to the Intel setup by utilizing the PCIe Gen2 adapter card DSPC-8681 from Advantech. It offers eight PCIe lanes [1] and a theoretical maximum bandwidth of $8 \cdot 5$ GBit/s = 40 GBit/s.

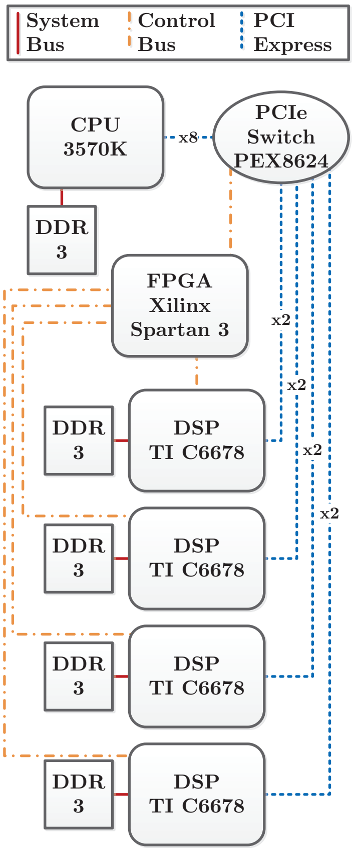


Fig. 12 Experimental hardware setup 'Intel setup' reflecting a PC-like, COTS embedded mainboard.

Figure 12 shows the chip interconnections of the Intel setup consisting of the Core i5 CPU connected to the PCIe adapter card. On the DSPC-8681, one 8x PCIe connector is multiplexed by a PCIe lane switch to four 2x PCIe connections – one for each DSP. The main processor and each DSP are equipped with dedicated DDR3 RAM. An FPGA – which is also part of the DSPC-8681 – controls the clocks and busses between the PCIe lane switch and the DSP coprocessors.

Such a device cannot be compared to a ready-to-use PCIe device for a personal computer that comes with a full set of firmware, software, and drivers. It was necessary to develop custom AMP (asymmetric multiprocessing) firmware and DMA (direct memory access) communication between the CPU and DSPs.

5.4 ARM setup (fully customized embedded boards)

Readers engaged in low-power, small-space embedded systems will be interested in the effects of the DSP coprocessors on a custom circuit board based on embedded microcontrollers such as ARM or PowerPC, represented

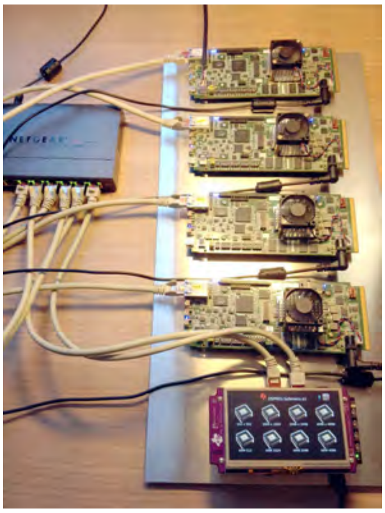


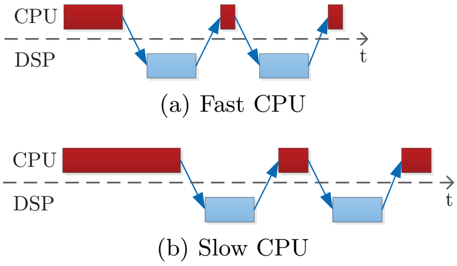
Fig. 13 Experimental hardware setup 'ARM setup' reflecting a fully customized embedded board. The experiments are based on Ethernet and an additional prediction is calculated on how a PCIe based custom circuit board would behave.

by the ARM setup. We are not aware of any off-the-shelf circuit boards that utilize an ARM microcontroller or SoC (System on Chip) being connected to four C6678 DSP coprocessors by a highspeed bus-system like PCIe, which means that a new specialized PCB (printed circuit board) and/or SoC would have to be designed for a measurement. Unfortunately, this is tremendously time and cost intensive and was out of our research scope. To present the effects of DSP-coprocessors to an ARM-based custom design nonetheless, we built the ARM setup by using readily available hardware evaluation modules with Gigabit Ethernet capability. As shown in Image 13, we chose an ARM Cortex A8 based TI Sitara evaluation board with a 720 MHz single core processor [41] and four TI C6678 evaluation boards [45].

5.4.1 Speed prediction for an ARM-PCIe based PCB

The ARM setup is based on Gigabit Ethernet, but in real PCB development, a rather fast bus system can be expected. Thus, we calculated a prediction of an ARM-PCIe based custom circuit board's speed, which may have some degree of uncertainty, but will still be useful for estimating the performance of such a setup.

