POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria del Cinema e dei Mezzi di Comunicazione

Tesi di Laurea Magistrale

Sviluppo di un sistema digitale interattivo

Relatore
prof. Giovanni Malnati

Candidata
Matilde Ugolini
matricola: 239432

Anno accademico 2018-2019

Sommario

L'oggetto di studio della presente Tesi di Laurea Magistrale è la progettazione finalizzata alla prototipazione di CUB8, un sistema digitale interattivo dalle fattezze cubiche, che si sviluppa nel sempre più in fermento mondo dell'IoT. Il progetto nasce dalla volontà di creare un dispositivo modulare e componibile che possa essere impiegato principalmente in ambito educativo per avvicinare bambini e tecnologia in modo non invasivo, così come nel mondo delle installazioni artistiche interattive audio-visive, che vedono al centro dell'opera lo spettatore e il suo interagire con essa. Un altro esempio di applicazione pratica degno di nota è l'utilizzo di CUB8 come intermediario in determinate interazioni sociali: dal proporsi come strumento di coordinamento per attività ludico-educative di monitoraggio delle dinamiche all'interno di gruppi più o meno eterogenei, alla psicoterapia infantile, come mezzo di agevolazione dell'interazione fra bambino e terapeuta rappresentando un punto di incontro reattivo e proattivo tra gioco e terapia. CUB8 rappresenta dunque un'interfaccia fisica e tangibile tra utente e applicativo finale, il quale può avere le più svariate implementazioni, in base al contesto di utilizzo. Ogni CUB8 oltre a possedere un'interattività intrinseca legata ai suoi movimenti relativi, possiede anche un'interattività derivante dalla composizione con uno o più dei suoi pari - composizione realizzata grazie alle facce magnetiche del cubo. L'utente infatti, manipolando uno o più cubi - facendoli ruotare, muovendoli e unendoli - interagisce con il sistema modificandone la natura a suo piacimento. Il sistema in questione necessiterà dunque di periferiche di output per dare feedback all'utente riguardanti la manipolazione in corso (d'opera). Queste periferiche di supporto saranno tendenzialmente rappresentate da schermi o proiettori per il lato visivo di riproduzione dell'ambiente virtuale e casse per la riproduzione dei suoni. Per quel che concerne il campo di applicazione educativo si vorrebbe mantenere il sistema quanto più analogico possibile, in modo da non dover costringere i bambini all'utilizzo di CUB8 davanti ad uno schermo, ma attivarne comunque la curiosità verso la tecnologia. Oltre alle periferiche di output, i CUB8 avranno anche bisogno di un elaboratore centrale che possa ricreare la topologia d'insieme dei cubi attivi e del loro "vicinato": deve dunque prendere coscienza della loro presenza/attività,

interpretarne lo stato e riprodurre sulle periferiche gli effetti che la manipolazione perpetuata/operata dall'utente ha avuto sul mondo virtuale. L'obbiettivo principale di tale Tesi di Laurea sarà dunque quello di celare la complessità del sistema in esame dietro alla semplicità di un cubetto.

Indice

Elenco delle tabelle 6

Elenco delle figure 7

I Capitolo Primo **9**

1 Introduzione 11
- 1.1 Scopo e ambito del progetto . . . 11
- 1.2 Requisiti di progettazione . . . 12
  - 1.2.1 Dimensioni . . . 13
  - 1.2.2 Componibilità . . . 13
  - 1.2.3 Illuminazione come feedback visivo . . . 14
  - 1.2.4 Progettazione estetica e materiali . . . 15

II Capitolo secondo **19**

2 Implementazione del progetto 21
- 2.1 CUB8 versione 1.0 . . . 21
- 2.2 Progettazione hardware ed estetica di CUB8 . . . 22
  - 2.2.1 Schema circuitale e produzione del circuito stampato . . . 22
  - 2.2.2 Design del contenitore e scelta dei materiali . . . 24
  - 2.2.3 Comunicazione infra-cubo e illuminazione . . . 25
- 2.3 Ambienti di sviluppo . . . 28
  - 2.3.1 Arduino IDE . . . 28
- 2.4 Componenti elettronici selezionati . . . 29
  - 2.4.1 Microcontrollore principale Wemos D1 mini . . . 29
  - 2.4.2 Microcontrollore ausiliario ATtiny85 . . . 33
  - 2.4.3 Modulo ricevitore a infrarossi VS1838B e LED trasmettitore . . . 36

2.4.4 IMU module: BNO055 . . . 37
2.4.5 Illuminazione tramite LED RGB WS2812 . . . 37
2.4.6 Alimentazione del circuito: batteria 18650 e modulo di ricarica TP4056 . . . 38
2.5 Progettazione software versione 1.0 . . . 40
2.5.1 Elaborazione dell'orientamento dei cubi tramite sensore BNO055 . . . 40
2.5.2 Protocollo di comunicazione IR degli Id tra cubi connessi 48
2.5.3 Controllo dei LED RGB WS2812 . . . 54
2.5.4 Protocollo di comunicazione su rete Wi-Fi del sistema CUB8 . . . 55
2.5.5 Applicazione sul campo: Game's Week 2018 (MI) . . . 58
2.5.6 Considerazioni tecniche e migliorie apportabili al progetto 60

III Capitolo terzo 63

3 Evoluzione del progetto 65
3.1 Componenti elettronici selezionati . . . 65
3.1.1 Modulo di lettura RFID PN532 V3 . . . 65
3.1.2 Microcontrollore ausiliario Arduino Pro Mini . . . 67
3.2 Evoluzione della progettazione software . . . 69
3.2.1 Acquisizione degli Id esterni tramite lettura RFID . . . 69
3.2.2 Protocollo di comunicazione su rete Wi-Fi del sistema CUB8 . . . 70
3.3 Evoluzione della progettazione hardware . . . 75
3.3.1 Schema circuitale aggiornato e produzione del circuito stampato . . . 75
3.3.2 Design del contenitore . . . 79

IV Conclusioni e sviluppi futuri 83

Elenco delle tabelle

2.1 Specifiche della board Wemos D1 Mini 32
2.2 Specifiche dettagliate del modulo di ricarica, da RobotStore [14] 39
2.3 Operating fusion modes overview 41
2.4 Protocol select pin mapping. Da BOSCH [4] 42
2.5 Protocol select pin mapping. Da BOSCH [4] 43
2.6 I2C address selection. Da BOSCH [4] 43
2.7 Tabella di traduzione dei codici di diagnostica. Da BOSCH [4] . 47
3.1 Switch settings 66
3.2 Collegamento fra i pin SPI del modulo PN532 e i pin di Arduino Pro Mini 68

Elenco delle figure

1.1 Logo di CUB8, creato da Segalini Simone 11
1.2 Evoluzione del design del contenitore di CUB8 nel tempo 15
1.3 Render relativi al terzo modello da sinistra in figura 1.2 16
1.4 Render della faccia modulo componibile 17
1.5 Cubo con facce modulari componibili con PCB esagonale: render e stampa 3D 17
2.1 Versione 1.0 di CUB8 21
2.2 Esempi di composizione direzionale dei CUB8 1.0 22
2.3 Schema circuitale di CUB8 versione 1.0, disegnato con il tool di design circuitale EasyEDA 23
2.4 Progetto PCB su EasyEDA e realizzazione fisica (top e bottom layer) 24
2.5 Scheletro e scassi in opalino bianco 24
2.6 In ordine: basette di supporto; Moduli a C complementari, con batteria e PCB; Struttura chiusa 25
2.7 Distanza e diffusione non ottimale e ottimale a confronto 26
2.8 Parallelepipedo di supporto 27
2.9 Sospensione della torretta con fascette 27
2.10 Risultato finale della prima versione di CUB8 28
2.11 Struttura di uno sketch Arduino 28
2.12 ESP8266EX 29
2.13 Schema da “ESP8266 Arduino Core Documentation” di Grokhotkov [9] 30
2.14 Schema da “ESP8266 Arduino Core Documentation” di Grokhotkov [9] 30
2.15 Schema da “ESP8266 Arduino Core Documentation” di Grokhotkov [9] 31
2.16 Modulo ESP-12S, by AI-T 31
2.17 Wemos D1 mini vista frontale con pinout e vista bottom 33
2.18 MCU ATtiny85, by Atmel 34
2.19 ATtiny25/45/85 pinout 35

2.20 Modulo VS1838B e LED trasmettitore . . . 36
2.21 IMU Module, by Robot Electronics . . . 37
2.22 BNO055 Sensor, by BOSCH Sensortec . . . 37
2.23 Battery 18650, by Sony . . . 38
2.24 TP4056 Module . . . 38
2.25 Collegamento modulo TP4056-batteria . . . 39
2.26 Collegamento modulo TP4056-batteria . . . 40
2.27 Canale LK1 . . . 42
2.28 Schematico del collegamento con l'mcu Wemos D1 mini . . . 42
2.29 Esempio di accesso I2C in lettura multipla. S: start; P: stop; ACKS: ack by slave; ACKM: ack by master; NACKM: Not ack by master; RW: read/write mode (dal datasheet di BOSCH [4]) 44
2.30 Esempio: rotazioni su x-y-z, da *Wolfram.com* . . . 45
2.31 Definendo le rotazioni in angoli di Eulero come dei sistemi cardanici con tre cerchi rappresentanti i gradi di libertà del sistema (yaw, pitch e roll), il gimbal lock accade quando due cerchi si allineano e l'intero sistema può muoversi solo in due dimensioni - da *gimbalsystem.com* . . . 45
2.32 Rappresentazione visiva del quaternione . . . 46
2.33 Segnale trasmesso modulando la luce di un LED . . . 49
2.34 Schema funzionamento Interrupt . . . 50
2.35 Segnale rilevato sul pin OUT del modulo VS1838B . . . 51
2.36 Esposizioni dei CUB8 presso lo stand di *Tonic Minds* alla Game's Week di Milano . . . 58
3.1 Modulo PN532 NFC RFID, by Elechouse . . . 66
3.2 Arduino Pro Mini, by SparkFun Electronics . . . 67
3.3 Macchina a stati del protocollo in rete di CUB8 . . . 71
3.4 Pagina web del captive portal di CUB8 . . . 72
3.5 Schema di massima della fase di *advertising* . . . 73
3.6 Schema di massima della fase *mqtt_connected* . . . 73
3.7 Render realizzato con *Blender* . . . 76
3.8 Schematici dei tre sotto-circuiti di CUB8 . . . 77
3.9 Design delle PCB di CUB8 . . . 78
3.10 Render della scanalatura del fill pattern e del CUB8 risultante atteso . . . 79
3.11 Render della struttura trasparente e del CUB8 risultante "acceso", contenente anche il cubo interno composto da PCB . . . 80

Parte I

Capitolo Primo

Capitolo 1

Introduzione

1.1 Scopo e ambito del progetto

L'oggetto di studio della presente Tesi di Laurea Magistrale è la creazione e sviluppo di *CUB8*, un sistema digitale interattivo dalle fattezze cubiche. CUB8 nasce dalla volontà di creare un dispositivo modulare e componibile che possa essere impiegato principalmente in ambito educational per avvicinare bambini e tecnologia in modo non invasivo, così come nel mondo delle installazioni artistiche interattive audio-visive, che vedono al centro dell'opera lo spettatore e il suo interagire con essa.

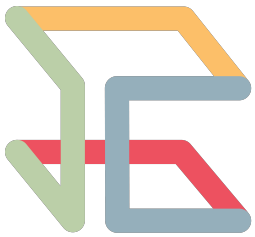


Figura 1.1: Logo di CUB8, creato da Segalini Simone

CUB8 rappresenta dunque un'interfaccia fisica e concreta tra utente e applicativo finale, il quale potrà avere le più svariate implementazioni, in base al contesto di utilizzo. Ogni CUB8 oltre a possedere un'interattività intrinseca legata ai suoi movimenti relativi, possiede anche un'interattività derivante dalla composizione con uno o più dei suoi pari. L'utente infatti, manipolando uno o più cubi, facendoli ruotare, muovendoli e unendoli, interagisce con il sistema modificandone la natura a suo piacimento.

Il sistema in questione necessiterà dunque di periferiche di output per dare feedback all'utente riguardanti la manipolazione in corso. Queste periferiche di supporto saranno tendenzialmente rappresentate da schermi o proiettori per il lato visivo di riproduzione dell'ambiente virtuale e casse per la riproduzione dei suoni. Per quel che concerne il campo di applicazione educativo si vorrebbe mantenere il sistema quanto più analogico possibile, in modo da non dover costringere i bambini all'utilizzo di CUB8 davanti ad uno schermo, ma attivarne comunque la curiosità verso la tecnologia.

Oltre a periferiche di output, i CUB8 avranno anche bisogno di un elaboratore centrale che prenda coscienza della loro presenza e/o attività, ne interpreti lo stato e ne riproduca sulle periferiche gli effetti che la manipolazione effettuata dall'utente sta avendo sul mondo virtuale.

L'obbiettivo di tale Tesi di Laurea, sarà dunque la realizzazione del primo prototipo di CUB8, ponendo principalmente il focus sull'oggetto fisico, piuttosto che sull'applicativo finale con il quale andrà utilizzato. Essendo considerabile un'interfaccia tangibile, l'implementazione dell'ambiente virtuale dipende prettamente dal contesto di utilizzo e/o dall'ambiente di sviluppo scelto, e non dal *controller* CUB8, ma si plasmerà su di esso.

1.2 Requisiti di progettazione

I principali requisiti di design sono legati all'usabilità di CUB8 da parte dell'utenza. In quanto dispositivo manipolabile le sue dimensioni devono essere quanto più ridotte, in modo da poter essere sostenuti con una singola mano. Ciò è fondamentale anche in quanto dispositivo componibile: deve infatti essere possibile interagire agevolmente con più cubi nello stesso istante. Il problema delle dimensioni assume ancora più spessore se visto nell'ottica delle finalità principali di CUB8: il settore *educational* legato dunque ai bambini - cubi di dimensioni ingenti con conseguente aumento del peso, sarebbero poco adatti all'usabilità per questo target.

Un altro aspetto importante è quello sensoriale, legato alla percezione tattile del cubo. Ci si è dunque posti come obbiettivo l'utilizzo del legno, in quanto materiale organico in grado di celare all'utente la sua vera natura sintetica ed elettronica. L'utilizzo del legno è legato anche all'ultimo aspetto percettivamente importante per la User Experience finale: l'estetica del cubo. Per quanto paradossale, riprodurre progettualmente la semplicità visiva di un cubo rappresenta una sfida non indifferente. Esternamente quello che si vuole idealmente ottenere è un cubo perfetto, senza giunzioni di sorta - seppur dovendo mantenere la possibilità di aprire il cubo per esigenza o manutenzione - e senza evidenze dello spessore dei lati.

I vincoli di design più strettamente legati alla User Experience finale, sono anche quelli che più si scontrano con i vincoli di progettazione del sistema finale:

- Il prototipo dovrebbe avere dimensioni adatte alle mani di un bambino, *seppur contenendo al suo interno tutta la componentistica elettronica necessaria al corretto funzionamento del sistema*;

- Deve essere piacevole al tatto e alla vista, da ciò la scelta del legno, *sebbene tale scelta comporti limitazioni in termini di feedback visivo (illuminazione responsiva del cubo complicata) e di lavorazione.*

Questi e ulteriori requisiti fanno sorgere non poche problematiche - problematiche che verranno sviscerate più dettagliatamente nel corso della qui presente tesi.

1.2.1 Dimensioni

Come anticipato, le dimesioni di CUB8 rappresentano un fattore cruciale nel processo di design del prototipo. La lunghezza *ideale massima* entro la quale si desiderava mantenersi, era inizialmente di 6 cm di spigolo.

Tuttavia una restrizione a tale situazione ideale, è la dimensione della batteria adottata per alimentare il circuito elettronico. Si necessitava di una batteria al litio ricaricabile da ~3.6 V e capacità 3000 mAh. Sfortunatamente, le batterie che rispondono a tali requisiti hanno tendenzialmente dimensioni fisse espresse dalla sigla *18650*: 18 cm x 65 cm. Per tale motivo la lunghezza minima dello spigolo del cubo dovrà necessariamente essere maggiore della lunghezza della batteria.

Un altro aspetto da tenere in considerazione è lo chassis di CUB8, il quale spessore andrà naturalmente a sommarsi alla lunghezza dello spigolo precedentemente ipotizzata.

Si è dunque arrivati ad un accettabile e realistico compromesso: adottare come dimensione *massima* dello spigolo esterno del cubo 8 cm - soglia massima oltre la quale non si risponderebbe più al requisito per le dimensioni ridotte del cubo.

1.2.2 Componibilità

Si è finora parlato di componibilità senza però entrarne propriamente nel merito. Per componibilità si intende la possibilità di unire ogni cubo con: al massimo tanti altri cubi quante le sue facce; al minimo un altro cubo.

Questa modularità vuole lasciare quanti più gradi di libertà possibile all'utente. Non si è dunque ricercata una componibilità in stile "puzzle", ovvero totalmente vincolata e univoca, ma uno stile di composizione più libero, per dar modo all'utente di esplorare le possibili combinazioni. Ogni CUB8 sarà quindi potenzialmente componibile con qualsiasi altro suo "pari".

Essendo il movimento relativo del cubo un aspetto importante nell'interazione utente-sistema si è pensato ad una soluzione che lo preservasse - si noti che per movimento relativo si intende principalmente la rotazione attorno ad un

determinato centro di rotazione. Per questo motivo sin dal principio si è deciso di adottare come soluzione finale il magnetismo fra cubi, unico metodo che realmente consente fluidità di movimento e garantisce stabilità nel contatto, se correttamente scelto il magnete. Ogni CUB8 dovrà dunque essere provvisto di sei magneti permanenti, uno al centro di ogni faccia, in modo tale da consentire la rotazione indipendente del cubo attorno al magnete seppur facendolo rimanere a contatto con i cubi con i quali è unito.

Unico vincolo che si pone adottando tale soluzione è la direzionalità della forza attrattiva dei magneti permanenti, che per natura si attraggono per poli opposti. Ciò comporta per ogni cubo avere tre facce il cui magnete abbia polo Nord rivolto all'esterno e le altre tre facce complementari (con polo Sud verso l'esterno). Conseguentemente non tutte le facce saranno compatibili, ma ciò risulta funzionale alla comunicazione infra-cubo che deve avvenire fra cubi composti: ogni cubo deve infatti inviare ai cubi circostanti il proprio identificativo, e riceverne uno per ogni altro cubo collegato alle sue facce, in modo tale che l'elaboratore centrale possa ricreare una topologia d'insieme dei cubi e del loro "vicinato". Per la natura complementare delle facce si riesce dunque facilmente a creare un parallelismo tra facce Nord/Sud con le facce ricevitrici degli Id circostanti e facce trasmettitrici del proprio Id. Se quindi si decidesse che per convenzione le facce trasmettitrici dell'Id sono quelle in cui il magnete ha il polo Nord verso l'esterno, le facce Sud riceverebbero gli Id dei cubi circostanti. Le scelte implementative/tecniche di questo protocollo saranno descritte in dettaglio nei seguenti capitoli.

1.2.3 Illuminazione come feedback visivo

Sebbene si voglia quanto più celare la natura elettronica di CUB8 e farlo apparire come un oggetto "analogico", si è ritenuto importante fornire all'utente alcuni feedback endogeni che lo accompagnino durante la User Experience di CUB8, oltre che a quelli esogeni generati dalle periferiche output di supporto. Questi feedback endogeni, propri del singolo cubo, avranno natura visuale; in particolare si è deciso di adottare l'illuminazione delle facce del cubo come feedback di stato. Un cubo "spento" rappresenta ovviamente un cubo inattivo. L'illuminazione invece, che significa stato di attività e possibilità di utilizzo, sarà coerente con la rotazione del cubo. Un esempio implementativo potrebbe essere il seguente: ogni qualvolta cambi la faccia rivolta verso l'alto di un CUB8 (o meglio viene superato un determinato angolo di soglia) cambia anche il colore della luce che illumina il cubo stesso.

1.2.4 Progettazione estetica e materiali

A livello di design estetico di CUB8 sono state effettuate numerose prove, utilizzando materiali e metodi di assemblaggio differenti. In figura 1.2 sono mostrati i risultati principali frutto di questo processo tuttora ancora in divenire.

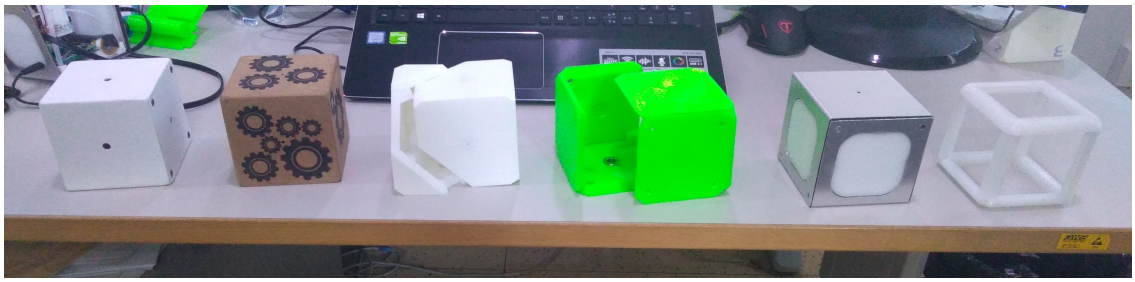


Figura 1.2: Evoluzione del design del contenitore di CUB8 nel tempo

I primi tentativi realizzati (in particolare i primi due cubi a partire da sinistra) non prevedevano ancora l'illuminazione dei cubi interattivi: presentano infatti un design molto semplice e sono stati utilizzati materiali totalmente opachi, che non permettono alla luce di passare attraverso. Il primo cubo mostrato a sinistra è stato realizzato in *forex*, materiale plastico in PVC semi-espanso o espanso, usualmente utilizzato nel campo della comunicazione per la realizzazione di cartelloni e insegne. Tale cubo è sigillato su cinque facce e permette l'apertura e chiusura (per l'inserimento della PCB del circuito e della batteria) avvitando la faccia fluttuante grazie a quattro viti poste sui vertici. Esternamente il cubo non mostra segni dello spessore delle facce; ciò è dovuto al fatto che queste sei sono state lavorate come un tronco di piramide, dello spessore ~6 mm, con svasatura a 45°. Il taglio a 45° permette alle facce di combaciare perfettamente. Il secondo cubo, sempre a partire da sinistra, è realizzato secondo gli stessi principi, ma è di MDF (*Medium Density Fiberboard*), materiale dalla finitura liscia e senza venature derivato da fibre di legno pressate ad alte temperature insieme a colla o resina. Si è voluto sperimentare questo tipo di materiale per avvicinarsi all'obbiettivo del legno compensato il quale, presentando venature naturali è più in linea con il risultato finale che si vorrebbe ottenere. L'MDF tuttavia è caratterizzato da una lavorazione più semplice, di conseguenza in fase di sperimentazione è risultato più comodo. Questi primi due cubi sono stati pensati per ospitare una PCB (Printed Circuit Board) quadrata, inseribile facendola scorrere grazie a due scanalature poste al centro di due facce parallele con lato di dimensione circa

$$l_{pcb} = l_{cubo} - 2 * spessore_{faccia} + 2 * profondità_{scanalatura} = \sim 7.3\,\text{cm}$$

Ciò forniva un'area operativa nella quale disporre tutti i componenti elettronici di ~5.3 cm.

