Available online at www.sciencedirect.com

ScienceDirect

SoftwareX 3–4 (2015) 6–12

SoftwareX

www.elsevier.com/locate/softx

# Skinware 2.0: A real-time middleware for robot skin

S. Youssefi*, S. Denei, F. Mastrogiovanni, G. Cannata

*Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, Via Opera Pia 13, 16145, Genoa, Italy*

Received 4 June 2015; received in revised form 24 September 2015; accepted 30 September 2015

**Abstract**

Robot skins have emerged recently as products of research from various institutes worldwide. Each robot skin is designed with different applications in mind. As a result, they differ in many aspects from transduction technology and structure to communication protocols and timing requirements. These differences create a barrier for researchers interested in developing tactile processing algorithms for robots using the *sense of touch*; supporting multiple robot skin technologies is non-trivial and committing to a single technology is not as useful, especially as the field is still in its infancy. The Skinware middleware has been created to mitigate these issues by providing abstractions and real-time acquisition mechanisms. This article describes the second revision of Skinware, discussing the differences with respect to the first version.


*Keywords:* Robot skin; Tactile; Middleware; Skinware; Software; Framework

## Code metadata

| | |
|---|---|
| Current code version | Git tag `v2.0.0` |
| Permanent link to code/repository used for this code version | https://github.com/ElsevierSoftwareX/SOFTX-D-15-00026 |
| Legal Code License | GPLv2.0+ |
| Code versioning system used | git |
| Software code languages, tools, and services used | C, C++, Python |
| Compilation requirements, operating environments & dependencies | Linux, gcc, autotools |
| If available Link to developer documentation/manual | https://github.com/maclab/skinware/tree/v2.0.0/doc and https://github.com/maclab/skinware/wiki |
| Support email for questions | https://github.com/maclab/skinware/issues, https://groups.google.com/d/forum/skinware and info@cyskin.com |

## 1. Motivation and significance

A wide variety of tactile sensors have been subject of active research for more than three decades [1,2]. Deploying such sensors on a large robot surface, in the form of a robot skin, however, has only recently been the subject of considerable research [3]. The majority of the effort in this direction addresses the various technological issues that arise when realizing such a system: issues such as scalability [4], conformance to curved surfaces [5] and communication networks [6] among others. Nevertheless, many examples of robot skin systems have been presented such as those in [7,8,5,9–12].

Each robot skin technology, often independently investigated as part of ongoing research efforts by laboratories worldwide, is realized with a different structure, uses a different communication network and provides data with different timing and significance [3]. In Fig. 1, three such technologies are visible. These differences manifest themselves at the presentation level in the form of various software APIs with conflicting semantics. As a result, algorithms developed for any particular robot skin

* Corresponding author.
*E-mail addresses:* shahbaz.youssefi@unige.it (S. Youssefi), simone.denei@unige.it (S. Denei), fulvio.mastrogiovanni@unige.it (F. Mastrogiovanni), giorgio.cannata@unige.it (G. Cannata).

http://dx.doi.org/10.1016/j.softx.2015.09.001

Fig. 1. Different robot skin technologies in [14] (left), [12] (middle) and [8] (right). In the left image, capacitive transducers are used to provide tactile feedback, and are placed on flexible PCB. In the middle image, rigid but small multi-modal modules sense touch, temperature and vibration. In the right image, touch sensors are placed on foldable modules.

technology are potentially difficult to translate to other robot skin technologies. Even more difficult is the implementation of those algorithms.

It is therefore necessary to be able to acquire robot skin data and present them with abstract semantics, independently of both the particular technological and system-level solutions. This can be realized through a middleware. One such middleware, and the only one to the best of the authors' knowledge, is Skinware [13]. This article introduces Skinware 2.0 with significant improvements to the software architecture. However, much of the terminology and algorithms are adopted from Skinware 1.0. The reader is referred to the work in [13] for a detailed description of such concepts as *sensor*, *module*, *patch*, *sensor type*, *sensor layer*, *region*, *writer*, *reader*, *driver*, *service* and *user*.

