A Middleware for Whole Body Skin-like Tactile Systems

Shahbaz Youssefi, Simone Denei, Fulvio Mastrogiavanni, Giorgio Cannata

Abstract—In this article a software architecture for acquiring, processing and presenting distributed tactile information in real-time is presented. When humanoid robots are endowed with large-scale tactile systems, the need arises to design a framework able to abstract from the underlying robot hardware structure. The article describes relevant functional requirements, architectural choices and specific technical solutions. Experiments show results with respect to a realistic use case, i.e., a humanoid robot fully covered with artificial skin.

I. INTRODUCTION

During the past few years, research activities on human-robot interaction [1] stressed the need for endowing humanoid robots with large-scale tactile sensing capabilities [2]. When moving from few transducers to large-scale tactile arrays, novel issues at the architectural level arise: scalability [3], conformance [4], [5], wiring [6] and networking [7], just to name but few. Humanoid robots are complex systems characterized by different mechanical and computational components. When considering large-scale skin-like tactile systems, these components become intertwined. As a matter of fact, huge amounts of tactile information originating from distributed sources are useless unless the data are properly processed in time and interpreted with respect to well-defined contact models [8]. Usually, it is common to consider tactile data processing as a problem similar to that faced by such research fields as Vision. It turns out that these problems are definitely different:

- Since tactile sensors are distributed over robot body parts of differing shape and morphology, there is no equivalent of the concept of bitmap, which assumes a regular and well-defined 2D grid. On the contrary, the tactile image can be hardly structured [9], [10] in a regular arrangement.
- Since robot motion commands can lead to robot postures in contact with the surrounding environment, the actual tactile image depends on the particular robot-environment configuration [11]. Furthermore, such interaction phenomena as self-touch can be considered, where the tactile image strictly depends on the robot configuration only.
- Different robot body parts are characterized by different requirements in terms of density, resolution and sensitivity. Differently from a single bitmap, many tactile images (i.e., corresponding to different body parts or skin areas) must be considered.

II. TERMINOLOGY AND REQUIREMENTS

A. Sensor-Module-Patch-Region: a Terminology for Large-scale Tactile Sensing

This Section introduces the main terminology of concepts that are useful when discussing large-scale tactile sensing for robots. Robot skin can be thought as a distributed and
a hierarchical system where a number of distinct components can be identified (Figure 1).

**Sensor.** The single tactile element (i.e., a taxel) is associated with two pieces of information, namely the sensed information and its position with respect to robot or workspace centred reference frames [9]. Taxels exploit specific transduction principles converting physical quantities, such as pressure [5], [3], strain and temperature in electrical signals.

**Module.** The physical structure embedding a set of sensors as well as the associated readout and transmission electronics. Based on flexible Printed Circuit Boards (fPCBs), modules can be characterized by specific shapes optimized to better conform to body parts with varying and non uniform curvatures, such as triangular [5], [3] or H-like [4]. Furthermore, the number of hosted sensors can greatly vary according to the specific implementation, e.g., from 12 in the robot skin presented in [5], [3] to 32 in [4].

**Patch.** The set of modules that can be interconnected to each other and managed by a single local computational node. Usually, the node hosts a microcontroller responsible for the acquisition of sensory data from all the modules, for early data processing and for data transmission to a centralized embedded workstation running higher level robot tasks. It is necessary to associate modules to microcontrollers in order to optimize a number of criteria, such as the minimization of the number of local wires or the computational load among microcontrollers [19]. Analogously to modules, patches are typically interconnected through communication buses, such as CAN [5], SMBus [4] or EtherCAT [7].

**Region.** The set of patches covering a specific robot body part. Patches belonging to a region share the same robot or workspace centred reference frame. This is particularly relevant in humanoid robot structures where, as a matter of fact, a region can be related to a particular part of the kinematic chain, such as the left forearm or the torso. Specifically, it is convenient to assume the common reference frame as the one associated with the related joint.

This functional terminology is of utmost importance to define data structures and query algorithms in the envisaged framework for tactile data management.

