UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un sistema di visione multi-camera per la registrazione di video di nuotatori nell'ambito del progetto SWIM4ALL

Design and Development of a multi-camera vision system to record video of swimmers in the context of the SWIM4ALL project

Relatore:
Ch.mo Prof. Mancini Adriano

Laureando:
Simoncini Iacopo

Anno Accademico 2019-2020

Indice

Capitolo 1

Introduzione

1.1 Obiettivo

Lo svolgimento di questo progetto di tesi è incentrato sullo sviluppo di un software che consenta di acquisire immagini da più camere per la registrazione di video di nuotatori nell'ambito del progetto SWIM4ALL. Il software deve dare la possibilità all'utente di settare i parametri delle camere e scegliere la struttura della directory di salvataggio. A supporto del progetto abbiamo un microcontrollore Arduino che trasmette il segnale di trigger per poter sincronizzare l'acquisizione delle immagini.

1.2 Struttura della tesi

La struttura della presente tesi è articolata in modo da costituire anche un supporto utile alla comprensione dell'effettivo funzionamento del software realizzato. Verrà approfondito tutto ciò che riguarda il progetto partendo dai concetti elementari per arrivare alla spiegazione del progetto vero e proprio in ogni dettaglio. Viene quindi strutturata come segue:

- nella prima parte verranno introdotti gli strumenti utilizzati sia a livello software che a livello hardware per il funzionamento e lo sviluppo del codice;
- nella seconda parte verranno spiegate le varie fasi di sviluppo e di progettazione del codice, spiegando in dettaglio tutte le parti fondamentali del progetto;
- nella terza parte verrà spiegato come impostare il dispositivo hardware e le camere per il corretto utilizzo del software;
- nell'ultima parte verranno tratte le conclusioni dopo aver esaminato i test effettuati sui dispositivi hardware elencati precedentemente.

Capitolo 2

Strumenti utilizzati

2.1 Camere IDS

Le camere utilizzate in questo progetto sono state le **IDS UI-3160CP Rev 2.1** [2]. Questo modello si presta molto bene alla tipologia di utilizzo effettuata nel progetto, infatti si ha la possibilità di implementare un software python per gestire tutti i parametri e le acquisizioni delle immagini attraverso la libreria PyuEye [8]. La camera ha due modalità di acquisizione: free run e trigger mode. Quella usata in questo progetto è la seconda. Per utilizzare questa modalità abbiamo bisogno di un supporto esterno per la produzione del segnale di trigger, nel nostro caso abbiamo utilizzato un Arduino. La massima risoluzione della camera è 2.3 MP (1920 x 1200 pixels) con un frame rate massimo di 169 frame al secondo. Ovviamente diminuendo la risoluzione è possibile aumentare il frame rate ad un massimo di 190 fps per una risoluzione di 1920 x 1080. Nel nostro caso abbiamo impostato l'AOI (Area Of Interest) a 920 x 600. I parametri descritti precedentemente sono modificabili grazie al software scrivendoli nel file di config. Per avere le massime prestazioni la camera utilizza un connettore USB 3.0, che rende il trasferimento delle immagini dalla memoria della camera a quella del dispositivo hardware utilizzato molto veloce.

Figura 2.1: IDS UI-3160CP-C-HQ rev 2.1

2.2 Dispositivo Hardware

Una parte fondamentale del progetto è data dal dispositivo hardware che viene utilizzato per eseguire il codice, nel nostro caso abbiamo utilizzato due dispositivi: il **NUC8i7BEH** [9] e il **Raspberry Pi4 4Gbyte** [12].

2.2.1 NUC8i7BEH

Next Unit of Computing (NUC) è una tipologia di personal computer ideata da Intel di dimensioni ridotte rispetto ai computer desktop e maggiormente performanti rispetto ai nettop, permettendo usi anche più gravosi rispetto alle applicazioni d'ufficio, navigazione e multimediale. Paragonata a quella di un desktop tradizionale, l'unità centrale di un NUC computer è più piccola, più leggera, più economica e consuma meno energia. I sistemi operativi supportati dai NUC sono solitamente tutti quelli con supporto all'architettura x86 e generalmente utilizzati nei personal computer. Nel nostro caso abbiamo installato Ubuntu 18.04 LTS per poter usufruire di tutti gli strumenti Linux necessari per lo sviluppo del codice.

Figura 2.2: NUC8i7BEH

2.2.2 Raspberry Pi 4

Il **Raspberry Pi** è un computer a scheda singola sviluppato dalla Raspberry Pi Foundation. La scheda fu progettata per ospitare sistemi operativi basati su kernel Linux e RISC OS. Successivamente fu concepito un sistema operativo dedicato, chiamato Raspberry Pi OS. Questo sistema operativo è basato sui rilasci ufficiali di Debian per l'architettura ARM. Rispetto a Debian, si trova l'interfaccia grafica LightDM, che consente di alleggerire il carico sulla CPU nell'esecuzione del sistema. Oltre a questo abbiamo l'aggiunta in APT di alcune repository, ufficiali e non, contenenti quasi tutti i pacchetti ricompilati in architettura ARM.

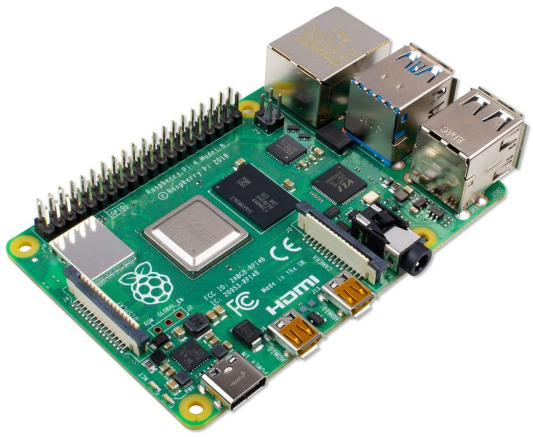

Figura 2.3: Raspberry Pi 4

Architettura ARM Un processore **ARM** fa parte di una famiglia di CPU basate sull'architettura RISC sviluppata da Advanced RISC Machines (ARM).

Essa realizza processori multi-core RISC a 32 e 64 bit. I processori RISC sono progettati per eseguire un numero inferiore di tipi di istruzioni per computer in modo che possano operare ad una velocità maggiore rispetto alle altre architetture. Eliminando le istruzioni non necessarie ed ottimizzando i percorsi, i processori RISC offrono presetazioni eccezionali in una frazione di secondo della richiesta di energia dei dispositivi CISC. I processori ARM sono ampiamente utilizzati nei dispositivi elettrinici di consumo come smartphone, tablet, lettori multimediali e altri dispositivi mobili. A causa del loro set di istruzioni ridotto, richiedono un numero inferiore di transistor, il che consente una dimensione più piccola del circuito integrato. Le dimensioni ridotte del processore ARM, la riduzione della complessità e il ridotto consumo energetico le rendono adatte a dispositivi sempre più miniaturizzati [3].

2.3 Software

In questa sezione verranno illustrati i vari software utilizzati per lo sviluppo del progetto.

2.3.1 IDS Camera Manager

Il software in questione, permette la gestione delle camere collegate al pc. All'interno del software troviamo la lista delle camere connesse con la disponibilità, la tipologia, il Cam.ID, il Dev.ID, il modello e il serial number. Il Cam.ID verrà impostato nella parte di codice python, mentre gli altri parametri sono propri della camera in questione.