Skinware 2.0, similar to the previous version, provides real-time robot skin data acquisition and presents the data from heterogeneous technologies to user applications uniformly and through abstract structures. This middleware is tailored specifically for robot skin technologies, while general purpose robotic middleware such as OROCOS [15], YARP [16] or ROS [17] could use Skinware and propagate the robot skin data to other respective modules in the network.

This article highlights the improvements to Skinware 1.0 and discusses their effect on Skinware's performance as well as how they facilitate development of tactile-based robot applications. Section 2 describes the architecture of Skinware 2.0 and its functionalities. Illustrative examples in Section 3 clarify the scenarios in which Skinware is useful. Section 4 discusses the aforementioned improvements and their impact, along with experimental tests to verify these improvements.

## 2. Software description

### 2.1. Terminology

As with Skinware 1.0 [13], the following entities are present in Skinware 2.0.

- *Sensor*. A sensor is a sensing element of the robot skin, e.g., it may be a tactile, temperature or vibration transducer [12].
- *Module*. A set of physically close sensors, often viewed as the building block of the robot skin. Non-modular robot skins can be viewed as having one module per sensor.
- *Patch*. A patch of the robot skin is a set of modules often controlled by the same microcontroller. Patches are interesting to hardware specific tasks, e.g., to monitor malfunctions.
- *Sensor type*. Knowing the types of the sensors, the applications can better interpret their values based on the semantics associated with each sensor type.
- *Driver*. A driver is a program that knows the specific details of the robot skin hardware, and is responsible for acquiring skin data and providing them to Skinware. With drivers, Skinware separates the hardware-specific acquisition tasks and the higher-level data propagation.
- *User*. A user is any application that receives robot skin data from Skinware.
- *Service*. A service is a part of a program that provides processed data to other applications. With services, Skinware reduces duplication of code, execution time and memory usage of applications by allowing them to share their results.
- *Writer*. A real-time thread responsible for writing data to internal Skinware buffers.
- *Reader*. A real-time thread that synchronizes with a corresponding writer thread to acquire coherent data from shared buffers.

Skinware 1.0 included the concepts of *sensor layers* and *regions*. Sensor layers are (possibly virtual) divisions of skin sensors based on their type, which allows an application to selectively acquire data from sensors of interest. Robot skin regions are sets of sensors that represent various areas of interest on the robot body, such as the *palm* or *forearm*. On the contrary, Skinware 2.0 does not enforce such divisions of the robot skin, as explained in the following.

With regards to sensor layers, experience has shown that this artificial division of sensor data unnecessarily increases the complexity of both the driver and the user programs. The driver becomes more complicated as it is required to provide data from various sensor types that are often received collectively, through unrelated writer threads. The user applications that require to interpolate those data also become more complicated as they receive them through unrelated reader threads. Skinware 2.0 thus removes any notion of layers, allowing a driver to provide