### B. Functional Requirements for a Robot Skin Middleware

In order to manage large-scale tactile information originating from distributed areas of the robot body, system requirements, relevant data structures and algorithms for fast and dependable data access must be carefully designed. On the one hand, different from what happens in vision, where the concept of *bitmap* corresponds to a regular matrix of pixels, raw distributed tactile information is spatially unstructured (since taxels are located on robot surfaces of varied shape) and its semantics strictly depends on the adopted transduction mode. On the other hand, high-level cognitive tasks assume to access well-defined and well-organized data structures before any further processing can occur.

The goal of the proposed middleware is to off-load high-level computational processing from the need of defining *ad hoc* data structures and retrieval algorithms for distributed tactile elements, as well as from dealing with heterogeneous hardware and networking mechanisms involved in the robot skin infrastructure framework. Relevant functional requirements are described as follows.

**Real-time performance.** When considering human-robot or environment-robot interaction tasks involving tactile feedback, real-time performance is particularly important. Specifically, two different requirements are tightly connected to real-time issues, namely human (and robot) safety [20] and the ability to perform fine-grained tactile-based control movements (e.g., manipulation or object lifting [21]). In particular, the first requirement implies to provide guarantees on the upper bounds of (even simple) feedback-based robot reaction times, such as a safety stop. During the time in between a physical contact and a robot safety stop, many components of a robot software architecture are involved: sensing transduction and transmission to a centralized embedded workstation, accessing and processing tactile data, issue of the control strategy, transmission to actuators and actual execution. In order to minimize the time needed to access and process tactile data, the proposed software framework devotes particular care to the design and implementation of redundant, hierarchical, data structures and layered data access mechanisms.

**Data consistency among tasks.** If multiple concurrent tasks are responsible for executing algorithms on tactile data, consistency mechanisms of shared tactile data structures must be provided. As a matter of fact, it is expected that the detection of complex contact phenomena requires multiple concurrent tasks to process tactile information referred to the same stimulus with the aim of merging the results. If coherency is not guaranteed, different tasks could ground their inferences with respect to different tactile images, thereby invalidating the

---

**Fig. 1:** Top: A graphical representation of the reference terminology for robot skin. Bottom: A skin patch covering a generic sketched robot part.
whole result. The proposed software framework allows for a number of data access and locking mechanisms to enforce and guarantee consistency between concurrent user applications.

Data coherency. Given the intrinsically distributed nature of robot skin patches and regions, tactile information originating from different robot body parts may refer to tactile images as frames taken at different time instants. Whilst the analysis of local contact phenomena could – to a limited extent – make use of spatially limited data (i.e., within a single region), more complex tactile interaction may require to integrate information from even topologically distant regions. For instance, let us consider the analysis of two forearms in contact: tactile data from distant regions must be temporarily coherent to precisely detect mutual contacts. The designed middleware is responsible to provide high-level tasks with tactile data originated coherent within the single tactile frame.

Multiple data indices. In order to provide functional access to the whole robot skin and to enforce real-time performance, it is necessary to exploit the intrinsic morphology of humanoid robots to define specific indices on the tactile data structure. These are relevant both in a functional and a structural perspective. In the first case, indices on data structures allow to access tactile data abstracting from the actual skin hardware. For instance, relevant data access mechanisms that must be directly available include: (i) retrieve a tactile image corresponding to a certain area of interest (AoI) in the skin that is a user-defined, possibly overlapping, set of taxels; (ii) identify the area of interest a specific taxel belongs to; (iii) retrieve the set of neighbouring taxels for a specified taxel. In the second case, indices allowing direct access to the physical hierarchy of taxels, modules, patches and regions are necessary to implement ancillary – yet important – mechanisms, such as fault-tolerance, at the software level. Related mandatory data access mechanisms include: (i) query the status of single taxels belonging to a specified module; (ii) query the status of modules belonging to a specified patch; (iii) retrieve the spatially close modules of a specified module; (iv) retrieve neighbouring patches of a specified patch.

The discussed requirements are the basis used to design and implement the presented robot skin middleware.

III. TACTILE DATA MIDDLEWARE DESIGN

This Section first introduces a case study where a humanoid robot must be provided with large-scale tactile sensing capabilities. Then, the overall software framework as well as specific design choices are introduced, which are grounded with respect to the introduced scenario.

