UNIVERSITA' DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI

CORSO DI LAUREA IN INGEGNERIA MECCATRONICA

TESI DI LAUREA TRIENNALE

Analisi e Sviluppo di una Serra Idroponica Controllata Mediante Scheda Arduino

Relatore: Paolo Magnone

Laureando: Filippo Tasca
1192960

ANNO ACCADEMICO: 2021/2022

Prefazione

La seguente tesi, frutto del progetto che ho sviluppato nel corso degli studi triennali di Ingegneria Meccatronica, tratta la tecnica *idroponica* per la coltivazione delle colture e degli ortaggi *indoor*. Può essere facilmente estesa anche ad una coltivazione *outdoor* adattando il programma automatico per la gestione della serra e scegliendo dei componenti adatti a questo utilizzo. L'idea partì durante il primo anno di università. Una domenica a pranzo ero seduto a tavola con la mia famiglia. Solitamente guardiamo il programma TV "Melaverde", in onda su Canale 5, nel quale i conduttori girano l'Italia portando al pubblico a casa le bellezze del territorio e tutto ciò che riguarda agricoltura e allevamento. Il servizio di quel giorno trattava la tecnica idroponica per la coltivazione delle fragole. Incuriosito dal nome specialmente, iniziai fin da subito ad informarmi sulla tecnica. Fu però durante il secondo anno di università che ebbi le idee chiare e potei partire con il progetto della serra idroponica controllata mediante un programma automatico. Infatti, dal secondo anno le materie di studio iniziarono a farsi più "tecniche" e specifiche, rispetto al primo. In quel momento, mi accorsi che molti professori insistevano sulla parola *efficienza* durante le loro lezioni. Anche se accostata a materie diverse tra loro, con nessun legame comune, il termine era comunque sempre ricorrente. L'intenzione dei professori era quella di far capire a noi studenti che l'efficienza è fondamentale al giorno d'oggi, ma specialmente per il futuro. Fu in quel momento che posi la domanda: che benefici potrebbe portare una serra idroponica in questo momento, con un occhio di riguardo anche al futuro? E' davvero così difficile creare una serra idroponica con un ciclo di controllo automatico che permetta di ottenere all'utente finale dei benefici, sia dal punto di vista del controllo ma anche dal punto di vista della resa? Ho sempre ammirato lo sviluppo che ha avuto l'agricoltura, specialmente negli ultimi anni, grazie all'introduzione di macchinari, sistemi e tecniche che fanno uso di soluzioni ingegneristiche, molte volte anche complesse. Fu così che iniziai ad informarmi sul metodo di coltivazione idroponica, leggendo articoli e documenti che trovavo, comprando libri e chiedendo anche ad amici agricoltori. Ancora non mi bastava però. Avevo ideato un sistema "rudimentale" su carta, una bozza di serra, altamente inefficiente. Frequentando un corso di ingegneria, certamente non ero soddisfatto del mio progetto. Potremmo dire che partì tutto per scherzo, per una serie di motivi. I miei genitori erano davvero stanchi di sentirmi parlare continuamente della coltivazione idroponica, e di tutte le mie idee a riguardo. Un giorno, decisero di prendere una "serra", una sorta di "giocattolo" educativo per bambini dai sei anni in su. Fu un modo scherzoso per dirmi che era ora di darsi da fare, e non solo parlare e basta. Lentamente iniziai ad acquistare i primi componenti, a testarli e ad assembrarli, fino a quando iniziò a prendere forma. E' sempre stata una sfida, fin dal primo giorno, un po' come lo spirito che dovrebbe avere un ingegnere quando gli si

presenta un problema davanti e prova soddisfazione nel vederlo risolto. Volevo dimostrare agli altri che credevo veramente nel progetto, e a me stesso che ero in grado di realizzare qualcosa con le mie forze. Quando la serra era ormai pronta iniziai a stilare le prime righe di codice. Quest'ultimo, è stato scritto molte volte, a causa del fatto che non ero mai pienamente soddisfatto del programma e di come funzionava. Ed ora eccomi qui, a scrivere questa tesi di laurea, su un progetto partito quasi per scherzo, ma con delle potenzialità rivolte verso il futuro. Non sarebbe la prima volta che dei ragazzi, partendo dalla loro camera da letto o dal loro garage, mettendo in pratica e sviluppando le loro idee, riescono a creare qualcosa di importante. Non voglio minimamente paragonarmi a loro, ma permettetemi di sognare ad occhi aperti ed essere ambizioso come ho sempre fatto. In fondo, è l'ambizione il vero motore alla base della crescita e dello sviluppo che ogni giorno è sotto agli occhi di tutti.

Indice

1 Introduzione

La **coltivazione idroponica**, che sempre più viene utilizzata al giorno d'oggi, in realtà ha origini molto antiche. Le parole derivano dal greco antico (*hýdor*, "acqua" e *pónos*, "lavoro"). Spesso viene chiamata anche con il nome di **idrocoltura**. A prescindere dal termine che si vuole attribuirle, questa tecnica prevede una coltivazione delle colture fuori suolo: il terreno comune viene sostituito da un substrato inerte (ad esempio argilla espansa, vermiculite, perlite, zeolite, lana di roccia, fibra di cocco, ecc.). Alla pianta viene fornita una *soluzione nutritiva*, formata da acqua e composti (per lo più inorganici) che rappresentano tutti gli elementi necessari per la crescita e lo sviluppo della pianta, e quindi al suo corretto apporto nutritivo. A differenza delle coltivazioni classiche, la coltura idroponica consente di ottenere delle produzioni maggiormente controllate, sia dal punto di vista qualitativo, sia in quello igienico-sanitario, grazie al modo in cui vengono coltivate le colture. Questi benefici inoltre, sono ottenibili e costanti per tutto l'anno. [1] Riguardo questa tecnica, non è ancora ben conosciuto il periodo storico della sua ideazione. Vi sono storici che ricollegano la coltivazione idroponica al periodo storico dei Babilonesi, tra il III e il II millennio a.C., utilizzata nei giardini pensili per l'irrigazione delle piante. Altri ancora al periodo degli Aztechi, intorno al XV secolo. Persino nell'antica Roma sono stati trovati reperti riconducibili all'utilizzo di questa tecnica. [2] A quel tempo, non era sicuramente avanzata come lo può essere ai nostri giorni. In qualsiasi caso se il suo reale utilizzo fosse dimostrato, si potrebbe pensare che i benefici introdotti, considerando le competenze tecniche e teoriche del tempo, erano apprezzabili rispetto ad altri metodi. Pensiamo per esempio a Babilonia, in Mesopotamia, terreno molto fertile e ricco d'acqua, adatto alla coltivazione grazie alla presenza del limo [1]. Anche nella zona del Messico centrale, territorio degli Aztechi, l'acqua e il terreno fertile non scarseggiavano, grazie alla composizione vulcanica del suolo. Sono tutti esempi che ribadiscono i benefici introdotti dall'utilizzo della coltivazione idroponica. Spostandosi verso il periodo moderno, negli ultimi anni questa tecnica ha subito un enorme sviluppo. Già nel XIX secolo iniziarono i primi studi e le prime sperimentazioni pratiche. Biologi e botanici iniziarono nuovamente a riprendere questa tecnica, e a studiarne i benefici che poteva introdurre nell'agricoltura. Persino durante la Seconda Guerra Mondiale questo metodo di coltivazione venne utilizzato dalle truppe americane per rifornire i soldati dispiegati nel Pacifico, a causa della tipologia di territorio presente nelle isole dell'Oceano, che non rendevano facile la coltivazione sul suolo roccioso. [2] Negli ultimi anni, una delle ultime notizie più importanti è sicuramente l'interesse, da parte della NASA, sull'utilizzo di questa tecnica durante le

[1] Il *limo* è un substrato inerte, trasportato in sospensione dai fiumi, con ottime caratteristiche per la crescita delle colture.

future spedizioni verso Marte e già applicata in assenza di gravità sulla ISS (*"International Space Station"*), in orbita attorno alla Terra. Purtroppo si parla solo di idee per il momento, infatti sarà necessario studiare bene tutti gli aspetti di interesse per la missione. Probabilmente potrebbe nascere un sistema *ibrido*, basato sulla coltivazione idroponica e adattato per il suolo del Pianeta Rosso, ricco di minerale di ferro, particolarmente utile per la regolazione delle funzioni vitali delle piante. Fondamentalmente questi sono i collegamenti più rilevanti ed interessanti da fare riguardo la coltivazione idroponica. Come per tutti gli ambiti, il mondo è in continua evoluzione ogni giorno. Grazie a ricercatori, studiosi, ideatori ma anche persone comuni, ma intraprendenti e volenterose, si compiono piccoli passi in avanti nel progresso. Nel futuro, con i cambiamenti climatici alle porte, le coltivazioni intensive e la continua crescita demografica della popolazione, con un incremento del fabbisogno energetico e nutrizionale direttamente proporzionato, vi sono forti fattori di rischio per il futuro dell'umanità stessa se non si interverrà subito. Come altre tecnologie e tecniche, anche la coltivazione idroponica può ritagliarsi uno spazio per evitare queste problematiche.

Nella tesi sviluppata, verranno trattati vari temi riguardo la coltivazione idroponica. Sicuramente la tematica ambientale odierna, come quella futura, saranno dei punti forti, per marcare ancor di più la necessità e l'importanza di apportare dei cambiamenti radicali nei metodi di coltivazione delle colture. Inoltre, con dati e descrizioni dettagliate, sarà possibile distinguere le varie tecniche di coltivazione idroponica e le differenze tra queste. Verranno trattati anche i vari tipo di substrato e i parametri di controllo, necessari per creare un clima e un ambiente favorevole alla crescita e allo sviluppo delle piante. Solo in seguito, la serra implementata. Saranno presenti calcoli numerici, schemi elettrici, datasheet e diagrammi di flusso utili a comprendere l'implementazione pratica della serra e la sua programmazione logica. Saranno riportate motivazioni sulle scelte fatte durante la fase di ideazione e realizzazione, anche riguardo la differenza con quelle già presenti nel WEB.

2 Tematica ambientale

Il pianeta Terra, che ci ospita assieme a molte altre forme di vita, sia animali che vegetali, è in continua evoluzione. Gli scienziati la definiscono "viva", non perché abbia una coscienza o un pensiero, ma perché muta ogni giorno. La scienza ci insegna che milioni di anni fa la struttura e la morfologia non erano sicuramente quelle odierne. Le placche tettoniche sono in continuo movimento, le montagne si innalzano a causa del continuo e lento spostamento di lembi di terra. L'uomo, essere vivente dotato di intelletto ed intelligenza, condivide con le altre specie animali e vegetali la propria presenza su questo splendido pianeta. L'umanità quindi, avendo consapevolezza ed intelligenza, sempre considerata superiore a tutte le altre specie, ha l'obbligo di rispettare e preservare il luogo in cui vive sia per se stessa, ma specialmente per tutti gli altri ospiti con cui condivide lo stesso luogo. Come insegnano storia e scienza la natura, in qualche modo, ristabilisce sempre gli equilibri e lo stato di quiete. L'uomo, con la sua esistenza e sussistenza sulla Terra, è già da molti anni che sta rompendo questi equilibri. Dal 1969, la comunità scientifica internazionale ha deciso di istituire il cosiddetto *"Earth Overshoot Day"* (EOD), un indice illustrativo che rappresenta il giorno dell'anno in cui l'umanità consuma interamente le risorse naturali prodotte dal pianeta Terra, che riesce a produrre nel ciclo di un anno. Si potrebbe pensare a questo indice come l'indicatore che rappresenta il numero di pianeti Terra necessari per il sostentamento della sola umanità. Dal 1969 al 1970 l'indice era pari a 1,00. In quegli anni, vi era equilibrio tra uomo e pianeta. Si prelevava un numero di risorse pari a quelle che la Terra era in grado di riprodurre nel ciclo di un anno. Dagli anni successivi invece, l'EOD iniziò ad aumentare notevolmente. Nel 2021 l'indice EOD era pari a 1,75, e il giorno dell'anno calcolato il 29 luglio. Per il 2022, la stima dell'indice è leggermente più alta. Viene stimato il 28 luglio, anticipando di un giorno quello dell'anno precedente. In 53 anni, l'umanità è passata dall'aver bisogno per la propria sussistenza di un solo pianeta Terra, a quasi due. Diversamente da quello che si può pensare, i cambiamenti climatici non sono così distanti quanto crediamo. Le difficoltà idriche di approvvigionamento dell'acqua, le falde che si abbassano, e l'acqua salata del mare che rientra verso l'entroterra, sono problemi da non sottovalutare. Nel territorio limitrofo a Bassano del Grappa, gli effetti climatici si sono visti in passato e sono ancora più marcati negli ultimi anni. Il fiume Brenta, che nasce dai laghi di Levico e Caldonazzo, attraversa il Trentino Alto Adige e la regione Veneto, sfocia poi nel Mar Adriatico, nella zona della Laguna di Venezia. Grazie al suo percorso, assicura una fonte di acqua preziosa per le colture che sono coltivate nei campi, anche a chilometri di distanza dal corso del fiume. Grazie ad un complesso sistema di canali, più o meno grandi, chiamati in gergo locale

"brentelle" [2], permettono agli agricoltori di avere sempre accesso ad una fonte di acqua nel periodo della piena coltivazione agricola. Come è facile pensare, se manca l'acqua nel fiume principale mancherà anche nei canali ad esso collegati, rappresentando un problema molto grave per gli agricoltori e le loro coltivazioni. Nel 2022, come accaduto nel 2003, la crisi idrica ha colpito gravemente la zona attorno al fiume Brenta, completamente prosciugato. Come si può notare nella *figura 1*, questo fenomeno deve preoccupare e far riflettere.

Figura 1: Fiume Brenta completamente in secca durante il periodo di crisi idrica - foto di Stefano Maruzzo - scattata in data 18/06/2022.

[2]Termine dialettale veneto per esprimere i canali di approvvigionamento dell'acqua che partono dal fiume Brenta e si diramano lungo tutto il territorio limitrofo, anche per molti chilometri di distanza, che permetto agli agricoltori di usufruire dell'acqua per irrigare i campi.

Figura 2: Fiume Brenta con la portata che normalmente garantisce per tutto l'anno - foto di Stefano Maruzzo - scattata in data 30/03/2014.

Nella regione Veneto, la produzione cerealicola è una tra le più alte d'Italia, e la carenza di acqua mette a serio pericolo l'economia della regione stessa, e il mercato nazionale. Inoltre, le ripercussioni più gravi comunque ricadono sempre sui cittadini, sia sul loro portafoglio ma anche sulla loro salute alimentare. In media, un metro quadro di terreno agricolo, assorbe più o meno acqua a seconda delle diverse piante che vi crescono. Le stime riportate in *Tabella 1* sono indicative, ma mettono in risalto la quantità necessaria per una resa media relativamente alta, senza considerare eventuali agenti atmosferici esterni o altri fattori di danno collegati alla produzione.

	Coefficiente di traspirazione		
Coltura	da	a	
Mais	300	400	L/Kg
Orzo	400	500	L/Kg
Frumento	400	600	L/Kg
Colza	600	700	L/Kg
Segale	400	500	L/Kg

Tabella 1: Quantità di acqua richiesta dalle varie colture per ottenere un Kg di sostanza secca [3].

E' possibile osservare una richiesta molto elevata di acqua, poiché gran parte di questa viene dispersa nel terreno e solo una minima parte realmente assimilata dalla pianta. Se estendiamo il concetto anche a tutte le piante da frutto, la quantità necessaria per la loro crescita, maturazione e fioritura

porta ad avere stime ancora superiori. La coltivazione idroponica non è l'unica ma può essere una delle soluzioni al problema, essendo a tutti gli effetti una strada percorribile e realizzabile. Difficile pensare di riuscire a coltivare interi ettari di terreno coltivati a colture cerealicole con questo metodo. Solo i costi sarebbero esorbitanti, per non parlare poi dell'impatto ambientale che avrebbero. Il terreno è un "micro-sistema", regolato naturalmente da piante, organismi ed insetti che vi vivono. Alterare questa composizione può portare dei benefici, ad esempio sulla resa e sulla quantità di prodotto ottenibile, ma andando a distruggere gran parte di questo sistema equilibrato. I pesticidi eliminano ciò che danneggia e compromette la crescita delle piante, ma anche gli insetti e gli organismi che naturalmente la migliorano. La coltivazione idroponica non presenta questo problema. E' anche vero però che questo metodo può essere applicato solo per frutta ed ortaggi di piccole dimensioni. D'altro canto, con la sola coltivazione di questi però, si andrebbe a migliorare lo spreco idrico nell'agricoltura in generale, contribuendo in parte a migliorare la situazione.

3 Tecnica di coltivazione idroponica

La tecnica di coltivazione idroponica si basa sull'uso dell'acqua. Le tecniche per garantire l'apporto di questo elemento essenziale alle piante sono molte. Le differenze che distinguono tra di loro i vari metodi di coltivazione idroponica sono essenzialmente: la modalità di irrigazione, il substrato utilizzato e il luogo di installazione della serra. Altri fattori che possono influenzare la distinzione sono ad esempio il metodo di gestione della serra, del clima al suo interno o altro ancora. Questi ultimi però, sono degli aspetti di contorno. Infatti, la tecnica di idroponica non richiede necessariamente un controllo automatizzato, ma può essere applicata anche in condizioni di assenza di quest'ultimo. Vari team di ricercatori negli anni hanno portato avanti studi e sperimentazioni per trovare un metodo migliore, che potesse introdurre dei benefici nettamente superiori agli altri. Sfortunatamente ogni soluzione adottata portava con se sia degli aspetti positivi, sia altri negativi. Molto spesso, la soluzione ottimale richiedeva l'uso di due o più metodi assieme, in modo da compensare gli aspetti negativi introdotti da un metodo con quelli positivi dell'altro.

3.1 Tipologie di serra

La prima distinzione importante da fare è sul luogo di installazione della serra. Questo influirà sul progetto della serra e sul metodo di coltivazione da adottare. Due sono le tipologia adottabili: *INDOOR*, quindi all'interno di un edificio, oppure *OUTDOOR*, all'esterno. Il clima e gli agenti atmosferici che intaccano la serra in un luogo all'aperto sono sicuramente diversi da quelli per una serra al chiuso, dove sono molto bassi in media o addirittura assenti in alcuni casi. Inoltre, adottando una tecnica di gestione automatizzata della serra, questa dovrà tenere conto anche di questo aspetto. La scelta dei componenti, i materiali utilizzati, le colture coltivabili in serra saranno diverse. Una serra esterna richiederà una resistenza all'usura nettamente superiore, e anche la grandezza sarà notevolmente differente.

3.1.1 Metodi di coltivazione

Oltre al luogo di installazione della serra, vi sono altre distinzioni importanti da osservare. La tipologia di irrogazione del substrato delle piante è la principale differenza per le coltivazioni idroponiche. L'acqua e la soluzione nutritiva possono raggiungere l'apparato radicale della pianta in modi molto diversi:

- *Sistemi passivi* (*wick systems*), detti anche "a stoppino", *figura 3a*. Questi sistemi non si avvalgono di una pompa: la soluzione nutritiva contenuta nel serbatoio, posto sotto alle piante, giunge per capillarità all'interno del substrato, dove insedia la zona radicale della pianta. Lo

svantaggio è legato principalmente all'approvvigionamento dell'acqua e dei sali minerali da parte della pianta: con questo metodo infatti, i substrati inerti [3], faticano a mantenersi umidi a causa della scarsa irrorazione offerta dal metodo stesso, e dunque rappresentano un limite non trascurabile per il sistema.

- *"Flood and drain"*. I seguenti sistemi, rappresentati in *figura 3b*, sono caratterizzati da un ciclo di irrigazione delle piante che, attraverso l'uso di una pompa e di un unico serbatoio, permettono di bagnare la zona radicale della pianta e fornire i sali minerali necessari a quest'ultima per crescere. L'acqua in eccesso che non può essere assorbita viene poi dispersa nell'ambiente o recuperata. In questo modo, è possibile garantire un afflusso costante e controllato di soluzione nutritiva alle piante e di operare un risparmio di acqua, se viene implementato un sistema a *ciclo chiuso* [4]; Altro sistema, può essere mediante l'utilizzo di due serbatoi e l'uso della gravità al posto della pompa. Un serbatoio viene posto a monte, più alto del livello delle piante, e funge da contenitore di pescaggio. L'altro viene posto a valle, sotto il livello dell'apparato radicale, fungendo da contenitore di recupero. In questo caso, il numero dei componenti si riduce, e il controllo si semplifica leggermente. E' un metodo che può essere applicato ovunque, in quanto non richiede fonti di energia.

- *NFT* (*"Nutrient Film Technique"*), *figura 3c*. Questo metodo è uno tra quelli più utilizzati, specialmente nelle grandi serre. Prevede l'uso di cubetti in fibra di cocco o lana di roccia principalmente, nei quali le radici delle piante crescono. Questi rappresentano il substrato che fa da ancoraggio alle piante durante la crescita. La soluzione nutritiva viene fornita alla pianta mediante un afflusso continuo di acqua che bagna i cubetti in materiale inerte, posti all'interno di tubi forati e inclinati che permettono lo spostamento dell'acqua solo in un verso. E' un metodo molto efficiente, facilmente regolabile e controllabile, ma ha lo svantaggio di non garantire sempre lo stesso afflusso di nutrienti dalle prime piante, da dove viene fatta fuoriuscire l'acqua che entra nel tubo, verso le ultime. Il problema è facilmente risolvibile però. Installando lungo i tubi diversi punti di immissione dell'acqua e della soluzione nutritiva, i nutrimenti e i sali minerali possono raggiungere tutte le piante in modo equilibrato, fornendo un apporto bilanciato.

[3] Con *substrato inerte* intendiamo la zona nella quale la parte radicale della pianta ramifica e assorbe le sostanze nutritive necessarie alla crescita. Substrati inerti sono ad esempio lana di roccia, cubi organici, torba, argilla espansa, perlite e vermiculite, fibra di cocco.

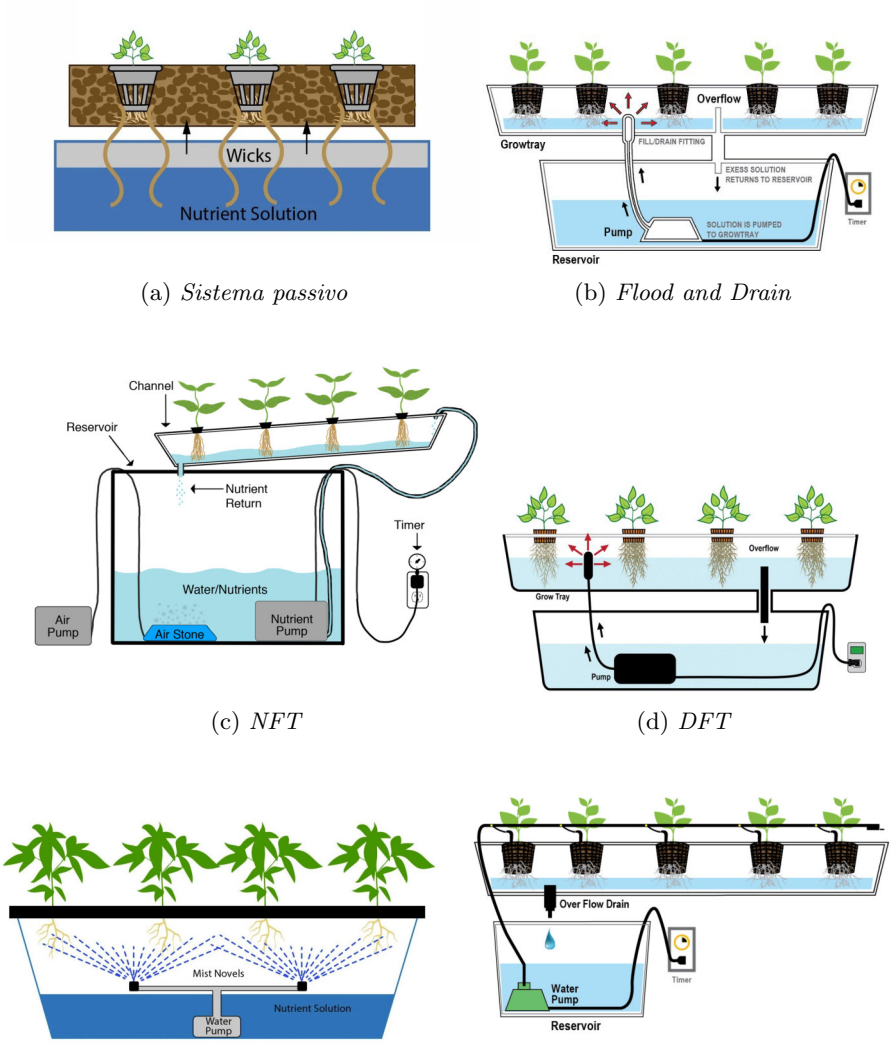
[4] Con *ciclo chiuso* intendiamo un sistema nel quale non vi sono perdite, in questo caso di acqua.

- *DFT* (*"Deep Flow Technique"*), *figura 3d.* E' un sistema di coltivazione leggermente differente da quello *NFT.* I cubi di materiale inerte, composti principalmente da fibra di cocco o lana di roccia, sono immersi in grandi vasche, poggiati su sistemi di galleggiamento in plastica o polistirolo. Questi contenitori mantengono le radici a filo con l'acqua. Appena piantate, le piante hanno radici che toccano a malapena l'acqua della vasca. La serra è sviluppata verso la direzione della linea delle vasche di immersione. Le vasche sono di lunghezza variabile, calcolata opportunamente per ospitare le piante durante il periodo necessario alla loro crescita. Le piante vengono immesse nella vasca da un lato, e mentre la percorrono nei vari giorni si sviluppano e crescono. Quando raggiungono l'altra estremità, le piante hanno raggiunto le dimensioni e la forma adeguate, e possono essere raccolte. Durante il periodo di crescita, l'apparato radicale ramifica e si immergere completamente all'interno dell'acqua. Questa tecnica viene applicata su serre particolarmente grandi. Ogni giorno vengono aggiunte tante piante vergini quante se ne raccolgono.

- *Aero-idroponica*, chiamata erroneamente anche *aeroponica*, *figura 3e.* Questo sistema è relativamente moderno. L'acqua, contenente le sostanze nutritive, si ossigena attraverso il movimento e raggiunge le piante mediante nebulizzazione. In seguito condensa a contatto con l'apparato radicale. E' un sistema a ciclo chiuso molto efficiente, che permette di rimuovere gas nocivi che stazionano nell'aria e concentrazioni saline che possono formarsi utilizzando alcune tipologie di substrato. Spesso nelle grandi serre, è utilizzato combinato con il sistema *NFT* per migliore la resa produttiva, quando la tipologia di piante coltivate lo permette.

- *Sistemi a gocciolamento*, *figura 3f.* E' il sistema maggiormente utilizzato, in quanto garantisce il giusto *trade-off* tra le varie tecniche. E' implementabile sia su scala ridotta, sia per produrre grandi volumi di prodotti. Prevede l'afflusso e l'ossigenazione dell'acqua contenete le sostanze nutritive attraverso il pompaggio lungo tubazioni, di dimensione variabile a seconda della portata della pompa, che raggiungono la zona radicale della pianta. In seguito, l'acqua raggiunge le radici nel substrato inerte attraverso gocciolamento. La lana di roccia è particolarmente adatta a questo metodo. Rappresenta inoltre un sistema facilmente regolabile e controllabile. Anche questa tecnica permette di risparmiare grossi volumi di acqua se viene implementata in un ciclo chiuso. L'unico inconveniente riguarda la formazione di concentrazioni saline nei cubetti di materiale inerte, che devono essere sciacquati attraverso cicli di irrigazione costanti in modo da non formare situazioni

pericolose per le piante, che potrebbero intaccarne la corretta crescita. *(Tutte queste informazioni sono contenute in [4]).*



(a) *Sistema passivo* (b) *Flood and Drain*

(c) *NFT* (d) *DFT*

(e) *Aero-idroponica* (f) *Sistema a gocciolamento*

Figura 3: Funzionamento delle varie tecniche di coltivazione idroponica descritte sopra [5]

3.2 Tipi di substrati per la coltivazione idroponica

Oltre alla tecnica di coltivazione utilizzata, anche il ruolo del substrato è di fondamentale importanza, in quanto influenza i cicli di irrigazione delle piante e l'accumulo più o meno marcato di sostanza nutritiva, che può causare gravi problemi alla pianta. Ostacolare l'ossigenazione della parte radicale può indurre gravi stati di sofferenza alla pianta, che possono poi sfociare fino alla sua morte. Sono molte le tipologie di substrati inerti, alcune delle quali generalmente applicabili mentre altre più specifiche per una determinata tecnica. I substrati inerti sono particolarmente adatti a questo metodo di coltivazione in quanto, a differenza del suolo agricolo classico formato da un impasto di vari elementi (riportati in *tabella 2*), permettono di avere una porosità [5] molto più elevata.

Elementi	da	a
sabbia	50%	70%
limo	25%	40%
argilla	5%	15%
altri elementi	0%	∼2%

Tabella 2: Composizione (in percentuale) degli elementi contenuti nel terreno agricolo.

Il substrato inerte garantisce una porosità che in media può variare in un intervallo da un 1/3 a 3 volte maggiore rispetto quella del suolo agricolo. Mediamente i valori si aggirano attorno al 35%. Una porosità maggiore, dunque un substrato meno compatto, ha inglobato al suo interno un maggiore volume di aria e permette dunque una migliore ossigenazione della zona radicale delle piante, influendo in modo decisamente positivo sulla crescita [6]. Inoltre, proprio grazie alla compattezza minore, le radici impiegano meno energia per radicarsi e ancorarsi; in questo modo la pianta risparmia energia che può convertire, come succede, in una crescita più rapida. Ecco perché la coltivazione idroponica offre una velocità di crescita 2 volte maggiore rispetto ai metodi classici. Questo valore però, è da intendersi in riferimento a tutta la serra, dunque inglobando anche il sistema di controllo e gestione dei parametri. Perché un materiale sia considerato un buon substrato, deve possedere alcune caratteristiche: non deve contenere una quantità eccessiva di cloruro di sodio (il normale sale da cucina), non deve contenere elementi potenzialmente nocivi per la pianta e nemmeno elementi chimici tossici,

[5] Per *porosità* intendiamo il volume di spazio vuoto nel terreno in rapporto percentuale al volume totale. Influenza direttamente la dinamica della fase liquida e aeriforme del terreno, e indirettamente la fertilità chimica.

come ad esempio i metalli pesanti, che raccolti dalle radici, si distribuiscono poi alla pianta e ai frutti che questa produce, creando un problema per l'uomo. Oltre alla composizione del substrato, i parametri chimici da tenere in considerazione sono pH, EC [6] e fitotossicità [7]. Parametri fisici sono invece densità apparente, porosità e granulometria. Il substrato deve essere il più neutro possibile, sia per quanto riguarda il pH, sia per l'EC. Tutti i parametri chimici possono essere corretti mediante l'utilizzo di opportune soluzioni nutritive per le piante. Per i parametri fisici invece, influisce la scelta del tipo di substrato. Con densità apparente, o del suolo, intendiamo una misura della sua compattezza. Valori elevati suggeriscono un suolo compatto. Al contrario invece un suolo meno compatto, con molti pori e sacche d'aria, è ideale per la coltivazione idroponica. Per tenere conto di tutti i parametri fisici assieme, si deve fare attenzione a due aspetti principali in idroponica durante la scelta del substrato: la *ritenzione idrica* e il *potenziale idrico*. Il primo esprime la quantità di acqua che un certo volume di substrato può assorbire mentre il secondo la forza che l'apparato radicale deve esercitare per assorbire l'acqua dal substrato (solitamente espresso in *Kilo Pascal*). Per avere un buon substrato si dovrebbe avere un potenziale idrico non troppo elevato in modo che la pianta riesca ad assorbire l'acqua senza troppo dispendio energetico e con una ritenzione idrica equilibrata, così da non avere problemi di possibile marcimento delle radici. Se questo valore è troppo elevato, il substrato non è in grado di trattenere l'acqua opportunamente. Meglio avere una ritenzione idrica leggermente più bassa e irrorare più frequentemente l'apparato radicale così da avere un continuo movimento di aria e acqua sulle radici, ossigenando la zona radicale e favorendo la crescita delle piante [6].

3.2.1 Substrati inorganici

- *Lana di roccia* e *lana di vetro*. La *lana di roccia* è composta da rocce basaltiche e rocce calcaree, mescolate assieme e fuse ad una temperatura di 1.600 °C mediante combustione di coke [8]. Il miscuglio ancora caldo, viene ridotto in fibre e in seguito sagomato in varie forme e misure. La *lana di vetro* ha un processo di fabbricazione simile, a differenza che le fibre possono essere ridotte in misure diverse, andando ad incidere sulla capacità di ritenzione idrica. La lana di roccia presenta però degli inconvenienti. Il più grande problema è legato al fatto che l'acqua non si diffonde in modo omogeneo dall'alto verso il basso

[6] *EC* definisce la *conducibilità elettrica*, ovvero la quantità di sali minerali disciolti nell'acqua.

[7] Per *fitotossicità* si intende l'insieme delle conseguenze negative per una pianta derivanti dall'uso di prodotti antiparassitari (prodotti usati sono per esempio il solfato di rame).

[8] Con *coke* ci si riferisce al combustibile grigio, duro e poroso, con alto contenuto di carbonio, usato in industria per la combustione per la fusione del minerale di ferro.

del cubo. Inoltre, rischia di saturare la parte inferiore del blocchetto di substrato e formare delle concentrazioni saline che danneggiano l'apparato radicale della pianta e non favoriscono il ricircolo dell'aria. Tra un'irrigazione e l'altra, la lana di roccia presenta un substrato avente quasi sempre un gradiente di umidità che va da saturo (in basso), a troppo secco (in alto), non ideale per le radici. Questo è dovuto al fatto che ha capacità di ritenzione idrica elevata ma basso potenziale idrico. Questo inconveniente è risolvibile con cicli di irrigazione frequenti regolando l'apporto di sali minerali, attraverso l'EC, in modo da evitare accumuli salini dannosi per la pianta. Il vantaggio della lana di roccia, e di vetro, è quello di poter essere riutilizzate per un secondo ciclo di coltivazione, o addirittura per un terzo in alcuni casi. Un cubo di lana di roccia si presenta nella forma rappresentata in *figura 4*.

Figura 4: Cubetto in lana di roccia compresso e sagomato, con foro centrale per inserire i semi o le piante. [7]

- *Pietre laviche.* Ne esistono diverse qualità, in base al luogo della loro raccolta. Vengono classificate in base al contenuto di silicato. Sono molto leggere e hanno una porosità elevata, che permette di rappresentarle solitamente come un buon substrato. Può presentare dei problemi con l'assorbimento, da parte della pianta, di piccole quantità di alluminio contenute naturalmente nelle pietre stesse. Questa sostanza viene poi assorbita dalle radici e finisce nella pianta o nei frutti che questa produce. Le pietre laviche si presentano nella forma e nelle dimensioni di *figura 5*.

Figura 5: Forma e dimensioni delle pietre laviche. [8]

- *Pomice.* E' un prodotto dell'attività vulcanica terrestre. Particolare tipologia di pietra lavica, ad alto contenuto di silicato (superiore al 65%), presenta una porosità elevata, dovuta al processo di formazione vulcanica che la forma. Data l'alta porosità, ha anche un peso contenuto e dunque è molto leggera. Utilizzata assieme ad altri substrati, come ad esempio la fibra di cocco, migliora le sue caratteristiche chimiche e fisiche. Viene solitamente posta sul fondo dei contenitori per migliorare il drenaggio e riduce il ristagno idrico. La pomice si presenta nella forma e nelle dimensioni di *figura 6.*

Figura 6: Forma e dimensioni della pomice, una particolare tipologia di roccia lavica. [9]

- *Perlite.* Anche questo è un substrato di origine vulcanica, ad alto contenuto di acqua. La sua produzione prevede prima una frantumazione del materiale e poi un processo di evaporazione dell'acqua ad una temperatura di circa 1.000 °C. In questo modo, l'espansione del vapore fa aumentare il volume delle rocce da 4 a 20 volte, producendo un materiale altamente poroso. Anche questa, dovuta all'origine vulcanica e al processo di produzione, è una pietra leggera ed estremamente

porosa. Ha inoltre una capacità di ritenzione idrica molto alta. L'unico inconveniente è quello legato all'estrema leggerezza che presenta. Molto spesso porta il materiale a galleggiare sull'acqua, spostandosi dalla posizione iniziale. Questo rischia di lasciare completamente o parzialmente scoperti alcuni apparati radicali delle piante e di intasare tubazioni o pompe. La perlite si presenta nello stato raffigurato dalla *figura 7*.

Figura 7: Forma e dimensione della perlite, lavorata e frantumata in pezzi piccoli. [10]

Materiale	densità apparente	porosità
Lana di roccia	0,1 g/cm^3	elevata (fino a 98%)
Lana di vetro	0,15 g/cm^3	elevata (fino a 95%)
Pietra lavica	1 g/cm^3	elevata (fino a 80%)
Pomice	$<$1 g/cm^3	elevata (da 98% a 99%)
Perlite	$\sim$1 g/cm^3	elevata (95%)

Tabella 3: Caratteristiche dei substrati inorganici - prima parte.

Materiale	pH	EC
Lana di roccia	neutro (5,5)	0,4÷0,5
Lana di vetro	acido/neutro (5)	0,2÷0,4
Pietra lavica	neutro (7÷8)	
Pomice	neutro (<6)	
Perlite	neutro (7÷7,5)	

Tabella 4: Caratteristiche dei substrati inorganici - seconda parte.

Solo per citarne altri, però meno utilizzati rispetto a quelli descritti in precedenza, troviamo la *vermiculite*, un minerale argilloso contenente acqua. Il processo prevede una cottura che, grazie all'evaporazione dell'acqua, permette di ottenere un materiale finale molto leggero e con densità molto bassa. Altro substrato, sostituto della perlite, può essere la *ghiaia*, più facilmente reperibile rispetto a quest'ultima e che offre una buona soluzione a basso costo. Ultimo materiale da menzionare è la *sabbia*. Si trova in grandi quantità, ha un costo contenuto, è leggera ed ha pH neutro. D'altro canto, data la grandezza molto ridotta, può rappresentare un ostacolo per serre idroponiche che utilizzano pompe se sprovviste di appositi filtri, come nelle serre a basso costo o fai da te.
(Informazioni contenute in [6].

3.2.2 Substrati organici

- *Fibra di cocco.* Questo substrato è ricavato dai gusci delle noci di cocco separandone la fibra. Di fatto rappresenta uno "scarto di lavorazione", che ottiene una nuova vita e un nuovo utilizzo. Derivando da sostanze organiche, può presentare parametri chimici variabili. Per questo motivo è impossibile calcolare dei valori medi. Il più adatto sarà certamente quello con pH maggiormente neutro e con EC più basso. E' un ottimo substrato, da paragonare senza alcun problema con quelli inorganici. Inoltre, essendo organico, facilita l'introduzione di microrganismi benefici alle piante. Purtroppo, come tutti i tipi di substrato, presenta anche alcuni difetti: trattiene troppo l'umidità e può portare, come già detto per i substrati inorganici, al marcimento delle radici. La fibra di cocco si presenta nella consistenza raffigurata dalla *figura 8*.

Figura 8: Pannelli in fibra di cocco per coltivazione idroponica. [11]

- *Segatura.* E' un tipo di substrato che non viene mai usato da solo, ma solitamente abbinato a delle miscele. Quella adatta è ricavata dal legno duro, e viene trattata per evitare il rischio di fitotossicità. E' un substrato leggero a bassa densità, dunque particolarmente adatto per l'ossigenazione dell'apparato radicale. Ha scarsa ritenzione idrica purtroppo e dunque richiede irrigazioni frequenti, specialmente in ambienti molto caldi. Il pH è generalmente neutro e ha un basso valore di EC. Altro vantaggio, come per la *fibra di cocco*, è di essere un derivato da origine organica. La segatura ha varie dimensioni e colori, a seconda della pianta, e presenta anche caratteristiche chimiche diverse, ma quella usata nella coltivazione si presenta solitamente come in *figura 9.*

Figura 9: Scarti della lavorazione del legno riutilizzati in coltivazione moderna come substrato. [12]

Altro substrato di origine organica da citare, senza entrare troppo nel dettaglio, è la *torba*. Viene ricavata da bacini riempiti di acqua poco profondi. Ha caratteristiche abbastanza variabili, a seconda del tipo di materiali contenuti nel bacino e dal luogo geografico dove si trova. Viene spesso associata ad altri substrati inorganici per migliorarne le caratteristiche offerte. *(Informazioni contenute in [6].*

3.3 Parametri di controllo

I parametri di controllo della serra idroponica possono essere molti, e molte volte anche difficili da rilevare o gestire. A seconda del livello tecnologico della serra, possono essere gestiti in modo automatico, come nelle serre più *hi-tech*, o in modo manuale, come nelle serre più "artigianali". Riducendoci ai più importanti, possiamo suddividerli in due macro gruppi: i *parametri ambientali*, legati alle piante e a quello che ricevono dall'ambiente, e i parametri legati alla *soluzione nutritiva*, dunque la composizione di sali e minerali fornita.

3.3.1 Parametri ambientali

Per quanto riguarda i *parametri ambientali*, sono legati alla serra stessa, e includono fattori come *temperatura* e *umidità*, *tempo di esposizione alla luce* delle piante, *ciclo di irrigazione* e *ventilazione*. Sembrano di facile gestione nel complesso, ma in realtà richiedono un'attenta calibrazione. Partendo innanzitutto dalla *temperatura* e dall'*umidità*, queste vengono regolate a seconda dello stato di crescita della pianta. Come erroneamente si pensa, le piante non hanno bisogno sempre dello stesso clima per crescere ma anzi, opportune soglie possono invece indurre nella pianta stessa il passaggio da una fase di crescita all'altra. Si pensi ad esempio ad una pianta nella fase precedente alla fioritura. In natura, è il clima esterno che induce nella pianta il segnale, nel momento opportuno, per fiorire. Se controllate opportunamente, come fatto in idroponica e in tutte le altre tecniche di coltivazione, è possibile rendere artificiale questo processo e risparmiare tempo e denaro. Le fasi della pianta possono essere pensate in sequenza come segue:

1. Germinazione;
2. Talea;
3. Pianta radicata;
4. Fase vegetativa;
5. Fioritura/fruttificazione;
6. Fase finale.

Ognuna di queste fasi, a seconda della pianta coltivata, richiede specifici valori di temperatura ed umidità. Dovranno nel tempo essere opportunamente cambiati, poiché la pianta impiegherà del tempo per crescere e svilupparsi. Una temperatura che in media va bene per quasi tutte le piante, si aggira attorno ai $21 \div 28$ °C (ma è comunque specifica a seconda della pianta). L'umidità invece, influenza la formazione delle piante in serra. Ambienti con

tasso di umidità [9] elevato permettono di sviluppare foglie più grandi rispetto ad ambienti secchi. Esperimenti condotti da ricercatori hanno dimostrato che la crescita è massima per valori di umidità che si aggirano tra il 60% e l'80%. Le talee però, a conferma di quanto detto prima, richiedono un tasso più elevato, circa del 90%. Per la fase finale invece, il tasso di umidità da mantenere è consigliabile non superi il 50%, per ridurre la formazione di muffe.
Anche il *tempo di esposizione alla luce* influenza la crescita della pianta, oltre ad influenzarne il "benessere". Come per tutte le forme di vita, anche le piante possono subire degli stati di sofferenza che le portano ad un non corretto sviluppo. Infatti, a differenza del pensiero diffuso, non necessitano di luce 24 ore al giorno, ma solo in opportune finestre giornaliere e per un preciso tempo. E' proprio durante il riposo, dunque in assenza di illuminazione diretta o indiretta, che la pianta cresce e si sviluppa. Inoltre, nello spettro luminoso, solo alcune bande sono utili alla pianta, il *blu* e il *rosso*. Lo spettro blu serve alla pianta durante tutta la fase di crescita poiché ne stimola lo sviluppo, mentre lo spettro rosso serve in particolare durante la fase di fioritura/fruttificazione. Altro spettro con caratteristiche simili a quelle del rosso, seppur in dose minore, è il *giallo*. Come detto in precedenza, anche le ore di esposizione giornaliera alla luce sono importanti. Nella fase dello sviluppo vegetativo della pianta, questo valore deve essere mantenuto attorno alle 18 h/g. Nella fase di fioritura/fruttificazione invece, per i primi $2 \div 3$ gg il valore scritto sopra, per poi passare a 12 h/g nell'ultima parte del processo. Non basta solo limitare le ore di luce perché la pianta cresca poiché è fondamentale avere delle ore di buio *continuative*. Infatti, è proprio in questo periodo continuo che si innescano i fenomeni di maturazione della pianta che la portano a crescere. Dunque, nell'arco della giornata, il periodo di riposo della pianta e quello di esposizione alla luce dovranno essere continui. Per questo motivo, in assenza di luce naturale, si dovrà provvedere a fornire luce artificiale alle piante.
Altro aspetto è legato al *ciclo di irrigazione*. Non incide così nettamente come i parametri elencati sopra, ma dipende in particolare dalla portata d'acqua dell'impianto e dal tipo di substrato scelto. Deve essere opportunamente regolato in modo da prevenire concentrazioni saline che possano portare al marcimento dell'apparato radicale della pianta o il contrario, ovvero la morte per mancanza di acqua. Dovrà quindi essere opportunamente calibrato, il più delle volte in modo pratico agendo direttamente sui tempi del ciclo di irrogazione.
Ultima, ma non per importanza, è la *ventilazione*. Permette di mantenere la concentrazione di CO_2 nella serra relativamente costante. La percentuale di questo gas assimilata dalle piante ne influenza il modo e la velocità di

[9] Quando si parla di *tasso di umidità* si intende il tasso di *umidità relativa* infatti, a seconda della temperatura dell'aria, questa può contenere più o meno acqua.

crescita. Il ricambio d'aria favorisce un apporto costante del mix di gas ed è perciò importantissimo. Inoltre la ventilazione permette di mantenere basso il livello di umidità che, se presenta valori molto elevati, riduce la fotosintesi delle piante (in particolare durante stadi di crescita che richiedo valori più contenuti di quest'ultimo parametro).

3.3.2 Parametri della soluzione nutritiva

La soluzione nutritiva, che verrà trattata in dettaglio nel capitolo successivo, ha parametri di controllo che permettono di verificare l'assunzione dei minerali da parte della pianta. Per la corretta crescita, la pianta deve assumere quantità adeguate di sali e minerali. I parametri di controllo sono il *pH* e l'*EC*. Il pH definisce il valore di acidità/basicità della soluzione nutritiva. Spazia in una scala tra 0 e 14, come raffigurato in *figura 10*, e per valori bassi indica una acidità della soluzione nutritiva sempre più elevata mentre per valori alti, al contrario, una sua basicità.

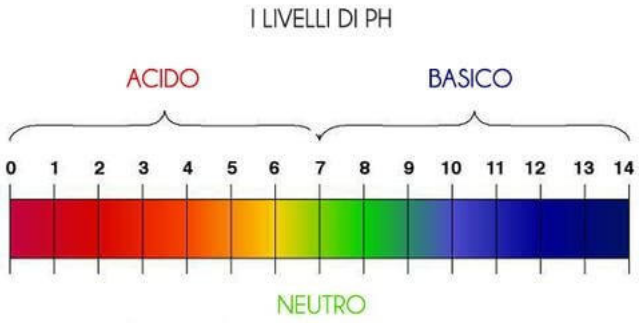


Figura 10: Scala del pH usato in chimica per definire la acidità/basicità di un liquido o di un gas.

Il pH ideale in idroponica si aggira tra 5,8÷6,2, dunque leggermente acido. Il pH della soluzione nutritiva può essere influenzato anche dal tipo di substrato utilizzato, e dunque la calibrazione dovrà essere fatta tendendo conto anche di questo aspetto. Buona norma è quella di iniziare a somministrare una soluzione nutritiva con un pH leggermente più basso di quello ideale in quanto, successivamente, tenderà ad aumentare. Per esempio, un pH di partenza di 5,5 può andare bene. In seguito, dovrà essere costantemente controllato e al raggiungimento del valore di 6,5 abbassato. Valori troppo alti o bassi di questo parametro possono danneggiare gravemente l'apparato radicale e dunque la pianta stessa. Altro aspetto importante è quello di variare lentamente il pH della soluzione nutritiva, in modo da non recare danno alle radici. Per la misurazione del valore di pH della soluzione nutritiva, possono essere utilizzati sensori collegati al controllo automatico della serra, se provvista, oppure adottando soluzioni più economiche come

ad esempio misuratori analogici (cartine tornasole) o digitali, come quello raffigurato in *figura 11*.

Figura 11: Strumento digitale per la misurazione del pH nei liquidi.

Oltre al pH, è importante mantenere controllato anche il valore di EC. Questo parametro definisce la quantità di sali contenuti nella soluzione nutritiva. Cambia a seconda dello stato di crescita della pianta, del substrato usato e dalle stagioni (anche se in modo indiretto). A seconda dello stato di crescita delle piante, il valore dovrà aggirarsi all'interno degli intervalli riportati in *tabella 5*.

EC	fase crescita pianta
0,2÷0,4 mS	Talea
0,8÷1,2 mS	Piante appena radicate
1,6÷1,8 mS	Fase vegetativa
1,8÷2,2 mS	Fioritura/Fruttificazione
2,4÷2,6 mS	Fase finale

Tabella 5: Valori medi di EC per ogni fase di crescita della pianta.

Anche se in modo indiretto, a seconda delle stagioni si deve intervenire su questo valore: in estate si devono assumere come intervalli ideali dei valori inferiori a quelli consigliati per ogni fase di crescita, mentre in inverno valori leggermente superiori. Per variare in modo semplice questo parametro, si può aggiungere acqua distillata durante la preparazione della soluzione nutritiva per abbassare il valore di EC oppure riducendo la quantità di acqua della miscela, per alzarlo. Valori di EC maggiori permettono una crescita di piante maggiormente robuste: per questo motivo nel periodo invernale è consigliabile assumere valori maggiori a quelli medi. Per la misurazione, come nel caso del pH, si possono installare sensori collegati con il programma automatico della serra. Una soluzione più economica è l'utilizzo di semplici

strumenti digitali come quello riportato in *figura 12*.

Figura 12: Strumento digitale per la misurazione del valore di EC nei liquidi.

(Informazioni contenute in [13].

3.4 Ossigenazione dell'apparato radicale e della soluzione nutritiva

Questo aspetto è molto delicato e contiene poche informazioni a riguardo. Nei vari tipi di substrato è stato spiegato che l'ossigenazione dell'apparato radicale della pianta, e della soluzione nutritiva, è di fondamentale importanza per una crescita rapida e ottimale delle piante. Maggiore sarà il movimento che riceve l'acqua prima di raggiungere le radici, e mentre tocca le radici stesse, maggiore sarà l'ossigenazione offerta sia alla soluzione che all'apparato radicale. La scelta del tipo di substrato è di fondamentale importanza quindi. La soluzione nutritiva è un concentrato di acqua, carbonati e sali minerali disciolti, almeno appena inserita nel ciclo di irrigazione della serra. Di fatto, data la presenza di tutti gli elementi necessari per la formazione di organismi, ben presto questi inizieranno a moltiplicarsi e modificare la composizione della soluzione stessa. Se i parametri rimangono dentro i limiti imposti, batteri e microbi che si formeranno saranno funzionali alla crescita della pianta al contrario, si dovrà provvedere alla purificazione della soluzione. Questo perché se il mix di organismi ed elementi non è più controllato, potrebbero nascere problemi per le piante. Anche il tipo di substrato usato può presentare problemi o benefici sulla formazione di una microflora interna alla soluzione. Le radici stesse della pianta rilasciano composti nella soluzione nutritiva, soprattutto CO_2, H^+, anioni organici, acido citrico e nitriti. L'apparato radicale si comporta come ricevente e mittente di sostanze da e verso l'acqua. Oltre alla composizione, anche la *temperatura della soluzione* è sicuramente importante, ma lo è ancor di più la *temperatura* nei pressi della *zona radicale*. A seconda di questo valore varia la quantità di ossigeno disponibile per le radici della pianta:

Temperatura	Quantità ossigeno disciolto
20°C	9,5 mg/L
30°C	7,6 mg/L

Tabella 6: Quantità media di ossigeno contenuta nella soluzione a diverse temperature (valori calcolati in acqua pura).

