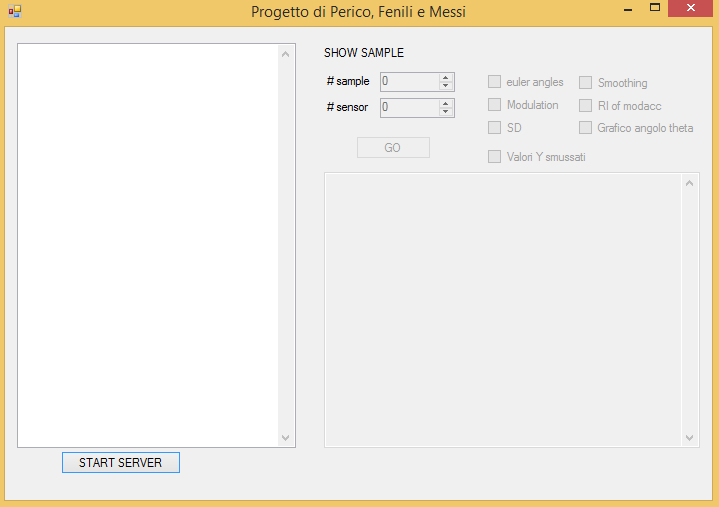
Programmazione e Amministrazione di Sistema

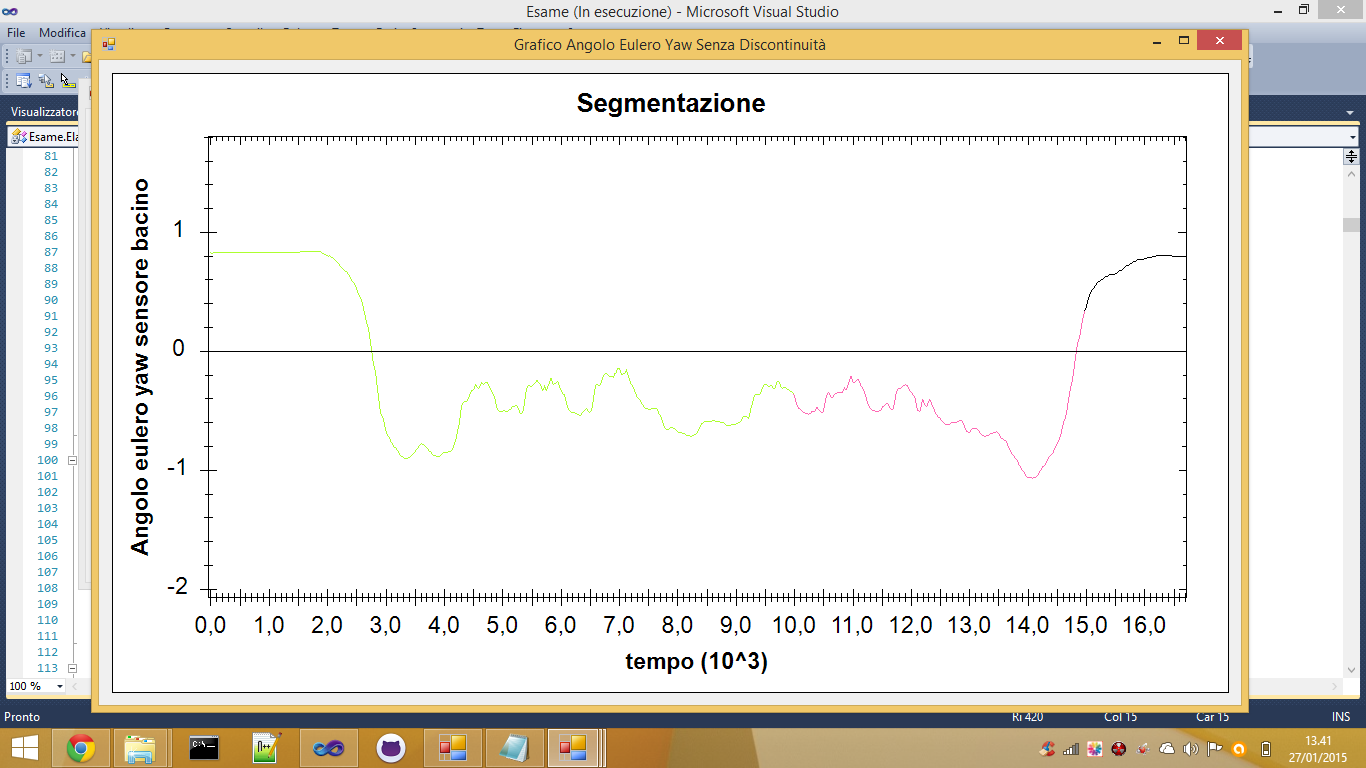
Progetto C#

Introduzione

L’ obiettivo del progetto è quello di implementare il server a cui i il simulatore (Xbus Simulator) si deve connettere per trasmettere i campioni. Abbiamo realizzato l’ interfaccia del server nel seguente modo



