Université de Technologie de Compiègne

SY31

Rapport de TP

6 - Projet libre

Automne 2014

Romain PELLERIN - Kyâne PICHOU - Nianfei SHI
Groupe de TP 2
9 janvier 2015

Table des matières

1 Introduction

Pour ce TP, dont le but est de faire un mini-projet en s'inspirant des travaux effectués lors des précédents TP, nous voulions initialement faire du Wifibot un robot suiveur. Nous voulions pouvoir mettre un objet très rouge devant la caméra du robot et faire en sorte que celui-ci suive cet objet selon la distance qui le séparait de l'objet, à l'aide du Lidar. Cela nous aurait permis de réutiliser les connaissances acquises lors de deux TP différents.

Après réflexion et après avoir mis nos idées sur papier, nous sommes partis dans l'idée de d'abord détecter l'objet sur la caméra, avant de nous occuper de la distance. Après avoir commencé à coder quelque chose en récupérant notre code du TP5, nous nous sommes heurté à un problème : **la structure d'image de sortie `MAPS::IplImage` n'était pas la même qu'en entrée**. Nous avons perdu plusieurs minutes à essayer d'obtenir en sortie une image en couleurs, avec les contours de l'objet rouge bien identifiés, mais sans succès. C'est pourquoi nous avons décidé de changer notre projet pour quelque chose de plus accessible.

Nous avons donc choisi de n'utiliser que la caméra et de réutiliser ce qui a été fait lors du TP5. Ainsi, nous allons développer un module RTMaps qui permet de :

— Afficher une image en noir et blanc classique (selon 256 niveaux de gris, de 0 à 255)

— Afficher une image en noir et blanc selon 5 niveaux de gris

— Afficher une image en noir et blanc classique et « en miroir » (inversée)

— Afficher une image en noir et blanc selon 5 niveaux de gris et « en miroir »

Finalement, lors de la troisième séance, nous sommes parvenus à afficher en sortie une image en couleurs. Ainsi, nous avons été à même d'afficher une image en couleurs founrie par la webcam ainsi qu'une image en couleur dont chaque pixel affiche uniquement la composante prédominante (soit rouge, soit bleu ou soit vert).

2 Développement lors de la première et de la deuxième séance

Intéressons-nous tout d'abord aux affichages en noir et blanc.

Pour l'ensemble des 4 affichages, nous avons choisi, pour évider la redondance de code, de ne faire qu'une seule fonction. Selon les paramètres qui lui sont passés, la fonction retourne en sortie l'un des 4 types d'images cités.

2.1 Noir et blanc classique selon 256 niveaux de gris

Pour afficher une image en noir et blanc selon 256 niveaux de gris, nous avons repris le code du TP5.

C++

```
// ...

int val = (src.imageData[3*i]*0.299) + (src.imageData[(3*i)+1]*0.587) + (src.
    imageData[(3*i)+2]*0.114);

// ...
```

2.2 Noir et blanc selon 5 niveaux de gris

Pour obtenir une image selon 5 niveaux de gris, nous sommes parti de la valeur `val` obtenue précédemment (qui correspond à une valeur de gris entre 0 et 255). Si cette valeur est dans une certaine « tranche », nous lui attribuons une valeur spécifique.

C++

```
// ...

if (val >=0 && val < 50)
  val = 25;
else if (val >= 50 && val < 100)
  val = 75;
else if (val >= 100 && val < 150)
  val = 125;
else if (val >= 150 && val < 200)
  val = 175;
else
  val = 225;

// ...
```

2.3 Noir et blanc classique en miroir

Pour cela, nous avons du créer un petit algorithme après avoir mis l'image en noir et blanc, pour inverser les pixels de l'image. Le premier pixel de l'image se retrouve être le dernier, le second devient l'avant-dernier, et ainsi de suite...

Voici donc la ligne de code qui inverse l'image :

C++

```
// ...

dst.imageData[((i/src.width)+1)*src.width-(i%src.width)] = val; // Affectation
    de la valeur noir et blanc du premier pixel de l'image source au dernier
    pixel de l'image de destination

// ...
```

2.4 Noir et blanc selon 5 niveaux de gris et en miroir

Pour obtenir une image en noir et blanc selon 5 niveaux de gris et en miroir, nous avons simplement combiné les autres morceaux de code ci-dessus.

