```
//
// Ce programme fonctionne sur carte ARDUINO MEGA
// So but est de récupérer les données du GPS NS-HP-GL et de les afficher sur un afficheur 2 lignes de 16 caractères
// On affiche seulement la distance du mobile EO et NS par rapport à la base
// un BP permet de désigner la position zéro
//
//
// Bibliotheques
#include <LiquidCrystal.h>  // pour afficheur cristaux liquides
#include <SPI.h> // bibliotheque de la liaison SPI pour écrire dans la mémoire SD ; 10: CS, 51: MOSI, 50: MISO, 52: SCK
#include <SD.h> // bibliotheque pour catrte mémoire SD
#include <Wire.h> // bibliotheque pour I/F I2C pour lire la boussole ;  MEGA: pins 20-SDA et 21-SCL
#include <math.h>
//
//   Fonctions utilisées
void lectureBoussole(); // declaration de fonction pour lire la boussole/inclinometre
void lectureTableau(); // declaration de fonction pour lire l'état du tableau de bord
void lectureTelecommande(); // declaration de fonction pour lire l'etat de la telecommande
void lectureGPS(); // declaration de fonction pour lire position GPS, nombre de satellites, HDOP et status (pas de fix, float, fix)
void commandeMoteurs(); // declaration de fonction pour commander les moteurs
void pilotageManu(); // lit la position télécommande et commande les moteurs
void affichage(); // affichage fonction du mode repére par un préfixe "M" ou "F"
void affichageFauchage(); // affichage spécial fauchage avec double Lti et Lgi (actuel / but)
void rotationCentre();  // permet de tourner autour du point central d'un certain nombre d'unités de la boussole. Gestion discontinuité 255-1 incluse. Pas de mesure GPS avant convergence.
void rotationRoue(); // permet de tourner autour d'une roue d'un certain nombre d'unités de la boussole. Gestion discontinuité 255-1 incluse. Pas de mesure GPS avant convergence.
void rotationRoueG();
void rotationRoueD();
void lecture_WP(); // va lire le Nième" Way Point" but_longi et but_lati dans le fichier champ n° M
void ecriture_SD();    // Ecriture dans un fichier lors de l'enregistrement du parcours (prefixe = E) ou lors du fauchage (prefixe F)
boolean OK_GPS();   // Fonction qui verifie la validité des données GPS: Status = 4 et écart par rapport à la position précédente < 50 cm
// int boussole();  // Fonction qui convertit l'azimut GPS en azimut boussole  (avec gestion discontinuité 255 - 1)
byte boussole();
//
// Variables et parametres pour lecture et traitements GPS
#define  navsparkSerial Serial1   // Sortie Tx1 du GPS reliée à 19 (Rx1) de la MEGA (hardware serial 1)
float RepGPS[5];
float D_lati0 = 0;
float D_longi0 = 0;
byte RepDateHeure[6];
//
// Variables et parametres pour afficheur
byte Af_D4 = 28;
byte Af_D5 = 26;
byte Af_D6 = 24;
byte Af_D7 = 22;
byte Af_RS = 32;
byte Af_E = 30;
byte COLS = 16;
byte ROWS = 2;

//
// Variables et parametres pour boussole / inclinometre
int RepBoussole[3];   // RepBoussole[0] = angle8 ; RepBoussole[1] = pitch ; RepBoussole[2] = roll ;
//
// Variables et parametres pour carte SD
byte pinCS = 10; // le shield SD prévu pour UNO a la pin Chip Select configurée sur 10; Cette pin est accessible par la MEGA;
```

```
// Par contre les pins 11,12,13 de l'interface SPI pour UNO doivent être reliées obligatoirement à 51(MOSI),50(MISO) et 52(SCK) de la MEGA
 byte Nfich; // numero du fichier
//
//
// Variables et parametres pour moteurs
byte pinPWM_D = 5;
byte pinSensRot_D = 4;
byte pinPWM_G = 6;
byte pinSensRot_G = 7;
byte pinPWM_AV = 2;
byte pinSensRot_AV = 3;
//
// Variables et parametres pour tableau
byte RepTableau[6];
//
// Variables et parametres pour telecommande manuelle
byte RepTelecommande[2];
//
// Variables et parametres pour commande de coupe
byte pinCoupe = 36;
//
LiquidCrystal LCD (Af_RS, Af_E, Af_D4, Af_D5, Af_D6, Af_D7); // initialisation afficheur
//
// Variables pour navigation
float but_longi; // contient la valeur lue dans le fichier parcours fauchage
float but_lati;   // idem
byte but_azi;  //
byte but_azi_D;  // stockage de but_azi + 180°  pour rotation 180 en bout
byte but_azi_G;
float ang;      //
byte Nwp;    //  contient le nombre de WayPoints lu dans le fichier fauchage
byte N = 1;   // valeur du waypoint en cours. Variable initialisée à 1 de façon à aller chercher la première ligne des waypoints du fichier fauchage.
