# RegTIRVIS : Thermo-optical image registration

Mohamed Ali CHEBBI

12 octobre 2017

## 1 Introduction

The present paper explains the steps followed to coregister Thermal to optical images using the Elise Library of MicMac. The main procedures are highlighted and their functionalities explained. It will allow the programmer to understand the pieces of code.

## 2 Class diagram

The following digram highlights the different classes that are defined to fulfill the registration task.

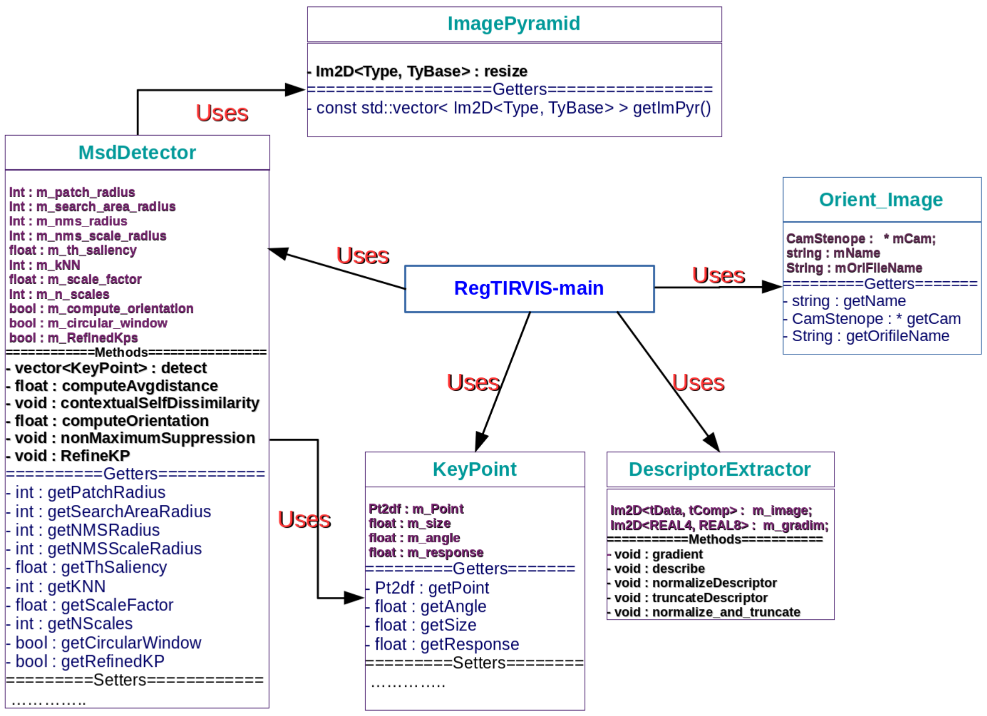

FIGURE 1 – Non exhaustive class diagram

# 3 Explaining the different classes and functionalities

## 3.1 RegTIRVIS_main

This is the main function that enbody the whole registration process. It is not explained in the class diagram because there a lot of functions and procedures. Therefore, it is handy to discuss them in detail. The overall process is based on searching algorithms that use KD tree structures.

$>$ $void \quad Resizeim(Im2D < Type, TyBase > im, Im2D < Type, TyBase > Out, Pt2drNewsize)$ : This procedure is used to resize an image using a certain interpolation method. The need to resize images is dictated by the multiscale analysis during MSD keypoints computation (Image pyramid)

$>$ $void \quad EnrichKps(vector < KeyPoint > Kpsfrom, ArbreKD * Tree, cElHomographie\ \&Homog, int\ \ NbIter)$ : As explained in the internship report, the algorithm of keypoints enriching is encaplsulated in this procedure. The latter takes the initial keypoints with the matching-based homography to enrich tie points using this a priori (Initial homography). This helps delocalize the homography so that it will be a good predictor for dyke registration.

$>$ $void \quad wallis(Im2D < U\_INT1, INT > \ \& \ image, Im2D < U\_INT1, INT > \ \& \ WallEqIm)$ : The Wallis filter algorithm is implemented using this procedure. It takes as entry an image and gives an output image with the algorithm applied.

