Gestione automatica dei parametri della telecamera degli Aibo

Autore: Gilberto Decaro.

Laboratory of Applied Intelligent Systems (AIS-Lab)
http://ais-lab.dsi.unimi.it

10th november 2005

Indice

1 Introduzione

1.1 Specifiche della Telecamera

La telecamera si trova sul muso degli Aibo tra la bocca e il sensore di distanza a raggi infrarossi. La telecamera è basata su un sensore *CMOS* e offre, per ogni frame (raggiunge i 30 Frame al secondo), quattro layer di immagini:

1. **bassa** risoluzione: 52x40
2. **media** risoluzione: 104x80
3. **alta** risoluzione: 208x160
4. **Color** D**etection**: **layer** per il riconoscimento dei colori; è possibile impostare il riconoscimento di 8 colori.

Ogni immagine viene restituita in formato **YCrCb** e risulta compressa con una tecnica chiamata *"Trasformata di Haar"*, è possibile utilizzare l'immagine direttamente senza effettuare trasformazioni, ma decomprimendola è possibile aumentare la risoluzione quindi la precisione dei propri algoritmi (l'immagine ad alta risoluzione se decompressa attraverso la trasformata di haar arriva a 416x320 px).

Di seguito alcuni dati tecnici della telecamera:

- Il **diaframma** ha un'apertura fissa di F/2.8
- **L'angolo visuale** orizzontale è di 56.9 deg mentre l'angolo di visione verticale è di 45.2 deg
- **Bilanciamento del Bianco**: il bilanciamento del bianco serve a rendere i colori dell'immagine il più possibile vicini ai colori reali operando sulla temperatura del colore e cercando di impostare correttamente il valore per il bianco. E' possibile impostare il bilanciamento del bianco con i seguenti valori:
  - **Outdoor**: temperatura del colore 6500 K
  - **Fluorescent** Light: temperatura del colore 5000 K
  - **Indoor**: temperatura del colore 2856 K
- Velocità **dell'otturatore** (**shutter**): opera sulla velocità di apertura dell'otturatore, come nelle macchine fotografiche velocità elevate permettono di avere immagini non mosse, ma rimanendo aperto per poco tempo l'otturatore, viene ridotta anche la quantità di luce che colpisce il sensore, rendendo quindi l'immagine più scura; una velocità bassa, viceversa, garantisce immagini più luminose, ma anche più mosse. Lo shutter puo' essere impostato come:
  - **Slow**: 1/50 secondo

- **Mid**: 1/100 secondo
- **Fast**: 1/200 secondo

- **Guadagno** (**gain**): imposta il guadagno sull'immagine, sostanzialmente “amplifica” l'immagine, rendendo luminose immagini altrimenti scure; purtroppo l'amplificazione amplifica anche il rumore dell'immagine. Il gain puo' assumere i valori:
  - **Low**: -6 db
  - **Mid**: 0 db
  - **High**: +6 db

1.2 Gestione dei parametri

Open-R offre la funzione ControlPrimitive, definita in OPENRAPI.h, per la gestione dei parametri delle varie primitive;

```
OStatus ControlPrimitive(OPrimitiveID primitiveID,
                         OPrimitiveRequest request,
                         void* param, size_t paramSize,
                         void* result, size_t resultSize);
```

I vari parametri dipendono dal tipo di primitiva da gestire, per quanto riguarda la telecamera sono:

- Bilanciamento del bianco:

```
 OPrimitiveControl_CameraParam wb(newWB);
 OPENR::ControlPrimitive (fbkID,oprmreqCAM_SET_WHITE_BALANCE, &wb,
sizeof(wb),0,0);
```

  - Dove newWB può assumere i valori: **ocamparamWB_FL_MODE**, **ocamparamWB_INDOOR_MODE**, **ocamparamWB_OUTDOOR_MODE**

- Velocità dell'otturatore:

```
 OPrimitiveControl_CameraParam shutter(newShutter);
 OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter ) , 0 , 0);
```

  - Dove newShutter può assumere i valori: **ocamparamSHUTTER_FAST**, **ocamparamSHUTTER_MID**, **ocamparamSHUTTER_SLOW**

- Guadagno:

```
 OPrimitiveControl_CameraParam gain(newGain);
 OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_GAIN, &gain ,
sizeof(gain), 0,0);
```

  - Dove newGain può assumere i valori: **ocamparamGAIN_LOW**, **ocamparamGAIN_MID,**

ocamparamGAIN_HIGH.

2 Bilanciamenti Automatici

Durante lo sviluppo di **VisionAdjuster**, un oggetto Open-R per la gestione automatica dei parametri della telecamera, analizzando alcuni header di Open-R sono stati scoperti alcuni comandi non documentati riguardandi la gestione dei parametri della telecamera; tali comandi non erano documentati ne nella documentazione ufficiale della Sony ne nei siti Internet visitati.