Figure 14 (a) shows a simplified (fewer iterations) execution flow of image registration between a fast main processor and one PCIe-connected DSP. The time for a complete image registration shall be I_w (Intel setup wallclock time). The CPU time, referring to the sum of time spans where the CPU is not idle, shall be I_c, reflected by the dark red bars above the dotted horizontal line. The time spans painted in light blue below the horizontal line refer to the operation times of the DSP and the blue arrows reflect the time demand for data transmission. The sum of the latter two shall be D_w (distributed calculation wallclock time). Then, a single threaded and uninterrupted system follows the equation $I_w = I_c + D_w$.



(a) Fast CPU

(b) Slow CPU

Fig. 14 Difference in the execution times of an image registration on a fast (a) and a slow (b) CPU and one DSP. The execution times on the DSP (lower blue bars) and for the data transfer (blue arrows) are equal in both cases and independent from the CPU speed.

Figure 14 (b) shows the impact of a slower main processor. For the wallclock and CPU times, the letter A (like ARM) shall be chosen instead of I: A_w and A_c. The DMA-based PCIe transfers and the DSPs operate independently from the main processor's operation speed. Therefore, D_w is equal in both Figures, and there is again an equation as above: $A_w = A_c + D_w$. The basis of our prediction is now to calculate A_w by summing A_c measured on the Ethernet-based ARM setup and D_w measured on the Intel setup for which PCIe was available.

Our measurement software was written in a way that makes an A_c value obtained by Ethernet similar to an A_c value on PCIe hardware by using the same memory copy scheme to the kernel buffers. Our experiments on the Intel setup confirmed that I_c is to some degree comparable between PCIe and Ethernet. Provided that a future PCB implementation also guarantees this for A_c, the sum of A_c of the ARM setup and D_w of the Intel setup yields the predicted registration duration on an ARM-PCIe-based PCB. This approach was experimentally verified on the Intel setup, and the error between the prediction and the verification measurement was in the range of 3.1%-4.5%[2], depending on the image size.

5.5 Embedded software for the experimental setup

To assess the computational performance of our method, the same algorithm was implemented with and without DSP coprocessor usage. This allows for a measurement of the DSP coprocessors' speedup as well as measurement of the extent to which the main processor will be offloaded.

As outlined in Figure 15, two points in time have to be defined at which a time difference will be taken to measure the algorithm's execution duration. The upper part of the figure shows the operation without DSP coprocessors (from left to right). For the DSP-free implementation, the starting point in time will be the point af-

[2] E.g. 4096x4096 px on 1 DSP: Ethernet measurement 1058ms, PCIe prediction 211ms, PCIe measurement 202ms.

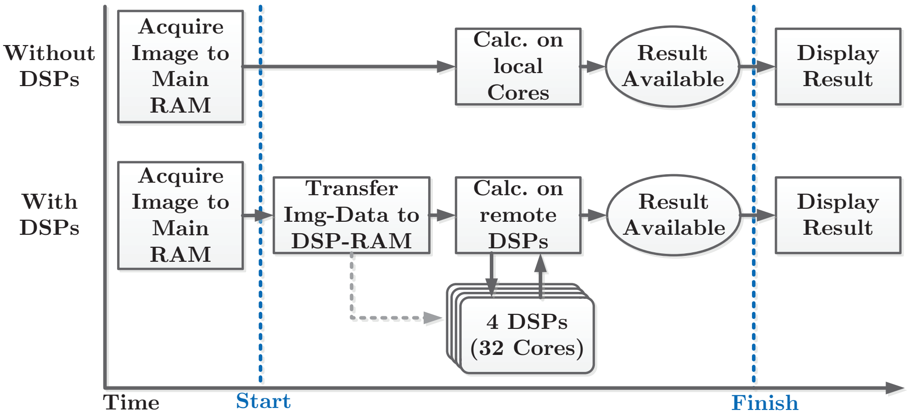


Fig. 15 Time measurement.

ter the image data has been loaded/acquired (e.g., from a sensor or the harddisk) and the Gauss-Newton optimization is about to start. The stop point in time will be the time when the Gauss-Newton optimization stops and the registration result has been computed.

The lower half of Figure 15 shows the alternative approach using DSP coprocessors. Here, the transfer for the image data to the DSPs' dedicated memory also has to be regarded as an overhead that belongs to the algorithm's execution time. Hence, the overall execution time here consists of the calculation time plus the overhead time for the data transfer between the main processor's and the DSP's RAM.

To retain the best comparability between both approaches, we spent the same effort of code optimization on both the DSP-free and the DSP-utilizing version. For example, all versions use the same computation accuracy (32 bit), compiler/linker optimizations (where possible), and full-speed processor options (e.g. Intel AVX, ARM NEON, SIMD, VLIW). The source code of the performance relevant calculation parts is almost identical between the two versions. Furthermore, the same extent of parallelization is used, i.e., a parallel multilevel pyramid generation and a parallel calculation of the function value, Jacobian, and Hessian on all available (DSP or CPU) cores.