$>$ $void \quad Migrate2Lab2wallis(Tiff\_Im \ \& \ image, Im2D < U\_INT1, INT > \ \& \ Output)$ : Prior to the description process, we perform RGB to lab colorimetric transition and apply the wallis filter. The whole process is managed using this procedure which calls a structure named RGB2Lab_b and the wallis procedure.

$>$ $void \quad StoreKps(vector < KeyPoint > KPs, string\ \ file)$ : Sicne the computation time is high, we rather store computed keypoints so that we won't do that again.

$>$ $vector < Pt2dr > \quad NewSetKpAfterHomog(std :: vector < \textbf{KeyPoint} > Kps, cElHomographieH)$ : Applies a homography H to a set of Keypoints.

$>$ $vector < Pt2dr > \quad NewSetKpAfterHomog(std :: vector < \textbf{SiftPoint} > Kps, cElHomographieH)$ : The same but for SiftPoint (s).

$>$ $vector < KeyPoint > \quad FromSiftP2KeyP(vector < SiftPoint > Kps)$ : Allows transitioning from SiftPoint class to KeyPoint class. It is applied to a whole set of SiftPoint (s).

$>$ $void \quad Readkeypoints(std :: vector < KeyPoint > \ \ \& \ Kps, string\ file)$ : Reads a set of KeyPoint (s) stored in a certain file.

$>$ $string \quad WhichThermalImage(string\ \ VisualIm, vector < string > \ \ ThermalSet)$ : Identifies a thermal having as entries its homologous Visual or otpical image and the whole set of thermal images.

$>$ $string \quad WhichVisualImage(string\ \ ThermalIm, vector < string > \ \ VisualSet)$ : Identifies an optical image having as entries its homologous thermal image and the whole set of optical images.

$>$ $void \quad ParseHomol(string\ \ MasterImage, vector < cCpleString > ImCpls, vector < string > \&ListHomol)$ : Having a set of overlpping images computed using the MicMac command GrapheHom, we can determine for each thermal the set of overlpping thermal images without an a priori on the orientation. Therefore, this procedure takes as entry a thermal image, the set of overlapping couples (GrapheHom), and gives a list of image names that overlap the entered thermal image.

```
struct PtAndIdx
{
    Pt2dr Pt;
    int CaptureIndex;
};
```

This structure encapsulates a Pt2dr point (2D) and its index in a vector of points. It hemps us find the corresponding KeyPoint.

The latter overview is not exhaustive. Now, we focus on the thermal orientation problem. We use a Test set to compute the homography predictor. Then, we use the homography to infer matches (thermal + thermo-optical).
Taking the **TestSet** :

+ **We compute MSD KeyPoint (s).**

+ **We move to the Lab and Wallis space.**

+ **We compute the Sift Descriptor for each KeyPoint==> DigeoFile thermal + DigeoFile optical.**
+ **We match KeyPoint (s) using the available command : Ann**

```
list<string> cmd;
string aCmd=MM3DStr +  " Ann "+ std::string("-ratio 0.9") +
   std::string(" DigeoTH.txt") + std::string(" DigeoV.txt") +
   std::string(" Matches.txt");
cmd.push_back(aCmd);
cEl_GPAO::DoComInParal(cmd);
```

+ **We compute an initial robust homography using the matched interest points.**

```
bool Exist= ELISE_fp::exist_file("Matches.txt");
if (Exist)
  {
     HomologousPts= ElPackHomologue::FromFile("Matches.txt");
  }
cElComposHomographie Ix(0,0,0);
cElComposHomographie Iy(0,0,0);
cElComposHomographie Iz(0,0,0);
cElHomographie H2estimate(Ix,Iy,Iz);

double anEcart,aQuality;
bool Ok;
Hout=H2estimate.RobustInit(anEcart,&aQuality,HomologousPts,Ok,50,80.0,2000);
Hout.Show();
std::cout<<"ecart "<<anEcart<<endl;
std::cout<<"quality "<<aQuality<<endl;
```

```
std::cout<<Ok<<endl;
```

