# Sources PushTo_Mega

## Table des matières



# 1 Fichier Astro.h

```
#ifndef MLL_ASTRO_H
#define MLL_ASTRO_H
#include <math.h>
#define DsR 57.2957795131     // 180/pi
#define HsR 3.819718634205    // 12/pi

/*
 * Conversion coordonnées horizontales en coordonnées horaires équatoriales
 * Entrees :
 *    azi_dg    : azimut en degrés, rétrograde d'origine le Nord (convention Stellarium)
 *    haut_dg   : hauteur en degrés
 * Sorties :
 *    anH_h     : angle horaire équatorial en heures
 *    DEC_dg    : déclinaison en degrés
 */
void Loc2hor(double azi_dg, double haut_dg, double *anH_h, double *DEC_dg);

/*
 * Initialisation des paramètres astronomiques
 *  ATTENTION : heure TU et non heure locale (enlever le décalage / Greenwich)
 * Entrées :
 *      jmUT :      jour julien modifie MLL  (horaire TU)
 *      longi_dg :  longitude du lieu en degrés décimaux
 *      lati_dg :  latitude du lieu en degrés décimaux
 *      flagGPS :   VRAI si jjUT provient d'une vrai horloge GPS non retouchée TU
 *      hdec :      Heure décimale (pour pb manque de précision avec l'Arduino)
 * Sortie :
 *      tsla_h :    le temps sideral local apparent en heures decimales
 */
double initAstro(double jmUT, double longi_dg, double lati_dg, int flagGPS, double hdec);

/*
 * Jour julien MLL modifiés : j  = 0 le 1.5/1/2020, (1.5 cad à 12h)
 * Valable pour dates postérieures au 15/10/1582
 * ATTENTION aux décimales : Convention différente pour d et jj.
 * d est un jour décimal 1.0 le 1er à 0h et 1.5 le 1er à minuit
 * m est le mois : 1 pour janvier
 * a est l'année
 * Retour : jour julien décimal : xxxxx.0 à 12h et xxxxx.5 à 24h.
 * La modification MLL est censé contourner la perte de précision sur
 * l'Arduino les "double" sont sur 4 octets.
*/
double dmy2julm(double d, int m, int a);

/**
 *      Variation de hauteur astre due a la refaction
 * @param h_dg      : hauteur theorique ou apparente en degres
 * @param th2ap    : 1 si h_dg est theorique, 0 si apparente (observee)
 * @param tc_c      : temperature en Celsius
 * @param p_mb      : pression en millibar
 * @return         : dh_dg variation de hauteur en degres
 */
 double refraction_dg(double h_dg, int th2ap, double tc_c, double p_mb);
#endif
```

# 2 Fichier Astro.cpp

```
#include "astro.h"
#include <math.h>
#include "Arduino.h"

/*
 * Interpolation extrêmement grossière de l'écart deltaT = TT-UT
 * Pourrait être négligé ici car n'est important que pour le calcul de la
 * position des corps du système solaire.
 * Entree : jm = jour julien modifié MLL (début 1.5/1/2020)
 * Sortie : deltaT = TT - UT en secondes
*/
static double jul2deltaT_s(double jm)  // (local, non exporté)
{
  double d;
  double jj = jm + 2458850; // Corrige modif MLL
  if(jj > 2407716.) d = (jj-2407716.)*0.001369; // >= 1980
  else if (jj < 2339745.) d = (2339745. - jj)*0.0037114; // <= 1694
  else d = 0.;
  return d;
}



// Lignes trigo latitude en global à ce fichier
extern double sinPhi, cosPhi; // calculés par initAstro
extern double temperature_c, pression_mb;

/*
 * Conversion coordonnées horizontales en coordonnées horaires équatoriales
 * Entrees :
 *    azi_dg    : azimut en degrés, rétrograde d'origine le Nord (convention Stellarium)
