Corso di Laboratorio di Programmazione

Assegnamento 1 – Gestore dati laserscanner
25/11/2020

Il problema
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.

Ulteriori dettagli qui:
https://it.wikipedia.org/wiki/Lidar

Tale sensore genera un flusso di dati costituito da un insieme di double che rappresentano le letture ai vari angoli. L'angolo che separa due letture consecutive prende il nome di *risoluzione angolare* 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° fornirebbe ugualmente una stringa di numeri in successione, ma tali numeri sarebbero riferiti a letture con una differenza di 0.5° tra una lettura e la successiva. 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

rappresenta 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 di 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 laserscanner che dovete gestire possono avere una risoluzione compresa tra 1° e 0.1°.

Definiamo *lettura* il singolo dato, *scansione* il gruppo di letture consecutive corrispondenti a una passata da 0° a 180°.

Il modulo da sviluppare

Il modulo che dovete sviluppare è implementato tramite la classe LaserScannerDriver 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. Quindi, se (per esempio) un sistema ha BUFFER_DIM che vale 10 e una risoluzione angolare di 1°, il buffer è capace di contenere 10 scansioni, ciascuna costituita da 180 valori. 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 più vecchia in memoria. La risoluzione del laserscanner è fissa per tutta la durata di vita (il che deve avere un riscontro nel costruttore).

La classe LaserScannerDriver deve implementare:

- La funzione new_scan che accetta un vector<double> contenente una nuova scansione fornita dal sensore e la memorizza nel buffer (sovrascrivendo quella più vecchia 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 (in maniera opportuna) un vector<double> contenete la scansione più vecchia del sensore e la rimuove dal buffer;
- La funzione clear_buffer che elimina (senza ritornarle) tutte le letture salvate;
- La funzione get_distance che accetta un angolo (double) e ritorna il valore di distanza (cioè la lettura) corrispondente a tale angolo (tenendo conto della risoluzione angolare) nella scansione più recente acquisita dal sensore – tale scansione non è eliminata dal buffer, e se l'angolo richiesto non è disponibile la funzione ritorna il 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:

- Per la gestione del buffer interno alla classe è obbligatorio usare direttamente l'allocazione dinamica della memoria (new, new[], delete, delete[]) - non potete usare gli standard container (come std::vector e std::list) né altre forme di strutture dati che gestiscono la memoria in maniera automatica;
- Le funzioni dell'elenco precedente devono essere opportunamente completate con i tipi di ritorno; gli argomenti indicati devono essere opportunamente completati con 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 le funzioni membro a scelta del progettista.
- Deve essere implementata ogni altra funzione membro ritenuta necessaria (distruttore?) per evitare memory leak.
- **I memory leak comportano una forte decurtazione del punteggio**, perciò si consiglia di **testare bene** il funzionamento del codice (anche con il debugger).

Consegna del software

Il software deve essere organizzato in maniera rigida nei seguenti file:

- LaserScannerDriver.h: header contenente la dichiarazione della classe ed eventuali funzioni definite in-class;
- LaserScannerDriver.cpp: file contenente la definizione delle funzioni membro;
- main.cpp: file contenente il main che testa la classe LaserScannerDriver.

Questi file devono essere contenuti in un'unica directory chiamata LSDriver (**case sensitive!**), che deve essere compressa in un archivio .zip e caricata su moodle.
Non sono date specifiche riguardo al file main.cpp: è possibile testare la classe a piacere. Si fa presente che in fase di correzione la classe sarà testata con un main.cpp diverso, che verificherà l'eventuale presenza di errori o debolezze del codice.
La suddivisione del codice nei vari file deve essere effettuata seguendo le linee guida viste a lezione. Errori in questo passaggio (per esempio, includere un file .cpp) comportano una decurtazione del punteggio.

Note

- Non devono essere usati namespace – questo serve solo per semplificare la correzione, ma sarebbe comunque una buona idea usarli;
- Il codice sviluppato deve essere robusto, quindi gli input di ogni funzione devono essere verificati, e le funzioni membro devono sempre rispettare gli invarianti;
- Nella classe, inserite un commento che indica quali sono gli invarianti che la vostra classe deve rispettare;
- È possibile scambiare idee, ma **non è possibile scambiare codice**. Le consegne saranno confrontate tra loro con un apposito software di controllo anti-copia. Eventuali copiature porteranno a **gravi decurtazioni del punteggio**.