Altre tipologie di assemblaggio del cubo sono mostrate sempre in figura 1.2 al posto due e tre a partire da sinistra. Queste prevedono di ospitare al loro interno una PCB di forma esagonale. Tale scelta è scaturita dal fatto che tagliando trasversalmente l'interezza del cubo a 54° viene a formarsi uno spazio esagonale. Una PCB di questa forma, a partire da un cubo con lato interno di 7.3 cm produce un'area utile di ~5.7 cm, apportando un incremento rispetto alla PCB quadrata. Inoltre suddivide meglio il volume del cubo, rispetto alle nostre necessità. Per implementare il protocollo IR della comunicazione infracubo della versione 1.0 di CUB8, infatti, era importante che moduli ricevitori e LED emettitori del proprio Id fossero ben schermati gli uni dagli altri. In particolare questi moduli dovevano essere posti all'interno dei magneti forati situati al centro delle facce anch'esse propriamente forate per lasciar passare la luce IR (vedi la sotto-sezione 2.2.3 per maggiori dettagli). Dividendo il volume del cubo tramite PCB esagonale, avviene una separazione netta fra i tre magneti contenenti i moduli ricevitori (se posti su tre facce ortogonali) e i magneti contenenti i LED emettitori.

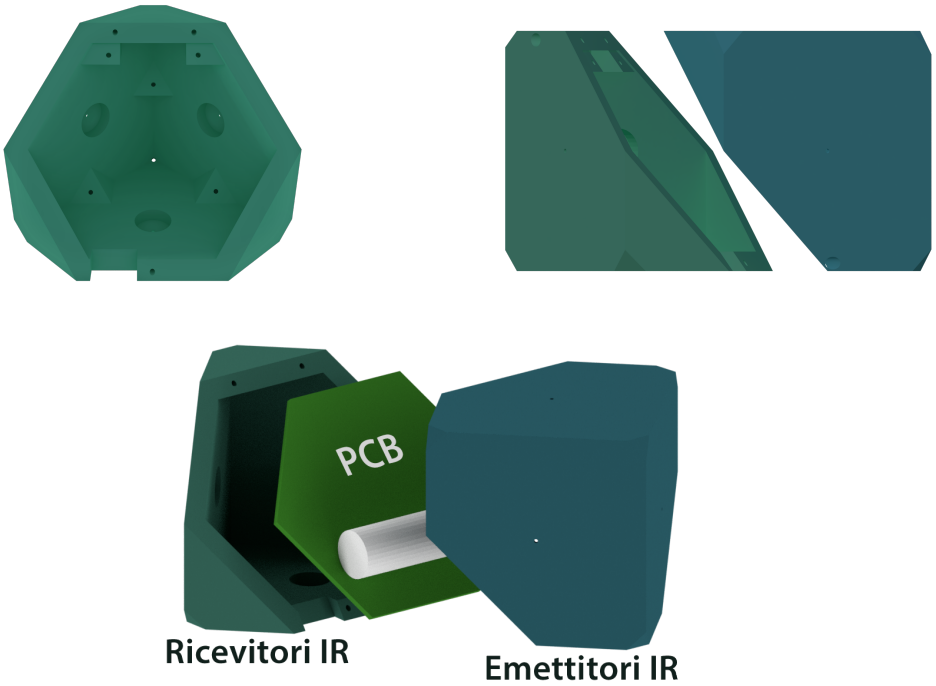


Figura 1.3: Render relativi al terzo modello da sinistra in figura 1.2

Il problema principale legato a tale soluzione era la scelta del materiale. Costruire un cubo in tale modo è fattibile solo se le due sue parti possono essere

create a partire da un "blocco unico". È quindi un processo poco sostenibile, a meno che non si decida di stampare in 3D i mezzi cubi, scelta però molto legata al campo della prototipazione e che poco si presta al prodotto finito.

L'integrazione delle tecniche fin qui esposte prende forma nel quarto prototipo da sinistra proposto in figura 1.2. A livello di realizzazione delle facce è del tutto simile ai primi due prototipi di forex e MDF essendo anch'esse tronco-coniche e come l'ultimo prototipo appena esposto ospita una PCB esagonale. Ciò è stato possibile effettuando una scanalatura trasversale che unisce i centri di due lati di ogni faccia (vedi figura 1.4), nella quale verrà adagiata la PCB esagonale, come si può intravedere nella figura 1.5.

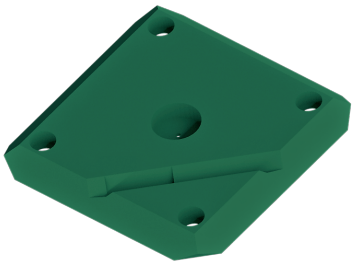


Figura 1.4: Render della faccia modulo componibile

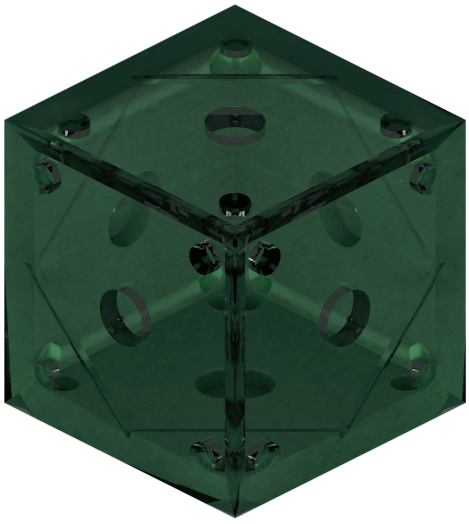


Figura 1.5: Cubo con facce modulari componibili con PCB esagonale: render e stampa 3D

Gli ultimi due prototipi rappresentati in figura 1.2 saranno discussi più approfonditamente rispettivamente nei Capitoli Secondo e Terzo essendo uno la realizzazione della versione 1.0 funzionante di CUB8 e l'altro il suo sviluppo futuro, attualmente in corso d'opera.

Parte II

Capitolo secondo

Capitolo 2

Implementazione del progetto

2.1 CUB8 versione 1.0

Dopo una prima fase di progettazione iniziale con annessa analisi dei requisiti, CUB8 ha visto la luce con una versione intermedia che indicheremo d'ora in avanti con *versione 1.0*, un giusto compromesso tra il soddisfare i requisiti quanto più possibile e l'effettiva necessità di realizzare un primo prototipo fisico da testare e sul quale poi plasmare la versione successiva di CUB8. Questo primo stadio di prototipazione ha portato alla riduzione del numero di facce componibili magneticamente da sei a due parallele - una ricevente ed una trasmettitrice degli Id - e all'utilizzo di un materiale diverso dal legno per facilitare l'illuminazione delle facce del cubo.

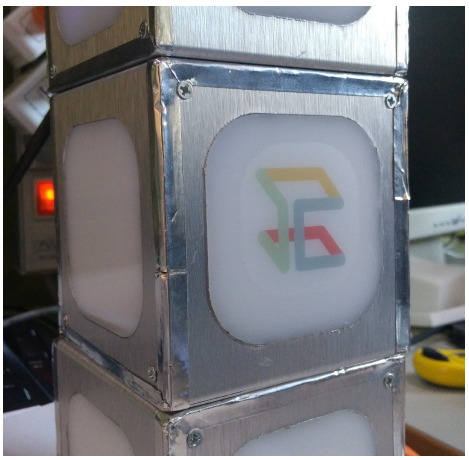


Figura 2.1: Versione 1.0 di CUB8

La versione 1.0 dunque prevede una componibilità direzionale: essendo componibili solo due facce parallele, la composizione del sistema durante l'utilizzo dell'applicativo si svilupperà nella direzione perpendicolare alle facce responsabili della comunicazione infra-cubo. La faccia rivolta verso il piano di appoggio (in situazione di riposo) *RX* è adibita al rilevamento degli Id del "vicinato" e, in caso di interazione con l'utente, essa verrà posta sulla faccia trasmettitrice

dell'Id *TX* di un altro cubo (ovvero la faccia rivolta verso l'alto). I magneti permanenti contenuti nella struttura interna del cubo, aiutano e guidano l'utente nel capire quali siano le facce effettivamente componibili, permettendo l'unione dei cubi solo se la configurazione $RX \Longleftrightarrow TX$ è soddisfatta e respingendoli in caso contrario.

Figura 2.2: Esempi di composizione direzionale dei CUB8 1.0

2.2 Progettazione hardware ed estetica di CUB8

Una volta ridefiniti i nuovi requisiti di sistema, si è passati alla progettazione hardware del circuito elettronico, in stretta correlazione con il design estetico della struttura portante del cubo stesso.

2.2.1 Schema circuitale e produzione del circuito stampato

Per prima cosa è stato disegnato lo schematico del circuito precedentemente creato e testato empiricamente su breadboard, come riportato in figura 2.3. Al contempo è stata modellata una simulazione 3D della *PCB* (*Printed Circuit Board*) e delle "bounding box" dei componenti elettronici coinvolti, passo fondamentale per poter fare un stima accurata degli spazi occupati dagli stessi e poter dare loro una disposizione spaziale efficiente e funzionale sulla superficie del circuito finale da stampare. Sulla base di questo modello 3D è stato possibile poi definire in modo più preciso le effettive misure del contenitore esterno e di tutte le strutture di supporto create ad hoc per la costruzione del primo prototipo.

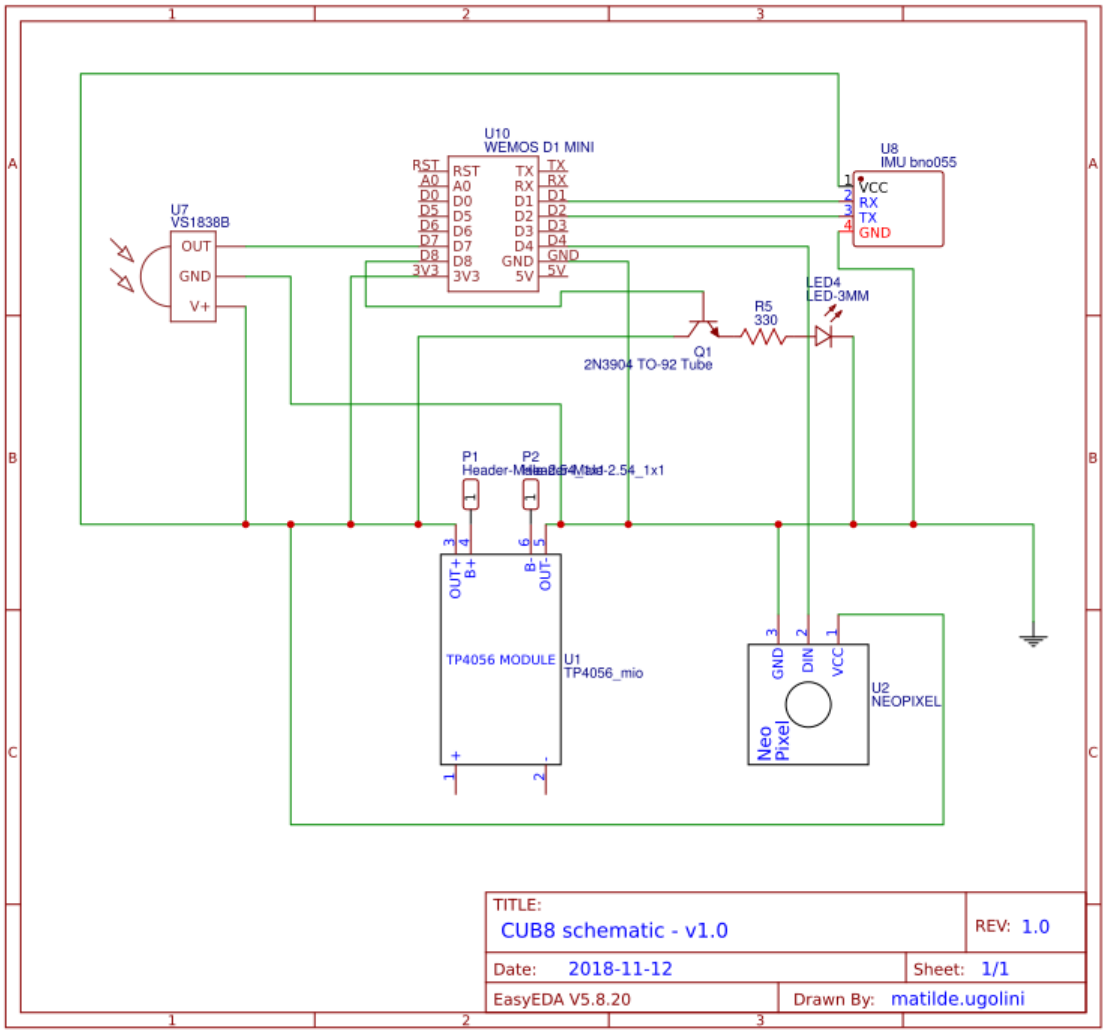


Figura 2.3: Schema circuitale di CUB8 versione 1.0, disegnato con il tool di design circuitale EasyEDA

Avendo definito la disposizione spaziale ottimale dei componenti elettronici si è infine passati alla fase di realizzazione della PCB vera e propria, utilizzando un apposito software di Circuit Design, EasyEDA. Il risultato ultimo e la sua riproduzione reale stampata sono riportati in figura 2.4.

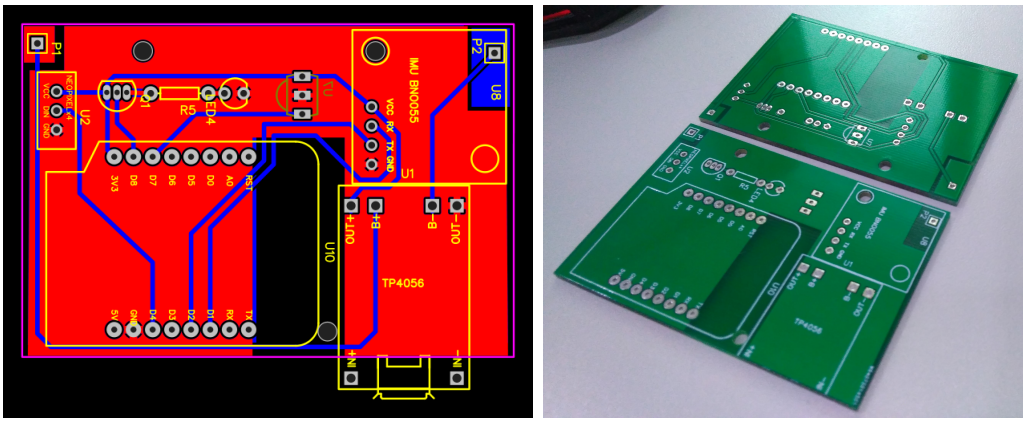


Figura 2.4: Progetto PCB su EasyEDA e realizzazione fisica (top e bottom layer)

2.2.2 Design del contenitore e scelta dei materiali

Per quel che concerne invece la scelta dei materiali del contenitore esterno di CUB8, sono stati utilizzati due diversi tipi di materiale:

- Materiale plastico rivestito in alluminio per la struttura di supporto del cubo, lo scheletro vero e proprio
- "Scassi" a incastro in plexiglass opalino bianco - per permettere un'ottimale diffusione della luce proveniente dall'interno del cubo - al centro delle facce non componibili

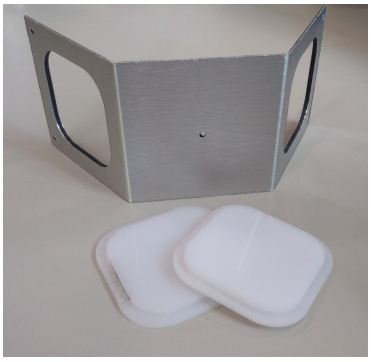


Figura 2.5: Scheletro e scassi in opalino bianco

In particolare il corpo del cubo è stato progettato per "incastrarsi a C": lo scheletro è infatti suddiviso in due parti modulari, che andranno a completarsi ricreandone così la figura intera (fare riferimento alla fotografia 2.6 in centro). Per ottenere una struttura solida sono state modellate e stampate in 3D due basette aventi stessa dimensione delle facce su cui andranno incollate - ovvero alla base e al "soffitto" del cubo. Tali strutture presentano ai vertici quattro volumetti che consentono di chiudere la struttura dall'esterno tramite l'utilizzo di due viti per faccia (per le quattro facce non componibili). Hanno inoltre un foro delle dimensioni dei magneti permanenti scelti ($\varnothing 15\,\text{mm}$) esattamente al centro, in modo da non smorzarne troppo la forza attrattiva. La basetta che

verrà incollata alla faccia sottostante, rispetto a quella del "soffitto", è più articolata. È infatti stata progettata per ospitare la PCB e i componenti elettronici ad essa saldati, e la batteria che alimenterà il circuito.

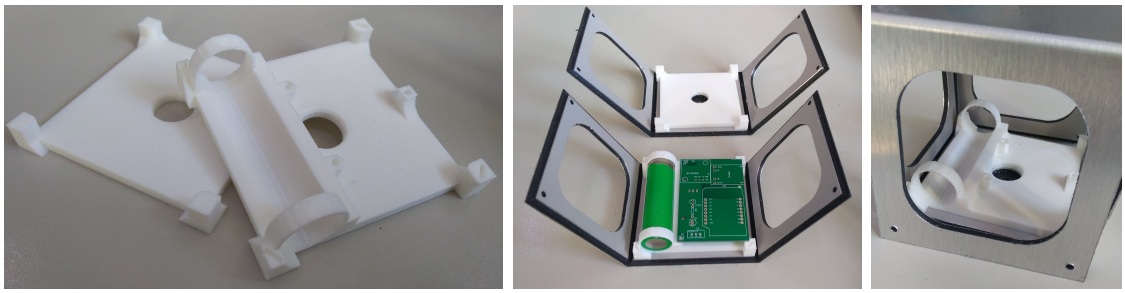


Figura 2.6: In ordine: basette di supporto; Moduli a C complementari, con batteria e PCB; Struttura chiusa

2.2.3 Comunicazione infra-cubo e illuminazione

Un aspetto fondamentale da tenere in considerazione in fase di design del contenitore esterno, è la modalità di comunicazione dell'Id fra cubi collegati magneticamente. A questo stadio si è deciso di implementare un protocollo di comunicazione a raggi infrarossi, quindi basato sull'utilizzo di un modulo ricevitore IR e un semplice diodo a emissione di luce, modulando quest'ultima a 38 kHz (per i dettagli vedere la sezione 2.4.3). Di conseguenza, le facce componibili (la base e la faccia a questa parallela) devono permettere alla luce di *penetrare dal basso* in fase di ricezione e di *fuoriuscire verso l'alto* in fase di trasmissione dell'Id. Ciò è stato effettuato realizzando un foro del diametro di ~2.5 mm nel centro delle facce d'interesse. I magneti occupano anch'essi il centro delle facce componibili, ma ciò non crea problemi nella comunicazione IR essendo stati scelti appositamente magneti permanenti forati. A questo punto il problema presentatosi è stata la modalità di collegamento del LED emettitore dell'Id al circuito elettronico situato sulla base del cubo. Dovendo il LED dirigere la luce verso l'alto, tale collegamento non è risultato semplice. In caso si fosse voluto mantenere il LED saldato direttamente alla PCB si sarebbe dovuto creare un sistema di incanalamento della luce fino al foro sul lato opposto del cubo, soluzione ovviamente realizzabile, ma con non poche problematiche, tra le quali la costruzione e l'assemblaggio del canale all'interno di uno spazio così ristretto, possibile distorsione e dispersione della luce lungo il tragitto, eventuale rischio di rilevamenti non voluti del segnale infrarosso da parte del modulo ricevitore dello stesso cubo.

Un altro ingente problema riscontrato è legato all'illuminazione dei cubi. A livello di feedback endogeno, quindi non relativo alla manipolazione dell'ambiente virtuale dell'applicativo finale, ma legato esclusivamente all'oggetto fisico CUB8, si è scelto di farli illuminare e far loro cambiare colore in modo coerente con la manipolazione che l'utente avrebbe operato.

Come anticipato, il problema dell'illuminazione è risultato molto più ostico di quanto ci si potesse aspettare. Illuminare dall'interno in modo uniforme e diffuso *tutte* le pareti di un cubo dalle così ridotte dimensioni, contenente al suo interno una serie di componenti che proietteranno per natura ombre sgradevoli, è tuttora infatti un problema irrisolto. Soprattutto se si considera il fatto che il centro di ogni faccia, nella versione finale a cui si tende, sarà per forza occupata dal magnete permanente che permette la componibilità dei cubi e che per forza di cose lascerà sempre una zona d'ombra molto netta.

Nella versione 1.0 di CUB8, tuttavia, solo due sono le facce comunicanti e dunque componibili. Le restanti quattro sono quindi libere dal magnete e dalla sua ombra. Di conseguenza si ha avuto più libertà in fase di progettazione visiva. Il centro delle facce che necessitano di illuminazione avranno uno "scasso" in plexiglass opalino bianco - che andrà ad incastrarsi nella struttura esterna del cubo - ed essendo l'opalino un materiale capace di diffondere la luce in ogni direzione con uguale intensità lascerà trapelare la luce dei LED all'esterno, diffondendola. Si voleva raggiungere un'illuminazione diffusa; di conseguenza i LED dovevano essere posti ad una certa distanza dalle pareti del cubo, in caso contrario l'illuminazione sarebbe risultata sgradevole e concentrata in un unico punto (immagine 2.7 a sinistra).

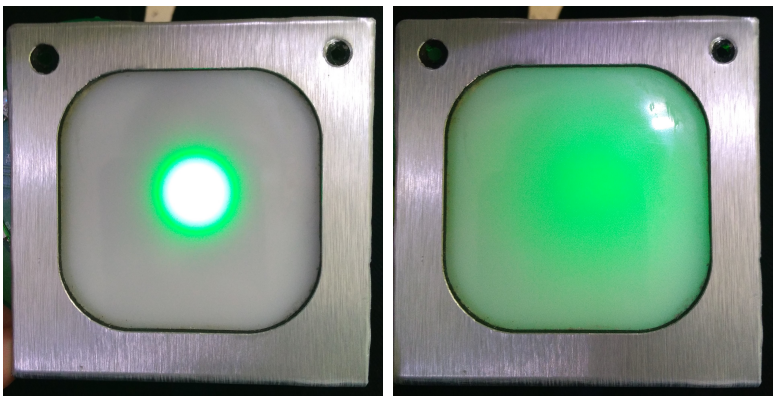


Figura 2.7: Distanza e diffusione non ottimale e ottimale a confronto

Soluzione adottata

La soluzione adottata infine è risultata ottimale, risolvendo tutti i problemi sopra elencati, sia riguardanti la disposizione del LED emettitore dell'Id che il problema dell'illuminazione diffusa.

È stato modellato e stampato in 3D un parallelepipedo (figura 2.8) di altezza ~4 cm e lato 2 cm internamente cavo, sulla quale estremità superiore è stato posto un quadrato 2 cm per 2 cm di basetta millefori. Su tale superficie è stato realizzato un piccolo circuito elettronico secondario, comprensivo del LED trasmettitore dell'Id, una resistenza da 390 Ω e un microprocessore *ATtiny85*, delegato a pilotare il LED stesso (non visibile in foto, essendo contenuto all'interno del parallelepipedo). Sulle pareti laterali del parallelepipedo è inoltre stata incollata una striscia di quattro LED RGB WS2812 in modo da circondarlo e far sì che ogni faccia del cubo avesse un LED parallelo alla stessa. Per alimentare questo circuito secondario sono stati sfruttati i pin *Vcc* e *GND* della striscia LED WS2812 la quale a sua volta viene alimentata grazie a tre jumper femmina collegati a tre connettori perpendicolari saldati alla PCB sottostante.