              // après la fin du fauchage N vaut Nwp. Seul un reset carte MEGA permettra une réinitialisation à 1 si par exemple on veut exécuter un second fauchage
boolean rot = false; // varaible booléenne vraie quand l'orientation initiale au début de chaque tronçon a bien été effectuée
float X = 0;  //  variables X et Y utilisées dans les calculs
float Y = 0;
float but_lati0[40];   // vecteurs servant à stocker les but lati et longi des waypoints
float but_longi0[40];
//  int offset = 0;       // angle entre boussole pointé réellement et angle calculé avec le GPS

float reflati = 0;  // variables globales contenant les positions latitude et longitude et utilisées dans la fonction de vérification OK_GPS()
float reflongi = 0;

float D =0;   // variable contenant la distance entre deux waypoints
float d = 0;  // variable contenant la distance parcourue depuis le dernier waypoint

boolean marcheCoupe = false; // cette variable est pilotée par le fichier fauchage (pour chaque WP, A ou M)
boolean fixValid = false; // cette variable est pilotée par le fichier fauchage (pour chaque WP, A ou M)

float fixTimer1; float fixTimer2;
byte Virage;  // 0: normal     1: roue gauche    2: roue droite

//--------------------------------------------------------------------------------------------------
void setup() {                                                  // SETUP
//--------------------------------------------------------------------------------------------------
  LCD.begin(COLS, ROWS);
  delay(100);
  LCD.clear();
  LCD.print ("Bonjour V22");  // on rajoute la gestion de la barre de coupe au SW suivant les WayPoints (V10) + fonction boussole avec fitting erreur
```

```
116     delay (2000);
117     //
118     navsparkSerial.begin(57600);   // Vitesse de transfert normalement sans probleme
        avec une inteface HW serial 1 de la carte MEGA
119     //
120     pinMode(pinPWM_D, OUTPUT); // sortie numérique PWM pour moteur droit
121     pinMode(pinPWM_G, OUTPUT); // sortie numérique PWM pour moteur gauche
122     pinMode(pinSensRot_D, OUTPUT); // sortie numérique pour sens de rotation moteur
        droit
123     pinMode(pinSensRot_G, OUTPUT); // sortie numérique pour sens de rotation moteur
        gauche
124     pinMode(pinPWM_AV, OUTPUT); // sortie numérique pour PWM moteur avant
125     pinMode(pinSensRot_AV, OUTPUT); // sortie numérique pour sens de rotation moteur
        avant
126     //
127     pinMode(pinCoupe, OUTPUT); // sortie numérique pour commande barre de coupe
128     //
129     pinMode(pinCS, OUTPUT); // On déclare pinCS comme sortie
130     SD.begin(pinCS); // initialisation de la carte mémoire avec pinCS = 10 comme
        ChipSelect
131     Wire.begin();
132     navsparkSerial.begin(57600);
133     Serial.begin(115200);  //L'interface serie de l'IDE est utilisee pour verifier le
        deroulement du programme
134     ///   while (!Serial); // attend tant que lla liaison série de l'IDE n'est pas prête
135     //
136     commandeMoteurs(1,1); // Tous les moteurs arrêtés
137     delay(100);
138     digitalWrite (pinCoupe,LOW) ; // Barre de coupe arrêtée
139     delay(100);
140
141     fixValid = false;
142     fixTimer1 = 0; fixTimer2 = 0;
143   }
144   //
145   //
146
147   //-------------------------------------------------------------------------------------
      ------------------------------------------------
148   void loop()                                                          // LOOP
149   //-------------------------------------------------------------------------------------
      ------------------------------------------------
150   {
151       lectureTableau();
152       lectureBoussole();
153       lectureTelecommande();
154       lectureGPS();
155
156