2.1 Codice per il bilanciamento automatico

Nel header **OPrimitiveControl.h**, in /usr/local/OPEN_R_SDK/OPEN_R/include/OPENR, sono definite le costanti per la gestione dei parametri di tutte le primitive presenti (o non presenti) nell'Aibo modello ERS-7 (sono presenti, per esempio, due "oscure" costanti *oprmreqBT_ATTACH* e *oprmreqBT_DETACH* dove BT sta per BlueTooth). In questo file sono definite le costanti: ocamparamWB_OUTDOOR_MODE, ocamparamGAIN_LOW, ocamparamSHUTTER_MID... e sono definite anche altre quattro costati:

```
● const OPrimitiveRequest oprmreqCAM_AE_ON = oprmreqCAM_BASE + 4;
● const OPrimitiveRequest oprmreqCAM_AE_OFF = oprmreqCAM_BASE + 5;
● const OPrimitiveRequest oprmreqCAM_AWB_ON = oprmreqCAM_BASE + 6;
● const OPrimitiveRequest oprmreqCAM_AWB_OFF = oprmreqCAM_BASE + 7;
```

Dove **AE** sta per **Automatic Exposition** mentre **AWB** sta per **Automatic White Balance**. Dalle prove effettuate risultano funzionanti, con buoni risultati soprattutto per quanto riguarda l'Esposizione; il bilanciamento del bianco automatico da risultati sufficienti, purtroppo soffre in parte del problema del *"alone blu"* attorno all'immagine, problema riscontrato da molti altri gruppi attualmente al lavoro sugli Aibo, problema difficilmente risolvibile anche utilizzando le impostazioni manuali ocamparamWB_*. Un vantaggio rispetto alle impostazioni manuali è la maggiore granularità nel controllo; manualmente sono disponibili solo 3 possibili valori per l'otturatore, per il gain e per il bilanciamento del bianco, mentre in modalità automatica si notano variazioni delle impostazioni altrimenti non effettuabili con le impostazioni manuali.

Grazie a queste nuove constati è possibile gestire i parametri della telecamera in modo automatico semplicemente aprendo la primitiva della telecamera e chiamando la funzione **autoAdjustCamera()**, possibilmente all'interno del metodo DoInit; di seguito il codice da utilizzare:

```
/* Primitiva della telecamera.*/
static const char* const FBK_LOCATOR = "PRM:/r1/c1/c2/c3/i1-
FbkImageSensor:F1"

OStatus OpenRObject::DoInit(const OSystemEvent& event){
  ...
  ...
```

```
  // Id della telecamera
  OPrimitiveID fbkID;

 // Apro la primitiva della telecamera
 OStatus result = OPENR::OpenPrimitive(FBK_LOCATOR, &fbkID);
  if (result != oSUCCESS)
    {
      // Se errore
      OSYSLOG1((osyslogERROR,"Error opening Camera"));
    }
  else
    {
       // Se successo allora attivo l'adjust automatico
       setCameraParamter(fbkID);
    }

   ...
   ...
}
```

Di seguito la funzione **autoAdjustCamera()**:

```
/** Attiva l'auto bilanciamento dei parametri della telecamera. @param
    fbkID OPrimitiveID della telecamera.*/
void autoAdjustCamera(OPrimitiveID& fbkID){
  OPrimitiveControl_CameraParam wb(ocamparamWB_FL_MODE);
  OPENR::ControlPrimitive (fbkID,oprmreqCAM_SET_WHITE_BALANCE, &wb,
sizeof(wb),0,0);

  OPrimitiveControl_CameraParam shutter(ocamparamSHUTTER_MID);
  OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter ) , 0 , 0);

  OPrimitiveControl_CameraParam gain(ocamparamGAIN_MID);
  OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_GAIN, &gain ,
sizeof(gain), 0,0);

  OPENR::ControlPrimitive (fbkID, oprmreqCAM_AWB_ON, 0, 0, 0, 0);

  OPENR::ControlPrimitive (fbkID, oprmreqCAM_AE_ON, 0, 0, 0, 0);
}
```

La funzione inizialmente imposta i parametri sui valori di default (operazione fatta per scrupolo) quindi attiva (le ultime due righe) il bilanciamento automatico del bianco e la gestione automatica

dell'esposizione.

2.2 Codice di VisionAdjuster

Come anticipato all'inizio del capitolo la scoperta della gestione automatica del bilanciamento del bianco e dell'esposizione è venuta dopo lo sviluppo di un oggetto Open-R in grado di effettuare tali impostazioni analizzando alcune caratteristiche dell'immagine ritornata dalla telecamera.

L'oggetto Open-R si chiama **VisionAdjuster**, raggiunge buoni risultati anche se è necessario perfezionare maggiormente alcuni parametri riguardo il grado di luminosità dell'immagine e la predominanza del colore blu (i Threshold definiti nell'header).

