Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Elettronica

Progettazione di un sistema di calcolo del punteggio per flipper meccanico

Relatore
Matteo Meneghini

Laureando
Giacomo Urbani

Anno accademico
2022-2023

Data di laurea
17 luglio 2023

Abstract

Questa tesi si pone l'obiettivo di illustrare le fasi di progetto e di realizzazione di un sistema di rilevazione ed elaborazione dati, applicato al caso di un flipper meccanico.

Lo scopo di questo sistema é di calcolare in tempo reale il punteggio ottenuto dall'utente durante una partita e il numero di palline ancora disponibili prima della fine del gioco, rendendo elettronico un flipper meccanico di cinquant'anni fa.

Verrà fatta un'analisi degli obiettivi di progetto, proseguendo con una descrizione dei componenti utilizzati per la raccolta dei dati e l'esecuzione del calcolo del punteggio. Seguirà il listato del codice sorgente del programma, completo di commenti per spiegare la dinamica di funzionamento. Infine verrà riportata una prova di utilizzo del sistema.

Indice

1 Obiettivi

1.1 Idea

L'idea che anima il progetto é il recupero e la valorizzazione di un vecchio flipper meccanico, costruito a fine anni Sessanta, per renderlo sfruttabile come un odierno modello elettronico, tipico di bar e sale giochi. Per fare ciò é necessario ideare un sistema che renda automatica la raccolta delle informazioni utili al giocatore e facile la loro lettura durante il gioco.
Più precisamente, il traguardo che si intende raggiungere si divide in due parti.
La prima consiste nel riuscire a calcolare in tempo reale:

- il punteggio ottenuto dal giocatore durante una partita;
- il numero di palline ancora disponibili all'utente prima della fine del gioco.

Per poter conseguire questo risultato, il sistema da progettare deve essere in grado di percepire la biglia:

- nei punti del piano di gioco dove è possibile guadagnare punteggio, per poterlo sommare al totale già ottenuto;
- nel cassetto di raccolta palline, per poter rilevare l'errore del giocatore e diminuire il numero di palline ancora a disposizione prima del *game over*.

Il secondo obiettivo da raggiungere é la visualizzazione di questi dati su uno schermo, così da essere disponibili all'utente giocante. Anche questa funzionalità dovrà essere aggiornata in tempo reale, di pari passo con il calcolo dei valori.

1.2 Progettazione

Il giocattolo su cui si é basata l'ideazione del sistema é un flipper meccanico (figura 1.1).

Figura 1.1: Il flipper usato per il progetto

Nella concezione originale del prodotto, il giocatore deve riuscire a colpire la pallina in maniera tale da farla fermare in una delle dieci buche del piano da gioco, così da vincere il punteggio specifico della buca. Se durante la partita il giocatore non colpisce la biglia, che tende a ricadere verso il basso data la pendenza del campo da gioco, questa viene raccolta in un cassetto per evitare che esca dalla struttura.

Per rendere possibile il raggiungimento di più buche consecutive e non interrompere continuamente l'azione, é stata effettuata una modifica alla struttura coprendo le buche con del nastro adesivo: la biglia riesce così a rotolare su tutto il piano, rimanendo in gioco fino al primo errore dell'utente, momento in cui finirà nel cassetto di raccolta.

Per rilevare il passaggio sulle buche si é scelto di porre sotto ogni foro un sensore ad effetto Hall e di dotare il flipper di una pallina di materiale magnetico: sfruttando il principio fisico detto appunto

”Effetto Hall”, il campo magnetico generato dalla biglia in movimento sopra la buca andrà a sollecitare il sensore, che invierà un segnale al microcontrollore: ciò permetterà l’aggiunta della somma di punti caratteristica della buca al totale corrente. É stato posto un sensore anche a lato del cassetto di raccolta, così da poter segnalare la perdita della pallina e poter di conseguenza calcolare il numero di palline ancora disponibili.

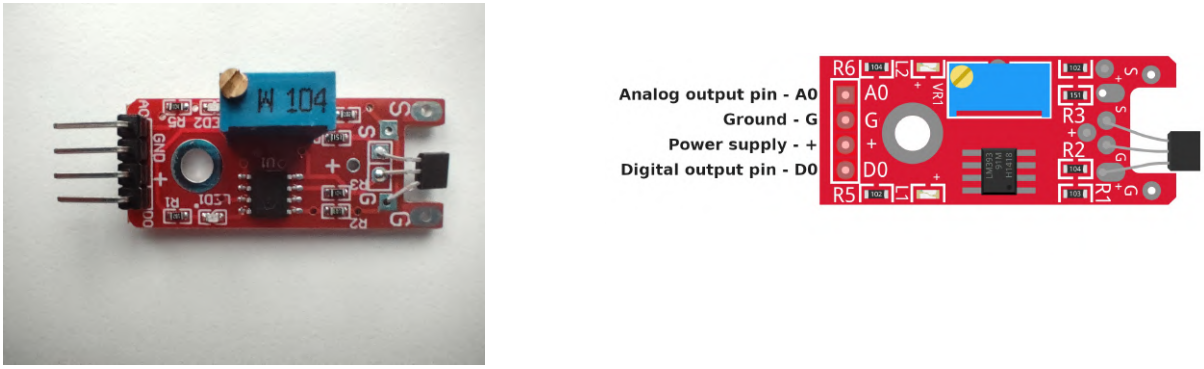
Per rilevare ed elaborare i dati é stata scelta la scheda Arduino Due, utilizzata durante il corso di ”Laboratorio di microelettronica” e adatta al progetto, grazie al semplice linguaggio di programmazione e alla grande disponibilità di pin a cui collegare i componenti di input ed output.

Infine per rendere disponibili all’utente i valori calcolati si é optato per uno schermo a cristalli liquidi TFT della Adafruit, già usato in laboratorio: la particolarità di questi display é la semplicità di utilizzo, grazie all’ottima libreria di istruzioni di cui sono corredati, perfettamente integrabile in Arduino.

2

Elenco dei componenti

2.1 Sensore ad effetto Hall KY-024



(a) Esemplare di sensore

(b) Piedinatura del sensore

Figura 2.1: Sensore ad effetto Hall KY-024

Il sensore serve a rilevare il passaggio della pallina sopra la buca, e sfrutta il principio dell'effetto Hall: la presenza di un campo magnetico perturba un circuito elettrico nel quale si formerà una differenza di potenziale ai capi, positiva o negativa in base al sistema di riferimento.
Il sensore possiede quattro piedini: due per l'alimentazione e due per le uscite, che forniscono segnali differenti:

- il pin Ao fornisce un segnale analogico di tensione, variabile fra 0 e 1024. In assenza di campo magnetico l'output fornisce un valore di circa 500: in base al polo magnetico che si avvicina al sensore, l'output varierà aumentando o diminuendo di conseguenza;

- il pin Do fornisce un segnale digitale di acceso/spento, in base al superamento di una certa soglia di tensione. Ciò limita l'accensione alla sola polarità del magnete che accresce la differenza di potenziale ai capi del sensore.

Regolando il potenziometro attraverso la vite é possibile modificare la soglia di sensibilità del sensore, che determina l'accensione del pin Do.
Sono presenti due LED: uno che si illumina quando il sensore è alimentato e uno che si illumina quando il pin Do fornisce segnale acceso.

