1 Introduction

Ce document a été réalisé durant la formation MICMAC développeur réalisé à l'ENSG du 08 au 12 décembre. Durant ces 5 jours de formation, plusieurs exercices ont été réalisés. Le premier concerne la conversion de calibration de caméra d'un format dans un autre en utilisant des points d'appuis fictifs. Le second concerne l'ajout d'équations d'observations concernant une paire de caméra rigidement liée et synchronisée.

Avant d'expliquer les différents exercices, voici deux rappels succincts concernant la mise à jour des sources depuis mercurial ainsi que la compilation des sources.

1.1 Rappel Mercurial

- **hg push** permet de mettre à jour le dépôt local (dossier micmac) en récupérant les sources du dépôt distant (ign.fr)
- **hg update** permet de mettre à jour les sources depuis le dépôt local (dossier micmac)

1.2 Rappel Compilation

- **make -j X** compile les sources et créer les binaires
- **make install** copie les binaires dans le dossier .../micmac/bin

2 Jour 1 : Conversion de calibration

Le but de cet exercice est de créer un programme permettant de transformer un format de camera en un autre format. Notre programme va:

- Lire le fichier caméra
- Générer une série de points en géométrie image (c, l)
- Calculer des points terrain (X, Y, Z) à partir des coordonnées images (c, l)
- Généré les fichiers de mesures et de points d'appuis nécessaires à Apero (qui effectuera le calcul de la nouvelle calibration à partir des données générées)

2.1 Objets utiles

2.1.1 Parseurs d'arguments

MicMac possède son propre parseur d'argument qui permet de spécifier les paramètres obligatoires et facultatifs. Ce parseur permet de vérifier le bon type de données ainsi que le bon nombre d'argument.

```
int main(int argc, char** argv)
        double a;
        std::string b;
        int c;
        bool d;
ElInitArgMain
(
//nb d'arguments
   argc,
//chaines de caracteres contenants tous les arguments
   argv,
//arg obligatoires
   LArgMain() << EAMC(a, "Param a",eSAM_IsExistFile) << EAMC(b,"Param b",eSAM_IsExistFile),
//arg facultatifs
   LArgMain() << EAM(c,"C",true,"Param C") << EAM(d,"D",true,"Param D")
);
   return EXIT_SUCCESS;
}
```

2.1.2 Caméras

La lecture d'un fichier XML de caméra se fait par la fonction *Std_Cal_From_File* qui retourne un pointeur sur un objet de type caméra.

```
CamStenope * aCamera =  Std_Cal_From_File("camera.xml");
```

Fonction permettant de passer d'un point image (coordonnées exprimées en (c, l)) et une distance à un point 3D ((X, Y, Z))

```
Pt2dr PtIm (123.4,567.8);
double distance=12.4;
Pt3dr aPGround = aCamera->ImEtProf2Terrain(PtIm,distance);
```

2.2 Génération des fichiers

2.2.1 Fichiers XML

MicMac utilise plusieurs fichiers xml pour fonctionner. Il existe des classes cpp permettant de les écrire automatiquement. Pour chaque balise XML, il existe une classe permettant de la lire. Par exemple pour la balise *DicoAppuisFlottant*, il existe une classe *cDicoAppuisFlottant*. Les balises filles sont des données membres de la classe issue du nom de la balise mère. Voici le code cpp ainsi que le fichier xml généré pour le fichier appui et le fichier mesure.

2.2.2 Fichier appui

Code cpp :

```
cDicoAppuisFlottant aDAF;

cOneAppuisDAF anApA;
Pt3dr aPGroundA(979.094 6533.994 1.055);
anApA.Pt() = aPGroundA;
anApA.Incertitude() = Pt3dr(1.,1.,1.);
anApA.NamePt() = "Megev1055";

cOneAppuisDAF anApB;
Pt3dr aPGroundB( 981.103 6553.402 1.102);
anApB.Pt() = aPGroundB;
anApB.Incertitude() = Pt3dr(1.,1.,1.);
anApB.NamePt() = "Carroz1002";

aDAF.OneAppuisDAF().push_back(anApA);
aDAF.OneAppuisDAF().push_back(anApB);

MakeFileXML(aDAF, "Mes3D.xml");
```

Fichier XML généré :

```
<?xml version="1.0" ?>
<DicoAppuisFlottant>
     <OneAppuisDAF>
          <Pt> 979.094 6533.994 1.055 </Pt>
          <NamePt> Megev1055 </NamePt>
          <Incertitude>1 1 1</Incertitude>
     </OneAppuisDAF>
     <OneAppuisDAF>
          <Pt>  981.103 6553.402 1.102 </Pt>
          <NamePt> Carroz1002 </NamePt>
          <Incertitude>1 1 1</Incertitude>
     </OneAppuisDAF>
</DicoAppuisFlottant>
```