VisionAdjuster analizza tre immagini successive ogni FRAME_TO_SKIP frame. La singola immagine (di default il layer a *media risoluzione*) viene suddivisa in NUM_OF_CELLS celle (di default in 16 celle), per ognuna di tali celle vengono analizzati PIXEL_TO_ANALYZE pixel scelti a caso all'interno della cella e sulla base dei loro valori viene calcolata la media dei canali Y, Cr e Cb di tali pixel. Una volta calcolate le NUM_OF_CELLS * 3 medie vengono calcolate le mediane per ogni singolo canale Y, Cr, Cb; quindi per ogni immagine viene restituita le mediane di Y, Cr e Cb. Questo calcolo viene fatto per tre frame successivo, questo per evitare che un ostacolo improvvisamente di fronte all'Aibo, oppure il passaggio di un altro cane o comunque improvvisi cambi di luminosità influenzino l'algoritmo. Per ogni canale viene fatta la media delle 3 mediane calcolate nei tre frame, avendo come risultato 3 valori uno per Y uno per Cr uno per Cp, questi tre valori vengono confrontati con alcuni threshold e in base al confronto vengono impostati i parametri.

VisionAdjuster offre il servizio **VisionAdjuster.Control.int.O** per attivare o disattivare l'adjusting dei parametri: se gli viene spedito in int 0 VisionAdjuster disattiva l'adjusting, con un valore positivo lo attiva.

Di seguito il codice sorgente:

```
[VisualAdjuster.h]

#ifndef VISIONADJUSTER_H_DEFINED
#define VISIONADJUSTER_H_DEFINED
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include <OPENR/RCRegion.h>
#include <OPENR/OFbkImage.h>
#include "def.h"
#include <string>

static const char* const FBK_LOCATOR = "PRM:/r1/c1/c2/c3/i1-
FbkImageSensor:F1";
```

```
class VisionAdjuster : public OObject{
public:
  VisionAdjuster();
  virtual ~VisionAdjuster();

  OSubject*    subject[numOfSubject];
  OObserver*   observer[numOfObserver];

  virtual OStatus DoInit   (const OSystemEvent& event);
  virtual OStatus DoStart  (const OSystemEvent& event);
  virtual OStatus DoStop   (const OSystemEvent& event);
  virtual OStatus DoDestroy(const OSystemEvent& event);

  /** Observer Avviato al ricevimento dell'immagine dalla
      telecamera.*/
  void NotifyImage(const ONotifyEvent& event);
  /** Observer di Controllo, permette di attivare o disattivare
      l'adjust dei parametri. Se riceve 0 allora VisionAdjuster viene
      disattivato e non effettua l'adjust dei parametri, con un valore
      positivo viene riattivato.*/
  void NotifyControl(const ONotifyEvent& event);

private:
  /** Struttura utilizzata da compute Mean per ritornare le medie dei
      tre canali*/
  struct meanValue
  {
    double y;
    double cr;
    double cb;
  };

  /** Numero di frame consecutivi da analizzare per la taratura.*/
  static const size_t SEQUENTIAL_FRAME=3;
  /** Indice corrente dell'array means*/
  size_t meanIndex;
  meanValue means[SEQUENTIAL_FRAME];

  /* Numero di celle in cui dividere l'immagine. L'immagine viene
     suddivisa nel numero specificato di celle, per ogniuna di esse
     viene calocata la media, quindi viene utilizzata la mediata tra
     le medie calcolate per valutare la luminosita' e il bianciamento
     del bianco dell'immagine. I possibili valori rientrano nella
```

```
     serie 1,4,16,64 (un valore accettabile di precisione e' 16);*/
  static const size_t NUM_OF_CELLS=16;
  /* Numero di celle sull'asse x dell'immagine*/
  int numOfXCell;
  /* Numero di celle sull'asse y dell'immagine*/
  int numOfYCell;
  /* Width della singola cella.*/
  int cellWidth;
  /* Height della singola cella.*/
  int cellHeight;
  /* Numero di Frame da scartare (30 Fps * 5) = 150: Analizzo
      un'immagine ogni sec.*/
  static const int FRAME_TO_SKIP=150;
  /* Layer utilizzato per l'elaborazione dell'immagine.*/
  static const int USED_LAYER=ofbkimageLAYER_M;
  /* Width dell'imagine al layer specificato.*/
  static const size_t LAYER_WIDTH=104;
  /* Height dell'immagine al layer specificato.*/
  static const size_t LAYER_HEIGHT=80;
  /* Numero massimo di tentativi consecutivi di Adjusting dei
     parametri.*/
  static const int MAX_READJUST=5;
  /* Numero di Pixel da analizzare nel singolo quadrante per valutare
     i parametri da modificare.*/
  static const size_t PIXEL_TO_ANALYZE=60;

  /* Stati di VisionAdjuster.*/
  enum
    {
      DISABLED, /**< Disabilitato, non effettuare aggiustamenti*/
      SKIPPING, /**< Skip di frame*/
      PREADJUST, /**< PreAdjust, serve per scartare un frame prima di
                                  fare l'adjust, per compensare i tempi
di
                                  aggiornamento dei parametri.*/
      ADJUST, /**< Adjust dei parametri*/
      CHECKADJUST, /**< Controllo dell'adjust.*/
    } state;

  /* Velocita' dello shutter precedente*/
  unsigned int previousShutter;
  /* Gain precedente*/
  unsigned int previousGain;
  /* Bilanciamento del bianco precedente.*/
  unsigned int previousWB;
```

```
 /* Frame attualmente scartati.*/
 int frameCount;
 /* Numero di adjusting consecutivi effettuati*/
 int adjustingCount;

 /* Id della camera una volta aperta.*/
 OPrimitiveID fbkID;

 /* Limite di luminosita', sotto tale valore l'immagine e'
    considerata scura. Luminosita' varia (dai dati raccolti) da 20 a
    222.*/
 static const double DARK_THRESHOLD=80.0;
 /* Limite di luminosita', sopra tale valore l'immagine e'
    considerata troppo chiara.*/
 static const double BRIGHT_THRESHOLD=125.0;

 /* Limite della predominanza del Rosso.*/
 static const double RED_THRESHOLD=170.0;
 /* Limite della predominanza del Blu.*/
 static const double BLUE_THRESHOLD=145.0;


private:
 /* Effettua l'adjust dei parametri della camera. @return true se ha
    effettuato un adjust dei parametri, false se non ha potuto
    effettuare l'adjust (raggiunti i limiti della telecamera).*/
 bool adjustParameter();
 /* Controlla la correttenzza del'adjust. Ritorna true se immagine
    OK, false se bisogna rifare un adjust.*/
 bool checkAdjust();

 /* Divide l'immagine in 16 blocchi di 26x20 px e per ogniuno di
    essi calcola le medie dei canali y,cr,cb considerando
    PIXEL_TO_ANALYZE pixel scelti a caso. Quindi calcola la media di
    y,cr,cb tra le 16 medie calcolate per ogni canale. Salva le tre
    mediane nel valore di ritorno meanValue.*/
 meanValue computeMean(OFbkImageVectorData* imageVec);

 /* Ritorna true se a e > di b. Serve per effettuare il sort sul
    vettore.*/
 static bool greater(double a, double b);
 /* Ritorna la stringa rappresentatante il parametro.*/
 std::string getWBParam(unsigned int wb);
 /* Ritorna la stringa rappresentatante il parametro.*/
```

```
  std::string getGainParam(unsigned int gain);
  /* Ritorna la stringa rappresentatante il parametro.*/
  std::string getShutterParam(unsigned int shutter);

  /* Effettua l'adjust della luminosita' lavorando sul gain e sullo
     shutter. @return true se ha effettuato un adjust dei parametri,
     false se non ha potuto effettuare l'adjust (raggiunti i limiti
     della telecamera).*/
  bool adjustBright(meanValue& mean);

  /* Effettua l'adjust del Bilanciamento del Bianco lavorando sul
     parametro WhiteBalance della telecamera. @return true se ha
     effettuato un adjust dei parametri, false se non ha potuto
     effettuare l'adjust (raggiunti i limiti della telecamera).*/
  bool adjustWB(meanValue& mean);

};


#endif
```