6 Experiments

Now that all aspects of the experimental setup have been described from the mathematics and the algorithm up to the embedded hardware and software, two kinds of experiments can be defined. The first experiment is a preparative examination to determine whether the mathematical approach in Chapter 3 is feasible and beneficial. This solely mathematical comparison can be executed on a PC workstation. However, the final performance measurement examining the effect of different hardware setups was made on a realistic embedded software implementation on the experimental hardware setups described in Chapter 5.

6.1 Preparative experiments on a PC workstation

As a preparative experiment, two implementations are compared to measure the advantage of our matrix-free mathematical framework. The first implementation follows the traditional matrix-based approach from which our method is derived, as described in [29]. The second implementation makes use of the matrix-free mathematics described in Section 3.3. The execution speed of both methods is compared to illustrate the speedup of our new method.

Since we only implemented our new, matrix-free approach on the embedded hardware setup instead of the less efficient, traditional matrix-based approach, this preparative experiment was executed on a PC workstation. We measured the combined execution times of the calculation of the function value (3), the Jacobian (8), and the Gauss-Newton approximation of the Hessian (9), which are the algorithm's most expensive operations. These times were measured for both matrix-based and matrix-free calculation.

Both the matrix-based and matrix-free calculations were implemented in C++, parallelized using Open-MP [39], and optimized individually and carefully to generate an objective and a comparable measurement. For the traditional implementation, sparse matrices had to be used because the amount of RAM available on a typical PC workstation would otherwise have been exceeded. In [29], for example, the biggest Jacobian has a dimension of $mn \cdot 2mn$ (m=image width, n=image height) which would consume 512 Terabytes of memory for a 4096x4096 pixel image if not implemented as a sparse matrix.

We implemented this experiment in a minimalistic way without any of our other algorithmic improvements (for instance, the padding borders introduced in Section 4.3). It was thus possible to emphasize the sole speedup of the matrix-free calculation which might also be relevant for applications outside the field of embedded computing.

6.2 Computational experiments on embedded hardware

In addition to the preparative experiment described above, our main experiments were executed on real embedded hardware setups as introduced in Chapter 5. On each particular hardware setup, one measurement series was made by solving several image registration problems multiple times and measuring two values in each run. One value is the calculation time needed to solve a particular registration problem, the wallclock time. The other value is the utilization of the main processor during this time, the CPU time.

ID	**CPU cores**	**DSP chips**	**DSP cores**	**Bus**
Intel-0	4	0	0	-
Intel-1	4	1	8	PCIe
Intel-4	4	4	32	PCIe
Arm-0	1	0	0	-
Arm-1	1	1	8	Eth

Table 3 Overview of all embedded hardware setups for our experiments. (Intel-1 had 4 physical DSPs available and only 1 DSP was used during the calculation.)

Table 3 presents an overview of all experimental hardware setups. On the Intel setup introduced in Section 5.3 reflecting PC-like, COTS embedded boards, three series of measurements were examined. The first (Intel-0) was calculated locally on the main processor without utilizing any DSP coprocessors, the other two series were calculated with one (Intel-1) and with four (Intel-4) DSPs.

As for the the ARM setup in Section 5.4, we found that four DSPs provide no advantage over one DSP: the ARM processor is so slow that the overhead for the image distribution to four DSPs costs more CPU time than can be compensated by the faster calculation on four DSPs instead of one. This effect is independent from the transport layer (Ethernet, PCIe), as it is caused by the CPU time A_c and not by the distributed calculation time D_w. Therefore, we show only the measurements made with one DSP (Arm-1) on the ARM setup.

In addition to the calculation speed, we also measure and compare the power consumption between the Intel setup and the ARM setup measured by a phase shift compensated power meter.

6.3 Example images

The mathematical framework presented here performs an image registration without any image recognition and without any landmarks. The registration algorithm will operate solely by providing a reference and a template image. To show its effectiveness, a challenging registration problem taken from [30] as shown in Figure 16 is used. The task is the registration of two subsequent histological slices of a biological cancer tissue. To produce these images, a series of thin slices of cancer tissue is digitized one by one using a microscope. One particular slice might be displaced (by means of shift and rotation) to its predecessor because a human moves the slice from a microtom slicing device to the microscope's object plate. By using image registration several of these slices can be combined to a 3D representation of the cancer tissue when the shift and rotation between the slices are digitally readjusted [37].

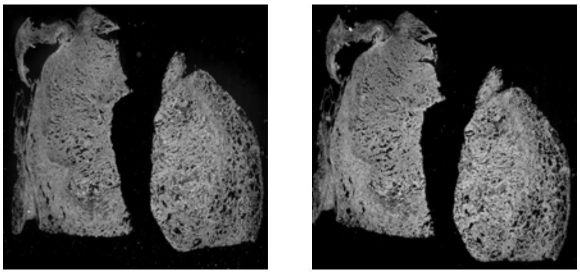

(a) reference image (b) template image

Fig. 16 Subsequent histological slices of cancer tissue used as example images for an image registration.

The difficulty of this registration problem is the arbitrary and unpredictable structure of the biological input data. The following chapter will focus on the calculation of the necessary shift and rotation to transform the template image shown in Figure 16 to the best possible overlay with the reference image.

