
**UNIVERSITÀ DEGLI STUDI DI MILANO**

FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA TRIENNALE IN INFORMATICA MUSICALE

# Utilizzo di sensori su dispositivi mobili per il controllo di parametri di una digital audio workstation

*Candidato*

**Daniele Mulas**

*Relatore*

**Prof. Ludovico L. Andrea**

*Correlatore*

**Stefano Baldan**

Anno Accademico 2013/2014

1. INTRODUZIONE 3

2. ARCHITETTURA E LINGUAGGI 5

2.1. LATO CLIENT 7

2.1.1. HTML5: STORIA E NOVITÀ 7

2.1.2. JAVASCRIPT 9

2.1.2.1. ORIENTATION E MOTION 10

2.2. LATO SERVER 12

2.2.1. NODE JS 13

2.2.2. PURE DATA 13

2.3. MIDI E OSC 14

2.3.1. MIDI 14

2.3.1.1. MESSAGGI 16

2.3.2. OSC 18

2.3.2.1. ELEMENTI 18

3. IMPLEMENTAZIONE 20

3.1. API JAVASCRIPT 20

3.2. WEBSOCKET CON SOCKET IO 22

3.2.1. LATO CLIENT 22

3.2.2. LATO SERVER 24

3.3. MESSAGGI 26

3.4. PURE DATA 29

4. SUGGERIMENTI E CONCLUSIONI 34

5. BIBLIOGRAFIA 36

# 1.Introduzione

Un MIDI controller è uno strumento che genera, trasmette e riceve messaggi MIDI, ovvero una serie d'informazioni su come dev'essere eseguita la performance. Questo fa si che i MIDI controller siano uno strumento da associare necessariamente a un generatore di suoni come un sintetizzatore o una DAW.

La nascita del MIDI ha portato alla diffusione di strumenti creati sul modello di quelli tradizionali come tastiere, chitarre, strumenti a fiato di vario tipo, batterie elettroniche. Brevemente diversi elementi sono stati aggiunti ai controller MIDI, per favorire la manipolazione della fonte sonora: knob, slider, pad sono quelli più tradizionali, per arrivare a touch-pad o sensori di distanza.

Una problematica comune a tutti i device MIDI risiede nella possibilità di controllare contemporaneamente un numero di parametri limitato. Si è pensato per ciò a metodi di controllo alternativi rispetto ai classici potenziometri che permettano col minimo di movimenti la resa maggiore in termini di caratteri sonori controllabili.
La scelta è ricaduta sui sensori di movimento (giroscopio, accellerometro e bussola) degli smartphone.

Dalla sua comparsa sul mercato, nel '92, vi è stata un'evoluzione tecnologica che ha reso il "telefonino intelligente" uno strumento multiuso e per molti indispensabile. Si parla di fotocamere e videocamera, gps, wi-fi e bluetooth, sensori di movimento e di luce sono un esemio degli accessori in dotazione.

Le statistiche mostrano un numero elevato di possessori di smartphone, destinata a crescere in futuro. Il fatto che quest'oggetto sia nelle tasche di molti è stata una motivazione in più che lo ha reso lo strumento ideale. L'utente in questo modo non ha bisogno di possedere hardware aggiuntivo per usufruire dell'applicazione.

La scoperta dell'esistenza di una serie di API Javascript che permettono la lettura dei sensori dal browser, è stato un punto decisivo nella scelta dell'architettura dell'applicazione. Si è deciso quindi di sviluppare

un'applicazione web, o comunemente chiamata web-app, che ha parecchi vantaggi rispetto alle tradizionali applicazioni cosiddette native.

Innanzitutto non vi è alcuna installazione da eseguire. L'utente accede all'indirizzo web e s'interfaccia con l'applicazione tramite il browser. Questo oltre a rendere più semplice e veloce l'uso, fa si che non sia necessario sviluppare più versioni per differenti sistemi operativi. Più rapida è anche la distribuzione degli aggiornamenti, poiché non essendo necessario installare alcun software, l'utente si trova ad usare sempre l'ultima release.

Il lavoro svolto in questi mesi ha come risultato un' applicazione che fornisce all'utente la possibilità di comandare un sintetizzatore o un software musicale o qualsiasi apparecchio che legga il MIDI; in modo semplice ma allo stesso tempo con risultati complessi grazie alla possibilità di controllare diversi parametri contemporaneamente, ma soprattutto utilizzando il proprio smartphone e un computer.

# 2.Architettura e linguaggi

L'applicazione è composta da due moduli: il client, con il quale l'utente si interfaccia dal suo smartphone, e il server, che permette la connessione tra il device mobile e il sintetizzatore.

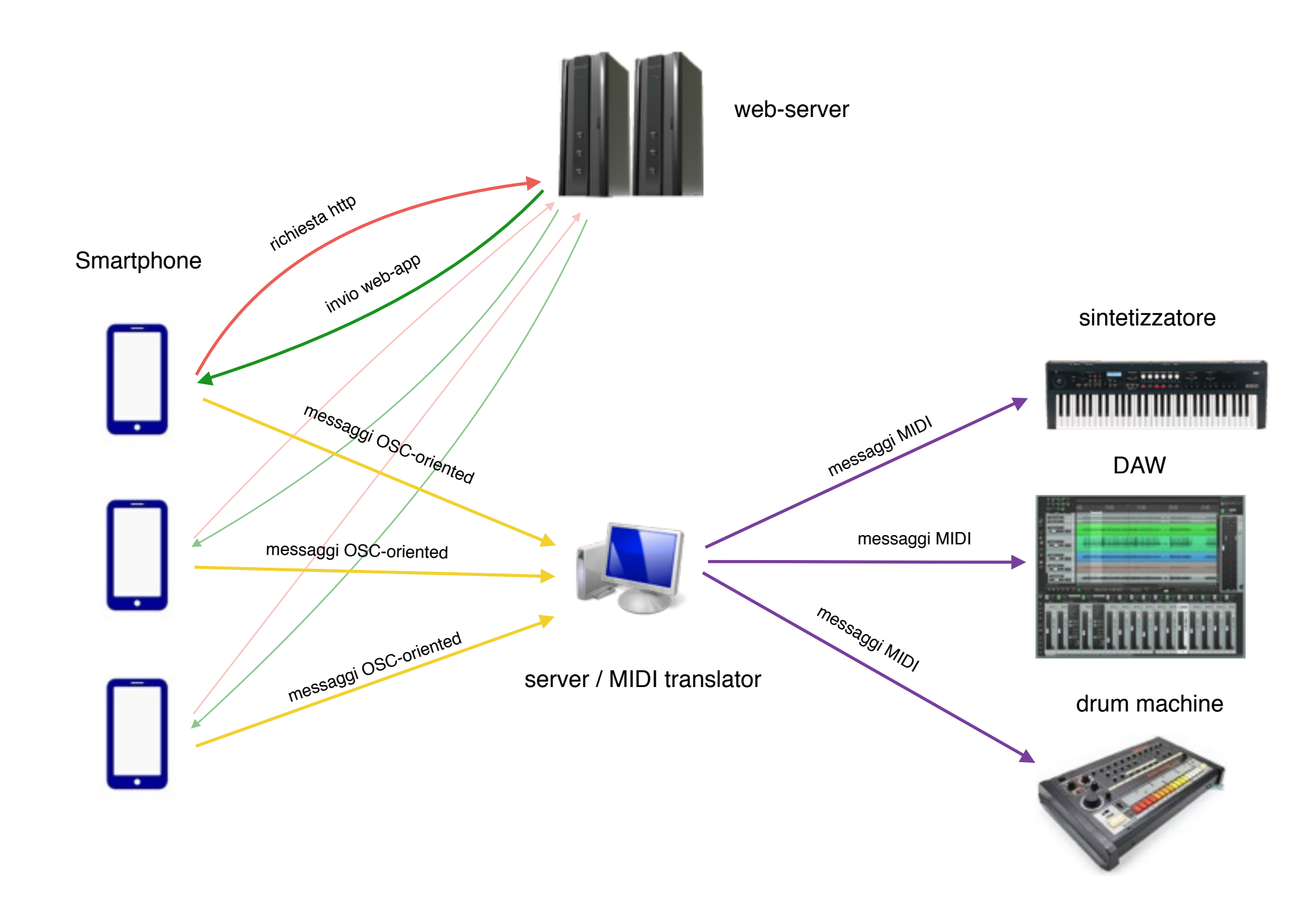
La tipica struttura nel web è l'architettura client-server: un computer si connette a un altro per la fruizione di un certo servizio. Colui che inizia la connessione è il client, mentre il server riceve le richieste. Il protocollo che gestisce queste richieste si chiama HTTP.