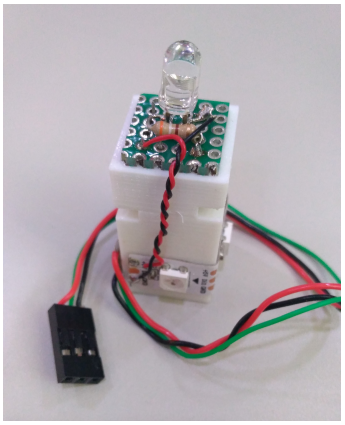


Figura 2.8: Parallelepipedo di supporto

Per mantenere poi sospeso il parallelepipedo in modo che l'illuminazione partisse dal centro del cubo e che il LED IR potesse essere vicino alla faccia superiore, si è utilizzato un sistema di fascette a incrocio passanti per le diagonali del parallelepipedo (e del cubo stesso) e terminanti sui volumetti della basetta montata sulla faccia di appoggio del cubo (fare riferimento alla figura 2.9). Le fascette, incrociandosi, formano un'ottima struttura di sostegno, se propriamente bloccate le estremità. Grazie alla loro resistenza è stato dunque possibile mantenere sospeso il parallelepipedo al centro del cubo, e al contempo grazie alla loro flessibilità, portare il LED trasmettitore dell'Id nel punto esatto del foro della faccia superiore del cubo, lasciandolo totalmente immerso nel magnete forato, permettendo quindi una perfetta trasmissione della luce all'esterno.

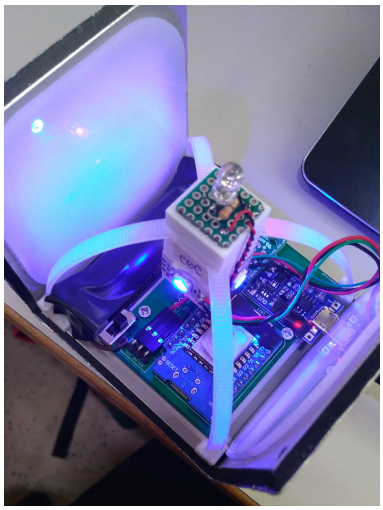


Figura 2.9: Sospensione della torretta con fascette

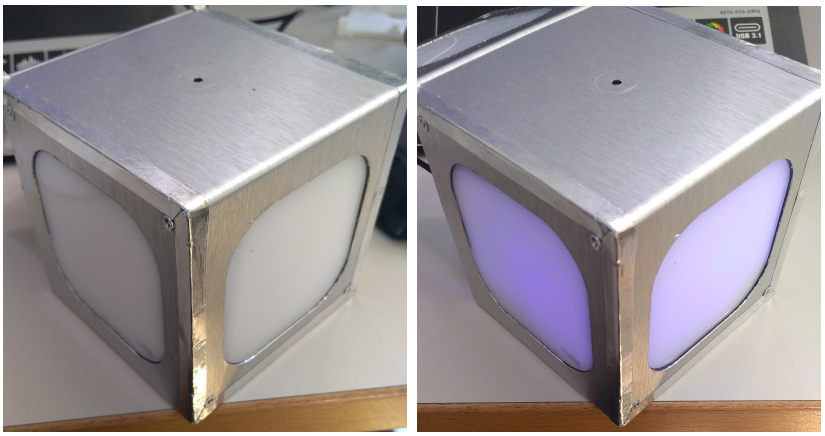


Figura 2.10: Risultato finale della prima versione di CUB8

2.3 Ambienti di sviluppo

2.3.1 Arduino IDE

Arduino è una piattaforma hardware open-source, diventata nell'ultimo decennio lo standard de facto nel mondo del Making e dei microcontrollori, grazie alla sua facilità di utilizzo e all'attivissima Community online. Il punto di forza di Arduino risiede proprio nei concetti di comunità e condivisione open-source.

Anche l'ambiente di sviluppo integrato (Integrated Development Environment) di Arduino è nato come software open-source multipiattaforma, da affiancare all'hardware e permetterne la programmazione. L'IDE consiste in un editor piuttosto intuitivo, integrato con un compilatore *C* e *C++*. La struttura generale di uno *sketch* (nome di un programma scritto su Arduino) è rappresentata nella figura 2.11. In termini di ambiente, l'IDE basa le sue fondamenta sull'altrettanto open-source software *Processing*, ambiente di programmazione in un contesto visivo utilizzato prevalentemente per avvicinarsi al mondo del coding, ai fini della prototipazione o per produrre visual arts generative.

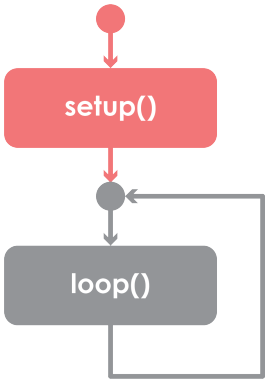


Figura 2.11: Struttura di uno sketch Arduino

Ciò che risulta maggiormente interessante ai nostri fini, è la possibilità di installare *Core* aggiuntivi nell'IDE di Arduino, in modo tale da rendere compatibili con questo ambiente di sviluppo diverse tipologie di microcontrollore e board. In particolare nel corso della presente Tesi si è usufruito della distribuzione *ESP8266 Arduino*

Core [6] che permette l'utilizzo di un'ampia famiglia di board basate sul microchip *ESP8266*. In particolare si è scelto di utilizzare la board Wemos D1 mini come microcontrollore principale, facente parte di tale famiglia (vedi sotto sezione 2.4.1).

2.4 Componenti elettronici selezionati

2.4.1 Microcontrollore principale Wemos D1 mini

Un qualsiasi sistema elettronico, in fase di prototipazione, richiede l'utilizzo di una *MicroController Unit* (MCU), un dispositivo elettronico single chip programmabile che integra al suo interno quanto gli sia necessario all'elaborazione di calcolo: dal microprocessore (*CPU*) alla memoria sia permanente (*FLASH/EEPROM/ROM*, per il programma da eseguire) che volatile (*RAM*), dall'oscillatore interno o esterno (*Clock*) alle periferiche (pin di *GPIO* - General Purpose Input Output - configurabili, porte seriali, interrupt, etc), fino ad arrivare ad una gamma di possibili estensioni nella funzionalità, in base al modello di MCU. Generalmente vengono impiegati in sistemi *embedded*, per ottimizzare costi, consumi e dimensioni.

I microcontrollori si suddividono in famiglie, all'interno delle quali condividono lo stesso microprocessore (stessi instruction set e compilatore), ma si differenziano gli uni dagli altri in termini di memoria integrata disponibile, periferiche aggiuntive (*DAC, ADC, uscite PWM, I^2C, SPI, USART, USB*) e contenitori (o *packages*) [7].

La famiglia di microcontrollori selezionata nel presente progretto di tesi è l'*ESP Series*, prodotta dalla casa manifatturiera *Espressif Systems*, che fornisce soluzioni SoC (System-on-a-chip) low-cost altamente integrate per applicazioni Wi-Fi che necessitino di un consumo energetico efficiente, un design compatto e affidabilità nel mondo dell'IoT (Internet Of Things) [15], ambito del quale fa parte CUB8.

In particolare si è fatto uso del microcontroller Wi-Fi *ESP8266EX*, un dispositivo integrato che implementa funzionalità avanzate come gli stack di rete (full TCP/IP stack) permettendo la creazione e utilizzo di reti Wi-Fi.

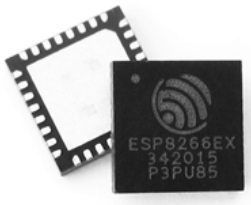


Figura 2.12: ESP8266EX

Andando ora a dettagliare il funzionamento dell'MCU ESP8266EX, questo a livello di specifiche tecniche supporta il protocollo Wi-Fi 802.11/b/g/n (HT20) ed opera dunque nel range di frequenze 2.4

~2.5 GHz della banda **ISM** (range di frequenze riservate all'uso **I**ndustriale, **S**cientifico e **M**edico). A livello software implementa inoltre tre modalità di collegamento alla rete Wi-Fi: *Station*, *Soft Access Point* e modalità mista *SoftAP+Station* [15].

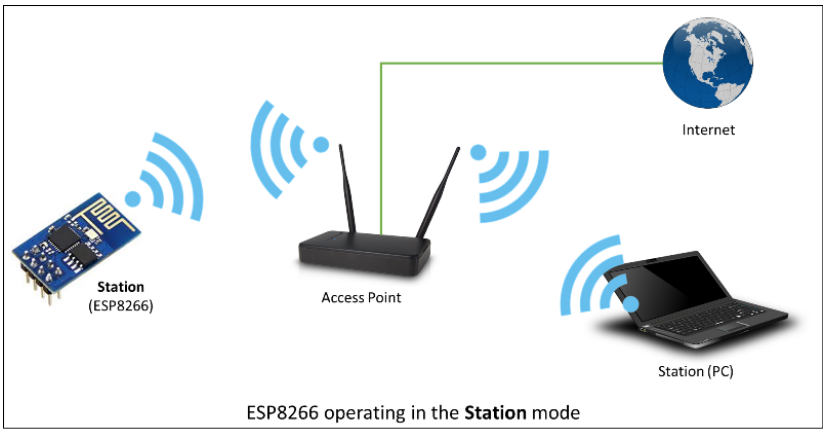


Figura 2.13: Schema da "ESP8266 Arduino Core Documentation" di Grokhotkov [9]

Quando il modulo ESP viene utilizzato in modalità *Station* (STA) (vedi figura 2.13), si connette ad una rete Wi-Fi pre-esistente tramite un Access Point che lo mette in comunicazione con altri device connessi alla medesima rete. La classe *Station* che implementa tale modalità ha diverse funzionalità di configurazione della connessione Wi-Fi e di management della stessa che ne agevolano l'utilizzo [9].

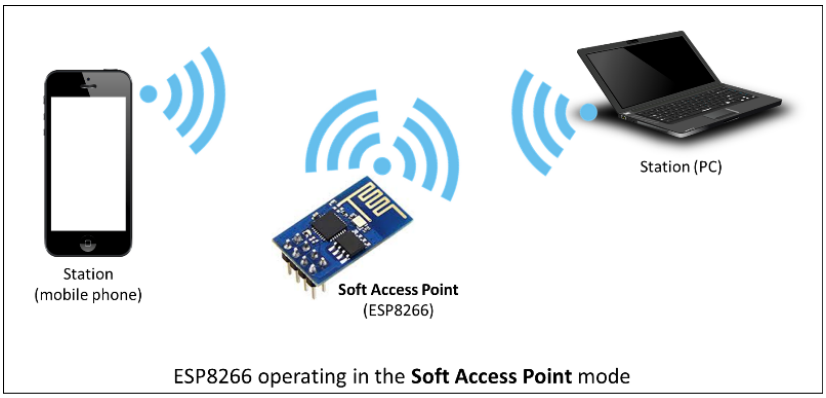


Figura 2.14: Schema da "ESP8266 Arduino Core Documentation" di Grokhotkov [9]

Se invece l'ESP si trova in modalità *SoftAP* (vedi figura 2.14) si comporterà come un *Access Point*, ovvero un dispositivo in grado di fornire ad altri dispositivi l'accesso ad una rete Wi-Fi, assegnando loro indirizzi IP tramite DHCP

(Dynamic Host Configuration Protocol) [7]. Il massimo numero di station che possono essere connesse contemporaneamente ad un SoftAP di default è 4, ma può essere modificato impostando in fase di configurazione un numero tra 0 a 8. Questa modalità viene spesso utilizzata come stato intermedio prima che l'ESP si connetta ad una rete Wi-Fi in Station mode. Non sempre infatti le credenziali della rete a cui si vuole accedere sono note a priori al modulo. L'ESP di conseguenza si avvierà dapprima in modalità Soft-AP, dando dunque la possibilità di collegarsi alla sua rete a device esterni (laptop o smartphone) i quali gli forniranno SSID e password per accedere alla rete Wi-Fi desiderata (e.g. tramite *Captive Portal*). Una volta verificato che la rete Wi-Fi target sia raggiungibile e che le credenziali fornite siano corrette, il modulo ESP passa alla modalità Station connettendosi a tale rete [9].

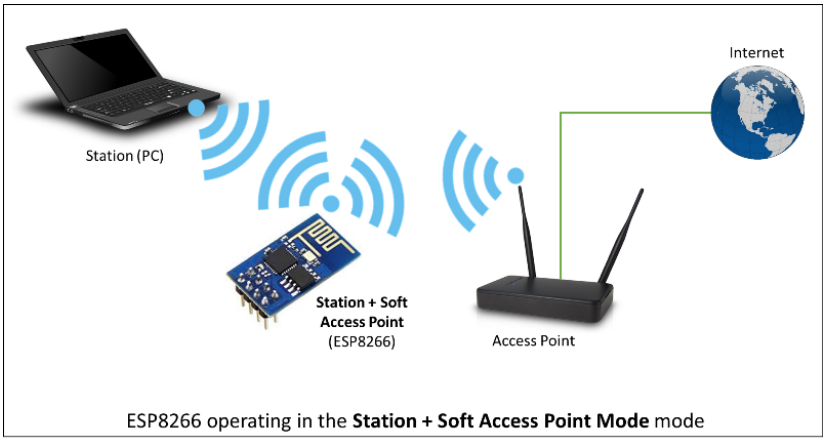


Figura 2.15: Schema da "ESP8266 Arduino Core Documentation" di Grokhotkov [9]

L'ultima modalità Wi-Fi implementata è la modalità così detta *Promiscuous mode* [15] (vedi figura 2.15) la quale prevede che il modulo operi simultaneamente sia in modalità Station che SoftAP come fosse un nodo di una WLAN: l'ESP sarà dunque Client di un Access Point esogeno, e allo stesso tempo fungerà egli stesso da AP per altri dispositivi [7].

Esistono numerose tipologie di moduli basati sull'ESP8266EX che si differenziano non solo in base al produttore/rivenditore, ma anche in base al design del modulo (forme, dimensioni e pinout diversi in relazione alle finalità del contesto applicativo). La casa produttrice di una delle più utilizzate serie di moduli ESP8266EX-based (non ufficiali *Espressif*) è l'*AI-Thinker* che produce moduli convenzionalmente chiamati *ESP-NN*, con *NN*

Figura 2.16: Modulo ESP-12S, by AI-T

codice progressivo legato al modello.

Solitamente questi moduli non sono dotati di componenti fondamentali ai fini di un processo di sviluppo più fluido, come ad esempio un modulo regolatore di tensione e un adattatore USB-to-serial, citando solo i moduli mancanti più importanti. Tali lacune vengono colmate tramite l'utilizzo di development board che integrano al loro interno questi moduli aggiuntivi, agevolando l'uso dei modulo ESP-based.

La board di sviluppo selezionata per la prototipazione di CUB8 è la board *Wemos D1 mini* (in figura 2.17) - prodotta dalla *Wemos* e basata sul modulo *ESP-12S* (in figura 2.16) della serie AI-Thinkers - scelta principalmente per le sue dimensioni compatte e i costi contenuti. Essa integra al modulo ESP-12S diversi moduli supplementari, fra cui i più importanti:

1. Uno strato-ponte USB-to-UART (in particolare grazie al modulo *WCH CH340G*) che funge da adattatore seriale per facilitare il debug da console

2. Una porta Micro-USB accoppiata ad un regolatore di tensione che trasforma la tensione in ingresso all'ESP8266EX in 3.3 V, essendo il livello di tensione operativa accettabile compresa nel range 2.5~3.6 V [15]

Queste integrazioni circuitali sulla board facilitano la programmazione e il collegamento al PC host del microcontrollore ESP [3].

Attributo	Valore
Program Memory Type	Flash
Operating Voltage	3.3V
Digital I/O Pins	11
Analog Input Pins	1(Max input: 3.2V)
Clock Speed	80MHz/160MHz
Flash	4M bytes
Length	34.2mm
Width	25.6mm
Weight	3g

Tabella 2.1: Specifiche della board Wemos D1 Mini

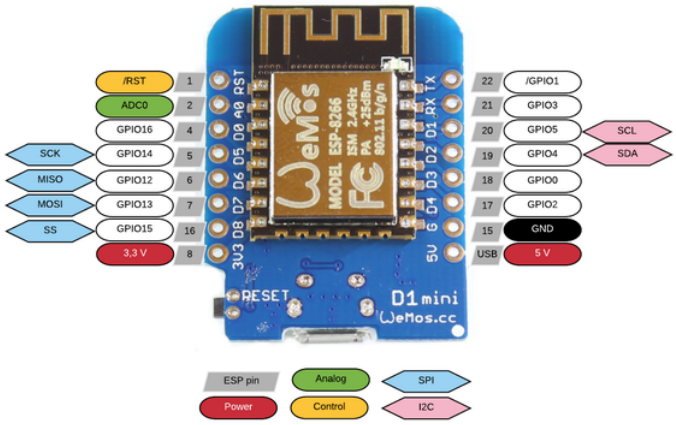


Figura 2.17: Wemos D1 mini vista frontale con pinout e vista bottom

Il modulo Wemos utilizzato mette a disposizione 11 pin di input/output predisposti ad attuare Interrupt, segnali in *PWM* (*Pulse Width Modulation*), protocolli *I2C* (*Inter-Integrated Circuit*) e SPI. In particolare la comunicazione tramite protocollo I2C può essere attuata tramite un bus che richiede solo due cavi dati, noti come *SDA* per i dati trasmessi, collegato al pin *D2* della MCU e *SCL* per il clock, collegato al pin *D1*. Questi due pin sono stati impiegati per connettere e far comunicare il microcontrollore con il modulo BNO055, rilevatore dei movimenti del cubo (vedi sotto-sezione 2.4.4). Per quanto riguarda gli altri componenti elettronici utilizzati nella prima versione di CUB8 sono stati impiegati il generico pin di GPIO *D4* per il collegamento dei led *NEOPIXEL* volti all'illuminazione reattiva del cubo, il pin *D7* per la rilevazioni a raggi infrarossi degli Id dei cubi vicini e la corrispettiva controparte trasmettitrice del proprio Id - un semplice LED pulsante nello spettro dell'infrarosso - sul pin *D8* (vedi sotto-sezione 2.4.3). Per quel che concerne l'alimentazione del circuito invece, il pin *3V3* sarà collegato al pin *OUT+* del circuito di ricarica *TP4056* (vedi sotto-sezione 2.4.6) che fornisce la tensione positiva in uscita dalla batteria *18650*.

2.4.2 Microcontrollore ausiliario ATtiny85

L'MCU ATtiny85 (in figura 2.18) fa parte della famiglia di microcontrollori *AVR* prodotti originariamente da *Atmel*. Gli AVR sono microcontrollori single-chip a 8-bit, con una Instruction Set Architecture di tipo RISC (Reduced Instruction

Set Computer), il che consente loro di eseguire meno cicli di esecuzione per istruzione (CPI) essendo le istruzioni più semplici e general purpose rispetto all'architettura CISC (Complexed Instruction Set). Sono stati inoltre fra le prime MCU ad utilizzare una memoria flash - riprogrammabile/riscrivibile e non volatile - per il caricamento e memorizzazione del programma da eseguire, ottimizzando i tempi di programmazione della scheda.

Figura 2.18: MCU ATtiny85, by Atmel

La sottofamiglia TinyAVR è tipicamente caratterizzata dall'avere un numero minore di pin di GPIO, meno memoria e meno funzionalità in generale rispetto agli altri microncontroller AVR (e.g. nessuna periferica USB), oltre che ad avere dimensioni più ridotte. Andiamo ora ad analizzare più nel dettaglio le specifiche tecniche dell'ATtiny85 [2]:

Attributo	Valore
Program Memory Type	Flash
Program Memory Size (KB)	8
CPU Speed (MIPS/DMIPS) Internal Clock Speed Grade	20 0 – 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
SRAM (Bytes)	512
Data EEPROM/HEF (Bytes)	512
Digital Communication Peripherals	1-SPI, 1-I2C
Capture/Compare/PWM Peripherals	5PWM
Timers Number of Comparators	2 x 8-bit 1
Interrupt Sources	External and Internal
Temperature Range (C)	-40 to 85
Operating Voltage Range (V)	1.8 to 5.5
Pin Count GPIO	8 6

Come si può vedere nella tabella sopra riportata, l'ATtiny85 fornisce una serie di funzionalità quali ad esempio (vedi anche pinout in figura 2.19):

1. Due Timer/Counter a 8-bit (dunque da 0 a 255)
   (a) Uno con Compare Mode
   (b) Uno ad alta velocità con due canali PWM ad alta frequenza
2. Possibilità di generazione di Interrupt interni ed esterni sui pin etichettati PCINT5:PCINT0
3. Sei porte (PB5:PB0) di GPIO programmabili via software in termini di direzione e output
4. Due periferiche di comunicazione seriale:
   (a) Una per *I2C* (Inter Integrated Circuit) sulle porte PB2⇒SCL per il clock e PB0⇒SDA per il dato
   (b) Una per *SPI* (Serial Peripheral Interface) sulle porte PB0⇒SCK collegato al valore del clock, PB1⇒MISO (Master Input / Slave Output) e PB0⇒MOSI (Master Output / Slave Input)

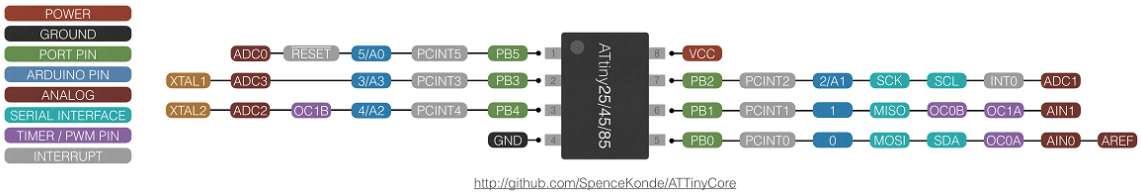


Figura 2.19: ATtiny25/45/85 pinout

Nel particolare caso studio, l'ATtiny85 risponde perfettamente all'esigenza di avere un microcontrollore secondario/ausiliario in supporto all'MCU Wemos D1 mini, in modo tale da non sovraccaricare quest'ultima di troppe operazioni in ogni *loop()*. In particolare lo si è sfruttato per un unico compito, che tuttavia sarebbe potuto risultare bloccante/dannoso se eseguito sulla MCU primaria Wemos insieme a tutte le altre operazioni del sistema: inviare continuativamente l'Id del proprio cubo, al vicinato esterno - ovvero al cubo collegato alla sua faccia ricevitrice (per i dettagli vedere la sezione 2.5.2). L'unico inconveniente nell'utilizzo dell'ATtiny risiede nella mancanza di un bootloader hardware; di conseguenza per caricare il programma che dovrà eseguire è necessario interfacciarlo serialmente con un Arduino (fungerà da *In Serial Programmer*) installando una libreria *Core* di estensione all'IDE di Arduino (e.g. ATtinyCore di

Konde [11]) che include numerosi supporti software alla programmazione degli ATtiny, fra cui un bootloader virtuale chiamato *Optiboot*.

2.4.3 Modulo ricevitore a infrarossi VS1838B e LED trasmettitore

Il primo aspetto considerato in fase di progettazione preliminare è stato il metodo di comunicazione degli Id fra cubi collegati. Come anticipato la comunicazione infra-cubo è fondamentale per ricostruire la topologia del sistema (capire la distribuzione dei e le relazioni fra i cubi) e inoltre influenza il design del prototipo.

La scelta implementativa iniziale del protocollo di comunicazione si è basata sull'utilizzo degli infrarossi; in particolare è stato scelto come componente elettronico il ricevitore IR VS1838B, uno fra i più comuni ricevitori a raggi infrarossi. Tale modulo è provvisto di tre pin (vedi figura 2.20):

1. OUT: il pin che sarà collegato ad uno dei GPIO del microcontrollore scelto
2. GND: il pin collegato a massa
3. VCC: il pin che alimenterà il sensore. Il valore di tensione nominale (vedi datasheet LFN [12]) è di 5 V, ma a livello operativo il range di tensioni accettabili affinchè il ricevitore funzioni correttamente è compreso fra 2.7 e 5.5 V. Ciò è dunque compatibile con la tensione di uscita della batteria 18650 scelta per alimentare il sistema (~3.6 V)

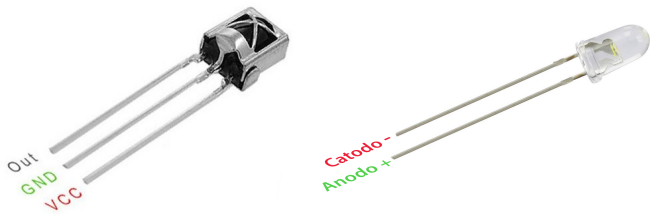


Figura 2.20: Modulo VS1838B e LED trasmettitore

Ogni CUB8 dunque, nella versione 1.0, che prevede parziale componibilità (solo due sono infatti le facce comunicanti) avrà un ricevitore ad infrarossi e un LED bianco (semplice diodo a emissione di luce) trasmettitore del proprio Id, per rispettare il parallelismo descritto nella sezione *Componibilità* 1.2.2. È fondamentale che i ricevitori siano isolati/schermati dai propri LED emettitori in modo tale che rilevino esclusivamente i segnali, e quindi gli Id, dei cubi collegati alle facce esterne.

