Sensors on Stage: Conquering the Requirements of Artistic Experiments and Live Performances

Simon Waloschek
Center of Music and Film Informatics
HfM Detmold / HS OWL
waloschek@hfm-detmold.de

Aristotelis Hadjakos
Center of Music and Film Informatics
HfM Detmold / HS OWL
hadjakos@hfm-detmold.de

ABSTRACT

With the rapid evolution of technology, sensor aided performances and installations have gained popularity. We identified a number of important criteria for stage usage and artistic experimentation. These are partially met by existing approaches, oftentimes trading off programmability for ease of use. We propose our new sensor interface SPINE-2 that presents a comprehensive solution to these stage requirements without that trade-off.

Author Keywords

Live Performance, Sensors, TUI Toolkits, Physical Computing, Arduino, Phidgets, Max/MSP

ACM Classification Keywords

H.5.2 [Information Interfaces and Presentation] User Interfaces — Input devices and strategies
H.5.5 [Information Interfaces and Presentation] Sound and Music Computing — Systems

1. INTRODUCTION

Modern digital sound synthesis methods open up many new possibilities for musical expression. Almost every technical parameter of sound and virtual instruments can be controlled by artists. Interactive elements may be incorporated into musical projects thanks to the mapping of these parameters into the digital domain. The prerequisite for this is an interface with sensors that captures real world input and provides the results in an adequate representation.

Although existing systems have already lowered the entry barrier for such scenarios significantly, not all requirements of live performances on stage and their conception have been covered satisfactorily. The requirements that are laid out in the following are based on our experience with sensing platforms in concert and educational settings: In April 2014 we used the first iteration of a platform called "SPINE" [1] to re-create and perform a piece called "Light Music", which applied inertial sensors to track the musician's hands. This was the first time that the piece was performed without the help of the composer Thierry De Mey and his team. Since the electronics originally used were not available, we successfully used the SPINE as a substitute. Furthermore, we regularly employ the Phidgets [9] sensor platform in our educational workshops with composers and media artists.

Both scenarios featured the following requirements:

Ease of use: Creative processes of media artists and composers are often characterized by experimentation and improvisation [7]. To support such processes, sensor platforms should be simple to use and allow switching from one sensor setup to another without much effort. A necessity to program in a low-level, text-based programming language to get access to sensor values will oftentimes switch the focus away from artistic experimentation. To be optimal for such scenarios, the systems should be usable out-of-the-box or at most require only little configuration.

Programmability: An enormous number of sensors are available today and new sensors are made available continually. Ideally, a sensor system should interface with a multitude of them. But embedded systems are able to do much more than just forwarding sensor data to a PC. They are capable of processing input data and producing visual and sonic output on their own.

Robustness of Communication: An immanent key requirement for all stage-worthy solutions is their reliability. If the underlying transmission technology is prone to errors, failures may distort the whole performance.

Speed: Slow transmission speeds may hamper complex live performances. In order to process input data in real time, the interface has to provide sensor values with sufficiently low latencies.

These challenges are addressed by our platform SPINE-2 (see Figure 1), that is further described in this paper.

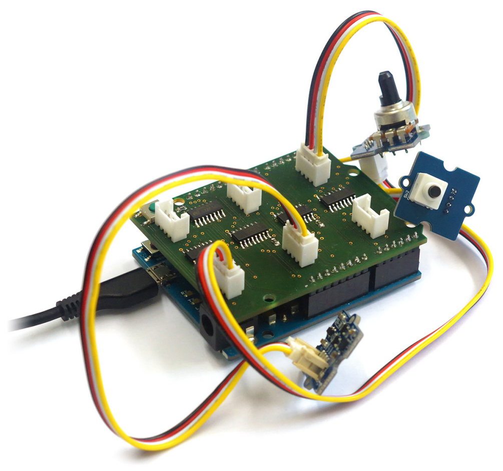


Figure 1: The SPINE-2 shield (green PCB) to which various Grove sensors (blue PCBs) are attached.

2. RELATED WORK

In the following we will discuss existing sensor platforms with regard to the requirements outlined in the previous section. We compare our solution to Phidgets, xOSC, Hiduino, Arduino, and Raspberry Pi. Of course there are many other sensor platforms that have been proposed, e.g., the CUI [13], d.tools [8], the Eobody [6], the TOASTER and KROONDE [4]. But those selected, we deem the closest competitors in various aspects: Phidgets (concerning ease of use), xOSC and Hiduino (robustness) and finally Raspberry Pi and Arduino, both being popular physical computing platforms.

