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 LIDAR. Un lidar è un sensore capace di effettuare misurazioni di distanza usando un fascio laser. Il principio di funzionamento è simile a quello di un metro laser, ma nel caso del lidar il fascio laser è posto in rotazione e la misurazione è effettuata a intervalli di angolo regolari. Un lidar perciò scandisce un piano nello spazio e riporta le misure di distanza rilevate su quel piano a intervalli regolari.
Potete trovare ulteriori dettagli riguardo ai lidar a questo link: https://it.wikipedia.org/wiki/Lidar

Un lidar genera un flusso di dati costituito da un insieme di `double` che rappresentano le letture ai vari angoli. Il termine **lettura** identifica la singola misurazione di distanza a un dato angolo, mentre il termine **scansione** identifica l'insieme di tutte le letture ai diversi angoli, acquisite in sequenza dal sensore. L'angolo che separa due letture consecutive prende il nome di *risoluzione angolare* ed è un dato di progetto di ogni lidar. Due modelli di lidar diversi possono avere risoluzioni angolari diverse.

Esempio 1. Supponete che un lidar con risoluzione angolare di 1° rilevi le distanze riportate in tabella:

Angolo [°]	0	1	2	3	4	5	6	7	8
Lettura [m]	1.01	1.10	1.65	1.45	1.3	0.98	0.9	0.88	0.75

Poiché la risoluzione angolare è una costante, il flusso dati di tale lidar è semplicemente la successione delle varie letture di distanza, quindi i dati corrispondenti alle misurazioni in tabella sono i seguenti:

```
1.01 1.10 1.65 1.45 1.3 0.98 0.9 0.88 0.75
```

Esempio 2. Un LIDAR con risoluzione angolare di 0.5° fornisce ugualmente una stringa di numeri in successione, ma tali numeri sono riferiti a letture con una differenza di 0.5° tra due letture consecutive. Quindi, un sensore con risoluzione angolare di 0.5° che fornisce i dati:

```
0.1 0.15 0.13 0.22 0.76 0.99
```

rileva le seguenti letture:

Angolo [°]	0	0.5	1	1.5	2	2.5
Lettura [m]	0.1	0.15	0.13	0.22	0.76	0.99

I lidar che dovete gestire hanno **un angolo di vista costante di 180°**, ovvero il range di misurazione è sempre di [0,180°], quindi un sensore con risoluzione di 1° fornisce 181 valori per ogni scansione, un sensore con risoluzione di 0.5° fornisce 361 valori per ogni scansione, un

sensore con risoluzione di 0.25° fornisce 721 valori per ogni scansione, ecc. I lidar che dovete gestire possono avere una risoluzione compresa tra 0.1° e 1°.

Il modulo da sviluppare

Il modulo che dovete sviluppare è implementato tramite la classe `LidarDriver` che deve essere in grado di ricevere i dati dal sensore (AKA produttore) e di fornirli a richiesta all'utilizzatore (AKA consumatore).

Il modulo deve implementare un buffer che salva le **scansioni** in ordine di arrivo e le rende disponibili nello stesso ordine al consumatore. Il buffer ha dimensione finita, capace di contenere un numero di scansioni pari a `BUFFER_DIM`, che è una costante definita in una posizione opportuna del codice. Si consideri come esempio un sistema il cui `BUFFER_DIM` vale 10 e dotato di una risoluzione angolare di 1°. Il buffer in questione deve essere capace di contenere 10 scansioni, ciascuna costituita da 181 letture. La politica di gestione è a buffer circolare, ovvero: quando il buffer è pieno, l'arrivo di una nuova scansione causa la sovrascrittura di quella meno recente.

La classe `LidarDriver` deve implementare:

- La funzione `new_scan` che accetta uno `std::vector<double>` contenente una scansione e lo memorizza nel buffer (sovrascrivendo la scansione meno recente se il buffer è pieno). Questa funzione esegue anche il controllo di dimensione: se le letture sono in numero minore del previsto, completa i dati mancanti a `0`; se sono in numero maggiore, li taglia;
- La funzione `get_scan` che fornisce in output uno `std::vector<double>` contenente la scansione più vecchia del sensore e la rimuove dal buffer;
- La funzione `clear_buffer` che elimina (senza restituirle) tutte le scansioni salvate;
- La funzione `get_distance` che accetta un angolo espresso in gradi (`double`) e ritorna la lettura corrispondente a tale angolo nella scansione più recente acquisita dal sensore; tale scansione non è eliminata dal buffer, e se l'angolo richiesto non è disponibile ritorna la lettura relativa al valore di angolo più vicino;
- L'overloading dell'`operator<<` che stampa a schermo l'ultima scansione salvata (ma non la rimuove dal buffer).

Note per l'implementazione:

- La risoluzione del `LidarDriver` è fissa per tutta la durata di vita del relativo oggetto (il che deve avere un riscontro nel costruttore).
- Per la gestione del buffer interno alla classe è possibile utilizzare un container come `std::vector` oppure usare direttamente l'allocazione dinamica della memoria (`new`, `new[]`, `delete`, `delete[]`);
- Le funzioni dell'elenco precedente devono essere opportunamente completate con i tipi di ritorno; gli argomenti indicati devono essere opportunamente completati con eventuali `reference` e `const`, se necessario o opportuno. È importante implementare le funzioni esattamente con il nome descritto (case sensitive) e con gli argomenti elencati;
- La classe può essere completata con le variabili membro utili e con funzioni membro a scelta;
- Deve essere implementata ogni altra funzione membro ritenuta necessaria.

La classe `LidarDriver` (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 inoltre **obbligatorio** allegare un file README in formato testo con la descrizione delle attività di **ciascun membro** del gruppo.