 *    haut_dg   : hauteur en degrés
 * Sorties :
 *    anH_h     : angle horaire équatorial en heures
 *    DEC_dg    : déclinaison en degrés
 */
void Loc2hor(double azi_dg, double haut_dg, double *anH_h, double *DEC_dg)
{
 double azi_rd = azi_dg/DsR;
  double dh_dg = refraction_dg(haut_dg, 0, temperature_c, pression_mb);
 double haut_rd = (haut_dg - dh_dg)/DsR;
 double sinA = sin(azi_rd);
 double cosA = cos(azi_rd);
 double sinH = sin(haut_rd);
 double cosH = cos(haut_rd);
 *anH_h = atan2(-sinA*cosH, cosPhi*sinH - sinPhi*cosA*cosH)*HsR;
 *DEC_dg = asin(sinPhi*sinH + cosPhi*cosA*cosH)*DsR;
}

/*
 * Initialisation des paramètres astronomiques
 *  ATTENTION : heure TU et non heure locale (enlever le décalage / Greenwich)
 * Entrées :
 *      jmUT :      jour julien modifie MLL  (horaire TU)
 *      longi_dg :  longitude du lieu en degrés décimaux
 *      lati_dg :  latitude du lieu en degrés décimaux
 *      flagGPS :   VRAI si jjUT provient d'une vrai horloge GPS non retouchée TU
 *      hdec :      Heure décimale (pour pb manque de précision avec l'Arduino)
 * Sortie :
 *      tsla_h :    le temps sideral local apparent en heures decimales
 */
double initAstro(double jmUT, double longi_dg, double lati_dg, int flagGPS, double hdec)
{
  if (flagGPS) jmUT -= 18./86400.;  // le vrai temps GPS avançait de 18 secondes le 1/1/2017
  // Temps sidéral moyen à Greenwich //
  double T = (floor(jmUT - 0.5) + 0.5)/36525.;
  double tsmg_dg = 100.614645 + T*(36000.7702+T*(0.0003879175-T/38710000));
  tsmg_dg += 15.04106864*hdec;
  int n = floor(tsmg_dg/360.); tsmg_dg -= n*360.;
  // Temps sidéral apparent à Greenwich
  double tsga_dg = tsmg_dg;
#undef PLUSPRECIS
#ifdef PLUSPRECIS
```

```
  T = (jmUT + 7305.)/36525.;
  double O = (125.04452  - 1934.136261*T)/DsR;            // Longi. noeud asc. Lune
  double dL = 2.*(280.4665 + 36000.7698*T)/DsR;           // Longi. moy. Soleil
  double dM = 2.*(218.3165 + 481267.8813*T)/DsR;          // Longi. moy. Lune
  // Nutation => Delta précession
  double dPsi_dg = (-17.20*sin(O) - 1.32*sin(dL) - 0.23*sin(dM) + 0.21*sin(2.*O))/3600.;
  // Nutation => Delta obliquité
  double dEps_dg = (9.20*cos(O) + 0.57*cos(dL) + 0.10*cos(dM) - 0.09*cos(2.*O))/3600.;
  // Obliquité moyenne
  double epsm_dg = 23. + 26./60. + (21.448 - T*(46.8150 + T*(0.00059 - 0.001813*T)))/3600.;
  // Obliquité apparente
  double epsa_dg = epsm_dg + dEps_dg;
  double cosEps = cos(epsa_dg/DsR);
  tsga_dg += dPsi_dg*cosEps;
#else
  tsga_dg += -0.0042; // Valeur début 2020
#endif
  // Temps sidéral local apparent
  double tsla_h = (tsga_dg + longi_dg)/15.;
  // Ligne trigo latitude
  double lat_rd = lati_dg/DsR;
  sinPhi = sin(lat_rd);
  cosPhi = cos(lat_rd);
  return tsla_h;
}

/*
 * Jour julien MLL modifiés : j  = 0 le 1.5/1/2020, (1.5 cad à 12h)
 * Valable pour dates postérieures au 15/10/1582
 * ATTENTION aux décimales : Convention différente pour d et jj.
 * d est un jour décimal 1.0 le 1er à 0h et 1.5 le 1er à minuit
 * m est le mois : 1 pour janvier
 * a est l'année
 * Retour : jour julien décimal : xxxxx.0 à 12h et xxxxx.5 à 24h.
 * La modification MLL est censé contourner la perte de précision sur
 * l'Arduino les "double" sont sur 4 octets.