For each experimental hardware setup listed in Table 3, a measurement series is made that covers the following image sizes:

- 512x512 pixel
- 1024x1024 pixel
- 2048x2048 pixel
- 4096x4096 pixel

7 Results and discussion

All experiments provided an adequate registration result in terms of accuracy. To demonstrate the level of accuracy achieved by our method in Figure 17, the difference images of the registration problem introduced in Figure 16 are shown. Zero difference is presented in gray, which means that the lighter or darker a pixel is, the more difference exists by comparing $\mathcal{R}$ and $\mathcal{T}$ at this coordinate. Figure 17(a) shows the differences between $\mathcal{R}$ and $\mathcal{T}$ before the image registration. In Figure 17(b), the transformed template image is compared to $\mathcal{R}$, which differs

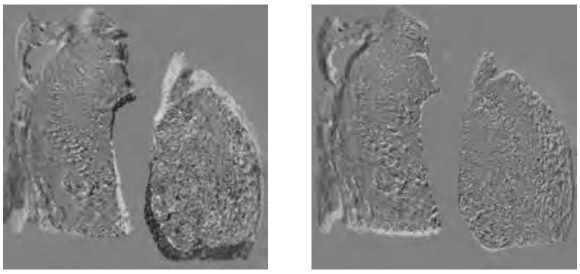

(a) before registration (b) after registration

Fig. 17 Difference images between the reference image of Figure 16 and the original (a) as well as the transformed (b) template image. Image registration eliminates the differences caused by the displacement and emphasizes the inherent inequalities.

significantly less. Only the inherent inequalities between $\mathcal{R}$ and $\mathcal{T}$ are left, while the differences related to the displacement between the images are eliminated.

Because the registration result is adequate, the execution times for the calculations are the most interesting figures, and these will be the focus of the rest of this chapter.

7.1 Speedup of the mathematical approach

Table 4 shows the speedup of our mathematical approach identified by the preparative experiment introduced in the beginning of Section 6.1. The table shows the execution time of function value, gradient, and Hessian computation together. All times are measured with the traditional matrix-based approach our method is derived from as well as with our new approach. As explained in Section 6.1, this preparative measurement took place on a PC workstation and focused solely on the speedup given by our matrix-free approach presented in Section 3.3, without considering further improvements.

Image Size	Tradition. Approach	New Approach	Speedup Factor
512x512	0.06772s	0.00461s	**14.69**
1024x1024	0.26068s	0.01534s	**16.99**
2048x2048	1.09102s	0.05921s	**18.43**
4096x4096	4.32099s	0.23972s	**18.03**

Table 4 Execution time and speedup between the traditional matrix-based and our new matrix-free approach for different image sizes calculated on a Linux workstation with Intel Core i7-2600 CPU (3.40GHz) and 16GB of RAM.

Our matrix-free approach produced a significant speedup, which means it was several times faster than the traditional sparse-matrix based equivalent. In larger images, the system overhead became less important and the speedup of the matrix-free approach increased.

This gives clear evidence of the superiority of the matrix-free approach and an embedded implementation will undoubtedly benefit from it as well. Additionally, this paper proposes several improvements to algorithms and hardware. Because the speedup of these improvements is related to the complex interaction and architecture of embedded hardware parts (e.g., processor caches), we did not simulate this on a PC workstation. We measured the benefits directly on target hardware comparable to state-of-the-art embedded systems.

7.2 Effectiveness of the memory architecture

As there are many advancements involved in our setup, the particular impact of each single advancement will not be analyzed here. For a selection of our methods, however, detailed values regarding speed and/or memory will be given before the overall performance of a complete image registration is discussed.

One important element of our approach is the generation of the multilevel pyramid as described in Section 4.4. For both images of Figure 16 ($\mathcal{R}_p$ and $\mathcal{T}_p$) on one particular DSP of the hardware setup Intel-4 we measured a time consumption ranging from 1.28 ms for a 512x512 pixel image to 25.23 ms for 4096x4096 pixels. Table 5 gives examples of the necessary image memory for $\mathcal{T}$ and $\mathcal{R}$ (including padding and multilevel pyramid) when $w_{\max}$ is assumed to allow ±10° rotation and ±10% translation.

Image Size	4 DSP	1 DSP
512x512	0.34 MB	0.87 MB
1024x1024	1.34 MB	3.48 MB
2048x2048	5.35 MB	13.90 MB
4096x4096	21.37 MB	55.55 MB

Table 5 Total memory consumption per DSP for the sum of both multilevel pyramids of $\mathcal{R}$ and $\mathcal{T}$ (for a 1 DSP and a 4 DSP setup) in MByte for different image sizes.

