Università del Salento

Dipartimento di Ingegneria dell'Innovazione

Corso di Laurea Magistrale in Ingegneria Informatica

DOCUMENTAZIONE DI PROGETTO

Signal Acquisition and Electronic Design

Un prototipo di Inseguitore Solare dotato di sensori ambientali e supporto di memoria realizzato con una scheda di prototipazione rapida Arduino Uno R3

Docenti

Prof. Paolo Visconti

Ing. Roberto de Fazio

Autore

Culcea Cezar Narcis

Matricola n° 20086225

ANNO ACCADEMICO 2022/2023

Indice

Capitolo 0

Abstract

Il progetto nasce dalla necessità di ottenere il massimo rendimento da un impianto di *pannelli solari*, fornendo essi una direzione in cui posizionarsi per ricevere i raggi solari in maniera perpendicolare.

Accanto a questa funzionalità principale, sono accostate altre funzionalità minori di monitoraggio meteorologico:

- rilevazione della *temperatura*
- rilevazione della *pressione*
- rilevazione della *pioggia*

I dati rilevati dalla stazione devono essere salvati su un *supporto di memoria* e visualizzati su un *display*.

Capitolo 1

Overview Progetto

1.1 Caso d'uso

Dall'analisi dei requisiti si evidenziano due casi d'uso:

- Rilevazione posizione del sole
- Pulizia dei pannelli solari in caso di pioggia

Rilevazione posizione del sole

In caso di condizioni meteorologiche normali il sistema deve:

1. Trovare la posizione del sole (**angolo orizzontale** e **angolo verticale**)
2. Rilevare i valori **pioggia**, **temperatura**, **pressione**
3. Mostrare i dati su un **display**
4. Salvare i dati in un *CSV* associandoli ad un **timestamp**

Pulizia dei pannelli solari in caso di pioggia

In caso di pioggia il sistema deve:

1. Comunicare ai pannelli di posizionarsi orizzontalmente
2. Rilevare i valori **pioggia**, **temperatura**, **pressione**
3. Mostrare i dati su un **display**
4. Salvare i dati in un *CSV* associandoli ad un **timestamp**

1.2 Schema circuitale

In questa sezione verrà analizzato il circuito del sistema progettato. Il sistema fa uso di tre diverse linee di alimentazione, evidenziate tramite cavi striati:

- *Linea 1*: 5 Volt forniti dalla Board Arduino R3 (usata per la maggior parte dei componenti)
- *Linea 2*: 3.3 Volt forniti dalla Board Arduino R3 (usata per il sensore BMP280)
- *Linea 3*: 5 Volt forniti esternamente (alimentazione dei servo motori)

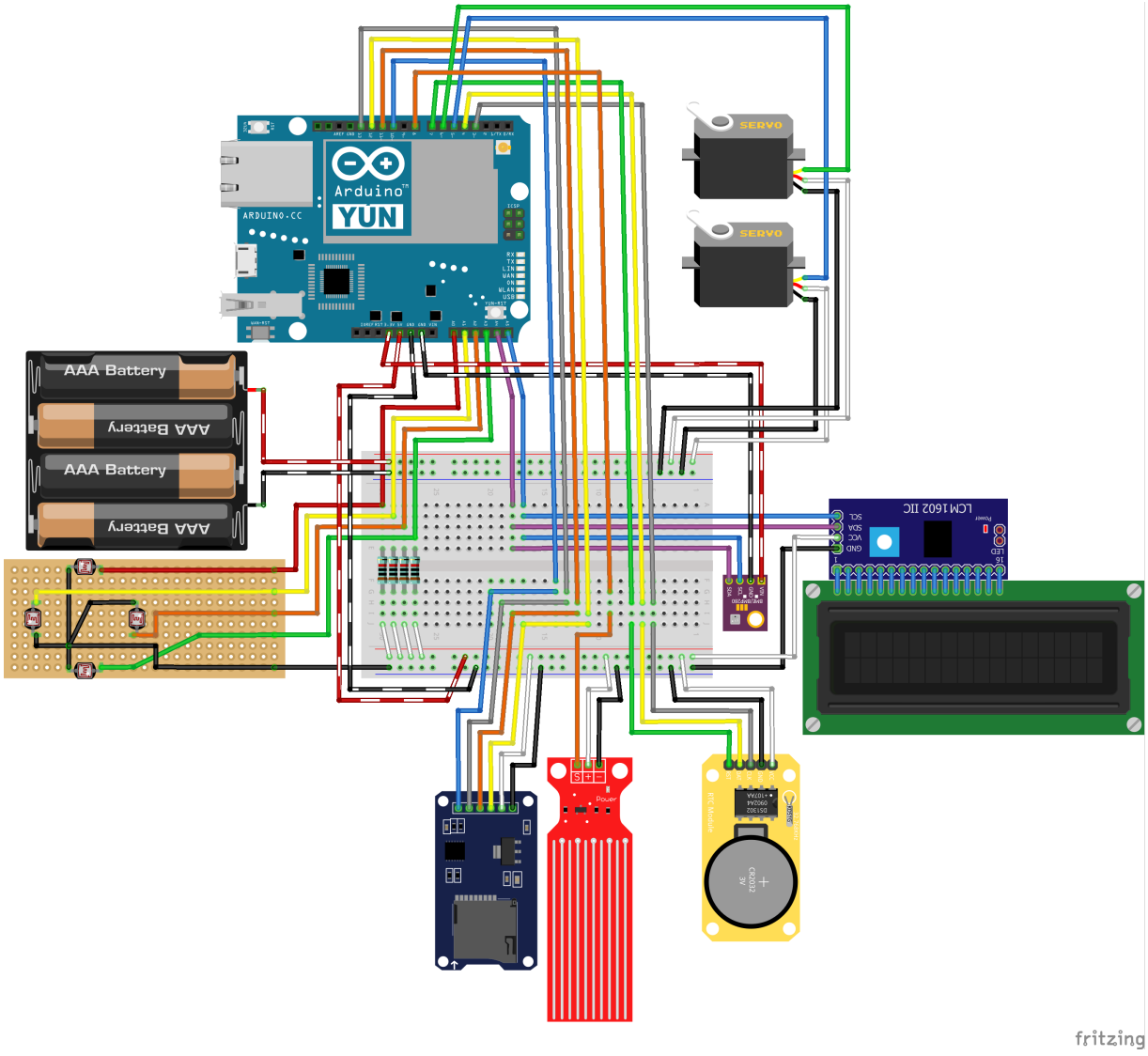

Figura 1.1: Schema circuitale del progetto

Analizzando nel dettaglio il circuito della **linea di alimentazione 1** abbiamo il seguente cablaggio:

- Display LCD: connesso alla linea I2C tramite SDA (cavo viola) e SCL (cavo blu)
- Water Sensor: connesso al GPIO 8
- MicroSD Card Adapter: connesso alla linea SPI tramite MISO (cavo giallo - GPIO 12), MOSI (cavo arancio - GPIO 11), CLK (cavo grigio - GPIO 13) e CS (cavo blu - GPIO 10)
- MH-Real-Time Clock Module 2: connesso con CLK (cavo grigio - GPIO 3), DAT (cavo giallo - GPIO 4), RST (cavo verde - GPIO 7)
- fotoresistori: inseriti in un partitore di tensione con resistenze da 10 kΩ e collegati ai GPIO A0, A1, A2, A3

Analizzando nel dettaglio il circuito della **linea di alimentazione 3** abbiamo il seguente cablaggio:

- Servo 1: collegato al GPIO 5
- Servo 2: collegato al GPIO 6

1.3 Assemblaggio

Il componente che insegue il sole è il pezzo principale del progetto. Esso ospita i *fotoresistori* ed è mosso dai due *servomotori* lungo l'asse orizzontale e verticale.

I modelli 3D da stampare sono stati ottenuti dal sito **Thingiverse**.

Link: https://www.thingiverse.com/thing:2467743.

