Efficient Distributed Torque Computation for Large Scale Robot Skin

Florian Bergner, Emmanuel Dean-Leon and Gordon Cheng

***Abstract*— The realization of a kinesthetic robot behavior using robot skin requires a reactive skin torque controller, which fuses skin information and robot information to an appropriate skin joint torque in real-time. This fusion of information in real-time is challenging when deploying large scale skin. In this paper, we present a system which efficiently computes the torque of distributed skin cells locally at the point of contacts, completely removing this complex computations from the real-time loop. We demonstrate the feasibility of realizing the skin joint torque computations on the local micro-controllers of the skin cells. Conducting experiments with a real robot, we compare the accuracy of the distributed skin joint torque computation with the computation on the control PC. We also show that the novel distributed approach completely eliminates the computational delay of computing skin joint torques in the robot's real-time control loop. As a result, this approach removes any limits for the maximum number of skin cells in control.**

I. INTRODUCTION

A. Motivation

Envisioning collaborative robots for assisting humans in industrial, health-care and household application scenarios, robots have to be safe and realize intuitive, effective and straight-forward human-robot interaction [1], [2]. Human-robot-interaction could benefit with the ability to manually/kinesthetically guide the robot simply by touch. Such an ability becomes particularly useful in teaching scenarios where the human teaches the robot a new task [1], [3]. Manual guidance through touch can be realized using different approaches such as installing force-torque sensors in robot's joints. A reactive controller with a dynamic robot model then discriminates between external and internal torques [4], [5]. A recent and effective approach is to deploy robot skin in order to realize a reactive skin controller [6]–[9]. Robot skin can overcome some drawbacks of force-torque sensors [8], [10]–[14]. Installing force-torque sensors in robot joints is expensive, and it is often infeasible to modify existing robots to enable safe human-robot-interaction. Furthermore, force-torque sensors can only sense the sum of all internal and external forces/torques and thus can not distinguish contact points. Whereas, robot skin can be deployed on existing robots to enhance their safety and their abilities in human-robot interaction. Robot skin can easily locate, sense, and discriminate contact points. However, to fully exploit the advantages of robot skin, robot skin has to cover the whole robot. This induces new challenges caused by large numbers of distributed skin sensors in large scale robot skin. The main challenges are: 1) transmitting huge amounts of tactile information from distributed sensors with low-latency to the reactive skin controller, and 2) processing and evaluating huge amounts of tactile information to extract necessary information in real-time. It has been shown that neuromorphic event-driven sensing, as conducted in the works of [15]–[19], is a valuable solution to extract novel information at sensor level. We introduced an event-driven reactive skin torque controller[1] in [20] to reduce computational load at control level. This paper exploits the distributed computing capabilities of our robot skin to remove the complex computation of skin joint torques from the real-time loop of the controller to the skin cells, enabling the robot to be more responsive.

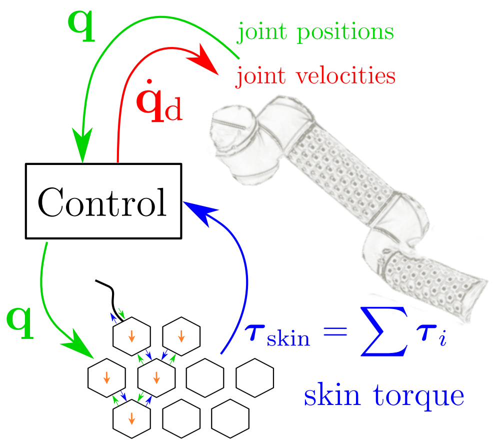

$\mathbf{q}$
joint positions
joint velocities
$\dot{\mathbf{q}}_\mathrm{d}$
Control
$\mathbf{q}$
$\boldsymbol{\tau}_\mathrm{skin} = \sum \boldsymbol{\tau}_i$
skin torque

Fig. 1. Instead of computing the skin joint torque $\boldsymbol{\tau}$ inside the real-time loop of the controller, the skin joint torque is distributedly computed by the skin cells.

B. Related Work

The proposed distributed computation of skin joint torques at the local skin cells for reactive skin torque control is related to a standard impedance control at torque level ($\boldsymbol{\tau} = \mathbf{J}^T\mathbf{F}$). To our best knowledge, computing distributed skin joint torques at skin cell level has not been fully addressed. The importance of reactive torque control for kinesthetic robot behaviors using force/torque sensors have been pointed out [4], [5]. Reactive controllers using robot skin have been introduced in the works of [6]–[9], [13], [14]. Realizing the heavy computation required in the dealing of large number of distributed skin cells and/or DOFs (degrees of freedom), the recent work of [21] introduced an efficient asynchronous algorithm for computing forward kinematics

All authors are with the Institute for Cognitive Systems, Technische Universität München, Munich, Germany, see `http://www.ics.ei.tum.de`

[1]A skin torque controller is a controller that uses tactile information from the skin to compute joint torques as commands for the robot.

for a large number of skin cells and DOFs. The authors use an update/query method to compute updates only when necessary.

C. Contributions

Instead of computing the skin joint torque contribution of each active skin cell in the real-time control-loop of the robot, we compute these skin joint torques locally at the distributed skin cells. We demonstrate the feasibility of realizing the complex skin joint torque computation on the micro-controllers of the skin cells. Conducting experiments with a real robot, we document that computing skin joint torques on the skin cells results in the same skin joint torque as when computing skin joint torques in the real-time loop of the robot. Furthermore, the experiments with the real robot demonstrate that the distributed computation of the skin joint torques completely removes the delays in the real-time loop, which are usually caused by computing the skin joint torque. The overall cycle time of the real-time control loop decreases and becomes independent to the number of active cells and the total number of skin cells, making it highly responsive.

II. SYSTEM DESCRIPTION

A. Robot skin

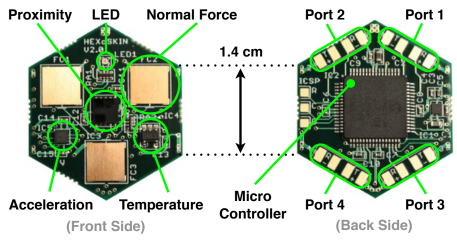


Fig. 2. Robot skin.