L'utente manda una richiesta al web-server sul quale risiede l'applicazione web attraverso il browser del suo dispositivo mobile. Instaurata la connessione, il server trasmette i dati richiesti al client.

Il server dell'applicativo ha due componenti: uno che attende la connessione dai client, ne riceve i messaggi e li trasmette in uscita senza apportare modifiche. L'altro in coda traduce i messaggi secondo lo standard MIDI, così da renderli "leggibili" da un dispositivo MIDI.

Questi due sotto-blocchi del server possono risiedere su due macchine distinte o nel caso anche sullo stesso computer dove risiede la DAW.

L'applicazione è stata progettata per consentire a più device mobili di connettersi contemporaneamente sullo stesso server e permettere delle performance collaborative a distanza.


web-server
Smartphone
richiesta http
invio web-app
messaggi OSC-oriented
messaggi OSC-oriented
messaggi OSC-oriented
server / MIDI translator
sintetizzatore
messaggi MIDI
DAW
messaggi MIDI
drum machine
messaggi MIDI

## 2.1. Lato client

Per l'implementazione del client si è optato per HTML5[1][2] e Javascript[3].

L'Html5 è il linguaggio standard per lo sviluppo di pagine e di applicazioni web.

La lettura dei sensori di movimento è stata possibile grazie alle API Javascript, essenziali per lo sviluppo dell'applicazione.

### 2.1.1. Html5: storia e novità

Il linguaggio di markup per il Wordl Wide Web è sempre stato l'HTML.
HTML venne creato all'inizio come linguaggio per descrivere semanticamente documenti scientifici cioè il design generale della pagina, e nel corso degli anni è stato adattato per descrivere anche altri formati.

Le prime versioni di HTML vennero sviluppate da Tim Bernars-Lee al CERN di Ginevra e successivamente allo IETF dagli anni 1990 al 1995. Successivamente, con la creazione del W3C, lo sviluppo dell'HTML cambiò sede nuovamente e nel 1995 venne rilasciata la versione HTML 3.0. Lo scarso successo di questa lasciò il posto alla versione 4.0 pochi anni dopo.

Negli anni a venire, ci fu un'importante svolta nello sviluppo del più importante linguaggio di markup web: i membri del W3C decisero di orientarsi verso un linguaggio basato sull' XML, per questo chiamato XHTML. La prima versione venne ufficializzata nel 2000 ed era sostanzialmente una riformulazione dell'HTML4 in XML.

L'intento del W3C era di abbandonare l'HTML in favore dell'XHTML e cercare di standardizzare in maniera sempre più rigida le regole del web. Al workshop W3C del 2004 un gruppo di sviluppatori della Apple, di Mozilla e di Opera decisero di continuare lo sviluppo dell'HTML e si riunirono nel WHATWG.

L'idea era di creare delle specifiche e un linguaggio che fosse retro compatibile (cosa che l'XHTML 2 non era). Incominciava a prendere forma l'HTML5.

Qualche anno dopo il W3C fece un passo indietro, abbandonando la strada dell' XHTML per concentrarsi completamente sullo sviluppo dell'HTML5. Ma quando sembravano finiti i contrasti, i due gruppi si resero conto di avere scopi differenti: il W3C voleva pubblicare una versione definitiva di HTML5, mentre la visione del WHATWG era di un linguaggio in continua evoluzione e aggiornamento. La prima versione ufficiale uscirà entro la fine del 2014, ma HTML5 è già presente in gran parte del web.

Le differenze con la precedente versione e le nuove funzionalità introdotte sono numerose.
La prima in assoluto, che mostra l'intento di semplificare HTML5, è la dichiarazione doctype. Il doctype definisce la versione di HTML e di conseguenza le regole alle quali il browser si dovrà attenere per leggere e codificare la pagina web.
Per definire una pagina Html5 si inserisce alla prima riga la dichiarazione:

<!DOCTYPE html>

In confronto alle versioni precedenti di html, il doctype è stato unificato e decisamente semplificato come si può vedere dal doctype di html 4.01:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

Di seguito si elencano le novità più significative:

- CANVAS: significa "tela" in inglese e questo elemento permette di selezionare una porzione di pagina sulla quale è possibile disegnare curve, creare sfumature di colore e grafiche per un videogioco.
- VIDEO: permette la riproduzione di video senza l'utilizzo di plug-in di terze parti come Apple QuickTime o Adobe Flash.
- WEB STORAGE: Html5 permette ai siti internet di memorizzare sul computer grandi quantità di dati e successivamente recuperarle. Il concetto è quello dei cookie ma si tratta di una mole di informazioni ben più grande.
- WEB APP OFFLINE: consente di memorizzare le pagine interessate sul computer. Se si è online il browser carica le pagine dalla rete altrimenti utilizza quelle memorizzate.

- GEOLOCATION: la capacità di localizzare l'utente attraverso gps, l'indirizzo ip oppure l'antenna alla quale è connesso il telefono.
- ELEMENT & INPUT TYPE: Html5 definisce tutta una serie di nuovi elementi semantici (<section>, <nav>, <article>, <aside>, <hgroup>, <header>, <footer>, <time>, <mark>) e nuovi tipi di input ("search", "number", "date", "color", "tel", "url", "email", "date", "month", "week", "time", "datetime", "datetime-local").
- WEBSOCKET: la possibilità di creare una singola connessione full-duplex sulla quale client e server si scambiano messaggi. Il canale di comunicazione rimane aperto fino a che il client non si disconnnette. Le websocket velocizzano le connessioni e ne aumentano l'efficenza.

