POLITECNICO DI MILANO

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Specialistica in Ingegneria Elettronica

Interaction design, arte e creatività elettronica

Relatore:

Prof. Giancarlo Storti Gajani

Tesi di Laurea di:

Matteo Riva
matr. 705097

Anno Accademico 2011/2012

SOMMARIO

INDICE DELLE FIGURE

ABSTRACT

Scopo di questa tesi è analizzare l'utilizzo creativo degli strumenti open-hardware e open-software di prototipazione rapida, al fine di individuare le attuali potenzialità e le possibilità di sviluppo future di tali mezzi a servizio della creatività e delle arti digitali interattive.

Ad una panoramica dei principali e più diffusi strumenti disponibili attualmente, seguirà una descrizione dei lavori svolti personalmente per conto di uno studio di design come progettista, attraverso cui evidenziare le criticità di progetti di digital interactive art sotto i vari aspetti della produzione: concept, design, sensoristica, elettronica di controllo, interfaccia, development.

L'ultima parte della trattazione verterà sul livello di difficoltà nell'utilizzo di strumenti open source e open hardware da parte di artisti non specializzati in discipline elettroniche e informatiche, e sulle potenzialità degli strumenti disponibili sul mercato; dall'analisi dei limiti imposti dalle problematiche di progetto emergerà il profilo di una figura professionale necessaria per l'armonioso dialogo tra le discipline artistiche e le competenze tecniche: l'interaction designer.

ABSTRACT (ENGLISH)

The aim of this thesis is to analize creative uses of open-software and open-hardware tools for rapid prototyping, in order to identify available features and posssbilities for future development in digital interactive art.

Taking in consideration all the most common aspects of interaction design, we begin with an overview of the most important and popular tools available nowadays, followed by a description of some personal works, realized in collaoration with a design studio in Milan, through which point out all the common issues in a digital art project: concept, desing, sensors, control electronic, interface, development.

The last section will focus on the use of open source and open hardware tools by artists and designers without a technical background, and how interaction designers have to communicate between the technical world and the creative one in order to obtain the best artistic results.

1. Dal tinkering alla digital art

"Ebbene, ora abbiamo tutta questa tecnologia. Ma a cosa serve?"

John Thackara, Doors of Perception

1.1 Human Computer Interaction

Si può definire come *interaction design* quella disciplina in grado di progettare nuove modalità di interazione tra la persona ed il prodotto tecnologico, di unire la semplicità dell'utilizzo dell'oggetto alla funzionalità del software, di realizzare una tecnologia che risponda realmente ai bisogni dell'utenza e che sia di arricchimento alla vita sociale e culturale. L'interaction design si avvale dell'approccio, metodi e visioni di svariate discipline: dall'ingegneria del software ed hardware all'architettura, dal product design e graphic design alla sociologia, ergonomia e psicologia.

L'Interaction design è un sottoinsieme di competenze nel più vasto campo della "human-computer interaction" (HCI), riferito allo studio e alla progettazione di ogni tipo di comunicazione uomo-macchina.

La prima definizione di "interaction design" viene coniata nel corso degli anni '80 da Bill Moggridge e Bill Verplank per definire ciò che in precedenza era noto come "user-interface design", poiché ritenevano necessario considerare il design dell'interfaccia per gli utenti come una disciplina integrata al processo di sviluppo del prodotto. Ciò che si riteneva ormai inappropriato nella progettazione dei dispositivi interattivi era l'approccio meramente ingegneristico, relegando il confezionamento

dell'interfaccia con l'utenza ad una fase successiva al lavoro degli sviluppatori di codice e dei progettisti hardware. Il suggerimento era rivolgere la ricerca non solo al miglioramento della funzionalità del prodotto tecnologico, ma al tipo di attività che si poteva svolgere con esso e di conseguenza considerare la centralità dell'utente all'interno di questo processo.

È su questa scia si pensiero che in quel periodo si inizia a parlare di "user-friendly", cioè di come rendere la tecnologia usufruibile anche da utenti non specializzati e senza competenze tecniche troppo approfondite: questo passaggio è da molti considerato come l'inizio della rivoluzione tecnologica che ha sconvolto la società moderna e ridefinito le relazioni e il modo di comunicare dell'uomo moderno. Il primo segnale di mutamento in questa direzione è il passaggio da *command line interface* (CLI) a *graphical user interface* (GUI) per i desktop computer, e l'introduzione delle interfacce a puntatore e del mouse, rivoluzionaria periferica di controllo; pionieri in questo campo sono i tecnici dei laboratori Xerox, che nel 1981 commercializzano il primo sistema controllato con puntatore con il nome di Xerox Star; lo stesso concetto viene prontamente rielaborato da Apple, con Apple Lisa nel 1983 e il celebre Macintosh nel 1984. La prima versione di un'interfaccia a puntatore a colori è da attribuire ad Atari (Atari 520ST, 1985), seguita di soli due mesi da Commodore International (Amiga 1000, 1985).

Da allora la tecnologia iniziò ad essere compresa, accolta e utilizzata da tutti in maniera estesa in ogni campo, e sono sotto gli occhi di tutti al giorno d'oggi le possibilità di comunicazione con i dispositivi tecnologici: la seconda grande rivoluzione dopo il mouse è considerata l'introduzione del *touch-screen*, che ha eliminato ogni barriera comunicativa tra l'utilizzatore e la risposta visiva dei dispositivi. Viene introdotto un altro concetto che va ben oltre allo user-friendly: le ricerche si concentrano ora sul creare la cosiddetta *natural user interface*, una modalità di utilizzo che ricalchi le abitudini e le modalità comunicative proprie dell'essere umano; la tecnologia compie un ulteriore passo verso l'uomo, per integrarsi maggiormente nella quotidianità in maniera sempre più "invisibile" e

discreta. Significativa espressione di questo movimento è l'incredibile successo commerciale della periferica di controllo Microsoft Kinect, nata come semplice controller per videogame ed ora pronta ad essere inserita pienamente nell'ultima release del sistema operativo Windows, che contempla il riconoscimento dei gesti dell'utente, secondo la formula utilizzata nella campagna pubblicitaria: *"You are the controller"*.

1.2 Digital interactive art

L'interaction design è una disciplina che mette la prototipazione al centro della sua metodologia; poiché deve creare esperienze significative tra esseri umani ed oggetti, in un certo senso è responsabile dell' "anima" che quei determinati oggetti avranno, del loro comportamento e della loro "personalità virtuale". Scrive Massimo Banzi, ideatore di Arduino: *"L'interaction design incoraggia il design attraverso un processo iterativo basato su prototipi sempre più fedeli. Questo approccio (che fa parte anche di alcuni tipi di design "convenzionale") può essere esteso per includere la prototipazione con la tecnologia; in particolare, la prototipazione con l'elettronica"*[1].

Il campo specifico dell'interaction design che interessa l'elettronica è il *Physical Computing* (o *Physical Interaction Computing*), ovvero la prototipazione elettronica per fornire nuovi materiali (in senso lato) per designer e artisti. Il suo scopo principale è la progettazione di oggetti interattivi che possano comunicare con gli esseri umani usando sensori e attuatori controllati da un comportamento implementato in forma di software ed eseguito da un computer o da un microcontrollore. Talvolta si suole indicare questo genere di dispositivi con il più generico termine di *sistemi embedded*, ovvero sistemi elettronici di elaborazione a microprocessore progettati appositamente per una determinata applicazione (*special purpose*, non riprogrammabili dall'utente per scopi differenti) e spesso con una piattaforma hardware progettata e costruita ad

[1] Banzi, M., *Getting Started with Arduino*, O'Reilly Media, pag. 2

[2] i concetti contenuti in questo paragrafo fanno riferimento agli studi sulla human-computer interaction compiuti da Chris Crawford alla fine degli anni '90 [2]

hoc, integrati nel sistema che controllano e in grado di gestire tutte o parte delle funzionalità che il sistema implementa.

La miniaturizzazione dei componenti elettronici e il loro costo decrescente hanno fatto sì che tali sistemi non fossero più soltanto rivolti alla realizzazione di infrastrutture e servizi, ma che gradualmente fossero impiegate nel mondo dell'intrattenimento e dell'arte. All'interno della *new media art*, esplosa con la diffusione del televisore e degli strumenti di registrazione e riproduzione video, grande spazio trovò la *digital interactive art*, con l'utilizzo di sensori, attuatori e logica di controllo a favore di un'esperienza coinvolgente e personale per lo spettatore nei confronti dell'opera d'arte, in un contesto mutevole e più soggettivo di espressione e percezione della stessa, in perfetta sintonia con la ricerca dell'effimero e dell'emozione soggettiva propri dell'arte contemporanea.

1.3 Prototipazione rapida e tinkering

Per molti anni però le figure professionali del designer (o artista) e del progettista elettronico (sia per il software che per l'hardware) sono rimaste ben distinte, portando ciascuno competenze differenti e vicendevolmente esclusive. Il risultato era non soltanto un rallentamento del processo produttivo, ma anche un grande limite da parte del creativo nel non poter sperimentare direttamente con il dispositivo; la maggior parte degli strumenti tecnologici disponibili fino a pochi anni fa erano infatti utilizzabili soltanto da specialisti e richiedevano una consistente preparazione teorica per il loro utilizzo. Negli ultimi anni i microcontrollori hanno cominciato a diventare più economici e facili da usare, permettendo di creare strumenti migliori in minor tempo, e di essere utilizzati anche senza necessità di troppe competenze. Strumenti come Arduino hanno permesso anche a principianti di apprendere rapidamente quanto basta per costruire prototipi e sperimentare nel campo dell'interazione con un investimento minimo in termini economici

e di tempo, agevolando in tal modo il processo creativo a favore della componente artistica del risultato finale.

La filosofia di utilizzo di tali dispositivi di prototipazione rapida (che analizzeremo più avanti nel dettaglio) consente sempre agli specialisti di impiegare un metodo classico di progettazione e poi implementare velocemente il loro circuito, ma contemporaneamente permette a chiunque non abbia conoscenze approfondite di elettronica di prototipare in tempo reale, senza schemi di progetto, e di sperimentare varie possibilità arrivando a volte a risultati inattesi, sempre stimolanti per la creatività e spesso ben oltre le aspettative. Questo è il processo definito *tinkering*: sperimentare liberamente con il mezzo tecnologico in maniera flessibile, lasciando spazio alla casualità e all'improvvisazione, trovando in tal modo qualcosa di imprevisto e inatteso.

1.4 Open source e open hardware

Questo modo di creare nasce sulla scia del grande successo di linguaggi/ambienti di programmazione come Processing, nato come strumento per insegnare in modo semplice i rudimenti della programmazione e presto diventato ambiente ideale per creare arte grafica digitale e non solo.

Quello che accomuna queste novità nel mondo software e hardware, sia per quanto concerne la semplicità di utilizzo che per un'apertura verso differenti applicazioni, è anche l'ingrediente principale del loro successo e della loro diffusione: si tratta nella quasi totalità dei casi di progetti *open source* e *open hardware*, liberamente modificabili e gratuitamente utilizzabili da chiunque. Si è assistito alla nascita di una vera e propria cultura dell'open source sin dalle prime distribuzioni del sistema operativo Linux, fino ad arrivare all'attuale offerta di un numero considerevole di programmi, ambienti di sviluppo, librerie, utility e quant'altro, disponibili per ogni piattaforma, sistema operativo e linguaggio di sviluppo.

Se per quanto riguarda il software open source (ovvero software di cui viene distribuito liberamente anche il codice sorgente, permettendone la modifica e la personalizzazione da parte di ogni utente) l'enorme diffusione di internet e delle comunità virtuali può facilmente spiegare il fenomeno (anche per la facilità con cui in rete ci si possono scambiare risorse e mettere a disposizione codice), è interessante notare come anche per l'hardware sia stato possibile un meccanismo molto simile, sebbene forse più contenuto perché meno immediato nelle modalità di condivisione: una comunità virtuale di progettisti coopera nella creazione di dispositivi mettendo a disposizione e modificandone gli schemi elettrici, producendoli autonomamente e condividendo poi i risultati delle proprie esperienze per migliorare il dispositivo o per sottolinearne nuove potenzialità.

Ancor più interessante è il risultato di questo processo: questi meccanismi hanno infatti velocemente portato alla creazione di prodotti che rispondevano alle esigenze che un vastissimo insieme di persone aveva espresso, generando strumenti che per la loro stessa modalità di progettazione sono immediatamente diventati di largo consumo e di incredibile interesse per tutti, alimentando ancor di più l'attività di condivisione e di avanzamento del prodotto stesso. Il supporto dei forum dedicati e delle community sorte attorno a questi oggetti è tutt'oggi il motore principale dello sviluppo degli stessi e la fonte più proficua di apprendimento e di aggiornamento riguardo le nuove tecnologie a disposizione di artisti, designer, hobbisti e progettisti. Possiamo dire che il mondo dell'arte digitale poggia oggi le sue basi e le sua possibilità di sviluppo sulla libera condivisione delle informazioni e delle esperienze, creando quella che può davvero essere definita una nuova forma d'arte sociale, non necessariamente nel contenuto dell'opera finale, ma sicuramente nel percorso attraverso cui la realizzazione di quell'opera è stata resa possibile.

La radice psicologica di questi fenomeni sta chiaramente nell'attitudine, propria della natura umana, alle relazioni tra esseri viventi e al bisogno di interazioni sociali, che si manifesta in questo caso anche nell'aggregarsi nel

costruire qualcosa insieme, sentirsi parte di una comunità attiva, valorizzati e stimolati nei propri interessi e realizzazioni. È stato appositamente coniato il termine *social computing* per descrivere i sistemi digitali che supportano interazione sociali online; è da rimarcare che queste interazioni non riguardano necessariamente la comunicazione con persone conosciute o ben definite, ma ci consentono ugualmente di interagire in quella che ormai è la società delle relazioni digitali, parallela alla vita reale ma con un'importanza ormai consolidata e innegabile nel nostro modo di esprimerci e sentirci parte della società stessa.

2. Physical Computing

"Until recently, rendering bits into human-readable form has been restricted mostly to displays and keyboards – sensory deprived and physically limited. By contrast, "tangible bits" allow us to interact with them with our muscles as well as our minds and memory"

Nicholas Negroponte, co-fondatore di MTI Media Lab

2.1 Physical Computing

La *computer revolution* descritta nel capitolo precedente ha reso computer e dispositivi tecnologici alla portata di chiunque, sia a livello economico che di usabilità, e questo è ormai un percorso compiuto che ha raggiunto pienamente i suoi obiettivi, rendendo la comunicazione visiva e tattile con i dispositivi un comportamento naturale per ciascuno di noi. Il passo ulteriore che dall'ultimo decennio la comunità di sviluppatori di human-computer interfaces sta compiendo è la realizzazione di computer che comunichino con tutto il resto del nostro essere nelle più svariate maniere: per molto tempo l'interazione "classica" si è limitata a schermo, tastiera e mouse. Per allargare l'orizzonte ed esplorare le possibilità future dell'informatica dobbiamo allontanarci da questo stereotipo di computer, pensando invece a dispositivi liberi in forma e capacità, che si adattano alle nostre necessità di automazione o interazione.

In una visione semplicistica potremmo dire che l'evoluzione dell'informatica insegue il pensiero umano. Le applicazioni e gli studi di

Artificial Intelligence (AI), che fanno uso dell'informatica per imitare (e forse un giorno sostituire) comportamenti umani, sono da sempre stati considerati molto importanti all'interno della *computer science*. Le tecnologie e la teoria utilizzate nel physical computing sono equivalenti a quelle proprie della robotica e dell'intelligenza artificiale, con una leggera diversità nelle applicazioni tipiche. L'approccio del physical computing arriva da una differente area di interesse, chiamata *Intelligence Amplification* (IA): invece che imitare o sostituire comportamenti umani, l'obiettivo è di aggiungere potenzialità ai comportamenti e ai metodi di comunicazione, siano essi tra persona e persona o tra uomo e macchina.