Our multi-modal, self-organizing, and self-calibrating robot skin consists of hexagonally shaped skin cells [8]. A collection of directly connected skin cells froms a skin patch. The robot skin automatically constructs a self-organized and redundant communication network. Each skin cell of this network acquires a bidirectional communication path to the PC such that the PC can send information to the skin cells (down-link) and receive information from the skin cells (up-link). All the skin cells use the same set of sensors to detect multi-modal tactile stimuli. The skin cells employ a 3D accelerometer to sense vibrations, two temperature sensors, three capacitive force sensors to measure contact forces, and a proximity sensor to detect pre-touch. The micro-controller firmware of the skin cells supports two different modes: 1) the continuous mode with a constant sampling rate; and 2) the event mode with non-constant novelty driven sampling rate [17], [18]. In this paper, we exploit the ability to update and change the skin cell firmware in order to realize skin joint torque computation locally on the skin cells.

B. The Tactile Omnidirectional Mobile Manipulator TOMM

The new approach is tested with our robot TOMM [22], see Fig. 3. TOMM's arms and grippers are covered with

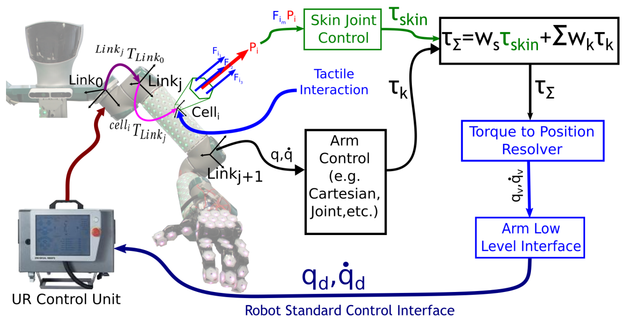


Fig. 3. The control framework of the robot TOMM.

robot skin. Furthermore TOMM is fully integrated in the ROS (Robot OS) middleware. A set of different robot behaviors is realized with a control loop as depicted in Fig. 3. The green box in Fig. 3 addresses the reactive skin torque controller of in this paper. This skin torque controller enables a kinesthetic behavior in the joint space but it can also be used for other robot behaviors.

C. Reactive skin torque control

A reactive skin torque controller, that creates a kinesthetic behavior, and that avoids contacts, needs to compute an appropriate skin joint torque $\boldsymbol{\tau}_{\mathrm{skin}}$. This skin joint torque $\boldsymbol{\tau}_{\mathrm{skin}}$ is the sum of all skin joint torques $\boldsymbol{\tau}_i$ of skin cells i that detected external forces F_{z}.

$$\boldsymbol{\tau}_{\mathrm{skin}} = \sum \boldsymbol{\tau}_i \tag{1}$$

1) Active skin cells: Skin cells i detect external normal forces along their **z**-axes. These external forces $F_{\mathrm{z},i}$ are determined by fusing the sum of the three capacitive contact forces F_{c} with the proximity value F_{p} (see Algo. 1). This fusion is weighted with positive gains α_{c} and α_{p} and thresholded by $F_{\mathrm{c,th}}$, $F_{\mathrm{p,th}}$, and $F_{\mathrm{z,th}}$. A skin cell i is *active*, e.i contributes a non-zero torque $\boldsymbol{\tau}_i$, whenever the external force F_{z} is greater than zero. The *number of active cells* is $N_{\mathrm{cell,act}}$.

Algorithm 1 Compute F_{z} for a given F_{c} and F_{p}

```
1:  if $F_c < F_{c,th}$ then
2:      $F_c := 0$
3:  end if
4:  if $F_p < F_{p,th}$ then
5:      $F_p := 0$
6:  end if
7:  $F_z := \alpha_c F_c + \alpha_p F_p$
8:  if $F_z < F_{z,th}$ then
9:      $F_z := 0$
10: else
11:     $N_{cell,act} := N_{cell,act} + 1$
12: end if
```

2) Computing the skin joint torque: The skin joint torque $\boldsymbol{\tau}_{\mathrm{skin}}$ of a reactive skin torque controller is computed cell-wise by computing the skin joint torque contribution $\boldsymbol{\tau}_i$ of an active skin cell i. A skin joint torque $\boldsymbol{\tau}_i$ of a cell is computed by mapping the skin cell wrench $\mathbf{w}_i \in \mathbb{R}^6$ properly to this skin joint torque $\boldsymbol{\tau}_i$ [9]. The skin joint torque $\boldsymbol{\tau}_i \in \mathbb{R}^{\mathrm{DOFs}}$ is a vector of joint torques τ_l. A skin cell i located on a robot

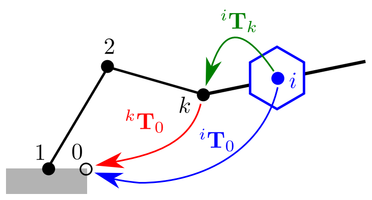


Fig. 4. Kinematic chain from a skin cell i on a robot limb actuated by a joint k to the world reference coordinate frame 0.

limb actuated by a joint k only produces torques τ_l in robot joints $1, \dots, k$ (see Fig. 4). In this case the torques τ_l in the robot joints $k+1, \dots, \mathrm{DOFs}$ are zero.

$$\boldsymbol{\tau}_i = \begin{bmatrix} \tau_1 & \cdots & \tau_k & 0 & \cdots & 0 \end{bmatrix}^T \in \mathbb{R}^{\mathrm{DOFs}} \tag{2}$$

The mapping from wrenches $\mathbf{w}_i$ to torques $\boldsymbol{\tau}_i$ is resolved by computing the transposed Jacobians $\mathbf{J}_i^T(\mathbf{q})$ of skin cells i and applying the principle of virtual work:

$$\boldsymbol{\tau}_i = \mathbf{J}_i^T(\mathbf{q})\ {}^{i}\mathbf{w}_0 \tag{3}$$

The wrench ${}^{i}\mathbf{w}_0$ is the wrench of the cell i with respect to the robot base frame 0. The external forces $F_{\mathrm{z},i}$ of active cells are aligned with the $\mathbf{z}$-axes of these cells. Thus the wrench vector $\mathbf{w}_i$ simplifies to:

$$\mathbf{w}_i = \begin{bmatrix} 0 & 0 & F_{\mathrm{z},i} & 0 & 0 & 0 \end{bmatrix}^T \in \mathbb{R}^6 \tag{4}$$