Le specifiche tecniche del sensore sono le seguenti[1]:

Tensione di alimentazione	$2.7\ \text{V} \leq \text{V}_{on} \leq 6.5\ \text{V}$
Sensibilità	1.0 mV/G minima 1.4 mV/G tipica 1.75 mV/G massima
Output	digitale, ad alta sensibilità
Dimensioni	15×19 mm

2.2 Scheda Arduino Due

Il microcontrollore Arduino Due si basa su una CPU con architettura ARM a 32 bit, e rappresenta una versione più potente e versatile della nota scheda Arduino Uno.
La particolarità di questo modello é l'aumento del numero di pin disponibili, che hanno permesso di implementare molte funzionalità rispetto alla gamma di schede più basiche e di dedicare dei piedini specifici a certe interfacce di comunicazione. L'abbondanza di pin digitali é risultata fondamentale per poter collegare tutte le uscite digitali dei sensori e alcuni pin dello schermo, e la presenza di un'interfaccia SPI dedicata ha agevolato il collegamento fra Arduino e display.[2]

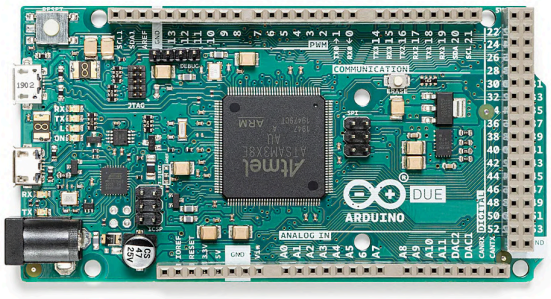


Figura 2.2: Esemplare di scheda

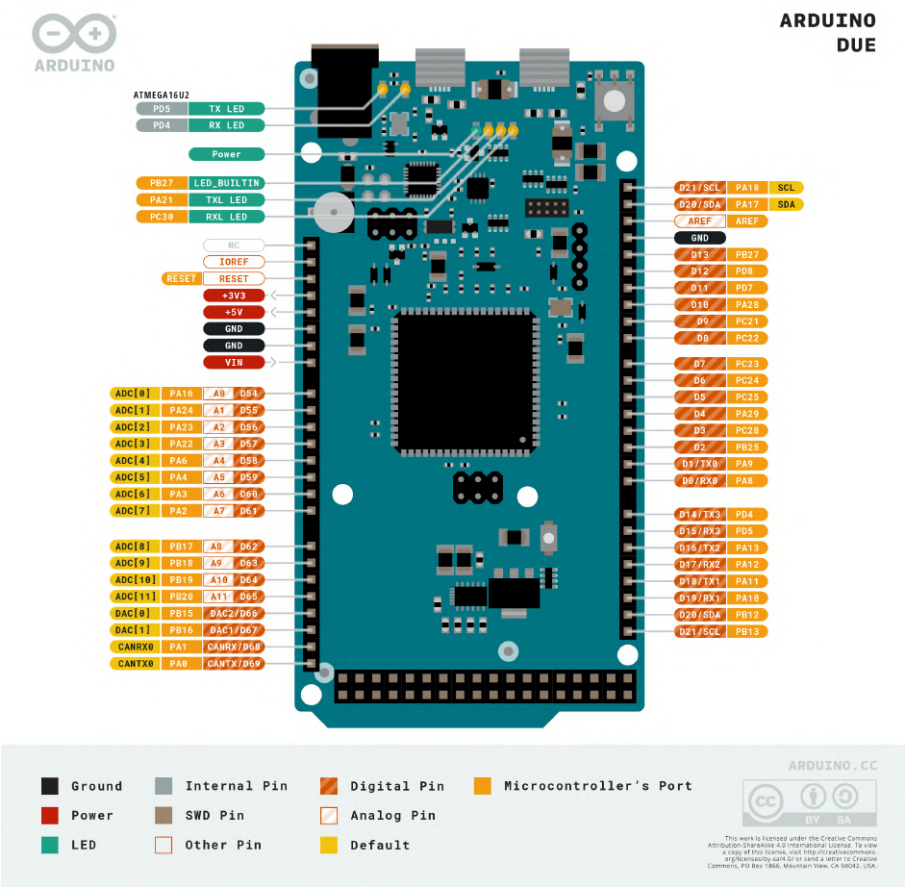


Figura 2.3: Piedinatura della scheda

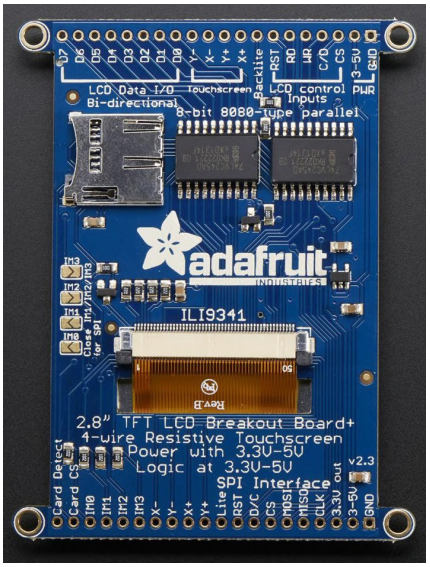
Le specifiche tecniche della scheda sono descritte nell'estratto del datasheet[3]:

Nome	Arduino® Due
N° pin digitali	54
N° pin analogici	12
N° pin con funzionalità PWM	12
Comunicazione I2C	Sì
Comunicazione SPI	Sì
Tensione ingresso/uscite	3.3V
Massima corrente entrante per pin	3 mA / 9 mA
Massima corrente entrante in tutte le linee I/o	130 mA
Processore	AT91SAM3X8E
Velocità di clock del processore	84 MHz
Memoria del processore	96KB SRAM, 512KB flash
Peso	36 g
Dimensioni	53.3 × 101.5mm

2.3 Schermo Adafruit Breakout v2

Evoluzione della prima serie, lo schermo Adafruit é uno schermo a colori touchscreen che offre due tipi di interfacce per collegarsi al microcontrollore: l'interfaccia SPI e l'interfaccia 8-bit.
La particolarità di questo schermo é il supporto nativo della scheda Arduino, grazie anche ad una libreria di funzioni che permette una facile programmazione dello schermo.

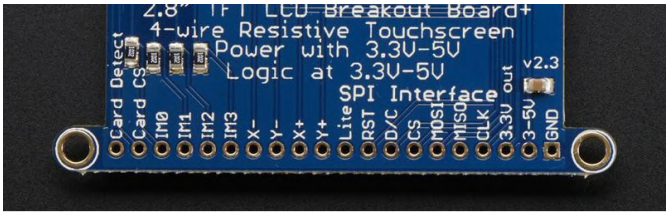
(a) Fronte display Adafruit



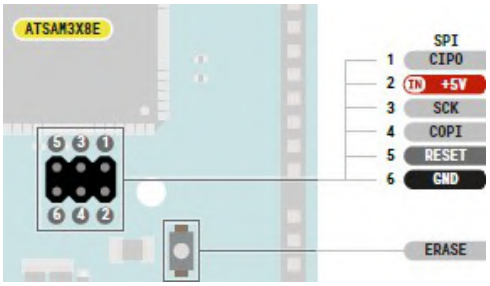
(b) Retro display Adafruit: la piedinatura superiore é per l'interfaccia 8-bit, la piedinatura inferiore é per l'interfaccia SPI

Figura 2.4: Display Adafruit

Per questo progetto é stata usata l'interfaccia SPI, che a fronte di una minore velocità di trasmissione dati rispetto alla 8-bit permette un cablaggio più semplice. Dovendo visualizzare dei dati numerici che non vengono aggiornati a velocità critiche per questo tipo di collegamento, la semplicità di connessione é stata determinante per la scelta.
Per poter scegliere fra le due modalità é necessario stagnare dei ponticelli sul retro dello schermo: nella figura 2.4b sono disposti a metà della lunghezza, sulla parte sinistra. Per usare la modalità SPI sono da saldare i jumper IM1, IM2, IM3 (i primi tre dall'alto).



