0.1 Programma che si occupa di post-elaborare i dati

Dopo aver acquisito i dati di arduino e del kinect bisogna effettuare una post elaborazione per ottenere coppie di valori riferiti a circa lo stesso istante temporale. Il programma usato per fare ciò riprende molte cose dal codice del programma "prg_get_data", le librerie usate sono le stesse ed anche le classi "kin_file_manager", "kin_file_manager" e "data_structure.h". L'unica classe nuova è "make_dataset":

```
class make_dataset
{
private:
  // dichiaro i puntatori alle classi per la gestione dei file cos\'i da renderle disponibili
      ↪ per tutta la classe
  ard_file_manager *f_a;
  kin_file_manager *f_k;

  // delta di accettazione in millisecondi
  const int range = 35;

  // dichiaro 2 liste per conservare i dati
  list<arduino_data> a_dat;
  list<kinect_data> k_dat;

  // devo dichiararare il numero di ingressi e di uscite della rete
  static const std::size_t n_in = 9;
  static const std::size_t n_out = 4;

  // dichiaro una lista di tipo dataset_data che verr\'a riempita con i dati accoppiati
  list<dataset_data<float, n_in, float, n_out>> dataset;

  // questa funzione si occupa di scrivere il dataset sul file
  void write_dataset(string dataset_file_name){...}

  // restituisce 1 solo se i 2 tempi che gli vengono passati differiscono meno della soglia
      ↪ impostata sopra
  bool check_sync(uint64_t t1, uint64_t t2){...}

  // questo metodo sposta i dati dalle strutture "arduino_data" e "kinect_data" alla struttura "
      ↪ dataset" dichiarata globalmente nella classe
  void transfer_data_to_dataset(arduino_data* ard_iter, kinect_data* kin_iter){...}

  // restituisce la differenza temporale tra i tempi passati alla funzione
  uint32_t time_distance(uint32_t t1, uint32_t t2){...}

  // sincronizza i dati caricati in "a_dat", "k_dat" e li sposta in "dataset" attraverso la
      ↪ funzione "transfer_data_to_dataset". Questa funzione non tiene conto dei "cloni", nella
      ↪ sezione successiva vi \'e una spiegazione pi\'u dettagliata.
  void sync_with_clones(){...}

  // stessa cosa della funzione precedente ma eliminando i cloni.
  void sync_without_clones(){...}

  // carica i dati dai file usando "f_a" e "f_k", sposta i dati acquisiti in "a_dat" e k_dat"
  void load_data(){...}

public:
  // l'unica funzione accessibile \'e il costruttore che provvede ad inizializzare i  file
      ↪ manager, caricare i dati, sincronizzarli (con o senza cloni) e scrivere il file del
      ↪ dataset con i dati appena generati.
  make_dataset(string ard_file_name, string kin_file_name, string dataset_file_name, bool
      ↪ want_clones)
  {
    f_a = new ard_file_manager(ard_file_name, std::ios::in);
    f_k = new kin_file_manager(kin_file_name, std::ios::in);

    load_data();

    if (want_clones)
    {
      sync_with_clones();
    }
```

```
    else
    {
      sync_without_clones();
    }

    write_dataset(dataset_file_name);

  }

  // distruttore della classe
  ~make_dataset(){...}

};
```