Come riportato in *tabella 6*, per valori elevati di temperatura la quantità di ossigeno cala bruscamente, danneggiando la crescita della pianta. Inoltre, la temperatura elevata accelera il metabolismo delle piante ed il consumo di ossigeno nella zona radicale. Questo rappresenta un problema poiché le piante reagiscono all'aumento della temperatura chiudendo gli stomi [10] per conservare l'acqua e quindi arrestando la crescita. Il compromesso è quello di avere una temperatura intermedia compresa tra una bassa, che comporta più ossigeno nella zona radicale e un metabolismo più lento, e una alta, con meno ossigeno disponibile e un metabolismo più veloce. Per mantenere ad una temperatura accettabile la soluzione nutritiva, si può far uso di scambiatori di calore oppure più semplicemente rinfrescando l'aria, in modo particolare quella localizzata nella zona dell'apparato radicale delle piante. Come descritto nella sezione precedente, pH ed EC della soluzione vanno controllati periodicamente. La sostituzione della soluzione in media deve essere effettuata ogni 12-15 gg, in modo da evitare la formazione di troppi microrganismi e di batteri non benefici alle piante. Sembra strano, ma il metodo migliore per capire cosa serve ad una pianta è guardarla: dall'aspetto che presenta è possibile capire cosa le serve. Purtroppo l'esperienza matura con il tempo, dunque non sempre si intraprendono le scelte corrette.
(Tutte queste informazioni sono contenute in [14].

[10] Gli *stomi* sono formati da due cellule, presenti negli organi aerei delle piante. Consentono lo scambio gassoso fra interno ed esterno del vegetale.

4 Implementazione della serra

Come descritto nella sezione precedente, la coltivazione idroponica garantisce numerosi metodi di implementazione. Sarà comunque sempre l'utente finale che dovrà beneficiare della serra a decidere quale è la migliore tecnica da adottare tenendo conto del tempo, da poter dedicare alla serra, e il costo stesso. Infatti, una serra gestita automaticamente richiederà meno tempo da dedicare alle piante, essendo tutto controllato, ma un costo maggiore rispetto a una tecnica meno avanzata. Nel progetto sviluppato, si è cercato di conciliare il beneficio dell'automatizzazione del processo di controllo della serra con un costo relativamente contenuto. Per il momento è solo un prototipo, con dimensioni ridotte, realizzato per testare e calibrare i parametri per poi poter applicare il controllo e la gestione in una serra con dimensioni maggiori. Per riuscire in breve tempo a registrare i risultati, si è scelta la pianta di *basilico* (non commestibile). A differenza di quello ad uso alimentare, questo cresce più velocemente e permette una raccolta dei dati nel breve periodo. Caratteristica importante della serra è l'implementazione di questa solo ad uso esclusivamente *indoor*.

4.1 Tecnica di coltivazione utilizzata

Il primo passo è la scelta del metodo di coltivazione. Nel capitolo 3 sono state elencate molte tipologie, più o meno adatte allo scopo. Le tecniche alle quali si poteva ricorrere per lo sviluppo del progetto erano:

- *NFT* (*"Nutrient Film Technique"*);
- *DFT* (*"Deep Flow Technique"*);
- *Aero-idroponica*;
- *Sistemi a gocciolamento*.

I primi tre metodi elencati richiedono una volume minimo di spazio interno alla serra nettamente maggiore di quello a disposizione per il progetto. La scelta è ricaduta dunque sul *sistema a gocciolamento*, maggiormente scalabile e che garantisce comunque degli ottimi risultati. Infatti, questa tecnica è usata anche nei metodi di coltivazione classici, garantendo principalmente un risparmio considerevole di acqua. Oltre alla scalabilità, permette di sviluppare un sistema di controllo che ne permetta la gestione sia a livello hardware che software. Questa tecnica fa uso di gocciolatori [11] (*figura 13*). Grazie al gocciolamento dell'acqua e della soluzione nutritiva, garantisce

[11] I *gocciolatori* sono delle particolari valvole con regolazione del flusso di liquido che gli attraversa. Sono in grado di garantire in uscita un gocciolamento del liquido che arriva, attraverso flusso continuo, dall'ingresso.

un'ossigenazione dell'apparato radicale della pianta, favorendone la crescita. Operando in un sistema chiuso, garantisce un ulteriore risparmio d'acqua.

Figura 13: Schema di un gocciolatore usato nella serra: presenta due canali di ingresso/uscita del liquido, a seconda del collegamento e una sola uscita a goccia con valvola arancione per la regolazione del flusso.

4.2 Substrato utilizzato nella serra

Come descritto nel *capitolo 3*, il tipo di substrato influenza la crescita delle piante. Il comportamento nei confronti dell'apparato radicale della pianta e verso la soluzione nutritiva sono differenti a seconda del substrato. Infatti, anche la soluzione diluita in acqua per fornire alle piante i sali e gli elementi per la loro crescita può subire della variazioni a contatto con questo. Per il progetto implementato, si è scelto di usare la *lana di roccia*, facilmente reperibile nel mercato con un costo contenuto e riutilizzabile per più di un ciclo di coltivazione. Anche le caratteristiche sono buone: è un substrato leggero, con elevata porosità e un pH neutro, ideale per la crescita delle piante. L'unico inconveniente è quello di asciugarsi velocemente e formare accumuli salini, nella zona sul fondo del cubetto (circa 1 cm), come raffigurato in *figura 14*. E' inoltre consigliato come substrato possibile per i sistemi a *gocciolamento*. Richiede attenzione in modo particolare quando è asciutto in quanto potrebbe provocare, in alcuni casi, delle leggere irritazioni cutanee.

Figura 14: Cubetto in lana di roccia con raffigurate le zone problematiche.

Per questo substrato, le concentrazioni saline possono risultare un problema. La soluzione è quella di prevedere cicli irrigui più frequenti e con soluzione meno concentrata, in maniera da razionare i nutrimenti contenuti nell'acqua su più volte. La quantità di sali e minerali richiesti dalla pianta saranno comunque soddisfatti, con l'ulteriore beneficio di non stressare troppo la pianta stessa. Come ulteriore rimedio alle concentrazioni saline che si formano e al problema del potenziale idrico basso, è stato posto del *materiale inerte* sui lati e sul fondo del contenitore contenente i cubi in fibra di roccia. In questo modo, si migliora l'ossigenazione dell'apparato radicale della pianta, si aumenta il potenziale idrico e si crea una protezione contro la formazione di concentrazioni saline troppo fitte. Per il materiale inerte è stata scelta la *perlite* perché offre ottime caratteristiche. E' leggera, grazie alla porosità interna, può trattenere l'acqua, ma comunque non da presentare degli accumuli salini, e ha pH neutro (*tabella 4*). Per queste caratteristiche, è spesso usata assieme ad altri substrati per migliorarne le caratteristiche. I benefici ottenuti abbinando i due substrati sono notevoli. La zona superiore del cubo di lana di roccia, che per caratteristica tende ad asciugare velocemente, resta in realtà umida grazie all'acqua inglobata nelle cavità porose della perlite stessa. La zona sottostante del cubetto invece, non toccando direttamente il fondo della serra e dunque non essendo al livello dell'acqua, accumula meno concentrazione salina rilassando l'apparato radicale della pianta. Questo semplice *trade-off* permette di ottenere un netto miglioramento in termini di crescita delle piante, che si avvicina a un 40÷50%. Il contenitore per le piante si presenta nel complesso come in *figura 15*.

Figura 15: Contenitore usato nella serra per la coltivazione delle piante.

I cubi di lana di roccia hanno dimensioni standard che si aggirano attorno a 25x25x40 mm. I contenitori hanno diametri interni che variano dai 50 mm della sommità ai 35 mm del fondo. I vincoli legati alla dimensione dei contenitori, utilizzati per ospitare il cubo in lana di roccia, richiedono l'utilizzo di un substrato inerte differente dalla perlite, con un potenziale idrico inferiore. I benefici, introdotti in precedenza, passano dal 40÷50% a un 15÷20%. Per dimostrare i benefici della lana di roccia, rispetto a un suolo classico, è stato realizzato un semplice test di crescita di una pianta di basilico. Il test prevedeva la sola somministrazione di acqua, senza minerali e nutrimenti, e il confronto sulla crescita tra una pianta coltivata in lana di roccia e una in suolo classico. I semi sono stati piantati in data 30/03/2022, come riportato nelle etichette raffigurate in *figura 17*. In figura *16* possiamo notare che lo stato di crescita iniziale delle piante di basilico non è lo stesso, ma è più rapido nella pianta nel cubetto di lana di roccia.

(a) *Basilico appena germogliato coltivato in lana di roccia.*

(b) *Basilico appena germogliato coltivato in terreno comune.*

Figura 16: Differenze di crescita per la pianta di basilico coltivata in due substrati differenti.

Le radici incontrano meno ostacoli durante lo sviluppo dell'apparato radicale rispetto ad un terreno classico, e dunque possono concentrare maggiormente l'energia sulla crescita della pianta. Fino a circa il mese di maggio 2022, la crescita era notevolmente più marcata in quella piantata nel cubo di lana di roccia. In seguito, a causa della mancanza di nutrimenti però, la pianta cresciuta in terreno classico ha manifestato una crescita maggiore. Si ricorda che il test prevedeva la sola somministrazione di normale acqua. Un terreno classico, per la sua composizione, presenta maggiori nutrimenti di un cubo di lana di roccia allo stato naturale, senza apporto dall'esterno. Infatti la lana di roccia, chimicamente neutra, non contiene gli elementi e gli organismi contenuti dal normale terreno usato in agricoltura, quindi non riesce a fornire alla pianta il nutrimento che ha bisogno per svilupparsi. Proprio per questo, dopo la radicazione iniziale, la pianta di basilico nel terreno classico ha trovato maggiori nutrimenti per la sua crescita. Dopo circa 3 mesi, in data 02/07/2022 le piante si presentano come in *figura 17*:

(a) *Basilico coltivato in lana di roccia.*

(b) *Basilico coltivato in terreno comune.*

Figura 17: Differenze sulle dimensioni di crescita dopo circa 3 mesi.

Dalla *figura 17b* è possibile estrapolare altre utili informazioni, spesso sottovalutate. Paragonando i due metodi, osserviamo come il basilico cresciuto in lana di roccia presenti un fondo del contenitore nettamente più pulito rispetto a quello cresciuto in terreno classico. E' la conferma che i benefici igienico-sanitari ottenibili con questa tecnica di coltivazione, di cui si è accennato nel capitolo introduttivo, sono facilmente osservabili anche con delle semplici prove.

4.3 Soluzione nutritiva

Come per il tipo di substrato e la tecnica di coltivazione, anche la soluzione nutritiva è un elemento fondamentale in idroponica. In tutte le coltivazioni gli elementi, forniti alle piante in modo liquido o solido, ne influenzano la crescita. La pianta raccoglie ciò che le serve dal terreno, attraverso le radici. In idroponica, gli elementi necessari alla crescita e allo sviluppo della pianta vengono immessi nella soluzione nutritiva, miscelati ad acqua. I vari rivenditori nel mercato forniscono soluzioni già pronte o miscelabili a seconda delle esigenze. I prodotti sono specifici per le varie fasi di crescita e adatti a rispondere alle esigenze della pianta. I prodotti scelti sono dell'azienda *"BioBizz"*, leader nei prodotti per la coltivazione idroponica e non solo. Fornisce soluzioni nutritive completamente biologiche e ad uso alimentare, che ricoprono gli stati di crescita della pianta dalla germogliazione fino alla fioritura. Per la coltivazione di basilico, sono stati scelti *tre prodotti*: due di questi fertilizzanti liquidi, adatti per la fase di crescita della pianta e per la fase di fioritura, e uno stimolante che permetta alla pianta di avere un aumento di dimensioni e di peso nei racemi floreali [12] della pianta. Per questi prodotti, è sconsigliato l'uso combinato. Ogni prodotto fornisce una tabella programmata sulla quantità di soluzione da utilizzare, per ogni litro di acqua, a seconda della fase di crescita della pianta (*figura 18*). Inoltre, sono concentrati che mantengono il pH costante al valore ottimale di crescita per la pianta. Se eventualmente, a causa dell'acqua utilizzata per la preparazione della soluzione, il pH dovesse scendere o salire, è possibile utilizzare prodotti dedicati per regolarne nuovamente il livello.

[12] Il *racemo floreale*, detto anche *grappolo*, è un'infiorescenza, ovvero un raggruppamento di rami che portano fiori, semplice, non ramificata, lungo una certa direzione nello spazio.

BIOBIZZ
WORLD WIDE ORGANICS

PRODUCT DATA SHEET

Original
BIO·GROW®

CERTIFICATI

FERTILIZZANTI LIQUIDI

COS'È BIO·GROW®?

Si può usare questo fertilizzante liquido per la crescita con la maggior parte delle miscele di terriccio e substrato. Ha come base il 100% di estratto di barbabietola da zucchero biologica (noto anche come borlanda). Questo sottoprodotto derivato dalla trasformazione di canna da zucchero in zucchero è il risultato del processo di fermentazione naturale. Gli amidi nello zucchero sono poi trattati con enzimi che producono glucosio, il quale insieme ad altri nutrienti vegetali, favorisce la produzione di fonti ricche di nutrimento per i microbi del suolo. Così la popolazione microbica cresce e il suolo diventa più prospero.

Gli zuccheri e il potassio naturale di Bio•Grow® attivano la flora batterica del substrato, e garantiscono raccolti generosi di frutti dolci e deliziosi.

FLACONI

250ML, 500ML, 1L, 5L, 10L, 20L

AMBIENTE	FASE	FUNZIONE
INTERNO	CRESCITA	NPK
ESTERNO	FIORITURA	NUTRIMENTO

USO

Bio•Grow® è un fertilizzante completo da usare per l'intero periodo della crescita. Da usare non appena appaiono le prime foglie fino a che la pianta non avrà raggiunto i 10-15 cm di altezza. Continua poi a usare Bio•Grow® fino alla fine della frutticoltura.

APPLICAZIONE

Acqua

Annaffiatura Classica

Dosaggio

1-4 ml per litro d'acqua

PRECAUZIONI E AVVERTENZE

Tenere fuori dalla portata dei bambini.
Non ingoiare.
Utilizzare le seguenti istruzioni.

SCHEDA PRINCIPI NUTRITIVI

Segui questa tabella quando usi **ALL·MIX*** * Può essere utile sostituire Bio·Grow®, con Fish·Mix™ durante la fase vegetativa. Passa a Bio·Grow®, quando vuoi attivare la fioritura.

	ST 1	ST 2	ST 3	ST 4	ST 5	ST 6	ST 7	ST 8	ST 9	ST 10	ST 11	ST 12
*BIO·GROW		1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua	1 ml/lt d'acqua		

Segui questa tabella quando usi **LIGHT·MIX*** o **COCO·MIX*** Essenziale per la coltivazione su fibra di cocco: regolare il pH & usare un prodotto enzimatico (es. Acti-Vera®)

	ST 1	ST 2	ST 3	ST 4	ST 5	ST 6	ST 7	ST 8	ST 9	ST 10	ST 11	ST 12
*BIO·GROW		2 ml/lt d'acqua	2 ml/lt d'acqua	2 ml/lt d'acqua	3 ml/lt d'acqua	3 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua		

FASE VEGETATIVA — FASE DI FIORITURA — LAVARE CON ACQUA — RACCOLTO

Polígono Industrial Lezama-Leguizamon C/ Gorbela nº 11-21 (48450) Etxebarri – Bizkaia Spain
Tel: +34 944 657 951 / sales@biobizz.com

IT

(a) *"Bio Grow" per la fase di crescita della pianta.*

BIOBIZZ
WORLD WIDE ORGANICS

PRODUCT DATA SHEET

Original **BIO·BLOOM™**

CERTIFICATI

FERTILIZZANTI LIQUIDI

COS'È BIO·BLOOM™?

Speri di avere fiori rigogliosi e frutti deliziosi? La risposta sta nel nome, con questo completo fertilizzante biologico liquido. Contiene la miscela ideale di azoto, fosforo e potassio, oltre ad enzimi e aminoacidi, in perfetta sintonia per lavorare al meglio il suolo. Oligoelementi e ormoni di origine vegetale sono stati aggiunti al mix per favorire la crescita di piante dal lungo stelo, forti e dalla fioritura rigogliosa.

Gli ingredienti nutritivi di Bio·Bloom™
Potassio e fosforo favoriscono la formazione e la sana crescita di bulbi, calici e petali. Il potassio in particolare lavora in sintonia con il naturale ritmo della pianta, di giorno e di notte, agevolando il processo di fioritura.

FLACONI

250ML, 500ML, 1L, 5L, 10L, 20L

AMBIENTE	FASE	FUNZIONE
INTERNO	FIORITURA	NPK
ESTERNO		NUTRIMENTO

USO

Bio•Bloom™ è un fertilizzante ad azione individuale, che può essere usato dalla fioritura alla raccolta. Aggiunto al terriccio o per integrare metodi fuori suolo come Coco•Mix™ che è composto da fibre di cocco. Biobizz® raccomanda una quantità di 2-4 ml Bio•Bloom™ per 1 litro d'acqua circa. Per risultati ottimali, segui le istruzioni nella tabella nutritiva sulla confezione.

APPLICAZIONE

Acqua

Annaffiatura Classica — Idroponica

Sistema di irrigazione

Dosaggio

1-4 ml per litro d'acqua

PRECAUZIONI E AVVERTENZE

Tenere fuori dalla portata dei bambini.
Non ingoiare.
Utilizzare le seguenti istruzioni.

SCHEDA PRINCIPI NUTRITIVI

Segui questa tabella quando usi **ALL·MIX**

** Per ogni sistema: terriccio, idroponico & aeroponica.

	ST 1	ST 2	ST 3	ST 4	ST 5	ST 6	ST 7	ST 8	ST 9	ST 10	ST 11	ST 12
**BIO·BLOOM			1 ml/lt d'acqua	2 ml/lt d'acqua	2 ml/lt d'acqua	3 ml/lt d'acqua	3 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua		

Segui questa tabella quando usi **LIGHT·MIX** o **COCO·MIX**

Essenziale per la coltivazione su fibra di cocco: regolare il pH & usare un prodotto enzimatico (es. Acti-Vera®)

	ST 1	ST 2	ST 3	ST 4	ST 5	ST 6	ST 7	ST 8	ST 9	ST 10	ST 11	ST 12
**BIO·BLOOM			1 ml/lt d'acqua	2 ml/lt d'acqua	2 ml/lt d'acqua	3 ml/lt d'acqua	3 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua	4 ml/lt d'acqua		

FASE VEGETATIVA — FASE DI FIORITURA — LAVARE CON ACQUA — RACCOLTO

Polígono Industrial Lezama-Leguizamon C/ Gorbela nº 11-21 (48450) Etxebarri – Bizkaia Spain
Tel: +34 944 657 951 / sales@biobizz.com

IT

(b) *"Bio Bloom" per la fase di fioritura della pianta.*

(c) *"Top Max" per la stimolazione durante la fase di fioritura della pianta.*

Figura 18: Tabelle per il periodo d'uso e somministrazione dei prodotti "BioBizz".

I test di crescita eseguiti hanno fatto emergere che seguire le tabelle fornite, permette una crescita costante e omogenea, con valori di pH ed EC sempre all'interno degli intervalli ottimali, senza ricorrere dunque a prodotti ulteriori per la variazione delle caratteristiche chimiche della soluzione nutritiva. Il programma di somministrazione dei prodotti è il seguente:

Giorni	Prodotto	Quantità soluzione	Funzione
da 0 a 4 gg	acqua pura	-	radicazione
da 5 a 45 gg	Bio Grow	1 ml/lt acqua	cresita
da 46 a 53 gg	Bio Bloom	1 ml/lt acqua	fioritura
da 54 a 60 gg	Bio Bloom	2 ml/lt acqua	fioritura

Tabella 7: Quantità e modo di utilizzo dei prodotti durante la crescita (in giorni) della pianta di basilico.

La tabella è comunque indicativa. Si deve in realtà sempre guardare le piante e capire di cosa hanno bisogno. Solitamente, il prodotto per la crescita delle piante, dunque il "Bio Grow", viene usato fino a quando la pianta non raggiunge i 10-15 cm di altezza. A seconda della stagione e del clima il periodo di crescita può variare. In seguito, viene stimolata la fioritura con il prodotto "Bio Bloom", con dosi sempre crescenti. Nel caso si voglia stimolare sin da subito la pianta, già dalle prime settimane di coltivazione è possibile aggiungere alla soluzione fino a 1 ml di prodotto "Top Max", per rafforzare la pianta e permetterle di generare fiori più grandi, ed eventualmente frutti più dolci e saporiti. Sono molte le soluzioni possibili, ma come per ogni cosa è il tempo che permette di affinarne la tecnica.

4.4 Componenti/sensori utilizzati e relativo collegamento

Il prototipo di serra idroponica sviluppata, oltre alla struttura esterna, presenta numerosi sensori e componenti installati. La parte elettrica è stata acquistata *ad hoc* per adempiere alla funzione richiesta, mentre la parte meccanica è stata realizzata quasi interamente con l'utilizzo di una stampante 3D. Mentre per sensori, connettori e dispositivi elettronici il mercato è molto ricco e rifornito, per particolari di piccole dimensione come quelli installati non lo era altrettanto. La stampa 3D ha aiutato la creazione dei particolari necessari.
Per la parte elettrica sono stati utilizzati vari dispositivi, fondamentali per il programma di gestione della serra. Per una migliore distinzione, verranno suddivisi in *Ingressi* ed *Uscite*.

4.4.1 Scheda Arduino

Il controllore utilizzato nel progetto, per la gestione del programma automatico, è la scheda *Arduino UNO R3*. Permette di avere un ottimo compromesso tra le prestazioni e il costo della scheda stessa. Utilizza il classico linguaggio di programmazione Arduino, dunque un insieme dei linguaggi C, C++ e Java. La scelta è ricaduta sulla scheda più piccola utilizzabile della gamma, ovvero quella che garantiva per prima cosa un numero di ingressi/uscite sufficienti allo scopo, oltre a delle prestazioni adeguate. La seguente scheda fornisce:

- 6 pin di ingresso analogici;
- 12 pin di ingresso/uscita digitali, tra cui 6 pin PWM.

Per lo sviluppo della serra erano necessari almeno i seguenti collegamenti con la scheda:

- 3 pin analogici di ingresso, di cui SCL e SDA per la comunicazione I2C;
- 9 pin di ingresso/uscita di cui 2 in uscita PWM, 3 di ingresso digitale e 4 di uscita digitale.

La disposizione e il collegamento delle parti con la scheda verranno trattati nella *sezione 5* in modo più dettagliato. Trattare la scheda Arduino nel dettaglio in questo progetto è al di fuori dello scopo del progetto stesso. Come descritto in precedenza, il concetto fondamentale è quello dell'uso della scheda, senza entrare nei particolari costruttivi e circuitali. Solo per completezza, viene riportato nell'appendice il datasheet del microcontrollore.

4.4.2 Ingressi

In questa categoria rientrano tutti i sensori necessari per la rilevazione dei parametri ambientali e per il corretto funzionamento del programma stesso.

- Sensore di temperatura ed umidità **DHT11**:
  Il sensore *DHT11*, molto economico e facilmente reperibile nel mercato, permette di leggere temperatura ed umidità interne alla serra. Si tratta di un dispositivo compatto, composto da un processore 8-bit single-chip. Quest'ultimo serve per gestire la parte resistiva del sensore, utilizzata per rilevare il livello di umidità dell'aria, e il sensore NTC [13] usato per il rilevamento della temperatura. Il segnale in uscita

[13] *Negative Temperature Coefficient*, si tratta di un "termistore", ovvero un dispositivo resistivo il cui valore della resistenza varia a seconda della temperatura alla quale si trova (*in questo caso la resistenza decresce al crescere della temperatura*)

dal sensore, a differenza dei classici segnali analogici, è invece digitale e permette un facile collegamento con la scheda Arduino. Attraverso l'uso di apposite librerie è possibile poi decodificare il segnale e trasformarlo in formato *float* o *int* [14], a seconda delle esigenze. Presenta 3 pin e i collegamenti con la scheda sono molto semplici: GND, da collegare al neutro della scheda, *Vcc* per l'alimentazione (5V) e *DATA* per il collegamento con il pin digitale 2 di ingresso della scheda Arduino (come in *figura 19* di seguito).

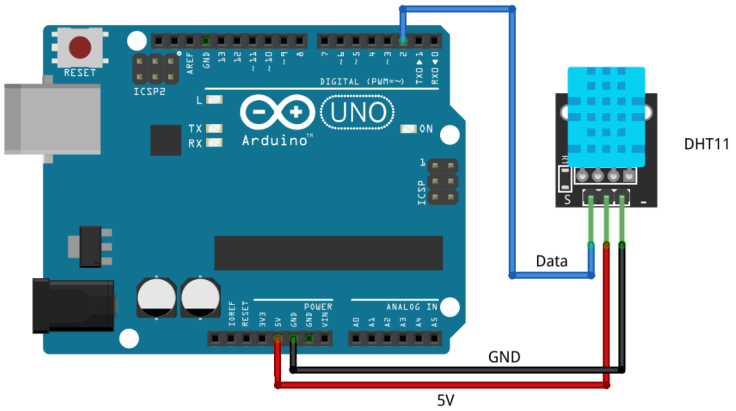


Figura 19: Schema di collegamento del sensore DHT11 alla scheda Arduino.

Dal datasheet in *figura 20*, si nota subito una risoluzione non particolarmente adatta ad un controllo automatico. Il valore varia in un intervallo di ±2 °C. D'altro canto, essendo ancora un prototipo di serra, può essere considerato comunque adatto alla funzione che deve svolgere. Per una implementazione futura, sicuramente si dovranno installare sensori con una risoluzione migliore, almeno di ±0,2 °C per avere un controllo ottimale e preciso. Range di funzionamento e temperatura di rilevamento sono compresi tra 0 e 50 °C. Data la tipologia di serra, adatta ad una coltivazione indoor, il sensore è adeguato alle temperature solitamente presenti all'interno di un edificio, ipotizzando come temperatura media quella compresa tra 20 e 25 °C. Riguardo a questo, osservando il datasheet si osserva che per valori attorno a 25 °C l'accuratezza della misurazione dell'RH [15] cala a ±4%RH.

[14] Con i termini *float* e *int* si intendono due dei vari tipi di variabili disponibili nel linguaggio di programmazione. In questo caso intendiamo rispettivamente un *numero con la virgola*, di dimensione 4 byte, e un *numero intero*, di dimensione 2 byte

[15] Con *RH* si intende l'umidità relativa dell'aria.

Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

Detailed Specifications:

Parameters	Conditions	Minimum	Typical	Maximum
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			±1%RH	
Accuracy	25℃		±4%RH	
	0-50℃			±5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0℃	30%RH		90%RH
	25℃	20%RH		90%RH
	50℃	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25℃, 1m/s Air	6 S	10 S	15 S
Hysteresis			±1%RH	
Long-Term Stability	Typical		±1%RH/year	
Temperature				
Resolution		1℃	1℃	1℃
		8 Bit	8 Bit	8 Bit
Repeatability			±1℃	
Accuracy		±1℃		±2℃
Measurement Range		0℃		50℃
Response Time (Seconds)	1/e(63%)	6 S		30 S

Figura 20: Datasheet del sensore DHT11, collegato alla scheda Arduino.

- Fototransistor **HW5P-1**:
  Questo componente è utilizzato per misurare la quantità di luce esterna alla serra, in modo da capire quando questa è insufficiente per la crescita delle piante. Si tratta di un BJT [16] in cui la giunzione tra la base ed il collettore è attivata quando viene colpita dalla luce. Questo genera una corrente di collettore con andamento lineare rispetto al livello di luminosità rilevato. Il collegamento prevede solo due pin: il piedino di anodo collegato all'alimentazione positiva (5V) e il piedino di catodo da collegare al pin analogico A0 della scheda e, mediante una resistenza di 10 kΩ, al terminale GND. Questo tipo di collegamento permette di misurare la differenza di potenziale generata ai capi della resistenza. La *d.d.p* [17] che si forma ai capi del resistore, secondo la legge di Ohm, viene rilevata dall'ingresso analogico della scheda. Questo è possibile grazie alla corrente di collettore che viene a formarsi nel fototransistor, quando viene colpito dalla luce, e la resistenza di valore fisso posta tra il catodo e il pin GND. Maggiore sarà l'intensità luminosa, maggiore sarà la corrente che scorrerà nella resistenza, e quindi proporzionalmente anche la tensione rilevata. Lo schema è quello riportato in *figura 21*.

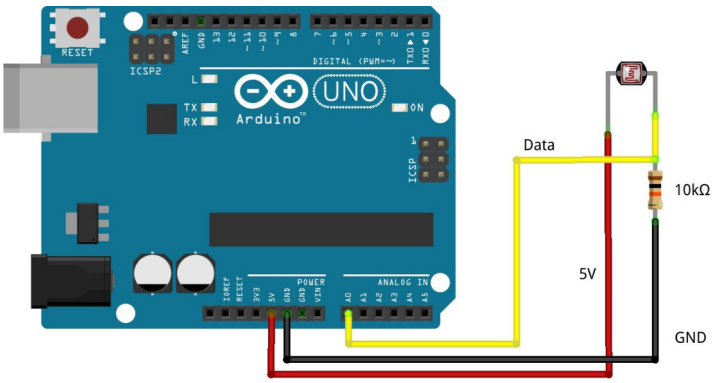


Figura 21: Schema del collegamento del sensore HW5P-1 alla scheda Arduino.

Anche in questo caso, come per il sensore *DHT11*, non si richiedeva un rilevamento dei valori esterni particolarmente accurato e sensibile. Il calcolo della quantità di luce necessaria alle piante è molto complicato, a seconda della grandezza di queste, della grandezza delle loro foglie e della posizione rispetto alla fonte di luce. Per il progetto era sufficiente

[16] Abbreviativo di *"Bipolar Junction Transistor"* è una tipologia di transistor a giunzione polare, largamente impiegato in elettronica

[17] Con *d.d.p.* intendiamo *differenza di potenziale*

avere un sensore che potesse rilevare cambiamenti di luminosità e comunicarli alla scheda Arduino per l'elaborazione dei dati e l'attuazione delle misure necessarie. Questo sensore, data la compattezza e il basso costo, era ideale e offriva uno dei migliori compromessi. Di seguito, si riporta in *figura 22* il datasheet del sensore utilizzato e in *figura 23* lo schema di collegamento.

HW5P-1

MAXIMUM RATINGS (Ta= 25° C)Ta=25℃

Characteristics	Symbol	Rating	Unit
supply voltage	Vdd	3-15	V
Quiescent current	I	<5uA@<1Lus	uA
Working temperature	Topr	-25~+90	℃
Storage temperature	Tstg	-40~+120	℃
Welding temperature (5 seconds)※※1	Tsol	260	℃

ELECTRICAL AND OPTICAL CHARACTERISTICS (Ta= 25° C)

Paramete		Symnol	Test conditions	Min	Type	Max	Unit
Collector photo-current	HW5P-1	IC （Vdd=5 V Rss=1k）	10 Lux,Vdd=5	50	60	70	uA
Collector dark current		I_{drk}	0Lux,Vdd=10V			100	nA
Spectrum sensitivity		λp	0Lux,Vdd=10V	480		1050	nM
The collector - emission level saturation pressure drop		Vdd-Vss	Lss=100mA		0.3		V
The reaction time (rise)		t_r	V_{dd}=10V, Iss=5mA R_L=100		2		μS
The reaction time (down)		t_f			2		μS

※ 2 Ev ,Ee With white LED light source test

Figura 22: Datasheet del sensore HW5P-1, collegato alla scheda Arduino.

La scheda Arduino acquisisce e legge il valore di tensione in ingresso al pin analogico corrispondente collegato al fototransistor. In seguito lo converte in un valore digitale da 0 a 1023 (conversione a 12 bit). Nel programma implementato vengono effettuate 5 letture consecutive di quest'ultimo e sommate in una variabile dedicata: successivamente viene eseguita una media e adattato il valore digitale in una scala da 0 a 100.

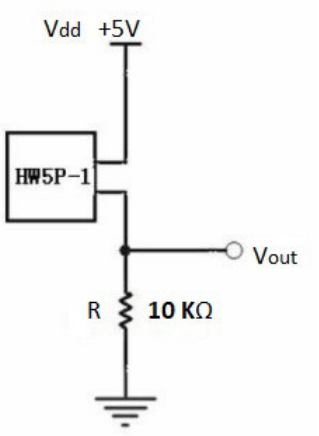


Figura 23: Schema di collegamento del sensore.

- RTC ("Real Time Clock") **DS1307**:
  Il *Real Time Clock* utilizzato è usato molto di frequente nei progetti a medio/basso costo. Purtroppo, l'ambiente semplificato di Arduino, non consente di sfruttare a pieno il RTC presente all'interno del microcontrollore. Per il progetto realizzato è stato installato un RTC esterno che consenta di memorizzare la data e l'ora. Questo dispositivo richiede il protocollo di comunicazione I2C [18], che tutte le schede, anche meno recenti, solitamente posseggono. Il collegamento tra Arduino e RTC è il seguente: un pin è collegato a GND e uno all'alimentazione 5V, mentre gli altri due servono per la comunicazione tra i due, attraverso protocollo I2C. Si utilizzano due linee seriali per la comunicazione: *SDA* collegato al pin analogico A4 della scheda Arduino, per i dati, e *SCL* collegato al pin analogico A5, per il clock. A4 e A5 sono i pin analogici di ingresso della scheda, utilizzabili inoltre per questo protocollo di comunicazione. La scheda Arduino usata nel progetto era provvista anche di pin dedicati, ma per ottenere una interscambiabilità tra le varie schede, si è preferito usare questi pin di ingresso. Lo schema di collegamento è quello riportato in *figura 24*.

[18] *I2C* è l'acronimo di *Inter Integrated Circuit* ed è un sistema di comunicazione seriale bifilare (due fili) utilizzato per il collegamento tra dispositivi. Il classico bus I2C è composto da almeno un *master* ed uno *slave*. Fa uso di due linee seriali di comunicazione: SDA (abbreviativo di *Serial "DATA"*), per i dati, e SCL (abbreviativo di *Serial "CLOCK"*), per il clock. La funzione di questi due segnali è di sincronizzare la velocità dei dati del processore della scheda con quella del dispositivo da comandare (attraverso SCL) e in seguito inviare o ricevere dati (attraverso SDA)

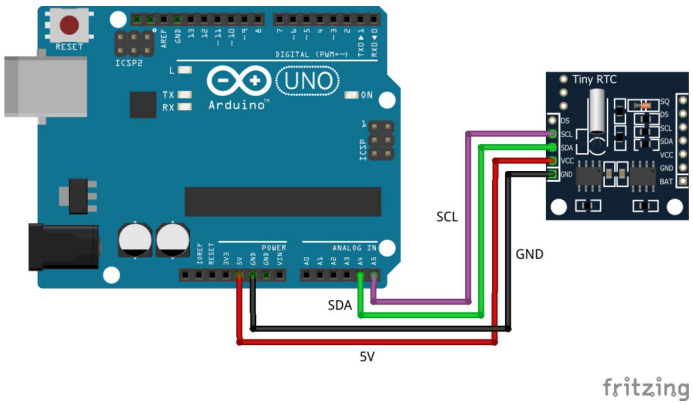


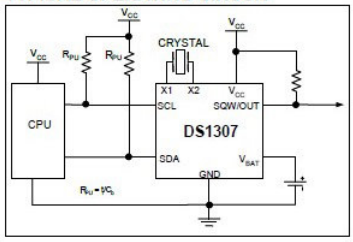
Figura 24: Schema del collegamento del RTC DS1307 alla scheda Arduino.

Nel programma della serra, il RTC è utilizzato per ottenere i dati sulla data e sull'ora disponibili, sia per l'utente ma in modo particolare per il programma stesso. Permette di distinguere i vari periodo dell'anno, in modo da regolare in automatico i parametri per le piante seguendo le varie stagioni dell'anno. Inoltre, permette di realizzare cicli temporizzati per irrigazione e ventilazione, e gestire parte del programma per la gestione dell'illuminazione della serra. Ecco che il suo ruolo non è solamente puramente estetico, ma funzionale al programma. In *figura 25* si riporta il datasheet del dispositivo utilizzato nella serra.

GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I^2C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

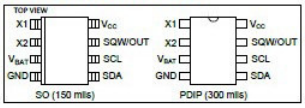
TYPICAL OPERATING CIRCUIT



BENEFITS AND FEATURES

- Completely Manages All Timekeeping Functions
  - Real-Time Clock Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week, and Year with Leap-Year Compensation Valid Up to 2100
  - 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes
  - Programmable Square-Wave Output Signal
- Simple Serial Port Interfaces to Most Microcontrollers
  - I^2C Serial Interface
- Low Power Operation Extends Battery Backup Run Time
  - Consumes Less than 500nA in Battery-Backup Mode with Oscillator Running
  - Automatic Power-Fail Detect and Switch Circuitry
- 8-Pin DIP and 8-Pin SO Minimizes Required Space
- Optional Industrial Temperature Range: -40°C to +85°C Supports Operation in a Wide Range of Applications
- Underwriters Laboratories® (UL) Recognized

PIN CONFIGURATIONS



ORDERING INFORMATION

PART	TEMP RANGE	VOLTAGE (V)	PIN-PACKAGE	TOP MARK*
DS1307+	0°C to +70°C	5.0	8 PDIP (300 mils)	DS1307
DS1307N+	-40°C to +85°C	5.0	8 PDIP (300 mils)	DS1307N
DS1307Z+	0°C to +70°C	5.0	8 SO (150 mils)	DS1307
DS1307ZN+	-40°C to +85°C	5.0	8 SO (150 mils)	DS1307N
DS1307Z+T&R	0°C to +70°C	5.0	8 SO (150 mils) Tape and Reel	DS1307
DS1307ZN+T&R	-40°C to +85°C	5.0	8 SO (150 mils) Tape and Reel	DS1307N

+Denotes a lead-free/RoHS-compliant package.
**A "+" anywhere on the top mark indicates a lead-free package. An "N" anywhere on the top mark indicates an industrial temperature range device.*
Underwriters Laboratories, Inc. is a registered certification mark of Underwriters Laboratories, Inc.

(a) *Pagina 1*

ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin Relative to Ground ..... -0.5V to +7.0V
Operating Temperature Range (Noncondensing)
Commercial ..... 0°C to +70°C
Industrial ..... -40°C to +85°C
Storage Temperature Range ..... -55°C to +125°C
Soldering Temperature (DIP, leads) ..... +260°C for 10 seconds
Soldering Temperature (surface mount) ..... Refer to the JPC/JEDEC J-STD-020 Specification.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect device reliability.

RECOMMENDED DC OPERATING CONDITIONS

(T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Logic 1 Input	V_{IH}		2.2		V_{CC} + 0.3	V
Logic 0 Input	V_{IL}		-0.3		+0.8	V
V_{BAT} Battery Voltage	V_{BAT}		2.0	3	3.5	V

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 4.5V to 5.5V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Input Leakage (SCL)	I_{LI}		-1		1	µA
I/O Leakage (SDA, SQW/OUT)	I_{LO}		-1		1	µA
Logic 0 Output (I_{OL} = 5mA)	V_{OL}				0.4	V
Active Supply Current (f_{SCL} = 100kHz)	I_{CCA}				1.5	mA
Standby Current	I_{CCS}	(Note 3)			200	µA
V_{BAT} Leakage Current	I_{BATLKG}			5	50	nA
Power-Fail Voltage (V_{BAT} = 3.0V)	V_{PF}		1.216 x V_{BAT}	1.25 x V_{BAT}	1.284 x V_{BAT}	V

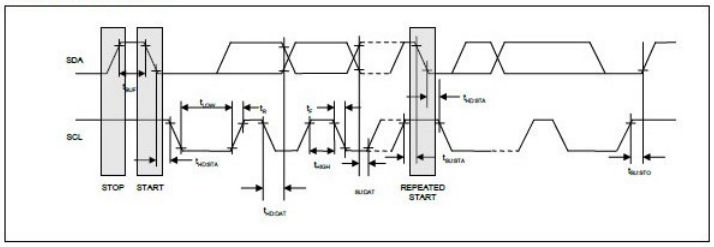
(b) *Pagina 2*

DC ELECTRICAL CHARACTERISTICS
(V_{CC} = 0V, V_{BAT} = 3.0V; T_A = 0°C to +70°C, T_A = -40°C to +85°C.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
V_{BAT} Current (OSC ON); SQW/OUT OFF	I_{BAT1}			300	500	nA
V_{BAT} Current (OSC ON); SQW/OUT ON (32kHz)	I_{BAT2}			480	800	nA
V_{BAT} Data-Retention Current (Oscillator Off)	I_{BATDR}			10	100	nA

WARNING: Negative undershoots below -0.3V while the part is in battery-backed mode may cause loss of data.

TIMING DIAGRAM



(c) *Pagina 3*

Figura 25: Datasheet del RTC DS1307, collegato alla scheda Arduino.

- Altri dispositivi utilizzati in ingresso: due **pulsanti**, collegati ai pin digitali di ingresso 10 e 11 della scheda Arduino. Permettono di cambiare la "pagina visualizzata" nello schermo (*pulsante 1*), oppure modificare i parametri del programma per la gestione della serra secondo opportune ricette [19] (*pulsante 2*). Il collegamento con la scheda Arduino prevede l'alimentazione a 5V di uno dei pin del pulsante e il collegamento in uscita posto in parallelo tra il pin digitale della scheda e una resistenza di 10 kΩ (*figura 26*).

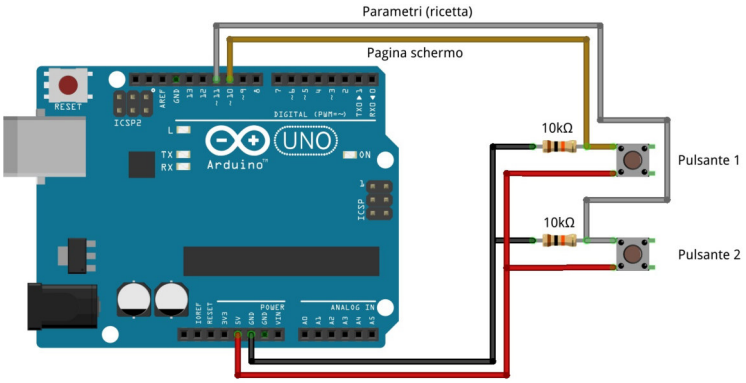


Figura 26: Schema del collegamento dei pulsanti alla scheda Arduino.

[19] Con *ricette* intendiamo una serie di dati, precompilati, da caricare durante il funzionamento del programma.

4.4.3 Uscite

Fanno parte di questo gruppo tutti gli attuatori e i dispositivi collegati in uscita alla scheda Arduino. Grazie all'acquisizione dei dati in ingresso e alla loro successiva elaborazione da parte del programma, permettono di attuare le tecniche di controllo per la gestione automatica del ciclo della serra idroponica.

- Servomotore **SM-S2309S**:
  Collegato alla scheda Arduino semplicemente con 3 cavi, permette di variare l'angolo dei deflettori [20] e cambiare la portata del flusso dell'aria che circola con ventola accesa e spenta. Il collegamento, come riportato in *figura 27*, prevede un cavo collegato a GND, uno all'alimentazione 5V e l'altro al segnale proveniente dal pin di uscita 3 della scheda, in grado di generare un segnale PWM [21].

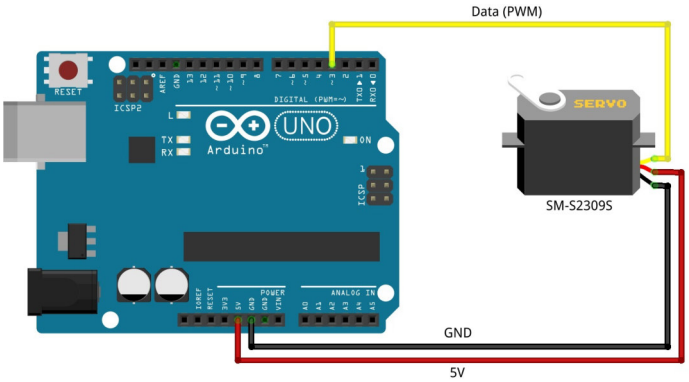


Figura 27: Schema del collegamento del servomotore SM-S2309S alla scheda Arduino.

La sua versatilità e il suo costo limitato hanno fatto in modo che la scelta ricadesse su questo componente. La possibilità di avere compatibilità con più schede e viceversa, è inoltre una peculiarità di questo attuatore. La coppia prodotta, come riportato nel datasheet in *figura 28*, è di 1,1 $Kg \cdot cm$, ovvero 10,78 $N \cdot cm$. E' decisamente sufficiente per lo scopo utilizzato, in quanto l'attrito che si genera tra le parti in plastica e il peso stesso, sono nettamente inferiori alla coppia erogabile

[20] I *deflettori*, come vedremo nella sezione sulla "Descrizione della serra", sono dei componenti installati sulla parte superiore della serra idroponica in grado di regolare il flusso d'aria, in funzione della rotazione dell'albero del servomotore, a cui sono opportunamente collegati.

[21] *PWM*, "*Pulse-Width Modulation*" o "*modulazione di larghezza d'impulso*", è un tipo di modulazione digitale che permette di avere una tensione media variabile in funzione del rapporto tra la durata dell'impulso positivo e la durata complessiva del periodo. Questo rapporto è detto anche *duty cycle*

dal dispositivo. L'angolo di rotazione richiesto per la movimentazione dei deflettori si aggira attorno ai 90°, compreso nell'intervallo offerto dal servomotore.

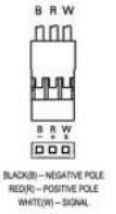
SM-S2309S MOTOR

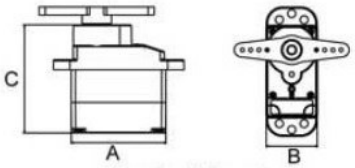
No. : SM-S2309S

Size: 22.9x12.3x22.2mm / 0.9x0.49x0.87in

Weight: 0.35oz

Specifications: Micro analog servo,4 plastic gears + 1 metal gear





Dimension（Unit：mm）

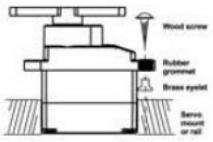


Diagram of installation

Products specification								Technical parameters						
Size(mm)					Weight		Wire	4.8V			6V			Rotation angle
								Speed	Torque		Speed	Torque		
A	B	C	D	E	g	oz	cm	sec/60°	kg•cm	oz•in	sec/60°	kg•cm	oz•in	
22.9	12.3	22.2	-	-	9.9	0.35	20.0	0.11	1.1	15.3	0.09	1.3	18.1	±60°

(Specifications are subjected to change without notice.)

Product brochure

Micro analog servo,4 plastic gear+ 1 metal gear.

Products packing

◦Packing with elevators (Elevators+PE bag)

Packaging content:Servo×1PCS、Servo arm×1bag、Manual×1PCS

Packaging specifications：Size-120×85mm、PE bag：120×85×0.07mm、Net weight-9.9g、Gross weight-11.5g

◦General packing (PE bag)

Packaging content：Packing with PE bag

Packaging specifications：PE bag size-90×85×0.07mm、Net weight-9.9g、Gross weight-11.5g

Figura 28: Datasheet del servomotore SM-S2309S, collegato alla scheda Arduino.

- Display OLED **SSD1306** I2C da 0.96 oz:
  Fabbricato da *"AZ-Delivery"*, è lo schermo con il rapporto qualità/prezzo migliore sul mercato. Il modello installato è provvisto di schermo monocromatico, ma è possibile acquistare ed installare anche il modello a colori. La sua compattezza nelle dimensioni, e la possibilità di rappresentare più informazioni contemporaneamente sullo schermo, lo rendevano particolarmente adatto al progetto rispetto al classico display LCD. Sullo schermo infatti sono rappresentate nello stesso istante tutte le informazioni indispensabili all'utente per capire lo stato della serra. Il collegamento, come raffigurato in *figura 29*, fa uso della comunicazione I2C.

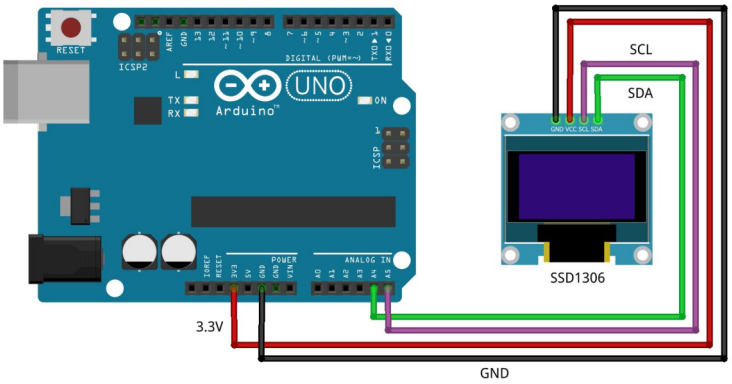


Figura 29: Schema del collegamento del display OLED SSD1306 alla scheda Arduino.

Come per *RTC DS1307*, anche qui troviamo collegati ai pin A4 e A5 della scheda Arduino rispettivamente i pin SDA e SCL dello schermo. A differenza di tutti gli altri dispositivi però, la tensione di alimentazione è 3.3V. La tensione massima di collegamento è *5.5V*, come riportato nel datasheet di *figura 30*, ma è consigliabile la tensione citata in precedenza. Il pin GND è collegato direttamente alla scheda.

AZ-Delivery
Ihr Experte für Mikroelektronik!

0,96 Zoll OLED Display
Datenblatt

1. Basic Specifications

- Module dimensions: 28 mm x 33 mm x 4mm
- Dot Matrix: 128 x 64
- Pixel size: 0.152 × 0.152 mm
- Pixel pitch: 0.175 × 0.175 mm
- Display Mode: Passive Matrix
- Duty: 1/64 Duty
- Display Color: White
- Interface: I2C
- Power consumption: Less then 11mA
- Size: 0.96 inch

2. Absolute Maximum Ratings

Parameter	Symbol	Min.	Max.	Unit	Notes
Supply Voltage for Logic	VDD	-0.3	5.5	V	1,2
Operating Temperature	Top	-40	+85	°C	
Storage Temperature	Tstg	-40	+85	°C	3
Life Time (120cd/□)	--	10.000	--	hour	4
Life Time (80cd/□)	--	30.000	--	hour	4
Life Time (60cd/□)	--	50.000	--	hour	4

Note 1: All the above voltages are on the basis of "VSS = 0V".

Note 2: When this module is used beyond the above absolute maximum ratings, permanent breakage of the module may occur.

Note 3: The defined temperature ranges do not include the polarizer. The maximum withstood temperature of the polarizer should be 80 °C.

(a) *Pagina 1*

Note 4: Ta=25 °C., 50% Checkerboard.

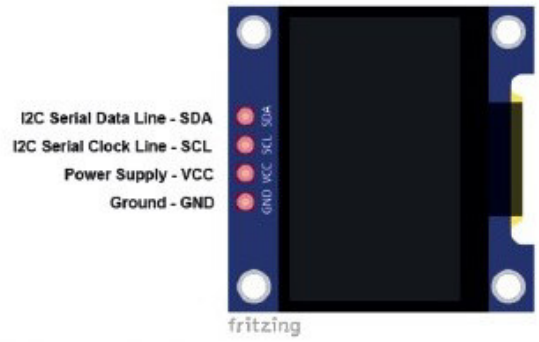
End of lifetime is specified as 50% of initial brightness reached. The average operating lifetime at room temperature is estimated by the accelerated operation at high temperature conditions.

3. Electrical Characteristics

Item	Symbol	Condition	Min.	Typ.	Max.	Unit
Supply Voltage for Logic	V_{DD}	External Supply	3.0	3.3	5.0	V
Supply Voltage for Logic IO	V_{DDIO}	Internal Supply	3.0	--	3.3	V
High Level Input	V_{IH}	-	0.8xV_{DDIO}	--	V_{DDIO}	V
Low Level Input	V_{IL}	-	0	--	0.2xV_{DDIO}	V
High Level Output	V_{OH}	I_{OUT}=100uA,3.3MHz	0.9xV_{DDIO}	--	V_{DDIO}	V
Low Level Output	V_{OL}	I_{OUT}=100uA,3.3MHz	0	--	0.1xV_{DDIO}	V
Operating Current for V_{DD}	I_{DD}	Note 5	--	28.5	33	mA
Sleep Mode Current for V_{DD}	I_{DD}, Sleep	--	--	-	1	mA

Note 5: V_{DD} = 3.3V, 100% Display Area Turn on.

4. Pinout



These displays have an on-board 3.3V voltage regulator.

The pins of 0.96 inch OLED display can be connected to either 3.3V or 5V logic and power supply without risk of damaging display.

NOTE: When using these displays with Raspberry Pi, power supply is 3.3V.

(b) *Pagina 2*

Figura 30: Datasheet dello schermo SSD1306, collegato alla scheda Arduino.

- Relay **JQC3F-5VDC-C**:
  Permettono la configurazione della scheda con i dispositivi esterni con tensione di alimentazione superiore (12V). Garantiscono l'isolamento galvanico [22] tra il circuito elettrico di Arduino, a tensione inferiore, e i circuiti elettrici della serra. Molto economici, non hanno funzioni troppo complesse da svolgere, se non mantenere un livello logico di uscita costante a seconda del segnale di ingresso comunicato, proveniente dalla scheda Arduino. Il collegamento è molto semplice: un pin viene collegato a GND, uno all'alimentazione 5V, e l'ultimo ai pin digitali di uscita della scheda Arduino, come riportato in *figura 31*. Per la serra idroponica sono stati utilizzati tre di questi componenti, collegati ai pin digitali di uscita 4, 7 e 8 della scheda e che comandano, rispettivamente, la pompa, l'illuminazione della serra e la ventola.