L'offerta del computer (intendendo da qui in avanti con questo termine tutta la vastità dei dispositivi di calcolo ed elaborazione disponibili) come "mezzo" è di arricchimento rispetto agli altri media classici perché consente di spezzare la linearità: se i media classici sono classificati come *linear media* (rendono infatti disponibile l'informazione in modo sequenziale e strettamente consecutivo; pensiamo ad una videocassetta o alla riproduzione di una registrazione audio), possiamo invece parlare di *random access media*, con la possibilità di accedere a differenti parti di memoria (elaborandole, visualizzandole o semplicemente traendone generiche informazioni) come se fossero contigue. I computer riducono le barriere spazio-temporali nell'operazione di manipolazione ed elaborazione delle idee e dei contenuti; in tal senso essi si avvicinano maggiormente al modo di pensare proprio degli esseri umani, in maniera dinamica.

A prescindere dalla visualizzazione di contenuti (campo di interesse della prima era dei new media moderni, ma ora abbastanza sorpassato), l'inserimento di un computer in un sistema di interfaccia concreto e tangibile offre la possibilità di rendere più complesse ed interessanti le relazioni tra azione umana e risposta della macchina, attraverso la scrittura di programmi che indichino al dispositivo le scelte da compiere a seguito delle azioni dell'utilizzatore. L'approccio della IA presuppone che la parte "interessante" dell'interazione sia compiuta dall'utilizzatore, mentre l'unico compito del dispositivo è di interpretare le intenzioni dello

specifico gesto dell'utente, senza necessariamente guidarlo nelle scelte o dare a sua volta istruzioni: da un interfaccia *user friendly* pensata per istruire velocemente l'utente con indicazioni e suggerimenti, l'interfaccia ora diventa il più possibile invisibile, l'interazione tende ad essere "naturale".

2.2 Concetti base di interazione

Il Physical computing riguarda essenzialmente la creazione della conversazione tra il mondo fisico reale e il mondo virtuale del computer. Ciò che rende possibile questo scambio reciproco è un insieme di processi di trasduzione, cioè conversioni di energia tra varie forme. Nella progettazione di un sistema interattivo spesso la maggior parte del lavoro si risolve nell'individuare il corretto flusso di informazioni e le modalità per interpretarlo (o restituirlo) attraverso sensori e attuatori adeguati. La bontà di un sistema interattivo risiede nella naturalezza del suo comportamento; una classica linea guida di progetto tenderà a far assomigliare il comportamento del sistema all'evolvere di una comune conversazione, rispettando il corretto bilanciamento delle tre componenti fondamentali: ascoltare, parlare e pensare.[2]

2.2.1 Ascoltare

Scomponiamo una comunicazione generica, e poniamoci innanzitutto nei panni del dispositivo che deve ascoltare l'utente.

Durante una conversazione reale non sussistono precisi momenti alternati di ascolto e di comunicazione; l'essere umano è in grado (entro certi limiti) di parlare ed ascoltare contemporaneamente. Quando l'ascoltatore vuole interrompere il parlatore invia inconsciamente e in

[2] i concetti contenuti in questo paragrafo fanno riferimento agli studi sulla human-computer interaction compiuti da Chris Crawford alla fine degli anni '90 [2]

maniera automatica dei segnali fisici istintivi, recepiti dal parlatore come la volontà di intervenire nella comunicazione.

La comunicazione presuppone anche delle naturali pause, principalmente nella preparazione delle risposte da parte dei due interlocutori, o per lasciare all'ascoltatore il tempo di elaborare e recepire le informazioni che gli abbiamo comunicato.

Altro elemento tipico di una conversazione è la verifica della comprensione di quello che stiamo comunicando, spesso attraverso domande spontanee suscitate da silenzi prolungati o da atteggiamenti dell'ascoltatore.

Scopo di un progettista di sistemi interattivi è riportare in maniera naturale queste e molte altre caratteristiche della conversazione naturale, analizzando soprattutto le componenti inconsce o istintive, che si rivelano spesso essere le più fondamentali. La difficoltà risiede proprio nel tradurre in termini di sensori e di logica di controllo questi stimoli, che il dispositivo dovrà essere in grado di decifrare correttamente; ovviamente non in tutti i tipi di interazione sarà necessario un grado così elevato di accuratezza nel riprodurre le condizioni di una conversazione uomo-macchina che sia il più naturale possibile: obiettivo dell'analisi di progetto sarà dunque individuare quali siano i canali comunicativi principali dell'utente che il dispositivo dovrà considerare in merito all'applicazione particolare che dovrà svolgere.

2.2.2 Parlare

È stato osservato che quando è il nostro turno di ascoltare mostriamo aspettative simili sull'andamento della conversazione sia che stiamo comunicando con altri esseri umani che con qualunque altro tipo di interlocutore. Pensiamo ad esempio alla comunicazione con un animale domestico: è naturale pensare che reagisca razionalmente ai nostri stimoli, e che lo faccia con tempi di reazioni paragonabili a quelli di un essere umano. La stessa aspettativa è quella che nutriamo inconsciamente nei

confronti dei dispositivi elettronici, ed è quella che maggiormente influenza la soddisfazione nell'utilizzo degli stessi.

Il dispositivo deve essere progettato perché reagisca in tempi naturali (e con questo intendiamo né troppo velocemente, perché sarebbe troppo artificioso, né troppo lentamente, o si perderebbe la consequenzialità nelle interazioni) oppure in modo che l'utente possa velocemente abituarsi a considerarli come naturali.

La capacità dei microprocessori di svolgere milioni di operazioni al secondo, e quindi di essere "pronti" all'interazione successiva prima che l'utente se ne possa accorgere, non va sopravvalutata nei sistemi embedded, le cui performance devono sempre mantenere la leggerezza sufficiente a celare agli occhi dell'utente i tempi di processo e di calcolo, pena la perdita del contatto naturale col dispositivo.

A tal proposito è utile una preventiva analisi delle aspettative sui tempi di risposta del sistema da sviluppare, accompagnata da un accorto inserimento di feedback da parte del dispositivo; la più semplice applicazione di questo principio è osservabile nei led di stato di tutti i dispositivi elettronici con cui abbiamo a che fare: anche se la risposta tarda a compiersi, l'accensione della classica "spia" comunica all'utente che il dispositivo ha perlomeno accolto la richiesta, e l'utente può aspettare il suo completamento, anche se richiede tempi elevati, senza alterare la naturalezza dell'interazione. Bisogna tuttavia sottolineare che nella maggior parte dei casi l'utente che interagisce con un sistema interattivo dedicato ha richieste molto più stringenti in termini di tempo e di soddisfazione delle richieste rispetto all'interazione con un classico computer: in quest'ultimo caso la tecnologia è manifesta, e i tempi di calcolo sono più accettabili per l'esperienza che abbiamo delle nostre interazioni con le macchine; nei sistemi embedded invece la tendenza è di nascondere la tecnologia, azzerando però questa tolleranza di attesa da parte dell'utente.

2.2.3 Pensare

La contemporaneità dell'ascoltare (ovvero leggere da sensori) e parlare all'utente (ovvero azionare attuatori e indicatori) deve essere coordinata da una logica che renda armoniosa l'interazione, corretta l'interpretazione dei messaggi da parte dell'utente e razionali le risposte che il sistema fornisce. Ovviamente non è realizzabile una vera contemporaneità, ma il tutto è reso possibile dall'elevata velocità con cui si possono eseguire operazioni in un microcontrollore (o microprocessore): compito del progettista è far sì che la somma dei tempi necessari all'acquisizione e interpretazione delle informazioni, all'elaborazione delle risposte e all'attuazione delle stesse sia inferiore a tempi sensibili all'utente.

Un particolare importante di cui avere cura è che i segnali provenienti dai trasduttori della sensoristica del sistema vanno interpretati ragionevolmente pensando alla causa che li ha generati, in termini di banda e di significato del segnale: a tal proposito si ricorre a metodi di pulizia del segnale, quali *edge detection*, *threshold setting*, *peak finding*, *debouncing*, *avearaging* e *smoothing*, per citarne alcuni.

3. Strumenti creativi

"To use a tool on a computer, you need do little more than point and click; to create a tool, you must understand the arcane art of computer programming"

John Maeda (insegnante MIT), "Creative code"

3.1 Strumenti hardware

Sono a disposizione attualmente moltissimi strumenti che rendono accessibile a tutti la progettazione di semplici sistemi embedded, in particolare tutta quella fascia di prodotti denominati "strumenti di prototipazione rapida"; lo scopo per cui inizialmente furono ideati e sviluppati questi dispositivi infatti era quello di fornire ai progettisti elettronici un modo veloce ed economico di sperimentare i propri sistemi complessi prima di iniziare la fase di produzione, per ottimizzare al meglio il progetto prima di iniziare la fase di investimento nella costruzione di prototipi. L'evoluzione di tali strumenti, affiancata dall'intuizione di poter rendere l'elettronica accessibile per altre finalità non strettamente professionali, ha portato alla diffusione anche nel campo artistico, dell'intrattenimento, della pubblicità e della didattica. È il caso sia di Arduino[3], concepito come strumento per insegnare l'elettronica attraverso semplici sperimentazioni pratiche, che del più recente Raspberry PI[4], il computer completo ed economico (disponibile attualmente a meno di 30$) che si rivolge a chi vuole imparare le basi dell'informatica e della

[3] vedi paragrafo 3.1.1
[4] vedi paragrafo 3.1.2

programmazione: entrambi hanno velocemente preso piede nei processi creativi di designers e artisti.

Questo nuovo punto di vista che ha avvicinato l'elettronica alle discipline umanistiche ha avuto come conseguenza anche un interessante fenomeno di reinterpretazione dei dispositivi elettronici: non di rado si sono verificati casi in cui un prodotto commerciale di larga diffusione sia diventato oggetto del cosiddetto hacking, come nel caso della periferica *Microsoft Kinect*[5] o, sempre nel campo della computer vision, della videocamera *Sony PS3 eye*, acquistabile a prezzo contenuto (grazie alla enorme popolarità della consolle *Playstation*) e facilmente modificabile per ottenere una camera sensibile agli infrarossi con buona risoluzione.

La comunità dei creativi ha saputo leggere oltre il semplice dispositivo elettronico pensato con uno scopo specifico, intravedendo le sue possibili e talvolta inusuali applicazioni: si assiste ad un riguardevole fenomeno di ricerca di dispositivi che siano personalizzabili ("*hackerabili*", come troviamo nei forum in rete) e riutilizzabili semplicemente da utenti che per passione o per lavoro seguono la tendenza a digitalizzare tutto ciò che ci circonda, al contempo fornendo però anche una certa umanità e valore artistico alla tecnologia che ci circonda; dal punto di vista sociale sembra un riappropriarsi dell'umanità perduta nel processo di tecnologizzazione e automazione che rapidamente negli ultimi decenni ha accolto e reso indispensabile la tecnologia nella realtà quotidiana.

[5] vedi paragrafo 3.3

3.1.1 Arduino

La cultura romantica ha reso popolare Arduino da Dadone, conosciuto anche come Arduino d'Ivrea, re d'Italia dal 1002 al 1014, riconoscendo in lui uno dei primi esponenti della lotta per la liberazione dell'Italia dalla dominazione straniera. La cultura digitale ha aggiunto popolarità internazionale a questo nome grazie ad una piccola ma geniale piattaforma hardware ideata nel 2005 da un team composto da Massimo Banzi (anima del progetto) David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis, tutti orbitanti attorno all'allora Design Institute di Ivrea, e sviluppata successivamente all'università di Los Angeles in Colombia.

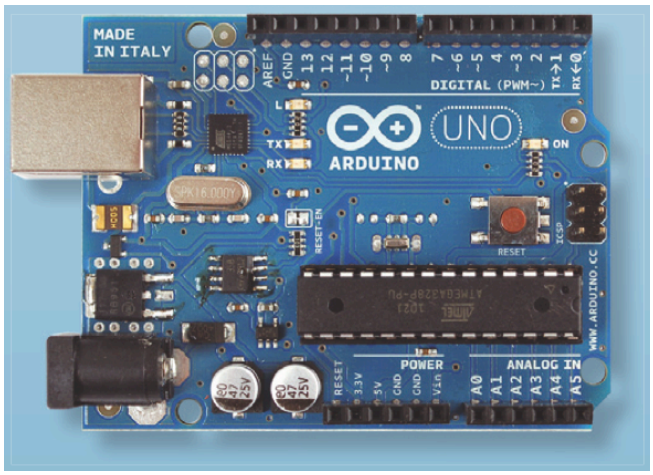


Figura 1 – La scheda Arduino UNO

Arduino è una piattaforma hardware di prototipazione rapida basata su microcontrollore ATMEL: sostanzialmente è una scheda di input/output di semplice e immediato utilizzo, grazie anche ad un ambiente di sviluppo dedicato (Arduino IDE) che fa uso di una libreria Wiring[6] per

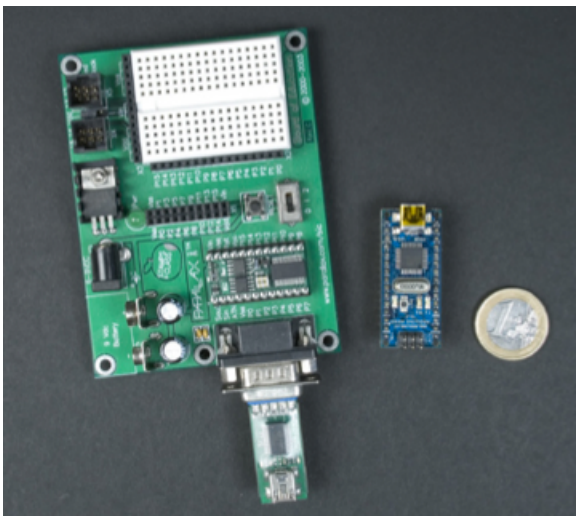


Figura 2 - Basic Stamp e Arduino Nano

[6] Wiring è un ambiente di programmazione open-source per impieghi su schede elettroniche

semplificare la scrittura di programmi in C e C++. Lo schema di progetto dell'hardware è open-source. Lo scopo era di rendere disponibile a progettisti, studenti e hobbisti un dispositivo di sviluppo semplice (che non richiedesse cioè competenze di elettronica approfondite) e al contempo più economico rispetto ai sistemi di prototipazione allora presenti sul mercato. Tra tutti ricordiamo Basic Stamp, diffuso tra gli hobbisti già dagli anni '90, basato su linguaggio di programmazione Basic. Arduino è funzionalmente molto simile a Stamp, ma introduce maggiore semplicità e un costo nettamente inferiore (quasi un quarto del prezzo a parità di caratteristiche).

3.1.1.1 Configurazione hardware

Il core della scheda è costituito da un microcontrollore a 8 bit AVR prodotto dalla Amtel; le schede ufficiali fanno uso dei chip della serie megaAVR (nello specifico, troviamo i modelli Atmega8, Atmega 168, Atmega328 e Atmega 1280), ma si trovano in commercio realizzazioni con altri microcontrollori equivalenti. Sulle schede sono presenti un regolatore di tensione a 5V, che stabilizza l'alimentazione della scheda e fornisce i riferimenti per i livelli logici delle porte di input/output digitale, e un oscillatore al quarzo da 16MHz (in alcune varianti sostituito da un risonatore ceramico); alcune eccezioni, come per esempio il modello LilyPad, hanno una frequenza di funzionamento fissata a 8 MHz, e non implementano lo stabilizzatore di tensione a 5V. All'acquisto la scheda è pre-programmata con un bootloader che semplifica il caricamento dei programmi nella memoria Flash incorporata nel chip; uno dei grossi vantaggio di Arduino è che non necessita di un programmatore esterno. Concettualmente tutte le schede vengono programmate attraverso una porta seriale standard RS-232, con differenti implementazioni hardware a seconda del modello: le prime schede seriali Arduino contengono un semplice circuito traslatore di livelli che permette la conversione da RS-232 ai livelli logici dei segnali TTL. Le versioni più recenti vengono gestite via USB, e sono equipaggiate di un adattatore USB-seriale come l'FT232

della FTDI. Alcune versioni, specialmente le più miniaturizzate, prevedono l'uso di una scheda o un cavo adattatore USB-to-Serial separato.