157     if (RepTableau[0] == 1)
        //-----------------------------------------------------------------// Mode manuel
158     {
159
160
161         if (RepTableau[1] == 0)  // mise à zero sur le point de départ. D_lati0 et
            D_longi0 contiennent les positions absolues du point de départ
162         {
163           D_lati0 = RepGPS[0];
164           D_longi0 = RepGPS[1];
165         }
166
167         if (RepTableau[4] == 2)  // inter barre de coupe en position haute
168         {
169          digitalWrite (pinCoupe,HIGH) ; // Barre de coupe en marche pour fauchage en
             mode manuel
170         }
171         else
172         {
173          digitalWrite (pinCoupe,LOW) ; // Barre de coupe arrêtée si inter en position
             basse ou médium
174         }
175
176           affichage('M');
```

```
177           pilotageManu();
178
179          Nfich = RepTableau[3] + RepTableau[2]*3 + 1; // numéro de fichier qui sera
             utilisé soit en mode Enregistrement, soit en mode Fauchage
180
181          reflati = RepGPS[0];     // on met en mémoire la première valeur de longitude
             et latitude pour la fonction de détection d'écart de position
182          reflongi = RepGPS[1];    // Au basculement en mode fauchage, c'est cette
             valeur qui sert de première valeur de référence
183      }
184
185
186
187      if (RepTableau[0] == 0)
         //------------------------------------------------------ // Mode Enregistrement
188      {
189
190          affichage('E');
191          pilotageManu();
192          ecriture_SD('E');
193
194         if (RepTableau[4] == 2)  // inter barre de coupe en position haute
195         {
196          digitalWrite (pinCoupe,HIGH) ; // Barre de coupe en marche pour fauchage en
             mode enregistrement
197         }
198         else
199         {
200          digitalWrite (pinCoupe,LOW) ; // Barre de coupe arrêtée si inter en position
             basse ou médium
201         }
202      }
203
204
205
206      if (RepTableau[0] == 2)
         //-------------------------------------------------------------// Mode fauchage
207      {
208
209
210         if (fixValid == false)
211         {
212             affichageFauchage();
213             if (RepGPS[4] != 4 )   //
214             {
215               fixTimer1 = millis();
216               fixTimer2 = millis();
217             }
218             if (RepGPS[4] == 4 )   //
219             {
220               fixTimer2 = millis();
221             }
222
223             if ((fixTimer2 - fixTimer1) > 60000 )
224             {
225               D_lati0 = RepGPS[0];
226               D_longi0 = RepGPS[1];
227               reflati = RepGPS[0];     // on met en mémoire la première valeur de
                  longitude et latitude pour la fonction de détection d'écart de position
228               reflongi = RepGPS[1];    // Au basculement en mode fauchage, c'est cette
                  valeur qui sert de première valeur de référence
229               fixValid = true;
230             }
231         }
232
233
234
235         if (fixValid == true)
236         {
237          but_lati0[0] = 0; but_longi0[0] = 0;
238
239         if (OK_GPS()== true)   // Si les données GPS sont de niveau fix RTK (Status =
             4) et si les écarts de position sont inférieurs à un seuil de 50 cm, on entre
```

```
        dans le pilotage automatique
240     {
241           lecture_WP(N);  // on va lire la Nième ligne du fichier des waypoints
              Champ_x ( x de 1 à 9 donné par la position des interrupteurs ) - N
              initialisé à 1 en début de séquence.
242                           // après cette lecture on dispose de but_lati, but_longi
                              et but_azi du Nième waypoint.