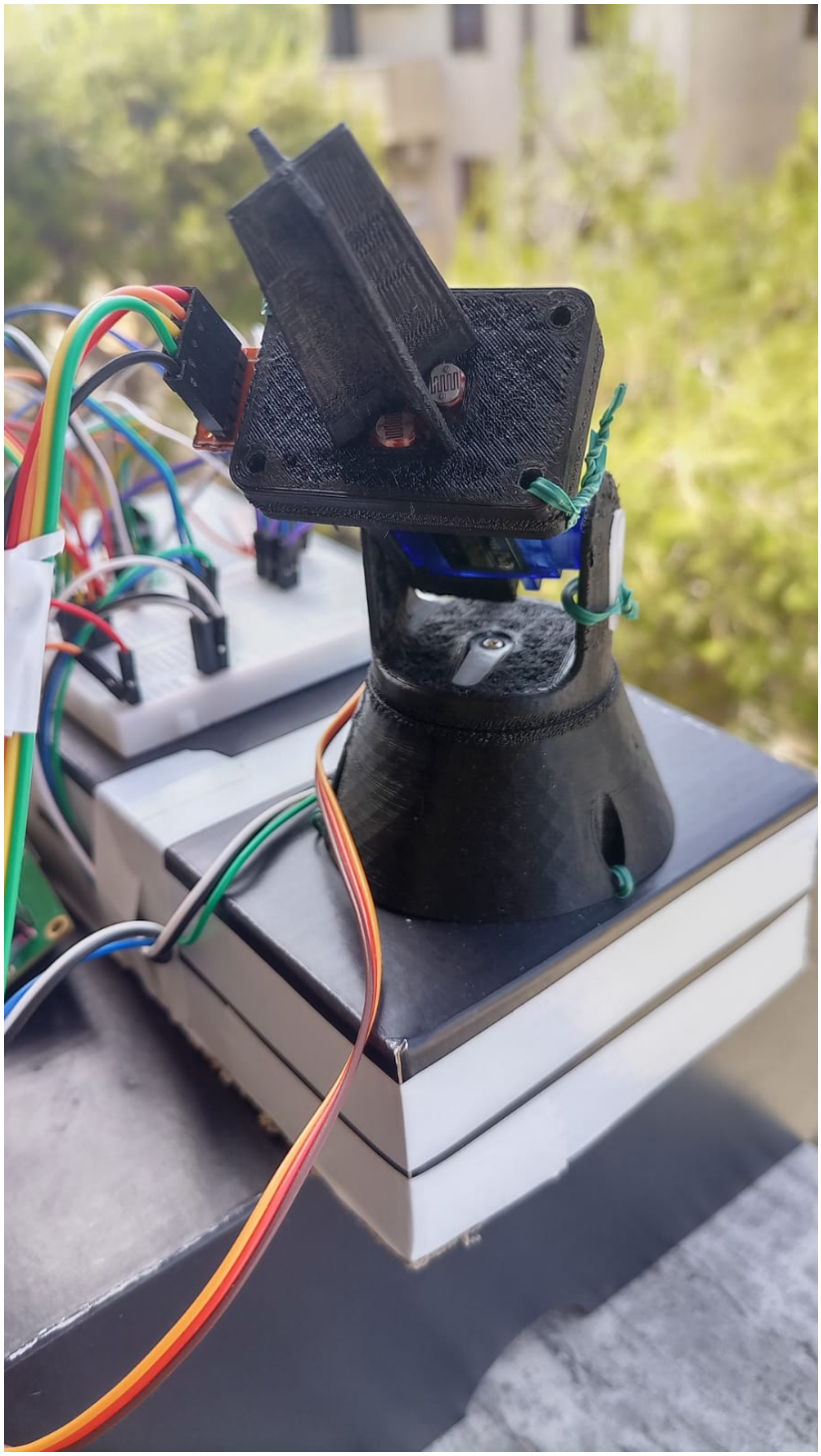

Figura 1.2: Inseguitore Solare 3D stampato e assemblato

I fotoresistori sono stati saldati su una millefori secondo lo schema illustrato nella Figura 1.1.

Successivamente l'inseguitore solare è stato montato su una base contenente dei pesi ed è stato collegato al resto del circuito.

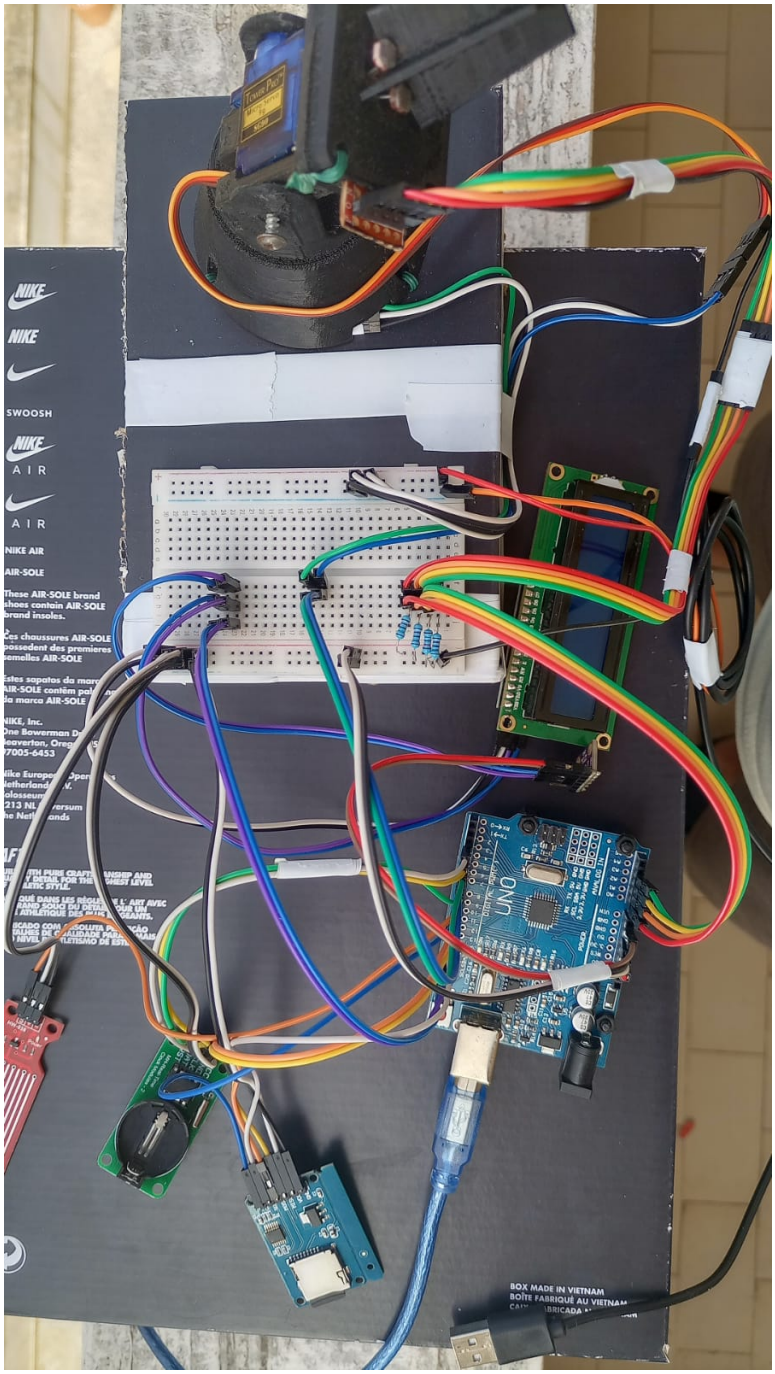

Figura 1.3: Overview sul sistema

Capitolo 2

Implementazione Hardware

2.1 Componenti

2.1.1 Arduino UNO R3

Arduino Uno è un dispositivo basato su un microcontrollore che offre infinite possibilità nella realizzazione di circuiti elettronici. La sua versatilità è evidente attraverso i suoi 14 pin digitali programmabili, che possono fungere sia da ingressi che da uscite, consentendo anche funzioni speciali come la generazione di segnali PWM o la comunicazione UART. Inoltre, sono disponibili 6 ingressi analogici per l'acquisizione e l'elaborazione di segnali analogici. Il cuore della scheda è costituito dal microcontrollore ATmega328 di Atmel, che opera a una frequenza di 16 MHz e vanta una memoria flash di 32 KB, una SRAM di 2 KB e una memoria EEPROM di 1 KB.

Figura 2.1: Arduino UNO R3

L'alimentazione di Arduino Uno avviene tramite la comoda porta USB o attraverso l'apposito connettore dedicato. Un aspetto particolarmente intelligente di questa scheda è la capacità di scegliere automaticamente la fonte di alimentazione esterna, qualora sia collegato sia il cavo USB che il connettore di alimentazione. Questo garantisce un funzionamento fluido e senza interruzioni, adattandosi alla situazione di utilizzo.

2.1.2 Fotoresistori

La fotoresistenza, un dispositivo fondamentalmente costituito da materiale semiconduttore, sfrutta il solfuro di cadmio come materiale comune. Questo componente presenta due elettrodi interdigitati, che permettono di rilevare la resistenza elettrica. La rilevazione si basa sull'effetto fotoelettrico: quando un fotone colpisce il materiale semiconduttore, promuove un portatore di carica dalla banda di valenza alla banda di conduzione, creando un aumento di portatori in grado di condurre la corrente. Di conseguenza, la resistenza elettrica del materiale diminuisce.

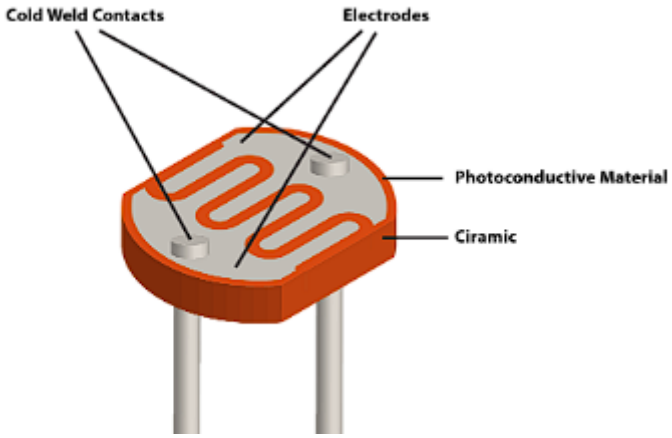

Figura 2.2: Fotoresistore

Il valore caratteristico della resistenza elettrica della fotoresistenza è calibrato a una specifica luminanza (10 Lux) e temperatura (10°C). Nel nostro caso, tale valore è di 10 kΩ. Questo significa che, in condizioni standard di 10 Lux e 10°C, la fotoresistenza avrà una resistenza elettrica di 10 kΩ.