2.2.3 Fichier mesure

Code cpp :

```
cSetOfMesureAppuisFlottants XMLMesuresImages;


cMesureAppuiFlottant1Im XMLImageA;
XMLImageA.NameIm()="T0-_MG_0001.JPG";

cOneMesureAF1I XMLMesureA;
XMLMesureA.NamePt()="Megev1055";
```

```
XMLMesureA.PtIm()=Pt2dr(189.32188537332104,2365.73434175065222);
XMLImageA.OneMesureAF1I().push_back(XMLMesureA);

cOneMesureAF1I XMLMesureB;
XMLMesureB.NamePt()="Carroz1002";
XMLMesureB.PtIm()=Pt2dr(11395.38751581008842,862.673280912829);
XMLImageA.OneMesureAF1I().push_back(XMLMesureB);


cMesureAppuiFlottant1Im XMLImageB;
XMLImageB.NameIm()="T0-_MG_0002.JPG";

cOneMesureAF1I XMLMesureC;
XMLMesureC.NamePt()="Megev1055";
XMLMesureC.PtIm()=Pt2dr(1703.75004653847122,2356.52137674056758);
XMLImageB.OneMesureAF1I().push_back(XMLMesureA);

cOneMesureAF1I XMLMesureD;
XMLMesureD.NamePt()="Carroz1002";
XMLMesureD.PtIm()=Pt2dr(1872.70867622042897,849.874687754920728);
XMLImageB.OneMesureAF1I().push_back(XMLMesureB);

MakeFileXML(XMLImage, "Mesure2D.xml");
```

Fichier XML généré :

```
<?xml version="1.0" ?>
<SetOfMesureAppuisFlottants>
     <MesureAppuiFlottant1Im>
          <NameIm>T0-_MG_0001.JPG</NameIm>
          <OneMesureAF1I>
               <NamePt>Megev1055</NamePt>
               <PtIm>1189.32188537332104 2365.73434175065222</PtIm>
          </OneMesureAF1I>
          <OneMesureAF1I>
               <NamePt>Carroz1002</NamePt>
               <PtIm>1395.38751581008842 862.673280912829</PtIm>
          </OneMesureAF1I>
     </MesureAppuiFlottant1Im>
     <MesureAppuiFlottant1Im>
          <NameIm>T0-_MG_0002.JPG</NameIm>
          <OneMesureAF1I>
               <NamePt>Megev1055</NamePt>
               <PtIm>1703.75004653847122 2356.52137674056758</PtIm>
          </OneMesureAF1I>
          <OneMesureAF1I>
```

```
            <NamePt>Carroz1002</NamePt>
             <PtIm>1872.70867622042897 849.874687754920728</PtIm>
        </OneMesureAF1I>
    </MesureAppuiFlottant1Im>
<SetOfMesureAppuisFlottants>
```

2.2.4 Utilisation de programmes MicMac

On peut également faire appel à un programme de la suite MicMac à partir d'un autre. Il suffit d'utiliser la fonction *MM3dBinFile* avec le nom du programme à utiliser ainsi que les arguments nécessaires au programme puis de faire appel à la commande *System*

```
std::string aCom =    MM3dBinFile("Apero")
                    + XML_MM_File("Apero-ConvCal.xml")
                    + " DirectoryChantier=" +DirOfFile(aCalibIn)
                    + " +AeroIn=ConvCalib"
                    + " +CalibIn="  + aCalibOut
                    + " +CalibOut=" + aNameCalibOut
                    + " +FocFree="  + ToString(FocFree)
                    + " +PPFree="   + ToString(PPFree)
                    + " +CDFree="   + ToString(CDFree)
                    + " +DRMax=" + ToString(aDRMax)
                    + " +DegGen=" + ToString(aDegGen)
                   ;


System(aCom);
```

2.2.5 Intégration dans la chaîne MicMac

Pour integrer son programme dans la chaîne MicMac et l'appeler avec la syntaxe *mm3d MonProgramme*, il faut ajouter dans le fichier *src/CBinaires/mm3d.cpp* :

- *extern int MonProgramme (int argc, char **argv)* : déclaration de la fonction
- *aRes.push_back(cMMCom("MonProgramme",MonProgramme,"Explication de mon programme"));* : intégration dans la suite mm3d avec le nom du programme "MonProgramme" ainsi que son explication "Explication de mon programme"