The Jacobian $\mathbf{J}_i$ of a skin cell i can be geometrically determined by applying the laws of physics for mapping linear and angular velocities $\dot{\mathbf{x}}$ and $\boldsymbol{\omega}$ to joint velocities $\dot{\mathbf{q}}$:

$$\mathbf{J}_i = \begin{bmatrix} {}^{0}\mathbf{z}_0 \times \left[{}^{i}\mathbf{t}_0 - {}^{0}\mathbf{t}_0\right] & \dots & {}^{l-1}\mathbf{z}_0 \times \left[{}^{i}\mathbf{t}_0 - {}^{l-1}\mathbf{t}_0\right] \\ {}^{0}\mathbf{z}_0 & \dots & {}^{l-1}\mathbf{z}_0 \end{bmatrix} \tag{5}$$

In general, the l-th column of $\mathbf{J}_i$ corresponds to a joint l:

$$\mathbf{j}_{l,i}(\mathbf{q}) = \begin{bmatrix} {}^{l-1}\mathbf{z}_0 \times \left({}^{i}\mathbf{t}_0 - {}^{l-1}\mathbf{t}_0\right) & {}^{l-1}\mathbf{z}_0 \end{bmatrix}^T \in \mathbb{R}^6 \tag{6}$$

The torque τ_l in joint l can be computed by

$$\tau_{l,i}(\mathbf{q}) = \mathbf{j}_{l,i}^T(\mathbf{q})\ {}^{i}\mathbf{w}_0 \in \mathbb{R} \qquad \textbf{if} \quad l \leq k \tag{7}$$

for a skin cell i on a robot limb actuated by a joint k. As the wrench $\mathbf{w}_i$ has only one entry Eq. 7 simplifies to

$$\begin{aligned} \tau_{l,i}(\mathbf{q}) &= \left[{}^{l-1}\mathbf{z}_0 \times \left({}^{i}\mathbf{t}_0 - {}^{l-1}\mathbf{t}_0\right)\right]^T\ {}^{k}\mathbf{R}_0\ {}^{i}\mathbf{z}_k\ F_{\mathrm{z},i} \\ &= J_{l,i}(\mathbf{q})\ F_{\mathrm{z},i} \end{aligned} \tag{8}$$

The vector ${}^{l-1}\mathbf{z}_0 \in \mathbb{R}^3$ is the $\mathbf{z}$-axis in coordinate frame $l-1$ and ${}^{i}\mathbf{t}_0$ is the origin of the coordinate frame of cell i, both with respect to the base coordinate frame 0. The transformation from l to 0 is defined by the homogeneous transformation matrix ${}^{l}\mathbf{T}_0$. This matrix contains the axes $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ and the origin $\mathbf{t}$ of coordinate frame l with respect to the base coordinate frame 0:

$${}^{l}\mathbf{T}_0(\mathbf{q}) = \begin{bmatrix} {}^{l}\mathbf{x}_0 & {}^{l}\mathbf{y}_0 & {}^{l}\mathbf{z}_0 & {}^{l}\mathbf{t}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{9}$$

To compute the torque contribution $\boldsymbol{\tau}_i$ of a cell i on a robot limb actuated by a joint k, we need to compute the forward kinematics from the base coordinate frame i to the coordinate frame of joint k, thus ${}^{l}\mathbf{T}_0$ for $l \leq k$, and also the static transformation ${}^{i}\mathbf{T}_k$ from the coordinate frame of cell i to the coordinate frame of joint k (see Fig. 4).

D. *Distributed computation of skin joint torque*

The computation of skin joint torques in the real-time control loop is computationally expensive and the induced delay increases with the number of active cells. In order to reduce these delays, we distribute the computation of skin joint torques $\boldsymbol{\tau}_i$ from the centralized computation on the PC to a distributed computation in the skin cells. Each skin cell exploits its micro-controller to compute its own skin joint torque. This skin joint torque is then sent to controller running on the PC (see Fig. 1). The decentralized computation of skin joint torques $\boldsymbol{\tau}_i$ on skin cells i requires the following steps:

1) provide the static transformation ${}^{i}\mathbf{T}_k$
2) receive the most recent joint positions $\mathbf{q} \in \mathbb{R}^{\mathrm{DOFs}}$
3) compute the forward kinematics to the cell, thus ${}^{l}\mathbf{T}_0(\mathbf{q})$ for $l \leq k$
4) compute the external force $F_{\mathrm{z},i}$
5) compute the components $\tau_{l,i}$ of $\boldsymbol{\tau}_i$, using Eq. 8
6) send the torques $\boldsymbol{\tau}_i$ from the skin cells to the reactive skin torque controller

We realize the distributed computation of skin joint torques for a UR5 robot arm with 6 DOFs. The computational power of the micro-controllers of the skin cells is very limited thus we optimize the computation procedure for this specific robot arm. Nevertheless this does not limit the generality of the approach as its optimization principles easily apply to other robots.

1) Fixed point arithmetic: The micro-controllers of the skin cells use a 16-bit architecture without a floating point unit (FPU) and process with 16 MIPS (mega instructions per second). Using floating point arithmetic on this micro-controller is infeasible because even simple floating point operations such as sums and products translate to hundreds of assembly instructions. In order to achieve fast computations on this micro-controller we realize all arithmetic operations with fixed point numbers and use 32 bit and 16 bit precision. Fixed point numbers reserve one bit to realize negative numbers with the two's complement and use the remaining bits for integer and fraction. We use the Q notation for fixed point numbers. A $Q1.14$ number uses bit 15 as sign bit, bit 14 for the integer part and bits 13 to 0 for the fractional part and can represent numbers in $[-2, 2[$ with a constant precision of $2^{-14} = 6.1035 \cdot 10^{-5}$. Additions and multiplications of fixed point numbers map to integer additions and multiplications. To add fixed point numbers both numbers must have the same number of fractional bits. While additions preserve the number of fractional bits, multiplications change it. The number of fractional bits can easily be adjusted by using shift operations. As long as arithmetic operations have a limited dynamic range, fixed point arithmetic can provide good accuracy and enable fast operations on architectures without a FPU. Additions and multiplications only require a few assembly instructions.