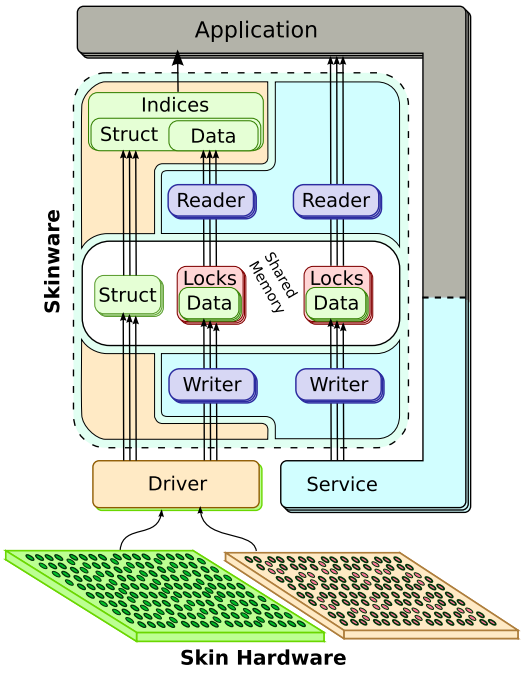


Fig. 2. The layered architecture of Skinware 2.0. The service layer provides means for real-time transfer of arbitrary data. The driver layer provides the structure of the robot skin while using the service layer to transfer sensor data. In this figure, *services* are provided through *writer* threads, and their *data* are retrieved by *reader* threads, synchronizing with the writer threads with *locks*. *Drivers* use services to transfer dynamic data while the *structure* of the robot skin is shared with *applications* separately. Skinware in the application side, creates *indices* over the complete robot skin structure for efficient access to data.

data from all sensor types it handles through the same writer thread.

While the idea of robot skin regions of interest are useful to certain applications, particularly higher level behavioral algorithms, they are not necessarily interesting for all user applications. Imposing this structure on the robot skin does not introduce performance penalties [13], but nevertheless increases the complexity of initialization. Therefore, the concept of regions is removed from the core of Skinware 2.0, although an organization in regions is still possible through services.

The main motivation for the change in the architecture of Skinware is that drivers and services are similar in many ways; they both provide data through buffers, taking care of synchronization. Skinware 1.0 had separate implementations for drivers and services, with a simpler implementation for services that was overlooked in [13]. Skinware 2.0 implements services as the core of communication, and implements drivers as specialized services. As a result, services are greatly empowered, amplifying their benefits. Furthermore, the mechanism that allows services to be introduced post-initialization now enables the same for the drivers, allowing a sort of hot-plugging for robot skins.

### 2.2. Skinware architecture

*Remark:* this section solely presents modifications to the architecture of Skinware 1.0. The arguments regarding the decisions made for other aspects of the architecture as well as the benefits of those decisions remain sound as discussed in [13].

Skinware 2.0 has a layered architecture (Fig. 2). At its core, Skinware provides a real-time inter-process data transfer mechanism, by means of writer and reader threads, shared memory and shared synchronization locks. The details of the data transfer mechanism have been discussed in [13]. This mechanism is encapsulated in Skinware services, and the service layer handles creation, removal, attachment to and detachment from named services.

Service writers can be either periodic or sporadic, i.e., they either provide data periodically or upon request by a reader. The service readers themselves are either periodic, sporadic or ASAP, similar to readers in Skinware 1.0. The algorithms for periodic writers and their corresponding readers have been presented in [13]. The algorithms for sporadic writers and their corresponding readers can be trivially derived and are not presented here for brevity.

Each driver of Skinware 2.0 contains a service that handles the task of transferring sensor measurements. The structure of the robot skin, which is fixed as long as the driver is attached to Skinware, is provided separately through shared memory. On the user application end, Skinware constructs a unified view of the robot skin, piecing together the structures from various drivers. As a result, a user application may inspect the whole robot skin without knowing exactly how it is divided among drivers, although that information is present for the interested user application. To allow for efficient traversal of sensors of a specific type, for example by a user application only interested in tactile sensors, an index over the sensors of each type is created.

In the architecture of Skinware 2.0, corresponding to each driver there is a service and thus a writer thread. This is in contrast with Skinware 1.0 where a driver could handle multiple sensor layers and consequently was assigned multiple writer threads. This greatly simplifies the task of a driver that acquires all sensor data from hardware collectively, as it does not need to separate those data and provide them through different writer threads. On the other hand, if the driver process acquires hardware data through separate means of communication, and thus would prefer to provide them through multiple writer threads, it is easily able to do so by providing multiple drivers to Skinware.

Similar to Skinware 1.0, for each user, one reader is created corresponding to each writer and takes upon the task of synchronization with the writer to acquire data from multiple shared buffers, possibly at a different rate from the writer. With readers corresponding to driver writers, Skinware by default copies sensor data in the user application's local replica of the robot skin. With all readers, Skinware allows the user application to inspect and possibly copy the data if necessary.

