UNIVERSITÀ DEGLI STUDI DI BERGAMO

SCUOLA DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

OBJECT FOLLOWER CAMERA

Corso:
Laboratorio di Automatica

EDOARDO CRIPPA 1040587
SIMONE ZANNI 1041241

Anno Accademico 19/20

Indice

Capitolo

1 Introduzione

Lo scopo del progetto è quello di sviluppare un sistema con una telecamera in grado di mantenere al centro della propria inquadratura uno specifico oggetto.

A tale scopo è stato sfruttato un gimbal a 3 assi precedentemente costruito, al quale è stato smontato il motore per il controllo dell'asse X (rollio). Il resto del sistema è stato montato su una base fissa, ed è quindi in grado di controllare beccheggio e imbardata della telecamera.

Figura 1.1. *Gimbal a due assi montato sulla base fissa*

Tutta la logica di controllo di alto livello è stata sviluppata nel linguaggio Python ed è eseguita su un PC, mentre il codice sviluppato per Arduino ha il compito di ricevere le velocità da settare sui motori e di conseguenza applicare le tensioni adeguate sulle fasi degli stessi.

Gli script Python fanno largo utilizzo delle librerie OpenCV per l'acquisizione e l'elaborazione delle immagini dalla telecamera, mentre il codice Arduino non include nessuna libreria aggiuntiva.

Capitolo

2 Hardware

2.1 Componenti

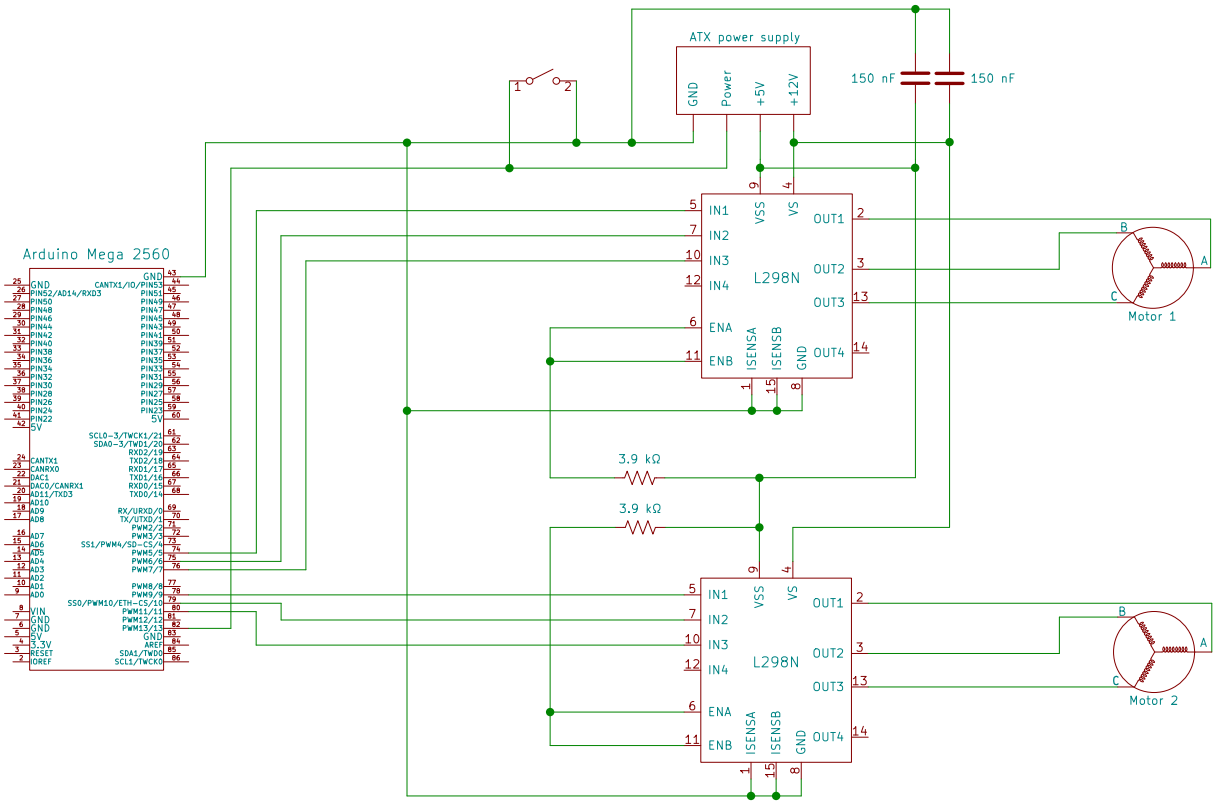


Figura 2.1. *Schema del circuito elettrico*

Per la realizzazione del progetto sono stati utilizzati i seguenti componenti hardware:

- 1x Arduino Mega 2560
- 2x Ponti H L298
- 2x Motori brushless BGM2606-90T
- 1x Camera USB 2.0
- 1x Alimentatore ATX 300 W
- 1x PC

Lo schema elettrico è riportato in figura 2.1. L'alimentatore ATX fornisce la tensione necessaria al funzionamento dei motori ($+12\,\mathrm{V}$) e della logica dei ponti H ($+5\,\mathrm{V}$), inoltre fornisce il riferimento di *ground* ed è accensibile cortocircuitando la linea *power* a massa attraverso un interruttore. Arduino e telecamera sono invece alimentati dal PC tramite cavo di tipo USB.

2.2 Motori

I motori utilizzati sono di tipo brushless DC, tipologia di dispositivi che non utilizzano il sistema collettore-spazzole (tipico dei motori in corrente continua) per la commutazione delle correnti negli avvolgimenti. Quest'ultimi sono alloggiati sullo statore e hanno il compito di generare il campo magnetico rotante che permette di orientare il rotore, che è costituito da magneti permanenti. La rotazione è resa possibile eccitando sequenzialmente gli avvolgimenti corretti in base alla posizione del rotore (ottenibile con dei sensori Hall), attraverso il controllo chiamato *six-step*.

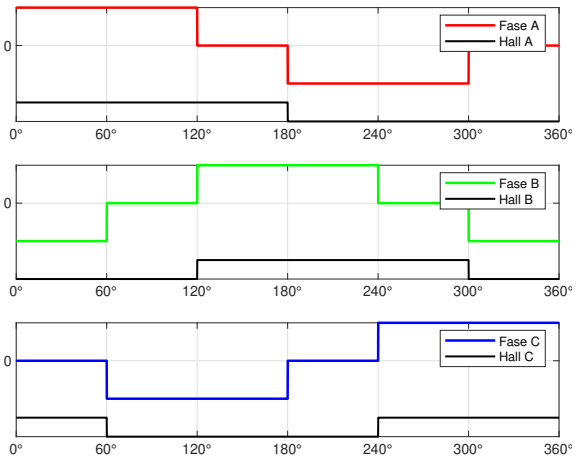


Figura 2.2. *Controllo six-step di un motore brushless con sensori Hall*

Il controllo *six-step* non è tuttavia adatto per far funzionare i motori a velocità molto basse (nella configurazione utilizzata essi non effettuano mai un giro completo): come suggerito dal nome questo tipo di controllo prevede la presenza di soli 6 punti di lavoro, che vengono ripetuti in sequenza. In un motore con una sola coppia polare di magneti rotorici, questa sequenza ne provoca un giro completo (con il rotore che quindi può assumere solo 6 posizioni nello spazio). Aumentando le coppie polari è possibile aumentare la risoluzione, i motori utilizzati sono di tipo 12N14P, ovvero hanno 12 avvolgimenti statorici e 14 magneti permanenti sul rotore (7 coppie polari), che permettono di ottenere una risoluzione $R = \frac{360°}{7 \cdot 6} \approx 8.571°$.

Per poter far girare i motori senza scatti a velocità basse, è necessario dunque modificare leggermente il controllo utilizzato: se i cambi di stato delle tensioni applicate sulle fasi non sono istantanei come in figura 2.2, ma avvengono seguendo delle rette inclinate, allora i segnali in figura assumono una forma trapezoidale. In questo modo in ogni istante vi è un punto di controllo diverso dal precedente e la risoluzione ottenibile è potenzialmente infinitesimale. Ovviamente in un'implementazione pratica questo è impossibile, la limitazione è la risoluzione PWM che il microcontrollore è in grado di offrire: nel caso di Arduino i possibili valori di PWM vanno da 0 a 255. In questo caso quindi, osservando ancora la 2.2 si nota che il segnale passa dal valore logico LOW a quello HIGH in uno spazio di $60°$, range in cui il controllo può assumere un massimo di 256 diversi valori. Nei $360°$ dunque i punti di controllo possibili sono $256 \cdot 6 = 1536$, prendendo in esame i motori utilizzati da 7 coppie polari la risoluzione è ora $R = \frac{360°}{7 \cdot 1536} \approx 0.034°$.

