

BAORadio

Mode d'emploi

5/03/2012 – v0.1

Conception du programme

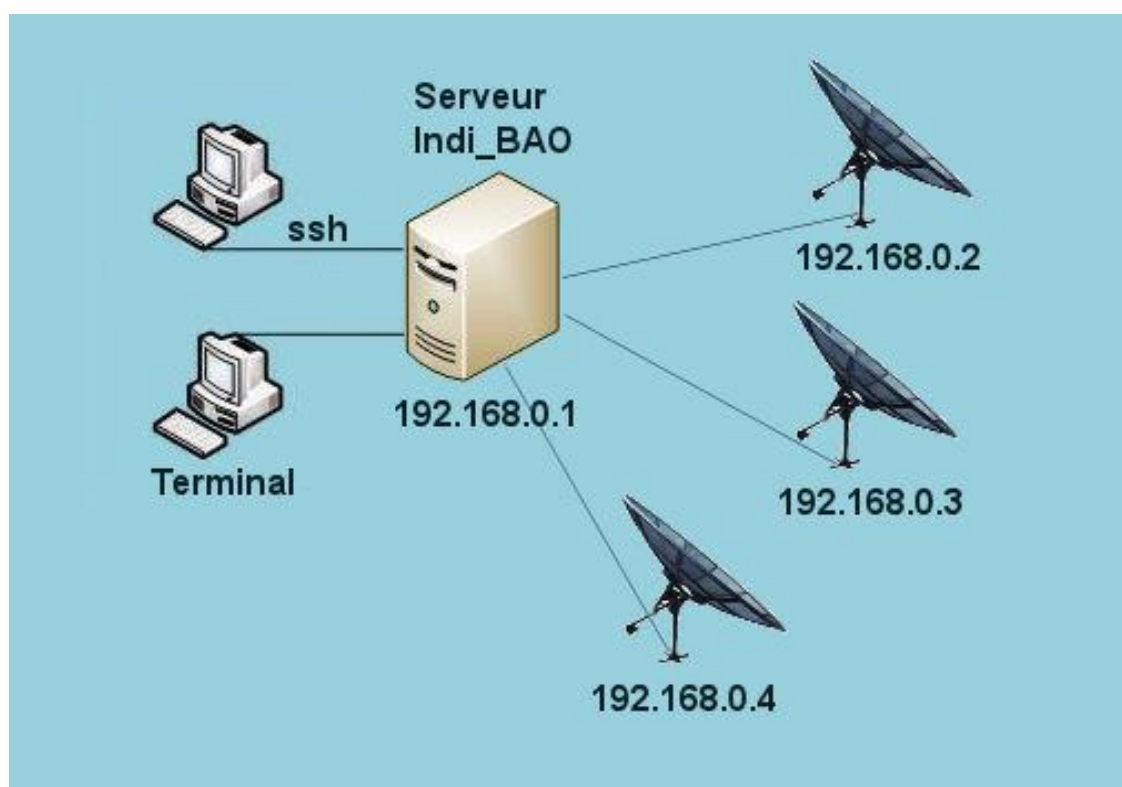
Le code du pilote BAO s'appuie sur la bibliothèque Indi (Instrument-Neutral-Distributed-Interface control protocol), protocole développé par Jasem Mutlaq et qui se démarque des autres solutions de pilotage de matériels astronomiques par les aspects suivants :

- Un logiciel entièrement écrit en C et donc reposant sur de solides efforts de normalisation,
- Une conception du programme qui supporte toutes sortes de liaisons physiques allant des traditionnels ports séries RS-232 très répandus dans le milieu des astronomes amateurs jusqu'aux connexions Ethernet
- La possibilité de gérer du matériel depuis un serveur Indi accessible via une connexion sécurisée ssh.
- Une solution multiplateforme
- Un protocole original utilisant le langage XML et offrant la possibilité d'utiliser les nombreuses bibliothèques développées pour ce langage

Malgré ces qualités, Indi n'était cependant pas tout à fait adapté à ce que nous voulions faire, c'est-à-dire piloter des dizaines d'antennes connectées à un réseau local Ethernet.

Nous avons donc ajouté de nouvelles fonctionnalités permettant la gestion de trames tcp/ip circulant sur un réseau local.

Le schéma suivant détaille le fonctionnement général de BAORadio :



On remarque qu'il est possible de se connecter sur le serveur Indi depuis plusieurs ordinateurs. Il est ainsi possible de gérer les antennes depuis le site de Nançay lui-même mais aussi depuis le laboratoire d'Orsay par l'intermédiaire d'une connexion sécurisée ssh.

L'interface de pilotage BAORadio se décline sous la forme de plusieurs modules :

- Un programme serveur (indi_BAO) qui communique avec toutes les antennes via un réseau local et qui reçoit ses instructions depuis une connexion Ethernet. Cette interface a été développée à partir de la bibliothèque Indi. C'est le chef d'orchestre du dispositif. Ce programme serveur est compatible avec les logiciels de cartographie grand public tournant sous Linux (KStars, Stellarium, XEphem).
- Une petite interface de pilotage des antennes en ligne de commande (BAOcontrol) et prenant en charge des opérations élémentaires comme la calibration des antennes ou la planification d'une série d'objets à observer.
- Une interface permettant de piloter l'installation depuis le logiciel Stellarium, un logiciel d'astronomie très populaire et bien plus prometteur que les logiciels du même type proposés sous environnement Linux. Il m'a semblé utile de proposer cette possibilité supplémentaire...
- Un petit programme de test (BAOtest) est également disponible. Il permet de simuler grossièrement le comportement des microcontrôleurs pilotant les moteurs les antennes. Il permet d'afficher les échanges de trames sur le réseau. Ce petit programme se révèle indispensable pour toutes les opérations de débogage du serveur indi_BAO...

Utilisation du programme

Compilation du programme

Après avoir récupéré le code depuis le serveur subversion du LAL, on se place dans le répertoire principal de l'application et on utilise le script `bao.sh` pour compiler, installer, configurer et éventuellement supprimer le programme de l'ordinateur.

Le script accepte plusieurs options :

- **install** pour compiler puis installer le pilote `indi_BAO` et les utilitaires `BAOcontrol` et `BAOtest` dans le répertoire `/usr/bin` de l'ordinateur
- **install config** pour copier les fichiers de configuration dans le répertoire personnel de l'utilisateur ayant ouvert la session linux
- **clean** pour effacer les fichiers issus de la compilation
- **remove** pour supprimer le pilote et les utilitaires de l'ordinateur

Ainsi, une installation typique prendra la forme suivante :

`./bao.sh install`

Compile et installe le logiciel,

`./bao.sh install config`

copie les fichiers de configuration et le catalogue d'étoile dans le répertoire de l'utilisateur. Enfin,

`./bao.sh clean`

libère de la place sur le disque dur.

En cas de problème de compilation, vérifiez les messages d'erreurs, corrigez les défauts éventuels et relancez le script.

Pour que tout fonctionne, il est indispensable que les paquets **KStars**, **xterm** et **boost**, **curl** soient installés sur l'ordinateur.

Alignement des antennes - principe

Lors de la première utilisation, il est indispensable de calibrer les antennes pour pointer convenablement les objets du ciel. On utilisera l'utilitaire BAOcontrol qui dispose d'une fonction conçue à cet effet. Il est à noter que cette opération n'est à réaliser qu'une seule fois après l'installation des antennes sur le terrain.

Le programme utilise trois méthodes d'alignements basés sur le même principe de fonctionnement :

- 1)- L'ordinateur calcule et affiche une liste d'étoiles brillantes visibles depuis le site d'observation à la date et à l'heure de la calibration.
- 2)- Le programme invite l'utilisateur à sélectionner les objets qu'il souhaite utiliser pour effectuer la calibration.
- 3)- l'ordinateur pointe chaque étoile sélectionnée et demande à l'utilisateur de pointer précisément l'objet dans le viseur de l'antenne.

Trois objets sont nécessaires pour l'alignement d'une antenne utilisant le modèle de correction le plus SIMPLE (c'est son nom). Les méthodes AFFINE et TAKI exigent un plus grand nombre d'objets mais permettent d'obtenir un alignement de meilleure qualité (reportez-vous à la section décrivant le principe de fonctionnement de ces trois méthodes).

À partir des objets pointés successivement et des corrections appliquées en ascension droite et en déclinaison, des matrices de pointage sont calculées.

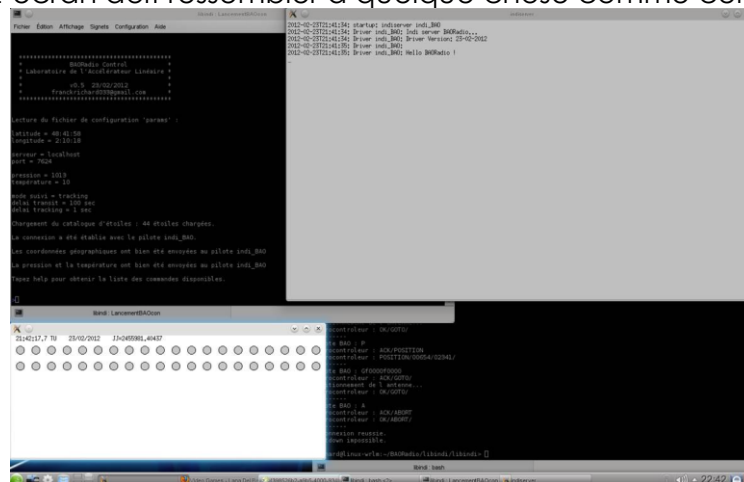
La méthode SIMPLE devrait suffire si la monture ne souffre pas trop d'effets de flexion, etc... A vérifier...

