UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA



Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Sistema di visione per la valutazione dell'entrata in acqua di un nuotatore nell'ambito del progetto SWIM4ALL

Vision system for the evaluation of the entry into the water of a swimmer within the SWIM4ALL project

Relatore: Ch.mo Prof. Mancini Adriano Laureando: Mori Nicola

Anno Accademico 2019-2020

Indice

El	enco	delle figure	4				
1	Intr	roduzione	5				
	1.1	Il progetto SWIM4ALL	5				
	1.2	Obiettivo	5				
	1.3	Struttura della tesi	6				
2	Stru	umenti Utilizzati	7				
	2.1	IDS UI-3160CP Rev. 2.1	7				
	2.2	Arduino MKR WAN 1300	8				
	2.3	Modello Client-Server	9				
		2.3.1 Client	10				
		2.3.2 Server	12				
	2.4	Linguaggi Utilizzati	14				
		$2.4.1 C/C++ \dots \dots \dots \dots \dots \dots \dots \dots \dots $	14				
		2.4.2 Pyhton	14				
		2.4.3 HTML	14				
		2.4.4 CSS	15				
		2.4.5 JavaScript	15				
	2.5	Editor di sviluppo	16				
		2.5.1 Visual Studio Code	16				
		2.5.2 Arduino IDE	17				
3	Sviluppo 18						
	3.1	Sviluppo codice Arduino	20				
		3.1.1 setup()	20				
		3.1.2 loop()	20				
		$3.1.3 \text{ startREC}() \dots \dots$	21				
		3.1.4 startNImpulsi()	23				
		3.1.5 recvWithStartEndMarkers()	24				
		$3.1.6 \text{parseData}() \ldots \ldots \vdots \ldots \ldots \ldots \ldots \ldots \ldots \ldots$	25				
	3.2	Sviluppo codice HTML	26				
		3.2.1 Gestione video	26				

INDICE

3.2.2 Form per la gestione di Arduino
Sviluppo codice CSS
Sviluppo codice JavaScript
3.4.1 Sincronizzazione video
3.4.2 Gestione modalità START/START N
Sviluppo codice Python
3.5.1 app.py
3.5.2 Arduino()
$3.5.3 \text{IlMioThread}() \dots \dots$
Generazione video
3.6.1 makevideo(path,framerate_num)
normaica dell'applicazione web 43
Gestione Video
Form Arduino
nclusioni 46
Miglioramenti

Capitolo 1 Introduzione

1.1 Il progetto SWIM4ALL

Il progetto SWIM4ALL fornisce un servizio per tutti coloro che lottano per inserirsi nelle strutture per imparare a nuotare nei centri regolari. Nasce dall'idea del comune di Fermo in collaborazione con l'Università Politecnica delle Marche, per aiutare ragazzi autistici ad approcciarsi al nuoto e all'ambiente acquatico. É un peccato che quindi non abbiano la possibilità di padroneggiare un'abilità in

L'un peccato che quindi non abbiano la possibilità di padroneggiare un abilità in un ambiente che dà loro grande piacere, ma presenta anche un grande pericolo, poiché molti bambini autistici non hanno paura presentando scarso controllo degli impulsi e quindi entrano in acqua senza capacità di prevenire l'annegamento. Un progetto molto simile lo portò avanti Natalie Clarke che decise di combinare anni di esperienza dovuta alla sua carriera di nuotatrice e anni di convivenza con il suo bambino, affetto dalla sindrome di Asperger, per cercare di aiutare questi ragazzi nelle piscine. Natalie inizio con un paio di nuotatori nella piscina della scuola elementare della zona, per poi aumentare il numero di iscritti del programma fino a cento in soli due anni.

La maggior parte dei nuotatori ha una diagnosi di autismo, però sono anche coinvolti ragazzi con sindrome di Down e altri tipi di disabilità.

1.2 Obiettivo

Il nostro obiettivo è quello di progettare un sistema di visione per la valutazione dell'entrata in acqua di un nuotatore. Attraverso un'applicazione Web potremmo monitorare le diverse registrazioni che verranno effettuate; ovvero decidere se iniziare la registrazione o fermarla e successivamente visionare i diversi video che verranno prodotti dal sistema. La fase di entrata in acqua di un ragazzo autistico è la fase più complicata e quindi anche quella alla quale prestare maggiore attenzione; il nostro sistema di visione avrà proprio come obiettivo quello di svolgere al meglio questo compito in maniera tale da riuscire ad analizzare, anche a posteriori, i movimenti dei nuotatori e le diverse fasi durante l'immersione in acqua.

1.3 Struttura della tesi

La tesi sarà strutturata in maniera tale da mostrare, inizialmente, quali strumenti hardware compongono il sistema, analizzando le loro caratteristiche peculiari, successivamente analizzeremo come lato software queste componenti sono state sviluppate e combinate, descrivendo le righe di codice interessate, vedremo poi come l'utente si interfaccia all'applicazione web e i possibili miglioramenti. La struttura quindi sarà la seguente:

- nella parte iniziale verranno mostrati gli strumenti utilizzati nel sistema, più specificatamente le componenti hardware.
- nella seconda parte verranno presentati i diversi linguaggi di programmazione coinvolti nella fase di sviluppo.
- la terza parte avrà al suo interno il codice sviluppato con relativi commenti riguardo al suo funzionamento.
- nella quarta parte verrà mostrata l'Applicazione Web, più specificatamente la sua interfaccia.
- la sezione finale sarà concentrata sui possibili miglioramenti del sistema sviluppato e sulle conclusioni.

Capitolo 2

Strumenti Utilizzati

2.1 IDS UI-3160CP Rev. 2.1

La telecamera industriale IDS UI-3160CP Rev. 2.1 [2] con sensore CMOS e otturatore globale 2/3" offre una risoluzione completa 2.3 MP, ovvero 1920x1200 pixel con un picco massimo di 165 fps e una risoluzione comune di 1600x1200 con un massimo di 200 fps. La telecamera è full HD (1920x1080) pixel, con attacco C-Mount che la rende molto utile per applicazioni di tipo medico industriale, per compiti di visualizzazione.

Nel nostro caso la telecamera verrà posizionata all'interno della piscina in maniera tale da monitorare l'entrata in acqua dei nuotatori, il suo utilizzo poi verrà combinato con quello di Arduino con il quale verrà collegata. Il nostro scopo finale sarà quello sincronizzare Arduino con tre telecamere, così facendo avremo tre registrazioni simultanee all'interno della piscina.



Figura 2.1: Telecamera IDS UI-3160CP Rev. 2.1 [2]

2.2 Arduino MKR WAN 1300

Arduino è una piattaforma hardware composta da una serie di schede elettroniche dotate di un microcontrollore, dove combinata con sensori, attuatori e altri strumenti può essere utilizzata per progettare diversi controllori di velocità, luci, azionamenti. Grazie ad un apposito ambiente di sviluppo integrato denominato Arduino IDE[17] è possibile programmare il microcontrollore.