Attraverso prove pratiche si è tuttavia scelto di utilizzare delle onde triangolari per il controllo dei motori (vedasi figura 3.1), che provocano minore rumore udibile di quest'ultimi e soprattutto minor riscaldamento dei ponti H: queste forme d'onda sono ottenute eliminando le parti "piatte" da quelle trapezoidali, al costo di perdere due terzi dei possibili punti di controllo. Questo porta la risoluzione a circa $R \approx 0.100°$, valore comunque più che sufficiente per ottenere una rotazione regolare anche a basse velocità.

Capitolo

3 Arduino

3.1 Comunicazione con PC

La comunicazione con il PC sul quale è eseguito l'algoritmo di tracking avviene tramite il protocollo seriale. La comunicazione è di tipo *one-way*, ovvero il PC è sempre il trasmittente e Arduino funge solo da ricevitore. I dati da trasmettere riguardano solamente le due velocità da settare sui motori, ognuna delle quali codificata in un byte con segno, quindi in un range $[-128, 127]$.

3.2 Controllo dei motori

Il controllo dei motori, come accennato precedentemente, avviene tramite una terna di segnali triangolari in tensione sfasati tra loro di $120°$, in modo tale da ottenere una rotazione senza scatti.

Per poter generare i segnali analogici appena descritti è stata utilizzata la funzionalità PWM di Arduino, che è in grado di fornire in uscita ai suoi pin un'onda quadra che oscilla da 0 a $5\,\mathrm{V}$, con un duty-cycle $[0-1]$ direttamente proporzionale al valore di PWM $[0-255]$ specificato.

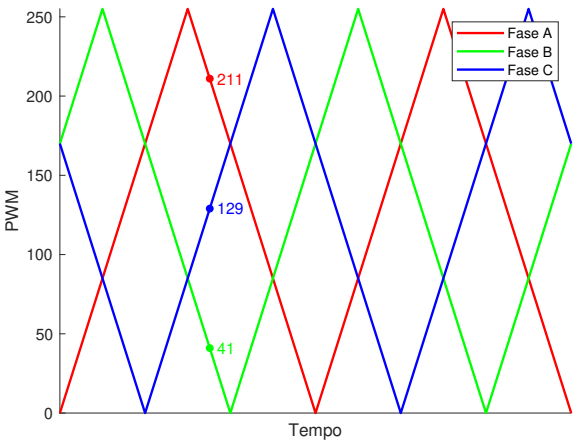


Figura 3.1. *Terna di segnali triangolari per il controllo di un motore*

La velocità di rotazione è controllabile variando il periodo dei segnali applicati. Il controllo non può essere continuo, quindi le forme d'onda in figura 3.1 sono in realtà discretizzate in diversi punti messi in sequenza: la velocità di rotazione è dunque controllabile variando il periodo con cui si passa da un punto di controllo a quello successivo. Per invertire la rotazione basta invertire a sua volta il controllo. Per esempio, considerando come punto di controllo quello evidenziato in figura 3.1, con valori di PWM sulle tre fasi di (211, 41, 129), il punto successivo di controllo è (210, 40, 130). Per far ruotare il motore nell'altro verso, il punto successivo è invece (212, 42, 128).

3.2.1 Conversione velocità-periodo di switching

I dati in ingresso allo script Arduino sono relativi alle velocità di rotazione calcolate dall'algoritmo di tracking, tuttavia il controllo dei motori necessita del periodo di *switching* tra i vari punti di lavoro. Per

quanto riguarda il verso di rotazione è sufficiente utilizzare il segno della velocità fornita, mentre per il valore assoluto del periodo è necessaria una conversione, in quanto la velocità dei motori è inversamente proporzionale a quest'ultimo. A tal proposito è stato progettato un algoritmo di conversione, stabilendo i valori minimo e massimo del periodo di *switching*, e assegnando rispettivamente i valori 127 e 1 della velocità in ingresso. I restanti valori sono convertiti attraverso interpolazione, come mostrato in figura 3.2, a eccezione del valore di velocità "0" che corrisponde a un valore infinito di periodo.