### 2.3. Software functionalities

Skinware, as a software library, provides the means for real-time inter-process data transfer. This functionality is

primarily used for periodically transferring sensor data to user applications, in a concurrent, consistent and coherent manner. It is also used for sharing processed data among those user applications. In this section as well, only the differences in functionality with respect to Skinware 1.0 are presented.

In contrast to Skinware 1.0 that was developed for RTAI [18], Skinware 2.0 uses an abstraction layer over real-time systems (URT: Unified Real-Time interface[1]) to increase its portability and facilitate development and testing. For example, a Skinware driver could be implemented and tested using the POSIX back-end of URT and not in real-time, for example under Linux, where debugging is easy and tools such as `valgrind`[2] are available. Once developed, the driver could be recompiled and used with the RTAI back-end of URT for real-time behavior.

Skinware 1.0 had a static view of the attached robot skin. While drivers were allowed to detach from and reattach to Skinware at runtime, they were required to be representing the same piece of robot skin. This property holds for many robots, where the robot skin is static, but it does not necessarily hold for more modern robot skins. It is quite possible for a skin module to adapt the number and precision of its sensors based on such phenomena as *focus*, i.e., reducing the number of sensors for energy saving when there is no activity in that area of robot skin, and increasing again once activity is detected, similar to biological systems [19]. Such behavior would have been impossible, or cumbersome at best, to achieve with Skinware 1.0. With Skinware 2.0, it is possible for a driver to present a different robot skin upon reattachment, given that all users have detached from it. Once done, users may reattach to the driver. With the static view of skin as in Skinware 1.0, hot-plugging would have also been impossible.

Spatial calibration, the act of identifying the precise positions and orientation of each sensor [20], is removed from the core of Skinware 2.0 and changed to a service. In Skinware 1.0, spatial calibration was an initialization step, which contributed to the rigidity of Skinware's view of the skin. Furthermore, while most applications would benefit from knowing the precise position and orientation of every sensor, this may not necessarily be true for all applications. For example, a force control application reading sensor data from the finger tips may only be interested in the average sensor value to control the finger motors, rather than the precise force vectors derived from the sensors' poses. Other examples include the calibration software itself, hardware configuration and quality inspection tools as well as machine learning algorithms [21]. A high level behavioral algorithm may also be interested solely in regions of the robot skin, rather than the detailed sensor locations.

## 3. Illustrative examples

Three target Skinware user classes are considered; end-users, creators and algorithm-developers of robot skins.

Research teams interested in using available robot skin technologies, evaluate and identify the technology that fulfills the requirements of the tasks at hand. It is possible that multiple technologies are identified, each suitable for a particular area on the robot body. Using Skinware, the research team is able to select the appropriate robot skin technologies without such considerations as software libraries and interoperability of components. As a result, there would be no compromises due to availability of software for any particular robot skin technology.

For the creators of a new robot skin technology, a typical scenario is as follows. The research team that develops a new robot skin technology has expert knowledge of all peculiarities regarding the communication network and data acquisition from their robot skin. For this team, it is enough to provide a Skinware driver and immediately all software and processing libraries written for other robot skin technologies through Skinware, would become available to them as well.

The developers of tactile-based robot behaviors are able to provide their software and processing libraries to a wide variety of robot skin technologies through Skinware. Furthermore, they develop algorithms with an abstract view of the robot skin, without considerations for peculiarities of the hardware implementation.

## 4. Impact

From a researcher's point of view. The use of Skinware has multiple advantages. First, the development of robot skin requires a heterogeneous design expertise involving computer and electronics engineering, material science and systems engineering: Skinware allows for abstracting from the technological peculiarities associated with all of these fields. Then, Skinware enables researchers to focus on specific algorithmic aspects, therefore enforcing new research activities: as soon as new technologies are available, they become immediately available, as long as Skinware supports them.