*/
double dmy2julm(double d, int m, int a)
{
  long y = a + 4800;
  if (m <= 2) {y--; m += 12;}
  long v = y/400 - y/100 + (36525*y)/100 + (306*(m+1))/10;
  v -= 2458850;  // modification MLL
  return (double)v + d - 32167.5;
}

/**
 *      Variation de hauteur astre due a la refaction
 * @param h_dg      : hauteur theorique ou apparente en degres
 * @param th2ap     : 1 si h_dg est theorique, 0 si apparente (observee)
 * @param tc_c      : temperature en Celsius
 * @param p_mb      : pression en millibar
 * @return          : dh_dg variation de hauteur en degres
 */
double refraction_dg(double h_dg, int th2ap, double tc_c, double p_mb)
{
    double arg_dg, dh_dg;
    double K = (283./(tc_c+273.))*p_mb/1010.;
    if (th2ap) // Pour calculer la hauteur à commander au télescope
    {
      arg_dg = h_dg + 10.3/(h_dg + 5.1);
      if (arg_dg < 90.) dh_dg = K*(1.02/tan(arg_dg/DsR))/60.;
      else dh_dg = 0.;
    }
    else // Pour calculer la hauteur de l'étoile visée par le télescope
    {
      arg_dg = h_dg + 7.31/(h_dg + 4.4);
      if (arg_dg < 90.) dh_dg = K*(1./tan(arg_dg/DsR))/60.;
      else dh_dg = 0.;
    }
    return dh_dg;
}
```

# 3 Fichier Codeurs.cpp

```
#include "Arduino.h"
// Connexions Mega aux 4 fils de la petite plaque d'interface codeurs
#define enc_1A 18 // vert
#define enc_1B 19 // blanc
#define enc_2A 2 // violet car 20 pris par SDA OLED
#define enc_2B 3 // gris car 21 pris par SCL OLED

// Modifier les 2 valeurs suivantes en fonction du rapport de réduction utilisé
long stepsTr1 = 72000; // (4*600)*30; => 30 tours codeurs pour 360° en azimut
long stepsTr2 = 72000; // (4*600)*30; => 30 tours codeurs pour 360° en site

volatile int  lastEncoded1 = 0;
volatile int  lastEncoded2 = 0;
volatile long encoderValue1 = 0;
volatile long encoderValue2 = 0;

void Encoder1() {
 int encoded1 = (digitalRead(enc_1A) << 1) | digitalRead(enc_1B);
 int sum  = (lastEncoded1 << 2) | encoded1;
 if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue1++;
 if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue1--;
 lastEncoded1 = encoded1;
}

void Encoder2() {
 int encoded2 = (digitalRead(enc_2A) << 1) | digitalRead(enc_2B);
 int sum  = (lastEncoded2 << 2) | encoded2;

 if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue2 ++;
 if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue2 --;
 lastEncoded2 = encoded2;
}

void setupCodeurs()
{
 pinMode(enc_1A, INPUT_PULLUP);
 pinMode(enc_1B, INPUT_PULLUP);
 pinMode(enc_2A, INPUT_PULLUP);
 pinMode(enc_2B, INPUT_PULLUP);
 attachInterrupt(digitalPinToInterrupt(enc_1A), Encoder1, CHANGE);
 attachInterrupt(digitalPinToInterrupt(enc_1B), Encoder1, CHANGE);
 attachInterrupt(digitalPinToInterrupt(enc_2A), Encoder2, CHANGE);
 attachInterrupt(digitalPinToInterrupt(enc_2B), Encoder2, CHANGE);
}

void readCodeurs(double *azi_dg, double *haut_dg)
{
 *azi_dg = (encoderValue1*360.)/stepsTr1;
 *haut_dg = (encoderValue2*360.)/stepsTr2;
 /*
 if (*haut_dg > 90.) *haut_dg = 90.;
 if (*haut_dg < -90.) *haut_dg = -90.;
 */
}
```

# 4 Fichier pave44b.h

```
#ifndef PAVE44B_H
#define PAVE44B_H
// ---------------------------------------------------------
//    VERSION AVEC SORTIE SUR MICRO-ECRAN Adafruit_SSD1306
// ---------------------------------------------------------
#include <Adafruit_SSD1306.h>
extern Adafruit_SSD1306 display; // Pour sortie de contrôle
extern int linepix;
// ---------------------------------------------------------
void affiche(int y, char *texte)
{
 display.setCursor(0, y); display.print(texte); display.display();
}

void delcar(int i, int y)
{
 display.fillRect(i*6, y, 6, 8, SSD1306_BLACK); display.display();
}

#include "Adafruit_Keypad.h"
#define ERRLNG -2147483648
#define ERRDBL -3.E+38

//definition des caractères du keypad
const byte NB_L = 4; // nb de lignes
const byte NB_C = 4; // nb de colonnes
char keys[NB_L][NB_C] = {
 {'1','2','3','A'},
 {'4','5','6','B'},
 {'7','8','9','C'},
 {'*','0','#','D'}
};

/* Classe gestion du keypad 4x4
 * Constructeur Pave44(byte *rowPins, byte *colPins)
 *    avec par exmple le brocahge suivant pour un Méga
 *    byte rowPins[4] = {45,43,41,39}; //les broches lignes 1 à 4
 *    byte colPins[4] = {37,35,33,31}; //les broches colonnes 5 à 8
 * Méthodes :
 * Lecture d'un caractère avec attente maximale d'un certain delai.