2.5 Module RTMaps

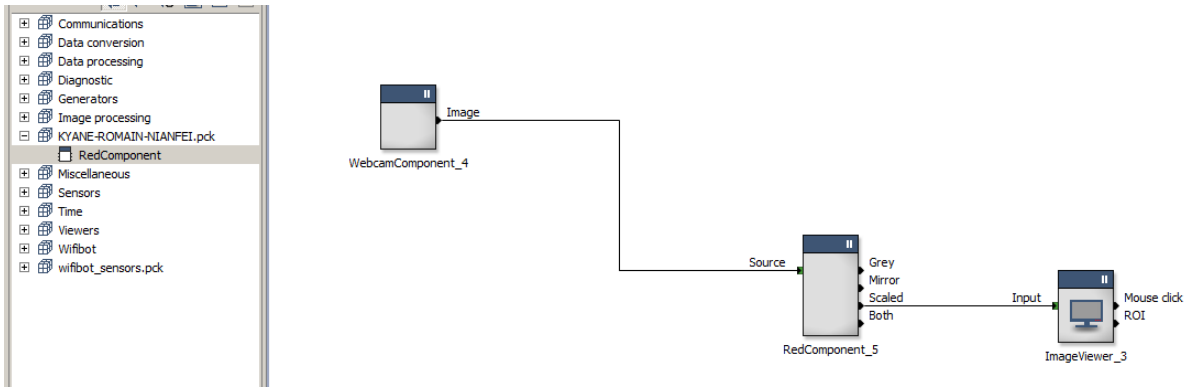

FIGURE 2.1 – Module RTMaps

2.6 Résultat visuel

Voici nos 4 affichages en noir et blanc, de gauche à droite et de haut en bas :

- Noir et blanc
- Miroir
- Noir et blanc selon 5 niveaux de gris
- Noir et blanc selon 5 niveaux de gris et miroir

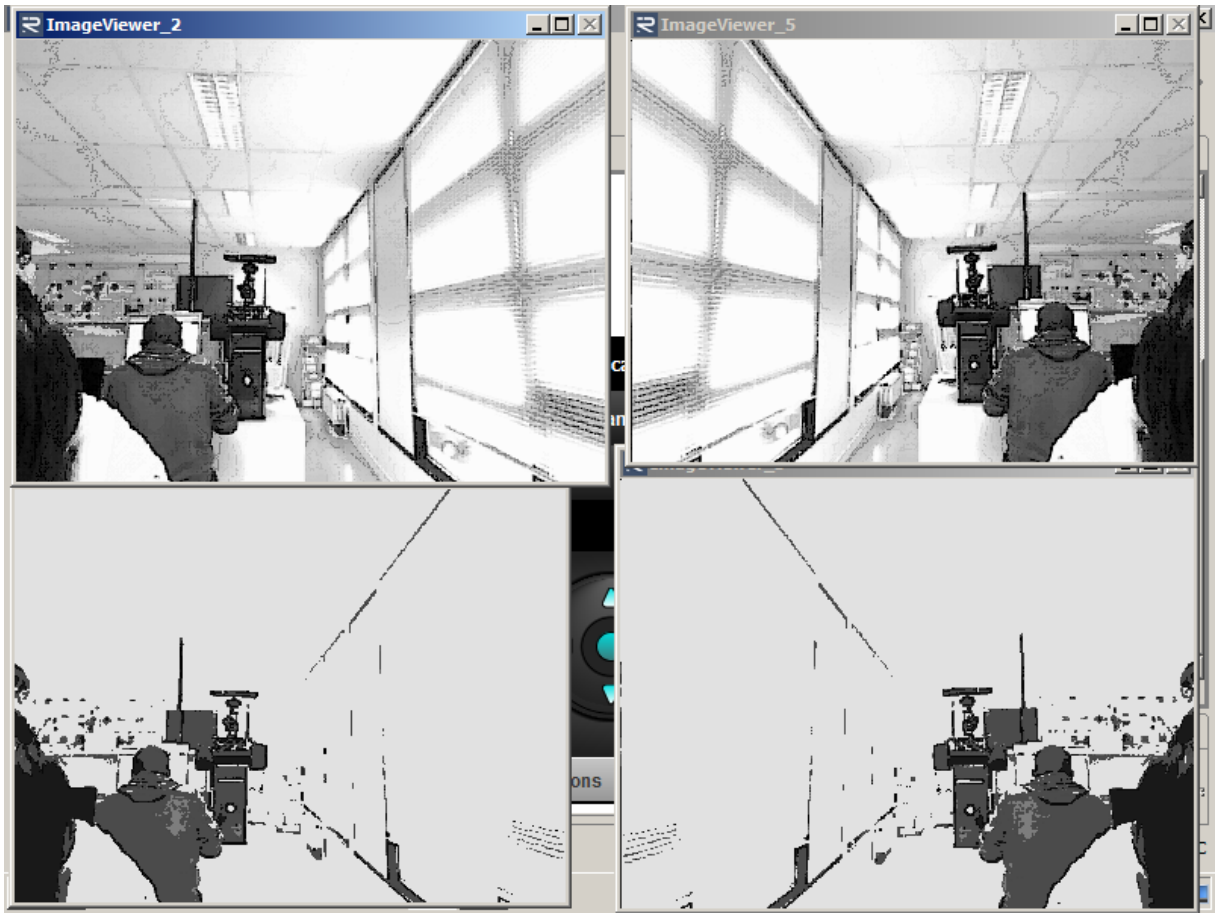

FIGURE 2.2 – Les 4 affichages en noir et blanc

2.7 Fichier .hpp

Voici donc le code final de notre module RTMaps.