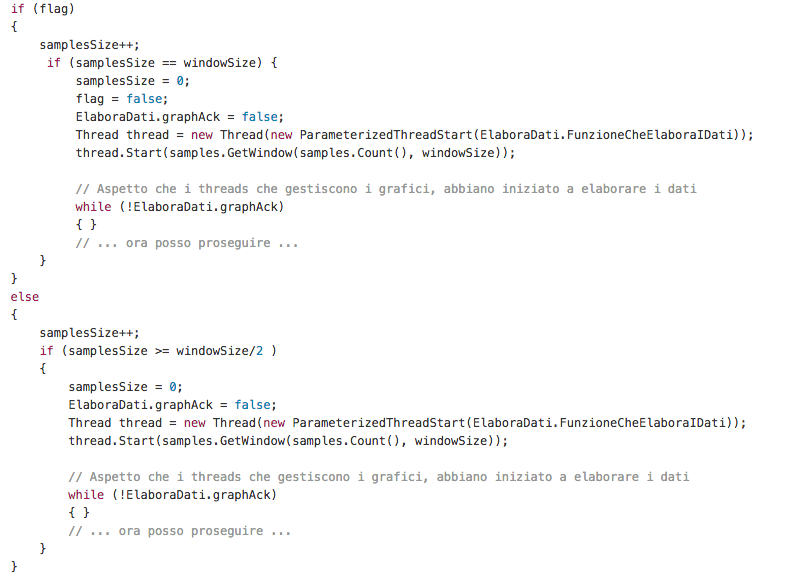
Praticamente abbiamo un tasto “START SERVER” che permette di metter il server in ascolto sulla macchina. Una volta avviato il server si mette in attesa del cliente e quando questo si connette e trasmette dei dati, vedremo comparire nella text box il numero di campioni ricevuto, aggiornato in tempo reale.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



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

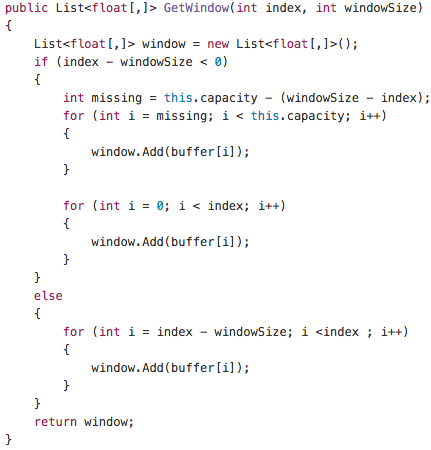
La variabile booleana “flag” a true indica che sto acquisendo i dati della prima finestra e dunque devo continuare ad acquisire i campioni fino a quando non ho riempito almeno una finestra (sampleSize == windowSize). Dalla seconda finestra in poi (flag = false), porto usare metà dei campioni acquisti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

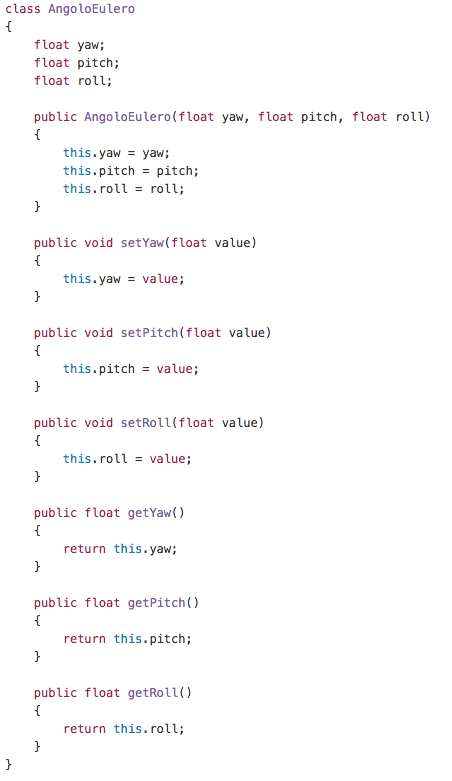
Il metodo fondamenta di questa classe e GetWindow che ci permette di estrapolare dal buffer la finestra di campioni che dobbiamo esaminare, semplicemente passando la dimensione della finestra (no dobbiamo preoccuparci se i campioni sono slavati un po in coda e un po in testa a buffer )

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)
* public static List<float> Smoothing(List<float> module)
* public static float MobileMean(int i,List<float> value\_mean)
* public static List<float> DeviazioneStandard(List<float> \_value)
* public static float[] RIfun(float[] input)
* public static List<AngoloEulero[]> AngoliEulero(List<float[,]> campioni)
* public static void FunzioneCheElaboraIDati(Object obj)
* public static void InizializzaGrafico()
* public static void DisegnaSulGrafico()
* public static void CalcolaMoto(List<float> SD)
* public static void CalcoloGirata(List<float> angoliTheta)
* public static List<float[]> FunzioneOrientamento(List<float[,]> samples)
* public static void CalcoloInclinazione(List<float[,]> window)