Compared with Skinware 1.0, Skinware 2.0 improves the quality of developed programs through a cleaner API and facilitates software management by providing loosely coupled modules. Furthermore, the view of the robot skin is more dynamic in Skinware 2.0, allowing online restructuring, addition and removal of robot skin patches, which significantly eases debugging and identifying malfunctioning sensors.

Skinware is a new software framework that is gaining traction as do robot skin technologies. As such, it is difficult to predict how exactly Skinware would be able to fulfill its role. It has nevertheless been extensively used internally by the authors during the development and deployment of the revisions of their robot skin system[3] to great success, and adopted in the European FP7 projects ROBOSKIN and CloPeMa.[4]

In the following paragraphs, the results of certain performance tests on Skinware 2.0 are presented to further emphasize its useability in diverse configurations.

[1] Unified Real-Time interface is available at https://github.com/ShabbyX/URT. At the time of this writing, URT supports POSIX systems as well as RTAI (both in user and kernel spaces).

[2] http://www.valgrind.org/.

[3] CySkin: http://www.cyskin.com.

[4] See acknowledgments.

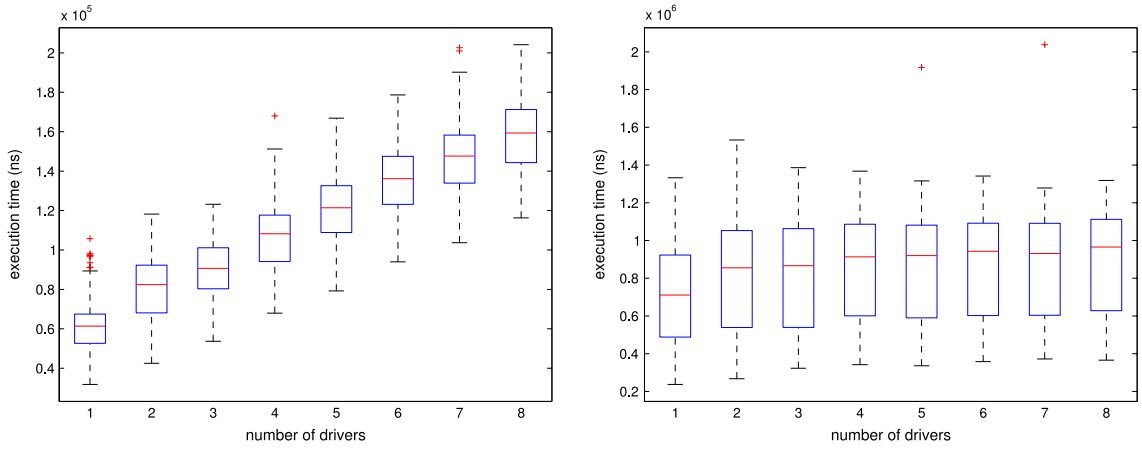


Fig. 3. Worst-case total execution time of writers (left) and readers (right) with number of drivers in the range 1–8.

Experimental tests on the implementation of Skinware 2.0 have been performed in a way similar to those discussed in [13] on the same workstation, but with upgraded software.

- Operating System: Ubuntu 14.04 using the Linux kernel 3.8.1 patched with modified RTAI 4.0,[5]
- Processor: Intel Core 2 Duo E8200 @2.66 GHz,
- Motherboard: FUJITSU SIEMENS D2581-A1,
- RAM: Four 1 GB 48 spaces DIMM DDR2 Synchronous 667 MHz,
- Compiler: gcc and g++ version 4.8.2 with -O2 optimization level.

The tests performed on Skinware 2.0 focus on the behavior of Skinware under heavy load and show the impact of number of drivers, number of user applications and number of communication buffers on the total execution time of writer and reader threads. In these tests, data from 30,848 sensors are acquired corresponding to $1m^2$ of robot skin with the ROBOSKIN technology [14], using virtual drivers similar to what those discussed in [13]. As the writer and reader threads operate on the sensor data only, details of the sensor structuring such as patches and modules, or other properties such as the sensor types are irrelevant to the performance of these threads.