The column for the 4 DSP setup refers to a significantly smaller memory amount than the column for the 1 DSP setup. This is caused by reducing the image data for a particular DSP, as shown in Section 4.2. Without this technique, the 4 DSP column would be equal to the 1 DSP column. As can be seen, our method operates with an efficient memory usage, which – depending on the image size – even allows using the DSP's internal memory (e.g., 4 MB on the C6678 [44]) without needing additional DDR3 chips.

7.3 Speedup through padding

In Section 4.3, we introduced padding to prevent pipeline hazards. Although this method increases the complexity of the software and consumes memory, outstanding performance benefits can be obtained. For 4096x4096 pixels, we measured a speedup factor of 6.32 solely for the function value calculation and 4.84 for the combined calculation of the function value, the Jacobian, and the Hessian on the Intel-1 setup, where preventing pipeline hazards can be observed on a single DSP without dividing the image into parts. These measurements were taken on the finest pyramid level and again, $w_{\max}$ was assumed to allow ±10° rotation and ±10% translation.

7.4 Speed comparison between the experimental setups

The most interesting measurement is the calculation speed for solving a complete image registration problem using the images shown in Figure 16. The execution times in ms measured in the experimental setups are shown in Table 6.

ID	512px	1024px	2048px	4096px
Intel-0	7.2	22.3	49.2	215.5
Intel-1	10.7	28.2	66.4	202.0
Intel-4	12.3	22.6	33.4	92.7
Arm-0	973.3	3,194.5	17,204.4	33,549.3
Arm-1	90.2	211.5	448.6	1,323.3

Table 6 Calculation duration in milliseconds for a complete image registration (according to Figure 15) for different image sizes on different hardware. The column ID refers to the experiments listed in Table 3.

Using our approach, a full image registration can be calculated in a few milliseconds and, depending on the image size, even the speed of a video (25 frames per second, 40 ms per frame) can be reached for realtime imaging. The bigger the images are, the more advantageous becomes the use of DSPs. The last two lines (Arm-1) show a remarkable speed increase on the ARM setup. The slow ARM processor takes 33.5 seconds to register a 4096x4096 image, whereas a DSP-equipped calculation finishes in 1.3s.

The Intel-0 row shows that the method and mathematical framework lead to a registration result in the range of milliseconds, even for hardware setups without any DSP utilization. Therefore, our method is also feasible for systems that cannot be equipped with DSP coprocessors.

7.5 Speedup comparison between systems with and without DSP

A more detailed analysis of the values in Table 6 can be obtained by investigating the DSPs' speedup by calculating the quotient of the execution time on a system with and without DSP usage, which is listed in Table 7. A value of 2.33 in the row for the ID $\frac{Intel-0}{Intel-4}$, for example, indicates that the DSP utilized system Intel-4 calculates 2.33 times faster than Intel-0.

ID	512px	1024px	2048px	4096px
$\frac{Intel-0}{Intel-1}$	0.67	0.79	0.74	1.07
$\frac{Intel-0}{Intel-4}$	0.58	0.99	1.48	2.33
$\frac{Arm-0}{Arm-1}$	10.79	15.10	38.35	25.35

Table 7 Speedup factors by the use of DSP coprocessors (wallclock time).

Table cells with a value lower than 1.0 correspond to a slowdown of the DSP-based approach on systems that are solely executing the image registration and would otherwise be idle. This, however, does not mean that there is no benefit at all, as shown in Section 7.6, because all of our DSP-based approaches still consume significantly less CPU time, even if more wallclock time is used.

As expected, the highest speedup can be seen on the Arm-1 system because the ARM microcontroller has only one single 720 MHz calculation core, which makes a great difference to the multicore DSPs' speed. On the Intel-4, a wallclock speedup can only be seen at 2048 and 4096 pixel sizes with speedup of 1.48 and 2.33, respectively. On the Intel-1 system, there is no or almost no wallclock speedup. There is, however, still an advantage of reduced CPU time, as shown below.

The speedup tends to be higher when the images are bigger. This can be explained by the relative image transmission overhead that is higher for small image sizes. The bigger the images are, the more the DSPs can account for the overall calculation performance and the less impact the initial image transmission has. A rule of thumb is that the bigger the image sizes are, the more advantageous the utilization of DSPs is.

7.6 Processor utilization

An embedded system's main processor or microcontroller is usually designed to cause as little procurement costs as possible for the device's operation requirements. It is often responsible for ongoing tasks like continuous readout of sensor data, reacting to real-world stimuli, controlling actuators, and performing data transfers. These tasks can easily consume a high amount of processing power. When calculations are offloaded to DSPs, the main processor remains simultaneously available for the tasks mentioned above.

Offloading tasks to coprocessors becomes especially attractive for embedded systems that operate the main processor constantly near the performance limit. In this case, the offload ratio (quotient of CPU times) can be more important than the speedup (quotient of the wallclock times). As previously mentioned, a coprocessor giving a negative speedup can still be beneficial when the offload ratio is positive.