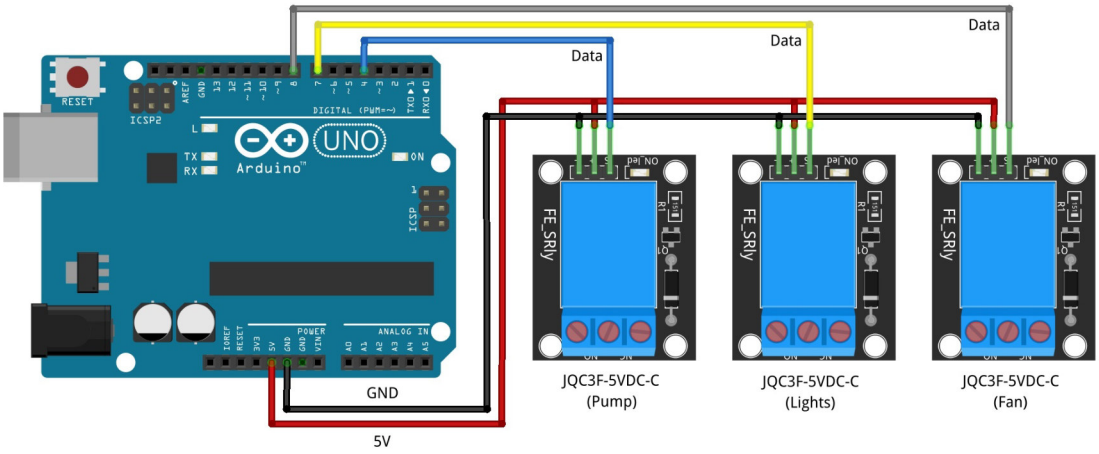


Figura 31: Schema del collegamento dei relay JQC3F-5VDC-C alla scheda Arduino.

Osservando il datasheet in *figura 32*, per una corrente di 10A la tensione massima di alimentazione della porta di uscita dei relay è 250V AC [23] e 30V DC [24]. I dispositivi esterni non collegati direttamente ad Arduino, hanno tensione di alimentazione di 12V e pochi ampere di corrente: i dispositivi sono dunque adeguati al loro uso nel progetto.

[22] Per *isolamento galvanico* s'intende la condizione per cui, tra due punti a differente potenziale elettrico, non ha e non potrà avere luogo una circolazione di corrente, in modo da preservare il circuito elettrico più sensibile e delicato.

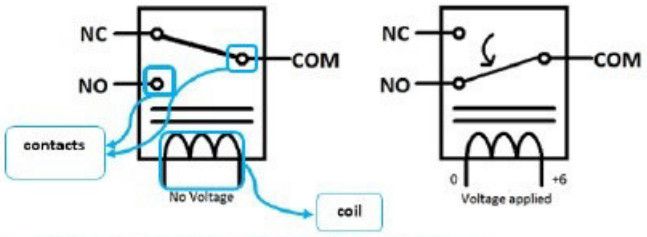
[23] Per AC si intende "Alternating Current", o corrente alternata.

[24] Per AC si intende "Direct Current", o corrente continua.

RELAY MODULES

RELAY WORKING IDEA

Relays consist of three pins normaly open pin , normaly closed pin, common pin and coil. When coil powerd on magntic field is generated the contacts connected to each other.



Relay modules 1-channel features

- Contact current 10A and 250V AC or 30V DC.
- Each channel has indication LED.
- Coil voltage 12V per channel.
- Kit operating voltage 5-12 V
- Input signal 3-5 V for each channel.
- Three pins for normally open and closed for each channel.

How to connect relay module with Arduino

As shown in relay working idea it depends on magnetic field generated from the coil so there is power isolation between the coil and the switching pins so coils can be easily powered from Arduino by connecting VCC and GND bins from Arduino kit to the relay module kit after that we choose Arduino output pins depending on the number of relays needed in project designed and set these pins to output and make it out high (5 V) to control the coil that allow controlling of switching process.

Figura 32: Datasheet dei relay JQC3F-5VDC-C, collegato alla scheda Arduino.

- Capsula piezo **PKM22EPP-4001-B0**:
  Utilizzato come *buzzer* sonoro per segnalare delle variazioni sui parametri del programma della serra, serve anche a segnalare acusticamente delle situazioni di allarme. Le funzioni svolte sono due: segnala il cambio dei parametri della serra o la pagina visualizzata sullo schermo OLED e, in caso di malfunzionamento dei dispositivi collegati (sensori), segnala uno stato di allarme all'utente. Il collegamento è riportato in *figura 33*. Un pin della capsula è collegato direttamente al pin di uscita analogico 12 (PWM) della scheda Arduino mentre l'altro pin invece a GND. E' necessario l'uso di un pin di uscita PWM per variare la frequenza del segnale acustico che viene riprodotto.

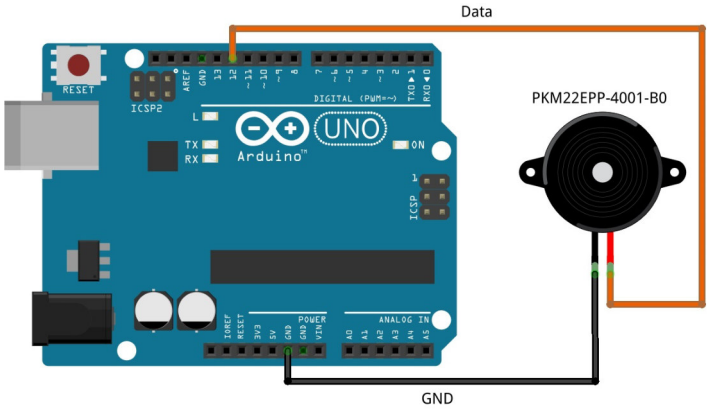


Figura 33: Schema del collegamento della capsula piezo PKM22EPP-4001-B0 alla scheda Arduino.

Il datasheet, con dimensioni e specifiche tecniche del dispositivo, è riportato in *figura 34*.

Piezoelectric Sound Components

Piezoelectric Sounders External Drive Pin Type

Now microcomputers are widely used for microwave ovens, air conditioners, cars, toys, timers, and other alarm equipment. Externally driven piezoelectric sounders are used in digital watches, electronic calculators, telephones and other equipment. They are driven by a signal (ex: 2048Hz or 4096Hz) from an LSI and provide melodious sound.

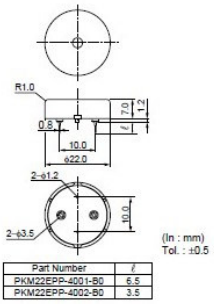
■ Features

1. Low power consumption
2. No contacts therefore, no noise and highly reliable

■ Applications

1. Telephone ringers
2. Various office equipment such as PPCs, printers and keyboards
3. Various home appliances such as microwave ovens
4. Confirmation sound of various audio equipment

PKM22EPP-4001-B0



Part Number	ℓ
PKM22EPP-4001-B0	6.5
PKM22EPP-4002-B0	3.5

Part Number	Sound Pressure Level (dB)	Sound Pressure Level (Ref. only) (dB)	Min. of Operating Voltage Range	Capacitance (nF)	Operating Temp. Range	Storage Temp. Range
PKM22EPP-4001-B0	75 min. [3Vp-p,4kHz,square wave,10cm]	75 min. [1Vrms,4kHz,sine wave,10cm]	25 Vp-p max.	12 ±30% [1kHz]	-20 to +70°C	-30 to +80°C

■ Freq. Response (Square Wave 3Vp-p, 10cm)

PKM22EPP-4001-B0

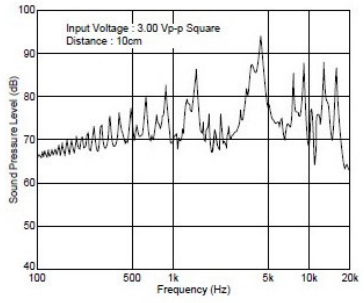


Figura 34: Datasheet della capsula piezo relay PKM22EPP-4001-B0, collegato alla scheda Arduino.

- Led **UR502DC**:
  E' il classico led utilizzato in elettronica, con la possibilità di molti colori. Il colore scelto è il rosso che, in sicurezza elettrica, serve a richiamare l'attenzione e riportare situazioni di errore o pericolo. Assieme alla capsula piezo, serve a segnalare le situazioni di errore dei componenti installati o del programma stesso. Il collegamento, come riportato in *figura 35*, prevede l'anodo del led (piede lungo) collegato al pin di uscita digitale 13 della scheda Arduino, e il catodo (piede corto) collegato ad una resistenza da 220Ω per poi collegarsi al GND. Quando il led viene attivato, sul pin 13 della scheda sono presenti 5V. La caduta di tensione del led è attorno ai 2.2V, come riportato in *figura 36*, dunque la corrente che circolerà sarà data da: Si avrà dunque:

$$I = \frac{(5V - 2.2V)}{220\Omega} \simeq 12.7mA$$

  Questo valore è sicuramente inferiore alla corrente massima ammissibile dal led, che da datasheet è di 30mA, e inferiore anche alla massima corrente erogabile (continuamente) dalla scheda Arduino, ovvero 20mA.

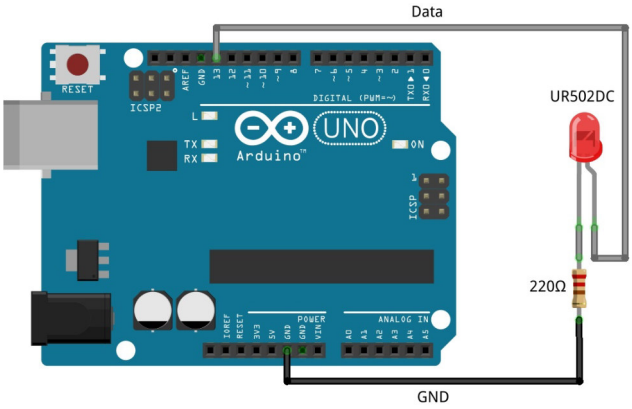
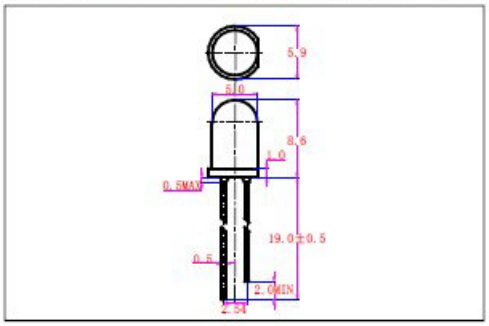


Figura 35: Schema del collegamento del led UR502DC alla scheda Arduino.

Di seguito (*figura 36*) il datasheet del componente:

产品型号（Part number system for led lamp） **UR502DC（φ5MM 短脚超亮红四元）**

外形图（Package Dimensions）单位：(Unit)：mm



晶片(CHIP)	
材质（Material）	InGaAlP
颜色(Color)	红色
胶体（Colloid）	
材质（Material）	环氧树脂
颜色(Color)	红色扩散

极限参数（Absolute Maximum Ratings）(Ta=25℃)

项目参数（Parameter）	符号（Symbol）	数值	单位（Unit）
最大功耗（Max Power Dissipation）	P_M	80	mW
最大正向电流（Max Continuous Forward Current）	I_{FM}	30	mA
最大反向电压（Max Reverse Voltage）	V_{RM}	5	V
最大脉冲峰值电流（Peak Forward Current）	I_{FP}	75	mA
焊接温度/时间（Lead Soldering Temperature/Time）	T_{SOL}	240/≤3S	℃/S
工作环境（Operating Temperature Range）	T_{OPR}	-25~+85	℃
储存温度（Storage Temperature Range）	T_{STR}	-30~+100	℃

光电参数（Initial Electrical Optical Characteristics）

项目参数（Parameter）	符号 Symbol	最小值 Min.	一般值 Typ.	最大值 Max.	单位 Unit	测试条件 Condition
发光强度（Luminous Intensity）	I_V	500	800	1000	mcd	I_F=20mA
发光角度（Viewing Angle）	2Ø1/2	/	35	/	deg	I_F=20mA
峰值波长（Peak Wave Length）	λ_p	/	630	/	nm	I_F=20mA
主波长（Dominant Wave Length）	λ_d	625	630	635	nm	I_F=20mA
频宽（Spectral Width at half height）	$\triangle\lambda$	/	30	/	nm	I_F=20mA
正向电压（Forward Voltage）	V_F	1. 9	2. 2	2. 4	V	I_F=20mA
反向电流（Reverse Current）	I_R	/	/	≤30	μA	VR=5V

Figura 36: Datasheet del led UR502DC, collegato alla scheda Arduino.

- Ventola **NMB 2408NL-05W-B50**: Utilizzata nel progetto per garantire un ricircolo dell'aria interna alla serra. Posta sulla parte superiore, durante la sua attivazione fornisce un apporto di aria fresca verso l'interno della serra che permette di arieggiare la parte interna abbassando temperatura ed umidità. La direzione del flusso non è casuale ma dovuta alla forma della serra. L'aria calda tende a salire e ad incanalarsi sulla parte superiore della serra. In questo modo, non si è costretti a forzare l'uscita dell'aria, ma basterà solamente lasciare aperti i deflettori presenti sul tetto della serra. Il collegamento non è diretto con la scheda Arduino, in quanto la tensione di alimentazione minima è di 6V e la scheda è in grado di generare in uscita massimo 5V. La ventola, viene alimentata quindi ad una tensione di 12V DC. Lo schema è riportato in *figura 37*: il connettore di alimentazione viene collegato al contatto NO [25] del relay e l'altro al GND del trasformatore. Il contatto comune ("COMMON") del relay è collegato direttamente al trasformatore che fornisce una alimentazione di 12V DC.

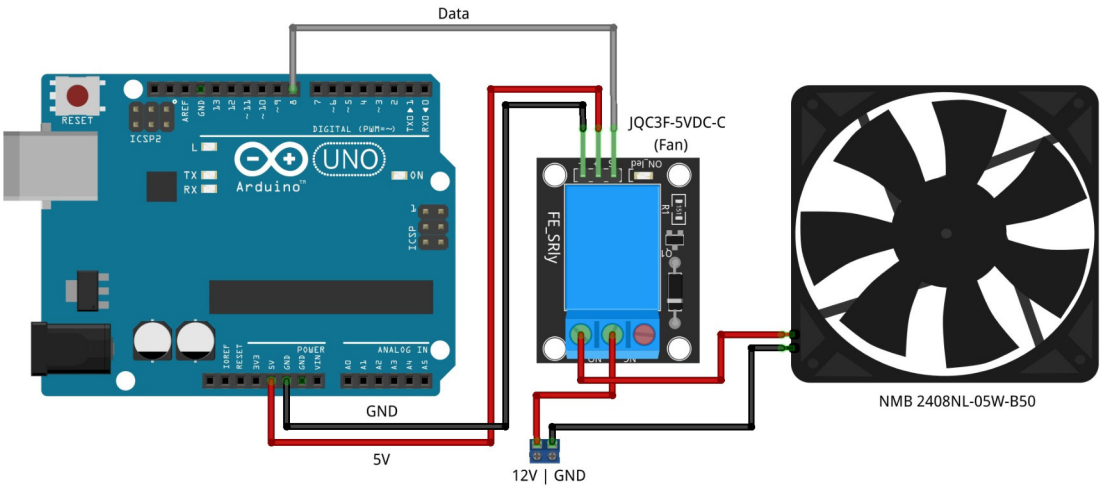
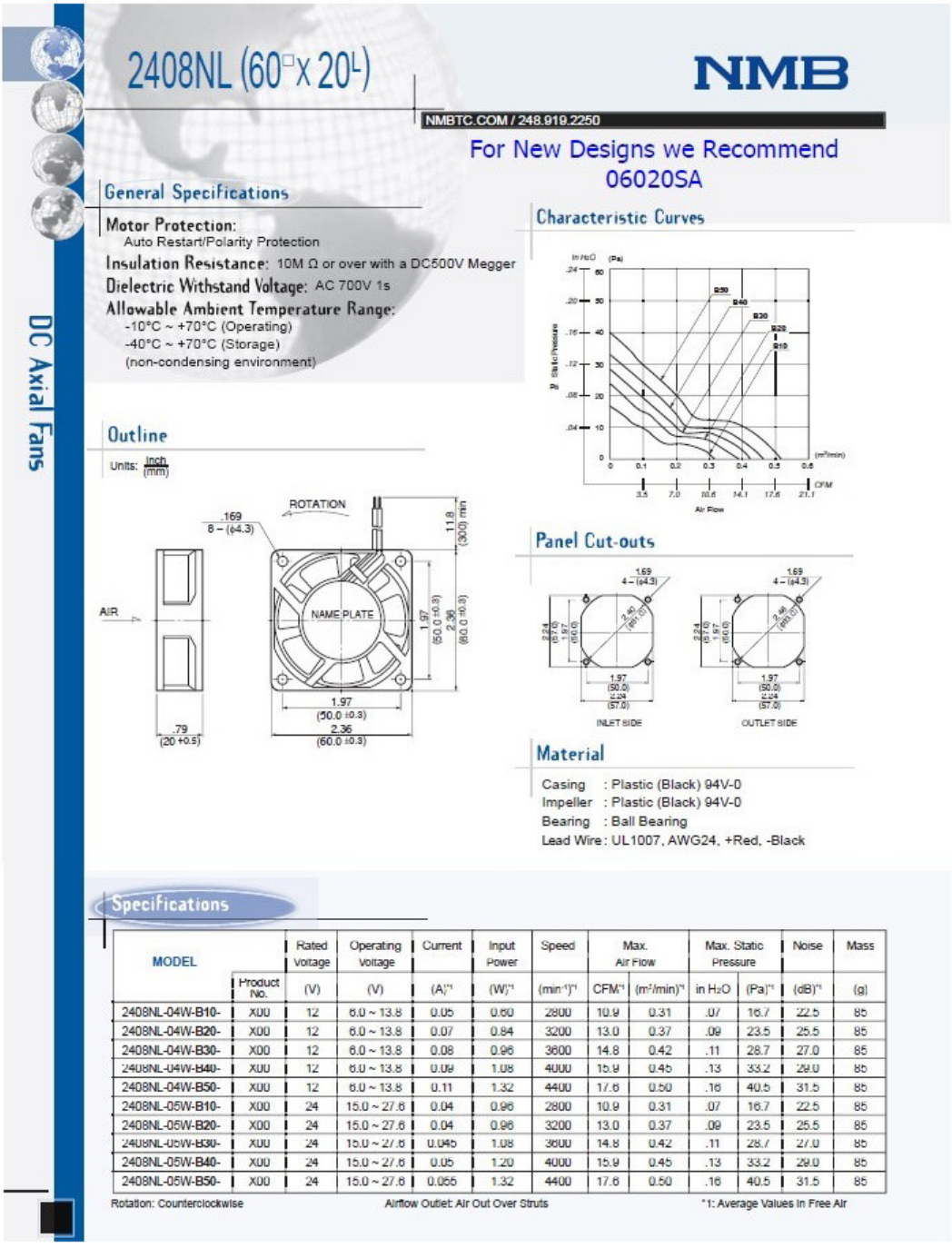


Figura 37: Schema del collegamento della ventola 2408NL-05W-B50 alla scheda Arduino.

Il relay garantisce l'isolamento galvanico tra la scheda Arduino e il circuito di potenza a 12V e permette di azionare, all'occorrenza, la ventola. Dal datasheet in *figura 38*, si nota il valore della portata d'aria: 0,50 m^3/min. Questo sarà un elemento importante da tenere in considerazione per il dimensionamento della ventola in funzione al volume della serra.

[25] Con NO intendiamo "*normal open*", ovvero *normalmente aperto*



MODEL	Product No.	Rated Voltage (V)	Operating Voltage (V)	Current (A)*1	Input Power (W)*1	Speed (min⁻¹)*1	Max. Air Flow CFM*1	Max. Air Flow (m³/min)*1	Max. Static Pressure in H₂O	Max. Static Pressure (Pa)*1	Noise (dB)*1	Mass (g)
2408NL-04W-B10-	X00	12	6.0 ~ 13.8	0.05	0.60	2800	10.9	0.31	.07	16.7	22.5	85
2408NL-04W-B20-	X00	12	6.0 ~ 13.8	0.07	0.84	3200	13.0	0.37	.09	23.5	25.5	85
2408NL-04W-B30-	X00	12	6.0 ~ 13.8	0.08	0.96	3600	14.8	0.42	.11	28.7	27.0	85
2408NL-04W-B40-	X00	12	6.0 ~ 13.8	0.09	1.08	4000	15.9	0.45	.13	33.2	29.0	85
2408NL-04W-B50-	X00	12	6.0 ~ 13.8	0.11	1.32	4400	17.6	0.50	.16	40.5	31.5	85
2408NL-05W-B10-	X00	24	15.0 ~ 27.6	0.04	0.96	2800	10.9	0.31	.07	16.7	22.5	85
2408NL-05W-B20-	X00	24	15.0 ~ 27.6	0.04	0.96	3200	13.0	0.37	.09	23.5	25.5	85
2408NL-05W-B30-	X00	24	15.0 ~ 27.6	0.045	1.08	3600	14.8	0.42	.11	28.7	27.0	85
2408NL-05W-B40-	X00	24	15.0 ~ 27.6	0.05	1.20	4000	15.9	0.45	.13	33.2	29.0	85
2408NL-05W-B50-	X00	24	15.0 ~ 27.6	0.055	1.32	4400	17.6	0.50	.16	40.5	31.5	85

Figura 38: Datasheet della ventola 2408NL-05W-B50, collegato alla scheda Arduino.

- Pump DECDEAL **QR50E**: Fondamentale per attuare il ciclo di irrigazione di una serra idroponica automatizzata. E' una classica pompa per acquario, ma riesce a garantire una buona portata e una buona pressione, in modo da poter raggiungere tutti i gocciolatori installati. Lo schema elettrico è analogo a quello della ventola, in quanto anch'essa è alimentata a 12V DC ed è collegata al suo relay dedicato, raffigurato in *figura 39*. Il connettore di alimentazione è collegato al contatto NO, e il comune del relay al trasformatore a 12V DC. L'altro contatto della ventola è portato al GND.

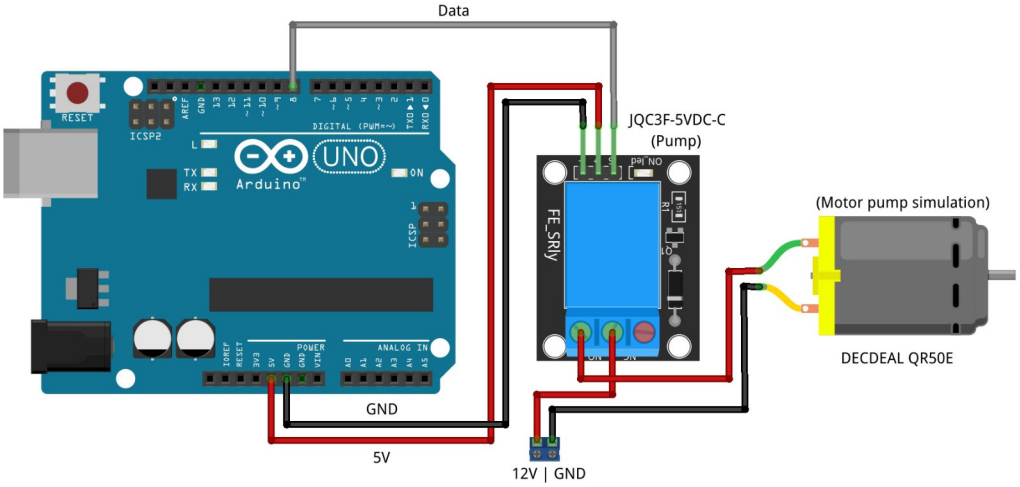
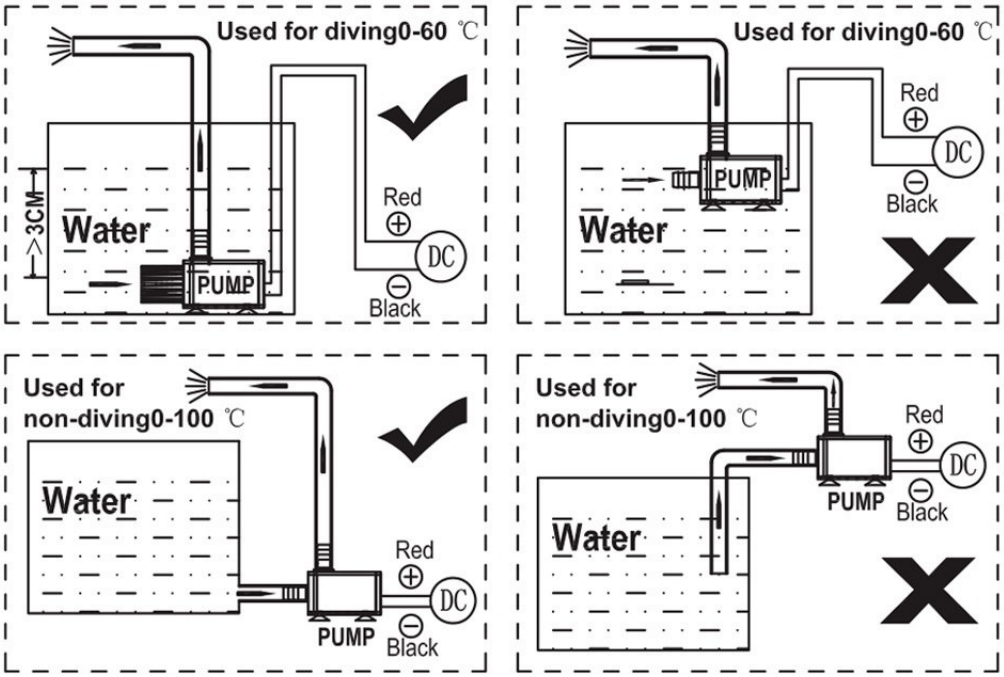


Figura 39: Schema del collegamento della pompa QR50E alla scheda Arduino.

Dal datasheet riportato in *figura 40*, è possibile osservare come la pompa abbia una portata di 280 L/h, circa 4,67 L/min, sufficienti a rifornire di acqua tutte le piante presenti nella serra. Anche il posizionamento corretto della pompa è di fondamentale importanza per il funzionamento di questa. Dal grafico di *figura 40b*, considerando la posizione di installazione della pompa tra 200 e 250 cm sotto al livello di uscita del liquido, ovvero dalla posizione dei gocciolatori, alla tensione nominale di servizio di 12V DC avremo il flusso massimo di 300 L/h, trascurando le perdite. Ipotizzando un calo del 5÷6% dovuto ad attriti e perdite nel sistema, torniamo nuovamente ad un valore di ∼280 L/h.

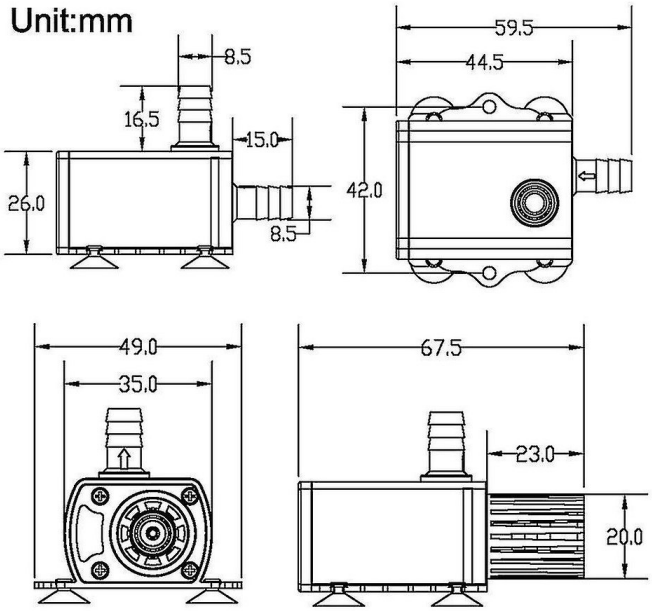
Installation instructions:



Specifications:

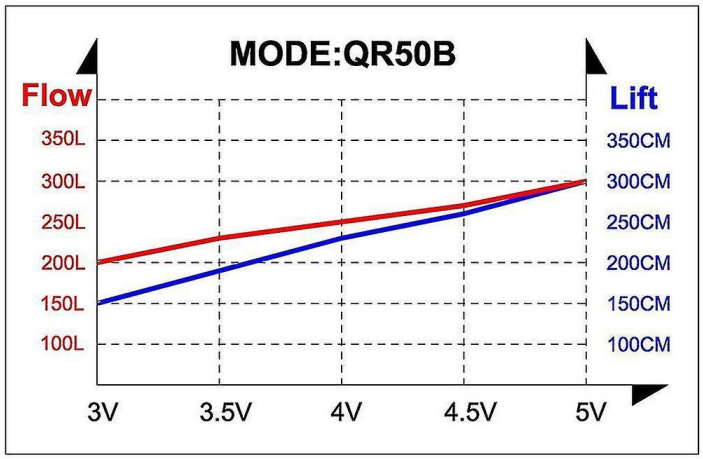
Color: Black
Pump material: PC + ABS
Rated Voltage: DC 12V
Max. Power: 5W
Max. Current: 416mA
Max. Flow Quantity: 280L/H
Max. Lift: 300cm / 9.84ft
Noise: Less than 35dB
Outer Dia. of Inlet/Outlet: 8.5mm / 0.33in
Inner Dia. of Inlet/Outlet: 6mm / 0.24in
Waterproof Class: IP68
Max. Liquid Temperature: 60°C
Cable Length: 100cm / 39.4in
Product Size: 59.5 * 49 * 42mm / 2.34 * 1.9 * 1.65in
Product Weight: 67g / 2.37oz
Package Size: 6.3 * 6 * 5.4cm / 2.5 * 2.36 * 2.12in
Package Weight: 82g / 2.9oz

(a) *Pagina 1*



Product curve performance

Amphibious pump, you can use it in the water or out of water(follow the installation instruction).
IP68 waterproof grade, totally submersible in the water.
With a filter, avoiding blocking the inlet.



(b) *Pagina 2*

Figura 40: Datasheet dello schermo SSD1306, collegato alla scheda Arduino.

- Led per illuminazione **LC LED 12V**: Led provvisti di opportuno cavo di lunghezza 12 cm. La tensione di alimentazione è di 12V DC, e sono collegati direttamente al loro relay dedicato. L'anodo è collegato al contatto NO del relay e il "COMMON" di questo al trasformatore che fornisce l'alimentazione di 12V DC. Il catodo dei led invece, è collegato al connettore GND. In totale sono 8 *led rossi* e 8 *led blu*, collegati tra loro in parallelo, in modo da garantire la continuità di servizio in caso di guasto di uno o più di questi. Un concetto da approfondire riguardo l'uso di questi led è quello al loro collegamento in parallelo senza l'uso di resistenze esterne che ne determinino la corrente che vi scorre. Come riportato dal datasheet in *figura 42*, nella descrizione dei led vi è una voce che specifica l'inutilità nell'uso di cavi o resistenze esterne. Infatti, come già detto, i led sono muniti di un apposito cavo e, sempre da datasheet, si può notare che sono provvisti anche di chip resistivo integrato, ovvero una resistenza di valore 1 kΩ che permette di collegarli direttamente a rete senza dover inserire ulteriori dispositivi elettronici. Considerando un valore di $V_F = 9V$, è possibile trovare la corrente che scorre su ogni ramo dei led:

$$I_f = \frac{(12V - 9V)}{1000\Omega} = 3mA$$

Questo valore è per il singolo ramo del led, inferiore al limite massimo in grado di sopportare continuativamente che è di 20 mA, e complessivamente richiedono dunque 48 mA. La scelta del colore non è casuale, ma rispecchia quanto spiegato nel *capitolo 3*. Per sfruttare lo spettro del colore necessario alla crescita e alla fioritura/fruttificazione delle piante, si dovevano fornire solamente i due colori che, una volta assorbiti, portassero dei benefici alla crescita delle piante stesse. I led di entrambi i colori sono accesi contemporaneamente, come succede per molte lampade per l'illuminazione presenti in commercio. La scelta di non differenziare i colori è legata alle stesse dimensioni della serra. Le piante al suo interno, non raggiungono lo stadio di crescita di fioritura/fruttificazione e dunque inserire nel programma, e nel progetto, questa gestione avrebbe comportato una complicazione a livello logico del programma e l'aggiunta di elementi elettrici superflui. Entrambi i colori sono stati inseriti per una futura implementazione: infatti, se i dati saranno confermati, l'ampliamento delle dimensioni della serra e l'uso di una gestione più efficiente porterà alla loro differenziazione. Lo schema elettrico è quello riportato in *figura 41*.

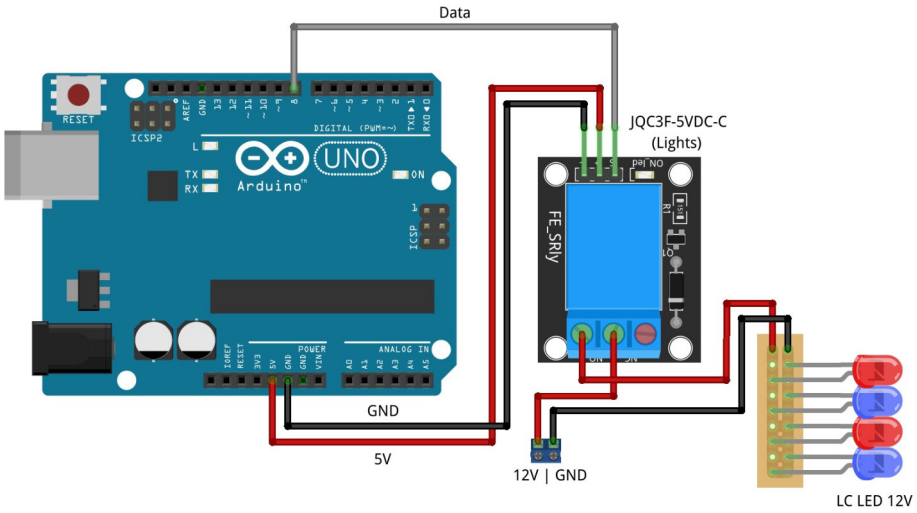


Figura 41: Schema del collegamento dei led LC LED 12V alla scheda Arduino (riportati solo 4 dei 16 led complessivi).

Il datasheet dei led, analogo per entrambi i colori, è riportato in *figura 42*. Come riportato, l'uso dei led è per *indoor* e per *outdoor*. Anche questa scelta non è casuale: all'interno della serra l'umidità relativa può arrivare anche a valori molto elevati ed è dunque importante che i led, essendo a stretto contatto con l'umidità, siano impermeabili e non subiscano danni.

LC LED 12V 5mm LED Lamp Water Clear (30 Degree)

PRODUCT DESCRIPTION	
(1) Design for 12VDC Operation	(4) High Intensity LED Die Suitable for Outdoor Uses
(2) No External Wiring or Resistor	(5) Max. Current Load of 20mA (12VDC)
(3) Incorporates Built-In Internal Resistor Chip	(6) RoHS Compliant Part

Absolute Maximum Rating (Ta = 25^0C)

PARAMETER	MAXIMUM RATING	UNITS
Max. DC Forward Voltage	12	V
DC Forward Current	20	mA
Avg. Forward Current	20	mA
Operating Temperature	-40 to +100	^{0}C
Storage Temperature	-40 to +100	^{0}C
Lead Soldering Temperature	260^0C for 6 seconds (1.0mm or 0.63 inch from Body)	

Electro-optical Characteristics (Ta = 25^0C)

PARAMETER	SYMBOL	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Forward Voltage	V_F	I_F = 20mA	8	12	13	V
Reverse Current	I_R	I_F = 20mA			100	μA
Dominant Wavelength	λ_D	I_F = 20mA		630		nm
Viewing Angle	2θ1/2	I_F = 20mA		30		Deg.
Luminous Intensity	I_V	I_F = 20mA	3,000		5,000	mcd

DEVICE DRAWING

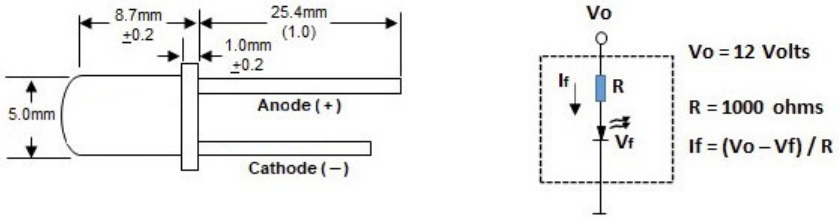


Figura 42: Datasheet dei led LC LED 12V, collegati alla scheda Arduino.

4.5 Schema elettrico della serra

Oltre ai singoli collegamenti tra la scheda Arduino e i componenti, è utile osservare anche lo schema elettrico complessivo. Per comodità, di seguito viene riportata una rappresentazione grafica dei collegamenti effettuati. L'immagine in *figura 43* permette di visualizzare tutti i componenti utilizzati, dai segnali di potenza (di alimentazione) della scheda Arduino e dei dispositivi a 12V DC fino ai componenti finali di utilizzo nella serra. Sicuramente si presenta molto più chiara di come si presentano i collegamenti e la disposizione dei cavi reali, in *figura 44*.

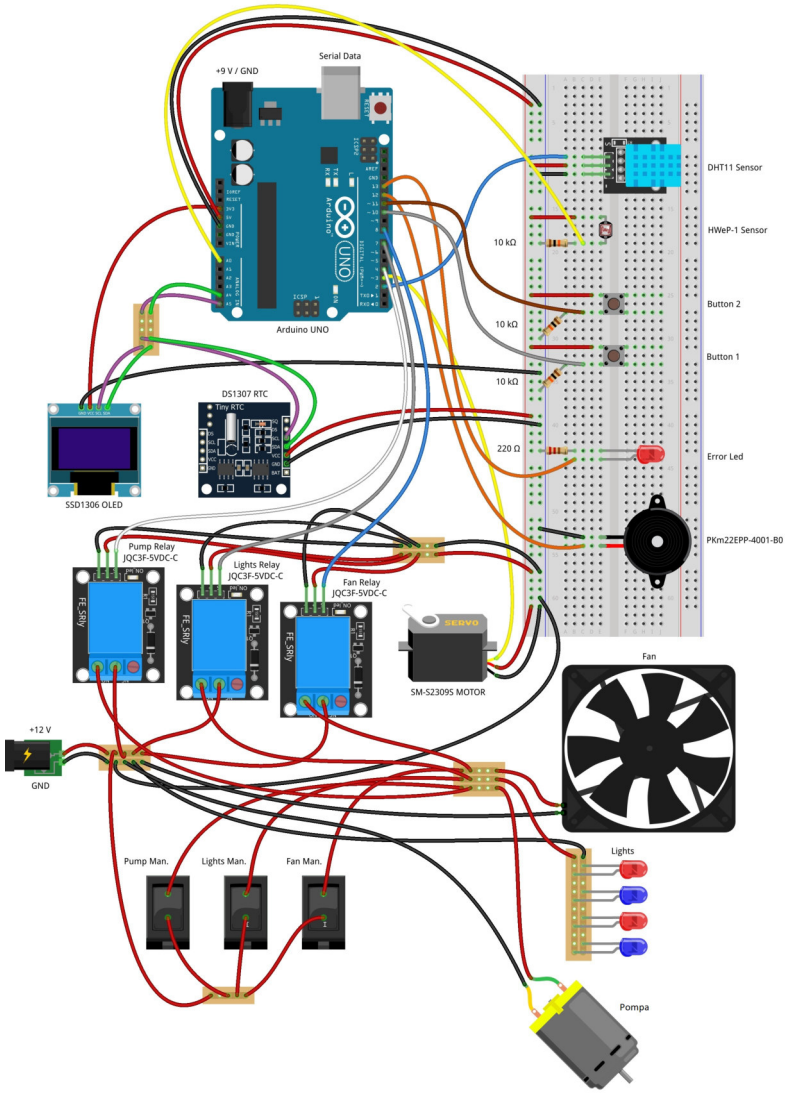
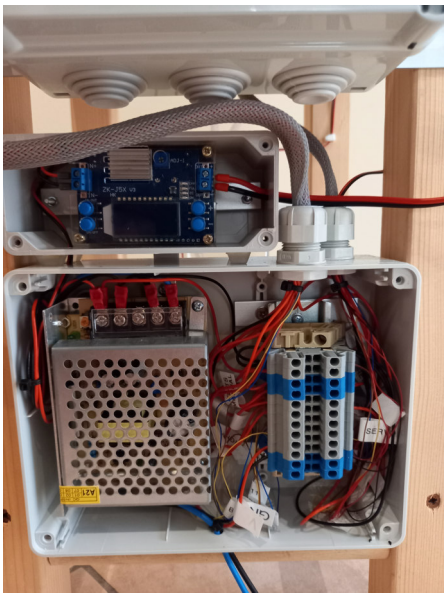


Figura 43: Schema elettrico schematico di quello realizzato nella serra.



(a) *Collegamenti elettrici inferiori.*

(b) *Collegamenti elettrici superiori.*

(c) *Quadro elettrico completo.*

Figura 44: Quadro elettrico installato per il funzionamento della serra.

Si possono notare tutti i collegamenti tra la scheda Arduino e i vari dispositivi, descritti in precedenza. I connettori di alimentazione della scheda e del circuito di potenza a 12V DC non sono collegati verso l'esterno, ma solo con il circuito della serra. La decisione è stata di permettere all'utente la scelta del tipo di alimentazione da fornire, rispettando comunque gli intervalli di tensione imposti. Gli interruttori nel circuito di potenza a 12V DC servono per l'azionamento in manuale della ventola, della pompa e del circuito di illuminazione della serra. In questo modo permettono di *bypassare* il circuito di comando controllato dalla scheda.
A scopo di chiarezza espositiva è riportato anche uno schema elettrico circuitale (*figura 45*), così da restringere l'attenzione ai soli collegamenti.

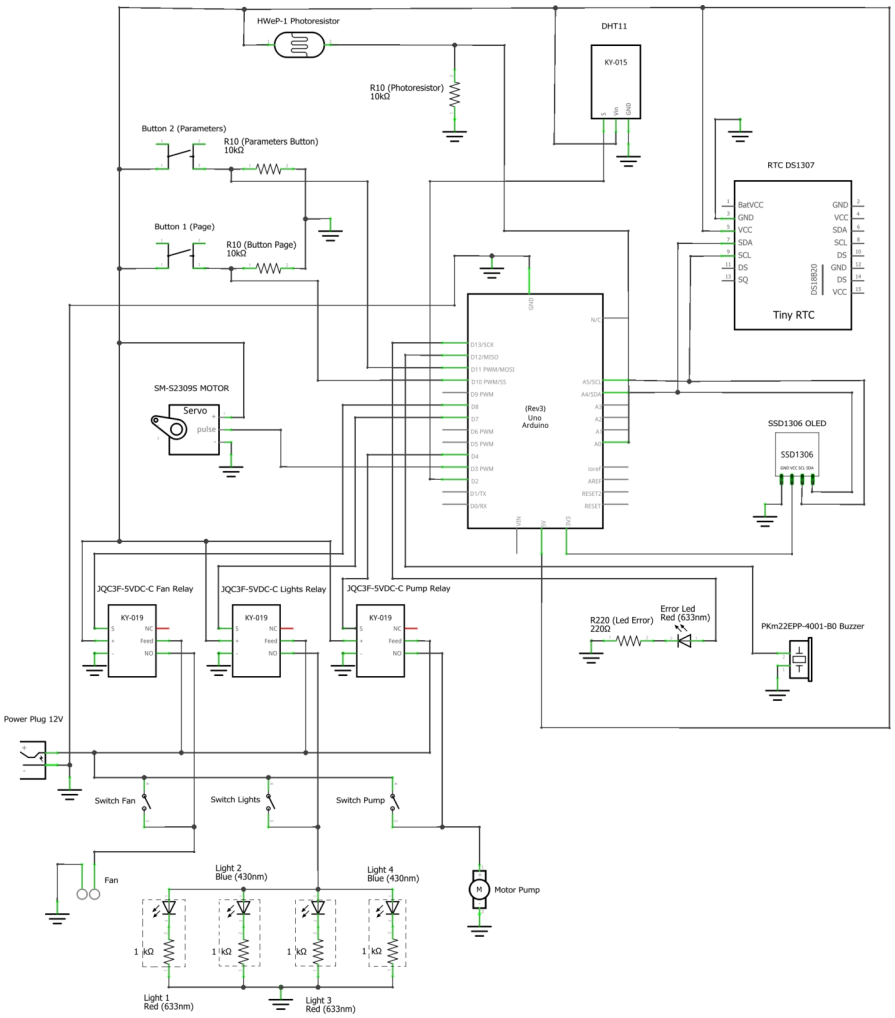


Figura 45: Schema elettrico circuitale.

4.6 Descrizione della serra

Per capire i vari nomi dei componenti all'interno del programma, è di fondamentale importanza dare una descrizione della serra. Come accennato in precedenza, molti dei componenti installati sono stati progettati e fabbricati mediante stampa 3D, per aggirare il problema della difficilissima reperibilità di questi nel mercato, specialmente perché sono realizzati su misura. Un'alternativa sarebbe potuta essere quella di appoggiarsi a qualche azienda privata della zona per realizzare la componentistica necessaria, ma avrebbe sicuramente inciso negativamente sul costo complessivo della serra. Oltre ai componenti realizzati, era di fondamentale importanza anche eseguire un calcolo preciso per verificare il corretto dimensionamento dei dispositivi installati, in questo caso la ventola.

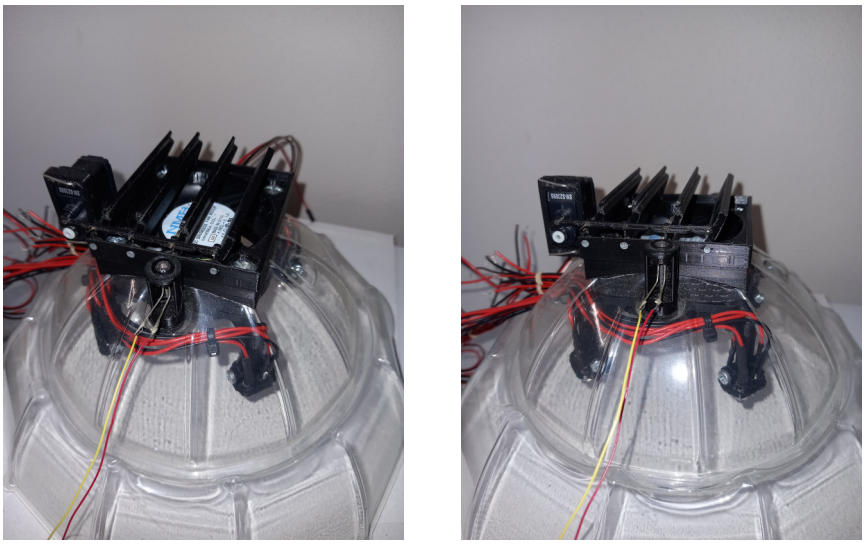
4.6.1 Parti e componenti installate nella serra

I componenti realizzati su misura per la serra non sono molti. Si tratta in particolare di modelli singoli copiati ed installati più volte, poiché il progetto ne richiedeva la presenza di più di uno. I componenti al di fuori del corpo della serra e dei dispositivi elettrici sono stati tutti realizzati *ad hoc* per lo scopo. Nel dettaglio ora i vari componenti realizzati:

1. *Deflettori*. Riportati in *figura 46a*, sono elementi spesso richiamati nel progetto. Sono delle "alette" ottenute attraverso stampa 3D in materiale *PLA* [26]. Sono collegate tra loro attraverso un'astina rigida, ottenuta anch'essa da stampa 3D, che le collega all'albero del servomotore. Il perno di quest'ultimo, ruotando all'interno di un opportuno intervallo, permette di variare l'angolo d'incidenza delle "alette" e quindi regolare il flusso d'aria entrante nella serra. Quando richiesto dal programma, possono chiudersi posizionandosi all'interno della base che le sostiene, ottenuta tramite stampa 3D, e dunque bloccare il ricircolo d'aria della serra. All'occorrenza, possono aprirsi e posizionarsi in verticale permettendo di generare il massimo flusso d'aria in simultanea con l'attivazione della ventola. Solitamente, in condizioni ambientali normali, si posizionano con una angolazione intermedia tra la posizione aperta e chiusa.

2. *Supporto sensore luminosità*. Ottenuto attraverso stampa 3D permette di posizionare il fototransistor, per la rilevazione della quantità di luce, direttamente sopra alla cupola della serra. In questo modo si garantisce un'acquisizione efficacie della quantità di luce.

[26] L'*acido polilattico*, chiamato più comunemente *PLA*, è una tipologia di polimero utilizzato nella stampa 3D



(a) *Deflettori e relativo collegamento al servomotore*

(b) *Supporto fototransistor*

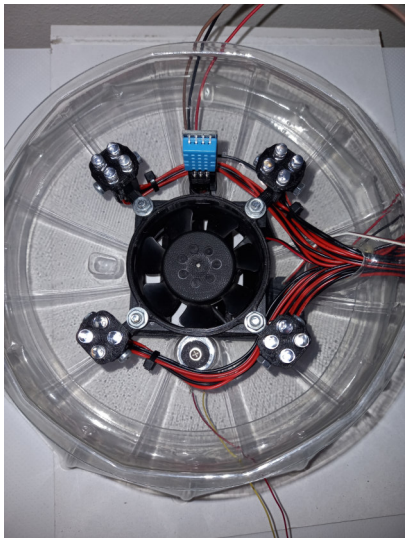
Figura 46: Deflettori in posizione intermedia e supporto fototransistor esterno alla cupola della serra.

3. *Supporto illuminazione serra.* Il volume interno della serra è molto contenuto. Il componente di *figura 47a* è quello che probabilmente ha richiesto più tempo di progettazione. Questo sistema di aggancio dei led per garantire l'illuminazione alle piante, è compatto ma allo stesso funzionale. Nella serra ci sono sei contenitori, quindi altrettante piante coltivate. Il sensore per il rilevamento della temperatura e dell'umidità (in azzurro nella foto) richiedeva spazio, e doveva essere posto all'interno della serra. Per questo motivo, non è stato possibile creare sei stazioni di luci sopra ogni pianta. Ne sono state installate solo quattro, ognuna composta da 4 led, 2 di colore rosso e 2 di colore blu. Il colore dei led permette di fornire lo spettro della luce di cui hanno bisogno le piante per crescere e fiorire. Tutti i connettori dei led sono convogliati verso l'uscita della serra per poi diramarsi verso il circuito elettrico di controllo dedicato.

4. *Supporto sensore temperatura ed umidità.* Come accennato al punto precedente, il sensore per il rilevamento della temperatura e dell'umidità doveva essere posto all'interno della serra, in una posizione il più vicina possibile alle piante, fin dalla loro germogliazione. Per questo motivo il supporto sviluppato in *figura 47b* è di forma allungata verso il basso: in questo modo, il sensore è posto ad una altezza favorevole per non risentire troppo del calore che sale verso l'alto, oppure del

flusso d'aria fresca prodotto durante l'attivazione della ventola, che potrebbero influenzarne le letture dei parametri ambientali.

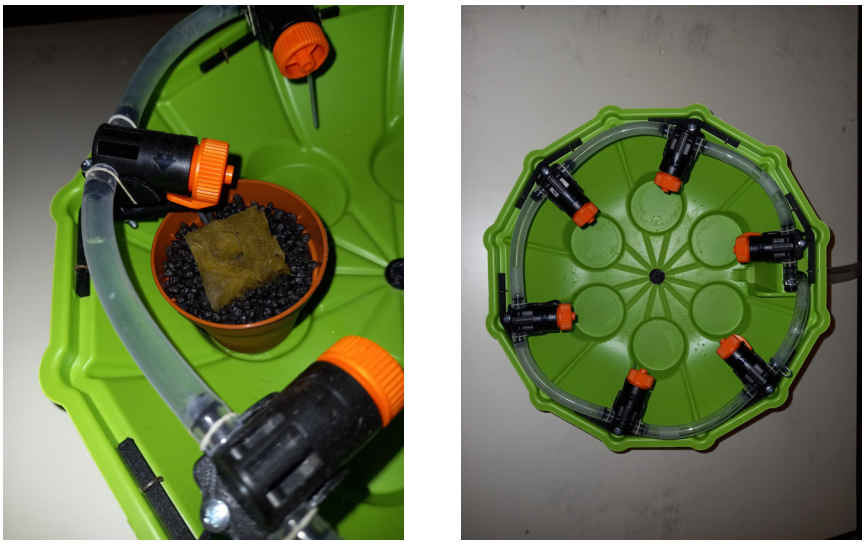
(a) *Supporto illuminazione led.*



(b) *Supporto e relativo sensore di temperatura ed umidità.*

Figura 47: Vista inferiore e laterale supporto illuminazione e sensore temperatura ed umidità.