The whole robot skin with 30,848 sensors is equally divided among a configurable number of virtual drivers in the range 1–8, and the data transfer is done through a configurable number of buffers in the range 2–8. In each configuration, measurements have been performed in the presence of a number of user applications in the range 1–15. The thread worst execution times have been sampled after a 60 s execution period. The test for each configuration has been performed 4 times, resulting in a total of 3360 tests. In each test, writer threads are created in periodic mode with a random period between 30 and 50 Hz and reader threads are created in periodic mode with a random period between 10 and 100 Hz. This implies that the worst thread execution times in each test configuration are themselves values calculated over 1800–3000 sample points for writer threads and 600–6000 sample points for reader threads.

*Remark:* as in [13], acquisition rates have been chosen randomly to test Skinware under unlikely conditions and edge cases as well as increase the chances for swap skips. In practice, where the task frequencies of writers and readers are harmonious, Skinware could only be more performant. The definition of swap skips, where a writer is unable to swap its working buffer, and the algorithms used to prevent such a situation are present in [13]. In all the tests discussed in this section, the measured number of swap skips have been a constant zero.

### 4.1. Effects of drivers

In this analysis, the effect of distributing robot skin sensors over a number of drivers is studied. Fig. 3 shows the box plots of the worst-case execution times of the writer and the reader threads, in presence of a varying number of drivers. Each entry in these box plots corresponds to measurements for varying numbers of user applications and communication buffers.

It can be observed that increasing the number of drivers increases the worst-case execution time of writers and readers. With a higher number of drivers, there is a higher number of pairs of writers and readers. As a result, the cost of synchronization is multiplied.

### 4.2. Effects of user applications

The effect of increasing the number of user applications can be viewed in Fig. 4 for the writer and the reader threads. In this figure, the box plots of the worst-case execution times of these threads in the presence of a varying number of user applications are presented. Measurements for varying numbers of drivers and communication buffers are present in each entry of these box plots.

It is apparent that the execution time of both the writer and the reader threads grow slowly with the number of user applications. A higher number of threads results in a greater cost of scheduler context switches. This affects the writer and the reader threads differently because writer threads have higher priority and are smaller in number with respect to readers. (With

[5] One such modification is to extend RTAI with the ability to measure precise execution times. The modified RTAI with which the tests have been performed can be found at https://github.com/ShabbyX/RTAI/.

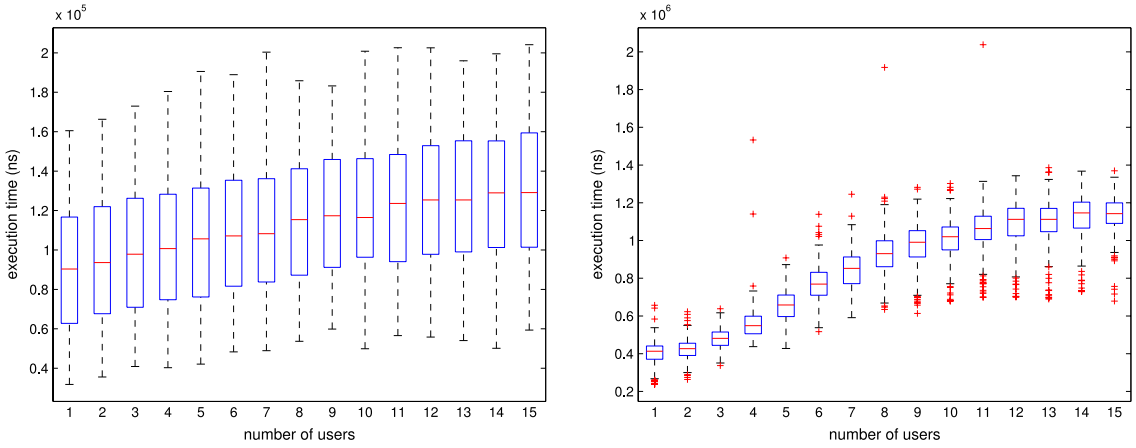


Fig. 4. Worst-case total execution time of writers (left) and readers (right) with number of user applications in the range 1–15.