Utilisation de BAOcontrol pour lancer la calibration

On lance BAOcontrol et le server indi_BAO en tapant

./LancementBAOcontrol.sh

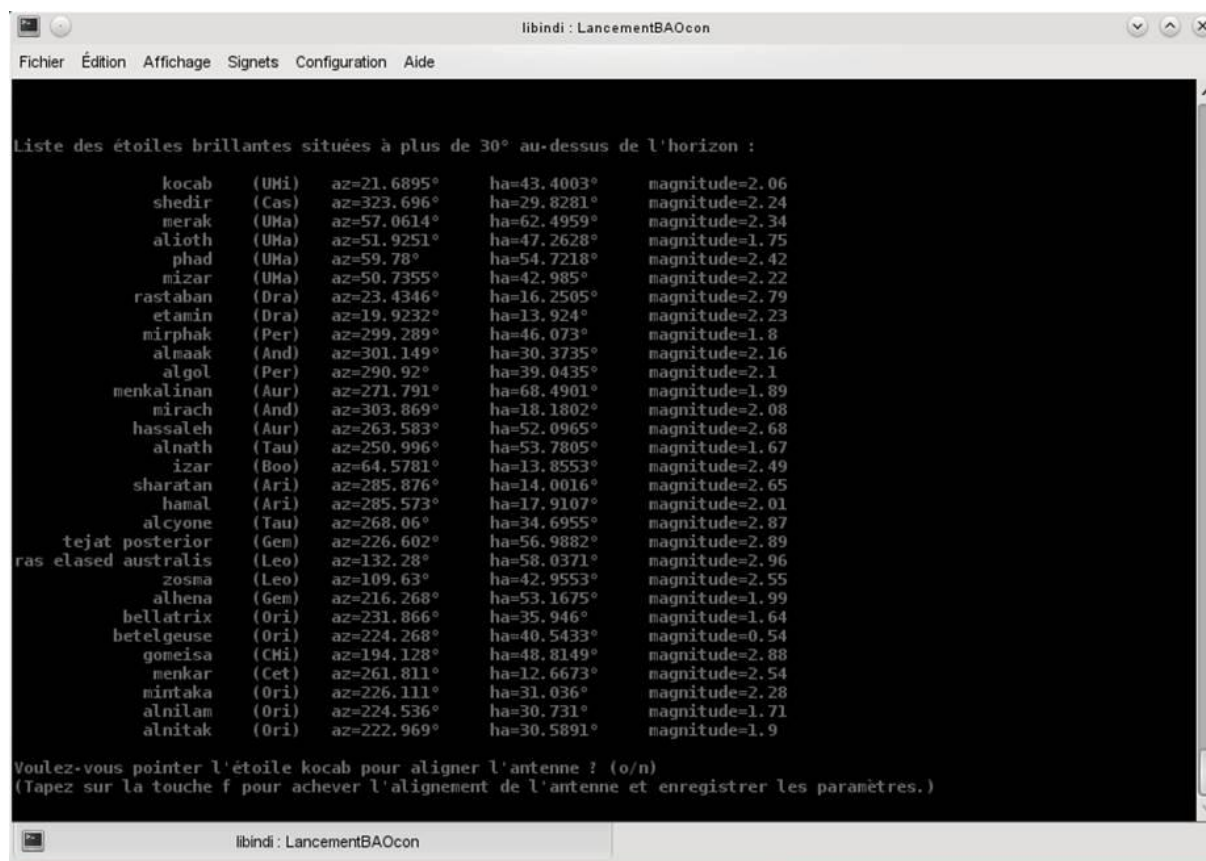
L'écran doit ressembler à quelque chose comme cela :



Vérifiez les dates de version des programmes BAOcontrol et indiserver. Les dates doivent correspondre à la dernière version connue.

- Connectez l'antenne au programme en suivant la procédure habituelle. Le programme doit afficher l'adresse ip de l'antenne. Cette adresse permet d'identifier l'antenne que l'on souhaite aligner.
- On initialise les paramètres d'alignement de l'antenne en utilisant la commande :
reset ip
L'ordinateur affiche un message d'erreur, si l'antenne n'est pas présente sur le réseau.
- Puis, on lance la procédure d'alignement de l'antenne en tapant :
align ip

Le programme indique la liste des étoiles brillantes situées à une hauteur supérieure à 30° au-dessus de l'horizon (et donc observables par l'instrument) :



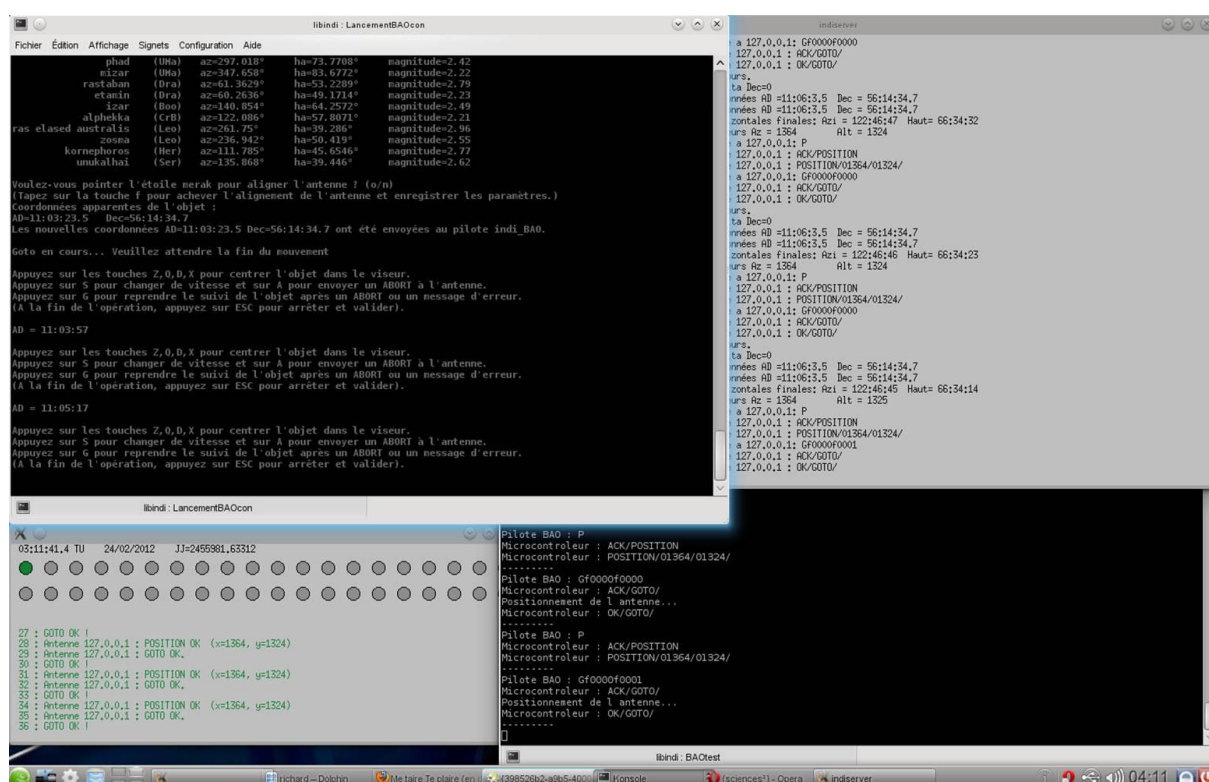
Il faut se munir d'une bonne carte du ciel pour trouver les étoiles indiquées. Il ne faut pas se tromper d'étoile durant la calibration de l'instrument. La matrice de correction en serait évidemment faussée...

La hauteur et l'azimut de l'étoile ainsi que la constellation sont mentionnés pour retrouver plus facilement l'objet...

Il est préférable de choisir des étoiles suffisamment espacées ($> 25^\circ$) dans le ciel. Avant de commencer l'opération, Il faut s'assurer qu'elles sont observables depuis le lieu d'installation de l'antenne.

La procédure d'alignement se poursuit ainsi :

- Le programme liste chaque étoile en demandant à l'utilisateur s'il veut ou non utiliser l'objet pour aligner l'antenne
- Si la réponse est positive, le programme propose d'utiliser le clavier comme une raquette de télescope :



Les touches du clavier s'utilisent de la façon suivante :

- ✓ La touche 'q' permet de décrétement l'ascension droite de 5 minutes d'arc
- ✓ La touche 'd' permet d'incrémenter l'ascension droite de 5 minutes d'arc
- ✓ La touche 'z' permet d'incrémenter la déclinaison de 5 minutes d'arc
- ✓ La touche 'x' permet de décrétement la déclinaison de 5 minutes d'arc
- ✓ La touche 's' change de vitesse de rotation sur les deux axes. Si le mouvement de l'instrument est trop lent, appuyez une fois sur 's' pour multiplier les déplacements par 10x. En appuyant de nouveau sur 's', on revient à la vitesse initiale 1x

- ✓ La touche 'a' envoie un ordre ABORT à l'antenne et permet d'interrompre un mouvement dangereux
- ✓ Lorsque l'étoile est bien centrée dans le viseur, appuyez sur la touche ESC pour valider l'étoile...
- ✓ Au bout de la troisième étoile, on peut interrompre la procédure d'alignement en tapant sur la touche 'f '. L'antenne est maintenant calibrée...