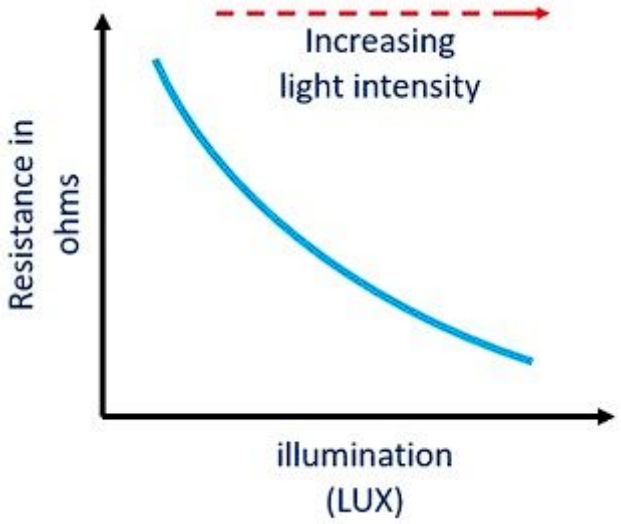

Figura 2.3: Caratteristica di un Fotoresistore

All'interno dell'inseguitore, i fotoresistori sono usati all'interno di un partitore di tensione con resistenze da 10 kOhm.

Quando sono colpiti da luce intensa, la lettura del valore effettuata attraverso i pin analogici, diminuisce fino in prossimità del valore di massa.

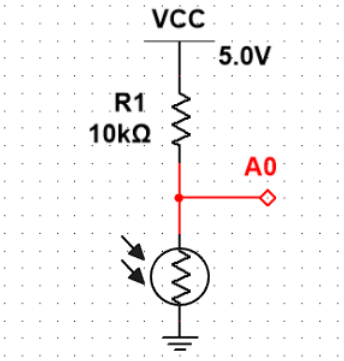

Figura 2.4: Schema del partitore di tensione utilizzato

Il valore di tensione letto corrisponde a:

$$V_{A0} = 5\text{Volt} * \frac{R_{fotoresistore}}{R_{fotoresistore} + R1}$$

2.1.3 Water Sensor

Il funzionamento del water sensor, utilizzabile come sensore di livello, è fondamentalmente semplice e si basa sulla conduttività dell'acqua. Quando l'acqua entra in contatto con

le piste del sensore, che sono alternativamente connesse alla tensione positiva e alla linea GND, permette alla corrente di fluire tra queste piste.

Questo valore di corrente viene poi elaborato elettronicamente all'interno del sensore e trasmesso sotto forma di un segnale analogico all'ingresso analogico della scheda di controllo. La scheda non misura direttamente la tensione elettrica, ma converte il segnale analogico in un valore numerico.

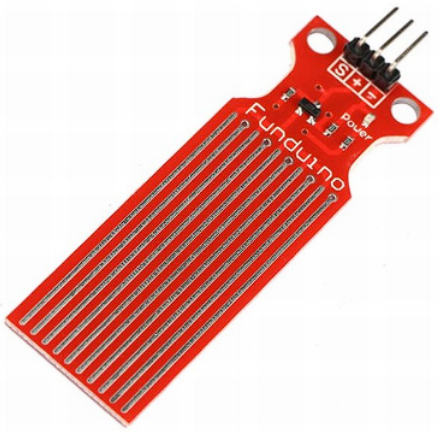

Figura 2.5: Water sensor

Lo schema circuitale è semplice. Quando connesso all'alimentazione, la corrente scorre nel led di accensione e sul collector del BJT Q1 si riportano 5V, così come su una parte della serpentina.

Quando l'acqua tocca il sensore, collega le piste, facendo scorrere una corrente di base nel BJT. Questa apre il BJT e riporta una valore in tensione sul Signal. Più acqua tocca il sensore, più corrente di base di genera, incrementando il valore di uscita.

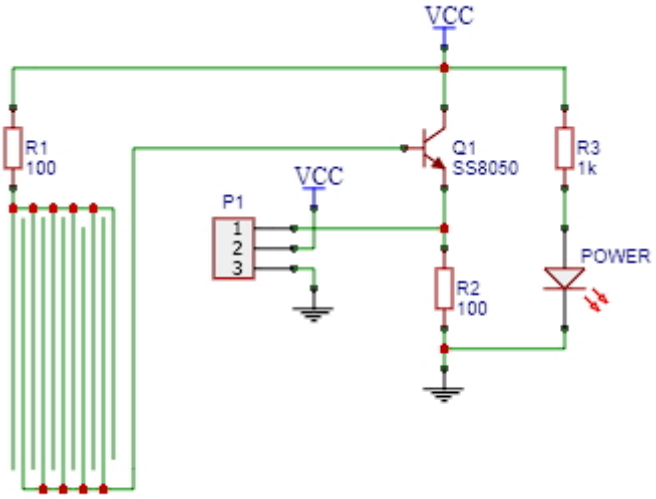

Figura 2.6: Circuito interno del Water Sensor

2.1.4 MicroSD Card Adapter

Il modulo offre la possibilità di integrare le memorie MicroSD (Micro Secure Digital) nei progetti basati su Arduino.

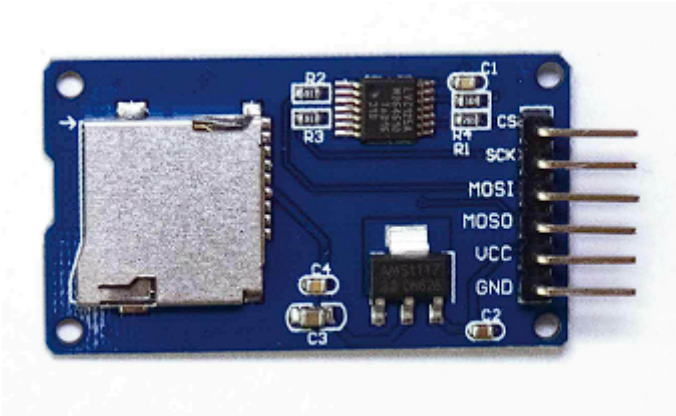

Figura 2.7: Modulo MicroSD Card Adapter

Dal punto di vista elettrico, lo schema del modulo è molto semplice e consiste nel collegamento dei pin della scheda di memoria al connettore. La comunicazione tra il microcontrollore e la scheda SD avviene tramite il protocollo SPI (Serial Peripheral Interface), un sistema di comunicazione tra il microcontrollore e altri circuiti integrati o tra più microcontrollori.

Il protocollo SPI si basa su quattro segnali:

- SCK: Serial Clock, emesso dal master.
- MISO: Serial Data Input, Master Input Slave Output (ingresso per il master ed uscita per lo slave).
- MOSI: Serial Data Output, Master Output Slave Input (uscita dal master).
- CS: Chip Select, Slave Select, emesso dal master per selezionare con quale dispositivo slave vuole comunicare.

Per utilizzare la scheda SD con Arduino, la comunicazione avviene attraverso i pin digitali 11 (MOSI), 12 (MISO) e 13 (CLK) (su maggior parte delle schede Arduino). Inoltre, un altro pin è utilizzato per selezionare la scheda SD (CS). Questo può essere il pin 10 (CS) hardware SS (su maggior parte delle schede Arduino) o un altro pin specificato nella chiamata a SD.begin (). In questo modo, il modulo consente una facile e veloce integrazione delle memorie MicroSD nei progetti basati su Arduino, aprendo la porta a numerose applicazioni e possibilità di archiviazione e gestione dei dati.

2.1.5 MH-Real-Time Clock Module 2

Il modulo MH-Real-Time Clock Module -2 si basa sull'integrato DS1302, che al suo interno ospita un preciso orologio in tempo reale e calendario, insieme a 31 byte di RAM statica. Questo chip comunica con il microprocessore tramite un'interfaccia seriale. Il modulo RTC/calendario fornisce dati come secondi, minuti, ore, giorno, data, informazioni sul mese e sull'anno.

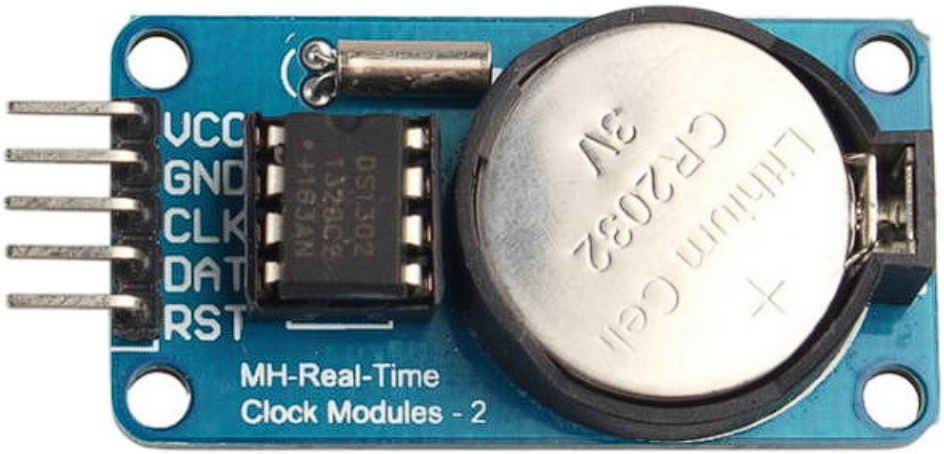

Figura 2.8: Modulo MH-Real-Time Clock Module 2

Per l'interfaccia con il microcontrollore, sono necessari solo tre fili: CE (Reset), I/O (linea dati) e SCLK (serial clock). La trasmissione dei dati tra il microcontrollore e l'orologio/RAM può avvenire in modalità singola, un byte alla volta, o in un'unica invio fino a 31 byte.

