HapticLib: a haptic feedback library for embedded platforms

Abstract

Mobile and wearable embedded devices connect the user with digital information in a continuous and pervasive way. The need for portability and unobtrusiveness limits the possibilities of user interaction with such devices, challenging the designer to exploit new input and output modalities. A key benefit is given by the possibility to use multi-modal interaction capabilities that can dynamically act on different human senses and the cooperative capabilities of the small and pervasive devices.

In this scenario we present HapticLib, a software library for the development and implementation of vibro-tactile feedback on resource-constrained embedded devices. It was designed to offer a high level programming interface for the rendering of haptic patterns, accurately modeling the nature of vibro-tactile actuators and different touch experiences.

CR Categories: H.5.2 [Information interfaces and presentation]: User Interfaces—Haptic I/O, Prototyping;
D.2.2 [Software Engineering]: Design Tools and Techniques—Software libraries;
C.3 [Special-purpose and Application-based systems]—Real-time and embedded systems, Signal processing systems;
G.2.3 [Discrete Mathematics]: Applications—Design, Experimentation;

Keywords: embedded platform, haptic feedback, haptic patterns, vibro-tactile feedback, vibrotactile actuators, hardware abstraction, human computer interface

Links:

1 Introduction

The last few years have seen a significant technological advance in hardware and software development of embedded systems, fostering a broad diffusion of mobile and wearable devices. This trend is contributing to the evolution of our intelligent information society, allowing the user to be seamlessly connected with digital information. The shift towards mobile and pervasive computing devices triggered the adoption of innovative interaction paradigms: touch responsive surfaces, tangible interfaces and gesture or voice recognition are finally entering our homes and workplaces [Shaer and Hornecker 2010], [Lawson et al. 2009].

We are experiencing the proliferation of smart objects and sensor networks embedded in our environment or worn on our body. This ubiquitous network of interconnected devices, always available and often referred to as the Internet of Things (IoT), is enabling new applications and services, ranging from enhancements of home and office environments, to remote healthcare assistance and to the birth of smart cities.

The efficient development and optimization of embedded systems is an enabling technology for the realization of this vision. Portability and unobtrusiveness constraints limit the dimensions of such devices, introducing the need for the development of novel interfacing paradigms, since traditional interfaces are no longer applicable. Today’s integration scale allows the design of ubiquitous and mobile devices, which are unobtrusive but capable of performing complex tasks. To fully exploit the user’s interacting capabilities, embedded systems should be able to dynamically use multiple input and output modalities, involving different senses [Dumas et al. 2012]. The way the user interacts and receives information from one or more digital devices is very important and can become critical in some scenarios and situations as, for example, in the case of digitally assisted surgery [Yamamoto et al. 2012]. The feedback provided to the user could be visual, auditive, tactile or, more likely, a combination of the three, as several studies show the benefits of multimodal interaction modalities [Lee and Spence 2008].

In the context of interaction with pervasive and wearable devices, vibro-tactile haptic feedback is rapidly gaining interest within the research community. Its main advantages are the versatility and the possibility to enhance the feedback modes of the system, without the need for direct user attention [Pielot et al. 2011].

Interfaces based on haptic feedback involve the user’s proprioceptive and kinesthetic senses through mechanical, thermal and/or chemical stimuli generated by the machine. In order for the haptic feedback to be effective, the user has to recognize those stimuli from known everyday-life patterns, or spend time training on them [Kohli et al. 2006].

This paper introduces HapticLib\(^1\), an open source library for the development and use of haptic feedback through vibro-tactile actuators in resource-constrained embedded systems. The library we propose allows the developers to accurately control the rendering of predefined haptic patterns or to create new ones. HapticLib targets low-level, microcontroller-based embedded systems characterized by reduced computational power, limited amount of memory and a low energy budget. It reduces the development effort and enhances the functionalities of a broad range of interactive devices, including wearable nodes, smart objects and sensor-actuator networks. The physical actuators may be driven by virtually any embedded device, which are usually battery-operated, placed directly on the user’s body and do not necessarily need an underlying operating system or additional higher level controllers.

