UNIVERSITÀ DI PISA

Goalkeeper Hardware

Informatica e Sistemi in Tempo Reale

Corso di Laurea Magistrale in Ingegneria Robotica e dell'Automazione

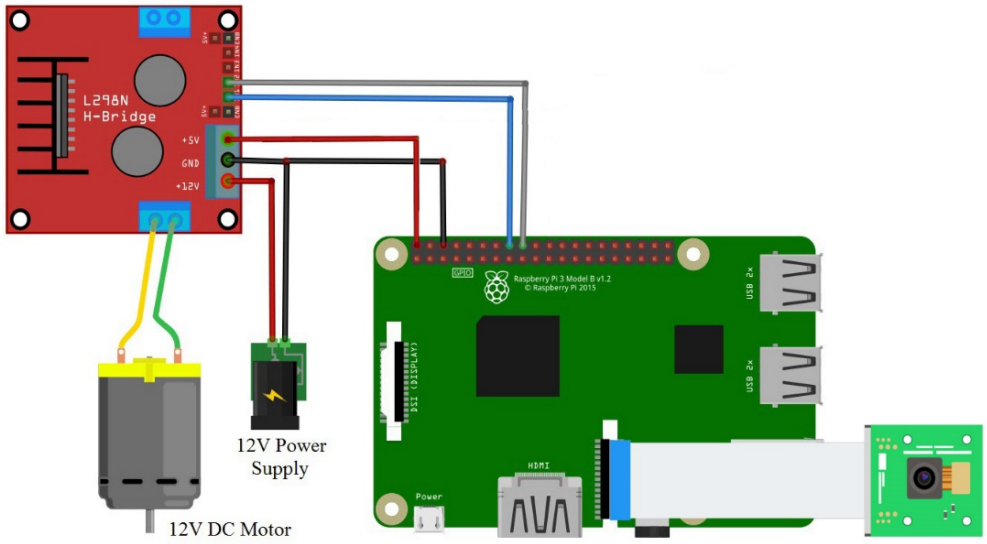
STUDENTI	MATRICOLA	EMAIL	DATA
ALESSIO TUMMINELLO	605871	a.tumminello1@studenti.unipi.it	20/05/2020
FEDERICO VITABILE	605862	f.vitabile@studenti.unipi.it	

Introduzione

L'obiettivo del progetto Goalkeeper Hardware è la realizzazione di un sistema fisico in grado di parare una palla lanciata su un piano di gioco, attraverso l'uso di tecniche di programmazione Real-Time.
Il software da realizzare deve essere in grado di catturare la posizione della palla, prevederne la posizione finale e infine spostare il portiere nella posizione opportuna per pararla.

Componenti utilizzati

Per la realizzazione del Goalkeeper si è partiti dalla scelta dell'hardware su cui implementare il codice. La scelta è ricaduta su un Raspberry Pi 3B+, in cui è possibile installare un sistema operativo; nel nostro caso Raspbian, una distro di Debian basata su Linux.
Per l'acquisizione delle immagini si è scelta una Raspicam da 5 megapixel per via del suo supporto nativo col Raspberry, mentre per la parte di attuazione un motore dc da 12V pilotato attraverso un ponte ad H, nella fattispecie un modulo L298N.
I collegamenti sono mostrati nella seguente figura.



Il campo da gioco è costituito da una base di legno, di dimensioni 60x49 cm, al cui lato, in una tavola di legno, sono posizionati il Raspberry, il modulo L298N e un supporto per la camera stampato in 3D.
Il sistema di attuazione del portiere si compone di una sagoma sostenuta da una cinghia GT2, posta fra due pulegge di diametro 8 cm; la prima attuata dal motore dc, mentre la seconda è folle.