Testez ensuite l'antenne en tapant par exemple **goto betelgeuse**

Utilisation de BAOcontrol:

BAOcontrol est une interface minimaliste entre l'utilisateur et le pilote Indi_BAO mais peut également être utilisé pour programmer et exécuter une série de mouvements. Il est également possible d'utiliser ce petit utilitaire dans des scripts.

Des commandes simples permettant d'interagir avec l'installation BAORadio : suivre une région du ciel, annuler un mouvement ou encore mettre les antennes en position de repos.

Avant toute utilisation de BAOcontrol, il convient de lancer le serveur Indi_BAO en tâche de fond. C'est le serveur indi_BAO qui va se charger de transmettre les ordres de l'utilisateur aux microcontrôleurs agissant sur les antennes. Il convient donc de le laisser fonctionner en permanence dans une fenêtre terminal.

Un script permet de lancer plus simplement BAOcontrol et le serveur :

./LancementBAOcontrol.sh

Le serveur démarre dans une fenêtre xterm et se présente de la façon suivante :

```
2010-11-05T23:59:27: startup: indiserver indi_BAO
2010-11-05T23:59:27: Driver indi_BAO: Initializing from BAO device...
2010-11-05T23:59:27: Driver indi_BAO: Driver Version: 2010-05-12
```

BAOcontrol doit indiquer les paramètres de configuration par défaut au démarrage du logiciel :

```
*****
*                BAORadio Control                *
* Laboratoire de l'Accélérateur Linéaire          *
*                                                  *
*                v0.2 20/06/2011                  *
*                franckrichard033@gmail.com        *
*****
```

Lecture du fichier de configuration 'params' :

```
latitude = 48:51:00
longitude = 2:19:59
```

```
serveur = localhost
port = 7624
```

```
pression = 1013
température = 10
```

```
mode suivi = tracking
delai transit = 100 sec
delai tracking = 5 sec
```

```
Chargement du catalogue d'étoiles : 200 étoiles chargées
La connexion a été établie avec le pilote indi_BAO.
Les coordonnées géographiques ont bien été envoyées au pilote indi_BAO
```

Tapez help pour obtenir la liste des commandes disponibles.

Ces paramètres sont lus depuis le fichier baocontrol_params situé dans le répertoire principal de l'utilisateur qui a ouvert une session sous Linux.

Ce fichier contient les paramètres suivant :

```
[connexion indi]
serveur=localhost
port=7624

[coordonnees geographiques]
latitude=48:51:00
longitude=2:19:59

[atmosphere]
pression=1013
temperature=10

[suivi]
mode=tracking
delai_transit=100
delai_tracking=5

[divers]
couleurs=1
```

On y trouve l'adresse IP de l'ordinateur faisant tourner le serveur indi (ici nous avons **localhost** ce qui signifie que BAOControl et indi_BAO partagent la même machine. Mais on peut imaginer de prendre le contrôle depuis Orsay de l'ordinateur connecté aux antennes et qui serait situé par exemple à Nançay via un port sécurisé SSH...). On peut également noter que le fichier contient les coordonnées géographiques du télescope (ici le bâtiment du LAL) et les délais entre deux actualisations dans les modes tracking et transit. La déviation des ondes radio par l'atmosphère est calculée en prenant en compte la pression et la température. Pour annuler le calcul de la réfraction, indiquez simplement pression=0.

Enfin, le paramètre couleurs (qui peut prendre les valeurs 1 ou 2) permet d'agir sur les couleurs utilisées par BAOcontrol pour afficher les messages dans la fenêtre terminal qui exécute le programme (si le fond du terminal est blanc, utilisez couleurs=1 sinon utilisez couleurs=2).

En cas de modifications, veuillez ne pas utiliser de lettres en capitale. Le fichier ne serait alors plus lisible par le programme...

Si le fichier params est valide, le programme doit afficher ces deux lignes de confirmation :

La connexion a été établie avec le pilote indi_BAO

Les coordonnées géographiques ont bien été envoyées au pilote indi_BAO

Dans le cas contraire, des messages d'erreur peuvent vous aider à résoudre les problèmes éventuels.

Les commandes

En tapant **help**, il est possible d'afficher les commandes disponibles :

```
connect :          Se connecte au pilote indi_BAO.
disconnect :      Se déconnecte du pilote indi_BAO.
goto AD Dec :     Pointe l'objet situé aux coordonnées apparentes AD Dec.
                  AD doit être au format xx:yy:zz et dec au format +/-xx:yy:zz
                  exemple goto 12:10:05 -05:10:11 permettra de pointer l'objet
                  situé aux coordonnées AD=12h10m5s et dec=-5°10'11''
goto AD Dec J2000 : Pointe l'objet situé aux coordonnées AD Dec dans le repère J2000.
                  Avant de pointer l'objet, le programme calcule la précession, la nutation et
                  l'aberration
                  pour ramener les coordonnées à l'écliptique et à l'équinoxe de l'observation.
goto nom_objet :  Interroge la base NED pour trouver les coordonnées de l'objet puis le pointe.
                  exemples: goto m 31, goto messier 1, goto ngc 175, goto ic 434 etc...
                  Dans ce mode, les coordonnées sont automatiquement rapportées
                  à l'écliptique et à l'équinoxe de l'observation.
                  Aussi, merci de ne pas faire suivre les coordonnées de l'indication J2000.
                  Goto sun permet de suivre le soleil.
search nom_objet : Interroge le serveur NED sur internet et retourne les coordonnées de l'objet.
align ip :        Lance la procédure d'alignement de l'antenne d'adresse ip.
reset ip :        Réinitialise les paramètres d'alignement de l'antenne d'adresse ip.
set transit on :   Active le mode de suivi transit.
set transit delay x : Fixe la durée entre deux actualisations à x seconde(s) dans le mode transit
set tracking on :   Active le mode de suivi tracking.
set tracking delay x : Fixe la durée entre deux actualisations à x seconde(s) dans le mode tracking
set alignment method x : Alignes les antennes en utilisant la méthode x (simple/affine/taki)
status :          Liste les paramètres du programme.
run filename :    Exécute le fichier de mouvements filename.
abort run :       Annule l'exécution du fichier de mouvements.
abort :           Annule le mouvement en cours.
park :           Place les antennes en position de repos.
updatetime :      Synchronise l'horloge du PC avec un serveur de temps.
exit :           Sortir de BAOControl.
```

- **connect :**
disconnect :
 permettent de se connecter /déconnecter au serveur indi_BAO. Habituellement, ces deux commandes sont inutiles puisque la connexion se fait automatiquement au moment du lancement de BAOcontrol. Cependant, la connexion avec le serveur peut s'interrompre à la suite de la déconnexion d'une antenne sur le réseau par exemple ou d'un autre incident.
- **goto AD Dec :**
 Usages : goto 01:00:20.5 +40:10:00.0
 Active le suivi et déplace les télescopes jusqu'aux coordonnées horaires apparentes AD et Dec. La valeur AD doit être de la forme xx:yy:zz.z et la valeur Dec doit respecter le format +/-xx:yy:zz.z
 Pour suivre un objet situé aux coordonnées 1h53m4s et +58°54'3", on tapera la commande **goto 01:53:04 +58:54:03**
 Il faut cependant toujours indiquer les secondes et les minutes même si elles sont nulles. Exemple 12h doit se noter 12:00:00 .
 Si tout ce passe bien, le programme doit confirmer l'envoi des nouvelles coordonnées au pilote indi.
 A noter : si l'objet est situé à moins de 30° au-dessus de l'horizon, BAOcontrol affichera un message d'erreur, les antennes ne pouvant suivre physiquement un objet situé si bas sur l'horizon...
- **Goto AD Dec J2000 :**
 Usages : goto 00:00:10.1 +20:10:10.0 J2000
 En faisant suivre les coordonnées horaires de l'indication **J2000**, on indique à BAOcontrol que les coordonnées indiquées se rapportent à l'époque J2000 (écliptique et équinoxe du 1 janvier 2000 à 0h TU). Avant de pointer l'objet, le

programme calcule la précession, la nutation et l'aberration pour ramener les coordonnées à l'écliptique et à l'équinoxe de l'observation.

- **goto nom_objet**

Usages : goto M31, goto ic434

Lorsque la commande goto est suivie du nom d'un objet, le programme consulte sa liste d'étoiles intégrées pour retrouver ses coordonnées ou interroge la base **NED** (nécessite une connexion internet) s'il s'agit d'un objet du ciel profond.

Les coordonnées étant exprimées dans le repère J2000, des calculs supplémentaires sont effectués pour rapporter les coordonnées à la date et à l'heure de l'observation.

Le programme envoie ensuite un ordre goto à toutes les antennes présentes sur le réseau afin de suivre l'objet.

On peut utiliser la commande sous cette forme : goto Abbel434 ou goto Abbel434, goto ngc 1101, goto messier 1 etc...

Puisque les coordonnées sont automatiquement rapportées à l'écliptique et à l'équinoxe de l'observation, il est inutile de faire suivre la commande goto du mot clé J2000, celui-ci étant implicite.