Il sorgente:

```
[VisionAdjuster.cc]

#include "VisionAdjuster.h"
#include <OPENR/OSyslog.h>
#include <OPENR/core_macro.h>
#include <OPENR/OPENRAPI.h>
#include <ant.h>
#include "entry.h"
#include <SystemTime.h>
#include <OPENR/OFbkImage.h>
#include <stdlib.h>
#include <time.h>
#include <string>
#include <vector>
#include <SystemTime.h>
#include <math.h>

using namespace std;

VisionAdjuster::VisionAdjuster()
  : meanIndex(0),
```

```
    state(ADJUST),
    frameCount(0),
    adjustingCount(0),
    fbkID(oprimitiveID_UNDEF)
{
  srand(clock());

  //
  // Valori di default alla partenza dell'Aibo
  //
  previousShutter=ocamparamSHUTTER_FAST;
  previousGain=ocamparamGAIN_MID;
  previousWB=ocamparamWB_FL_MODE;

  numOfXCell=static_cast<int>(sqrt(static_cast<double>(NUM_OF_CELLS)));
  numOfYCell=numOfXCell;

  cellWidth=LAYER_WIDTH/numOfXCell;
  cellHeight=LAYER_HEIGHT/numOfYCell;
}

VisionAdjuster::~VisionAdjuster(){
}


OStatus VisionAdjuster::DoInit(const OSystemEvent& event){
  OSYSDEBUG(("VisionAdjuster::DoInit()\n"));

  NEW_ALL_SUBJECT_AND_OBSERVER;
  REGISTER_ALL_ENTRY;
  SET_ALL_READY_AND_NOTIFY_ENTRY;

  // Apro la telecamera
  OStatus result = OPENR::OpenPrimitive(FBK_LOCATOR, &fbkID);

  if (result != oSUCCESS)
    {
      OSYSLOG1((osyslogERROR,"VisionAdjuster: Error opening Camera"));
    }
  else
    {
      //
      // Reimposto i parametri di default.
      //
```

```
      OPrimitiveControl_CameraParam wb(previousWB);
      OPENR::ControlPrimitive (fbkID,oprmreqCAM_SET_WHITE_BALANCE, &wb,
sizeof(wb),0,0);

      OPrimitiveControl_CameraParam shutter(previousShutter);
      OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter ) , 0 , 0);

      OPrimitiveControl_CameraParam gain(previousGain);
      OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_GAIN, &gain ,
sizeof(gain), 0,0);

//       //
//       // Credo che AWB stia per Auto White Balance: sembra non
funzinoare.
//       //
//       OPENR::ControlPrimitive (fbkID, oprmreqCAM_AWB_ON, 0, 0, 0,
0);


//        //
//        // Credo che AE stia per Auto Exposition: Funziona !
//        //
//        OPENR::ControlPrimitive (fbkID, oprmreqCAM_AE_ON, 0, 0, 0,
0);



    }

  return result;
}

/** Attiva l'auto bilanciamento dei parametri della telecamera. @param
    fbkID OPrimitiveID della telecamera.*/
void autoAdjustCamera(OPrimitiveID& fbkID){
  OPrimitiveControl_CameraParam wb(previousWB);
  OPENR::ControlPrimitive (fbkID,oprmreqCAM_SET_WHITE_BALANCE, &wb,
sizeof(wb),0,0);

  OPrimitiveControl_CameraParam shutter(previousShutter);
  OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter ) , 0 , 0);
```

```
  OPrimitiveControl_CameraParam gain(previousGain);
  OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_GAIN, &gain ,
sizeof(gain), 0,0);

  //
  // Credo che AWB stia per Auto White Balance: sembra non funzinoare.
  //
  OPENR::ControlPrimitive (fbkID, oprmreqCAM_AWB_ON, 0, 0, 0, 0);


  //
  // Credo che AE stia per Auto Exposition: Funziona !
  //
  OPENR::ControlPrimitive (fbkID, oprmreqCAM_AE_ON, 0, 0, 0, 0);
}


/*DoStart: fa partire Telepati*/
OStatus VisionAdjuster::DoStart(const OSystemEvent& event){
  OSYSDEBUG(("VisionAdjuster::DoStart()\n"));

  ENABLE_ALL_SUBJECT;
  ASSERT_READY_TO_ALL_OBSERVER;

  OStatus st = OPENR::SetMotorPower(opowerON);

  return st;
}  //DoStart() END


/*DoStop: arresta Telepati*/
OStatus VisionAdjuster::DoStop(const OSystemEvent& event){

  OSYSDEBUG(("VisionAdjuster::DoStop()\n"));

  DISABLE_ALL_SUBJECT;
  DEASSERT_READY_TO_ALL_OBSERVER;

  OStatus st = OPENR::SetMotorPower(opowerOFF);
  return st;
}  //DoStop() END


/*DoDestroy: libera la memoria occupata da Telepati*/
```

```
OStatus VisionAdjuster::DoDestroy(const OSystemEvent& event){
  OSYSDEBUG(("VisionAdjuster::DoDestroy\()\n"));

  DELETE_ALL_SUBJECT_AND_OBSERVER;

  return oSUCCESS;
}  //DoDestroy() END

void VisionAdjuster::NotifyControl(const ONotifyEvent& event){
  int controlValue=*((int*)event.Data(0));
  if(controlValue <= 0)
    state=DISABLED;
  else
    {
      frameCount=0;
      state=ADJUST;
    }
}

void VisionAdjuster::NotifyImage(const ONotifyEvent& event){
  switch(state)
    {
    case DISABLED:
      {}   // do nothing
      break;

    case SKIPPING:
      {
                              if(frameCount > FRAME_TO_SKIP)
                                {
                                  state=ADJUST;
                                  frameCount=0;
                                  adjustingCount=0;
                                  meanIndex=0;
                                }
                              else