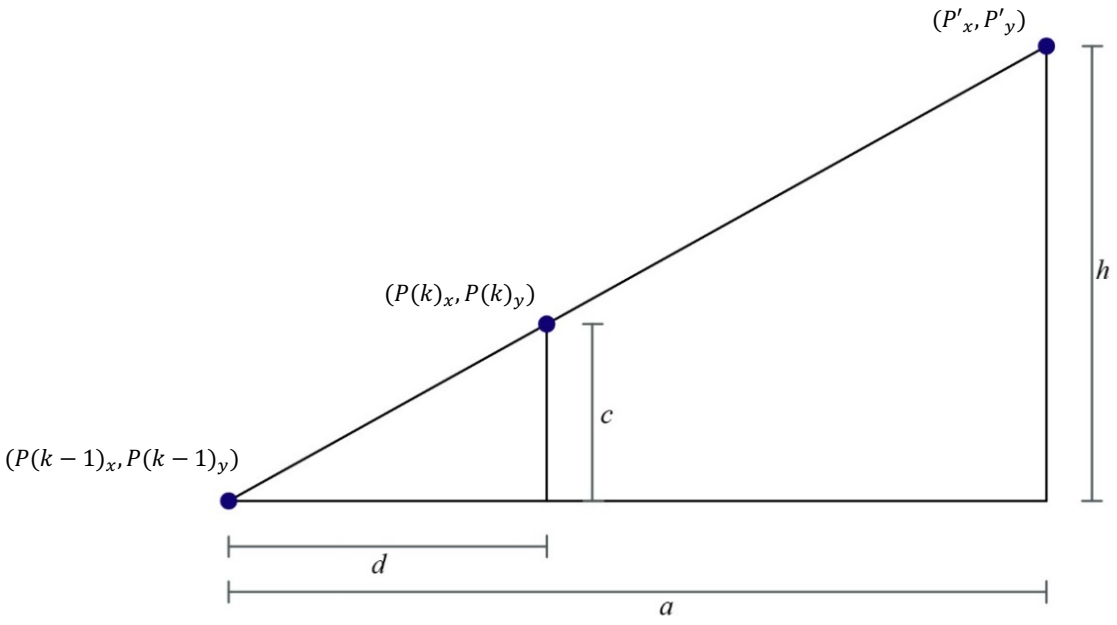
Tracking palla e portiere

Le coordinate della palla vengono determinate attraverso il calcolo del centroide; in particolare, una volta acquisito un frame, viene analizzato il colore dei singoli pixel e confrontato con quello della palla. Alla fine del processo si calcola la media delle coordinate dei pixel con lo stesso colore del target, ottenendo così le coordinate della palla all'istante di campionamento k.
In maniere analoga viene calcolata la posizione del portiere.

Modello di predizione

Per via dell'inevitabile presenza di jitter nel task che calcola la posizione della palla all'istante corrente, si è preferito un metodo di predizione che non utilizzasse il periodo del task della camera.
Approssimando la traiettoria della palla come lineare si è quindi scelto di sfruttare una proprietà dei triangoli simili, che si vengono a formare se si considerano:

- $\boldsymbol{P(k)}$ la posizione della palla al tempo *k* (nota)
- $\boldsymbol{P(k-1)}$ la posizione della palla al tempo *k-1* (nota)
- $\boldsymbol{P'_y}$ la coordinata y del portiere (nota)
- $\boldsymbol{P_f}$ la posizione finale della palla (incognita)



Indicando con:

- ***a*** la distanza lungo l'asse y fra la posizione della palla al tempo *k-1* e la posizione del portiere
- ***d*** la distanza lungo l'asse y fra la posizione della palla al tempo *k-1* e *k*
- ***c*** la distanza lungo l'asse x fra la posizione della palla al tempo *k* e *k-1*
- ***h*** la distanza lungo l'asse x fra la posizione della palla al tempo *k-1* e la posizione finale della palla

per la proprietà dei triangoli simili si ha che $a : h = d : c$ e quindi $h = \frac{a*c}{d}$.
Infine la posizione finale della palla sarà pari a $P_f = P_x(k-1) + h$.