Enfin, il est possible de suivre le soleil en indiquant simplement, **goto sun**

La commande **goto** a besoin de l'utilitaire **curl** pour activer cette fonctionnalité. Vérifiez que **curl** est bien installé sur votre machine.

- **search nom_objet**

La commande search marche sur le même principe que la command goto mais se contente de retourner les coordonnées de l'objet spécifié sans suivre l'objet.

- **align ip**

Lance la procédure d'alignement de l'antenne située à l'adresse ip (cf Alignement des antennes).

- **reset ip**

Réinitialise les paramètres d'alignement de l'antenne d'adresse ip (cf Alignement des antennes).

- **Commande set**

La commande set permet de fixer certains paramètres du programme :

- **set transit on** : Active le mode de suivi transit.
- **set transit delay x** : Fixe la durée entre deux actualisations de la position dans le mode transit à x seconde(s).
(x est un nombre entier positif non nul)
- **set tracking on** : Active le mode de suivi tracking.
- **set tracking delay x** : Fixe la durée entre deux actualisations de la position dans le mode tracking à x seconde(s)
(x est un entier positif non nul)

- **set alignment method x** : Aligne les antennes en utilisant la méthode x (x est un mot clé qui peut prendre les valeurs SIMPLE, AFFINE ou TAKI)

- **status**
Liste les paramètres du programme.
- **run filename** :
Exécute le fichier de mouvements filename (cf Executer un fichier de mouvements).
- **abort run** :
Annule l'exécution du fichier de mouvements.
- **abort** :
Annule le mouvement en cours.
- **park** :
Place les antennes en position de repos.
- **updatetime** :
Synchronise l'horloge du PC avec un serveur de temps (nécessite une connexion internet)
- **exit** :
Sortir de BAOControl.

La commande run :

La commande run exécute une liste de mouvements préprogrammés présente dans un fichier qui est donné en paramètre (ex : **run filename**). Le fichier filename doit se trouver dans le répertoire de BAORadio.

Le fichier filename doit se présenter sous cette forme :

```
[objet 1]
date=05/11/2010
heure=23:12:00
duree=30
ad=12:25:00
de=88:00:00
[objet 2]
date=05/11/2010
heure=23:13:00
duree=30
ad=00:25:00
de=89:00:00
```

Chaque objet est décrit par une rubrique qui commence par une étiquette [objet n°x] suivie par une liste de paramètres.

Les lignes date et heure indiquent le début de l'observation de l'objet i (Attention, les heures doivent être exprimées en temps universel (TU) !). Les antennes se déplaceront à l'heure indiquée pour suivre l'objet sur une durée exprimée en secondes (ligne duree=...).

L'ascension droite (ligne ad=) et la déclinaison (ligne de=) de l'astre achèvent de décrire l'observation de l'objet.

Il faut que les dates et les durées d'observation des objets qui se suivent soient cohérentes entre elles. Dans notre exemple, une durée d'observation fixée à 200s de l'objet numéro 1 ne permettrait pas l'observation de l'objet suivant. Veuillez donc à bien vérifier l'enchaînement des objets dans votre le fichier filename...

Utilisation de la commande run en ligne de commande

Il peut être utile de lancer l'exécution d'une série de mouvements au démarrage de BAOControl, en particulier pour réaliser des scripts plus élaborés.

Pour cela, ajouter l'indicateur -r au moment de lancer le programme :

./BAOcontrol -r NomDuFichier

NomDeFichier étant un fichier de mouvements préprogrammés tel que décrit un peu plus haut...

Utilisation des messages et des fichiers log

Au moment du démarrage du programme, vous avez peut-être noté qu'une fenêtre s'ouvrait automatiquement :

Les cercles représentent les antennes (ici il y en a 100) présentes sur l'installation BAORadio.

Les cercles gris représentent les antennes non connectées et les cercles verts indiquent des antennes fonctionnant normalement. En cas d'erreur sur une antenne, le cercle correspondant devient rouge et un message également en rouge apparaît dans la liste des derniers messages envoyés par les antennes.

Les antennes apparaissent dans la liste aléatoirement en fonction de l'ordre des connexions au serveur indi_BAO. Ici, la première antenne connectée était l'antenne portant l'IP 192.168.0.57

Pour connaître l'adresse IP associée à un cercle (et donc identifier l'antenne physiquement), cliquez sur l'un d'entre eux...

Tous les échanges entre BAOControl, le pilote indi_BAO et les antennes sont consignés dans un fichier portant le nom BAOControl.log .

On peut le consulter en ouvrant un éditeur texte (kwrite par exemple).

Documentation technique

Class BAOcontrol : liste des méthodes employées :

```
BAOcontrol::BAOcontrol()
```

Constructeur de la classe BAOcontrol.

Initialisation des variables globales de la classe, des paramètres des antennes. Chargement du fichier de configuration de BAOcontrol, du catalogue d'étoiles nécessaires à la calibration des antennes...

```
void BAOcontrol::initialiserFenetre()
```

Initialise la fenêtre graphique de BAOcontrol.

```
void BAOcontrol::Dessiner()
```

Dessine les éléments dans la fenêtre graphique.

Utilise XCopyArea pour éviter le scintillement de la fenêtre graphique lors de l'actualisation des éléments graphiques.

```
void BAOcontrol::LireReponse()
```

Récupère les messages envoyés par le serveur indi_BAO par la redirection :

*client_socket >> reponse;

La variable reponse doit contenir la chaîne « message= ». La partie que l'on doit traiter suit ce mot clé.

En cas de présence des chaînes « ALERTE antenne » ou « Erreur sur l antenne » dans la chaîne réponse, on change le statut de l'antenne émettrice et on affiche le message d'erreur.

```
bool BAOcontrol::VerifReponse  
(  
    string reponseattendue  
)
```

Récupère le dernier message envoyé par le serveur indi_BAO et retourne true si ce message contient la chaîne spécifiée dans la variable reponseattendue.

Si le message n'est pas conforme à la réponse souhaitée, on affiche le message reçu dans un message d'erreur.

On consulte le buffer contenant la réponse du serveur toutes les millisecondes. Si la réponse ne vient pas au bout de 20 millisecondes, on sort de la fonction avec la réponse false.

```
Void BAOcontrol::UpdateTime()
```

Récupère la date et l'heure du système et envoie ces informations à la classe Astro. De là, on lance la méthode CalculTSL qui évalue le temps sidéral local, le jour julien et la longitude de la terre dans le repère héliocentrique.

```
Void BAOcontrol::my_thread_process ()
```

Le thread de l'application gère :

- l'exécution d'une liste de mouvements programmés
- l'actualisation de la fenêtre graphique
- la récupération des messages du pilote indi_BAO
- La sauvegarde des paramètres d'alignement des antennes (qui n'est effectuée que si une seconde vient juste de basculer...)

Le thread contient une boucle while dont on ne sort que si la variable globale NoExit vaut false...

```
bool BAOcontrol::DecompositionCommande  
(  
    string chaîne,  
    string commande,  
    string *chaine1,  
    string *chaine2  
)
```

Décrypte les commandes saisies par l'utilisateur

Exemples :

DecompositionCommande("goto 10:10:10 25:10:10", "goto", &chaine1, &chaine2)

retourne true avec chaine1 = "10:10:10", chaine2 = "25:10:10"

DecompositionCommande("goto messier 1","goto", &chaine1, NULL) (avec chaine2=NULL)

retourne true avec chaine1 = "messier 1", chaine2 = NULL

Dans cette deuxième situation, on observe que « messier 1 » qui contient pourtant un caractère espace se retrouve exclusivement dans chaine1.

```
bool BAOcontrol::EnvoyerCoordGeographiques()
```

Nous entrons ici dans l'une des particularités d'Indi : dans la philosophie d'Indi, chaque instrument poste ou répond à des vecteurs de données exprimés dans le langage xml.

Nous en avons un exemple dans la méthode EnvoyerCoordGeographiques qui transmet les coordonnées géographiques du lieu d'observation au serveur Indi_BAO en utilisant le bloc de code :

```
*client_socket << "<newNumberVector device=\"BAO\"  
name=\"GEOGRAPHIC_COORD\">";  
*client_socket << "<oneNumber name=\"LAT\">";  
*client_socket << LatitudeChar;  
*client_socket << "</oneNumber>";  
*client_socket << "<oneNumber name=\"LONG\">";  
*client_socket << LongitudeChar;  
*client_socket << "</oneNumber>";  
*client_socket << "</newNumberVector>";
```

Les deux nombres LAT et LONG appartenant au vecteur GEOGRAPHIC_COORD sont envoyés au pilote indi_BAO qui va recevoir et traiter ces informations par l'intermédiaire de la fonction

```
void BAO::ISNewNumber (const char *dev, const char *name, double values[],  
char *names[], int n) ;
```

(voir le détail de la procédure dans la partie consacrée à la classe BAO)

Les procédures suivantes fonctionnent selon le même principe :

```
bool BAOcontrol::EnvoyerPressionTemperature()  
bool BAOcontrol::EnvoyerDelaiesModesTransitEtTracking()  
bool BAOcontrol::EnvoyerMethodeAlignement()  
bool BAOcontrol::Park()  
bool BAOcontrol::Abort()  
bool BAOcontrol::Connect(bool connect)
```

```
bool BAOcontrol::Goto(  
string ar,          // Chaîne contenant l'ascension droite de l'objet visé  
string dec,         // chaîne contenant la déclinaison " " "  
bool Transit,       // = true si le mode transit est activé  
bool J2000          // les coordonnées horaires sont-elles dans le repère  
                    // J2000 ?  
)
```

Dirige l'antenne vers les coordonnées ar et dec et suit l'objet en activant le mode transit (si true) ou tracking.