2.1 Ease of Use vs. Programmability

Tangible user interface (TUI) toolkits are non-programmable integrated sensor solutions. Phidgets [9] is perhaps the most commonly used TUI toolkit. A standardized connector system is used to attach simple analog and digital sensors to a master unit, where the values are digitized and transmitted via USB in a vendor-specific protocol. Furthermore, there are some specialized sensor and actuator units that are directly connected to the computer without using a master unit as intermediary. Physical computing platforms like the Arduino [11], the Rasperry Pi, or Beagle Boards on the other hand are open and programmable. Almost any sensor can be connected but this requires the development of a specific firmware. While media artists may be proficient in visual programming languages such as Max/MSP, they usually lack the text-based programming background that is necessary to program these platforms successfully. Our platform merges these two approaches by presenting different layers to programmers and media artists.

We have evaluated the usability of SPINE-2 and compared it to the Phidgets platform (see Section 4.1). Phidgets is rated with '+' (see Table 2). The usability of the SPINE-2 is significantly better compard to our first iteration [1], which we rate with '◯'.

2.2 Robustness of Communication

RS232 over USB: Microcontroller-based physical computing systems, like the Arduino, oftentimes use RS232 over USB for communication. The main problem with RS232 over USB is that the drivers in modern operating systems are not robust enough. In [1] we have reported severe usability problems. The most common error we are experiencing is that the virtual RS232 port stops working until the computer is rebooted. We have also experienced Max/MSP freezes and even blue screen errors. These issues are usually related to user (or software) errors, e.g., the user may unplug the USB cable while the communication is still taking place or open more than one connection to the same port. Due to the problems when starting communication, we find RS232 over USB to be problematic for live musical performances and rate Arduino with '–' in Tab. 1. Phidgets uses a vendor-specific implementation and thus shows similar problems, although to a lesser extent. Therefore, we rate Phidgets with '◯'.

Wi-Fi: Wireless transmission protocols operating in the 2.4 GHz band are vulnerable to interference from other devices [12]. Although TCP/IP would provide reliable communication over the unreliable Wi-Fi network, it is not suited for time-critical transmission of sensor values in the context of live musical performances, since it introduces high and uncontrollable latency. Since xOSC relies primarily on wireless communication, we rated it to have intermediary transmission robustness (◯) in Tab. 1.

Ethernet: Today, Ethernet uses full-duplex and point-to-point connections between endpoints and switches so that collisions do not occur, making lost packets rather unlikely. Because most Raspberry Pi models provide Ethernet we rate its transmission robustness as good (+).

Hiduino and SPINE-2: SPINE-2 is a Human Interface Device (HID, see Section 3.2). The communication is very robust as our evaluation shows (see Section 4.2). The Hiduino [5] uses a comparable strategy and is also rated with '+'.

2.3 Speed

For an objective comparison of transmission speed we calculated the maximum throughput for all the above mentioned related systems. These values represent only the theoretical connection speed without considering potential speed losses due to other load (value conversions, overhead etc.). Arduino and its variations (including SPINE-1 [1]) use a regular RS232 (UART) over USB connection for data transmission with a maximum baud rate of 115.2 kBd/s, resulting in a throughput of 11.25 kB/s. xOSC sends up to 400 UDP unicast packets over a Wi-Fi connection per second with a fixed size of 104 bytes which results in 40.625 kB/s [12]. Values for Phidgets could not be evaluated due to its closed design. We assume that Hiduino transmits data rather slowly due to the limited speed specified in the used MIDI protocol. Raspberry Pi is able to communicate via a 100 Mbit/s connection [10] and thus can be considered very fast. We measured the transmission speed of SPINE-2 in Section 4.3.

3. THE SPINE-2

We wanted to create a sensor platform that is easy to use for composers and media artists, such as Phidgets. However, we wanted our platform to be fully programmable so that users can learn about low-level programming, develop code for new sensor modules or use the platform in not yet anticipated ways. As Arduino is a very popular physical computing platform, we decided to make it the basis of our platform. The solution consists of an Arduino Leonardo, our SPINE-2 shield (see Figure 1), and specific software to program the SPINE-2 system.