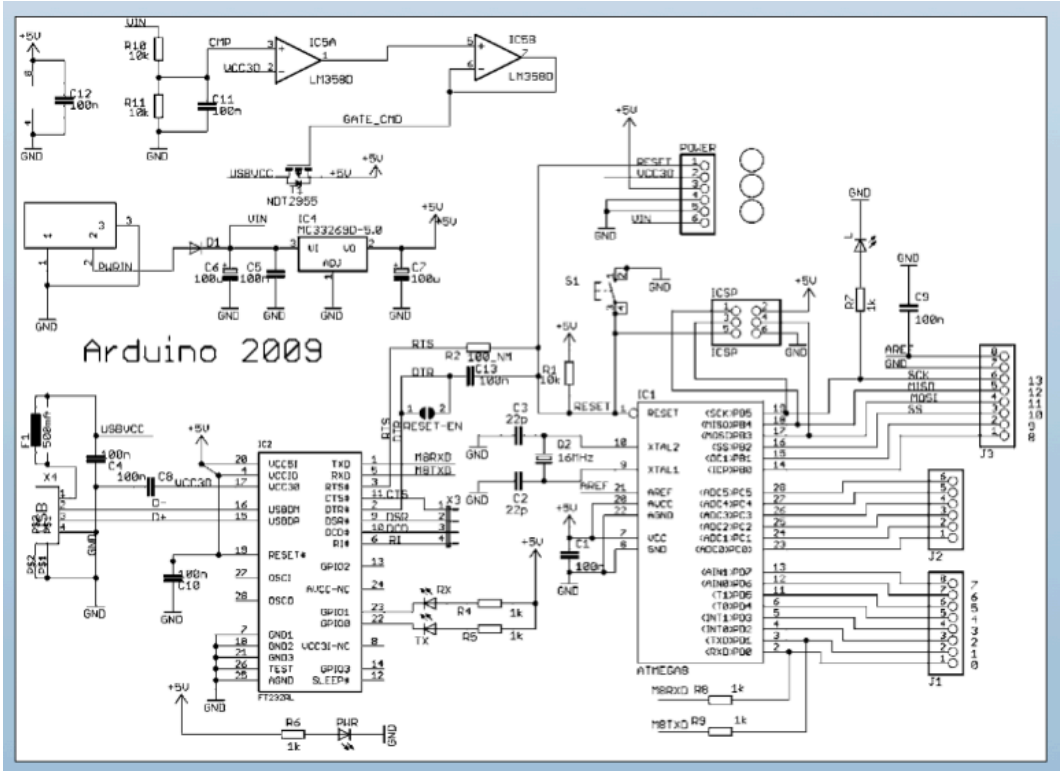


Figura 3 – Schema elettrico di Arduino Duemilanove

Le schede Arduino dispongono di numerosi connettori di input/output utilizzabili come estensione per altri circuiti esterni, nonché il collegamento di sensori e attuatori, fornendo un comodo controllo attraverso la porta seriale del computer. Il numero e la tipologia degli ingressi/uscite varia a seconda del modello della scheda: le più diffuse (Arduino UNO e Arduino Duemilanove) sono basate sul modello Diecimila, che offre 14 connettori per input/output digitale, 6 dei quali possono generare segnali in PWM, e altri 6 connettori per input analogici. Questi pin sono disponibili nella parte superiore della scheda mediante connettori femmina a passo standard 0,1 pollici. Questa particolare configurazione presente su tutte le più diffuse schede Arduino ha portato alla produzione di numerosissime *Arduino shield*, ovvero schede dedicate per funzioni particolari, pensate appositamente per essere collegate ad

Arduino ed estenderne le capacità: possiamo trovare in commercio shield per controllare motori, per riprodurre segnali audio, per ricevere ed elaborare segnali provenienti da svariati sensori, connettere display, comunicare wireless con altri dispositivi, e così via. Le vaste funzionalità disponibili costituiscono un campionario perfetto per chi non ha competenze tali da saper progettare autonomamente un circuito di elaborazione, acquisizione o controllo, ma vuole utilizzarlo in maniera semplice attraverso Arduino per realizzare il proprio sistema embedded o la propria installazione interattiva.

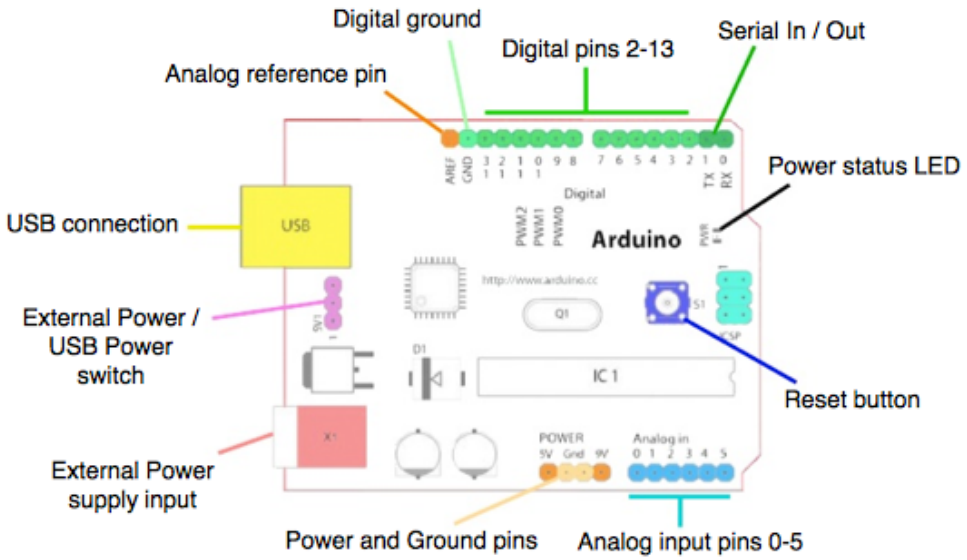


Figura 4 - Connessioni di una scheda Arduino UNO

3.1.1.2 Ambiente di sviluppo

L'ambiente di programmazione integrato (IDE) di Arduino è un'applicazione multipiattaforma scritta in linguaggio Java, derivata dell'IDE creato per il linguaggio di programmazione Processing[7] e adattato al progetto Wiring. Per consentire la stesura di codice sorgente in maniera intuitiva e semplificata per gli utenti inesperti, il programma include alcune funzionalità come il *syntax highlighting*, il controllo delle parentesi

[7] vedi paragrafo 3.2.1

e l'identificazione automatica delle istruzioni. L'editor è inoltre in grado di compilare e caricare il programma eseguibile sul microcontrollore in una sola passata, con un solo click dell'utente. L'IDE di Arduino usa la GNU *toolchain* e la *AVR Libc* per compilare i programmi, e si avvale del tool *avrdude* per caricarli sulla scheda.

3.1.1.3 Licenza di distribuzione

Gli schemi hardware di Arduino vengono distribuiti, in modo da poter essere utilizzati nei termini legali, con una licenza *Creative Commons Attribution Share-Alike 2.5*, e sono disponibili sul sito ufficiale Arduino. Per alcune versioni della scheda sono disponibili anche il layout e i file di produzione. Il codice sorgente per l'Ambiente di sviluppo integrato e la libreria residente sono disponi- bili, e concessi in uso, secondo i termini legali contenuti nella licenza *GPLv2*. La *GNU General Public License* è una licenza per software libero. È comunemente indicata con l'acronimo GNU GPL o semplicemente GPL. Contrariamente alle licenze per software proprietario, la GNU GPL assicura all'utente libertà di utilizzo, copia, modifica e distribuzione, ed è oggi la più diffusa licenza per il software libero: è sufficiente allegare un file contenente il testo della licenza o un collegamento web ad essa per rilasciare un software sotto tale licenza, e il licenziatario ottiene automaticamente il permesso di modificare il programma, copiarlo e ridistribuirlo (con o senza modifiche), gratuitamente o a pagamento. Rispetto alle altre licenze di software libero la GPL è classificabile come *persistente* e *propagativa*. È *persistente* perché impone un vincolo alla redistribuzione: se l'utente distribuisce copie del software deve farlo secondo i termini della GPL stessa. In pratica, deve distribuire il testo della GPL assieme al software e corredarlo del codice sorgente o di istruzioni per poterlo ottenere. Questa è la caratteristica principale della GPL, il concetto ideato da Richard Stallman e da lui simpaticamente battezzato *copyleft*. Il suo scopo è di mantenere libero un programma una volta che esso sia stato posto sotto GPL, anche se viene migliorato correggendolo e ampliandolo. È *propagativa* perché

definisce nel testo una particolare interpretazione di *codice derivato*, tale che in generale l'unione di un programma coperto da GPL con un altro programma coperto da altra licenza può essere distribuita sotto GPL.

3.1.2 Raspberry PI

L'arrivo sul mercato dei circuiti integrati e dei microcontrollori aprì nuove possiblità di realizzazioni elettroniche "intelligenti" incrementando l'interesse e la necessità di competenze informatiche anche da parte dei sistemisti elettronici, che dovettero occuparsi anche di programmazione per la scrittura del firmware in esecuzione permanente sull'hardware computazionale inserito nel sistema. Il processo di avvicinamento è avvenuto anche nell'altro senso, con una sempre maggior complessità e miniaturizzazione dei comupter che portò gli informatici ad una analoga necessità di aggiornamento nei confronti delle architetture hardware su cui programmare. Il punto di contatto tra le due realtà è sempre più vicino nei sistemi embedded: se da una parte i microcontrollori si evolvono mettendo a disposizione strumenti per comunicare sempre più facilmente con i computer, contemporaneamente compaiono schede PC-embedded dotate di pin di input/output per interfacciarsi con circuiti esterni. La scheda *Raspberry PI* nasce proprio sul confine di questi due mondi: è un vero e proprio computer su scheda singola che rende disponibili direttamente sulla scheda una serie di pin di I/O, una interfaccia seriale, una SPI, ina I2C e alimentazione a 3,3V e 5V.

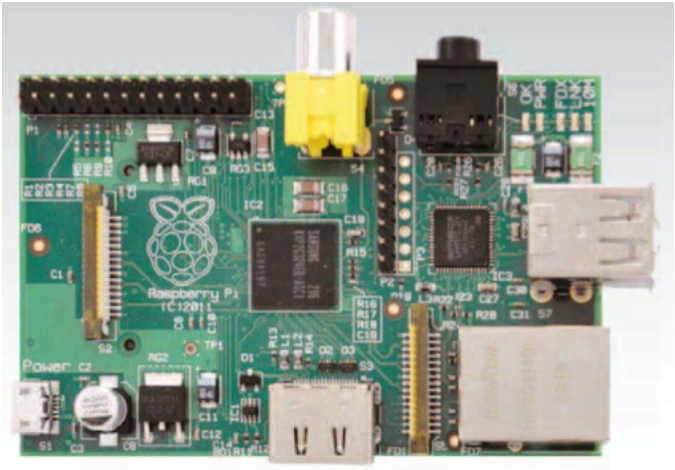


Figura 5 - Raspberry PI

3.1.2.1 Configurazione hardware

Raspberry PI È stato sviluppato nel Regno Unito dalla *Raspberry PI Foundation*, con l'intento di realizzare un computer a basso costo per stimolare l'insegnamento dell'informatica nelle scuole, in modo particolare nei paesi in via di sviluppo.

Il cuore del computer è il SoC Broadcom BCM2835, che include un processore ARM da 700 MHz, un processore grafico VideoCore IV, un processore di segnali digitali e 256 Mb di memoria RAM condivisi con la GPU. Non sono previsti dispositivi dischi fissi o allo stato solido, in quanto il sistema operativo e la memoria di massa sono allocati su SD Card, che svolge anche il ruolo di unico device di boot[8].

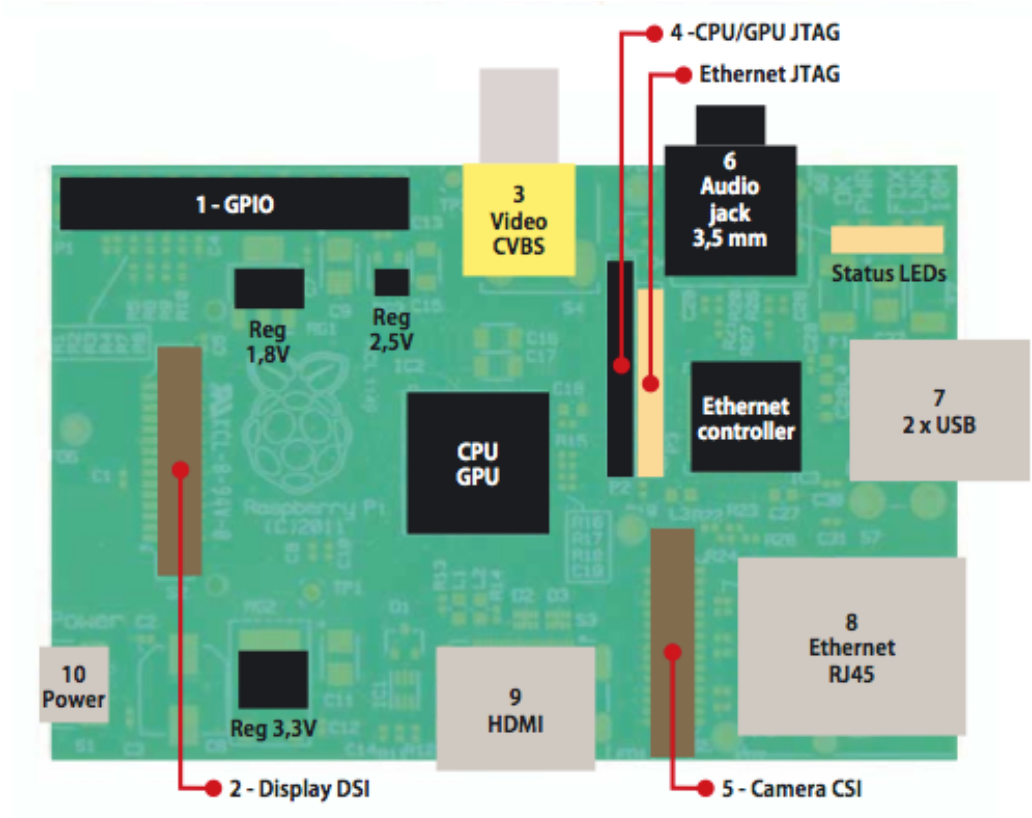


Figura 6 - Schema dell'hardware sulla scheda Raspberry PI

[8] Il termine "boot" è l'abbrevazione di "bootstrap", letteralmente il laccetto posto dietro a scarponi e stivali che serve ad aiutare a calzarli, tirandolo da dietro

Facendo riferimento allo schema sovrastante *[Figura 6]*, analizziamo i connettori e i componenti principali presenti sulla scheda:

1. Connettore GPIO: 26 pin in totale, su due file di 13 pin ciascuna a passo standard 2,54 mm; da questo connettore sono accessibili l'alimentazione a 3,3V e a 5V, la massa, e una serie di pin di input/output che possono essere riconfigurati per ulteriori funzioni come I2C, SPI, UART, e Seriale; tutti i pin di I/O sono configurati di dafault come output, e funzionano con livelli di tensione 3,3V; non tollerano il livello logico alto a 5V, in quanto direttamente connessi con il processore: il progettista dovrà premurarsi di convertire adeguatamente i livelli di eventuali schede a cui si può interefacciare.
2. Connettore per un monitor LCD/OLED con interfaccia DSI (*Display Serial Interface*), del tipo montato sui telefoni cellulari.
3. Connettore per monitor o televisore con uscita video composito.
4. Connettore JTAG[9] per il test della scheda.
5. Connettore CSI (*Camera Serial Interface*) che rende compatibili con Raspberry PI la maggior parte delle videocamere con sensore CMOS utilizzate negli Smartphone.
6. Jack audio 2,5 mm in uscita; non è disponibile un ingresso audio, per realizzarlo bisgona utilizzare una periferica di acquisizione sonora interfacciata tramite USB.
7. Due connettori USB, espandibili con HUB esterno con alimentazione autonoma.
8. Connettore Ethernet RJ45 per il collegamento in rete.
9. Connettore HDMI per utilizzare un monitor video ad alta definizione come dispositivo di output.
10. Connettore di alimentazione in formato micro USB.