Controllo del motore

Come già anticipato, per via della sua velocità e della sua facilità di controllo, l'attuazione è stata affidata ad un motore dc da 12V. Il pilotaggio di quest'ultimo, grazie all'uso del modulo L298N, si riduce alla scrittura di due pin fisici (16 e 18) del Raspberry come mostrato nella seguente tabella:

Pin 16	Pin 18	Stato motore
LOW	*LOW*	fermo
HIGH	*LOW*	direzione puleggia attuata
LOW	*HIGH*	direzione puleggia folle
HIGH	*HIGH*	fermo

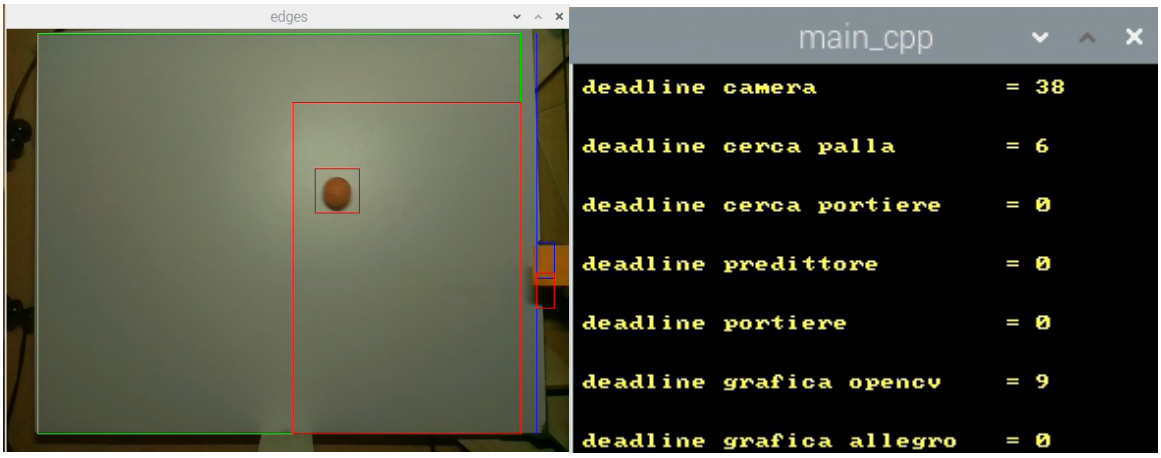
Per il controllo in posizione si è optato per un controllo in feedback, l'idea di base è quella di controllare ad ogni istante di campionamento k la posizione del motore e, se la distanza dalla posizione desiderata risultasse essere maggiore/minore di un certo margine di tolleranza, far ruotare il motore nella direzione opportuna o mantenerlo fermo, nel caso contrario.
Per via dei limiti sulla frequenza di campionamento, imposti dalla latenza della camera, non è stato possibile implementare il controllo descritto in precedenza. In particolare avendo a disposizione un nuovo frame, su cui rilevare la posizione del portiere, ogni 45 ms e ipotizzando che la posizione venga letta un istante prima che essa venga aggiornata, si

arriverebbe ad un worst-case di 90 ms in cui il motore permarrebbe in uno dei tre stati. Ciò, unito all'alta velocità del motore, comporterebbe un'inevitabile instabilità del sistema.
Di conseguenza si è modificata l'idea di partenza, permettendo al motore di stare attivo solo per un intervallo di tempo t_{on} e non per tutto il periodo. Inoltre, per cercare di limitare i casi di letture non aggiornate, si è deciso di assegnare al task relativo al controllo priorità minore rispetto al task relativo alla ricerca della posizione del portiere.

Interfaccia grafica

L'interfaccia grafica si compone di due finestre, la prima contiene ciò che viene acquisito dalla camera, mentre nella seconda vengono visualizzate le deadline miss dei vari task.
La prima finestra è stata realizzata con l'ausilio della libreria grafica OpenCv in C++ e mostra in tempo reale ciò che viene acquisito dalla camera con l'aggiunta di elementi grafici che evidenziano:

- il campo di gioco (indicato da un rettangolo verde)
- l'area di ricerca della palla (indicata da un rettangolo rosso)
- la linea di ricerca del portiere (indicata da una linea blu)
- la posizione della palla (indicata da un quadrato rosso)
- la posizione del portiere (indicata da un rettangolo blu)
- la posizione finale predetta della palla (indicata da un rettangolo rosso)



La seconda finestra è stata invece realizzata con la libreria grafica Allegro 4.2 in C ed è composta da uno sfondo nero dove sono mostrate in giallo le deadline miss dei task.