3 Jour 2 et 3 : Implémentation d'un bloc rigide de 2 caméras

3.1 Données utilisées

La caméra utilisé pour ce jeux de données genere des images au format MPO. Ce format permet de stocker dans un même fichier les deux images prises par la caméra. Un outil de MicMac permet de séparer le fichier MPO en 2 fichiers images. Voici la commande *mm3d SplitMPO ".MPO" pour récupérer les paires d'images.*

3.2 Description des différents processus

Plusieurs étapes sont nécessaires pour ajouter des observations dans le système de compensation de MicMac :

1. Écriture mathématique des équations dans le système de calcul formel distribué par MicMac
2. Génération des fichiers cpp contenant les classes avec entre autre la différentielle et le calcul de résidus
3. Ajout dans le système de compilation MicMac
4. Modification du fichier de paramètres
5. Création des fichiers cpp pour la lecture du fichier de paramètres
6. Interaction entre tout ça

La plupart du code décrit ci-après provient du fichier *micmac/src/uti_phgrm/Apero/cImplemBlockCam.cpp*

3.2.1 Écriture mathématique des équations

On appelle i l'instant ou des images sont prises en même temps. On appelle a et b les deux caméras de la paire stéréo. Considerer un block rigide de 2 caméras a et b signifie que la transformation T comportant rotation R et translation T entre 2 instants est constant.

$$T_{a,b,i} = T_{a,b,i+1} \tag{1}$$

MicMac possède son sytème de calculs formels. Cela permet de calculer automatiquement la dérivée des observations par rapport aux paramètres mais également d'optimiser le code généré (minimisation du nombre de calculs élémentaires).

Quelques éléments se trouvent dans les fichiers */micmac/include/general/phgr_formel.h* et dans */micmac/src/uti_phgrm/Apero/Conventions.txt*

Voici les 2 fonctions concernées par le calcul formel

```
void GenerateCode()
{
          cSetEqFormelles *   mSet;
          cRotationFormelle * mRotRT0;
          cRotationFormelle * mRotLT0;
          cRotationFormelle * mRotRT1;
          cRotationFormelle * mRotLT1;
          cIncListInterv      mLInterv;
          cElCompiledFonc*    mFoncEqResidu;

// permet de donner un nom aux inconnues
// sert a la localisation dans le systeme d'equation

   mRotRT0->IncInterv().SetName("OriR0");
   mRotLT0->IncInterv().SetName("OriL0");
   mRotRT1->IncInterv().SetName("OriR1");
   mRotLT1->IncInterv().SetName("OriL1");

   mLInterv.AddInterv(mRotRT0->IncInterv());
   mLInterv.AddInterv(mRotLT0->IncInterv());
   mLInterv.AddInterv(mRotRT1->IncInterv());
   mLInterv.AddInterv(mRotLT1->IncInterv());

// equation d'observation
    Pt3d<Fonc_Num>     aTrT0;
    ElMatrix<Fonc_Num> aMatT0(3,3);
    CalcParamEqRel(aTrT0,aMatT0,*mRotRT0,*mRotLT0,0,1);

    Pt3d<Fonc_Num>     aTrT1;
    ElMatrix<Fonc_Num> aMatT1(3,3);
    CalcParamEqRel(aTrT1,aMatT1,*mRotRT1,*mRotLT1,2,3);

    Pt3d<Fonc_Num> aResTr = aTrT1-aTrT0;
    ElMatrix<Fonc_Num> aResMat = aMatT1-aMatT0;

// residus
    std::vector<Fonc_Num> aVF;
    aVF.push_back(aResTr.x);
    aVF.push_back(aResTr.y);
    aVF.push_back(aResTr.z);
    for (int aKi=0 ; aKi<3 ; aKi++)
        for (int aKj=0 ; aKj<3 ; aKj++)
           aVF.push_back(aResMat(aKi,aKj));

// generation des classes cpp a partir du code precedent
    cElCompileFN::DoEverything
```

```
    (
        DIRECTORY_GENCODE_FORMEL,  // Directory ou est localise le code genere
        "cCodeBlockCam",  // donne les noms de fichier .cpp et .h ainsi que les nom de class
        aVF,  // expressions formelles utilisees pour les observations
        mLInterv  // intervalle de reference
    );
}


void GenerateCodeBlockCam()
{
   cSetEqFormelles aSet;

   ElRotation3D aRot(Pt3dr(0,0,0),0,0,0);
   cRotationFormelle * aRotRT0 = aSet.NewRotation (cNameSpaceEqF::eRotLibre,aRot);
   cRotationFormelle * aRotLT0 = aSet.NewRotation (cNameSpaceEqF::eRotLibre,aRot);
   cRotationFormelle * aRotRT1 = aSet.NewRotation (cNameSpaceEqF::eRotLibre,aRot);
   cRotationFormelle * aRotLT1 = aSet.NewRotation (cNameSpaceEqF::eRotLibre,aRot);


  cEqObsBlockCam * aEOBC = aSet.NewEqBlockCal (*aRotRT0,*aRotLT0,*aRotRT1,*aRotLT1,true);
  DoNothingButRemoveWarningUnused(aEOBC);
}
```