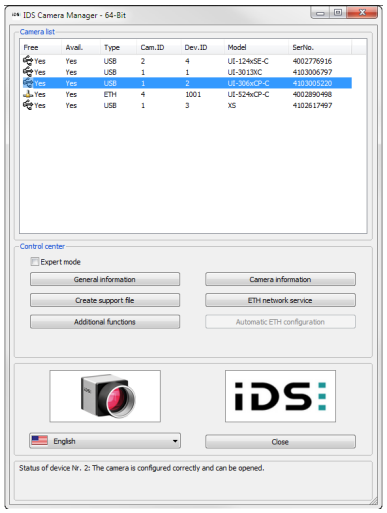

Figura 2.4: IDS Camera Manager

2.3.2 Ueye Demo

Il software Ueye Demo consente di avviare l'acquisizione di immagini con la telecamera uEye. Il programma consente di settare i parametri della camera senza necessità di programmazione e confrontare le acquisizioni fatte dalla camera con impostazioni diverse. Il software si è rivelato utile soprattutto per capire come settare i parametri in modo tale da ottenere gli fps desiderati.

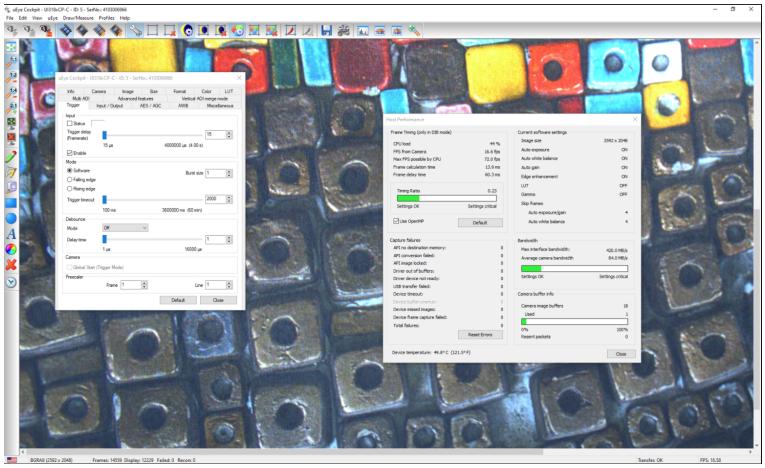

Figura 2.5: Ueye Demo

2.3.3 Visual Studio Code

Per lo sviluppo del codice abbiamo utilizzato l'IDE Visual Studio Code in modalità debug remota. Tramite il protocollo ssh e l'estensione "Remote ssh" [1], è stato possibile scrivere codice ed eseguirlo in modalità debug da remoto, rendendo lo sviluppo più efficiente non avendo il bisogno di essere fisicamente vicini al dispositivo. Per connettersi al dispositivo sarà necessario inserire il nome dell'host, la porta alla quale accedere, il nome utente e la password all'interno del file /.ssh/config, ovvero il file di configurazione del protocollo SSH. Per rendere accessibile la lettura e il download del codice abbiamo creato una repository su GitHub in quanto Visual Studio Code offre la possibilità di connettersi tramite il proprio nome utente e password al proprio profilo.

2.4 Protocollo SSH

SSH (Secure SHell, shell sicura) è un protocollo che permette di stabilire una connessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete informatica. Usando un'architettura del tipo client/server, permette agli utenti di registrarsi in sistemi host server in modo remoto. A differenza

di altri protocolli remoti di comunicazione, come FTP o Tenet, SSH cripta la sessione di login, impedendo alle persone non autorizzate di ottenere le password in chiaro. Il protocollo SSH fornisce le seguenti misure di protezione:

- dopo la prima connessione, il client verifica che il collegamento sia avvenuto con lo stesso server al quale ci si era connessi in precedenza;
- tutti i dati inviati e ricevuti durante le sessioni, vengono trasferiti usando una codifica a 28 bit rendendo complessa la decodifica da parte di estranei.

Tramite il protocollo SSH è possibile rendere sicura una qualsiasi connessione TCP attraverso il **TCP Port Forwarding**. Possiamo suddividere il protocollo SSH in 3 strati:

- **Connection Layer**: divide la connessione in canali logici;
- **User Authentication Layer**: autentica il client al server;
- **Transport Layer**: fornisce autenticazione host, confidenzialità, integrità e compressione.

[5]

2.5 Linguaggio di Programmazione

Il linguaggio di programmazione utilizzato per questo progetto è **Python3** versione 3.7.3. Questo è uno dei linguaggi "di più alto livello" rispetto alla maggior parte dei linguaggi orientati agli oggetti. Python è un linguaggio multi-paradigma, una delle caratteristiche principali sono le variabili non tipizzate e l'uso dell'indentazione per la sintassi delle specifiche, al posto delle più comuni parentesi. Anche se Python viene considerato un linguaggio interpretato, i programmi vengono automaticamente compilati in un formato chiamato bytecode prima dell'esecuzione. Questa caratteristica lo rende molto efficiente, compatto e garantisce prestazioni elevate. Inoltre, come nel caso della libreria PyuEye, diverse strutture dati e funzioni di Python sono implementate in C per rendere il tutto ancora più performante. Queste caratteristiche rendono Python un linguaggio adatto allo scopo del progetto poiché era necessario creare uno script che si potesse lanciare direttamente da terminale, senza bisogno di interfacce grafiche che avrebbero reso il codice più pesante dal punto di vista computazionale [11].

2.5.1 Thread

Un **thread** è un singolo flusso sequenziale di istruzioni all'interno di un programma. La scelta di utilizzare questo strumento per realizzare il progetto deriva dal

fatto che le tre camere dovranno necessariamente eseguire le catture in modo indipendente tra loro. Infatti, i thread permettono l'esecuzione di un pezzo di codice in modo indipendente dal software che lo contiene. L'esecuzione di più thread nello stesso codice fa sorgere il problema della sincronizzazione, soprattutto se i thread condividono delle strutture dati. Nel nostro caso, la prima idea fu di sviluppare un codice con all'interno due thread, uno per la cattura e uno per il salvataggio. Questa scelta ha portato a dover sincronizzare i due thread attraverso un **mutex**, ovvero uno strumento in grado di far rispettare la mutua esclusione tra i due thread. Essendo il thread di cattura molto più veloce rispetto a quello di salvataggio, si aveva una perdita di frame molto alta la quale ci ha portato a dover ottimizzare il codice in modo da non avere più questo problema. Questi problemi verranno affrontati nel capitolo successivo legato alla progettazione e allo sviluppo del progetto [6].

2.6 JSON

Il **JSON** (JavaScript Object Notation) è un formato testuale per la strutturazione di dati, viene prevalentemente usato nell'ambito dei Web Services, poiché abbiamo la possibilità di ottenere la descrizione di una risorsa in un formato semplice da gestire. Pur contenendo il nome Javascript, il formato è del tutto indipendente da esso, possiamo infatti usare questo formato in programmi sviluppati in linguaggi diversi, come nel nostro caso Python. JSON è basato su due strutture:

- un insieme di coppie chiave/valore, che in Python può essere visto come un dizionario;
- un elenco ordinato di valori.

Al suo interno possiamo definire diverse strutture:

- **Value**: può essere un numero, un booleano o una stringa tra virgolette. Queste strutture possono essere annidate.