                                frameCount++;
      }
      break;

    case ADJUST:
      {
                              // Faccio l'adjust
                              // Quindi imposto state in CHECKADJUST
```

```
                              if(meanIndex < SEQUENTIAL_FRAME)

means[meanIndex++]=computeMean((OFbkImageVectorData*)event.Data(0));
                              else
                                {
                                  OSYSDEBUG(("ADJUST\n"));
                                  meanIndex=0;
                                  if(adjustParameter())
                                    state=CHECKADJUST;
                                  else
                                    state=SKIPPING;
                                }
      }
      break;
    case CHECKADJUST:
      {
                              // Se ho superato il massimo numero di
adjust consecutivi
                              if(adjustingCount > MAX_READJUST)
                                {
                                  // Torno in skipp
                                  state=SKIPPING;
                                }
                              else
                                {
                                  if(meanIndex < SEQUENTIAL_FRAME)

means[meanIndex++]=computeMean((OFbkImageVectorData*)event.Data(0));
                                  else
                                    {
                               OSYSDEBUG(("CHECKADJUST\n"));
                               // altrimenti effettuo il check.
                               adjustingCount++;
                               meanIndex=0;
                               if(checkAdjust())
                                 {
                                   // Se l'immagine e' OK torno a Skip
                                   state=SKIPPING;
                                   OSYSDEBUG(("Check OK\n"));
                                 }
                               else
                                 {
                                   // Altrimenti vado a readjust.
                                   state=PREADJUST;
```

```
                                    OSYSDEBUG(("Check  FAIL\n"));
                                  }

                                     }
                                  }
      }
      break;
    case PREADJUST:
      {
                              OSYSDEBUG(("PREADJUST\n"));
                              state=ADJUST;
      }
      break;
    }

  observer[event.ObsIndex()]->AssertReady(event.SenderID());
  return;
}

bool VisionAdjuster::greater(double a, double b){
  return a>b;
}

VisionAdjuster::meanValue
VisionAdjuster::computeMean(OFbkImageVectorData* imageVec){
#ifdef OPENR_DEBUG
  SystemTime startTime;
  GetSystemTime(&startTime);
#endif

  vector<double> yMean;
  vector<double> crMean;
  vector<double> cbMean;

  meanValue retVal;

  OFbkImageInfo* info = imageVec->GetInfo(ofbkimageLAYER_M);
  byte*          data = imageVec->GetData(ofbkimageLAYER_M);

  OFbkImage yImage(info, data, ofbkimageBAND_Y);
  OFbkImage crImage(info, data, ofbkimageBAND_Cr);
  OFbkImage cbImage(info, data, ofbkimageBAND_Cb);

  //
```

```
  // Calcolo le medie dei vari quadranti
  //
  if(yImage.IsValid() && crImage.IsValid() && cbImage.IsValid())
    {
      double cbTempMean;
      double crTempMean;
      double yTempMean;

      int x,y;
      int xMin=0;
      int yMin=0;

      for(int i=0; i< numOfXCell; i++)
                              for(int j=0; j<numOfYCell; j++)
                                {
                                  xMin=(i*cellWidth);
                                  yMin=(j*cellHeight);

                                  yTempMean=0;
                                  cbTempMean=0;
                                  crTempMean=0;
                                  for(size_t z=0; z < PIXEL_TO_ANALYZE;
z++)
                                    {
                                 x=xMin+(rand() % cellWidth); //un numero
da 0 a 25
                                 y=yMin+(rand() % cellHeight); //un numero
da 0 a 19

                                 yTempMean+=yImage.Pixel(x,y);
                                 cbTempMean+=(cbImage.Pixel(x,y));
                                 crTempMean+=(crImage.Pixel(x,y));
                                      }

yMean.push_back(yTempMean/PIXEL_TO_ANALYZE);

crMean.push_back(crTempMean/PIXEL_TO_ANALYZE);

cbMean.push_back(cbTempMean/PIXEL_TO_ANALYZE);
                                 }
      //
      // Tiro fuori la mediana tra le medie dei vari quadranti