243
244           if (N <= Nwp)    // pour chacun des waypoints de N=1 à Nwp. La variable
              nombre de waypoints Nwp a été lue lors de l'appel à la fonction
              lecture_WP()
245           {
246               affichageFauchage();
247   //            ecriture_SD('F');
248
249              if (RepTableau[4] == 0)  // inter barre de coupe en position basse:
                 barre de coupe commandée par le fichier
250               {
251               if (marcheCoupe == true) digitalWrite(pinCoupe,HIGH);
252               else
253               digitalWrite(pinCoupe,LOW);
254               }
255
256              if (RepTableau[4] == 2)  // inter barre de coupe en position haute:
                 barre de coupe toujours ON
257               {
258               digitalWrite(pinCoupe,HIGH);
259               }
260              if (RepTableau[4] == 1)  // inter barre de coupe en position moyenne:
                 barre de coupe toujours OFF
261               {
262               digitalWrite(pinCoupe,LOW);
263               }
264
265             but_lati0[N] = but_lati;         // on met en mémoire but_lati et
                but_longi, pour les futurs calculs. but_lati0[0] et but_longi0[0] ont
                été initialisés à 0
266             but_longi0[N] = but_longi;
267
268             Y = but_lati0[N]-but_lati0[N-1];   // Calcul de la distance D entre
                les deux waypoints
269             X = but_longi0[N]-but_longi0[N-1];
270             D = sqrt(X*X + Y*Y);
271
272
273             X = (RepGPS[1]-D_longi0) - but_longi0[N-1];     // calcul de la
                distance d entre position courante et way point origine
274             Y =  (RepGPS[0]-D_lati0) - but_lati0[N-1];
275             d = sqrt(X*X + Y*Y);
276
277
278             if (d <= D-0.1)   // on programme un arrêt à 10 cm du but de façon à
                compenser l'effet du retard d'exécution
279             {
280                   if (rot == false)                    // à chaque début de
                      ligne, rotation vers le prochain Way Point et calcul de
                      l'offset valable pour l'étape
281                   {
282                     ang =  atan2 (but_lati0[N] - but_lati0[N-1],but_longi0[N] -
                        but_longi0[N-1]) * 180. / 3.1415;  // angle calculé à partir
                        des positions GPS
283                     but_azi = boussole(ang); // calcul de l'angle boussole à
                        partir coordonées GPS
284                     if (Virage == 0) rotationCentre(but_azi);
285                     rot = true;
286                     if (Virage == 1)
287                     {
288                     rotationRoue(but_azi_G);    // but_azi_G est l'angle
                        précédent + 160°
289                       N = N+1;
290                       rot = true;
291                       goto Fin_virage;
292                     }
```

```
293                         if (Virage == 2)
294                         {
295                           rotationRoue(but_azi_D);    // but_azi_D est l'angle
                              précédent - 160°
296                           N = N+1;
297                           rot = true;
298                           goto Fin_virage;
299                         }
300
301
302
303                       }
304
305                       X = but_longi0[N-1] + (d + 1) / D * ( but_longi0[N]- but_longi0[
                          N-1] );
306                       Y = but_lati0[N-1] + (d + 1) / D * ( but_lati0[N]- but_lati0[N-1
                          ] );
307                       X = X - (RepGPS[1]-D_longi0);
308                       Y = Y - (RepGPS[0]-D_lati0);
309                       ang = atan2 (Y,X) * 180. / 3.1415;  // (Y, X) OK pour la
                          fonction atan2
310                       but_azi = boussole(ang);
311
312                       ecriture_SD('F');
313
314                       commandeMoteurs (2,2);                          // après
                          l'orientation vers le prochain way point, on lance les moteurs
                          en MAV
315
316                       if (abs(RepBoussole[0] - but_azi) > 2)       // corrrection de
                          direction
317                       {
318                         rotationRoue (but_azi);
319                       }
320                       commandeMoteurs (2,2);                          // après
                          correction de direction, on re-lance les moteurs en MAV
321
322                   }
323                 else
324                   {
325                       commandeMoteurs (1,1);
326                       N = N+1;
327                       rot = false;
328                       but_azi_G = boussole (ang + 160);    //   On arrête avant un
                          retournement complet (180°) de façon à avoir un mouvement plus
                          fluide  // +160 = vers la Gauche
329                       but_azi_D = boussole (ang - 160);    //   On arrête avant un
                          retournement complet (180°) de façon à avoir un mouvement plus
                          fluide // -160 = vers la Droite
330                   }
331               }
332
333               Fin_virage:
334
335               if (N > Nwp)
336               {
337               commandeMoteurs(1,1);           //  Moteurs arrêtés
338               digitalWrite (pinCoupe,LOW) ;   // Barre de coupe arrêtée
339               }
340         }
341
342         if (OK_GPS() != true)          //  En l'absence de RTK valide ou d'écart de
            position trop fort, tous les moteurs sont arrêtés et la fauchause se met en
            attente. On continue à afficher et à enregistrer dans la carte SD.