[9] JTAG è l'acronimo di Joint Test Action Group, in consorzio di 200 imprese produttrici di circuiti integrati costituito allo scopo di definire un protocollo standard per il test funzionale dei circuiti stampati; attivo tra il 1985 e il 1990 ha dato vita allo standard IEEE 1149.1

3.1.3 Kinect

Quando Microsoft rilasciò Kinect, la reazione di interaction designers e hackers di tutto il mondo di fronte alle enormi possibilità aperte da questa periferica fu istantanea e di proporzioni smisurate.

Matt Webb, CEO di un noto studio di design londinese, leggendo nello sviluppo della tecnologia lo specchio della società (e individuando nella ricerca militare il motore principale di sviluppo tecnologico), considerò Kinect il prodotto rappresentativo del giorno d'oggi, commentando "*WW2 and ballistics gave us digital computers. Cold War decentralisation gave us the Internet. Terrorism and mass surveillance: Kinect.*"[10]. La vera rivoluzione portata da questo dispositivo è stata di rendere disponibile a tutti uno strumento a basso costo e di semplice utilizzo che fornisce ai sistemi informatici la possibilità di percepire la realtà tridimensionale e di interpretare i gesti dell'utente: nell'ottica del physical computing è un'abbassamento totale della barriera tattile con l'utilizzatore, un enorme passo avanti nella naturalezza della comunicazione tra uomo e macchina.

Figura 7 - Kinect senza rivestimento plastico esterno

[10] "La seconda guerra mondiale e la tecnica missilistica ci hanno dato i computer digitali. La decentralizzazione portata dalla guerra fredda ci ha dato internet. Il terrorismo e la sorveglianza delle masse: Kinect."

Kinect è una *depth camera* (anche definita *RGBD camera*), con interfaccia USB, in grado di restituire, oltre all'immagine prodotta dalla luce ambientale riflessa, anche una mappa di distanze per ogni pixel acquisito, ottenuta come riflessione di un pattern di luce infrarossa. Notiamo al suo interno *[Figura 7]* due sensori di visione separati, uno per la luce nello spettro del visibile e uno nel campo degli infrarossi; sul lato è visibile l'emettitore di luce infrarossa, che proietta nell'ambiente un pattern di punti noto all'unità di elaborazione del dispositivo: lo scostamento dei punti di tale pattern nell'immagine riflessa fornisce l'informazione necessaria a ricostruire la distanza tra il sensore e l'oggetto che riflette la luce emessa. La risoluzione delle due videocamere è limitata a 640x480 pixels. Inoltre Kinect è equipaggiato con un array di quattro microfoni (per consentire l'acquisizione sonora sensibile alla posizione della sorgente), led di stato di tre differenti colori, un accelerometro in grado di rilevare l'inclinazione del dispositivo e l'eventualità di movimenti durante l'utilizzo (per consentire di aggiustare la calibrazione di conseguenza) e un servo motore che consente all'intero sistema, poggiato su un suo piedistallo antiscivolo, di ruotare in direzione verticale con un'escursione massima di 30°.

Figura 8 - ASUS Xtion Pro e Microsoft Kinect

3.1.3.1 Origini del dispositivo

Kinect è un prodotto di Microsoft, commercializzato come periferica di controllo per la consolle videoludica Xbox 360; tuttavia si tratta del risultato di molti anni di ricerche condotte sia dalla Microsoft Research Division che da altre società, ma anche dall'intera comunità di progettisti e appassionati di computer vision.

L'hardware costitutivo di kinect venne sviluppato da PrimeSense, una società israeliana che aveva già precedentemente prodotto altre depth cameras basate sullo stesso sistema di proiezione a luce infrarossa; PrimeSense lavorò su questo nuovo hardware con lo scopo di renderlo adatto agli algoritmi e al software su cui Microsoft stava lavorando nelle sue attività di ricerca; ha dato licenza a Microsoft per produrre Kinect, ma detiene il brevetto e la paternità della tecnologia. Infatti ha già collaborato con ASUS nella realizzazione di un prodotto molto simile a Kinect, chiamato Wavi Xtion, e ha preso parte anche ad altri progetti.

Fino a novembre 2010, la combinazione di hardware PrimeSense e software Microsoft era conosciuta con il nome in codice di "Project Natal"; il 4 novembre il dispositivo venne lanciato sul mercato con il nome commerciale di Kinect, e rivelò subito la sua potenzialità divenendo la periferica più velocemente venduta della storia del computer, con più di 10 milioni di unità vendute nel solo primo mese di commercializzazione.

L'interesse del mondo creativo scaturì lo stesso giorno, grazie ad un concorso indetto da Adafruit, una società di New York che si occupa di distribuire kit e materiale elettronico per progetti hardware open source (una parte considerevoli dei quali riguardanti Arduino): Limor Fried, fondatore di Adafruit, mise in palio 2000$ alla prima persona che avesse prodotto un driver open source per accedere ai dati acquisiti da Kinect. Una iniziale reazione negativa di Microsoft a questo concorso ebbe l'unico risultato di far alzare il premio a 3000$, e in meno di una settimana Hector Martin si aggiudicò l'ambita somma, realizzando la prima versione del driver e dando vita al progetto *Open Kinect*, una community di sviluppatori di software open source che ha contribuito a perfeziona reil

driver e a fornire gratuitamente risorse e supporto a chi volessse utilizzarlo nei propri progetti. La disponibilità di un driver open source portò rapidamente al sorgere di numerose librerie per i più diffusi ambienti di sviluppo software. Un importante passo nella diffusione di Kinect tra i creative coders è sicuramente il lavoro di Dan Shiffman, docente dell'università di New York nel programma di Telecomunicazioni Interattive, che elaborò una libreria per Processing.

In risposta a tutto questo interesse PrimeSense distribuì il proprio sistema di riconoscimento degli utenti e di tracciamento delle articolazioni degli stessi, chiamato OpenNI (Natural Interaction), fornendo un prezioso strumento per creare interazioni gestuali; una grande peculiarità di OpenNI è la portabilità del codice su dispositivi diversi da Kinect. Microsoft successivamente rilasciò la propria SDK (con le stesse funzionalità disponibili) per sviluppatori in ambiente Windows.

Le diverse origini degli strumenti che permettono l'utilizzo di Kinect al di fuori del suo scopo di vendita originale sono causa della diversità delle licenze che li accompagnano. I driver sviluppati dal progetto Open Kinect sono completamente open source, e rilasciati sotto la duplice licenza Apache 2.0 / GPL 2.0, consentendo l'utilizzo del codice distribuito da Open Kinect in qualsiasi tipo di progetto commerciale o open source senza alcun tipo di restrizione, gratuitamente e senza riconoscimento di proprietà intellettuale agli autori. OpenNI invece è distribuito sotto licenza LGPL, con caratterisctiche molto simili alla licenza di Open Kinect, ma una parte del codice rilasciato, riguardante il tracciamento delle articolazioni degli utenti rilevati, non ricade sotto questa licenza, in quanto implementata in un modulo esterno chiamato NITE: pur essendo un prodotto commerciale di PrimeSense a tutti gli effetti, è disponibile gratuitamente una licenza di utilizzo, ma non è reso pubblico il codice sorgente.

3.2 Strumenti software

"Concetti ed emozioni che altri media non possono esprimere, possono essere espressi tramite il software. [...] La storia ci insegna che tecniche come la pittura ad olio, la macchina fotografica e le pellicole cinematografiche hanno creato nuove forme di arte, e sebbene affermando che le nuove tecnologie non fanno necessariamente progredire l'arte, siamo convinti che redano disponibili forme differenti di comunicazione ed espressione."[11] La scrittura di codice occupa un posto del tutto particolare tra i linguaggi di espressione artistica perché consente di produrre forme dinamiche, interpretare e elaborare gesti, definire comportamenti, simulare sistemi fisici, e soprattutto integrare dinamicamente altri media: suoni, immagini, testi.

Osserviamo in ogni forma d'arte che la scelta del materiale è influenzata dalle capacità dell'artista, dall'ispirazione del momento e dal risultato che si vuole ottenere. Questa affermazione è ancora valida anche nel mondo dell'interaction design: nel corso degli anni sono comparsi differenti ambienti di sviluppo e linguaggi di programmazione, ciascuno con una propria "personalità" e caratteristiche potenziali, per venire incontro alle esigenze di artisti mediali che mostravano necessità diverse proprio a partire dalla loro esperienza, dalle conoscenze informatiche in loro possesso e dal risultato che volevano ottenere. Potremmo dire che ogni linguaggio di programmazione consente di realizzare praticamente le stesse applicazioni, consentendo ai programmatori di scegliere quello a loro più congeniale: nel campo della digital art la scelta del software ricopre un ruolo più importante, influenzando molto lo stile, l'aspetto e il comportamento del prodotto finale.

I software per sviluppare applicazioni di interaction design sono tipicamente rivolti ad artisti, la cui necessità è di realizzare le proprie idee senza un'eccessiva preparazione e competenza in campo informatico; ciò non toglie che vengano utilizzati anche linguaggi più complessi e di basso

[11] [2]

livello, né tantomeno che tali software "semplificati" non abbiano potenzialità estese: semplicemente possono essere utilizzati senza un approccio tipicamente ingegneristico da chi ha competenze differenti, pur consentendo ai programmatori esperti di avere a disposizione ambienti di sviluppo completi e validi. Questa possibilità ha dato una spinta decisiva nella trasformazione del computer da dispositivo in grado di eseguire velocemente calcoli a vero e proprio strumento espressivo nelle mani degli artisti.

La visione del computer come mezzo espressivo ha le sua radici già all'inizio degli anni '70, quando si iniziò a discutere del potenziale futuro dell'informatica come nuovo media, introducendo due fondamentali concetti: *hypertext*, cioè testi collegati tra loro (che portò alla nascita del web) e *hypergram*, ovvero rappresentazioni grafiche interattive (concetto che sta alla base delle prime interfacce così come della realtà virtuale).[12] È in questo contesto che nei laboratori Xerox viene teorizzato *Dynabook*, un prototipo ideale degli odierni personal computer che comprendeva un linguaggio di programmazione con cui, a detta degli ideatori, i musicisti potessero scrivere il proprio software di composizione, e i bambini il proprio software per disegnare e creare favole personalizzate: in questa visione delle cose per la prima volta non c'è più distinzione tra utente e programmatore. A quasi 40 anni da questa visione ottimistica ci troviamo in una situazione differente a causa della rivoluzione tecnologica e culturale che ha portato i computer e internet a una diffusione incredibilmente vasta, ma con utenti che nella quasi totalità dei casi fanno uso di software scritti da sviluppatori professionisti, piuttosto che crearsi i propri strumenti personalizzati: la facilità di utilizzo di questi software *ready-to-use* ha oscurato la potenzialità della programmazione dedicata; gli artisti digitali devono esplorare le reali potenzialità dei computer senza farsi costringere dai limiti imposti dai software esistenti. Il mercato del software è infatti dominato dalle grandi case produttrici, che impongono degli standard di lavoro agli artisti, e lasciano nell'ombra i tentativi di

[12] neologismi coniati in Nelson T. (1974) *Computer Lib / Dream Machines*

creare nuovo software, mancando agli artisti le capacità tecniche di sviluppare codice indipendentemente: è qui che si collocano i software *open source* più recenti per *creative coders*, supportati da enormi comunità di utilizzatori che in tal modo non solo contribuiscono a migliorare il prodotto, ma realizzano anche un'auto-formazione continua degli utenti, rendendo i software stessi competitivi e di maggior risalto.

Sono riportati di seguito i principali linguaggi utilizzati in ambito grafico e creativo, con le loro caratteristiche principali che li rendono più o meno adatti al progetto dal realizzare ed alle competenze tecniche del programmatore; successivamente verranno analizzati in dettaglio due casi particolari molto significativi, un ambiente di sviluppo integrato e un framework che ad esso si ispira: Processing e openFrameworks.

ACTIONSCRIPT: è un linguaggio sviluppato per Flash, il celebre software di Adobe (originariamente di Macromedia). Flash nacque come software per la creazione di animazioni per il Web, e ActionScript venne pensato successivamente per aggiungere la possibilità di codificare il comportamento dei MovieClips. ActionScript è basato su *ECMAScript* e pertanto presenta molte analogie con JavaScript.

BASIC: nato nel 1963 come linguaggio per permettere agli studenti di discipline non tecniche di utilizzare i computer, BASIC ebbe la sua grande diffusione su tutti gli home computer negli anni '80. Pensato per i principianti ma utilizzabile anche come linguaggio general-purpose per applicazioni più complesse, venne successivamente usato come base per altri linguaggi più specifici, come PIC BASIC per i microcontrollori.

C, C++: il linguaggio C nacque all'inizio degli anni '70, ma è tutt'oggi molto utilizzato. Molti linguaggi successivi, tra cui PHP e Processing, hanno avuto come modello C; oltre che per lo sviluppo di sistemi operativi e di software per PC, è un linguaggio molto popolare anche nella programmazione di microcontrollori. C++ venne ideato per estendere le potenzialità di C attraverso l'aggiunta di concetti object-oriented (lo stesso

nome lo definisce come un'evoluzione di C: '++' è infatti l'istruzione C per aggiungere 1 ad una variabile numerica). A causa dell'enorme diffusione e utilizzo di C, C++ divenne subito il più popolare linguaggio orientato agli oggetti. Il controllo di basso di livello che offrono C e C++ permette la realizzazione di applicazioni veloci ed efficienti. Non hanno però metodi diretti di disegno, e devono appoggiarsi a librerie grafiche come OpenGL (procedimento reso più semplice da tool come openFrameworks, ad esempio)

CHUCK: è un linguaggio di programmazione audio per sintesi sonora in *real-time*, composizione ed esecuzione musicale. Consente di aggiungere e modificare il codice durante l'esecuzione del programma stesso, ed offre un preciso controllo temporale delle istruzioni, data la sua natura puramente musicale.

DBN (Design By Numbers): anche questo linguaggio nasce per insegnare i concetti fondamentali della programmazione ad artisti e designer senza esperienza; fu ideato da John Maeda, nello stesso laboratorio che diede origine a Processing. DBN offre un ambiente e un linguaggio estremamente minimalista, ed è pertanto semplice da apprendere ma fortemente limitato nel creare applicazioni di grafica avanzate.

DRAWBOT: è un linguaggio pensato esclusivamente per l'insegnamento; combina Python con una libreria grafica 2D in un semplice ambiente di sviluppo. Sviluppato solo per Macintosh consente di creare grafica ed esportarla in differenti formati.

JAVA: venne creato da Sun Microsystems negli anni '90 come alternativa a C++. Il linguaggio è rivolto a creare programmi *cross-platform* con supporto integrato per la comunicazione in rete; la popolarità di Java ebbe enorme crescita assieme al Web grazie alle Java Applets, programmi che possono essere eseguiti all'interno di un browser. Diversamente da C e C++, Java consente di scrivere programmi più

velocemente, ma più lenti nell'esecuzione, a causa della compilazione *run time* che li rende però portatili su ogni piattaforma.

JAVASCRIPT: nasce come linguaggio di programmazione per rendere dinamiche le pagine Web. Sebbene il nome sia molto simile, JavaScript non è direttamente collegato a Java; inizialmente venne sviluppato da Netscape con il nome di LiveScript, mutando nome in JavaScript quando Netscape incluse il supporto a Java nel proprio browser.

LINGO: è il linguaggio scritto per Macromedia Director, uno dei principali strumenti per realizzare progetti su CD-ROM, che perse di importanza nell'era del Web a causa del successo di Flash, ma è ancora largamente utilizzato grazie alle ottime librerie di codice disponibili per estendere le sua funzionalità. La particolarità di Lingo è un ambiente di sviluppo che fa uso di termini come "*stage*" e "*cast*" per descrivere i vari elementi di un progetto, ed è caratterizzato da una sintassi che lo rende simile alla lingua inglese. Director è stato modificato nel corso degli anni per supportare la programmazione ad oggetti e la creazione di grafica 3D.