 *    char getC(int attente_ms) -> char
 * Lecture d'une chaine de nbc char au max, jusqu'au caractère '#'
 * avec attente maximale de tmax_ms par caracatère
 * Le caractère C n'est pas lu, il sert à effacer le caractère précédent
 *    char *getS(char *ligne, int nbc, long tmax_ms) -> char*
 * Lecture d'un entier jusqu'au caractère '#'
 * Le caractère C sert à effacer le caractère précédent
 *    long getL() -> long ou ERRLNG si erreur
 * Lecture d'un réel jusqu'au caractère '#'
 * Le caractère C sert à effacer le caractère précédent
 * le caractère '*' sert de virgule
 *    double getD() -> double ou ERRDBL si erreur
 */
class Pave44b : public Adafruit_Keypad
{
public :
 int dl, dc; // Position du curseur sur l'écran
 // Constructeur
 Pave44b(byte *rowPins, byte *colPins) : Adafruit_Keypad(makeKeymap(keys), rowPins, colPins, NB_L, NB_C)
 {
  ;
 }

 // Méthode utilisée pour surveiller une appel de l'utilisateur avec un '#'
 // Renvoi faux si rien ou pas encore laché le '#'
 // Renvoi true après un lacher de '&'
 bool userIT()
 {
  tick();
  while(available())
  {
   keypadEvent e = read();
```

```
   if (e.bit.KEY == '#' && e.bit.EVENT == KEY_JUST_RELEASED) return true;
  }
  return false;
}

// Lecture d'un caractère avec attente maximale d'un certain delai.
char getC(int attente_ms)
{
long tdeb = millis();
  while (millis()-tdeb < attente_ms)
  {
   tick();
   while(available())
   {
    keypadEvent e = read();
    if(e.bit.EVENT == KEY_JUST_RELEASED) return (char) e.bit.KEY;
   }
   delay(1);
  }
  return '\0';
}

// Lecture d'une chaine de nbc char au max, jusqu'au caractère '#'
// avec attente maximale de tmax_ms par caractère
// Le caractère C n'est pas lu, il sert à effacer le caractère précédent
char *getS(char *ligne, int nbc, long tmax_ms)
{
  int i = 0, c;
  while (i < nbc)
  {
   c = getC(tmax_ms);
   if (c == 0) { ligne[0] = 0; return ligne;}
   if (c == 'C')
   {
    ligne[i] = 0;
    if (i > 0) {i--; delcar(i, linepix); }
    continue;
   }
   if (c == '#') {ligne[i] = 0;  break;}
   ligne[i] = c; i++; ligne[i] = 0;
   affiche(linepix, ligne);
  }
  linepix += 8;
  return ligne;
}

// Lecture d'un entier jusqu'au caractère '#'
// Le caractère C sert à effacer le caractère précédent
// Le caractère A sert de signe moins
#define MAXNBL 12
long getL(long tmax_ms)
{
  int i = 0, c;
  bool negatif = false;
  char *ligne = malloc(MAXNBL);
  if(ligne == NULL) return ERRLNG;
  ligne[i] = '0'; // Si on ne reçoit rien cette valeur est utilisée
  while (i < MAXNBL-1)
  {
   c = getC(tmax_ms);
   if (c == 0) {free(ligne); return ERRLNG;}
   if (c == 'C')
   {
    ligne[i] = 0; if (i == 0) negatif = false;
    if (i > 0)
    {
     i--; if (i == 0) negatif = false;
     delcar(i, linepix);
    }
    continue;
   }
   if (i == 0 && c == 'A') {c = '-'; negatif = true;}
   if (c == '#') break;
   ligne[i] = c; i++; ligne[i] = 0;
   affiche(linepix, ligne);
  }
```

```
  ligne[i] = 0;
  long v = atol(ligne);
  free(ligne);
  linepix += 8;
  return v;
 }

 // Lecture d'un réel jusqu'au caractère '#'
 // Le caractère C sert à effacer le caractère précédent
 // le caractère '*' sert de virgule
 #define MAXNBD 14
 double getD(long tmax_ms)
 {
  int i = 0, c;
  bool negatif = false;
  char *ligne = malloc(MAXNBD);
  if(ligne == NULL) return ERRLNG;
  ligne[i] = '0'; // Si on ne reçoit rien cette valeur est utilisée
  while (i < MAXNBD-1)
  {
   c = getC(tmax_ms);
   if (c == 0) {free(ligne); return ERRDBL;}
   if (c == 'C')
   {
    ligne[i] = 0; if (i == 0) negatif = false;
    if (i > 0)
    {
     i--; if (i == 0) negatif = false;
     delcar(i, linepix);
    }
    continue;         }
   if (i == 0 && c == 'A') {c = '-'; negatif = true;}
   if (c == '*') c = '.';
   if (c == '#') break;
   ligne[i] = c; i++; ligne[i] = 0;
   affiche(linepix, ligne);
  }
  ligne[i] = 0;
  double v = atof(ligne);
  free(ligne);
  linepix += 8;
  return v;
 }
};
#endif
```

# 5 Fichier PushToMega.ino

```
// BROCHAGE - BRANCHEMENTS
// Codeur Azimut 18 -19 (vert - blanc)
// Codeur Site   2 - 3 (violet - gris)
// Bluetooth Rx-Tx en 14-15 (Serial3)
// GPS Rx-Tx en 16-17 (Serial2)
// Ecran Oled SDA-SCL en 20-21