343         {
344           commandeMoteurs(1,1);           //  Moteurs arrêtés
345           digitalWrite (pinCoupe,LOW) ;   // Barre de coupe arrêtée
346           affichageFauchage();
347           ecriture_SD('F');
348         }
349         }
350     }
351
```

```
352 //--------------------------------------------------------------------------------
    ------------------------------------------------
353 }    // fin loop
354 //--------------------------------------------------------------------------------
    ------------------------------------------------
355
356 ////////////////////////////////////////////////////////////////////////////////
    ////////////////////
357 //*****************************************  FONCTIONS
    ************************************************
358 ////////////////////[////////////////////////////////////////////////////////////
    ////////////////////
359
360 //*****************************************************************************
    *************************
361 void lectureBoussole()   // lit la boussole et  retourne la direction de 0-255 ainsi
    que le roulis et le tangage
362 {
363 #define CMPS11_ADDRESS 0x60  // Address of CMPS11 shifted right one bit for arduino
    wire library
364 #define ANGLE_8  1           // Register to read 8bit angle from
365   unsigned char high_byte, low_byte, angle8;
366   char pitch, roll;
367   unsigned int angle16;
368
369   //
370   Wire.beginTransmission(CMPS11_ADDRESS);  //starts communication with CMPS11
371   Wire.write(ANGLE_8);                     //Sends the register we wish to start
    reading from
372   Wire.endTransmission();
373
374   // Request 5 bytes from the CMPS11 - This will give us the 8 bit bearing,  both
    bytes of the 16 bit bearing, pitch and roll
375   Wire.requestFrom(CMPS11_ADDRESS, 5);
376
377   while (Wire.available() < 5);       // Wait for all bytes to come back
378
379   angle8 = Wire.read();               // Read back the 5 bytes
380   high_byte = Wire.read();
381   low_byte = Wire.read();
382   pitch = Wire.read();
383   roll = Wire.read();
384
385   angle16 = high_byte;                 // Calculate 16 bit angle
386   angle16 <<= 8;
387   angle16 += low_byte;
388
389   RepBoussole[0] = int(angle8); // angle8 byte de 0-255
390   RepBoussole[1] = int(pitch);  // pitch signed interger
391   RepBoussole[2] = int(roll);   // roll signed integer
392 }
393 //
394
395 //*****************************************************************************
    ***************************************
396 void lectureTelecommande()  // lit les positions des 2 boutons de la commande codées
    sur 3 valeurs: 0 (bas), 1 (milieu), 2 (haut)
397 {
398   byte analogPinTCG = 14;   // telecommande moteur G
399   int valTCG = 5;
400   byte analogPinTCD = 15;  // telecommande moteur D
401   int valTCD = 5;
402
403   valTCG = analogRead(analogPinTCG);
404   valTCD = analogRead(analogPinTCD);
405
406   if (valTCG < 250) RepTelecommande [0] = 0;
407   if ((valTCG > 250) & (valTCG < 750)) RepTelecommande [0] = 1;
408   if (valTCG > 750) RepTelecommande [0] = 2;
409
410   if (valTCD < 250) RepTelecommande [1] = 0;
411   if ((valTCD > 250) & (valTCD < 750)) RepTelecommande [1] = 1;
412   if (valTCD > 750) RepTelecommande [1] = 2;
```

```
413   }
414   //
415
416   //*************************************************************************************
      ************************************************************************
417   void lectureTableau()  // lit les positions des 6 boutons du tableau de commande