Arduino IDE è un'applicazione multipiattaforma scritta in Java, la quale facilita la scrittura di codice sorgente attraverso strumenti come il syntax highlighting, ovvero evidenziare con colori diversi particolari sintassi del testo che vengono riconosciute dall'ambiente di sviluppo. I codici vengono definiti 'sketch' che possono essere caricati sulle schede grazie ad un semplice click.

L'Arduino MKR WAN 1300[14] è una piattaforma progettata per offrire una soluzione semplice ed economica per chiunque vuole approcciarsi a questo tipo di tecnologia, è perfettamente funzionante con Arduino IDE grazie al quale possiamo sviluppare codice per la scheda. Verrà utilizzata nel nostro progetto per inviare impulsi alla telecamera con la quale interagirà per creare sessioni di registrazione dove potremmo controllare il frame rate inviando impulsi ad una velocità controllata da noi stessi.



Figura 2.2: Arduino MKR WAN 1300 [14]

2.3 Modello Client-Server

É un modello di interazione tra una coppia di processi uno dei quali richiede un servizio **Client** ad un altro che lo fornisce **Server**.

Il Client e il Server sono moduli funzionali con interfacce ben definite le funzioni possono essere realizzate da moduli hardware, software o da una combinazione dei due, possono anche risiedere su computer dedicati.

L'interazione tra Client e Server si attiva quando, il primo invia una richiesta al secondo e durante l'interazione tra i due i ruoli rimangono costanti, mentre può capitare che nell'interazione successiva i ruoli siano invertiti.

I moduli possono risiedere su macchine diverse interconnesse da una rete oppure sullo stesso calcolatore, inoltre lo scambio di informazioni tra i moduli avviene mediante l'utilizzo di particolari **messaggi**. I messaggi sono dei contenitori all'interno dei quali viene inserito il contenuto che il Client ha richiesto al Server, inviandolo al modulo corrispondente; tipicamente sono **interattivi** e generano l'attivazione di processi lato Client



Figura 2.3: Architettura Client-Server [11]

2.3.1 Client

Il Client nell'ambito delle reti informatiche e dell'architettura logica di rete Client-Server indica una qualunque componente software, presente tipicamente su una macchina host, che utilizzando opportuni protocolli di comunicazione riesce ad effettuare una richiesta ad un altra componente detta Server.

Nel web il Client è il **browser** che comunicano con un server web attraverso il protocollo **Hypertext Transfer Protocol** di comunicazione, ma in generale esistono protocolli in base alle richieste che eseguiamo; per esempio nel caso in cui effettuiamo una richiesta di email il protocollo utilizzato è il **Simple Mail Transfer Protocol**.

🍯 Nuova scheda 🛛 🗙	+	- @ ×
\leftrightarrow \rightarrow C \textcircled{a}	Q Cerca con Google o inserisci un indirizzo	<u>↓</u> II\ 🗉 🛎 Ξ
Come iniziare		Altri segnalibri
Come iniziare	G Cerca sul Web →	È Atri segnalibri

Figura 2.4: Esempio di Web browser Firefox

2.3.1.1 Architettura Firefox

L'architettura Firefox ha un cuore che realizza le funzionalità di base e viene definito **Gecko**, il quale gestisce le principali funzioni di un browser e il rendering per la visualizzazione. Gecko si interfaccia con la **User Interface**, la quale produce il risultato del rendering e gestisce l'interazione con l'utente.

La **Necko** si occupa della gestione dei collegamenti di rete, controllando i protocolli web, gestisce lo scambio di informazioni.

Lo **Spider-Monkey** è l'interprete JavaScript per gestire gli eventi e gli elementi interattivi dei documenti HTML. Il **Display Backend** è l'architattura per la libreria grafica mentre l'**Expat** è l'XML parser di Firefox per gestire l'informazione parziale. Per la persistenza di memoria nella sessione di login, per la gestione della sicurezza invece vi è la componente **Data Persistence**



Figura 2.5: Architettura Web Browser Firefox [16]

2.3.2 Server

Il Server è un componente o un sottosistema di gestione e traffico dell'informazione, e fornisce ad un Client un determinato documento oppure un servizio attraverso il protocollo HTTP nel mondo web.

In altre parole si tratta di un computer o un programma che fornisce i dati che gli vengono richiesti ad un altro elaboratore, quindi si può trattare sia di un componente puramente hardware che di un componente puramente software.

Un **Server Web**, invece, è un'applicazione software in esecuzione su di un Server, ed è in grado di gestire le richieste di trasferimento di pagine web di un Client, nella maggior parte dei casi un web browser. Possiamo anche installare un Server web all'interno di un normale PC al fine di provare localmente le pagine web, questo soprattutto per chi si occupa di sviluppo di siti web.

La connessione che il Server stabilisce con il Client è una connessione **stateless**, ovvero una volta che si interrompe la connessione tra i due non vi è nessun sistema di memoria che ci permette di vedere quale interazione vi è stata; questo meccanismo comporta problemi ad esempio per procedure di login, le quali si devono 'ricordare' ad ogni nuova interazione con il Server che precedentemente l'accesso al sito è stato effettuato.

2.3.2.1 Web Server Apache

Apache HTTP Server è il principale web server libero sviluppato dalla casa **Apache Software Foundation**, ed è la piattaforma server web più diffusa nel mercato.

La sua architettura è di tipo modulare, ovvero ad ogni richiesta che viene effettuata dal client vengono eseguite funzioni specifiche da ogni modulo di cui il web server è composto, ogni modulo si occupa di una specifica funzionalità ed è gestito dal HTTP_CORE. La gestione delle richieste viene effettuata tramite polling.

Un client quando effettua una richiesta ad un web server Apache attiva al suo interno l'**HTTP_PROTOCOL** che gestisce tutti i messaggi provenienti, il successivo modulo che si attiva è l'**HTTP_MAIN** richiama tutti i contenuti. L'HTTP_CORE gestisce le funzionalità di base del server, mentre lo scopo del **HTTP_REQUEST** è quello di acquisire la richiesta che il client ha effettuato, generare il risultato e inviarlo all'HTTP_MAIN, attraverso l'HTTP_PROTOCOL e restituisce il risultato al Client.



Figura 2.6: Architettura Web Server Apache

2.4 Linguaggi Utilizzati

Si affrontano diversi linguaggi di programmazione in base a cosa è necessario implementare nel sistema. Quelli che si sono presentati sono C/C++ nello sviluppo di codice Arduino, Python per la creazione dell'Applicazione Web e per gestire le richieste nella porta seriale di Arduino ed infine HTML e CSS come linguaggi per creare interfacce web.

2.4.1 C/C++

Il linguaggio di programmazione C è un linguaggio di tipo imperativo procedurale, viene definito come un linguaggio ad alto livello che riesce ad integrare anche operazioni di basso livello. Il linguaggio C nel corso del tempo si è imposto come uno dei principali linguaggi per la creazione di software, sistemi operativi, applicazioni e molto altro.