MAX/MSP/JITTER: Max (che prende il suo nome da Max Mathews, pioniere della computer music) era originariamente un linguaggio di programmazione visuale per controllare dati MIDI. L'interfaccia di Max ricorda quella dei sintetizzatori analogici, e permette al programmatore di collegare visivamente tra loro diversi moduli per controllare l'elaborazione dei dati. Gli oggetti MSP vennero aggiunti in seguito per permettere al software di generare audio in tempo reale; Jitter è un'estensione successiva per controllare video e grafica tridimensionale. Max/MSP/Jitter è comunemente usato per realizzare performance live audio-visuali.

MEL (Maya Embedded Language): è il linguaggio del software di modellazione 3D Maya; è utile per automatizzare operazioni ripetitive e per raggruppare in macro riutilizzabili sequenze di istruzioni. La sintassi è

simile a C, pertanto non supporta la programmazione orientata agli oggetti.

MOBILE PROCESSING: è una variazione di Processing specifica per sviluppare software per smartphone. La libreria grafica è ottimizzata per i processori di tali dispositivi, e vi sono alcune funzioni aggiuntive per accedere direttamente all'interfaccia disponibile (sia da tastiera che attraverso schermo touch) e all'hardware in dotazione (Bluetooth, videocamera, microfono, etc...).

PERL: è un linguaggio flessibile, la cui sintassi è una commistione di molti linguaggi differenti tra cui C, awk, sed, sh e altri; è molto diffuso per la programmazione rivolta al Web e alle applicazioni di rete, ha avuto il suo picco di popolarità negli anni '90 e ha ispirato altri linguaggi come PHP. I suoi punti di forza sono l'analisi e l'elaborazione testuale e la facilità con cui si possono processare grandi quantità di dati.

PHP: è un linguaggio semplice ma con grandi potenzialità, ideato per creare contenuti dinamici per il Web. La sintassi è simile al C, ed è facilmente integrabile con HTML. Il suo utilizzo più comune è la gestione di database.

PD (Pure Data): è un linguaggio visuale per creare computer music e grafica live; è l'alternativa open-source a Max, creata dallo stesso autore, e si differenzia per la possibilità intrinseca di sintesi audio in real time. Come per Max i programmi sono scritti collegando patch visuali.

PYTHON: è considerato un linguaggio di programmazione ad oggetti molto adatto per scopi didattici grazie alla sua semplice sintassi e struttura molto leggibile. Python è tipicamente utilizzato per applicazioni non grafiche, non avendo una libreria nativa a supporto, ma è comunque possibile generare grafica con toolkit appositi. Il nome deriva dalla passione del suo creatore, Guido van Rossum, per il gruppo di comici inglese Monty Python.

QUARTZ COMPOSER: è un linguaggio visuale incluso in Mac OS X per rendering ed elaborazione di grafica usando OpenGL. Come negli altri linguaggi visivi, l'elemento fondamentale è la *patch* (l'equivalente di una funzione), e l'utente può connettere diverse patch semplicemente tracciando linee tra i loro ingressi e uscite. Le composition (così sono chiamati i programmi scritti in Quartz Composer) possono essere eseguite all'interno di altre applicazioni Mac OS X.

RUBY: orientato agli oggetti, si caratterizza per la facilità nel processare file di testo e nell'eseguire operazioni di gestione del sistema. La sintassi consente grande flessibilità nello strutturare il codice, rendendolo i programmi difficile lettura ad occhi diversi da quelli dell'autore. La sua popolarità è aumentata grazie a Ruby on Rails, un framework per realizzare applicazioni Web.

SQL (Structured Query Language): è il linguaggio più comunemente usato per creare e gestire database. Le sue origini risalgono al 1969, ma venne standardizzato soltanto nel 1986. Pur non consentendo di produrre applicazioni, fornisce un ottimo strumento per interrogare un database attraverso *query* e *procedure*.

SUPERCOLLIDER: è un ambiente di sviluppo (disponibile solo per Mac OS) per la sintesi di audio real-time. Il linguaggio usato contiene elementi di Smalltalk e di C; SuperCollider comprende, oltre al linguaggio di programmazione, un sistema di classi *object-oriented*, un costruttore di interfaccia per creare un pannello di controllo per l'utente, controllo MIDI e una vasta libreria di funzioni per processare e sintetizzare segnali audio.

WIRING: è un linguaggio specifico per programmare microcontrollori; viene molto usato per insegnare i concetti basilari della prototipazione elettronica. Il linguaggio è basato su Processing, con un taglio che lo rende più adatto all'elettronica; l'ambiente di sviluppo stesso è una versione modificata dell'IDE di Processing. Quando si compila un programma in

Wiring, viene tradotto in codice C e poi ri-compilato come programma in C. Wiring è il linguaggio usato anche per Arduino.

vvvv: è un linguaggio visuale, simile a Max e Pure Data, ma con un'interfaccia più elaborata e minor accento sulla produzione di audio; comprende strumenti per creare grafica 3D e rende possibili multi-proiezioni in modo semplice. È disponibile solo per Windows, ed è distribuito gratuitamente per uso non commerciale.

3.2.1 Processing

Processing mette in relazione i paradigmi della programmazione con i principi della visualizzazione grafica, del movimento e dell'interazione; integra in sé un linguaggio di programmazione, un ambiente di sviluppo e una metodologia di insegnamento, rendendoli accessibili in forma compatta e semplice: questi i punti di forza che lo hanno reso a tutti gli effetti la IDE per eccellenza dei creative coders nell'ultimo decennio.

Processing fu il risultato delle ricerche del gruppo di lavoro di "Aesthetics + Computation" del MIT Media Lab attorno al 2001. Precedentemente, a partire dal 1999, il docente responsabile del gruppo, John Maeda, assieme ai suoi studenti (tra cui anche Casey Reas e Ben Fry, futuri promotori e sviluppatori di Processing) aveva ideato DBN, un linguaggio di programmazione molto semplice, con il suo relativo ambiente di sviluppo. DBN era utilizzabile in remoto via Web o come applicazione stand-alone, e fu concepito con lo scopo di semplificare enormemente la scrittura di codice per le arti visive; possiamo vedere Processing come il diretto discendente, molto più evoluto, di questa prima realizzazione molto semplice: infatti utilizzando DBN la finestra di output era limitata a pochi centimetri quadrati di grafica, per di più in bianco e nero, ma comprendeva molte delle caratteristiche che ci aspettiamo da un linguaggio di programmazione rendendole semplici da implementare. Sorto dall'esperienza maturata con DBN, oltre a migliorare le prestazioni e

le possibilità grafiche, si distingue dal suo predecessore (e trova ancora oggi il suo punto di forza) principalmente nell'approccio multimodale, che consente di programmare ad alto livello (di astrazione, quindi in maniera più semplice ma con minor controllo diretto), a basso livello (direttamente in Java) o con un approccio ibrido tra i due.

3.2.1.1 Struttura del linguaggio

Processing non può essere considerato a tutti gli effetti un linguaggio di programmazione, ma piuttosto un ambiente di sviluppo, dal momento che l'IDE di processing è in realtà un programma Java in esecuzione. Processing consente essenzialmente di scrivere codice molto più semplice rispetto al puro Java; al momento della compilazione l'output è convertito in un file classe Java (identico ad un comune programma scritto direttamente in Java) e tale classe viene poi interpretata all'interno della *Java Virtual Machine* durante l'esecuzione.

Processing permette di strutturare il codice sia in uno stile procedurale (*function-based*) che *object-oriented*. Nel primo approccio, più intuitivo per i neofiti, l'algoritmo di programmazione viene implementato attraverso funzioni (procedure, subroutines), ovvero blocchi di codice riutilizzabili che il programmatore può richiamare quando necessario. Diversamente, uno stile object-oriented è più astratto, e porta a dividere il codice in blocchi chiamati oggetti, svincolandosi dalla sequenzialità procedurale più vicina al comune modo di pensare e suddividere gli obiettivi da implementare. La possibilità di scegliere la modalità più congeniale, e addirittura di utilizzarle contemporaneamente, rende Processing uno strumento adatto sia ai principianti che ai professionisti, allargandone gli orizzonti e mettendo in comunicazione in maniera diretta i programmatori e gli artisti.

Nel modo più semplice di comporre il codice, Processing rende libero l'utente dall'apprendere anche solo le basi dell'approccio procedurale: è infatti possibile scrivere semplicemente una sequenza di istruzioni e metterla in esecuzione; questo è il cosiddetto "*basic mode*", di grande utilità per sperimentare brevi porzioni di codice senza dover scrivere

troppo, ma anche nella fase di apprendimento per concentrarsi sul risultato ottenuto da semplici funzioni. La modalità più utilizzata è comunque il "*Continuous mode*", che si basa su due primitive fondamentali di Processing che consentono di aggiungere il codice personalizzato ed eseguirlo: la prima funzione, *setup()*, conterrà le dichiarazioni e le impostazioni che verranno eseguite una sola volta all'avvio del programma, mentre la seconda, *draw()*, farà sì che il codice in essa contenuto venga eseguito ciclicamente, introducendo la possibilità di realizzare animazioni e più in generale comportamenti dinamici del programma. Un terzo modo, "*Java mode*" consente di utilizzare la IDE di Processing per sviluppare codice in puro Java, mettendone a disposizione l'intera API; è chiaramente riservato ai programmatori esperti.

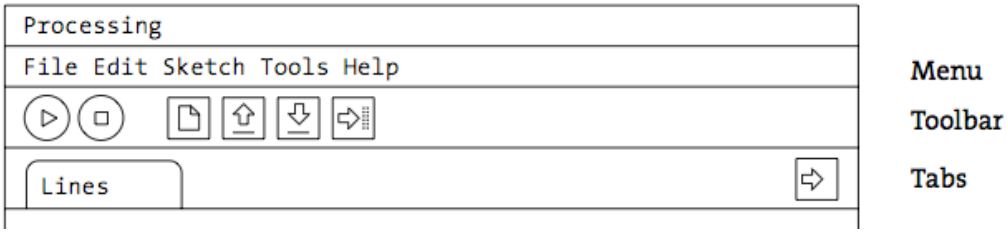


Figura 9 - Menu della prima IDE di Processing

3.2.2 openFrameworks

Come dice il nome stesso, openFrameworks non è un linguaggio di programmazione e non comprende un ambiente di sviluppo, ma è un *framework*, cioè una raccolta di codice disponibile al programmatore per realizzare obiettivi complessi, nello specifico rivolto al campo dell'interaction design e della media art. Dal momento che è scritto in C++ richiede più competenze specifiche rispetto a Processing, ma consente un livello molto maggiore di flessibilità e di prestazioni del prodotto finale.

openFrameworks è stato sviluppato da Zach Lieberman, Theo Watson, Arturo Castro e Chris O'Shea, con il supporto di collaboratori della Parson School of Design, del MediaLabMadrid e del centro di ricerca arti visive

Hangar di Barcellona, e ha visto la sua espansione grazie ad una vasta comunità di creative coders che hanno alimentato e supportato il progetto. Il nucleo del framework ha origine alla Parson School of Design, di cui Lieberman era studente: nella necessità di avere maggior flessibilità e potenzialità per i suoi progetti interattivi si avventura nello studio del C++, e nota che, tra le numerose librerie e frameworks disponibili per ogni altro tipo di applicazione, praticamente nessun supporto era presente per facilitare l'approccio alla programmazione in C++ agli artisti digitali; nasce allora l'idea di realizzare openFrameworks per fornire una modalità simile a quella di Processing nella programmazione di grafica in C++.

3.2.2.1 Struttura del framework

openFrameworks si presenta come una cartella da scaricare dal sito ufficiale del progetto (o da un repository on line nel caso si volesse utilizzare una versione differente dalla ultima release stabile) da copiare semplicemente sul proprio hard disk. Fondamentalmente programmare con openFrameworks significa scrivere del codice in C++ in una IDE a piacimento (sono consigliate Code::Blocks per Linux e Windows, e Xcode per Mac) e di utilizzare una classe (*ofBaseApp.h*) che consente di sfruttare tutte le potenzialità contenute nel framework, unitamente a numerose altre classi e librerie in continuo aggiornamento ed evoluzione, scaricabili da una sezione del sito ufficiale. Quello che troviamo nella directory di openFrameworks è organizzato nelle seguenti sotto-cartelle:

- *addons*: contiene tutte le funzionalità aggiunte al nucleo centrale del framework, create e rese disponibili dalla comunità di utenti durante il corso degli anni;
- *apps*: cartella che racchiude i programmi di esempio e dedicata a contenere le applicazioni scritte dall'utente: in quanto basate su inclusioni di librerie, le applicazioni di openFrameworks devono trovarsi in questo percorso ben definito per evitare problemi nei collegamenti alle risorse esterne.
- *libs*: contiene tutte le librerie utili al funzionamento e alla corretta compilazione del codice; come per gli addons vengono fornite le

librerie fondamentali per le funzioni basilari, ma è possibile aggiungerne a piacere secondo necessità per operazioni più complesse.

- *openFrameworks*: qui è contenuto il nucleo di openFrameworks, ovvero tutte le risorse direttamente utilizzabili dallo sviluppatore; è ulteriormente suddivisa in sei sotto-cartelle:
    - *app*: contiene l'header di *ofBaseApp*, cioè il prototipo di ogni applicazione sviluppata con openFrameworks, e che rende disponibili funzioni di alto livello per controllare mouse e tastiera e semplificare le operazioni di creazione e visualizzazione grafica;
    - *communication*: rende disponibile un protocollo di comunicazione seriale, e il metodo *ofEvents*, che gestisce l'invio e la ricezione di informazioni relative agli eventi in openFrameworks;
    - *graphics*: contiene le cinque classi fondamentali per creare grafica digitale: *ofBitmapFont* e *ofTrueTypeFont* per lavorare con i caratteri alfanumerici, *ofGraphics* per il disegno bidimensionale, *ofImage* per le immagini bitmap e *ofTexture* per operare con textures di OpenGL;
    - *sound*: contiene due classi, *ofSoundPlayer* per creare e riprodurre file audio, e *ofSoundStream*, che fornisce un controllo di basso livello sulle funzionalità della scheda audio del sistema;
    - *video*: classi specifiche per la riproduzione e la modifica di file video, e per l'acquisizione da fonti esterne di contenuti mediali;
    - *utils*: contiene l'implementazione di alcune funzioni matematiche, codice per misurare i tempi, definizione di tipi di dato utilizzati nel resto del codice, e così via.

La struttura di un programma scritto in openFrameworks è riportata nella figura sottostante *[Figura 10]*; si può notare che un'applicazione scritta in

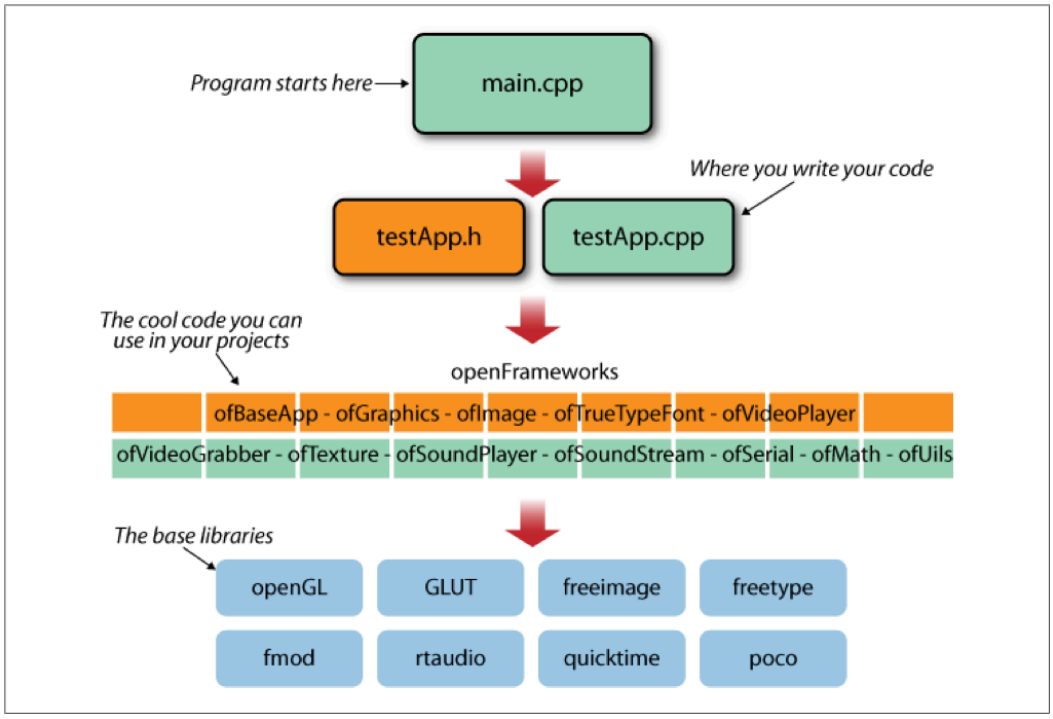


Figura 10 – Struttura di una applicazione openFrameworks

openFrameworks poggia le sue basi su una serie di librerie appositamente sviluppate per la produzione di grafica e contenuti interattivi, a loro volta collegate e rese funzionanti dalle librerie di base che comunicano a più basso livello con il sistema in cui si sta sviluppando. Questo significa che la portabilità di un'applicazione di openFrameworks su diversi sistemi operativi non è così immediata come in Processing, ma nemmeno troppo complessa, in quanto si limita al corretto collegamento delle librerie di base con il sistema stesso.