3.2.2 Génération du code CPP

Un fois la fonction permettant l'équation formelle écrite, il faut appeler l'utilitaire qui permet de générer la classe effectuant le calcul. Il s'agit de *GenCode* fichier : *src/uti_files/CPP_GenCode.cpp* et d'appeler la fonction *GenerateCodeBlockCam()*.

Deux fichiers sont alors créés *cCodeBlockCam.h* et *cCodeBlockCam.cpp* dans le dossier */micmac/CodeGenere/photogram/* . Ces fichiers contiennent plusieurs fonctions dont :

- ComputeVal() : calculs les résidus
- ComputeValDeriv() : calcul la dérivée par rapport aux inconnues

3.2.3 Ajout du code dans MicMac

Ensuite, il faut ajouter ces fichiers dans le code de MicMac. Pour cela il faut ajouter

- dans *src/photogram/phgr_or_code_gen00.cpp*

```
#include "../../CodeGenere/photogram/cCodeBlockCam.h"
AddEntry("cCodeBlockCam",cCodeBlockCam::Alloc);
```

- dans *src/photogram/phgr_or_code_gen6.cpp*

```
#include "../../CodeGenere/photogram/cCodeBlockCam.cpp"
```

1

3.2.4 Modification du fichier de paramètres

Lorsqu'on veut écrire de nouvelles observations, il faut pouvoir indiquer au programme via les fichiers xmls traditionnels, les différents paramètres qui sont nécessaires. Il s'agit du fichier *include/XML_GEN/ParamApero.xml*

Cette partie a été ajouté afin de tenir compte des pondérations sur un bloc rigide. La pondération sur la translation et la rotation a été différenciée.

```
<RigidBlockWeighting Nb="1" Class="true" ToReference="true">
    <PondOnTr Nb="1" Type="double"> </PondOnTr>
   <PondOnRot Nb="1" Type="double"> </PondOnRot>
</RigidBlockWeighting>

<UseForBundle Nb="?">
    <GlobalBundle Nb="1" Type="bool">      </GlobalBundle>
    <RelTimeBundle Nb="1" Type="bool">     </RelTimeBundle>
</UseForBundle>
```

Lien vers la pondération déclaré précédemment. *GlobalBundle* correspond à un bloc rigide pour toute une acuqisition tandis que *RelTimePond* correspond à un bloc rigide de proche en proche (on tolère une variation de la rigidité pour toute une acuqisition)

```
<GlobalPond Nb="?" RefType="RigidBlockWeighting"> </GlobalPond>
<RelTimePond Nb="?" RefType="RigidBlockWeighting"> </RelTimePond>
```

3.2.5 Intégration complète dans MicMac

Dans la fonction *void cAppliApero::AddObservations* du fichier *micmac/src/uti_phgrm/Apero/cAA_Compensati*
on doit ajouter :

```
AddObservationsRigidBlockCam(anSO.ObsBlockCamRig(),IsLastIter,aSO);
```

Avec dans le même fichier son implémentation :

```
void cAppliApero::AddObservationsRigidBlockCam
     (
         const std::list<cObsBlockCamRig> & anOBCR,
         bool IsLastIter,
```

[1] MicMac possède un mécanisme de rangement des inconnues afin de minimiser le nombre de coefficients non nul de la matrice à inverser (gain de place et gain de temps de calcul). Ce mécanisme n'est pas présenté dans ce document mais est présent dans les sources. Il s'agit de toutes les fonctions et objets contenant AMD.

```
          cStatObs & aSO
     )
{
    for
    (
       std::list<cObsBlockCamRig>::const_iterator itO=anOBCR.begin();
       itO !=anOBCR.end();
       itO++
    )
    {
                //cette fonction renvoie a l'implementation dans
                // cImplemBlockCam.cpp
          AddObservationsRigidBlockCam(*itO,IsLastIter,aSO);
    }

}
```

La classe cImplemBlockCam du fichier *micmac/src/uti_phgrm/Apero/cImplemBlockCam.cpp* effectue tout le travail. Le constructeur permet d'initialiser toutes les données (lecture de la transformation initiale, tri des caméras...). La fonction *DoCompensation* pose les équations d'observations dans la matrice normale