      codées sur 3 valeurs: 0 (bas), 1 (milieu), 2 (haut)  / 0(gauche), 1(milieu), 2(droite)
418   {
419     byte analogPinModSel = 8;
420     int valModSel = 5;
421     byte analogPinMarque = 9;
422     int valMarque = 5;
423     byte analogPinVselect = 10;
424     int valVselect = 5;
425     byte analogPinHselect = 11;
426     int valHselect = 5;
427     byte analogPinCoupe = 12;
428     int valCoupe = 5;
429     byte analogPinAffich = 13;
430     int valAffich = 5;
431
432     valModSel = analogRead(analogPinModSel);
433     valMarque = analogRead(analogPinMarque);
434     valVselect = analogRead(analogPinVselect);
435     valHselect = analogRead(analogPinHselect);
436     valCoupe = analogRead(analogPinCoupe);
437     valAffich = analogRead(analogPinAffich);
438
439     if (valModSel < 250) RepTableau [0] = 0;
440     if ((valModSel > 250) & (valModSel < 750)) RepTableau [0] = 1;
441     if (valModSel > 750) RepTableau [0] = 2;
442
443     if (valMarque < 400) RepTableau [1] = 0;
444     if (valMarque > 600) RepTableau [1] = 1;
445
446     if (valVselect < 250) RepTableau [2] = 0;
447     if ((valVselect > 250) & (valVselect < 750)) RepTableau [2] = 1;
448     if (valVselect > 750) RepTableau [2] = 2;
449
450     if (valHselect < 250) RepTableau [3] = 0;
451     if ((valHselect > 250) & (valHselect < 750)) RepTableau [3] = 1;
452     if (valHselect > 750) RepTableau [3] = 2;
453
454     if (valCoupe < 250) RepTableau [4] = 0;
455     if ((valCoupe > 250) & (valCoupe < 750)) RepTableau [4] = 1;
456     if (valCoupe > 750) RepTableau [4] = 2;
457
458     if (valAffich < 250) RepTableau [5] = 0;
459     if ((valAffich > 250) & (valAffich < 750)) RepTableau [5] = 1;
460     if (valAffich > 750) RepTableau [5] = 2;
461
462   //  LCD.clear();
463   //  LCD.print (String(valAffich));
464   //  delay(1000);
465   }
466   //
467
468   //
469   //*************************************************************************************
      ************************************************************************
470   void lectureGPS()  // lecture du flux NMEA émis par TX1 du GPS. TX1 est reliée à RX1
      (pin 19) de l'interface série hardware n°1 de la MEGA (pins 18=Tx1 et 19=Rx1)
471   {
472     char c[1100];
473    // long lati0;
474   // long longi0;
475     long lati;
476     long longi;
477     long k0 = 1000000;
478     long k1 = 100000;
479     long k2 = 10000;
480     float D_longi;
481     float D_lati;
```

```
482     float k_lati = 0.001851852;   // transforme les millionièmes de minutes de latitude
        en mètres
483     float k_longi = 0.001332111;  // transforme les millionièmes de minutes de
        longitude en mètres  (à la latitude de Lizac)
484     float Hdop;
485     int i;
486     int i0 = 0;
487     byte Nsat;
488     int D0;
489     int D1;
490     int D2;
491     int D3;
492     int D4;
493     int D5;
494     int D6;
495     byte fixStatus = 0;
496     //
497     // lecture de plus de caracteres que ceux contenus dans un paquet NMEA
498     for ( i = 0; i <= 1010; i = i + 1)
499     {
500       while (!navsparkSerial.available()); // attend tant que navsparkSerial n'est pas
        OK
501       c[i] = navsparkSerial.read(); // lecture d'un caractère dès que OK
502     }
503
504      if ((RepTableau[0] == 0) || (RepTableau[0] == 1))
505      {
506      lectureTelecommande();
507      pilotageManu();
508      }
509
510
511     //
512
513    //
       $GPGGA,145854.000,4336.7808980,N,00128.2748093,E,1,05,13.0,157.793,M,50.700,M,,0000*6