(a) Piedinatura lato SPI schermo Adafruit



(b) Dettaglio della piedinatura di Arduino Due, riguardante l'interfaccia SPI

Figura 2.5: Interfacce di collegamento fra Arduino e display

La piedinatura per la connessione SPI del display é così composta, osservando la figura 2.5a da destra a sinistra:

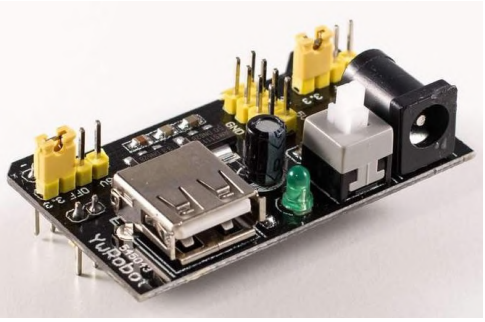
GND	collegamento a terra
3-5V	alimentazione del display, possibile sia a 3.3V sia a 5V
3.3V out	uscita alimentata a 3.3V
CLK	pin di clock, da collegare al pin SPI n°3 di Arduino Due
MISO	Master In Slave Out, pin di comunicazione da schermo a Arduino. Da collegare al pin SPI n°1 di Arduino Due (CIPO - Controller In Peripheral Out)
MOSI	Master Out Slave In, pin da comunicazione da Arduino a schermo. Da collegare al pin SPI n°4 di Arduino Due (COPI - Controller Out Peripheral In)
CS	Chip Select, da collegare ad un pin digitale di Arduino Due. Per questo progetto é stato scelto il pin n°46
D/S	Data and Command Selector, da collegare ad un pin digitale di Arduino Due. Per questo progetto é stato scelto il pin n°50

Le specifiche tecniche dello schermo sono descritte nell'estratto del datasheet[4]:

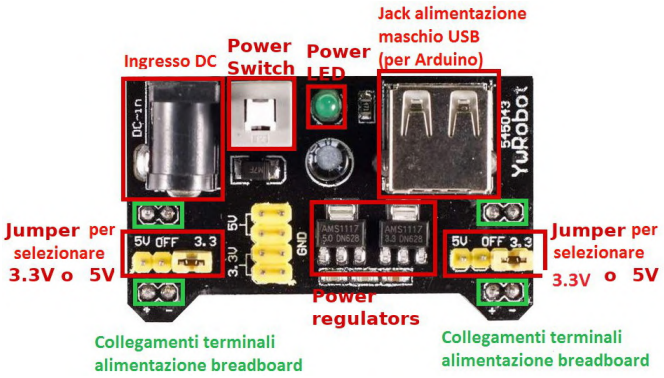
Dimensioni schermo	2.8" (43.2 × 57.6 mm)
Disposizione pixel	240 × 320 pixel RGB
Retroilluminazione	4 LED bianchi in parallelo
Touchscreen	Sì
Interfacce di collegamento	8-bit (12 pin), SPI (5 pin)
Circuito integrato controllo display	ILI9341
Tensione di alimentazione	3.3V oppure 5V

2.4 Alimentatore MB102

Data la necessità di alimentare 11 sensori e uno schermo, la corrente erogata dalla scheda Arduino Due non é sufficiente a garantire il funzionamento di tutti i componenti.
Per supplire a ciò, tutti componenti sono stati collegati a questo alimentatore, la cui tensione di uscita é regolabile per ciascuna delle due linee che alimenta.[5]



(a) Esemplare di alimentatore



(b) Dettaglio del componente

Tensione di alimentazione	6.5V ÷ 12V
Tensione erogabile	0V, 3.3V, 5V (regolabile tramite ponticelli)
Numero di uscite	2, indipendenti
Massima corrente erogabile	700 mA
Dimensioni	53 × 33 mm

2.5 Breadboard

Componente necessario per permettere dei collegamenti veloci e modificabili fra alimentatore, Arduino, schermo e sensori.

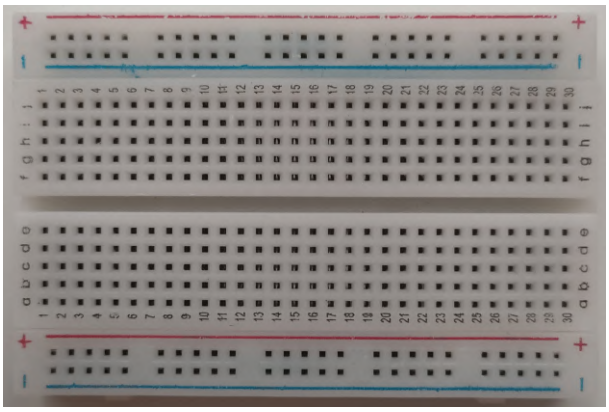


Figura 2.7: Breadboard usata nel progetto

Terminali di alimentazione	100
Terminali di distribuzione	300
Tensione massima	300 V
Corrente massima	3 A

2.6 Cavi

Cavi di 3 tipologie: maschio-maschio, maschio-femmina e femmina-femmina, per permettere qualunque tipo di collegamento ed eventuali prolunghe.

Figura 2.8: Cavi usati nel progetto

3 Codice

Il codice sviluppato avrà le seguenti funzioni:

- inizializzazione dei dispositivi, associando la scheda Arduino al resto dell'hardware che compone il sistema;
- inizializzazione delle variabili necessarie alla gestione della partita;
- gestione degli input derivanti dai sensori, che dovranno essere separati fra input validi e falsi positivi derivanti dal rumore di fondo del segnale;
- elaborazione degli input validi, per stampare a schermo l'andamento della partita;
- controllo delle condizioni necessarie per il proseguimento del gioco, per interrompere la partita quando non sono più disponibili palline.