ID	512px	1024px	2048px	4096px
Intel-0	31.2	105.5	202.1	851.9
Intel-1	1.1	1.8	4.3	10.6
Intel-4	3.3	7.4	13.5	42.6
Arm-0	980.0	3,190.0	17,180.0	33,510.0
Arm-1	20.0	50.0	140.0	530.0

Table 8 CPU time of the main CPU in milliseconds for a complete image registration (according to Figure 15) for different image sizes on different hardware. The column ID refers to the experiments listed in Table 3.

Table 8 shows the amount of CPU time used for the image registration. It is a sum of the time periods in which each core was busy with registration processing. In the case of a multicore system, the CPU time can be higher than the wallclock time. A remarkable decrease in processor utilization can be seen on all setups when DSP coprocessors are used.

A tradeoff between CPU time and wallclock time shows up in the Intel setup. The Intel-4 setup in Table 6 was clearly superior to the Intel-1 setup in terms of wallclock time. However, the Intel-1 setup in Table 8 exhibits superior CPU time.

ID	512px	1024px	2048px	4096px
$\frac{Intel-0}{Intel-1}$	27.9	59.6	47.0	80.3
$\frac{Intel-0}{Intel-4}$	9.5	14.3	15.0	20.0
$\frac{Arm-0}{Arm-1}$	49.0	63.8	122.7	63.2

Table 9 Offload factors by the use of DSP coprocessors.

In Table 9, the offload factor is shown, which is calculated the same way the speedup is calculated, but using the CPU times instead of the wallclock times. The relative savings of CPU resources are very high. This can be explained by the fact that the main processor is only busy with the image distribution to the DSPs in the beginning, and most of the mathematical calculation work is done in the DSPs, which do not affect the CPU. In this case, they neither consume processor cycles nor RAM bandwidth because our approach is based on dedicated RAM for each DSP.

This way, a cost-effective and energy efficient main processor can be chosen that has enough free calculation resources available for other tasks during the image registration.

7.7 Prediction for a custom circuit board

As clarified in Section 5.4, it was out of our research scope to design and manufacture a custom circuit board connecting the ARM and DSP chips on one single PCB with a highspeed bus. Therefore, we do not present measurement values for such a circuit board. In Section 5.4.1, however, we proposed and verified a measurement and calculation method that predicts how our Ethernet based ARM setup would behave if Ethernet was replaced by PCIe.

Value	512px	1024px	2048px	4096px
Wallclock	29.6ms	76.4ms	202.1ms	721.4ms
Speedup	32.86	41.79	85.13	46.50

Table 10 Prediction of an ARM and PCIe based custom circuit board. The values were not directly measured, but instead calculated using the formula in Section 5.4.1.

Table 10 shows the results of this calculation. The nature of a prediction is to have some degree of uncertainty. In our verification of the prediction method in Section 5.4.1, we measured an error between 3.1%-4.5%; other setups might have a higher or lower error. A prediction is only made for the wallclock time because the CPU time is not affected as much by changing the transport layer and the values in the Arm-1 row of the Tables 8, and 9 can be taken as an orientation.

7.8 Power consumption

Table 11 lists the average electrical power consumption of the measurement setups during a consecutive series of image registrations with an image size of 4096x4096 pixels and the energy necessary for one particular image registration. The consumption of an Intel-1 setup could not be measured because single DSP chips could not be deactivated on the DSPC8681.

Setup	Power	Energy
Intel-0	103.6 W	20.33 Ws
Intel-4	70.2 W	6.51 Ws
Arm-0	1.3 W	43.61 Ws
Arm-1	21.9 W	28.98 Ws

Table 11 Power consumption of the experimental setups and energy consumption of one particular image registration (4096x4096 pixel, no display connected).

In the first column, it can be seen that the overall device power consumption is lowered on the Intel setup when DSP coprocessors are in use. The higher energy demand of the DSP processors is obviously overcompensated by the savings of the lower CPU utilization.

Since Intel-4 additionally calculates 2.33 times faster than Intel-0, which amplifies the effect, the energy savings for one particular image registration become quite attractive: 3.12 times less energy.

The power consumption of the ARM setup is significantly increased by the additional DSP evaluation module, but because of the enhanced calculation speed of factor 25.35, we can see in the second column of Table 11 that one particular image registration still consumes less energy with DSPs than without. The higher power demand is overcompensated by the higher calculation speed. On a real PCB, this effect can be assumed to be even higher because the registration would be faster (as shown in Section 7.7) and the overhead of adding only a sole DSP chip would be lower than by adding a whole experimental evaluation board.

7.9 Conclusion

Our work shows that image registration using derivative based optimization is possible on low cost, low power and low space embedded systems at very high speed, which is even fast enough for real-time imaging applications needing response times within a couple of milliseconds (depending on the image size).

To achieve this, this work derived special mathematical and algorithmical approaches optimized for state-of-the-art embedded hardware based on multicore processors. By means of efficient algorithm parallelization as well as by reducing the utilization of memory and bus systems (PCIe, RAM), these concepts provide tremendous speedup compared to a classical approach. By using additional DSP coprocessors with dedicated RAM, an even higher speedup could be observed, while the calculations were also being offloaded from the main processor. Thus, an energy-efficient and cost-effective main processor can operate the embedded system's sensors, actuators, and realtime tasks without conflicting with the high-speed image registration.