2) Storing skin cell poses into skin cells: The distributed computation of the skin joint torque $\tau_{l,i}$ of a cell i requires the static transformation ${}^{i}\mathbf{T}_k$. The new skin cell firmware provides an interface to store this transformation using $Q1.14$ into the non-volatile flash memory. Note that the transformation ${}^{i}\mathbf{T}_k$ is different for each skin cell i. The skin cells also store the id of the robot joint k, the robot joint which directly actuates the limb of skin cell i. The transformation ${}^{i}\mathbf{T}_k$, the static transformation from skin cell i to joint k, has been determined using our 3D self-calibration algorithm [3], [23].

3) Fast trigonometric functions: Fast and precise sine and cosine functions are essential for updating the forward kinematics from $l = 1, \ldots k$ sufficiently fast and with sufficient precision. The homogeneous transformation matrix ${}^{l}\mathbf{T}_{l-1}(q_l)$ from joint l to $l-1$ contains a rotation matrix ${}^{l}\mathbf{R}_{l-1}(q_l) \in \mathbb{R}^{3\times3}$ and a translation vector ${}^{l}\mathbf{t}_{l-1} \in \mathbb{R}^3$:

$$ {}^{l}\mathbf{T}_{l-1}(q_l) = \begin{bmatrix} {}^{l}\mathbf{R}_{l-1}(q_l) & {}^{l}\mathbf{t}_{l-1} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{10}$$

All the transformations ${}^{l}\mathbf{T}_{l-1}(q_l)$ between joints of the kinematic chain use the Denavit-Hartenberg convention, thus each of these transformations contains $\sin(q_l) = s_l$ and $\cos(q_l) = c_l$ in ${}^{l}\mathbf{R}_{l-1}$. In worst case, for a skin cell i actuated by joint $k = 6$, the micro-controller has to compute 6 different sines and cosines. Sine/cosine functions can be approximated by polynomials, look-up-tables (LUTs) with interpolation, Taylor series, or the CORDIC algorithm [24]. LUTs require memory and are slow since memory access is slow. Taylor series approximations are inaccurate since an infinite series is truncated and the special properties of sine/cosine functions are lost. The CORDIC algorithm uses only shift and addition operations but is slow in architectures where shift, addition and multiplication operations have the same cost since it only converges linearly. To get fast and precise fixed point results for sine/cosine operations, we approximate the $\sin(x)$ with a 5-th order polynomial in the interval $x \in [-\frac{\pi}{2}, \frac{\pi}{2}[$. In order to simplify the computation and notation, we transform coordinates and substitute x by z:

$$ z := \frac{x}{0.5\,\pi} \tag{11}$$

This normalizes angular input arguments x in rad. The sine function $\sin(z)$ has a period of $z = 2\,k$:

$$ \sin(z) = \sin(z \pm 2\,k) \qquad \forall k \in \mathbb{Z} \tag{12}$$

The sine function is an odd function, thus all parameters of the 5th order polynomial connected to even powers are zero:

$$ p_5(z) = a\,z - b\,z^3 + c\,z^5 \tag{13}$$
$$ \dot{p}_5(z) = a - 3\,b\,z^2 + 5\,c\,z^4 \tag{14}$$

We determine appropriate parameters with three conditionals which define the behavior of the polynomial on its boundaries and preserve the special properties of a sine function:

$$ p_5(z=1) \overset{!}{=} 1 \quad \dot{p}_5(z=1) \overset{!}{=} 0 \quad \dot{p}_5(z=0) \overset{!}{=} \frac{\pi}{2} \tag{15}$$

Solving this equation system with 3 unknown parameters and 3 conditions results into the following parameters:

$$ a = \frac{\pi}{2} \qquad b = \pi - \frac{5}{2} \qquad c = \frac{\pi}{2} - \frac{3}{2} \tag{16}$$

The simplified approximation of $\sin(z)$ for $z \in [-1, 1[$ is thus:

$$ \sin(z) \approx \sin_{5,p}(z) \qquad \forall z \in [-1, 1[$$
$$ \sin_{5,p}(z) = 0.5z\left(\pi - z^2\left[(2\pi - 5) - z^2\,(\pi - 3)\right]\right) \tag{17}$$

The sine function is a periodic function thus the approximation of Eq. 17 can be easily extended for general $z \in \mathbb{R}$. Changing the fixed point precision of z from $Qm.f$ to $Q1.14$ or respectively to $Q1.30$, easily and reliably maps, in a modulo-4-like operation, all $z \in \mathbb{R}$ to the interval $z \in [-2, 2[$. For this interval the sine function can be approximated in the following way:

$$ \sin_5(z) = \begin{cases} \sin_{5,p}(2 - z) & \textbf{if } z \in [-2, -1[\\ \sin_{5,p}(z) & \textbf{if } z \in [-1, 1[\\ \sin_{5,p}(2 - z) & \textbf{if } z \in [1, 2[\end{cases} \tag{18}$$

This approximation of the sine function is fast and only uses around 20 assembly instructions. Its realization on the micro-controller takes fixed point numbers of $Q1.14$ as argument and returns fixed point numbers in $Q1.14$. The approximation of the cosine is simply:

$$ \cos_5(z) = \sin_5(z + 1) \tag{19}$$

The mean error for $y = \sin_5(z)$ with z and y in $Q3.12$ is $6.7 \cdot 10^{-4}$.