4. Case studies

"Let's make better mistakes tomorrow"

Anonimo

Sono riportate in questo capitolo le descrizioni di alcune installazioni interattive di diverso genere realizzate durante la stesura di questa tesi, sia autonomamente che per conto di ZetaLab, studio di comunicazione di Milano; in particolare si ringrazia Lucio Lazzara, socio fondatore di ZetaLab, per il tempo e le competenze dedicati a questi progetti, per la fiducia riposta e per l'opportunità concessa. Grazie anche al professor Giancarlo Storti Gajani per il supporto tecnico e i preziosi consigli nelle ultime concitate ore prima della consegna dei lavori stessi ai clienti.

La documentazione dei lavori svolti in collaborazione con ZetaLab è consultabile on-line:

Zetalab Brand & communication design *http://www.zetalab.com/works/interactive/*

MAGIC BOX

Interactive installation. A "magic" book is the remote controller to select and display movies and poetic texts in artistic ways.

Customer: Moleskine
Design: Lucio Lazzara
Engineering: Matteo Riva

Expositions:
Milano (IT) - "Fuorisalone 2011"
New York (USA)

4.1 Magic Box

All'inizio del 2011 ho sviluppato l'hardware e il software per un'installazione commissionata da Moleskine, che abbiamo chiamato "*Magic Box*", esposta per una settimana a Milano e successivamente a New York; per la realizzazione hardware è stato utilizzato Arduino per acquisire le informazioni dai sensori ed inviarle tramite Bluetooth al software di visualizzazione, sviluppato con Processing.

4.1.1 Concept

Un videoproiettore permette di visualizzare dei video o alcune frasi tratte da una raccolta di citazioni; attraverso un controller remoto l'utente deve poter interagire scegliendo se riprodurre i video o leggere le frasi; nella modalità video deve essere possibile controllare la velocità di riproduzione e il passaggio ad un video diverso, scelto casualmente da un database preesistente; nella modalità testuale l'utente deve avere la sensazione di far muovere le lettere sullo schermo come se fossero fisicamente presenti all'interno del controllore e ha la possibilità di rimescolarle in frasi differenti.

4.1.2 Interaction Analysis

Il controller remoto ha preso la forma di un'agenda, per richiamare chiaramente la linea di prodotti del cliente. L'interazione principale è stata individuata nella visualizzazione testuale, più vicina anche intuitivamente alla contestualizzazione del controller; per rendere spontanea l'interazione

dell'utente, all'interno del controller (oltre ovviamente all'elettronica necessaria) sono stati inseriti dei piccoli oggetti liberi di muoversi (nello specifico dei piccoli grani di pasta) per restituire la sensazione delle lettere in movimento nell'agenda. L'utente ha possibilità di far muovere le lettere che compongono la frase visualizzata inclinando il controller lentamente, mentre scuotendolo più velocemente vede riposizionarsi le lettere della frase attuale che sono presenti anche nella successiva, mentre escono dallo schermo le lettere superflue ed entrano quelle mancanti per comporre una nuova frase. Le animazioni sono fluide e naturali, e per il movimento delle lettere si è scelta una simulazione fisica, assegnando ad ogni carattere un peso proporzionale ai pixel occupati, e spostando il centro di gravità del sistema simulato in relazione allo spostamento fisico del controller; le lettere possono rimbalzare (per fermarsi con uno smorzamento adeguato alla massa virtuale a loro assegnata) sui bordi della proiezione, o accumularsi se l'utente mantiene il controller inclinato verso un bordo o un angolo; se il controller rimane fermo per un periodo sufficiente di tempo la frase si ricompone, per evitare che rimanga una visualizzazione illeggibile tra un utilizzo e il successivo nel caso in cui un utente abbandoni il controller in posizione non orizzontale. Per passare da questa modalità testuale alla riproduzione dei video l'utente deve capovolgere il controller: è un gesto abbastanza spontaneo che nasce dalla curiosità di esplorare l'interazione, e che porta quindi ad usufruire in maniera naturale di tutte e due le esperienze senza necessità di istruzioni o indicazioni per l'utente. In questa modalità è possibile cambiare il video in riproduzione con un veloce scuotimento del

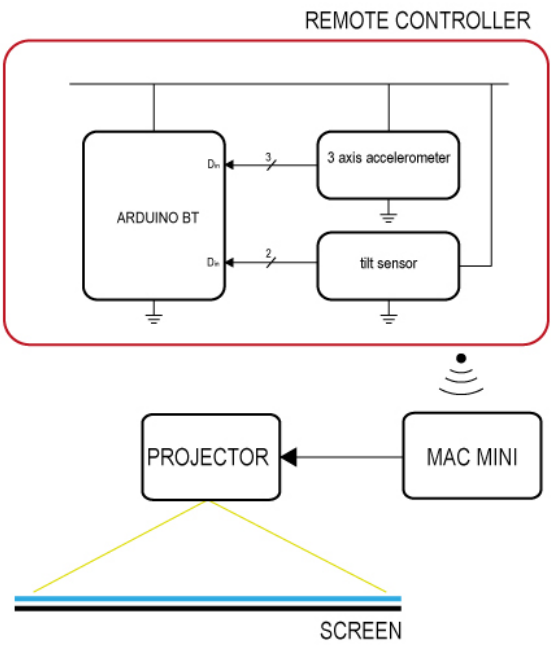


Figura 11 - "Magic Box" setup

controller (interazione analoga al cambio di frase, così che l'utente non debba imparare un nuovo "linguaggio" per interagire con contenuti differenti) e la velocità di riproduzione è controllabile tramite l'inclinazione del controller stesso, consentendo sia avanzamento che riavvolgimento del video a velocità modulabile.

Tutte queste considerazioni fatte in fase di progetto hanno avuto riscontro positivo osservando le reazioni degli utilizzatori durante i giorni di permanenza dell'installazione a Milano: la quasi totalità degli utilizzatori ha usufruito dell'esperienza interattiva senza bisogno di indicazioni, ed ha esplorato tutte le funzioni del controller interagendovi con naturalezza.

4.1.3 Hardware

Per rilevare l'inclinazione del controller la scelta è ricaduta su un accelerometro a 3 assi, a cui è stato aggiunto un *tilt sensor* per capire quando la scatola era capovolta. Le informazioni fornite dall'accelerometro si sono rivelate sperimentalmente molto sensibili anche a spostamenti impercettibili: è stato pertanto necessario scrivere del codice, in esecuzione su Arduino, che mediasse il flusso di dati in ingresso, operando un filtraggio passabasso per rilevare soltanto spostamenti volontari

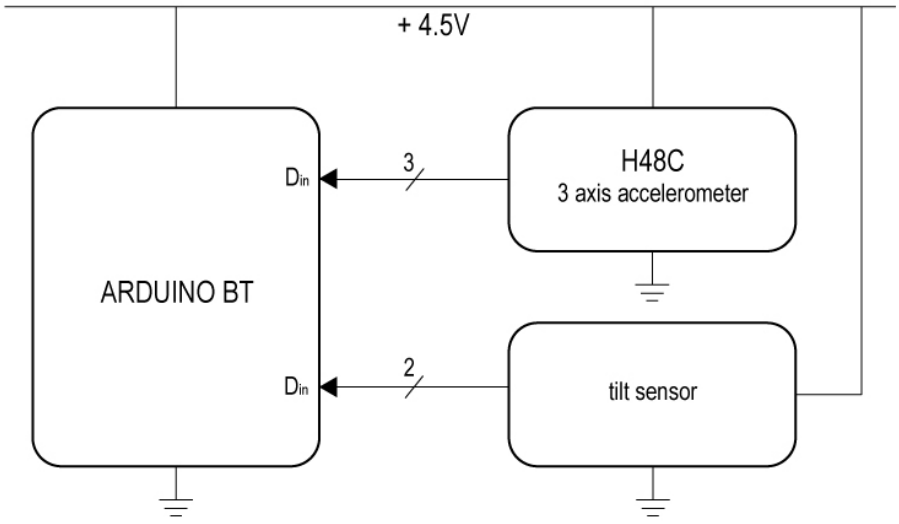


Figura 12 - Schematizzazione dell'hardware

dell'utente. Il tilt sensor ha aggiunto l'informazione necessaria ad interpretare correttamente i dati dall'accelerometro, con la corretta inversione dell'asse x nel calcolo dell'inclinazione svolto dal microcontrollore.

I dati provenienti da questi due sensori principali sono raccolti e interpretati di una particolare scheda Arduino in grado di comunicare con un computer tramite BlueTooth. Sul computer è in esecuzione il software che elabora i dati ricevuti e genera la grafica da proiettare. Le dimensioni circoscritte dell'ambiente (e quindi la distanza massima del controller dal computer) hanno reso possibile l'utilizzo del protocollo BlueTooth senza rischi di errori di comunicazione, anche grazie al flusso dei dati trasmessi, ridotto dalla pre-elaborazione effettuata da Arduino prima della trasmissione.

L'alimentazione del controller è a batteria, con 4 pile stilo che forniscono alla scheda Arduino i 5V necessari; l'alimentazione e i riferimenti di tensione per l'accelerometro sono poi prelevati direttamente dalla scheda Arduino.

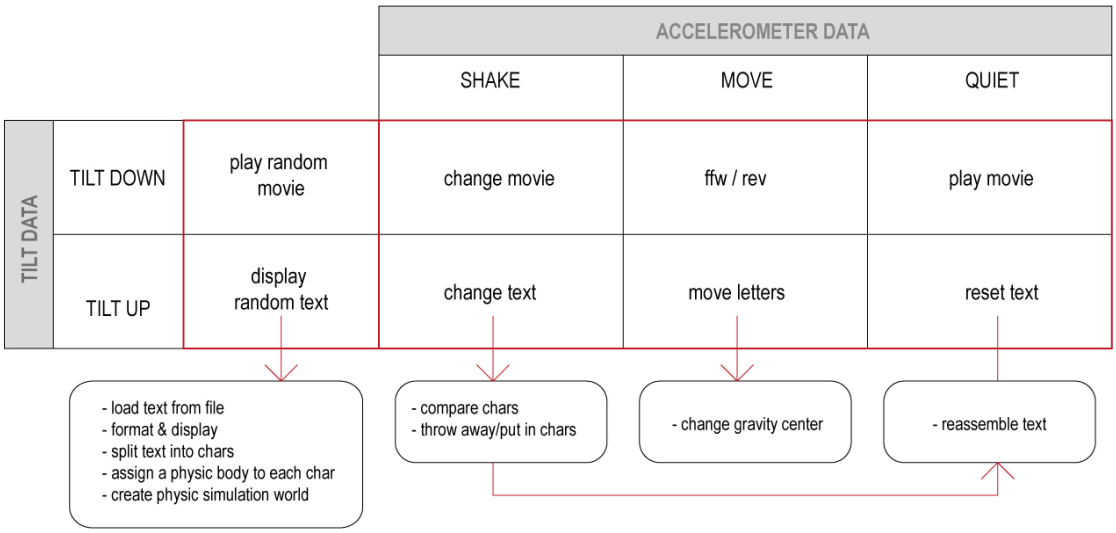
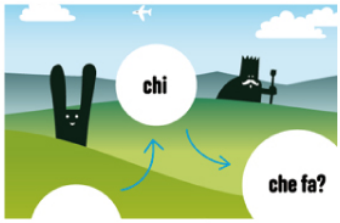


Figura 13 - Algoritmo decisionale del software

4.1.4 Software

Il software è stato sviluppato in Processing, e svolge la duplice funzione di interpretare i dati provenienti dei sensori (attraverso comunicazione seriale) e di generare la grafica che andrà proiettata. La facilità di programmazione offerta dall'IDE di Processing ha consentito di sviluppare il software modificando il codice in maniera molto dinamica, sperimentando continuamente la risposta grafica attraverso l'interazione reale con il prototipo del controllore. Per una maggior naturalezza nel comportamento delle lettere sullo schermo si è utilizzata una libreria Java di simulazione fisica, molto utilizzata nello sviluppo di videogame e di simulazioni realistiche, anch'essa open-source e liberamente utilizzabile.



SALTAFAVOLE

Interactive carpet; jumping over six sensible areas
users can create more than one million different fairy tales.

Design: Lucio Lazzara
Texts: Lara Aldeghi & Lucio Lazzara
Engineering: Matteo Riva

Expositions:
Milano (IT) - "Uovo Kids" @ MUST
Istanbul (TR)

4.2 SaltaFavole

La seconda collaborazione con ZetaLab ha portato alla realizzazione di un'installazione interattiva rivolta ai bambini, che saltando su aree definite di un tappeto possono creare più di un milione di brevi favole generate casualmente. Esposto inizialmente al Museo della Scienza e della Tecnica di Milano in occasione del festival *UOVO Kids* nel giugno 2011, il *SaltaFavole* è stato anche a Istanbul il mese successivo.

4.2.1 Concept

I bambini devono poter interagire saltando su un tappeto sul quale sono riportate sei parti di un breve racconto generico: *dove? chi? che fa? cosa? perché? e...?*. Ad ogni domanda è associata una raccolta di circa 20 frasi da cui attingere casualmente, consentendo così la formazione di più di un milione di favole diverse, generate casualmente in maniera interattiva.

4.2.2 Interaction Analysis

Per questa installazione è stato realizzato un grande tappeto illustrato, con sei aree bianche, ciascuna con una delle sei domande. I bambini osservano la proiezione delle sei parti della favola rappresentate come se fossero appese ad un filo ancorato tra una parte e la successiva. Saltando sopra ad un'area possono osservare che la parte di frase corrispondente nella proiezione oscilla, come se avessero scosso il filo a cui sono appese le sue lettere; un altro salto provoca ancora maggiore disordine, e un terzo

salto consecutivo fa disperdere le lettere della frase attuale e induce l'ingresso delle lettere della frase successiva. Nell'intervallo di tempo tra un salto e il successivo le lettere tendono a tornare alla posizione originale, con movimenti naturali generati da una simulazione fisica di corpi elastici vincolati tra loro dinamicamente e fissati agli estremi, per garantire la sensazione di interazione diretta e naturale con la proiezione. Per stimolare la curiosità dei bambini ad esplorare l'installazione, attorno alle frasi prendono vita animazioni grafiche; l'inutilizzo per lungo tempo provoca dei cambi casuali delle frasi proiettate, per istruire l'osservatore all'utilizzo del tappeto, che riporta disegni simili a quelli animati nella proiezioni.

