Visual Tracking (Real-Time)

INDICE

1. Introduzione

Lo scopo di questo progetto è l'implementazione di un programma (scritto in linguaggio C) che permetta il tracciamento di un oggetto di forma rotonda all' interno di uno spazio di dimensioni finite e in cui l'oggetto si muove. L' oggetto che deve occuparsi del compito è visto come un quadrato (obiettivo della camera) che dovrà sempre muoversi cercando di mantenere il bersaglio rotondo all' interno dei margini dell'obiettivo.
Per semplicità (senza comunque togliere generalità al progetto) consideriamo il bersaglio come una palla di colore diverso dal nostro colore di background.

L' applicazione è composta da tre diversi task, ognuno dei quali si occupare di gestire una parte dell'applicazione.
Un task è dedicato alla gestione della palla, un altro alla gestione della camera e l'ultimo alla gestione dell'errore (compito di quest'ultimo è calcolare la distanza fra il centro della camera e la palla e disegnare un grafico in scala 1:8).

Il task dedicato alla gestione della palla si occupa di muovere quest'ultima all'interno dell'area rettangolare delimitata dai bordi. Deve essere anche gestito il contatto coi bordi in modo tale che questi non vengano oltrepassati.

Il task camera serve per muovere la camera e far sì che questa insegua la palla cercando di mantenerla dentro l'obiettivo.
Per fare ciò si effettua una previsione sulla possibile futura posizione della palla in modo tale da poter sempre seguire quest'ultima con precisione.

Utilizzando come algoritmo di scheduling FIFO il task a priorità maggiore sarà quello relativo alla camera, media priorità sarà assegnata al task relativo alla palla e priorità minore al task che gestisce l'errore.

Nel codice del progetto ci sono inoltre dei frammenti commentati per poter utilizzare la politica SCHED_DEADLINE. Non essendo questa standard e non trovandosi negli header file le strutture richieste per il suo utilizzo è stato necessario includere nel progetto un ulteriore file (sched_deadline.h) dove sono dichiarate le strutture e le chiamate di sistema necessarie per settare tale politica di scheduling ai task.

Tutta l'applicazione si basa su due file di libreria che consentono la creazione e la gestione di thread periodici.

Per eseguire il progetto è necessario avere:
- Compilatore gcc
- Libreria allegro

Per compilare:
- Spostarsi nella directory track.
- Eseguire il comando make.

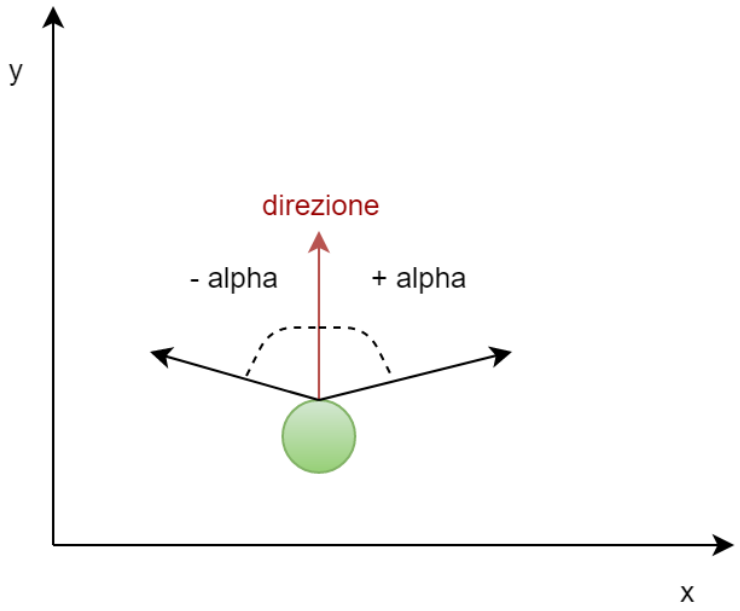
2. Palla

La palla possiede inoltre diverse modalità di movimento che possono essere selezionate dall' utente tramite la tastiera. I movimenti previsti dall' applicazione sono:

- Casuale
- Sinusoidale
- Mouse

Nella modalità casuale la palla è libera di muoversi in maniera randomica. Ogni movimento tiene però in considerazione anche la direzione precedente, andando a generare una direzione casuale entro un certo range rispetto a quella precedente.
Nella modalità sinusoidale la palla segue la classica traiettoria descritta dal seno.
Nella modalità mouse viene utilizzato il mouse all' interno della nostra interfaccia grafica per muovere la palla.
In ogni modalità viene assunta costante la velocità della palla.

a. Modalità Random



alpha = PI / 6

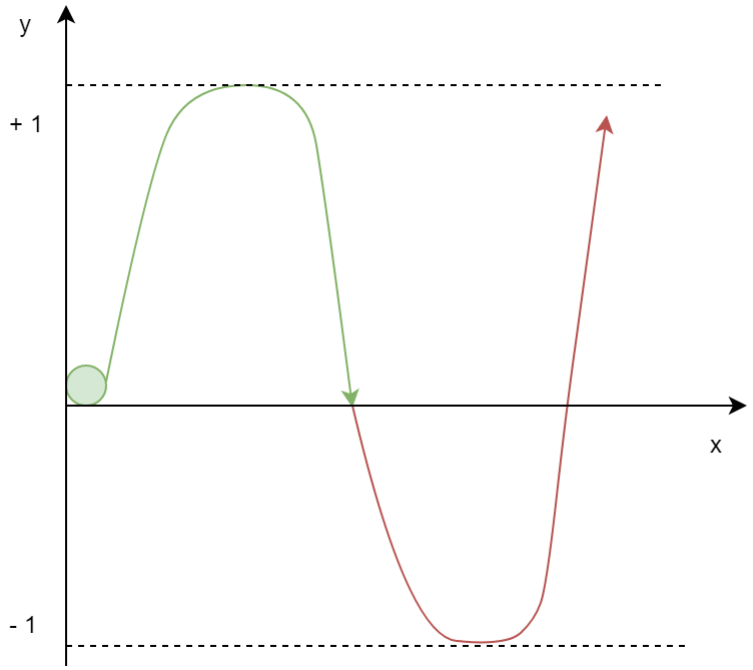
In modalità random il range entro cui viene scelta la direzione successiva è:

$$[direzione\ attuale - alpha, direzione\ attuale + alpha]$$

L'aggiornamento della velocità e quindi della posizione è descritto dalle due equazioni:

$$vx = ball.v * cos(ball.a)$$
$$vy = ball.v * sin(ball.a)$$

b. Modalità Seno



Nella modalità sinusoidale l'aggiornamento della posizione avviene tramite la formula:

$$ball.y = \frac{YMAX}{2} + [ball.v * sin\left(2 * PI * \frac{1}{dt} * ball.x\right)]$$

Viene calcolato il seno del valore della posizione x della palla e assegnato alla coordinata y della stessa.

c. Modalità Mouse

Nella modalità mouse è l'utente a stabilire, con il movimento del mouse la posizione della palla. Questa non segue alcuna traiettoria descrivibile tramite delle equazioni.
La libreria grafica Allegro mette a disposizione due funzioni legate all'utilizzo del mouse, quali:
mouse_x: restituisce la posizione lungo l'assse x del puntatore del mouse
mouse_y: restituisce la posizione lungo l'assse y del puntatore del mouse
Queste funzionalità forniteci da Allegro sono utilizzabili previa precedente inizializzazione del mouse tramite la funzione install_mouse();