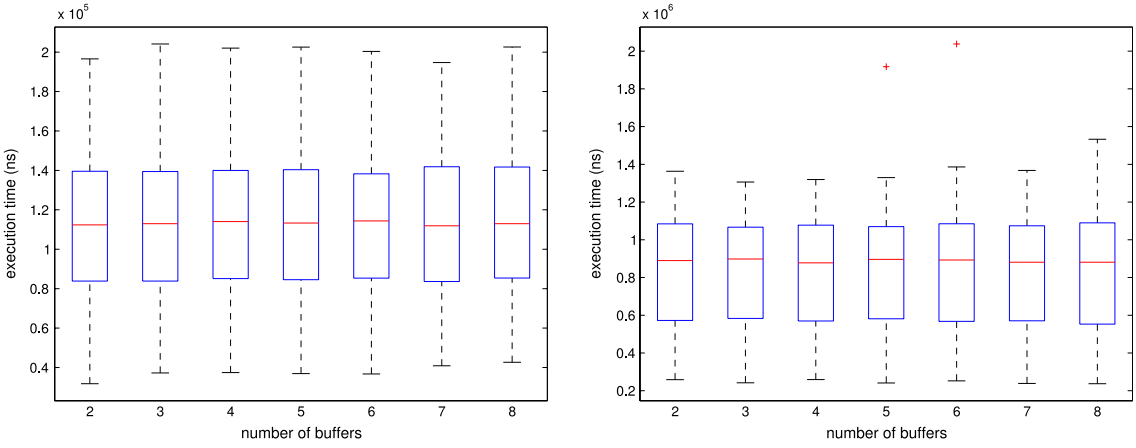


Fig. 5. Worst-case total execution time of writers (left) and readers (right) with number of communication buffers in the range 2–8.

$D$ drivers and $U$ users, there are $D$ writer threads and $U \times D$ reader threads.)

### 4.3. Effects of communication buffers

Finally, the effect of the number of communication buffers on the performance of Skinware is demonstrated. In Fig. 5, this effect is shown for the writer and the reader threads, with box plots of the worst-case execution times of these threads against the number of communication buffers. Each entry of these box plots corresponds to measurements for varying numbers of drivers and user applications.

It is observed that changes to the number of communication buffers does not particularly affect the worst-case performance of either writers or readers. For writer threads, a higher number of buffers means a higher chance to find a free buffer to swap. However, the execution time of the writer is unaffected because it consists mainly of *sleeping*, i.e., it does not contribute to the measurement of the *execution* time. For reader threads, they wait on the buffer being currently filled, regardless of how many other buffers are present.

It can be concluded from the results of these tests that heterogeneous robot skin technologies can be handled by Skinware through different drivers albeit with minor performance penalty. Users in Skinware can grow as necessary without a considerable penalty as well. Finally, the number of buffers may be increased with no penalty to eliminate the remotest chances of swap skips from happening.

## 5. Conclusions

Through Skinware, developed behaviors based on robot skins are made portable to all current and (in principle) future robot skin technologies. This allows new robot skins to thrive and be benchmarked against well-known robot skins, at the same time allowing researchers to move from one robot skin technology to a more suitable one, without concerns for software compatibility. Skinware 2.0 improves upon the first revision by reducing complexity of the already simple drivers, empowering services, providing a cleaner API and being considerably more portable.

## Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under Grant 231500 (project ROBOSKIN) and Grant 288553 (project CloPeMa).

## References


[1] Lee M, Nicholls H. Tactile sensing for mechatronics—a state of the art survey. Mechatronics 1999;9:1–31.

[2] Lumelsky V, Shur M, Wagner S. Sensitive skin. IEEE Sens J 2001;1: 41–51.