Il C++ invece è una variante del linguaggio C in cui, oltre alle funzionalità già presenti, aggiunge anche la programmazione orientata agli oggetti e le sue peculiarità vengono aggiornate continuamente.

Arduino IDE, come già detto, utilizza questi linguaggi per la creazione di sketch che vengono caricati direttamente nella scheda in maniera tale da svolgere operazioni che vengono opportunamente scritte nell'editor dell'ambiente di sviluppo.

2.4.2 Pyhton

Python è un linguaggio orientato agli oggetti e il suo utilizzo è diffuso in molte aree dell'informatica, dalla creazione di applicazioni distribuite, computazione numerica etc. Nasce agli inizi degli anni novanta ed è un linguaggio multi-paradigma che come caratteristiche fondamentali ha la flessibilità, la dinamicità e semplicità. Due peculiarità di Python sono l'importanza dell'indentazione e le variabili non tipizzate, tanto che il controllo dei tipi viene fatto a **runtime**.

2.4.2.1 Flask

Flask[13] è un micro-framework scritto in Python, viene definito micro-framework poiché ha un nucleo molto semplice ma estendibile, infatti supporta estensioni che permettono di aggiungere delle funzionalità all'applicazione come se fossero dello stesso Flask.

2.4.3 HTML

L'Hyper Text Markup Language è un linguaggio di markup, nacque grazie alla necessità di scambio di documenti ipertestuali in ambito dello sviluppo web dove era necessario un linguaggio comune per la strutturazione di documenti. Questo tipo di linguaggio serve per descrivere il contenuto dei documenti attraverso delle 'etichette' denominate TAG, deriva dall'SGML[12] (altro linguaggio di markup).

La visualizzazione di un documento HTML dipende dal browser dove esso viene aperto, quindi c'è un problema di incompatibilità crossbrowser, anche se esiste comunque una standardizzazione su come determinati TAG devono essere interpretati da diversi browser.

L'HTML è un linguaggio di dominio pubblico e il suo organo di organizzazione è il W3C (World Wide Web Consortium).

Non è un linguaggio di programmazione quindi non è necessario andare a definire variabili, strutture dati, funzioni. I file HTML vengono interpretati senza eventuali errori di sintassi blocchino la traduzione, differentemente da quanto accade, per esempio, in C o Java; l'interprete che svolge questo compito infatti viene definito lasco.

2.4.4 CSS

Il CSS è un linguaggio che permette di definire le modalità di visualizzazione degli oggetti presenti nella pagina, a differenza dell'HTML che invece ne definisce la struttura. Queste regole di visualizzazione le possiamo associare o a singoli elementi oppure direttamente ad un gruppo di oggetti.

Per integrare queste regole all'interno di un documento HTML possiamo inserire all'interno dei TAG l'attributo 'style', possiamo inserirle in una sezione (solitamente nella head), oppure creare un foglio di stile esterno e poi includerlo all'interno del file.

2.4.5 JavaScript

Il linguaggio JavaScript è orientato agli oggetti e agli eventi per la creazione di oggetti interattivi e dinamici all'interno dei documenti HTML che vengono inseriti nel web. Questo tipo di linguaggio viene sfruttato principalmente lato Client ma vengono prodotti codici anche lato Server.

Eún linguaggio interpretato dove il codice non viene compilato ma direttamente eseguito, l'interprete solitamente è contenuto nel browser. Quando viene aperta una pagina web contenente codice JavaScript viene interpretato dal browser grazie all'interfaccia denominata DOM che permette proprio quata interazione tra browser e codice JavaScript.

Il DOM è un'API quindi è un'interfaccia per la programmazione di applicazioni, ovvero è un insieme di funzioni, metodi e proprietà che i programmi possono richiamare, inoltre è un modello che definisce come i diversi oggetti presenti in un documento HTML sono collegati tra di loro. Attraverso JavaScript possiamo accedere ai vari elementi del DOM, con i relativi handler che intercettano gli eventi che avvengono nelle pagine.

2.5 Editor di sviluppo

Per sviluppare il nostro dispositivo dal lato software si utilizzano principalmente due editor ovvero Visual Studio Code, per sviluppare codice in Python, HTML, CSS, JavaScript e Arduino IDE invece per C/C++ per la parte relativa ad Arduino.

2.5.1 Visual Studio Code

Visual studio code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Il vantaggio principale di Visual Studio Code è quello di non essere progettato per supportare un solo tipo di linguaggio, ma dal suo marketplace è possibile scaricare le relative estensioni che permettono di non limitarsi a programmare unicamente in un linguaggio, come si può intuire, quindi, è un tipo di editor molto dinamico, efficace, e elastico.

📢 File Edit Selection View Go Run Terminal Help	Welcome - Visual Studio Code	-	٥	×
↓ Welcome ×				
Visual Studio Code Editing evolved				
Start New file	Customize			
Open folderor done repository	Tools and languages Install support for JavaScript, Python, Java, PHP, Azure, Docker and more			
E5 Recent Progetto [S3H: homefmit.ddns.net] /home/pi/swim4all	Settings and keybindings Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others			
services (SSH: homefinit.ddns.net) /home/pi/xwim4al/Progetto Progetto C.\xampyDi/ddocs Progetto D:\Xivola\Samiid adare\Tirocinio + Tesi Python D:\Xivola\Samiid adare\Tirocinio + Tesi	Color theme Make the editor and your code look the way you love			
Help Primable keyboard cheatsheet	Find and run all commands Rapidly access and search commands from the Command Palette (Ctrl+Shift+P)			
	Interface overview Get a visual overlay highlighting the major components of the UI			
Stack Overflow Join our Newsletter	Interactive playground Try out essential editor features in a short walkthrough			
Show welcome page on startup				
⊗0 ∆ 0			R	φ

Figura 2.7: Interfaccia Visual Studio Code

2.5.2 Arduino IDE

Arduino IDE è un'applicazione multipiattaforma scritta in Java, ed è derivata dall'IDE creato per il linguaggio di programmazione Processing e per il progetto Wiring.

É presente un editor dotato di particolari proprietà come il syntax highlighting, l'indentazione automatica, controllo delle parentesi. Questa piattaforma permette anche di compilare e caricare sulla scheda Arduino il programma che è stato scritto nell'editor. Come già detto il linguaggio supportato è C/C++ si una particolare libreria denominata Wiring.

Ogni sketch per Arduino contiene due moduli principali che sono il metodo setup e il metodo loop; la differenza tra le due è la seguente, ovvero la funzione setup viene chiamata una sola volta allo start del programma, mentre la funzione loop viene chiamata ripetutamente.