Il DS1302 è stato progettato per funzionare con un consumo energetico molto basso, conservando i dati e le informazioni di clock con meno di 1μW di potenza. Inoltre, grazie alla batteria di backup non ricaricabile da 3.3V tipo CR2032, è in grado di mantenere i dati e il funzionamento del clock per più di 10 anni.

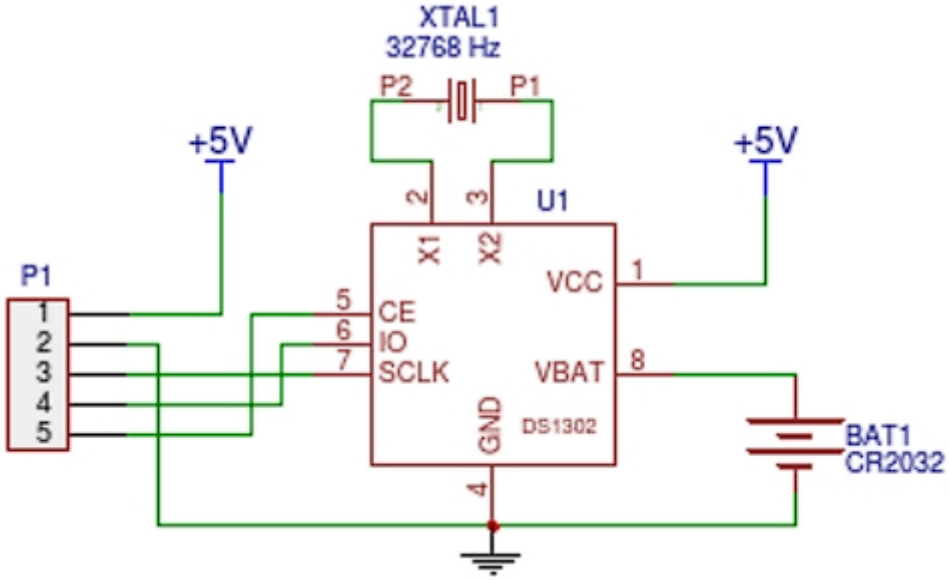

Figura 2.9: Circuito interno del MH-Real-Time Clock Module 2

Il pin VCC è dedicato all'alimentazione primaria, di solito 3.3V ma può essere utilizzato anche 5V. Il pin VBAT, invece, viene alimentato dalla batteria di backup CR2032 da 3.3V, per garantire il funzionamento continuo anche in caso di interruzione dell'alimentazione principale.

2.1.6 BMP280

Il sensore BMP280 è un dispositivo barometrico di pressione a bassa potenza che offre anche la capacità di misurare la temperatura. Grazie alla sua elevata precisione nel rilevare la pressione, questo sensore è ideale per l'utilizzo a bordo di droni e nelle stazioni di rilevazione meteo.

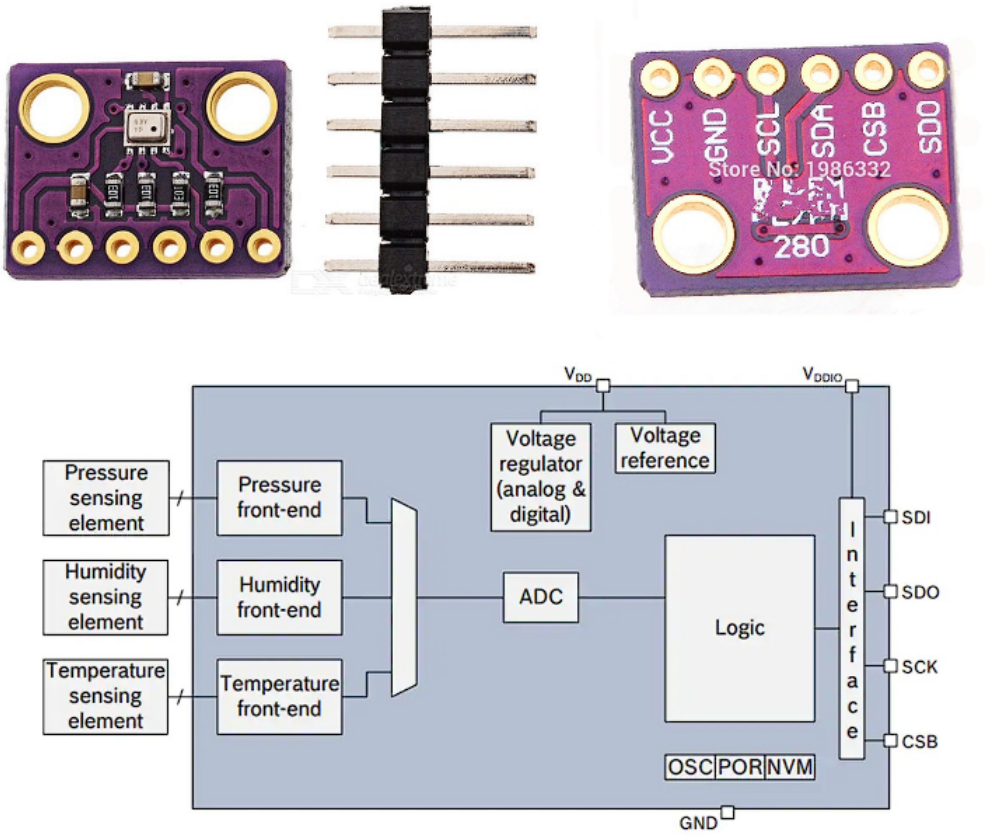

Figura 2.10: Modulo BMP280 e schema logico interno

All'interno del sensore, è presente un MEMS (Micro Electro-Mechanical System), un sistema elettro-meccanico che converte uno spostamento meccanico in un segnale elettrico. Questo utilizza un elemento piezo-resistivo che, deformandosi in risposta alla pressione, genera un segnale elettrico proporzionale alla pressione.

Il chip include anche un convertitore analogico-digitale (ADC) e un modulo di comunicazione I2C o SPI. Inoltre, è dotato di un filtro digitale che permette di eseguire diverse misurazioni e successivamente filtrarle per ridurre al minimo l'errore di misurazione.

Il sensore BMP280 può effettuare una singola misura oppure eseguire fino a 16 misurazioni per ottenere la massima risoluzione, ma questa opzione comporta un consumo di energia più elevato. L'errore minimo ottenibile è di 0.16 Pascal, ma ciò richiede un tempo di misura più lungo e una corrente più alta. Le considerazioni sulla risoluzione si applicano anche alla temperatura

Questo sensore ha i seguenti parametri:

- Temperatura di misura: da -40°C a +85°C.

- Range di misura della pressione: da 300 a 1100 hPa.
- Precisione assoluta di misura pressione: ±7hPa (-20°C a 0°C), ±1hPa (0°C a 65°C).
- Precisione assoluta di misura temperatura: ±1°C.

2.1.7 LCD Display

Il display LCD 16x2 è un'interfaccia di uscita che offre la possibilità di visualizzare messaggi su due righe, ognuna composta da 16 caratteri. Al suo interno, è presente il circuito integrato di interfaccia HD44780, che svolge il ruolo di ricevere comandi e dati dal microcontrollore, elaborandoli per la visualizzazione delle informazioni sullo schermo LCD.

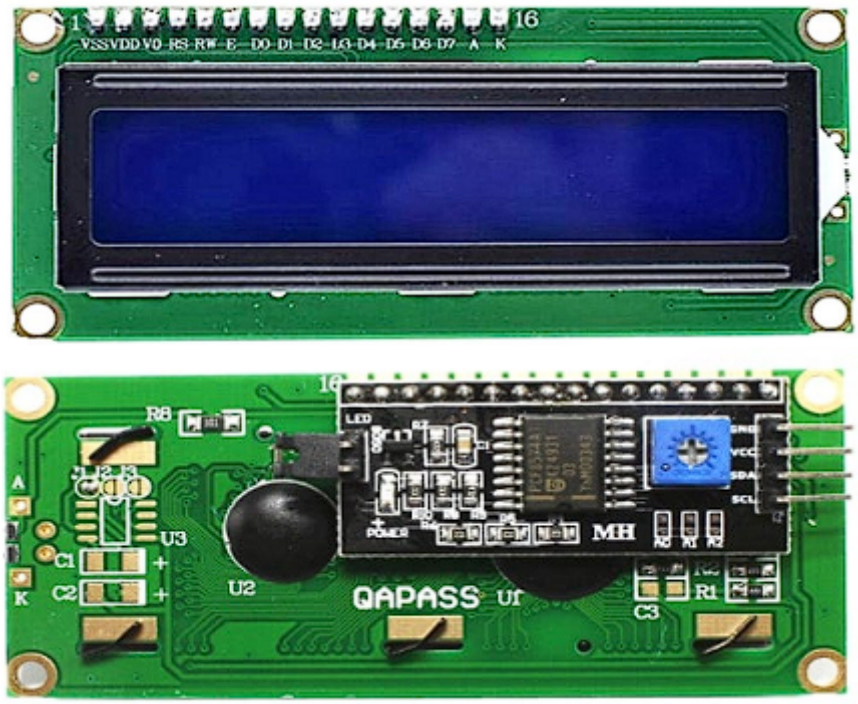

Figura 2.11: Display LCD con Driver I2C

Il display LCD 16x2 supporta due modalità di funzionamento: una a 8 bit e una a 4 bit. La disposizione totale dei pin è di 16, dove i pin dall'7 al 14 rappresentano i byte di dati che verranno visualizzati sul display. Questi pin possono essere impostati come 0 o 1 per trasmettere il byte al display, il quale leggerà e memorizzerà il dato in un registro, successivamente mostrandolo sullo schermo in base alle istruzioni ricevute tramite i pin 4, 5 e 6.