A. Working with tactile data: a case study

The considered case study includes a typically sized humanoid robot equipped with tactile sensors throughout its whole body (Figure 2). In this scenario, skin surface can sum up to 1m². Since, in principle, each robot body part is typically involved in specific tasks (e.g., hands are mostly related to grasping and manipulation, whereas arms in obstacle detection and avoidance), the related skin patch can be specifically structured with respect to, for instance, taxel density and pitch. Furthermore, varying computational requirements and bandwidth for distributed tactile information processing are needed for different body parts. For instance, such body parts as the torso or the back do not require high spatial tactile resolution: still, tactile frames need to be checked at the fastest possible rate to detect unexpected collisions with the environment.

To reflect these aspects, a software framework able to decouple (i) actual data acquisition from actual processing, and (ii) processing related to different robot body parts, is needed. Taking these issues into account, the proposed software framework has been designed to comply with a precise specification. The body of a humanoid robot covered with 1m² of skin can be subdivided, according to [22], in 9%, 16.5% and 26% for a limb, a leg and the torso respectively (Figure 2). Using the technology presented in [5] and assuming to have a constant spatial resolution on the whole body, the goal is to be able to process tactile images of

\[
\text{taxels} = \frac{1m^2}{\frac{3.89cm^2}{\text{module area}}} \times \frac{12}{\text{taxel per module}} \approx 30848
\]

taxels with a temporal resolution compliant with typical robot safety behaviours, such as reflexes [20] and safety stops.

B. Middleware Architecture and Design Choices

To meet the functional requirements introduced in Section II-B, a 2-layer architecture has been designed (see Figure 3). The bottom layer consists of different components, namely the writer service, a tactile data structure and a number of optional services.

The writer service is a hardware dependent component responsible for the real-time periodic acquisition of a tactile image from the underlying communication bus and for its storage within the data structure. As a matter of fact, data is a huge array of taxels. For each represented taxel, data stores a structure representing the current numerical value and the type of sensor.
Optional services can be assigned with tactile-based processing tasks. This mechanism is introduced to provide algorithms as common and shared services useful for any user level task. Each running service has reserved access to its memory space external to data. Since different services in the bottom layer can perform heterogeneous operations on the actual tactile image, different processed tactile images can be available at the same time. Furthermore, services can be combined in pipeline-like structures, therefore implementing complex cascades of tactile processing stages. In the current implementation, one service has been implemented, namely the propagation of active taxel data upwards in the AoI structure. This allows for a rapid identification of active skin areas. Therefore, the process of identifying active taxels is reduced to a search in the fewer regions rather than the numerous taxels. In Section IV, it is shown how this service significantly improves performance of such queries.

Analogously to the bottom layer, the top layer consists of different components, namely a reader service, a tactile data structure, a set of indices and a number of optional services.

The reader service is responsible for obtaining a local copy of data to work on. Since many top layers can be concurrently executed in a realistic robot architecture, this mechanism reflects the single writer – multiple readers nature of the data access problem. As it can be noticed from Figure 3, the copied tactile image is only part of the data structure maintained in each top layer: in principle, additional information for each taxel can be added, e.g., the 3D position with respect to a robot centred reference frame (i.e., dependent on the current robot configuration) and the 2D position with respect to a flattened representation of the skin surface [10], which is useful to ground high level tactile data processing and control tasks.

A similar mechanism holds for indices. The top layer data structure is initialized by copying from the bottom layer the required parts. However, each top layer can add new indices thereby imposing its own data structuring. In principle, the sensor–module–patch–region paradigm can be completely dismissed. In the current implementation, indices allow for three basic query mechanisms: the first based on the physical skin structure (i.e., based on modules, patches and regions); the second on the logical definition of AOs, as can be seen in Figure 4; the third is a 2D map representing the topographic arrangement of taxels.