Il flusso del programma é riassunto nel diagramma in fig.3.1 :

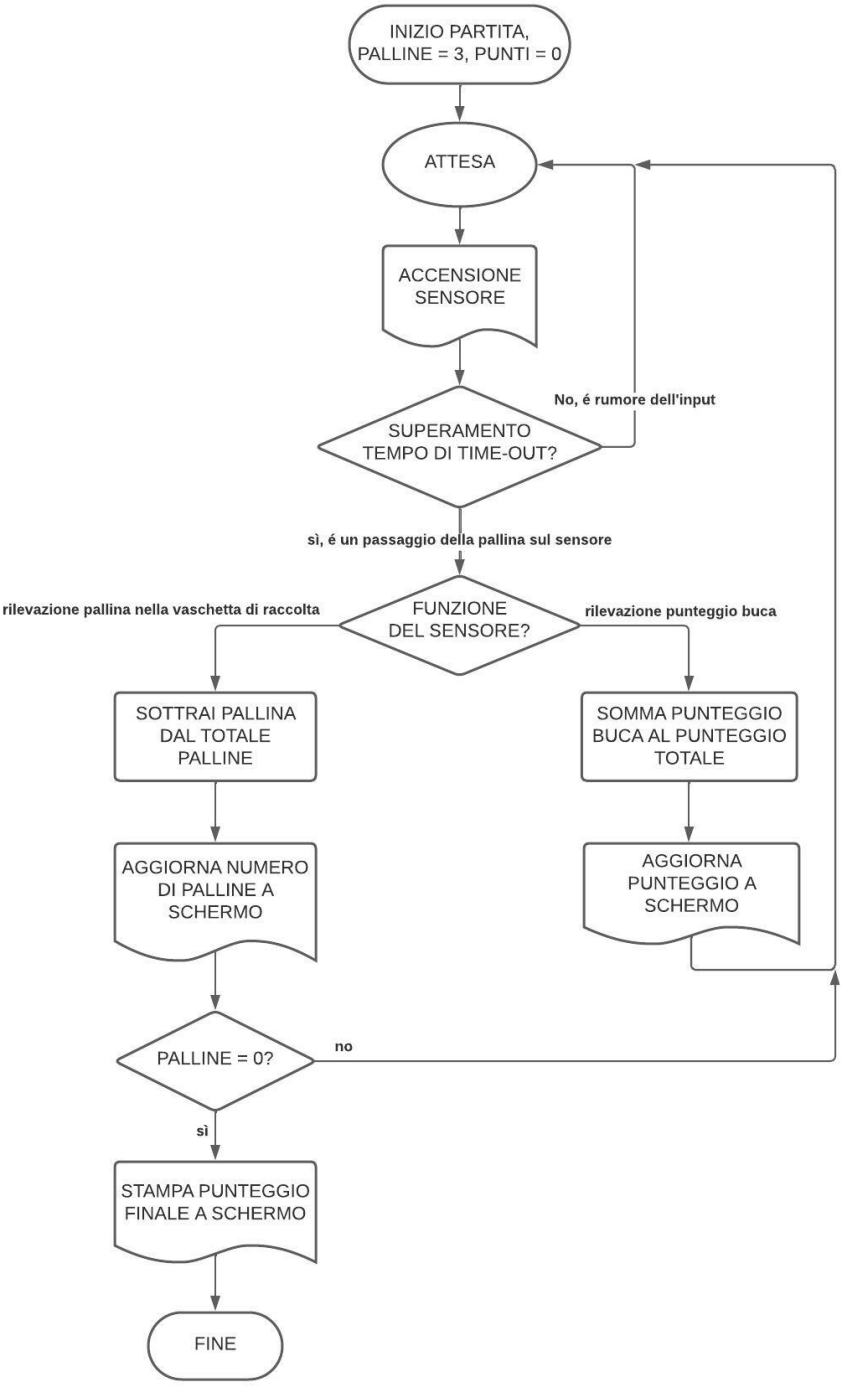


Figura 3.1: Diagramma di flusso del programma

La prima parte del programma é adibita alla definizione dei pin della scheda Arduino collegati alle uscite dei sensori e alla definizione dei principali colori rappresentabili sullo schermo Adafruit: in questo modo non sarà necessario ricordare il codice dello standard 565 abbinato al tal colore, ma la sua più pratica definizione in linguaggio naturale.

```
1     #define HALL_D_PIN_A 13
2     #define HALL_D_PIN_B 11
3     #define HALL_D_PIN_C 9
4     #define HALL_D_PIN_D 7
5     #define HALL_D_PIN_E 5
6     #define HALL_D_PIN_F 3
7     #define HALL_D_PIN_G 22
8     #define HALL_D_PIN_H 27
9     #define HALL_D_PIN_I 30
10    #define HALL_D_PIN_J 35
11    #define HALL_D_PIN_K 38
12
13    #define ILI9341_BLACK 0x0000
14    #define ILI9341_BLUE 0x001F
15    #define ILI9341_RED 0xF800
16    #define ILI9341_GREEN 0x07E0
17    #define ILI9341_CYAN 0x01FF
18    #define ILI9341_MAGENTA 0xFB1F
19    #define ILI9341_YELLOW 0xFFE0
20    #define ILI9341_WHITE 0xFFFF
```

I sensori hanno una lettera diversa per ciascuno, data la varietà di funzioni e di punteggi che contraddistingue ognuno: la legenda é disponibile a fig.3.2.

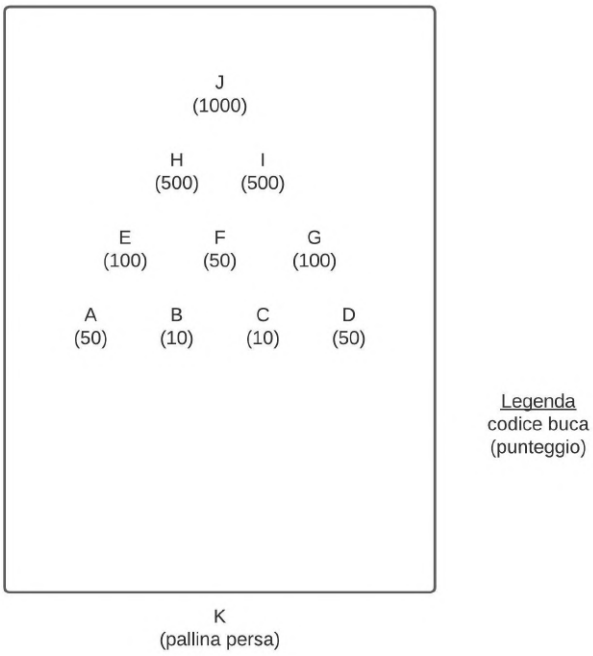


Figura 3.2: Codici delle buche usati nel programma

Segue la chiamata alle librerie necessarie alla comunicazione SPI display - Arduino e per la visualizzazione di elementi a schermo, la definizione di ulteriori pin di comunicazione esterni al socket SPI di Arduino (CS - chip select e DC - data selector) e la conseguente creazione dell'oggetto display necessario per la comunicazione.

```
1  #include "SPI.h"
2  #include "Adafruit_GFX.h"
3  #include "Adafruit_ILI9341.h"
4
5  #define TFT_CS 46
6  #define TFT_DC 50
7
8  Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
```

Si inizializzano ora le variabili globali che verranno usate durante la partita:

- il punteggio totale, con annessa variabile temporanea per l'aggiornamento del valore a schermo;

- il totale delle palline giocabili, con annessa variabile temporanea per l'aggiornamento del valore a schermo;
- il numero di millisecondi di input continuativo da un sensore per non essere un falso segnale d'ingresso.

```
int tot = 0;
int old_tot = tot;
int ball_num = 3;
int old_ball_num = ball_num;
int sens_time = 30;
```