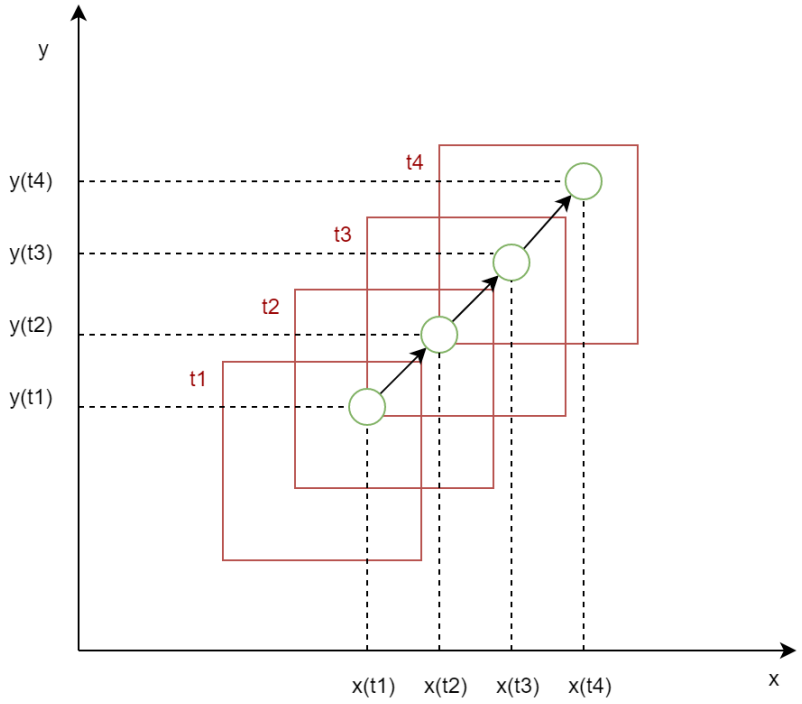
L'aggiornamento della posizione della palla avviene semplicemente tramite due assegnamenti:

$$ball.x = mouse\_x$$
$$ball.y = mouse\_y$$

3. Camera

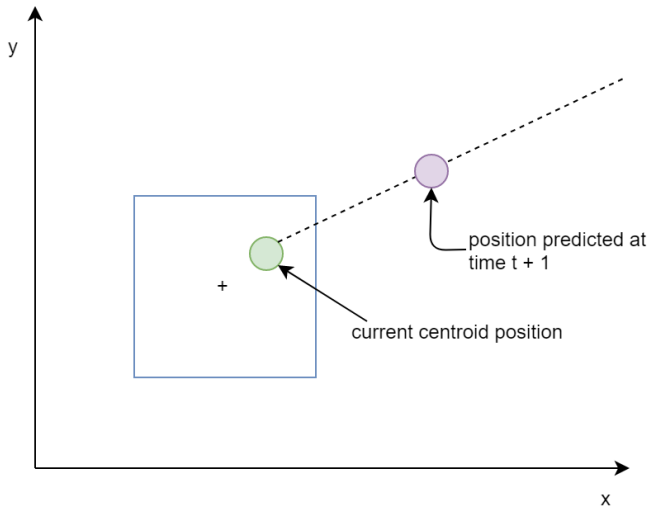
Si cattura l'immagine all'interno del quadrato che identifica la camera e si controlla se un pixel dentro questa area abbia colore uguale a due (il codice due identifica il colore verde).
Se ciò non viene trovato, si incrementano le dimensioni della camera e si fa muovere quest'ultima in una direzione pseudo-casuale (si utilizza lo stesso meccanismo con cui si aggiorna la posizione della pallina, ovvero la direzione futura viene calcolata all'interno di un range tenendo in considerazione la direzione attuale).

a. Calcolo centroide



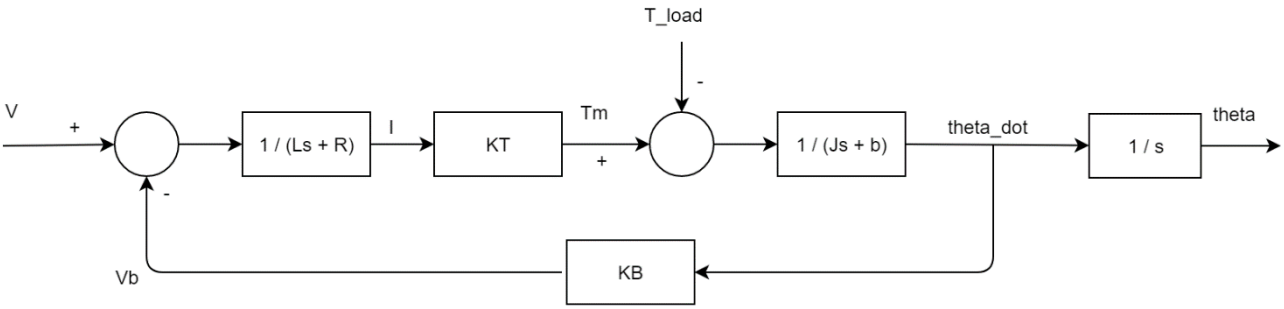
La camera ad ogni movimento cerca di prevedere la futura posizione della pallina calcolando il centroide di questa assumendo costante la velocità tramite la formula:

$$x(t + 1) = 2x(t) - x(t - 1)$$
$$y(t + 1) = 2y(t) - y(t - 1)$$



b. Motori

Per rendere maggiormente reale il movimento della camera consideriamo che questa sia mossa da due motori (pan and tilt), uno per gestire il movimento lungo l'asse x e l'altro per gestire il movimento lungo l'asse y.
Il diagramma a blocchi seguente mostra il modello del motore.



Semplificando il modello e ponendo:

$$K = \frac{K_t}{Rb + K_t K_b}$$

$$Tau = \frac{RJ}{Rb + K_t K_b}$$

Otteniamo:

$$out(k) = A * in(k - 1) + B * in(k - 2) + (1 + p) * out(k - 1) - p * out(k - 2)$$

dove

$$p = e^{-T/Tau}$$

$$A = K(T - Tau + pTau)$$

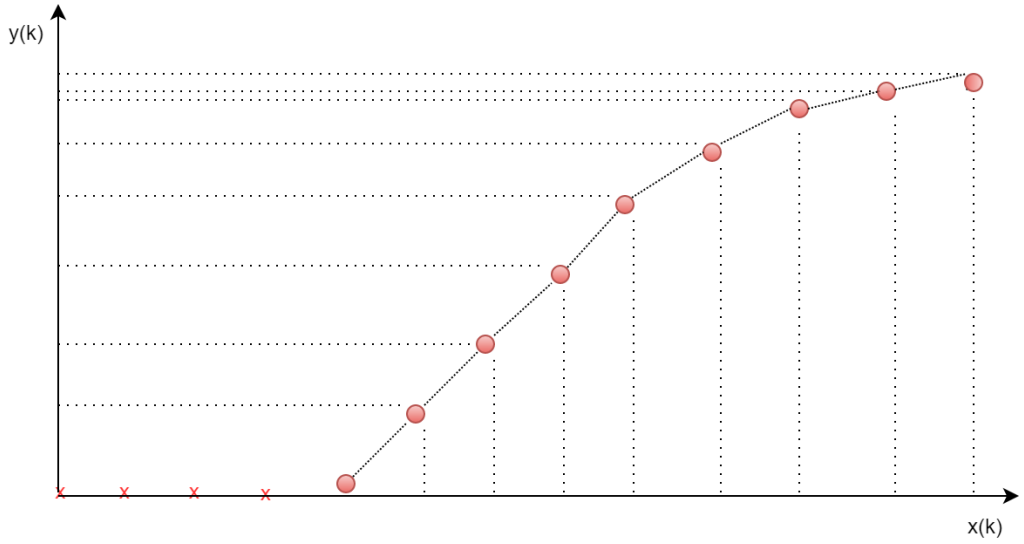
$$B = K(Tau - PT - pTau)$$

c. Filtri

Introduciamo anche un filtro nel ciclo chiuso per far fare in modo che i movimenti lungo i due assi da parte della camera vengano sempre ponderati in modo da avere maggior controllo tramite il controllore PID.
Utilizziamo un filtro passa basso descritto da:

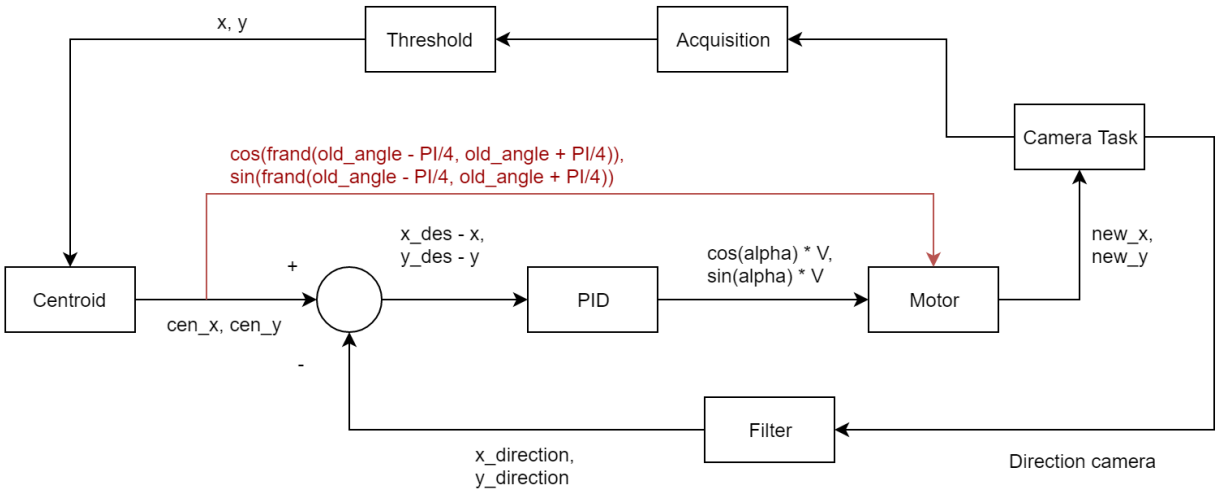
$$out(k) = p * out(k - 1) + (1 - p) * in(k - 1)$$

dove p è una costante del filtro e nella figura sotto viene esaminato il caso in cui p = 0.5.



Il PID che prende in ingresso la differenza tra la posizione desiderata e l'attuale posizione restituisce come output il valore controllato da dare in ingresso al motore.

d. Schema a blocchi per il controllo della camera



L'alternativa evidenziata in rosso nello schema viene eseguita ogni volta che la palla non viene rilevata all'interno dell'obiettivo della camera. In questo caso la camera oltre ad essere ridimensionata viene mossa in modo tale da poter ricatturare il prima possibile la palla.
In questo caso viene dato in input al motore un valore pseudo-randomico senza passare attraverso il PID (poiché se la x e la y in uscita dalla funzione threshold assumono valore -1 il centroide non viene calcolato, e le variabili cen_x e cen_y assumono valore non rilevante per poter effettuare il giusto controllo).
Dobbiamo considerare che la struttura nel closed-loop è duplicata (due PID, due motori e due filtri) per rendere maggiormente reale il controllo della camera poiché in genere le camere utilizzate per programmi di visual tracking utilizzano due motori per rendere indipendenti i movimenti lungo i due assi. Per questo motivo, essendo le due variabili da controllare non dipendenti l'una dall'altra bisogna duplicare anche tutta la struttura che fornisce gli input ai due plant del nostro sistema (i motori).

4. Errore

Nell' interfaccia grafica può anche essere osservato l'errore che in ogni istante la camera commette cercando di prevedere la futura posizione della palla.
Questo è identificabile come la distanza tra il centro del nostro obiettivo (la camera) e la palla.
Andando ad aumentare la velocità, si ha un incremento di questa distanza.
Per calcolare tale distanza ricorriamo alle seguenti formule:

$$mod\_x = (ball.x - camera.x) \text{ ^ } 2;$$
$$mod\_y = (ball.y - camera.y) \text{ ^ } 2;$$
$$e = sqrt(mod\_x + mod\_y) / 8;$$

La divisione per 8 viene eseguita per non aver mai un modulo dell'errore troppo grande che non sarebbe possibile disegnare sulla nostra interfaccia di dimensioni ridotte.
Per semplicità si è scelto di usare una scala 1:8, in modo da poter comunque osservare variazioni di tale errore all'aumentare del modulo della velocità senza comunque ricorrere in un grafico di dimensioni notevoli (non essendo questo lo scopo principale di questo lavoro).

5. Ptime

Nel file ptime.h sono dichiarate tutte le funzioni di utilità per poter convertire in struttura timespec i vari valori numerici e viceversa, per sommare due strutture timespec e per effettuare la comparazione fra due timespec. Oltre a queste vi sono delle macro per poter accedere alla tabella di conversione (definita nel file ptime.c tramite le espressioni corrispondenti).
Nel file ptime.c vengono implementate queste funzioni.
Tutto ciò è necessario per poter implementare con maggior precisione tutti le funzionalità legate alla periodicità dei vari task.