### 2.1.2. Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nella creazione di siti web (programmazione Web).

Si dice di scripting un linguaggio di programmazione interpretato, cioè la cui esecuzione avviene direttamente dal codice sergente attraverso un programma detto interprete, senza dover essere pre compilato.

La prima versione venne sviluppata alla Netscape Communication nel '95 con il nome di Mocha prima e successivamente di LiveScript. Microsoft ne sviluppò una versione "personale" per Internet Explorer (Jscript) e successivamente venne standardizzato dalla ECMA con il nome di ECMAScript (seppure rimane Javascript il nome più comunemente usato).

Sintatticamente Javascript ricorda linguaggi come il C, C++ o Java ma sostanzialmente è molto diverso. Ad esempio non è un linguaggio tipizzato ovvero in cui bisogna dichiarare il tipo di variabile. Gli oggetti assomigliano più agli array associativi del linguaggio Pearl piuttosto che agli oggetti del C++ o di Java e come il Pearl è interpretato. I meccanismi di ereditarietà degli oggetti assomigliano a quelli di linguaggi come il Self.

Altri aspetti di interesse: in JavaScript lato client, il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il web server non viene sovraccaricato a causa delle richieste dei client.

#### 2.1.2.1. Orientation e Motion

Molti dei nuovi computer, smartphone e tablet vengono messi in commercio con sensori come l'accelerometro, il giroscopio e la bussola che permettono di ricavare dati sul movimento e l'orientamento del dispositivo.

La standardizzazione delle interfacce di programmazione – API Javascript DeviceOrientation e DeviceMotion[4] – ha permesso che queste funzionalita' vengano gestite dai browser e da altre applicazioni.

Le Device Orientation API forniscono informazioni sulla disposizione fisica e l'orientamento del dispositivo nello spazio. L'evento restituisce tre attributi: alpha, beta e gamma. Questi rappresentano la rotazione rispetto a 3 assi.

- **alpha**: è la rotazione attorno all'asse z, cioè l'asse perpendicolare alla superficie terrestre. Restituisce un valore che va da 0 a 360 come quello di una bussola.

- **beta**: restituisce la rotazione rispetto all'asse x, con valori compresi tra -90 e 90.

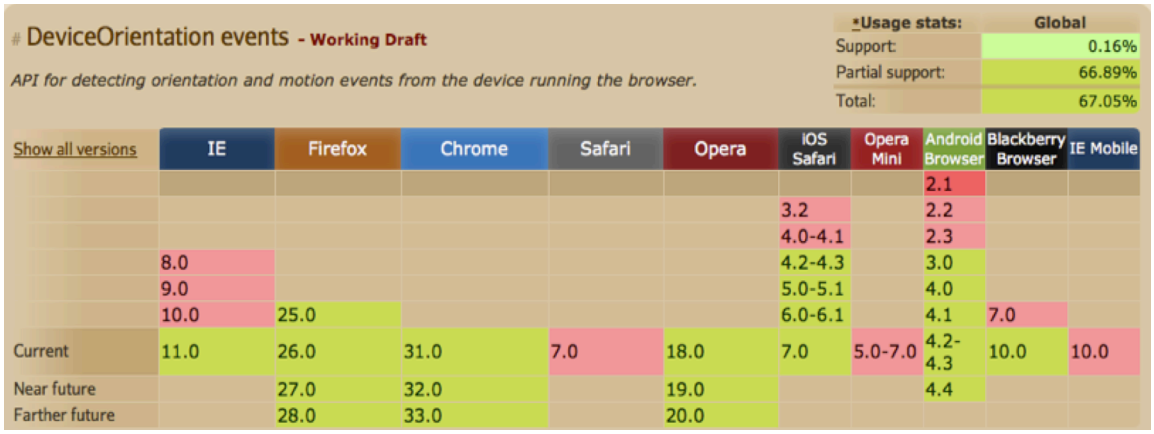

- **gamma**: si riferisce alla rotazione rispetto all'asse y, con valori compresi da -90 a 90.


Per esempio un device appoggiato su una superficie orizzontale, con lo schermo rivolto verso l'alto e in direzione del nord, restituirà 0 da tutti e tre gli attributi.

L'evento Motion Device fornisce l'accelerazione del dispositivo espressa in coordinate cartesiane. Restituisce 4 attributi:

- acceleration: restituisce l'accelerazione del dispositivo rispetto ai tre assi di riferimento, calcolato in m/s^2.
- accelerationIncludingGravity: uguale all'attributo precedente, tenendo conto della forza di gravità.
- rotationRate: è la frequenza di rotazione espressa in gradi/secondo
- interval: è una costante che indica l'intervallo in millisecondi tra i dati forniti dai sensori.

Di seguito è mostrata una tabella dei browser che supportano gli eventi in questione.



# DeviceOrientation events - Working Draft

*API for detecting orientation and motion events from the device running the browser.*

| *Usage stats: | Global |
|---|---|
| Support: | 0.16% |
| Partial support: | 66.89% |
| Total: | 67.05% |

| Show all versions | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser | IE Mobile |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 2.1 | | |
| | | | | | | 3.2 | | 2.2 | | |
| | | | | | | 4.0-4.1 | | 2.3 | | |
| | 8.0 | | | | | 4.2-4.3 | | 3.0 | | |
| | 9.0 | | | | | 5.0-5.1 | | 4.0 | | |
| | 10.0 | 25.0 | | | | 6.0-6.1 | | 4.1 | 7.0 | |
| Current | 11.0 | 26.0 | 31.0 | 7.0 | 18.0 | 7.0 | 5.0-7.0 | 4.2-4.3 | 10.0 | 10.0 |
| Near future | | 27.0 | 32.0 | | 19.0 | | | 4.4 | | |
| Farther future | | 28.0 | 33.0 | | 20.0 | | | | | |

## 2.2. Lato Server

Si è detto che il server svolge due funzioni: una di ricevere i messaggi dal client e l'altra di convertirli in messaggi MIDI.

Il server vero e proprio è stato sviluppato con Node Js[5]. La scelta è stata dettata principalmente dal fatto che le websocket[6] , attraverso cui avviene la comunicazione tra il client e il server, sono state realizzate tramite una libreria di Node js.

La comunicazione tra client e server avviene attraverso una tipologia di messaggi OSC, come si spiegherà meglio nel capitolo successivo.

Pure Data[7] è un linguaggio che fornisce una serie di semplici funzioni per la codifica di messaggi OSC e per la generazione di messaggi MIDI. Questo e il fatto di essere open source, a differenza di Max/Msp, hanno fatto optare all'uso di Pure Data.

### 2.2.1. Node Js

Node js è un framework basato sul motore Javascript V8, sviluppato da Google e componente del browser Chrome, particolarmente indicato per lo sviluppo server-side.