Inizia ora la fase di setup in avvio di partita, in cui tutti i pin dei sensori definiti sopra vengono inizializzati come input e viene inizializzata la comunicazione seriale per messaggi di debug:

```
void setup() {
    pinMode(HALL_D_PIN_A, INPUT);
    pinMode(HALL_D_PIN_B, INPUT);
    pinMode(HALL_D_PIN_C, INPUT);
    pinMode(HALL_D_PIN_D, INPUT);
    pinMode(HALL_D_PIN_E, INPUT);
    pinMode(HALL_D_PIN_F, INPUT);
    pinMode(HALL_D_PIN_G, INPUT);
    pinMode(HALL_D_PIN_H, INPUT);
    pinMode(HALL_D_PIN_I, INPUT);
    pinMode(HALL_D_PIN_J, INPUT);
    pinMode(HALL_D_PIN_K, INPUT);
    Serial.begin(9600);
```

Si prosegue con la scrittura dei dati a schermo.
Dopo aver aperto la comunicazione SPI si definisce l'orientamento dello schermo, in questo caso orizzontale, e si colora lo schermo completamente di nero per creare lo sfondo.

```
    tft.begin();
    tft.setRotation (1);
    tft.fillScreen(ILI9341_BLACK);
```

Poi si vanno a scrivere le parole e le variabili, seguendo sempre lo stesso ordine:

- posizionamento del cursore;
- impostazione del colore, sfruttando le definizioni scritte all'inizio;
- impostazione della dimensione del carattere;
- scrittura della parola/variabile.

```
        tft.setCursor(5,5);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(3);
        tft.println("Punteggio");

        tft.setCursor(5,40);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(6);
        tft.println(tot);

        tft.setCursor(5,125);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(3);
        tft.println("Palline rimanenti");

        tft.setCursor(5,160);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(6);
        tft.println(ball_num);

    }
```

Concluso il setup, gli ingressi sono inizializzati e lo schermo é inizializzato con zero punti e il numero di palline definito con la variabile `ball_num`.
Inizia quindi il ciclo di loop, che verifica ininterrottamente lo stato di tutti i sensori in sequenza: quando un sensore rileva la pallina magnetica, si entra nell'if corrispondente e si inizia il conteggio del tempo. Se il sensore non rileva più la pallina entro il tempo stabilito con la variabile `sens_time`,

allora non é da considerare come segnale valido e si esce dall'if. Se invece il sensore rileva la pallina per il tempo impostato si aggiorna la variabile punteggio in base al sensore su cui sta passando la pallina. I sensori A-J deputati al calcolo del punteggio hanno tutti un caso if molto simile, ad eccezione del punteggio che attribuiscono in base alla buca sotto cui sono posti.

```
1     void loop() {
2         if(digitalRead(HALL_D_PIN_A)==HIGH){
3             int ball_time = millis();
4             while((millis()-ball_time)<=sens_time){
5                 if(digitalRead(HALL_D_PIN_A)!=HIGH){
6                 break;
7                 }
8             }
9             if((millis()-ball_time)>=sens_time){
10                tot += 50;
11                Serial.println(tot);
12            }
13        }
14        if(digitalRead(HALL_D_PIN_B)==HIGH){
15            int ball_time = millis();
16            while((millis()-ball_time)<=sens_time){
17                if(digitalRead(HALL_D_PIN_B)!=HIGH){
18                    break;
19                }
20            }
21            if((millis()-ball_time)>=sens_time){
22                tot += 10;
23                Serial.println(tot);
24            }
25        }
26        if(digitalRead(HALL_D_PIN_C)==HIGH){
27            int ball_time = millis();
28            while((millis()-ball_time)<=sens_time){
29                if(digitalRead(HALL_D_PIN_C)!=HIGH){
30                    break;
```

```
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 10;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_D)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
                if(digitalRead(HALL_D_PIN_D)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 50;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_E)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
                if(digitalRead(HALL_D_PIN_E)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 100;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_F)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
```

```
                if(digitalRead(HALL_D_PIN_F)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 50;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_G)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
                if(digitalRead(HALL_D_PIN_G)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 100;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_H)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
                if(digitalRead(HALL_D_PIN_H)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 500;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_I)==HIGH){
```

```
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
                if(digitalRead(HALL_D_PIN_I)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 500;
                Serial.println(tot);
            }
        }
        if(digitalRead(HALL_D_PIN_J)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=sens_time){
                if(digitalRead(HALL_D_PIN_J)!=HIGH){
                    break;
                }
            }
            if((millis()-ball_time)>=sens_time){
                tot += 1000;
               Serial.println(tot);
            }
        }
```

Il sensore K, che rileva la perdita della pallina nella vaschetta in fondo al piano di gioco, ha un caso if diverso. Se la pallina smette di essere rilevata per un tempo pari al doppio della variabile `sens_time`, si esce dall'if: il tempo é aumentato rispetto agli altri sensori per evitare rilevamenti multipli, nel caso la pallina rimbalzi all'interno del cassetto prima di fermarsi. Se la pallina viene rilevata per il tempo necessario, allora viene considerata persa e viene scalata dal conteggio delle palline ancora disponibili. Il loop finale é inserito per fermare momentaneamente il gioco, fino a quando il giocatore non rimuove la pallina dalla vaschetta.

```
        if(digitalRead(HALL_D_PIN_K)==HIGH){
            int ball_time = millis();
            while((millis()-ball_time)<=(sens_time*2)){
```

```
            if(digitalRead(HALL_D_PIN_K)!=HIGH){
                break;
            }
        }
        if((millis()-ball_time)>=(sens_time*2)){
            Serial.println("pallina persa");
            ball_num--;
        }
        while(true){
            if(digitalRead(HALL_D_PIN_K)!=HIGH){
                break;
            }
        }
    }
```

Terminata la sequenza di controllo di input, si procede a confrontare i valori di punteggio e palline rimanenti con quelli dell'iterazione precedente: se sono diversi vengono aggiornati i valori e scritti a schermo, mediante l'apposizione di una banda nera per ripristinare lo sfondo e la successiva scrittura del nuovo numero.

```
    if(tot != old_tot){
        old_tot = tot;
        tft.fillRect(5,40,315,80,ILI9341_BLACK);
        tft.setCursor(5,40);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(6);
        tft.println(tot);
    }
    if(old_ball_num != ball_num){
        old_ball_num = ball_num;
        tft.fillRect(5,160,315,80,ILI9341_BLACK);
        tft.setCursor(5,160);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(6);
        tft.println(ball_num);
```

```
16          }
```

La condizione finale verifica le condizioni per il proseguimento del gioco, dopo l'ultimo aggiornamento dei valori: se il numero di palline giocabili è sceso a zero ferma il loop, scrive un messaggio rivolto al giocatore con le istruzioni per iniziare una nuova partita e il punteggio finale che ha ottenuto. Entra poi in un loop infinito, in attesa del reset da parte del giocatore.

```
1           if(ball_num==0) {
2               Serial.println("Palline esaurite, premi sull'Arduino il
                    pulsante RESET vicino alla presa USB per cominciare una
                    nuova partita!");
3               Serial.print("Il tuo punteggio totale e' : ");
4               Serial.println(tot);
5
6               tft.fillScreen(ILI9341_BLACK);
7               tft.setCursor(0,5);
8               tft.setTextColor(ILI9341_WHITE);
9               tft.setTextSize(2);
10              tft.println("Palline esaurite!");
11              tft.println("Premi sull'Arduino");
12              tft.println("il pulsante RESET");
13              tft.println("(vicino alla presa USB)");
14              tft.println("per cominciare");
15              tft.println("una nuova partita!");
16              tft.println();
17              tft.println("Punteggio totale");
18              tft.println();
19              tft.setTextSize(6);
20              tft.println(tot);
21
22              while(true){}
23          }
24      }
```