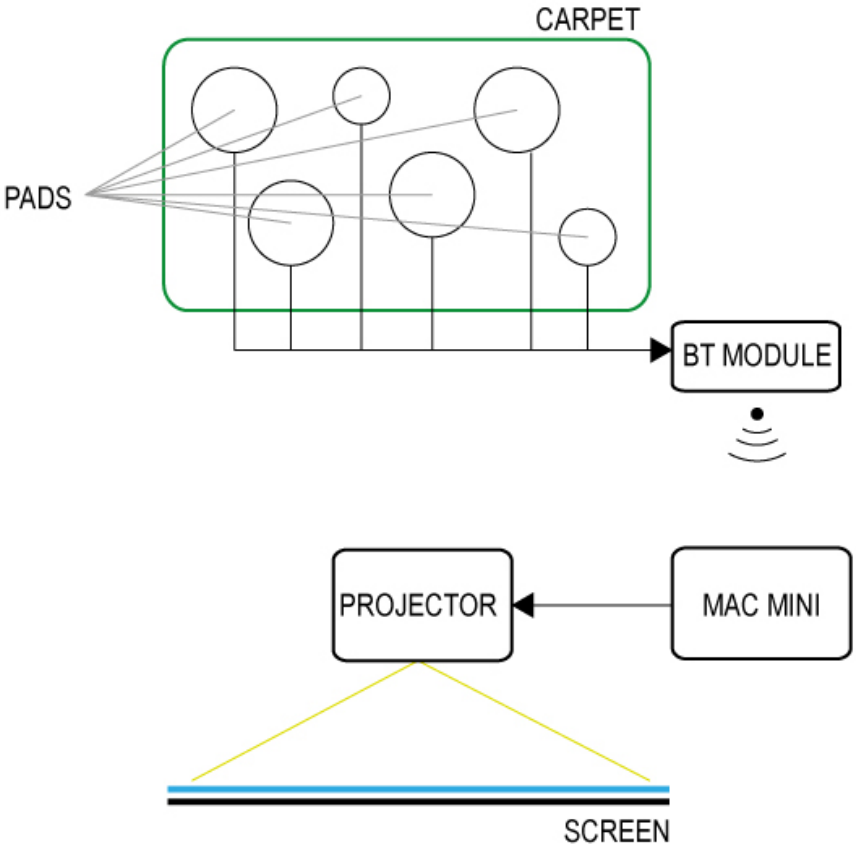


Figura 14 - Setup del "SaltaFavole"

4.2.3 Hardware

Per l'elaborazione dei dati provenienti dai sei pulsanti posti sotto alle altrettante aree sensibili si è scelto anche in questo in caso un modulo Arduino in grado di comunicare via BlueTooth le informazioni ad un calcolatore remoto, connesso al videoproiettore. Per realizzare i contatti dei pulsanti sotto al tappeto è stata realizzata una superficie isolante comprimibile, forata e senza memoria di forma, delle dimensioni del tappeto stesso, interposta tra due porzioni di tessuto conduttivo della forma delle aree sensibili.

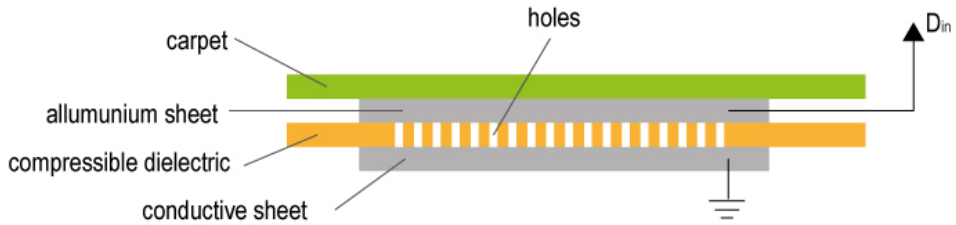


Figura 15 - Struttura dei pulsanti

Il tessuto permette di realizzare contatto elettrico in corrispondenza dei fori nell'isolante quand'esso viene compresso dal peso del bambino che sta saltandoci sopra; un apposito algoritmo di anti-rimbalzo software permette di isolare i singoli salti e gestire correttamente il comando dell'utente.

4.2.4 Software

Il software che genera la grafica ricevendo tramite comunicazione seriale wireless le informazioni sulla pressione delle aree sensibili del tappeto è stato sviluppato in Processing. Un contatore azzerato con tempi predefiniti e misurati rispetto allo studio dell'interazione tipica permette di valutare il comportamento delle lettere da simulare; un motore indipendente si occupa di generare animazioni casuali con grafica creata a priori, per movimentare casualmente la proiezione e renderla più

interessante agli occhi dell'utente. Come nel caso precedente, si è fatto uso di una libreria di simulazione fisica open-source per modellare il comportamento delle singole lettere animate nella proiezione, per garantire la massima naturalezza nella risposta dell'interazione.

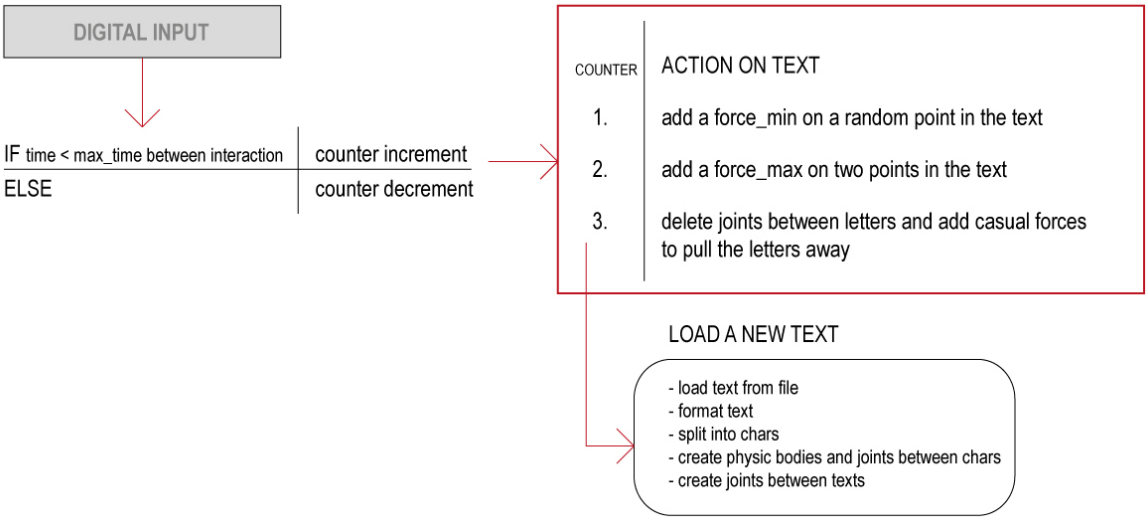


Figura 16 - Algoritmo decisionale implementato in Processing

AUDREY II

A little plant acting on the stage in a musical;
embedded system interacting with actors.

Customer: Musical Box
Design: Matteo Riva
Engineering: Matteo Riva

Expositions:
"Little Shop of Horrors" tour
by Musical Box

4.3 Audrey II

Per la produzione del musical "Little Shop of Horrors" della Compagnia Teatrale MusicalBox nella stagione teatrale 2011/12 ho realizzato un robot interattivo che recitasse la parte di una piccola pianta carnivora assieme al protagonista in una delle prime scene dello spettacolo. Il risultato è un sistema embedded basato su Arduino che controlla i movimenti di un braccio robotico a 4 gradi di libertà.

4.3.1 Concept

La trama di questo musical prevede che una pianta carnivora cresca gradualmente dall'inizio alla fine dello spettacolo ogniqualvolta si nutra di sangue umano. Per le scene iniziali è necessaria una pianta dalle dimensioni contenute, che il protagonista possa tenere in mano, e che reagisca coerentemente con la scena che si va rappresentando: in particolare dovrà appassire ad un preciso momento, risollevarsi successivamente e poi interagire in maniera complessa col protagonista, inseguendo una sua mano o rifuggendola, ed infine ergersi in tutta la sua statura per simulare la crescita.

Figura 17 - La pianta in scena durante una canzone

4.3.2 Interaction Analysis

Trattandosi di una performance live, l'attenzione si è concentrata sul mettere a proprio agio il protagonista interazione della pianticella robotica; se inizialmente si è pensato ad un controllo tramite radiocomando, si è giunti ad una soluzione più semplice tramite un pulsante nascosto, e comportamenti temporizzati con la musica di accompagnamento della scena, così da fornire un riscontro diretto all'attore riguardo all'azionamento dell'automazione, guadagnando naturalezza nell'esecuzione e maggior libertà di recitazione. Ovviamente il sistema è stato progettato per funzionare a batteria, e tutta l'elettronica è stata nascosta all'osservatore per fornire al pubblico l'impressione di una vera (seppur strana e interessante) pianta carnivora, ai fini della buona riuscita dello spettacolo.

4.3.3 Hardware

Ancora una volta a gestire il funzionamento del sistema è una scheda Arduino, utilizzata in questo caso autonomamente, senza un calcolatore con cui comunicare: in ingresso riceve la lettura del pulsante di azionamento che in modo sequenziale provoca i movimenti temporizzati precedentemente in fase di progettazione, e in uscita comanda i cinque servomotori che compongono il braccio robotico da azionare. L'azionamento è realizzato tramite la comunicazione digitale con gli ingressi dell'integrato L293, alle cui uscite sono collegati i motori.

La scelta di un braccio robotico a 5 gradi libertà consente una maggior naturalezza nei movimenti della pianticella, fattore accentuato dal lavoro scenotecnico a copertura dei giunti meccanici.

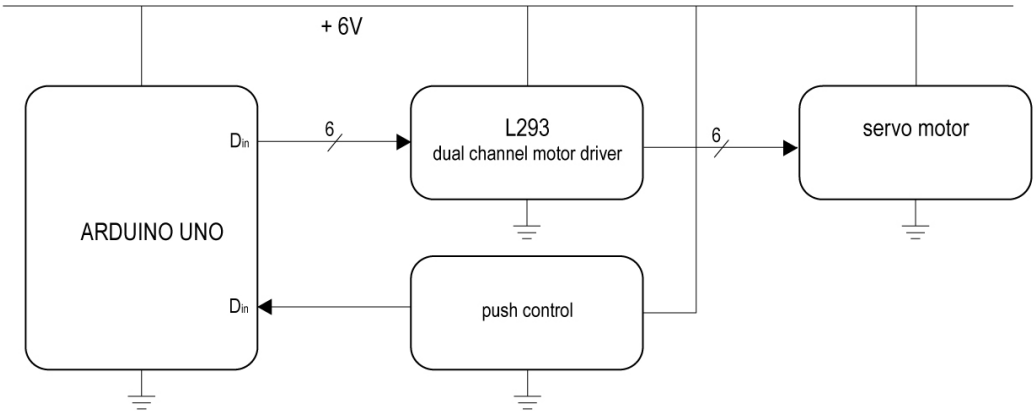


Figura 18 – Schema delle connessioni hardware

4.3.4 Software

Il software è stato sviluppato con l'IDE di Arduino, ed è in esecuzione sulla scheda stessa. La scelta di far eseguire movimenti registrati e testati in precedenza consente di evitare lo studio della cinematica inversa e il controllo di posizione dei giunti, semplificando notevolmente il carico computazionale che grava sulla scheda, il cui unico scopo è di inviare con la corretta temporizzazione i segnali di comando ai driver dei motori.

PER SPECULUM

Can you see your face in the mirror?
Come closer, and see what happens...

Customer: Kernel Festival
Design: Matteo Riva
Engineering: Matteo Riva

Expositions:
Kernel Festival 2012
Villa Tittoni Traversi, Desio (MB)

4.4 perSpeculum

Come risultato di un laboratorio di interactive digital art con Davide Sgalippa (per il progetto *Kernel Lab*, a Monza) e di una residenza artistica presso il centro di ricerca arti visive Hangar di Barcellona sotto il tutoraggio di Alex Posada, ho ideato e realizzato un'installazione interattiva che ho esposto nel giugno 2012 al Kernel Festival di Desio, festival di video mapping e arte digitale.

4.4.1 Concept

L'installazione deve svilupparsi attorno ad una frase tratta dalla prima lettera ai Corinzi di San Paolo: *"Videmus nuns per speculum in enigmate, tunc autem facie ad faciem"*[13]. L'idea è di far riflettere sul limite di auto-conoscenza che le nostre categorie finite impongono all'uomo, in contrasto alla conoscenza assoluta e perfetta dell'infinito, che né la scienza né la filosofia potranno mai dare.

4.4.2 Interaction Analysis

Il visitatore deve essere invitato ad avvicinarsi allo specchio: sono pertanto presenti sopra di esso delle luci che lo rendono ben visibile anche dalla distanza, consentendo a chiunque passi nei dintorni di specchiarsi. Nell'allestire l'esposizione è bene che il dispositivo venga integrato il

[13] *"Pochè ora vediamo come in uno specchio, in maniera confusa; ma allora (nella vita eterna, ndt.) vedremo faccia a faccia"*

meglio possibile nell'ambiente per farlo sembrare un semplice e normale specchio (una soluzione è stata l'inserimento della struttura all'interno di una porta della sala di esposizione, creando continuità con il muro della stanza e riprendendone i motivi e le decorazioni). Chi si lascia incuriosire intraprende un breve percorso osservando la propria immagine, e man mano che si avvicina la vede progressivamente sempre più sfocata e meno definita, mentre invece si aspetta di potersi specchiare meglio. Arrivato in prossimità dello specchio non potrà più distinguere la propria faccia riflessa ma si vedrà di spalle, scorgendo alla fine del simbolico cammino di conoscenza soltanto l'immagine di se stesso ancora in ricerca, che guarda oltre dove noi non possiamo vedere. Il cammino che doveva portare ad una conoscenza di se stessi più perfetta porta invece all'incontro con il proprio limite di creatura umana.

4.4.3 Hardware

L'effetto di sovrapposizione della propria immagine ad un'altra retroproiettata su un telo posto dietro lo specchio si ottiene utilizzando un particolare vetro semi-specchiante (detto "*silver mirror*") e controllando il rapporto tra illuminazione ambientale e illuminazione della zona retrostante lo specchio stesso. Per l'illuminazione si è utilizzato un circuito di controllo comandato da una scheda Arduino, che contemporaneamente comanda anche un motore su rotaia, per lo spostamento del telo da retro-proiezione posto dietro lo specchio: l'effetto ottico di sovrapposizione delle due immagini (quella riflessa e quella proiettata) si ottiene infatti soltanto se la proiezione si trova alla stessa distanza dell'osservatore rispetto allo specchio (così che l'osservatore confonda il raggio di luce riflesso dallo specchio con quello trasmesso dal retro). Un sensore di distanza a raggi infrarossi comunica ad Arduino la distanza del telo dallo specchio, consentendo un controllo più preciso della sua posizione. La distanza dell'utente dallo specchio è invece rilevata tramite l'informazione RGBD proveniente da un primo Kinect, il cui scopo è individuare la presenza di possibili utenti e calcolare la distanza media della sagoma in movimento per comunicarla ad Arduino, che controlla di conseguenza l'illuminazione

ambientale, la posizione del telo da proiezione e la luminosità della proiezione stessa. Un secondo Kinect posto alle spalle dell'osservatore si occupa invece di rilevarne la sagoma e di trasmettere questa informazione ad un videoproiettore, che illuminerà soltanto chi si sta avvicinando allo specchio, permettendo ad una normale videocamera HD posta a fianco del Kinect di acquisire correttamente l'immagine a colori che andrà proiettata sul telo dietro lo specchio, creando l'illusione desiderata.