5. *Supporto gocciolatore e gocciolatori.* Anche se dalla *figura 48a* non sembra, è un componente decisamente complesso. Come già ribadito nei punti precedenti, lo spazio interno alla serra è decisamente ristretto, considerando anche la crescita delle piante. I gocciolatori quindi, dovevano essere posti in una posizione tale che le gocce potessero cadere direttamente sul cubetto di lana di roccia, contenuto nel contenitore sottostante. La disposizione doveva avvenire senza creare un ostacolo fisico durante la crescita della pianta. Per questo motivo, sia la posizione inclinata dei gocciolatori e sia la rotazione rispetto l'asse della serra non sono casuali, ma calcolati per venire incontro alle esigenze della pianta.



(a) *Supporto per gocciolatore e relativa posizione.*

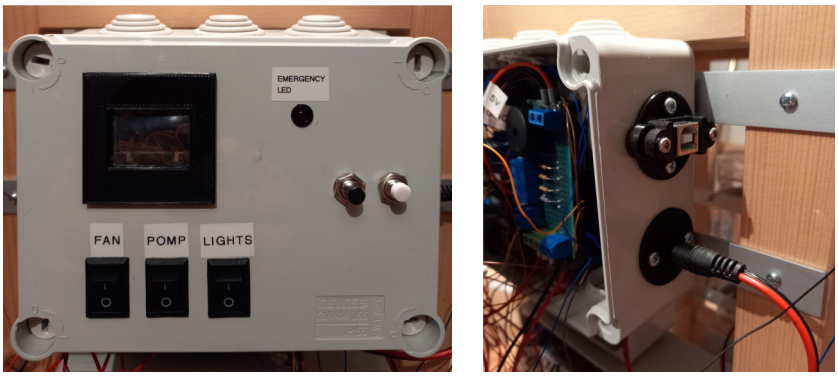
(b) *Gocciolatori installati nella serra.*

Figura 48: Deflettori in posizione intermedia e supporto fototransistor esterno alla cupola della serra.

6. *Contenitore soluzione nutritiva.* Riportato in *figura 49*, presenta una forma a semi-cilindro con una volume di acqua contenibile pari a 1 L . E' stato recuperato e riciclato nel progetto. Avendo la parte posteriore piatta, è facilmente agganciabile alla propria struttura di sostegno. Il volume d'acqua che contiene è perfetto per la serra poiché le percentuali di liquido dei prodotti utilizzati per creare la soluzione nutritiva sono riferite al litro d'acqua. Sul fondo, ancorata attraverso delle ventose, è posizionata la pompa per il sistema di irrigazione della serra.

Figura 49: Contenitore per acqua miscelata alla soluzione nutritiva per le piante della serra.

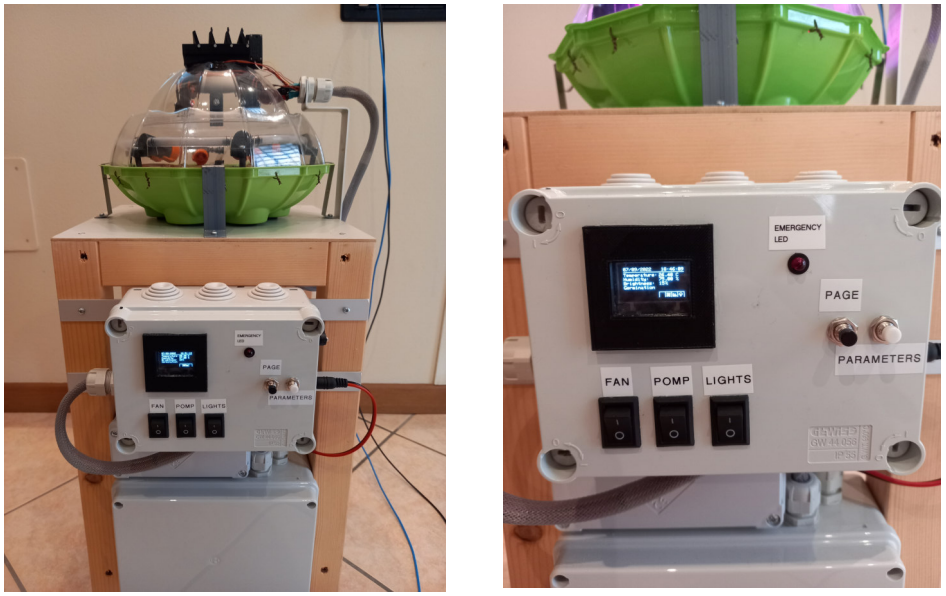
7. *Pannello controllo e pulsantiera.* Riportato in *figura 50*, questo pannello permette l'interfaccia tra l'utente e la serra. Riporta sullo schermo OLED, istante per istante, i dati significativi riguardanti la serra. E' provvisto di due pulsanti per il cambio dei parametri della serra e per il cambio della pagina visualizzata, tre interruttori che permettono il *bypass* del programma automatico utilizzati per l'attivazione manuale dei componenti. Sempre sulla parte frontale è installato anche il led per la segnalazione degli errori dei componenti o del programma. Dietro il pannello, sono fissati la scheda di controllo Arduino, il buzzer sonoro per la segnalazione acustica, i relay e tutti i connettori necessari. Sulla parte laterale invece, illustrata in *figura 50b*, sono presenti i connettori per l'alimentazione della scheda e per il collegamento seriale della scheda Arduino al PC.



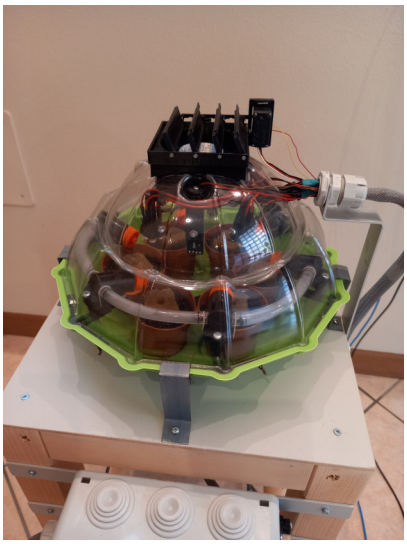
(a) *Pannello di controllo frontale* (b) *Pannello di controllo laterale*

Figura 50: Pannello di controllo della serra.

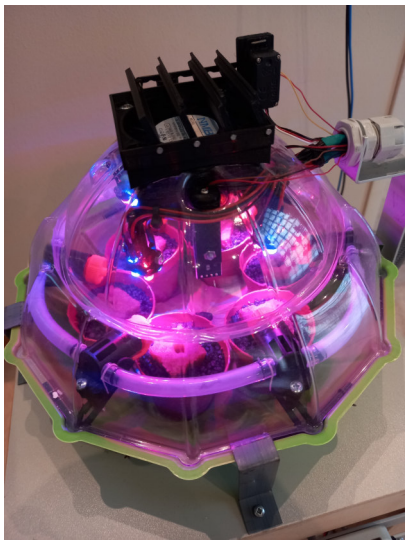
Ora che sono stati descritti tutti i componenti installati e le varie parti della serra, è possibile osservare quest'ultima, assieme al supporto che la sostiene, in *figura 51a*. Verranno inoltre riportate alcune immagini durante il normale funzionamento del programma automatico di gestione della serra. In particolare è possibile osservare il pannello di controllo e i dati riportati nella pagina principale del diplay OLED, in *figura 51b*, e la serra durante il periodo di funzionamento con luce naturale e artificiale, rispettivamente in *figura 51c* e *figura 51d*.



(a) *Serra nel suo complesso.* (b) *Pannello di controllo con dati riportati nel display OLED*



(c) *Serra con luce naturale.*



(d) *Serra con luce artificiale*

Figura 51: Complessivo della serra sviluppata.

Nelle figure appena riportate non è possibile scorgere il contenitore per l'acqua e la soluzione nutritiva. Si trova sul retro della struttura di sostegno, come rappresentato in *figura 52*.

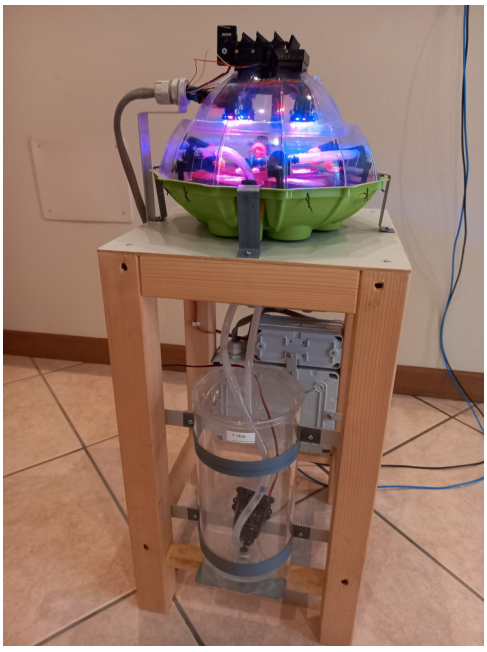
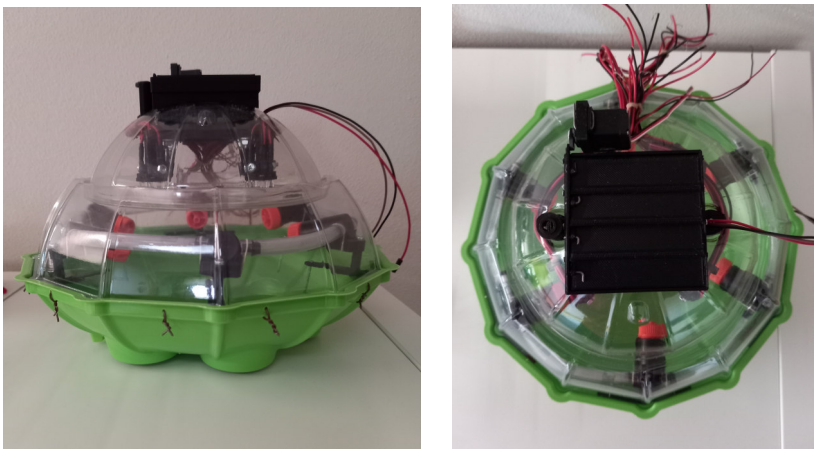


Figura 52: Parte posteriore della struttura di sostegno della serra.

4.6.2 Calcolo del volume

Per il dimensionamento della ventola installata, era necessario capire il volume interno della serra per poter scegliere un componente che fosse adatto allo scopo. La serra idroponica, si presenta come in *figura 53*.



(a) *Vista laterale* (b) *Vista superiore*

Figura 53: Forma della serra idroponica sviluppata.

Si è definita una parametrizzazione del guscio trasparente come grafico di una funzione radiale definita a tratti, su spicchi di ampiezza $\pi/6$ (si può notare che la base della serra è dodecagonale). Il risultato complessivo della parte trasparente sarà questo valore moltiplicato per 12.
In uno spicchio $\theta \in \left[-\frac{\pi}{12}; \frac{\pi}{12}\right]$ consideriamo una porzione di calotta sferica fino ad un raggio fissato r_0 e poi un arco di parabola definito radialmente fino alla retta $\{x = x_1\}$: questo ci darà una superficie regolare a tratti della quale possiamo calcolare il volume del sottografico. Ricordiamo che una parabola con la convessità rivolta verso il basso, valore massimo h assunto al punto x_0 e che si annulla in x_1, ha l'espressione:

$$h\left(1 - \frac{(x - x_0)^2}{(x_1 - x_0)^2}\right)$$

Il dominio della parametrizzazione è l'insieme, scritto come forma $r-semplice$:

$$D := \left\{(r, \theta) \in [0, +\infty) \times \left[-\frac{\pi}{12}, \frac{\pi}{12}\right] : 0 \leq r \leq r\cos\theta\right\}$$

che dividiamo in:

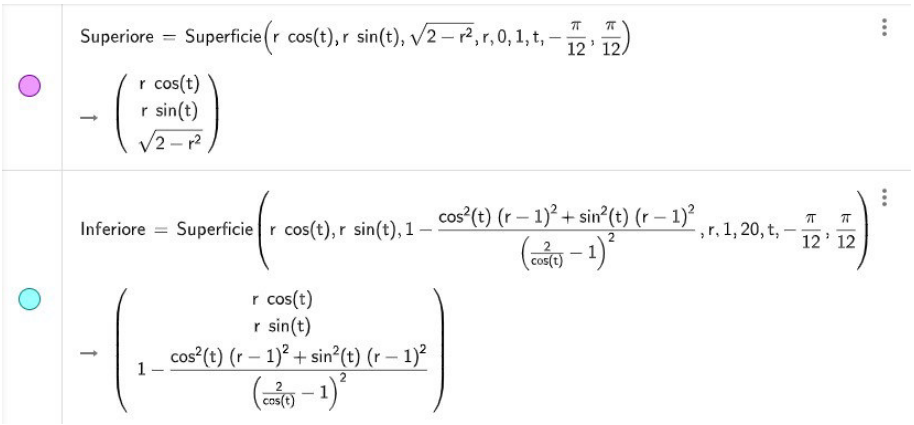
$$D_1 = \left[0, r_0\right] \times \left[-\frac{\pi}{12}, \frac{\pi}{12}\right]$$

$$D_2 = \left\{(r,\theta) \in D : r_0 < r \leq x_1 \cos\theta\right\}$$

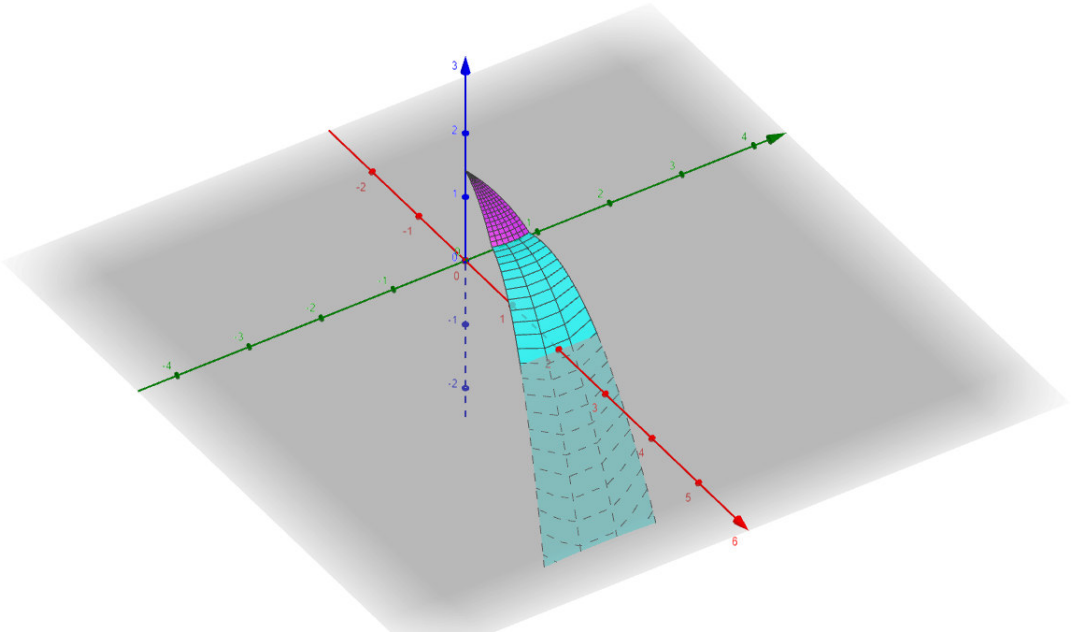
Definiamo la superficie regolare a tratti come:

$$\underline{\sigma}(r,\theta) = \begin{cases} (r\cos\theta, r\sin\theta, \sqrt{r_0^2 - r^2}) & (r,\theta) \in D_1 \\ \left(r\cos\theta, r\sin\theta, \frac{r_0}{2}\left(1 - \frac{(r-r_0)^2}{\left(\frac{x_1}{\cos\theta} - r_0\right)^2}\right)\right) & (r,\theta) \in D_2 \end{cases}$$

dove la prima parte è una parametrizzazione di una porzione di sfera centrata nell'origine e di raggio r_0 limitatamente agli angoli $\theta \in \left[-\frac{\pi}{12}, \frac{\pi}{12}\right]$ e $\varphi \in \left[0, \frac{\pi}{3}\right]$, dove φ è l'angolo formato dal vettore posizione con l'asse z. La seconda parte della parametrizzazione è, ristretta ad un angolo fissato θ, una porzione di parabola dal punto $(r_0\cos\theta, r_0\sin\theta, \frac{r_0}{2})$ fino al punto $(x_1, x_1\tan\theta, 0)$. I parametri r_0 e x_1 sono scelti in base ai valori della serra.



(a) *Parametrizzazione in GeoGebra della parte superiore della serra*



(b) *Risultato della parametrizzazione definita in precedenza*

Figura 54: Parametrizzazione dei domini in GeoGebra.

I calcoli dei volumi dei sottografici vengono calcolati separatamente, entrambi in coordinate cilindriche:

1. Relativamente alla porzione di sfera, dunque per il dominio D_1, calcoliamo:

$$\int_{-\frac{\pi}{12}}^{\frac{\pi}{12}} \int_0^{r_0} r\sqrt{r_0^2 - r^2}\, dr d\theta = \frac{\pi}{6}\left(-\frac{1}{3}(r_0^2 - r^2)^{\frac{3}{2}}\right)\Big|_0^{r_0} = \frac{\pi}{18} r_0^3$$

2. Per il dominio D_2 invece, calcoliamo:

$$r_0 \int_0^{\frac{\pi}{12}} \int_{r_0}^{\frac{x_1}{\cos\theta}} r\left(1 - \frac{(r - r_0)^2}{(\frac{x_1}{\cos\theta} - r_0)^2}\right) dr d\theta$$

$$= r_0 \int_0^{\frac{\pi}{12}} \int_{r_0}^{\frac{x_1}{\cos\theta}} r - \frac{(r^3 - 2r_0 r^2 + r_0^2 r)}{(\frac{x_1}{\cos\theta} - r_0)^2} dr d\theta$$

$$= r_0 \int_0^{\frac{\pi}{12}} \frac{1}{2}\frac{x_1^2}{\cos^2\theta} - \frac{r_0^2}{2} - \frac{\frac{1}{4}\frac{x_1^4}{\cos^4\theta} - \frac{2}{3} r_0 \frac{x_1^3}{\cos^3\theta} + \frac{1}{2} r_0^2 \frac{x_1^2}{\cos^2\theta} - \frac{1}{4} r_0^4 + \frac{2}{3} r_0^4 - \frac{1}{2} r_0^4}{(\frac{x_1}{\cos\theta} - r_0)^2} d\theta$$

$$= r_0 \int_0^{\frac{\pi}{12}} \frac{\frac{1}{4}\frac{x_1^4}{\cos^4\theta} - \frac{1}{3} r_0 \frac{x_1^3}{\cos^3\theta} - \frac{1}{2} r_0^2 \frac{x_1^2}{\cos^2\theta} + r_0^3 \frac{x_1}{\cos\theta} - \frac{5}{12} r_0^4}{(\frac{x_1}{\cos\theta} - r_0)^2} d\theta \qquad (1)$$

$$= r_0 \int_0^{\frac{\pi}{12}} \frac{1}{4}\frac{x_1^2}{\cos^2\theta} + \frac{1}{6} r_0 \frac{x_1}{\cos\theta} + \frac{5}{12} r_0^2 d\theta$$

$$= r_0 \left(\frac{1}{4} x_1^2 \tan\frac{\pi}{12} + \frac{5}{144}\pi r_0^2 + \frac{1}{6} r_0 x_1 \int_0^{\tan\frac{\pi}{24}} \frac{2}{1-t^2} dt \right) \qquad (2)$$

$$= \frac{1}{4} r_0 x_1^2 \tan\frac{\pi}{12} + \frac{1}{6} r_0^2 x_1 \log\frac{1+\tan\frac{\pi}{24}}{1-\tan\frac{\pi}{24}} + \frac{5}{144}\pi r_0^3. \qquad (3)$$

Per passare da (1) a(2) abbiamo usato il fatto che:

$$\frac{1}{4}a^4 - \frac{1}{3}a^3 b - \frac{1}{2}a^2 b^2 + ab^3 - \frac{5}{12}b^4 = \big(a-b\big)^2 \left(\frac{1}{4}a^2 + \frac{1}{6}ab + \frac{5}{12}b^2 \right).$$

Per la risoluzione di (2) abbiamo usato le formule trigonometriche di bisezione per riscrivere

$$\cos\theta = \frac{\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}}{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}} = \frac{1-\tan^2\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}}$$

e abbiamo poi fatto il cambio di variabile con $t = \tan\frac{\theta}{2}$. Infine, per ottenere (3) abbiamo riscritto:

$$\frac{2}{1-t^2} = \frac{1}{1-t} + \frac{1}{1+t}.$$

Nel complesso, estendendo ora il volume a tutte e dodici le fette del dodecagono, concludiamo che il volume della calotta superiore cercato è:

$$V_{CalottaSuperiore} = \frac{13}{12}\pi r_0^3 + 2r_0^2 x_1 \log\frac{-1-\sqrt{3}+2\sqrt{2+\sqrt{3}}}{3+\sqrt{3}-2\sqrt{2+\sqrt{3}}} + 3r_0 x_1^2 \frac{3-\sqrt{3}}{3+\sqrt{3}} \qquad (4)$$

3. La parte inferiore della serra, di colore verde in *figura 53a*, è uguale al dominio D_2 in quanto hanno entrambi la stessa forma. Otterremo dunque:

$$V_{CalottaInferiore} = \frac{1}{4} r_1 x_1^2 \tan\frac{\pi}{12} + \frac{1}{6} r_1^2 x_1 \log\frac{1+\tan\frac{\pi}{24}}{1-\tan\frac{\pi}{24}} + \frac{5}{144}\pi r_1^3$$

Moltiplicando anche questa parte per 12, in quanto anche la base è dodecagonale, otteniamo:

$$= \frac{5}{12}\pi r_1^3 + 2r_1^2 x_1 \log \frac{-1-\sqrt{3}+2\sqrt{2+\sqrt{3}}}{3+\sqrt{3}-2\sqrt{2+\sqrt{3}}} + 3r_1 x_1^2 \frac{3-\sqrt{3}}{3+\sqrt{3}} \quad (5)$$

Sostituendo ora i valori corretti in 4, dunque un valore di $r_0 = 80mm$ e $x_1 = 115mm$, si ottiene:

$$V = 2762768,367mm^3 \quad (6)$$

Per la parte inferiore invece, assumiamo un valore di $r_1 = 80mm$ (come r_0) e manteniamo lo stesso valore di x_1. Il volume di 5 sarà dunque:

$$V = 1690438,074mm^3 \quad (7)$$

Il volume totale sarà quindi la somma dei due, ovvero:

$$V_{totale} = 4453206,441mm^3 =\sim 4453206mm^3 \quad (8)$$

Considerando ora il volume dei contenitori interni alla serra. Il singolo contenitore ha un volume $V_{contenitori} = 112221mm^3$. Considerando tutti sei i contenitori, il volume da sottrarre a 8 è $676326mm^3$, quindi complessivamente il volume d'aria interno sarà di:

$$V_{aria} = 4453206 - 676326 = 3776880mm^3 =\sim 0,00377m^3 \quad (9)$$

Questo riportato in realtà è un valore sovrastimato. Infatti, tenendo in considerazione anche il sistema di irrigazione composto da tubi, gocciolatori e relativi supporti, lo spazio occupato dal sistema di illuminazione e dalle piante stesse, il volume d'aria effettivo cala. Riprendendo ora il datasheet della ventola di *figura 38*, si osserva essere riportato un valore di $0,5m^3/min$ per la portata d'aria, ampiamente sufficiente per arieggiare la serra, considerando il volume di aria che contiene.

4.7 Elementi di innovatività del progetto

Nel WEB sono sicuramente presenti molti progetti di serre automatizzate che possono essere più o meno complessi. E' bene specificare le differenze che sono state introdotte rispetto a molti di questi, in modo da distinguere bene le scelte compiute in fase di progettazione e realizzazione. Innanzitutto è bene ricordare che si tratta di un prototipo, sviluppato per poter rilevare dati sui tempi di crescita delle piante usando la tecnica idroponica, e sul beneficio introdotto dall'automatizzazione del processo di controllo. Questo aspetto è fondamentale perché implica una costruzione più semplice e ridotta nella struttura della serra. Nel caso in cui i dati fossero confermati,

l'obiettivo sarà quello di applicare la tecnica ed il controllo in un ambiente più grande e complesso, sviluppando le dimensioni della serra e il relativo programma di controllo. Un aspetto da non trascurare è anche lo schermo OLED installato nel progetto che, a differenza dei display LCD installati in quasi tutte le serre presenti nel WEB, garantisce una visualizzazione dei dati più completa e personalizzabile. Permette di creare figure stilizzate per rappresentare i componenti con la possibilità di personalizzare lo spazio e la disposizione delle informazioni sullo schermo. I dati riportati inoltre, non si limitano ai semplici parametri ambientali (temperatura, umidità e luminosità), ma vengono riportati anche i parametri riguardanti i vari cicli di crescita delle piante. Questi sono ad esempio la temperatura e l'umidità minime e massime, la luminosità minima, la fase di crescita delle piante, le ore di luce giornaliere da fornire e i relativi orari di accensione e spegnimento dell'illuminazione led. Il programma, oltre a riportare i dati sul display OLED, permette anche una modalità di *DEBUG*. Attraverso quest'ultima, l'operatore è in grado di visualizzare tutte le informazioni che il display OLED fornisce accedendo ulteriormente all'andamento logico del programma, visualizzando ulteriori informazioni che garantiscono un facile *debugging* dell'applicazione. Oltre alla parte logica e programmabile, le differenze con i progetti già presenti nel WEB riguardano anche l'aspetto meccanico e costruttivo. La struttura della serra è stata acquistata, ma tutte le parti sono state progettate e stampate in 3D a seconda delle esigenze. Simula la forma e la geometria di alcune serre idroponiche verticali, alle quali sono stati dedicati anni di studi e progettazione. Un progetto con forma simile è il *"Progetto Plantagon"*, una serra idroponica verticale completamente automatizzata (*figura 55*) che dovrebbe abbattere gli ostacoli riguardanti l'impatto ambientale dovuto alle tecniche di coltivazione, e che permetterebbe una produzione di colture sane e fresche nelle grandi metropoli. Sempre riguardo alla forma e alle dimensioni, un ambiente interno con uno spazio ristretto garantisce un'uniformità tra la zona più bassa della serra e quella più in alto permettendo di rilevare, grazie ad un solo sensore, la temperature e l'umidità presenti. La forma inoltre, ha permesso di installare i dispositivi per la ventilazione in alto, dove tende a risiedere l'aria più calda. La posizione è inoltre strategica in quanto permette un'uscita dell'aria più calda, e dunque meno densa, senza l'attivazione della ventola ma lasciando aperti i soli deflettori. In questo modo, naturalmente, l'aria calda sale e dunque è possibile forzare solo quella in ingresso, attraverso l'uso della ventola, per immettere aria più fresca proveniente dall'esterno. Sempre riguardo la ventilazione, la serra offre una gestione temporizzata. Con temperatura ed umidità che non eccedono dai limiti imposti, la ventilazione viene attivata ad intervalli regolari in modo da garantire sempre un apporto di aria fresca. Altra caratteristica che viene introdotta, rispetto a molti progetti simili, è la parte sull'illuminazione delle piante e la sua gestione. Spesso, per comodità e anche per disinformazione, si crede che fornendo luce

Figura 55: Progetto “Plantagon”, serra idroponica verticale con impatto ambientale ridotto, con costruzione in aree urbane. [15]

per 24 ore al giorno alle piante ne favorisca una crescita più veloce. Come già ribadito, questo fenomeno genera stress alle stesse e non è benefico nei loro confronti. Per questo motivo è stato implementato un controllo che, a seconda della fase di crescita della pianta, garantisce un apporto giornaliero di luce artificiale, in caso di mancanza di quella naturale, rispettando le esigenze della pianta. Questo controllo tiene conto del periodo dell’anno in modo da spostare i tempi di accensione e spegnimento dei led così da garantire il maggior quantitativo di luce solare possibile tutto l’anno, in ottica di risparmio energetico. Questi sono gli aspetti chiave del progetto, uniti alla possibilità per l’utente di personalizzare i parametri a livello software, creando delle ricette personalizzate a seconda del tipo di coltura coltivata.

5 Controllo della serra e programma implementato per la sua gestione automatica

Oltre alla realizzazione stessa della serra, con la scelta dei componenti e la progettazione e realizzazione di alcuni di questi, il programma per la gestione automatica è sicuramente il punto cardine del progetto. Permette di gestire completamente la serra idroponica grazie al rilevamento dei parametri, l'elaborazione di questi e la successiva attuazione delle misure necessarie al controllo. Data la complessità del programma, scritto in codice Arduino, è di aiuto per prima cosa riportare un *flowchart* del suo funzionamento.

5.1 Flowchart logico del programma implementato

Per avere una maggiore chiarezza sul funzionamento del programma, è utile riportare più *flowcharts* in modo da introdurne gradualmente il funzionamento. In *figura 56* viene illustrato il metodo di funzionamento del programma, in modo semplificato ma utile alla sua successiva comprensione. Per prima cosa, ad ogni accensione della scheda, vengono caricate le librerie necessarie e viene allocato dello spazio in memoria per le variabili del programma. In seguito, viene eseguita la funzione *SETUP* della scheda Arduino, che permette di impostare lo stato dei pin ed inizializzare i componenti. Vengono inoltre caricati i parametri della serra, che ne permettono il controllo a seconda dei valori ambientali rilevati dai sensori. Queste prime azioni sono di fondamentale importanza per il seguito del programma in quanto, se qualche componente non è correttamente collegato, il programma automatico viene arrestato per motivi di sicurezza. Dopo queste prime azioni, viene eseguita la funzione *LOOP* che permette di eseguire in sequenza tutte le azioni necessarie per il controllo della serra, in modo ciclico. Queste due funzioni infatti, sono definite di default nel microcontrollore Arduino, e sono indispensabili per il funzionamento di qualsiasi programma caricato. La funzione *SETUP* viene eseguita solo all'avvio della scheda mentre la funzione *LOOP* continua ad essere eseguita in modo ciclico per ripetere il programma caricato. L'ordine delle operazioni da svolgere non è banale, ma rispetta uno schema che permetta di accedere a dei dati sempre aggiornati ad ogni ciclo prima di attuare le operazioni necessarie per il controllo della serra. Per prima cosa vengono letti i parametri ambientali, gli stati degli ingressi e delle uscite. Vengono poi elaborati dalla scheda i dati raccolti e confrontati con i parametri caricati in precedenza. Solo alla fine, vengono generati i segnali che permettono di modificare lo stato delle uscite. Solo in seguito la funzione di *LOOP* viene eseguita nuovamente. L'ordine delle operazioni ed il funzionamento appena descritti altro non sono quelli eseguiti della gran parte dei sistemi di controllo.

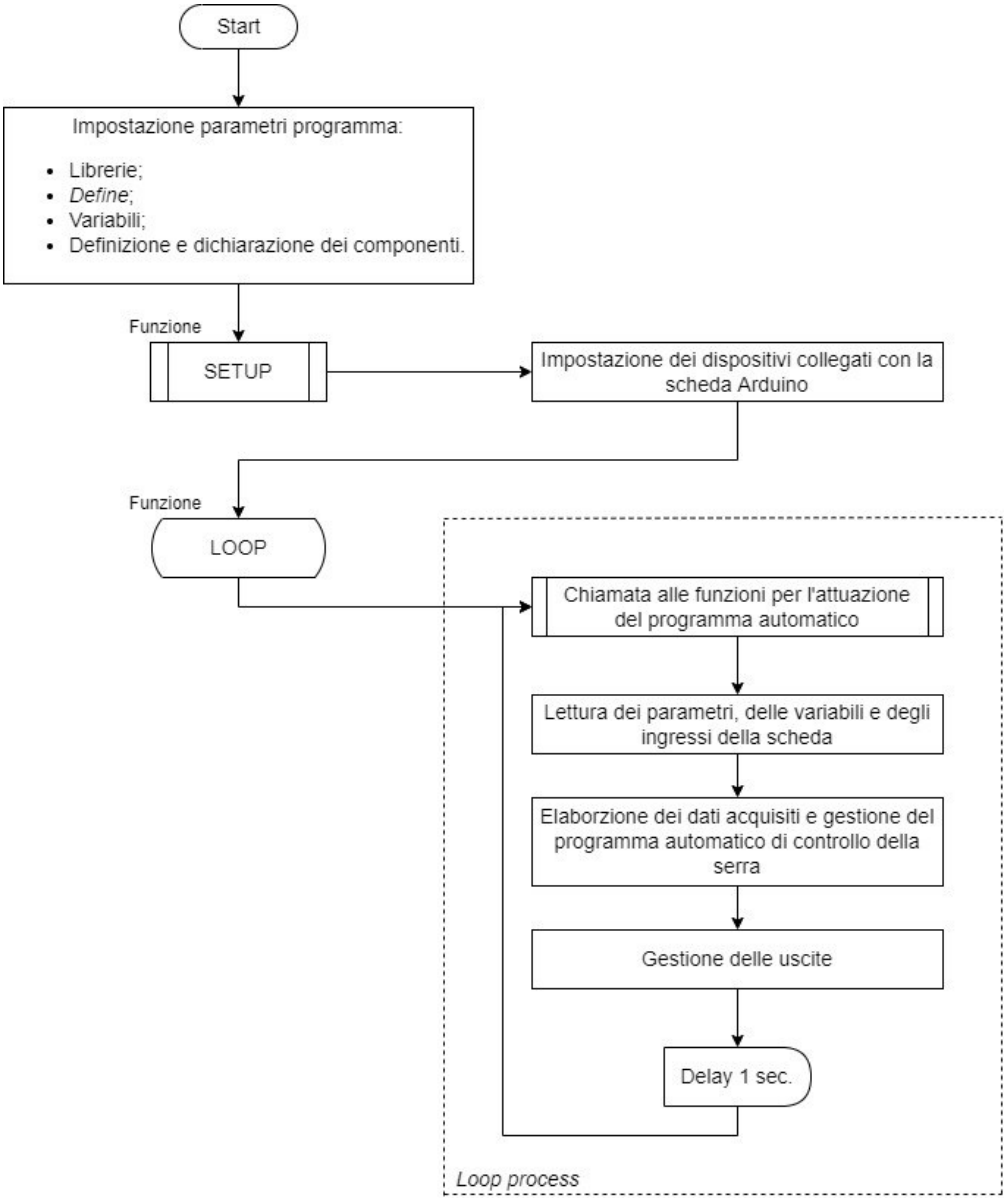
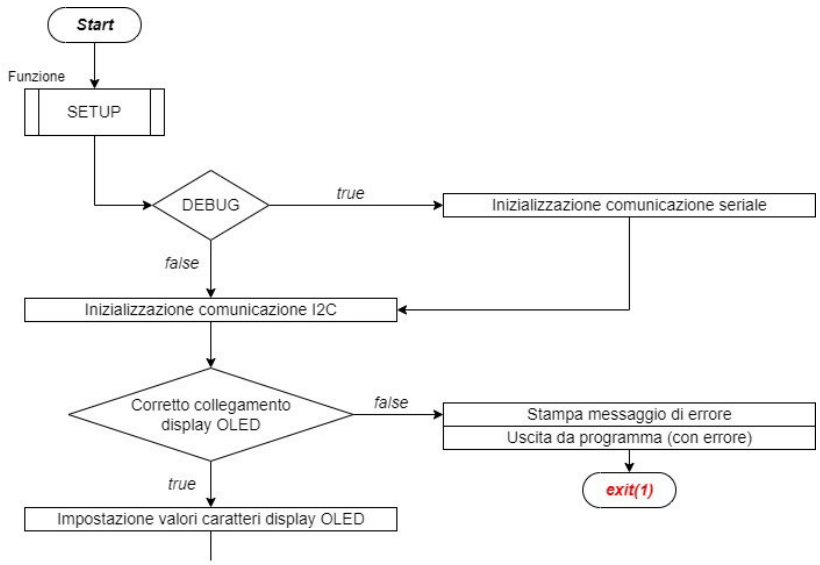


Figura 56: Flowchart schematico del funzionamento della serra.

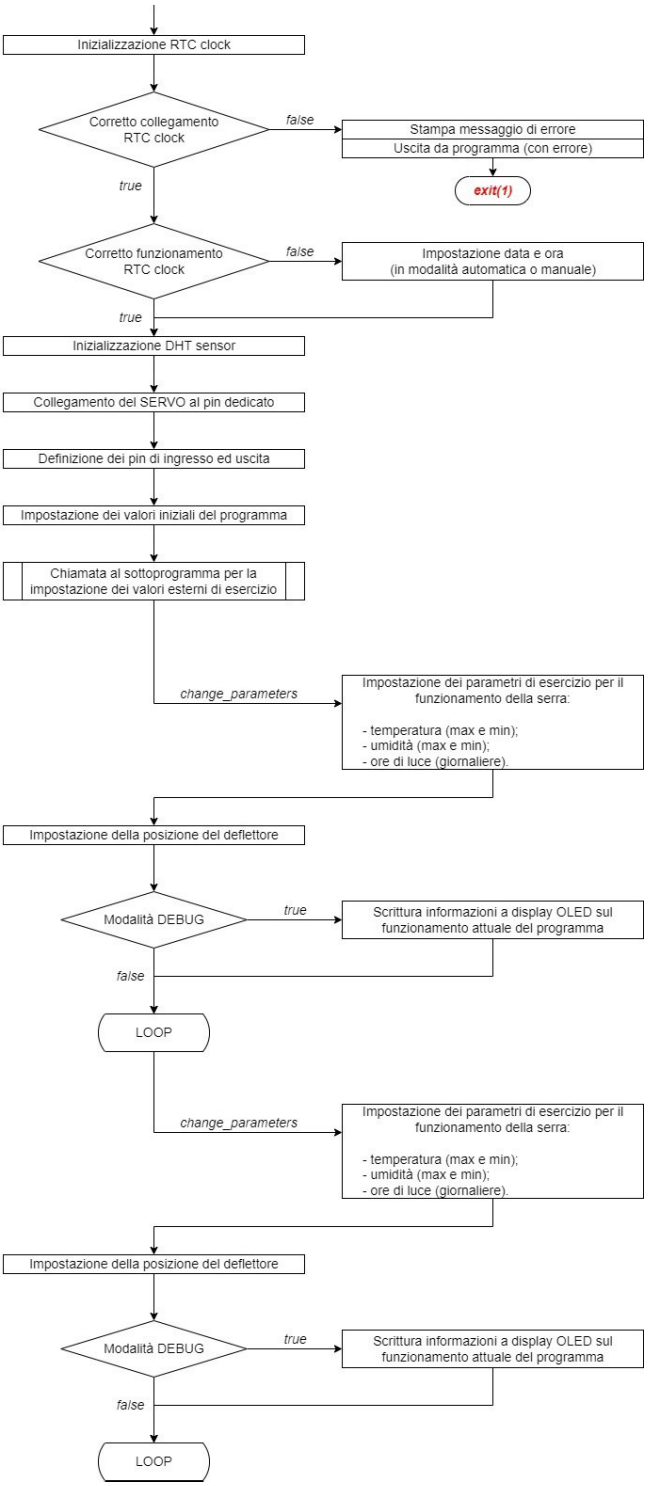
Dopo una prima introduzione al funzionamento del programma, è possibile vedere nel dettaglio le due macro parti implementate per la gestione automatica della serra, tralasciando le operazioni svolte in automatico dalla scheda Arduino per permettere questo. Le due parti sono: la funzione *SETUP* e la funzione *LOOP*.

5.1.1 Funzione SETUP

Come descritto in precedenza, in questa funzione vengono eseguite tutte le operazioni necessarie per configurare la scheda, i dispositivi collegati e il programma stesso alla successiva gestione automatica. Viene controllato il corretto funzionamento dei dispositivi tramite test di comunicazione tra questi e la scheda Arduino. In caso di uno o più errori, il programma viene interrotto e si procede all'aborto. Il fatto che un sensore o un componente non siano correttamente collegati implica una scorretta gestione del programma, e dunque deve essere segnalato e interrotto il processo. Se i dispositivi sono correttamente collegati alla scheda, si procede con l'impostazione dei vari pin, definendone gli ingressi e le uscite e lo stato logico di queste. Solo in seguito, vengono caricati i parametri necessari al funzionamento del programma automatico e gestita la modalità per il *DEBUG*. Quest'ultima permette di ricevere, a terminale, ulteriori informazioni rispetto quelle visualizzate sullo schermo OLED. In particolare, come suggerisce il nome, è molto utile in fase di gestione e risoluzione dei *bug* del programma o per la verifica di eventuali errori dei dispositivi collegati. La funzione *SETUP* viene eseguita solamente una volta dopo l'accensione della scheda. In *figura 57* vengono riportate le operazioni che questo svolge.



(a) *Pagina 1*

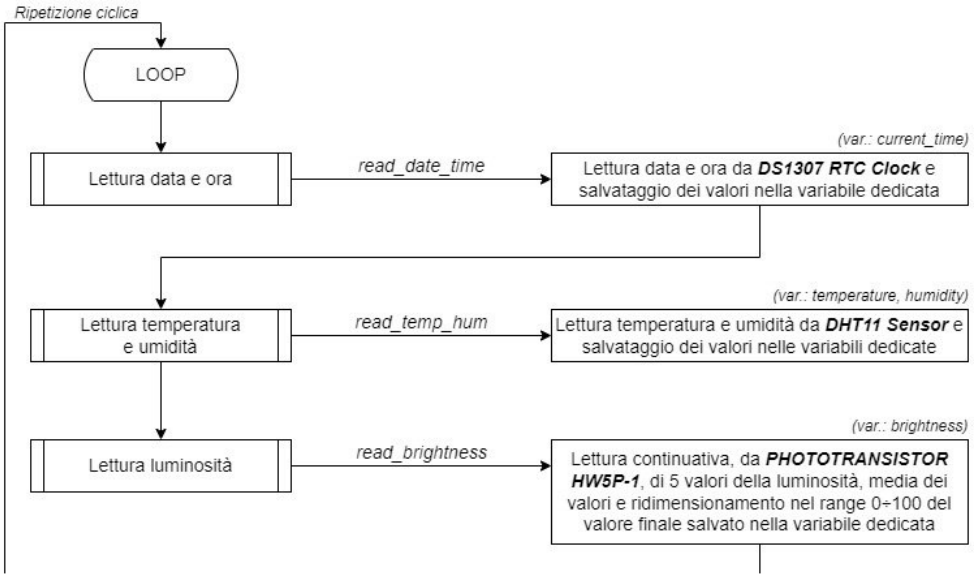


(b) *Pagina 2*

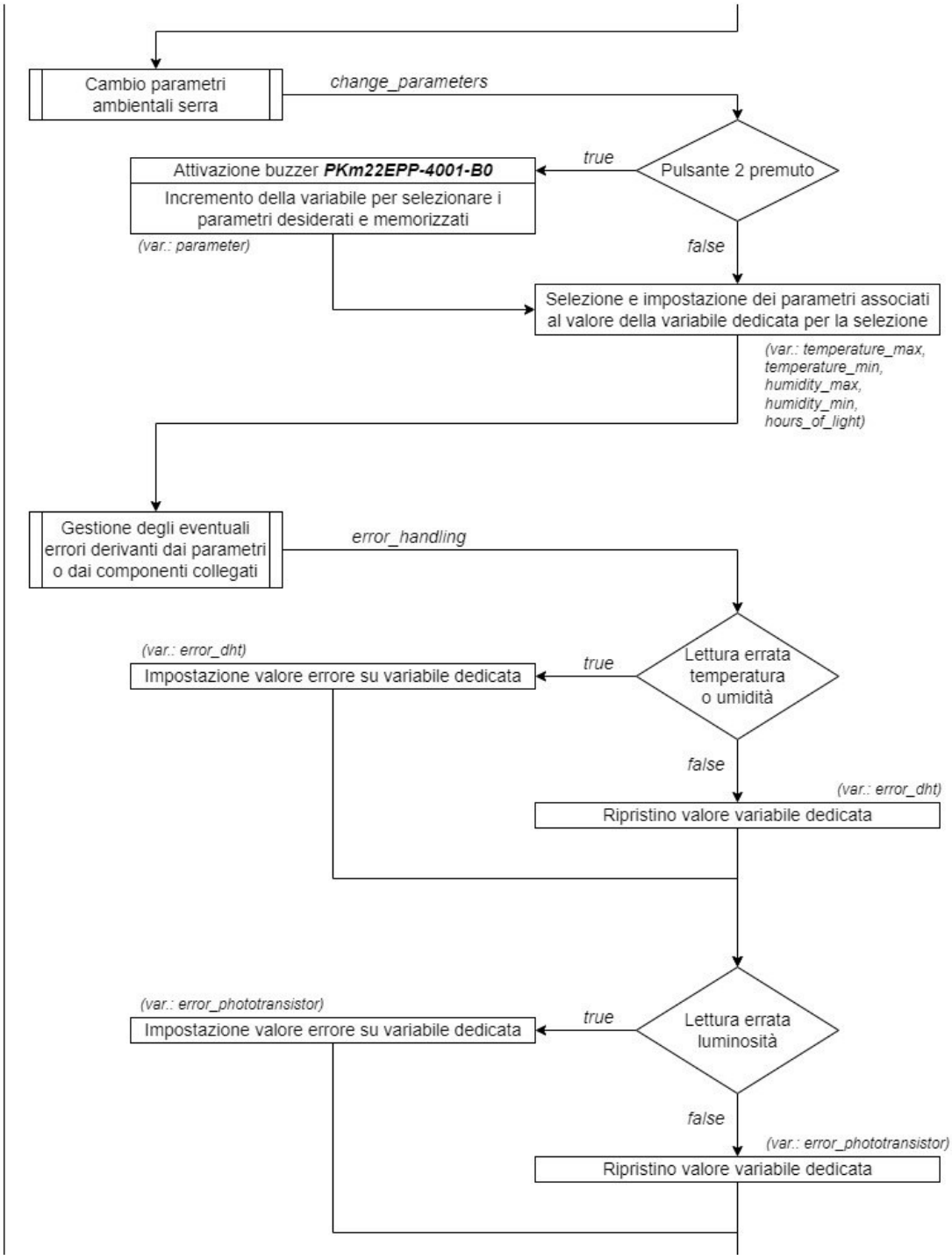
Figura 57: Flowchart della funzione SETUP del programma della serra idroponica.

5.1.2 Funzione LOOP

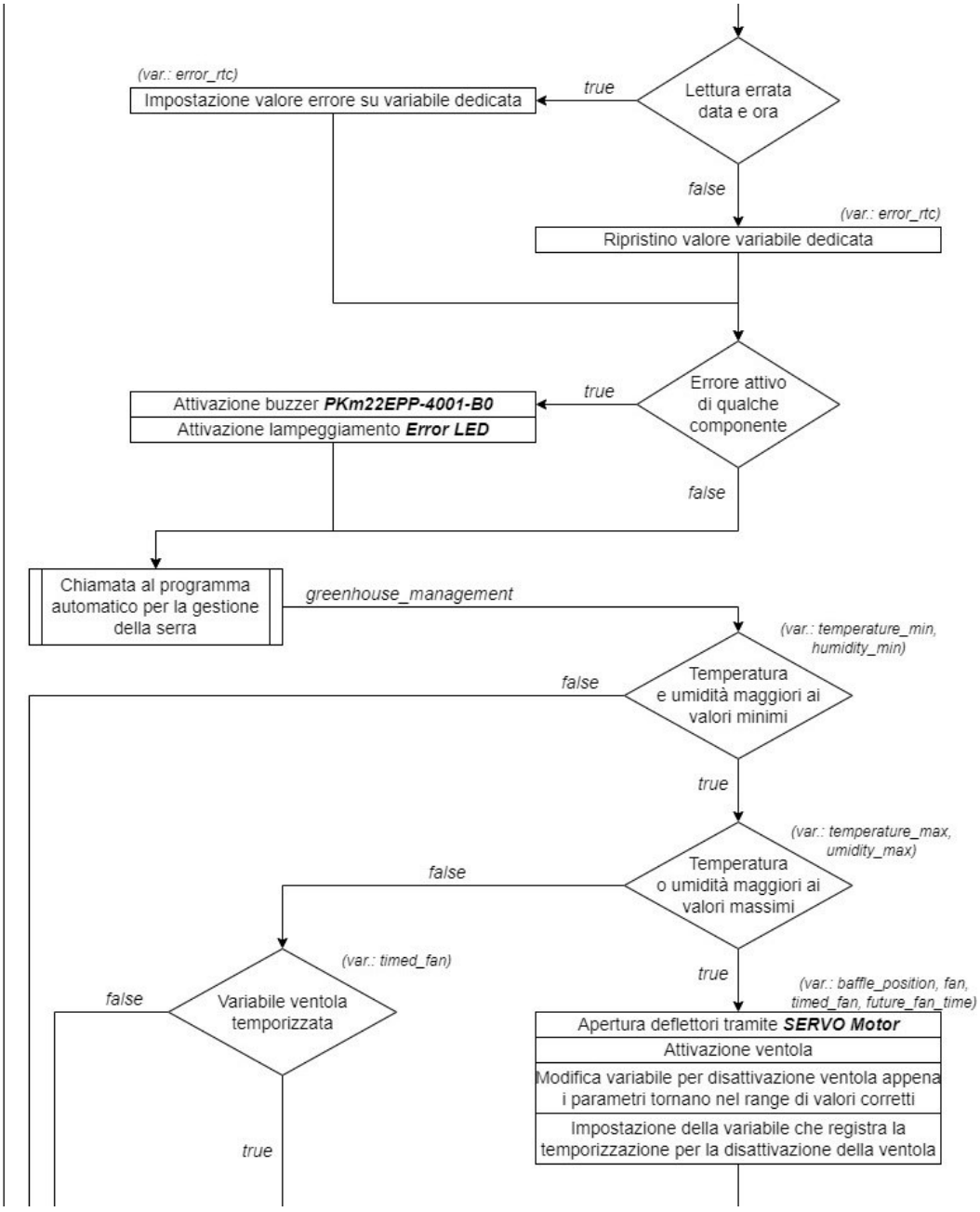
Per caratteristica di Arduino, la funzione *LOOP* viene eseguita in modo ciclico. Qui risiedono tutti i sottoprogrammi necessari per la gestione automatica della serra. Come accennato nell'introduzione del capitolo, la sequenza di esecuzione dei sottoprogrammi non è casuale, ma segue uno specifico ordine logico. Per prima cosa, la scheda Arduino legge data e ora dal RTC collegato. In seguito, legge i parametri ambientali, aggiornando il valore memorizzato fino a quell'istante nelle rispettive variabili all'interno del programma. Solo dopo procede ad impostare i parametri della serra, mantenendo quelli precedenti o eventualmente aggiornandoli a seconda delle richieste dell'utente. Prima di generare un segnale di uscita verso gli attuatori, viene controllata la presenza o meno di errori nei dispositivi collegati; se è presente un mal funzionamento, viene riportato lo stato di errore del programma e non vengono modificate le uscite. Solo in seguito viene richiamato il programma per la gestione della serra e vengono modificate le uscite a seconda dei segnali generati. Come ultima operazione, la scheda invia tutti i dati significativi allo schermo OLED oppure a terminale in caso di attivazione della modalità *DEBUG*. Il ritardo di 1 secondo posto al termine della funzione *LOOP* viene posto poiché il sensore *DHT11* ha tempo di *refresh* di 1 Hz: devo dunque attendere il tempo corretto prima di acquisire nuovamente i dati ambientali aggiornati. Mantenere un tempo ciclo variabile non è la soluzione preferibile, ma in questa situazione non si poteva fare altrimenti, a causa delle limitazioni tecnologiche imposte. In *figura 58*, è riportato il flowchart della funzione *LOOP* del programma per la gestione della serra.