Implementazione

Il sistema così descritto è stato implementato attraverso l'uso di sette task periodici concorrenti, comunicanti fra loro attraverso l'uso di variabili globali.
Il codice così scritto è stato suddiviso in quattro librerie (*camera_plus.h, predittore.h, dc_motor.h, task_allegro.h)* compilate insieme attraverso l'uso di un Makefile.

Il *task camera* ad ogni attivazione acquisisce un nuovo frame dalla camera e lo salva in una struttura Mat di nome *frame*, globale per la libreria *camera_plus.h.*

Il task *cerca_palla* prende in ingresso la struttura Mat *frame,* la analizza per trovare le coordinate della palla e infine salva quest'ultime in una struttura globale *struct coord coord_palla*. La struttura *coord* si compone di quattro elementi rappresentanti le componenti lungo gli assi x e y della coordinata della palla all'istante di campionamento corrente *k* e quelle dell'istante precedente *k-1,*
Per ridurre il tempo di calcolo del task la ricerca della palla non viene effettuata su tutta l'immagine, ma solo su una sezione in base alla direzione in cui si sta muovendo la palla. L'aggiornamento della finestra di ricerca viene affidato sempre al task *cerca_palla,* attraverso la chiamata di una funzione di nome *aggiorna_finestra,* poco prima che esso si sospenda. La finestra di ricerca è realizzata da una struttura globale, di nome *area_ricerca,* di quattro elementi rappresentanti le coordinate x e y di due vertici opposti di un rettangolo.

Il task *cerca_ portiere* così come il precedente prende in ingresso la struttura Mat *frame,* e una volta individuate, salva le coordinate del portiere in una variabile globale di tipo *int* di nome *pos_goalk*.

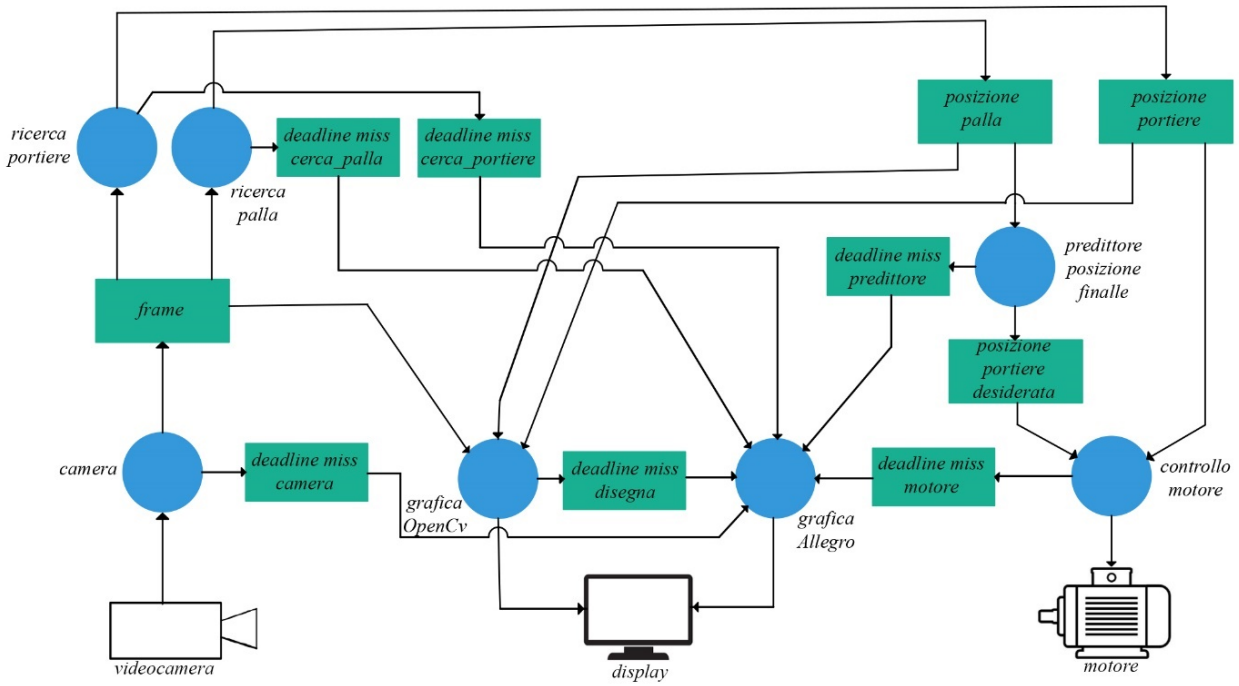
Il task *predittore* prende in ingresso la struttura globale *coord_palla* e, come spiegato nel capitolo sul modello di predizione, calcola la posizione finale della palla. Questa viene successivamente salvata su una variabile globale di tipo *int* di nome *pos_des.*