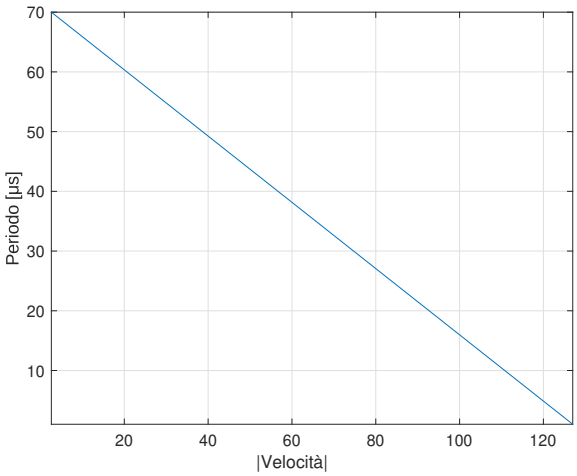


Figura 3.2. *Retta di conversione velocità-periodo di switching*

3.3 Avvio del sistema

Lo stato di alimentazione del sistema è controllato attraverso il cavo verde dell'alimentatore ATX, che si trova alla tensione di $+5\,\mathrm{V}$ e permette l'accensione del dispositivo quando è cortocircuitato a massa. La fase di set-up del sistema prevede il posizionamento dell'orizzonte della telecamera sopra la linea parallela al terreno, quindi è necessario azionare il motore che controlla l'angolo di tilt per un breve periodo di tempo. Questo è eseguito all'interno della funzione *initialPosition*.

Arduino fornisce la funzione *setup* per l'esecuzione di codice prima di iniziare il ciclo di controllo. Tuttavia, utilizzando questa funzione, sarebbe stato necessario resettare il microcontrollore a ogni avvio del sistema. Per questo è stato collegato il cavo verde di accensione dell'alimentatore al digital pin 13 dell'Arduino, in modo tale da rilevare i fronti negativi (ovvero quando la tensione passa a GND, l'alimentatore viene acceso e viene fornita corrente ai motori) ed eseguire di conseguenza la funzione *initialPosition* senza dover resettare il microcontrollore.

Capitolo

4 Programma PC

L'algoritmo eseguito sul PC consente di tracciare l'oggetto desiderato nell'immagine fornita dalla fotocamera e, utilizzando un controllo PID sulla posizione relativa di quest'ultimo, di comunicare all'Arduino le velocità da impostare per i vari motori.

L'intero programma è scritto in Python3, usando OpenCV come libreria grafica di visione artificiale in tempo reale. Tutti i metodi di riconoscimento utilizzati (per colore, usando Haar Cascades o DNN) sono implementati nativamente in questa libreria, e non è stato necessario aggiungere software esterni.

4.1 MultiThreading

Il programma si compone di 3 parti indipendenti tra loro, con variabili condivise gestite dalla classe Manager del pacchetto *multiprocessing* nativo di Python.

4.1.1 Tracking

Questo processo riceve lo stream video dalla fotocamera e calcola per ogni frame il valore delle coordinate (xy) del centro dell'oggetto, se viene rilevato, altrimenti restituisce come valore il centro dello schermo.
Il componente che ha il compito di rilevare l'oggetto cercato nell'inquadratura è deciso a runtime dalle impostazioni passate come argomento al programma, in modo da risultare modulare e permettere, con lo stesso codice, di riconoscere oggetti diversi e adattarsi a più situazioni (questo funzionamento è descritto nella sezione 4.2).

Il risultato di questi calcoli è salvato in due variabili condivise, che vengono usate dal controllore PID per ricavare le velocità da assegnare ai motori.

4.1.2 Controllori PID

Con lo stesso codice vengono inizializzate due istanze della classe PID, in questo modo i calcoli dei movimenti di imbardata e beccheggio avvengono indipendentemente tra loro.
In un loop infinito questo processo calcola la distanza euclidea dal centro dell'inquadratura al centro dell'oggetto tracciato precedentemente, e il risultato è usato come errore per aggiornare il controllore, il quale a ogni passo restituisce il valore di velocità che deve avere il motore corrispondente.

Si può notare che, siccome se nessun oggetto viene trovato il processo di tracking restituisce come output il centro dello schermo, l'errore calcolato da questo processo è esattamente "0", il che tenderà a fermare i movimenti dei motori, a meno di un'azione del controllo integrale ancora in corso.