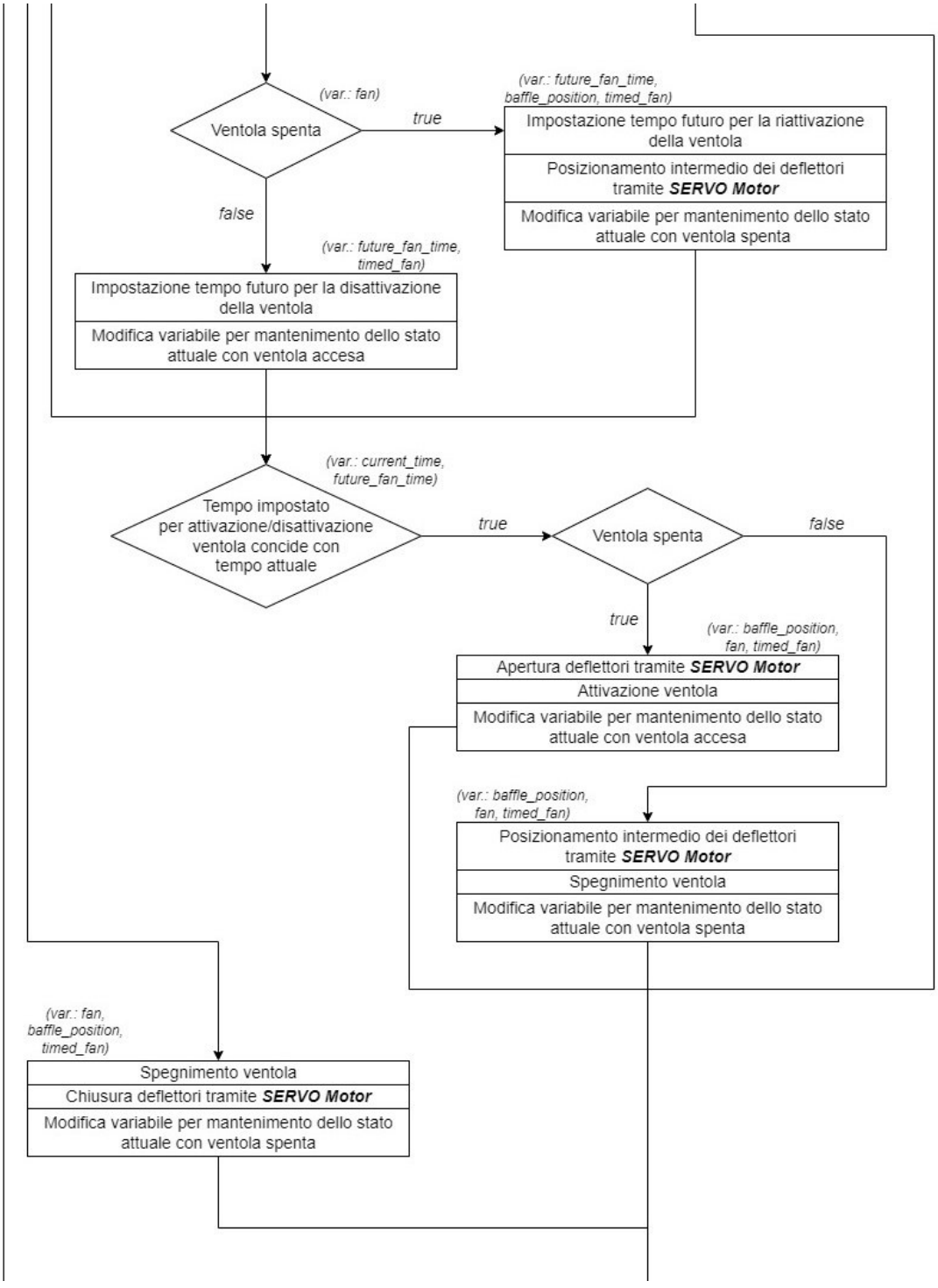
(a) *Pagina 1*


Cambio parametri ambientali serra
change_parameters
Pulsante 2 premuto
true
Attivazione buzzer PKm22EPP-4001-B0
Incremento della variabile per selezionare i parametri desiderati e memorizzati
(var.: parameter)
false
Selezione e impostazione dei parametri associati al valore della variabile dedicata per la selezione
(var.: temperature_max, temperature_min, humidity_max, humidity_min, hours_of_light)
Gestione degli eventuali errori derivanti dai parametri o dai componenti collegati
error_handling
Lettura errata temperatura o umidità
true
(var.: error_dht)
Impostazione valore errore su variabile dedicata
false
(var.: error_dht)
Ripristino valore variabile dedicata
Lettura errata luminosità
true
(var.: error_phototransistor)
Impostazione valore errore su variabile dedicata
false
(var.: error_phototransistor)
Ripristino valore variabile dedicata

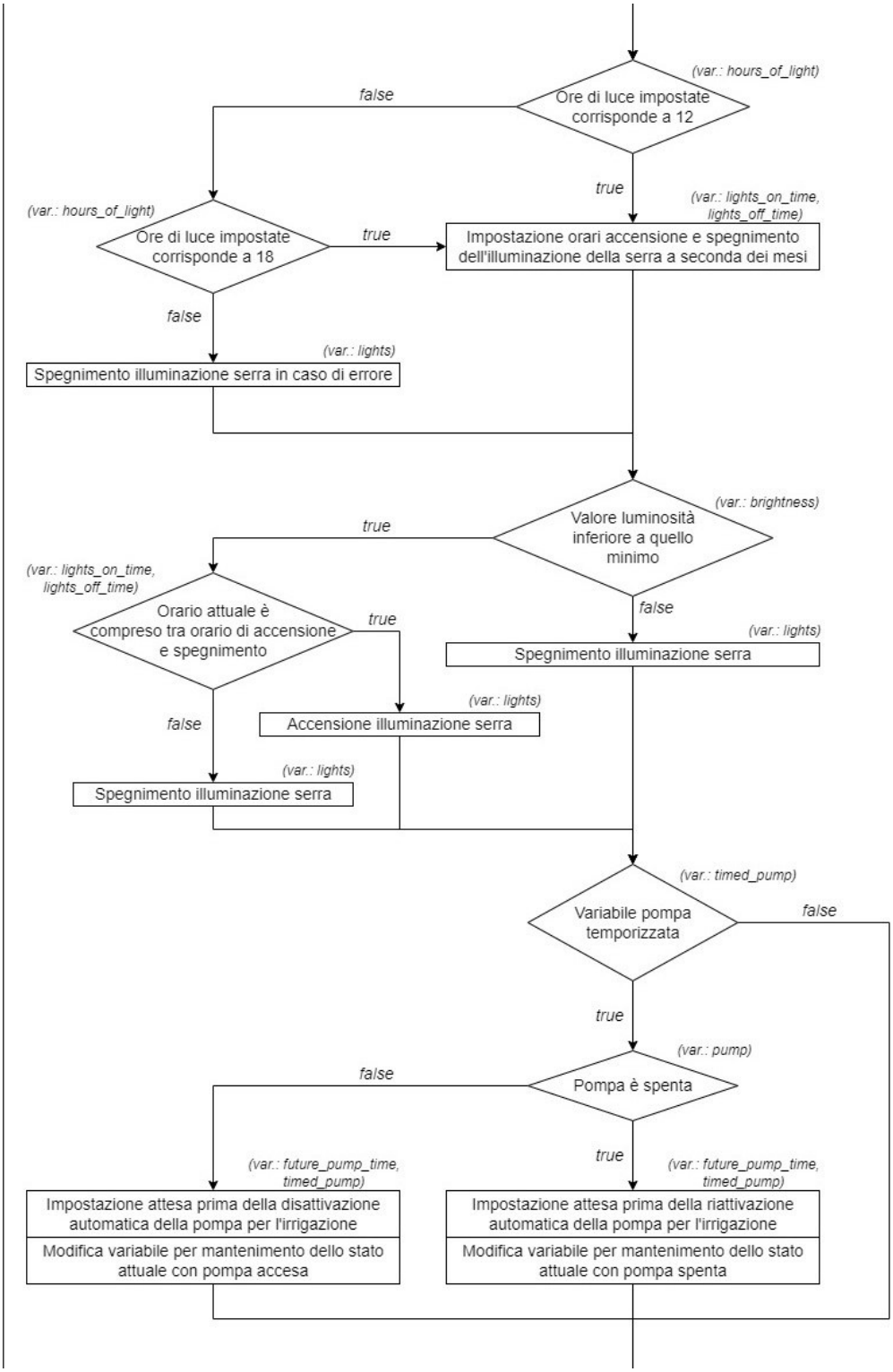
(b) *Pagina 2*


Lettura errata data e ora
true
(var.: error_rtc)
Impostazione valore errore su variabile dedicata
false
(var.: error_rtc)
Ripristino valore variabile dedicata
Errore attivo di qualche componente
true
Attivazione buzzer PKm22EPP-4001-B0
Attivazione lampeggiamento Error LED
false
Chiamata al programma automatico per la gestione della serra
greenhouse_management
(var.: temperature_min, humidity_min)
Temperatura e umidità maggiori ai valori minimi
false
true
(var.: temperature_max, umidity_max)
Temperatura o umidità maggiori ai valori massimi
false
true
(var.: timed_fan)
Variabile ventola temporizzata
false
true
(var.: baffle_position, fan, timed_fan, future_fan_time)
Apertura deflettori tramite SERVO Motor
Attivazione ventola
Modifica variabile per disattivazione ventola appena i parametri tornano nel range di valori corretti
Impostazione della variabile che registra la temporizzazione per la disattivazione della ventola

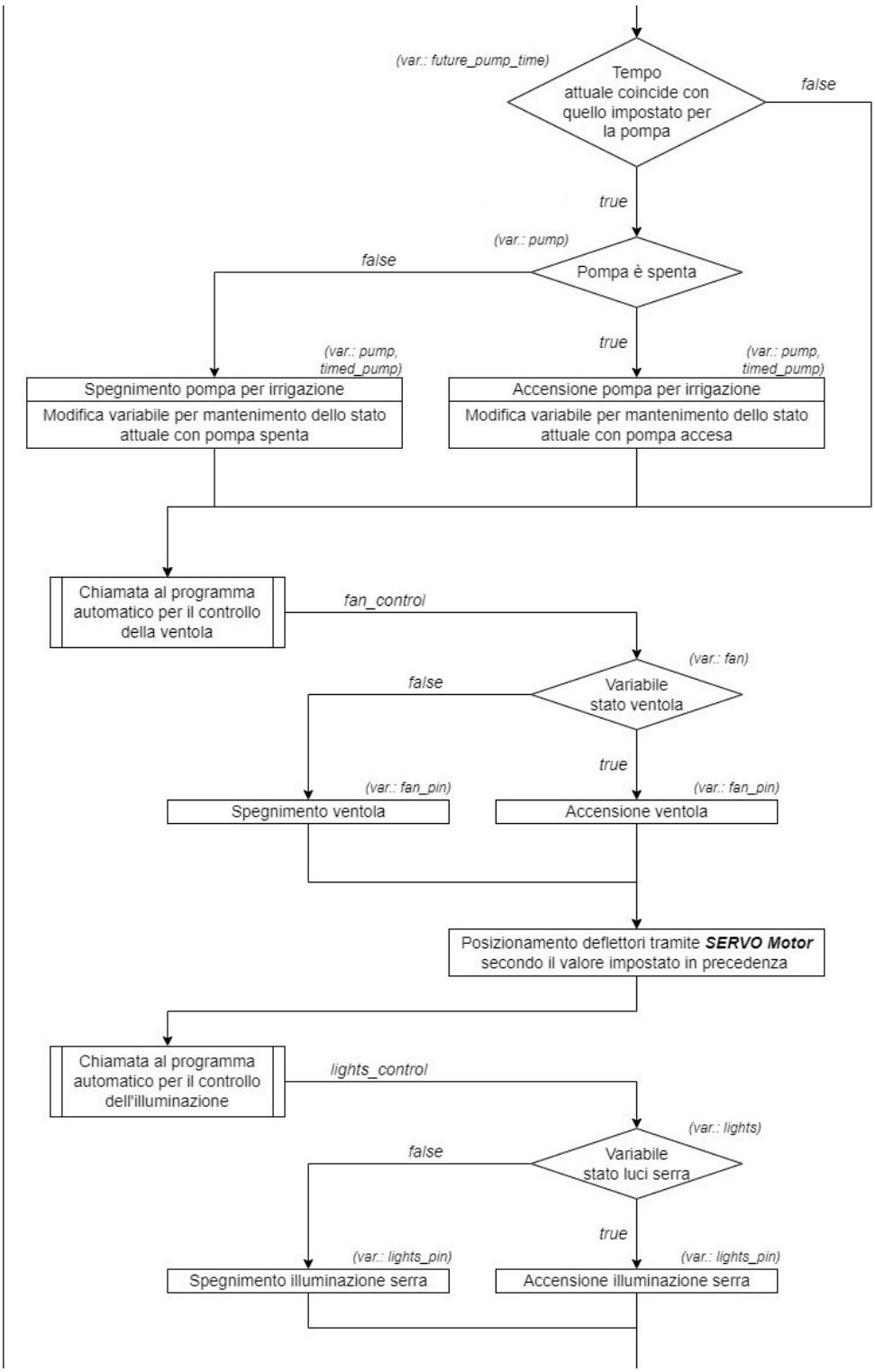
(c) *Pagina 3*


(var.: fan)
Ventola spenta
true
(var.: future_fan_time, baffle_position, timed_fan)
Impostazione tempo futuro per la riattivazione della ventola
Posizionamento intermedio dei deflettori tramite SERVO Motor
Modifica variabile per mantenimento dello stato attuale con ventola spenta
false
(var.: future_fan_time, timed_fan)
Impostazione tempo futuro per la disattivazione della ventola
Modifica variabile per mantenimento dello stato attuale con ventola accesa
(var.: current_time, future_fan_time)
Tempo impostato per attivazione/disattivazione ventola concide con tempo attuale
true
Ventola spenta
false
true
(var.: baffle_position, fan, timed_fan)
Apertura deflettori tramite SERVO Motor
Attivazione ventola
Modifica variabile per mantenimento dello stato attuale con ventola accesa
(var.: baffle_position, fan, timed_fan)
Posizionamento intermedio dei deflettori tramite SERVO Motor
Spegnimento ventola
Modifica variabile per mantenimento dello stato attuale con ventola spenta
(var.: fan, baffle_position, timed_fan)
Spegnimento ventola
Chiusura deflettori tramite SERVO Motor
Modifica variabile per mantenimento dello stato attuale con ventola spenta

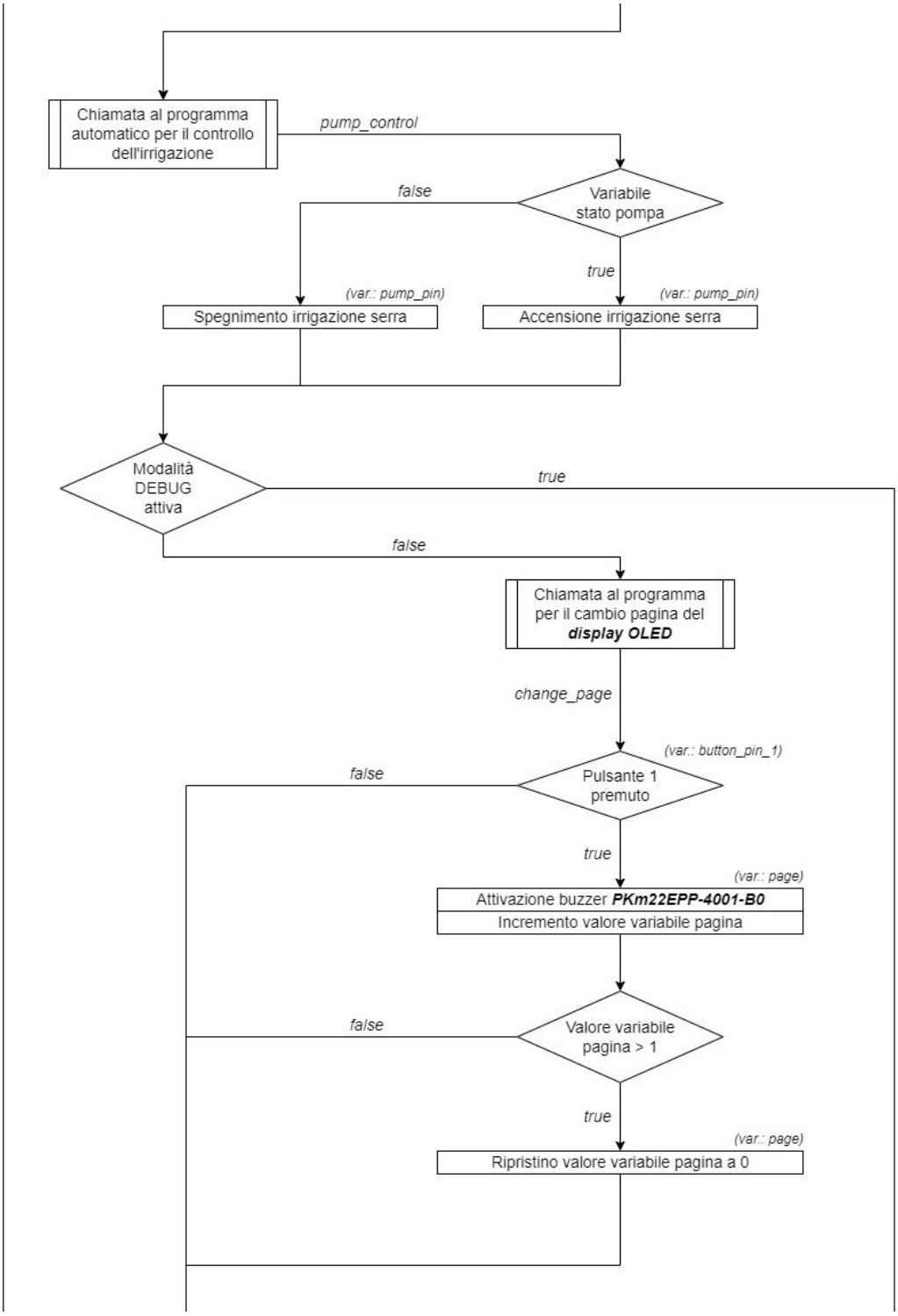
(d) *Pagina 4*


(var.: hours_of_light)
Ore di luce impostate corrisponde a 12
false
true
(var.: hours_of_light)
Ore di luce impostate corrisponde a 18
true
(var.: lights_on_time, lights_off_time)
Impostazione orari accensione e spegnimento dell'illuminazione della serra a seconda dei mesi
false
(var.: lights)
Spegnimento illuminazione serra in caso di errore
(var.: brightness)
Valore luminosità inferiore a quello minimo
true
false
(var.: lights_on_time, lights_off_time)
Orario attuale è compreso tra orario di accensione e spegnimento
true
(var.: lights)
Spegnimento illuminazione serra
false
(var.: lights)
Accensione illuminazione serra
(var.: lights)
Spegnimento illuminazione serra
(var.: timed_pump)
Variabile pompa temporizzata
false
true
(var.: pump)
Pompa è spenta
false
true
(var.: future_pump_time, timed_pump)
Impostazione attesa prima della disattivazione automatica della pompa per l'irrigazione
Modifica variabile per mantenimento dello stato attuale con pompa accesa
(var.: future_pump_time, timed_pump)
Impostazione attesa prima della riattivazione automatica della pompa per l'irrigazione
Modifica variabile per mantenimento dello stato attuale con pompa spenta

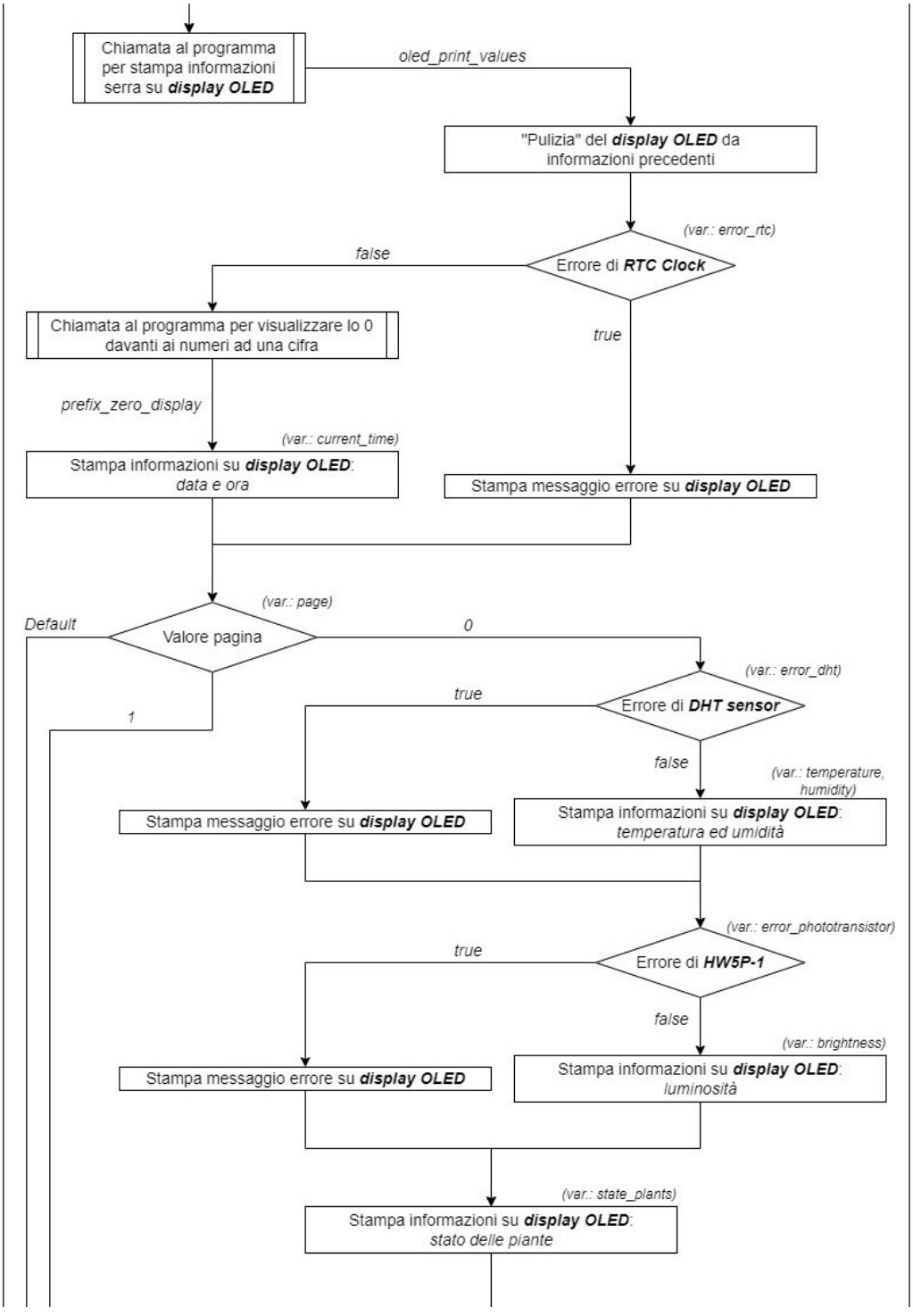
(e) *Pagina 5*


(var.: future_pump_time)
Tempo attuale coincide con quello impostato per la pompa
false
true
(var.: pump)
Pompa è spenta
false
true
(var.: pump, timed_pump)
Spegnimento pompa per irrigazione
Modifica variabile per mantenimento dello stato attuale con pompa spenta
(var.: pump, timed_pump)
Accensione pompa per irrigazione
Modifica variabile per mantenimento dello stato attuale con pompa accesa
Chiamata al programma automatico per il controllo della ventola
fan_control
(var.: fan)
Variabile stato ventola
false
true
(var.: fan_pin)
Spegnimento ventola
(var.: fan_pin)
Accensione ventola
Posizionamento deflettori tramite SERVO Motor secondo il valore impostato in precedenza
Chiamata al programma automatico per il controllo dell'illuminazione
lights_control
(var.: lights)
Variabile stato luci serra
false
true
(var.: lights_pin)
Spegnimento illuminazione serra
(var.: lights_pin)
Accensione illuminazione serra

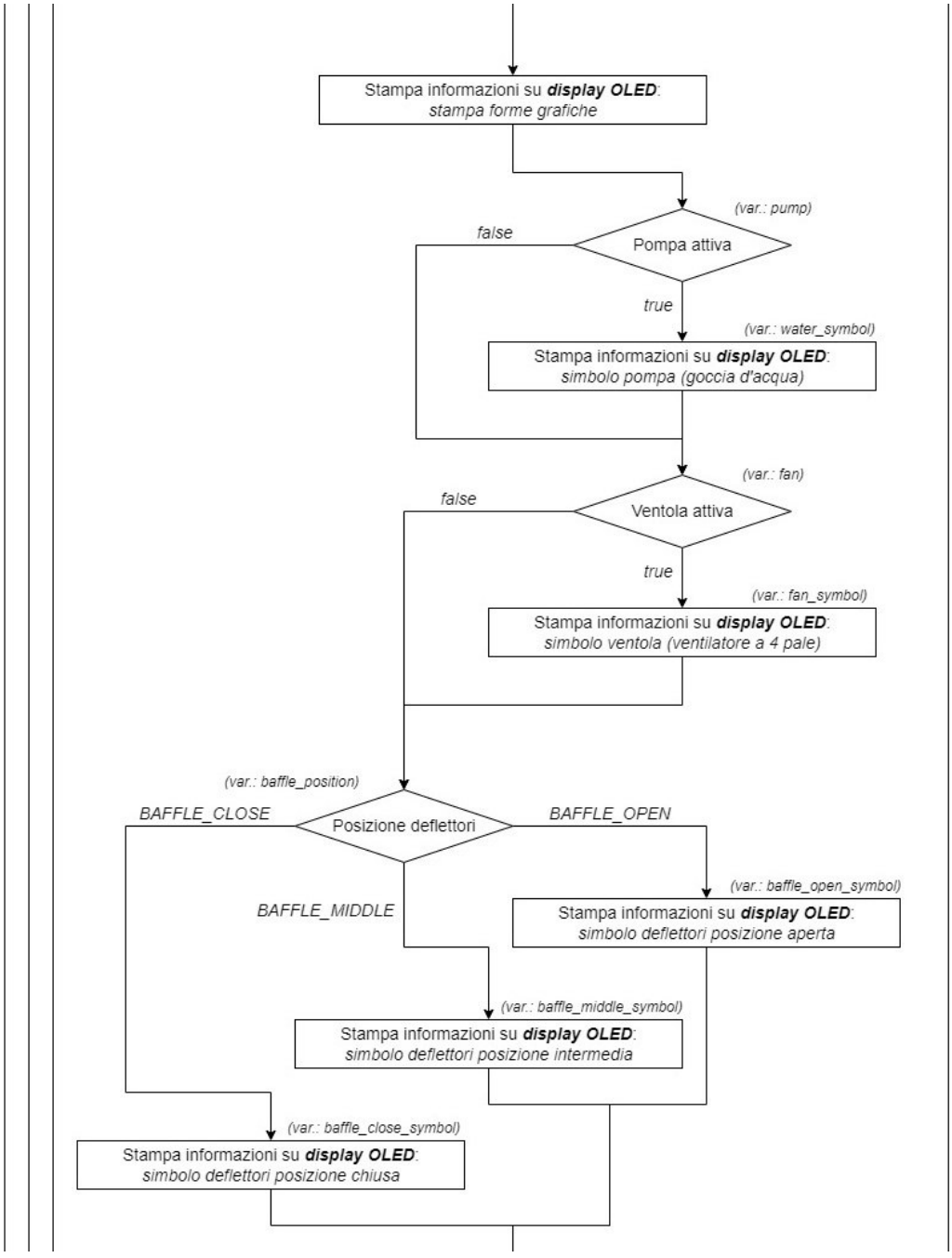
(f) *Pagina 6*


Chiamata al programma automatico per il controllo dell'irrigazione
pump_control
Variabile stato pompa
false
true
(var.: pump_pin)
Spegnimento irrigazione serra
(var.: pump_pin)
Accensione irrigazione serra
Modalità DEBUG attiva
true
false
Chiamata al programma per il cambio pagina del display OLED
change_page
(var.: button_pin_1)
Pulsante 1 premuto
false
true
(var.: page)
Attivazione buzzer PKm22EPP-4001-B0
Incremento valore variabile pagina
Valore variabile pagina > 1
false
true
(var.: page)
Ripristino valore variabile pagina a 0

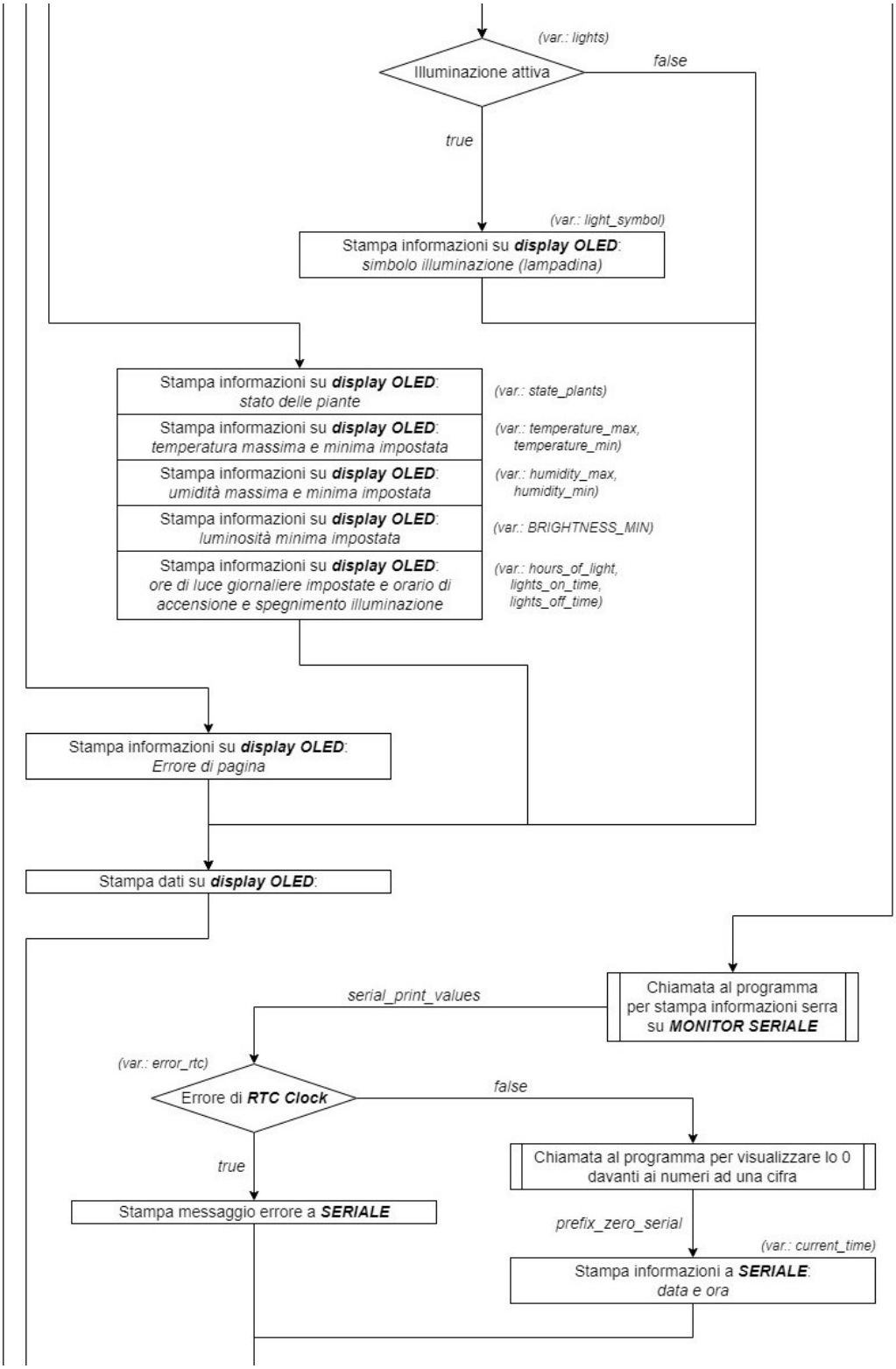
(g) *Pagina 7*


Chiamata al programma per stampa informazioni serra su display OLED
oled_print_values
"Pulizia" del display OLED da informazioni precedenti
(var.: error_rtc)
Errore di RTC Clock
false
true
Chiamata al programma per visualizzare lo 0 davanti ai numeri ad una cifra
prefix_zero_display
(var.: current_time)
Stampa informazioni su display OLED: data e ora
Stampa messaggio errore su display OLED
(var.: page)
Default
Valore pagina
0
1
(var.: error_dht)
Errore di DHT sensor
true
false
(var.: temperature, humidity)
Stampa messaggio errore su display OLED
Stampa informazioni su display OLED: temperatura ed umidità
(var.: error_phototransistor)
Errore di HW5P-1
true
false
(var.: brightness)
Stampa messaggio errore su display OLED
Stampa informazioni su display OLED: luminosità
(var.: state_plants)
Stampa informazioni su display OLED: stato delle piante

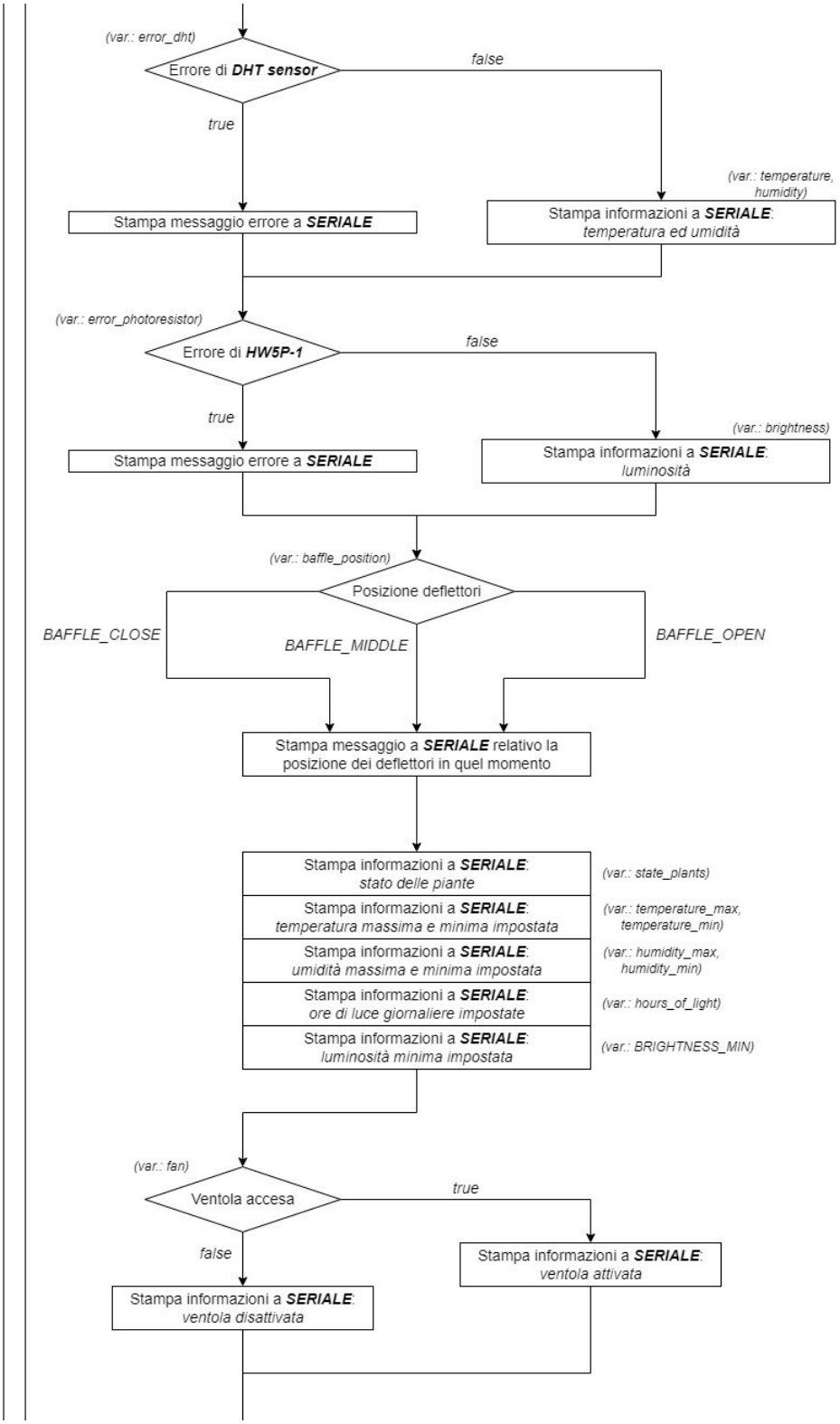
(h) *Pagina 8*


Stampa informazioni su **display OLED**:
stampa forme grafiche
(var.: pump)
Pompa attiva
false
true
(var.: water_symbol)
Stampa informazioni su **display OLED**:
simbolo pompa (goccia d'acqua)
(var.: fan)
Ventola attiva
false
true
(var.: fan_symbol)
Stampa informazioni su **display OLED**:
simbolo ventola (ventilatore a 4 pale)
(var.: baffle_position)
Posizione deflettori
BAFFLE_CLOSE
BAFFLE_OPEN
BAFFLE_MIDDLE
(var.: baffle_open_symbol)
Stampa informazioni su **display OLED**:
simbolo deflettori posizione aperta
(var.: baffle_middle_symbol)
Stampa informazioni su **display OLED**:
simbolo deflettori posizione intermedia
(var.: baffle_close_symbol)
Stampa informazioni su **display OLED**:
simbolo deflettori posizione chiusa

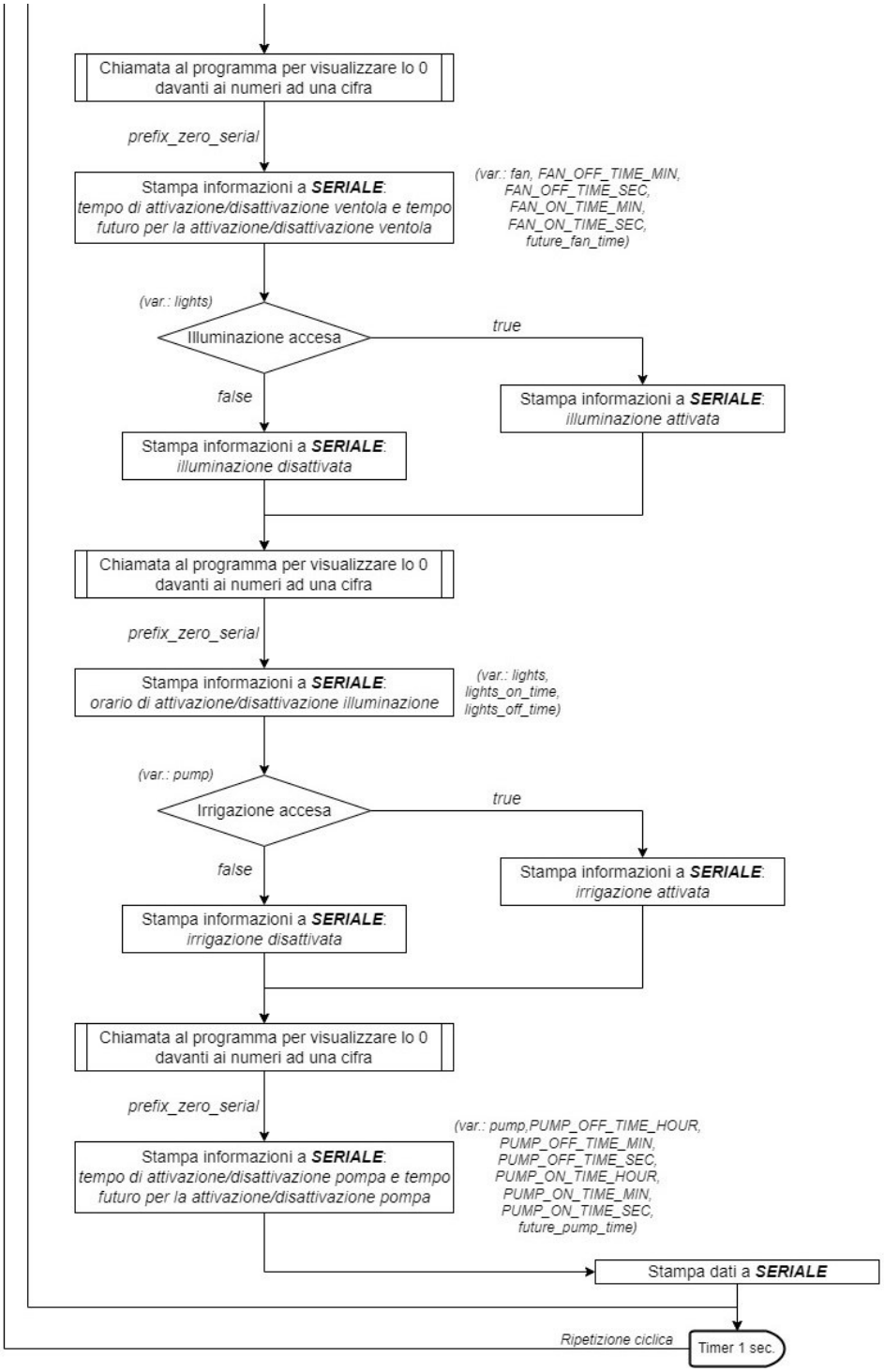
(i) *Pagina 9*


(var.: lights)
Illuminazione attiva
false
true
(var.: light_symbol)
Stampa informazioni su display OLED:
simbolo illuminazione (lampadina)
Stampa informazioni su display OLED:
stato delle piante
(var.: state_plants)
Stampa informazioni su display OLED:
temperatura massima e minima impostata
(var.: temperature_max, temperature_min)
Stampa informazioni su display OLED:
umidità massima e minima impostata
(var.: humidity_max, humidity_min)
Stampa informazioni su display OLED:
luminosità minima impostata
(var.: BRIGHTNESS_MIN)
Stampa informazioni su display OLED:
ore di luce giornaliere impostate e orario di accensione e spegnimento illuminazione
(var.: hours_of_light, lights_on_time, lights_off_time)
Stampa informazioni su display OLED:
Errore di pagina
Stampa dati su display OLED:
Chiamata al programma per stampa informazioni serra su MONITOR SERIALE
serial_print_values
(var.: error_rtc)
Errore di RTC Clock
false
true
Chiamata al programma per visualizzare lo 0 davanti ai numeri ad una cifra
Stampa messaggio errore a SERIALE
prefix_zero_serial
(var.: current_time)
Stampa informazioni a SERIALE:
data e ora

(j) *Pagina 10*


(var.: error_dht)
Errore di DHT sensor
false
true
(var.: temperature, humidity)
Stampa messaggio errore a SERIALE
Stampa informazioni a SERIALE: temperatura ed umidità
(var.: error_photoresistor)
Errore di HW5P-1
false
true
(var.: brightness)
Stampa messaggio errore a SERIALE
Stampa informazioni a SERIALE: luminosità
(var.: baffle_position)
Posizione deflettori
BAFFLE_CLOSE
BAFFLE_MIDDLE
BAFFLE_OPEN
Stampa messaggio a SERIALE relativo la posizione dei deflettori in quel momento
Stampa informazioni a SERIALE: stato delle piante
(var.: state_plants)
Stampa informazioni a SERIALE: temperatura massima e minima impostata
(var.: temperature_max, temperature_min)
Stampa informazioni a SERIALE: umidità massima e minima impostata
(var.: humidity_max, humidity_min)
Stampa informazioni a SERIALE: ore di luce giornaliere impostate
(var.: hours_of_light)
Stampa informazioni a SERIALE: luminosità minima impostata
(var.: BRIGHTNESS_MIN)
(var.: fan)
Ventola accesa
true
false
Stampa informazioni a SERIALE: ventola attivata
Stampa informazioni a SERIALE: ventola disattivata

(k) *Pagina 11*



(l) *Pagina 12*

Figura 58: Flowchart della funzione *LOOP* del programma per la gestione della serra idroponica.

5.2 Programma automatico per la gestione della serra

Oltre alla realizzazione della serra e la scelta dei componenti, il programma per la gestione automatica è il vero punto forte del progetto. I dispositivi collegati alla scheda possono essere sostituiti, ma la struttura del codice implementata rimarrà la stessa. Il programma sviluppato permette variazioni dei parametri sia da parte dell'utente, in modo più limitato e specifico solo per alcuni valori, ma anche a livello di programmazione. Il codice riportato di seguito rispecchia, attraverso righe di comando, i flowcharts di *figura 58* e *57*. Nei diagrammi di flusso venivano semplicemente spiegati i passi, all'interno del programma, necessari per ottenere il controllo desiderato, mentre ora si potrà analizzare in modo più approfondito il codice necessario per svolgere quelle operazioni. Di seguito verranno riportate le sezioni del programma implementato e una descrizione di ogni riga.

5.2.1 Librerie di progetto

Hydroponic_Greenhouse

```
1  /*************************************************************************************************************************************************
2   *                                                  HYDROPONIC GREENHOUSE
3   *                                  Three-year Degree Thesis, ING. Mechatronics, Padua (DTG, Vicenza)
4   *                                        (unauthorized distributions are not permitted)
5   *  Version:          1.0
6   *  Author:           Filippo Tasca
7   *  Pin connection:
8   *    Analog:         A0 -> HW5P-1,  A4 (SDA) -> SSD1306 and DS1307,  A5 (SCL) -> SSD1306 and DS1307;
9   *
10  *    Digital:        2 -> DHT11,  3 -> SM-S2309S,  4 -> JQC3F-5VDC-C (pump),  7 -> JQC3F-5VDC-C (lights),  8 -> JQC3F-5VDC-C (fan),
11  *                    10 -> 101-TS6111T1601-EV (button 1),  11 -> 101-TS6111T1601-EV (button 2),  12 -> PRm22EPP-4001-B0,  13 -> UR502DC (led).
12  *  Description:      Automated program for management hydroponic greenhouse
13  *  First update:     17/02/2022
14  *  Last update:      15/03/2022
15  *************************************************************************************************************************************************/
16
17  /* ---------------------------------------  Libraries  --------------------------------------- */
18
19  #include <DHT.h>                                                    // Library that includes a information and control of DHT with Arduino
20  #include <SPI.h>                                                    // Library that includes the communication for SPI device
21  #include <Wire.h>                                                   // Library that includes I2C communication process
22  #include <RTClib.h>                                                 // Library that includes the control and manage of RTC clock with Arduino
23  #include <Servo.h>                                                  // Library that includes the control of SERVO motor with Arduino
24  #include <Adafruit_GFX.h>                                           // Library that includes a common set of graphic primitives
25  #include <Adafruit_SSD1306.h>                                       // Library that includes the control of OLED display with Arduino
26
27
```

Figura 59: Introduzione del programma e librerie di progetto.

In *figura 59* viene riportato l'inizio del programma implementato. Per prima cosa ci sono della brevi righe di commento per riportare autore, data di inizio del progetto e data dell'ultima modifica. Inoltre, viene riportata la configurazione dei pin della scheda Arduino con i dispositivi collegati.
La prima sezione del progetto riguarda le librerie: caricare le librerie necessarie al funzionamento del programma è la prima operazione importante da compiere. Senza di queste, molti dispositivi collegati non riuscirebbero a comunicare con Arduino, e viceversa, compromettendo il funzionamento del programma.

- riga 19: $< DHT.h >$
  Descrizione: Libreria di progetto per sensori di temperatura *DHT11*, *DHT22* e gli altri modelli *DHT*. Prodotta da *Adafruit*, permette di

controllare i sensori di temperatura ed umidità e ricevere informazioni da questi.
Uso: Questa libreria viene utilizzata per poter interfacciare la scheda con il sensore di temperatura ed umidità, in modo da rilevare i parametri ambientali interni alla serra.

- riga 20: $< SPI.h >$
Descrizione: Libreria usata per comunicare con i dispositivi provvisti di comunicazione *SPI*, usando Arduino come controllore (con funzionamento *master/slave*).
Uso: Viene utilizzata per il controllo dello schermo OLED *SSD1306*.

- riga 21: $< Wire.h >$
Descrizione: Libreria usata per il protocollo di comunicazione I2C/TWI usando i pin diretti dedicati SDA (linea dati) e SCL (linea di clock), oppure rispettivamente i pin di ingresso analogici A4 e A5 della scheda Arduino. Anche questa permette, come per la libreria precedente, la comunicazione tra la scheda e i dispositivi che ne fanno uso.
Uso: Usata per comunicare con il Clock RTC *DS1307* e con lo schermo OLED *SDD1306*.

- riga 22: $< RTClib.h >$
Descrizione: Libreria usata per il controllo del Clock RTC *DS1307* e l'interfaccia di questo con Arduino.
Uso: Viene utilizzata per ricevere i dati (data e ora) dal Clock RTC e per usare specifiche funzioni contenute nella libreria stessa.

- riga 23: $< Servo.h >$
Descrizione: Permette il controllo, da parte della scheda Arduino, di una grande varietà di servomotori.
Uso: Viene usata nel programma per controllare il servomotore collegato ai deflettori nel condotto dell'aria posizionato sopra alla ventola della serra. Garantisce il posizionamento senza uso di altri sensori di posizione.

- riga 24: $< Adafruit\_GFX.h >$
Descrizione: Libreria grafica di base per l'uso di schermi e dispositivi grafici *Adafruit*, prodotta e distribuita sempre dagli stessi.
Uso: Viene utilizzata per tracciare simboli e altri elementi grafici sullo schermo OLED collegato alla scheda Arduino.

- riga 25: $< Adafruit\_SSD1306.h >$
Descrizione: Libreria contenente i driver per il funzionamento degli schermi OLED *SSD1306* per display monocromatici di dimensione 128x64 e 128x32 pixels.
Uso: Viene usata per il controllo del display OLED *SSD1306*.

5.2.2 #define

Questa sezione contiene, come descritto dal nome, i *#define*, un utile componente che permette di dare un nome ad un valore costante prima che il programma venga compilato. I parametri contenuti in questa sezione sono utilizzati per memorizzare valori costanti da utilizzare in seguito nel programma. Principalmente sono usati per definire i componenti esterni collegati alla scheda, ma sono contenuti anche valori costanti che permettono l'esecuzione del programma per il controllo della serra. Garantiscono rapide variazioni sul programma automatico e sui componenti installati, in caso di sostituzione di questi. In *figura 60* viene riportata la sezione dedicata.

```
28
29 /* ---------------------------------------- Defines ---------------------------------------- */
30
31 // Debug Mode
32 #define DEBUG false                                                    // Variable for using "DEBUG MODE" (false: normal, true: debug mode, with serial print values)
33
34 // DHT11 sensor
35 #define DHT_TYPE DHT11                                                // Type of DHT sensor used in the program
36
37 // Buzzer values
38 #define PITCH 500                                                     // Pitch frequency of buzzer
39 #define BUZZER_TIME 100                                               // Pitch time of buzzer
40
41 // Brightness
42 #define BRIGHTNESS_MIN 20                                             // Minimum value of brightness
43
44 // Fan time values
45 #define FAN_ON_TIME_MIN 0                                             // Time fan ON (in minute)
46 #define FAN_ON_TIME_SEC 30                                            // Time fan ON (in second)
47 #define FAN_OFF_TIME_MIN 5                                            // Time fan OFF (in minute)
48 #define FAN_OFF_TIME_SEC 0                                            // Time fan OFF (in second)
49
```

(a) *Parte 1*

```
50 // Pump time values
51 #define PUMP_ON_TIME_HOUR 0                                           // Time pump ON (in hour)
52 #define PUMP_ON_TIME_MIN 0                                            // Time pump ON (in minute)
53 #define PUMP_ON_TIME_SEC 30                                           // Time pump ON (in second)
54 #define PUMP_OFF_TIME_HOUR 1                                          // Time pump OFF (in hour)
55 #define PUMP_OFF_TIME_MIN 30                                          // Time pump OFF (in minute)
56 #define PUMP_OFF_TIME_SEC 00                                          // Time pump OFF (in second)
57
58 // Baffle position values
59 #define BAFFLE_OPEN 90                                                // Value of baffle open position (in degrees)
60 #define BAFFLE_MIDDLE 130                                             // Value of baffle middle position (standard position, in degrees)
61 #define BAFFLE_CLOSE 175                                              // Value of baffle close position (in degrees)
62
63 // SSD1306 OLED values
64 #define SCREEN_WIDTH 128                                              // OLED display width (in pixels)
65 #define SCREEN_HEIGHT 64                                              // OLED display height (in pixels)
66 #define SCREEN_ADDRESS 0x3C                                           // See datasheet for Address; 0x3D for 128x32, 0x3C for 128x64
67 #define OLED_RESET -1                                                 // Reset pin # (or -1 if sharing Arduino reset pin)
68
69
70
```

(b) *Parte 2*

Figura 60: Sezione contenente tutti i *#define* utilizzati nel programma.

- riga 32: #define DEBUG false
  Variabile usata per attivare o disattivare la modalità per il *DEBUG* del programma. In particolare, permette di deviare l'invio delle informazioni all'operatore dal display OLED (con *DEBUG = false*) al monitor seriale del programma di Arduino (con *DEBUG = true*). Attivando la modalità di *DEBUG*, verranno riportate informazioni ulteriori rispetto a quelle visualizzate sullo schermo OLED, che permettono di controllare eventuali errori di programmazione. Infatti, le informazioni visualizzate dal display OLED sono sintetiche e mirate, mentre quelle

inviate al monitor seriale sono adatte a riportare lo stato completo del programma.