Di solito, per il funzionamento a 8 bit, il modulo richiede l'utilizzo di 11 pin digitali, mentre per il funzionamento a 4 bit ne richiede 6. Tuttavia, esiste un modulo (Fig. 2.11) che consente di controllare il display LCD utilizzando solamente due pin analogici, oltre ai pin di alimentazione e massa. Questo modulo converte il bus seriale I2C in un bus parallelo compatibile con il display LCD, semplificando notevolmente la gestione del display mediante il protocollo I2C. Grazie a questo approccio, è possibile risparmiare risorse sui pin del microcontrollore e semplificare l'interfacciamento con il display LCD 16x2.

Il Driver è dotato un trimmer che permette di regolare il contrasto del display, consentendo di ottenere una visualizzazione ottimale dei caratteri. Inoltre, è presente un jumper removibile che permette di attivare o disattivare la retroilluminazione del display a seconda delle esigenze.

Per la configurazione dell'indirizzo I2C, sono presenti tre connessioni denominate A1, A2 e A3. Queste connessioni possono essere utilizzate per impostare l'indirizzo I2C a 3 bit, consentendo di selezionare uno dei diversi indirizzi disponibili nell'intervallo tra 0x20 e 0x27. L'indirizzo predefinito impostato di fabbrica è 0x27, e i tre pin A0, A1 e A2 vengono lasciati aperti in questa configurazione predefinita.

2.1.8 MicroServo SG90 a rotazione continua

I servomotori a rotazione continua sono dispositivi elettromeccanici che offrono una rotazione continua dell'albero di uscita, a differenza dei tradizionali servomotori che hanno un angolo di rotazione limitato. All'interno del servomotore a rotazione continua, il cuore del sistema è costituito da un motore a corrente continua (DC) e da un meccanismo di feedback a potenziometro.

Il motore DC riceve l'energia elettrica dai driver di controllo, che a loro volta sono comandati da un segnale di controllo proveniente da un microcontrollore o da un circuito di controllo. Questo segnale di controllo determina la velocità e la direzione della rotazione dell'albero di uscita del servomotore.

Figura 2.12: Microservo SG90 a rotazione continua

Il potenziometro è una componente fondamentale per il feedback del servomotore. Esso è collegato all'albero di uscita del motore e misura continuamente la sua posizione angolare. Il segnale di tensione generato dal potenziometro viene inviato al circuito di controllo che lo utilizza per controllare e regolare costantemente la velocità e la direzione di rotazione del motore.

I servomotori utilizzati sono pilotati da un segnale PWM a 50 Hz. A differenza dei servomotori normali, non è possibile comandarli indicando loro una posizione da raggiungere. Il segnale PWM decide solo la direzione e la velocità di rotazione.

In particolare, considerando un periodo di 20 ms si ha:

- Impulso di 1 ms: rotazione a velocità massima verso destra
- Impulso di 1,5 ms: il servomotore si ferma
- Impulso di 2 ms: rotazione a velocità massima verso sinistra.

2.2 Protocolli

2.2.1 Protocollo SPI

L'interfaccia SPI, originariamente ideata dalla Motorola (ora Freescale) per i propri microprocessori e microcontrollori, descrive una comunicazione sincrona e full-duplex tra un singolo Master e un singolo Slave.

L'interfaccia SPI utilizza quattro linee di collegamento (oltre alla massa) ed è conosciuta come 4 Wire Interface. Il Master è responsabile di iniziare la comunicazione e fornire il clock allo Slave. Le linee nell'interfaccia SPI sono di solito indicate come segue:

- MOSI: Master Output Slave In - MISO: Master Input Slave Output - SCLK: Serial Clock (generato dal Master) - SS: Slave Select (Selezione dello Slave)

È importante notare che la comunicazione SPI richiede la presenza di un segnale SS (Slave Select), consentendo al Master di selezionare lo Slave con cui comunicare, sia per inviare che ricevere dati.

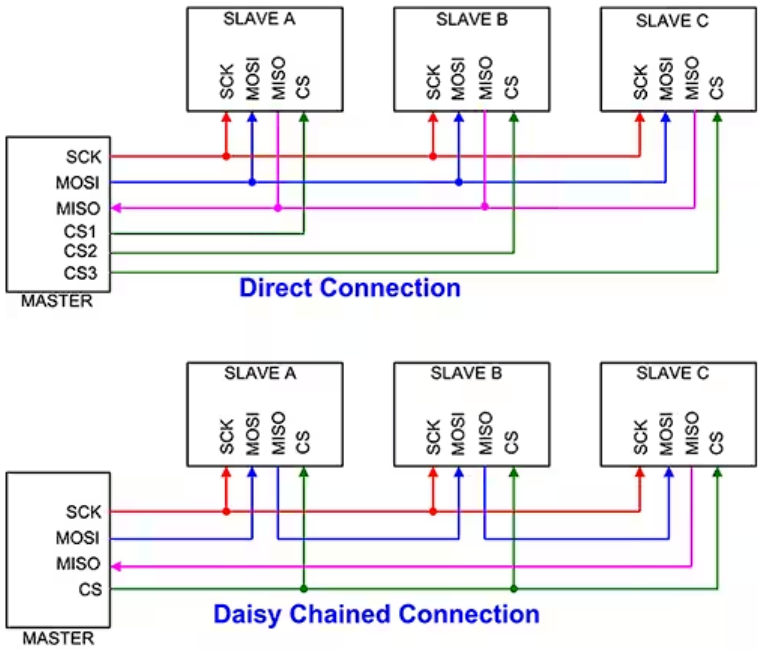

Figura 2.13: Modalità di connessione multislave per SPI

Sebbene la comunicazione SPI sia inizialmente progettata per una connessione tra un singolo Master e un singolo Slave, è possibile collegare più Slave, a condizione che le periferiche Slave supportino l'opzione di avere la linea MISO configurata come three states o floating (alta impedenza).

Esistono quattro modalità di trasmissione per l'interfaccia SPI e vengono solitamente configurate attraverso due parametri, denominati CPOL (Clock Polarity) e CPHA (Clock Phase), spesso implementati con due bit.

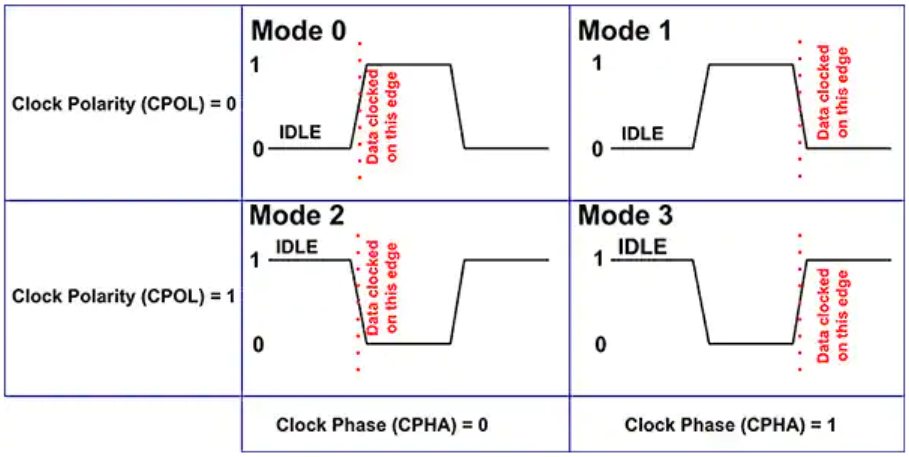

Figura 2.14: Modalità di trasmissione per SPI

2.2.2 Protocollo I2C

Il protocollo I2C è una forma di comunicazione seriale che permette di sostituire le comunicazioni parallele. Questa interfaccia utilizza un bus che può essere configurato in due modi:

- Modalità multimaster: in cui sono presenti più master, ma solo uno può essere attivo alla volta, mentre gli altri funzionano come slave.
- Modalità multislave: in cui ci sono più dispositivi slave.

Quando il master desidera iniziare una comunicazione, invia l'indirizzo dello slave con cui vuole comunicare attraverso il bus I2C, che è composto da due linee bidirezionali:

- SDA (Serial Data): utilizzata per inviare dati a 8 bit.
- SCL (Serial Clock): utilizzata per trasmettere il segnale di clock che sincronizza la trasmissione.