4) Fast forward kinematics: The forward kinematics for a skin cell i actuated by joint k can be computed as follows:

$$ {}^{k}\mathbf{T}_0 = {}^{1}\mathbf{T}_0\,{}^{2}\mathbf{T}_1 \cdots {}^{k}\mathbf{T}_{k-1} \tag{20}$$

Multiplying two 4×4 matrices $\mathbf{AB} = \mathbf{C}$ results in:

$$ c_{ij} = \sum_{k=1}^{4} a_{ik} b_{kj} \tag{21}$$

an operation which has an arithmetic complexity of 64 `mul`+ 48 `add`. In worst case, for $k = 6$, the arithmetic complexity of updating the forward kinematics is 5 times a 4×4 matrix multiplication resulting in 320 `mul` + 240 `add`. Only computing the rotation ${}^{k}\mathbf{R}_0$ the arithmetic complexity reduces to 5 times a 3×3 matrix multiplication, which is 135 `mul` + 90 `add`. However, only computing ${}^{k}\mathbf{R}_0$ is not sufficient to compute the skin joint torque $\boldsymbol{\tau}_i$. Eq. 8 also requires the projections $\mathbf{p}_{i,l-1,0}$, the projection of ${}^{i}\mathbf{t}_0$ over ${}^{l-1}\mathbf{t}_0$ with respect to the base coordinate frame 0:

$$ \mathbf{p}_{i,l-1,0} = {}^{i}\mathbf{t}_0 - {}^{l-1}\mathbf{t}_0 \tag{22}$$

At first glance, this requires the computation of ${}^{i}\mathbf{T}_0$ which induces one additional 4×4 matrix multiplication resulting in 384 `mul` $+$ 288 `add`. However the projection $\mathbf{p}_{i,l-1,0}$ can be computed much more efficiently:

$$\begin{aligned}\mathbf{p}_{i,l-1,0} &= {}^{l-1}\mathbf{R}_0\,{}^{i}\mathbf{t}_{l-1}\\ &= {}^{l-1}\mathbf{R}_0\left({}^{k}\mathbf{R}_{l-1}\,{}^{i}\mathbf{t}_k + {}^{k}\mathbf{t}_{l-1}\right)\\ &= {}^{k}\mathbf{R}_0\,{}^{i}\mathbf{t}_k + {}^{l-1}\mathbf{R}_0\,{}^{k}\mathbf{t}_{l-1}\\ &= \mathbf{p}_{i,k,0} + \mathbf{p}_{k,l-1,0} \qquad (23)\end{aligned}$$

The projection $\mathbf{p}_{i,k,0}$ is needed for the computation of all the components $\tau_{l,i}$ of $\boldsymbol{\tau}_i$ and the projection $\mathbf{p}_{k,l-1,0}$ can be computed easily by:

$$\mathbf{p}_{k,l-1,0} = \sum_{m=l-1}^{k-1} {}^{m}\mathbf{R}_0\,{}^{m+1}\mathbf{t}_m \qquad \forall l \le k \qquad (24)$$

All the required rotation matrices are determined when computing ${}^{k}\mathbf{R}_0$. In the worst case for $k = 6$, the computation of $\mathbf{p}_{k,l-1,0}$ requires 6 times a matrix-vector multiplication (9 `mul` $+$ 6 `add`) and 5 times a vector addition (3 `add`). This results in 54 `mul` $+$ 66 `add`. To sum up, the computation of

$$\begin{aligned}{}^{k}\mathbf{R}_0 &\rightarrow 135\ \texttt{mul} + 90\ \texttt{add}\\ \mathbf{p}_{i,0,0} &\rightarrow 63\ \texttt{mul} + 78\ \texttt{add}\\ {}^{k}\mathbf{R}_0\,{}^{i}\mathbf{z}_k &\rightarrow 9\ \texttt{mul} + 6\ \texttt{add} \qquad (25)\end{aligned}$$

results in 207 `mul` $+$ 174 `add`. In comparison to ${}^{i}\mathbf{T}_0$ which needs 384 `mul` $+$ 288 `add`, the exploitation of the properties of projections reduce the number of multiplications by 48% and the number of additions by 40%.
We further reduced the number of multiplications and additions in the final realization on the micro-controller by reusing sums and products in the different processing steps. The kinematics for $k = 6$ using the formulas of Eq. 25 results in 66 `mul` $+$ 26 `add`, reducing the final number of multiplications by 80% and the number of additions by 85%.

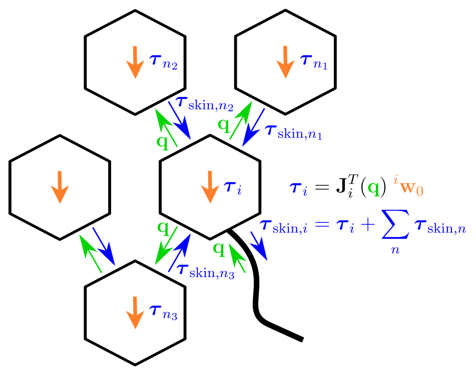
5) Torque computation and propagation: The skin cells receive the joint positions of the robot $\mathbf{q} \in \mathbb{R}^6$ as a broadcast message with a constant update rate f_q. $\mathbf{q}$ has six components q_i, $i = 1, \ldots 6$ which are trimmed and transformed into the z representation (see Eq. 11) such that:

$$\mathbf{q} = (q_i) \in \mathbb{R}^6 \quad \textbf{with} \quad q_i \in [-2, 2[\qquad (26)$$

These six joint positions q_i are encoded in $Q1.14$ and fit into one single skin packet. The ability to use broadcast messages and to pack $\mathbf{q}$ into one packet limits the transmission overhead in the down-link (PC-to-cell link) enabling high update rates f_q. A skin cell i receiving a new $\mathbf{q}$ immediately starts to update $J_{l,i}(\mathbf{q})$ using Eq. 8, Eq. 23, and 24. The realization of these equations on the skin cell is optimized such that only what is necessary is computed and, if possible, intermediate results are reused. In the worst case for $k = 6$, these computations for six $J_{l,i}(\mathbf{q})$ have an arithmetic complexity of:

$$(J_{l,i}) \rightarrow 103\ \texttt{mul} + 57\ \texttt{add} + 6\ \texttt{sin} + 6\ \texttt{cos} \qquad (27)$$

and require around 1096 assembly instructions. Thus updating $(J_{l,i})$ takes in worst case $68.5\ \mu$s.


$\boldsymbol{\tau}_{n_2}$
$\boldsymbol{\tau}_{n_1}$
$\boldsymbol{\tau}_{\text{skin},n_2}$
q
q
$\boldsymbol{\tau}_{\text{skin},n_1}$
$\boldsymbol{\tau}_i$
$\boldsymbol{\tau}_i = \mathbf{J}_i^T(\mathbf{q})\,{}^{i}\mathrm{w}_0$
q
q
$\boldsymbol{\tau}_{\text{skin},i} = \boldsymbol{\tau}_i + \sum_n \boldsymbol{\tau}_{\text{skin},n}$
$\boldsymbol{\tau}_{\text{skin},n_3}$
$\boldsymbol{\tau}_{n_3}$
Fig. 5. Propagating the sums of torques.