E' una piattaforma sviluppata su modello event-driven: significa che il flusso del programma è determinato da eventi. Per evento si intende ad esempio azioni dell'utente (click del mouse, tocco dello schermo), l'output di sensori, messaggi da altri programmi. A ognuna di queste situazioni è possibile associare una funzione.

Socket IO è una libreria di Node JS che offre una serie di strumenti per l'implementazione delle web socket, sia lato client che lato server.

### 2.2.2. Pure Data

Pure Data, noto anche come PD, è un linguaggio grafico open source sviluppato da Miller Puckette nel 1990 all'IRCAM di Parigi. Fa parte della famiglia del linguaggio di programmazione Max (Max / FTS, ISPW Max, Max / MSP, jMax, DesireData, ecc), sviluppato sempre da Puckette.

E' stato ideato per permettere ad artisti, musicisti, grafici di poter lavorare in modo facile e intuitivo in ambiti come la computer music o in generale il multimedia. La sua modularità permette la condivisione di parti di programma che lo rendono uno strumento diffuso e semplice da utilizzare.

Pure Data è un linguaggio grafico. Significa che a differenza dei tradizionali linguaggi in cui c'è un codice sorgente composto da caratteri alfanumerici, in

PD vi sono una serie di oggetti grafici che l'utente posiziona sullo schermo. Si collegano questi oggetti tra di loro per permettere il passaggio dei dati e l'esecuzione dell'applicazione. Per ciò si dice essere un linguaggio orientato al flusso dati.

Un vantaggio rispetto ad altri linguaggi dove il codice dev'essere processato prima di essere eseguito è il fatto che Pure Data permette di effettuare modifiche durante la performance rendendolo uno strumento adatto ad esecuzioni live.

## 2.3. MIDI e OSC

Il fine ultimo di questa applicazione è la generazione di messaggi MIDI.

Il MIDI[8] è un protocollo che descrive l'esecuzione, senza dare indicazioni specifiche sul suono.

Recentemente ha fatto la sua comparsa un protocollo simile al MIDI ma pensato per essere utilizzato in rete, chiamato OSC[9][10].

Grazie alla sua flessibilità e alla sua compatibilità con Pure Data si è utilizzato per la comunicazione tra client e server una tipologia di messaggi ispirati all'OSC che lo hanno reso preferibile al MIDI.

Ciononostante OSC ha una diffusione molto minore rispetto al MIDI che viene utilizzato da tutti i sintetizzatori e i software musicali e quindi ha una portata più ampia.

### 2.3.1. MIDI

Il protocollo MIDI è lo standard da 30 anni a questa parte per l'interazione tra strumenti musicali elettronici. Il suo utilizzo si allarga anche ad altri device: dai

computer (sempre nell'ambito musicale) ai cellulari, fino ad arrivare alle luci e a strumentazione da palcoscenico come macchine del fumo.

Il MIDI nasce dall'idea dei produttori di strumenti musicali elettronici di creare un protocollo standard che permetta la comunicazione tra i loro device.

All'inizio degli anni 80, diversi costruttori, ad esempio Oberheim e Roland, offrivano già sui propri strumenti, sistemi di interfacciamento personalizzati molto limitati.
Sotto la spinta in particolare di Sequential Circuit, Oberheim e Roland, con la collaborazione di altre case produttrici americane e giapponesi, vengono scritte le prime specifiche e nell'83 viene immesso sul mercato il primo strumento ad utilizzare il MIDI.

Pur non essendo cambiati certi concetti basilari, dalla prima versione delle specifiche MIDI ad oggi sono stati apportati diversi aggiornamenti: il General MIDI, nuovi meccanismi di connessione (usb, FireWire, Wi-Fi), nuovi prodotti come telefoni cellulari e videogiochi ecc.

L'introduzione di uno standard ha portato parecchi vantaggi ai musicisti.

Il MIDI ha reso possibile la separazione della tastiera dal sintetizzatore, inteso come generatore di suono, e ha permesso di suonare diversi strumenti contemporaneamente. Questo ha significato la nascita di vari tipi di input device, o MIDI controller, basati su strumenti tradizionali (chitarra, batteria, strumenti a fiato) e non (un microfono collegato a un convertitore pitch-midi, riconosce il pitch del suono in input e genera messaggi MIDI).

Pur essendo il MIDI il protocollo standard ormai da molto tempo, esso presenta alcune limitazioni abbastanza significative.

Il primo problema è la velocità di trasmissione massima pari a 31,250 bit/secondo, non abbastanza da sopportare performance particolarmente "pesanti", tenendo conto del fatto che un ritardo seppur piccolo viene introdotto per la lettura dei messaggi da parte dei microprocessori. Inoltre il suo carattere sequenziale nella trasmissione dei dati può introdurre un asincronia, ad esempio nell'esecuzione di un accordo.

Problemi d'interconnessione tipo cavi unidirezionali e necessità di Patch Bay, seppure ormai superati con la comparsa di connessioni come l'USB, hanno reso

il MIDI uno strumento scomodo e costoso in principio.
Queste e altre limitazioni non hanno evitato che il MIDI si affermasse così saldamente e per così tanto tempo sul mercato. Pur rimanendo un protocollo dinamico e in continua evoluzione, alcuni problemi alla base come quelli appena elencati rimangono irrisolti e hanno portato alla nascita di alternative come l'OSC.

#### 2.3.1.1. Messaggi

La comunicazione tra dispositivi MIDI avviene tramite lo scambio di messaggi. Ognuno di questi è composto da uno o più "parole" di 10 bit l'uno. Il primo bit (start bit) è impostato a 0 mentre l'ultimo bit (stop bit) è uguale a 1 e non vengono considerati nella lettura del messaggio.

I byte che compongono il messaggio sono di due tipi: byte di stato e byte di dati.

Ogni messaggio è composto da un byte di stato che ne indica il tipo e uno o più byte di dati che indicano il valore.

Il byte di stato ha il primo bit più significativo impostato a 1 (che lo rende riconoscibile come byte di stato), tre bit (dal secondo al quarto) identificano il tipo di messaggio e gli ultimi 4 bit specificano il canale MIDI.

Il byte di dato ha il primo bit impostato a 0 e sette bit per specificare il valore associato al byte di stato.

I messaggi MIDI si dividono in 5 gruppi:

- Channel Voice: riferiti a un canale MIDI specifico; ne fanno parte i messaggi "note on" e "note off" (inizio e rilascio della nota), "polyphonic key pressure" o "channel pressure" (cambiamenti di pressione tra un note on e note off), "control change" (un sottogruppo di messaggi per il controllo di altri parametri), "pitch bend (controllo del pitch).
- Channel Mode: ne fanno parte i messaggi "mode" che danno indicazioni sul modo in cui i messaggi devono essere interpretati. Altri tipi sono "all

note off" (un messaggio di emergenza che manda un note off a tutti i device collegati) simile a "all sound off" (disattiva istantaneamente tutti gli apparecchi).