#define global // Pour signaler transmission directe avec les autres fichiers
#include "Arduino.h"
#include "astro.h"         // Les fonctions astro

// Includes pour écran Oled
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
// ECRAN OLED
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1    // car cet écran n'a pas de broche reset
#define SCREEN_ADDRESS 0x3C // la norme indique 0x3D, mais ma puce est chinoise
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// POUR LE KEYPAD
#include "pave44b.h"        // Pour l'entrée keypad
byte rowPins[4] = {45,43,41,39}; //les broches lignes 1 à 4
byte colPins[4] = {37,35,33,31}; //les broches colonnes 5 à 8
Pave44b pave = Pave44b(rowPins, colPins);
global int linepix = 0; // Numéro ligne du display

// Gestion Liaisons série via pointeurs, plus souple en cas de modifs
// en particulier pour substituer le bluetooth au câble USB vers PC
#include "HardwareSerial.h"
HardwareSerial *serieGPS = &Serial2;
HardwareSerial *comSTELL = &Serial3; // Liaison PC vers STELLARIUM

// Gestion puce GPS
#include <TinyGPS++.h>
static TinyGPSPlus gps; // Branchée sur serieGPS

void setupCodeurs(); // Evite de faire un codeurs.h
void readCodeurs(double *azi_dg, double *haut_dg) ;

global double temperature_c = 10., pression_mb = 1013.;
global double sinPhi, cosPhi; // calculés par initAstro

// La localisation exacte est fournie par le GPS
static double lati_dg=45, longi_dg=0;// Localisation
static char strLongi[12], strLati[12]; // Pour affichage

// Angle horaire, A.D. et Dec inexacts tant que GPS non initilaisé
static double angleHor_h=0., AD_h=0., DEC_dg=90.;

void setup() {
 // Initialisation clavier
 pave.begin();
 // ini sin et cos lati, pour affichage bidon en attendant la loc
 sinPhi = sin(lati_dg/57.2957795131);
 cosPhi = cos(lati_dg/57.2957795131);
 // Conditionnement écran
 if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) exit(0);
 delay(2000);
 display.clearDisplay();
 display.setTextColor(WHITE);
 display.setTextSize(1);
 display.setCursor(0, 0);
 display.display();

 // Initialisation codeurs
 setupCodeurs();


 // Initialisation liaison PC (pour Stellarium
 display.println("Attente com. PC");
 display.display();
```

```
 comSTELL->begin(9600);  while (!(*comSTELL)) delay(10);

 // Initialisation liaison GPS
 display.println("Attente com. GPS");
 display.display();
 serieGPS->begin(9600) ;
 while (!(*serieGPS)) delay(10); // Attente connexion avec GPS
 display.clearDisplay();

 // Pour affichage localisation (inexacte pour le moment)
 formatLng(longi_dg, strLongi);
 formatLat(lati_dg, strLati);

}

// Pour le dialogue utilisateur
#define WAITUSERMS 20000
static bool userIT = false;
enum UserCde {FAUX, DATE, HEURE, LONGD, LATID, LONGS, LATIS, REINIASTRO, NEANT};
static long userCde = NEANT;

void loop()
{

// La date exacte est fournie par le GPS
static int annee, mois, jour;
static int heure, minutes, seconde;
static double hTUini_h;       // heure Tu initiale
static unsigned long tIni_ms; // temps millis() initial à l'instant de l'heure initale
static double hSIDini_h;      // temps sideral local initial
static double jmTUini_j;      // jour julien initial

// Etat des données fournies par le GPS
static bool okgps = false, oklocgps = false, okdategps = false, oktimegps = false;
static bool okuser = false, okdateuser = false;;

#define GainSid 1.0027379 // pour conversion temps TU en temps sidéral
static unsigned long tAffich_ms = 0;  // instant dernier affichage

char tampon[32];  // tampon de travail de conversion de strings
double hTUnow_h, hSIDnow_h;           // temps TU et sidéral courant
double azi_dg, haut_dg;               // Azimut, site de la visée
unsigned long tEcoule_ms;             // temps écoulé depuis l'ini temps GPS
double tEcoule_h;                     // idem en heures
// Coordonnées pointées envoyées à stellarium
char strTxAD[10], strTxDEC[11];

 // Lecture des codeurs
 readCodeurs(&azi_dg, &haut_dg);

 // Conversion en angle horaire et déclinaison
 Loc2hor(azi_dg, haut_dg, &angleHor_h, &DEC_dg);
 // Heure TU
 tEcoule_ms = millis() - tIni_ms;
 tEcoule_h = tEcoule_ms/3600000.;
 hTUnow_h = hTUini_h + tEcoule_h;