In più, se un oggetto si trova a un errore dal centro minore di un valore arbitrariamente piccolo (è stato scelto $\frac{1}{20}$ della larghezza del frame) viene bypassato il calcolo del controllore e viene restituito il valore di velocità "0", per evitare oscillazioni di assestamento quando l'oggetto designato è già al centro dell'inquadratura. Fatto ciò viene anche reinizializzata la classe, per perdere lo storico dei valori che, se integrati, porterebbero a una deriva non voluta in caso di piccole oscillazioni dell'oggetto.

Ogni iterazione del ciclo ha un tempo di pausa di 0.01 secondi, per permettere al processo di tracking di calcolare le distanze e quindi avere dati eterogenei ogni volta.

4.1.3 Comunicazione con Arduino

Questo processo prende i valori generati dai controllori PID e li invia ad Arduino grazie al comando *write()* della libreria *serial*.

I valori di velocità calcolati vengono quindi limitati e codificati in interi a 8 bit, la grandezza dell'unità di trasmissione più piccola ammissibile dal canale seriale di Arduino, ma che permette una risoluzione più che sufficiente per un buon controllo dei motori ($2^8 = 256$ valori ammissibili di velocità, compresi tra -128 e 127).

A ogni ciclo di controllo sono dunque trasmessi ad Arduino due byte di informazioni, uno per ogni velocità degli assi da controllare, in questo modo la comunicazione è abbastanza veloce e non introduce colli di bottiglia nel sistema. La lettura su Arduino avviene tramite un array di due byte, i valori letti vengono poi convertiti in *signed char* per poterli interpretare con la stessa codifica utilizzata dallo script Python.

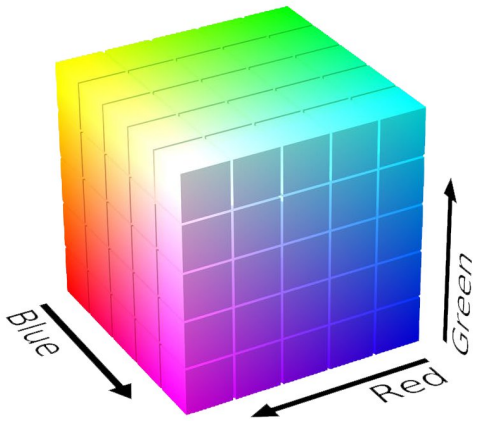
4.2 Come avviene il riconoscimento

Esiste la possibilità di usare diversi metodi di riconoscimento di oggetti nell'ambito della computer vision. In questo caso per ogni tipologia di oggetto da tracciare è stata scelta la modalità da si possono trarre più benefici.
In particolare, il riconoscimento per colore è affidato a una maschera che seleziona i valori desiderati e considera l'oggetto con il contorno più grande, mentre il riconoscimento di volti si può effettuare tramite Haar Cascades o Deep Neural Networks, e il sistema tiene traccia dell'oggetto più vecchio nell'inquadratura, in quanto la camera ne può seguire solo uno per volta.

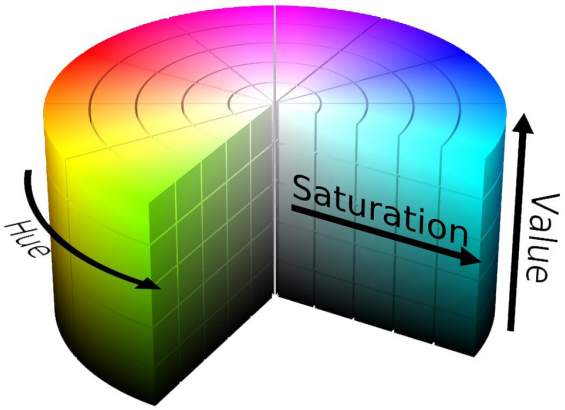
Tutte le varie classi hanno la stessa interfaccia in output e ritornano 3 parametri: cordinate x e y del centro e raggio del cerchio che racchiude l'oggetto riconosciuto.

4.2.1 Per colore

OpenCV lavora con colori descritti dalla convenzione HSV (Hue Saturation Value), un modello alternativo all'RGB sviluppato negli anni '70 da ricercatori di computer grafica per essere particolarmente orientato alla prospettiva umana, essendo basato sulla percezione che si ha di un colore in termini di tinta, sfumatura e luminosità.



Modello RGB



Modello HSV

Da un frame quindi, convertito in HSV, viene estratta una matrice con valori booleani le cui celle rappresentano se quel determinato pixel è nel range di colore cercato oppure no.