Finally, each top layer can implement its own local optional services, which may or may not exploit additional services in the common bottom layer. It must be noticed that, in the current implementation, it is not mandatory for a reader to copy all the data, thus it can skip over not requested additional information provided by bottom layer’s services. It is up to the designer to decide when and how to access these information. Moreover, user-made services can be loaded or unloaded, even at runtime, and the reader is not aware of the services that are currently running in the architecture.

Whilst the bottom layer is unique, many different instances of the top layer (each one retrieving tactile data from the bottom layer) can be concurrently executed. The bottom layer can be considered as an interface to the actual skin underlying hardware. This is enforced by implementing it as a part of the operating system, allowing a direct communication with the
hardware and bypassing the operating system services. To this aim, in the current implementation on a Linux OS, the bottom layer is implemented in kernel space, whereas top layers are parts of applications in user space. The separation between these two layers is motivated by a number of reasons: (i) each top layer corresponds to a different robot subsystem, each one possibly processing tactile frames for a different goal; (ii) it enforces decoupling between the bottom layer and user level applications by allowing the respective top layers to communicate with the unique bottom layer at different rates, according to the task to perform; (iii) redundancy in the tactile data representation prevents any reader to block the writer.

As a matter of fact, the tactile data structure in the bottom layer is a memory resource whose access by top layers must be monitored through a properly defined communication interface. Moreover, each top layer provides user applications with a number of functionalities retrieving tactile images at a needed rate. Depending on the particular task to execute, three data acquisition modes are available, namely as-soon-as-possible, periodic and event-based.

As-soon-as-possible. In this mode the reader tries to synchronize with the writer in the bottom layer. As a consequence, it is evoked by the writer. This mode is targeted at subsystems requiring as many tactile images as possible such as fine-grained control strategies or safety behaviours.

Periodic. In this mode the reader is scheduled at a given rate, obviously slower than the writer. This mode is targeted at algorithms running at slow rates performing possibly complex operations on tactile images. It is worth mentioning that proper synchronization mechanisms are needed between specific user tasks and the reader, in order to enforce their scheduling within their shared period.

Event-based. In this mode the reader awaits the user tasks to request a read operation and then performs it once. This mode is specifically targeted at non real-time or non periodic applications.

C. Functional Requirements and Technical Solutions

Functional requirements introduced in Section II-B can be mapped to this framework. Whilst multiple data indices are structurally embedded within the architecture, real-time performance must encompass all the bottom layer and at least the behaviour of each reader in any top layer. This has been implemented adopting the RTAI/LXRT\textsuperscript{1} extension, which guarantees hard real-time behaviour for the bottom layer and soft real-time behaviour for top layers. Data consistency within tasks is intrinsically met since each top layer has its own consistent local copy of tactile images. Data coherency must be guaranteed by adding proper locking mechanisms to access the bottom layer data as well as memory areas of additional services.

Consistency and coherence of data deserve special attention. Two additional requirements are needed to guarantee a deadlock free behaviour for the communication process. The former, ceaseless acquisition, states that the bottom layer writer must never be blocked due to any locking mechanism that may exist for synchronization between any reader. The latter, independence from readers, states that data acquisition in the bottom layer must be guaranteed to be sound regardless of the existence of any number of reader instances. This is to prevent unpredictable behaviour in systems where the number of top layers could dynamically change.

A double buffer mechanism has been implemented to tackle this issue. At a given time instant, the first buffer is locked by the writer, whereas the second, used by any reader, is free. In the subsequent writer execution period, a buffer swap occurs: the writer locks the free buffer and frees the unlocked one. As a consequence, all the reader instances read from the unlocked buffer and store a local copy of the tactile image.

However, there are many situations where the locking mechanism needs particular attention. After the writer completes its operation on the locked buffer, it would require a buffer swap: if a reader were still reading from the free buffer, swapping buffers would jeopardize the requirement of data consistency for that reader. Since the writer cannot wait indefinitely for the reader to finish, an undesirable swap skip behaviour occurs: the writer ignores the buffer swap and rewrites the same locked buffer. Figure 5a shows how a swap skip happens. In such a case, all the reader instances miss the tactile frame being overwritten.