      //
      sort(yMean.begin(), yMean.end(), VisionAdjuster::greater);
```

```
      sort(crMean.begin(), crMean.end(), VisionAdjuster::greater);
      sort(cbMean.begin(), cbMean.end(), VisionAdjuster::greater);

      int sxIndex=int(((double)NUM_OF_CELLS/2)+0.5);
      int dxIndex=int(((double)NUM_OF_CELLS/2)-0.5);
      retVal.y=(yMean[sxIndex]+yMean[dxIndex])/2;
      retVal.cr=(crMean[sxIndex]+crMean[dxIndex])/2;
      retVal.cb=(cbMean[sxIndex]+cbMean[dxIndex])/2;
      OSYSDEBUG(("CALCOLATO y= %f  cr= %f  cb=%f\n",
retVal.y,retVal.cr,retVal.cb));
    }
  else
    {
      retVal.y=-1;
      retVal.cr=-1;
      retVal.cb=-1;
    }

#ifdef OPENR_DEBUG
  SystemTime endTime;
  GetSystemTime(&endTime);
  SystemTime elapsed= (endTime - startTime);  //tempo di consegna
  OSYSDEBUG(("TEMPO ESECUZIONE calculateMean: %u sec %u microsec\n",
elapsed.seconds, elapsed.useconds));
#endif

  return retVal;
}


bool VisionAdjuster::adjustBright(meanValue& mean){
  //
  // Gestisco la luminosita'
  //
  bool retValue;
  unsigned int newShutter;
  unsigned int newGain;
  if(mean.y < DARK_THRESHOLD)
    {
      OSYSDEBUG(("Immagine troppo scura\n"));
      // Se troppo scura

      if(previousGain == ocamparamGAIN_HIGH)
                              {
```

```
                                newGain=ocamparamGAIN_HIGH;
                                if(previousShutter ==
ocamparamSHUTTER_FAST)
                                  newShutter=ocamparamSHUTTER_MID;
                                else
                                  newShutter=ocamparamSHUTTER_MID;
                              }
      else if(previousGain == ocamparamGAIN_MID)
                              {
                                newGain = ocamparamGAIN_HIGH;
                                newShutter=ocamparamSHUTTER_FAST;
                              }
      else if(previousGain == ocamparamGAIN_LOW)
                              {
                                newGain = ocamparamGAIN_MID;
                                newShutter=ocamparamSHUTTER_FAST;
                              }

      OSYSDEBUG(("Shutter: %s\nGain: %s\n",
getShutterParam(newShutter).c_str(), getGainParam(newGain).c_str()));

      retValue=(previousShutter != newShutter) || (previousGain !=
newGain);

      previousShutter=newShutter;
      OPrimitiveControl_CameraParam shutter(newShutter);
      OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter ) , 0 , 0);

      previousGain=newGain;
      OPrimitiveControl_CameraParam gain(newGain);
      OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_GAIN, &gain ,
sizeof(gain), 0,0);

    }
  else if(mean.y > BRIGHT_THRESHOLD)
    {
      OSYSDEBUG(("Immagine troppo chiara\n"));
      // Se troppo chiara
      if(previousGain == ocamparamGAIN_HIGH)
                              {
                                newShutter=ocamparamSHUTTER_FAST;
                                newGain=ocamparamGAIN_MID;
                              }
```

```
      else if(previousGain == ocamparamGAIN_MID)
                              {
                                newShutter=ocamparamSHUTTER_FAST;
                                newGain=ocamparamGAIN_LOW;
                              }
      else if(previousGain == ocamparamGAIN_LOW)
                              {
                                newShutter=ocamparamSHUTTER_FAST;
                                newGain=ocamparamGAIN_LOW;
                              }

      OSYSDEBUG(("Shutter: %s\nGain: %s\n",
getShutterParam(newShutter).c_str(), getGainParam(newGain).c_str()));

      retValue=(previousShutter != newShutter) || (previousGain !=
newGain);

      previousShutter=newShutter;
      OPrimitiveControl_CameraParam shutter(newShutter);
      OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_SHUTTER_SPEED,
&shutter, sizeof(shutter ) , 0 , 0);

      previousGain=newGain;
      OPrimitiveControl_CameraParam gain(newGain);
      OPENR::ControlPrimitive(fbkID, oprmreqCAM_SET_GAIN, &gain ,
sizeof(gain), 0,0);
    }
  else
    {
      retValue=false;
      OSYSDEBUG(("Luminosita' OK; Shutter: %s\nGain: %s\n",
getShutterParam(previousShutter).c_str(),
getGainParam(previousGain).c_str()));
      ; // Non far nulla Luminosita' OK.
    }

  return retValue;
}

bool VisionAdjuster::adjustWB(meanValue& mean){
  //
  // Gestisco il bilanciamento del bianco
  //
  bool retValue;
```

```
  unsigned int newWB;
  if(mean.cb > BLUE_THRESHOLD)
    {
      OSYSDEBUG(("Immagine troppo blu\n"));
      // Se il colore blu predomina nell'immagine allora la telecamera