3.1 Universal Connectors

In order to have a pluggable solution we needed to choose a standard for the connector which is used to connect sensor modules to the SPINE-2 shield. We chose the Grove [14] sensor module system due to its sufficient selection of relatively inexpensive sensors. The Grove connector has two power supply pins and two sensor-specific pins, usually either analog, digital, I^2C or UART.

To make it possible to connect any Grove sensor to any connector on the SPINE-2 shield, each of the 6 connectors is "universal". This means that they can be used as analog inputs, digital inputs or for more complex communication protocols commonly used by sensors. These protocols are already implemented in the Arduino microcontroller. Several multiplexers redirect the needed data lines to the corresponding pins of the connectors.

3.2 Robust and Fast Communication

To avoid the errors mentioned in Section 2.2, we made SPINE-2 a HID. Extending the idea of "Hiduino" [5], an implementation for sending and receiving MIDI messages with an Arduino via USB, the transmission speed and safety have been significantly enhanced. HIDs require no driver installation and are supported out-of-the-box by modern operating systems.

Each data value is transmitted as a 32-bit float to represent data with sufficient resolution. Although an analog

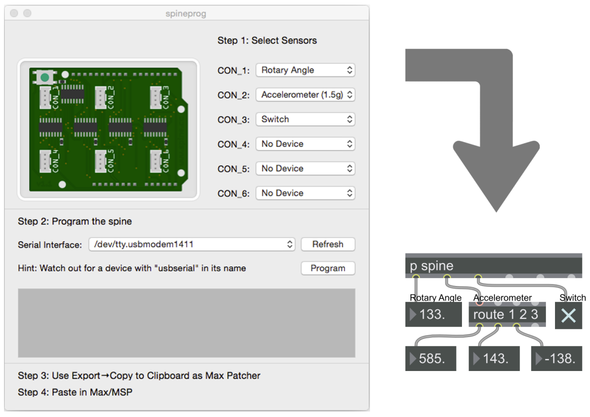
Table 1: Comparison of solutions

	Phidgets	xOSC	Hiduino	Arduino	RaspberryPi	SPINE	SPINE-2
Ease of Use	+	–	–	–	–	○	+
Programmability	–	+	+	+	+	+	+
Comm. Robustness	○	○	+	–	+	–	+
Speed [kB/s]	N/A	40.625	N/A	11.25	≫ 1 MB/s	11.25	242.9

value will typically be a positive integer number, negative and fractional values can be represented if necessary. Every fixed-sized HID message that is being sent to the computer contains up to 12 float32 `sensorValues` and uses a 1 byte header with the number of sensor values (`valueCount`) and a per-shield `spineID`. 12 sensor values have proven to be a suitable trade-off between the overall message size regarding the transmission overhead and fast reaction time for new sensor data. The single values are later identified by a `valueID` in case of multiple values per sensor (e.g., three axes of an accelerometer) and a `connID` which specifies the corresponding connector.The Arduino Leonardo offers up to four USB endpoints. That enables the simultaneous use of both the standard Arduino UART and the new HID implementation in a single sketch and a single USB cable. During development, the regular UART can be a helpful tool for debugging purposes. However, it is not recommended to use the UART in a final setup.

3.3 Using the SPINE-2

In order to function properly, the SPINE-2 must know which sensor is attached to which connector. This is done with a configuration application called *spineprog* (see Figure 2). First, the user attaches the sensors to the connectors on the SPINE-2 shield. The particular sensor for each of the 6 connectors from the list of already available sensors firmwares is select. After chosing a serial interface, the Arduino can be programmed. In the background, existing and newly generated C code is compiled and flashed to the Arduino. The spineprog also prepares a ready-to-use Max/MSP patch for the particular sensor setup selected by the user. The user can copy this patch into the clipboard and paste it into Max/MSP (Figure 2). Digital values are represented as toggle switches. Analog values are represented as floating point numbers in that patch.


spineprog

Step 1: Select Sensors

CON_1: Rotary Angle
CON_2: Accelerometer (1.5g)
CON_3: Switch
CON_4: No Device
CON_5: No Device
CON_6: No Device

Step 2: Program the spine

Serial Interface: /dev/tty.usbmodem1411

Hint: Watch out for a device with "usbserial" in its name

Step 3: Use Export→Copy to Clipboard as Max Patcher

Step 4: Paste in Max/MSP

Figure 2: The spineprog application is used to program the Arduino and prepares an adequate Max/MSP patch for the user.

