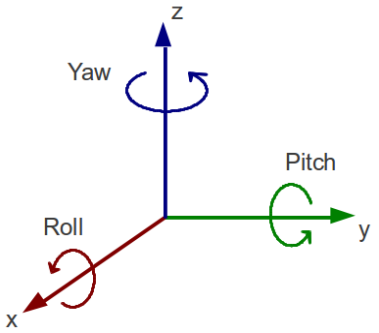
Corso di Laboratorio di Programmazione

Prova intermedia di programmazione

Questo assegnamento richiede di sviluppare un modulo C++ per la gestione dei dati provenienti da un sistema di sensori inerziali per il tracciamento della posa del corpo umano - per ulteriori dettagli, si veda ad esempio:

https://www.sense4motion.com/en/donanim-yazilim/mvn-awinda/

Il sistema è composto da 17 sensori fissati a punti specifici del corpo. Ciascuno di essi misura velocità e accelerazione delle rotazioni attorno ai tre assi cartesiani (si veda la figura a fianco). Ogni sensore fornisce un gruppo di 6 valori di tipo `double`, chiamato **lettura**, che deve essere rappresentato usando una opportuna struttura dati. I 6 valori prendono il nome di: `yaw_v`, `yaw_a`, `pitch_v`, `pitch_a`, `roll_v`, `roll_a`.



Il sistema nel suo complesso fornisce un insieme di dati, chiamato **misura**, che si compone di 17 letture.

Il modulo da sviluppare

Il modulo software che dovete sviluppare è implementato tramite la classe `InertialDriver` che deve essere in grado di ricevere le misure dal sensore (AKA produttore) e di fornirli a richiesta all'utilizzatore (AKA consumatore). Il modulo software deve implementare una struttura dati che rappresenta la misura, composta quindi da 17 letture, organizzato obbligatoriamente come array stile C di 17 elementi (ciascun elemento rappresentato in maniera opportuna). Tali misure devono essere salvate in ordine di arrivo e rese disponibili nello stesso ordine al consumatore, organizzando le misure in arrivo in un buffer gestito obbligatoriamente con la classe MyVector sviluppata in laboratorio (a vostra discrezione, con o senza i template). Il buffer ha dimensione finita, capace di contenere un numero di misure pari a `BUFFER_DIM`, che è una costante definita in una posizione opportuna del codice.
La politica di gestione è a buffer circolare, ovvero: quando il buffer è pieno, l'arrivo di una nuova misura causa la sovrascrittura di quella meno recente.

La classe `InertialDriver` deve implementare:

- La funzione `push_back` che accetta un array stile C contenente una misura e la memorizza nel buffer (sovrascrivendo la misura meno recente se il buffer è pieno);
- La funzione `pop_front` che fornisce in output un array stile C contenente la misura più vecchia e la rimuove dal buffer;
- La funzione `clear_buffer` che elimina (senza restituirle) tutte le misure salvate;
- La funzione `get_reading` che accetta un numero tra 0 e 16 e ritorna la corrispondente lettura della misura più recente, senza cancellarla dal buffer;
- L'overloading dell'`operator<<` che stampa a schermo l'ultima misura salvata (ma non la rimuove dal buffer).

Note per l'implementazione:

- Le funzioni dell'elenco precedente devono essere opportunamente completate con i tipi di ritorno; i parametri indicati devono essere opportunamente completati con eventuali

`reference` e `const`. È importante implementare le funzioni esattamente con il nome descritto (case sensitive) e con gli argomenti elencati;
- La classe può essere completata con variabili membro e funzioni membro a scelta;
- Deve essere implementata ogni altra funzione membro ritenuta necessaria.

Organizzazione del progetto
La classe `InertialDriver` (e altre eventuali classi) devono essere **correttamente separate nei file `.h` e `.cpp`**. Un ulteriore file `.cpp` deve contenere il `main`, usato per i test. Il progetto deve essere opportunamente strutturato in cartelle e sottocartelle analogamente a quanto illustrato nello schema seguente (relativo al progetto `Rational`):

```
Rational:
├── include
│   └── rational.h
├── README.txt
└── src
    ├── main.cpp
    └── rational.cpp
```

È **obbligatorio** allegare un file README in formato testo con la descrizione delle attività di **ciascun membro** del gruppo. Questo file non è la documentazione, che deve essere inserita sotto forma di commenti nel codice, ma un elenco di note aggiuntive per chi corregge, per esempio problemi riscontrati e non risolti.

Note allo sviluppo
Il progetto sviluppato deve essere compilabile sulla macchina virtuale (VLab/Taliercio.2020). Non è necessario sviluppare tutto sulla macchina virtuale, ma è importante fare verifiche periodiche per essere sicuri di non aver utilizzato codice fuori standard.

Plagi
Il codice verrà controllato per verificare eventuali plagi tra gruppi appartenenti allo stesso canale e in canali diversi. Verrà effettuato anche un controllo rispetto a codice disponibile online.

Consegna
Il compito deve essere consegnato su moodle. Il progetto dovrà essere compresso in un archivio che include una sola directory contenente il codice sorgente (organizzato in sottodirectory come descritto sopra).

L'archivio non deve contenere l'eseguibile, perché il sorgente sarà compilato in fase di correzione. Dopo la consegna su moodle, è necessario **verificare ciò che è stato consegnato** scaricando il progetto da moodle in una directory diversa da quella usata per sviluppare e provando a compilare il codice ed eseguire il programma.
Si suggerisce di consegnare anche compiti non completi. È tuttavia necessario che il software consegnato sia compilabile ed eseguibile.