A questo punto si usa il metodo *cv2.findContours* per trovare i contorni degli oggetti rilevati dalla maschera. Un contorno è una semplice curva che collega tutti i punti di frontiera dello stesso colore e intensità. Applicando questo metodo sulla maschera si otterrà il contorno degli oggetti riconosciuti.

Di tutte queste curve si sceglie quella con il perimetro maggiore, e si trova il centro dell'oggetto con il metodo *cv2.minEnclosingCircle*, che restituisce centro e raggio del cerchio di area minore che racchiude interamente il set di punti passati come parametro.

Si può notare dunque che il metodo di scelta tra più oggetti rilevati per decidere quale tracciare si basa unicamente sull'area occupata nell'inquadratura. Questo permette anche di ignorare il rumore di fondo nell'immagine, in quanto dei pixel erroneamente riconosciuti come dello stesso colore da cercare non avranno mai un'area maggiore di un oggetto fisico presente nel frame, e in più si è scelto di non considerare oggetti il cui *minEnclosingCircle* ha un raggio minore di 10 pixel.

4.2.2 Volti con Haar Cascade

Il riconoscimento di volti tramite Haar Feature-based Cascade è un metodo computazionalmente leggero ma necessita di volti quasi perfettamente allineati con la camera per poterli riconoscere.

Questo modello di machine learning utilizza delle feature basate sulla somma delle differenze tra pixel in regioni diverse del frame (si veda figura 4.1). Il calcolo di ogni possibile differenza sulle diverse combinazioni di pixel porterebbe a un numero enorme di variabili (più di 160 mila), e dunque durante la fase di traninig è utilizzato AdaBoost per selezionare le migliori features e costruire un classificatore accurato che ne utilizza solo circa 6000.

Il processo di training dell'algoritmo restituisce un file in formato XML, utilizzabile per classificare in modo efficiente oggetti in un immagine: a ogni quadrato di 24x24 pixel non vengono applicate le 6000 features, ma venogno raggruppate su più livelli e controllate una alla volta "a cascata". Se una regione dell'immagine non passa il riconoscimento di una feature, tutto il gruppo viene scartato e i controlli seguenti non vengono effettuati, rendendo l'algoritmo molto veloce. Se invece la regione dell'immagine

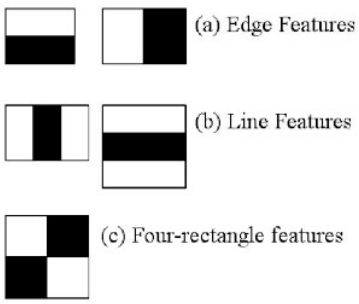


Figura 4.1. *Tipi di feature*

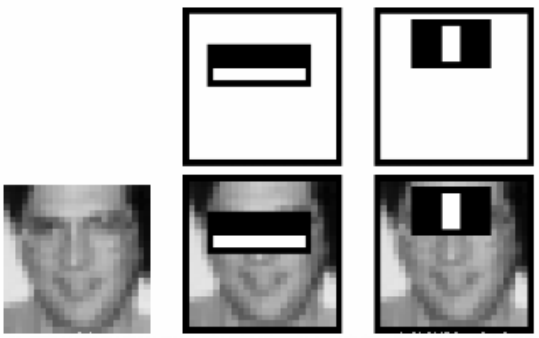


Figura 4.2. *Esempio di applicazione*

passa la verifica di tutto il gruppo di features significa che c'è corrispondenza e viene riconosciuto un volto.

Il metodo utilizzato è implementato in OpenCV e si può richiamare con il seguente comando: *cv2.CascadeClassifier.detectMultiScale*. Questo restituisce una lista di rettangoli, cioè coordinate (x, y) di due vertici opposti, per gli oggetti riconosciuti.

Se più di un volto viene trovato nell'inquadratura si è scelto di tenere traccia dell'oggetto più vecchio, nel senso di tempo passato nell'inquadratura. A ogni volto viene associato un ID incrementale, e i motori si muovono per centrare quello con l'ID minore. L'algoritmo di assegnazione e update degli ID è descritto nella sezione 4.2.4.

4.2.3 Volti con Deep Neural Networks

Dalla versione 3.3 di OpenCV, rilasciata in agosto 2017, è disponibile il modulo *dnn*, che supporta molteplici framework di deep learning, come Caffe, TensorFlow e Torch.
Nonostante sia computazionalmente più oneroso dell'algoritmo basato su Haar features riesce a individuare gli oggetti desiderati anche se non perfettamente allineati con la camera o parzialmente coperti. Su un PC con basse prestazioni o addirittura un Raspberry non è fattibile eseguire questo metodo in quanto il calo di FPS porta a uno squilibrio nel sistema degli ID (descritto nella sezione 4.2.4), nel controllo PID e una errata comunicazione di velocità ai motori.