+ **We enrich keypoints having the intial a priori (Homography) and compute the final homography predictor.**

```
// Construct a Kd tree for the destination keypoints (slave image: optical)


 ArbreKD * ArbreV= new ArbreKD(Pt_of_Point, box, KpsV.size(), 1.0);
 for (uint i=0; i<KpsV.size(); i++)
 {
   ArbreV->insert(pair<int,Pt2dr>(i, Pt2dr(KpsV.at(i).getPoint().x,
    KpsV.at(i).getPoint().y)));
 }
// Call Enrich Keypoints to use the homography as predictor

EnrichKps(KpsTh,ArbreV,Hout,5);
```

Now that the homography predictor is computed using a couple of thermal and optical image (From the tractor dataset : TestSet), we can move forward to compute the Dyke thermal set orientation and the 3D similarity that link both optical and thermal coordinate systems.

==> **Dataset : Dyke**

The overall process is explained hereafter :

+ **First of all, we need to check that each thermal image has its corresponding optical one. The whole set is therefore coherent.**
  Thermal images should have the prefix **TIR**
  Optical images should have the prefix **VIS**

+ **An a priori on the thermal set visibility or overlapping is acquired using the optical set orientation ==> GrapheHom**

```
 //Compute the graph of image correspondences
 list< string > CMD;
 string aCMD;

 //Define the pattern of visual images
 string aPatImVIS="VIS*"+aPatIm;

 //1. Read the xml file containing couples of images

std::vector< cCpleString> ImCpls;
if (!DoesFileExist("GrapheHom.xml"))
{
     aCMD= MM3DStr + "␣GrapheHom␣" + aDirImages + "␣\""
     + aPatImVIS + "\"␣" + Oris_VIS_dir;
      std::cout<<"␣Command␣"<<aCMD<<endl;
     CMD.push_back(aCMD);
```

```
    cEl_GPAO::DoComInParal(CMD);

    std::cout<<"Graph hom built \n";
}
```

+ **The depth images are therefore computed. We choose to work in the geometry ==> ZBufferRaster**

```
ELISE_ASSERT(ELISE_fp::exist_file(PlyFileIn),"Mesh file not
computed or is corrupted ");
std::cout<<" We checked the ply file \n";
//3. Compute ZBuffer Images to get depth information for each pixel
string dirDepthImages="./Tmp-ZBuffer/" + PlyFileIn;

if (!ELISE_fp::exist_file(dirDepthImages))
{
    aCMD=MM3DStr + " TestLib ZBufferRaster " + PlyFileIn + " \"" +
    aPatImVIS + "\" " + Oris_VIS_dir;
    CMD.push_back(aCMD);
    cEl_GPAO::DoComInParal(CMD);
}

ELISE_ASSERT(ELISE_fp::IsDirectory("./Tmp-ZBuffer"),
"ZBuffer Directory has not been created");
```

+ **MSD and Sift keypoints are therefore computed for thermal and optical images**
**==> MSD keypoints**

```
// Thermal image

Tiff_Im ImTh=Tiff_Im::UnivConvStd(ThermalImages.at(i));
std::vector<KeyPoint> KpsTh=msd.detect(ImTh);
StoreKps(KpsTh,FileTh);

//Optical image
Tiff_Im ImV=Tiff_Im::UnivConvStd(VisualImages.at(i));
std::vector<KeyPoint> KpsV=msd.detect(ImV);
StoreKps(KpsV,FileV);
```

**==> SIFT keypoints**

```
CMD.clear();
for (int i=0;i<ThermalImages.size();i++)
    {
    std::string FileTh=KpsfileSIFT + "/" + ThermalImages.at(i) + ".key";
    if (!DoesFileExist(FileTh.c_str()))
    {
        aCMD= MM3DStr + " Sift " + ThermalImages.at(i) + " -o " + FileTh;
        CMD.push_back(aCMD);
    }
    std::string FileV=KpsfileSIFT + "/" + VisualImages.at(i) + ".key";
    if (!DoesFileExist(FileV.c_str()))
    {
        aCMD= MM3DStr + "␣Sift␣" + VisualImages.at(i) + "␣-o␣" + FileV;
        CMD.push_back(aCMD);
    }
    }
cEl_GPAO::DoComInParal(CMD);
```