HapticLib offers an intuitive high-level Application Programming Interface (API) and hides the low-level details using a Hardware Abstraction Layer (HAL), greatly reducing the effort of developing advanced interactive applications and easing their port to different embedded platforms.

---

\(^1\)Kinesthesia and proprioception relate to self awareness of relative positions and movements of body parts and feeling of received stimuli.

\(^2\)HapticLib is available on its website at [https://guardati.it/hapticLib].
2 Background

Information produced by the human touch sense is called the haptic experience and comes from two channels: kinesthetic and cutaneous [Boff et al. 1986]. The kinesthetic channel provides kinetic information about forces applied on body parts such as the reaction force obtained by pushing an object or the speed with which fingers move. On the other side, the cutaneous channel provides information that comes from the skin such as shape, consistence and temperature of the touched surface. To deepen these aspects, a large amount of literature is available, such as [Kandel et al. 2000].

Haptic science focuses on the study of the touch sense and on the creation of accurate models of the tactile perception. Understanding and modeling the human tactile perception enables the design of haptic interfaces and artificial techniques for touch stimulation, also known as haptic rendering. It consists in the generation of a proper haptic feedback in response to user interactions with digital devices, to enhance the user experience.

2.1 Haptic interfaces

Haptic interfaces can be separated in two categories: contact point interfaces and tactile displays. Contact point interfaces consist of platforms that have one or more terminal elements, called contact points, of known position and orientation, where forces and torques can be applied. Tactile displays integrate an ensemble of actuation points allowing to encode and express more complex information through the sense of touch. Their main features are the small dimensions, the low power consumption and the possibility of integration in classical interface devices such as mouses, keyboards or joysticks.

Several works validate the effectiveness of a multi-modal feedback approach, like the one in [Prewett et al. 2006]. The authors of this work have conducted a meta-analysis to compare user effectiveness when employing visual and tactile feedback versus only the visual one. Their results indicate that using visual and tactile feedback always enhances the task effectiveness and it is even more evident in high-workload and multi-task scenarios. The work in [Kim and Kim 2011] explores how multimodal feedback can improve multitasking performance. Their results show that, for tasks with reasonable difficulty, non-redundant multimodal feedback is more effective than a single feedback or redundant multimodality.

The work in [Altini et al. 2011] presents an indoor navigation system based on vibro-tactile feedback and Bluetooth localization. In this approach, way-finding cues are provided to the user employing coded vibro-tactile messages, targeting high sensitivity areas to reach an optimal stimulus detection while minimizing the power consumption. Moreover, the system was validated exploring the use of EEG to analyze and quantitatively assess effects of the vibrotactile stimulation. In a similar way, the work in [Rofouei et al. 2010] deals with a low power wearable haptic system. The authors propose new techniques to minimize the power consumption in such systems by scattering the activation instant of multiple tactors and proposes ways to efficiently manage tactor networks used i.e. to create phantom sensation.

2.2 Related Work

The software control of haptic systems involves several aspects and domains, from the high-level coordination and management to the driving of the actual physical end-devices. Finding the right combination of all these aspects is crucial for a successful design and there is a limited variety of software tools to support the developer. For example, a closed source commercial haptic development kit from Geomagic, the Sensable OpenHaptics [Geomagic Corp. ], runs on Windows and Linux PCs and targets only their haptic devices. This solution focuses on the 3D contextualization of employed devices, relying on OpenGL as an additional dependency. CHAI3D [Conti et al.] is an open source product for full featured platforms (Windows, Linux, OS X). This library abstracts several low-level details of the hardware devices supported, assuming the Operating System driver is present and correctly working. In this case, the abstraction refers to 3D applications, such as solid modeling and virtual reality applications. Haptic Library [De Pascale and Prattichizzo 2007] is another C++ open source library that offers different language interfaces (Java, Matlab, python), includes plugins to control some commercial devices, but again needs a complete workstation to run.

Immersion Corporation [Immersion Corporation 1993 –2013] sells proprietary SDKs and libraries to control their haptic interfaces in various environments and for various platforms, including game consoles and smartphones. One of their products focuses on the Android operating system, enabling a somewhat embedded solution that doesn’t need a full featured workstation, yet the dependency on a complex underlying environment, such as an Android device, is a strong requirement for the developer.