- System Common: inviati a tutti i dispositivi connessi e su tutti i canali. Esempi sono "song position pointer" (indica una posizione sulla traccia), "song select" (seleziona uno dei 128 file in libreria), "tune request" (manda un segnale ai sintetizzatori analogici per accordare gli oscillatori), "quarter frame" (sono impulsi di segnale per sincronizzare gli apparecchi utilizzano il MIDI Time Code).
- System Real Time: come gli altri messaggi di sistema, vengono inviati su tutti i canali e danno informazioni sulla riproduzione. Ne fanno parte i messaggi "start", "stop" e "continue" o i messaggi "MIDI clock" (simile al MIDI Time Code, servono per sincronizzare).
- System Exclusive: sono messaggi esclusivi di case produttrici, per permettere di mandare messaggi personalizzati e ricoprire funzioni che i messaggi standard MIDI non svolgono.

Si riportano di seguito alcuni esempi di messaggi MIDI che vengono generati dall'applicazione.

NOTE ON: 1 byte di stato + 2 byte di dati (key number e velocity)

1001 0000 00111100 01111111

messaggio "note on" sul canale 1 con pitch 60 (do centrale) e velocity 127 (valore massimo)

NOTE OFF: come "note on", 1 byte di stato + 2 byte di dati

1000 0110 010001001 01000110

messaggio "note off" sul canale 6 con pitch 69 e velocità 70

MODULATION WHEEL: messaggio "control change", con il primo byte di dati che indica il tipo di messaggio "control change" e il secondo byte di dati che ne indica il valore.

1byte di stato + 2 byte di dati

1011 0010 00000001 0xxxxxxx

### 2.3.2. OSC

Open Sound Control (OSC) è un protocollo di comunicazione tra computer, sintetizzatori e alti strumenti multimediali, ottimizzato per l'uso sulla rete.

Questo protocollo semplice ma potente è indicato per il controllo in tempo reale del suono, pur rimanendo flessibile e facile da implementare.

Caratteristiche principali di OSC sono:

- linguaggio open-ended (aperto e personalizzabile) e dinamico, con uno schema in stile URL.
- valori ad alta risoluzione (compreso il time tag).
- bundle di messaggi per l'esecuzione contemporanea di messaggi specifici.

I campi di utilizzo dell' OSC sono moltissimi, dalle applicazioni musicali in tempo reale agli strumenti web interattivi, da linguaggi di programmazione per sintesi sonora alla sensoristica.

#### 2.3.2.1. Elementi

I pacchetti OSC sono l'unità di trasmissione del protocollo. L'applicazione che invia pacchetti viene chiamata Client OSC mentre chi li riceve è il Server OSC.

Un pacchetto OSC è composto dal *contenuto*, un blocco contiguo di dati binari, e dalla *dimensione*, cioè un numero sempre multiplo di 4.

I pacchetti vengono consegnati a protocolli di trasmissione come UDP, nel quale non vengono apportate modifiche; altrimenti se si utilizza TCP, ogni pacchetto è preceduto da un numero di 32 bit che ne indica la dimensione.

Il contenuto può essere o un messaggio OSC o un cosiddetto bundle OSC. Il primo byte del contenuto del pacchetto distingue tra queste due alternative.

I messaggi OSC sono composti da 3 parti:

1. Indirizzo: stringa che inizia con il carattere "/". Questa parte del messaggio rappresenta il dato a cui il messaggio si riferisce. Ha una composizione in stile path-name.
2. Type tag: stringa in cui ogni singolo carattere rappresenta il tipo di dato contenuto nell'argomento. Inizia con una ",". Questa tabella riporta la corrispondenza tra i Type tag OSC e l'argomento OSC corrispondente.

| OSC Type Tag | Type of corresponding argument |
|---|---|
| i | int32 |
| f | float32 |
| s | OSC-string |
| b | OSC-blob |

3. Argomento: rappresenta il valore dell' "oggetto" a cui ci si riferisce. In base al Type Tag sarà un certo tipo.

Un Bundle OSC è composto da una stringa "#bundle" iniziale, seguita da un Time Tag, seguito da zero o più OSC Bundle Elementi.

Un Bundle Elemento OSC è costituito (come per il pacchetto OSC) da una dimensione e dal suo contenuto. I contenuti però possono essere sia un messaggio OSC sia un Bundle OSC a sua volta.

# 3.Implementazione

In questo capitolo si analizza l'applicazione a un livello più approfondito, riportando il codice e commentandolo.

Si parte dal lato client, in particolare trattando gli eventi DeviceOrientation e DeviceMotion. Si passa poi all'implementazione delle websocket e del server con Node.js e Socket.IO, descrivendo come viene gestita la connessione di più client. Si riporta la funzione che genera e invia i messaggi tra client e server per finire con le patch di Pure Data.

## 3.1. API Javascript

### A. Verificare la compatibilità

Si verifica che il browser supporti gli eventi Orientation e Motion.

Se non lo fosse, viene mandato un messaggio di errore.

```
if(window.DeviceOrientationEvent) {…}
else {
      alert("Mi spiace ma il tuo browser non supporta l'evento
      Orientation");
};
```

### B. Impostare un listener

Per poter sfruttare le funzionalità offerte dalle API Javascript, bisogna creare un listener per gli eventi in questione.

AddEventListener() permette di impostare delle funzioni che verranno chiamate al verificarsi dell'evento specificato.

```
window.addEventListener('deviceorientation', orientation, false);
```

## C. Funzione

I valori restituiti dalle seguenti funzioni, vengono salvati in una variabile globale di tipo array. L'utilizzo di una variabile vettore è stata pensato come metodo per mantenere la corrispondenza, tramite l'indice del vettore, tra i valori restituiti e i parametri sonori scelti dall'utente.

### ORIENTATION

Il metodo "round" dell'oggetto Math arrotonda all'intero più vicino.

```
function orientation(event){
            var x = Math.round(event.gamma);
            var y = Math.round(event.beta);
            var z = Math.round(event.alpha)
            risultato[0] = x;
            risultato[1] = y;
            risultato[2] = z;
    };
```

### MOTION

Dell'evento "motion" si considera solamente l'attributo "acceleration". Poiché restituisce un valore in coordinate cartesiane, viene fatta la media e moltiplicato per 10 per rendere il risultato più significativo.

Il metodo "abs" restituisce il valore assoluto.

```
function motion(event) {
           var acceleration = event.acceleration;
            x=Math.abs(acceleration.x);
            y=Math.abs(acceleration.y);
            z=Math.abs(acceleration.z);
            risultato[3]=Math.round(10*((x+y+z)/3)); ;
        };
```

## 3.2. Websocket con Socket IO

Si è detto che le websocket sono uno strumento utile in casi in cui si ha bisogno di connessioni performanti in termini di efficenza.

Per la loro implementazione si è usato una libreria di Node JS che fornisce gli strumenti necessari sia da lato client che dal lato server.

### 3.2.1. Lato client

La funzione "connetti", come suggerisce il nome, permette la connessione al server. Viene richiamata all'inserimento dell'indirizzo ip della macchina su cui risiede il server.

L'argomento della funzione è l'indirizzo ip che l'utente inserisce nel form apposito.