 // Heure sidérale
 hSIDnow_h = hSIDini_h + tEcoule_h*GainSid;
 // Conversion coord azimutales -> equatoriales
 AD_h = hSIDnow_h - angleHor_h;
 while(AD_h < 0.) AD_h += 24.;
 while(AD_h > 24.) AD_h -= 24.;

 // Traitement données de la puce GPS
 if(!okgps) while(serieGPS->available() > 0)
 {
  gps.encode(serieGPS->read());
  if (okdategps && !oktimegps && gps.time.isUpdated())
  {
   heure = gps.time.hour();
   minutes = gps.time.minute();
   seconde = gps.time.second();
   tIni_ms = millis();
   hTUini_h = heure + minutes/60. + seconde/3600.;
   oktimegps = true;
  }
  if (!oklocgps && gps.location.isUpdated())
```

```
   {
    lati_dg = gps.location.lat();
    longi_dg = gps.location.lng();
    oklocgps = true;
    formatLng(longi_dg, strLongi); // Pour affichage
    formatLat(lati_dg, strLati); // Pour affichage
   }
   if (!okdategps && gps.date.isUpdated())
   {
    annee = gps.date.year();
    mois = gps.date.month();
    jour = gps.date.day();
    okdategps = true;
   }
   okgps = oklocgps && okdategps && oktimegps;
   // ENFIN la localisation est ok, ON INITIALISE LES FORMULES ASTRO
   if (okgps)
   {
    // serieGPS->end(); // Fermeture liaison série avec GPS
    // Initialisation astro (jour julien initial imprécis sur Arduino : au quart de jour près)
    jmTUini_j = dmy2julm(jour + hTUini_h/24., mois, annee);
    hSIDini_h = initAstro(jmTUini_j, longi_dg, lati_dg, false, hTUini_h);
    display.clearDisplay();
   }
  }
  // ENVOI DES DONNEES A STELLARIUM (Strings LX200)
  val_s2ams(strTxAD, AD_h*3600., 0);
  val_s2ams(strTxDEC, DEC_dg*3600., 1);
#if 0
  // Version standard
  if (comSTELL->available() > 0) comWithStellarium(strTxAD, strTxDEC);
#else
  // Version modifiée pour éviter le blocage en cas de problème sur la liaison
  comWithStellarium2(strTxAD, strTxDEC);
#endif



  // ENVOI INFOS VERS AFFICHEUR LOCAL 1 fois par seconde seulement
  if (tEcoule_ms - tAffich_ms >= 1000)
  {
   display.clearDisplay(); display.setCursor(0, 0);
   if (okgps || okuser) {
    display.print("H TU :"); display.print(vdec2sex(hTUnow_h, tampon)); display.println(okgps ? " gps" : "
user");
    display.print("H Sid:"); display.print(vdec2sex(hSIDnow_h, tampon)); display.println(okgps ? " gps" : "
user");
    }
   else {
    display.println(oktimegps ? "Temps GPS OK" : "Attente TempsGPS");
    display.println(oklocgps  ? "Loca. GPS OK" : "Attente Loca.GPS");
   }
   display.print("Long : "); display.println(strLongi);
   display.print("Lati : "); display.println(strLati);
   display.print("Azim : "); display.println(vdec2sex(azi_dg, tampon));
   display.print("Haut : "); display.println(vdec2sex(haut_dg, tampon));
   display.print("AD_h : "); display.println(vdec2sex(AD_h, tampon));
   display.print("Dec. : "); display.println(vdec2sex(DEC_dg, tampon));
   display.display();
   tAffich_ms = tEcoule_ms;
  }

  // DIALOGUE UTILISATEUR
char ligne[16];
long vl;
double vd;
int vi;
char str[12];
bool ok;
  if (!userIT)
  {
   userIT = pave.userIT();
  }
  if (userIT)
  {
   userIT = false; // Prêt pour recevoir un nlle userIT
   display.clearDisplay();
```

```
linepix = 0;
affiche(linepix, "Commande ?#");  linepix += 8;
userCde = pave.getL(WAITUSERMS);
if (userCde == ERRLNG || userCde <= 0 || userCde >= NEANT)
{
 display.clearDisplay();
 display.setCursor(0, 0);
 display.println("0# : cette aide");
 display.println("1# : date");
 display.println("2# : heure");
 display.println("3# : long.deci");
 display.println("4# : lati.deci");
 display.println("5# : long.sexa");
 display.println("6# : lati.sexa");
 display.println("7# : Reinit gps");
 display.display();
 delay(15000); // Pour laisser le temps de lire
 display.clearDisplay();
 display.setCursor(0, 0);
 display.println("A(-) pour Ouest, Sud");
 display.println("* pour virgule (.)");
 display.println("# pour executer");
 display.display();
 delay(5000); // Pour laisser le temps de lire
}
else switch (userCde)
{
 case DATE :
 {
  affiche(linepix, "(Date :) JJMMAA#");  linepix += 8;
  vl = pave.getL(WAITUSERMS);
  ok = false;
  if(vl != ERRLNG)
  {
   int an = vl%100;
   int mo = (vl/100)%100;
   int jo = (vl/10000);