All the introduced libraries and software tools have limitations such as their closed source nature or the limited and high-end hardware supported.

We, in contrast, adopted an open source approach and designed the library to be platform independent, not tied to any particular third-party tool and not to have particular hardware requirements.

Haptik Library is a tool to ease the development of low-level software for an embedded node equipped with multiple tactors. There are no assumptions on the final haptic application (user interface, virtual 3D world, etc.) and the target embedded platform does not even require an operating system (but neither excludes its use). The proposed library offers a high-level abstraction of the node’s haptic structure to the developer, shifting the focus on tactors executing haptic patterns rather than timer driven outputs.

3 HaptiLib Architecture

HaptiLib is a C software library, developed for MicroController Unit (MCU) based embedded platforms. To make library usage safer from the developer’s point of view, advanced coding techniques have been used that exploit specific C features. For example union typical and implement parameter encapsulation to enforce compile-time type checking or function pointer arrays map patterns. A comprehensive documentation facilitates its use and extension.

As illustrated in Figure 1, HaptiLib is made out of three main modules: the Hardware platform, the Pattern generator and the User API. Each one of these modules takes care of a specific aspect of the end application.

The Hardware platform module performs the initialization phase and all the hardware interaction services. The Pattern generator module deals with the rendering of haptic patterns and coordinates the activity of multiple tactors while the User API module proxies the application calls for the pattern selection and activation to the correct pattern generator functions. This layered organization allows a transparent library extension and the integration of other modules avoids breaking the existing code.

3.1 Hardware Platform

This module is a Hardware Abstraction Layer (HAL) and separates the low-level code that manages the interface with the specific hard-
ware from the high-level code of the rest of the library. This module enables a seamless integration with existing low-level libraries to interface with the hardware in a more efficient and easy way.

HapticLib offers two different kinds of template: the “application template” to help developers start coding a new application and the “pattern template” intended to ease the extension of the pattern repository of the library.

The library also supply some demos showing different use scenarios. Another feature is the debugging functions; they allow the developer to output some information during the development phase or on the deployed system. These functions can be conditionally compiled into the firmware.

### 3.2 Pattern generator

This module implements the logic that handles the rendering of haptic patterns. A pattern generator is a self-contained entity composed of two parts: an *initiator* to setup the initial state of the desired pattern and a *continuator* to keep track of its state evolution.

A well documented template has been designed to help developers create new and personalized pattern generators, extending the existing library’s repository. The developer has complete freedom to design creative, dynamic or realistic physical models. Every pattern can be designed to use runtime parameters and can drive a single haptor or be linked to multiple ones.

HapticLib allows for multiple patterns to concurrently run on one node, with limitations given only by the available hardware resources. An internal scheduler manages the coordination of the different patterns, which is transparent to the application code.

### 3.3 User API

A great effort has been put to design a clear and simple programming interface allowing the library user to easily develop haptic feedback applications. The API consists of functions for system initialization, pattern setup and pattern start or stop.

The API functions offered by HapticLib are:

- `haptor_desc* hl_initPattern(pattern_name patternName, user_param* userParams)`: This function is used to initialize a specific pattern instance. `patternName` is a mnemonic name description of the pattern generator provided (enum). `userParam` is the pointer to pattern specific, user-tunable parameters. The return value points to the descriptor of the pattern instance.

- `pattern_desc* hl_addHaptor(haptor_desc* newHaptor, pattern_desc* pattern)`: The function `hl_addHaptor` is used to link a pattern instance with a haptor. `newHaptor` points to one of the haptor descriptors returned by `hl_configure()`, `pattern` points to the instance returned by `hl_initPattern()`. The return value is a pattern argument.

- `uint8* hl_startPattern(pattern_desc* pattern)`: This function is used to start an initialized pattern instance. `pattern` points to the pattern instance. The return value contains the exit condition.

- `uint8* hl_stopPattern(pattern_desc* pattern)`: Function used to force the stop of a running pattern instance. `pattern` points to the pattern instance. The return value contains the exit condition.