The skin cells also perform the computation of the external force $F_{z,i}$ according to Algorithm 1. The final computation of $\boldsymbol{\tau}_i = (J_{l,i})\,F_{z,i}$ is only performed when the cell is active. The skin joint torque $\boldsymbol{\tau}_i \in \mathbb{R}^6$ of a skin cell i uses a fixed point precision of $Q3.12$ and can be packed into one single skin packet. The skin joint torque $\boldsymbol{\tau}_i$ is only sent to the PC (up-link) when the cell is active. This limits the transmission overhead in the up-link.
The skin joint torques $\boldsymbol{\tau}_i$ don't have a frame of reference and can simply be added up as shown in Eq. 1. We exploit this nice property such that each skin cell i adds the skin joint torques $\boldsymbol{\tau}_{\text{skin},n}$ of its neighbors to its own skin joint torque $\boldsymbol{\tau}_i$ before passing the result $\boldsymbol{\tau}_{\text{skin},i}$ to the next skin cell (see Fig. 5):

$$\boldsymbol{\tau}_{\text{skin},i} = \boldsymbol{\tau}_i + \sum_n \boldsymbol{\tau}_{\text{skin},n} \in \mathbb{R}^6$$
$$n \in \{\text{neighbors of cell } i\} \qquad (28)$$

In order to allow for the higher numbers of the sums, we reduce the fixed point precision to $Q7.8$ for $\boldsymbol{\tau}_{\text{skin},i}$. Naturally this reduces the transmission rate in the up-link further at the cost of a slight decrease of precision of the final skin joint torque $\boldsymbol{\tau}_{\text{skin}}$.

III. EXPERIMENTS

A. Experimental setup

We conduct experiments to validate the accuracy of the distributed skin joint torque computation, and the reduction of computational load in the real-time loop of the controller. The experiments are performed on the left arm of our robot TOMM (see Fig. 3), employing 253 skin cells on its upper and lower arm. We use a reactive skin controller to generate the kinesthetic behavior in the joint space (KIN J). This controller sends joint positions $\mathbf{q}$ to the skin cells with a rate of $f_q = 62.5$ Hz and receives skin joint torques $\boldsymbol{\tau}_{\text{skin,dstb}}$. The reactive skin controller supports two modes:

1) use $\boldsymbol{\tau}_{\text{skin,dstb}}$ and compute $\boldsymbol{\tau}_{\text{skin}}$ in parallel
2) use $\boldsymbol{\tau}_{\text{skin,dstb}}$

The first mode is used to compare the distributedly computed skin joint torques $\boldsymbol{\tau}_{\text{skin,dstb}}$ with the skin joint torques $\boldsymbol{\tau}_{\text{skin}}$, which are computed in the real-time control loop. We use the second mode to evaluate the reduction of delay in the real-time loop when $\boldsymbol{\tau}_{\text{skin}}$ is not computed. In both modes, we use

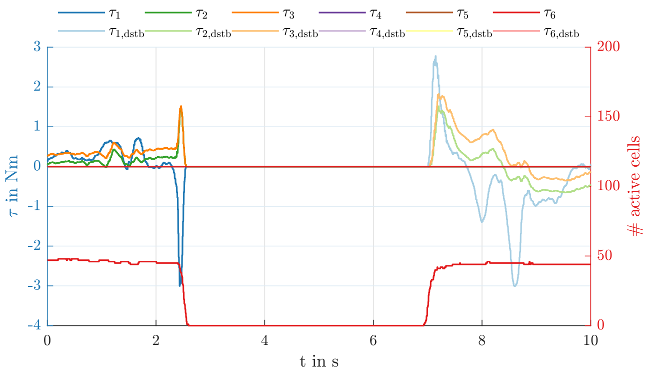


Fig. 6. Comparison of the skin joint torques computed on the skin cells ($\boldsymbol{\tau}_{\rm dstb}$) and torques computed in real-time control loop ($\boldsymbol{\tau}$); the torques are almost identical; the mean absolute error of $\boldsymbol{\tau}_{\rm dstb}$ with respect to $\boldsymbol{\tau}$ is 0.0054 Nm and the error $\boldsymbol{\tau}_{\rm dstb,add}$ is 0.0356 Nm.

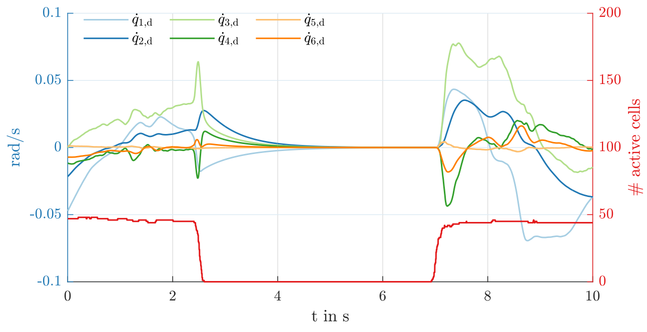


Fig. 7. The joint velocities resulting from the skin joint torque $\boldsymbol{\tau}_{\rm skin,dstb}$ computed on the skin cells and number of active cells $N_{\rm cell,act}$.

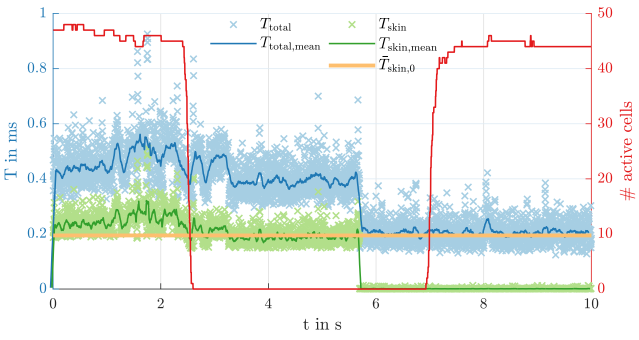


Fig. 8. The cycle times in ms when using the distributedly computed skin joint torque $\boldsymbol{\tau}_{\rm skin,dstb}$; $T_{\rm skin}$ is the delay in the real-time control loop when computing the skin joint torque $\boldsymbol{\tau}_{\rm skin}$; the cycle time $T_{\rm total}$ of the controller drops when switching to the distributed computation in the skin cells; $\bar{T}_{\rm skin,0}$ is the average time for computing $\boldsymbol{\tau}_{\rm skin}$ when the number of active cells is zero (for a total number of 253 cells).

