Programmazione e Amministrazione di Sistema

Progetto C#

Università Milano Bicocca

Dipartimento di Informatica Disco

CdL triennale in Informatica

* Lorenzo Fenili
* Silvio Messi
* Luca Perico

Introduzione

L’ obiettivo di questo progetto è quello di implementare il server a cui un simulatore (Xbus Simulator) si deve connettere per trasmettere dei campioni generati ad un set di 5 sensori inerziali XSens. Questo server, otre ad essere in grado di acquisire correttamente questi campioni, deve essere in grado di analizzarli, visualizzarli su grafici ed eseguire algoritmi che ricavano informazioni utili riguardo alla persona che indossa tali sensori. Nelle prossime pagine verrano spiegate in maniera dettagliata la varie parti che compongono l’ algoritmo, in sostanza le classi che compongono il progetto sono:

Interfaccia grafica:

* Form1
* FormGraph

Classi elaborazioni dati e d’ appoggio:

* Server
* Buffer
* ElaboraDati
* AngoloEulero

***Il principio fondamentale con il quale abbiamo realizzato il progetto (seguendo quanto indicato nella consegna) è il seguente;***

***Il server acquisisce una finestra di 500 campioni e li analizza grazie ad un thread che lavora in parallelo con il server. Successivamente vengono acquisti altri 250 campioni, che insieme agli ultimi 250 campioni della finestra precede compongono la nuova finestra di analisi. Questo permette di analizzare finestra di campioni sovrapposte e di non perdere aventi che si verificano a cavallo di due finestre.***

Classe Form1

Questo form rappresenta l’ interfaccia grafica del server. E’ stata progettata nel seguente modo:

FOTO FORM1 AGGIORNATO

Praticamente abbiamo un tasto “START SERVER” che permette di metter il server in ascolto sulla macchina. Il server, una volta avviato, si mette in attesa del cliente e quando questo si connette e trasmette dei dati, vedremo comparire nella text box il numero di campioni ricevuto, aggiornato in tempo reale. Il server inoltre ci avvisa quando viene avviato il thread che analizza i dati.

L’interfaccia grafica permette, tramite gli apposite check box, di selezionare fino ad un massimo di 4 grafici che verranno visualizzati appena si avrà un numero sufficiente di campioni da analizzare. I grafici che il nostro programma è in grado di visualizzare sono:

* modulo dell’accelerometro (sensore a piacere)
* modulo del giroscopio (sensore a piacere)
* angolo theta;
* angolo theta senza discontinuità
* deviazione standard del modulo dell’accelerometro/ giroscopio (sensore a piacere)
* angolo Yaw
* angolo Yaw senza discontinuità
* angolo Pitch;
* angolo Pitch senza discontinuità
* angolo Roll;
* angolo Roll senza discontinuità
* dead reckoning

Nella parte destra dell’interfaccia a grafica invece, abbiamo la possibilità (una volta che tutti i campioni sono stati acquisiti) di andare a verificare il contenuto di un campione, oppure testare la funzioni che analizzano i dati. Questa funzionalità inizialmente e stata utilizzate per debuggare il programma, ma è stata mantenuta data la sua utilità.

Class FormGraph

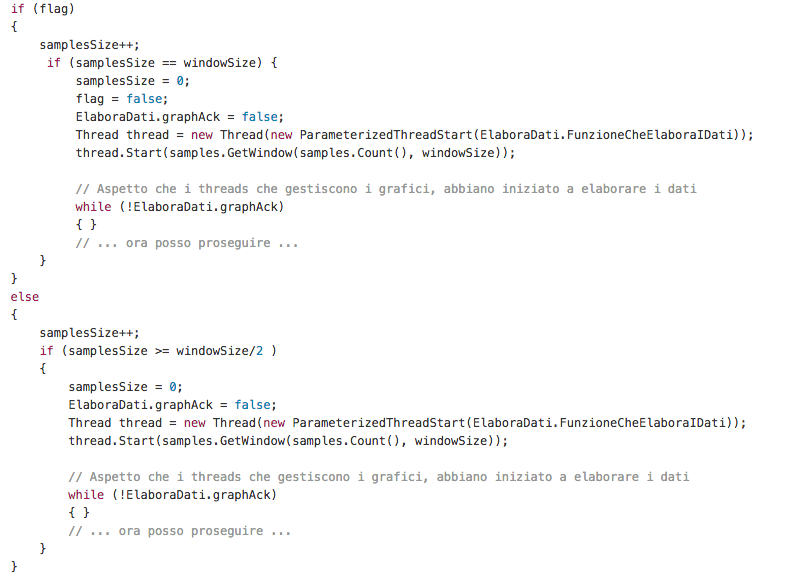
FOTO FORM GRAPH AGGIORNATO

SEGUITO CON LOGICA DEI GRAFICI (Luca)