### 3.4 Simple application execution flow

To clarify the usage of the library we will illustrate an example application, with the execution flow diagram shown in Figure 2. The diagram is made of three stacked sections: on the top we have the User APIs, in the middle the Pattern Generators code is shown and on the bottom there are the function calls to the Hardware platform module. The Intersample Delay intervals are constant in this example and correspond to the time elapsed between two consecutive calls of the system tick generator `SysTick_Handler()`. In this example two different patterns are initialized, started and stopped by calling the appropriate library functions. The first one (pattern A) is a single haptor pattern, acting on the hardware haptor 1, while the second one (pattern B) is a multi-haptor pattern acting on 2 and 3.

The application code first calls `hl_configure()` to setup the HapticLib’s internal structures and the hardware peripherals. Then the first pattern is initialized with the call to `hl_initPattern(A)` and linked to the haptor `hl_addHaptor(1,A)`. Finally the pattern is executed and scheduled calling `hl_startPattern(A)`. The same sequence applies for the pattern B, with the only difference being two calls to link the two desired haptors: `hl_addHaptor(2,B)` and `hl_addHaptor(3,B)`. In the final part of the graph the patterns are stopped using the `hl_stopPattern()` function, first pattern B is stopped, then pattern A.

The graph also shows what happens internally to HapticLib during the execution of the application. The first thing to note is the periodic execution of the system tick handler `SysTick_Handler()`, a low-level and hardware related Interrupt Service Routine (ISR), that calls the HapticLib’s internal scheduler `patternScheduler()`. After the scheduling has been accomplished by the initiator, the specific pattern continuator will be called by the `patternScheduler()` every intersample delay. This will be executed until the pattern naturally ends (if designed to do so) or when the client code calls `hl_stopPattern()` (like in this example application).

When there are patterns running, `patternScheduler()` will call all the

---

4 haptor: haptic actuator.
continuators and each one of them will ultimately update the PWM duty-cycles driving the haptors linked to the pattern, following the designed logic. In this case $A_{\text{continuator}}()$ will update only the PWM related to haptor $\odot$, while $B_{\text{continuator}}()$ will update the PWMs related to haptor $\oplus$ and haptor $\ominus$.

4 Pattern examples

HapticLib comes with some pre-coded patterns, which can be divided into two classes: purely creative or physically related ones. Another pattern classification can be made based on their stop condition: patterns that unconditionally extinguish are called impulsive patterns, meanwhile continuous patterns are stopped only with an external condition.

Among the included patterns, the Test pattern is a very simple example belonging to the impulsive and creative classes. This pattern consists of two periods of a saw tooth profile, as shown in Figure 3, acting on the intensity of a single haptor. Another creative pattern provided is the Constant pattern. It allows to update the shaking amplitude of the actuator at run-time and it never stops unless an external stop command is given, making it a continuous pattern. Some examples of possible constant patterns are reported in Figure 4. The library also includes an example of an impulsive pattern based on a physical phenomenon. This Impact pattern reproduces the sensation felt on an impact with a determined surface. The phenomenon, studied in [Okamura et al. 2001], is reproduced using a parametric exponential function to model the material of the surface and the impact velocity. The model used is:

$$Q(t) = A \cdot v \cdot e^{-B \cdot t} \sin(2 \cdot \pi \cdot f \cdot t)$$

where $A$, $B$, and $f$ depend on the surface material and $v$ is the impact velocity. This model is processed offline using Matlab, where 256 samples are extracted from the obtained profile and stored in the library. Compared to the originally proposed model, we made some manipulations to prevent negative and fractional samples, avoiding floating-point operations. This simulation is repeated using different parameter sets encompassing three distinct impact velocities and three possible materials, as suggested in [Okamura et al. 2001] and reported in Table 1. Simulation results are arrays of samples for each pattern version and are directly embedded in the library. In Figure 5 the sample values successions are shown for three different materials and the same impact velocity.