Listing 1: classe make_dataset

In questa classe le 2 funzioni principali sono "sync_with_clones" e "sync_without_clones" che provvedono a costruire le coppie di dati ed a salvarle nella struttura dati "dataset". Analizziamo la funzione "sync_with_clones":

```
void sync_with_clones()
{
  uint64_t time_old_good_pair = 0;

  uint64_t time_old_ard = 0;
  uint64_t time_old_kin = 0;

  auto ard_iter = a_dat.begin();
  auto kin_iter = k_dat.begin();

  uint32_t ard_cont = 0;
  uint32_t kin_cont = 0;

  uint32_t old_ard_cont = 0;
  uint32_t old_kin_cont = 0;

  while (true)
  {
    // controllo la sincronia
    if (check_sync(ard_iter->frame_time, kin_iter->frame_time))
    {
      // aggiungo il dato al dataset
      transfer_data_to_dataset(&(*ard_iter), &(*kin_iter));

      // stampo delle info utili a capire la qualit\'a dei dati acquisiti
      cout << abs((long long)(ard_iter->frame_time - kin_iter->frame_time)) << "  kin: " <<
          ↪ kin_cont << " ard: " << ard_cont << endl;
      cout << "time since another good pair: " << ((ard_iter->frame_time + kin_iter->frame_time)
          ↪  / 2 - time_old_good_pair) << endl;
      cout << "ard frame rate: " << (1.0f / ((ard_iter->frame_time - time_old_ard) / 1000.0f)) <
          ↪ < "Hz" << endl;
      cout << "kin frame rate: " << (1.0f / ((kin_iter->frame_time - time_old_kin) / 1000.0f)) <
          ↪ < "Hz" << endl;

      // salvo i tempi
      time_old_good_pair = (ard_iter->frame_time + kin_iter->frame_time) / 2;
      time_old_ard = ard_iter->frame_time;
      time_old_kin = kin_iter->frame_time;

      // controllo se ci sono pi\'u dati che si accoppiano con uno solo (solo debug)
      if ((kin_cont == old_kin_cont) || (ard_cont == old_ard_cont))
      {
        cout << "dati multipli nello stesso range" << endl;
      }
      else
      {
        old_kin_cont = kin_cont;
        old_ard_cont = ard_cont;
      }

      cout << endl;
```

```
48
49      }
50      else
51      {
52        cout << "dato scartato" << endl << endl;
53      }
54
55      // incremento chi tra i 2 dati \'e il pi\'u vecchio
56      if (ard_iter->frame_time >= kin_iter->frame_time)
57      {
58        kin_iter++;
59        kin_cont++;
60      }
61      else
62      {
63        ard_iter++;
64        ard_cont++;
65      }
66
67      // se sono arrivato alla fine esco dal ciclo
68      if ((kin_iter == k_dat.end()) || (ard_iter == a_dat.end())) { break; }
69    }
70  }
```