To lower the likelihood of swap skips, a possible solution, shown in Figure 5b, requires that each reader $R_i$ is able to keep track of its Worst Case Execution Time (WCET), i.e., the time spent reading the free buffer. With this information, $R_i$ can ascertain whether it can complete the read operation before the writer period begins. In particular, if we define $T_c$ as the current time instant, $WCET_i$ as the Worst Case Execution Time for $R_i$, $T_{free}$ as the time instant when the

\textsuperscript{1}Please refer to the official RTAI website at www.rtai.org.
buffer is free, $T_w$ as the buffer period and $k$ as the number of subsequent swap skips since the last buffer swap, if the following condition

$$T_c + WCET_i \leq T_{free} + (k + 1)T_w$$

(1)

holds, the reader decides to delay the access to the tactile image to lower the chance for all the other reader instances missing a tactile frame. In Section IV it is shown that the likelihood of swap skips is negligible, although non null.

IV. PERFORMANCE ASSESSMENT

In this Section we show experimental results aimed at assessing the overall middleware performance. In particular, we are interested in quantitatively measuring the temporal delay $\Delta T_d$ introduced by tactile information management in the embedded central workstation, i.e., from the time instant $T_0$ when a tactile image is available from the underlying hardware subsystem to the time instant $T_n$ when the data are available to user level applications.

Let us consider a global configuration with a bottom layer and $n_t$ top layers. The bottom layer hosts $n_b$ services that must sequentially operate on the tactile image. If we consider that each reader $R_i$ can host a number of services that sequentially operate on the tactile image, we can define $n_t$ as the number of different services among all the readers.

The user level applications needs processed tactile images. In this scenario, the temporal delay $\Delta T_{di} = T_{di} - T_0$ for the $i$th top layer can be modelled as:

$$\Delta T_{di} = \Delta T_{acq} + \sum_{j=0}^{n_b} \Delta T_{bsj} + \Delta T_{comm} + \sum_{k=0}^{n_t} \Delta T_{tsk} + \Delta T_{acc},$$

(2)

where $\Delta T_{acq}$ is the time needed by the writer to acquire the tactile image and store it with data, $\Delta T_{bsj}$ and $\Delta T_{tsk}$ are the times needed to perform, respectively, the correspondent bottom service and top layer service w.r.t. reader $R_i$ (i.e., equal to zero if the top layer service does not belongs to that reader). $\Delta T_{acc}$ is the time needed for the user application to access useful tactile data, and $\Delta T_{comm}$ is the time for the reader and the writer to communicate.

To assess the middleware performance a number of tests has been set up. The kernel module has been disconnected from the real hardware and the tactile data were generated simulating an underlying tactile system as described in Section III-A. This choice has been made for stressing the architecture without a costly and hardly modifiable test platform. According to Equation 2, performance are evaluated by measuring the time required by the middleware to present the data once they are available, disregarding the time required by the hardware to produce a tactile image. Tests have been performed on a an Intel Core2 Duo@2.66 GHz workstation, provided with 4×1GB DIMM DDR2 at 667 MHz running a Linux OS, with a kernel 2.6.35.7 patched with RTAI magma.

TABLE I: Performance measures for the writer, the reader, the service allowing fast detection of active taxels and complete scanning of skin taxels

<table>
<thead>
<tr>
<th></th>
<th>Average (ns)</th>
<th>Worst-Case (ns)</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>writer $\Delta T_{acq}$</td>
<td>33848</td>
<td>39438</td>
<td>570</td>
</tr>
<tr>
<td>reader $\Delta T_{comm}$</td>
<td>40482</td>
<td>58639</td>
<td>166</td>
</tr>
<tr>
<td>Service $\Delta T_{tsk}$</td>
<td>114979</td>
<td>136572</td>
<td>14536</td>
</tr>
<tr>
<td>Complete Scan</td>
<td>57282</td>
<td>222646</td>
<td>16629</td>
</tr>
</tbody>
</table>

Performed tests target different middleware aspects, such as overall performance, swap skip occurrence and service response time. The service used for the last test implements a naive contact detection mechanism building a bitmask that represents whether in a certain AoI there is a contact. In this context, a contact is defined as any taxel response below a predetermined threshold.