5 Implementation and Results

In this section, we will present a reference hardware platform implementation and two demonstrative applications, showing the low requirements of a typical haptic application and the simplicity of the HapticLib usage for developers.
5.1 Reference hardware platform

The reference implementation where the library was developed is the STM’s STM32VL-DISCOVERY evaluation board. This board offers an STM32F100RB ARM Cortex M3 MCU and breakout connectors for its I/O pins, along with LEDs and buttons for simple user interaction. A diagram of the hardware setup is shown on Figure 6. From the haptic development point of view, this platform offers some interesting specifications, such as a core and peripheral bus speed clock up to 24Mhz, more than 20 PWM channels directly mappable to GPIOs and 16 bit system timers to measure time delays. STM offers the StdPeriph library to easily manage on-chip low-level peripherals and HapticLib uses it to reduce code complexity.

HapticLib, at its current stage of development, fully supports Eccentric Rotating Mass (ERM) vibro-tactile actuators, driven by a single PWM-controlled voltage. For the implementation of the library we employed Precision Microdrive’s Pico Vibe 312-103 actuators [Precision Microdrives Corp.]. This device is quite small (12 × 2.7mm, 1.3g), offers a maximum shaking amplitude of 1.25G and works at 3V with low power consumptions (147mW). Moreover, the typical rise time of the actuator is around 80ms and it has a lag time of about 30μs.

5.2 Use cases

Two of the example applications implemented to show the capabilities and the performance of the library are mentioned here. The HapticWorld application shows how simple it is to use the library and its high-level features offered to the developer. This application initializes the library components and runs one instance of the test pattern on one haptor.

The second example is the HapticLevel application. It uses a 3-axial accelerometer to display the tilt of a board equipped with four haptors, one for each edge. In this case, the haptic actuation gives a feedback on the tilt of the device, proportionally actuating the haptor placed on the edge of the board facing the direction of inclination.
5.3 HapticLib resources overhead

To evaluate the overhead introduced by our library, we analyzed its memory requirements and compared the HapticLevel application with an equivalent one, which performs the same task without the use of HapticLib. In Table 2 the memory requirements of the proposed library is shown. The first column reports the library code size at various optimization profiles and the second one shows its static RAM requirements. We can observe that the overhead related to HapticLib is not critical, if compared to the program and RAM memory sizes present in today’s value-line microcontrollers, especially considering the benefit of the lower design complexity and the reduced time to market for a given haptic application. Also note that RAM requirements could be reduced in future library versions using dynamic allocation for specific internal descriptor structures.

Next, we compared the main() function’s source code size in the two solutions. We included in the bare application, along with the application-specific logic code, all the timer related configuration code because this is what HapticLib does for the application. The results, shown in the central column of Table 3, indicate how HapticLib leads to a reduced main() body size made only of code related to the haptic application domain. This advantage would increase for more complex applications.

The latency introduced by HapticLib in the sensor-actuator information flow, using the simple scheduler that coordinates the patterns’ rendering, is negligible when compared to the characteristic times of the actuators as shown in the second column of Table 3. Furthermore, in many cases a scheduler already exists, further reducing the relative overhead of the library. For this test we used the reference hardware platform specifications and considered the code executed from the instant in which the sensor is sampled to the actuation of the haptor.

Both applications use a 10Hz sampling frequency, the bare application code samples the sensor and directly updates the timer’s PWM duty-cycle, whereas HapticLevel samples the sensor and updates the pattern’s run-time parameter (the constant value representing the PWM’s duty-cycle). In the latter case, the updated value is fed to the actual timer peripheral only when the scheduler runs its update. This happens at regular intervals, hence introduces a random delay, uniformly distributed in the range [0, interval period]. The interval period regulates the time between two consecutive updates of the library’s scheduler and is tunable using the API. This should be performed keeping in mind the underlying hardware characteristics and the number and timing constraints of the haptic actuators.

The used devices have a 30ms lag in driving the rotating mass that generates the vibrations and we used an interval period of 10ms.

6 Conclusions

The library presented in this paper offers a new approach for embedded haptic feedback applications and represents a versatile tool for a broad range of developers. HapticLib is not designed for any particular commercial device, it uses an open source approach and provides features to ease the porting of the library and the developed code to multiple platforms. In fact, being on its initial development status, there is only the reference implementation described on the previous section; but as shown in the paper, the architectural solutions make it very easy to port HapticLib to other platform like Arduino, PIC, MSP430. The project has a solid architectural design and gives a robust framework for developers to enrich applications with haptic interaction.

References