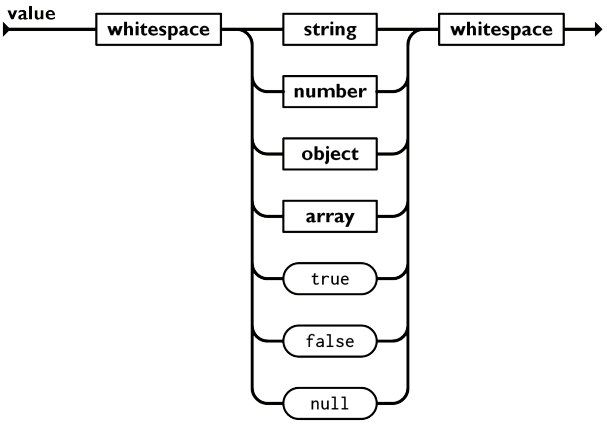

Figura 2.6: Struttura value JSON

- **String**: una raccolta di zero o più caratteri Unicode, tra virgolette.

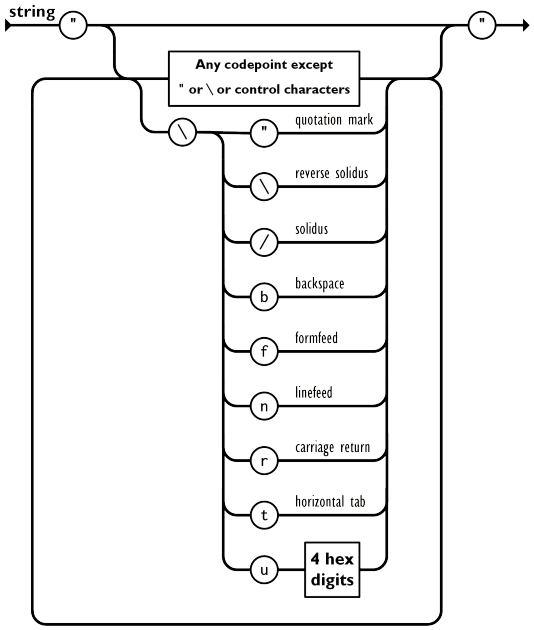

Figura 2.7: Struttura stringa JSON

- **Number**: simile al C e Java, non possono essere rappresentati numeri esadecimali.

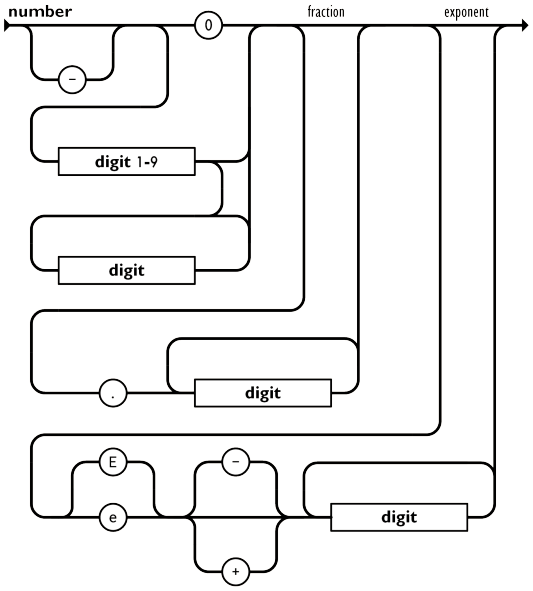

Figura 2.8: Struttura number JSON

- **Array**: raccolta ordinata di valori. Questa struttura comincia con la [e finisce con]. I valori sono separati sempre da una virgola.

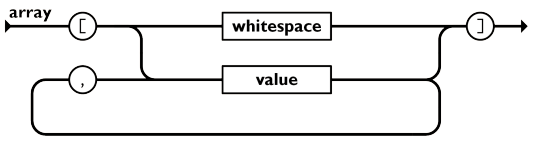

Figura 2.9: Struttura array JSON

- **Object**: una serie non ordinata di chiavi/valori. Questa struttura inizia con e finisce con . Ogni chiave è divisa dal valore dai due punti e ogni coppia è divisa da una virgola [10].

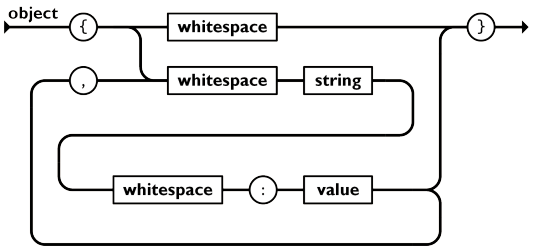

Figura 2.10: Struttura object JSON

Capitolo 3

Progettazione e Sviluppo

3.1 Workflow del progetto

Il progetto complessivo è articolato in diverse fasi. Nella presente tesi viene presa in esame unicamente la fase inerente il software per l'acquisizione e il salvataggio delle foto. L'applicazione legata alla camera è composta di 4 moduli: main.py, utilities.py, thread_capture.py e camera.py. I suddetti moduli, verranno spiegati. In questa sezione verrà illustrato unicamente il comportamento generale del software. Il programma si comporta come un demone, ovvero una volta entrato in esecuzione svolge tutte le sue funzioni senza che sia sotto il controllo diretto dell'utente. Infatti, una volta avviato, aspetta che la camera abbia scattato un'immagine per poi prenderla dalla memoria della camera e salvarla nella memoria fisica del dispositivo sul quale stiamo eseguendo il codice. Escludendo l'impostazione dei parametri della camera che viene effettuato solo una volta tramite l'uso di un file json, il software svolge le sue funzioni in modo completamente autonomo e indipendente.
Di seguito troviamo uno schema che definisce tutte le componenti del progetto.

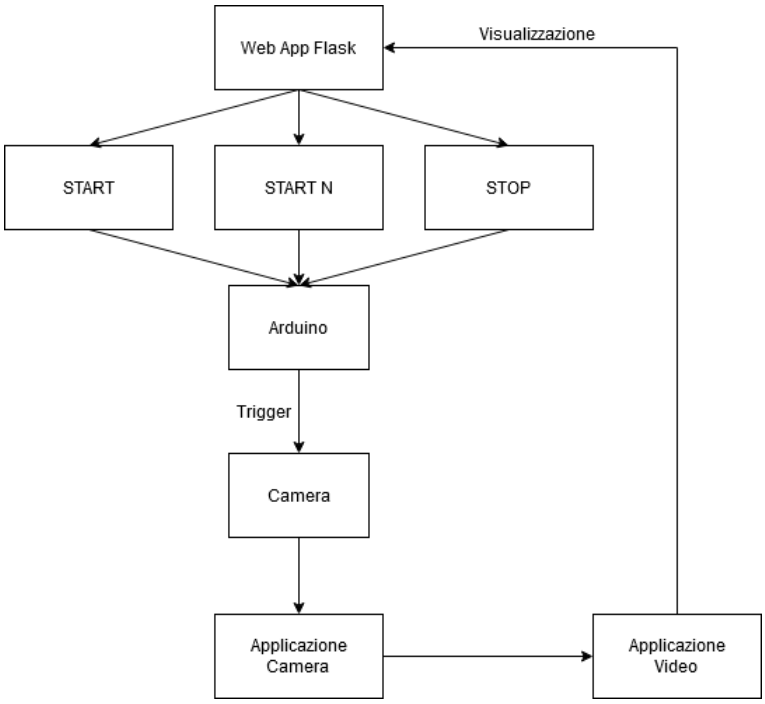

Figura 3.1: Workflow intero del progetto

Le altre componenti, come la generazione del segnale di trigger e la web App, sono state realizzate dal collega Nicola Mori.