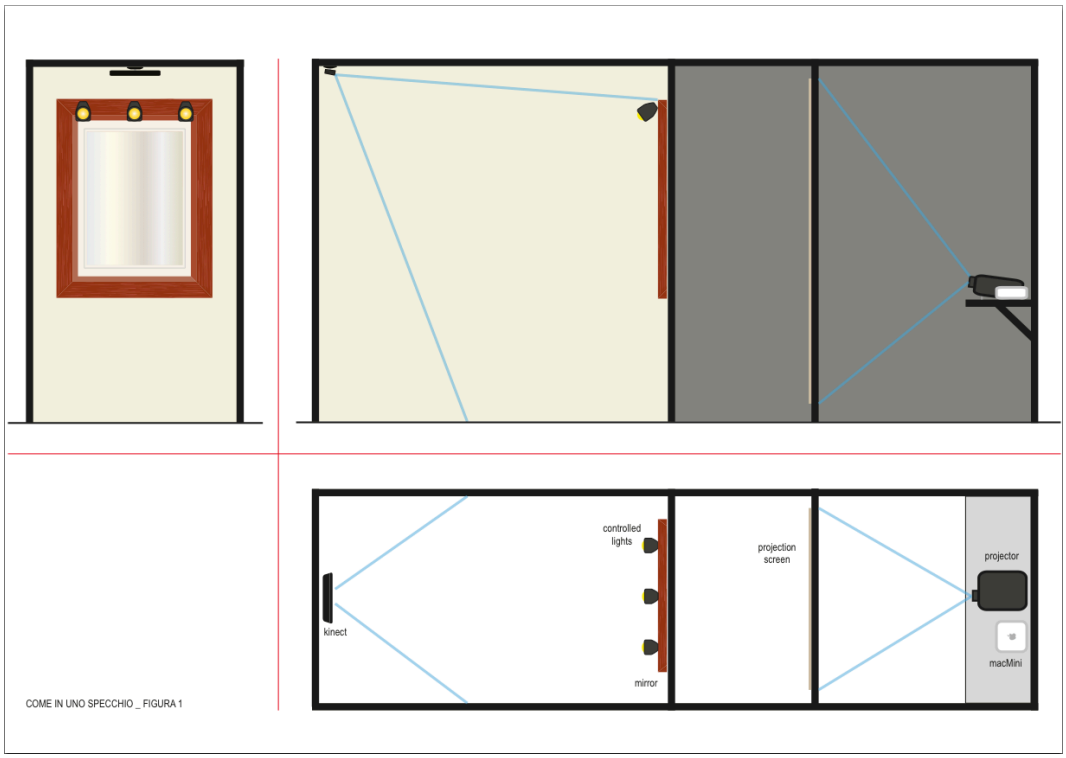


Figura 19 - Schema semplificato del prototipo di "perSpeculum"

4.4.4 Software

Per la complessità delle operazioni da eseguire, in particolare per l'elevato peso computazionale richiesto dalle operazioni di computer vision, ho scelto di sviluppare il software in C++, facendo uso di openFrameworks. Successivamente, sperimentando le richieste stringenti di risorse da parte di due Kinect (ed evitando al contempo la contaminazione dell'ambiente espositivo con il passaggio di cavi di segnale) sono stati utilizzati due computer differenti, uno posto dietro lo

specchio e l'altro diametralmente opposto, alle spalle del visitatore, comunicanti tra loro, utilizzando la rete wireless. Un primo computer, alle spalle dell'utente, è quindi connesso alla videocamera HD, ad un Kinect e ad un videoproiettore; l'altro, ospitato dalla struttura retrostante lo specchio, riceve dati (immagini a colori e informazioni di distanza) dal primo, comunica via USB con Arduino (al quale è riservato il compito di movimentare il telo da proiezione ed inviare i corretti segnali di controllo per il dimmer di potenza delle luci ambientali) ed è collegato ad un secondo Kinect e ad un secondo videoproiettore.

Per implementare questo sistema complesso ho fatto largo uso degli *addons* di openFrameworks, che hanno facilitato notevolmente lo sviluppo del software:

- *ofxOpenNI* per l'acquisizione immagini da Kinect, sfruttando la capacità propria della libreria OpenNI di identificare i singoli utenti nello spazio e tenerne traccia mediante assegnazione di identificativi progressivi;
- *ofxRemoteCameraServer*, modificato per trasmettere lo stream video HD in rete senza perdere in qualità o in velocità di riproduzione;
- *ofxSimpleGui* per realizzare un'interfaccia user-friendly per la taratura del sistema (dal momento che si tratta di un'installazione fortemente dipendente dall'ambiente in cui è posizionata e dalle geometrie esatte tra i proiettori e le periferiche di acquisizione video).

OpenFrameworks si è rivelato valido strumento per lo sviluppo del software, fornendo sufficiente controllo nella gestione delle risorse, fattore molto importante a causa della gran quantità di dati da trasmettere ed elaborare, in aggiunta alla necessità di mantenere in esecuzione l'applicativo per molte ore di seguito senza rallentamenti o malfunzionamenti.

5. Conclusioni

"Art is making something out of nothing and selling it"

Frank Zappa

5.1 L'interaction designer

Le necessità imposte dal mercato e dall'evoluzione degli strumenti disponibili hanno portato alla comparsa di una nuova figura professionale forse ancora oggi non del tutto definita, ma identificata nell'interaction designer. Non possiamo infatti classificare le competenze necessarie a plasmare interazioni, in quanto un sistema interattivo cela sotto l'apparente semplicità del risultato finale (se progettato bene) una grande complessità di problemi, obiettivi e studi di diverso genere. Di seguito si propone una semplice schematizzazione delle aree disciplinari coinvolte nella realizzazione di un buon design, in una gerarchia di complessità e profondità di analisi:

- *Antropometria*, ovvero la progettazione "a misura d'uomo" nel senso più letterale della parola: l'utente deve poter interagire con dispositivi proporzionati a se stesso (ad esempio: schermo visibile e leggibile, tasti accessibili o touch screen sufficientemente sensibile, possibilità di reggere l'oggetto in mano nel caso di *hand-held device*, etc..) ed utilizzabili per la totalità delle persone a cui il dispositivo è rivolto; è una prima importantissima specifica di progetto a cui attenersi, che influenzerà tutte le scelte successive.
    - *Fisiologia*: dopo aver considerato l'oggetto fisico, si considerano le azioni; al pari delle dimensioni e della

conformazione nel corpo umano, le specifiche di progetto devono assecondarne le funzionalità, il modo in cui è possibile agire.

- *Psicologia cognitiva*: soprattutto nel caso di progettazione di dispositivi che presentano comportamenti "intelligenti", ma anche in ogni tipo di interazione che si indirizza all'essere umano, è particolarmente fondamentale conoscere come il nostro cervello recepisce ed elabora gli stimoli; ciò consente di progettare sistemi di interazione più naturale, elevando il livello comunicativo sia sul piano della ricezione degli stimoli da parte del sistema che nel restituire in maniera naturale delle risposte all'utente
- *Sociologia*: sia che il sistema che stiamo progettando preveda un'interazione tra gli utenti o che debba semplicemente essere integrato nel contesto del suo utilizzo, vanno considerati gli aspetti di interazione tra esseri umani, per far sì che il dispositivo elettronico sia un amplificatore della comunicazione o comunque si inserisca in maniera naturale tra le interazioni sociali.
- *Antropologia culturale*: se si sviluppano prodotti per il mercato globale bisogna tener conto anche dei fattori culturali, degli interessi e del tenore di vita medio dell'utenza a cui ci si rivolge; questo è valido in parte anche nello sviluppo di un qualsiasi sistema interattivo, per cui è bene sviluppare un linguaggio che si indirizzi ad ogni cultura, nei due versi della comunicazione. Siamo comunque ad un livello di complessità di design non sempre richiesto da tutte le realizzazioni, e riservato a produzioni su larga scala o per vasta utenza.
- *Ecologia:* da non trascurare l'impatto ambientale dell'utilizzo (o della realizzazione) del sistema interattivo, a favore della sostenibilità, in termini di scelta dei materiali ed energia utilizzata.

Se questo tipo di progettazione stratificata si applica normalmente per la produzione di oggetti di design su grande scala tramite la cooperazione, non è da escludere che l'interaction designer che si occupa di installazioni o

dispositivi specifici debba racchiudere in sé perlomeno un'infarinatura generale di tutte le discipline sopra citate, a cui aggiungere ovviamente le competenze tecniche per la progettazione elettronica del dispositivo, non più sufficienti in uno scenario di mercato come quello odierno.

Una definizione di *interaction design* lo descrive come *"The design of the subjective and qualitative aspects of everything that is both digital and interactive, creating designs that are useful, desirable, and accessible"*[14]: il progettista si muove nel mondo artificiale di bit, pixel, dispositivi di input e output, modelli concettuali dell'utente e organizzazione delle informazioni, ma deve saper soddisfare anche le categorie dell'utile, del bello e dell'accessibile, spaziando tra tutte le discipline sopracitate; si rivela pertanto essere una professione da costruire su solide basi di una delle discipline coinvolte, ma da estendere con l'esperienza a tutti gli altri campi di studio.

5.2 Opera d'arte o esercizio di stile

Partendo da queste considerazioni, e anche a fronte della semplicità offerta da numerosi strumenti descritti finora in questo elaborato, possiamo notare come il mondo della progettazione elettronica di sistemi interattivi spalanchi le sue porte ad esperti di ogni altro settore, rendendo secondarie in questo campo di applicazioni le capacità tecniche rispetto alla sensibilità artistica dell'ideatore. Il rischio di questa apertura è però un abbassamento del livello delle realizzazioni, ed un osservabile appiattimento della creatività: se da una parte si assiste infatti ad una sempre maggior diffusione di sistemi embedded presentati come strumenti di intrattenimento o didattici, pubblicitari o educativi, o addirittura come opere d'arte, d'altro canto si tratta nella maggior parte dei casi di un utilizzo abbastanza convenzionale di strumenti disponibili. La disponibilità di strumenti e l'aumento delle potenzialità forniscono materiale interessante ai veri artisti, ma inevitabilmente generano un

[14] *"Il progetto degli aspetti qualitativi e soggettivi di tutto ciò che è digitale e interattivo, rendendolo utile, accessibile e gradevole"* - vedi bibliografia [9]

substrato di realizzazioni di scarso valore, proprio perché alla portata di tutti; il movimento naturale dal concetto alla realizzazione (proprio di ogni opera degna di nota) si è troppo spesso invertito, diventando un'operazione da "cosa è disponibile" al "che significato posso dargli". Questo limite alle grosse potenzialità della digital art proviene egualmente dai deu estremi: da una parte gli artisti di formazione più umanistica possono accedere soltanto agli strumenti tecnici più semplici, e quindi vedono limitate le possibilità realizzative; dall'altra, ingegneri e designer di formazione scientifica possiedono conoscenze tecniche più approfondite che consentirebbero loro realizzazioni più complesse e interessanti, ma non hanno sufficiente formazione per eccellere sul piano creativo.

Le eccezioni che portano invece a risultati notevoli, che incoraggiano il riconoscimento della digital art come espressione vera e propria dell'arte contemporanea, sono spesso frutto della collaborazione tra eccellenze di più discipline, oppure sono firmate da chi, partendo da una specializzazione su uno dei due versanti, ha saputo avvicinarsi all'altro ed apprendere di volta in volta competenze dai collaboratori con cui ha lavorato. Ipotizzando una crescita futura dell'interaction design, la figura che acquisirà maggior valore sarà, a mio parere, proprio quella dell'ingegnere che saprà apprendere maggiormente dalle altre discipline, e potrà fornire le sua capacità tecniche per la produzione di dispositivi non convenzionali ed innovativi; contemporaneamente il mondo dell'arte dovrà maturare sempre minor diffidenza nell'interpellare la comunità tecnico-scientifica per creare punti d'incontro e nuovi stimoli in questo campo.

5.3 Il dispositivo e il linguaggio ideali

A conclusione dell' analisi del mondo della digital interactive art, facendo riferimento anche alla mia personale esperienza lavorativa in

questo campo, è spontaneo interrogarsi sui possibili futuri sviluppi della materia.

Osservando il mondo dell'open source rivolto alla creatività si può notare un'iniziale forte differenziazione degli strumenti software, ciascuno con il suo scopo preciso e specifico, seguita da una lenta riunificazione dei linguaggi in strumenti di sviluppo più generici che offrono la possibilità di realizzare applicazioni più complesse e diversificate, mantenendo al contempo le potenzialità specifiche dei singoli strumenti; questa tendenza unificativa ha fatto sì che, dopo un periodo di incontrastata dominanza dei tool con IDE integrata (in particolar modo Processing), ad essere privilegiati nell'utilizzo (e quindi nel supporto della community di utenti, che genera nuovi utilizzatori e crea i presupposti per l'auto formazione e l'evoluzione dello strumento) sono i frameworks, per la semplicità di estensione ai contributi degli sviluppatori e per l'implicito punto di forza di non dover richiedere all'utente di imparare un nuovo linguaggio di programmazione. Questo movimento mostra anche come il mondo dell'interaction si stia spostando verso competenze più specifiche di sviluppatori che hanno già una base di conoscenze informatiche, lasciando presagire una progressiva crescita positiva del livello delle realizzazioni. Lo scenario più plausibile è che pochi tra questi linguaggi sopravvivranno, e i più usati diventeranno sempre più portabili su diverse piattaforme ed utilizzabili anche con differenti linguaggi di programmazione.

Nel campo dell'hardware, relativamente più "giovane", possiamo prevedere un movimento analogo, seppur a mio parere più lento: dalle prime piattaforme di prototipazione a microcontrollore siamo passati a sistemi molto più potenti e più accessibili, anche economicamente; le schede come Arduino hanno davanti a sé ancora molte possibilità di sviluppo e di crescita di popolarità, perlomeno fintanto che sistemi embedded più complessi non saranno utilizzabili in maniera più semplice: la natura "open" di questi ultimi infatti impone l'impiego di software indipendenti e talvolta meno intuitivi di quanto ci si aspetta, come è il caso della distribuzione Linux consigliata per Raspberry. Ci si può aspettare che con l'evoluzione ulteriore dell'hardware compariranno anche sistemi

operativi pensati appositamente per lo sviluppo di sistemi embedded, computer art e sistemi interattivi o generativi in genere.

Lo scenario futuro vede l'integrazione in spazi sempre più ridotti di veri e propri computer sempre più potenti; in maniera naturale sorgeranno metodi semplici ed immediati di interfacciarsi a sensori e attuatori, anch'essi lanciati in uno sviluppo che li porterà ad essere più precisi, miniaturizzati ed accessibili; di pari passo, se l'attività finora ininterrotta e straordinariamente produttiva delle community di sviluppatori open source continuerà a questo ritmo, saranno disponibili strumenti software per ogni tipo di applicazione da realizzare, spostando gli scenari dell'interazione ad un livello sempre più umano e realistico.

BIBLIOGRAFIA

[1] BANZI, M. (2009) *Getting started with Arduino*, O'Reilly Media

[2] CRAWFORD, C. (2002) *The Art of Interactive Design: A Euphonious and Illuminating Guide to Building Succesful Software*, No Starch Press

[3] IGOE, T., O'SULLIVAN, D. (2004) *Physical Computing*, Thomson Course Technology PTR

[4] LOWGREN, JONAS (2008): *Interaction Design*. In: Soegaard, Mads and Dam, Rikke Friis (eds.). "Encyclopedia of Human-Computer Interaction". Aarhus, Denmark: The Interaction Design Foundation.

[5] NOBLE, J. (2009) *Programming Interactivity*, O'Reilly Media

[6] REAS, C., FRY, B. (2007) *Processing: a programming handbook for visual designers and artists*, MIT Press

[7] SEGATELLO, M. (2010) *Conoscere e usare Arduino*, inserto monografico di Elettronica IN, Maggio-Giugno 2010

[8] BORENSTEIN, G. (2012) *Making things see*, O'Reilly Media

[9] MORRIDGE, B. (2007) *Designing Interaction*, MIT Press

INDICE DELLE ABBREVIAZIONI

AI: Artificial Intelligence
CLI: Command Line Interface
CSI: Camera Serial Interface
DSI: Display Serial Interface
GPIO: General Purpose Input/Output
GPL: General Public License
GPU: Graphics Processing Unit
GUI: Graphical User Interface
HCI: Human Computer Interaction
I/O: Input / Output
I2C: (I^2C) Inter Integrated Circuit
IA: Intelligence Amplification
IDE: Integrated Development Environment
MIT: Massachusetts Institute of Technology
PWM: Pulse-Width Modulation
RAM: Random-Access Memory
RGBD: Red-Green-Blue-Depth
SDK: Software Development Kit
SoC: System On a Chip
SPI: Serial Peripheral Interface
TTL: Transistor-Transistor Logic
USB: Universal Serial Bus