2.4.4 IMU module: BNO055

Viene definita *Inertial Measurement Unit* (IMU) un dispositivo elettronico che rilevi e misuri l'accelerazione lineare, la velocità angolare di un corpo e talvolta anche il campo magnetico che lo circonda utilizzando una combinazione di rispettivamente uno o più accelerometri, uno o più giroscopi ed eventualmente magnetrometri. Questa combinazione di sistemi inerziali è fondamentale dal momento che l'utilizzo di un singolo accelerometro o singolo giroscopio spesso non è sufficiente a descrivere in modo affidabile l'orientamento, la posizione e la velocità del corpo monitorato. L'integrazione dei sensori invece offre una più accurata rilevazione dei movimenti, arrivando fino a sei gradi di libertà (DOF) con fusione di soli accelerometro e giroscopio (solitamente 3 DOF per il primo e da 1 a 3 per il secondo), e fino a nove DOF se integrato anche il magnetometro.

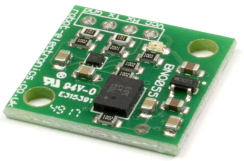


Figura 2.21: IMU Module, by Robot Electronics

Figura 2.22: BNO055 Sensor, by BOSCH Sensortec

Nel caso in esame, volendo tenere traccia ed interpretare il movimento assoluto dei cubi al fine di poter manipolare uno specifico ambiente/oggetto virtuale si è fatto uso proprio di una IMU, in particolare si è scelto di utilizzare il modulo basato sul sensore *BNO055* (della BOSCH Sensortec, vedi figura 2.22) e prodotto dalla *Robot Electronics* (vedi figura 2.21).

2.4.5 Illuminazione tramite LED RGB WS2812

Per l'illuminazione sono stati utilizzati i LED RGB *WS2812* (in figura 2.4.5) più comunemente noti come *Adafruit NeoPixel*. La NeoPixel è una famiglia di strisce, anelli e board di LED RGB 5050 (5 mm x 5 mm) già integrati con un driver chip *WS2811* che facilita il controllo dello stato, della luminosità e del colore di tutte e tre le componenti Red, Green e Blue di ogni LED presente sulla striscia tramite un unico data-pin in input *Din*. In particolare è stata utilizzata una striscia di quattro LED, in modo che ogni faccia libera potesse essere illuminata da uno di questi.

2.4.6 Alimentazione del circuito: batteria 18650 e modulo di ricarica TP4056

Essendo CUB8 un dispositivo che necessita di libertà di movimento per essere propriamente manipolato e unito ad altri suoi pari, viene da sè il bisogno di una fonte di alimentazione a se stante, come una batteria. Come anticipato è stata scelta la batteria ricaricabile agli ioni di litio *18650*, in particolare il modello *US18650VTC6* della Sony (in figura 2.23). Questa ha capacità nominale di 3120 mAh e minima di 3000 mAh, capacità che ci consente di alimentare correttamente il circuito del prototipo. Ha inoltre tensione nominale 3.6 V, tensione che soddisfa il fabbisogno voltaico in ingresso di tutti i componenti elettronici.

Figura 2.23: Battery 18650, by Sony

Ovviamente è stata scelta una batteria ricaricabile, sia per minimizzare l'impatto ambientale ed economico di CUB8, sia per circoscrivere l'apertura del cubetto alla solo manutenzione e/o riparazione dell'elettronica e aggiornamento del meglio software?.

La batteria sarà dunque accompagnata dal modulo di ricarica micro-usb *TP4056* a corrente/tensione costanti (CC/CV) che integra al suo interno un circuito di protezione da sovraccarichi e cortocircuiti (in figura 2.24). In particolare questo circuito di protezione è basato sul chip di controllo di scarica MOSFET *FS8205A* e sul chip di protezione della batteria *DW01A*, da sovraccarichi/sovrascarichi voltaici e sovracorrenti e cortocircuiti. Insieme questi moduli impediscono che la batteria superi il limite minimo di carica 2.4 V, per evitarne il danneggiamento. Quando questo limite viene superato, il modulo interrompe il segnale di output della batteria fino a chè questa non raggiunge almeno i 3 V di tensione. È bene notare che sebbene l'output venga interrotto, il diodo parassita MOSFET del circuito permette comunque la ricarica della batteria connessa. Oltre a situazioni di sovrascarica il modulo gestisce anche la problematica duale, non permettendo la carica della batteria oltre ai 4.2 V.

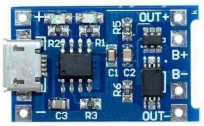


Figura 2.24: TP4056 Module

SPECIFICHE	
Circuito di carica	TP4056
Circuito di protezione	DW01A
Metodo di carica	Corrente/Tensione costante (CC/CV)
Tensione di input	4.5V 6V
Tensione di carica completa (Fluttuante)	4.2V (±1.5%)
Tensione di rilevamento sovraccarica Tensione di rilascio sovraccarica	4.3V (±50mV) 4.1V (±50mV)
Tensione di rilevamento sovra-scarica Tensione di rilascio sovra-scarica	2.4V (±100mV) 3V (±100mV)
Soglia di protezione da sovraccarichi Ritardo di interruzione da sovraccarico	3A 10ms 20ms
Ritardo di interruzione cortocircuiti	5µs 50µs
Soglia di tensione della carica di compensazione Corrente di carica di compensazione	2.9V (±0.1V) 130mA (±10mA)

Tabella 2.2: Specifiche dettagliate del modulo di ricarica, da RobotStore [14]

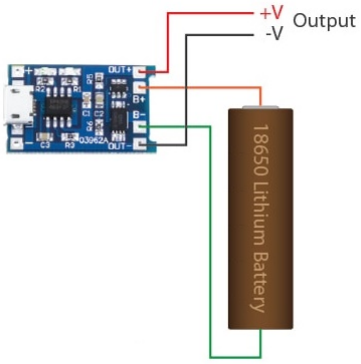


Figura 2.25: Collegamento modulo TP4056-batteria

Il modulo viene collegato alla batteria come rappresentato in figura 2.25, ovvero collegando i pin *B+* e *B-* rispettivamente ai poli positivo e negativo della batteria e i pin *OUT+* al pin *3V3* del microcontrollore Wemos D1 mini e *OUT-* al piano di massa del circuito elettrico.

In fase di design del contenitore esterno di CUB8, naturalmente dovrà essere tenuta in considerazione la fase di ricarica della batteria e quindi dovrà essere prevista una modalità di connessione della micro-usb al modulo TP4056 che limiti al minimo l'apertura del cubo. Nella prima versione realizzata di CUB8, su una delle sei facce è stata creata una fessura in prossimità della quale, internamente, è allocato il connettore femmina micro-usb al quale, dall'esterno e senza dover aprire la struttura, è possibile collegare il cavo.

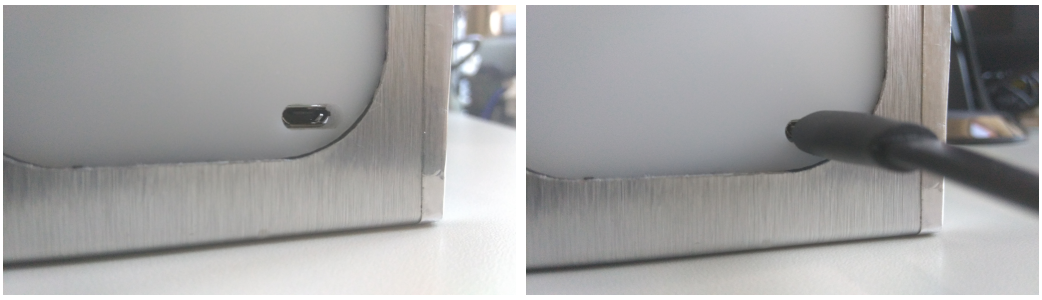


Figura 2.26: Collegamento modulo TP4056-batteria

2.5 Progettazione software versione 1.0

2.5.1 Elaborazione dell'orientamento dei cubi tramite sensore BNO055

Il sensore utilizzato BNO055 è un chip SiP (System in Package) che integra al suo interno un accelerometro triassiale a 14 bit, un giroscopio triassiale a 16-bit (range di ±2000 gradi al secondo), un sensore geomagnetico anch'esso a tre assi e un microprocessore Cortex M0+ a 32-bit grazie al quale il sensore può produrre dati in output in modalità *sensor/data fusion* e gestire in modo efficiente l'alimentazione. La *Fusion Mode* è una modalità di elaborazione dei dati grezzi rilevati dai vari sensori della IMU e permette di misurare l'orientamento del dispositivo nello spazio. È inoltre in grado di discriminare l'orientamento relativo dell'oggetto da quello assoluto. L'orientamento assoluto è dato dall'orientamento del sensore rispetto al campo magnetico della Terra (viene calcolata la direzione del polo nord magnetico). In un sistema non assoluto o relativo, invece, i dati prodotti dal sensore possono variare in relazione alla sua posizione iniziale. Tutte le modalità di data fusion producono come output i dati o in formato quaternione o in angoli di Eulero (*roll*, *pitch* e *yaw*). Il sensore BNO055 fornisce varie modalità operative in Fusion Mode (oltre a quelle Non-Fusion Mode):

Fusion modes	Available Sensor			Fusion Data	
	Acc	Mag	Gyro	Rel	Abs
IMU	x	-	x	x	-
COMPASS	x	x	-	-	x
M4G	x	x		x	-
NDOF_MC_FF	x	x	x	-	x
NDOF	x	x	x	-	x

Tabella 2.3: Operating fusion modes overview

Per quel che concerne il lato software di acquisizione dei dati derivanti dal modulo IMU, è stata utilizzata la libreria *Adafruit_BNO055*, di Adafruit. Questa libreria è parte della libreria Adafruit Unified Sensor Library (Adafruit_Sensor) che fornisce un'interfaccia comune per i sensori supportati e per ognuno di questi restituisce i dati da loro rilevati nelle unità di misura conformi al Sistema Internazionale. Grazie al livello di astrazione tra l'applicativo e l'hardware del sensore, facilita in gran misura l'utilizzo del sensore BNO055:

1. I dati di cui si può usufruire vengono convertiti in modo automatico dalla libreria in unità di misure standard, senza dover dunque tradurre dati grezzi ma potendoli fin da subito interpretare ed elaborare ad alto livello
2. Fornisce strumenti di diagnostica del sensore per:
   (a) Controllare che la auto-calibrazione che questo effettua all'accensione sia andata a buon fine
   (b) Verificarne il corretto funzionamento nell'arco dell'esecuzione dell'applicativo
3. L'astrazione permette inoltre di cambiare il modulo del sensore o il sensore stesso con uno che meglio si adatti alle necessità di sviluppo del sistema (e.g. maggiore sensibilità, minor costo) purché il nuovo sensore sia compatibile con tale libreria

Di default, quando nel software viene inizializzato l'oggetto *Adafruit_BNO055* basato sulla omonima libreria, viene impostata la modalità data fusion *NDOF* (vedi tabella 2.3), modalità che è stata mantenuta nella prototipazione di CUB8.

La modalità *NDOF* è caratterizzata da nove gradi di libertà (DOF) che calcola l'orientamento assoluto del dispositivo combinando i dati rilevati dall'accelerometro, dal giroscopio e sul magnetometro. I vantaggi di combinare tutti e tre i sensori sono un'elaborazione veloce con conseguente elevate frequenza di trasferimento dati in output, e alta resistenza alle distorsioni del campo magnetico.

Comunicazione I2C con l'MCU

Il modulo BNO055-based che è stato utilizzato (in figura 2.21) offre un regolatore di tensione e diverse interfacce di comunicazione, permettendo al dispositivo di essere operativo con qualsiasi valore di tensione in ingresso compreso fra 3.3~5 V. La modalità di comunicazione di default fra slave e host è quella seriale *UART*, ma semplicemente chiudendo circuitalmente il canale *LK1* evidenziato nella figura a sinistra, è possibile utilizzare come protocollo *I2C*, così com'è stato fatto per CUB8.

Figura 2.27: Canale LK1

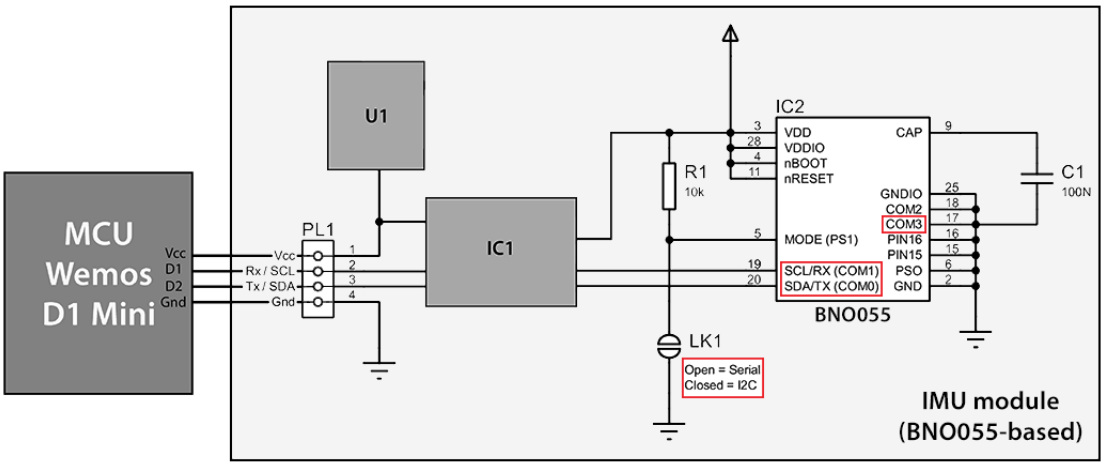


Figura 2.28: Schematico del collegamento con l'mcu Wemos D1 mini

Entrambe le interfacce di comunicazione condividono parzialmente gli stessi pin:

PIN	I2C	UART
COM0	SDA	Tx
COM1	SCL	Rx
COM2	GNDIO	
COM3	I2C address select	

Tabella 2.4: Protocol select pin mapping. Da BOSCH [4]

Per tale motivo esistono i Select Pin *PS0* e *PS1* tramite i quali stati viene selezionata l'interfaccia attiva. In particolare il valore di *PS1* dipende direttamente dal collegamento *LK1*: come anticipato se aperto $PS1 = 1$, mentre se chiuso $PS1 = 0$.

PS1	PS0	Funzionalità
0	0	I2C
0	1	HID-I2C
1	0	UART
1	1	Reserved

Tabella 2.5: Protocol select pin mapping. Da BOSCH [4]

In standard I2C l'indirizzo di default del dispositivo BNO055 è *0x29* o in alternativa *0x28* (discriminato da un valore di tensione *HIGH* o *LOW* sul pin COM3) ed opera fino a 400 kHz di frequenza di clock.

Configurazione I2C	Stato COM3	Indirizzo I2C
Slave	HIGH	0x29
Slave	LOW	0x28
HDI-I2C	X	0x40

Tabella 2.6: I2C address selection. Da BOSCH [4]

I2C funziona secondo il seguente protocollo [4]:

1. START: La trasmissione dei dati sul bus inizia con una transizione da *HIGH* a *LOW* sulla linea *SDA* (dato seriale in input/output) mentre il clock seriale *SCL* è mantenuto *HIGH* (condizione di *Start* imposta dal Master). Una volta che il segnale di START è stato trasmesso dal Master, il bus viene considerato *Busy*

2. STOP: Ogni dato trasmesso deve sempre essere terminato da un segnale di STOP generato dal Master. La condizione di STOP viene rappresentata con un fronte di salita *LOW to HIGH* sulla linea SDA mentre SCL è mantenuto *HIGH*

3. ACK: Ogni byte di dato trasferito deve essere confermato (*Aknowledged*). Ciò avviene tramite l'invio di un ACK bit al mittente da parte del destinatario. *(Durante il segnale di ACK la linea SDA deve essere rilasciata*

dal trasmettitore (no pull down), mentre il ricevente deve abbassare SDA in modo tale da rimanere LOW stabilmente durante il periodo alto del ciclo di clock dell'ACK)

L'azione che ci interessa maggiormente è quella di lettura dai registri del sensore *BNO055* per rilevare l'orientamento del cubo (riferimento alla figura 2.29). Un'azione di lettura consiste in una fase di scrittura I2C di un byte, seguita da una fase di lettura. Queste due parti di trasmissione devono essere separate da una sequenza di start (Sr). Nella fase di scrittura il master indirizza lo slave e gli trasmette gli indirizzi dei registri che devono essere letti dallo stesso. Dopo che lo slave conferma l'avvenuta ricezione degli indirizzi, il master genera un'altra condizione si start e trasmette l'indirizzo dello slave insieme ad un flag di lettura (RW = 1). A questo punto il master rilascia il bus e attende che lo slave legga i byte di dato interessati. Dopo ogni byte il master genera un bit di conferma (ACK = 0) per abilitare nuovi trasferimenti di dati. Un NACKM (ACK = 1) dal master può generare la condizione di STOP e terminare quindi la trasmissione dei dati. L'indirizzo del registro viene automaticamente incrementato e dunque più di un byte può essere letto sequenzialmente. Quando una nuova lettura inizia, l'indirizzo di start verrà settato all'indirizzo del registro specificato nell'ultima scrittura I2C avvenuta. Di default, l'indirizzo di start è impostato a 0x00. Ciò consente di iterare la lettura di byte multipli dallo stesso indirizzo di partenza.

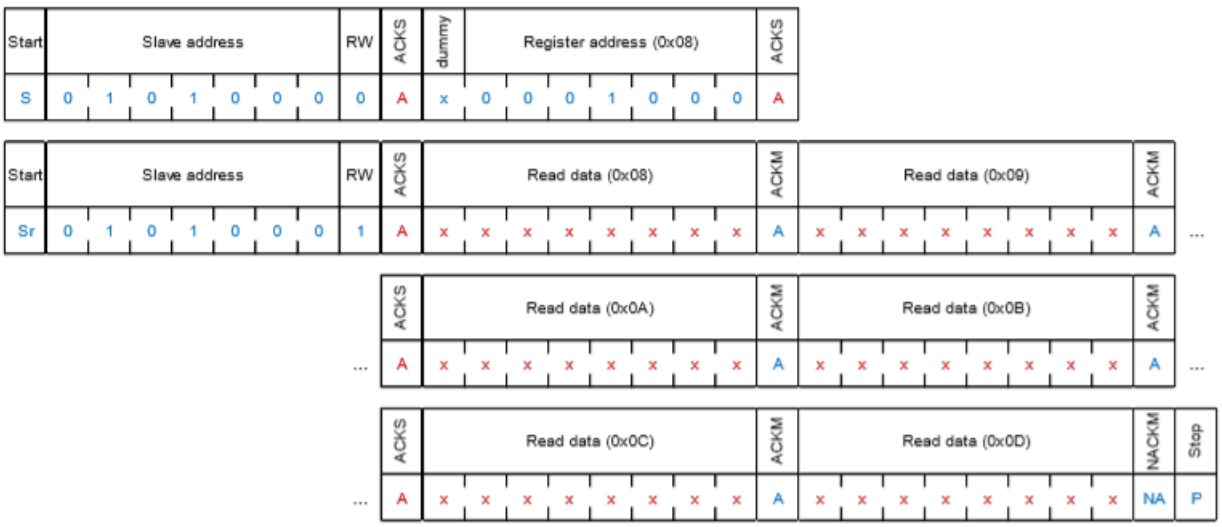


Figura 2.29: Esempio di accesso I2C in lettura multipla. S: start; P: stop; ACKS: ack by slave; ACKM: ack by master; NACKM: Not ack by master; RW: read/write mode (dal datasheet di BOSCH [4])

Formato dati in output: Quaternioni

Come anticipato la modalità fusion-data produce in output dati non grezzi, ma pronti ad essere interpretati, che vengono trasferiti tramite protocollo I2C all'MCU Wemos D1 mini. Tali dati sono prodotti in formato angoli di Eulero (roll, pitch e yaw) o in formato Quaternione. Gli angoli di Eulero vengono usati per rappresentare l'orientamento del sistema di riferimento *X'Y'Z'* (tre assi perpendicolari) di un corpo rigido attraverso la composizione di tre rotazioni a partire da un sistema di riferimento fisso *XYZ* che abbia stessa origine *O* del nuovo sistema di riferimento *X'Y'Z'*. Qualsiasi configurazione di rotazione di un oggetto può essere descritta attraverso la combinazione di queste rotazioni.

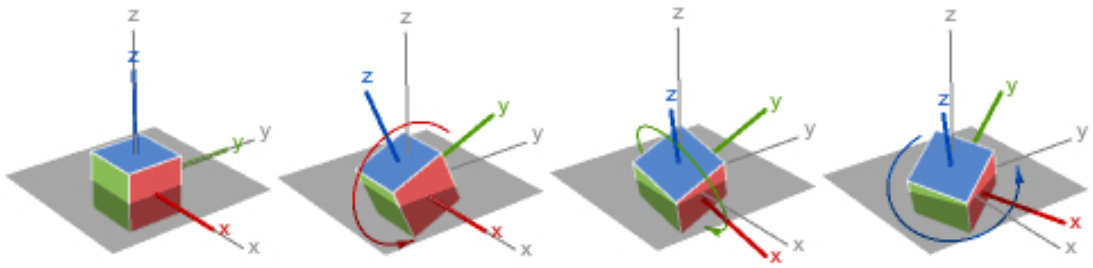


Figura 2.30: Esempio: rotazioni su x-y-z, da *Wolfram.com*

Utilizzando gli angoli di Eulero si rischia tuttavia di incorrere in quello che viene chiamato *blocco cardanico* o *gimbal lock*, situazione in cui è impossibile re-orientare il sistema senza avere un riferimento esterno. Il blocco avviene quando, durante le rotazioni degli assi, due di essi finiscono per descrivere la stessa rotazione facendo quindi perdere un grado di libertà al sistema (da ciò la perdita dell'informazione sull'orientamento).

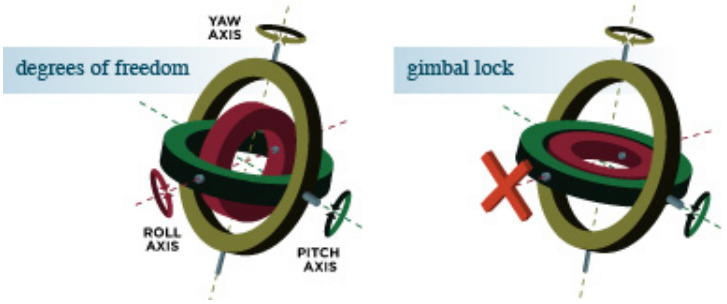


Figura 2.31: Definendo le rotazioni in angoli di Eulero come dei sistemi cardanici con tre cerchi rappresentanti i gradi di libertà del sistema (yaw, pitch e roll), il gimbal lock accade quando due cerchi si allineano e l'intero sistema può muoversi solo in due dimensioni - da *gimbalsystem.com*

L'utilizzo della notazione matematica del *quaternione* per rappresentare l'orientamento e la rotazione di un oggetto tridimensionale, consente di evitare il problema del blocco cardanico, oltre ad essere una rappresentazione più compatta ed efficiente di altre notazioni matematiche. Il quaternione consente di moltiplicare e dividere vettori ruotandoli ed è formato da un numero scalare w e un vettore tridimensionale r:

$$q = (w, r) = (w, x\mathbb{i}, y\mathbb{j}, z\mathbb{k})$$
$$q = w + x\mathbb{i} + y\mathbb{j} + z\mathbb{k}$$

Con w, x, y e z numeri reali e $\mathbb{i}$, $\mathbb{j}$ e $\mathbb{k}$ versori. La direzione tra due qualsiasi punti può essere espressa tramite i tre numeri x, y e z ognuno dei quali è compreso nell'intervallo $[-1,1]$ e complessivamente soddisfano l'equazione

$$\sqrt{x^2 + y^2 + z^2} = 1$$

Collettivamente questi quattro numeri descrivono la rotazione applicata ad un punto tridimensionale formando il quaternione, rappresentato in arancione nella figura riportata a destra. Questo viene applicato al vettore di posizione blu e il risultato di tale applicazione è rappresentato tramite il nuovo vettore di posizione in rosso.