       2   /// exemple NMEA
514
515     //  Recherche debut de chaîne $GPGGA
516       i0 = 0;
517       for (int i = 0; i <= 1010; i = i + 1)
518       {
519         if (c[i] == '$' & c[i + 1] == 'G' & c[i + 2] == 'P'& c[i + 3] == 'G'& c[i + 4]
          == 'G' & c[i + 5] == 'A')
520         {
521           i0 = i;
522           break;
523         }
524       }
525
526
527     //  Transformation des caractères de la latitude en chiffres
528       D0 = c[21 + i0] - '0';
529       D1 = c[23 + i0] - '0';
530       D2 = c[24 + i0] - '0';
531       D3 = c[25 + i0] - '0';
532       D4 = c[26 + i0] - '0';
533       D5 = c[27 + i0] - '0';
534       D6 = c[28 + i0] - '0';
535    // Serial.print (D1); Serial.print (D2); Serial.print (D3); Serial.print (D4);
       Serial.print (D5); Serial.println (D6);
536       lati = k0 * D0 + k1 * D1 + k2 * D2 + 1000 * D3 + 100 * D4 + 10 * D5 + D6; //
          millionièmes de minutes
537
538     // Transformation des caracteres de la longitude en chiffres
539       D0 = c[37 + i0] - '0';
540       D1 = c[39 + i0] - '0';
541       D2 = c[40 + i0] - '0';
542       D3 = c[41 + i0] - '0';
543       D4 = c[42 + i0] - '0';
544       D5 = c[43 + i0] - '0';
545       D6 = c[44 + i0] - '0';
546
```

```
547    // Serial.print (D1); Serial.print (D2); Serial.print (D3); Serial.print (D4);
       Serial.print (D5); Serial.println (D6);
548       longi = k0 * D0 + k1 * D1 + k2 * D2 + 1000 * D3 + 100 * D4 + 10 * D5 + D6; //
          millionièmes de minutes
549
550     //  Transformation des caracteres du nombre de satellites en chiffres
551       D1 = c[51 + i0] - '0';
552       D2 = c[52 + i0] - '0';
553       Nsat = 10 * D1 + D2;  // Le nombre de satellites reste toujours inférieur à 20.
          En général entre 10 et 15
554
555   //  Transformation du caractere du status en chiffres (No fix = 0, Float = 1, Fix = 2)
556       fixStatus = c[49 + i0] - '0';
557
558   //  Transformation des caracteres de HDOP en chiffres
559       D1 = c[54 + i0] - '0';
560       D2 = c[56 + i0] - '0';
561
562       Hdop = D1 + D2*0.1;  // Indicateur de qualité
563     //
564
565       // Transformation des caracteres de l'heure en chiffres
566       D0 = c[7 + i0] - '0';
567       D1 = c[8 + i0] - '0';
568       D2 = c[9 + i0] - '0';
569       D3 = c[10 + i0] - '0';
570       D4 = c[11 + i0] - '0';
571       D5 = c[12 + i0] - '0';
572
573      RepDateHeure[3] = D0*10 + D1;
574      RepDateHeure[4] = D2*10 + D3;
575      RepDateHeure[5] = D4*10 + D5;
576
577
578   //  D_lati = (lati - lati0) * k_lati; // distance NS en mètres  (lati0 est l'origine
     marquée par BP dans programme principal)
579   //  D_longi = (longi - longi0) * k_longi; // distance EO en mètres   (longi0 est
     l'origine marquée par BP dans programme principal)
580     D_lati = lati * k_lati; // distance NS en mètres  (lati0 est l'origine marquée par
        BP dans programme principal)
581     D_longi = longi * k_longi; // distance EO en mètres   (longi0 est l'origine marquée
        par BP dans programme principal)
582
583
584     //  Recherche debut de chaîne $GPRMC
585       i0 = 0;
586       for (int i = 0; i <= 1010; i = i + 1)
587       {
588         if (c[i] == '$' & c[i + 1] == 'G' & c[i + 2] == 'P'& c[i + 3] == 'R'& c[i + 4]
          == 'M' & c[i + 5] == 'C')
589         {
590           i0 = i;
591          break;
592         }
593       }
594
595     // Transformation des caracteres de la date en chiffres
596       D0 = c[65 + i0] - '0';
597       D1 = c[66 + i0] - '0';
598       D2 = c[67 + i0] - '0';