- riga 35: #define DHT_TYPE DHT11
  Variabile usata per la definizione del tipo di sensore DHT utilizzato. Serve per comunicare alla libreria $< DHT.h >$ quale modello di sensore si utilizza nel programma e di conseguenza a quale deve interfacciarsi.

- riga 38: #define PITCH 500
  Variabile usata per definire il "tono" del buzzer collegato alla scheda quando viene attivato.

- riga 39: #define BUZZER_TIME 100
  Variabile usata per definire il tempo di attivazione del buzzer quando riceve il segnale.

- riga 42: #define BRIGHTNESS_MIN 20
  Variabile usata per definire la soglia minima di luminosità sufficiente per la crescita delle piante, in un intervallo tra 0 e 100.

- riga 45: #define FAN_ON_TIME_MIN 0
  riga 46: #define FAN_ON_TIME_SEC 30
  riga 47: #define FAN_OFF_TIME_MIN 5
  riga 48: #define FAN_OFF_TIME_SEC 0
  Variabili usate per definire rispettivamente, come suggerito dai nomi, il tempo in minuti e secondi per i quali la ventola rimane accesa una volta ricevuto il comando e il tempo, sempre in minuti e secondi, per i quali deve rimanere spenta. Il comando di commutazione della ventola viene inviato dal programma al verificarsi di determinate condizioni. Il ricircolo dell'aria della serra deve essere frequente, per evitare concentrazioni pericolose di CO_2. Per questo motivo, sono state definite solo le variabili per i minuti e secondi.

- riga 51: #define PUMP_ON_TIME_HOUR 0
  riga 52: #define PUMP_ON_TIME_MIN 0
  riga 53: #define PUMP_ON_TIME_SEC 30
  riga 54: #define PUMP_OFF_TIME_HOUR 1
  riga 55: #define PUMP_OFF_TIME_MIN 30
  riga 56: #define PUMP_OFF_TIME_SEC 0
  Variabili usate per definire rispettivamente il tempo in ore, minuti e secondi per i quali la pompa deve rimanere accesa una volta ricevuto il comando e sempre il tempo in ore, minuti e secondi per i quali deve invece rimanere spenta. Anche per la pompa, come per la ventola, deve esserci un opportuno comando di attivazione e disattivazione. In questo caso, sono state prese in considerazione anche le ore: infatti,

il tempo di attesa tra un ciclo irriguo e l'altro può essere anche notevole. A seconda del substrato scelto, e dei parametri ambientali che influiscono su questo, è possibile variare il valore di queste variabili.

- riga 59: #define BAFFLE_OPEN 90
  riga 60: #define BAFFLE_MIDDLE 130
  riga 61: #define BAFFLE_CLOSE 175
  Variabili usate per definire la posizione del servomotore collegato ai deflettori posizionati, assieme alla ventola, sopra alla serra. A seconda del valore inviato dalla scheda Arduino, permettono di variare l'angolo di incidenza delle alette e cambiare la portata d'aria tramite una opportuna rotazione dell'albero del servomotore. I valori sono stati scelti proprio per permettere il posizionamento corretto dei deflettori.

- riga 64: #define SCREEN_WIDTH 128
  riga 65: #define SCREEN_HEIGHT 64
  riga 66: #define SCREEN_ADDRESS 0x3C
  riga 67: #define OLED_RESET -1
  Variabili usate per la configurazione dei parametri dello schermo OLED *SSD1306*. Rispettivamente identificano la larghezza e l'altezza dello schermo in pixels, l'indirizzo di questo (necessario per la comunicazione I2C tra Arduino e lo schermo) e il pin di reset. Quest'ultimo non è presente nello schermo e dunque deve essere condiviso con quello di Arduino. E' importante avere queste variabili personalizzabili per avere un rapido e semplice cambiamento in caso di sostituzione dello schermo che implichi delle caratteristiche differenti.

5.2.3 Definizione dei pin I/O

La seguente sezione è dedicata alla definizione degli ingressi e delle uscite, sia digitali che analogici. Di fondamentale importanza è rinominare i pin di Arduino con nomi riferiti al loro collegamento in modo da individuare istantaneamente il pin di cui si ha bisogno in fase di programmazione. In *figura 61* la sezione riguardante la definizione dei pin della scheda Arduino.

```
71 /* ------------------------------------  I/O digital/analog pin  ------------------------------------ */
72
73 // Digital pins
74 const int dht_pin = 2;                                          // Variable for using Arduino pin connected at DHT sensor
75 const int baffle_pin = 3;                                       // Variable for using Arduino pin connected at SERVO motor
76 const int pump_pin = 4;                                         // Variable for using Arduino pin connected at JQC3F-5VDC-C relay for manage pump
77 const int lights_pin = 7;                                       // Variable for using Arduino pin connected at illumination LED
78 const int fan_pin = 8;                                          // Variable for using Arduino pin connected at JQC3F-5VDC-C relay for manage fan
79 const int button_pin_1 = 10;                                    // Variable for using Arduino pin connected at button (display page change)
80 const int button_pin_2 = 11;                                    // Variable for using Arduino pin connected at button (greenhouse parameters change)
81 const int warning_buzzer = 12;                                  // Variable for using Arduino pin connected at PKm22EPP-4001-B0 buzzer
82 const int warning_led = 13;                                     // Variable for using Arduino pin connected at UR502DC led (red)
83
84 // Analog pin
85 const int phototransistor_pin = A0;                             // Variable for using Arduino pin connected at HW5P-1 phototransistor
86
87
88
```

Figura 61: Sezione contenente la definizione dei pin I/O della scheda.

- riga 74: const int dht_pin = 2;
  Variabile usata per definire il pin collegato al sensore *DHT* per la misurazione della temperatura e dell'umidità.

- riga 75: const int baffle_pin = 3;
  Variabile usata per definire il pin collegato al servomotore per l'orientamento dei deflettori.

- riga 76: const int pump_pin = 4;
  Variabile usata per definire il pin collegato al relay che comanda l'attivazione della pompa.

- riga 77: const int lights_pin = 7;
  Variabile usata per definire il pin collegato al relay che comanda l'attivazione dell'illuminazione della serra.

- riga 78: const int fan_pin = 8;
  Variabile usata per definire il pin collegato al relay che comanda l'attivazione della ventola.

- riga 79: const int button_pin_1 = 10;
  Variabile usata per definire il pin collegato al pulsante di selezione della pagina del display OLED. Premendo questo pulsante è possibile cambiare sezione e visualizzare altri dati rispetto a quelli normalmente raffigurati nella pagina iniziale.

- riga 80: const int button_pin_2 = 11;
  Variabile usata per definire il pin collegato al pulsante che permette il cambio dei parametri della serra. A seconda dell'avanzamento e della crescita delle piante, è possibile selezionare diversi stati di sviluppo e quindi parametri di controllo della serra opportunamente definiti per lo stadio nel quale si trovano le colture.

- riga 81: const int warning_buzzer = 12;
  Variabile usata per definire il pin collegato al buzzer sonoro, usato per segnalare stati di errore dei componenti o il cambiamento dei parametri di crescita delle piante.

- riga 82: const int warning_led = 13;
  Variabile usata per definire il pin collegato al led per la segnalazione di stati di errore. Questo led segnala errori legati al programma o ai singoli componenti, richiamando l'attenzione dell'utente.

- riga 85: const int phototransistor_pin = A0;
  Variabile usata per definire il pin collegato al fototransistor. Questo è l'unico pin analogico di ingresso utilizzato nel programma, escludendo i pin A4 e A5 usati per il protocollo di comunicazione I2C tra la scheda e i componenti che ne fanno uso.

5.2.4 Impostazione dei dispositivi collegati alla scheda

L'impostazione dei componenti collegati alla scheda Arduino è fondamentale per ogni programma. In questa sezione in particolare, si definiscono le impostazioni sul modello e sulle altre caratteristiche del display OLED, del sensore di temperatura ed umidità, dell'RTC Clock e del servomotore. Come si può notare, è un'impostazione dei componenti molto semplice: infatti, in caso di sostituzione di uno di questi, è possibile modificare l'opportuno #define, definito in precedenza, senza dover ricercare queste righe di codice nel programma. In *figura 62* la sezione riguardante l'impostazione dei componenti.

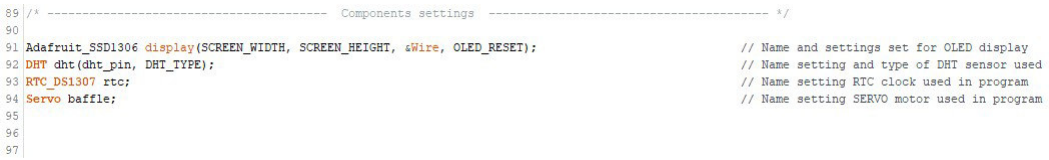


Figura 62: Sezione contenente la regolazione dei componenti collegati alla scheda per la configurazione con il programma automatico.

- riga 91: Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
  Con questa dichiarazione viene definita la tipologia dello schermo OLED installata, definendo i parametri caratterizzanti definiti in precedenza nella sezione dei #define. Si fa riferimento rispettivamente a larghezza e altezza dello schermo, in pixels, tipologia di protocollo utilizzata e pin di reset del display. Inoltre con la parola "display" si definisce il nome con cui si richiama lo schermo all'interno del programma.

- riga 92: DHT dht(dht_pin, DHT_TYPE);
  Questa dichiarazione serve per definire il pin al quale è collegato il sensore di temperatura ed umidità e il tipo di *DHT* utilizzato. Anche qui come per lo schermo è possibile definire il nome con cui verrà richiamato il sensore all'interno del programma; in particolare si è scelto il nome "dht".

- riga 93: RTC_DS1307 rtc;
  Questa dichiarazione, se pur molto minimale, specifica il tipo di RTC Clock utilizzato e collegato ad Arduino permettendo inoltre di rinominare il dispositivo all'interno del programma con il nome "rtc".

- riga 94: Servo baffle;
  Con questa dichiarazione è possibile solo rinominare il nome del servomotore collegato, senza specificarne il tipo o specificare altri parametri. Arduino UNO offre un controllo fino a 12 servomotori contemporanea-

mente, purché siano compatibili con la gestione che offre la libreria ad essi dedicata.

5.2.5 Variabili del programma

Questa sezione è usata per la definizione di tutte le variabili globali utilizzate nel programma. In *figura 63* la sezione del codice dedicata.

```
98  /* ---------------------------------------  Variables  --------------------------------------- */
99
100 float temperature = 0, humidity = 0;                                  // Variables for acquisition of temperature and humidity values
101 int brightness = 0;                                                   // Variable for brightness acquisition
102 int temperature_max = 0;                                             // Variable for setting greenhouse max temperature
103 int temperature_min = 0;                                             // Variable for setting greenhouse min temperature
104 int humidity_max = 0;                                                // Variable for setting greenhouse max humidity
105 int humidity_min = 0;                                                // Variable for setting greenhouse min humidity
106 int hours_of_light = 0;                                              // Variable for setting greenhouse hours of light
107 int parameter = 0;                                                   // Variable for auto-setting greenhouse parameters (temperature [max and min], humidity [max an
108 int baffle_position = 0;                                             // Variable for save position of baffle (in degrees)
109 uint8_t page = 0;                                                    // Variable to save the number of the page shown in the display
110 uint8_t lights_on_time = 0;                                          // Variable for setting time of switching ON lights
111 uint8_t lights_off_time = 0;                                         // Variable for setting time of switching OFF lights
112 DateTime current_time (0, 0, 0, 0);                                  // Variable for recording data of time from RTC
113 DateTime future_fan_time (0, 0, 0, 0);                               // Variable that setting the time for control of the fan
114 DateTime future_pump_time (0, 0, 0, 0);                              // Variable that setting the time for control of the pump
115 bool error_dht = false;                                              // Variable for manage DHT sensor error (false: OK, true: Error!)
116 bool error_phototransistor = false;                                  // Variable for manage HW5P-1 phototransistor error (false: OK, true: Error!)
117 bool error_rtc = false;                                              // Variable for manage RTC clock error (false: OK, true: Error!)
118 bool fan = false;                                                    // Variable for manage FAN (false: Fan OFF, true: Fan ON)
119 bool timed_fan = true;                                               // Auxiliary variable for manage automatic ventilation (used in the program)
120 bool lights = false;                                                 // Variable for manage LED illumination (false: lights OFF, true: lights ON)
121 bool pump = false;                                                   // Variable for manage PUMP (false: Pump OFF, true: Pump ON)
122 bool timed_pump = true;                                              // Auxiliary variable for manage automatic pumping (used in the program)
123 char state_plants[6][19] = {"Germination",                           // Char array to store plant state names
124                             "Cutting",
125                             "Rooted plant",
126                             "Vegetative phase",
127                             "Flowering/fruiting",
128                             "Final stage"};
129
130
131
```

Figura 63: Sezione contenente le variabili globali usate nel programma.

- riga 100: float temperature = 0, humidity = 0;
  Definizione ed inizializzazione delle variabili per la memorizzazione dei valori di temperatura ed umidità ambientali all'interno della serra.

- riga 101: int brightness = 0;
  Definizione ed inizializzazione della variabile per la memorizzazione del valore della luminosità rilevato all'esterno della serra idroponica.

- riga 102: int temperature_max = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il valore massimo di temperatura ammissibile nella serra, a seconda degli stadi della pianta e dell'impostazione che gli viene data dal programma di gestione della serra.

- riga 103: int temperature_min = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il valore minimo di temperatura ammissibile nella serra, a seconda degli stadi della pianta e dell'impostazione che gli viene data dal programma di gestione della serra.

- riga 104: int humidity_max = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il

valore massimo di umidità ammissibile nella serra, a seconda degli stadi della pianta e della impostazione che gli viene data dal programma di gestione della serra.

- riga 105: int humidity_min = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il valore minimo di umidità ammissibile nella serra, a seconda degli stadi della pianta e dell'impostazione che gli viene data dal programma di gestione della serra.

- riga 106: int hours_of_light = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il valore di ore di luce giornaliere che dovranno subire le colture nella serra. Anche questo valore, è scelto in funzione dello stadio di sviluppo della pianta e a seconda dell'impostazione che viene data dal programma.

- riga 107: int parameter = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare gli stati di crescita, e i valori caratteristici di ognuno, sotto forma di ricette. Questa variabile permette di selezionare una sola ricetta tra quelle presenti, fornendo al programma i parametri ad essa collegati.

- riga 108: int baffle_position = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il valore dell'angolo dell'albero del servomotore. Grazie a questa variabile è possibile inviare al servomotore la posizione desiderata per effettuare l'opportuno posizionamento dei deflettori ad esso collegati.

- riga 109: uint8_t page = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare il valore della pagina visualizzata sullo schermo Display OLED. Ogni pagina definita all'interno del programma riporta dei dati significativi della serra. Con questa variabile è possibile scegliere una tra le varie pagine visualizzabili sullo schermo.

- riga 110: uint_t lights_on_time = 0;
  Definizione ed inizializzazione della variabile usata per memorizzare l'orario dopo cui è permessa l'attivazione dell'illuminazione della serra, se vi sono le opportune condizioni ambientali esterne. Inizialmente impostata al valore "0" in seguito verrà caricato il valore opportuno contenuto nelle ricette dedicate ad ogni stato di crescita della pianta.

- riga 111: uint_t lights_off_time = 0;
  Analogamente alla riga 110, definizione ed inizializzazione della variabile usata per memorizzare l'orario dopo cui viene disattivata automa-

ticamente l'illuminazione della serra, indipendentemente dalle condizioni esterne. Anche qui, come in precedenza, viene impostato il valore corretto richiamando l'opportuna ricetta.

- riga 112: DateTime current_time (0, 0, 0, 0);
  Definizione ed inizializzazione della variabile usata per memorizzare l'orario e la data correnti. Ad ogni ciclo del programma, questo valore viene aggiornato.

- riga 113: DateTime future_fan_time (0, 0, 0, 0);
  Definizione ed inizializzazione della variabile usata per memorizzare l'orario e la data per la attivazione/disattivazione automatica della ventola. Quando lo stato di quest'ultima commuta, è possibile definire il tempo e l'orario di mantenimento dello stato fino alla successiva commutazione.

- riga 114: DateTime future_pump_time (0, 0, 0, 0);
  Definizione ed inizializzazione della variabile usata per memorizzare l'orario e la data del prossimo ciclo irriguo della serra. Ciclicamente, viene eseguito un ciclo di irrigazione gestito dal programma.

- riga 115: bool error_dht = false;
  Definizione ed inizializzazione della variabile usata per memorizzare lo stato di funzionamento/errore del sensore *DHT11* per la misurazione della temperatura e dell'umidità. In caso di errore, viene utilizzata questa variabile per comunicare la presenza di un malfunzionamento nel sensore.

- riga 116: bool error_phototransistor = false;
  Analogamente a sopra, definizione ed inizializzazione della variabile usata per memorizzare lo stato di funzionamento/errore del fototransistor usato per la misurazione del livello di illuminazione esterna. In caso di valori errati, ovvero al di fuori dell'intervallo compreso tra 0 e 100, viene segnalato nel programma la presenza di questo problema.

- riga 117: error_rtc = false;
  Analogamente alle due righe precedenti, definizione ed inizializzazione della variabile usata per memorizzare lo stato di funzionamento/errore del clock RTC *DS1307*. In caso di fallimento della comunicazione tra Arduino e il Clock, viene usata questa variabile per comunicare il problema al programma.

- riga 118: fan = false;
  Definizione ed inizializzazione della variabile usata per memorizzare lo stato logico della ventola. Grazie a questa variabile, è possibile leggere una sola volta lo stato del pin collegato al relay della ventola e avere a

disposizione durante tutto il ciclo del programma lo stato aggiornato. In questo modo è possibile evitare continue letture del pin.

- riga 119: timed_fan = false;
  Definizione ed inizializzazione della variabile usata per la attivazione/disattivazione della ventilazione automatica all'interno del programma.

- riga 120: lights = false;
  Definizione ed inizializzazione della variabile usata per memorizzare lo stato logico dell'illuminazione della serra in modo da averlo a disposizione ad ogni ciclo del programma (motivazioni analoghe a quelle della variabile "fan").

- riga 121: pump = false;
  Definizione ed inizializzazione della variabile usata per memorizzare lo stato logico della pompa. Analogamente a quanto detto per la ventola e per l'illuminazione, permette di avere ad ogni ciclo lo stato logico della pompa a disposizione e all'occorrenza modificarlo.

- riga 122: timed_pump = false;
  Definizione ed inizializzazione della variabile usata per la attivazione/disattivazione dell'irrigazione automatica.

- riga 123: char state_plants[6][19] =
  {"Germination",
  "Cutting",
  "Rooted plant",
  "Vegetative phase",
  "Flowering/fruiting",
  "Final stage"};
  Definizione ed inizializzazione dell'array di char usato per definire lo stato delle piante durante la loro crescita, e definire una distinzione tra le fasi della pianta. Viene usato per memorizzare il nome dello stato di crescita delle piante a seconda della ricetta attiva.

5.2.6 Simboli dello schermo

In questa sezione sono memorizzati array multidimensionali con codifica binaria di 0 e 1. Ogni array è composto da 8 righe e 8 colonne che formano una sorta di griglia, dove in ogni cella è contenuto il valore binario. A seconda del valore della cella, è possibile spegnere ("0") o accendere ("1") i pixel dello schermo OLED creando dunque delle immagini. In *figura 64* sono riportati gli array multidimensionali utilizzati nel programma per la rappresentazione dei simboli sullo schermo.

```
132 /* ------------------------------------  Components symbol  ------------------------------------ */
133
134 // Light symbol
135 static const unsigned char PROGMEM light_symbol[] = {
136   B00111100,
137   B01000010,
138   B10000001,
139   B10011001,
140   B01011010,
141   B00100100,
142   B00011000,
143   B00011000
144 };
145
146 // Fan symbol
147 static const unsigned char PROGMEM fan_symbol[] = {
148   B11110011,
149   B11100111,
150   B01101111,
151   B00111001,
152   B10011100,
153   B11110110,
154   B11100111,
155   B11001111
156 };
157
```

(a) *Parte 1*

```
158 // Water symbol
159 static const unsigned char PROGMEM water_symbol[] = {
160   B00010000,
161   B00011000,
162   B00111100,
163   B01111110,
164   B01111110,
165   B01111110,
166   B01111110,
167   B00111100
168 };
169
170 // Baffle-close symbol
171 static const unsigned char PROGMEM baffle_close_symbol[] = {
172   B00000000,
173   B00000000,
174   B01111111,
175   B11111111,
176   B11001100,
177   B11111111,
178   B11111111,
179   B11111111
180 };
181
```

(b) *Parte 2*

```
182 // Baffle-middle symbol
183 static const unsigned char PROGMEM baffle_middle_symbol[] = {
184   B00000000,
185   B00000000,
186   B00110011,
187   B01100110,
188   B11001100,
189   B11111111,
190   B11111111,
191   B11111111
192 };
193
194 // Baffle-open symbol
195 static const unsigned char PROGMEM baffle_open_symbol[] = {
196   B00000000,
197   B11000000,
198   B11001100,
199   B11001100,
200   B11001100,
201   B11111111,
202   B11111111,
203   B11111111
204 };
205
206 // Arrow symbol
207 static const unsigned char PROGMEM arrow_symbol[] = {
208   B00000000,
209   B00000000,
210   B00000100,
211   B00000110,
212   B11111111,
213   B00000110,
214   B00000100,
215   B00000000
216 };
```

(c) *Parte 3*

Figura 64: Parte di codice riguardante i simboli usati nello schermo OLED.

- riga 135: static const unsigned char PROGMEM [27] light_symbol[] =
  {
  B00111100,
  B01000010,
  B10000001,
  B10011001,
  B01011010,
  B00100100,
  B00011000,
  B00011000
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo della "lampadina". Questo viene usato nello schermo per segnalare l'attivazione dell'illuminazione della serra.

[27] PROGMEM è un comando usato per memorizzare nella memoria flash grandi quantità di dati, in questo caso array di char, che sono immutabili (const) in modo da ottimizzare il codice ed avere un accesso più rapido ai dati.

- riga 147: static const unsigned char PROGMEM fan_symbol[] = {
  B11110011,
  B11100111,
  B01101111,
  B00111001,
  B10011100,
  B11110110,
  B11100111,
  B11001111
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo della "ventola". Questo viene usato nello schermo per segnalare l'attivazione della ventilazione nella serra.

- riga 159: static const unsigned char PROGMEM water_symbol[] = {
  B00010000,
  B00011000,
  B00111100,
  B01111110,
  B01111110,
  B01111110,
  B01111110,
  B00111100
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo della "goccia d'acqua". Questo viene usato nello schermo per segnalare l'attivazione del ciclo di irrigazione delle piante della serra.

- riga 171: static const unsigned char PROGMEM baffle_close_symbol[] = {
  B00000000,
  B00000000,
  B01111111,
  B11111111,
  B11001100,
  B11111111,
  B11111111,
  B11111111
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo dei "deflettori chiusi". Questo viene usato nello schermo per segnalare la posizione delle alette.

- riga 183: static const unsigned char PROGMEM baffle_middle_symbol[] = {
  B00000000,
  B00000000,
  B00110011,
  B01100110,
  B11001100,
  B11111111,
  B11111111,
  B11111111
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo dei "deflettori semi-aperti". Questo viene usato nello schermo per segnalare la posizione delle alette.

- riga 195: static const unsigned char PROGMEM baffle_open_symbol[] = {
  B00000000,
  B11000000,
  B11001100,
  B11001100,
  B11001100,
  B11111111,
  B11111111,
  B11111111
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo dei "deflettori aperti". Questo viene usato nello schermo per segnalare la posizione delle alette.

- riga 207: static const unsigned char PROGMEM arrow_symbol[] = {
  B00000000,
  B00000000,
  B00000100,
  B00000110,
  B11111111,
  B00000110,
  B00000100,
  B00000000
  };
  Definizione ed inizializzazione dell'array di char per la memorizzazione del simbolo della "freccia" usato a scopo puramente estetico nello schermo.

5.2.7 Subroutines

La sezione riguardante le subroutines del programma, riportata in *figura 65*, comprende tutte le funzioni implementate per ottenere il ciclo automatico della serra. Di fatto, sono contenute tutte quelle richiamate nella funzione *LOOP* del codice caricato nella scheda Arduino.

```
217
218
219
220 /* ---------------------------------------  Subroutines  --------------------------------------- */
221
222 // Function for read temperature and humidity values
223 void read_temp_hum(float *temp, float *hum)
224 {
225   *temp = dht.readTemperature();                                    // Temperature acquisition
226   *hum = dht.readHumidity();                                        // Humidity acquisition
227 }
228
229
230
231 // Function for read the brightness value
232 void read_brightness(int *brig)
233 {
234   for (int i = 0; i < 5; i++)
235     *brig += analogRead(phototransistor_pin);                       // Brightness acquisition
236
237   *brig = *brig / 5;                                                // Average of the acquired values
238   *brig = map(*brig, 0, 1023, 0, 100);                              // Mapping of values into range [0, 100]
239 }
240
241
242
243 // Function for read date and time values
244 void read_date_time(DateTime *cur_time)
245 {
246   *cur_time = rtc.now();                                            // Read data from the RTC Chip
247 }
248
249
250
```

(a) *Parte 1*

- Funzione “read_temp_hum”:
  Riceve: puntatori alle variabili “temperature” e “humidity”.
  Restituisce: void.
  Descrizione: Funzione che permette di aggiornare il valore della temperatura e dell’umidità nella variabile puntata, rispettivamente “temperature” e in seguito “humidity”. Viene utilizzato un comando della libreria $< DHT.h >$ per leggere il valore rilevato dal sensore di temperatura ed umidità *DHT11*, e imporre quindi l’assegnamento e l’aggiornamento sulle variabili dedicate.

- Funzione “read_brightness”:
  Riceve: puntatore alla variabile “brightness”.
  Restituisce: void.
  Descrizione: Funzione che permette di aggiornare il valore di luminosità. Per mezzo del ciclo *for*, vengono eseguite 5 letture del valore di luminosità rilevato dal fototransistor, collegato alla scheda Arduino. In questo modo, è possibile avere dei valori più attendibili rispetto ad eseguire una sola misurazione. Successivamente alle misurazioni, viene eseguita una media dei valori ottenuti. In seguito, il valore medio viene “rimappato”, grazie alla funzione *map*, all’interno di un intervallo

[0, 100]: questo rappresenterà la percentuale di luce esterna. Viene eseguita questa operazione poiché ogni segnale analogico, che sia di ingresso o di uscita, viene convertito internamente ad una soglia di valori compresa tra 0 e 4095 (ovvero 2^{12}, dove 12 sono i bit dell'architettura di Arduino) che rappresenta il range di tensione del piedino tra 0 e 3.3V.

- Funzione "read_date_time":
  Riceve: puntatore alla variabile "current_time".
  Restituisce: void.
  Descrizione: Funzione che permette di aggiornare il valore della data e dell'orario all'interno del programma. Grazie al comando della libreria $< RTClib.h >$, è possibile leggere dal Clock la data e l'orario per memorizzarli nell'opportuna variabile.

```
251 // Function for select and change greenhouse parameters
252 void change_parameters(int *param, int *temp_max, int *temp_min, int *hum_max, int *hum_min, int *h_light)
253 {
254   if (digitalRead(button_pin_2)) {
255     tone(warning_buzzer, PITCH, BUZZER_TIME);
256     *param = *param + 1;
257 
258     if (*param > 5)
259       *param = 0;
260   }
261   switch (*param) {
262     case 0:
263       // Germination
264       *temp_max = 28;
265       *temp_min = 21;
266       *hum_max = 65;
267       *hum_min = 50;
268       *h_light = 18;
269       break;
270     case 1:
271       // Cutting
272       *temp_max = 28;
273       *temp_min = 24;
274       *hum_max = 90;
275       *hum_min = 65;
276       *h_light = 18;
277       break;
278     case 2:
279       // Rooted plant
280       *temp_max = 28;
281       *temp_min = 23;
282       *hum_max = 75;
283       *hum_min = 65;
284       *h_light = 18;
285       break;
```

(b) *Parte 2*

```
286     case 3:
287       // Vegetative phase
288       *temp_max = 28;
289       *temp_min = 23;
290       *hum_max = 75;
291       *hum_min = 65;
292       *h_light = 18;
293       break;
294     case 4:
295       // Flowering/Fruiting
296       *temp_max = 28;
297       *temp_min = 21;
298       *hum_max = 50;
299       *hum_min = 40;
300       *h_light = 12;
301       break;
302     case 5:
303       // Final stage
304       *temp_max = 28;
305       *temp_min = 21;
306       *hum_max = 50;
307       *hum_min = 40;
308       *h_light = 12;
309       break;
310     default:
311       // Wrong parameter!
312       *temp_max = 0;
313       *temp_min = 0;
314       *hum_max = 0;
315       *hum_min = 0;
316       *h_light = 0;
317   }
318 }
319
320
```

(c) *Parte 3*

- Funzione "change_parameters:"
  Riceve: puntatori alle variabili "parameter", "temperature_max", "temperature_min", "humidity_max", "humidity_min", "hours_of_lights"
  Restituisce: void.
  Descrizione: Funzione che permette di aggiornare i valori delle ricette definite per ogni stato di crescita delle piante. La prima operazione da svolgere è quella di verificare se il pulsante "button_pin_2" è premuto. Questo pulsante è adibito al cambiamento dei parametri di crescita delle piante. Se questa condizione è verificata, viene incrementato il valore della variabile "parameter", usata per selezionare una sola delle ricette contenute nel successivo *switch-case*. Se il valore della variabile "parameter" assume valori fuori dall'intervallo definito (ovvero >5, in quanto cinque sono le ricette disponibili), si provvede al ripristino al valore 0 della variabile. Successivamente nel ciclo *switch* si seleziona la ricetta imposta dal valore della variabile "parameter": in questo modo si acquisiscono i valori per le variabili "temperature_max", "temperature_min", "humidity_max", "humidity_min", "hours_of_lights", a seconda dello stato di crescita della pianta selezionato. In caso di un

valore della variabile “parameter” errato, i parametri sono posti a 0 per motivi di sicurezza. I valori dei parametri delle ricette sono personalizzabili a seconda del tipo di pianta coltivata in quel momento nella serra. Nell’immagine si riferiscono a quelli ottimali per la crescita della pianta di basilico.

```
321
322 // Function for switch page on display OLED
323 void change_page(uint8_t *pag)
324 {
325   if (digitalRead(button_pin_1)) {
326     tone(warning_buzzer, PITCH, BUZZER_TIME);
327     *pag = *pag + 1;
328
329     if (*pag > 1)
330       *pag = 0;
331   }
332 }
333
334
335
```

(d) *Parte 4*

- Funzione “change_page”:
  Riceve: puntatore alla variabile “page”.
  Restituisce: void.
  Descrizione: Funzione che permette di cambiare la pagina visualizzata nello schermo OLED. Se il pulsante “button_pin_1” è premuto, ovvero il pulsante adibito al cambio di pagina, viene fatto un incremento del valore contenuto nella variabile “page” in quel momento e viene memorizzato. Se il valore assunto da “page” è >1, viene ripristinato il valore a “0” e dunque visualizzata la pagina principale.

```
// Subroutine for the management of sensor errors
void error_handling(float *temp, float *hum, int *brig, bool *err_dht, bool *err_photo, bool *err_rtc)
{
  // Temperature and Humidity values
  if (isnan(*temp) || isnan(*hum))                                  // Wrong reading temperature or humidity values
    *err_dht = true;                                                // Activation the DHT sensor error
  else
    *err_dht = false;                                               // Deactivation the DHT sensor error


  // Brightness value
  if (*brig < 0 || *brig > 100)                                     // Wrong reading/mapping brightness value
    *err_photo = true;                                              // Activation the HW5P-1 phototransistor error
  else
    *err_photo = false;                                             // Deactivation the HW5P-1 phototransistor error


  // Date and Time
  if (!rtc.begin())                                                 // RTC library code correct started
    *err_rtc = true;                                                // Activation the RTC clock error
  else
    *err_rtc = false;                                               // Deactivation the RTC clock error



  // System error report
  if (*err_dht || *err_photo || *err_rtc) {
    digitalWrite(warning_led, !digitalRead(warning_led));           // Simple blinking of warning led
    tone(warning_buzzer, PITCH, BUZZER_TIME);                       // Buzzer sound to signal the error
  } else {
    digitalWrite(warning_led, LOW);                                 // Deactivation of blinking warning led
  }
}
```

(e) *Parte 5*

- Funzione “error_handling”:
  Riceve: puntatori alle variabili “temperature”, “humidity”, “brightness”, “error_dht”, “error_phototransistor”, “error_rtc”.
  Restituisce: void.
  Descrizione: Funzione per la gestione degli errori dei componenti collegati alla scheda. Grazie a una serie di cicli *if*, e funzioni contenute nelle librerie $< DHT.h >$ e $< RTClib.h >$, permette di verificare eventuali stati di errore rispettivamente del sensore di temperatura ed umidità e del Clock. Per prima cosa si verifica il corretto funzionamento del sensore *DHT11*, in seguito quello del fototransistor e infine dell’RTC Clock *DS1307*. Uno stato logico *false* delle variabili “error_dht”, “error_phototransistor” ed “error_rtc” implica che tutti i dispositivi sono collegati alla scheda e funzionano correttamente. Uno stato *true* di almeno una delle variabili implica il rispettivo malfunzionamento del dispositivo a cui si riferiscono. Per il sensore *DHT11* si verifica la presenza di un errore di lettura dei valori ambientali esterni, attraverso una funzione della libreria dedicata. Per il fototransistor invece, si verifica che il “rimappamento” eseguito successivamente all’acquisizione dei valori, e alla media di questi, sia contenuto nell’intervallo opportuno, ovvero [0, 100]. Infine, per il Clock RTC si verifica la corretta inizializzazione, e dunque che comunichi con Arduino. Nell’ultima parte della funzione, se una o più di queste variabili si trova nello stato *true*, viene segnalata la presenza del/degli errore/i attraverso il lampeggiamento del led collegato al pin “warning_led” e attivando il buzzer collegato al pin di uscita “warning_buzzer”, con la tonalità ed il tem-

po impostati nelle rispettive variabili “PITCH” e “BUZZER_TIME”. Per eseguire il lampeggiamento del led, semplicemente ad ogni ciclo si impone uno stato logico del pin opposto a quello del ciclo precedente.

```
371
372 // Subroutine for the management of the automatic cycle and the parameters of the greenhouse, and for the management of related errors
373 void greenhouse_management(DateTime *cur_t, DateTime *fut_fan_t, DateTime *fut_pump_t, float *temp, float *hum, int *brig, int *temp_max, int *temp_min, int *hum_max,
374                            int *hum_min, int *h_light, bool *fan_ctrl, bool *t_fan, int *baffle_pos, bool *lights_ctrl, uint8_t *lights_on_t, uint8_t *lights_off_t, bool *pump_ctrl,
375                            bool *t_pump)
376 {
377   // Fan management automated cycle
378   if (*temp > *temp_min && *hum > *hum_min) {                                          // Check parameters are over the minimum values
379     if (*temp > *temp_max || *hum > *hum_max) {                                        // Temperature and humidity over of maximum values
380       *baffle_pos = BAFFLE_OPEN;                                                     // Open the baffle for activation of fan
381       *fan_ctrl = true;                                                              // Fan activation
382       *t_fan = false;                                                                // Setting the value for deactivate fan when parameters come into corret range
383       *fut_fan_t = *cur_t;                                                           // Copy "current time" for deactivate fan when parameters come into correct range
384     } else {
385       if (*t_fan) {                                                                  // Cycle for setting the future times for activate/deactivate automatic fan
386         if (!*fan_ctrl) {                                                            // Fan OFF condition
387           *fut_fan_t = (*cur_t + TimeSpan(0, 0, FAN_OFF_TIME_MIN, FAN_OFF_TIME_SEC));// Setting the future time before fan activation
388           *baffle_pos = BAFFLE_MIDDLE;                                               // Middle position for baffle (standard position)
389           *t_fan = false;                                                            // Variable setting for subsequent cycles (setting future time only after completion of interme
390         } else {
391           *fut_fan_t = (*cur_t + TimeSpan(0, 0, FAN_ON_TIME_MIN, FAN_ON_TIME_SEC));  // Setting the future time before fan deactivation
392           *t_fan = false;                                                            // Variable setting for subsequent cycles (setting future time only after completion of interme
393         }
394       }
395       // "Current time" and "future time" matching condition; automatic activation/deactivation cycle
396       if ((cur_t->hour() >= fut_fan_t->hour()) && (cur_t->minute() >= fut_fan_t->minute()) && (cur_t->second() >= fut_fan_t->second())) {
397         if (!*fan_ctrl) {                                                            // Fan OFF condition
398           *baffle_pos = BAFFLE_OPEN;                                                 // Open the baffle for activation of fan
399           *fan_ctrl = true;                                                          // Fan activation
400           *t_fan = true;                                                             // Variable setting for repeat setting new future time (time before deactivation)
401         } else {
402           *baffle_pos = BAFFLE_MIDDLE;                                               // Middle position for baffle (standard position)
403           *fan_ctrl = false;                                                         // Fan deactivation
404           *t_fan = true;                                                             // Variable setting for repeat setting new future time (time before activation)
405         }
```

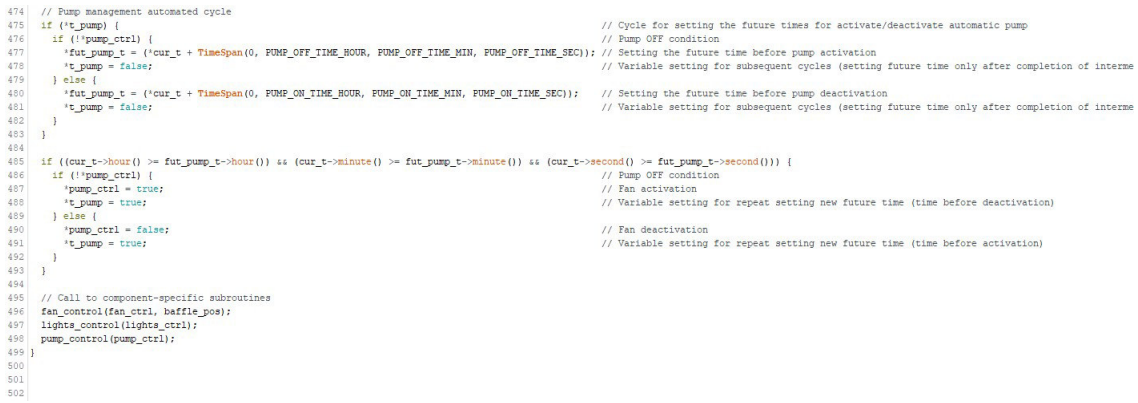
(f) *Parte 6*

```
406       }
407     }
408   } else {
409     *fan_ctrl = false;                                                               // Fan deactivation
410     *baffle_pos = BAFFLE_CLOSE;                                                      // Close the baffle (Fan OFF)
411     *t_fan = true;                                                                   // Setting the value for setting new value of "future fan time" the next cycle when parameters
412   }
413
414   // Led management automated cycle
415   if (*h_light == 12) {
416     switch (cur_t->month()) {                                                        // Parameters setting when "hours of light" is set on 12
417       // Months: January, February, November, December
418       case 1: case 2: case 11: case 12:
419         *lights_on_t = 7;
420         *lights_off_t = 19;
421         break;
422       // March, April, September, October
423       case 3: case 4: case 9: case 10:
424         *lights_on_t = 7;
425         *lights_off_t = 19;
426         break;
427       // May, June, July, August
428       case 5: case 6: case 7: case 8:
429         *lights_on_t = 6;
430         *lights_off_t = 18;
431         break;
432       // Default settings in case of problems
433       default:
434         *lights_on_t = 0;
435         *lights_off_t = 0;
436         break;
437     }
```

(g) *Parte 7*

```
438   } else if (*h_light == 18) {                                          // Parameters setting when "hours of light" is set on 18
439     switch (cur_t->month()) {
440       // Months: January, February, November, December
441       case 1: case 2: case 11: case 12:
442         *lights_on_t = 4;
443         *lights_off_t = 22;
444         break;
445       // March, April, September, October
446       case 3: case 4: case 9: case 10:
447         *lights_on_t = 4;
448         *lights_off_t = 22;
449         break;
450       // May, June, July, August
451       case 5: case 6: case 7: case 8:
452         *lights_on_t = 4;
453         *lights_off_t = 22;
454         break;
455       // Default settings in case of problems
456       default:
457         *lights_on_t = 0;
458         *lights_off_t = 0;
459         break;
460     }
461   } else {                                                              // Incorrect "hour of lights" condition (every other case)
462     *lights_ctrl = false;                                               // Setting value of "hours of lights" failed, led OFF
463   }
464
465   if (*brig <= BRIGHTNESS_MIN) {
466     if ((cur_t->hour() >= *lights_on_t) && (cur_t->hour() < *lights_off_t))
467       *lights_ctrl = true;                                              // Lights activation
468     else
469       *lights_ctrl = false;                                             // Reset lights activation out of lights period for plants
470   } else if (*brig >= (BRIGHTNESS_MIN + 8)){                            // Hysteresis condition (to prevent continuous activation/deactivation)
471     *lights_ctrl = false;                                               // Lights deactivation
472   }
```

(h) *Parte 8*



```
474   // Pump management automated cycle
475   if (*t_pump) {                                                        // Cycle for setting the future times for activate/deactivate automatic pump
476     if (!*pump_ctrl) {                                                  // Pump OFF condition
477       *fut_pump_t = (*cur_t + TimeSpan(0, PUMP_OFF_TIME_HOUR, PUMP_OFF_TIME_MIN, PUMP_OFF_TIME_SEC)); // Setting the future time before pump activation
478       *t_pump = false;                                                  // Variable setting for subsequent cycles (setting future time only after completion of interme
479     } else {
480       *fut_pump_t = (*cur_t + TimeSpan(0, PUMP_ON_TIME_HOUR, PUMP_ON_TIME_MIN, PUMP_ON_TIME_SEC));    // Setting the future time before pump deactivation
481       *t_pump = false;                                                  // Variable setting for subsequent cycles (setting future time only after completion of interme
482     }
483   }
484
485   if ((cur_t->hour() >= fut_pump_t->hour()) && (cur_t->minute() >= fut_pump_t->minute()) && (cur_t->second() >= fut_pump_t->second())) {
486     if (!*pump_ctrl) {                                                  // Pump OFF condition
487       *pump_ctrl = true;                                                // Fan activation
488       *t_pump = true;                                                   // Variable setting for repeat setting new future time (time before deactivation)
489     } else {
490       *pump_ctrl = false;                                               // Fan deactivation
491       *t_pump = true;                                                   // Variable setting for repeat setting new future time (time before activation)
492     }
493   }
494
495   // Call to component-specific subroutines
496   fan_control(fan_ctrl, baffle_pos);
497   lights_control(lights_ctrl);
498   pump_control(pump_ctrl);
499 }
500
501
502
```

(i) *Parte 9*

- Funzione “greenhouse_management”:
  Riceve: puntatori alle variabili “current_time”, “future_fan_time”, “future_pump_time”, “temperature”, “humidity”, “brightness”, “temperature_max”, “temperature_min”, “humidity_max”, “humidity_min”, “hours_of_lights”, “fan”, “timed_fan”, “baffle_position”, “lights”, “lights_on_time”, “lights_off_time”, “pump”, “timed_pump”.
  Restituisce: void.
  Descrizione: Funzione che gestisce tutta la parte automatica di controllo e gestione della serra idroponica. In questa funzione vengono gestiti il ciclo di ventilazione, quello di irrigazione e l’illuminazione della serra. Per prima cosa, viene gestita la ventola. Il primo ciclo *if* (riga 378) controlla che le condizioni ambientali siano superiori a quelle minime richieste, in modo da poter attuare il controllo che segue dopo. Il secondo ciclo *if* (riga 379), verifica se la temperatura o l’umidità rilevate sono superiori alle soglie critiche: se questa condizione

è verificata, il programma passa ad attivare il segnale di controllo per l'attivazione della ventola. Inoltre provvede ad impostare i parametri per il mantenimento della ventola accesa, fino a quando i valori ambientali non saranno ritornati all'interno dell'intervallo corretto. Se questa condizione però non è soddisfatta, dunque i parametri sono all'interno del range corretto, il programma prosegue attraverso l'*else* (riga 384) del secondo ciclo *if*. Al suo interno sono presenti un altri due cicli *if* (righe 385 e 386) e un ciclo *else* (riga 390) che permettono il controllo della ventola in condizioni ambientali normali. Grazie a questi, è possibile gestire l'attivazione o la disattivazione del ciclo di ventilazione automatico impostando l'orario della futura commutazione della ventola, utilizzando le variabili definite all'inizio. Tutto questo viene eseguito solo se i parametri ambientali esterni rimangano all'interno dell'intervallo corretto. Se questi ultimi invece, nei cicli successivi, si troveranno a dei valori al di fuori del range, questa parte verrà esclusa e sarà attuato un altro tipo di controllo, a seconda del valore dei parametri ambientali. In questo caso, la commutazione della ventola sarà gestita automaticamente dal programma e i valori sul tempo futuro di commutazione della ventola esclusi. Nel caso in cui il programma proceda con dei valori ambientali all'interno dell'intervallo corretto, ad ogni ciclo viene eseguito un controllo per verificare che il tempo attuale combaci con il tempo futuro per la commutazione della ventola, attraverso un ulteriore ciclo *if* (riga 396). Se il confronto è positivo, si procede a commutare lo stato della ventola altrimenti si mantiene lo stato precedente. Ad ogni commutazione della ventola, si definisce inoltre la corretta posizione dell'albero del servomotore collegato ai deflettori, attraverso l'uso delle variabili dedicate definite in precedenza. Se i parametri sono inferiori a quelli minimi (*else* di riga 408), si disattiva la ventola e si chiudono i deflettori in modo da mantenere temperatura ed umidità interni alla serra.
In seguito alla gestione della ventilazione si passa a quella dell'illuminazione della serra. Per prima cosa, si impostano i valori di accensione e spegnimento dell'illuminazione della serra, facendo uso della funzione *switch-case*. Si verifica, attraverso i cicli *if* (riga 415 e 438), il valore contenuto nella variabile che riporta le ore di luce necessarie alle piante. Questo valore era stato definito in precedenza nelle ricette che differenziano gli stati di crescita. In seguito, dopo il confronto, si selezionano gli orari di accensione e spegnimento dell'illuminazione della serra. I valori variano a seconda del mese per cercare di ridurre al minimo l'apporto di luce artificiale alle piante e sfruttare il più possibile quella naturale, in ottica di risparmio energetico. Con il successivo ciclo *if* (riga 465) si verifica che la quantità di luce esterna sia maggiore a quella minima impostata all'inizio del programma. In caso di una quantità di luce inferiore, si effettua un ulteriore controllo: se l'orario

attuale è contenuto all'interno tra l'orario di attivazione e quello di disattivazione dell'illuminazione, allora vuol dire che in quel momento le piante dovrebbero ricevere luce, ma l'ambiente esterno non gliela sta fornendo naturalmente. Si provvede dunque ad accendere l'illuminazione per fornire la luce mancante in modo artificiale. In caso contrario, si provvede a disattivare l'illuminazione della serra.
Dopo il ciclo di ventilazione ed illuminazione, viene gestito il ciclo di irrigazione della serra. Similmente a quanto accade per la ventola, anche in questo caso si usano le funzioni contenute nella libreria $< RTClib.h >$ per l'impostazione degli orari per la commutazione della pompa. Attraverso i cicli *if* e *else* (righe 476 e 479) è possibile impostare l'orario e la data per la successiva commutazione della pompa. Come ultima operazione sul ciclo di irrigazione, viene effettuato un controllo attraverso un ciclo *if* (riga 485) per verificare che il tempo attuale combaci con la data e l'orario impostati per la commutazione: se la condizione è verificata, lo stato della pompa viene commutato altrimenti rimane invariato.
Dopo questa parte di gestione automatica, vengono richiamate le funzioni usate per il controllo dei dispositivi (righe 496, 497 e 498). In particolare, permettono l'attuazione dell'eventuale commutazione logica dei componenti gestiti in precedenza, ovvero la ventola, l'illuminazione della serra e la pompa.

```
503 // Fan management function
504 void fan_control(bool *fan_ctrl, int *baffle_pos)
505 {
506   if (*fan_ctrl)
507     digitalWrite(fan_pin, HIGH);                    // Activation fan
508   else
509     digitalWrite(fan_pin, LOW);                     // Deactivation fan
510
511   baffle.write(*baffle_pos);                        // Set baffle position
512 }
513
514
515
516 // Lights management function
517 void lights_control(bool *lights_ctrl)
518 {
519   if (*lights_ctrl)
520     digitalWrite(lights_pin, HIGH);                 // Activation illumination led
521   else
522     digitalWrite(lights_pin, LOW);                  // Deactivation illumination led
523 }
524
525
526
527 // Pump management function
528 void pump_control(bool *pump_ctrl)
529 {
530   if (*pump_ctrl)
531     digitalWrite(pump_pin, HIGH);                   // Activation pump
532   else
533     digitalWrite(pump_pin, LOW);                    // Deactivation pump
534 }
535
536
537
```

(j) *Parte 10*

- Funzione "fan_control":
  Riceve: puntatori alle variabili "fan" e "baffle_position".
  Restituisce: void.
  Descrizione: Semplice funzione che commuta lo stato della ventola a seconda del valore della variabile dedicata. Inoltre, imposta l'an-

golo dei deflettori andando ad applicare la rotazione all'albero del servomotore collegato a questi ultimi.

- Funzione "lights_control":
  Riceve: puntatore alla variabile "lights".
  Restituisce: void.
  Descrizione: Semplice funzione che commuta lo stato dei led dell'illuminazione della serra a seconda del valore della variabile dedicata.