3.2 Moduli python

Come specificato in precedenza, il software è suddiviso in 4 moduli python i quali svolgono funzioni differenti per il corretto funzionamento del progetto finale. Grazie a questi moduli abbiamo la possibilità di raggruppare costanti, funzioni e classi, che ci consentono di suddividere e organizzare meglio il nostro progetto. In questa sezione approfondiremo nel dettaglio come sono strutturati i singoli moduli del progetto in questione.

3.2.1 main.py

In questo modulo troviamo il main loop principale che permette al thread di acquisizione di continuare la sua esecuzione fintanto che la classe definita dentro camera.py non rivela un eccezione. Nel **main.py** troviamo anche il modo per interrompere il loop principale e terminare l'esecuzione del thread in modo sicuro, implementato attraverso l'uso del **try**. All'interno della sezione except troviamo **KeyboardInterrupt** ovvero l'interruzione da tastiera provocata dalla pressione dei tasti Ctrl e c.

Listing 3.1: Main loop

```
id = [1, 2, 3]
dev = ["Camera" + str(id[0]), "Camera" + str(id[1]),
       "Camera" + str(id[2])]

Cam0 = camera.Cam(id[0], 10)
Cam0.Setup()

p = tc.Capture_Thread(Cam0)
p.start()

while Cam0.nRet == ueye.IS_SUCCESS:
    try:
        if not p.joined:
            p.join()
            print("p joined")

    except KeyboardInterrupt:
        p.kill()
        break

ueye.is_FreeImageMem(Cam0.cam, Cam0.pcImageMemory, Cam0.MemID)
Cam0.exit()
```

Infatti, premendo questi due tasti il codice passa nella sezione di except dove sono richiamate le funzioni **p.kill()** e **Cam0.exit()**. Queste due funzioni sono rispettivamente per la terminazione del thread e per la chiusura della connessione con la camera.

Listing 3.2: Funzione p.kill() definita nel modulo thread_capture.py

```
def kill(self):
    self.isRunning = False
    self.file.close()
```

La funzione che termina il thread cambia il valore della variabile booleana che viene utilizzata come condizione del ciclo while interno al thread stesso. Nelle sezioni successive parleremo del thread e di come viene impiegato all'interno del codice.

Listing 3.3: Funzione Cam0.exit() definita nel modulo camera.py

```
def exit(self):
    if self.cam is not None:
        self.nRet = ueye.is_ExitCamera(self.cam)
    if self.nRet == ueye.IS_SUCCESS:
        self.cam = None
    print()
    print("Camera exit.")
```

Per chiudere la connessione con la camera c'è bisogno di richiamare la funzione **is_ExitCamera** definita nel modulo PyuEye che restituisce **IS_SUCCESS** se tutto è andato a buon fine. IS_SUCCESS è una costante intera definita sempre nello stesso modulo che ci permette di capire se una funzione ha completato la propria esecuzione senza problemi.

3.2.2 camera.py

In questo modulo è definita la classe **Cam**, ovvero la classe che rappresenta la camera con le proprie funzioni e i propri parametri. La prima funzione è quella di setup della camera, ovvero l'inizializzazione, il setup dei driver e il setup di tutti i parametri specificati all'interno del file config.json. Inoltre, viene allocata l'area di memoria per eseguire l'operazione di cattura dell'immagine e viene settata la modalità di acquisizione dell'immagine.

Listing 3.4: Funzione di allocazione della memoria

```
def alloc(self):
    rect = self.get_aoi()
    bpp = self.nBitsPerPixel

    for buff in self.img_buffer:
        self.nRet = ueye.is_FreeImageMem(self.cam, buff.mem_ptr, buff.mem_id)
        if self.nRet != ueye.IS_SUCCESS:
            error_log(self.nRet, "is_FreeImageMem")

    self.img_buffer = []

    for i in range(self.buffer_count):
        buff = ImageBuffer()
        ueye.is_AllocImageMem(self.cam,
                              rect.width,
                              rect.height,
                              bpp,
                              buff.mem_ptr,
                              buff.mem_id)
        ueye.is_AddToSequence(self.cam, buff.mem_ptr, buff.mem_id)
        self.img_buffer.append(buff)

    self.nRet = ueye.is_InitImageQueue(self.cam, 0)
    if self.nRet != ueye.IS_SUCCESS:
        error_log(self.nRet, "is_InitImageQueue")
```

Le modalità sono due: la **free run mode** e la **trigger mode**. Queste due modalità possono essere impostate a seconda del valore passato alla funzione capture_video definita sempre nello stesso modulo.

Listing 3.5: Funzione per definire la modalità di acquisizione

```
def capture_video(self, wait=True):
    wait_param = ueye.IS_WAIT if wait else ueye.IS_DONT_WAIT
    return ueye.is_CaptureVideo(self.cam, wait_param)
```

Altre due funzioni da dover menzionare sono la **capture_status** e la **trigger_miss**. Queste due funzioni si occupano rispettivamente di restituire qual è lo stato della cattura e se è stato perso un frame.

Listing 3.6: Funzioni capture_status e trigger_miss

```
def capture_status(self):
    info = ueye.c_uint()
    nRet = ueye.is_CaptureStatus(self.cam,
                                 ueye.IS_CAPTURE_STATUS_INFO_CMD_GET,
                                 info,
                                 ueye.sizeof(info))
```

```
7       return nRet
8
9   def trigger_miss(self):
10      error_code = ueye.is_CameraStatus(self.cam,
11                                        ueye.IS_TRIGGER_MISSED,
12                                        ueye.IS_GET_STATUS)
13      print("error_code: ",error_code)
```

Sono state usate principalmente durante la fase di debug poiché successivamente è stato implementato un metodo per capire se un frame è stato perso all'interno del thread di acquisizione.

3.2.3 thread_capture.py

In questo modulo troviamo il thread per l'acquisizione e per il salvataggio delle foto. Nella prima versione del codice veniva semplicemente acquisito l'array della foto dalla camera e aggiunto ad una queue che veniva poi passata al thread per il salvataggio.

Listing 3.7: Codice di cattura dell'immagine prima versione

```
1   def Capture(self, queue):
2       t_old = timeit.default_timer()
3
4       self.nRet = ueye.is_FreezeVideo(self.cam, ueye.IS_WAIT)
5       if self.nRet != ueye.IS_SUCCESS:
6           raise uEyeException(self.nRet)
7
8       t = timeit.default_timer()
9       self.FPS = 1/(t - t_old)
10      t_old = t
11
12      array = ueye.get_data(self.pcImageMemory,
13                            self.width, self.height,
14                            self.nBitsPerPixel,
15                            self.pitch, copy=True)
16
17      queue.append(array.copy())
```

Di seguito verranno spiegate nel dettaglio le funzioni utilizzate nella prima versione del codice:

- **is_FreezeVideo**: aspetta fintanto che il segnale di trigger non cambia stato, bloccando la ripresa e salvando il frame in memoria;
- **get_data**: restituisce l'immagine bloccata dalla funzione precedente sotto forma di array numerico.