Entrambe le linee sono di tipo open-drain, il che significa che richiedono una resistenza di pull-up per mantenere le linee a livello logico alto quando non sono in uso (idle).

Ogni periferica collegata al bus I2C avrà un proprio indirizzo, costituito da 7 o 10 bit, che la identifica.

La comunicazione tra il master e lo slave si svolge attraverso diverse fasi:

1. Inizializzazione: il master occupa la linea inviando un segnale di start (livello logico alto).

2. Il master controlla se gli slave sono impegnati in altre comunicazioni: se la linea è a livello logico alto, gli slave sono occupati e il master attende; se la linea è a livello logico basso, il master assume il controllo del bus.

3. Start: il master invia l'indirizzo dello slave con cui vuole comunicare sulla stessa linea.

4. Ascolto: lo slave interessato entra in modalità di ascolto.

5. Invio dei pacchetti: il master invia i dati in pacchetti da 1 byte.

6. Ricezione: lo slave riceve i pacchetti e invia un ACK (Acknowledge) al master per confermare la ricezione (questa fase continua finché tutti i dati non sono stati scambiati).

7. Stop: per indicare la fine della comunicazione.

Il protocollo I2C consente sia operazioni di scrittura (ottavo bit impostato a 0) che di lettura (ottavo bit impostato a 1). Durante la comunicazione, si verifica una commutazione dei livelli logici tra SDA e SCL solo quando SCL è a livello logico basso. SDA diventa alto prima che SCL cambi il suo stato.

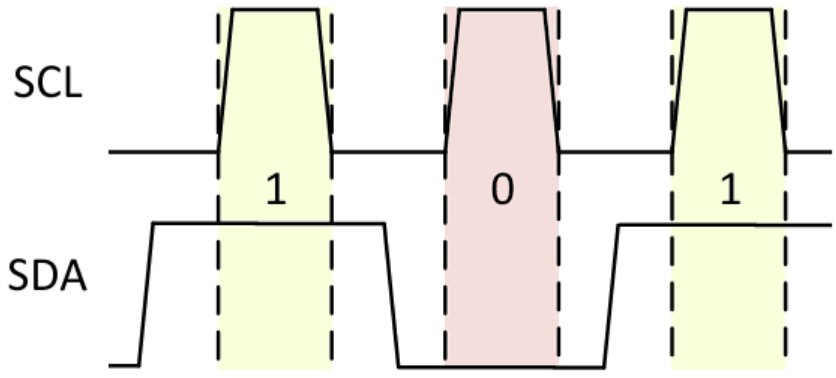

Figura 2.15: Commutazione durante la comunicazione I2C

Le uniche due eccezioni, entrambe generate dal master, sono i due simboli di Start e Stop che identificano l'inizio e la fine di un frame.

Capitolo 3

Implementazione Software

Il codice sviluppato è accessibile pubblicamente sulla piattaforma GitHub al seguente link:

https://github.com/Kynesy/SolarTracker.git

Figura 3.1: QR Code della Repository

3.1 Librerie, Costanti e Variabili

La logica inizia con l'import delle librerie. Lo scopo di queste è l'utilizzo delle interfacce *SPI* e *I2C*. Esse vengono inoltre utilizzate per la gestione dei moduli *BMP280*, *MicroSD Card Adapter*, *LCD Dispay* e *Real Time Clock*.

```
1 #include <Wire.h>
2 #include <SPI.h>
3 #include <Adafruit_BMP280.h>
4 #include "SD.h"
5 #include <LiquidCrystal_I2C.h>
6 #include <DS1302.h>
```

Si procede con la definizione dei GPIO utilizzati.

```
#define CM_CLK_PIN 3 /**< Clock pin for Clock Module  */
#define CM_DATA_PIN 4 /**< Data pin for Clock Module */
#define CM_RST_PIN 7 /**< Reset pin for Clock Module */
#define SD_SELECT 10 /**< ChipSelect for MicroSD Card Adapter*/
#define UP A0 /**< Analog Pin for UP photoresistor */
#define DX A1   /**< Analog Pin for RIGHT photoresistor */
#define SX A2 /**< Analog Pin for LEFT photoresistor  */
#define DW A3 /**<  Analog Pin for DOWN photoresistor  */
#define WATER_SENSOR_PIN_DIGITAL 8 /**< Digital Pin Of Water Sensor */
#define SERVO_X_PIN 5 /**< Digital pin for Horizontal Servo */
#define SERVO_Y_PIN 6 /**< Digital pin for Vertical Servo */
```

Infine vengono definite le costanti e le variabili utilizzate all'interno di tutto il file.

```
//COSTANTS
#define MAX_INTERVAL 5000 /**< Maximum interval for data update in milliseconds. */
const int threeshold = 15; /**< Threshold value for sun tracking servo control. */
LiquidCrystal_I2C lcd(0x27, 16, 2); /**< LCD module object. */
Adafruit_BMP280 bmp; /**< BMP280 sensor object. */
DS1302 rtc(CM_RST_PIN, CM_DATA_PIN, CM_CLK_PIN); /**< Real-time clock module object. */
char* fileName = "dati1.csv"; /**< File name used to store data on SD */
File file; /**< File Object */

//VARIABLES
int startInterval; /**< Stores the starting interval time */
float temperature; /**< Stores the temperature value in Celsius degrees */
float pressure; /**< Stores the pressure value in hPa */
bool isRaining; /**< Stores the raining value: 1 if is raining, 0 otherwise */
int x_angle; /**< Stores the horizontal angle of the Servo_X */
int y_angle; /**< Stores the vertical angle of the Servo_Y */
```

3.2 Logica principale

La logica dell'intero progetto è riassumibile ad alto livello con il seguente FlowChart.

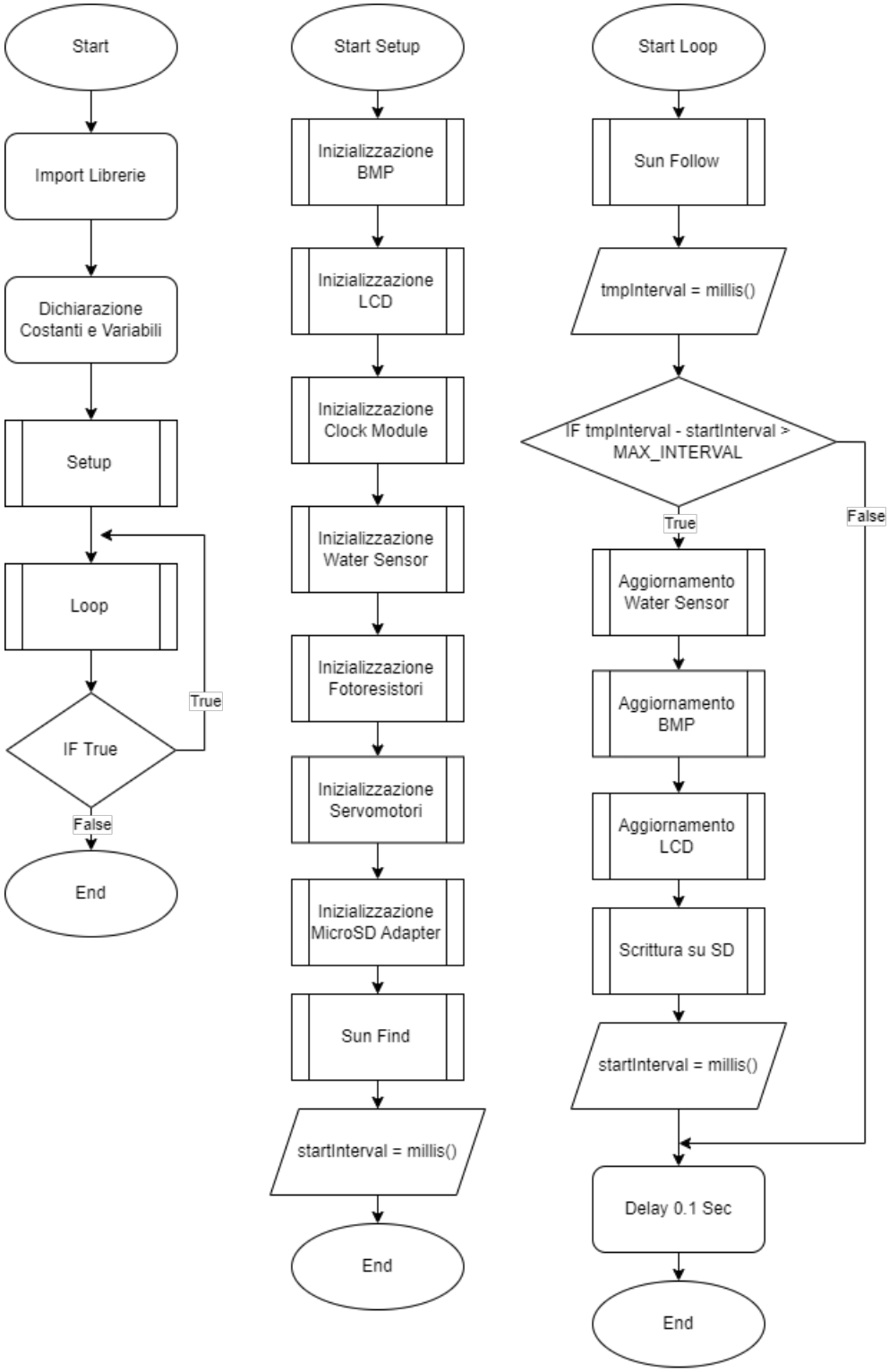

Figura 3.2: FlowChart della logica main, del setup e del loop

3.2.1 setup()