Listing 2: funzione sync_with_clones

Questa prima funzione effettua il pair dei dati senza però tener conto che, dati i differenti frame rate di arduino e del kinect, un singolo dato del kinect potrebbe accoppiarsi con più di un dato di arduino e viceversa. Questi "cloni" interferiscono con il processo di apprendimento della rete e quindi si è passati alla funzione "sync_without_clones" che effettua la stessa operazione ma tenendo conto che potrebbero esserci più dati di un tipo accoppiati con uno dell'altro e salva solo la coppia che ha la differenza di tempo minore.

```
1  void sync_without_clones()
2  {
3    uint64_t time_old_good_pair = 0;
4
5    uint64_t time_old_ard = 0;
6    uint64_t time_old_kin = 0;
7
8    auto ard_iter = a_dat.begin();
9    auto kin_iter = k_dat.begin();
10
11   uint32_t ard_cont = 0;
12   uint32_t kin_cont = 0;
13
14   uint32_t old_ard_cont = 0;
15   uint32_t old_kin_cont = 0;
16
17   kinect_data k_tmp = *kin_iter;
18   arduino_data a_tmp = *ard_iter;
19
20   for(;;)
21   {
22     // controllo la sincronia
23     if (check_sync(ard_iter->frame_time, kin_iter->frame_time))
24     {
25       // stampo i tempi per valutere la qualit\'a del dataset
26       cout << "common " << abs((long long)(ard_iter->frame_time - kin_iter->frame_time)) << "
           ↪ kin: " << kin_iter->contatore << " ard: " << ard_iter->contatore << endl;
27       cout << "ard frame rate: " << (1.0f / ((ard_iter->frame_time - time_old_ard) / 1000.0f)) <
           ↪ < "Hz" << endl;
28       cout << "kin frame rate: " << (1.0f / ((kin_iter->frame_time - time_old_kin) / 1000.0f)) <
           ↪ < "Hz" << endl;
29
30       // salvo i tempi attuali
31       time_old_ard = ard_iter->frame_time;
32       time_old_kin = kin_iter->frame_time;
33
34       // se il contatore del kinect non \'e variato dall'ultima volta significa che ci sono pi\'
           ↪ u dati riferiti ad arduino che si accoppiano ad un solo dato del kinect
35       if (kin_cont == old_kin_cont)
```

```
36        {
37
38          cout << "multi ard in the same kin" << endl;
39          if (time_distance(kin_iter->frame_time, ard_iter->frame_time) < time_distance(k_tmp.
                ↪ frame_time, a_tmp.frame_time))
40          {
41            a_tmp = *ard_iter;
42          }
43        }
44        // se il contatore dei arduino non \'e variato dall'ultima volta significa che ci sono pi
              ↪ \'u dati riferiti al kinect che si accoppiano ad un solo dato di arduino
45        else if (ard_cont == old_ard_cont)
46        {
47          cout << "multi kin in the same ard" << endl;
48          if (time_distance(kin_iter->frame_time, ard_iter->frame_time) < time_distance(k_tmp.
                ↪ frame_time, a_tmp.frame_time))
49          {
50            k_tmp = *kin_iter;
51          }
52        }
53        \\ se entrambi sono variati significa che la sequenza \'e finita e posso salvare i dati
54        else
55        {
56          cout << "sequence reset" << endl;
57
58          // salvo la coppia salvata
59          transfer_data_to_dataset(&a_tmp, &k_tmp);
60
61          // stampo i tempi per le valutazioni
62          cout << "best " << abs((long long)(a_tmp.frame_time - k_tmp.frame_time)) << "  kin: " <<
                ↪  k_tmp.contatore << " ard: " << a_tmp.contatore << endl;
63          cout << "time since another good pair: " << ((a_tmp.frame_time + k_tmp.frame_time) / 2 -
                ↪  time_old_good_pair) << endl;
64          time_old_good_pair = (a_tmp.frame_time + k_tmp.frame_time) / 2;
65
66          // salvo i dati correnti e i contatori correnti
67          old_kin_cont = kin_cont;
68          old_ard_cont = ard_cont;
69          a_tmp = *ard_iter;
70          k_tmp = *kin_iter;
71        }
72
73        cout << endl;
74
75      }
76      else
77      {
78        cout << "dato scartato" << endl << endl;
79      }
80
81      // incremento chi tra i 2 dati \'e il pi\'u vecchio
82      if (ard_iter->frame_time >= kin_iter->frame_time)
83      {
84        kin_iter++;
85        kin_cont++;
86      }
87      else
88      {
89        ard_iter++;
90        ard_cont++;
91      }
92
93      // se sono arrivato alla fine dei dati esco
94      if ((kin_iter == k_dat.end()) || (ard_iter == a_dat.end())) { break; }
95    }
96  }
```

Listing 3: funzione sync_without_clones

La funzione che si occupa di acquisire i nomi dei file e altro dall'utente è "post_elaborate":

```
void post_elaborate()
{
  string ard_file_mane = "";
  string kin_file_mane = "";
  string dataset_file_name = "";
  string tmp = "";
  bool clones = false;

  // acquisisco i nomi dei file dei dati di arduino e del kinect
  cout << "insert the input file names: " << endl
    << "\t arduino -> ";
  std::getline(std::cin, ard_file_mane);
  cout << "\t kinekt -> ";
  std::getline(std::cin, kin_file_mane);

  // acquisisco il nome del file in cui verr\'a scritto il dataset
  cout << "insert the ouptut file name -> : ";
  std::getline(std::cin, dataset_file_name);

  // chiedo se si vuole il dataset con o senza cloni
  cout << "do you want clones in the data? [y/n] -> : ";
  std::getline(std::cin, tmp);

  if (tmp == "y")
  {
    clones = true;
  }

  // chiamo la classe make_dataset passandogli i dati appena acquisiti
  make_dataset(ard_file_mane, kin_file_mane, dataset_file_name, clones);
}
```

Listing 4: funzione "post_elaborate"