Figura 2.8: Interfaccia Arduino IDE

Capitolo 3 Sviluppo

Il sistema di visione è schematizzato in figura:



Figura 3.1: Workflow del sistema

L'applicazione Flask ci permette di gestire i trigger di Arduino attraverso due possibili modalità, START, generatrice di impulsi finché non decidiamo di effettuare lo STOP e la START N che invece genera un numero determinato di impulsi, specificato nella stessa app in un'opportuna form. Una volta che viene attivato il comando dall'applicazione, Arduino genera dei trigger con una certa frequenza che vengono inviati ad una telecamera, la quale una volta che li riceve ad ogni trigger acquisisce un frame. Successivamente si attiva l'applicazione sviluppata dal mio collega di progetto Iacopo Simoncini che permette di salvare questi frame in una cartella. Una volta che i trigger si interrompono a quel punto si può generare il video attraverso un'applicazione sfruttando la libreria ffmpeg[3]. Nel mio sviluppo mi sono concentrato nel programmare Arduino, l'applicazione Flask, e il codice relativo alla creazione del video.

3.1 Sviluppo codice Arduino

Di seguito verrà mostrato il codice Arduino sviluppato e commentato durante la fase di implementazione, poi caricato sulla scheda MKR 1300 WAN.

3.1.1 setup()

```
void setup() {
    // put your setup code here, to run once:
    pinMode(buttonPin, OUTPUT);
    Serial.begin(9600);
    while(!Serial);
    Serial.println("0");
}
```

- **pinMode(buttonPin, OUTPUT)** : configura il PIN specificato, in questo caso il buttonPin, in modo che si comporti come input o output, in questo caso OUTPUT;
- Serial.begin(9600) : Imposta la velocità dati in bit al secondo (baud) per la trasmissione dati seriale. Nel nostro caso 9600.
- while(!Serial) : Comando per eseguire un comando quando la comunicazione seriale ancora non è attiva, nel nostro caso verrà stampato lo 0 sul monitor seriale.

3.1.2 loop()

```
void loop() {
    // put your main code here, to run repeatedlyz
    recvWithStartEndMarkers();
3
    if (newData == true) {
      strcpy (tempChars, receivedChars);
      parseData();
      if (strcmp(command, "START") = 0) {
        Serial.println("1");
        startREC();
9
        Serial.println("0");
      } else if (strcmp(command, "START N") == 0) {
11
        Serial.println("1");
        startNImpulsi();
13
        Serial.println("0");
      else
```

- recvWithStartEndMarkers() : la prima funzione che viene chiamata nel loop, serve per leggere il comando che viene inserito nella porta seriale di Arduino;
- if (newData == true) : controllo della variabile newData che inizialmente falsa dentro la funzione sopra citata diventerà vera, una volta entrati nell'if la stringa receivedChars verrà assegnata alla stringa tempChars. Subito dopo viene chiamata la funzione parseData() che ci permetterà di effettuare il parsing dei dati che vengono inseriti nella stringa da seriale.
- if(strcmp(command, "comando") == 0) : viene letto il comando corrispondente al primo elemento parsato dalla funzione **parseData** e comparato con due possibili comandi, 'START' e 'START N', se entriamo nell'if dello START verrà stampato nel monitor lo stato "1" che corrisponde allo stato di attivazione degli impulsi di Arduino, poi viene chiamata la funzione startREC(), poi una volta conclusa viene stampato lo "0" che indica lo stato di spegnimento degli impulsi di Arduino.

Stessa cosa accadrà se invece dello START entriamo in START N dove invece che la funzione startREC() verrà chiamata la funzione startNImpulsi().

• else : nel caso in cui il comando non corrisponde ai due casi precedenti verrà stampato nel monitor seriale il messaggio "Inserisci il corretto comando".

3.1.3 startREC()

Questa funzione ci permette di inviare impulsi con Arduino finché non viene scritto nel monitor seriale un qualsiasi carattere che ne provoca l'immediato stop.

```
void startREC() {
   boolean wait = true;
   while (wait) {
     digitalWrite (buttonPin, HIGH);
     delay(tOn);
     digitalWrite (buttonPin, LOW);
     delay(tacquisizione - tOn);
```

- wait : variabile che servirà per uscire dal ciclo di generazione degli impulsi.
- while (wait) {...} : ciclo continuo di impulsi che vengono generati da Arduino attraverso la funzione digitalWrite(buttonPin, HIGH) che genera un impulso nel PIN corrispondente al buttonPin, segue delay(tOn) che permette di mantenere il PIN alto per la quantità tOn che viene data dalla stringa inserita da seriale, poi abbassiamo il segnale con digital-Write(buttonPin,LOW), ed infine attraverso la delay(tacquisizione tOn) facciamo in modo che il segnale prima di riattivarsi aspetti un tempo pari alla differenza tra il tempo di acquisizione e il tOn, ovvero il corrispondente tOff.

Successivamente per fermare questo ciclo di impulsi basta scrivere da seriale un qualsiasi carattere che verrà salvato nella variabile **incomingByte** e verrà resa falsa la variabile**wait** così si potrà uscire dal ciclo.

3.1.4 startNImpulsi()

Questa funzione ci permette di inviare un numero N di impulsi, una volta generato il numero corretto di trigger fermeremo l'inoltro degli impulsi automaticamente, a differenza della startREC(), la quale necessitava si un carattere nel monitor per fermarsi.

```
void startNImpulsi() {
    int tempo = 0;
    int i = 1;
    while (tempo <= nimpulsi * tacquisizione) {
        digitalWrite(buttonPin , HIGH);
        delay(tOn);
        digitalWrite(buttonPin , LOW);
        delay(tacquisizione - tOn);
        i++;
        tempo = tacquisizione * i;
}
</pre>
```

- while(tempo <= nimpulsi * tacquisizione) : si può entrare nel ciclo solo se la variabile tempo non ha superato il tempo totale calcolato come numero di impulsi moltiplicato con il tempo di acquisizione che risulta il tempo totale per generare gli impulsi.
- digitalWrite(buttonPin)...delay(tacquisizione tOn) : genera il segnale per un periodo pari a tOn, per poi spegnerlo per un periodo tOff (corrispondente a tacquisizione - tOn)
- tempo = tacquisizione*i aggiornamento della variabile tempo ad ogni ciclo di impulsi per poi essere controllata nella condizione del while.

3.1.5 recvWithStartEndMarkers()

Questa funzione ci permette di leggere ciò che viene scritto sul terminale come una vera e propria stringa.

```
void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
2
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;
6
    while (Serial.available() > 0 & newData == false) {
      rc = Serial.read();
8
      if (recvInProgress == true) {
10
         if (rc != endMarker) {
           received Chars [ndx] = rc;
12
           ndx++;
           if (ndx \ge numChars) {
14
             ndx = numChars - 1;
           }
16
         }
         else {
18
           received Chars [ndx] = \sqrt[3]{0}; // terminate the string
           recvInProgress = false;
20
           ndx = 0;
           newData = true;
22
         }
      }
24
      else if (rc == startMarker) {
26
         recvInProgress = true;
      }
28
    }
  }
30
```

- **rc** = **Serial.read()** : viene letto il carattere della stringa.
- if(recvInProgress == true) {...} : se il carattere letto è diverso dal carattere del termine della stringa, nel nostro caso ">" specificato nella variabile endMarker, salviamo il carattere nell'array receivedChars.
- else {...} : verifichiamo che la stringa sia finita per poi procedere con il codice.
- else if (rc == startMarker) : se il carattere iniziale è uguale alla variabile startMarker, nel nostro caso il carattere "<" allora settiamo il valore di recvInProgress a true così entriamo nel ciclo di lettura dei caratteri.