Il codice completo é riportato di seguito.

```
#define HALL_D_PIN_A 13
#define HALL_D_PIN_B 11
#define HALL_D_PIN_C 9
#define HALL_D_PIN_D 7
#define HALL_D_PIN_E 5
#define HALL_D_PIN_F 3
#define HALL_D_PIN_G 22
#define HALL_D_PIN_H 27
#define HALL_D_PIN_I 30
#define HALL_D_PIN_J 35
#define HALL_D_PIN_K 38

#define ILI9341_BLACK 0x0000
#define ILI9341_BLUE 0x001F
#define ILI9341_RED 0xF800
#define ILI9341_GREEN 0x07E0
#define ILI9341_CYAN 0x01FF
#define ILI9341_MAGENTA 0xFB1F
#define ILI9341_YELLOW 0xFFE0
#define ILI9341_WHITE 0xFFFF

#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

#define TFT_CS 46
#define TFT_DC 50

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

int tot = 0;
int old_tot = tot;
int ball_num = 3;
```

```
34      int old_ball_num = ball_num;
35      int sens_time = 30;
36
37      void setup() {
38          pinMode(HALL_D_PIN_A, INPUT);
39          pinMode(HALL_D_PIN_B, INPUT);
40          pinMode(HALL_D_PIN_C, INPUT);
41          pinMode(HALL_D_PIN_D, INPUT);
42          pinMode(HALL_D_PIN_E, INPUT);
43          pinMode(HALL_D_PIN_F, INPUT);
44          pinMode(HALL_D_PIN_G, INPUT);
45          pinMode(HALL_D_PIN_H, INPUT);
46          pinMode(HALL_D_PIN_I, INPUT);
47          pinMode(HALL_D_PIN_J, INPUT);
48          pinMode(HALL_D_PIN_K, INPUT);
49
50          Serial.begin(9600);
51
52          tft.begin();
53          tft.setRotation (1);
54          tft.fillScreen(ILI9341_BLACK);
55
56          tft.setCursor(5,5);
57          tft.setTextColor(ILI9341_WHITE);
58          tft.setTextSize(3);
59          tft.println("Punteggio");
60
61          tft.setCursor(5,40);
62          tft.setTextColor(ILI9341_WHITE);
63          tft.setTextSize(6);
64          tft.println(tot);
65
66          tft.setCursor(5,125);
67          tft.setTextColor(ILI9341_WHITE);
```

```
68          tft.setTextSize(3);
69          tft.println("Palline rimanenti");
70
71          tft.setCursor(5,160);
72          tft.setTextColor(ILI9341_WHITE);
73          tft.setTextSize(6);
74          tft.println(ball_num);
75
76      }
77
78      void loop() {
79          if(digitalRead(HALL_D_PIN_A)==HIGH){
80              int ball_time = millis();
81              while((millis()-ball_time)<=sens_time){
82                  if(digitalRead(HALL_D_PIN_A)!=HIGH){
83                  break;
84                  }
85              }
86              if((millis()-ball_time)>=sens_time){
87                  tot += 50;
88                  Serial.println(tot);
89              }
90          }
91          if(digitalRead(HALL_D_PIN_B)==HIGH){
92              int ball_time = millis();
93              while((millis()-ball_time)<=sens_time){
94                  if(digitalRead(HALL_D_PIN_B)!=HIGH){
95                      break;
96                  }
97              }
98              if((millis()-ball_time)>=sens_time){
99                  tot += 10;
100                 Serial.println(tot);
101             }
```

```
102         }
103         if(digitalRead(HALL_D_PIN_C)==HIGH){
104             int ball_time = millis();
105             while((millis()-ball_time)<=sens_time){
106                 if(digitalRead(HALL_D_PIN_C)!=HIGH){
107                     break;
108                 }
109             }
110             if((millis()-ball_time)>=sens_time){
111                 tot += 10;
112                 Serial.println(tot);
113             }
114         }
115         if(digitalRead(HALL_D_PIN_D)==HIGH){
116             int ball_time = millis();
117             while((millis()-ball_time)<=sens_time){
118                 if(digitalRead(HALL_D_PIN_D)!=HIGH){
119                     break;
120                 }
121             }
122             if((millis()-ball_time)>=sens_time){
123                 tot += 50;
124                 Serial.println(tot);
125             }
126         }
127         if(digitalRead(HALL_D_PIN_E)==HIGH){
128             int ball_time = millis();
129             while((millis()-ball_time)<=sens_time){
130                 if(digitalRead(HALL_D_PIN_E)!=HIGH){
131                     break;
132                 }
133             }
134             if((millis()-ball_time)>=sens_time){
135                 tot += 100;
```

```
136                 Serial.println(tot);
137             }
138         }
139         if(digitalRead(HALL_D_PIN_F)==HIGH){
140             int ball_time = millis();
141             while((millis()-ball_time)<=sens_time){
142                 if(digitalRead(HALL_D_PIN_F)!=HIGH){
143                     break;
144                 }
145             }
146             if((millis()-ball_time)>=sens_time){
147                 tot += 50;
148                 Serial.println(tot);
149             }
150         }
151         if(digitalRead(HALL_D_PIN_G)==HIGH){
152             int ball_time = millis();
153             while((millis()-ball_time)<=sens_time){
154                 if(digitalRead(HALL_D_PIN_G)!=HIGH){
155                     break;
156                 }
157             }
158             if((millis()-ball_time)>=sens_time){
159                 tot += 100;
160                 Serial.println(tot);
161             }
162         }
163         if(digitalRead(HALL_D_PIN_H)==HIGH){
164             int ball_time = millis();
165             while((millis()-ball_time)<=sens_time){
166                 if(digitalRead(HALL_D_PIN_H)!=HIGH){
167                     break;
168                 }
169             }
```

```
170             if((millis()-ball_time)>=sens_time){
171                 tot += 500;
172                 Serial.println(tot);
173             }
174         }
175         if(digitalRead(HALL_D_PIN_I)==HIGH){
176             int ball_time = millis();
177             while((millis()-ball_time)<=sens_time){
178                 if(digitalRead(HALL_D_PIN_I)!=HIGH){
179                     break;
180                 }
181             }
182             if((millis()-ball_time)>=sens_time){
183                 tot += 500;
184                 Serial.println(tot);
185             }
186         }
187         if(digitalRead(HALL_D_PIN_J)==HIGH){
188             int ball_time = millis();
189             while((millis()-ball_time)<=sens_time){
190                 if(digitalRead(HALL_D_PIN_J)!=HIGH){
191                     break;
192                 }
193             }
194             if((millis()-ball_time)>=sens_time){
195                 tot += 1000;
196                Serial.println(tot);
197             }
198         }
199         if(digitalRead(HALL_D_PIN_K)==HIGH){
200             int ball_time = millis();
201             while((millis()-ball_time)<=(sens_time*2)){
202                 if(digitalRead(HALL_D_PIN_K)!=HIGH){
203                     break;
```

```
204                 }
205             }
206             if((millis()-ball_time)>=(sens_time*2)){
207                 Serial.println("pallina persa");
208                 ball_num--;
209             }
210             while(true){
211                 if(digitalRead(HALL_D_PIN_K)!=HIGH){
212                     break;
213                 }
214             }
215         }
216 
217         if(tot != old_tot){
218             old_tot = tot;
219             tft.fillRect(5,40,315,80,ILI9341_BLACK);
220             tft.setCursor(5,40);
221             tft.setTextColor(ILI9341_WHITE);
222             tft.setTextSize(6);
223             tft.println(tot);
224         }
225         if(old_ball_num != ball_num){
226             old_ball_num = ball_num;
227             tft.fillRect(5,160,315,80,ILI9341_BLACK);
228             tft.setCursor(5,160);
229             tft.setTextColor(ILI9341_WHITE);
230             tft.setTextSize(6);
231             tft.println(ball_num);
232         }
233 
234        if(ball_num==0) {
235             Serial.println("Palline esaurite, premi sull'Arduino il
                pulsante RESET vicino alla presa USB per cominciare una
                nuova partita!");
```

```
236             Serial.print("Il tuo punteggio totale e' : ");
237             Serial.println(tot);
238 
239             tft.fillScreen(ILI9341_BLACK);
240             tft.setCursor(0,5);
241             tft.setTextColor(ILI9341_WHITE);
242             tft.setTextSize(2);
243             tft.println("Palline esaurite!");
244             tft.println("Premi sull'Arduino");
245             tft.println("il pulsante RESET");
246             tft.println("(vicino alla presa USB)");
247             tft.println("per cominciare");
248             tft.println("una nuova partita!");
249             tft.println();
250             tft.println("Punteggio totale");
251             tft.println();
252             tft.setTextSize(6);
253             tft.println(tot);
254 
255             while(true){}
256         }
257     }
```