Sulla pagina GitHub di OpenCV sono disponibili dei modelli pre-allenati, per cui è stato usato uno di questi basato su Caffe. I file necessari sono:

- il file **.prototxt**, che definisce il modello (la descrizione dei vari layer).
- il file **.caffemodel**, che contiene i valori dei pesi dei vari layer ottenuti alla fine della fase di training.

Il programma prende il frame e ne crea un *blob*, una fase di preprocessing necessaria per ottenere le corrette predizioni dalla rete neurale ridimensionando e normalizzando l'immagine (riducendo errori dovuti a differenti dimensioni o illuminazione).

Il blob viene poi dato in input alla *dnn*, la quale restituisce una lista di rettangoli con abbinata la confidenza (cioè la probabilità) associata con la predizione. Vengono filtrati i risultati con una probabilità maggiore di un valore arbitrario (in questo caso 80%) e i rimanenti vengono usati per aggiornare l'algoritmo dei centroidi (descritto nella sezione 4.2.4) che restituisce gli ID dei volti rilevati, potendo scegliere quello con il valore minore, e dunque il più vecchio nell'inquadratura.

4.2.4 Algoritmo dei Centroidi

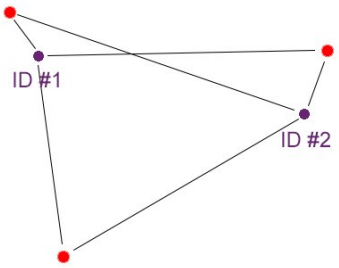
Algoritmo che permette di prendere un set iniziale di oggetti, assegnare loro un ID univoco e tenere traccia dei loro movimenti in frame successivi in un video, mantenendo l'assegnamento dell'ID.

Un buon algoritmo di tracking deve poi garantire una certa robustezza alle occlusioni e agli oggetti "persi" tra un frame e l'altro.

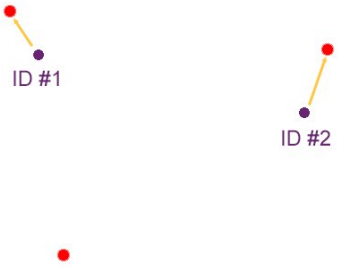
L'algoritmo chiamato *Centroid Tracking* si basa sulla distanza euclidea tra i centroidi di un frame e quelli del frame successivo e si basa su 4 step.

Nel seguente esempio nel primo frame sono riconosciuti due oggetti (in viola) e nel successivo tre oggetti (in rosso). Si nota come alla fine dell'ultimo step gli oggetti con ID #1 e #2 hanno cambiato posizione, e un nuovo oggetto viene aggiunto al set con un ID univoco e maggiore degli altri.

Step n.1: assegnamento ID



Step n.2: calcolo distanze



Step n.3: aggiornamento oggetti esistenti

Step n.4: registrazione nuovi ID

- **Step n.1**: Si ricevono dei *bounding box* e se ne calcolano le coordinate dei centri. Si assegna a ogni oggetto un ID univoco incrementale.
- **Step n.2**: Ricevuto il frame successivo, si calcola la distanza Euclidea tra i vecchi centri e quelli del frame corrente, senza assegnare nuovi ID. Prima bisogna verificare se i nuovi centri (in rosso) possono essere asseganti a qualcuno dei vecchi centri (in viola).

- **Step n.3**: Vengono aggiornate le coordinate degli oggetti esistenti. Questo processo si basa sulla seguente assunzione: dato un oggetto che può muoversi in frame seguenti la distanza tra i centrodi dello stesso corpo per i frame F_t e F_{t+1} sarà la minore di tutte le altre distanze tra oggetti. L'aggiornamento delle posizioni degli oggetti quindi tende a minimizzare la distanza Euclidea tra i centri degli stessi in frame successivi.
- **Step n.4**: Se ci sono più oggetti riconosciuti rispetto al frame precedente, a quelli rimasti orfani (cioè senza un ID assegnato durante lo step n.3) viene assegnato un nuovo ID univoco e maggiore degli altri. Nel caso invece ce ne siano di meno, gli oggetti scomparsi saranno cancellati dalla lista. Un oggetto viene ritenuto *scomparso* quando non si riesce a trovare un match per N frame consecutivi, con N a piacere.