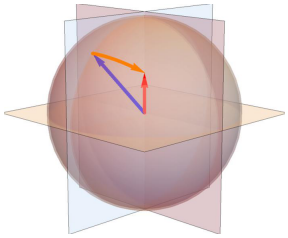


Figura 2.32: Rappresentazione visiva del quaternione

Grazie alla libreria di supporto utilizzata *Adafruit_BNO055* e grazie al fatto che il sensore BNO055 opera data-fusion a monte, è possibile accedere ai dati già trasformati in quaternioni ed elaborarli direttamente per interpretare l'orientamento assoluto del cubo di riferimento. In particolare grazie alla funzione di libreria *imu::Quaternion getQuat(void)* che restituisce i dati nella apposita struttura dati *imu::Quaternion* che salva i valori di w, x, y e z letti dai registri di sistema riservati ai quaternioni:

REGISTER NAME	ADDRESS
QUA_DATA_W_LSB	0X20
QUA_DATA_W_MSB	0X21
QUA_DATA_X_LSB	0X22
QUA_DATA_X_MSB	0X23
QUA_DATA_Y_LSB	0X24
QUA_DATA_Y_MSB	0X25
QUA_DATA_Z_LSB	0X26
QUA_DATA_Z_MSB	0X27

La lettura del quaternione di ogni CUB8 viene fatta all'inizio di ogni loop dello sketch Arduino, con previa azione diagnostica di sistema del sensore per verificare che non si siano riscontrati errori e che i dati oltre ad essere disponibili siano anche corretti. La funzione di controllo del sistema messa a disposizione dalla libreria *Adafruit_BNO055* è la funzione *void getSystemStatus(uint8_t *system_status, uint8_t *self_test_result, uint8_t *system_error)* che si occupa di salvare nelle variabili in ingresso alla funzione, i risultati riguardo lo stato del sistema:

DATA	DESCRIPTION
System Status Code	0 = Idle 1 = System Error 2 = Initializing Peripherals 3 = System Iniitalization 4 = Executing Self-Test 5 = Sensor fusion algorithm running 6 = System running without fusion algorithms
Self Test Results	1 = test passed, 0 = test failed Bit 0 = Accelerometer self test Bit 1 = Magnetometer self test Bit 2 = Gyroscope self test Bit 3 = MCU self test 0x0F = all good!
System Error Code	0 = No error 1 = Peripheral initialization error 2 = System initialization error 3 = Self test result failed 4 = Register map value out of range 5 = Register map address out of range 6 = Register map write error 7 = BNO low power mode not available for selected operat ion mode 8 = Accelerometer power mode not available 9 = Fusion algorithm configuration error A = Sensor configuration error

Tabella 2.7: Tabella di traduzione dei codici di diagnostica. Da BOSCH [4]

Una volta acquisiti i dati riguardanti l'orientamento del cubo, questi verranno poi inviati all'elaboratore centrale - che potrà così interpretarli ed agire conseguentemente - tramite lo scambio di pacchetti UDP sulla rete Wi-Fi alla quale entrambi dovranno essere connessi (per i dettagli sul protocollo di comunicazione in rete del sistema in esame, vedere la sezione 2.5.4).

2.5.2 Protocollo di comunicazione IR degli Id tra cubi connessi

Encodig PWM e Decoding dell'ID

Il modulo VS1838B - il quale scopo è quello di ricevere gli Id dei cubi esternamente connessi alle proprie facce - riesce a decodificare segnali con frequenza portante pari a 38 kHz, motivo per cui in fase di trasmissione dell'Id si è dovuto modulare in PWM (Pulse Width Modulation) la luce di un semplice LED a tale frequenza. Modulare un segnale in PWM significa agire sulla variazione della larghezza d'impulso al fine di ottenere una tensione media che varia in relazione al rapporto tra la durata dell'impulso positivo τ e il periodo totale $T = \frac{1}{f_0}$, trasportando quindi l'informazione discreta/binaria tramite un segnale pulsato. Tale rapporto è chiamato *duty cycle* δ o ciclo utile. Il valore medio V_{avg} della tensione è controllato agendo sulla durata dell'alimentazione data al carico e quindi accendendo e spegnendo a velocità elevata il LED ed è direttamente proporzionale al duty cycle: $V_{avg} = V\frac{\tau}{T} = V\delta$, con V tensione di linea disponibile all'ingresso del carico.

Encoding

Ogni cubo deve inviare il proprio Id in modo continuativo. Il ciclo di invio viene ripetuto ogni ~54 ms e ha durata variabile in base alla quantità di "uni" e "zeri" nel codice binario da trasmettere. L'Id (e.g. *C1*), verrà convertito in codice binario e trasmesso bit a bit per un totale di 12 bit - 8 bit di dato e 4 bit ridondanti per il controllo d'errore - secondo le seguenti regole:

1. Il **periodo** di riferimento T, dovendo modulare un segnale con frequenza portante $f_0 = 38\,\text{kHz}$ sarà: $T = \frac{1}{f_0} = 26\,\mu\text{s}$

2. Bit di **start**: la trasmissione inizia con 128 impulsi positivi (livello alto del segnale) ognuno da $26\,\mu$s per un intervallo totale di $3328\,\mu$s seguiti da 32 periodi di pausa ($832\,\mu$s)

3. Bit di **dato**: la trasmissione di ogni bit logico viene effettuata suddividendo l'intervallo di trasmissione in due parti, la prima *HIGH*, tenendo quindi alto il segnale, e la seconda *LOW*, ovvero spegnendo il LED emettitore. L'aspetto discriminante fra bit logico a '1' e bit a '0' è la durata dell'intervallo in cui il segnale è alto:

   (a) Bit a '**1**': la trasmissione di un 1 logico avviene tenendo acceso il LED per 1664 μs (64 impulsi ognuno di 26 μs) e spegnendolo per 832 μs (32 periodi di inattività)

   (b) Bit a '**0**': la trasmissione di uno 0 logico avviene invece tenendo acceso il LED per 832 μs (32 impulsi ognuno di 26 μs) e spegnendolo per altrettanto tempo

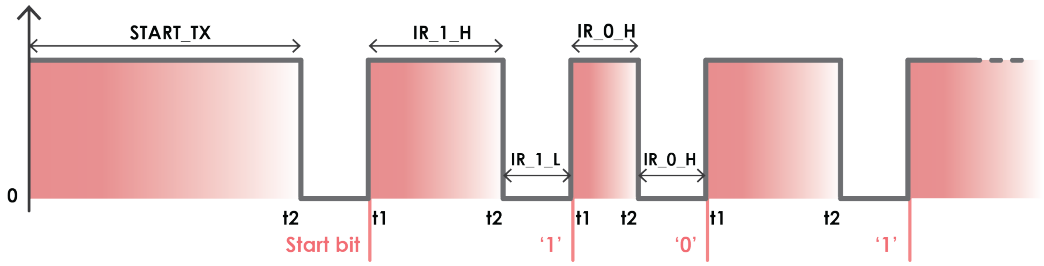


Figura 2.33: Segnale trasmesso modulando la luce di un LED

Listing 2.1: Definizione durate in microsecondi degli impulsi

```
1 const int START_TX = 3328; // Bit di Start
2
3 const int IR_1_H = 1664; // Parte alta di un 1
4 const int IR_1_L = 832; // Parte bassa di un 1
5
6 const int IR_0_H = 832; // Parte alta di uno 0
7 const int IR_0_L = 832; // Parte bassa di uno 0
8
9 const int E = 100; // Margine di errore nella lettura
```

Decoding

Non potendo prevedere il momento esatto in cui avverrà la ricezione degli Id esterni - evento dipendente dalla volontà dell'utente che collegherà due cubi - si è deciso di sfruttare l'asincronicità degli *Interrupt*.

Gli interrupt sono eventi asincroni speciali, causati anche da stimoli esterni non sempre prevedibili, ai quali il microcontrollore deve reagire entro un tempo prestabilito. Quando avviene un interrupt, l'esecuzione del loop principale si interrompe e viene eseguita una funzione di routine detta *Interrupt Handler* o *ISR* (Interrupt Service Routine) che gestirà propriamente l'evento. Una volta terminata si ritornerà all'esecuzione del programma principale, lì dove ci si era interrotti.

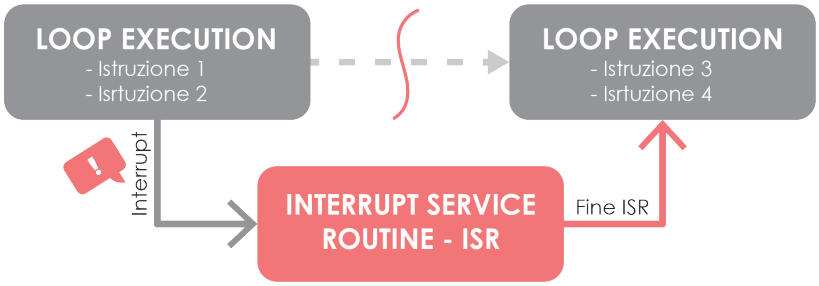


Figura 2.34: Schema funzionamento Interrupt

Gli interrupt possono essere generati internamente o esternamente. Nel nostro caso avranno natura esogena: saranno dunque provocati da eventi esterni al microcontrollore. Ciò è molto utile perchè consente di ricevere i dati dai ricevitori IR solo quando realmente necessario, evitando di fare *polling*. L'evento scatenante gli interrupt è il cambiamento di valore in un pin digitale di I/O del microcontrollore. Nel microcontrollore Wemos D1 mini, tutti i pin di GPIO possono essere configurati come pin di interrupt, fatta eccezione per GPIO16 (D0) (sezione 2.4.1). Ogni pin utilizzabile può essere configurato secondo tre modalità di interrupt:

1. CHANGE: l'interrupt viene scatenato quando il pin cambia il valore in uscita da *HIGH* a *LOW*, o viceversa
2. RISING: l'interrupt viene scatenato quando il segnale passa da *LOW* a *HIGH*
3. FALLING: è la modalità duale di RISING, l'interrupt viene scatenato quando il segnale passa da *HIGH* a *LOW*

Entrando nel merito della ISR scritta per la decodifica degli Id, si è andati a tenere traccia dei *cambiamenti* del segnale ricevuto sul pin OUT del modulo VS1838B collegato al pin D7 del microcontrollore, posto sulla base di CUB8. Si è scelto di tenere traccia dei cambiamenti del segnale in modalità *CHANGE*, quindi ad ogni fronte di salita e di discesa (*rising-or-falling-edges*). In particolare il modulo VS1838B produce come output un segnale *LOW* nel momento in cui

rileva la presenza della luce IR del LED. Di conseguenza si avrà un'inversione di tensione: un valore di tensione *HIGH* trasmesso equivarrà ad un valore di tensione *LOW* in fase di rilevamento.

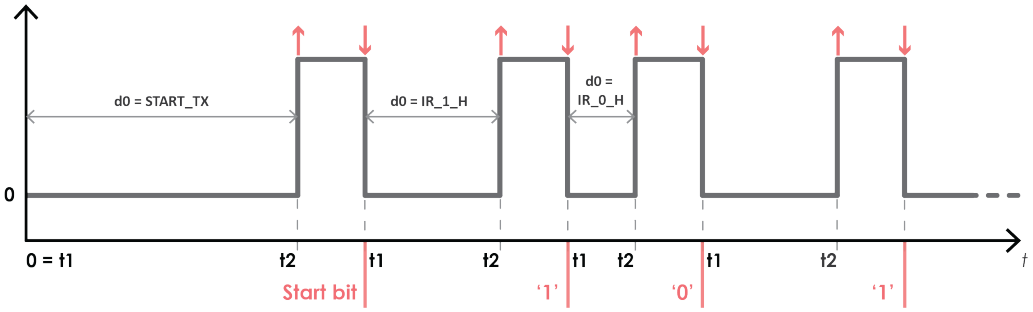


Figura 2.35: Segnale rilevato sul pin OUT del modulo VS1838B

Per poter rilevare correttamente l'Id dall'esterno, si necessita di cinque variabili globali:

1. *t1_fallingEdge*: rappresenta l'istante di ricezione del fronte HIGH ⇒ LOW, precedente a *t2_risingEdge*

2. *t2_risingEdge*: rappresenta l'istante di ricezione del fronte LOW ⇒ HIGH calcolato con *micros()*, funzione che conta quanti microsecondi sono passati dall'avvio del programma (tempo corrente)

3. *count*: rappresenta il numero di bit letti fino al momento corrente. Essendo i bit trasmessi 12 per ogni trasmissione, una volta raggiunto tale stato viene resettato a -1 (stato di blocco alla decodifica, non essendo in presenza di bit di start)

4. *dataPin*: variabile in cui viene ricostruito bit a bit il dato ricevuto. La decodifica effettiva parte solo ed esclusivamente se in presenza dei bit di start, in qual caso la variabile *count* viene settata a 0 (stato di inizio della decodifica)

5. *receivedCubeID*: variabile in cui viene salvato il valore ricevuto una volta finita la decodifica. Rappresenta l'Id del cubo collegato magneticamente alla faccia su cui opera il ricevitore IR. Verrà poi inviata all'elaboratore centrale, insieme ad altre informazioni di sorta, per ricreare la topologia del sistema

Un altro parametro di cui si deve tenere traccia è la durata dell'ultimo intervallo temporale registrato (*d0*), dato dalla differenza fra tempo corrente *t2_risingEdge* e fronte di discesa precedente *t1_fallingEdge* ovvero:

$$d0 = t2\_risingEdge - t1\_fallingEdge$$

Tale differenza viene calcolata solo quando si è in presenza di un fronte di salita: se tale condizione è verificata significa che è appena terminato un intervallo a zero. È quindi possibile calcolarne la lunghezza *d0* ed in base ad essa discriminare se si è in presenza di bit di start, di un '1' logico o di uno '0' logico:

1. se $d0 \geq (START\_TX - E)$ si è in presenza dei bit di start. Si possono dunque inizializzare le variabili *dataPin* e *count* a 0 ed uscire dalla ISR, in attesa del prossimo cambiamento di fronte e quindi di dati interpretabili.

2. In qualsiasi altro caso, finchè la variabile *count* non viene inizializzata a 0, si deve semplicemente uscire dalla funzione e ignorare/non decodificare qualsiasi valore venga rilevato sul pin, in attesa dei bit di start.

3. se invece $(count > -1) \wedge [(IR\_1\_H - E) \leq d0 \leq (START\_TX - E)]$ si è in presenza di un bit a '1'. I bit in *dataPin* vengono quindi "shiftati" a sinistra di un posto; tale posto viene dapprima riempito con uno '0', dall'operatore *shift left* (<<) ed in seguito, a questo viene sommato un '1'

4. se infine $(count > -1) \wedge [(IR\_0\_H - E) \leq d0 < (IR\_1\_H - E)]$ è stato ricevuto uno '0'. L'unica operazione da effettuare in questo caso è quella di *shift* che muove i bit in dataPin verso sinistra di un posto ed inserisce uno '0' nel posto appena liberato.

Quando *count* raggiunge lo stato 12 significa che l'ultimo bit è ormai stato letto e il dato ricostruito è stato memorizzato in *dataPin*. A questo punto si inizializza *count* a *-1* (stato in cui ci si limita ad attendere la ricezione dei prossimi bit di start) e si verifica che il dato rilevato sia valido, tramite la funzione *validate(dataPin)*. Questa implementa la codifica a correzione d'errore di Hamming.

Robustezza del protocollo: codifica di Hamming

In fase di trasmissione d'informazione fra sistemi elettronici può accadere che ciò che viene rilevato non sia la rappresentazione esatta del segnale trasmesso, ma una sua distorsione. Tale distorsione è solitamente causata dal rumore introdotto per natura dal canale di comunicazione. Nel nostro caso il canale di comunicazione è rappresentato da null'altro se non dal mezzo in cui si propagano le onde a infrarosso dei LED che stanno inviando il proprio ID: l'aria e l'ambiente circostante.

In particolare una possibile causa di distorsione può essere dovuta ad interferenze solari, che possono generare sporadiche pulsazioni non desiderate nel

segnale di output del modulo ricevitore, come dimostrato empiricamente nell'esperimento effettuato da AnalysisIR [1]. Tuttavia, nel caso in esame, essendo il ricevitore VS1838B all'interno del cubo, risulta propriamente schermato da questo tipo di interferenze.

Un tipo di rumore del quale invece è potenzialmente a rischio, è il rumore dato dall'alimentazione elettrica e dagli apparati elettronici stessi coinvolti nella trasmissione. Per tale motivo risulta necessario implementare una tecnica di controllo e/o correzione d'errore all'interno del protocollo.

La codifica scelta - come anticipato - è la codifica di Hamming, un codice lineare che sfrutta la ridondanza dei bit nella code-word (blocco codificato) trasmessa per la rilevazione e/o correzione d'errore. Per codificare M simboli distinti con un codice binario occorrono $m \geq \log_2 M$ bit. Un codice si dice ridondante, quando codifica gli M simboli con r bit in più rispetto agli m bit strettamente richiesti dalla codifica binaria. Esistono due tipi di codici ridondanti:

1. A rivelazione d'errore, possono individuare la presenza di uno o più bit errati.

2. A correzione d'errore, possono correggere il bit errato una volta individuata la sua posizione all'interno della code-word

La capacità di rilevare e/o correggere eventuali errori è correlata alla distanza di Hamming del codice scelto: per rivelare N errori si necessita di un codice con distanza di Hamming minima di $(N + 1)$ tra le configurazioni valide, mentre per la correzione si necessita di un codice con distanza minima di $(2N + 1)$. La distanza di Hamming fra due code-word $\in M$ è data dal numero di bit diversi nelle corrispondenti posizioni.

Verrà utilizzato per CUB8 un codice di Hamming a correzione di errore su singolo bit, in grado cioè di correggere un solo bit errato. Dunque per ogni messaggio da trasmettere di m bit e per ogni numero di bit ridondanti $r \geq 2$, esiste una parola codificata di lunghezza $n = m + r$. Ognuna delle 2^m parole legali, ha n code-word errate a distanza di Hamming 1, ottenute cambiando un bit alla volta nella parola originaria e ognuna richiede $(n + 1)$ configurazioni di bit dedicate. Per cui varrà la disuguaglianza:

$$\begin{gathered} 2^n \geq 2^m(n+1) \\ 2^m 2^r \geq 2^m(m+r+1) \\ 2^r \geq (m+r+1) \end{gathered}$$

Gli Id dei cubi hanno per convenzione il formato "C" seguito da un numero progressivo in base alla quantità dei cubi utilizzati. Tale Id è rappresentabile in binario con 8 bit di messaggio ($m = 8$), ai quali ne sono stati aggiunti 4 ulteriori ($r = 4$) per rispettare la disuguaglianza:

$$2^r \geq (m + r + 1)$$
$$\Rightarrow 2^4 \geq (8 + 4 + 1) \Rightarrow 16 \geq 13)$$

È facilmente verificabile che con $r = 3$ tale disuguaglianza non è soddisfatta. Si è dunque ottenuto un codice *Hamming(n,m) = Hamming(12,8)*, con ridondanza $r = 4$.

Invio degli Id ricevuti

In situazione di "riposo", il singolo CUB8 è isolato. In questa particolare condizione dunque, il suo modulo ricevitore ad infrarossi non rileverà alcun tipo di informazione esterna e la variabile *receivedCubeID* ha valore 0. Ogni qualvolta l'utente colleghi magneticamente due CUB8 - in particolare nella versione 1.0, questa composizione avverrà ponendoli uno sopra o sotto l'altro - il sensore VS1838B del cubo soprastante rileverà un cambiamento di fronte del segnale sul suo pin di output. Questo cambiamento avvia l'esecuzione della ISR precedentemente descritta che andrà a ricostruire bit a bit l'Id del cubo sottostante. Il CUB8 che sta ricevendo l'Id, avrà così preso consapevolezza del suo vicinato e potrà dunque inviare tale informazione all'elaboratore centrale via comunicazione UDP (vedi sezione 2.5.4). Per rendere questo invio valido e reattivo ai cambiamenti nella topologia del sistema, la variabile *receivedCubeID* viene azzerata ad ogni invio e ricostruita bit a bit solo se necessario (ovvero se è presente un altro cubo): in questo modo se l'utente decide di staccare due cubi collegati o ne collega uno nuovo il sistema potrà istantaneamente avere l'informazione più aggiornata e corretta.

2.5.3 Controllo dei LED RGB WS2812

A livello software, per pilotare i LED RGB WS2812 è stata utilizzata la libreria di Arduino *FastLED* (Garcia [8]), compatibile con molti tipi di board, fra cui quelle basate sul microcontrollore ESP8266. La libreria agevola la programmazione delle LED strip in modo efficiente, integrando al suo interno moltissimi chipset per i LED più utilizzati nel mondo del making. Il chipset di interesse nel caso in esame è quello 3-wire (collegamento della striscia LED tramite 3 cavi: massa, alimentazione e dati) che permette l'utilizzo dei LED scelti, più compatto rispetto ai chipset basati su SPI aventi un cavo di collegamento aggiuntivo riservato per il clock. La libreria FastLED fornisce strutture dati e funzioni di supporto per l'inizializzazione e l'utilizzo dei LED RGB. In particolare viene utilizzata una struttura dati di tipo *CRGB* sulla quale verrà dichiarato un array *leds* con dimensione il numero di LED della striscia da utilizzare (*NUM_LEDS = 4*). In seguito, grazie alla funzione *FastLED.addLeds<CHIPSET, PIN>(leds,*

NUM_LEDS) si andrà ad inizializzare tale vettore dichiarando il tipo di chipset da utilizzare (*WS2812*) e il numero del pin di GPIO del microcontrollore al quale è collegato il data pin della striscia (*D4*). Configurato il set-up iniziale, è possibile modificare i singoli valori RGB che controlleranno il colore e la luminosità dei LED. Esistono molti metodi per effettuare tale operazione; per minimizzare i tempi di risposta della striscia si è deciso di accedere direttamente alle allocazioni in memoria di questi valori, evitando l'uso di funzioni di libreria: *leds[i][0] ⇒ Green; leds[i][1] ⇒ Red; leds[i][2] ⇒ Blue.*

Nell'applicativo sviluppato per la versione 1.0 di CUB8, il colore di cui i vari cubi si illuminano è la discriminante per il loro ruolo all'interno dell'ambiente virtuale di "gioco". Inizialmente, quando ancora l'utente non ha avviato la simulazione, tutti i cubi brillano dello stesso colore (configurato nella funzione *setup()* dello sketch), trasmettendo visivamente neutralità ed equivalenza. Nel procedere all'utilizzo dell'applicativo, l'utente sceglie di volta in volta un CUB8 e gli assegna un determinato ruolo, ruolo che l'elaboratore centrale si impegna a distinguere visivamente cambiando il colore del cubo in questione. Tale aggiornamento di colore e quindi di stato, in questo stadio di sviluppo del prototipo, viene implementato tramite lo scambio di pacchetti UDP fra l'elaboratore centrale e il microcontrollore ESP8266, contenenti le informazioni delle componenti RGB con cui illuminare i LED.