Una volta acquisito, l'array viene aggiunto alla queue tramite l'append. Questo sistema impiega parecchio tempo per l'elaborazione, il problema con questa versione del codice è che la funzione is_FreezeVideo era pensata per una sola cattura per poi terminare l'esecuzione. Inoltre, salvare l'immagine su un altro thread compromette le prestazioni del thread di cattura, poiché i due thread devono essere sincronizzati. Infatti, il thread di salvataggio rispetto a quello di cattura

impiega decisamente più tempo per concludere la propria esecuzione, tempo che viene sottratto al thread di cattura.
Per ovviare a questo problema abbiamo utilizzato funzioni diverse definite nel modulo PyuEye proprio per questo scopo.

Listing 3.8: Codice principale della cattura e del salvataggio dell'immagine

```
if not os.path.isdir("/home/pi/swim4all/Tirocinio/Photo/" +
                     str(self.x.day) + "-" +
                     str(self.x.month) + "-" +
                     str(self.x.year) + "/"):
    os.makedirs("/home/pi/swim4all/Tirocinio/Photo/" +
                str(self.x.day) + "-" +
                str(self.x.month) + "-" +
                str(self.x.year) + "/")

tempo_conteggio = time.time()

count = 0

while self.isRunning:
    t_old = time.time()
    img_buffer = ImageBuffer()
    self.cam.nRet = ueye.is_WaitForNextImage(self.cam.cam,
                                             self.timeout,
                                             img_buffer.mem_ptr,
                                             img_buffer.mem_id)

    if self.cam.nRet == ueye.IS_SUCCESS:
        mem_info = MemoryInfo(self.cam.cam, img_buffer)
        array = ueye.get_data(img_buffer.mem_ptr,
                              mem_info.width,
                              mem_info.height,
                              mem_info.bits,
                              mem_info.pitch,
                              copy=True)
        self.cam.unlock_seq(img_buffer.mem_id, img_buffer.mem_ptr)

        if self.cam.mode_filename == 1:
            filename = "/home/pi/swim4all/Tirocinio/Photo/" +
                       str(self.x.day) + "-" +
                       str(self.x.month) + "-" +
                       str(self.x.year) + "/" +
                       str(self.cam.camID) + "-" +
                       str(time.time()) + ".png"
        else:
            filename = "/home/pi/swim4all/Tirocinio/Photo/" +
                       str(self.x.day) + "-" +
                       str(self.x.month) + "-" +
                       str(self.x.year) + "/" +
                       str(time.time()) + "-" +
                       str(self.cam.camID) + ".png"

        self.file_param.pwchFileName = filename
        self.file_param.nFiletype = ueye.IS_IMG_PNG
        self.file_param.ppcImageMem = None
        self.file_param.pnImageId = None
        nRet = ueye.is_ImageFile(self.cam.cam,
                                 ueye.IS_IMAGE_FILE_CMD_SAVE,
                                 self.file_param,
                                 ueye.sizeof(self.file_param))
        if nRet != ueye.IS_SUCCESS:
            error_log(nRet, "is_ImageFile")
```

```
57              if not self.file.closed:
58                  self.file.write("FPS: " +
59                                  "Salvataggio non riuscito" + "\n")
60          else:
61              t = time.time()
62              self.FPS = 1 / (t - t_old)
63              if not self.file.closed:
64                  self.file.write("FPS: " + str(self.FPS) + "\n")
65      else:
66          error_log(self.cam.nRet, "is_WaitForNextImage")
67          if not self.file.closed:
68              self.file.write("FPS: " + "Frame perso" + "\n")
```

In questo modo la differenza sostanziale consiste nella modalità di cattura che viene modificata in trigger mode tramite la funzione **is_CaptureVideo** definita nel modulo camera.py. Inoltre, la funzione **is_WaitForNextImage** aspetta fintanto che il trigger non cambia di stato. In caso di successo, cioè quando il trigger ha cambiato stato, si seleziona l'area di memoria in cui è salvata l'immagine grazie alla classe **MemoryInfo** e si salva l'immagine in un file .png.La funzione aspetta che arrivi un'altra immagine in memoria ma se il frame non arriva entro un certo tempo ovvero il timeout, definito nella sezione **__init__** del thread, viene restituito il codice di errore **IS_TIMED_OUT**.
Un'affinità con il codice precedente è la funzione get_data, questa funzione è effettivamente superflua poiché l'array non viene salvato, ma comunque c'è la possibilità per uno sviluppo futuro di processare l'immagine con qualche algoritmo ad esempio di rilevazione di oggetti. Questo processo rappresenta l'acquisizione di un frame e viene eseguito fintanto che la variabile all'interno del ciclo while del thread è True. Dal punto di vista del frame rate, abbiamo deciso di misurare il tempo iniziale e il tempo finale di acquisizione di un frame tramite la libreria timeit. Per calcolare il frame rate basta fare l'inverso della differenza tra il tempo finale e il tempo iniziale. Il valore del calcolo viene salvato in un file di testo generato all'inizio dell'esecuzione del thread.

3.2.4 utility.py

In questo modulo sono state inserite tutte le classi di supporto per la realizzazione del progetto. La prima è la classe che lancia l'eccezione di PyuEye quando questi si verificano, principalmente utilizzata nella fase di debug del codice.

Listing 3.9: classe per la gestione delle eccezioni in fase di debug

```
1   class uEyeException(Exception):
2   def __init__(self, error_code, name_funct):
3       self.error_code = error_code
4       self.name = name_funct
5   def __str__(self):
6       return self.name + ": " + str(self.error_code)
```

A sostituzione di questa classe troviamo **error_log**, ovvero, una funzione che scrive nel syslog linux l'errore che si potrebbe presentare qualora una funzione non restituisca il valore desiderato. In seguito verrà spiegato il syslog e il suo

funzionamento nell'intero progetto. Come spiegato in precedenza, i parametri della camera verranno presi da un file di configurazione all'interno della directory del progetto, la funzione che raccoglie questi parametri è **param_from_json**, anch'essa descritta all'interno del modulo utility.

Listing 3.10: funzione per caricare i parametri della camera

```
def param_from_json(cam):
    with open('/home/pi/swim4all/Tirocinio/Tirocinio/config.json') as json_file:
        data = json.load(json_file)
        for j in data['config']:
            cam.rectAOI.s32Width = ueye.int(j['width'])
            cam.rectAOI.s32Height = ueye.int(j['height'])
            cam.rectAOI.s32X = ueye.int(j['x'])
            cam.rectAOI.s32Y = ueye.int(j['y'])
            cam.exposure = j['exp']
            cam.pixelclock = j['pixclock']
            cam.current_fps = j['fps']
            cam.gain = j['gain']
            cam.rGain = j['rGain']
            cam.bGain = j['bGain']
            cam.gGain = j['gGain']
            cam.mode_filename = j['directory_structure']
            cam.m_nColorMode = j['color_mode']
```

Di seguito verranno spiegate il funzionamento delle classi restanti nel file:

- **Rect**: classe che acquisisce i dati relativi alla AOI di una camera;
- **ImageBuffer**: collegata all'acquisizione delle immagini, ci permette di stabilire il puntatore e l'id dell'area di memoria allocata precedentemente durante l'inizializzazione della camera;
- **MemoryInfo**: classe che permette di salvare i parametri relativi all'AOI e permette di leggere i parametri relativi all'area di memoria.

Queste tre classi sono fondamentali per l'acquisizione delle immagini all'interno del thread, infatti, permettono alle funzioni is_WaitForNextImage e get_data di sapere dove si trova la locazione di memoria nella quale è salvata l'immagine.