Classe Server

Il compito principale di questa classe è quello di aprire un socket in ascolto sulla porta 45555 della macchina. Una volta che il client si connette viene avviato un thread per gestire la comunicazione tra client e server. Nel metodo eseguito dal thread (ReadFromSocket) viene analizzato, come indicato da documentazione fornita, lo stream di byte che il sensore ci manda in modo da poter estrapolare i campioni i quali verranno salvati in una struttura dati idonea, rappresentata dalla classe Buffer (descritta successivamente).

NOTA: un singolo campione è salvato in una matrice dimensionale float[a,b] dove l’ indice a rappresenta il sensore (da 0 a 4) e l’ indice b la tipologia di dato (da 0 a 12) ossia accelerometro asse x,y,z, giroscopio asse x,y,z, magnetometro asse x,y,z e i 4 quaternioni.

A questo punto, quando ho acquisto un certo numero di campioni (500 nel nostro caso), viene avviato un thread che si occupa di elaborare i campioni acquisti fino a quel punto. Dopo aver acquisito la prima finestra il thread viene richiamato in modo da avere delle finestre sovrapposte (nel nostro caso ogni 250 campioni).

COMMENTO CODICE

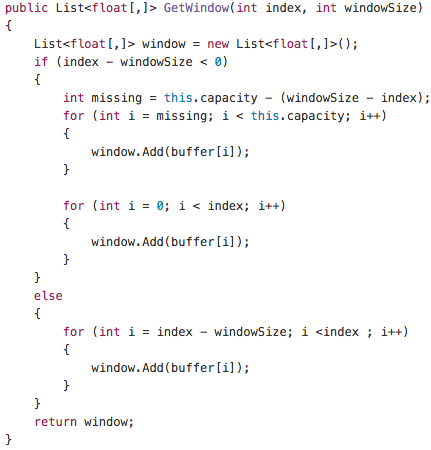
La variabile booleana “flag” a true indica che sto acquisendo i dati della prima finestra e dunque devo continuare ad acquisire i campioni fino a quando non ho riempito almeno una finestra (sampleSize == windowSize). Dalla seconda finestra in poi (flag = false), porto usare metà dei campioni acquisti nella finestra precedente insieme a quelli appena acquisiti.

Un altra funzionalità importante di questa classe è il salvataggio dei campioni, una volta che sono stati acquisti tutti, nel file Samples.csv

Classe Buffer

Sostanzialmente questa classe rappresenta un struttura dati ad hoc per il nostro problema-

In pratica è un buffer di dimensione fissa (750), in grado di contenere più di una finestra di campioni. I campioni vengono inseriti uno dopo l’ altro nel buffer fino a quando questo non è pieno. A questo punto i nuovi campioni saranno inseriti partendo dalla posizione 0 del buffer. Questa logica ci permette salvare i campioni di una finestra e mantenerli anche per la finestra successiva, senza rischiare di perdere dei campioni.

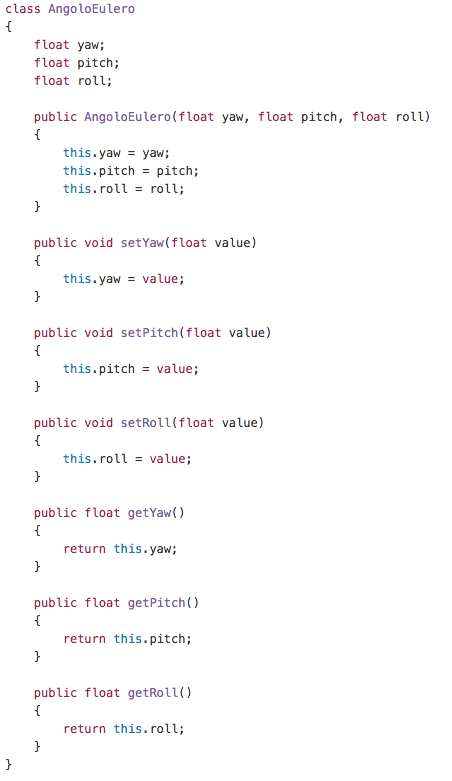
Il metodo fondamenta di questa classe e GetWindow che ci permette di estrapolare dal buffer la finestra di campioni che dobbiamo esaminare, semplicemente passando la dimensione della finestra (no dobbiamo preoccuparci se i campioni sono slavati un po in coda e un po in testa a buffer )