3.4 Developing for the SPINE-2

In order to provide best familiarity to Arduino users, every sensor specific module looks and feels as an individual Arduino Sketch and can also be edited with the Arduino IDE. Exactly like regular sketches, SPINE-2 modules consist of a `setup()` method that is executed once after a successful boot of the Arduino to initialize connected hardware. A second method named `loop()` is called repeatedly and usually contains the code for obtaining new sensor values (see Figure 3). The new function `write(...)` puts a value into the transmission buffer and encapsulates the USB-HID internals.

```
void setup() {
  pinMode(PIN_A, INPUT); //Set PIN_A to input
}
void loop() {
  write(1, digitalRead(PIN_A)); //Send via channel 1
write(2,  analogRead(PIN_B)); //Send via channel 2
}
```

```
#include <Spine.h>
namespace sensor0 {
  const uint8_t SENSOR_ID = 0;
  uint8_t PIN_A = A0; //Definition of hardware pins
  uint8_t PIN_B = A1;
  inline void write(uint8_t c, float f) {
    Spine.write(SENSOR_ID, c, f);
  }
  #include "foo.h" //Insert SPINE module code
}
namespace sensor1 {
  [...]
}
void setup() {
  Spine.begin();
  sensor0::setup();
  sensor1::setup();
}
void loop() {
  Spine.select(0); //Select multiplexer channel 0
  sensor0::loop();
  Spine.select(1); //Select multiplexer channel 1
  sensor1::loop();
}
```

Figure 3: A simple sensor module (top) and the automatically generated main file (bottom)

During the flashing process, all selected SPINE-2 modules are merged via an internally generated Arduino sketch. Each module is wrapped in its own C++ namespace (`sensor0`, `sensor1` etc.) to avoid variable and function name conflicts. This main sketch is generated by the spineprog. To compile the C code, the spineprog contains a stripped down version of the Arduino toolchain with our own changes to the USB-HID part of the Arduino platform.

4. EVALUATION

We performed a pilot user test where we compared our system to the Phdigets system to see if our system is as easy to use as a (non-programmable) TUI toolkit. Furthermore, we evaluated technical aspects of the system.

4.1 Pilot User Test

The pilot user test was performed with five students of our university. In the test we compared our SPINE-2 system with the Phidgets system. The participants were instructed to install the necessary software on their own computer and control the frequency of a sine generator with a sensor in Max/MSP. After that they were encouraged to use the platform creatively. The user test was carried out as part of a weekly Max/MSP course.

The participants received an anonymized package containing an Arduino Leonardo, our SPINE-2 shield, various Grove sensors and the software including a PDF manual. After the task they filled out the System Usability Scale (SUS) [3] questionnaire. In a second step the participants received Phidgets system components and links for the software and the manual. The participants again performed the task and filled out the SUS questionnaire. Each of the two tasks took about 40 minutes.

SPINE-2 received an average SUS score of 86 ($\sigma = 7.0$), which makes it a system with excellent usability according to Bangor et al. [2]. Phidgets received an average score of 76 points ($\sigma = 18.0$). Accordingly it has a good usability. An unpaired sample, unequal variance, 2-sided t-test was conducted to compare the previous iteration of the SPINE with the SPINE-2. A statistically significant improvement ($P < 0.05$) was found. In a previous user study it had obtained an average SUS score of 74 ($\sigma = 8.4$) [1]. We are aware of the influence that the testing order of the systems had but were restricted to a very limited number of participants.

Our evaluation indicates that the usability of SPINE-2 is at least as good as Phidgets. Furthermore, the SPINE-2 has other benefits. Most importantly it is still a fully-fledged Arduino system that can be programmed freely.

4.2 Reliability of Communication

Experiences with UART-based systems revealed serious problems in case of unpredictable disconnections and multiple instances accessing the same hardware. The severity ranged from Max/MSP freezes to complete system reboots. The software architecture of HID prevents such consequences. In this way the SPINE-2 is invulnerable to connection interruptions and automatically continues to receive new values after a reconnection.

To test the robustness, we intentionally unplugged and replugged a SPINE-2 50 times while using it with Max/MSP. Our Max External successfully recognized the absence of the interface and resumed its work without any further impairment after reconnection.