6. Ptask

Le vere e proprie funzionalità periodiche dei task vengono definite ed implementate nei file ptask.h e ptask.c.
Come prima cosa vengono definiti due nuovi tipi enum, sem_protocol e ptask_state che rappresentano rispettivamente il protocollo che viene utilizzato per implementare i semafori di mutua esclusione e lo stato in cui un task può trovarsi.
I protocolli utilizzabili per la realizzazione dei semafori sono PRIO_IINHERITANCE e PRIO_CEILING.
Se alla creazione di un semaforo non viene specificato nessun argomento, questo verrà creato senza alcun protocollo di gestione.
Nel file di implementazione sono definite due diverse funzioni che servono per la creazione dei semafori quando devono essere usati protocolli di gestione.
È stata dichiarato un nuovo tipo tpars per semplificare la gestione dei vari thread periodici avendo la possibilità di settare alcuni argomenti in maniera standard.
La vera e propria struttura di un task periodico viene definita tramite la struttura task_par (descrittore di thread periodico).

Le funzioni allocate_tp e release_tp allocano e rilasciano un descrittore di processo se questo è disponibile per essere utilizzato.

La funzione ptask_init avente come parametri la politica di scheduling e il protocollo da utilizzare per la creazione dei semafori inizializza il campo free di ogni descrittore in modo da sapere quanti descrittori sono disponibili per l'allocazione.
Viene anche dato l'indice 0 alla variabile first_free che tiene in considerazione il primo descrittore libero in modo che la allocate_tp possa allocare il descrittore di processo avente indice contenuto in questa variabile.
Vengono poi inizializzati i semafori per la protezione dei descrittori di processo che servono per proteggere i descrittori da accessi concorrenti.

La funzione ptask_create al cui interno avviene la creazione del task serve anche per inizializzare il primo descrittore libero in modo da salvare nei campi di questa struttura i parametri per rendere periodico il task e su cui le altre funzione dedicate alla gestione del periodo lavorano. All' interno di questa funzione vengono inoltre inizializzati i parametri, quali lo scheduling e la priorità che devono poi essere passati come secondo argomento alla pthread_create.
Se la pthread_create non va a buon fine viene rilasciato il descrittore precedentemente occupato.

La ptask_get_period effettua la conversione del campo periodo (di tipo timespec) in intero e ritorna tale valore. Per accedere alla struttura del descrittore sarà necessario acquisire e rilasciare il lock sul semaforo appartenente alla struttura in modo da garantire la mutua esclusione.

La ptask_wait _for_period cambia lo stato del task e lo pone in WFP (wait for period) e lo addormenta fino al prossimo activation time.
Quando viene poi riattivato (quando viene rincontrato il nuovo istante di attivazione) avviene l'aggiornamento della deadline e del prossimo activaction time.

La deadline_miss effettua una comparazione tra l'istante attuale e il valore salvato all' interno del campo del descrittore che identifica la deadline. Se questo valore è 1 ciò significa che si è verificata una miss di una deadline.

7. Control Motor

Questi file (control_motor.h e control_motor.c) implementano le funzionalità di gestione e controllo della camera.
Nell'header vengono dichiarate le strutture che caratterizzano i componenti che fanno parte del ciclo chiuso e che forniscono le funzionalità alla camera.
Si definisce un nuovo tipo, quale motor_type per differenziare i due motori.
Il motore PAN deve gestire lo spostamento laterale, lungo l'asse x.
Il motore TILT deve gestire lo spostamento verticale, lungo l'asse y.
Vengono inoltre dichiarate le funzioni che servono per inizializzare i vari componenti e per aggiornare lo stato di ognuno di essi.
La funzioni update_filter, update_motor e update_PID aggiornano gli array contenuti nelle rispettive strutture del filtro, dei motori e del controllore.
La funzione control_motor è quella che implementa il closed-loop per il controllo della camera aggiornando il filtro, il controllore e fornendo la variabile di controllo in ingresso al motore la cui uscita viene identificata come l'uscita del plant.