      // e' tarata su una temperatura del colore piu' bassa rispetto
      // alla realta'. Quindi devo alzare la temperatura del colore.
      //                        newWB=ocamparamWB_FL_MODE;
      // PERO': questi simpatici cani funzinoano al contrario ?!
      switch(previousWB)
                              {
                              case ocamparamWB_INDOOR_MODE: // 2856K
                                newWB=ocamparamWB_FL_MODE;
                                break;
                              case ocamparamWB_FL_MODE: // 5000K
                                newWB=ocamparamWB_OUTDOOR_MODE;
                                break;
                              case ocamparamWB_OUTDOOR_MODE: //6500K
                                newWB=ocamparamWB_OUTDOOR_MODE;
                                break;
                              }

      OSYSDEBUG(("White Balance: %s\n", getWBParam(newWB).c_str()));

      retValue=(previousWB != newWB);
      previousWB=newWB;
      OPrimitiveControl_CameraParam wb(newWB) ;
      OPENR::ControlPrimitive (fbkID,oprmreqCAM_SET_WHITE_BALANCE, &wb,
sizeof(wb),0,0);
    }
  else if(mean.cr > RED_THRESHOLD)
    {
      OSYSDEBUG(("Immagine troppo giallognola\n"));
      // Se il colore giallognolo predomina nell'immagine allora la
      // telecamera e' tarata su una temperatura del colore piu' alta
      // rispetto alla realta'. Quindi devo abbassare la temperatura
      // del colore della telecamera.
      // PERO': questi simpatici cani funzinoano al contrario ?!
      switch(previousWB)
                              {
                              case ocamparamWB_INDOOR_MODE: //2856K
                                newWB=ocamparamWB_INDOOR_MODE;
                                break;
                              case ocamparamWB_FL_MODE: // 5000k
```

```
                                newWB=ocamparamWB_INDOOR_MODE;
                                break;
                              case ocamparamWB_OUTDOOR_MODE: //6500K
                                newWB=ocamparamWB_FL_MODE;
                                break;
                              }

      OSYSDEBUG(("White Balance: %s\n", getWBParam(newWB).c_str()));

      retValue=(previousWB != newWB);

      previousWB=newWB;
      OPrimitiveControl_CameraParam wb(newWB);
      OPENR::ControlPrimitive (fbkID,oprmreqCAM_SET_WHITE_BALANCE, &wb,
sizeof(wb),0,0);
    }
  else
    {
      retValue=false;
      OSYSDEBUG(("White Balance OK; White Balance: %s\n",
getWBParam(previousWB).c_str()));
      ; // Nulla Bilanciamento del bianco OK
    }

  return retValue;
}

bool VisionAdjuster::adjustParameter(){

  meanValue mean={0,0,0};

  // Calcolo la media delle letture di y,cr,cb
  for(size_t i=0; i< SEQUENTIAL_FRAME; i++)
    {
      mean.y+=means[i].y;
      mean.cr+=means[i].cr;
      mean.cb+=means[i].cb;
    }
  mean.y/=SEQUENTIAL_FRAME;
  mean.cr/=SEQUENTIAL_FRAME;
  mean.cb/=SEQUENTIAL_FRAME;


  bool WBAdjusted=adjustWB(mean);
```

```
  bool brightAdjusted=adjustBright(mean);

  return (WBAdjusted || brightAdjusted);
}

bool VisionAdjuster::checkAdjust(){

  meanValue mean={0,0,0};

  // Calcolo la media delle letture di y,cr,cb
  for(size_t i=0; i< SEQUENTIAL_FRAME; i++)
    {
      mean.y+=means[i].y;
      mean.cr+=means[i].cr;
      mean.cb+=means[i].cb;
    }
  mean.y/=SEQUENTIAL_FRAME;
  mean.cr/=SEQUENTIAL_FRAME;
  mean.cb/=SEQUENTIAL_FRAME;

  // Luminosita'
  if(mean.y < DARK_THRESHOLD || mean.y > BRIGHT_THRESHOLD)
    return false;

  // Bilanciamento del bianco
  if(mean.cb > BLUE_THRESHOLD || mean.cr > RED_THRESHOLD)
    return false;


  return true;
}

string VisionAdjuster::getShutterParam(unsigned int shutter){
  string str;
  switch(shutter)
    {
    case ocamparamSHUTTER_FAST:
      str="FAST";
      break;
    case ocamparamSHUTTER_MID:
      str="MID";
      break;
    case ocamparamSHUTTER_SLOW:
      str="LOW";
```

```
      break;
    }
  return str;
}

string VisionAdjuster::getGainParam(unsigned int gain){
  string str;
  switch(gain)
    {
    case ocamparamGAIN_LOW:
      str="LOW";
      break;
    case ocamparamGAIN_MID:
      str="MID";
      break;
    case ocamparamGAIN_HIGH:
      str="HIGH";
      break;
    }
  return str;
}

string VisionAdjuster::getWBParam(unsigned int wb){
  string str;

  switch(wb)
    {
    case ocamparamWB_INDOOR_MODE:
      str="WB_INDOOR_MODE";
      break;
    case ocamparamWB_FL_MODE:
      str="WB_FL_MODE";
      break;
    case ocamparamWB_OUTDOOR_MODE:
      str="WB_OUTDOOR_MODE";
      break;
    }
  return str;
}
```