3.1.6 parseData()

Questa funzione prende la stringa che abbiamo salvato prima per poi "parsare" i dati che sono contenuti in essa.

```
// split the data into its parts
  void parseData() {
2
    char * strtokIndx; // this is used by strtok() as an index
    strtokIndx = strtok(tempChars, ",");
    strcpy (command, strtokIndx);
    strtokIndx = strtok(NULL, ","); // this continues where the
     previous call left off
    tacquisizione = atoi(strtokIndx); // converte questa parte in
     un intero
    strtokIndx = strtok(NULL, ",");
10
    tOn = atoi(strtokIndx);
12
    strtokIndx = strtok(NULL, ",");
    nimpulsi = atoi(strtokIndx);
14
```

Viene letta la stringa separando i quattro elementi che la compongono separati dal carattere "virgola":

- Il primo elemento della stringa viene copiato nella variabile command.
- Il secondo elemento della stringa viene trasformato in numero dalla funzione **atoi** e assegnata alla variabile **tacquisizione**.
- Il terzo elemento della stringa viene trasformato anch'esso dalla funzione atoi e assegnato alla variabile **tOn**.
- Il quarto elemento della stringa come i due precedenti viene trasformato in numero e assegnato alla variabile **nimpulsi**.

3.2 Sviluppo codice HTML

Ora analizzeremo il codice HTML, dove potremmo caricare e sincronizzare i video di tre diverse telecamere, oltre a questo è possibile anche gestire Arduino attraverso un'opportuna form.

3.2.1 Gestione video

```
<body>
      <div id="titolo">
          < h1 >
               SWIM4All Video
           </h1>
      </div>
      <div id="video-container">
          <video id="video1" src={{ url for('static', filename="video.")</pre>
     mp4" ) }} controls> </video>
          <video id="video2" src={{ url_for('static', filename="video.")</pre>
     mp4" ) }} muted></video>
          <video id="video3" src={{ url for('static', filename="video.")</pre>
11
     mp4" ) }} muted></video>
      </div>
13
```

Questa è la zona dove vengono gestiti i video delle tre telecamere. Nella zona superiore del documento HTML avremo il titolo, "SWIM4ALL Video", poco più in basso ci saranno tre TAG video dove in ognuno verrà caricato il rispettivo video. (La sincronizzazione verrà gestita in JavaScript). Da notare l'attributo **controls** presente solamente nel TAG con id **video1** che ci permetterà di avere i controlli per il video, come alzare o abbassare il volume, oppure spostarci in un preciso punto del video. **Muted** presente invece negli altri due TAG ci permette di rendere muti gli altri due video.

3.2.2 Form per la gestione di Arduino

Qui descriveremo la form con la quale gestiremo gli impulsi della scheda Arduino.

```
<div id="bot">
      <div id="formdati">
          <form id="invio" action="{{url for('home')}}" method="POST">
              <div id="titoloform">
                   <h3> Controllo Arduino: </text>
               </div>
              < br >
              < div >
                   <label> Scegli la modalit&#224 </label>
9
                   <select id="modalita" name="modalita">
                       < option value = "0" > START N < / option >
11
                       <option value="1">START </option>
                   </\operatorname{select}>
13
               </div>
              < br >
              < div >
                   <label for="acquisizione">Tempo Acquisizione: 
     label >
                   <input id="acquisizione" name="acquisizione" type="
     number">
                   <text> ms </text>
19
               </div>
              < br >
21
              < div >
                   <label for="ton">tOn: </label>
23
                   <input id="ton" name="ton" type="number">
                   <text> ms </text>
25
               </div>
              < br >
27
              <div id="divimpulsi">
                   <label>Numero impulsi: </label>
29
                   <input id="impulsi" name="impulsi" type="number">
               </div>
31
              < br >
              <input type="submit" name="submit" value="Invia"> </p
33
              <input type="submit" name="submit" value="STOP"> 
          </form>
35
      </div>
      <div id="stato">
37
           Stato Arduino: < b>{\{ value \}} < / b> 
      </div>
39
  </div>
```

La form è composta da diversi elementi:

- **modalita** : è un menù a tendina dove possiamo scegliere se avviare Arduino in modalità START o START N.
- **acquisizione** : possiamo scrivere all'interno il valore in millisecondi del tempo di acquisizione di Arduino.
- ton : possiamo scrivere all'interno il valore in millisecondi del tOn di Arduino.
- **nimpulsi** : nel caso in cui scegliamo lo START N possiamo scegliere il numero di impulsi da inviare.
- Invia : bottone che serve per inviare la richiesta alla porta seriale di Arduino.
- **Stop** : bottone utilizzato per fermare gli impulsi quando Arduino è in modalità START.

3.3 Sviluppo codice CSS

Foglio di stile associato al documento HTML, accediamo ai vari elementi attraverso l'attributo id presente nei TAG.

```
#titolo {
       text-align: center;
2
  }
4
  #video-container {
      width: 100%;
6
       height: auto;
       text-align: center;
8
  }
10
  #video1 {
      width: 30\%;
12
  }
14
  #video2 {
      width: 30%;
16
  }
18
  #video3 {
      width: 30%;
20
  }
22
  #bot {
       display: inline;
24
  }
  #titoloform {
26
      font-style: serif;
28 }
  #formdati {
      margin-top: 50px;
30
       margin-left: 20\%;
       width: auto;
32
  }
```

3.4 Sviluppo codice JavaScript

3.4.1 Sincronizzazione video

Di seguito il codice JavaScript che ci permette sincronizzare i video.

```
var video1 = document.getElementById('video1');
  video1.addEventListener("play", function () {
3
      document.getElementById('video2').play();
      document.getElementById('video3').play();
5
  });
7
  video1.addEventListener("pause", function () {
      document.getElementById('video2').pause();
9
      document.getElementById('video3').pause();
 })
 video1.addEventListener("seeking", function () {
13
      document.getElementById('video2').currentTime = video1.
     currentTime;
      document.getElementById('video3').currentTime = video1.
     currentTime;
  })
17
  video1.addEventListener("seeked", function () {
      document.getElementById('video2').currentTime = video1.
19
     currentTime;
      document.getElementByID('video3').currentTime = video1.
     currentTime;
 })
21
```

In questo caso abbiamo la variabile **video1** che prende l'elemento del DOM avente come id **video1**, successivamente mettiamo all'ascolto questo elemento a quattro possibili eventi ovvero il play, pausa, seeking, seeked. Quando viene premuto il tasto play del primo video grazie al metodo **getElementById** con il quale agganciamo gli altri due video concatenato con il metodo play facciamo in modo di far iniziare la riproduzione simultaneamente per tutti e tre i video presenti nella schermata. La medesima operazione viene fatta anche per gli altri tre comandi. 2

3.4.2 Gestione modalità START/START N

```
document.getElementById('modalita').addEventListener('change',
    function () {
    var style = this.value == 0 ? 'block' : 'none';
    document.getElementById('divimpulsi').style.display = style;
});
```

Questa porzione di codice JavaScript, invece, ci aiuta a gestire il menù a tendina poiché se scegliamo la modalità START non abbiamo bisogno di inserire il numero di impulsi, in quanto questo comando indica la generazione del segnale finché non si prema il pulsante stop.