+ **Computing thermal tie points using the algorithm explained in the internship report**

```
for each couple of optical images given by GrapheHom
  {
   ==> Determine their corresponding thermal images

   ==> Apply the homography predictor to the thermal images relative
   keyPoints (MSD+ SIFT)

   ==> Use the orientation of optical images to compute homologous points
       Orient_Image ImV1(Oris_VIS_dir,ImCpls.at(i).N1(),aICNM);
       Orient_Image ImV2(Oris_VIS_dir,ImCpls.at(i).N2(),aICNM);

   ==> Use the depth image to obtain Ground truth values
       Depth=Im2D<REAL4,REAL>::FromFileStd(FileDepthImV1);
   if (Prof!=-1)
        {
        Pt3dr pTerrain= ImV1.getCam()->ImDirEtProf2Terrain(Kps1H.at(j),Prof,
        ImV1.getCam()->DirVisee());
        Pt2dr PtImage2= ImV2.getCam()->Ter2Capteur(pTerrain);

        Voisins.clear();

      ==> Search for homologous points by seraching for nearest
          neighbours in the Slave keypoints Tree


        SlaveTree->voisins(PtImage2, distMax, Voisins);

         if (Voisins.size()>0)
            {
              Pt2dr P1(Kps1.at(j).getPoint().x, Kps1.at(j).getPoint().y);
              Pt2dr Pnew(Kps2.at(Voisins.begin()->first).getPoint().x,
              Kps2.at(Voisins.begin()->first).getPoint().y);
              HomologousPts.Cple_Add(ElCplePtsHomologues(P1,Pnew));
            }
        }
   }
```

The Overall process gives birth to a directory named ./Homol (same convention as MicMac). This file can be entered to Tapas to compute the thermal set orientation (Interior + Exterior).

+ **Computing thermo-optical tie points using the relevant algorithm explained in the internship report**

```
for all images
```

```
{
  Compute Keypoint masks and store them
      if (thermal)
         {
           Apply homography to the set of Keypoints
         }
      else
         {
           Use Keypoints as they are
         }
}

for each image in the dataset (thermal or optical)
  {
   1)Search for overlapping images (thermal and optical)
       //search inside ImCpls
         std::vector<string> VisuHomols;
         ParseHomol(VisualImages[i],ImCpls,VisuHomols);
   2) Get these images corresponding masks

   for (all Keypoints in the master image)
   {
     3) Project to ground
        Pt3dr pTerrain= ImV.getCam()->ImDirEtProf2Terrain
        (AllKpsThermalHomog.at(i).at(k),Prof,ImV.getCam()->DirVisee());




     4) Search for tie points in the overlapping images
     for (int n=0;n<(int)ThermalImages.size();n++)
       {
         // there is a couple of thermal and visual images
         seen by the master image
         if(WhichIsSeen[2*n])
           {
            Orient_Image ImVslave(Oris_VIS_dir,VisualImages[n],aICNM);

            4.1) Reproject Ground point in the overlpping image space

               Pt2dr Ptslave= ImVslave.getCam()->Ter2Capteur(pTerrain);

            4.2) Search for homologous points

                 AllTrees->at(2*n)->voisins(Ptslave, distMax, Voisins);
           }
       }
      5) Check for multiplicity: If the tie points are mulitple
   }

  }
```

The 5th step mandates the use of multiple tie points to make sure that there are sufficient points seen by thermal and optical images. A subsequent link is then built between the two modalities. We store 2D points coordinates according to their mother images. We

also store the ground truth optical point which has led to the 2D measures.
By the end, we obtain a file of 3D and 2D measures where only thermal images are involved (Remember we register thermal ====> Optical images, So we need thermal 2D points that are seen in optical images and have a ground values)

The 2D and 3D measure files are then used by the command Bar available on MicMac software whic computes a robust 3D similarity between TWO coordinate systems and applies the computed transformation to the first set (THERMAL) orientation files. The latter is moved to the new frame (Optical).

## 3.2 MsdDetector

Maximum Self Dissimilarity (MSD) interest points are computed using this class. As highlighted in the class diagram several parameters are involved in this process.