   if (an > 20 && mo > 0 && mo <= 12 && jo > 0 && jo <= 31)
   {
    ok = true;
    annee = 2000+ an; mois = mo; jour = jo; okdateuser = true;
    sprintf(ligne, "Date: %d/%02d/%d",jour, mois, an);
    affiche(linepix, ligne);  linepix += 8;
   }
  }
  if (!ok) { affiche(linepix, "Date incorrecte!"); linepix += 8; }
  delay(2000);
 break;
 }
 case HEURE :
 {
  if (!okdateuser) {
   affiche(linepix, "D'abord la date !!!");  linepix += 8;
   delay(2000); break;
  }
  affiche(linepix, "(Heure TU :) HHMM#");  linepix += 8;
  vl = pave.getL(WAITUSERMS);
  ok = false;
  if(vl != ERRLNG)
  {
   int mm = vl%100;
   int hh = vl/100;
   if (hh >=0 && hh <= 24 && mm >= 0 && mm < 60)
   {
    ok = true;
    sprintf(ligne, "Heure TU: %d:%02d:00",hh, mm);
    affiche(linepix, ligne);  linepix += 8;
    affiche(linepix, "Entrer * pour top");  linepix += 8;
    if (pave.getC(60000) == '*') {
     tIni_ms = millis();
     heure = hh;
     minutes = mm;
     seconde = 0;
     hTUini_h = heure + minutes/60. + seconde/3600.;
     jmTUini_j = dmy2julm(jour + hTUini_h/24., mois, annee);
     hSIDini_h = initAstro(jmTUini_j, longi_dg, lati_dg, false, hTUini_h);
```

```
      okuser = true;
     }
     else
     {affiche(linepix, "Heure ignoree !");  linepix += 8; delay(2000);}
    }
   }
   if (!ok) { affiche(linepix, "Heure incorrecte!"); linepix += 8; delay(2000);}
   break;
}
case LONGD :
case LATID :
{
   affiche(linepix, userCde==LONGD ? "Longi.decimale" : "Lati. decimale"); linepix += 8;
   affiche(linepix, "(-D)DD.dddd#"); linepix += 8;
   vd = pave.getD(WAITUSERMS);
   ok = false;
   if(vd != ERRDBL)
   {
    bool neg = false;
    if (vd < 0) neg = true;
    if ((userCde==LONGD && vd >= -180. && vd < 360) || (userCde==LATID && vd >= -90. && vd <= 90))
    {
     ok = true;
     if (userCde==LONGD)
     {
      longi_dg = vd;
      formatLng(longi_dg, strLongi);
      sprintf(ligne, "Longi.: %s",strLongi);
      affiche(linepix, ligne);  linepix += 8;

     }
     else
     {
      lati_dg = vd;
      formatLat(lati_dg, strLati);
      sprintf(ligne, "Lati.: %s",strLati);
      affiche(linepix, ligne); linepix += 8;
     }
    }
   }
   if (!ok) { affiche(linepix, "Coord. incorrecte!"); linepix += 8; }
   delay(2000);
   break;
}
case LONGS :
case LATIS :
{

   affiche(linepix, userCde==LONGS ? "Longi.sexage." : "Lati. sexage."); linepix += 8;
   affiche(linepix, "(-D)DDMMSS#"); linepix += 8;
   vl = pave.getL(WAITUSERMS);
   ok = false;
   if(vl != ERRLNG)
   {

    bool neg = false;
    if (vl < 0) {neg = true; vl = -vl;}
    int ss = vl%100;
    int mm = (vl/100)%100;
    int dd = vl/10000;
    if (ss >=0 && ss < 60 && mm >= 0 && mm < 60 && dd > 0 && dd < ((userCde==LONGS)? 360 : 91))
    {
     ok = true;
     vd = dd+mm/60.+(ss+0.1)/3600.; if (neg) vd = -vd;
     if (userCde==LONGS)
     {
      formatLng(vd, str);
      sprintf(ligne, "Longi.: %s",str);
      affiche(linepix, ligne); linepix += 8;
     }
     else
     {
      formatLat(vd, str);
      sprintf(ligne, "Lati.: %s",str);
      affiche(linepix, ligne);  linepix += 8;
     }
```

```
       }
      }
      if (!ok) { affiche(linepix, "Coord. incorrecte!"); linepix += 8; }
      delay(2000);
      break;
     }
     case REINIASTRO:
     {
      okgps = false;
      okdategps = false;
      oktimegps = false;
      oklocgps = false;
      break;
     }
   } // fin du switch userCde
 } // Fin du if (userIT)

}

void comWithStellarium(char *strTxAD, char *strTxDEC)
{
#define MSW 1  // 1 milliseconde d'attente
 char a, b, c, d;
 int i;
 do
 {
  do {delay(MSW); a = comSTELL->read();} while (a != ':');
  delay(MSW); b = comSTELL->read(); if (b != 'G') continue;
  delay(MSW); c = comSTELL->read(); if (c != 'R' && c != 'D') continue;
  delay(MSW); d = comSTELL->read();
 } while (d != '#');
 if (c == 'R') comSTELL->print(strTxAD);
 if (c == 'D') comSTELL->print(strTxDEC);
}

// VERSION COM AVEC STELLARIUM MODIFIEE
/*
 * La version précédente peut bloquer si la communication est altérée et qu'on ne
 * trouve pas de caractères ':' ou '#' dans le message. La version suivante, plus
 * lourde, évite ce problème peu problable, mais sait-on jamais ?.