Si la variable si J2000 vaut true, cela signifie que les coordonnées ar et dec sont données dans le système de coordonnées J2000 (écliptique et équinoxe du 1 janvier 2000 à 0 h TU).

Des calculs supplémentaires (précession, nutation, aberration) sont alors réalisés pour ramener les coordonnées ad et dec à l'époque de l'observation. (Voir le détail dans la classe Astro.)

```
void mode_raw(  
int activer  
)
```

Gestion du clavier en mode raw pour la procédure d'alignement.

Cela permet de ne pas à avoir à taper sur la touche 'entrée' entre chaque pression sur les touches du clavier.

```
bool BAOcontrol::AlignementAntenneIP( string ip )
```

Le principe consiste à utiliser des étoiles comme point de repère pour la calibration des antennes : on compare la position calculée de chaque objet avec la position effectivement mesurée lors de la procédure d'alignement. On en tire alors une matrice de rotation qui doit corriger les défauts d'alignement de l'antenne (le 0 pas codeur pas parfaitement dirigé vers le sud et un axe de rotation az pas forcément normal au sol...).

```
bool BAOcontrol::Decomposition(  
string chaîne,      // chaîne à traiter  
char type,          // type, entier entre 0 et 3  
float *a1,          // Valeurs de retour  
float *a2,  
float *a3)
```

Décomposition et vérification des dates, heures, AD et déclinaisons

La variable type indique le type d'objet à traiter :

Si type = 0 -> la chaîne contient une déclinaison ou une latitude

type = 1 -> longitude

type = 2 -> AD ou heure

type = 3 -> date

Exemple de Décomposition("15:23:12", 2, a, b, c) retourne true avec a=15, b=23, c=12
Si la chaîne a un format incorrect, la fonction retourne false.

```
void BAOcontrol::DecodageEntreesUtilisateur(string chaîne)
```

Identification des commandes entrées par l'utilisateur puis exécution des commandes...

```
bool BAOcontrol::Run(  
    string fichier)
```

Lance l'exécution d'un fichier de mouvements.

```
CoordonneesHoraires BAOcontrol::ServeurNED(  
    string objet)
```

Interroge le serveur NED sur internet et retourne les coordonnées dans le repère J2000. Si il n'y a pas de réponse du serveur, un message d'erreur est affiché et les coordonnées retournées sont égales à AD="" dec=""
Utilise l'utilitaire curl pour interroger le serveur...

```
bool BAOcontrol::ChargementParametres(  
    string fileName  
)
```

Chargement des paramètres contenus dans le fichier params. Utilise la classe filetools (fichier .cpp et.h présents dans le fichier communs. Cependant cette classe que j'ai récupérée sur Internet est très lente et nécessitera certainement une réécriture).

```
bool BAOcontrol::LectureFichierMouvements(  
    string fileName  
)
```

Lecture des mouvements planifiés contenus dans le fichier fileName
Ces fichiers de mouvement planifiés permettent des observations automatisées d'une liste d'objets (ou zone du ciel) pour lesquels on passe de l'un à l'autre en fonction de l'heure.

```
bool BAOcontrol::ChargementCatalogueEtoiles(  
    string fileName)
```

Chargement du catalogue d'étoiles nécessaire à la procédure d'alignement des antennes.

```
int BAOcontrol::init(  
    int argc,  
    char **argv)
```

Routine principale de la classe BAOcontrol

Classe BAO : liste des méthodes employées :

```
void BAO::InitAntennes()
```

Initialise les paramètres des antennes.

```
void BAO::init_properties()
```

Initialise les boutons et des zones d'affichage dans la boîte de dialogue INDI apparaissant dans KStars en particulier...
(se référer à la documentation d'INDI)

```
void BAO::ISGetProperties(  
const char *dev)
```

Initialisation de la boîte de dialogue INDI (suite).
Cette procédure doit être présente dans les tous pilotes Indi.

```
void BAO::reset_all_properties()
```

Initialisation des vecteurs de la boîte Indi.
Cette procédure doit être présente dans tous les pilotes Indi.

```
void BAO::ISNewText(  
const char *dev,  
const char *name,  
char *texts[],  
char *names[],  
int n  
)
```

En cas de changement de texte dans la boîte de dialogue (par exemple : changement du nom de l'objet à suivre) alors effectuer le goto correspondant...
Cette fonction n'est pas utilisée en raison de la redondance avec les fonctions déjà présentes dans KStars et BAOcontrol.

```
void BAO::ISNewNumber(  
const char *dev,  
const char *name,  
double values[],  
char *names[],  
int n)
```

En cas de changement d'une valeur numérique dans la boîte de dialogue Indi
Exemple : longitude, latitude, ar, dec etc...) -> prendre en compte les modifications.

const char *dev contient le nom du dispositif Indi recevant le message (ici indi_BAO)
const char *name reçoit le nom de la rubrique modifiée par l'utilisateur
(ex : les coordonnées géographiques, les coordonnées horaires de l'objet etc...)
double values[] contient la liste de tous les champs modifiables dans chaque rubrique.
(ex : longitude et latitude dans la rubrique coordonnées géographiques.)
char *values[] contient toutes les valeurs (numériques ou non) mais toujours exprimées sous la forme d'une chaîne de caractères.

```
void BAO::ISNewSwitch(  
const char *dev,  
const char *name,  
ISState *states,  
char *names[],  
int n)
```

L'utilisateur clique sur l'un des boutons de la boîte Indi.
Même observation que pour la procédure précédente.

```
void *BAO::pThreadSocket()
```

Gestion du thread : permet de suivre la connexion/déconnexion des antennes toutes les secondes.

L'utilisation d'un thread permet de contourner le problème de la fonction accept qui est bloquante.

L'adresse IP de l'antenne qui a effectué la connexion est consignée dans la variable IP de la structure Sockets. L'état de l'antenne (connected) est placé à true.

```
bool BAO::ExtractPosition(  
string str,  
Position *result)
```

Extraction de la position de l'antenne après l'envoi de la commande **P**

Le retour de la commande **P** est **POSITION/valeur_az/valeur_alt/**

Ce retour est envoyé dans la chaîne str

ExtractPosition retourne une structure Position contenant Valeur_az et Valeur_alt

Ex: **ExtractPosition("POSITION/0100/0001/", result)** retourne true avec result.x=100 et result.y=1

```
void BAO::ADDEC2Motor(  
double newRA,  
double newDEC)
```

Cette procédure effectue les conversions en unités codeurs des paraboles après avoir appliqué la matrice de correction aux coordonnées de l'objet visées par l'utilisateur.

```
int BAO::AntennesConnectees()
```

Retourne le nombre d'antennes actuellement connectées.

```
void BAO::DeconnecterSocket(int num)
```

déconnecte l'antenne utilisant le socket num (en cas de détection d'une anomalie la concernant par exemple).

```
void BAO::ISPoll()
```

Procédure principale de la classe BAO. Elle est appelée toutes les ms.

L'exécution complète d'un goto s'effectue en effectuant les étapes suivantes :

1ère étape : envoi de la commande POSITION à toutes les antennes

2ème étape : Vérification de l'acknowledge de la commande POSITION pour chaque antenne.

3ème étape : Les positions retournées par les antennes sont-elles valides ?

4ème étape : Si oui, envoi de la commande goto à toutes les antennes.

5ème étape : est-ce que toutes les antennes ont envoyé l'acknowledge de la commande goto ?

6ème étape : les antennes ont toutes répondu GOTO OK !

Remarque concernant la réalisation de l'étape 3 :

On place cette partie du traitement en dehors du switch et de la boucle de traitement afin d'attendre que toutes les antennes soient rendues à l'étape 3 et soient donc en mesure d'effectuer le goto simultanément -> meilleure synchronisation

```
bool BAO::process_coords()
```

Passe en mode transit ou tracking

```
void BAO::connect_telescope()
```

Connexion / Déconnexion du télescope

```
bool BAO::COMMANDE(  
int numsocket,  
char* Commande,  
char* Params)
```

Envoie une commande sur le socket numsocket.

```
bool BAO::GOTO(  
int numsocket,  
int deltaAz,  
int deltaAlt)
```

Envoie l'ordre Goto à l'antenne occupant le socket numsocket.