L'assunzione nello step n.3 dimostra l'infattibilità dell'utilizzo del metodo basato su *dnn* in PC con bassa potenza di calcolo: un drastico calo dei FPS nel video porta ad avere frame successivi che sono temporalmente distanti anche 0.5 secondi, e in questo caso l'ipotesi di minore distanza tra i centri dello stesso oggetto rispetto ad altri non è più valida, in quanto è un lasso di tempo sufficiente per muovere due oggetti abbastanza velocemente da far perdere la traccia al sistema degli ID.

Un altro problema riguarda la **sovrapposizione**: non potendo supportare una vista prospettica, e quindi identificare anche la profondità, due oggetti che hanno il centro nelle stesse coordinate porteranno il sistema basato su distanze Euclidee a fare errori di tracciamento nel 50% dei casi. Nella pratica questo significa che se nell'inquadratura due oggetti si sovrappongono potrebbero scambiarsi gli ID.

4.3 Valori del controllore PID

I valori di kP, kI e kD (rispettivamente i pesi per le componenti proporzionale, integrativa e derivativa del controllore) devono essere regolati per permettere le migliori prestazioni possibili nell'applicazione pratica, in questo caso per fare in modo che la telecamera segua abbastanza velocemente l'oggetto in questione senza compiere movimenti troppo bruschi.

I controllori dei due motori saranno settati in modo indipendente tra loro, iniziando dal motore di beccheggio, utilizzando il metodo di taratura manuale:

- Impostare kI e kD a 0, lasciando un controllo unicamente proporzionale.
- Incrementare kP da 0 fino al primo valore per cui la camera oscilla con l'oggetto in centro, e scegliere come valore di kP la metà di quello appena trovato.
- Incrementare kI in modo da ridurre in fretta qualsiasi offset ci sia. Un valore troppo alto causa instabilità. In questa configurazione non esistono offset, e il controllo integrale serve a inseguire oggetti che si muovono insieme alla camera, simulando un offset "virtuale".
- Incrementare kD fino a che il sistema è accettabilmente veloce a raggiungere l'oggetto dopo un disturbo improvviso (come muovere l'oggetto molto velocemente). Un valore troppo alto di kD causerà una risposta eccessiva e farà in modo che l'oggetto venga superato dalla camera.

Capitolo

5 Conclusioni e Sviluppi futuri

Di seguito sono elencate le principali problematiche riscontrate durante lo svolgimento del progetto e le migliorie possibili:

- I motori utilizzati non sono ottimizzati per questa applicazione, in quanto non sono in grado di fornire informazioni sulla posizione del rotore (ad esempio con sensori Hall). Il mancato utilizzo di dispositivi esterni come accelerometri o giroscopi, obbliga a effettuare un controllo in anello aperto, il risultato è che il sistema non è robusto a perturbazioni esterne. Ad esempio se i cavi di alimentazione di un motore dovessero opporre resistenza meccanica, il rotore potrebbe trovarsi in una posizione diversa da quella stimata dal controllo e ciò risulterebbe in movimenti inaspettati del motore.
- I ponti ad H utilizzati per controllare la corrente sulle fasi del motore sono ottimizzati per il classico controllo *six-step*, ovvero sono in grado di aprire e chiudere degli interruttori (transistor BJT) per decidere su quali fasi fluisce la corrente e il verso della stessa. Con il controllo utilizzato i transistor sono sempre in conduzione, e la corrente viene controllata attraverso la tensione applicata sulla base degli stessi, con relativo riscaldamento eccessivo dei dispositivi.
- Le prestazioni sono influenzate da diversi fattori, come la potenza del PC, la qualità della camera, la qualità dell'illuminazione ambientale, ecc... e dunque non è possibile dare giudizi oggettivi sul corretto funzionamento ma solo probabilistici.
- L'algoritmo dei Centroidi per l'assegnamento degli ID agli oggetti non è robusto, come si è visto, alla sovrapposizione di due volti nell'inquadratura. Si possono implementare metodi più avanzati di tracking, come ad esempio quello basato su kernel (link) o sul correlation-tracking (link)
- Utilizzando altri file XML per le Haar-features o modelli di deep learning si può estendere il tracciamento di oggetti oltre a quello dei volti.