$\boldsymbol{\tau}_{\rm skin,dstb}$ to control the robot. To test this controller during the experimentations, we kinesthetically move the robot arm in its workspace.

B. Accuracy of skin joint torque

Figure 6 compares the skin joint torques $\boldsymbol{\tau}_{\rm skin} = (\tau_l)$ computed by the PC with the skin joint torques $\boldsymbol{\tau}_{\rm skin,dstb} = (\tau_{l,dstb})$ computed distributedly by the skin cells. The resulting reaction of the skin joint torque controller can be seen in Figure 7. Up to the time $t = 5.7$ s the controller computes $\boldsymbol{\tau}_{\rm skin}$ and receives $\boldsymbol{\tau}_{\rm skin,dstb}$. Visually there is no difference between these two torques. The average error for

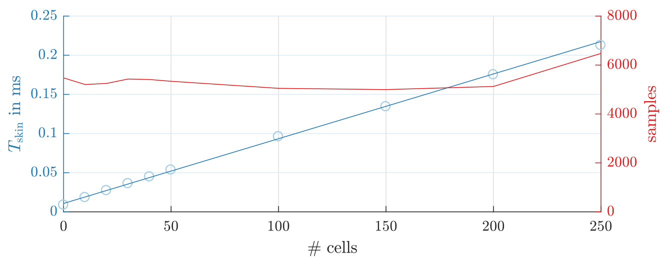


Fig. 9. The cycle time $T_{\rm skin}$ of the reactive skin controller computing $\boldsymbol{\tau}_{\rm skin}$ in the case that the number of active cells is zero ($N_{\rm cell,act} = 0$).

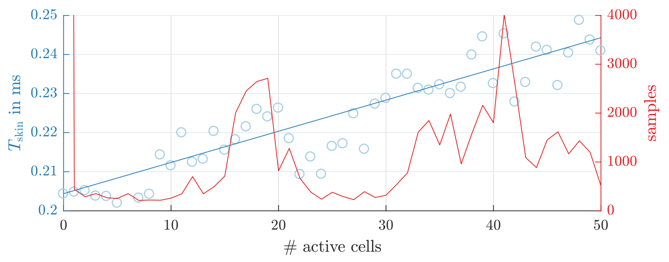


Fig. 10. The dependency between number of active cells and cycle time for skin joint torques computed by the controller for a total of 253 skin cells; the linear approximation takes the number of samples as weight.

all experiments is 0.0054 Nm for non-summed up $\boldsymbol{\tau}_i$ in $Q3.12$ and 0.0356 Nm for summed up $\boldsymbol{\tau}_i$ in $Q7.8$.

C. Computational cost in the real-time control loop

Figure 8 shows the total cycle time $T_{\rm total}$ of the real-time control loop. The control loop runs at 500 Hz [2], therefore, if the total cycle time crosses the border of 2 ms, then the controller breaks its real-time constraint. The cycle time of the reactive skin controller $T_{\rm skin}$ is part of the total cycle time $T_{\rm total}$ and $\bar{T}_{\rm skin,0}$ is the average of $T_{\rm skin}$ under the condition that the number of active cells $N_{\rm cell,act}$ is zero. The drop of both, $T_{\rm total}$ and $T_{\rm skin}$ at $t = 5.7s$ clearly indicates the switching off of the computation of $\boldsymbol{\tau}_{\rm skin}$ in the real-time control loop. $T_{\rm skin}$ drops to zero and the total cycle time reduces to more or less 0.2 ms. Most notably $T_{\rm total}$ is now independent to the number of activated cells $N_{\rm cell,act}$, which has been not possible if the computing of $\boldsymbol{\tau}_{\rm skin}$ is within the control loop. This is because, whenever the $\boldsymbol{\tau}_{\rm skin}$ is computed within the control loop, $T_{\rm skin}$ will depend on the number of cells $N_{\rm cell}$ and also on the number of active cells $N_{\rm cell,act}$.

D. Relationship between number of cells and cycle time

The cycle time $T_{\rm skin}$ of a reactive skin controller that computes $\boldsymbol{\tau}_{\rm skin}$ in the real-time loop depends on the total number of cells $N_{\rm cell}$. Even if there are no active cells, $T_{\rm skin}$ increases linearly with the total number of cells $N_{\rm cell}$ (see Fig. 9). The linear approximation for this relationship is

$$T_{\rm skin}(N_{\rm cell}) = 0.827 \cdot 10^{-6} N_{\rm cell} + 0.01073 \cdot 10^{-3} \quad (29)$$

[2]The closed loop dynamics use a virtual dynamic model which requires a fast update frequency, in this case 500 Hz. We noticed that slower frequencies lead to instabilities.

with a fitting accuracy of 99.95%. The linear approximation is weighted with the number of samples. The dependency of T_{skin} on the number of cells is the result of traversing the sensor values of skin cells in memory in order to detect active cells. For a control rate of 500 Hz with a cycle time of 0.2 ms for the rest of the controller (see Fig. 8), the theoretical limit for the total number of skin cells is 2160 cells. However, activating even only one of these cells will result in breaking the real-time constraint for 500 Hz.