2.5.4 Protocollo di comunicazione su rete Wi-Fi del sistema CUB8

Come spiegato precedentemente CUB8 è un dispositivo modulare da utilizzare in parallelo con altri suoi pari. Il sistema nel suo insieme prevede l'utilizzo di una piattaforma di elaborazione centrale che coordinerà e interpreterà la topologia di tale sistema ripercuotendo le interazioni reali effettuate dall'utente nel modo virtuale dell'applicativo. Per realizzare quest'opera di coordinamento, si necessita di una modalità di comunicazione fra i cubi utilizzati dall'utente e l'elaboratore centrale. Tale modalità è stata implementata sfruttando le reti Wi-Fi; da qui la scelta di una board programmabile che contenesse a monte gli strumenti necessari per un'appropriata connessione (board *ESP8266EX-based*). Sono state dunque utilizzate le librerie:

- *ESP8266WiFi library* (Grokhotkov [9], capitolo 5) fornita dal Core di Arduino per i moduli ESP8266 che permette loro di connettersi ad una determinata rete Wi-Fi e la gestione della connessione stessa

- *WiFiUDP* per lo scambio di pacchetti UDP con altri dispositivi connessi alla medesima rete

Fase di discovering delle rete Wi-Fi e connessione

I CUB8, nella progettazione della versione 1.0, si connettono in modalità *Station* ad un Access Point Wi-Fi (vedi figura 2.13) dichiarandolo con la funzione di libreria *WiFi.mode(WIFI_STA)*. La procedura base per effettuare la connessione alla rete di interesse avviene tramite l'utilizzo della funzione *WiFi.begin("network-name", "pass-to-network")* sostituendo opportunamente i due argomenti con l'SSID e la passphrase effettivi della rete, seguita da un ciclo di attesa che termina nel momento in cui lo stato - dato da *WiFi.status()* - della struttura *WiFi* non risulta *CONNECTED*. Tale procedura standard, è stata rivisitata in modo che i vari CUB8 potessero avere in memoria un set di reti alle quali tentare l'accesso per rendere il sistema nel suo insieme dinamico e adattativo alle diverse situazioni.

In particolare ciò è stato effettuato creando un partizione *SPIFFS* nella memoria Flash della board Wemos D1 Mini delle dimensioni di 1 MB su 4 MB totali. SPIFFS (*SPI Flash Filing System*) permette di dedicare una parte della memoria dei sistemi embedded (di default dedicata al codice eseguibile) ad un sistema di *filing*, per lo storage di file contenenti dati che raramente necessitano di essere modificati (pagine web, file di configurazione, calibrazione dei sensori, etc). Ciò permette di non sovraccaricare la RAM del microcontrollore, solitamente caratterizzata da dimensioni molto ridotte. I file del file system SPIFFS sono poi accessibili sia in lettura che scrittura attraverso API Posix-like per l'elaborazione di testi come *open, close, read, write, seek* invocabili nello sketch eseguibile. SPIFFS è stato dunque utilizzato per memorizzare un file *networks.txt* contenente la lista di SSID e passphrase relative a reti Wi-Fi note e alle quali i CUB8 si dovevano connettere più frequentemente. Tale file viene poi letto riga per riga in fase di *setup()* nello sketch, andando ad inserire i valori letti in una struttura dati *Network* (creata ad hoc) costituita da SSID e password, la quale a sua volta viene inserita in un collettore *std::list<Network> networks* di oggetti di tipo *Network*. In fase di connessione Wi-Fi, CUB8 scorre tale lista effettuando un confronto fra gli SSID e password della rete corrente con quelli delle reti disponibili in quel dato luogo e momento. In caso di confronto fallito passa a confrontare la rete successiva della lista *networks*; se il confronto invece risulta andato a buon fine, stabilizza la connessione all'Access Point individuato.

In questo stadio i LED RGB WS2812 sono stati progettati per accendersi ad intermittenza di colore blu per dare un feedback visivo immediato di ciò che sta accadendo, dal momento che in caso d'uso reale di CUB8 non sarebbe possibile acceder al monitor seriale per il debug.

Comunicazione UDP

Una volta stabilita la connessione Wi-Fi, viene inizializzata una struttura dati di tipo *WiFiUDP* memorizzata in una variabile chiamata *UDP*, atta a gestire lo scambio di pacchetti attraverso il protocollo omonimo *User Datagram Protocol* sulla porta scelta. Inizialmente si è scelto questo tipo di comunicazione vista l'ingente quantità di pacchetti da scambiare: ad ogni *loop()* (con un delay di 90 ms funzionale all'acquisizione di nuovi dati dalla IMU) infatti viene inviato un pacchetto UDP in broadcast sulla porta stabilita che verrà poi filtrato dall'elaboratore centrale. Si è quindi preferito implementare una comunicazione efficiente, piuttosto che affidabile (come ad esempio una comunicazione *TCP*, affidabile, ma più dispendiosa in termini di risorse).

Ogni pacchetto Datagram uscente da CUB8 è composto secondo il seguente formato:

$$messageUDP = \text{“}cube\_id,\ qW,\ qX,\ qY,\ qZ,\ receivedCubeID\text{”}$$

con

1. *cube_id*: valore dell'ID del cubo che sta inviando il datagramma
2. *qW, qX, qY, qZ*: i valori del quaternione letto grazie alla funzione *imu::Quaternion bno.getQuat()* e rispettivamente reperiti accedendo alle funzioni della libreria *utility/imumaths.h* ⇒ *quaternion.w(), quaternion.x(), quaternion.y(), quaternion.z()*
3. *receivedCubeID*: il valore dell'Id rilevato dal modulo IR VS1838B. In caso nessun cubo sia collegato al CUB8 di riferimento, questo valore varrà 0

L'elaboratore centrale, in ascolto sulla porta concordata, leggerà poi i pacchetti ogni qualvolta ne riceva e aggiornerà l'ambiente virtuale di testing (creato con il software *Processing*) coerentemente con quanto letto. In particolare grazie alle informazioni ricevute potrà:

1. Sapere quali sono i CUB8 attivi grazie all'indicazione dell'Id trasmessa da *cube_id*
2. Interpretare ed aggiornare l'orientamento della rappresentazione virtuale del cubo avente come Id *cube_id* grazie ai valori *w, x, y* e *z* del quaternione. Elaborando tali dati ruoterà e/o sposterà l'oggetto virtuale (legato al tipo di applicazione) relativo al cubo di riferimento
3. Identificare le relazioni di composizione in atto tra i vari cubi attivi e ricreare più o meno implicitamente la topologia dell'ambiente virtuale. Se

per esempio il CUB8 con identificativo “C0C” trasmette come *receivedCubeID* “C3F” significa che questo cubo è posto al di sotto del cubo “C0C”. Questa relazione fra i due cubi produrrà un determinato tipo di reazione nell’applicativo sulla base del contesto di utilizzo.

Un altro tipo di informazione che l’elaboratore centrale può acquisire indirettamente, essendo contenuta nell’header del pacchetto UDP, è l’indirizzo IP assegnato ai cubi connessi alla rete Wi-Fi utilizzata. Sarà dunque in grado di rispondere ai diretti interessati inviando loro direttive circa colore e luminosità con cui pilotare i loro LED RGB, in base all’interpretazione elaborata sul quaternione ricevuto e/o altre discriminanti specifiche dell’ambiente virtuale in esecuzione. Il pacchetto UDP di risposta, inviato ovviamente sulla stessa porta, conterrà esclusivamente i tre byte relativi ai valori Green, Red e Blue con cui illuminare i LED. Questo, una volta ricevuto dal CUB8 interessato, verrà *parsato* e i valori letti inseriti direttamente nella corrispettiva area di allocazione dedicata ai LED, cambiandone lo stato.

2.5.5 Applicazione sul campo: Game’s Week 2018 (MI)

Abbiamo avuto la fortuna di poter testare sul campo la prima versione di CUB8 alla Game’s Week, tenutasi a Milano dal 5 al 7 ottobre 2018. In tale sede abbiamo potuto constatare pregi e difetti della prima versione del prototipo, informazioni fondamentali per poter procedere alle versioni future e sempre più definitive di CUB8.

Figura 2.36: Esposizioni dei CUB8 presso lo stand di *Tonic Minds* alla Game’s Week di Milano

Per l'occasione sono state portate due applicazioni demo, ideate per avere come target bambini compresi fra i 3 e gli 8 anni circa, ed utilizzate in fase di testing dei sette prototipi creati.

La prima, più semplice, prevede l'utilizzo di un singolo cubo ed è pensata per agevolare la comunicazione dello stato d'animo del *bambino utente* ad un adulto di riferimento. CUB8, in questo contesto, vuole quindi porsi come strumento di supporto nel caso in cui il bambino abbia difficoltà ad esprimere se stesso e i propri sentimenti (e.g. terapie di superamento di traumi, più o meno gravi forme di autismo o comportamenti di chiusura in generale). L'applicazione prevede uno scenario molto semplice, che vede come protagonista un ragazzino in stile *cartoon* inquadrato a mezzo busto. Il focus della scena è quindi sul volto e sulle sue espressioni. Cambiando la faccia su cui poggia il CUB8 utilizzato come controller dal bambino, cambieranno anche le espressioni del protagonista facendolo passare da felice a triste, a disgustato, ad arrabbiato o a sorpreso. Se invece si ruota il cubo in senso orario o antiorario attorno all'asse verticale, il ragazzino creato virtualmente muoverà gli occhi seguendo la rotazione del cubo.

La seconda applicazione presentata è invece una sorta di "teatrino virtuale", in cui il bambino è lasciato libero di esplorare le varie ambientazioni e personalizzare la scena a suo piacimento. Avrà a disposizione cinque cubi con i quali controllare l'ambiente: inizialmente i CUB8 sono illuminati dello stesso colore e sono allineati su una board di plexiglass trasparente creata ad hoc per l'occasione (vedi immagine 2.36). La board ha cinque "postazioni" magnetiche che tengono in posizione i cubi in stato di riposo. Quando l'utente-bambino interagisce con i cubi può staccarli dalla loro posizione e manipolarli. Ogni cubo ha un ruolo all'interno del teatrino: un cubo controlla la scelta dell'ambientazione, un altro rappresenta un oggetto di scena, due cubi impersonano i due protagonisti (un ragazzino e una ragazzina) e l'ultimo CUB8 è il "configuratore", ovvero un tool di personalizzazione dell'abbigliamento dei protagonisti. Sono stati creati diversi scenari ognuno corredato da un oggetto di scena che ben si adatta all'ambientazione e dal "corretto" abbigliamento dei protagonisti. Nonostante la possibilità di creare scene realisticamente coerenti, al bambino è lasciata la totale libertà di creare nuove combinazioni mischiando i vari oggetti e i look dei protagonisti. In questo contesto CUB8 si pone come strumento di storytelling interattivo.

Nel complesso abbiamo potuto notare che i CUB8 attiravano molto l'attenzione di adulti e di bambini in visita alla Game's Week sebbene questi ultimi fossero quasi tutti lievemente al di sopra della fascia d'età d'interesse del target. Questo superamento di soglia si è fatto sentire maggiormente durante l'utilizzo dell'applicativo: esso infatti risultava tendenzialmente troppo semplice e troppo

poco interattivo. Tuttavia, un risultato di questo tipo era atteso; al giorno d'oggi i bambini iniziano ad essere immersi e bombardati da videogiochi e tecnologia fin da piccoli. Ciò ha comportato un innalzamento della soglia di capacità allo stupore e di aspettativa nei confronti di questo mondo, specialmente all'interno di un contesto denso di innovazione in campo video-ludico come la Game's Week. Il fatto però che i CUB8 suscitassero curiosità e interesse lascia ben sperare e pone buoni presupposti per la continuazione del progetto.

2.5.6 Considerazioni tecniche e migliorie apportabili al progetto

A livello di considerazioni tecniche ponderate post esperienza durante la Game's Week, si è potuto verificare che la durata della carica della batteria è risultata ampiamente soddisfacente, considerato lo stato di accensione dei cubi e la loro attività costante durante l'arco di tutta la giornata (circa dalle ore 9:30 alle 18:30). Un aspetto più problematico è stata la connessione alla rete Wi-Fi privata del router portato per la presentazione delle varie applicazioni e la comunicazione UDP dei CUB8 con il server centrale. In fase di connessione non tutti e sette i cubi sono riusciti a connettersi al primo tentativo, ma è stato necessario aprirne il contenitore e resettarli manualmente più volte. Ciò è probabilmente stato dovuto al fatto che all'interno del padiglione della mostra erano presenti centinaia di reti Wi-Fi con conseguente ingente traffico di pacchetti di ogni sorta. La stessa comunicazione UDP, non si è rivelata robusta in tale contesto applicativo reale e caotico, come tuttavia si poteva prevedere. Queste ed altre considerazioni avvenute nel periodo di sviluppo e testing in laboratorio hanno portato a prendere in considerazione nuove strategie implementative per le versioni future di CUB8, parte delle quali sono già state effettuate o in corso d'opera:

1. Modifica del protocollo di comunicazione in rete fra i cubi e il server centrale: si è deciso di migrare da una messaggistica senza conferma di ricezione come UDP ad una più affidibile e robusta come *MQTT* (protocollo molto usato per device nell'IoT).

2. Aggiungere la possibilità di resettare i CUB8 dall'esterno, senza dover per forza aprire il contenitore.

3. Ottenere componibilità totale dei moduli-cubi, ovvero consentire il collegamento magnetico dei cubi su tutte e sei le facce. Questo consentirebbe molta più libertà di espressione nella User Experience di CUB8, facendo aumentare esponenzialmente il numero delle combinazioni effettuabili e quindi permettendo la creazione di applicativi interessanti e complessi.

4. Ricerca di nuovi metodi di illuminazione dei cubi: sebbene l'effetto ottenuto sia molto vicino all'illuminazione diffusa che si stava ricercando, il tipo di soluzione adottata per la versione 1.0 di CUB8 è praticabile solo se si prevede una componibilità limitata su due facce parallele del cubo (come spiegato più in dettaglio nella sotto-sezione 2.2.3). Tuttavia, come espresso nel punto 3, ciò che si vuole ottenere per la versione finale di CUB8 è componibilità totale. Di conseguenza si dovranno adottare nuove soluzioni per illuminare i cubi.

5. Modifica del protocollo di comunicazione infra-cubo: ciò che è stato scoperto empiricamente è che - paradossalmente - i ricevitori IR VS1838B rilevano fin troppo i segnali inviati dai LED trasmettitori dell'Id dei cubi circostanti. Ci si è resi conto che se per esempio due cubi sono spazialmente molto vicini e poggiano uno sulla faccia contenente il modulo VS1838B (rilevatrice dell'Id) e l'altro sulla faccia opposta (trasmettitrice dell'Id), il cubo con la faccia rilevatrice rivolta verso il basso rischia di ricevere l'Id del cubo affianco, *sebbene non siano effettivamente composti magneticamente*. Si è quindi pensato di adottare come nuova soluzione la rilevazione degli Id esterni tramite *tag* e lettore *RFID* che, se calibrate nel giusto modo le distanze, possono permettere la comunicazione solo se due cubi sono "a contatto".

Parte III

Capitolo terzo

Capitolo 3

Evoluzione del progetto

La realizzazione della versione iniziale 1.0 di CUB8 e la possibilità di testarlo in un contesto reale, hanno fatto sì che si arrivasse ad una ridefinizione nel design del progetto. È importante notare che lo sviluppo della versione definitiva è attualmente ancora in corso d'opera, di conseguenza rimarranno alcuni punti irrisolti, divenendo argomento di sviluppi futuri.

3.1 Componenti elettronici selezionati

La componentistica elettronica adottata nella prima versione dei cubi interattivi è rimasta pressapoco invariata, fatta eccezione per i componenti relativi alla comunicazione infra-cubo, ovvero i moduli ricevitori IR VS1838B, i diodi a emissione di luce per la trasmissione dell'Id e il microcontrollore ATtiny che li pilotava. Questi sono stati rimpiazzati con un sistema di lettura *contact-less* basato su RFID pilotati dal microncontrollore Arduino Mini collegato in seriale con la board Wemos D1 Mini. Le restanti parti adottate nella versione 1.0, quali l'MCU Wemos, il modulo di ricarica TP4056 con annessa batteria 18650 e il modulo IMU basato sul sensore BNO055 sono state mantenute.

3.1.1 Modulo di lettura RFID PN532 V3

Il primo punto sul quale si è voluto lavorare, è la componibilità su tutte e sei le facce dei CUB8. Questo, unito alle problematiche riscontrate con l'utilizzo dei moduli VS1838B in combinazione con i LED emettitori dell'Id, ha portato ad adottare un nuovo sistema di comunicazione infra-cubo. È stata scelta la tecnologia a RFID (*Radio Frequency Identification*) che sfrutta i campi elettromagnetici permettendo l'identificazione automatica di componenti chiamati *tag* o *etichette* posti sugli oggetti di cui si vuole tenere traccia. Le frequenze radio,

portando energia al tag ne trasferiscono l'informazione identificativa contenuta all'interno al lettore RFID ed infine al microcontrollore che potrà elaborare tale informazione. Passando senza problemi attraverso la maggior parte dei materiali, come la plastica o il legno, lettori e tag possono essere incorporati all'interno dei cubotti realizzando una comunicazione contact-less, senza che lo spessore delle pareti interferisca con essa. È fondamentale tuttavia che la distanza di lettura sia uguale a tale spessore, per consentire il trasferimento dell'identificativo da un cubo all'altro solo se questi sono effettivamente collegati magneticamente e non solo se si trovano spazialmente vicini.

Il *reader* selezionato è il modulo *PN532 NFC RFID* V3, prodotto dalla *Elechouse*, basato appunto sul circuito integrato PN532 della *NXP*. Questo modulo supporta la lettura e scrittura di vari tipi di RFID (come ad esempio le card Mifare 1k, 4k, Ultralight e DesFire o le carte ISO/IEC 14443-4 e altri standard nel settore), comunicazione *P2P* peer-to-peer fra reader e la recentemente molto popolare tecnologia NFC per smartphone Android. Supporta comunicazione seriale con un dato microcontrollore in I2C, in SPI e in HSU (High Speed UART). Nel particolare caso in esame, essendo i pin di GPIO della board Wemos D1 Mini dedicati al protocollo I2C impegnati nella comunicazione con la IMU BNO055, è stata selezionata l'interfaccia SPI per lo scambio dei dati con il modulo PN532. Tale interfaccia è stata mantenuta anche in seguito all'introduzione del microcontrollore secondario Arduino Mini, introdotto per alleggerire il carico computazionale sulla scheda Wemos ed effettuare la lettura sui reader RFID al posto suo. L'interfaccia di default impostata dal modulo RFID è HSU, ma si può facilmente modificare tale configurazione semplicemente alternando lo stato di due piccoli interruttori posizionati sul modulo secondo lo schema seguente:

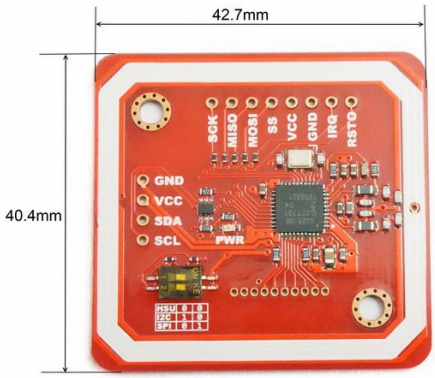


Figura 3.1: Modulo PN532 NFC RFID, by Elechouse

Interfaccia	Switch 1	Switch 2
HSU	OFF	OFF
I2C	ON	OFF
SPI	OFF	ON

Tabella 3.1: Switch settings

Ogni CUB8 avrà dunque tre lettori RFID PN532, posti internamente su tre facce adiacenti/ortogonali del cubo. Questo minimizzerà le possibilità di interferenze durante le letture sulle facce in quanto la lettura dei tag avviene solo se questi sono posti parallelamente ai reader: di conseguenza se i tre lettori si trovano ortogonali gli uni agli altri la possibilità di rilevare un tag posto su uno degli altri lettori RFID è bassa. Sulle restanti tre facce complementari, verranno poi posti i tag contenenti l'informazione dell'Id relativa al cubo nel quale sono contenuti, informazione che verrà rilevata dai lettori RFID dei cubi esterni, una volta che questi ultimi verranno collegati magneticamente dall'utente. Il parallelismo facce magnetiche Nord trasmettitrici dell'Id e Sud rilevatrici rimane soddisfatto (per i dettagli vedere la sezione 1.2.2).

3.1.2 Microcontrollore ausiliario Arduino Pro Mini

Come anticipato è stato utilizzato un microcontrollore secondario, *Arduino Pro Mini*, adibito alla sola lettura degli RFID, per evitare rallentamenti nel *loop()* dello sketch caricato sulla MCU principale Wemos. Tuttavia, gli Id letti dai reader dovranno poi essere inviati all'elaboratore centrale così che questo possa ricostruire la topologia dell'ambiente virtuale. Di conseguenza è necessario che tali Id vengano trasmessi all'MCU principale, l'unico componente di CUB8 in grado di comunicare tramite Wi-Fi.

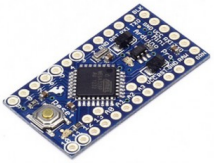
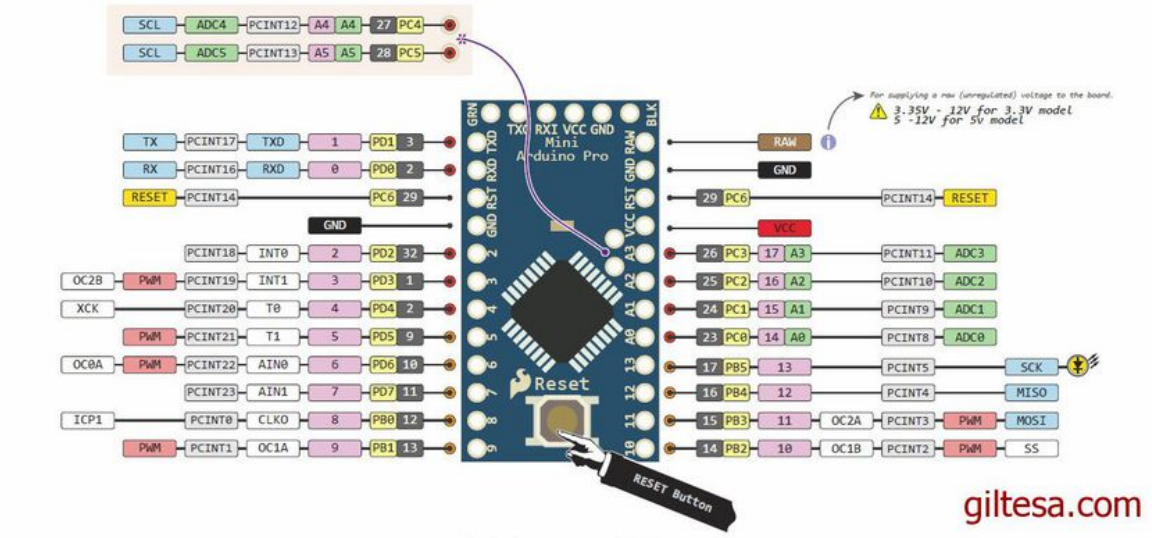


Figura 3.2: Arduino Pro Mini, by SparkFun Electronics



Arduino Pro Mini fornisce una serie di agevolazioni per la comunicazione con un computer, un secondo Arduino o altri microcontrollori. Il pin *TX0* di Arduino Mini è stato dunque collegato al pin *RX* della board Wemos. Essendo una comunicazione unilaterale (le informazioni fluiscono esclusivamente dal pin *TX0* al pin *RX*), il monitor seriale per fare debug sulla porta della Wemos è rimasto disponibile in fase di sviluppo, dal momento che il suo pin *TX* era libero.