Per utilizzare la libreria Socket.IO lato client è necessario caricare degli script presenti sul server. Non sapendo al momento in cui la pagina viene caricata qual'è l'indirizzo ip (poichè è l'utente che lo inserisce prima di utilizzare l'applicazione) è necessario generare il comando per caricare la libreria in questo modo.

Si crea un elemento script, si dichiara la sua sorgente (src) e s'inserisce all'interno dell'elemento body.

```
function connetti(ipAdd) {
      script = document.createElement('script');
      script.src = "http://" + ipAdd + ":8080/socket.io/socket.io.js";
      document.body.appendChild(script);
```

In questo modo viene caricata la libreria da supporto e si esegue la connessione correttamente.

La connessione avviene sulla porta 8080. Il numero della porta deve coincidere con quello sulla quale il server attende la connessione e naturalmente

dev'essere una porta libera quindi non utilizzata da altre applicazioni o dal sistema.

Avvenuta la connessione si manda un messaggio al server e un messaggio di sistema che verrà visualizzato sulla console del browser. Questo strumento per il debugging è stato di grande aiuto nello sviluppo dell'applicazione.

```
script.onload=function(){
      socket = io.connect('http://' + ipAdd ,{ port: 8080 });
      socket.on('connect', function() {
      socket.send('CONNESSO;');
      console.log("CONNESSO");
```

Con l'instaurazione della connessione, si può dire conclusa la fase di configurazione. Viene modificato un parametro CSS dell'elemento "configurazione" (quello dove l'utente inserisce canale MIDI ed indirizzo ip) nascondendolo alla vista dell'utente.

```
var hidden = document.getElementById("configurazione");
hidden.style.display="none";
```

Il metodo "setTimeout" richiama la funzione (primo parametro) dopo il numero di millisecondi specificato nel secondo parametro. La funzione esegue un controllo sulla variabile socket; nel caso in cui non risulti settata, quindi inteso come se la connessione non fosse andata a buon fine, viene mandato un messaggio di errore e si ricarica la pagina.

```
setTimeout(function(){
      if(socket == 0){
            alert("indirizzo ip non valido");
            location.reload();
      }
 },5000);
```

### 3.2.2. Lato server

La parte server sviluppata in Node JS gestisce le connessioni con i diversi client, ne riceve i dati e li invia a Pure Data.

Il modulo Socket.IO viene utilizzato anche per l'implementazione delle websocket lato server.

Come è stato accennato poco più su, la connessione avviene sulla porta 8080. Il modulo "net" instaura la connessione con Pure Data per la quale è stata scelta la porta 13467.

```
var io = require('socket.io').listen(8080);
var pd = require('net').createConnection(13467);
```

Avvenuta la connessione con PD, si manda un messaggio di sistema e si inizializza una variabile per il conteggio dei messaggi ricevuti.

```
pd.on('connect',function(){
    console.log('connesso a pd');
    var x=0
```

Instaurata la connessione con il client, il server esegue un controllo sul canale MIDI che l'utente intende utilizzare e invia il risultato di questo controllo al client. Dopodichè si tiene traccia dell'utente e del rispettivo canale.

```
io.sockets.on('connection', function (socket) {
      socket.on('new', function(data){
      var risp=controlla(data);
      if(risp == 1){
            socket.send('libero');
      }
      else if(risp == 0){
```

```
            socket.send('occupato');
      };
      aggiungi(data,socket.id);
});
```

Si utilizza un oggetto array per tenere traccia dei diversi client e del canale MIDI.

```
function aggiungi(can,sock){
      connessi[conteggio]=can;
      sock_connessi[conteggio]=sock;
      console.log('Aggiunto elemento ' + sock + ' sul canale ' + can);
};
```

Attraverso il metodo "indexOf" degli array si verifica che il canale MIDI non sia già in uso. IndexOf restituisce l'indice dell'array in cui si trova un valore passatogli come argomento. Se il risultato è un valore minore di 0 si suppone che il valore non sia presente e quindi che non il canale MIDI sia libero.

```
function controlla(midi){
      if(connessi.indexOf(midi)>=0){
            console.log('Canale ' + midi + ' già in uso');
            return 0;
            }
      else if(connessi.indexOf(midi)<0){
            console.log('Canale ' + midi + ' libero');
            return 1;
      }
};
```

Ogni volta che si riceve un messaggio dal client, viene aggiunto un carattere ";" alla fine del messaggio e inviato a sua volta a Pure Data. Viene inviato un messaggio di ack al client per l'avvenuta ricezione.

```
socket.on('message', function (message) {
      console.log("ricevuto: ",message);
      pd.write(message + ';');
      x++;
      socket.send('ack' + x);
```

Quando il client si disconnette viene aggiornato l'array e cancellata la voce con l'id-socket e il canale MIDI del client.

```
socket.on('disconnect',function(){
      var indice=sock_connessi.indexOf(socket.id);
      sock_connessi[indice]=0;
      connessi[indice]=0;
      console.log('Disconnesso client' + socket.id);
});
```

## 3.3. Messaggi

Come si è già detto in precedenza, i messaggi inviati dal client sono basati sul modello di messaggi OSC, cioè come un path address.

I messaggi hanno una struttura di questo tipo:

**/canale MIDI / tipo di messaggio / parametro_valore;**

Il canale MIDI viene impostato dall'utente al momento della configurazione iniziale tramite un form e salvato all'interno di una variabile. La funzione "canale" esegue un controllo sul valore inserito che per le specifiche MIDI, che dev'essere un numero compreso tra 1 e 16.

```
function canale(number){
	if(number<1 || number>16){
		alert("Errore, i canali MIDI vanno da 1 a 16");
		location.reload();
	}
	else
		channel=number;
};
```

In base al tipo di messaggio, si è pensato a due metodi diversi per la creazione e l'invio.

Si inizia trattando i messaggi che trasmettono i valori dei sensori di movimento, principalmente messaggi *control change*.

L'utente ha la possibilità di iniziare e di stoppare l'invio dei messaggi cliccando il pulsante INVIO, al quale è associata la seguente funzione.

```
function invio(){
	if(bool==0){
		bool=1;
		intervallo = setInterval(function(){
			var i;
			for(i=0; i<4; i++) {
				socket.send(creaMsg(i));
			};
			console.log(risultato)},50);
		}
	else if(bool==1){
		bool=0;
		clearInterval(intervallo);
	};
};
```

Innanzitutto si esegue un controllo sulla variabile "bool". Se settata a zero, s'inizia la trasmissione dei messaggi, se settata a uno si stoppa la trasmissione.

Il metodo setInterval permette di eseguire un'azione ciclicamente. Il primo argomento è la funzione da eseguire, il secondo sono i millisecondi tra una esecuzione della funzione e l'altra. Si è impostato un valore di 50 millisecondi, considerato adeguato per avere una buona qualità a livello uditivo nel controllo di parametri sonori.

La funzione esegue un ciclo al cui interno si richiama il metodo "send" della classe socket che invia messaggi al server. Come argomento di send, si richiama la funzione CreaMsg alla quale si passa l'indice del ciclo.