Con il metodo **getElementeById** nel DOM agganciamo l'elemento con id **modalità** e lo mettiamo in ascolto dell'evento **change**, se il valore corrispondente di modalità è 0 allora nascondiamo il blocco **divimpulsi**, viceversa invece verrà mostrato.

3.5 Sviluppo codice Python

Di seguito sarà mostrato come il codice relativo a Python, verranno descritti i diversi metodi e le librerie utilizzate, vedendo come abbiamo messo in comunicazione una Applicazione Web con Arduino e come vengonon gestite le richieste che vengono inviate dall'applicazione. Il framework utilizzato è Flask.

3.5.1 app.py

```
app = Flask(__name__, template_folder="templates", static_folder="
    static")
a = Arduino()
time.sleep(3)
LED_PIN = 6
```

Questa porzione di di codice ci permette di creare un'istanza dell'applicazione Flask, per convenzione i templates e i file statici vengono memorizzati in sottodirectory all'interno dell'albero dei sorgenti Python dell'applicazione rispettivamente con in nomi **templates** e **static**, come specificato nella creazione dell'istanza. Come già detto nell'introduzione Flask è un micro-framework e con "micro" intendiamo che mantiene un nucleo semplice ma estensibile, ciò però non vuol dire che tutta l'applicazione debba essere contenuta in un unico file python. Nella riga successiva creiamo un'istanza della classe **Arduino()** che verrà descritta successivamente, settiamo i PIN che ci interessano da dove gli impulsi di Arduino devono uscire per il funzionamento desiderato (**LED_PIN**). Il valore di LED_PIN può essere cambiato in base a come Arduino attraverso i PIN interagisce con altri dispositivi.

```
@app.route('/', methods = ['POST', 'GET'])
  def home():
2
      11 11 11
      This function just responds to the browser ULR
      localhost:5000/
6
                        the rendered template 'home.html'
      :return:
      11 11 11
8
      stato = 'OFF'
      time.sleep(1)
      if request.method == 'POST':
          \# if we press the turn on button
12
           if request.form ['submit'] == 'Invia':
               if request.form ['modalita'] == "1" :
14
                    acquisizione = request.form ['acquisizione']
                    ton = request.form['ton']
16
                   impulsi = "0"
```

```
a.string write ("START", acquisizione, ton, impulsi)
18
                   stato = 'ON'
                   print ('TURN ON')
20
               elif request.form['modalita'] == "0" :
                   acquisizione = request.form ['acquisizione']
                   ton = request.form ['ton']
                   impulsi = request.form['impulsi']
24
                   a.string_write("START N", acquisizione, ton, impulsi)
                   stato = 'ON'
26
                   print ('TURN ON')
28
          \# if we press the turn off button
           elif request.form['submit'] == 'STOP':
30
               a.string_write("STOP","0","0","0")
               stato = , OFF'
32
               print ('TURN OFF')
           else:
34
               pass
36
      return render template ('video.html', value = stato)
```

Questa porzione di codice descrive come la nostra applicazione gestisce le richieste che l'utente può effettuare in essa.

@app.route('/', methods = ['POST', 'GET'])

@app.route ci permette di creare una rotta all'indirizzo localhost:5000/ in cui la nostra applicazione viene attivata. All'interno dell'@app.route si può notare la presenza di methods =['POST','GET'], che indica che questo indirizzo si prepara a gestire richieste di tipo POST e GET.

3.5.1.1 Gestione delle richieste POST

```
if request.method == 'POST':
```

Nell'HTML avevamo visto la presenza di due bottoni, **Invia** e **Stop**, entrambi avevano come tipo richiesta la POST, questa semplice riga ci permette di gestire la richiesta quando l'utente clicca uno dei due bottoni.

```
if request.form['submit'] == 'Invia':
               if request.form ['modalita'] == "1" :
                   acquisizione = request.form ['acquisizione']
                   ton = request.form['ton']
                   impulsi = "0"
5
                   a.string_write("START", acquisizione, ton, impulsi)
                   stato = 'ON'
                   print ('TURN ON')
               elif request.form['modalita'] == "0" :
9
                   acquisizione = request.form ['acquisizione']
                   ton = request.form ['ton']
11
                   impulsi = request.form['impulsi']
                   a.string_write("START N", acquisizione, ton, impulsi)
13
                   stato = 'ON'
                   print ('TURN ON')
15
```

Quando l'utente clicca il bottone Invia la richiesta POST viene gestita come descritto dal codice, ovvero:

- 1. Viene controllata la modalità, ovvero il valore del menù a tendina, se il valore è 1 siamo in modalità START, mentre e si entra nel primo if, mentre se il valore è 0 siamo in modalità START N.
- Una volta entrati in uno dei due if vengono estratti dalla form i valori che l'utente scrive da tastiera e vengono copiati nelle variabili acquisizione, ton, impulsi (se ci troviamo nell'if corrispondente alla modalità START sarà settata a 0).
- 3. Infine viene invocata la funzione appartenente alla classe Arduino() concatenandola all'istanza a creata all'inizio string_write() alla quale passiamo come parametri le variabili prima inizializzate. Avrà come compito quello di scrivere nella porta seriale il comando corrispondente per interagire con il codice caricato nella scheda Arduino descritto precedentemente.

```
elif request.form['submit'] == 'STOP':
    a.string_write("STOP","0","0","0")
    stato = 'OFF'
    print ('TURN OFF')
```

Se l'utente ha attivato Arduino in modalità START, ovvero vengono inviati impulsi finché non arriva un segnale di stop, questa richiesta POST ci permette di farlo; richiamando la funzione **string_write()** inviamo una stringa al seriale di dove Arduino è collegato corrispondente al segnale di STOP nel codice installato dentro la scheda.

```
3.5.1.2 if __name__ == '_main_'
```

```
if __name__ == '__main__':
    thread1 = timeofsession.IlMioThread(a)
    thread1.start()
```

Viene innescata la funzione main, è usata per testare e quindi definire un comportamento diverso in caso che il codice sia eseguito come programma o come modulo.