I lettori RFID sono tutti collegati in parallelo agli stessi pin di Arduino Pro Mini, in particolare:

Pin PN532 per SPI	Pin Arduino Mini Pro
Clock - SCK	11
Master In, Slave Out - MSO	12
Master Out, Slave In - MOSI	13

Tabella 3.2: Collegamento fra i pin SPI del modulo PN532 e i pin di Arduino Pro Mini

I pin di SS (Slave Select) sono invece dedicati ai singoli lettori e sono collegati ai pin *A1*(15), *A2*(16) e *A3*(17) di Arduino. Ogniqualvolta dunque, l'MCU Wemos rilevi dati in ingresso sulla sua porta seriale li acquisisce e conseguentemente li trasmette via Wi-Fi al server centrale.

3.2 Evoluzione della progettazione software

3.2.1 Acquisizione degli Id esterni tramite lettura RFID

Per l'acquisizione dei dati sui lettori RFID collegati ad Arduino Pro Mini è stata utilizzata la libreria di Adafruit *Adafruit_PN532*, compatibile con il tipo di microprocessore dell'MCU utilizzato, ATmega328P. Una volta definita la corrispondenza fra i vari pin dell'interfaccia SPI e relativi Slave Select, sono state create tre variabili di tipo *Adafruit_PN532*, ognuna con SS diverso. Dopo un preliminare controllo in fase di *setup()* che le periferiche siano funzionanti, si può passare alla fase di lettura sugli RFID, effettuata una volta al secondo nel *loop()*. La lettura consiste in due passaggi:

1. Per prima cosa viene invocata la funzione *readPassiveTargetID(uint8_t cardbaudrate, uint8_t * uid, uint8_t * uidLength, uint16_t timeout)* la quale restituisce un boolean che indica se il reader sta effettivamente rilevando un tag all'interno del suo raggio di azione.

2. Se il *bool* restituito dopo il punto 1 risulta vero, si può passare alla lettura del contenuto del tag rilevato. Ogni card Mifare Classic 1K (tag su cui si sono fatti i test) è composta da sedici macro-settori numerati da 0 a 15 ognuno a sua volta suddiviso in quattro blocchi da 16 byte. Il settore 0 è il *Manifacturer Block* e contiene dati riguardanti la manifattura del circuito integrato e il numero seriale UID (Unique Identifier). Il settore che invece contiene l'informazione di nostro interesse si trova nel blocco quattro, (ovvero il primo blocco del settore 1) di conseguenza si può limitare la lettura a tale blocco. Questo blocco può essere sovrascritto con le informazioni d'interesse, nel caso specifico con l'Id del cubo e in seguito letto dal reader RFID.

Ognuna di queste fasi appena descritte deve essere ripetuta per ogni modulo PN532 contenuto nel cubo. Una volta effettuata la lettura per tutti e tre i moduli si può passare alla composizione del messaggio da inviare all'MCU principale Wemos tramite comunicazione seriale. Il messaggio sarà composto secondo il seguente formato:

$$message = \text{“}Id1/Id2/Id3\text{”}$$

con *Id1*, *Id2*, *Id3* gli Id letti sulle facce di CUB8. Nel caso in cui non sia stato rilevato alcun cubo su una delle facce, al posto del valore dell'Id viene inserito il carattere *X* ad indicare che in quella determinata faccia non è collegato alcun cubo.

3.2.2 Protocollo di comunicazione su rete Wi-Fi del sistema CUB8

Una delle modifiche che si desiderava apportare riguardava il protocollo di comunicazione in rete dei CUB8 con l'elaboratore centrale. Si è dunque migrati da un'architettura UDP ad una più affidabile MQTT (*Message Queuing Telemetry Protocol*).

MQTT è un protocollo di messaggistica *light-weight* ideale per la comunicazione M2M (Machine to Machine); sta infatti diventando lo standard nel mondo dell'IoT, avendo un impatto a livello di risorse molto basso e ben adattandosi a sistemi che prevedono la presenza di device e banda limitati, alta latenza o reti poco affidabili. Lo scopo di MQTT è proprio quello di garantire affidabilità e garanzia di ricezione anche all'interno di contesti operativi difficili. È un protocollo ISO standard posto al di sopra del livello TCP/IP ed è basato sul principio del *Publish/Subscribe*. Il design principle Publish/Subscribe fornisce un'alternativa all'architettura classica Client/Server, nel quale è prevista comunicazione diretta fra i due (o più) attori coinvolti. Nel pattern pub/sub avviene invece un disaccoppiamento fra chi pubblica i messaggi (Publisher) e chi li riceve (i Subscribers). Questi non sono mai in contatto diretto, ma la loro comunicazione è gestita da un terzo componente, il Broker. Il Broker filtra tutti i *messaggi* trasmessi dai Client e li smista ai corrispettivi sottoscrittori. I vari Client possono pubblicare i propri messaggi all'interno di *topic* creati ad hoc, ovvero aree d'interesse dedicate a un particolare tipo di informazione, alle quali i Client possono iscriversi (*Subscribe*) se interessati a ricevere aggiornamenti su quel particolare "argomento", e/o in cui pubblicare messaggi (*Publish*) a riguardo. I topic sono rappresentati da stringhe UTF-8 separate da *slash* "/", i quali individuano il livello del topic (simile alla gerarchia di sub-directories di un file system).

Nel sistema prototipale CUB8, il Broker e il Client MQTT che dovrà interpretare i dati inviati dai cubi, risiedono sullo stesso elaboratore centrale. Il Broker installato è *Mosquitto*, un broker open-source che implementa le versioni 3.1 e 3.1.1 di MQTT. Il Client MQTT sul quale sarà eseguito l'applicativo con cui utilizzare e testare i cubi, allo stato attuale è scritto in *NodeJS*, per il quale è stato installato il pacchetto *npm* relativo appunto a MQTT [13]. Perchè l'insieme del sistema (gli N CUB8 attivi, il Broker e il Client MQTT) sia in grado di comunicare e quindi essere operativo, tutti gli attori coinvolti dovranno essere connessi sulla medesima rete Wi-Fi.

Nel particolare caso in esame, il protocollo comunicativo nel suo complesso implementa una "macchina a stati" come rappresentato nello schema generale in figura 3.3.

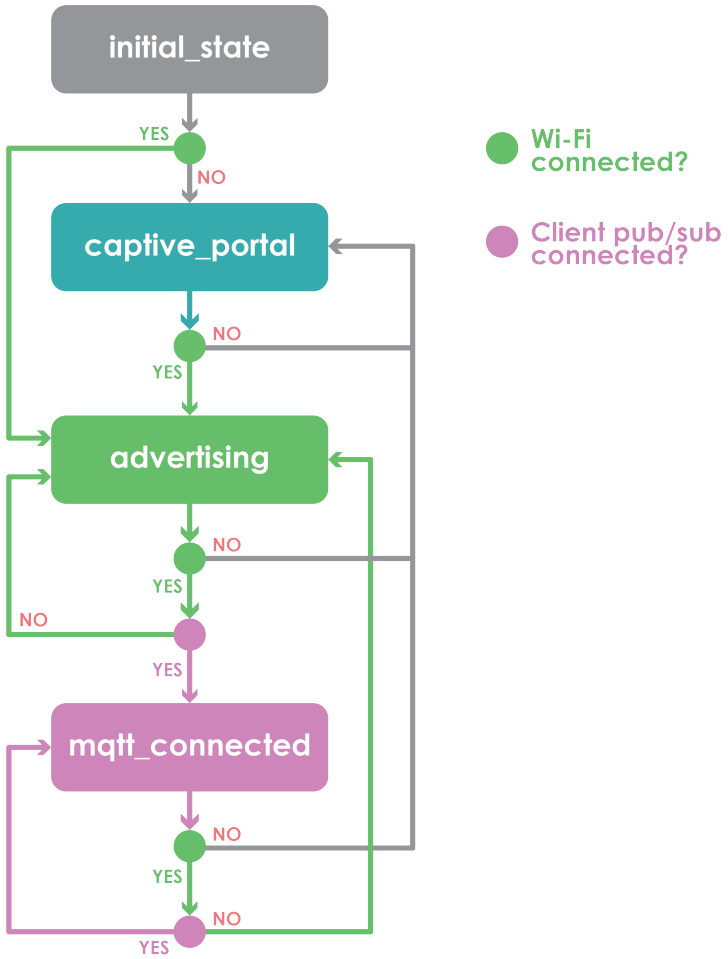


Figura 3.3: Macchina a stati del protocollo in rete di CUB8

Fase Captive Portal

CUB8, inizialmente assume lo stato *initial_state*, stato nel quale viene fatta la lettura delle reti note contenute nel file *networks.txt* sulla partizione SPIFFS nella memoria Flash dell'MCU principale (dettagli in sotto-sezione 2.5.4). Queste vengono inserite in una lista contenente "oggetti" di tipo *Network*. A questo punto si entra nella funzione di *loop()* dello sketch nella quale verrà ciclata la lista *networks* per tentare la connessione con le reti Wi-Fi disponibili in quel dato momento e luogo. Se nessuna rete nota soddisfa la condizione di uguaglianza fra SSID e password delle reti disponibili si passa allo stato *captive_portal*, stato in cui il microcontrollore ESP8266 entrerà in modalità *Soft-AP*, fornendo un punto di accesso in rete attraverso l'indirizzo privato *"http://cub8.local/"*. A tale indirizzo si troverà il Captive Portal contenente il form tramite il quale sarà possibile inserire SSID e password della nuova rete alla quale vogliamo

far accedere CUB8. Se le credenziali fornite risultano corrette, verranno quindi memorizzate nel file *networks.txt*, in modo tale da potervi accedere in modo automatico, senza aver bisogno di entrare nel portale e re-inserire i dati nel form ad ogni utilizzo. In questo stadio l'illuminazione dei CUB8 sarà blu, per avere un feedback visivo in fase di testing.

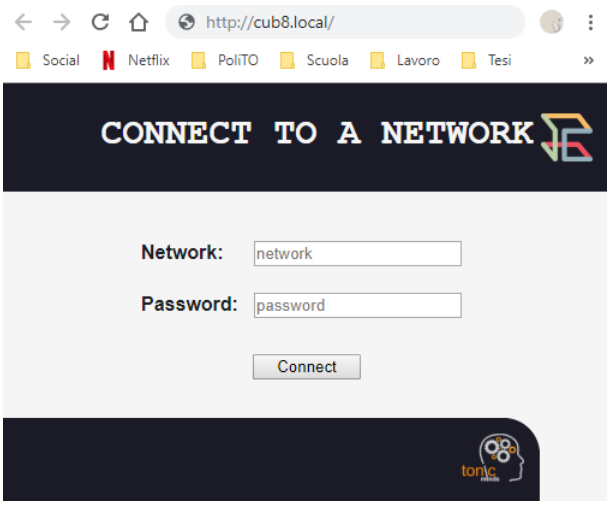


Figura 3.4: Pagina web del captive portal di CUB8

Fase Advertising

Nel momento in cui CUB8 riesce ad effettuare l'accesso alla rete Wi-Fi d'interesse, entrerà nello stato di *advertising*. A questo punto dunque, i cubi opereranno in modalità *Station* e passeranno ad illuminarsi da luce blu a verde. Per potersi connettere al Broker Mosquitto, CUB8 necessita dell'indirizzo IP della macchina su cui questo sta venendo eseguito e la porta scelta. Per rendere il sistema più dinamico e adattativo possibile si è deciso di implementare una breve fase di *advertise/discover* UDP, grazie alla quale il Client MQTT sull'elaboratore prenderà coscienza dei cubi attivi che stanno richiedendo accesso al Broker e gli fornirà le informazioni di cui hanno bisogno. In tale stato di advertising, viene dunque aperta una porta UDP, utilizzata del cubo per mandare (una volta al secondo) un datagramma in broadcast con il formato "message = cub8|cube_id" per segnalare la propria presenza e stato di attività. Quando il Client nodejs riceve questo pacchetto, risponderà all'indirizzo IP del mittente con il proprio IP (risiedendo il Broker sulla sua stessa macchina hanno lo stesso indirizzo) e la porta (1883). Sul fronte CUB8, una volta ricevuto e *parsato* il pacchetto UDP di risposta, si tenterà di stabilire la connessione MQTT all'indirizzo IP ricevuto.

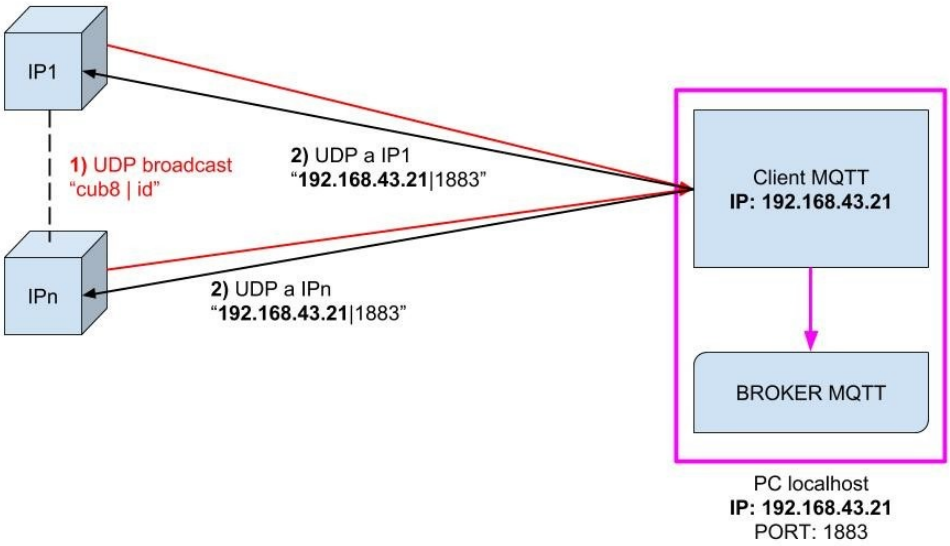


Figura 3.5: Schema di massima della fase di *advertising*

Fase MQTT connected

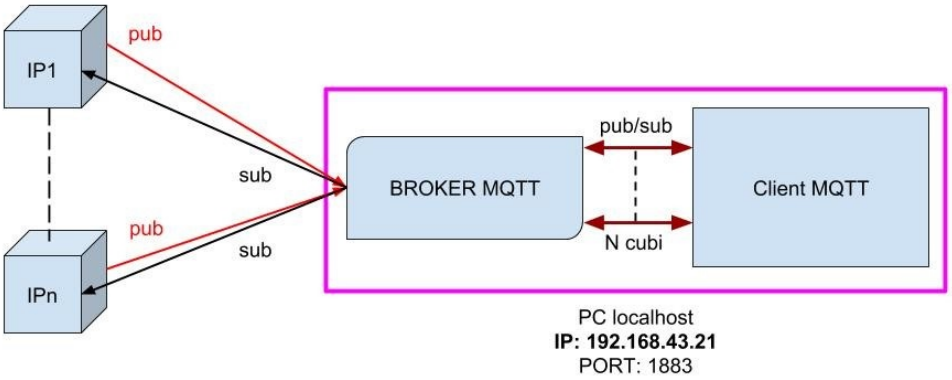


Figura 3.6: Schema di massima della fase *mqtt_connected*

Una volta che la fase di Advertising è andata a buon fine e sono dunque stati ricevuti l'indirizzo IP del Server MQTT e la porta di riferimento sulla quale avverrà lo scambio dei messaggi fra i CUB8 e l'elaboratore, si passa a stabilire la connessione MQTT all'indirizzo fornito. Per implementare tale architettura è stata utilizzata la libreria *PubSubClient* (Imroy [10]). Tramite la funzione

bool connect(String id, String willTopic, uint8_t willQos, bool willRetain, String willMessage) si prova a stabilire la connessione MQTT, previa configurazione di server e porta, e della funzione di callback che verrà invocata ogni qualvolta il cubo riceva messaggi sui topic ai quali è iscritto (tramite le funzioni *setServer()* e *setCallback()*). Se il risultato della chiamata a funzione *connect()* è positivo si passa dunque dallo stato di *Advertising* allo stato *MQTT_connected* della macchina stati, avanzamento di stato che viene evidenziato visivamente passando da un'illuminazione verde ad una violetta (agevolazione in fase di debug e testing).

Nel caso in esame i topic sono sfruttati per creare uno specifico sistema gerarchico proprio di ogni CUB8, che vede alla radice la stringa relativa all'Id del cubo che sta pubblicando il messaggio circa il suo stato, ovvero il proprio orientamento corrente (grazie alla IMU), e l'identificativo degli eventuali cubi collegati magneticamente alle sue facce (grazie agli RFID). I topic sui quali i vari CUB8 pubblicano i messaggi sono strutturai nel seguente modo:

- *<cube_ID>/rfid* ⇒ topic reltivo ai dati rilevati sui sensori RIFD del cubo con specifico *cube_ID*. I messaggi pubblicati su tale topic hanno formato "ID1|ID2|ID3" secondo la convenzione spiegata nella sotto-sezione 3.2.1.

- *<cube_ID>/imu* ⇒ topic relativo all'orientamento della IMU del cubo con specifico *cube_ID*. I messaggi pubblicati su tale topic hanno formato "qW|qX|qY|qZ", valori relativi al quaternione corrente letto dalla IMU.

L'elaboratore centrale, per ricevere i dati aggiornati pubblicati su questi topic dovrà ovviamente essere iscritto agli stessi. Ciò viene effettuato grazie all'indicazione dell'Id dei cubi attivi nel sistema ricevuta tramite il pacchetto UDP nella fase di *advertising*, quindi ogni qualvolta rilevi la presenza di un nuovo CUB8. Una volta interpretate le informazioni su orientamento e vicinato dei cubi attivi, l'elaboratore centrale pubblica a sua volta messaggi di risposta relativi a colore e luminosità con cui il cubo iscritto a questo specifico topic dovrà illuminarsi come conseguenza dei dati inviati. Il topic su cui l'elaboratore pubblica e a cui il CUB8 deve essere iscritto è il seguente: *<cube_ID>/led* ⇒ topic reltivo ai LED del cubo con specifico *cube_ID*. Il formato dei messaggi scambiati su questo topic è "G:=[0-255]|R:=[0-255]|B:=[0-255]" con GRB i valori delle componenti rispettivamente verdi, rosse e blu con cui il cubo con <cube_ID> deve illuminarsi.

3.3 Evoluzione della progettazione hardware

NOTA BENE: A differenza delle modifiche effettuate al software, le modifiche hardware sono ancora in fase di sviluppo e prototipazione, di conseguenza ciò che verrà esposto in questa sezione resta ancora legato al piano della teoria e sarà obbiettivo di sviluppi futuri prossimi in contesto differente rispetto al progetto universitario prettamente relativo alla presente Tesi.

3.3.1 Schema circuitale aggiornato e produzione del circuito stampato

Conseguentemente alle modifiche apportate al set di componenti elettronici utilizzati si è dovuto ridefinire lo schematico del circuito elettrico, così come la PCB del circuito stampato. Inoltre volendo adottare la soluzione di CUB8 a componibilità sulla totalità delle sei facce, si è dovuto rivedere anche la progettazione del contenitore.

L'introduzione dei lettori RFID in particolare, ha stravolto il modo di pensare alle soluzioni della progettazione hardware: si passa da una situazione iniziale che vede i moduli VS1838B e i LED emettitori dell'Id posizionati perpendicolarmente alle facce del cubo e, date le dimensioni ridotte, tutti contenuti sulla stessa PCB, ad una situazione totalmente complementare. Ora infatti i reader devono per forza di causa maggiore essere posizionati parallelamente alle facce del cubo e con loro anche la PCB alla quale saranno connessi. Avere tre reader RFID ~40 mmx43 mm che necessitano di essere paralleli a tre facce significa dunque disporre di altrettante PCB poste ortogonalmente le une alle altre, formanti quindi una sorta di "mezzo cubo". Questo mezzo cubo, oltre agli RFID conterrà anche i restanti componenti elettronici utilizzati. L'altra metà del cubo invece, dovrà essere creata, oltre che per dare robustezza alla struttura di PCB, anche per poter sostenere i tag passivi che trasmettono l'Id del cubo - ognuno al centro di ogni faccia - ma non conterrà componenti elettronici.

Si è dunque deciso di costruire un cubo formato dalle sole PCB che andrà poi a porsi all'interno del contenitore vero e proprio di CUB8. Il problema da risolvere in seguito a tale decisione risiedeva nel *come* far fluire l'elettricità da una PCB all'altra per permettere al circuito di funzionare correttamente, essendo queste tre parti disgiunte e per di più ortogonali. La soluzione è stata trovata nell'utilizzo di connettori per PCB perpendicolari maschio posti sulle estremità di alcune facce che andranno a collegarsi a connettori per PCB femmina presenti sulle estremità delle facce con cui si devono unire. Unendo le facce in questo modo, viene così composto il cubo di PCB, permettendo inoltre il passaggio della corrente elettrica da una scheda all'altra tramite i pin connettori.

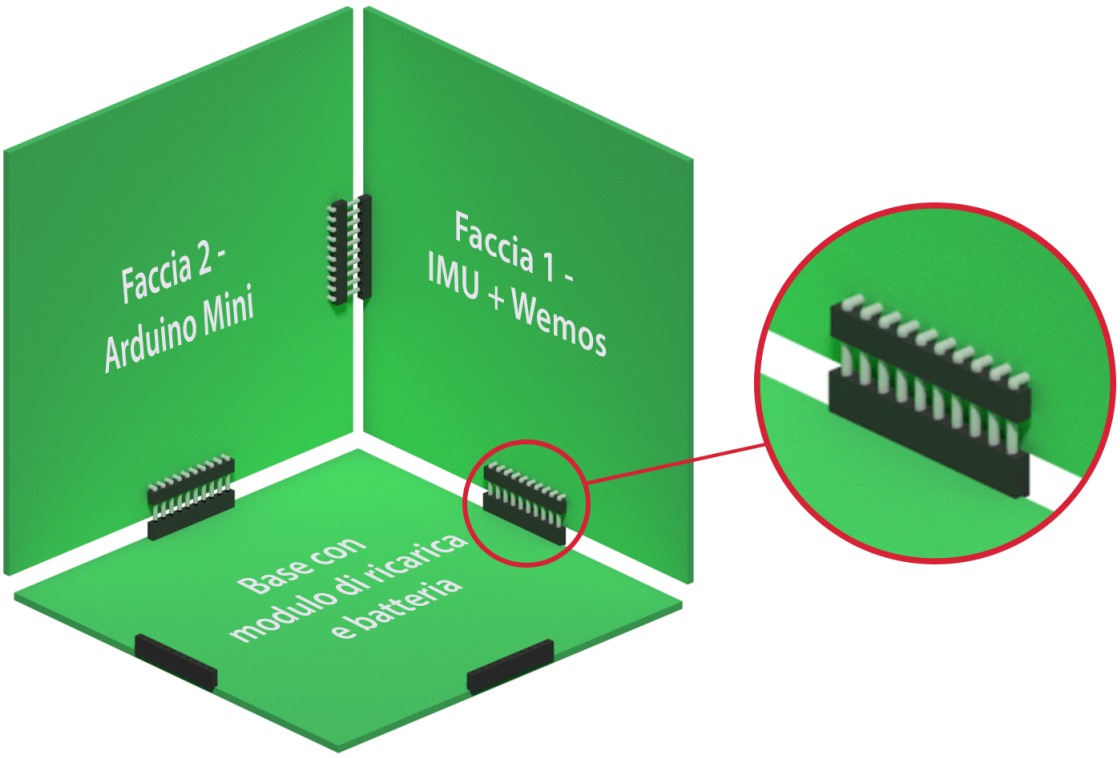


Figura 3.7: Render realizzato con *Blender*

Il circuito elettronico di CUB8 sarà dunque formato da tre sotto-circuiti, ognuno legato ad una delle tre facce perpendicolari costituenti il "mezzo cubo" (vedi figura 3.8).