Il metodo "clearInterval" blocca l'esecuzione ricorrente della funzione.

```
function creaMsg(indice){
      var messaggio;
      messaggio = '/' + channel + '/cc/' + comando[indice] + ' '  +
risultato[indice] + ';' ;
      return messaggio;
};
```

L'argomento passato alla funzione è una variabile numerica, cioè la variabile "i" del ciclo di cui si è parlato sopra. L'applicazione utilizza due variabili globali array: in una vengono salvati i valori che le funzioni orientation e motion device restituiscono. Nell'altro vettore sono salvano i parametri selezionati dall'utente. Così si riesce a mantenere le corrispondenze e l'utente può modificare i parametri a proprio piacimento.

Ritornando alla funzione creaMsg, questa restituisce una stringa così formata:

- channel: canale MIDI che l'utente inserisce al momento della configurazione
- cc: rappresenta il tipo di messaggio MIDI, cioè il control change.
- comando[indice]: è il vettore di cui si è accennato sopra. In questa variabile si copia il parametro selezionato dall'utente.
- risultato[indice]: infine il valore restituito dagli eventi orientation e motion.

I messaggi di tipo note on e note off, non essendo necessario inviarli in continuazione, vengono generati e trasmessi attraverso la funzione “note”. La funzione viene richiamata agli eventi touchstart e touchend, rispettivamente al tocco dello schermo e al rilascio da parte dell’utente. Attraverso lo strumento “canvas” di HTML5 si è riprodotta una tastiera di un ottava che permette all’utente di inviare messaggi note con pitch differenti, con la possibilità di suonare accordi.

```
function note(val,pitch){
	var trasposition=60+pitch;
	if(val==1){
		messaggio = '/' + channel + '/note/on ' + transposition +
';';
		socket.send(messaggio);
	}
	if(val==0){
		messaggio = '/' + channel + '/note/off ' + transposition +
';';
		socket.send(messaggio);
	}
}
```

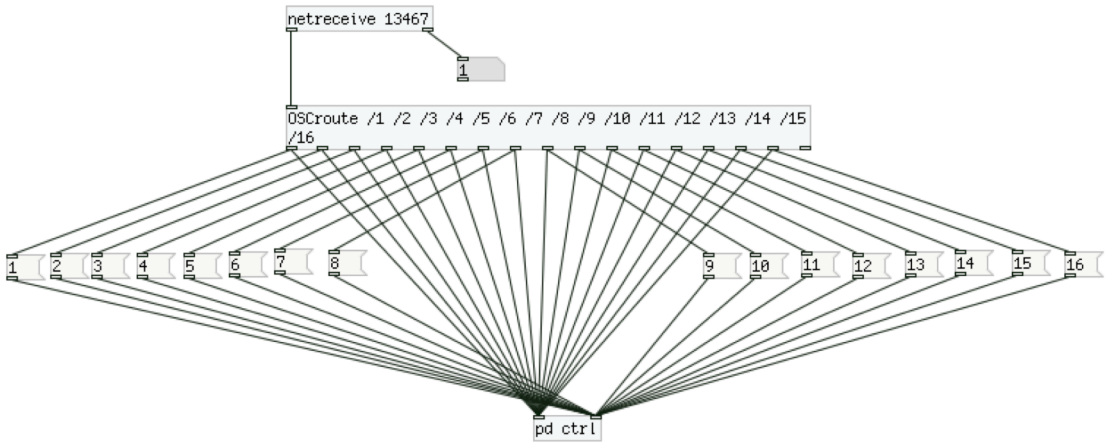
La funzione “note” riceve come argomenti una variabile “val”, impostata a 1 per l’evento “touchstart” e 0 per quello “touchend”, e una variabile “pitch” che identifica la nota suonata dall’utente.

Il messaggio viene quindi generato e inviato allo stesso modo di come si è visto in precedenza con la funzione “creaMsg”.

## 3.4. Pure Data

L’ultimo modulo dell’applicazione, quello che si occupa della traduzione dei messaggi OSC in messaggi MIDI, è stato sviluppato con Pure Data.

Di seguito si riporta la patch principale.




L'oggetto "netreceive" permette di ricevere dati sulla porta specificata come argomento. Come uscita, sulla sinistra viene trasmesso il dato ricevuto, sulla destra il numero di client connessi.

Grazie all'oggetto OSCroute è possibile leggere e trattare messaggi OSC in modo semplice ed efficace. Del messaggio che riceve in input, OSCroute considera solamente la parte di stringa compresa tra il primo carattere slash partendo da sinistra e il successivo. Se questa corrisponde a uno dei suoi argomenti, trasmette sull'uscita corrispondente la rimanente parte di messaggio. Quindi avrà un numero di uscite pari al numero degli argomenti, più l'ultima a destra nel caso in cui la stringa non coincida con alcun argomento.

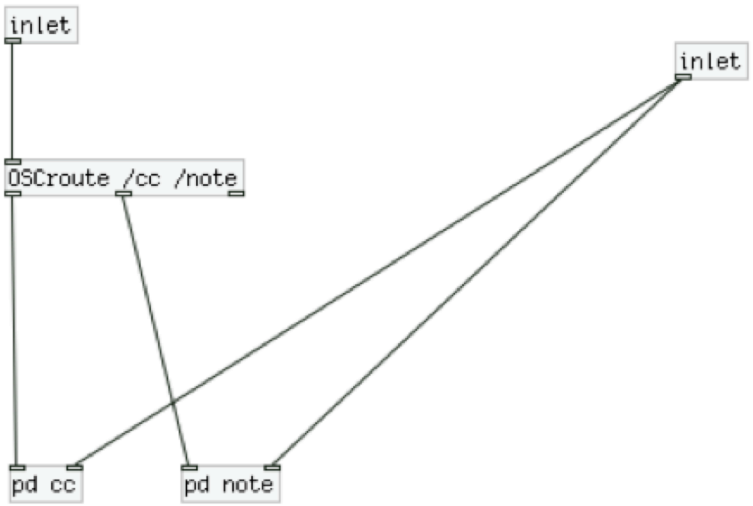
Ad esempio in questo caso, come si è già detto, il messaggio OSC è strutturato con il canale MIDI come primo oggetto per cui OSCroute avrà come argomenti numeri che vanno da 1 a 16. In uscita si passa la parte di messaggio senza il canale MIDI. Questo viene trasmesso alle patch successive come valore a parte, poiché è necessario per la generazione del messaggio MIDI.

Pure Data permette di raggruppare in sub-patch frammenti di programma, creando un oggetto che inizi con le lettere "pd". La sub-patch "ctrl" riceve sull'input di sinistra il resto del messaggio osc e a destra il canale MIDI.

## PATCH CTRL

Il messaggio continua con il tipo di messaggio MIDI, control change oppure note (on/off).