Class Alignement, liste des méthodes employées :

```
void Alignement::InitAlignement()
```

Initialisations des vecteurs et des paramètres d'alignement.

```
void Alignement::TransmettreParametresClasseAstro(double Annee, double  
Mois, double Jour, double Heu, double Min, double Sec, double Longitude,  
double Latitude, double Pression, double Temp)
```

Initialisation des paramètres de la classe astro.

```
void Alignement::Identity()
```

```
1 0 0  
0 1 0  
0 0 1
```

```
void Alignement::RotationAutourDunAxe(  
double t,  
double r,  
double alpha)
```

Calcule la matrice de rotation d'un angle alpha autour d'un axe défini par les paramètres t et r dans un système de coordonnées sphériques.
Fonction utile pour de débogage de la fct d'alignement

```
void Alignement::Calculer_Matrice_LMN(  
Coord p1,  
Coord p2,  
Coord p3)
```

Matrice intermédiaire nécessaire au calcul de la matrice de correction dans le mode d'alignement TAKI. Consultez la rubrique consacrée à l'alignement pour comprendre le principe de la correction AFFINE/TAKI.

```
int Alignement::Calculer_Matrice_Taki(  
double x,  
double y,  
Coord a1,  
Coord a2,  
Coord a3,  
Coord m1,  
Coord m2,  
Coord m3)
```

Calcule la matrice de correction dans le mode d'alignement TAKI.

```
void Alignement::AppliquerMatriceCorrectionTaki(Coord * result, Coord vect
```

Applique la matrice MATRICE au vecteur vect

```
void Alignement::Calculer_Matrice_GETPQ(  
Coord p1,  
Coord p2,  
Coord p3)
```

Matrice intermédiaire nécessaire au calcul de la matrice de correction dans le mode d'alignement AFFINE

```
int Alignement::Calculer_Matrice_Affine( double x, double y, Coord a1,  
Coord a2, Coord a3, Coord m1, Coord m2, Coord m3)
```

Calcule la matrice de correction dans le mode d'alignement AFFINE.

```
void Alignement::AppliquerMatriceCorrectionAffine(  
Coord * result,  
Coord ob)
```

Calcule la quantité CoordObjets * MATRICECorrection + VECT_offset

```
int Alignement::Calculer_Matrice_Simple( Coord a1, Coord a2, Coord a3,  
Coord m1, Coord m2, Coord m3)
```

Calcule la matrice de correction issue de la procédure d'alignement des antennes

Cette méthode qui utilise trois points de correction seulement est particulièrement simple mais repose sur la qualité des pointages effectués lors de l'alignement sur les trois étoiles et ne prend en compte la flexion de la monture etc...

Les variables a1, a2, a3 contiennent les coordonnées horaires calculées (AD & Dec) des trois étoiles visées...

Les variables m1, m2, m3 contiennent les coordonnées horaires effectivement mesurées de ces trois objets

```
double Alignement::Surface_Triangle(double px1, double py1, double px2,
double py2, double px3, double py3)
```

Calcule la surface d'un triangle de coordonnées (xa,ya) (xb, yb) (xc, yc)

La formule utilisée est :

$$S = \frac{1}{2} \left| \det \begin{pmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{pmatrix} \right| = \frac{1}{2} |x_A y_C - x_A y_B + x_B y_A - x_B y_C + x_C y_B - x_C y_A|.$$

```
bool Alignement::PointSitueDansSurfaceTriangle(double px, double py,
double px1, double py1, double px2, double py2, double px3, double py3)
```

Est-ce que le point de coordonnées (px, py) se trouve dans la surface du triangle de coordonnées (px1,py1) (px2, py2) (px3, py3) ?

```
void Alignement::CalculerMatriceCorrection(
double ar,
double dec)
```

Calcule la matrice de correction issue de la procédure d'alignement des antennes.

Routine principale. On construit ici trois vecteurs c1, c2 et c3 qui contiennent les coordonnées (issues du catalogue) de trois objets visés pendant la procédure d'alignement. Les vecteurs m1, m2 et m3 contiennent les coordonnées mesurées de ces mêmes objets dans le viseur de l'instrument.

```
bool Alignement::ChargementParametresAlignement(  
    string IP,  
    string fileName,  
    double a,  
    double b)
```

Chargement des paramètres d'alignement de l'antenne d'adresse ip IP et visant actuellement le point de coordonnées horaires a et b.

```
bool Alignement::EnregistrementParametresAlignement(  
    string IP,  
    string fileName  
)
```

Sauvegarde des paramètres d'alignement de l'antenne d'adresse IP dans le fichier de configuration fileName.

Class Astro, liste des méthodes employées :

```
void Astro::DefinirDateHeure(double A, double M, double J, double H,  
double Mi, double S)
```

Initialisation des variables globales de la classe Astro

```
double Astro::VerifAngle(double Angle)
```

Paramètre : angle en radians
retourne l'angle dans un intervalle $0 \leq \text{angle} \leq 2\pi$

```
double Astro::VerifDistance(double Angle)
```

Paramètre : angle en radian
retourne angle dans un intervalle $0 \leq \text{angle} \leq \pi$

```
double Astro::DistanceAngulaireEntre2Points(double az1, double ha1, double  
az2, double ha2)
```

Distance entre deux points situés sur une sphère

```
double Astro::Arrondi(double a)
```

retourne l'arrondi du nombre donné en paramètre
Exemples: Arrondi(5.7) = 6 Arrondi(-5.7) = -6

```
string Astro::DHMS(double mema, bool HMS)
```

convertit un angle (exprimé en degrés) dans le format hh:mm:ss.s ou deg:mm:ss.s
selon la valeur du booléen HMS (format hh:mm:ss.s si HMS = true)

```
double Astro::CalculLongitudeSoleil()
```

Calcul de la longitude du Soleil
La quantité PS n'est pas indispensable. Elle permet d'améliorer la précision

```
void Astro::CalculARDecSoleil(CoordonneesHoraires *Soleil)
```

Calcule l'AD et la déclinaison du Soleil. C'est indispensable pour exécuter la commande "goto sun"

```
void Astro::Nutation()
```

Calcul de la nutation en longitude et en inclinaison
Indispensable pour calculer la position apparente d'un objet (étoile, galaxie etc...) à partir des coordonnées horaires données dans le repère J2000.

```
void Astro::NutationEtoile(double *ar, double *de)
```

Nutation des étoiles. Modifie les coordonnées horaires données en paramètres.
ar et de sont exprimées en radians

```
void Astro::Precession(double *ar, double *de)
```

Précession des équinoxes. Modifie les coordonnées horaires données en paramètres.
ar et de sont exprimés en radians

```
void Astro::Obliquite(double JJ)
```

Calcule l'inclinaison de l'axe terrestre par rapport au plan de l'écliptique.
Quantité indispensable pour calculer le temps sidéral local et les coordonnées horaires du soleil.


```
void Astro::AberrationAnnuelle(double *ar, double *de)
```

Aberration annuelle des étoiles, ar et de sont exprimés en radians

```
double Astro::CalculJJ(double A, double M, double J, double Heure)
```

Calcule le jour julien en fonction des paramètres : Année, Mois, Jour, et Heure

```
void Astro::CalculTSL()
```

Calcule le jour julien, temps sidéral local, la longitude du soleil.

```
void Astro::Azimut(  
double Ar,    //Ascension droite de l'objet  
double De,    //Déclinaison de l'objet  
double *azi,  //Azimut de l'objet  
double *hau   // hauteur de l'objet  
)
```

Calcule la hauteur et l'azimut d'un objet en fct du lieu d'observation et de l'ascension droite et de la déclinaison.

```
double Astro::RefractionAtmospherique(double ht)
```

Réfraction atmosphérique : formule simple de Jean Meeus

la hauteur ht est exprimée en radians. Calcule les effets de la réfraction atmosphérique sur la hauteur d'un objet donné en paramètre... La fct retourne la hauteur corrigée.

```
void Astro::slaRefro ( double zobs, double hm, double tdk, double pmb,  
                      double rh, double wl, double phi, double tlr,  
                      double eps, double *ref )
```

```
-----  
slaRefro  
-----
```

Atmospheric refraction for radio and optical wavelengths.

Given:

zobs double observed zenith distance of the source (radian)
hm double height of the observer above sea level (metre)
tdk double ambient temperature at the observer (deg K)
pmb double pressure at the observer (millibar)
rh double relative humidity at the observer (range 0-1)
wl double effective wavelength of the source (micrometre)
phi double latitude of the observer (radian, astronomical)
tlr double temperature lapse rate in the troposphere (degK/met)
eps double precision required to terminate iteration (radian)

Returned:

ref double refraction: in vacuo ZD minus observed ZD (radian)

Notes:

- 1 A suggested value for the tlr argument is 0.0065D0. The refraction is significantly affected by tlr, and if studies of the local atmosphere have been carried out a better tlr value may be available.
- 2 A suggested value for the eps argument is 1e-8. The result is usually at least two orders of magnitude more computationally precise than the supplied eps value.
- 3 The routine computes the refraction for zenith distances up to and a little beyond 90 deg using the method of Hohenkerk and Sinclair (NAO Technical Notes 59 and 63, subsequently adopted in the Explanatory Supplement, 1992 edition - see section 3.281).
- 4 The C code is an adaptation of the Fortran optical refraction subroutine AREF of C.Hohenkerk (HMNAO, September 1984), with extensions to support the radio case. The following modifications to the original HMNAO optical refraction algorithm have been made:
 - . The angle arguments have been changed to radians.
 - . Any value of zobs is allowed (see note 6, below).
 - . Other argument values have been limited to safe values.
 - . Murray's values for the gas constants have been used (Vectorial Astrometry, Adam Hilger, 1983).
 - . The numerical integration phase has been rearranged for extra clarity.
 - . A better model for Ps(T) has been adopted (taken from

Gill, Atmosphere-Ocean Dynamics, Academic Press, 1982).

- . More accurate expressions for Pwo have been adopted (again from Gill 1982).
- . Provision for radio wavelengths has been added using expressions devised by A.T.Sinclair, RGO (private communication 1989), based on the Essen & Froome refractivity formula adopted in Resolution 1 of the 13th International Geodesy Association General Assembly (Bulletin Geodesique 1963 p390).
- . Various small changes have been made to gain speed.