+ Int : m_patch_radius : The radius of the patch needed to compute the correlation criterion (SSD or NCC)
+ Int : m_search_area_radius : The radius of the region over which the patches are to be sliding
+ Int : m_nms_radius : The non maxima suppression step is defined over a certain vicinity
+ Int : m_nms_scale_radius : Non maxima suppression is extended to the scale space
+ float : m_th_saliency : The saliency threshold.
+ Int : m_kNN : To compute the saliency operator, an average value is computed rather than taking the distance to the most similar pixel (SSD or NCC). This provides robustness under noise nuisances.
+ float : m_scale_factor : Defines the scale factor for the pyramid computation.
+ Int : m_n_scales : The number of layers in the image pyramid
+ bool : _compute_orientation : Dictates whether we need to compute each keypoint orientation or not.
+ bool : m_circular_window : Use a weighted wircular patch or not
+ bool : m_RefinedKps : Refine Keypoints by fiiting a quadric to the saliency map at the initial keypoint location.

==> **Methods**

> $float \quad computeAvgDistance(std :: vector < float > \& \ minVals, intden)$ : Computes the saliency operator based on a set of nearest neighbours.

> $void \quad contextualSelfDissimilarity(Im2D < U\_INT2, INT > \& \ img, int \ xmin, int \ xmax, float \ \ *saliency)$ : Allows to compute saliency maps for the image pyramid (scale space) using a certain correlation criterion.

> $float \quad computeOrientation(Im2D < U\_INT2, INT > \& \ img, int \ x, int \ y, vector < Pt2df > \quad circle)$ : Computes the orientation of a certain interest point. It is needed to make the description robust under rotation changes.

> $void \quad nonMaximaSuppression(std \ :: \ vector \ < \ float* \ > \ \& \ saliency, vector \ < KeyPoint > \& \ keypoints)$ : This method performs non maxima suppression in the scale space and in the current image space. It gives birth to the set of Keypoints.

> $vector < KeyPoint > \quad detect(Tiff\_Im \ \& \ img)$ : This methods wraps up the whole process and calls all the other methods. It returns the set of keypoints computed by the non Maxima Suppression method.

## 3.3 KeyPoint

We introduce a new class that define a certain interest point by its coordinates, its orientation (angle) and the scale at which it appears. Thses ingredients are necessary for the description step to succed.

+ Pt2df : m_Point : Interest point coordinates
+ float : m_size : Interest point scale or size
+ float : m_angle : Interest point orientation
+ float : m_response : Interest point saliency value.

## 3.4 DescriptorExtractor

This class computes the SIFT descriptor for each interest point. It follows the steps described in the original Paper (Lowe, 2004). It is inspired from the work of Arnaud Le Bris under the Library Elise of MicMac.

+ Im2D<tData, tComp> : m_image : The image that is used to compute interest points.
+ Im2D<REAL4, REAL8> : m_gradim ; The gradient image.

==> **Methods**

> $void \quad gradient(REAL8 \;\; i\_maxValue)$ : Computation of the gradient image.

> $void \quad describe(REAL8 \;\; i\_x, REAL8 \;\; i\_y, REAL8 \;\; i\_localScale, REAL8i\_angle, REAL8 \;\; *o\_descriptor)$; Compute the sift descriptor for a ceratin interest point knowing its coordinates, orientation and characteristic scale.

> $void \quad normalizeDescriptor(REAL8 \;\; *io\_descriptor)$ : The SIFT descriptor is therefore normalized to enhance robustness under radiometric changes.

> $void \quad truncateDescriptor(REAL8 \;\; *io\_descriptor)$ : Tuncate the descriptor values using a threshold of value 0.2.

## 3.5 ImagePyramid

This class is only used by the MsdDetector class to compute the image pyramid for a multiscale analysis. It uses a resizing method which in turn exploits a Nearest Neighbour interpolation scheme to assign radiometric values to the resized image.

## 3.6 Orient_Image

This class orients a certain image by using its orientation file and the stenope prjection assumption (CamStenope). It Inherits the classes CamStenope that have methods allowing to orient the camera according to the orientation file. It is mainly used to take advantage from the optical set orientation.

# 4 Conclusion

The present gives a brief explanation of the coding scheme that is followed to address the registration of thermal to optical images. It is not exhaustive but discusses almost all the coding steps so that the work could be exended.