La funzione di *setup()* inizializza la *Seriale* ed i vari moduli utilizzati. Si occupa anche di indirizzare il dispositivo verso il sole grazie alla funzione *sun_find()*.

```
void setup() {
  // Initialize Serial communication
  Serial.begin(9600);

  // Initialize modules and sensors
  bmp_init();
  lcd_init();
  clock_init(12, 0, 0, 20, 7, 2023);
  water_sensor_init();
  photoresistor_init();
  servo_init();
  sd_init();
  sun_find();
  startInterval = millis();
}
```

3.2.2 loop()

La funzione di *loop()* si occupa di aggiustare ad ogni ciclo la posizione del Solar Tracker. Inoltre ogni 5 secondi, effettua un aggiornamento dei valori, li mostra sul display e li salva su un file *CSV* all'interno dell'SD.

```
void loop() {
  sun_follow();
  // Perform data update after a certain interval
  int tmpInterval = millis();
  if(tmpInterval - startInterval > MAX_INTERVAL){
    water_sensor_update();
    bmp_update();
```

```
8       lcd_update();
9       sd_write();
10      //sd_read();
11      startInterval = millis();
12    }
13    delay(100);
14  }
```

3.3 Logica Sun Tracking

3.3.1 Sun Find()

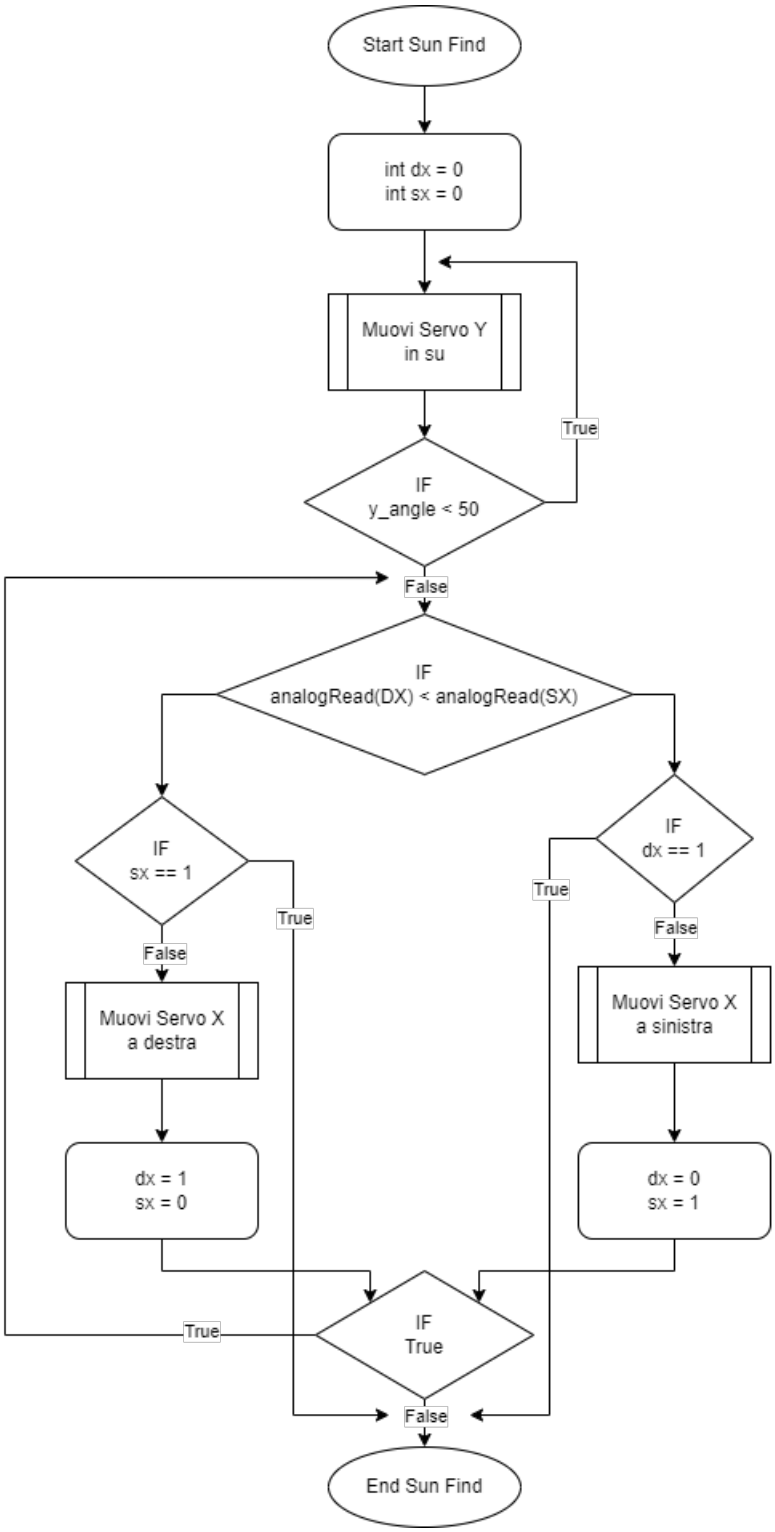

Figura 3.3: FlowChart della funzione sun_find()

La funzione *sun_find()* si ha lo scopo di indirizzare il device verso il Sole. All'avvio si occupa di inclinare di 50° in verticale. Successivamente, analizzando i valori dei fotoresistori laterali, la torretta viene ruotata fino ad essere rivolta verso il sole.

```
void sun_find(){
  int sx=0;
  int dx=0;

  while(y_angle < 50){
    Y_moveUp();
  }

  while(true){
    if(analogRead(DX) < analogRead(SX)){
      if(sx==1){
        break;
      }
      X_moveDx();
      sx=0;
      dx=1;
    }else{
      if(dx==1){
        break;
      }
      X_moveDx();
      sx=1;
      dx=0;
    }
  }
}
```

3.3.2 Sun Follow()

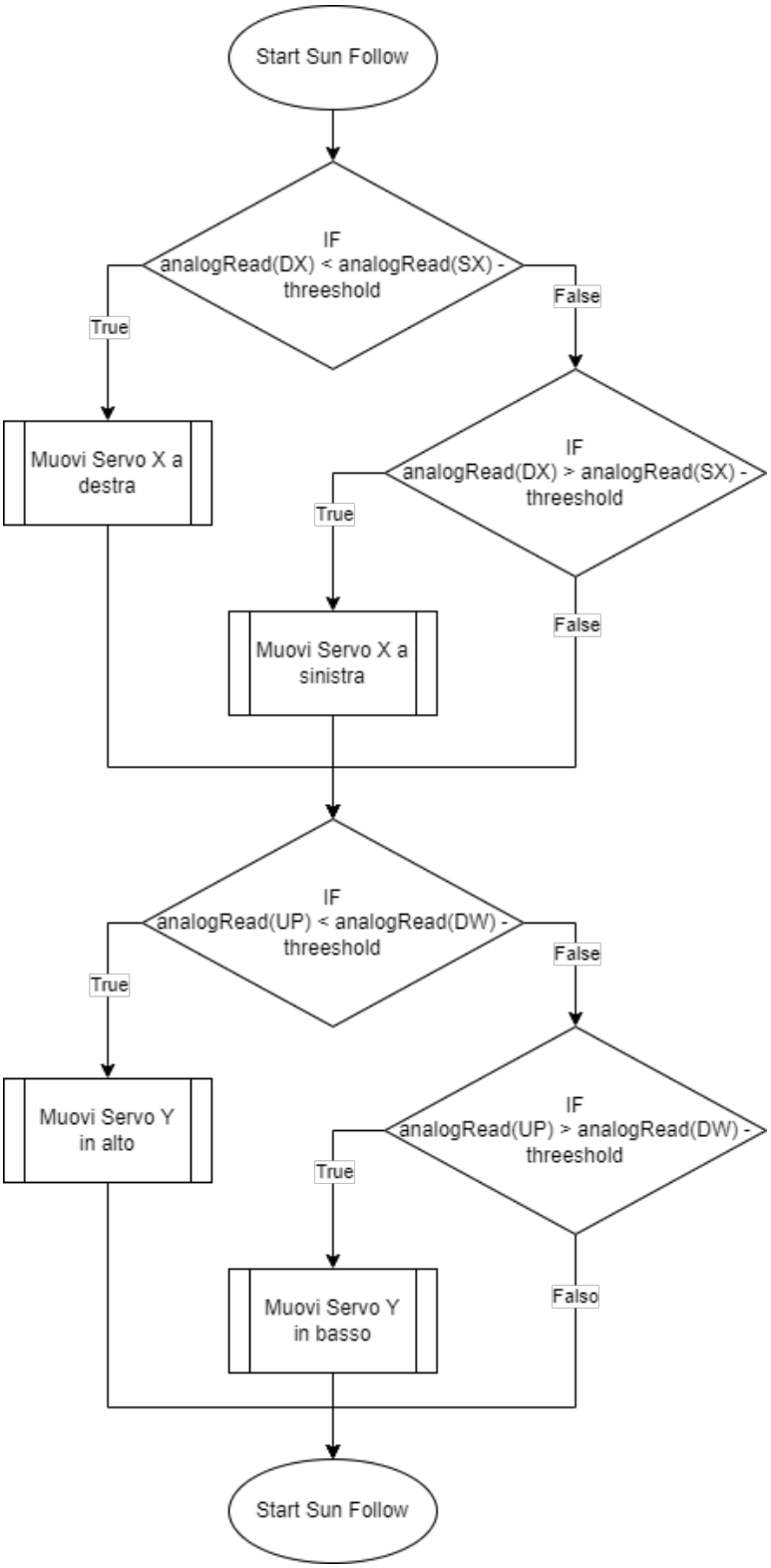

Figura 3.4: FlowChart della funzione sun_follow()

La funzione si occupa di muovere la torretta nella direzione con più luce. Non contiene un loop interno e fa fare solo piccoli movimenti ai servomotori, a seconda dei valori letti dai fotoresistori. Per questo motivo viene eseguita nel *loop()* ogni 0.1 secondi.