The mathematic and algorithmic principles are generic and can easily be applied to different numbers of DSPs as well as to different mathematical components, such as another distance measure or a higher spatial dimension. Therefore, our method is versatile for use in many registration problems in various fields.

Our unique combination of super-efficient software on super-fast, yet cost-effective DSP coprocessors makes image registration using derivative-based optimization available to applications that depend on small, cost-effective, and energy-efficient embedded computers.

Acknowledgements The software created during this work is open-source and can be accessed at http://github.com/RoelofBerg/fimreg.

In deep sorrow, we commemorate Prof. Dr. rer. nat. Bernd Fischer who passed away during the creation of this paper. Our thoughts are with his family.

References

1. Advantech (2013) DSPC-8681 – half-length PCI express card with 4 TMS320C6678 DSPs. URL http://downloadt.advantech.com/ProductFile/PIS/DSPC-8681/Product%20-%20Datasheet/DSPC-8681_DS%2803.31.14%2920140519134025.pdf
2. Alavi A, et al (2007) Is PET-CT the only option? European Journal of Nuclear Medicine and Molecular Imaging 34:819–821
3. Brown LG (1992) A survey of image registration techniques. ACM Comput Surv 24(4):325–376
4. Capek K (1999) Optimisation strategies applied to global similarity based image registration methods. In: International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG), vol 2, pp 369–374
5. Castro-Pareja CR, Jagadeesh JM, Shekhar R (2003) FAIR: a hardware architecture for real-time 3-D image registration. Information Technology in Biomedicine, IEEE Transactions on 7(4):426–434
6. Dennis JJE, Schnabel RB (1983) Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM
7. Evans JR, Arslan T (2002) The implementation of an evolvable hardware system for real time image registration on a system-on-chip platform. In: Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on, IEEE, pp 142–146
8. Eyre J, Bier J (2000) The evolution of DSP processors. IEEE Signal Processing Magazine 17(2):43–51
9. Fischer B, Modersitzki J (2008) Ill-posed medicine – an introduction to image registration. Inverse Problems 24(3):034,008
10. Geronimo D, Lopez AM, Sappa AD, Graf T (2010) Survey of pedestrian detection for advanced driver assistance systems. Pattern Analysis and Machine Intelligence, IEEE Transactions on 32(7):1239–1258
11. Gigengack F, Ruthotto L, Burger M, Wolters CH, Jiang X, Schafers KP (2012) Motion correction in dual gated cardiac PET using mass-preserving image registration. Medical Imaging, IEEE Transactions on 31(3):698–712
12. Gonzalez RC, Woods RE (1992) Digital Image Processing, vol 2. Addison-Wesley
13. Haber E, Modersitzki J (2006) A multilevel method for image registration. SIAM Journal on Scientific Computing 27(5):1594–1607
14. Haber E, Modersitzki J (2007) Intensity gradient based registration and fusion of multi-modal images. Methods of Information in Medicine 46:292–9