Listing 3.11: classi Rect, ImageBuffer e MemoryInfo

```
class Rect:
    def __init__(self, x=0, y=0, width=0, height=0):
        self.x = x
        self.y = y
        self.width = width
        self.height = height

class ImageBuffer:
    def __init__(self):
        self.mem_ptr = ueye.c_mem_p()
        self.mem_id = ueye.int()

class MemoryInfo:
    def __init__(self, cam, img_buffer):
        self.x = ueye.int()
        self.y = ueye.int()
```

```
17          self.bits = ueye.int()
18          self.pitch = ueye.int()
19          self.img_buff = img_buffer
20          rect_aoi = ueye.IS_RECT()
21          nRet = ueye.is_AOI(cam,
22                             ueye.IS_AOI_IMAGE_GET_AOI,
23                             rect_aoi,
24                             ueye.sizeof(rect_aoi))
25
26          if nRet != ueye.IS_SUCCESS:
27              error_log(nRet, "is_Aoi")
28
29          self.width = rect_aoi.s32Width.value
30          self.height = rect_aoi.s32Height.value
31          nRet = ueye.is_InquireImageMem(cam, self.img_buff.mem_ptr,
32                                         self.img_buff.mem_id, self.x,
33                                         self.y,
34                                         self.bits, self.pitch)
35
36          if nRet != ueye.IS_SUCCESS:
37              error_log(nRet, "is_InquireImageMem")
```

3.3 Configurazione camera

3.3.1 config.json

Nel file di configurazione config.json è possibile settare i diversi parametri della camera quali pixelclock, esposizione, gain, ecc. In questa sezione verranno spiegati i parametri e i possibili valori da utilizzare per il corretto funzionamento della camera. Width e height rappresentano le dimensioni dell'Area Of Interest mentre x e y rappresentano la posizione dell'AOI. Questi sono parametri fissi cioè indipendenti dall'esposizione, pixelclock e dagli fps, tuttavia per avere prestazioni migliori è consigliabile ridurre la dimensione dell'AOI in modo da rendere la computazione dell'immagine più leggera per la CPU. Per quanto riguarda l'esposizione e il frame rate, questi variano in funzione del pixelclock e del gain, dunque per settare questi parametri c'è bisogno di fare dei tentativi e trovare la soluzione che ci è più congeniale in termini di prestazioni e di resa dell'immagine. L'inserimento di valori errati all'interno del file di config viene gestito dal codice Python che provvederà all'inserimento di valori consentiti dalla camera. Questo vale per i valori come il pixelclock, che ha un range da 200 a 400, per l'esposizione e per gli fps. Questi ultimi sono "settabili" solamente in modalità free run e non in trigger mode, se vogliamo modificare il valore degli fps in trigger mode dobbiamo necessariamente modificare la durata del trigger. Gli ultimi due parametri rappresentano rispettivamente la struttura delle directory per il salvataggio dell'immagine e la color mode. Se settiamo directory_structure ad un valore diverso da 1, avremo come struttura del nome dell'immagine l'istante di tempo in Linux epoch time in cui è stato catturata l'immagine e l'id della camera che l'ha acquisita. Settando il valore a 1, avremo come struttura l'id della camera e l'istante di tempo sempre in epoch time. Per quanto riguarda la

color_mode abbiamo la possibilità di inserire solo uno tra i possibili valori che verranno descritti nella tabella seguente, altrimenti si verificherà un'eccezione che verrà segnalata nel syslog.

color_mode	value	Available
IS_CM_SENSOR_RAW8	11	Yes
IS_CM_SENSOR_RAW10	33	No
IS_CM_SENSOR_RAW12	27	No
IS_CM_SENSOR_RAW16	29	No
IS_CM_MONO8	6	Yes
IS_CM_RGB8_PACKED	129	Yes
IS_CM_BGR8_PACKED	1	No
IS_CM_RGBA8_PACKED	128	Yes
IS_CM_BGRA8_PACKED	0	Yes
IS_CM_BGR10_PACKED	25	No
IS_CM_RGB10_PACKED	153	Yes
IS_CM_BGRA12_UNPACKED	31	No
IS_CM_BGR12_UNPACKED	30	No
IS_CM_BGRY8_PACKED	24	Yes
IS_CM_BGR565_PACKED	2	Yes
IS_CM_BGR5_PACKED	3	Yes
IS_CM_UYVY_PACKED	12	Yes
IS_CM_UYVY_MONO_PACKED	13	Yes
IS_CM_UYVY_BAYER_PACKED	14	Yes
IS_CM_CBYCRY_PACKED	23	Yes

Tabella 3.1: Tabella valori color_mode

I valori nella tabella 3.1 rappresentano tutti i colori possibili per le camere IDS, quindi non tutti i valori sono validi per la camera utilizzata nel progetto. Infatti, abbiamo inserito la terza colonna per far capire al lettore quali sono i colori validi e quali non lo sono. L'inserimento di un valore non valido genererà un codice di errore per quanto riguarda il salvataggio che verrà inserito all'interno del syslog.

Listing 3.12: File JSON per la configurazione della camera

```
{
    "config":[
        {
            "width": 960,
            "height": 600,
            "x": 0,
            "y": 0,
            "exp": 5,
            "pixclock": 400,
            "fps": 50,
            "gain": 50,
            "rGain": 0,
            "bGain": 0,
            "gGain": 0,
            "directory_structure": 1,
            "color_mode": 0
        }
]}
```

3.4 Particolari soluzioni adottate

3.4.1 Modulo PyuEye

Il modulo PyuEye permette di implementare complesse funzioni scritte in C all'interno del codice Python, questi tipi di moduli vengono chiamati "wrapper". Un wrapper permette la decodifica di librerie scritte in linguaggi diversi da quello che stiamo utilizzando per scrivere un programma. PyuEye consente di utilizzare tutte le funzionalità della camera quali l'impostazione dei parametri (gain, exposure, focus, ecc.), l'attivazione dei driver, l'inizializzazione della camera, la cattura in trigger mode, ecc. In sostanza questo è il modulo che ci permette di interagire con la camera e con tutte le sue funzionalità.

3.4.2 Modulo threading

Il modulo threading di Python permette la creazione di thread e la sincronizzazione di questi ultimi. Un thread è una parte di codice del programma che viene eseguito in maniera indipendente dal programma stesso. Vediamo nel dettaglio come funziona questo modulo:

- Innanzitutto bisogna importare il modulo threading.

```
1         import threading
2
```

- Una volta importato il modulo possiamo passare alla effettiva creazione del thread. Nel nostro progetto abbiamo definito nel modulo thread_capture una classe che estende la classe thread, dunque quello è il nostro thread. Per inizializzarlo c'è bisogno di istanziare un oggetto della classe Capture_Thread, successivamente possiamo richiamare il metodo start() per mandare in esecuzione il nostro thread.

```
1         import thread_capture as tc
2         p = tc.Capture_Thread(Cam0)
3         p.start()
4
```

Abbiamo scelto di utilizzare i thread poiché in questo modo è possibile utilizzare più di una camera ed eventualmente poter sincronizzare i thread di cattura delle singole camere.