 */
static int ncl = 0;
static char rcu;
#define Av comSTELL->available()
#define Rd comSTELL->read()

void comWithStellarium2(char *strTxAD, char *strTxDEC)
{
 switch(ncl) {
 case 0 :
  while(Av > 0) if (Rd == ':') {ncl = 1; break;}
  if (ncl != 1) break;
 case 1 :
  if (Av <= 0) break;
  if (Rd != 'G') {ncl = 0; break;}
  ncl = 2;
 case 2 :
  if (Av <= 0) break;
  rcu = Rd;
  if (rcu != 'R' && rcu != 'D') {ncl = 0; break;}
  ncl = 3;
 case 3 :
  if (Av <= 0) break;
  if (Rd != '#') {ncl = 0; break;}
  if (rcu == 'R') comSTELL->print(strTxAD);
  if (rcu == 'D') comSTELL->print(strTxDEC);
  ncl = 0;
 }
}



static char *vdec2sex(double vdec, char *tampon)
{
int v,m,s;
double reste;
bool signeNEG;
```

```
  signeNEG = false;
  if (vdec < 0)  {signeNEG = true; vdec = -vdec;}
  v = vdec; reste = (vdec - v)*60.;
  m = reste; s = (reste - m)*60.;
  if (signeNEG) sprintf(tampon,"-%2d:%02d'%02d\"",v,m,s);
  else sprintf(tampon," %2d:%02d'%02d\"",v,m,s);
  return tampon;
}

/*
 * Formattage des heures et degres decimaux en sexagsimaux au format LX200.
 * Entrees :
 *    tampon : buffer dans lequel sera écrite la valeur sexagesimale
 *    val_s  : Valeur a écrire (heure ou angle) DONNEE EN SECONDES (A T T E N T I O N)
 *    casdeh : 1 si l'entrée est en " de degrés, 0 si secondes d'heures
 * Sortie :
 *    renvoi un pointeur sur le tampon (qui contient le résultat)
 */
static char *val_s2ams(char *tampon, long val_s, int casdeg)
{
long w, a, m, s, r;
int negatif = 0;
 if (val_s >= 0) {negatif = 0; w = val_s;}
 else  {negatif = 1; w = -val_s;}
 a = w/3600; w -= a*3600;
 m = w/60;
 s = w - 60*m;
 if (casdeg)
  sprintf(tampon, "%c%02d*%02d:%02d#", negatif ? '-' : '+', int(a), int(m), int(s));
 else
  sprintf(tampon, "%02d:%02d:%02d#", int(a), int(m), int(s));
 return tampon;
}
/*
 * Formatage longitude
 */
static char *formatLng(double longi_dg, char *buf)
{
int isLngEst = 1;
int lng_dd, lng_mm, lng_ss;
double reste, lng_dg = longi_dg;
 if (lng_dg < 0.) {lng_dg = -lng_dg; isLngEst = 0;} else isLngEst = 1;
 lng_dd = lng_dg; reste = (lng_dg - lng_dd)*60.;
 lng_mm = reste;
 lng_ss = (reste - lng_mm)*60.;
 sprintf(buf,"%3d:%02d'%02d\"%c",lng_dd, lng_mm, lng_ss, isLngEst?'E':'W');
 return buf;
}

/*
 * Formatage latitude
 */
static char *formatLat(double lati_dg, char *buf)
{
int isLatNord = 1;
int lat_dd, lat_mm, lat_ss;
double reste, lat_dg = lati_dg;;
 if (lat_dg < 0.) {lat_dg = -lat_dg; isLatNord = 0;} else isLatNord = 1;
 lat_dd = lat_dg; reste = (lat_dg - lat_dd)*60.;
 lat_mm = reste;
 lat_ss = (reste - lat_mm)*60.;
 sprintf(buf,"%3d:%02d'%02d\"%c",lat_dd, lat_mm, lat_ss, isLatNord?'N':'S');
 return buf;
}
```