15. Hossny M, Nahavandi S, Creighton D, Bhatti A (2010) Towards autonomous image fusion. In: Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on, IEEE, pp 1748–1754
16. Intel Corporation (2013) Desktop 3rd generation Intel Core processor family, desktop Intel Pentium processor family, and desktop Intel Celeron processor family. URL www.intel.com/content/dam/www/public/us/en/documents/datasheets/3rd-gen-core-desktop-vol-1-datasheet.pdf
17. Irani M, Peleg S (1991) Improving resolution by image registration. CVGIP: Graphical models and image processing 53(3):231–239
18. Kabus S, Lorenz C (2010) Fast elastic image registration. Grand Challenges in Medical Image Analysis pp 81–89
19. Karam LJ, AlKamal I, Gatherer A, Frantz GA, Anderson DV, Evans BL (2009) Trends in multicore DSP platforms. Signal Processing Magazine, IEEE 26(6):38–49
20. Kessler CW (2013) Compiling for VLIW DSPs. In: Handbook of Signal Processing Systems, Springer, pp 1177–1214
21. König L, Rühaak J (2014) A fast and accurate parallel algorithm for non-linear image registration using normalized gradient fields. In: Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on, IEEE, pp 580–583
22. Kontron AG (2009) Infotainment POS/POI. URL http://www.kontron.com/resources/collateral/industry_brochures/pos_poi_2010_global_single.pdf
23. Kontron AG (2013) Embedded Computer Solutions for Advanced Automation Control. URL http://www.kontron.com/resources/collateral/industry_brochures/folder_automation_2013.pdf
24. Leon FP, Kammel S (2003) Image fusion techniques for robust inspection of specular surfaces. In: AeroSense 2003, International Society for Optics and Photonics, pp 77–86
25. Maes F, Collignon A, Vandermeulen D, Marchal G, Suetens P (1997) Multimodality image registration by maximization of mutual information. Medical Imaging, IEEE Transactions on 16(2):187–198
26. Mahapatra NR, Venkatrao B (1999) The processor-memory bottleneck: problems and solutions. Crossroads 5(3es):2
27. Mattes D, Haynor DR, Vesselle H, Lewellen TK, Eubank W (2003) PET-CT image registration in the chest using free-form deformations. Medical Imaging, IEEE Transactions on 22(1):120–128
28. Modersitzki J (2004) Numerical Methods for Image Registration. Oxford University Press
29. Modersitzki J (2009) FAIR - Flexible Algorithms for Image Registration. SIAM, Philadelphia
30. Mueller B, Olesch J, Lotz J, Barendt S, Sedlaczek O, Lahrmann B, Grabe N, Bestvater F, Kauczor U, Schnabel P, Hoffmann H, Fischer B, Schirmacher P, Warth A, Breuhahn K (2013) 3D reconstruction of lung adenocarcinomas – one module for the development of mathematical multiscale models of lung cancer. Der Pathologe 34(1):140
31. Nocedal J, Wright S (2006) Numerical optimization, 2nd edn. Springer, Berlin, Heidelberg
32. Reed JM, Hutchinson S (1996) Image fusion and subpixel parameter estimation for automated optical inspection of electronic components. Industrial Electronics, IEEE Transactions on 43(3):346–354
33. Remagnino P, Jones G (2002) Automated registration of surveillance data for multi-camera fusion. In: Information Fusion, 2002. Proceedings of the Fifth International Conference on, IEEE, vol 2, pp 1190–1197
34. Rühaak J, Heldmann S, Kipshagen T, Fischer B (2013) Highly accurate fast lung CT registration. In: SPIE Medical Imaging, International Society for Optics and Photonics
35. Rühaak J, König L, Hallmann M, Papenberg N, Heldmann S, Schumacher H, Fischer B (2013) A fully parallel algorithm for multimodal image registration using normalized gradient fields. In: Biomedical Imaging (ISBI), 2013 IEEE 10th International Symposium on, pp 572–575
36. Saban N (2011) Multicore DSP vs GPUs. URL www.sagivtech.com/contentManagment/uploadedFiles/fileGallery/Multi_core_DSPs_vs_GPUs_TI_for_distribution.pdf
37. Schmitt O, Modersitzki J, Heldmann S, Wirtz S, Fischer B (2007) Image registration of sectioned brains. International Journal of Computer Vision 73(1):5–39
38. Sen M, Hemaraj Y, Plishker W, Shekhar R, Bhattacharyya SS (2008) Model-based mapping of reconfigurable image registration on FPGA platforms. Journal of Real-Time Image Processing 3(3):149–162
39. Stotzer E, Jayaraj A, Ali M, Friedmann A, Mitra G, Rendell A, Lintault I (2013) OpenMP on the low-power TI keystone II ARM/DSP system-on-chip. In: Rendell A, Chapman B, Müller M (eds) OpenMP in the Era of Low Power Devices and Accelerators, Lecture Notes in Computer Science, vol 8122, Springer Berlin Heidelberg, pp 114–127
40. Texas Instruments (2013) AM335x sitara processors. URL http://www.ti.com/lit/ds/symlink/am3359.pdf
41. Texas Instruments (2014) AM335x starter kit. URL www.ti.com/tool/tmdssk3358
42. Texas Instruments (2014) C6678 power consumption model (rev. d). URL www.ti.com/litv/zip/sprm545d
43. Texas Instruments (2014) SYS/BIOS (TI-RTOS kernel) v6.40. URL http://www.ti.com/lit/ug/spruex3n/spruex3n.pdf

44. Texas Instruments (2014) TMS320C6678 - multicore fixed and floating-point digital signal processor. URL www.ti.com/lit/ds/symlink/tms320c6678.pdf
45. Texas Instruments (2014) TMS320C6678 evaluation modules. URL www.ti.com/tool/tmdsevm6678
46. Tramnitzke F, Rühaak J, König L, Modersitzki J, Köstler H (2014) GPU Based Affine Linear Image Registration using Normalized Gradient Fields. In: Proc. Seventh International Workshop on High Performance Computing for Biomedical Image Analysis (HPC-MICCAI), Boston, MA, USA
47. Vercauteren T, Pennec X, Perchant A, Ayache N (2009) Diffeomorphic demons: Efficient non-parametric image registration. NeuroImage 45(1):S61–S72
48. Viola P, Wells III WM (1997) Alignment by maximization of mutual information. International Journal of Computer Vision 24(2):137–154
49. Wu H, Kim Y (1998) Fast wavelet-based multiresolution image registration on a multiprocessing digital signal processor. International Journal of Imaging Systems and Technology 9(1):29–37
50. Zitová B, Flusser J (2003) Image registration methods: a survey. Image and Vision Computing 21(11):977 – 1000