4

Verifica di funzionamento

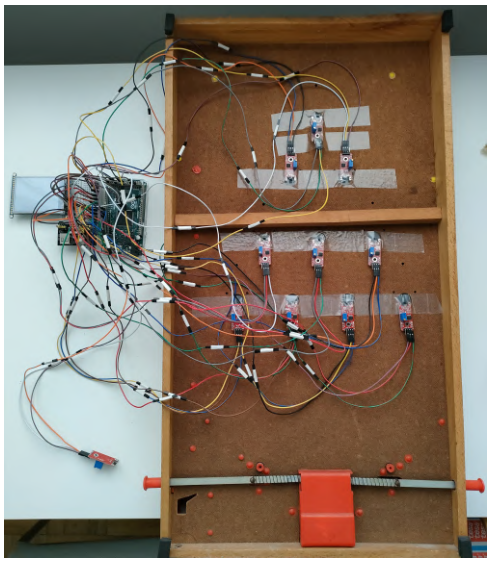
4.1 Risultato costruttivo

Ogni sensore é stato attaccato al lato inferiore del piano di gioco con dello scotch, e da ogni componente si diramano tre collegamenti: due all'alimentazione e uno al pin digitale Do di Arduino Due. Dal microcontrollore i 5 cavi necessari alla comunicazione con lo schermo si collegano alla breadboard, su cui é inserito il display.

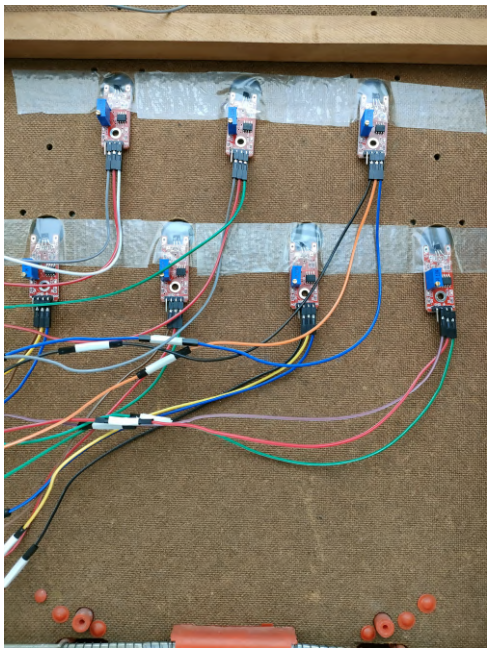
(a) Fronte flipper

(b) Lato flipper: particolare del sensore per il conteggio delle palline rimanenti

Figura 4.1

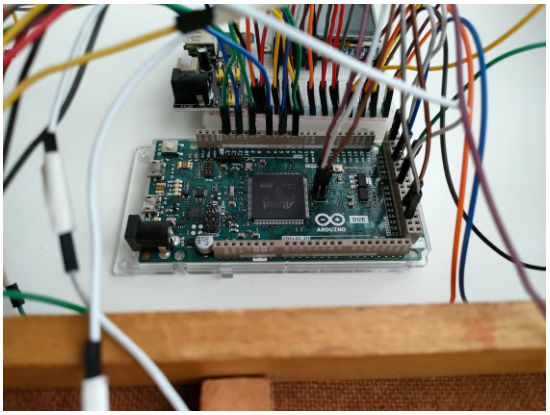


(a) Retro flipper: la parte di rilevazione del sensore é posizionata in corrispondenza della buca

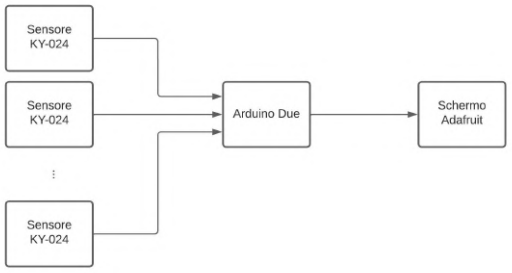


(b) Particolare della sistemazione dei sensori

Figura 4.2



(a) Particolare della scheda Arduino, con il collegamento SPI al display



(b) Schema a blocchi del circuito

Figura 4.3

4.2 Dinamica di gioco

All'accensione l'Arduino visualizza a schermo zero punti e tre palline rimanenti, come mostrato in figura 4.4.

Figura 4.4: Situazione all'accensione del microcontrollore

Iniziando a giocare si nota l'aumento progressivo del punteggio in base alle buche su cui passa la pallina, come visibile in fig. 4.5.