- Funzione "pump_control":
  Riceve: puntatore alla variabile "pump".
  Restituisce: void.
  Descrizione: Semplice funzione che commuta lo stato della pompa a seconda del valore della variabile dedicata, per la gestione del ciclo automatico d'irrigazione della serra.

```
539 void serial_print_values(float *temp, float *hum, int *brig, int *temp_max, int *temp_min, int *hum_max, int *hum_min, int *h_light, int *param, char state[][19],
540                         DateTime *cur_t, DateTime *fut_fan_t, DateTime *fut_pump_t, bool *err_dht, bool *err_photo, bool *err_rtc, bool *fan, int *baffle_pos, bool *lights,
541                         uint8_t *lights_on_t, uint8_t *lights_off_t, bool *pump)
542 {
543   // Date and Time (1° line)
544   if (!*err_rtc) {                                                                    // Correct reading of date and time values
545     Serial.print(F("Date:  "));
546     prefix_zero_serial(cur_t->day());
547     Serial.print('/');
548     prefix_zero_serial(cur_t->month());
549     Serial.print('/');
550     Serial.print(cur_t->year());
551     Serial.print(F("    Time:  "));
552     prefix_zero_serial(cur_t->hour());
553     Serial.print(':');
554     prefix_zero_serial(cur_t->minute());
555     Serial.print(':');
556     prefix_zero_serial(cur_t->second());
557     Serial.println();
558   } else {                                                                            // Wrong reading of date and time values
559     Serial.println(F("Couldn't find RTC: Error!"));
560   }
561
562   // Temperature and Humidity (2° line)
563   if (!*err_dht) {                                                                    // Correct reading of temperature and humidity values
564     Serial.print(F("Temperature:  "));
565     Serial.print(*temp);
566     Serial.print(F("°C"));
567     Serial.print(F("    Humidity:  "));
568     Serial.print(*hum);
569     Serial.print('%');
```

(k) *Parte 11*

```
570   } else {                                                    // Wrong reading of temperature and humidity values
571     Serial.print(F("Temperature:  "));
572     Serial.print(F("Error!"));
573     Serial.print(F("    Humidity:  "));
574     Serial.print(F("Error!"));
575   }
576   // Brightness
577   if (!*err_photo) {                                          // Correct reading of brightness value
578     Serial.print(F("    Brightness:  "));
579     Serial.print(*brig);
580     Serial.println('%');
581   } else {                                                    // Wrong readding of brightness value
582     Serial.print(F("    Brightness:  "));
583     Serial.println(F("Error!"));
584   }
585
586   // Baffle position (3° line)
587   Serial.print(F("Baffle position:  "));
588   switch (*baffle_pos) {
589     case BAFFLE_OPEN:
590       Serial.println(F("open"));
591       break;
592     case BAFFLE_MIDDLE:
593       Serial.println(F("middle"));
594       break;
595     case BAFFLE_CLOSE:
596       Serial.println(F("close"));
597       break;
598   }
599
600   // Greenhouse parameters (4° line)
601   Serial.print(F("State of plants:  "));
602   Serial.println(state[*param]);
603
```

(l) *Parte 12*

```
604   // Greenhouse parameters (5° line)
605   Serial.print(F("Temperature max:  "));
606   Serial.print(*temp_max);
607   Serial.println(F("°C"));
608
609   // Greenhouse parameters (6° line)
610   Serial.print(F("Temperature min:  "));
611   Serial.print(*temp_min);
612   Serial.println(F("°C"));
613
614   // Greenhouse parameters (7° line)
615   Serial.print(F("Humidity max:  "));
616   Serial.print(*hum_max);
617   Serial.println('%');
618   Serial.print(F("Humidity min:  "));
619   Serial.print(*hum_min);
620   Serial.println('%');
621
622   // Greenhouse parameters (8° line)
623   Serial.print(F("Hours of light:  "));
624   Serial.print(*h_light);
625   Serial.println('h');
626
627   // Greenhouse parameters (9° line)
628   Serial.print("Brightness min:  ");
629   Serial.print(BRIGHTNESS_MIN);
630   Serial.println('%');
631
```

(m) *Parte 13*

```
632   // Fan parameters (10° line)
633   Serial.print(F("Fan state:     "));
634   if (*fan)
635     Serial.print(F("activate  "));
636   else
637     Serial.print(F("deactivate"));
638   Serial.print(F("    Time OFF:  "));
639   Serial.print(F("00:"));
640   prefix_zero_serial(FAN_OFF_TIME_MIN);
641   Serial.print(':');
642   prefix_zero_serial(FAN_OFF_TIME_SEC);
643   Serial.print(F("    Time ON:  "));
644   Serial.print(F("00:"));
645   prefix_zero_serial(FAN_ON_TIME_MIN);
646   Serial.print(':');
647   prefix_zero_serial(FAN_ON_TIME_SEC);
648   Serial.print(F("    (FUTURE:  "));
649   prefix_zero_serial(fut_fan_t->hour());                    // Print of the future time for the activation/deactivation of the fan
650   Serial.print(':');
651   prefix_zero_serial(fut_fan_t->minute());
652   Serial.print(':');
653   prefix_zero_serial(fut_fan_t->second());
654   Serial.println(')');
655
```

(n) *Parte 14*

```
656   // Led parameters (11° line)
657   Serial.print(F("Lights state:  "));
658   if (*lights)
659     Serial.print(F("activate  "));
660   else
661     Serial.print(F("deactivate"));
662   Serial.print(F("    Time OFF:  "));
663   prefix_zero_serial(*lights_off_t);
664   Serial.print(F(":00:00"));
665   Serial.print(F("    Time ON:  "));
666   prefix_zero_serial(*lights_on_t);
667   Serial.println(F(":00:00"));
668
669   // Pump parameters (12° line)
670   Serial.print(F("Pump state:    "));
671   if (*pump)
672     Serial.print(F("activate  "));
673   else
674     Serial.print(F("deactivate"));
675   Serial.print(F("    Time OFF:  "));
676   prefix_zero_serial(PUMP_OFF_TIME_HOUR);
677   Serial.print(':');
678   prefix_zero_serial(PUMP_OFF_TIME_MIN);
679   Serial.print(':');
680   prefix_zero_serial(PUMP_OFF_TIME_SEC);
681   Serial.print(F("    Time ON:  "));
682   prefix_zero_serial(PUMP_ON_TIME_HOUR);
683   Serial.print(':');
684   prefix_zero_serial(PUMP_ON_TIME_MIN);
685   Serial.print(':');
686   prefix_zero_serial(PUMP_ON_TIME_SEC);
687   Serial.print(F("    (FUTURE:  "));
688   prefix_zero_serial(fut_pump_t->hour());                   // Print of the future time for the activation/deactivation of the pump
```

(o) *Parte 15*

```
689   Serial.print(':');
690   prefix_zero_serial(fut_pump_t->minute());
691   Serial.print(':');
692   prefix_zero_serial(fut_pump_t->second());
693   Serial.println(')');
694
695   // Wait time for printing lines
696   Serial.println();
697   Serial.flush();
698 }
699
700
701
```

(p) *Parte 16*

- Funzione “serial_print_values”:
  Riceve: puntatori alle variabili “temperature”, “humidity”, “brightness”, “temperature_max”, “temperature_min”, “humidity_max”, “humidity_min”, “hours_of_lights”, “parameter”, “state_of_plants”, “current_time”, “future_fan_time”, “future_pump_time”, “error_dht”, “error_photoresistor”, “error_rtc”, “fan”, “baffle_position”, “lights”, “lights_on_time”, “lights_off_time”, “pump”.
  Restituisce: void.
  Descrizione: Funzione per la stampa dei parametri del programma a terminale. Ponendo a “1” la variabile *DEBUG* all’inizio del programma, è possibile spostare la stampa delle informazioni dal display OLED al terminale. In questo modo, si ricevono più informazioni di quelle mostrate per default e si possono controllare eventuali problemi o malfunzionamenti a livello logico del programma. La funzione è decisamente lunga, ma ripetitiva nelle operazioni svolte. Nella prima riga troviamo la stampa della data e dell’ora, con eventuali messaggi di errore nel caso in cui il Clock RTC non sia correttamente collegato con Arduino. La distinzione sulla stampa dei valori viene eseguita attraverso i cicli *if* e *else* (righe 544 e 558). La seconda riga serve per visualizzare i valori di temperatura ed umidità rilevati dal sensore *DHT11* e nel caso segnalare un messaggio di errore. Anche in questo caso, la distinzione viene decisa a seconda di cicli *if* e *else* (righe 563 e 570). Sempre in questa riga, viene riportato anche il valore di luminosità o eventualmente il suo messaggio di errore, attraverso i cicli *if* e *else* dedicati (righe 577 e 581). Nella terza riga viene invece riportata la posizione dei deflettori. Nella quarta lo stato di crescita delle piante, dunque la ricetta selezionata in quell’istante. Dalla sesta alla nona riga invece, vengono riportati i parametri di temperatura ed umidità (massimi e minimi), le ore di luce e la luminosità minima che sono associati al alla ricetta caricata nel programma. Nella decima riga viene riportato lo stato logico della ventola con i relativi tempi di mantenimento dello stato attivo o spento e l’orario della prossima commutazione logica. Nell’undicesima è riportato lo stato dell’illuminazione della serra: inoltre vengono mostrati anche il tempo di accensione e spegnimento dei led, a seconda della ricetta selezionata. Nella dodicesima riga lo stato della pompa con il tempo di mantenimento dello stato attivo o spento, e l’orario della prossima commutazione logica. In tutta questa parte del codice viene fatto uso di una funzione grafica ausiliaria, riportata in *figura 65w*, per porre uno “0” davanti ai numeri minori di 10, in quanto Arduino restituisce i valori interi. Si tratta solo di un’implementazione a puro scopo grafico.

```
// Subroutine for printing the values to the OLED display ("DEBUG" mode -> deactivated)
void oled_print_values(float *temp, float *hum, int *brig, int *temp_max, int *temp_min, int *hum_max, int *hum_min, int *h_light, int *param, char state[][19],
                       DateTime *cur_t, bool *err_dht, bool *err_photo, bool *err_rtc, bool *fan, int *baffle_pos, bool *lights, uint8_t *lights_on_t,
                       uint8_t *lights_off_t, bool *pump, uint8_t *pag)
{
  display.clearDisplay();

  // Date and Time (1° line)
  if (!*err_rtc) {                                                                    // Correct reading of date and time values
    display.setCursor(0, 0);
    prefix_zero_display(cur_t->day());
    display.print('/');
    prefix_zero_display(cur_t->month());
    display.print('/');
    display.print(cur_t->year());
    display.setCursor(80, 0);
    prefix_zero_display(cur_t->hour());
    display.print(':');
    prefix_zero_display(cur_t->minute());
    display.print(':');
    prefix_zero_display(cur_t->second());
  } else {
    display.setCursor(0, 0);
    display.print(F("DATE & TIME, FAILED!"));
  }
  display.drawLine(0, 9, display.width()-1, 9, SSD1306_WHITE);                        // Draw the line under date and time

  // Section for printing different values depending on the page number
  switch (*pag) {
    // Page: 0 (first page, greenhouse variables)
    case 0:
      // Temperature and Humidity (2°-3° lines)
      display.setCursor(0, 12);
```

(q) *Parte 17*

```
      if (!*err_dht) {                                                                // Correct reading of temperature and humidity values
        // Temperature (2° line)
        display.print(F("Temperature:"));
        display.setCursor(75, 12);
        display.print(*temp);
        display.setCursor(111, 12);
        display.print('C');

        // Humidity (3° line)
        display.setCursor(0, 21);
        display.print(F("Humidity:"));
        display.setCursor(75, 21);
        display.print(*hum);
        display.setCursor(111, 21);
        display.print('%');
      } else {                                                                        // Wrong reading of temperature and humidity values
        // Temperature (2° line)
        display.print(F("Temperature:"));
        display.setCursor(75, 12);
        display.print(F("Error!"));

        // Humidity (3° line)
        display.setCursor(0, 21);
        display.print(F("Humidity:"));
        display.setCursor(75, 21);
        display.print(F("Error!"));
      }

```

(r) *Parte 18*

```
763      // Brightness (4° line)
764      display.setCursor(0, 30);
765      display.print(F("Brightness: "));
766      if (!*err_photo) {                                                          // Correct reading of brightness value
767        display.setCursor(75, 30);
768        display.print(*brig);
769        display.setCursor(88, 30);
770        display.print('%');
771      } else {                                                                    // Wrong readding of brightness value
772        display.setCursor(75, 30);
773        display.print(F("Error!"));
774      }
775
776      // State of plants (5° line)
777      display.setCursor(0, 39);
778      display.print(state[*param]);
779
780
781      // Quadrant of symbols
782      display.drawLine(display.width()-54, display.height()-12, display.width()-1, display.height()-12, SSD1306_WHITE);              // Top line
783      display.drawLine(display.width()-54, display.height()-1, display.width()-54, display.height()-12, SSD1306_WHITE);              // Left line
784      display.drawLine(display.width()-1, display.height()-1, display.width()-1, display.height()-12, SSD1306_WHITE);                // Right line
785      display.drawLine(display.width()-41, display.height()-1, display.width()-41, display.height()-12, SSD1306_WHITE);              // Intermediate line
786      display.drawLine(display.width()-28, display.height()-1, display.width()-28, display.height()-12, SSD1306_WHITE);              // Intermediate line
787      display.drawLine(display.width()-15, display.height()-1, display.width()-15, display.height()-12, SSD1306_WHITE);              // Intermediate line
788
789      // Symbols
790      if (*pump)
791        display.drawBitmap(display.width()-51, display.height()-9, water_symbol, 8, 8, 1);           // Water symbol
792      if (*fan)
793        display.drawBitmap(display.width()-38, display.height()-9, fan_symbol, 8, 8, 1);             // Fan symbol
```

(s) *Parte 19*

```
794      switch (*baffle_pos) {
795        case BAFFLE_OPEN:
796          display.drawBitmap(display.width()-25, display.height()-9, baffle_open_symbol, 8, 8, 1);    // Baffle open symbol
797          break;
798        case BAFFLE_MIDDLE:
799          display.drawBitmap(display.width()-25, display.height()-9, baffle_middle_symbol, 8, 8, 1);  // Baffle middle symbol
800          break;
801        case BAFFLE_CLOSE:
802          display.drawBitmap(display.width()-25, display.height()-9, baffle_close_symbol, 8, 8, 1);   // Baffle close symbol
803          break;
804        default:
805          break;
806      }
807      if (*lights)
808        display.drawBitmap(display.width()-12, display.height()-9, light_symbol, 8, 8, 1);           // Light symbol
809      break;
810    // Page: 1 (second page, greenhouse variables for manage automatic cycle)
811    case 1:
812      // State of plants (2° line)
813      display.setCursor(0, 12);
814      display.print(state[*param]);
815
816      // Temperature values (3° line)
817      display.setCursor(0, 21);
818      display.print(F("Temp max:"));
819      display.setCursor(58, 21);
820      display.print(*temp_max);
821      display.setCursor(75, 21);
822      display.print(F("min:"));
823      display.setCursor(102, 21);
824      display.print(*temp_min);
825      display.setCursor(120, 21);
826      display.print('C');
827
```

(t) *Parte 20*

```
      // Humidity values (4° line)
      display.setCursor(0, 30);
      display.print(F("Hum max:"));
      display.setCursor(58, 30);
      display.print(*hum_max);
      display.setCursor(75, 30);
      display.print(F("min:"));
      display.setCursor(102, 30);
      display.print(*hum_min);
      display.setCursor(120, 30);
      display.print('%');

      // Brightness values (5° line)
      display.setCursor(0, 39);
      display.print(F("Bri min:"));
      display.setCursor(77, 39);
      display.print(BRIGHTNESS_MIN);
      display.setCursor(95, 39);
      display.print('%');

      // Hours light values (6° line)
      display.setCursor(0, 48);
      display.print(F("H light:"));
      display.setCursor(58, 48);
      display.print(*h_light);
      display.setCursor(82, 48);
      display.print('(');
      display.print(*lights_on_t);
      display.drawBitmap(95, 48, arrow_symbol, 8, 8, 1);                    // Arrow symbol
      display.setCursor(105, 48);
      display.print(*lights_off_t);
      display.print(')');
```

(u) *Parte 21*

```
      break;
    // Wrong page value
    default:
      display.setCursor(38, 35);
      display.print(F("Error page"));
      break;
  }

  display.display();                                                        // Send to display
}
```

(v) *Parte 22*

```
873 // Graphic function for add "0" before numbers < 10 in Date and Time in serial print
874 void prefix_zero_serial(uint8_t number)
875 {
876   if (number < 10) {                                                    // Numbers less than ten
877     Serial.print('0');                                                  // Print '0' before number value
878     Serial.print(number);
879   } else {
880     Serial.print(number);                                               // Print the number without '0'
881   }
882 }
883
884
885
886 // Graphic function for add "0" before numbers < 10 in Date and Time in display print
887 void prefix_zero_display(uint8_t number)
888 {
889   if (number < 10) {                                                    // Numbers less than ten
890     display.print('0');                                                 // Print '0' before number value
891     display.print(number);
892   } else {
893     display.print(number);                                              // Print the number without '0'
894   }
895 }
896
897
898
```

(w) *Parte 23*

Figura 65: Sezione subroutines con i programmi necessari al funzionamento della serra idroponica.

- Funzione "oled_print_values":
  Riceve: puntatori alle variabili "temperature", "humidity", "brightness", "temperature_max", "temperature_min", "humidity_max", "humidity_min", "hours_of_lights", "parameter", "state_of_plants", "current_time", "error_dht", "error_photoresistor", "error_rtc", "fan", "baffle_position", "lights", "lights_on_time", "lights_off_time", "pump", "page".
  Restituisce: void.
  Descrizione: Funzione per la stampa dei parametri del programma sullo schermo OLED. La funzione fa uso delle librerie $< Adafruit\_GFX.h >$ e $< Adafruit\_SSD1306.h >$ per la stampa dei valori. Ponendo a "0" la variabile *DEBUG*, all'inizio del programma, è possibile deviare la stampa dal terminale al display OLED, usato come dispositivo di default per la rappresentazione delle informazioni. Anche questa funzione è molto lunga, ma come per "serial_print_values", si tratta di semplici comandi ripetuti per eseguire la stampa delle informazioni. Per prima cosa viene pulito il *buffer* del display per poter poi rappresentare le nuove informazioni aggiornate (riga 707). Nella prima riga troviamo la stampa della data e dell'ora ed eventualmente il messaggio di errore collegato, tramite i cicli *if* e *else* dedicati (righe 710 e 723). Viene in seguito eseguito un controllo della pagina selezionata attraverso il ciclo *switch-case* (riga 730). Per la prima pagina (riga 732), nella seconda e terza riga vengono visualizzate temperatura ed umidità rilevate o eventualmente il messaggio di errore, attraverso i cicli *if* e *else* de-

dicati (righe 735 e 750), se la rilevazione è fallita. Nella quarta la luminosità o il messaggio di errore ad esso collegata, sempre attraverso i cicli *if* e *else* dedicati (righe 766 e 771). Nella quinta riga lo stato delle piante contenuto nella ricetta selezionata in quel momento. In seguito, grazie a delle funzioni grafiche contenute nelle librerie citate in precedenza, vengono visualizzate delle linee per formare dei riquadri così da visualizzarne all'interno i simboli che rappresentano la ventola, l'illuminazione della serra, la pompa e la posizione dei deflettori, a seconda dello stato dei componenti. Se la pagina selezionata invece è la seconda (riga 811), viene visualizzato a display, nella seconda riga, lo stato delle piante collegato alla ricetta selezionata. Nella terza riga invece i valori massimi e minimi di temperatura previsti per quello stato di crescita e analogamente, nella quarta, quelli riguardanti l'umidità. Nella quinta riga la luminosità minima e nella sesta invece i valori dell'orario per l'accensione e lo spegnimento delle luci dell'illuminazione della serra. In caso di valore errato della pagina (riga 862), viene visualizzato a display un messaggio di errore. Anche in questa sezione di codice, solo a scopo grafico, si fa utilizzo di una funzione ulteriore per anteporre uno "0" ai numeri minori di 10 (*figura 65w*). L'ultima operazione è quella dell'invio a display dei valori, attraverso la funzione dedicata (riga 868).

5.2.8 Funzione SETUP

Come già descritto, la funzione *SETUP* permette di impostare inizialmente la scheda Arduino e i relativi dispositivi collegati. Attraverso opportune righe di codice si configurano gli ingressi e le uscite, lo stato logico dei componenti e il collegamento di questi con la scheda. In *figura 66*, viene riportata la seguente funzione.

```
899 /* ------------------------------------  Setup  ------------------------------------ */
900
901 void setup()
902 {
903   if (DEBUG)                                                                    // DEBUG mode is activated
904     Serial.begin(9600);                                                         // Start serial communication
905
906   // I2C communication
907   Wire.begin();                                                                 // Start Wire library (I2C communication)
908
909   // OLED display
910   if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {                    // Control the correct connection with OLED display: SSD1306_SWITCHCAPVCC = generate display vo
911     Serial.println(F("Error: SSD1306 allocation failed! NOT connected!"));
912     Serial.flush();
913     exit(1);
914   } else {
915     display.setTextSize(1);                                                     // Text height
916     display.setTextColor(WHITE);                                                // Text color
917     display.cp437(true);                                                        // Text font
918     display.clearDisplay();                                                     // Clear the display
919   }
920
921   // RTC clock
922   rtc.begin();                                                                  // Start the RTC library code
923   if (!rtc.begin()) {                                                           // RTC library code incorrect started (rtc_name.begin() is a bool function)
924       Serial.println(F("Couldn't find RTC: Error!"));
925       Serial.flush();
926       exit(1);                                                                  // Exit from program with error
927   }
928   if (!rtc.isrunning()) {                                                       // RTC is not running
929     /*************** stampare qualcosa ? *****************/
930     rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));                             // Set automatically date parameters from pc at compilation project time
931     /*
932      * rtc.adjust(DateTime(2022, 02, 17, 14, 43, 58));                            // Set manually date parameters inserting: (DateTime(Year, Month, Day, Hour, Minute, Second)
933      */
```

(a) *Parte 1*

- righe 903 e 904:
  Queste righe di codice servono per attivare o meno la comunicazione seriale, a seconda dello stato della variabile *DEBUG*. Infatti, si procede ad attivare questo tipo di comunicazione solo se la variabile è posta sul valore logico *true*: in questo caso si impone come terminale di stampa delle informazioni quello seriale, escludendo lo schermo OLED.

- riga 907:
  Questa riga serve per avviare la comunicazione I2C tra i dispositivi collegati che la richiedono e Arduino.

- righe da 910 a 919:
  Queste righe servono per impostare il display OLED collegato ad Arduino. Fanno uso delle variabili definite all'inizio del programma per l'impostazione di questo. Viene inoltre eseguito un test di corretto funzionamento del dispositivo che, in caso di esito negativo, porta il programma ad interrompersi e a uscire dall'esecuzione. Solo in caso di esito positivo viene terminata l'impostazione del display.

- righe da 922 a 934:
  Queste righe servono per collegare il Clock RTC ad Arduino. Per prima cosa si avvia la comunicazione con il Clock e si verifica la corretta inizializzazione. In caso di esito positivo si prosegue con l'esecuzione della funzione *SETUP*, altrimenti si segnala che il Clock non è correttamente collegato e il programma viene interrotto nello stato di errore. Inoltre, se il collegamento è corretto ma non sono impostati data e ora, in modo automatico viene acquisito l'orario del dispositivo collegato in quel momento con Arduino attraverso il collegamento seriale, e caricato nel Clock. Eventualmente è possibile impostare questi ultimi valori anche manualmente.

```
934  }
935
936  // DHT sensor
937  dht.begin();                                                                    // Start the DHT library code
938
939  // SERVO motor
940  baffle.attach(baffle_pin);                                                      // Declaration of the pin to which the SERVO is connected
941
942
943  // Pins configuration
944  pinMode(dht_pin, INPUT);                                                        // DHT sensor pin configuration on INPUT value
945  pinMode(baffle_pin, OUTPUT);                                                    // SERVO motor pin configuration on OUTPUT value
946  pinMode(pump_pin, OUTPUT);                                                      // JQC3F-5VDC-C relay pin (pump) on OUTPUT value
947  pinMode(lights_pin, OUTPUT);                                                    // JQC3F-5VDC-C relay pin (lights) on OUTPUT value
948  pinMode(fan_pin, OUTPUT);                                                       // JQC3F-5VDC-C relay pin (fan) on OUTPUT value
949  pinMode(button_pin_1, INPUT);                                                   // Button pin configuration on INPUT value
950  pinMode(button_pin_2, INPUT);                                                   // Button pin configuration on INPUT value
951  pinMode(warning_buzzer, OUTPUT);                                                // Buzzer pin configuration on OUTPUT value
952  pinMode(warning_led, OUTPUT);                                                   // Led pin configuration on OUTPUT value
953
954
955  // Initial values
956  digitalWrite(pump_pin, LOW);                                                    // Set initial value of pump relay pin
957  digitalWrite(lights_pin, LOW);                                                  // Set initial value of illumination led relay pin
958  digitalWrite(fan_pin, LOW);                                                     // Set initial value of fan relay pin
959  digitalWrite(warning_buzzer, LOW);                                              // Set initial value of buzzer pin
960  digitalWrite(warning_led, LOW);                                                 // Set initial value of led pin
961
962
963
964
965  // Call function for selection initial greenhouse parameters (at the start parameter = 0)
966  change_parameters(&parameter, &temperature_max, &temperature_min, &humidity_max, &humidity_min, &hours_of_light);
967  baffle.write(BAFFLE_MIDDLE);                                                    // Set initial position of SERVO
968
```

(b) *Parte 2*

- riga 937:
  Questa riga serve per avviare la comunicazione con il sensore *DHT11* e collegarlo alla scheda Arduino.

- riga 940:
  Questa riga serve per inizializzare il servomotore e collegarlo ad Arduino, definendo il pin di collegamento della scheda.

- righe da 944 a 952:
  Queste righe servono per definire lo stato dei pin della scheda: definiscono quali di questi sono di entrata (*INPUT*) e quali di uscita (*OUTPUT*).

- righe da 956 a 960:
  Queste righe servono per impostare lo stato logico iniziale dei pin di uscita, e dunque dei componenti collegati.

- riga 966:
  Questa riga serve per richiamare la funzione per il cambio dei parametri: in questo modo è possibile caricare i valori della ricetta alla prima accensione della scheda, prima di procedere ad eseguire la funzione *LOOP*. Questa operazione è importante in quanto permette il funzionamento del programma che viene eseguito in seguito.

- riga 967:
  Questa riga serve per posizionare i deflettori nella posizione intermedia:

questa sarà la posizione iniziale che avranno all'inizio della funzione *LOOP*.

```
969   // DEBUG mode active: displays the message on the screen
970   if (DEBUG) {
971     display.clearDisplay();
972     display.setCursor(35, 30);
973     display.print(F("DEBUG MODE"));
974     display.display();
975   }
976 }
977
978
979
```

(c) *Parte 3*

Figura 66: Sezione della funzione *SETUP* della scheda Arduino e dei componenti.

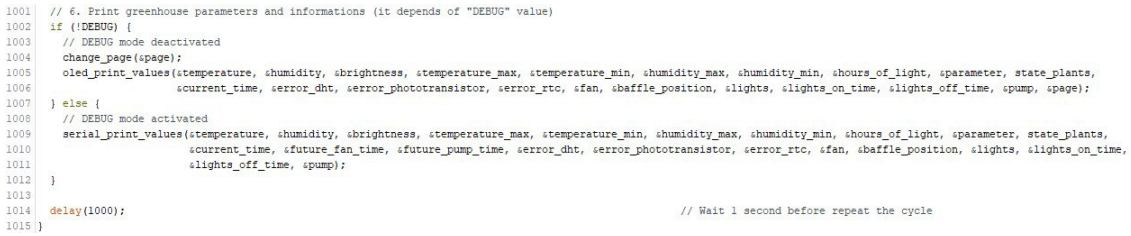
- righe da 970 a 975:
  Queste righe servono per visualizzare nello schermo Display OLED le informazioni che indicano l'attivazione della modalità di *DEBUG*.

5.2.9 Funzione LOOP

Come già descritto, la funzione *LOOP* del programma permette di ripetere l'esecuzione ciclica e continua delle funzioni implementate. E' una funzione molto semplice in quanto si trovano solo le chiamate alla funzioni implementate in precedenza nella sezione delle *subroutines*. In *figura 67* la parte corrispondente del programma.

```
980 /* ---------------------------------------  Loop  --------------------------------------- */
981
982 void loop()
983 {
984   // 1. Reading Date and Time values
985   read_date_time(&current_time);
986
987   // 2. Reading greenhouse parameters (temperature, humidity and brightness)
988   read_temp_hum(&temperature, &humidity);
989   read_brightness(&brightness);
990
991   // 3. Change greenhouse parameters
992   change_parameters(&parameter, &temperature_max, &temperature_min, &humidity_max, &humidity_min, &hours_of_light);
993
994   // 4. Error handling
995   error_handling(&temperature, &humidity, &brightness, &error_dht, &error_phototransistor, &error_rtc);
996
997   // 5. Greenhouse management
998   greenhouse_management(&current_time, &future_fan_time, &future_pump_time, &temperature, &humidity, &brightness, &temperature_max, &temperature_min, &humidity_max,
999                         &humidity_min, &hours_of_light, &fan, &timed_fan, &baffle_position, &lights, &lights_on_time, &lights_off_time, &pump, &timed_pump);
1000
```

(a) *Parte 1*



(b) *Parte 2*

Figura 67: Sezione della funzione *LOOP* della scheda Arduino e dei componenti.

- riga 985:
  Chiamata alla funzione per la lettura della data e dell'orario.
- riga 988:
  Chiamata alla funzione per la lettura della temperatura e dell'umidità.
- riga 989:
  Chiamata alla funzione per la lettura della luminosità esterna.
- riga 992:
  Chiamata alla funzione per il cambiamento della ricetta e dei parametri relativi allo stato di crescita delle piante.
- riga 995:
  Chiamata alla funzione per il controllo e la gestione degli errori legati ai componenti collegati.
- riga 998:
  Chiamata alla funzione per la gestione automatica della serra idroponica.
- righe da 1002 a 1012:
  Gestione della modalità *DEBUG*. Attraverso i cicli *if* e *else* (righe 998 e 1003) si definiscono i casi: se la variabile *DEBUG* è posta nello stato *false* si richiamano le funzioni per il cambio della pagina visualizzata nello schermo OLED e quella per la stampa dei dati verso questo altrimenti, se è posta nello stato *true*, si devia la stampa delle informazioni sul terminale seriale.
- riga 1014:
  Attesa di 1 secondo prima della nuova esecuzione della funzione *LOOP* dovuto, come già introdotto, al tempo di *refresh* del sensore *DHT11* che vincola il tempo ciclo.

5.3 Programmi di testing dei componenti

Il programma sviluppato per la serra permette una modalità di *DEBUG*, con stampa dei valori sul terminale seriale. Le informazioni che vengono fornite non sono le stesse illustrate sullo schermo OLED, ma sono molto più dettagliate. Dato che molto spesso è difficile identificare dei problemi all'interno di codici così complessi, sono stati sviluppati anche dei programmi di *testing* dei singoli componenti, in modo da verificare il singolo funzionamento di questi. Le librerie, i metodi di configurazione dei dispositivi e i pin di collegamento sono analoghi a quelli usati nel programma principale, in modo da potersi collegare direttamente senza effettuare delle variazioni.

5.3.1 Test schermo

Lo schermo OLED prevedeve un programma di test che verifica la connessione tra questo e la scheda Arduino. Vengono definite le variabili necessarie per la configurazione dello schermo e vengono stampati dei messaggi a seconda della corretta, o meno, configurazione di quest'ultimo. Se l'esito del test eseguito è positivo, viene visualizzato un messaggio sullo schermo stesso altrimenti, in caso di errori, il messaggio di avvertimento viene visualizzato sul terminale seriale collegato alla scheda. Di seguito, in *figura 68*, il programma di *testing*.

Test_SSD1306

```
/*
 * Name: Test_SSD1306
 * Version: 1.0
 * Description: Library testing OLED SSD1306 Display
 * Component: Adafruit OLED 128x64 SSD1306 I2C (4 pin: GND [GND], VCC [3.3V], SCL [A5] and SDA [A4])
 *
 * Author: Filippo Tasca, 08/02/2022
 */

// Libraries
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>


// Defines
#define SCREEN_WIDTH 128                // OLED display width, in pixels
#define SCREEN_HEIGHT 64                // OLED display height, in pixels
#define OLED_RESET -1                   // -1: sharing Arduino reset pin, 4: reset pin (example: # 4)
#define SCREEN_ADDRESS 0x3C             // 0x3C: for 128x64 display, 0x3D: for 128x32 display


// Settings
Adafruit_SSD1306 OLED(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);        // Setting display OLED


// Setup
void setup()
{
  Serial.begin(9600);

  Serial.println(F("Test OLED DISPLAY:"));
  delay(1500);

  // OLED SSD1306 Display correct connectioning test
  if (!OLED.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
```

(a) *Parte 1*

```
    Serial.println(F("Error: SSD1306 NOT connected!"));
    Serial.flush();
    exit(1);
  } else {
    Serial.println(F("SSD1306 correctly connected!"));
    Serial.flush();
    OLED.setTextSize(1);
    OLED.setTextColor(WHITE);
    OLED.cp437(true);
    OLED.clearDisplay();

    // Launch OLED SSD1306 Display test
    test_ssd1306();
  }

  // Exit program (stop)
  exit(0);
}


// Loop
void loop()
{

}


// Program test display OLED SSD1306 Display
void test_ssd1306()
{
  // Print informations on display
  OLED.setCursor(0,0);
  OLED.print(F("OLED SSD1306, ok!"));
  OLED.display();
}
```

(b) *Parte 2*

Figura 68: Programma di *testing* dello schermo OLED collegato.

5.3.2 Test sensore di temperatura ed umidità

Il programma per il test del sensore di temperatura ed umidità prevede l'esecuzione di semplici funzioni: la lettura dei parametri e la verifica del corretto collegamento analizzando ed elaborando i valori raccolti. La misurazione dei parametri esterni viene eseguita più volte per questioni di affidabilità. Se tutti i valori misurati sono corretti, allora il sensore è collegato correttamente. Alla fine del programma, il *testing* viene concluso e il programma arrestato. In *figura 69*, il codice implementato.

Test_DHT11

```
/*
 * Name: Test_DHT11
 * Version: 1.0
 * Description: Library testing DHT11 temperature/umidity sensor
 * Component: DHT11 sensor (3 pin: GND [GND], DATA [2] and VCC [5V])
 *
 * Author: Filippo Tasca, 08/02/2022
 */

// Libraries
#include <DHT.h>


// Defines
#define DHT_TYPE DHT11                    // Define DHT sensor type (DHT 11, DHT 22 or DHT 21)
#define DHT_PIN 2                         // PIN that connect the data signal from DHT sensor


// Settings
DHT dht(DHT_PIN, DHT_TYPE);               // DHT sensor initialize, with "dht" name

float temperature, humidity;              // float variables for temperature and humidity detected


// Setup
void setup()
{
  Serial.begin(9600);

  Serial.println(F("Test DHT11:"));
  delay(1500);

  dht.begin();                            // Initialize DHT11

```

(a) *Parte 1*

```
  // Reading initial parameters
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();

  // DHT11 correct connectioning test
  if (!isnan(temperature) && !isnan(humidity)) {
    Serial.println(F("DHT11 correctly connected!"));
    Serial.flush();
    delay(1500);
  } else {
    Serial.println(F("Error: DHT11 NOT connected!"));
    Serial.flush();
    exit(1);
  }
}


// Loop
void loop()
{
  for (int i = 0; i < 5; i++) {
    // Launch DHT11 sensor test
    dht_measurements_test();

    delay(2000);


    // Exit program (stop)
    if (i == 4)
      exit(0);
  }
}

```

(b) *Parte 2*

```
// Program test DHT11 sensor
void dht_measurements_test()
{
  // Reading parameters
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();

  // Print informations
  Serial.print(F("DETECTED DATA: \t"));
  Serial.print(F("Temperature: "));
  if (!isnan(temperature)) {
    Serial.print(temperature);
  } else {
    Serial.print(F("Error!"));
  }
  Serial.print(F("\t"));
  Serial.print(F("Humidity: "));
  if (!isnan(humidity)) {
    Serial.println(humidity);
  } else {
    Serial.println(F("Error!"));
  }
  Serial.flush();
}
```

(c) *Parte 3*

Figura 69: Programma di *testing* del sensore di temperatura ed umidità.

5.3.3 Test Real Time Clock

Per il test del Real Time Clock, collegato alla scheda Arduino, si procede nel seguente modo: verifica del corretto collegamento e funzionamento e successiva stampa dei valori letti da quest'ultimo. Se tutto è configurato correttamente, si dovrebbe visualizzare a terminale seriale una serie di orari e date scritte in modo sequenziale, dalla meno recente a quella attuale. Di seguito, in *figura 70*, il codice di *testing* del dispositivo.

Test_DS1307

```
/*
 * Name: Test_DS1307
 * Version: 1.0
 * Description: Library testing RTC DS1307
 * Component: RTC DS1307 clock (4 pin: GND [GND], SCL [A4], SDA [A5] and VCC [5V])
 *
 * Author: Filippo Tasca, 09/02/2022
 */

// Libraries
#include <RTClib.h>


// Settings
RTC_DS1307 rtc;                                  // Setting RTC DS1307 program name


// Setup
void setup()
{
  Serial.begin(9600);

  Serial.println(F("Test RTC:"));
  delay(1500);

  // RTC DS1307 correct connectioning test
  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    exit(1);
  }
```

(a) *Parte 1*

```
  // RTC DS1307 correct functioning test
  if (!rtc.isrunning()) {
    Serial.println("RTC is NOT running, set the time!");
    Serial.flush();
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));                // Automatic time setting from PC
    // rtc.adjust(DateTime(2022, 2, 9, 12, 26, 33));               // Manual time setting: (year, month, day, hour, minute, second)
  }
}


// Loop
void loop()
{
  for (int i = 0; i < 5; i++) {
    // Launch RTC SD1307 test
    rtc_serial_print();

    delay(1000);

    // Exit program (stop)
    if (i == 4)
      exit(0);
  }
}


// Program test RTC DS1307
void rtc_serial_print()
{
  // Time acquisition
  DateTime now = rtc.now();

```

(b) *Parte 2*

```
  // Print informations
  if (rtc.begin()) {
    Serial.print(now.day(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.year(), DEC);
    Serial.print("\t");
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.println(now.second(), DEC);
  } else {
    Serial.println("Couldn't find RTC");
  }

  Serial.flush();
}
```

(c) *Parte 3*

Figura 70: Programma di *testing* dello schermo OLED collegato.

5.3.4 Test servomotore

Il programma di *testing* del servomotore permette di verificare il collegamento di quest'ultimo andando a ruotare l'albero. Al termine dell'esecuzione di ogni comando di rotazione, viene visualizza a terminale seriale la posizione teorica dell'albero per verificare che la comunicazione sia forte e continua. Di seguito il programma implementato (*figura 71*).

Test_SM-S2309S

```
/*
 * Name: Test_SM-S2309S
 * Version: 1.0
 * Description: Library testing SM-S2309S servo motor
 * Component: SM-S2309S servo motor (3 pin: GND [GND], DATA [3] and VCC [5V])
 *
 * Author: Filippo Tasca, 09/02/2022
 */

// Libraries
#include <Servo.h>


// Define
#define SERVO_PIN 3                             // Servo connection Arduino pin


// Settings
Servo servo;                                    // Setting SM-S2309S servo motor program name

int angle = 0;                                  // Variable for rotation testing


// Setup
void setup()
{
  Serial.begin(9600);

  servo.attach(SERVO_PIN);                      // Setting SM-S2309S servo motor

  Serial.println(F("Test SERVO MOTOR:"));
  delay(1500);

```

(a) *Parte 1*

```
  // Setting initial position SM-S2309S servo motor
  servo.write(angle);
  Serial.print(F("Initial position: "));
  Serial.print(angle);
  Serial.println(F("°"));
  Serial.flush();
}


// Loop
void loop()
{
  // Launch SM-S2309S servo motor test
  move_servo();

  // Print informations
  Serial.println(F("Servo test, complete!"));
  Serial.flush();
  exit(0);
}


// Program test SM-S2309S servo motor
void move_servo()
{
  // Setting angle = 180°
  for (angle = 0; angle <= 180; angle++) {
    servo.write(angle);
    if (angle == 180) {
      // Print informations
      Serial.print(F("Angle: "));
      Serial.print(angle);
      Serial.println(F("°"));
      Serial.flush();
    }
```

(b) *Parte 2*

```

    delay(20);
  }

  // Setting angle = 90°
  for (angle = 180; angle >= 90; angle--) {
    servo.write(angle);
    if (angle == 90) {
      // Print informations
      Serial.print(F("Angle: "));
      Serial.print(angle);
      Serial.println(F("°"));
      Serial.flush();
    }

    delay(20);
  }
}
```

(c) *Parte 3*

Figura 71: Programma di *testing* dello schermo OLED collegato.

5.3.5 Test buzzer

Per quanto riguarda il codice di *testing* del buzzer, questo ha una struttura molto semplice. Di fatto si verifica solo il corretto funzionamento sonoro riportando a seriale i messaggi relativi all'esecuzione del test, per avere un riferimento visivo dello stato del programma. In *figura 72*, il codice implementato.

Test_PKm22EPP-4001-B0

```
/*
 * Name: Test_PKm22EPP-4001-B0
 * Version: 1.0
 * Description: Library testing PKm22EPP-4001-B0 buzzer
 * Component: PKm22EPP-4001-B0 (2 pin: GND [GND] and DATA [12])
 *
 * Author: Filippo Tasca, 13/02/2022
 */

// Define
#define BUZZER_PIN 12


// Setup
void setup()
{
  pinMode(BUZZER_PIN, OUTPUT);                    // Setting Arduino buzzer pin

  Serial.begin(9600);

  // Print informations
  Serial.println(F("Test BUZZER:"));
  delay(1500);

  // Launch test buzzer program
  test_tone();
  delay(1000);

  // Print informations
  Serial.println(F("Test completed!"));
  Serial.flush();

  // Exit program (stop)
  exit(0);
}

```

(a) *Parte 1*

```

// Loop
void loop()
{

}


// Program for testing HW5P-1 buzzer
void test_tone()
{
  // Commands for tone buzzer ("tone(buzzer_pin, frequency, time)")
  tone(BUZZER_PIN, 700, 500);
  delay(700);
  tone(BUZZER_PIN, 1000, 500);
  delay(700);
  tone(BUZZER_PIN, 1500, 300);
}
```

(b) *Parte 1*

Figura 72: Programma di *testing* del buzzer collegato.

5.3.6 Test relay

La serra richiede tre relay per gestire rispettivamente ventola, pompa e illuminazione led. Il test permette di verificare il corretto collegamento e funzionamento di entrambi. Per prima cosa vengono configurati i pin, e lo stato logico di questi, dei dispositivi alla scheda: solo in seguito viene lanciato il programma di test. Viene commutato lo stato dei relays riportando ogni cambiamento a terminale seriale. Di seguito, in *figura 73*, il codice implementato.

Test_JQC3F-5VDC-C

```
/*
 * Name: Test_JQC3F-5VDC-C
 * Version: 1.0
 * Description: Library testing JQC3F-5VDC-C relay
 * Component: JQC3F-5VDC-C (x3) (3 pin: GND [GND], VCC [5V], DATA [4] (pomp), DATA [7] (lights) and DATA [8] (fan))
 *
 * Author: Filippo Tasca, 14/02/2022
 */

// Defines
#define POMP_PIN 4                                      // Arduino pin connected to the pomp
#define LIGHTS_PIN 7                                    // Arduino pin connected to the lights
#define FAN_PIN 8                                       // Arduino pin connected to the fan


// Setup
void setup()
{
  Serial.begin(9600);

  // Settings
  pinMode(POMP_PIN, OUTPUT);
  pinMode(LIGHTS_PIN, OUTPUT);
  pinMode(FAN_PIN, OUTPUT);

  // Initial states
  digitalWrite(POMP_PIN, LOW);
  digitalWrite(LIGHTS_PIN, LOW);
  digitalWrite(FAN_PIN, LOW);

  // Print informations
  Serial.println(F("Test RELAYS:"));
  delay(1500);

```

(a) *Parte 1*

```
  // Launch programs testing relays
  test_relays();
  delay(1000);

  setting_relays();
  delay(1000);

  test_relays();
  delay(1000);

  setting_relays();
  delay(1000);

  test_relays();
  delay(1000);

  Serial.println(F("Test completed!"));
  Serial.flush();

  // Exit program (stop)
  exit(0);
}

// Loop
void loop()
{

}
```

(b) *Parte 2*

```
// Program for test relays connected to Arduino pin
void test_relays()
{
  // Print informations
  Serial.print(F("POMP state: "));
  Serial.println(digitalRead(POMP_PIN));
  delay(1000);
  Serial.print(F("LIGHTS state: "));
  Serial.println(digitalRead(LIGHTS_PIN));
  delay(1000);
  Serial.print(F("FAN state: "));
  Serial.println(digitalRead(FAN_PIN));
  Serial.flush();
  delay(1000);
}


// Program that change states of relays
void setting_relays()
{
  // Print informations
  Serial.println(F("Changing relays states..."));
  Serial.flush();

  // Changing relays states
  digitalWrite(POMP_PIN, !digitalRead(POMP_PIN));

  digitalWrite(LIGHTS_PIN, !digitalRead(LIGHTS_PIN));

  digitalWrite(FAN_PIN, !digitalRead(FAN_PIN));
}
```

(c) *Parte 3*

Figura 73: Programma di *testing* dei relays collegati.

5.3.7 Test del fototransistor

Il codice di *testing* del fototransistor collegato permette di verificare se i valori rilevato sono corretti. Per prima cosa viene definito il pin di collegamento con la scheda Arduino e solo in seguito vengono acquisiti i valori attraverso delle misurazioni. Si esegue una media e si verifica che il risultato sia all'interno dell'intervallo coretto. A seconda dell'esito, viene visualizzato il valore medio ottenuto oppure un messaggio di errore. In *figura 74* il codice sviluppato.

Test_HW5P-1

```
/*
 * Name: Test_HW5P-1
 * Version: 1.0
 * Description: Library testing HW5P-1 phototransistor
 * Component: HW5P-1 (2 pin: DATA [A0] (with 10 Kohm resistance connected to GND), VCC [5V])
 *
 * Author: Filippo Tasca, 14/02/2022
 */

// Define
#define PHOTOTRANSISTOR_PIN A0                    // Arduino pin connected to phototransistor


// Settings
int brightness = 0;                               // Variable to detect brightness


// Setup
void setup()
{
  Serial.begin(9600);

  Serial.println(F("Test PHOTOTRANSISTOR:"));
  delay(1500);
}


// Loop
void loop()
{
  for (int i = 0; i < 5; i++) {
    // Launch brightness test
    test_brightness();
    delay(1500);

```

(a) *Parte 1*

```
37      // Exit program (stop)
38      if (i == 4)
39        exit(0);
40    }
41  }
42
43
44  // Program tests correct brightness reading
45  void test_brightness()
46  {
47    // Reading paramater
48    for (int i = 0; i < 5; i++) {
49      brightness += analogRead(PHOTOTRANSISTOR_PIN);                // Reading value of brightness and accumulation
50    }
51    brightness = brightness / 5;                                    // Average of values
52    brightness = map(brightness, 0, 1023, 0, 100);                  // Change range of value [0, 1023 --> 0, 100]
53
54    // Print informations
55    if (brightness < 0 || brightness > 100) {
56      Serial.println(F("Wrong brightness value!"));
57    } else {
58      Serial.print(F("Brightness value: "));
59      Serial.println(brightness);
60    }
61  }
```

(b) *Parte 1*

Figura 74: Programma di *testing* del fototransistor collegato.

5.3.8 Test led errore

Ultimo test, ma non certamente per importanza, è quello riguardante il led di errore collegato alla scheda Arduino. Permette solo di verificare il corretto collegamento e funzionamento del led attraverso un controllo visivo. Inoltre, vengono riportati a terminale seriale messaggi riguardanti lo stato di esecuzione del programma. In *figura 75*, il codice.

Test_LED

```
/*
 * Name: Test_LED
 * Version: 1.0
 * Description: Library testing LED for incorrect program value
 * Component: LED (2 pin: GND [GND] and VCC [13])
 *
 * Author: Filippo Tasca, 14/02/2022
 */

// Define
#define LED_PIN 13                          // Arduino pin connected to the led


// Setup
void setup()
{
  Serial.begin(9600);


  // Setting
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);

  // Print informations
  Serial.println(F("Test LED:"));
  delay(1500);

  // Launch program that test led
  test_led();

  // Exit program (stop)
  exit(0);
}


```

(a) *Parte 1*

```
// Loop
void loop()
{

}


// Program for test led
void test_led()
{
  // Changing led states
  digitalWrite(LED_PIN, HIGH);
  delay(2000);
  digitalWrite(LED_PIN, LOW);
  delay(1000);

  // Print informations
  Serial.println(F("Test completed!"));
  Serial.flush();
}
```

(b) *Parte 1*

Figura 75: Programma di *testing* del led di errore collegato.

5.4 Controllo del programma automatico

I programmi di testing sviluppati hanno lo scopo di verificare il corretto funzionamento dei singoli dispositivi collegati. Oltre a questo aspetto, è importante verificare anche che il programma segua l'ordine di esecuzione delle funzioni. Il codice implementato ha molte variabili da controllare e alcune di queste, a causa delle limitazioni legate al programma di Arduino, non sono rappresentabili. L'attenzione è stata indirizzata quindi ai parametri ambientali di ingresso, alle relative variabili collegate e alla gestione delle uscite. In *figura 76* viene riportato l'andamento degli ingressi (dalla prima alla terza riga dal basso) e delle relative variabili associate, e lo stato delle uscite (dalla quarta alla sesta riga dal basso). In particolare, troviamo i seguenti segnali:

- temperature (*verde*);
- temperature_min (*arancione*);
- temperature_max (*azzurro*);
- humidity (*giallo*);
- humidity_min (*grigio chiaro*);
- humidity_max (*rosa*);
- brightness (*rosso*);
- BRIGHTNESS_MIN (*blu*);
- fan (*grigio*);
- lights (*viola*);
- pump (*azzurro cielo*).

Osservando il grafico di *figura 76*, è possibile osservare la commutazione delle uscite a seconda del valore dei segnali di ingresso e delle variabili collegate gestite dal programma. Osservando i segnali, è possibile notare come questi partano dal valore 0 nell'instante di avvio del programma. Per come è stato implementato il codice, questo andamento è corretto in quanto le variabili nel programma vengono dichiarate ed inizializzate al valore 0. Questo vale per tutti i segnali e le variabili eccetto per la variabile *BRIGHTNESS_MIN*, che invece viene fin da subito inizializzata al valore di 20. Partendo dalla riga più in basso, è possibile osservare come il segnale della temperatura si aggiri all'interno del range definito dalle variabili *temperature_min* e *temperature_max*. Durante la transizione del segnale all'interno dell'intervallo, lo stato della ventola viene mantenuto basso (quarta riga). Nel momento

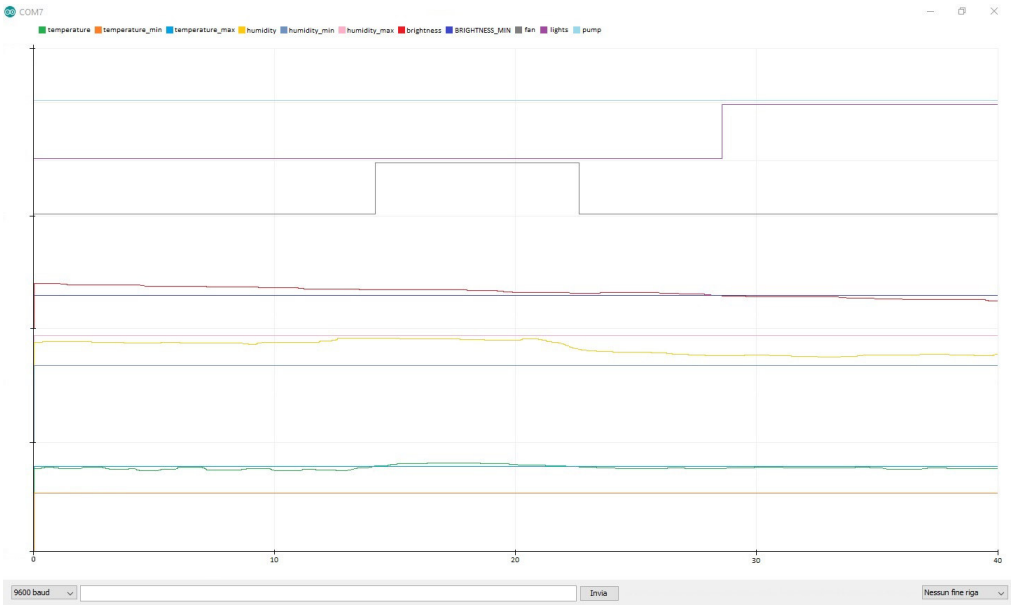
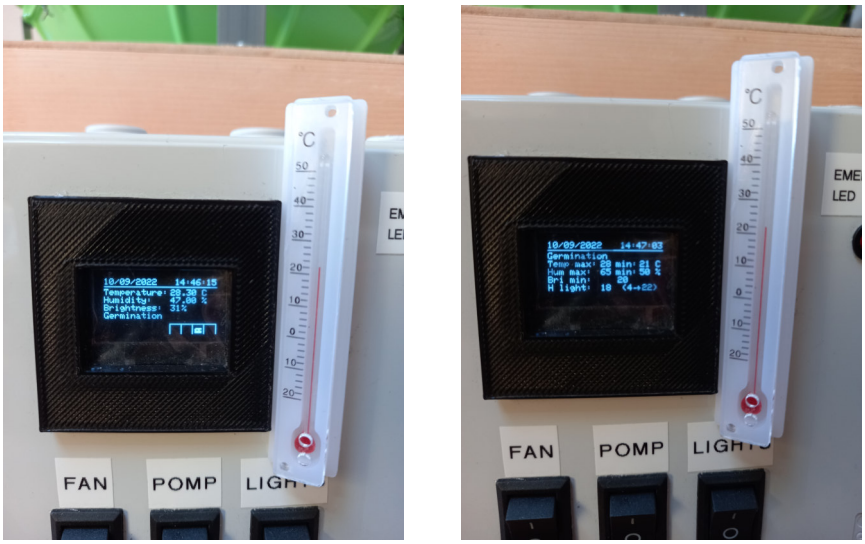


Figura 76: Grafico dei principali segnali utilizzati nel programma, nel periodo di tempo tra l'accensione e i primi 40 sec. di esecuzione.

in cui il valore di temperatura supera la soglia impostata, come avviene a circa 14 sec., lo stato della ventola viene commutato e mantenuto fino a quando il segnale non ritorna all'interno del range corretto, circa a 22 sec., per poi essere mantenuto nuovamente in quello stato. Anche l'umidità, analogamente alla temperatura, può controllare la commutazione della ventola ma in questo caso rimane sempre dentro il range definito dalle variabili *humidity_min* e *humidity_max* (seconda riga). Altro parametro ambientale riportato è quello della luminosità (terza riga). Analogamente a quanto succede per la temperatura, in questo caso un valore inferiore a quello minimo, definito dalla variabile *BRIGHTNESS_MIN*, provoca la commutazione dello stato del pin che controlla l'illuminazione della serra, come è possibile notare (quinta riga). L'ultimo segnale presente invece (sesta riga), non commuta mai. Potrebbe sembrare errato ma in realtà l'andamento è corretto in quanto, il ciclo di irrigazione delle piante, è temporizzato e viene azzerato ad ogni accensione della scheda. Per notare una commutazione di questo segnale si dovrà quindi aspettare un tempo molto più lungo, a seconda del valore impostato nel programma. Per la visualizzazione dei dati sono stati utilizzati anche il display OLED e il terminale seriale del programma di Arduino. In particolare, il primo permette di osservare i soli dati utili all'utente finale, mentre per il secondo ulteriori dati a scopo di *debugging*. In *figura 77* vengono riportate le schermate del display, con visualizzati i dati misurati.