Successivamente viene istanziato il **thread1** dalla classe **IlMioThread(a)** per poi eseguire la sua esecuzione con il metodo .start() che attiva il modulo .run() contenuto nella definizione del thread.

3.5.2 Arduino()

La classe Arduino grazie ai suoi metodi ci permette di comunicare con una scheda attraverso la porta seriale con la quale stabiliamo la connessione.

3.5.2.1 __init__()

In Python, __init__, è un metodo speciale, e viene chiamato automaticamente appena andiamo a creare un oggetto e ci permette di inizializzare gli attributi della classe. Nel nostro caso viene vengono inizializzate la variabile **serial_port**, ovvero la porta seriale dove Arduino è collegato, **baud_rate**, cioè la velocità di trasmissione, e **read_timeout**.

La connessione con la porta seriale avviene grazie serial.Serial(serial_port, baud_rate) che ci permette di creare un collegamento con la porta avendo un determinato baud rate specificati in fase di chiamata della __init__.

3.5.2.2 set pin mode()

```
def set_pin_mode(self, pin_number, mode):
    """
    Performs a PINMode() operation on PIN_number
    Internally sends b'M{mode}{pin_number} where mode could be:
    - I for INPUT
    - O for OUIPUT
    - O for OUIPUT
    P for INPUT_PULLUP MO13
    """
    command = (''.join(('M',mode,str(pin_number)))).encode()
    #print 'set_pin_mode =',command,(''.join(('M',mode,str(
    pin_number))))
    self.conn.write(command)
```

Questo modulo ci permette di definire il pinMode() corrispondente al codice Arduino, ovvero configurare un PIN in modo che si comporti come input oppure come output.

3.5.2.3 digital read()

```
def
          digital read(self, pin number):
          Performs a digital read on pin number and returns the value
     (1 \text{ or } 0)
          Internally sends b'RD{pin number}' over the serial
     connection
          11 11 11
5
          command = (''.join(('RD', str(pin number)))).encode()
          self.conn.write(command)
          line received = self.conn.readline().decode().strip()
          header, value = line received.split(':') # e.g. D13:1
9
          if header == ('D'+ str(pin number)):
              \# If header matches
11
               return int (value)
```

Legge in un determinato PIN, se il valore è HIGH oppure LOW, ovvero se vi è segnale o meno.

3.5.2.4 digital write

```
def digital_write(self, pin_number, digital_value):
    """
    Writes the digital_value on pin_number
    Internally sends b'WD{pin_number}:{digital_value}' over the
    serial
        connection
        """
        command = (''.join(('WD', str(pin_number), ':',
            str(digital_value)))).encode()
        self.conn.write(command)
```

Scrive un valore alto o basso in un determinato PIN specificato come parametro della funzione.

Se il PIN è configurato come OUTPUT, il voltaggio è settato come 5 V come HIGH e 0 V come LOW, se invece è configurato come INPUT, digitalWrite () abiliterà (HIGH) o disabiliterà (LOW).

3.5.2.5 analog_read()

```
analog read(self, pin number):
      def
          Performs an analog read on pin number and returns the value
     (0 to 1023)
          Internally sends b'RA{pin number}' over the serial
     connection
          11 11 11
          command = (''.join(('RA', str(pin number)))).encode()
          self.conn.write(command)
          line received = self.conn.readline().decode().strip()
          header, value = line_received.split(':') # e.g. A4:1
9
          if header == ('A' + str(pin number)):
              \# If header matches
11
              return int (value)
```

Legge il valore dal PIN analogico specificato. Le schede Arduino contengono un convertitore da analogico a digitale a 10 bit multicanale. Ciò significa che mapperà le tensioni di ingresso comprese tra 0 e la tensione operativa (5 V o 3,3 V) in valori interi compresi tra 0 e 1023.

3.5.2.6 analog write()

```
def analog_write(self, pin_number, analog_value):
    """
    Writes the analog value (0 to 255) on pin_number
    Internally sends b'WA{pin_number}:{analog_value}' over the
    serial
        connection
    """
        command = (''.join(('WA', str(pin_number), ':',
            str(analog_value)))).encode()
        self.conn.write(command)
```

Scrive un valore analogico (onda PWM) su un PIN. Può essere utilizzato per accendere un LED a luminosità variabile o azionare un motore a varie velocità. Dopo una chiamata ad analogWrite (), il PIN genererà un'onda rettangolare costante del duty cycle specificato fino alla chiamata successiva ad analogWrite () (o una chiamata a digitalRead () o digitalWrite ()) sullo stesso PIN.

3.5.2.7 string_write()

```
def string_write(self, modalita, tempoacquisizione, ton, impulsi)
:
    "Scrive una stringa nel monitor seriale"
    finalString = "<" + modalita + "," + tempoacquisizione + ","
    + ton + "," + impulsi + ">"
        self.conn.write(finalString.encode("utf-8"))
```

Questo metodo ci permette di scrivere una stringa nella porta seriale nel giusto formato per attivare i comandi START e START N all'interno di Arduino, così da attivare le funzioni startREC() e startNImpulsi() nel codice.

3.5.2.8 readlineArduinoState()

```
def readlineArduinoState(self):
    "Legge lo stato di Arduino, ovvero l'ultima riga del seriale
"
    msg = self.conn.readline()
    return msg
```

Questa funzione della classe Arduino ci permette di leggere le stringhe che vengono scritte sulla porta seriale da parte della scheda alla quale siamo collegati.

3.5.2.9 close()

```
def close(self):
    """
    To ensure we are properly closing our connection to the
    Arduino device.
    """
    self.conn.close()
    print ('Connection to Arduino closed')
```

Questa funzione ci permette di chiudere la connessione con la porta seriale.

3.5.3 IlMioThread()

Questo thread ci permette di 'domandare' continuamente ad Arduino il suo stato. Nel codice di Arduino abbiamo visto che ogni volta che partono i trigger dalla scheda nella porta viene stampato il valore 1, mentre quando i trigger vengono fermati, nel caso della funzione **START**, o si fermano, nel caso della funzione **START** N, viene stampato il valore 0. Il thread ci permette di intercettare questo stato e di salvare la data e l'ora di quando avviene lo start del trigger o lo stop, questi valori poi ci serviranno per ricostruire il video assemblando i frame che vengono raccolti dalla telecamera

```
def run(self):
        while (True):
           time.sleep(1)
3
           stato = self.a.readlineArduinoState()
           stato = stato [:-2]
            if str(stato, 'utf-8') = '1' :
               self.id = self.id + 1
               print(self.id)
               now = datetime.now()
9
               print(now.day)
               print(now.month)
11
               print(now.year)
               print(time.time()) #epoctime
13
               # dd/mm/YY H:M:S
              \#dt \ string = now.strftime("\%d/\%m/\%Y \%H:\%M:\%S")
               #print("Inizio =", dt string)
               flag = True
17
               while (flag) :
                  stato = self.a.readlineArduinoState()
                  stato = stato [:-2]
                  if str(stato, 'utf-8') = '0' :
21
                     now = datetime.now()
                     print(now.day)
23
                     print(now.month)
                     print(now.year)
25
                     print(time.time()) #epoctime
                     # dd/mm/YY H:M:S
                     #dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
                     \#print("Fine =", dt string)
29
                     flag = False
```

- 1. Il **while** è infinito perché è necessario chiedere continuamente lo stato in cui si trova Arduino.
- 2. Attraverso la funzione **readlineArduinoState()** leggiamo lo stato che viene stampato solamente quando vi è un cambiamento nel funzionamento della scheda.