3.4.3 Modulo syslog

SYSLOG (System Log) è un protocollo di rete particolarmente diffuso in Unix, appartenente alla suite di protocolli Internet utilizzato per trasmettere attraverso una rete, semplici informazioni di log. In generale, il client invia un messaggio di testo al server chiamato "syslogd", "sislog daemon" o "syslog server". Il server riesce a gestire i messaggi provenienti da diverse tipologie di macchine grazie alla semplicità del protocollo. In questo progetto abbiamo implementato l'invio di questi messaggi grazie al modulo syslog che permette di scrivere messaggi di errore direttamente all'interno del file di log contenuto all'interno della directory /var/log/syslog. Gli errori hanno la seguente struttura: nome della funzione che non restituisce il valore corretto accostato al valore che restituisce la funzione.

```
1     syslog.openlog(logoption=syslog.LOG_PID,facility=syslog.LOG_ERR)
2     syslog.syslog(syslog.LOG_ERR,
3                   str(name_funct) + ": " + str(error_code))
```

In generale, la gestione degli errori riguarda quasi esclusivamente le funzioni del modulo PyuEye in quanto la maggior parte delle funzioni restituisce un valore nRet che identifica un codice di errore il quale ci indica se la funzione ha terminato con successo l'esecuzione o meno.

3.5 Codici di errore

Numero	Errore	Descrizione
-1	IS_NO_SUCCESS	Messaggio di errore generale
0	IS_SUCCESS	Funzione eseguita con successo
2	IS_IO_REQUEST_FAILED	Una richiesta di I/O dal driver uEye non è riuscita
3	IS_CANT_OPEN_DEVICE	Il tentativo di inizializzare o selezionare la telecamera non è riuscito
11	IS_CANT_OPEN_REGISTRY	Errore durante l'apertura di una chiave di registro di Windows
12	IS_CANT_READ_REGISTRY	Errore durante la lettura di una chiave di registro di Windows
15	IS_NO_IMAGE_MEM_ALLOCATED	Il driver non è riuscito ad allocare memoria
16	IS_CANT_CLEANUP_MEMORY	Il driver non è riuscito a rilasciare la memoria
17	IS_CANT_ COMMUNICATE_WITH_DRIVER	Comunicazione con il driver fallita perchè non ci sono driver caricati
18	IS_FUNCTION_NOT_ SUPPORTED_YET	La funzione non è ancora supportata
30	IS_INVALID_IMAGE_SIZE	Dimensione dell'immagine non valida
32	IS_INVALID_CAPTURE_MODE	La funzione non può essere eseguita nella modalità corrente (free run, trigger or standby)
49	IS_INVALID_MEMORY_POINTER	Puntatore non valido o ID memoria non valido
50	IS_FILE_WRITE_OPEN_ERROR	Il file non può essere aperto per la scrittura o la lettura
51	IS_FILE_READ_OPEN_ERROR	Il file non può essere aperto.
52	IS_FILE_READ_INVALID_ BMP_ID	Lo specifico file non è di un bitmap
53	IS_FILE_READ_INVALID_ BMP_SIZE	La dimensione del bitmap non è corretta

Tabella 3.2: Tabella dei codici di errore

Numero	Errore	Descrizione
108	IS_NO_ACTIVE_IMG_MEM	Nessuna memoria immagine attiva disponibile.
112	IS_SEQUENCE _LIST_EMPTY	L'elenco delle sequenze è vuoto e non può essere eliminato
113	IS_CANT_ ADD_TO_SEQUENCE	La memoria immagini è già inclusa nella sequenza e non può essere aggiunta di nuovo
117	IS_SEQUENCE_BUF_ ALREADY_LOCKED	La memoria non può essere bloccata. Il puntatore al buffer non è valido
118	IS_INVALID_ DEVICE_ID	L'ID del dispositivo non è valido
119	IS_INVALID_BOARD_ID	L'ID della scheda non è valido. Gli ID validi sono compresi tra 1 e 255
120	IS_ALL_DEVICES_BUSY	Tutte le telecamere sono in uso
122	IS_TIMED_OUT	Si è verificato un timeout
123	IS_NULL_POINTER	Array non valido
125	IS_INVALID_PARAMETER	Uno dei parametri inviati è al di fuori dell'intervallo valido o non è supportato per questo sensore o non è disponibile in questa modalità
127	IS_OUT_OF_MEMORY	Non può essere allocata alcuna memoria
129	IS_ACCESS_VIOLATION	Si è verificato un errore interno
139	IS_NO_USB20	La fotocamera è collegata a una porta che non supporta lo standard USB 2.0 ad alta velocità
140	IS_CAPTURE_RUNNING	È in corso un'operazione di acquisizione e deve essere prima terminata

Tabella 3.3: Tabella dei codici di errore

Le tabelle sopra riportate contengono i codici di errore che si sono presentati con maggiore frequenza durante i test del software. É possibile consultare la tabella completa, scaricando la suite di pacchetti completi dal sito della IDS.

Capitolo 4

Test e set up del progetto

4.1 Differenze tra NUC e Raspberry Pi 4

In questa sezione elencheremo le differenze tra il NUC e il Raspberry Pi 4 e come settare i parametri della camera per ottenere le prestazioni volute.

4.1.1 Test effettuati con NUC

I test effettuati con il NUC hanno coinvolto la fase di debug dell'applicazione. Infatti, durante tutto lo sviluppo del codice è stato utilizzato questo dispositivo poiché le prestazioni del NUC sono maggiori rispetto al Raspberry Pi 4. L'obiettivo fondamentale era raggiungere i 75 fps in trigger mode, ciò è stato possibile con il NUC anche impostando valori che avrebbero rallentato la normale esecuzione del codice. Di seguito è riportata la tabella con la configurazione usata per i test.

Width	960
Height	600
X	0
Y	0
Exposure	5
Pixelclock	400
Gain	50
Red gain	0
Blue gain	0
Green gain	0
Color mode	0

Tabella 4.1: Valori parametri della camera durante i test

Il valore degli fps viene settato tramite il trigger con valore di tempo di acquisizione pari a 13 millisecondi e 1 millisecondo come tempo in cui il segnale rimane nello stato on, raggiungendo un frame rate di 75 frame per secondo. Il calcolo dei frame effettivo viene fatto all'interno del codice calcolando il tempo intercorso tra un frame e l'altro, per avere una precisione migliore si è calcolato il tempo per catturare 100 frame, dividendo 100 per il tempo trascorso si ottengono i frame per secondo. I test sono stati effettuati cambiando anche i valori dell'esposizione, del pixelclock e della color mode, ovviamente le prestazioni cambiano diminuendo il valore dell'esposizione, impostando la color mode a IS_CM_SENSOR_RAW8 e tenendo a 400 il pixelclock. In generale, il NUC ottiene risultati più che soddisfacenti e, portando il trigger anche ad avere fps maggiori di 75, abbiamo comunque una situazione stabile, senza cali di frame rate e senza perdita di frame.

4.1.2 Test effettuati con Raspberry Pi 4

I test effettuati con il Raspberry Pi 4 sono stati effettuati con gli stessi parametri utilizzati per quelli descritti in precedenza. L'obiettivo anche in questo caso era quello di raggiungere i 75 fps in trigger mode. Rispetto al NUC abbiamo provato anche altri parametri poiché quelli utilizzati in precedenza non davano risultati positivi e abbiamo notato che cambiando la color_mode le prestazioni migliorano o peggiorano a seconda della modalità inserita. Di seguito troviamo la tabella con tutte le color_mode provate durante i test.