None of the changes significantly affects the optical results with respect to the algorithm given in the 1992 Explanatory Supplement. For example, at 70 deg zenith distance the present routine agrees with the ES algorithm to better than 0.05 arcsec for any reasonable combination of parameters. However, the improved water-vapour expressions do make a significant difference in the radio band, at 70 deg zenith distance reaching almost 4 arcsec for a hot, humid, low-altitude site during a period of low pressure.

- 5 The radio refraction is chosen by specifying $wl > 100$ micrometres. Because the algorithm takes no account of the ionosphere, the accuracy deteriorates at low frequencies, below about 30 MHz.
- 6 Before use, the value of zobs is expressed in the range $\pm \pi$. If this ranged zobs is -ve, the result ref is computed from its absolute value before being made -ve to match. In addition, if it has an absolute value greater than 93 deg, a fixed ref value equal to the result for zobs = 93 deg is returned, appropriately signed.
- 7 As in the original Hohenkerk and Sinclair algorithm, fixed values of the water vapour polytrope exponent, the height of the tropopause, and the height at which refraction is negligible are used.
- 8 The radio refraction has been tested against work done by Iain Coulson, JACH, (private communication 1995) for the James Clerk Maxwell Telescope, Mauna Kea. For typical conditions, agreement at the 0.1 arcsec level is achieved for moderate ZD, worsening to perhaps 0.5-1.0 arcsec at ZD 80 deg. At hot and humid sea-level sites the accuracy will not be as good.
- 9 It should be noted that the relative humidity rh is formally defined in terms of "mixing ratio" rather than pressures or densities as is often stated. It is the mass of water per unit mass of dry air divided by that for saturated air at the same

temperature and pressure (see Gill 1982).

Called: slaDrange, atmt, atms

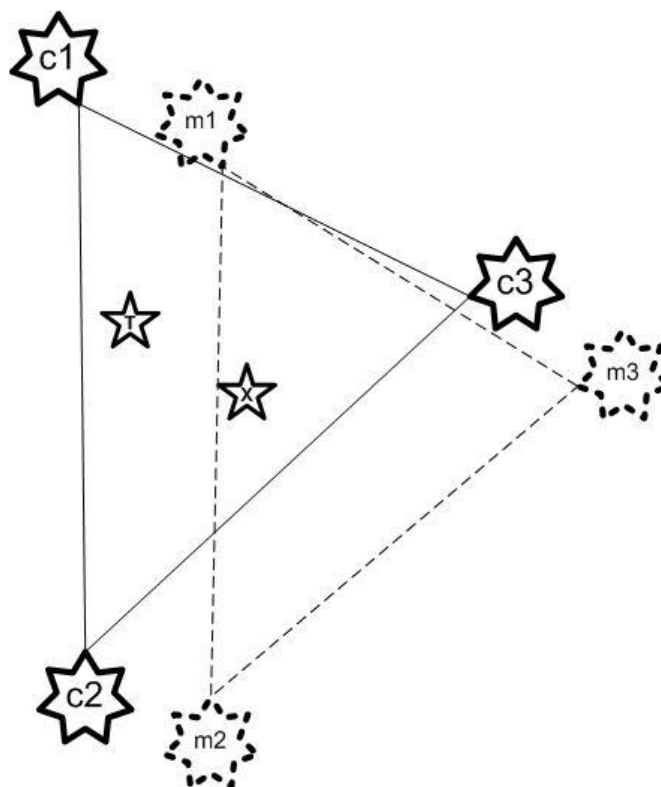
Defined in slamac.h: TRUE, FALSE

Last revision: 30 January 1997

Copyright P.T.Wallace. All rights reserved.

Méthodes d'alignement

Trois méthodes d'alignement sont implantées dans le pilote indi_BAO. Elles reposent sur le même principe : calculer la position de trois objets puis mesurer les positions réellement observées par l'instrument.



Sur ce schéma, c1 c2 et c3 représentent trois objets calculés pour la calibration. Les positions réellement mesurées par l'antenne sont en m1, m2 et m3.

Pour un objet quelconque T, on applique alors la matrice calculée pour passer des point c aux points m.

1ère méthode : méthode SIMPLE

Cette méthode propose un modèle de pointage calculé à partir de trois étoiles visées successivement puis dont le pointage a été affiné par l'utilisateur.

On calcule d'abord l'azimut θ et la hauteur δ à partir de l'ascension droite α et la déclinaison β de l'objet que l'on place à la surface d'une sphère de rayon 1.

On passe des coordonnées sphériques aux coordonnées cartésiennes en utilisant la transformation bien connue :

$$x = \cos \theta \sin \delta$$

$$y = \cos \theta \cos \delta$$

$$z = \sin \delta$$

soient les vecteurs $\vec{a}_1 = \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix}$, $\vec{a}_2 = \begin{pmatrix} a_{21} \\ a_{22} \\ a_{23} \end{pmatrix}$, $\vec{a}_3 = \begin{pmatrix} a_{31} \\ a_{32} \\ a_{33} \end{pmatrix}$ représentant les coordonnées cartésiennes des trois étoiles sélectionnées par l'utilisateur pour réaliser la procédure d'alignement.

Soient les vecteurs $\vec{n}_1 = \begin{pmatrix} n_{11} \\ n_{12} \\ n_{13} \end{pmatrix}$, $\vec{n}_2 = \begin{pmatrix} n_{21} \\ n_{22} \\ n_{23} \end{pmatrix}$, $\vec{n}_3 = \begin{pmatrix} n_{31} \\ n_{32} \\ n_{33} \end{pmatrix}$ contenant les coordonnées cartésiennes des étoiles mesurées.

La matrice M doit vérifier les relations :

$$\begin{pmatrix} n_{11} \\ n_{12} \\ n_{13} \end{pmatrix} = M \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix}, \quad \begin{pmatrix} n_{21} \\ n_{22} \\ n_{23} \end{pmatrix} = M \begin{pmatrix} a_{21} \\ a_{22} \\ a_{23} \end{pmatrix}, \quad \begin{pmatrix} n_{31} \\ n_{32} \\ n_{33} \end{pmatrix} = M \begin{pmatrix} a_{31} \\ a_{32} \\ a_{33} \end{pmatrix}$$

$$\text{avec } M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

On trouve finalement :

$$m_{11} = -\frac{-a_{23}a_{32}n_{11} + a_{22}a_{33}n_{11} + a_{13}a_{32}n_{21} - a_{12}a_{33}n_{21} - a_{13}a_{22}n_{31} + a_{12}a_{23}n_{31}}{d}$$

$$m_{12} = -\frac{a_{23}a_{31}n_{11} - a_{21}a_{33}n_{11} - a_{13}a_{31}n_{21} + a_{11}a_{33}n_{21} + a_{13}a_{21}n_{31} - a_{11}a_{23}n_{31}}{d}$$

$$m_{13} = -\frac{-a_{22}a_{31}n_{11} + a_{21}a_{32}n_{11} + a_{12}a_{31}n_{21} - a_{11}a_{32}n_{21} - a_{12}a_{21}n_{31} + a_{11}a_{22}n_{31}}{d}$$

$$m_{21} = -\frac{-a_{23}a_{32}n_{12} + a_{22}a_{33}n_{12} + a_{13}a_{32}n_{22} - a_{12}a_{33}n_{22} - a_{13}a_{22}n_{32} + a_{12}a_{23}n_{32}}{d}$$

$$m_{22} = -\frac{a_{23}a_{31}n_{12} - a_{21}a_{33}n_{12} - a_{13}a_{31}n_{22} + a_{11}a_{33}n_{22} + a_{13}a_{21}n_{32} - a_{11}a_{23}n_{32}}{d}$$

$$m_{23} = -\frac{-a_{22}a_{31}n_{12} + a_{21}a_{32}n_{12} + a_{12}a_{31}n_{22} - a_{11}a_{32}n_{22} - a_{12}a_{21}n_{32} + a_{11}a_{22}n_{32}}{d}$$

$$m_{31} = - \frac{-a_{23}a_{32}n_{13} + a_{22}a_{33}n_{13} + a_{13}a_{32}n_{23} - a_{12}a_{33}n_{23} - a_{13}a_{22}n_{33} + a_{12}a_{23}n_{33}}{d}$$

$$m_{32} = - \frac{a_{23}a_{31}n_{13} - a_{21}a_{33}n_{13} - a_{13}a_{31}n_{23} + a_{11}a_{33}n_{23} + a_{13}a_{21}n_{33} - a_{11}a_{23}n_{33}}{d}$$

$$m_{33} = - \frac{-a_{22}a_{31}n_{13} + a_{21}a_{32}n_{13} + a_{12}a_{31}n_{23} - a_{11}a_{32}n_{23} - a_{12}a_{21}n_{33} + a_{11}a_{22}n_{33}}{d}$$

avec $d = a_{13}a_{22}a_{31} - a_{12}a_{23}a_{31} - a_{13}a_{21}a_{32} + a_{11}a_{23}a_{32} + a_{12}a_{21}a_{33} - a_{11}a_{22}a_{33}$

Une fois que l'on a obtenu les nouvelles coordonnées les nouvelles coordonnées d'un objet visé après avoir appliqué :

$$\begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = M \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