- Una volta che la variabile stato viene inizializzata applichiamo un controllo condizionale e vediamo se il trigger è iniziato confrontandolo con il valore 1.
- 4. Ora nella variabile **now** salviamo l'istante in cui avviene il cambio di stato e stampiamo poi i valori del giorno, del mese, dell'anno, e del tempo in epoctime.
- 5. Successivamente intercettiamo quando il segnale viene stoppato, quindi quando stato diventa 0 grazie alla funzione readlineArduinoState().
- 6. Una volta intercettato l'istante in cui avviene il cambio di stato in 0, stampiamo l'istante di fine sessione come abbiamo fatto per il primo if.

3.6 Generazione video

Grazie alla libreria **ffmpeg** è possibile "assemblare" i frame, che vengono generati dalla telecamera e salvati in una cartella dall'applicazione, per generare un video.

3.6.1 makevideo(path,framerate_num)

Il modulo **makevideo** ci permette di sfruttare la libreria ffmpeg per creare i video partendo dai frame; specificando il **path** ovvero la cartella dove si trovano questi frame, passato come parametro, e il **framerate**, anch'esso passato come parametro, prende i frame contenuti nella cartella e li assembla formando un video in formato .**mp4**.

Capitolo 4 Panormaica dell'applicazione web

L'interfaccia della nostra applicazione web si presenta in questo modo: Possiamo

SWIM4All Video				
► 0.00 /3:40 ◆ → ★ ::				
Controllo Arduino:				
Tempo Acquisizione:	6 ms			
Invia STOP				

gestire nella nostra app nella stessa schermata sia la sincronizzazione dei tre video, sia la gestione di Arduino attraverso l'inserimento dei dati della Form ed effettuando il submit attraverso il bottone Invia.

4.1 Gestione Video

Nella gestione dei video possiamo notare la presenza di tre video, con i comandi presenti unicamente nel primo slot a sinistra; nel caso in cui si prema play nel primo avverrà la riproduzione anche negli altri due, lo stesso se premiamo pause o spostiamo il cursore del video.



4.2 Form Arduino

La form per inserire i dati che verranno inviati ad Arduino si presenta così. Inserendo i dati, scegliendo la modalità e premendo il tasto invia possiamo attivare i trigger che poi Arduino destinerà alla telecamera. Premendo STOP, quando siamo in modalità START N, fermiamo i triggger.

Controllo Arduino:

Scegli la modalità ST/	ART N ~	
Tempo Acquisizione:		 🖨 ms
tOn:	🖨 ms	
Numero impulsi:		\$
Invia		
STOP		

Capitolo 5 Conclusioni

Lo sviluppo di questo progetto mi ha portato ad affrontare un nuovo linguaggio, ovvero Python, ed ha reso possibile vedere come diversi tipi di linguaggi possono essere combinati al fine di ottenere un risultato complesso. Un'altra novità dalla mia parte è stata l'utilizzo di Arduino, sia lato hardware che lato software, mi ha permesso di comprendere come funzionino questo tipo di schede.

Il progetto poteva essere sviluppato in altri ambienti, quindi potevano essere portato avanti anche in modo diverso. L'utilizzo di Flask ha facilitato il lavoro, infatti il ruolo di questo framework è proprio quello di facilitare lo sviluppo di un'applicazione web.

5.1 Miglioramenti

I miglioramenti che possono essere fatti sono:

- l'Applicazione Web può essere migliorata sicuramente nell'interfaccia, in quanto lo sviluppo effettuato fino a questo momento in questo settore risulta essere molto scarno e semplice.
- Il codice in generale sia di Arduino, che Pyhton può essere ottimizzato, tra tutti possiamo migliorare il codice JavaScript utilizzando per esempio jQuery che facilita la scrittura di codice ed ha anche una sintassi più breve.
- Nel codice Arduino potremmo utilizzare una sintassi più semplice per la stringa che viene inviata nella porta seriale, così il comando che ne legge il contenuto diventerebbe più semplice.
- Inserire dei controlli nella form in maniera tale che non ci siano errori di input nelle zone dedicate, ad esempio, l'inserimento del tempo di acquisizione in millisecondi.
- Inserire nell'app.py un elemento che permetta di vedere live lo stato di Arduino, ovvero se in esecuzione o meno.

• Gestione dell'archivio dei video che vengono prodotti dalla telecamera a cui siamo collegati, ed un sistema di scelta del video che si vuole riprodurre nell'applicazione.

Bibliografia

- [1] https://code.visualstudio.com/.
- [2] https://en.ids-imaging.com/store/ui-3160cp-rev-2-1.html.
- [3] https://ffmpeg.org/.
- [4] https://httpd.apache.org/.
- [5] https://it.wikipedia.org/wiki/client.
- [6] https://it.wikipedia.org/wiki/c_(linguaggio).
- [7] https://it.wikipedia.org/wiki/css.
- [8] https://it.wikipedia.org/wiki/html.
- [9] https://it.wikipedia.org/wiki/javascript.
- [10] https://it.wikipedia.org/wiki/server.
- [11] https://it.wikipedia.org/wiki/sistema_client/server.
- [12] https://it.wikipedia.org/wiki/standard_qeneralized_markup_language.
- [13] https://palletsprojects.com/p/flask/.
- [14] https://store.arduino.cc/arduino-mkr-wan-1300-lora-connectivity-1414.
- [15] http://swim4all.com.au/swim-4-all-story/.
- [16] https://wvganjana.medium.com/how-web-browsers-use-process-threads-67e156abb3c5.
- [17] https://www.arduino.cc/en/software.
- [18] https://www.mozilla.org/it/firefox/new/.
- [19] https://www.python.org/.

Elenco delle figure

2.1	Telecamera IDS UI-3160CP Rev. 2.1	7
2.2	Arduino MKR WAN 1300	8
2.3	Architettura Client-Server	9
2.4	Esempio di Web browser Firefox	0
2.5	Architettura Web Browser Firefox	1
2.6	Architettura Web Server Apache	3
2.7	Interfaccia Visual Studio Code	6
2.8	Interfaccia Arduino IDE	7
		_
3.1	Workflow del sistema	8