Nell'immagine in *figura 77a* è possibile visualizzare la schermata principale dello schermo OLED installato nel pannello della serra. Di fianco



(a) *Pagina principale.* (b) *Pagina secondaria.*

Figura 77: Schermate visualizzate nel display OLED installato.

viene riportato un termometro ad alcool, con risoluzione bassa, ma utile per l'analisi. La temperatura rilevata dal dispositivo analogico si aggira attorno ai 22°C, esterni alla serra, mentre la temperatura interna è di circa 28°C. E' bene chiarire che la differenza evidenziata non è una casualità, ma dovuta al fatto che all'interno della cupola trasparente si creano umidità e calore, trattandosi di un ambiente chiuso e compatto. I raggi del sole che entrano portano ad un riscaldamento dell'aria e ad una evaporazione molto lenta, e in minima parte, dell'acqua contenuta nei cubetti di lana di roccia. Nella pagina principale sono riportati i dati sui parametri ambientali rilevati dai sensori (prime 3 righe) oltre a data ed ora. I valori visualizzati nella schermata secondaria, in *figura 77b*, fanno riferimento alla fase di crescita selezionata (nella foto *"Germination"*), e rappresentata sia nella pagina secondaria (prima riga) sia in quella principale (quarta riga dei dati). Nella pagina principale, in basso a destra, è riportato inoltre un piccolo riquadro con delle immagini personalizzate. Questo spazio dello schermo serve per rappresentare lo stato delle uscite, in particolare quelle della ventola, della pompa, dell'illuminazione e la posizione dei deflettori. Osservando entrambe le immagini è possibile notare il corretto funzionamento del programma. I valori massimi e minimi di temperatura ed umidità sono riportati nella pagina secondaria dello schermo, in *figura 77b*. La temperatura rilevata eccede quella massima impostata mentre per l'umidità, al contrario, è al di sotto della soglia minima. In questo caso, il programma dovrebbe mantenere spenta la ventola e chiudere i deflettori, come rappresentato nel riquadro dello schermo in *figura 77a*. Osservando ora la luminosità, si può notare

come la soglia minima sia impostata al 20%. Quella rilevata è superiore e dunque l'illuminazione della serra viene mantenuta spenta (simbolo dell'illuminazione assente). Attivando la modalità di *DEBUG* del programma, è possibile deviare la stampa dei dati dal display OLED al monitor seriale del PC. In *figura 78* viene riportata la schermata visualizzabile.

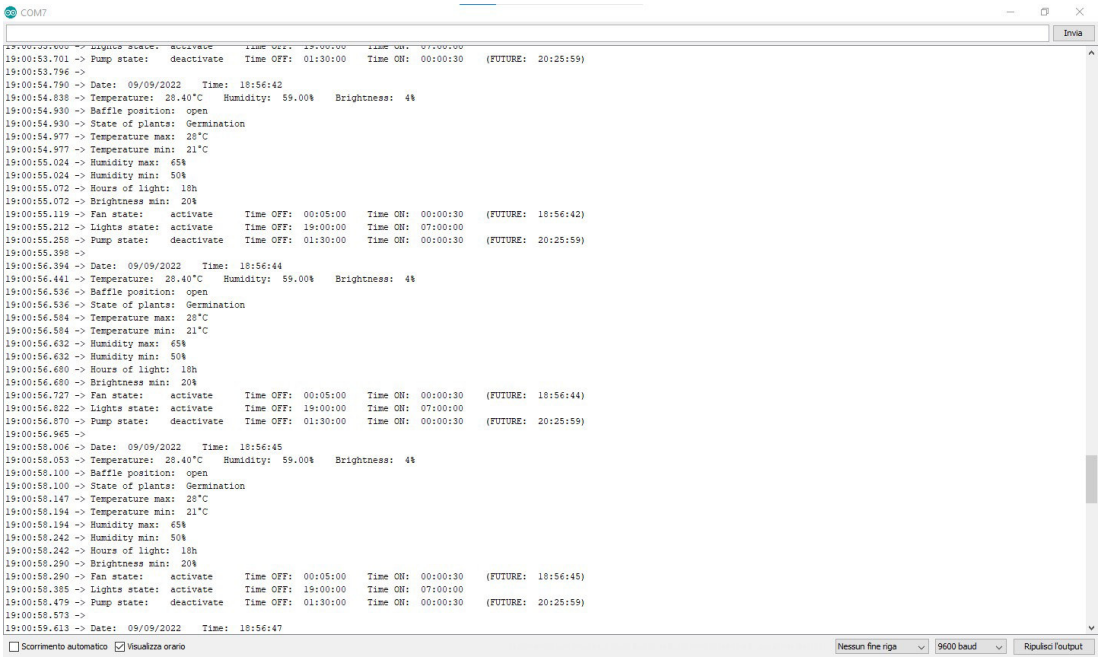


Figura 78: Monitor seriale con riportati i dati rilevati dai sensori e informazioni per il debugging.

Si può notare come le informazioni e i dati riportati in precedenza nel display siano rappresentati ora anche nel monitor seriale, con aggiunte altre informazioni riguardo i tempi di attivazione/disattivazione dei componenti e il tempo futuro di commutazione dello stato dei dispositivi. L'immagine riportata presenta il rilevamento di parametri corretti da parte dei dispositivi. Nel caso in cui, ad esempio, non siano rilevati correttamente i valori di temperatura ed umidità (scollegando il cavo di collegamento tra la scheda Arduino e il sensore *DHT11*), la schermata e le informazioni riportate sono quelle rappresentate in *figura 79*. La lettura errata viene riportata e segnalata, come succede anche per il display, e il programma attiva la modalità di segnalazione degli errori. Questo vale anche per la lettura dei dati dal fototransistor, dal modulo RTC installato, e per molte parti logiche del programma non legate direttamente ai dispositivi esterni.

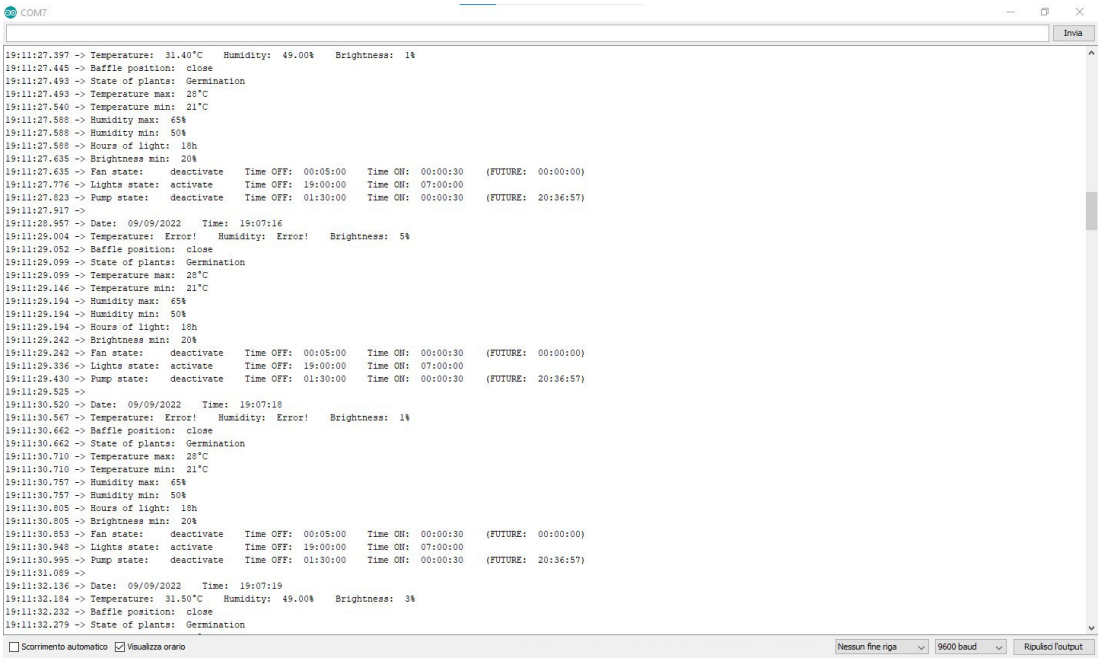


Figura 79: Monitor seriale con riportati gli errori dovuti al malfunzionamento/scollegamento del sensore DHT11.

6 Raccolta e validazione dei dati

Lo scopo finale del progetto, come spiegato più volte, era quello di creare una serra idroponica che potesse introdurre dei benefici, sfruttando i pregi dovuti al tipo di coltivazione e quelli introdotti dall'automatizzazione del processo di controllo e di gestione. Per poter verificare i dati, sicuramente è bene distinguere i due punti appena elencati.

6.0.1 Dati sul metodo di coltivazione idroponico

Per prima cosa è bene parlare del metodo di coltivazione applicato. Come già ampiamente detto, l'uso di questa tecnica introduce dei benefici riguardo il tempo di crescita delle piante e la loro qualità finale. Le caratteristiche che sono possibili evidenziare al termine di un ciclo di coltivazione, sono:

- Crescita più rapida delle piante coltivate in substrato inerte fornendo solo acqua. In un test eseguito irrorando le piante solo con acqua, senza alcuna soluzione nutritiva, è emerso che nelle prime 4 settimane di vita, i semi piantati in cubetti di lana di roccia mostrano uno sviluppo maggiore rispetto a quelli in terreno comune. Con "sviluppo maggiore" facciamo riferimento ad un parametro semplice e facile da misurare per eseguire il confronto: l'altezza della pianta. Il valore riportato in *figura 80* s'intende rispetto alla profondità, in millimetri, dei semi piantati, ed è una media tra le grandezze delle 6 piante coltivate.

Test di crescita con sola acqua

	Altezza [mm]				
Settimana	**Lana di roccia**	**Delta crescita**	**Terreno classico**	**Delta crescita**	**Rapporto di crescita (*)**
1	7	7	5	5	40%
2	13	6	8	3	63%
3	18	5	11	3	64%
4	23	5	14	3	64%
5	26	3	17	3	53%
6	29	3	22	5	32%
7	31	2	28	6	11%
8	33	2	35	7	-6%
9	35	2	43	8	-19%
10	37	2	52	9	-29%
11	39	2	65	13	-40%
12	41	2	77	12	-47%

Test di crescita in serra

	Altezza [mm]		Rapporto di crescita rispetto:	
Settimana	**Lana di roccia**	**Delta crescita**	**Lana di roccia (**)**	**Terreno classico (***)**
1	7	7	0%	40%
2	14	7	8%	75%
3	21	7	17%	91%
4	27	6	17%	93%
5	35	8	35%	106%
6	44	9	52%	100%
7	55	11	77%	96%
8	-	-	-	-
9	-	-	-	-
10	-	-	-	-
11	-	-	-	-
12	-	-	-	-

(*) Rapporto tra la grandezza della pianta in lana di roccia rispetto a quella in terreno classico (con sola acqua).
(**) Rapporto tra la grandezza della pianta in lana di roccia, in serra, rispetto a quella in lana di roccia, con sola acqua.
(***) Rapporto tra la grandezza della pianta in lana di roccia, in serra, rispetto a quella in terreno classico, con sola acqua.

Figura 80: Dati sulla crescita delle piante e sui test eseguiti.

Per maggiore comprensione, si riporta in *figura 81* il solo grafico di crescita.

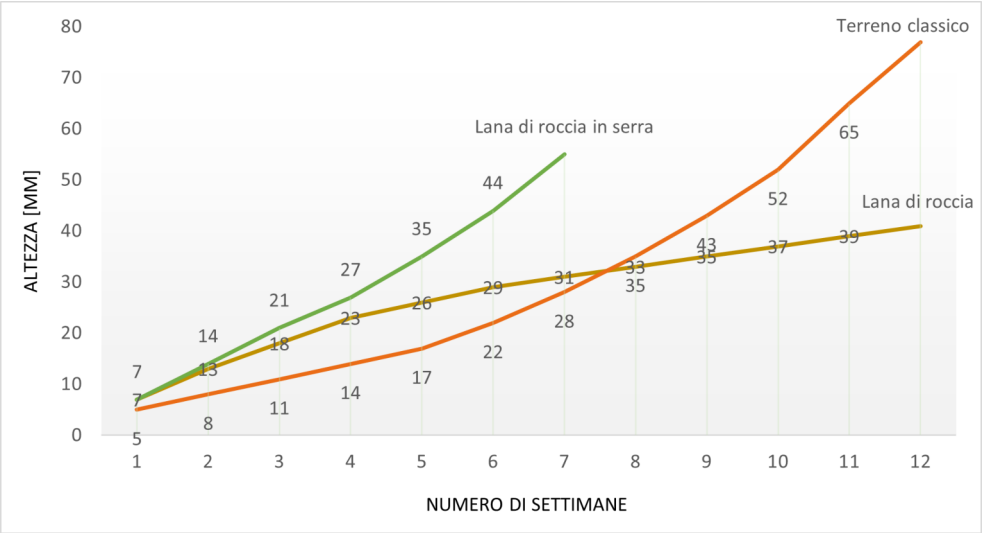


Figura 81: Grafico di crescita delle piante. Si ricordano i colori: marrone = lana di roccia (solo acqua), arancione = terreno classico (solo acqua), verde = lana di roccia (in serra).

Nel terreno classico, i semi sono inseriti ad una profondità di circa 3 mm, analoga alla posizione alla quale si trovano nel cubo. L'altezza riportata tiene conto di questa profondità e non solo del fusto visibile della pianta. Nel grafico riportato in *figura 81*, le linee rappresentative del test sono distinte a seconda del colore: *marrone* per la crescita in lana di roccia e *arancione* per il terreno classico. Dal grafico emerge che circa dopo 1 settimana, o poco più, il germoglio in lana di roccia presenta un'altezza maggiore, rispetto al terreno classico, del 40%. Questo è dovuto al materiale e alle proprietà fisiche del cubo, grazie alla sua porosità. Le radici infatti, in questa prima fase, trovano meno ostacoli nel radicarsi e dunque possono concentrare le energie nella crescita del fusto e della pianta stessa. Fino alla terza settimana abbiamo una crescita costante che arriva a toccare il punto più alto con il 64%. I cubi in lana di roccia rispettano le aspettative ipotizzate e anzi, in questo test, le superano leggermente. Nelle settimane successive invece, avviene un calo drastico. Questo è normale in quanto, il terreno classico, contiene naturalmente gli elementi nutritivi per la pianta che le permettono di svilupparsi in autonomia a differenza della lana di roccia, che è invece neutra. Tra la settima e l'ottava settimana le grandezze delle piante si eguagliano; in seguito, quelle coltivate nei cubi mantengono un trend stabile e leggermente in crescita a differenza

delle altre che presentano un andamento esponenziale. Dopo circa 3 mesi (12 settimane), il distacco è netto: si tratta di un 47% di crescita maggiore per le piante coltivate nel terreno classico. Il fenomeno è del tutto normale in quanto, nel cubo, si fermano solo gli elementi e i sali contenuti nell'acqua usata per irrigare la pianta. Se venisse usata dell'acqua distillata, priva di ogni impurità, probabilmente la pianta non riuscirebbe a svilupparsi e arriverebbe a marcire prima, e alla morte poi.

- Crescita e resa maggiore delle piante coltivate in serra. Il test di somministrazione di sola acqua ha sicuramente messo in luce i benefici introdotti dall'uso di un substrato inerte nella prima fase di crescita della pianta, nella quale il seme contiene tutto l'occorrente per farla germogliare. Da dimostrare in serra invece, rimaneva il calo subito dopo la terza settimana. Analogamente ai dati ottenuti nell'altro test, e le relative linee nel grafico, in *figura 81* è possibile osservare anche l'andamento delle piante coltivate in cubi di lana di roccia con metodo idroponico (linea *verde*). Come già descritto nei paragrafi precedenti, durante i primi giorni di coltivazione ai semi viene somministrata solo acqua, anche se coltivati in serra. Proprio per questo motivo, il trend della linea *verde* e di quella *marrone* si sovrappongono. Dalla prima settimana invece, ovvero dopo l'inizio della somministrazione di soluzione nutritiva, l'andamento sembra essere abbastanza lineare, con una leggera concavità verso l'alto. Il picco di crescita lo si tocca alla settima settimana, con un incremento del 77% rispetto alle piante in lana di roccia e un 96% rispetto quelle in terreno classico, ricordando che a queste ultime veniva somministrata solo acqua. I dati ottenuti sono il segnale che il metodo di coltivazione, abbinato alle scelte concettuali e alla gestione automatizzata, introducono dei significativi benefici. Purtroppo i dati si interrompono alla settima settimana. Questo è dovuto alle dimensioni interne della serra e al relativo spazio di crescita per le piante. Infatti le piante, alla settima settimana hanno raggiunto la massima altezza disponibile e non avevano a disposizione più spazio per svilupparsi e crescere. Probabilmente, se i dati fossero stati rilevati fino alla dodicesima settimana (3 mesi), si sarebbe potuto avere un paragone più netto tra i vari test e osservare un andamento esponenziale più marcato della curva.

7 Conclusioni e possibili implementazioni future

La realizzazione pratica del progetto ha sicuramente permesso di mettere in risalto dei dati che, altrimenti, sarebbero rimasti solo delle ipotesi. Lo scopo del progetto era quello di ideare, progettare e realizzare un prototipo di serra che utilizzasse il metodo di coltivazione idroponica, e che permettesse inoltre di ottenere un controllo e una gestione del processo automatici, attraverso l'uso di un microcontrollore. Unendo questi aspetti, nella teoria era possibile stimare una crescita più veloce delle piante, attorno circa ad un 50%, rispetto ai metodi di coltivazione classici. Inoltre, sarebbero stati introdotti altri benefici legati allo stato fitosanitario e qualitativo delle piante, oltre ad un significativo risparmio d'acqua (circa un 80%). Il progetto finito ha reso possibile la conferma di queste ipotesi. Le scelte progettuali applicate, con alla base delle motivazioni pensate, ne hanno determinato lo sviluppo e la realizzazione. La grandezza della serra è legata infatti alle dimensioni che deve avere un prototipo, ovvero in scala rispetto al progetto finale. Doveva essere qualcosa di trasportabile e facilmente osservabile, per poter tenere d'occhio tutti i parametri, in grado di applicare con semplicità la gestione del processo e la tecnica di coltivazione idroponica. Si voleva dimostrare che il progetto, e il metodo applicato, avrebbero funzionato indipendentemente dal budget investito o dalle dimensioni della serra. I componenti installati, dovevano anch'essi rispettare le specifiche dettate in precedenza, garantendo però una funzionalità adeguata. La stampa 3D ha permesso di ottenere molti pezzi, difficilmente realizzabili o acquistabili altrimenti. Si è dovuta prestare particolare attenzione anche a tutti gli aspetti legati al metodo di coltivazione, effettuando scelte di progetto che permettessero, come per il sistema a gocciolamento installato, una facile regolazione e scalabilità. L'insieme di questi aspetti e di queste scelte ha permesso alla fine di ottenere una analisi sul progetto complessivo. I dati di crescita ottenuti, hanno confermato le ipotesi fatte all'inizio del progetto. E' possibile dire anzi, che i dati acquisiti superano le stime ipotizzate. La crescita ottenuta, nonostante si tratti di un prototipo, è raddoppiata rispetto al metodo di coltivazione classico. Inoltre, il programma risponde bene alle esigenze per le quali è stato progettato, garantendo un controllo efficiente e semplice della serra. Sicuramente è una buona base di partenza per il futuro anche se, gli eventuali risultati di un progetto di dimensioni maggiori, dovranno comunque essere confermati. Sicuramente il progetto della serra non si ferma a questa applicazione. Le possibili implementazioni sono tante, ma sicuramente le più rilevanti sono:

- Sviluppo di una serra con dimensioni maggiori, in modo da accogliere più piante e ottenere una resa maggiore. Questo sicuramente richiederà una gestione più complessa, con l'aggiunta ulteriore di dispositivi con risoluzioni migliori di quelle attuali;

- Convertire la serra, e quindi tutti i componenti ed il programma, verso una coltivazione *OUTDOOR*. Questo infatti permetterebbe l'installazione in tutti i luoghi e non necessariamente all'interno di un edificio;

- Sviluppo di grafici sullo schermo OLED installato per visualizzare i rendimenti delle piante e altri dati utili, avendo a disposizione ad esempio dei registri storici dei parametri ambientali e degli stessi allarmi, in modo da garantire anche una diagnostica del sistema;

- Implementazione di un controllo automatico sul dosaggio e il rilevamento delle caratteristiche chimiche della soluzione nutritiva. Per il momento questo processo viene svolto in modo manuale, ma è possibile automatizzare l'operazione;

- Possibilità di collegamento wireless o bluetooth con il programma tramite device e smartphone, per migliorare la connettività e lo scambio dati oppure implementare una piattaforma sul *WEB* così da risolvere i problemi legati alla distanza ridotta dei due metodi precedenti;

- Utilizzo di fonti rinnovabili per l'alimentazione della serra e dei componenti installati. Questo garantirebbe l'efficientamento energetico e la possibilità di installazione anche in luoghi remoti.

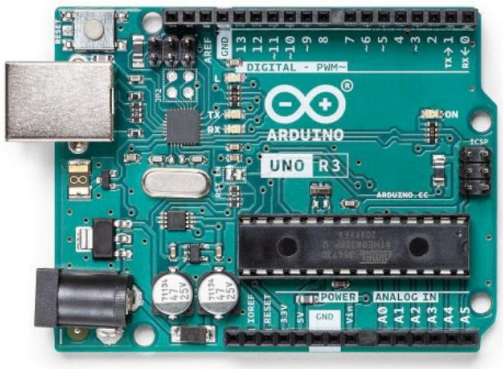
A Datasheet scheda Arduino

Di seguito, come descritto nel Capitolo 4, viene riportato il datasheet della scheda *Arduino UNO R3*. Il documento è quello ufficiale scaricabile dal sito di Arduino. Quella riportata è però la traduzione dalla lingua originale, in quanto il datasheet tradotto non era disponibile. Nel documento sono trattati tutti gli aspetti della scheda, dalla condizioni operative di funzionamento, al consumo di energia fino a passare alla parte circuitale e logica. E' possibile comprendere meglio i collegamenti tra i pin disponibili all'utente e il circuito integrato al suo interno, oltre che la lista dei pin e una loro descrizione. Ultime ma non per importanza, la certificazione CE di conformità e le certificazioni utili del caso.

Arduino® UNO R3

Manuale di riferimento del prodotto

COD: A000066



Descrizione

Arduino UNO R3 è la scheda perfetta per familiarizzare con l'elettronica e la codifica. Questo versatile microcontrollore è dotato del noto ATmega328P e del processore ATMega 16U2.
Questa scheda ti darà un'ottima prima esperienza nel mondo di Arduino.

Aree di destinazione:

Maker, introduzione, industrie

Caratteristiche

- Processore **ATMega328P**
  - **Memoria**
    - CPU AVR fino a 16 MHz
    - Flash da 32 KB
    - VERGOGNA 2KB
    - EEPROM da 1 KB
  - **Sicurezza**
    - Reset all'accensione (POR)
    - Rilevamento Brown Out (BOD)
  - **periferiche**
    - 2x timer/contatore a 8 bit con registro del periodo dedicato e confronto dei canali 1x timer/contatore
    - a 16 bit con registro del periodo dedicato, cattura in ingresso e confronto dei canali 1x USART con generatore di baud
    - rate frazionario e rilevamento dell'inizio del frame 1x controller/ periferica Serial Peripheral Interface (SPI) 1x controller
    - dual mode/I2C periferico 1x comparatore analogico (AC) con ingresso di riferimento scalabile
    -
    -
    - Watchdog Timer con oscillatore su chip separato
    - Sei canali PWM
    - Interruzione e riattivazione al cambio pin
  - **Processore ATMega16U2**
    - Microcontrollore basato su RISC AVR® a 8 bit
  - **Memoria**
    - Flash dell'ISP da 16 KB
    - EEPROM 512B
    - SRAM 512B
    - interfaccia debugWIRE per il debug e la programmazione su chip
  - **Potenza**
    - 2,7-5,5 volt

CONTENUTI

1 Il Consiglio

1.1 Esempi di applicazione

La scheda UNO è il prodotto di punta di Arduino. Indipendentemente dal fatto che tu sia nuovo nel mondo dell'elettronica o utilizzerai l'ONU come strumento per scopi educativi o attività legate al settore.

Primo ingresso all'elettronica: se questo è il tuo primo progetto nell'ambito della codifica e dell'elettronica, inizia con la nostra scheda più utilizzata e documentata; Arduino UNO. È dotato del noto processore ATmega328P, 14 pin di input/output digitali, 6 input analogici, connessioni USB, header ICSP e pulsante di reset. Questa scheda include tutto il necessario per un'ottima prima esperienza con Arduino.

Scheda di sviluppo standard del settore: utilizzando la scheda Arduino UNO nelle industrie, ci sono una serie di aziende che utilizzano la scheda UNO come cervello per i loro PLC.

Scopi educativi: sebbene il consiglio dell'ONU sia con noi da circa dieci anni, è ancora ampiamente utilizzato per vari scopi educativi e progetti scientifici. Gli standard elevati e le prestazioni di alta qualità della scheda la rendono un'ottima risorsa per acquisire il tempo reale dai sensori e attivare complesse apparecchiature di laboratorio per citare alcuni esempi.

1.2 Prodotti correlati

- Kit di partenza
- Tinkerkit Braccio Robot
- Esempio

2 valutazioni

2.1 Condizioni operative consigliate

Simbolo	Descrizione	min	Massimo
	Limiti termici conservativi per l'intera tavola:	-40 °C (-40 °F)	85 °C (185 °F)

NOTA: a temperature estreme, la EEPROM, il regolatore di tensione e l'oscillatore a cristallo potrebbero non funzionare come previsto a causa delle condizioni di temperatura estreme

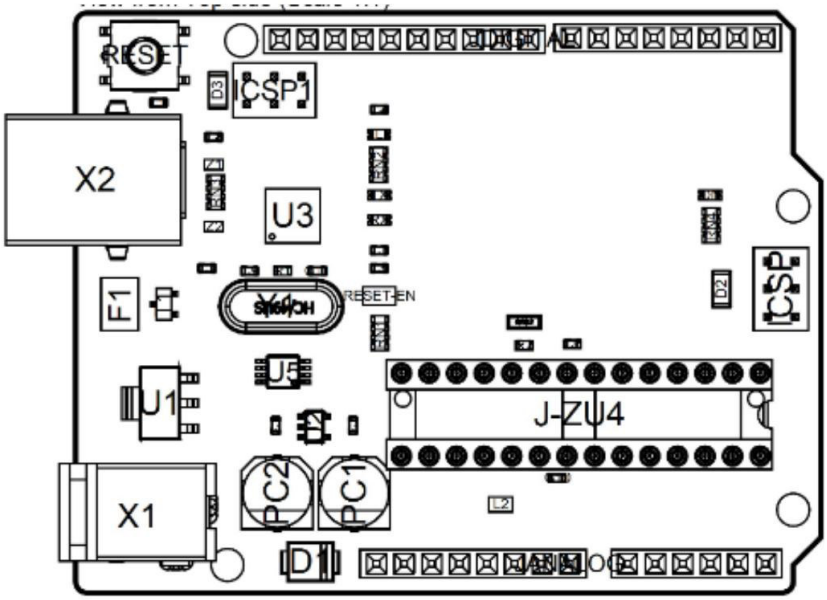
2.2 Consumo di energia

Simbolo	Descrizione	Min Tipo	Max Unità		
VINMax	Massima tensione di ingresso dal pad VIN	6	-	20	IN
VUSB Max	Massima tensione di ingresso dal connettore USB		-	5.5	IN
PMax	Massimo consumo di energia	-	-	xx	mA

3 Panoramica funzionale

3.1 Topologia della scheda

Vista dall'alto



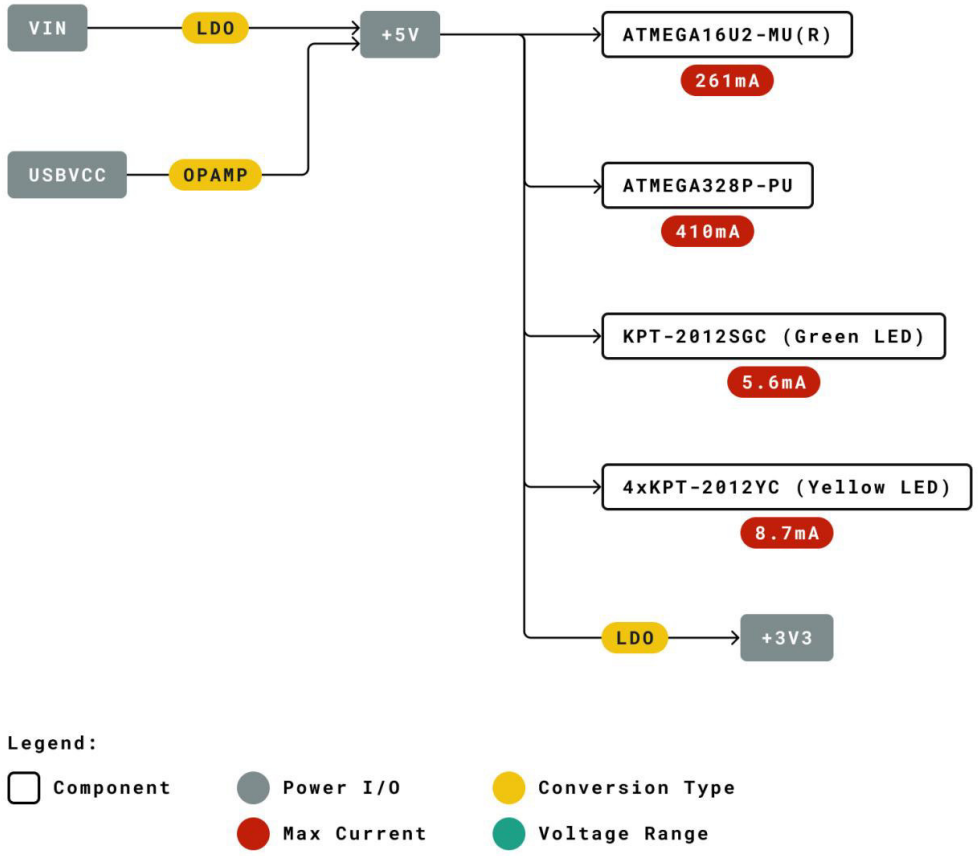
Topologia della scheda

Rif.	Descrizione	Rif.	Descrizione
X1	Presa di alimentazione 2,1x5,5 mm	U1	Regolatore SPX1117M3-L-5
X2	Connettore USB B	U3	Modulo ATMEGA16U2
PC1	Condensatore SMD EEE-1EA470WP 25V	U5	LMV358LIST-A.9 IC
PC2	Condensatore SMD EEE-1EA470WP 25V	F1	Condensatore a chip, ad alta densità
D1	Raddrizzatore CGRA4007-G	ICSP	Connettore con basetta a pin (attraverso il foro 6)
Modulo J-ZU4 ATMEGA328P		ICSP1	Connettore con basetta a pin (attraverso il foro 6)
Y1	Oscillatore ECS-160-20-4X-DU		

3.2 Processore

Il processore principale è un ATmega328P che funziona fino a 20 MHz. La maggior parte dei suoi pin sono collegati all'esterno intestazioni, tuttavia alcune sono riservate alla comunicazione interna con il coprocessore USB Bridge.

3.3 Albero del potere



Albero di potere

4 Operazioni di bordo

4.1 Per iniziare - IDE

Se vuoi programmare il tuo Arduino UNO offline devi installare Arduino Desktop IDE [1] Per collegare Arduino UNO al tuo computer, avrai bisogno di un cavo Micro-B USB. Questo fornisce anche alimentazione alla scheda, come indicato dal LED.

4.2 Per iniziare - Editor Web Arduino

Tutte le schede Arduino, inclusa questa, funzionano immediatamente sull'editor Web Arduino [2], semplicemente installando un semplice plug-in.

Arduino Web Editor è ospitato online, quindi sarà sempre aggiornato con le ultime funzionalità e supporto per tutte le schede. Segui **[3]** per iniziare a codificare sul browser e caricare i tuoi schizzi sulla tua bacheca.

4.3 Per iniziare - Arduino IoT Cloud

Tutti i prodotti Arduino abilitati per IoT sono supportati su Arduino IoT Cloud che ti consente di registrare, rappresentare graficamente e analizzare i dati dei sensori, attivare eventi e automatizzare la tua casa o la tua azienda.

4.4 Esempi di schizzi

Esempi di schizzi per Arduino XXX possono essere trovati nel menu "Esempi" nell'IDE di Arduino o nel Sezione “Documentazione” del sito Arduino Pro [4]

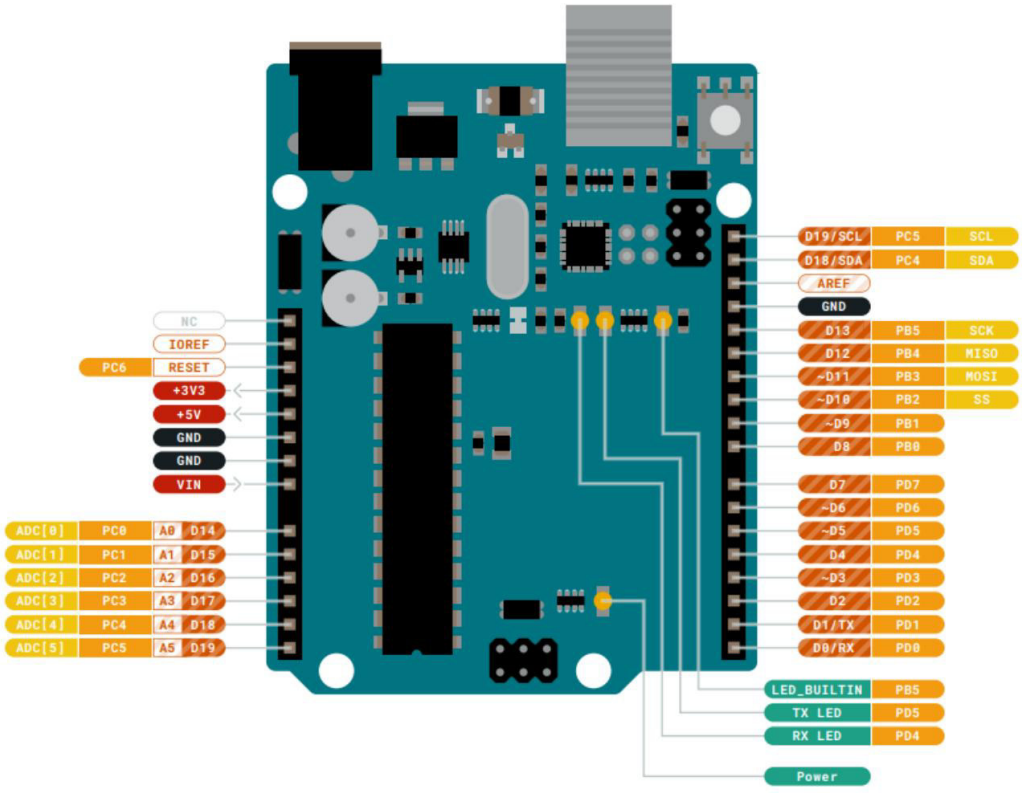
4.5 Risorse in linea

Ora che hai esaminato le basi di ciò che puoi fare con la scheda, puoi esplorare le infinite possibilità che offre controllando progetti entusiasmanti su ProjectHub **[5],** Arduino Library Reference **[6]** e il negozio online **[7]** dove sarà in grado di completare la tua scheda con sensori, attuatori e altro ancora

4.6 Recupero della scheda

Tutte le schede Arduino hanno un bootloader integrato che consente di eseguire il flashing della scheda tramite USB. Nel caso in cui uno schizzo blocchi il processore e la scheda non sono più raggiungibili tramite USB è possibile entrare in modalità bootloader toccando due volte il pulsante di reset subito dopo l'accensione.

5 piedini del connettore



piedinatura

5.1 JANALOG

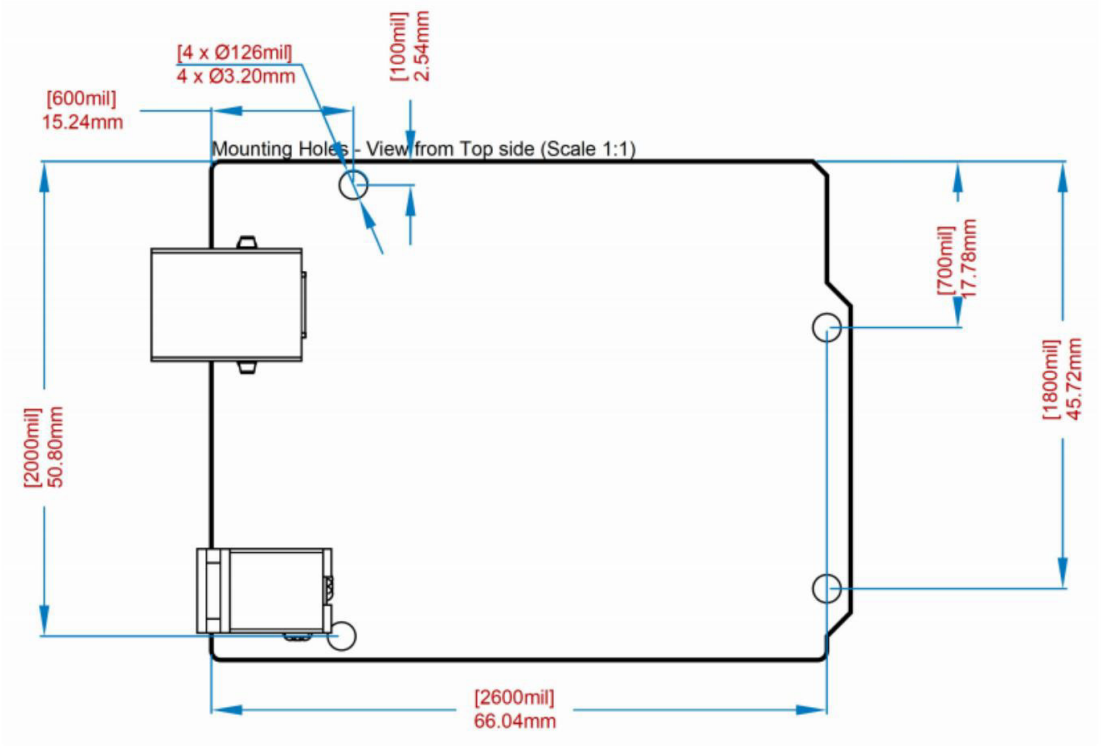
Funzione Pin		Tipo	Descrizione
1	NC	NC	Non collegata
2	IOREF	IOREF	Riferimento per logica digitale V - collegata a 5V
3	Ripristina	Ripristina	Ripristina
4	+3V3	Potenza	+3V3 Power Rail
5	+5V	Potenza	Linea di alimentazione +5V
6	GND	Potenza	Terra
7	GND	Potenza	Terra
8	VENIRE	Potenza	Ingresso di tensione
9	A0	Analogico/GPIO	Ingresso analogico 0 /GPIO
10	A1	Analogico/GPIO	Ingresso analogico 1 /GPIO
11	A2	Analogico/GPIO	Ingresso analogico 2 /GPIO
12	A3	Analogico/GPIO	Ingresso analogico 3 /GPIO
13	A4/SDA	Ingresso analogico/I2C	Ingresso analogico 4/I2C Linea dati
14	A5/SCL	Ingresso analogico/I2C	Ingresso analogico 5/I2C Linea orologio

5.2 JDIGITALE

Spillo	Funzione	Tipo	Descrizione
1	D0	Digitale/GPIO	Pin digitale 0/GPIO
2	D1	Digitale/GPIO	Pin digitale 1/GPIO
3	D2	Digitale/GPIO	Pin digitale 2/GPIO
4	D3	Digitale/GPIO	Pin digitale 3/GPIO
5	D4	Digitale/GPIO	Pin digitale 4/GPIO
6	D5	Digitale/GPIO	Pin digitale 5/GPIO
7	D6	Digitale/GPIO	Pin digitale 6/GPIO
8	D7	Digitale/GPIO	Pin digitale 7/GPIO
9	D8	Digitale/GPIO	Pin digitale 8/GPIO
10	D9	Digitale/GPIO	Pin digitale 9/GPIO
11	SS	Digitale	Selezione del chip SPI
12	FUMO	Digitale	SPI1 Uscita principale Ingresso secondario
13	MISO	Digitale	SPI Main In Secondary Out
14	SCK	Digitale	Uscita orologio seriale SPI
15	GND	Potenza	Terra
16	AREF	Digitale	Tensione di riferimento analogica
17	A4/SD4	Digitale	Ingresso analogico 4/I2C Linea dati (duplicata)
18	A5/SD5	Digitale	Ingresso analogico 5/I2C Linea orologio (duplicata)

5.3 Informazioni Meccaniche

5.4 Profilo della scheda e fori di montaggio



Schema di bordo

6 Certificazioni

6.1 Dichiarazione di conformità CE DoC (UE)

Dichiariamo sotto la nostra esclusiva responsabilità che i prodotti di cui sopra sono conformi ai requisiti essenziali delle seguenti Direttive UE e quindi qualificarsi per la libera circolazione all'interno dei mercati comprendenti quello europeo Unione (UE) e Spazio economico europeo (SEE).

Direttiva ROHS 2 2011/65/UE	
Conforme a:	EN50581:2012
Direttive 2014/35/UE. (LVD)	
Conforme a:	EN 60950-1:2006/A11:2009/A1:2010/A12:2011/AC:2011
Direttive 2004/40/CE e 2008/46/CE e 2013/35/UE, EMF	
Conforme a:	EN 62311:2008

6.2 Dichiarazione di conformità alla RoHS UE e REACH 211 19/01/2021

Le schede Arduino sono conformi alla Direttiva RoHS 2 2011/65/UE del Parlamento Europeo e RoHS 3 Direttiva 2015/863/UE del Consiglio, del 4 giugno 2015, sulla restrizione dell'uso di determinate sostanze pericolose in apparecchiature elettriche ed elettroniche.

Sostanza	Limite massimo (ppm)
Piombo (Pb)	1000
Cadmio (Cd)	100
Mercurio (Hg)	1000
Cromo esavalente (Cr6+)	1000
Poli bifenili bromurati (PBB)	1000
Eteri di difenile polibromurati (PBDE)	1000
Bis(2-Etilesil}ftalato (DEHP)	1000
Benzil butil ftalato (BBP)	1000
Dibutilftalato (DBP)	1000
Diisobutilftalato (DIBP)	1000

Esenzioni: non sono richieste esenzioni.

Le schede Arduino sono pienamente conformi ai relativi requisiti del Regolamento dell'Unione Europea (CE) 1907/2006 in materia di registrazione, valutazione, autorizzazione e restrizione delle sostanze chimiche (REACH). Dichiariamo nessuno di le SVHC (https://echa.europa.eu/web/guest/candidate-list-table), la Candidate List of Very High La preoccupazione per l'autorizzazione attualmente rilasciata dall'ECHA, è presente in tutti i prodotti (e anche nella confezione) in quantità totalizzando una concentrazione uguale o superiore allo 0,1%. Al meglio delle nostre conoscenze, dichiariamo anche che i nostri prodotti non contengono nessuna delle sostanze elencate nell'"Elenco delle autorizzazioni" (allegato XIV del regolamento REACH) e Sostanze estremamente problematiche (SVHC) in quantità significative come specificato dall'allegato XVII dell'elenco dei candidati pubblicato dall'ECHA (Agenzia europea per le sostanze chimiche) 1907/2006/CE.

6.3 Dichiarazione sui minerali di conflitto

In qualità di fornitore globale di componenti elettronici ed elettrici, Arduino è consapevole dei nostri obblighi in merito a leggi e regolamenti relativi ai minerali provenienti da conflitti, in particolare il Dodd-Frank Wall Street Reform and Consumer Protection Act, Sezione 1502. Arduino non genera o elabora direttamente i conflitti minerali come stagno, tantalio, tungsteno o oro. I minerali di conflitto sono contenuti nei nostri prodotti sotto forma di saldature o come componenti di leghe metalliche. Nell'ambito della nostra ragionevole due diligence, Arduino ha contattato i fornitori di componenti all'interno della nostra catena di fornitura per verificare la loro continua conformità alle normative. Sulla base delle informazioni finora ricevute, dichiariamo che i nostri prodotti contengono minerali provenienti da zone di conflitto.

7 FCC Attenzione

Eventuali cambiamenti o modifiche non espressamente approvati dal soggetto responsabile della conformità potrebbero invalidare il diritto dell'utente all'utilizzo dell'apparecchiatura.

Questo dispositivo è conforme alla parte 15 delle norme FCC. L'operazione è soggetta alle due seguenti condizioni:

(1) Questo dispositivo non può causare interferenze dannose

(2) questo dispositivo deve accettare qualsiasi interferenza ricevuta, comprese le interferenze che potrebbero causare un funzionamento indesiderato.

Dichiarazione FCC sull'esposizione alle radiazioni RF:

1. Questo trasmettitore non deve essere posizionato o funzionare insieme ad altre antenne o trasmettitori.
2. Questa apparecchiatura è conforme ai limiti di esposizione alle radiazioni RF stabiliti per un ambiente non controllato.
3. Questa apparecchiatura deve essere installata e utilizzata con una distanza minima di 20 cm tra il radiatore e il tuo corpo.

Inglese: i manuali dell'utente per gli apparati radio esenti da licenza devono contenere il seguente avviso o un avviso equivalente in una posizione ben visibile nel manuale dell'utente o in alternativa sul dispositivo o entrambi. Questo dispositivo è conforme agli standard RSS esenti da licenza di Industry Canada. L'operazione è soggetta alle due seguenti condizioni:

(1) questo dispositivo non può causare interferenze

(2) questo dispositivo deve accettare qualsiasi interferenza, comprese le interferenze che potrebbero causare un funzionamento indesiderato del dispositivo.

Inglese: questo dispositivo è conforme agli standard RSS esenti da licenza di Industry Canada. Lo sfruttamento è autorizzato alle seguenti due condizioni:

(1) il dispositivo non può causare interferenze

(2) l'utente del dispositivo deve accettare qualsiasi interferenza radio subita, anche se l'interferenza rischia di comprometterne il funzionamento.

Avvertimento SAR IC:

Italiano Questa apparecchiatura deve essere installata e utilizzata con una distanza minima di 20 cm tra il radiatore e il corpo.

Italiano: Durante l'installazione e il funzionamento di questo dispositivo, la distanza tra il radiatore e il corpo è di almeno 20 cm.

Importante: la temperatura di esercizio dell'EUT non può superare gli 85 ÿ e non deve essere inferiore a -40 ÿ.

Con la presente Arduino Srl dichiara che questo prodotto è conforme ai requisiti essenziali e ad altri pertinenti disposizioni della Direttiva 2014/53/UE. Questo prodotto può essere utilizzato in tutti gli stati membri dell'UE.

8 Informazioni sull'azienda

Nome della ditta	Arduino Srl
indirizzo aziendale	Via Andrea Appiani 25 20900 MONZA Italy

9 Documentazione di riferimento

Riferimento	Collegamento
Arduino IDE (Desktop)	https://www.arduino.cc/en/Principale/Software
Arduino IDE (Cloud)	https://create.arduino.cc/editor
Ottenere IDE cloud Cominciato	https://create.arduino.cc/projecthub/Arduino_Genuino/iniziare-con-arduino web-editor-4b3e4a
Sito Web Arduino Pro https://www.arduino.cc/pro	
Riferimento	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trend
alla libreria Project Hub	https://www.arduino.cc/reference/en/
Negozio online	https://store.arduino.cc/

10 Cronologia delle revisioni

Data	Revisione	I cambiamenti
xx/06/2021	1	Rilascio scheda tecnica

Bibliografia

[1] Wikipedia. *Idroponica — Wikipedia, L'enciclopedia libera*. [Online; in data 14-novembre-2021]. 2021. URL: http://it.wikipedia.org/w/index.php?title=Idroponica&oldid=119856486.

[2] William Texier. *Idroponica per tutti – Tutto sull'orticoltura domestica*. Italiano. Trad. da Alessia Fisichella. Con annot. di Loriel Verlomme. Mama Editions, 2015. Cap. Introduzione. ISBN: 978-2-84594-112-0.

[3] BOKU (Università delle risorse naturali e delle scienze della vita - Vienna). *Aratura nella coltivazione*. Articolo online dedicato alla coltivazione del mais, ma con dati riportati anche se altre colture cerealicole. URL: https://www.kws.com/it/it/consulenza/semina/aratura/l'aratura-nella-coltivazione-del-mais/.

[4] William Texier. *Idroponica per tutti – Tutto sull'orticoltura domestica*. Italiano. Trad. da Alessia Fisichella. Con annot. di Loriel Verlomme. Mama Editions, 2015. Cap. I diversi sistemi idroponici. ISBN: 978-2-84594-112-0.

[5] *Immagini reperibili online*. URL: https://www.verticalfarmitalia.cloud/vertical-farm/.

[6] William Texier. *Idroponica per tutti – Tutto sull'orticoltura domestica*. Italiano. Trad. da Alessia Fisichella. Con annot. di Loriel Verlomme. Mama Editions, 2015. Cap. Substrati idroponici. ISBN: 978-2-84594-112-0.

[7] Wikipedia. *Lana di roccia*. URL: https://it.wikipedia.org/wiki/Lana_di_roccia.

[8] Obi. *Pietre laviche*. URL: https://www.obi-italia.it/utensili-ed-accessori-per-il-barbecue/pietre-laviche-cmi/p/4103339.

[9] Agrobio. *Pomice*. URL: https://www.agrobio-mi.it/prodotto/pomice-agricola-florovivaismo-50-lt-europomice/.

[10] Agri Pet Garden. *Perlite*. URL: https://www.agripetgarden.it/perlite-4lt.html.

[11] Idroponica.it. *Fibra di cocco*. URL: https://www.idroponica.it/utilizzo-fibra-cocco-indoor_28-163.html.

[12] Wikipedia. *Segatura*. URL: https://it.wikipedia.org/wiki/Segatura_%28materiale%29.

[13] William Texier. *Idroponica per tutti – Tutto sull'orticoltura domestica*. Italiano. Trad. da Alessia Fisichella. Con annot. di Loriel Verlomme. Mama Editions, 2015. Cap. La grow room. ISBN: 978-2-84594-112-0.

[14] William Texier. *Idroponica per tutti – Tutto sull'orticoltura domestica*. Italiano. Trad. da Alessia Fisichella. Con annot. di Loriel Verlomme. Mama Editions, 2015. Cap. Gestire la soluzione nutritiva. ISBN: 978-2-84594-112-0.

[15] GenitronSviluppo. *Plantagon: Agricoltura Urbana Verticale e Scalabile. Un Progetto Svedese fra Ambizione e Soluzione per le Grandi Metropoli di Tutto il Mondo*. URL: `http://www.genitronsviluppo.com/2009/07/10/plantagon-agricoltura-urbana-verticale/`.

Ringraziamenti

Non sono molto bravo con i ringraziamenti, ma ci terrei in particolare a ringraziare tutti coloro che hanno contribuito alla realizzazione del progetto. Prima di tutti vorrei ringraziare il mio relatore, il Prof.re Paolo Magnone per la disponibilità nell'accettare e supportare la mia proposta di tesi. Anche alla Prof.ssa Annalisa Massaccesi va un enorme ringraziamento poiché, senza di lei, sarebbe stato difficile calcolare il volume d'aria della serra.

Non posso però non nominare la mia famiglia, che mi ha permesso di intraprendere questo percorso universitario, lasciandomi libertà di scelta e supportandomi in ogni mia decisione. La loro presenza è stata fondamentale, in particolar modo nei momenti più bui. Mi hanno trasmesso la forza e la voglia di stringere i denti ed andare avanti, più forte di prima. Altra persona che ci tengo a ringraziare è mia nonna Olimpia. Un grazie per ogni candela accesa durante ogni singolo esame sostenuto non sarebbe abbastanza.

Infine gli amici, che ci sono tutt'ora o che ci sono stati negli anni. Anche il loro contributo è stato importantissimo nel mio percorso universitario, sia nei momenti di gioia sia in quelli di amarezza. Hanno saputo supportarmi nei momenti di delusione, e gioire assieme in quelli di felicità.

Spero di non aver scordato nessuno, ma poco importa: è solo un foglio di carta con scritti dei nomi e qualche parola. I veri ringraziamenti si vedono nella vita di tutti i giorni.

Grazie ancora a tutti quanti e, anche se forse può sembrare da egoisti, il ringraziamento più grande lo faccio a me stesso, per la tenacia e la voglia che sono riuscito a mettere in questi anni, facendo molti sacrifici e ricevendo molte porte chiuse in faccia. Tutto questo mi ha cambiato, ha temprato il mio carattere, e mi ha permesso di diventare quello che sono, senza dimenticare da dove sono partito e dove voglio arrivare.