A. Performance Tests

This test consists of running the kernel module at 30 Hz and initializing it with 30848 taxels divided in 100 AoIs, not overlapping, and creating a reader process in as-soon-as-possible mode. Execution times of writer, reader the implemented service are logged. This test has been repeated 3000 times. Table I shows the mean, WCET and variance of, respectively, the writer, the reader and the implemented service, as well as of a complete scan of the taxels. The scan is obtained by inspecting all the taxels individually.

B. Swap Skip Tests

In these tests, the kernel module is running at 30 Hz, double buffered, and initialized with 30848 taxels in random AoIs. Each test runs 20 processes, creating them one by one with a 10s delay, logging the number of swap skips per seconds.

1) As-soon-as-possible: In this test, all the reader instances are requested to work in as-soon-as-possible mode, in which they try to synchronize with the writer. It has been observed that in this case the number of swap skips is always 0, except for an occasional swap skip when a process is first created, i.e., one swap skip is the cost of getting in synchrony with the writer. This swap skip can not be avoided with the reader swap skip prediction property, because at its first run, the reader does not have an estimate of its execution time.

2) Periodic Mode - Random Periods: This test is run twice, the first time for setting the prediction on and the second time off. The reader periods are randomly generated according to a uniform distribution in $[T_w, 10 \times T_w]$ with a granularity of one nanosecond. As a result, the all the reader instances are evoked at their periods, each time at a different time offset with respect to the writer evocation, thereby creating a high chance of executing shortly before the writer evocation and thus creating a swap skip. It has been observed that swap skips are rare in the first place and that, with the swap skip prediction, these swap skips are completely removed.
C. Active Taxel Discovery Tests

In these tests, the kernel module is running at 300 Hz, double buffered, and initialized with 30848 taxels. The taxels are equally assigned to each AoI, but differently in each test. Each test runs 200 times, thereby generating a random skin with an incremented number of AIs, i.e., starting from one AoI for the whole skin up to 200 AIs. On each run, the module is loaded and initialized, whereas a test task is created in as-soon-as-possible mode. In all these tests, the writer activates a different single taxel on each frame writing. Then, the test task performs a thorough search on the skin using the result of the service to find the active taxel, logging the search time.

Figure 6 presents trends of average execution times for the complete search over the whole skin even if the active taxel is found early. It is evident, as it was expected, that the search with overlapping AIs has a larger execution time since every active taxel appears in more than one AoI. Therefore, there is a trade-off between this search time and the advantages of having overlapping regions such as increased reliability through redundancy. It is worth noting that, as shown in Table I, the search time (i.e., the complete scan time), when not using the result from the mentioned service, is considerably greater compared to the values reported in Figure 6, even in the worst case. However, to that time the time required by the service to build the structure must be added (i.e., the $\Delta T_{\text{struct}}$ in Table I). However, in a generic scenario with multiple reader instances, this is anyway an improvement w.r.t. multiple subsequent searches.

On the one hand, if the skin is mapped to a single region, then in order to find an active taxel, each single taxel must be inspected, i.e., the mentioned service becomes ineffective. However, since this service creates packed bit-patterns of regional activities, the search is still substantially faster than a search on the taxels themselves. As the number of regions increases, the search time gets reduced due to the fact that no search is performed on AIs identified as *not active*. Furthermore, it is even more faster if we want just to check if there is activity in a specified AoI, without knowing the activated taxel. On the other hand, the introduction of more AIs introduces a small overhead in search times as it can be observed in Figure 6.

V. CONCLUSION

This paper describes a software frameworks for managing huge amount of tactile data. When humanoid robots must be provided with large-scale tactile sensing systems, the need arises for designing a middleware able to abstract from the underlying sensing mechanisms as well as from the specific robot structure. The paper discusses a number of relevant functional requirements, design choices and specific solutions, with respect to a specific – yet realistic – use case. Experiments show that the developed architecture can be considered as a first step towards this goal.

Fig. 6: Average search times for active taxels with disjoint and overlapping AIs. As it can be noted, a number of spikes are present when the number of areas increases. To understand the causes some tests are on going, but we believe that they are related to memory management of the underlying operating system.

REFERENCES