COMMENTO CODICE

Il metodo ritorna una lista di campioni (List<float[,]>) che sarà passata al thread che si occupa di analizzare i dati

Classe AngoloEulero

Viene riportato il codice della classe, dato che è auto esplicativo



Classe Elabora Dati

Questa classe è un po' il cuore del progetto in quanto contiene tutte le funzioni che si occupano di elaborare i campioni.

Di seguito ci sono le signature dei metodi principali che la classe implementa con una breve spiegazione

* public static List<float> Modulation(List<float[,]> samples, int \_S, int \_type)

Questa funzione mi permette di calcolare il modulo dei tre assi dell’accelerometro o del giroscopio. Grazie ai nuovi risultati ottenuti io ho dei dati che non sono specifici per asse.

La funzione prende in ingresso i campioni, il numero di sensore e un identificatore che mi permette di capire se applicare la modulazione sull’accelerometro o giroscopio.

* public static List<float> Smoothing(List<float> module)

Lo Smoothing è una funzione molto utile nel progetto, ci permette di ridurre il rumore causato dal segnale e appiattire/mediare i dati. La funzione non fa altro che prendere in ingresso una lista di float modulati e mediarli tramite una media mobile.

* public static float MobileMean(int i,List<float> value\_mean)

Questa funzione mi permette di restituire la media mobile calcolata in quella determinate finestra di valori grazie all’indice passato alla funzione.

* public static List<float> DeviazioneStandard(List<float> \_value)

La funzione DeviazioneStandard mi calcola deviazione standard mobile(grazie alla funzione MobileMean che mi restituisce anch’essa una media mobile) dei valori modulati dell’accelerometro e grazie ai risultati ottenuti riesco a capire se una persona è in movimento o no.

* public static float[] RIfun(float[] input)

La funzione prende un vettore di valori in ingresso e restituisce il vettore monodimensionale RI[].

L' elemento RI[i] rappresenta il Rapporto Incrementale della funzione nel punto i.

* public static List<AngoloEulero[]> AngoliEulero(List<float[,]> campioni)

La funzione ricava gli angoli di Eulero di ogni campione per ogni sensore tramite le formule indicate nel testo del progetto.

* public static void FunzioneCheElaboraIDati(Object obj)

Questa è la funzione principale della classe (quella che viene eseguita del thread che si occupa dell’analisi dei campioni) . Sostanzialmente non fa altro che chiamare tutte le funzione di analisi dei dati e salvare il risultato in apposite strutture dati. Inoltre se non è ancora partito il thread che gestisce i grafici, provvede ad avviarlo.

* public static void InizializzaGrafico()

Viene chiamata solo al primo avvio del thread che gestisce il grafico e inizializza gli assi e le etichette del grafico stesso.

* public static void DisegnaSulGrafico()

Questa funzione viene chiamata quando si lancia il thread che gestisce il grafico e passa le strutture dati che contengono le elaborazioni dei vari algoritmi ai grafici

* public static void CalcolaMoto(List<float> SD)

Passandogli i valori delle deviazioni standard dell’accelerometro questa funzione ha il compito di scrivere su un file quando la persona è in movimento o no, scrivendo anche l’inizio e la fine di ogni evento.

* public static void CalcoloGirata(List<float> angoliTheta)

La funzione grazie all’ angolo theta è in grado di calcolare quando avvengono girate significative (> 6 gradi) . Una volte che queste sono state rilevate vengono salvate sul file degli eventi

* public static void CalcoloInclinazione(List<float> samples)

Questa funzione mi permette di calcolare quando una persona è in piedi, seduta o sdraiata e

scriverlo su un file con i rispettivi tempi di inizio e fine. I dati utilizzati per capire l’inclinazione

sono i valori sull’asse delle Y dell’accelerometro e del sensore che si trova sul bacino.

* public static List<float[]> DeadReckoning(List<float> SD, List<float>yaw)

Sono state volontariamente omesse funzioni secondari che vengono usate dalle funzioni sopra elencate. Inoltre non sono stati riportati frammenti di codice in quanto il progetto e ben commentato ed eventuali dubbi possono essere chiariti durante la presentazione del progetto.

Conclusione