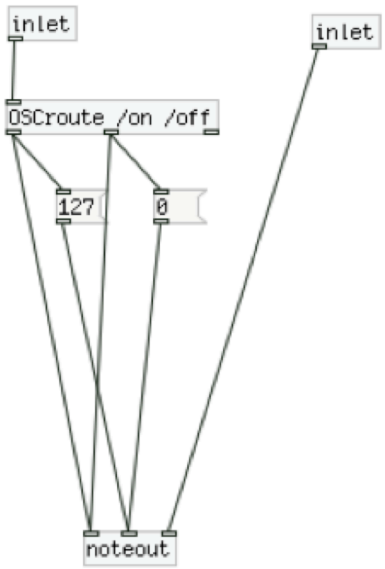
"Inlet" è un oggetto per "recuperare" i dati che la sub-patch ha preso in input. Il canale MIDI (l'input di destra) viene trasmesso alle relative sub-patch "cc" e "note".




## PATCH NOTE

Questa è la patch che genera messaggi di note on e note off. L'oggetto "noteout" riceve come argomenti,da destra verso sinistra, la nota MIDI (intero che va da 0 a 127), la velocity (per il note on si è impostata di default a 127) e il canale MIDI.

Si noti che il note off viene inviato come messaggio note on con velocity 0.

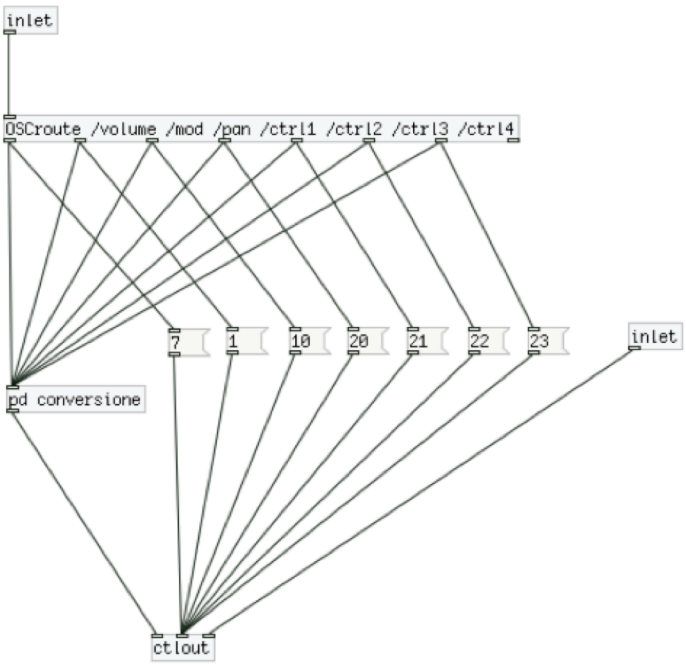



## PATCH CC

Questa patch genera messaggi MIDI control change.

OSCroute riceve il messaggio passatogli attraverso l'oggetto inlet e lo indirizza sull'uscita relativa. Ogni uscita è collegata con una sotto pach "conversione" e con un numero corrispondente al numero control change  secondo le specifiche MIDI.

L'oggetto "ctlout" riceve in input il valore numerico convertito dalla sub-patch "conversione", il numero control change e il canale MIDI.


inlet
OSCroute /volume /mod /pan /ctrl1 /ctrl2 /ctrl3 /ctrl4
7
1
10
20
21
22
23
inlet
pd conversione
ctlout

# 4.Suggerimenti e conclusioni

L'obbiettivo di questo progetto, quello cioè di sviluppare un'applicazione che permetta di sfruttare i sensori di movimento di uno smartphone per controllare parametri del suono, può dirsi certamente raggiunto.

Rispetto a come era stata pensata inizialmente, l'applicativo finale offre delle funzionalità aggiuntive che lo rendono uno strumento di maggior utilità (ad esempio l'implementazione di una tastiera grafica suonabile attraverso lo screen-touch).

Ciò nonostante rimangono diversi aspetti che possono essere migliorati e che in sintesi riguardano tre punti: fruibilità, funzionalità e performance collaborativa.

Si è già parlato nei capitoli precedenti delle motivazioni che hanno portato all'uso di Node.JS e  di Pure Data, entrambi fondamentali per il funzionamento dell'applicazione. Per contro l'utente che utilizza la web-app è obbligato a installare e a eseguire questi componenti aggiuntivi il che richiede anche una certa conoscenza informatica. Ripensare l'applicazione svincolandola da tali componenti rappresenta il primo punto da risolvere.

Per quanto riguarda le funzionalità, sono molte quelle che il MIDI implementa, mentre l'applicazione si limita a mandare messaggi di note on e note off e alcuni messaggi control change. L'idea potrebbe essere quella di trasformare lo smartphone in un controller MIDI a tutti gli effetti aggiungendo una lunga serie di funzionalità che il linguaggio MIDI mette a disposizione.

Inoltre è forte la spinta per lo sviluppo di nuove API Javascript, soprattutto da parte di Mozilla e Google, per sfruttare le nuove tecnologie adottate dagli smartphone. Ad esempio sono in fase di elaborazione le API per la lettura dei sensori di luminosità che potrebbe rappresentare un ulteriore add-on.

Infine, per quanto riguarda la performance collaborativa, si è detto come il server gestisce le connessioni dei diversi client. Semplicemente il client viene informato dal server se il canale MIDI è già in uso oppure no. Se il canale è utilizzato da un altro client e entrambi hanno mappato lo stesso parametro, l'effetto che ne risulterà potrà essere interessante ma anche fastidioso. Si possono pensare metodi per avere risultati di diverso tipo durante una

performance collaborativa, in modo tale che sia l'utente a scegliere se preferisce collaborare con l'altro client o passare su un altro canale.

Con queste considerazioni, concludo affermando la mia convinzione in merito alle potenzialità dell'applicazione: penso che il risultato di questi mesi di lavoro possa risultare effettivamente utile all'utente che ne farà uso, che sia uno studente, un musicista o chiunque voglia provare a produrre musica con questa applicazione.

# 5.Bibliografia

[5] Cantelon M., Harter M., TJ Holowaychuk and Rajlich N., *Node.js in Action*, Manning, 2013

[8] Curtis Roads, *The computer music tutorial*, Mit press, 1996

[7] Johannes Kreidler, *Loadbang: Programming Electronic Music in Pure Data*, Perfect Paperback, Mar 2009

[1] Mark Pilgrim, *HTML5: Up and Running*, O Reilly, 2010

[9] Matt Wright, *Open Sound Control: an enabling technology for musical networking*, Cambridge University Press, 2005

[3] Shelley Powers, *JavaScript Cookbook*, O' Reilly, 2011

## Sitografia

[2] AA.VV, *Html5, a vocabulary and associated APIs for HTML and XHTML*, http://www.w3.org/TR/html5/

[4] Block & Popescu, *DeviceOrientation Event Specifica*tion, http://w3c.github.io/deviceorientation/spec-source-orientation.html

[6] Fette & Melnikov, *The WebSocket Protocol*, http://www.w3.org/TR/2009/WD-websockets-20091222/

[10] Matt Wright, *The Open Sound Control 1.0 Specification*, http://opensoundcontrol.org/spec-1_0