C++

```
#ifndef DEF_RED_COMPONENT_HPP
#define DEF_RED_COMPONENT_HPP

// Includes, rtmaps
#include "maps.hpp"

namespace sy31
{
  class RedComponent
    : public MAPSComponent
  {
    MAPS_COMPONENT_STANDARD_HEADER_CODE(RedComponent)
    protected:
      /// Allocate the output images.
      void allocateOutputs(IplImage const& src);

      /// Notre travail
     void fonction(IplImage const& src,IplImage & dst, bool scaled, bool mirror)
```

```
        ;

    private:
      bool mIsAllocated;
  };
}

#endif
```

2.8 Fichier .cpp

C++

```
// Includes, project
#include "red-component.hpp"

// Includes, standar
#include <cmath>

using namespace sy31;

////////////////////////////////////////////////////////////////
// RTMaps - Input
////////////////////////////////////////////////////////////////
MAPS_BEGIN_INPUTS_DEFINITION(RedComponent)
  MAPS_INPUT("Source", MAPS::FilterIplImage, MAPS::FifoReader)
MAPS_END_INPUTS_DEFINITION

////////////////////////////////////////////////////////////////
// RTMaps - Output
////////////////////////////////////////////////////////////////
MAPS_BEGIN_OUTPUTS_DEFINITION(RedComponent)
  MAPS_OUTPUT("Grey", MAPS::IplImage, NULL, NULL, 1)
  MAPS_OUTPUT("Mirror", MAPS::IplImage, NULL, NULL, 1)
  MAPS_OUTPUT("Scaled", MAPS::IplImage, NULL, NULL, 1)
  MAPS_OUTPUT("Both", MAPS::IplImage, NULL, NULL, 1)
MAPS_END_OUTPUTS_DEFINITION

////////////////////////////////////////////////////////////////
// RTMaps - Properties
////////////////////////////////////////////////////////////////
MAPS_BEGIN_PROPERTIES_DEFINITION(RedComponent)
MAPS_END_PROPERTIES_DEFINITION

////////////////////////////////////////////////////////////////
// RTMaps - Actions
////////////////////////////////////////////////////////////////
MAPS_BEGIN_ACTIONS_DEFINITION(RedComponent)
MAPS_END_ACTIONS_DEFINITION

////////////////////////////////////////////////////////////////
```

```
// RTMaps - Definition
////////////////////////////////////////////////////////////////
MAPS_COMPONENT_DEFINITION(
  RedComponent, "RedComponent", "1.0", 128,
  MAPS::Threaded|MAPS::Sequential,MAPS::Threaded,
  1, // Nb of inputs
  4, // Nb of outputs
  0, // Nb of properties
  0) // Nb of actions

void
RedComponent::Birth()
{
  mIsAllocated = false;
}

void
RedComponent::Core()
{
  MAPSIOElt* input = StartReading(Input("Source"));
  IplImage& source = input->IplImage();
  if (!mIsAllocated)
    allocateOutputs(source);

  MAPSIOElt* outGrey  = StartWriting(Output("Grey"));
  MAPSIOElt* outScaled = StartWriting(Output("Scaled"));
  MAPSIOElt* outMirror = StartWriting(Output("Mirror"));
  MAPSIOElt* outBoth = StartWriting(Output("Both"));

  IplImage& grey  = outGrey->IplImage();
  IplImage& scaled = outScaled->IplImage();
  IplImage& mirror = outMirror->IplImage();
  IplImage& both = outBoth->IplImage();

  /* Convertir l'image en niveaux de gris */
 fonction(source,grey,false,false);
 fonction(source,scaled,true,false);
 fonction(source,mirror,false,true);
 fonction(source,both,true,true);

  /* Set the timestamps */
  outGrey->Timestamp() = MAPS::CurrentTime();
  outScaled->Timestamp() = MAPS::CurrentTime();
  outMirror->Timestamp() = MAPS::CurrentTime();
  outBoth->Timestamp() = MAPS::CurrentTime();

  StopReading(Input("Source"));
  StopWriting(outGrey);
  StopWriting(outScaled);
  StopWriting(outMirror);
  StopWriting(outBoth);
```

```
}

void
RedComponent::Death()
{
  // Couic! :(
}

void
RedComponent::allocateOutputs(IplImage const& src)
{
  IplImage model = MAPS::IplImageModel(src.width, src.height,
      MAPS_CHANNELSEQ_GRAY);

  Output("Grey").AllocOutputBufferIplImage(model);
  Output("Scaled").AllocOutputBufferIplImage(model);
  Output("Mirror").AllocOutputBufferIplImage(model);
  Output("Both").AllocOutputBufferIplImage(model);

  mIsAllocated = true;
}

void
RedComponent::fonction(IplImage const& src,IplImage & dst, bool scaled, bool
    mirror)
{
  for(int i=0; i<src.width*src.height;i++) {

    int val = (src.imageData[3*i]*0.299) + (src.imageData[(3*i)+1]*0.587) + (src
        .imageData[(3*i)+2]*0.114);

    if (scaled) {
      if (val >=0 && val < 50)
        val = 25;
      else if (val >= 50 && val < 100)
        val = 75;
      else if (val >= 100 && val < 150)
        val = 125;
      else if (val >= 150 && val < 200)
        val = 175;
      else
        val = 225;
    }

    if (mirror)
      dst.imageData[((i/src.width)+1)*src.width-(i%src.width)] = val;
    else
      dst.imageData[i] = val;
  }
}
```