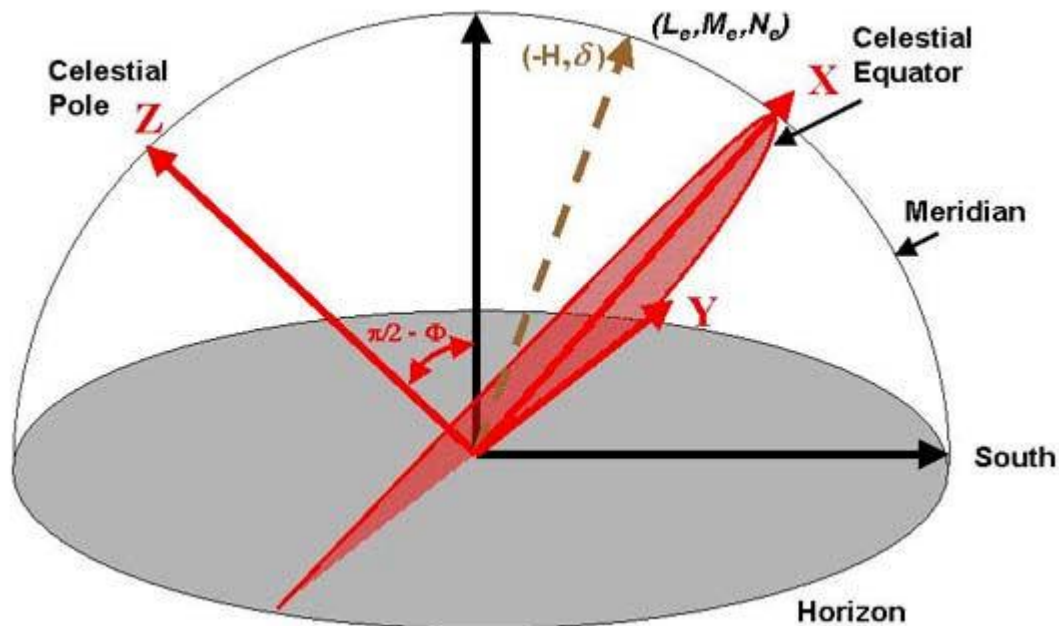
On revient aux coordonnées sphériques (et donc horizontales : l'azimut α et la hauteur β) en appliquant les relations :

$$\alpha = \tan^{-1}\left(\frac{n_2}{n_1}\right) \text{ avec } \alpha = \alpha + \pi \text{ si } n_1 < 0$$

$$\beta = \sin^{-1} n_3$$

2ème et 3ème méthodes : Méthode AFFINE et TAKI

Ces méthodes sont expliquées dans un article Toshimi Taki. Elles sont conçues pour aligner un télescope équatorial mais peuvent également permettre de corriger les défauts d'alignement d'un télescope sur monture alt-az ;



La figure ci-dessus représente un télescope aligné sur l'axe polaire et dans la direction de visée pointe un objet de coordonnées avec les coordonnées équatoriales $(-H, \delta)$. Sur une sphère unité, ces coordonnées peuvent être exprimées par un vecteur directeur (L_e, M_e, N_e) .

$$\begin{pmatrix} L_e \\ M_e \\ N_e \end{pmatrix} = \begin{pmatrix} \cos \delta \cos(-H) \\ \cos \delta \sin(-H) \\ \sin \delta \end{pmatrix}$$

Où H = angle horaire = Temps sidéral local – ascension droite

Dans ce repère de référence, l'axe Z est dirigé vers l'axe polaire

L'axe des Y est dirigé vers l'Est

L'axe des X est dirigé vers le Sud

Pour une monture présentant un défaut d'alignement, ce vecteur devient:

$$\begin{pmatrix} L_e' \\ M_e' \\ N_e' \end{pmatrix} = \begin{pmatrix} \cos \delta' \cos(-H') \\ \cos \delta' \sin(-H') \\ \sin \delta' \end{pmatrix}$$

Où

$$\delta' = \delta + \Delta$$

$$H' = H - h$$

Avec :

Δ = offset en déclinaison du au défaut d'alignement de la monture

h = offset en ascension droite du au défaut d'alignement de la monture

Fichier type produit par la procédure de calibration :

```
[Alignement antenne ip x.x.x.1]
delta_az_polar=0
align=0
alignment_method=3
ad 1=3.88645
de 1=1.29328
delta_ad 1=701
delta_de 1=234
ts1 1=2.0774
ad 2=2.89122
de 2=0.982848
delta_ad 2=-80
delta_de 2=173
ts1 2=2.08377
ad 3=3.21169
de 3=0.994143
delta_ad 3=23
delta_de 3=224
ts1 3=2.09165
```

ad x et de x contiennent les coordonnées horaires de l'objet x ayant servi à la calibration. Delta_ad contient le nombre de fois où la quantité PasDeltaAD est ajoutée à ad x. Delta_de contient le nombre de fois où la quantité PasDeltaDec est ajoutée à dec x.

Fichier const.h contenant les paramètres du programme

```
////////////////////////////////////////
// Paramètres utilisés dans le logiciel //
////////////////////////////////////////

// dimensions des tableaux

#define MAXLOG 100000
#define MAXOBJETS 1000
#define MAXETOILES 250
#define MAXANTENNES 50
#define MAXALIGNEMENTANTENNE 40
#define TAILLEMAXLOGS 3000000

// Taille maximale des chaines de caractères

#define MAXCARACTERES 1024

// nombre de pas sur l'axe azimuth

#define NBREPASCODEURSAZ 4000

// Hauteur minimale au-dessus de laquelle un objet devient accessible à l'antenne

#define HAUTMIN 30.0

////////////////////////////////////////
// Définition des méthodes d'alignement //
////////////////////////////////////////

#define NONE 0
#define SIMPLE 1
#define AFFINE 2
#define TAKI 3

// Magnitude maximale des étoiles qui serviront à la procédure de calibration

#define MAGNITUDEMAXETOILESCALIBRATION 3.5

// distance minimale entre deux étoiles de référence utilisées par les méthodes d'alignement
AFFINE et TAKI

#define MIN_DISTANCE_ALIGNEMENT 0.3

////////////////////////////////////////
// Constantes Mathématiques //
////////////////////////////////////////

#define Pi 3.14159265358979323846264338327
#define Pidiv180 0.01745329251994329576923690768
#define N180divPi 57.2957795130823208767981548141
#define Pi2 6.28318530717958647692528676655
#define Pidiv2 1.57079632679489661923132169163

// 1 ' en rad

#define MINUTE_ARC 0.000290888209

// valeurs des angles (exprimés en rad) ajoutés à l'ascension droite et à la déclinaison
lorsque
// l'utilisateur agit sur les touches fléchées durant la procédure d'alignement des antennes

#define PasDeltaAD 2.0 * MINUTE_ARC
#define PasDeltaDe 2.0 * MINUTE_ARC
```

```
#define gmax(A,B) ((A)>(B)?(A):(B))
#define gmin(A,B) ((A)<(B)?(A):(B))

////////////////////////////////////
// dimensions de la fenêtre graphique //
////////////////////////////////////

#define haut_fenetre 10+22*10
#define larg_fenetre 605

////////////////////////////////////
// couleurs utilisées dans le terminal //
////////////////////////////////////

#define red1 "\033[0;31m"
#define blue1 "\033[0;34m"
#define green1 "\033[0;32m"
#define black1 "\033[0;30m"
#define grey1 "\033[1;30m"
#define red2 "\033[1;31m"
#define blue2 "\033[1;34m"
#define green2 "\033[1;32m"
#define black2 "\033[1;30m"
#define grey2 "\033[1;37m"

////////////////////////////////////
// Macros //
////////////////////////////////////

#define SAFEDELETE(pointer) if ((pointer)) { delete ((pointer)); pointer = NULL; }
#define SAFEDELETE_TAB(pointer) if ((pointer)) { delete [] ((pointer)); pointer = NULL; }

////////////////////////////////////
// Fonctionnement des trames sur le réseau //
// Ne pas modifier //
////////////////////////////////////

// Port utilisé sur le réseau tcp/ip pour communiquer avec les antennes

#define BAO_PORT 8000

// Délai maximum en ms d'attente de la réponse des microcontrôleurs

#define MAX_DELAI_REPONSE 40

// Si une commande ne reçoit pas d'acknowledge. Alors refaire 80 tentatives en renvoyant la
commande

#define MAXATTENTE 80

// Si pas de réponse au bout de 80 tentatives -> erreur critique -> socket perdu ?

#define MAXANOMALIES 2

// attendre plus de 2 mn pour considérer qu'une antenne n'est pas en mesure de faire un goto

#define MAXANOMALIESGOTO 1500
```

Sommaire :

Conception du programme.....	2
Utilisation du programme.....	5
Compilation du programme	6
Alignement des antennes - principe	7
Utilisation de BAOcontrol pour lancer la calibration	7
Utilisation de BAOcontrol:.....	11
Documentation technique.....	17
Class BAOcontrol : liste des méthodes employées :.....	18
Classe BAO : liste des méthodes employées :.....	23
Class Alignement, liste des méthodes employées :.....	27
Class Astro, liste des méthodes employées :.....	31
Méthodes d'alignement	37
1ère méthode : méthode SIMPLE	38
2ème et 3ème méthodes : Méthode AFFINE et TAKI.....	41
Fichier const.h contenant les paramètres du programme.....	44