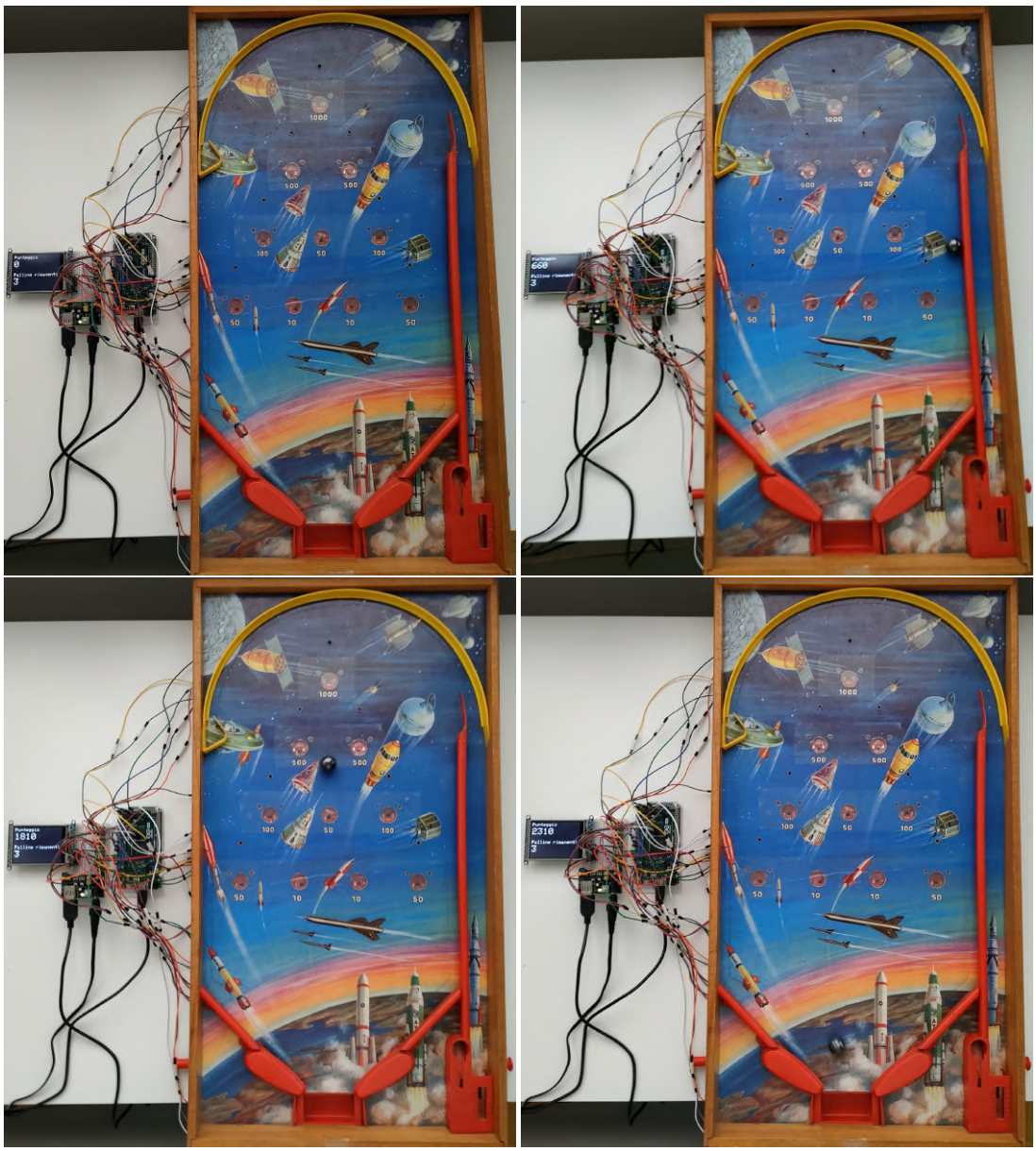


Figura 4.5: Incremento del punteggio durante il gioco

Nel caso di perdita della pallina, il sensore posto sul lato del cassetto di raccolta farà diminuire il totale delle palline rimanenti. Il sistema rimarrà in attesa finché il giocatore non rimetterà in gioco la biglia, momento nel quale il programma ricomincerà a calcolare il punteggio.

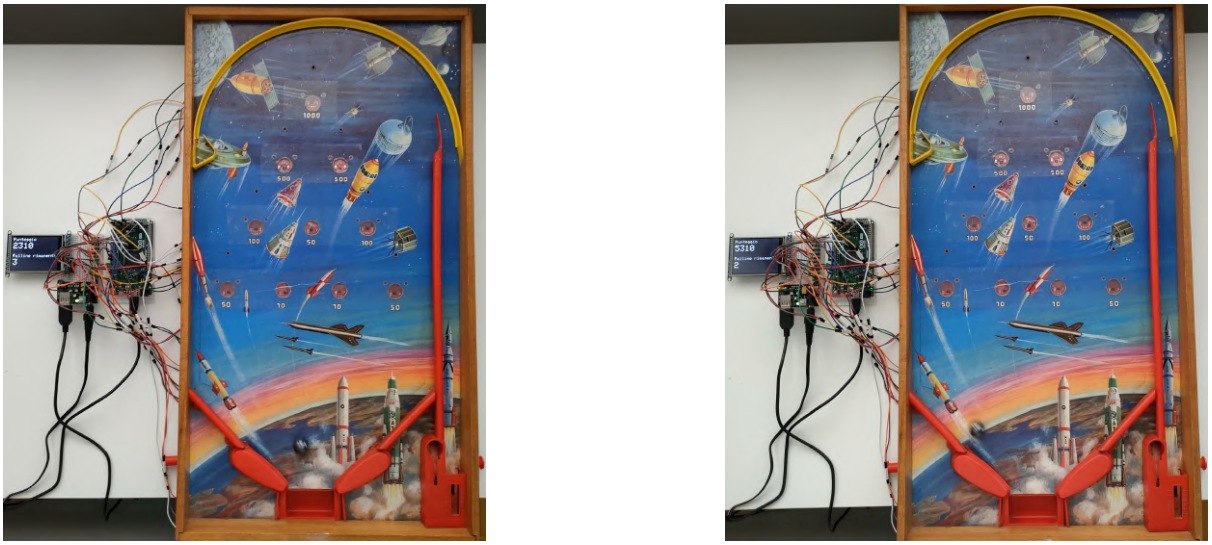


Figura 4.6: Fasi di gioco, con aumento del punteggio e diminuzione delle palline

Il gioco continua così fino alla perdita dell'ultima pallina rimanente: il programma fermerà la rilevazione dei sensori, mostrerà a schermo il punteggio totale ottenuto durante la partita e le istruzioni necessarie al giocatore per iniziare una nuova partita.

Figura 4.7: Situazione a fine partita: nell'immagine di destra si può notare la comparsa del messaggio rivolto al giocatore

5
Conclusioni

Con questa tesi é stata descritta la progettazione e realizzazione di un sistema elettronico di rilevazione ed elaborazione dati, impiegato ai fini del calcolo di un punteggio di una partita di flipper.
Si é iniziato spiegando l'idea che ha fatto nascere il progetto, proseguendo con la presentazione degli obiettivi di funzionamento del sistema.
Si é poi dato spazio alla progettazione del sistema, spiegando i passaggi che hanno condotto alla scelta del meccanismo di rilevazione, del microcontrollore per la gestione del sistema e dello schermo per la visualizzazione dei dati.
Un capitolo é stato dedicato alla spiegazione dei vari componenti usati nel progetto, con l'aggiunta di foto, schemi e tabelle dati per chiarire la loro funzione e le loro caratteristiche tecniche.
Il codice che controlla le funzioni di rilevazione e calcolo é stato riportato interamente, con descrizioni per ogni parte di programma.
Si é concluso questa dissertazione con una descrizione fotografica del prodotto finito ed una sua prova di funzionamento, per dimostrare il raggiungimento degli obiettivi prefissati inizialmente.

Bibliografia

[1] AZ-Delivery, *Guida all'uso del sensore ad effetto Hall KY-024*.

[2] M. Meneghini, *Slide del corso di Laboratorio di microelettronica*, 2022.

[3] Arduino, *Sito web di documentazione Arduino,* https://docs.arduino.cc/hardware/due.

[4] Adafruit, *Guida all'uso dello schermo Adafruit 2.8" breakout v2*.

[5] AZ-Delivery, *Guida all'uso dell'alimentatore MB102*.