3 Développement lors de la troisième séance

3.1 Affichage en couleurs

Pour afficher une image en couleurs, il suffisait simplement de modifier la structure `IplImage` comme ceci dans la fonction `allocateOutputs(IplImage const& src)` :

C++

```
void
RedComponent::allocateOutputs(IplImage const& src)
{
  IplImage model = MAPS::IplImageModel(src.width, src.height,
     MAPS_CHANNELSEQ_BGR);
  Output("Color").AllocOutputBufferIplImage(model);
  mIsAllocated = true;
}
```

Il suffisait ensuite de modifier la fonction `fonction(...)` utilisée précédemment comme ceci :

C++

```
void RedComponent::fonction(IplImage const& src,IplImage & dst)
{
  for(int i=0; i<src.width*src.height;i++) {

    int blue = src.imageData[3*i];
    int green = src.imageData[3*i+1];
    int red  = src.imageData[3*i+2];

    dst.imageData[3*i] = blue;
    dst.imageData[3*i+1] = green;
    dst.imageData[3*i+2] = red;
  }
}
```

3.2 Affichage en couleurs de la composante prédominante

C++

```
void RedComponent::fonction(IplImage const& src,IplImage & dst)
{
  for(int i=0; i<src.width*src.height;i++) {

    int blue = src.imageData[3*i];
    int green = src.imageData[3*i+1];
    int red  = src.imageData[3*i+2];
```

```
    dst.imageData[3*i] = (blue>=red)&&(blue>=green)?255:0;
    dst.imageData[3*i+1] = (green>=blue)&&(green>=red)?255:0;
    dst.imageData[3*i+2] = (red>=blue)&&(red>=green)?255:0;
  }
}
```

3.3 Résultat visuel

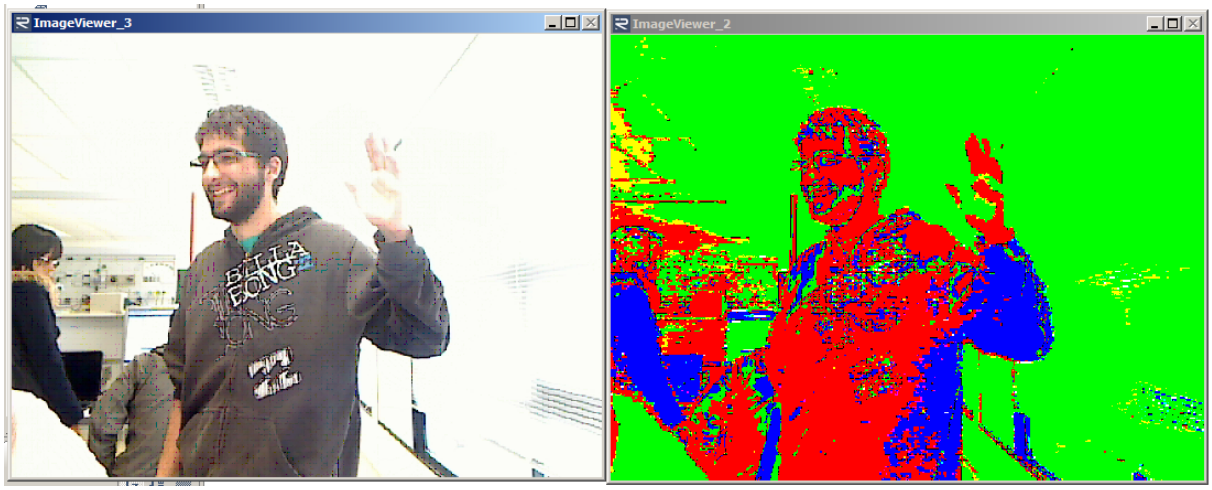

FIGURE 3.1 – Affichages en couleur et de la prédominante