Il task *portiere* prende le variabili globali *pos_goalk* e *pos_des* e, come spiegato nel capitolo sul controllo del motore, si occupa dell'attivazione del motore per un certo t_{on}. Questo procedimento viene implementato attraverso l'uso di un task aperiodico di nome a*spetta* e di un semaforo di sincronizzazione di nome *sem_aspetta* che blocca il task *portiere* fino a che non sia terminato *aspetta.*

Il task *disegna* si occupa di mostrare sullo schermo tutti gli elementi grafici corrispondenti alla finestra di OpenCv prendendo i dati in ingresso dalle corrispondenti variabili globali.

Infine il task *stampa_deadline* stampa sulla finestra di allegro le deadline di tutti i task periodici. I valori del numero di deadline miss dei vari task vengono acquisiti dalle relative variabili globali, una per ogni task ad eccezione di *stampa_deadline* che è locale al task stesso; queste in caso di sforamento della propria deadline, vengono incrementate da ogni task.

Tutte le variabili globali condivise dai vari task sono state protette da dei semafori, in particolare le variabili relative alle deadline miss e quella della finestra di ricerca della palla sono state protette attraverso dei semafori classici sem_t. Mentre per il frame, le coordinate della palla, pos_des e pos_goalk, essendo condivisi fra più di due task, si sono usati i Mutex. Il sistema così realizzato viene mostrato nel seguente diagramma.



Come si è visto, per il discorso di stabilità già affrontato, si è scelto di suddividere in tre task separati l'acquisizione dell'immagine, la ricerca del portiere e la ricerca della palla in maniera tale da minimizzare la frequenza di aggiornamento della posizione del portiere.
Si è quindi deciso di assegnare la priorità maggiore (50) al task *camera,* intermedia (40) al task *cerca_portiere* e infine la più bassa fra le tre (30) al task *cerca_palla*.
Successivamente, sempre per il discorso della stabilità, fra la predizione e il controllo del motore si è assegnata priorità maggiore al secondo. In particolare al task *motore* è stata

assegnata la priorità 35, poco inferiore a quella di *cerca_palla,* per ridurre le letture di posizioni non aggiornate. Mentre al task *predittore* è stata assegnata una priorità poco inferiore (25) rispetto al task *cerca_palla,* sempre per il discorso di letture non aggiornate. Infine sono state assegnate le priorità minori ai due task grafici, nello specifico 20 a *disegna* e 10 a *stampa_deadline*.

Una volta fissate le priorità si è trovato in maniera empirica il periodo del task *camera*, pari a 45 ms. Lo stesso periodo è stato assegnato anche ai task *cerca_portiere* e *motore*.
Per non sovraccaricare troppo il processore, senza però perdere di efficienza si è assegnato ai task *cerca_palla* e *predittore* un periodo pari a 55 ms. Infine ai task grafici è stata assegnato un periodo di 120 ms.
Le deadline sono state assegnate a tutti i task pari ai rispettivi periodi.
Come scheduler si è usato OTHER a causa dell'incompatibilità con la libreria OpenCv dello scheduler FIFO.

Considerazioni finali

Il sistema complessivo così ottenuto riesce a parare le palle lanciate con delle velocità moderate, questo limite di velocità è dettato dai problemi di stabilità precedentemente discussi.
I task riescono a rispettare le proprie deadline, ad eccezione di qualche caso isolato. Mentre si hanno alcune deadline miss nella fase di avviamento del software, in particolare per il task *camera,* dovute al tempo necessario per avviare la camera.