Dato che i valori delle fotoresistenze non saranno mai uguali, l'uso della variabile *threeshold*, consente di impostare una sensibilità ed evita di muovere costantemente la torretta. La soglia deve essere regolata in base alla luminosità ambientale.

```
1  void sun_follow(){
2    if(analogRead(DX) < analogRead(SX) - threeshold){
3      X_moveDx();
4    }else if(analogRead(DX) > analogRead(SX) + threeshold){
5      X_moveSx();
6    }
7
8    if(analogRead(UP) < analogRead(DW) - threeshold){
9      Y_moveUp();
10   }else if(analogRead(UP)> analogRead(DW) + threeshold){
11     Y_moveDown();
12   }
13 }
```

3.4 Logica Fotoresistori

I fotoresistori sono inizializzati con la funzione *photoresistor_init()*. I relativi GPIO analogici sono impostati come INPUT.

```
1  void photoresistor_init(){
2    pinMode(UP, INPUT);
3    pinMode(DW, INPUT);
4    pinMode(SX, INPUT);
5    pinMode(DX, INPUT);
6    Serial.println("PR | Successo inizializzazione");
7  }
```

3.5 Logica Water Sensor

La logica del modulo di rilevamento della pioggia consta di due funzioni:

- *water_sensor_init()*: imposta il GPIO come input
- *water_sensor_update()*: aggiorna la variabile globale con 1 se sta piovendo, 0 altrimenti

```
void water_sensor_init(){
  pinMode(WATER_SENSOR_PIN_DIGITAL, INPUT);
  Serial.println("WTR| Successo inizializzazione");
}

void water_sensor_update() {
  int water_presence = digitalRead(WATER_SENSOR_PIN_DIGITAL);
  isRaining = water_presence ? true : false;
}
```

3.6 Logica MicroSD Card Adapter

La logica del modulo MicroSD consta di due funzioni:

- *sd_init()*: inizializza il Chip Select del modulo
- *sd_write()*: crea il file sul modulo e vi appende una riga contenente i dati (timestamp, pressione, temperatura, pioggia, angolo orizzontale, angolo verticale). Nel caso in cui piova, comunica di posizionarsi verso l'alto.

```
static void sd_init(){
  if (!SD.begin(SD_SELECT)) {
    Serial.println("SD | Errore inizializzazione");
    while (1);
  }
  Serial.println("SD | Successo inizializzazione");
}
```

```
8
9  static void sd_write(){
10   file = SD.open(fileName, FILE_WRITE);
11   if (file) {
12     if(isRaining){
13       file.println(String(rtc.getTimeStr()) + "-" + String(rtc.getDateStr()) + ","
14             + String(temperature) + "," + String(pressure) + "," + String(isRaining) +
15             "," + "0" + "," + "90");
16     }else{
17       file.println(String(rtc.getTimeStr()) + "-" + String(rtc.getDateStr()) + ","
18             + String(temperature) + "," + String(pressure) + "," + String(isRaining) +
19             "," + String(x_angle) + "," + String(y_angle));
20     }
21     file.close();
22     Serial.println("SD | Scrittura su file riuscita");
23   } else {
24     Serial.println("SD | Errore apertura file");
25   }
26 }
```

3.7 Logica MH-Real-Time Clock Module 2

La logica del Modulo Clock Real Time consta della sola funzione di inizializzazione, con la quale è possibile impostare *data* e *ora*.

```
1  void clock_init(int hour, int min, int sec, int day, int month, int year){
2    rtc.halt(false);
3    rtc.writeProtect(false);
4    rtc.setTime(hour, min, sec);
5    rtc.setDate(day, month, year);
6
7    Serial.println("CLK| Successo inizializzazione");
8  }
```

3.8 Logica BMP280

La logica del sensore BMP280 consta di due funzioni:

- *bmp_ init()*: accede al BMP attraverso il suo indirizzo ed imposta le risoluzioni del sampling
- *bmp_ update()*: legge i valori di temperatura e pressione ed aggiorna le variabili globali

```
1  void bmp_init() {
2    unsigned status;
3    status = bmp.begin(0x76);
4    if (!status) {
5      Serial.println("BMP| Errore inizializzazione");
6
7      while (1) delay(10);
8    }else{
9      Serial.println("BMP| Successo inizializzazione");
10   }
11
12   bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
13                   Adafruit_BMP280::SAMPLING_X2,
14                   Adafruit_BMP280::SAMPLING_X16,
15                   Adafruit_BMP280::FILTER_X16,
16                   Adafruit_BMP280::STANDBY_MS_500);
17 }
18
19 void bmp_update(){
20   temperature = bmp.readTemperature();
21   pressure = bmp.readPressure()/100;
22 }
```

3.9 Logica LCD Display

La logica del display LCD consta di due funzioni:

- *lcd_ init()*: inizializza il diplay ed imposta la retroilluminazione
- *lcd_ update()*: aggiorna i dati (time, temperature, pressure, rain) mostrati sul display

```
1  void lcd_init(){
2    lcd.init();
3    lcd.init();
4    lcd.backlight();
5
6    Serial.println("LCD| Successo inizializzazione");
7  }
8
9  void lcd_update(){
10   lcd.clear();
11   lcd.setCursor(0, 0);
12   lcd.print(String(rtc.getTimeStr()) + " " + String(temperature) + "*C");
13
14   lcd.setCursor(0, 1);
15   if(isRaining){
16     lcd.print(String(pressure) + "hPa" + "  Rain");
17   }else{
18     lcd.print(String(pressure) + "hPa" + "  Sun");
19   }
20 }
```

3.10 Logica MicroServo SG90

La logica dei servomotori consta di varie funzioni:

- *servo_init()*:
- *X_moveSX(), X_moveDX(), Y_moveUP(), Y_moveDW()*: muovono i servomotori di 10°
- *generatePWM*: genera un segnale PWM sul GPIO specificato di una certa durata.

La gestione del movimento dei servomotori avviene tramite la funzione *generatePWM()*. Questa funzione è responsabile della generazione di impulsi di durata variabile per ciascuna tipologia di rotazione. Le diverse durate degli impulsi corrispondono a diverse velocità, le quali sono state stabilite attraverso prove sperimentali, in modo tale che ogni impulso sposti il servomotore corrispondente di circa 10° gradi.

Tuttavia, nonostante questi sforzi, risulta praticamente impossibile determinare con precisione la posizione dei servomotori. Ciò è dovuto al fatto che le condizioni ambientali, come la temperatura e la tensione di alimentazione, non sono costanti, e queste variazioni influenzano il comportamento dei servomotori.

```
1  static void servo_init() {
2    x_angle = 0;
3    y_angle = 0;
4
5    pinMode(SERVO_X_PIN, OUTPUT);
6    pinMode(SERVO_Y_PIN, OUTPUT);
7
8    Serial.println("SRV| Successo inizializzazione");
9  }
10
11 void X_moveSx(){
12   generatePWM(SERVO_X_PIN, 1183);
13   x_angle += 10;
14   if(x_angle == 360){
15     x_angle = 0;
```

```
16    }
17    delay(100);
18  }
19
20  void X_moveDx(){
21    generatePWM(SERVO_X_PIN, 1840);
22
23    x_angle -= 10;
24    if(x_angle == -360){
25      x_angle = 0;
26    }
27    delay(100);
28  }
29
30  void Y_moveDown(){
31    if(y_angle >= 10){
32      generatePWM(SERVO_Y_PIN, 1270);
33      y_angle -= 10;
34      delay(100);
35    }
36  }
37
38  void Y_moveUp(){
39    if(y_angle <= 80){
40      generatePWM(SERVO_Y_PIN, 1780);
41      y_angle += 10;
42      delay(100);
43    }
44  }
45
46  void generatePWM(int servo, unsigned long pulseWidth) {
47    digitalWrite(servo, HIGH);
48    delayMicroseconds(pulseWidth);
```

```
49    digitalWrite(servo, LOW);
50    delayMicroseconds(20000- pulseWidth);
51  }
```