E. Relationship between number of active cells and cycle time

The cycle time T_{skin} of a reactive skin controller that computes $\boldsymbol{\tau}_{\text{skin}}$ in the real-time loop also depends on the number of active cells $N_{\text{cell,act}}$ (see Fig. 8). The relationship between $N_{\text{cell,act}}$ and T_{skin} can be seen in more detail in Fig. 10. The linear approximation, using the number of samples as weight, is

$$T_{\text{skin}}(N_{\text{cell,act}}) = 0.798 \cdot 10^{-6} N_{\text{cell,act}} + 0.2 \cdot 10^{-3} \quad (30)$$

with a fitting accuracy of 93.9%. This results finally in

$$\begin{aligned} T_{\text{skin}}(N_{\text{cell,act}}, N_{\text{cell}}) = {} & 0.798 \cdot 10^{-6} N_{\text{cell,act}} \\ & + 0.827 \cdot 10^{-6} N_{\text{cell}} \\ & + 0.01073 \cdot 10^{-3} \end{aligned} \quad (31)$$

Under the assumption that the maximum number of active cells is $\max(N_{\text{cell,act}}) \leq 100$, the limit for the total number of cells when computing $\boldsymbol{\tau}_{\text{skin}}$ in the control loop, is $N_{\text{cell}} = 2067$ cells.

IV. CONCLUSIONS

This paper presented that the distributed computation of skin joint torques at the skin cells is efficient, feasible, accurate and applicable on a real robotic system. The error induced by the fixed-point computation of the skin torques is negligible. Our new approach is not limited to a specific robot and the complex skin joint torque computation no longer needs to take place within the real-time control loop. This removes the delay of computing skin joint torques in the control loop such that a controller no longer sets any upper bounds for the number of skin cells. Our approach enables large scale skin without the need to relax/break real-time in the control loop and removes any limits for the maximum number of skin cells in control.

REFERENCES

[1] R. D. Schraft and C. Meyer, "The need for an intuitive teaching method for small and medium enterprises," *VDI BERICHTE*, vol. 1956, 2006.

[2] T. Lens, J. Kunz, O. Von Stryk, C. Trommer, and A. Karguth, "Biorob-arm: A quickly deployable and intrinsically safe, light-weight robot arm for service robotics applications," in *German Conference on Robotics (ROBOTIK)*, 2010.

[3] E. Dean-Leon, K. Ramirez-Amaro, F. Bergner, I. Dianov, and G. Cheng, "Integration of robotic technologies for rapidly deployable robots," *IEEE Transactions on Industrial Informatics*, 2017.

[4] G. Grunwald, G. Schreiber, A. Albu-Schäffer, and G. Hirzinger, "Programming by touch: The different way of human-robot interaction," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 4, pp. 659–666, 2003.

[5] D. Massa, M. Callegari, and C. Cristalli, "Manual guidance for industrial robot programming," *Industrial Robot*, vol. 42, no. 5, pp. 457–465, 2015.

[6] T. Wösch and W. Feiten, "Reactive motion control for human-robot tactile interaction," in *International Conference on Robotics and Automation (ICRA)*, vol. 4, 2002, pp. 3807–3812.

[7] F. Nori, S. Traversaro, J. Eljaik, F. Romano, A. D. Prete, and D. Pucci, "icub whole-body control through force regulation on rigid non-coplanar contacts," *Frontiers in Robotics and AI*, vol. 2, no. 6, pp. 1–18, 2015.

[8] P. Mittendorfer, E. Yoshida, and G. Cheng, "Realizing whole-body tactile interactions with a self-organizing, multi-modal artificial skin on a humanoid robot," *Advanced Robotics*, vol. 29, no. 1, pp. 51–67, 2015.

[9] E. Dean-Leon, F. Bergner, K. Ramirez-Amaro, and G. Cheng, "From multi-modal tactile signals to a compliant control," in *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 892–898.

[10] L. D. Harmon, "Automated tactile sensing," *The International Journal of Robotics Research*, vol. 1, no. 2, pp. 3–32, 1982.

[11] G. Cannata, M. Maggiali, G. Metta, and G. Sandini, "An embedded artificial skin for humanoid robots," in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. IEEE, 2008, pp. 434–438.

[12] M. Strohmayr and D. Schneider, "The dlr artificial skin step i: Uniting sensitivity and collision tolerance," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 1012–1018.

[13] R. Bosch, *APAS Intelligent Systems for Man-Machine Collaboration*, Robert Bosch GmbH, Postfach 30 02 20, 70442 Stuttgart, Germany, 2016.

[14] M. Fritzsche, N. Elkmann, and E. Schulenburg, "Tactile sensing: A key technology for safe physical human robot interaction," in *Proceedings of the 6th International Conference on Human-robot Interaction*, 2011, pp. 139–140.

[15] S. Caviglia, L. Pinna, and C. Bartolozzi, "An event-driven posfet taxel for sustained and transient sensing," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 349–352.

[16] W. W. Lee, S. L. Kukreja, and N. V. Thakor, "A kilohertz kilotaxel tactile sensor array for investigating spatiotemporal features in neuromorphic touch," in *Biomedical Circuits and Systems Conference (BioCAS)*, 2015, pp. 1–4.

[17] F. Bergner, P. Mittendorfer, E. Dean-Leon, and G. Cheng, "Event-based signaling for reducing required data rates and processing power in a large-scale artificial robotic skin," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 2124–2129.

[18] F. Bergner, E. Dean-Leon, and G. Cheng, "Event-based signaling for large-scale artificial robotic skin - realization and performance evaluation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4918–4924.

[19] C. Bartolozzi, P. M. Ros, F. Diotalevi, N. Jamali, L. Natale, M. Crepaldi, and D. Demarchi, "Event-driven encoding of off-the-shelf tactile sensors for compression and latency optimisation for robotic skin," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 166–173.

[20] F. Bergner, E. Dean-Leon, and G. Cheng, "Efficient event-driven reactive control for large scale robot skin," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 394–400.

[21] R. Wakatabe, Y. Kuniyoshi, and G. Cheng, "O (logn) algorithm for forward kinematics under asynchronous sensory input," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2502–2507.

[22] E. Dean-Leon, B. Pierce, P. Mittendorfer, F. Bergner, K. Ramirez-Amaro, W. Burger, and G. Cheng, "Tomm: Tactile omnidirectional mobile manipulator," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2441–2447.

[23] P. Mittendorfer and G. Cheng, "3d surface reconstruction for robotic body parts with artificial skins," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 4505–4510.

[24] D. Timmermann, H. Hahn, and B. J. Hosticka, "Low latency time cordic algorithms," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 1010–1015, 1992.