[3] Dahiya R, Mittendorfer P, Valle M, Cheng G, Lumelsky V. Directions toward effective utilization of tactile skin: A review. IEEE Sens J 2013; 13:4121–38.

[4] Schmitz A, Maiolino P, Maggial M, Natale L, Cannata G, Metta G. Methods and technologies for the implementation of large-scale robot tactile sensors. IEEE Trans Robot 2011;27:389–400.

[5] G Cannata, M Maggiali, G Metta, G Sandini, An embedded artificial skin for humanoid robots. In: Multisensor fusion and integration for intelligent systems. MFI 2008. IEEE international conference on. 2008. p. 434.

[6] E Baglini, S Youssefi, F Mastrogiovanni, G Cannata, A real-time distributed architecture for large-scale tactile sensing. In: Intelligent robots and systems, 2014 IEEE/RSJ international conference on. 2014. p. 1663–9.

[7] O Kerpa, K Weiss, H Worn, Development of a flexible tactile sensor system for a humanoid robot. In: Intelligent robots and systems. Proceedings. 2003 IEEE/RSJ international conference on, vol. 1. 2003. p. 1.

[8] Y Ohmura, Y Kuniyoshi, A Nagakubo, Conformable and scalable tactile sensor skin for curved surfaces. In: Robotics and automation. Proceedings 2006 IEEE international conference on. 2006. p. 1348.

[9] J Ulmen, M Cutkosky, A robust, low-cost and low-noise artificial skin for human-friendly robots. In: Robotics and automation. 2010 IEEE international conference on. 2010. p. 4836–41.

[10] E Baglini, G Gannata, F Mastrogiovanni, Design of an embedded networking infrastructure for whole-body tactile sensing in humanoid robots. In: Humanoid robots (Humanoids), 2010 10th IEEE-RAS international conference on. 2010. p. 671.

[11] D Tawil, D Rye, M Velonaki, Touch modality interpretation for an EIT-based sensitive skin. In: Proceedings of the 2011 IEEE international conference on robotics and automation. (Shanghai, China); 2011.

[12] Mittendorfer P, Cheng G. Humanoid multimodal tactile-sensing modules. IEEE Trans Robot 2011;27:401–10.

[13] Youssefi S, Denei S, Mastrogiovanni F, Cannata G. A real-time data acquisition and processing framework for large-scale robot skin. Robot Auton Syst 2015;68:86–103.

[14] Billard A, Bonfiglio A, Cannata G, Cosseddu P, Dahl T, Dautenhahn K, F M, Metta G, Natale L, Robins B, Seminara L, Valle M. The ROBOSKIN project: Challenges and results, vol. 544. 2013. p. 351–8.

[15] H Bruyninckx, Open robot control software: the OROCOS project. In: Robotics and automation, 2001. Proceedings 2001 ICRA. IEEE international conference on, vol. 3. 2001. p. 2523–8.

[16] Metta G, Fitzpatrick P, Natale L. YARP: Yet another robotic platform. Int J Adv Robot Syst 2006;3.

[17] M Quigley, B Gerkey, K Conley, J Faust, T Foote, J Leibs, E Berger, R Wheeler, A Ng, ROS: an Open-source robot operating system. In: Proceedings of the IEEE international conference on robotics and automation (ICRA) workshop on open source robotics (Kobe, Japan); 2009.

[18] Mantegazza P, Dozio E, Papacharalambous S. RTAI: Real time application interface. Linux J 2000;2000.

[19] Frings C, Bader R, Spence C. Selection in touch: Negative priming with tactile stimuli. Percept & Psychophysics 2008;70:516–23.

[20] Denei S, Mastrogiovanni F, Cannata G. Towards the creation of tactile maps for robots and their use in robot contact motion control. Robot Auton Syst 2015;63:293–308.

[21] J N, P Byrnes-preston, R Salleh, C Sammut, Texture recognition by tactile sensing. In: Australasian conference on robotics and automation. (Sydney, N.S.W., Australia); 2009.