Di seguito il file stub.cfg:

```
[stub.cfg]
```

```
ObjectName: VisionAdjuster
NumOfOSubject: 1
NumOfOObserver: 2
Service: "VisionAdjuster.DummySbj.byte.S", null, null
Service: "VisionAdjuster.Image.OFbkImageVectorData.O", null,
NotifyImage()
Service: "VisionAdjuster.Control.int.O", null, NotifyControl()
```

Quindi il file ocf:

```
[VisionAdjuster.ocf]

object VisionAdjuster 3072 16384 128 cache tlb user
```

E infine il Makefile:

```
[Makefile]

OPENRSDK_ROOT?=/usr/local/OPEN_R_SDK
INSTALLDIR=../MS
CXX=$(OPENRSDK_ROOT)/bin/mipsel-linux-g++
STRIP=$(OPENRSDK_ROOT)/bin/mipsel-linux-strip
MKBIN=$(OPENRSDK_ROOT)/OPEN_R/bin/mkbin
STUBGEN=$(OPENRSDK_ROOT)/OPEN_R/bin/stubgen2
MKBINFLAGS=-p $(OPENRSDK_ROOT)
LIBS=-L$(OPENRSDK_ROOT)/OPEN_R/lib -lObjectComm -lOPENR  -lantMCOOP
CXXFLAGS= \
                              -O2 \
                              -I. \
                              -Wall \
                              -I$(OPENRSDK_ROOT)/OPEN_R/include/R4000 \
                              -I$(OPENRSDK_ROOT)/OPEN_R/include/MCOOP \
                              -I$(OPENRSDK_ROOT)/OPEN_R/include

#
# When OPENR_DEBUG is defined, OSYSDEBUG() is available.
#
CXXFLAGS+= -DOPENR_DEBUG
TARGET= VisionAdjuster.bin
TARGET_NAME=VisionAdjuster
TARGET_MS=VISADJ.BIN
.PHONY: all install clean
```

```
all: $(TARGET)

%.o: %.cc
                              $(CXX) $(CXXFLAGS) -o $@ -c $^

$(TARGET_NAME)Stub.cc: stub.cfg
                              $(STUBGEN) stub.cfg

$(TARGET): $(TARGET_NAME)Stub.o $(TARGET_NAME).o $(TARGET_NAME).ocf
                              $(MKBIN) $(MKBINFLAGS) -o $@ $^ $(LIBS)
                              $(STRIP) $@


install: $(TARGET)
                              gzip -c $(TARGET) > $(INSTALLDIR)/OPEN-
R/MW/OBJS/$(TARGET_MS)

clean:
                              rm -f *.o *.bin *.elf *.snap.cc
                              rm -f $(TARGET_NAME)Stub.h
$(TARGET_NAME)Stub.cc def.h entry.h
                              rm -f $(INSTALLDIR)/OPEN-
R/MW/OBJS/$(TARGET_MS)
```