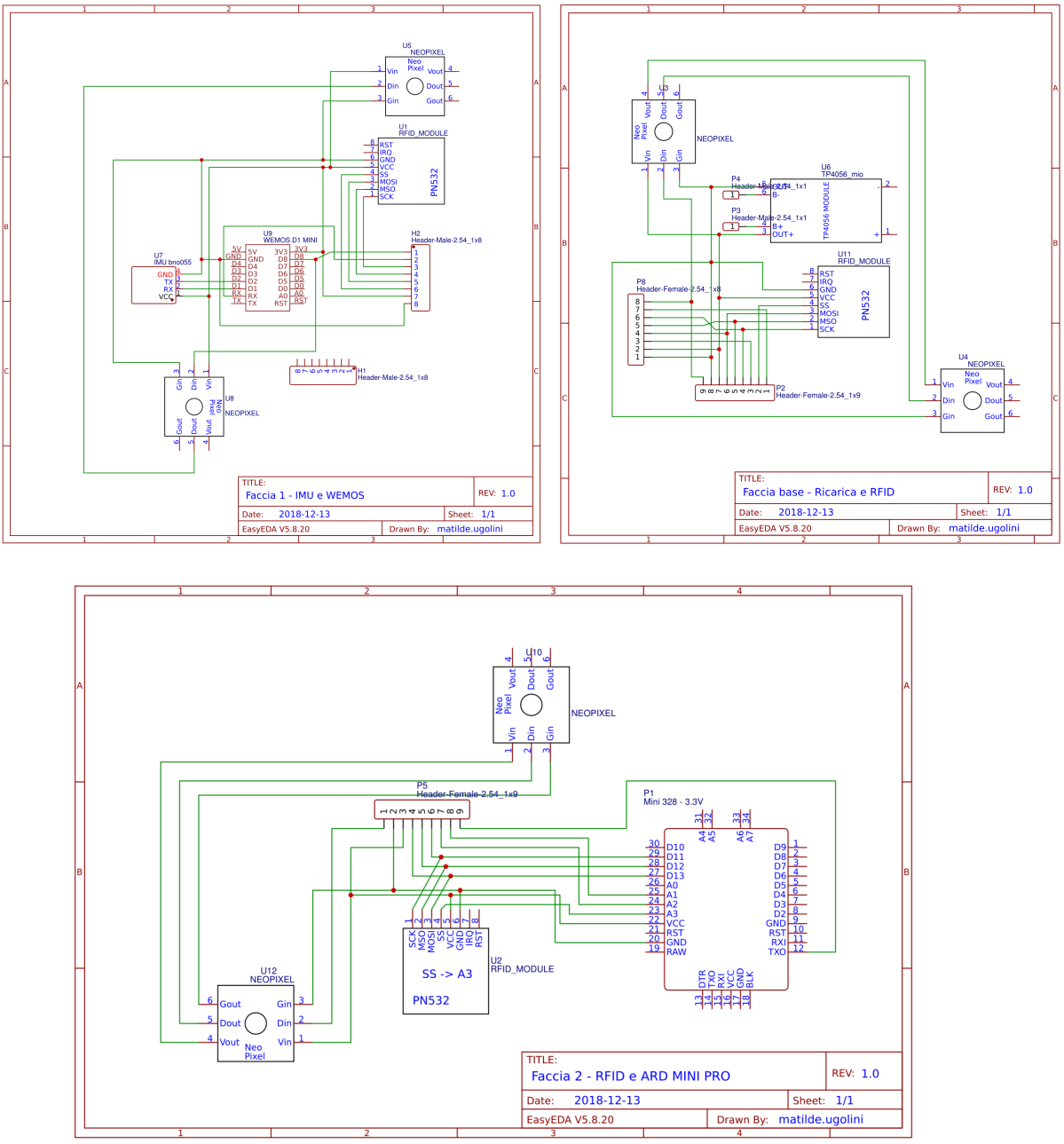


Figura 3.8: Schematici dei tre sotto-circuiti di CUB8

Gli schematici sono poi stati tradotti nelle corrispettive PCB, di cui è stato fatto il seguente design.

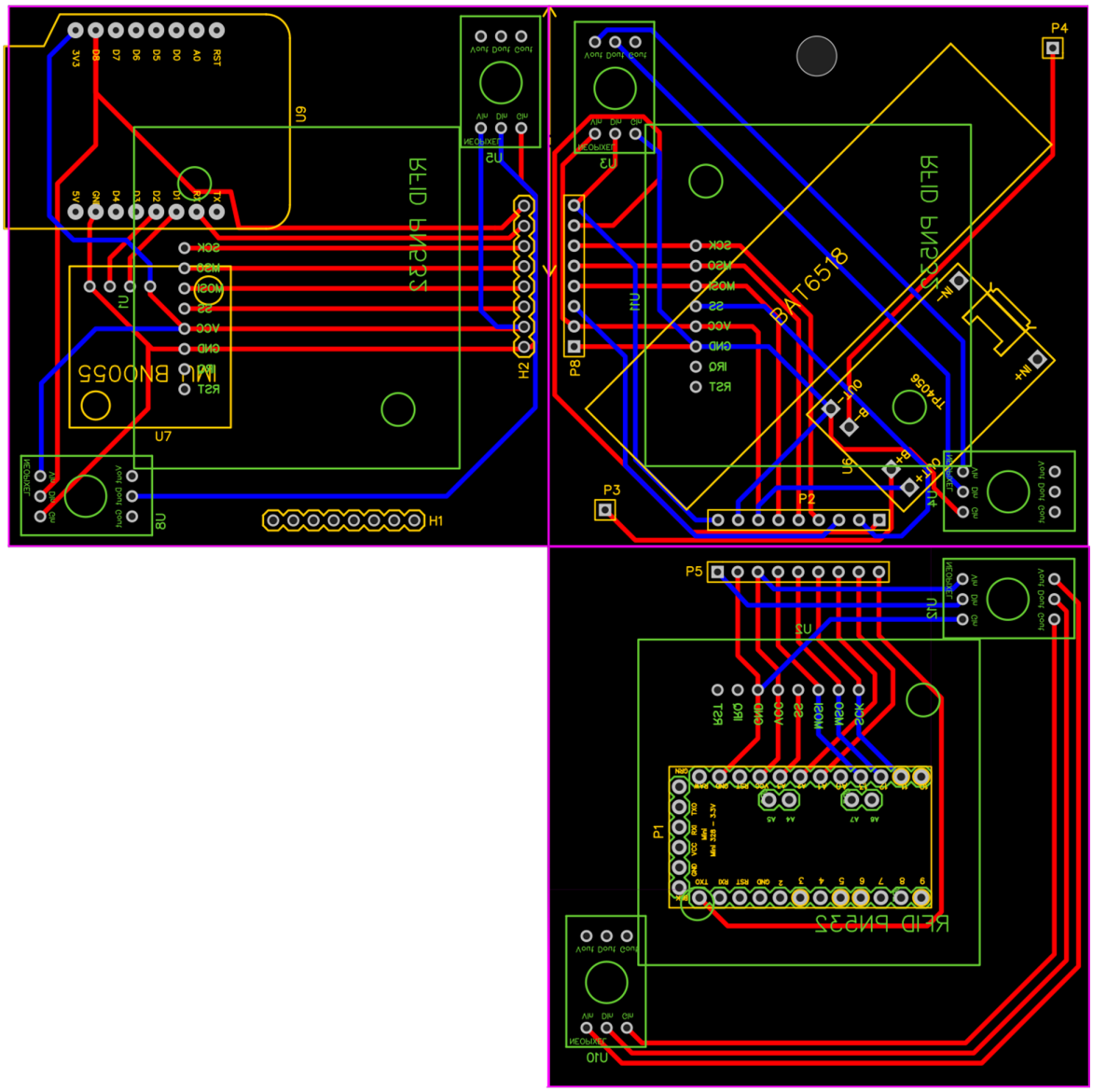


Figura 3.9: Design delle PCB di CUB8

La faccia in alto a destra nella figura di riferimento 3.9 rappresenta la base del "mezzo cubo", mentre le altre due sono le facce anche andranno montate meccanicamente a questa tramite l'utilizzo dei connettori perpendicolari. Per effettuare tali connessioni meccaniche sono stati utilizzati i componenti *Header* maschio (siglati con *H* seguito da un numero progressivo) e femmina (identificati con *P* seguito da un numero progressivo) visibili sia negli schematici 3.8 che nelle PCB 3.9. Dal momento che non tutti i componenti elettronici che necessitano di collegamenti elettrici si trovano sulla stessa PCB, si sono sfruttati questi

Header come tramite per far passare i segnali da una PCB all'altra, realizzando le connessioni elettriche tra i vari componenti, laddove necessario. In particolare l'Header *P5* sarà collegato con *P2* e l'Header *P8* con *H2*.

3.3.2 Design del contenitore

Allo stadio attuale dello sviluppo, non è ancora stata presa una decisione definitiva circa la realizzazione del contenitore della nuova versione di CUB8. Sono state intraprese e sperimentate molte strade, ma nessuna di queste ha portato ad un risultato degno di nota, nè tantomeno risolutivo. Il problema principale, come anticipato, rimane l'illuminazione del cubo: essendo ora potenzialmente componibili tutte e sei le facce del cubo non è più possibile adottare come soluzione la sospensione di una struttura illuminata (vedi sotto-sezione 2.2.3) al centro di CUB8. Questo perchè ormai il centro del cubo contenitore sarà occupato dall'altro cubo di PCB, più interno. Di conseguenza, realizzare un tipo di illuminazione diffusa al centro delle facce risulta attualmente impossibile. Si è dovuto dunque pensare a metodi alternativi per sfruttare l'illuminazione dei LED RGB WS2812 per fornire feedback visivi all'utente.

Un primo tentativo è stato effettuato utilizzando una fibra ottica *full light*, fibra che se illuminata su un'estremità, trasporta la luce su tutta la sua lunghezza, e non solo *end-to-end*, al fine di illuminare un *fill pattern* riempitivo, in modo reattivo alla manipolazione dell'utente. Si è quindi tornati a pensare all'utilizzo del legno come materiale per la costruzione del contenitore, non avendo più il problema di dover lasciar passare la luce all'esterno. L'idea era creare delle scanalature sulle facce di legno del cubo e inserire al loro interno la fibra ottica (⌀2 mm), illuminandone un'estremità con un LED.

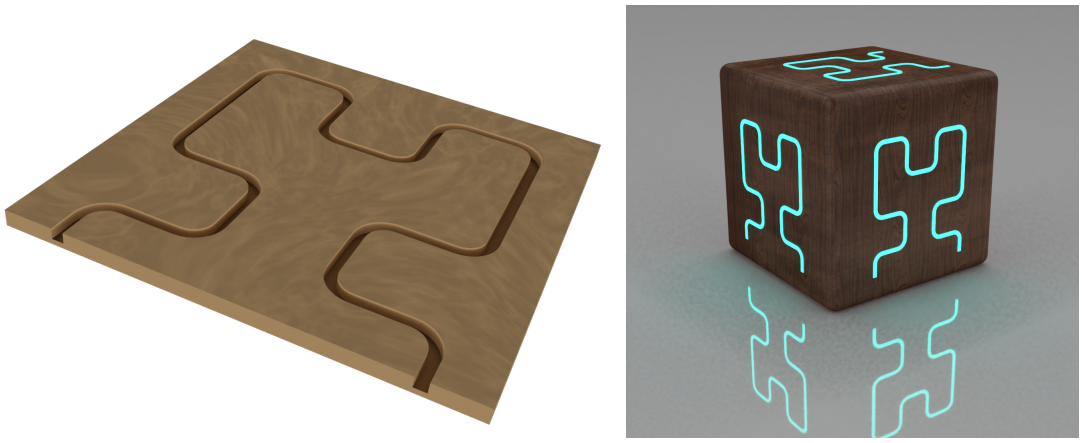


Figura 3.10: Render della scanalatura del fill pattern e del CUB8 risultante atteso

Purtroppo i risultati non sono stati soddisfacenti e comunque non in linea con le aspettative: il problema principale risiede nel fatto che più strada viene percorsa dalla luce attraversando la fibra e più si indebolisce, fornendo scarsa illuminazione. Un altro problema riscontrato riguardava la disposizione spaziale dei LED RGB all'interno delle PCB e sul come mantenere a contatto l'estremità delle fibre con la superficie del LED. Non essendo soddisfatti dell'effetto visivo finale ottenuto, si è preferito non indugiare ulteriormente nella ricerca di una soluzione basata su fibra ottica.

Alla luce degli esperimenti effettuati, è stato deciso di cambiare totalmente prospettiva. La soluzione verso la quale si sta dunque procedendo non riguarda più l'illuminazione delle facce di CUB8, ma l'illuminazione degli spigoli della struttura, i quali necessitano meno di avere un'illuminazione diffusa. Tuttavia, anche questa soluzione è in fase di studio e sviluppo e non è stata ancora implementata interamente. La situazione ottimale sarebbe avere un'infrastruttura di supporto in materiale trasparente o semi tale, che possa far trapelare la luce dei LED RGB posti ai vertici di CUB8. Tale struttura per facilitare l'assemblaggio delle parti interne, è stata progettata per essere composta di due parti come visibile nella figura a sinistra sotto riportata, le quali possono essere "avvitate" insieme grazie alle scalanature interne previste sulla metà superiore che permettono l'accesso a un cacciavite dalle dimensioni ridotte.

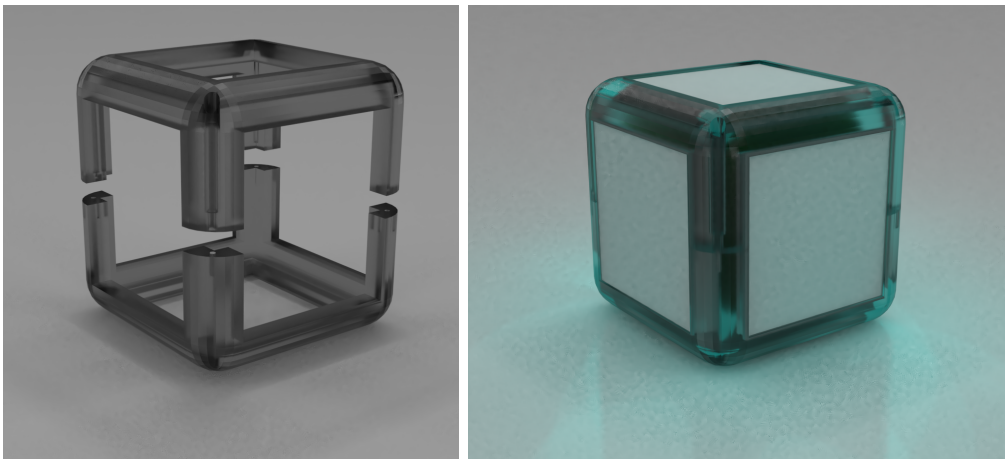


Figura 3.11: Render della struttura trasparente e del CUB8 risultante "acceso", contenente anche il cubo interno composto da PCB

Le facce interne di CUB8 saranno poi adeguatamente chiuse con degli scassi simili a quelli della versione 1.0, di materiale ancora da definire. Il legno potrebbe essere un ottimo candidato, per rimanere in linea con i requisiti postisi inizialmente. Tuttavia, non è certo che il connubio materiale plastico trasparente

e legno sia vincente. Un altro plausibile materiale di copertura delle facce potrebbe essere il silicone o un altro materiale flessibile. Tale flessibilità potrebbe agevolare la fase di assemblaggio del cubo.

Parte IV

Conclusioni e sviluppi futuri

Il progetto di Tesi proposto ha portato alla produzione di un primo prototipo funzionante di CUB8, che sebbene non rispecchi ancora il prodotto che si desidera ottenere, ha permesso di farci avere un'idea del potenziale e di gettare le basi per uno sviluppo futuro. Tale sviluppo, come anticipato è già in corso d'opera, e si prevede di riuscire ad avere un nuovo prototipo aggiornato nel corso dei prossimi mesi, grazie al quale potranno essere effettuati nuovi test e ridefiniti laddove necessario nuovi requisiti di progetto.

Attualmente l'ostacolo più ingente è la costruzione di un contenitore esterno di CUB8 che permetta di essere illuminato dall'interno, in modo che fornisca feedback visivo all'utente, e al contempo permetta la comunicazione RFID sulle sei facce del cubo e insieme a questa la componibilità magnetica fra i CUB8 attivi nell'applicativo virtuale. È proprio la mancanza di un'idea definita su come produrre lo chassis che sta impedendo l'avanzamento della prototipazione: senza sapere esattamente come deve essere implementato è difficile stabilire dove posizionare i LED RGB WS2812 all'interno delle PCB e di conseguenza non è sostenibile ordinare la stampa di queste ultime, e senza avere le PCB fisiche non è possibile avviare la fase di testing tra prototipi. Lo sviluppo futuro prossimo, al quale si lavorerà attivamente, *tenderà a* e *tenterà di* spezzare tale circolo vizioso, al fine di produrre la versione finale del *prodotto* CUB8, dunque non più prototipo.

Un altro obbiettivo per gli sviluppi futuri, sarà la definizione precisa dei requisiti di progetto che dovranno avere gli applicativi software che utilizzeranno CUB8 come controller "analogico" per pilotare e manipolare l'ambiente virtuale definito dall'applicazione. Il lavoro della presente Tesi si è concentrato quasi esclusivamente sulla progettazione e produzione dell'oggetto fisico CUB8. Essendo questo un'interfaccia - tangibile e concreta ma pur sempre interfaccia - l'implementazione dell'applicativo posto sull'elaboratore centrale si plasmerà su essa, interpretando i dati circa lo stato dei cubi attivi in modo specifico e diverso in base al contesto di utilizzo e i requisiti richiesti. Ogni set di CUB8 dunque può essere controllato e controllare potenzialmente qualsiasi applicativo legato a questi dal momento che possono essere considerati dispositivi quasi totalmente passivi: il loro unico scopo è quello di comunicare all'esterno i dati sul proprio orientamento e sugli Id dei CUB8 esterni collegati alle loro facce magneticamente e pilotare i propri LED RGB come ordinato dall'esterno. Non operano un'interpretazione di questi dati, di conseguenza sono disgiunti dallo scopo finale dell'applicazione fornendo astrazione sufficiente per essere considerati interfacce, implementabili in modi differenti.

I campi di applicazione in cui possono essere adoperati questi cubi componibili interattivi sono molteplici e non esclusivamente legati al mondo dell'intrattenimento ludico infantile, come si potrebbe pensare.

Il campo principale di applicazione individuato è legato al mondo dei bambini, in particolare al settore EDU e della psicoterapia infantile. CUB8 vorrebbe inserirsi nell'educazione come intermediario fra bambini e tecnologia. È importante infatti al giorno d'oggi che le nuove generazioni - assorbite fin dalla nascita in un contesto altamente tecnologizzato - continuino a percepire che esiste una distinzione fra analogico/reale e digitale, aspetto che applicazioni mobile o tablet, utilizzate fin dalla tenera età, tendenzialmente non tengono in considerazione. CUB8 invece, essendo un oggetto concreto e tangibile può aiutare in questa presa di coscienza, seppur comunque insegnando al bambino che esistono delle relazioni fra i due mondi, i quali anzi si trovano oggigiorno fortemente integrati. Il focus è quindi sull'insegnamento più o meno indiretto ad avere maggiore consapevolezza della tecnologia in quanto mezzo e non in quanto fine, e a farci porre come soggetti attivi e non solo passivi.

È prassi comune per logopedisti, psicologi e psichiatri infantili fare utilizzo di attività ludiche per mediare l'interazione con i bambini durante le sessioni di terapia. Il gioco è visto infatti come strumento di esplorazione e scaricamento delle tensioni per il bambino e strumento di supporto nell'analisi per lo specialista. CUB8 potrebbe dunque farsi intermediario in questo processo, implementando una serie di attività utili o rendendo quelle pre-esistenti più interattive.

Un altro campo di applicazione potenziale è quello delle installazioni artistico audio-visive interattive, in cui è lo spettatore ad *essere* e *manipolare* l'opera stessa tramite le interazioni che applica al controller CUB8. L'applicativo dunque fornirebbe supporto visivo riproducendo entità virtuali, ognuna legata ad un cubo diverso o alla composizione magnetica di più cubi, e modificandole sulla base di un set di *gesture* che l'utente può effettuare e che il software si impegna a riconoscere. Sempre legato a questo mondo, CUB8 ben si potrebbe prestare anche ad esibizioni musicali: i principali software volti alla produzione di musica elettronica interattiva (e.g. *MAX*) basano infatti il loro funzionamento su linguaggi di programmazione visuale a nodi/blocchi di oggetti che si comportano come programmi auto-contenuto, i quali ricevono valori in input generando output se connessi tra loro, producendo così segnali acustici digitali. Si potrebbe dunque creare un parallelismo tra queste entità modulari e il modulo CUB8, e fra le relazioni che uniscono tali entità con il magnetismo che collega i cubi. Ruotando e componendo cubi diversi tra loro si andrebbe a modificare interattivamente i valori delle variabili/messaggi scambiati fra componenti connessi, andando dunque a produrre effetti sonori diversi.

Gli obbiettivi per il futuro prossimo sono dunque la realizzazione di un secondo prototipo funzionante di CUB8, del quale creare molteplici copie per poter poi passare alla fase di testing sulla componibilità e usabilità del prodotto. Parallelamente a ciò, dovrà essere presa una decisione circa il principale campo

di applicazione in cui si vuole provare ad inserire CUB8 ed iniziare un'effettiva indagine di mercato e/o analisi dei requisiti facendo riferimento alle figure professionali del settore individuato.

Bibliografia

[1] AnalysisIR. *Infrared receiver showdown – TSOP34438 vs VS1838B winner revealed*. URL: https://www.analysir.com/blog/2014/12/08/infrared-receiver-showdown-tsop34438-vs-vs1838b-winner-revealed/.

[2] Atmel. *Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash. ATtiny25/V / ATtiny45/V / ATtiny85/V Summary*.

[3] *Boards - Generic ESP8266 Module*. URL: https://arduino-esp8266.readthedocs.io/en/latest/boards.html.

[4] Sensortec BOSCH. *BNO055 Datasheet. Intelligent 9-axis absolute orientation sensor*.

[5] *ESP-12S User Manual*.

[6] *ESP8266 Arduino Core's documentation*. URL: https://arduino-esp8266.readthedocs.io/en/latest/index.html#.

[7] M. Zarro G. Malnati. "Introduzione ai microcontrollori". 2015-2017.

[8] Daniel Garcia. *FastLED Animation Library*. URL: http://fastled.io/.

[9] Ivan Grokhotkov. *ESP8266 Arduino Core Documentation*. Ver. Release 2.5.0. URL: https://media.readthedocs.org/pdf/arduino-esp8266/latest/arduino-esp8266.pdf.

[10] Imroy. *PubSubClient*. URL: https://github.com/Imroy/pubsubclient#MQTT_version.

[11] Spence Konde. *ATtinyCore*. URL: https://github.com/SpenceKonde/ATTinyCore.

[12] LFN. *Infrared Receiver Module*. Chinese. URL: http://www.electronicoscaldas.com/datasheet/VS1838B_LFN.pdf.

[13] *Pacchetto MQTT per nodejs*. URL: https://www.npmjs.com/package/mqtt.

[14] RobotStore. *CARICABATTERIE LI-ION TP4056 CON CIRCUITO DI PROTEZIONE*. URL: https://www.robotstore.it/Caricabatterie-Li-Ion-TP4056-con-circuito-di-protezione.

[15] Espressif System. *ESP8266EX Datasheet*. English. 2018. URL: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.