Multiple instances of the SPINE-2 object in Max/MSP are internally managed and do not disturb each other. In contrast to UART connections, which allow only one instance to access the data, our implementation forwards all incoming data to newly instantiated SPINE-2 objects if the hardware is already used instead of accessing it on its own. We evaluated this feature with multiple simultaneously opened Max Patches using a single SPINE-2 shield.

4.3 Performance

We first measured the transmission speed that could be theoretically achieved with no sensors attached. A stable communication with a computer is established at an average rate of 242.878 kB/s. Compared to the solutions using UART with a commonly used rate of 11.25 kB/s (115200 baud/s, 10 bits per transmitted byte), this is a theoretical speed up of about 21. With a single 3 axis accelerometer connected to the SPINE-2 that is utilized via I^2C, we measured a throughput of 78.401 kB/s.

A more realistic setup scenario example would consist of 3 analog (e.g. light sensors, sliders, rotation angle sensors) and 3 digital (e.g. buttons) sensors. Such a configuration achieved a data rate of 69.939 kB/s. Since each sensor value is represented as a 32 bit float regardless of the sensor type, this equals a sample rate of about 2310 samples/s for each sensor. The HID approach performs significantly better than typical UART solutions.

5. CONCLUSION AND FUTURE WORK

The use of sensors in live performances leads to very specific requirements for sensor interfaces. We have shown that our approach meets these criteria and outperforms competing systems in several regards. Our SPINE-2 shield extends the Arduino system and thus enables programmers to use the widely-used Arduino language. Composers on the other hand do not need to know the internals. A simple graphical user interfaces lets the user easily configure a sensor setup and export a customized Max/MSP patch for a quick integration into new or existing projects. The ease of use has been evaluated with the SUS method and was considered excellent.

Making the SPINE-2 a HID enhanced the transmission speed and safety compared to similar systems and eliminated the need of driver installation.

We plan to increase the number of supported sensors. This goal could ideally be accomplished by users that contribute, creating a SPINE community with an active exchange of code, experience and knowledge.

6. REFERENCES

[1] A. Hadjakos and S. Waloschek. SPINE: A TUI Toolkit and Physical Computing Hybrid. In *NIME*, pages 625–628, 2014.

[2] A. Bangor, P. T. Kortum, and J. T. Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[3] J. Brooke. SUS: A quick and dirty usability scale. *Usability evaluation in industry*, vol. 189, 1996.

[4] T. Coduys, C. Henry, and A. Cont. TOASTER and KROONDE: high-resolution and high-speed real-time sensor interfaces. In *NIME*, 2004.

[5] D. Diakopoulos and A. Kapur. HIDUINO: A firmware for building driverless USB-MIDI devices using the Arduino microcontroller. *NIME*, pages 405–408, 2011.

[6] E. Gallin and M. Sirguy. Eobody3: a ready-to-use pre-mapped and multi-protocol sensor interface. In *NIME*, 2011.

[7] S. Gelineck and S. Serafin. From idea to realization - understanding the compositional processes of electronic musicians. In *Audio Mostly*, 2009.

[8] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Symposium on user interface software and technology (UIST)*, 2006.

[9] Phidgets Inc. Phidgets Inc. – unique and easy to use USB interfaces. http://www.phidgets.com, 2014. [Online; accessed 25-January-2015].

[10] Raspberry Pi Foundation. Raspberry pi model specifications. http://www.raspberrypi.org/documentation/hardware/raspberrypi/models/specs.md, 2015. [Online; accessed 29-January-2015].

[11] D. Mellis, M. Banzi, D. Cuartielles, and T. Igoe. Arduino: An open electronic prototyping platform. In *Conf. on Human Factors in Computing*, 2007.

[12] T. Mitchell, S. Madgwick, S. Rankine, G. Hilton, A. Freed, and A. Nix. Making the most of wi-fi: Optimisations for robust wireless live music performance. In *NIME*, 2014.

[13] D. Overholt. Musical interaction design with the CUI32Stem: wireless options and the GROVE system for prototyping new interfaces. In *NIME*, 2012.

[14] SeeedStudio. Grove system – wiki. http://www.seeedstudio.com/wiki/GROVE_System, 2014. [Online; accessed 02-January-2015].