Color Mode	FPS
IS_CM_SENSOR_RAW8	77.64
IS_CM_MONO8	76.95
IS_CM_RGB8_PACKED	54.77
IS_CM_RGBA8_PACKED	24.75
IS_CM_RGB10_PACKED	50.00
IS_CM_BGR565_PACKED	64.45
IS_CM_BGR5_PACKED	56.7
IS_CM_UYVY_PACKED	56.44
IS_CM_UYVY_MONO_PACKED	58.28
IS_CM_UYVY_BAYER_PACKED	58.3

Tabella 4.2: Tabella dei risultati dei test con color_mode differenti

Dalla tabella possiamo rilevare che il valore degli fps cambia a seconda della color_mode scelta. In generale, l'obiettivo prefissato viene raggiunto soltanto da certi parametri che sono IS_CM_SENSOR_RAW8 e IS_CM_MONO8. Tuttavia, lo scopo finale del progetto era quello di raggiungere quel frame rate indipendentemente dalla color mode impostata.

4.2 Set up dei parametri

Come accennato in precedenza nella sezione legata al file config.json, i parametri della camera sono:

- **width**: larghezza dell'area of interest;
- **height**: altezza dell'area of interest;
- **x**: posizione dell'area of interest rispetto all'asse x;
- **y**: posizione dell'area of interest rispetto all'asse y;
- **exp**: tempo di esposizione, ovvero il tempo durante il quale l'otturatore della fotocamera rimane aperto per permettere alla luce di raggiungere il sensore [4];
- **pixclock**: pixel clock, Numero massimo di pixel generati al secondo. A seconda del numero di pipeline, per ogni ciclo di clock, vengono renderizzati un numero corrispondente di pixel. Va da un massimo di 400 ad un minimo di 200;
- **fps**: numero di frame per secondo (utilizzabile solo in free run mode);
- **gain**: sensibilità alla luce del sensore. Misurata in valori numerici da 0 a 100, dove più è grande il numero più il sensore è sensibile alla luce [7];
- **rGain**: gain relativo al colore rosso;
- **gGain**: gain relativo al colore verde;
- **bGain**: gain relativo al colore blu;
- **color_mode**: spiegato in modo approfondito nella sezione 3.2.1 config.json.

Le prestazioni del software sono influenzate dai parametri inseriti. In generale, possiamo dire che per aumentare le prestazioni della camera ci basta diminuire l'AOI, diminuire l'esposizione ed aumentare il pixel clock. Questo è direttamente collegato agli fps della camera in quanto più alto è il valore di pixel clock più alti sono i frame per secondo.
I valori possono essere impostati solamente prima dell'esecuzione del codice. Il caricamento dei parametri all'interno dell'oggetto Cam avviene prima dell'esecuzione del thread. Se il valore non viene impostato correttamente prima dell'esecuzione, il codice esce e scriverà nel syslog l'errore **IS_INVALID_PARAMETER**, con a fianco il nome della funzione che l'ha restituito.

Capitolo 5

Conclusioni

5.1 Differenze tra prima versione e la versione finale

La prima versione del codice era strutturata in modo diverso rispetto alla versione finale in quanto era composta di due thread, uno per la cattura e uno per l'acquisizione. Il problema nasce nel dover gestire la sincronizzazione dei due thread che poteva essere risolto tramite l'uso di un semaforo binario. L'acquisizione della variabile del semaforo binario veniva fatto molto più velocemente dal thread dell'acquisizione rispetto a quello del salvataggio, portando inevitabilmente ad un overflow della queue. In più i frame al secondo non erano quelli desiderati in quanto il thread di salvataggio impiega più tempo per svolgere la propria esecuzione, dovendo effettuare il reshape dell'array per poterlo salvare in png e poi grazie all'utilizzo del modulo PIL, salvare l'array. Per questo motivo è stato necessario strutturare il progetto in modo diverso, scrivendo un unico thread sia per l'acquisizione che per il salvataggio dell'immagine.

La soluzione adottata nella versione finale è risultata più comprensibile a livello di scrittura del codice e non c'è stato bisogno di utilizzare un modulo separato per il salvataggio dell'array in file .png in quanto PyuEye è dotato di una funzione per il salvataggio dell'immagine. All'interno della directory di progetto troviamo comunque il modulo per thread_save per far notare le differenze rispetto alla versione finale.

Listing 5.1: Thread per il salvataggio dell'immagine

```
class Save_Thread(threading.Thread):
    def __init__(self, cam, list):
        super(Save_Thread, self).__init__()
        self.camera = cam
        self.queue = list
        self.killed = False
        self.isRunning = False
        self.isStarted = False
```

```
9
10      def run(self):
11          self.isStarted = True
12          self.isRunning = True
13
14          while not self.killed:
15              start = time.time()
16              tc.Lock.acquire()
17              self.camera.Save(self.queue)
18              tc.Lock.release()
19              end = time.time()
20              print("Time Save Thread: " + str(end - start))
21
22          if self.killed:
23              self.isRunning = False
24              raise SystemExit
25
26          self.isRunning = False
27
28      def kill(self):
29          self.killed = True
```

Listing 5.2: Funzione Save richiamata dal thread

```
1  def Save(self, list):
2      if(len(list)):
3          filename = "Camera" + str(self.camID) + "-" +
4                     str(time.time())
5          array = list.pop(0)
6          reshape = np.reshape(array, (self.height.value,
7                                       self.width.value,
8                                       self.bytes_per_pixel))
9          Image.fromarray(reshape).save("/home/pi/swim4all/Tirocinio/Photo/" +
       filename + ".png", "PNG")
10         print("Image saved \n")
11     else:
12         print("Image not saved \n")
13         pass
```

5.2 Possibili miglioramenti

Durante lo sviluppo, ci siamo accorti che se il codice veniva lanciato senza segnale di trigger, si doveva aspettare un certo tempo prima che l'esecuzione terminasse. Per risolvere il problema si potrebbe inserire un if che verifica se il segnale è presente o meno tramite una delle funzioni della libreria PyuEye.

Un'altra miglioria potrebbe consistere nell'inserire nel file di configurazione anche il tempo di attesa tra un'immagine e l'altra. Tale parametro, è rappresentato dalla variabile timeout interna al thread di cattura. Questo tempo è uno dei parametri della funzione is_WaitForNextImage, spiegata in dettaglio nella sezione 3.2.3 thread_capture.py e serve per comunicare alla funzione quanto tempo aspettare nel caso jn cui non ci sia nessuna immagine scattata.

5.3 Conclusioni

Lo sviluppo del software, secondo gli obiettivi fissati, è stato eseguito con successo. Per la realizzazione del progetto sono state usate le conoscenze della programmazione ad oggetti in Python, linguaggio adatto a questo scopo. Inoltre, sono state usate le conoscenze della programmazione procedurale in C per tutto ciò che riguarda la camera. In generale, si sarebbe potuto utilizzare il linguaggio C per permettere la realizzazione del progetto ma sarebbe risultato più complicato e inutile ai fini del progetto stesso.

Bibliografia

[1] https://code.visualstudio.com/docs/remote/ssh.

[2] https://en.ids-imaging.com/store/ui-3160cp-rev-2-1.html.

[3] https://it.wikipedia.org/wiki/architettura_arm.

[4] https://it.wikipedia.org/wiki/esposizione_(fotografia).

[5] https://it.wikipedia.org/wiki/secure_shell.

[6] https://it.wikipedia.org/wiki/thread_(informatica).

[7] https://it.wikipedia.org/wiki/velocità_della_pellicola.

[8] https://pypi.org/project/pyueye/.

[9] https://www.intel.it/content/www/it/it/products/boards-kits/nuc/kits/nuc8i7beh.html.

[10] https://www.json.org/.

[11] https://www.python.org/.

[12] https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.

Elenco delle figure