599       D3 = c[68 + i0] - '0';
600       D4 = c[69 + i0] - '0';
601       D5 = c[70 + i0] - '0';
602
603      RepDateHeure[0] = D0*10 + D1;
604      RepDateHeure[1] = D2*10 + D3;
605      RepDateHeure[2] = D4*10 + D5;
606
607   RepGPS[0] = D_lati ;
608   RepGPS[1] = D_longi ;
609   RepGPS[2] = Nsat ;
610   RepGPS[3] = Hdop ;
611   RepGPS[4] = fixStatus;
```

```
612
613     delay (10);
614   }
615
616   //*************************************************************************************
      *********************
617   void commandeMoteurs (byte CD, byte CG)
618   {
619
620   if ((CG == 2) & (CD == 2))
621   {
622     digitalWrite(pinSensRot_G, HIGH);
623     analogWrite(pinPWM_G, 255);
624     digitalWrite(pinSensRot_AV, HIGH);
625     analogWrite(pinPWM_AV, 200);
626     digitalWrite(pinSensRot_D, HIGH);
627     analogWrite(pinPWM_D, 255);
628   }
629
630   if ((CG == 1) & (CD == 1))
631   {
632     digitalWrite(pinSensRot_G, HIGH);
633     analogWrite(pinPWM_G, 0);
634     digitalWrite(pinSensRot_AV, HIGH);
635     analogWrite(pinPWM_AV, 0);
636     digitalWrite(pinSensRot_D, HIGH);
637     analogWrite(pinPWM_D, 0);
638   }
639
640   if ((CG == 0) & (CD == 0))
641   {
642     digitalWrite(pinSensRot_G, LOW);
643     analogWrite(pinPWM_G, 250);
644     digitalWrite(pinSensRot_AV, LOW);
645     analogWrite(pinPWM_AV, 200);
646     digitalWrite(pinSensRot_D, LOW);
647     analogWrite(pinPWM_D, 250);
648   }
649
650   if ((CG == 2) & (CD == 0))
651   {
652     digitalWrite(pinSensRot_G, HIGH);
653     analogWrite(pinPWM_G, 255);
654     digitalWrite(pinSensRot_AV, HIGH);
655     analogWrite(pinPWM_AV, 250);
656     digitalWrite(pinSensRot_D, LOW);
657     analogWrite(pinPWM_D, 255);
658   }
659
660   if ((CG == 0) & (CD == 2))
661   {
662     digitalWrite(pinSensRot_G, LOW);
663     analogWrite(pinPWM_G, 255);
664     digitalWrite(pinSensRot_AV, HIGH);
665     analogWrite(pinPWM_AV, 250);
666     digitalWrite(pinSensRot_D, HIGH);
667     analogWrite(pinPWM_D, 255);
668   }
669
670   if ((CG == 2) & (CD == 1))
671   {
672     digitalWrite(pinSensRot_G, HIGH);
673     analogWrite(pinPWM_G, 255);
674     digitalWrite(pinSensRot_AV, HIGH);
675     analogWrite(pinPWM_AV, 200);
676     digitalWrite(pinSensRot_D, LOW);
677     analogWrite(pinPWM_D, 40);
678   }
679
680   if ((CG == 1) & (CD == 2))
681   {
682     digitalWrite(pinSensRot_G, LOW);
683     analogWrite(pinPWM_G, 40);
```

```
  digitalWrite(pinSensRot_AV, HIGH);
  analogWrite(pinPWM_AV, 200);
  digitalWrite(pinSensRot_D, HIGH);
  analogWrite(pinPWM_D, 255);
}

}

//*************************************************************************************************
void pilotageManu()
{
      if ((RepTelecommande[0] == 2) & (RepTelecommande[1] == 2))
      {
          commandeMoteurs (2,2);
      }

      if ((RepTelecommande[0] == 1) & (RepTelecommande[1] == 1))
      {
        commandeMoteurs (1,1);
      }

      if ((RepTelecommande[0] == 0) & (RepTelecommande[1] == 0))
      {
        commandeMoteurs (0,0);
      }

      if ((RepTelecommande[0] == 2) & (RepTelecommande[1] == 0))
      {
          do
          {
          commandeMoteurs (2,0);
          lectureTelecommande();
          delay(100);
          } while ((RepTelecommande[0] == 2) & (RepTelecommande[1] == 0));
      }

      if ((RepTelecommande[0] == 0) & (RepTelecommande[1] == 2))
      {
          do
          {
          commandeMoteurs (0,2);
          lectureTelecommande();
          delay(100);
          } while ((RepTelecommande[0] == 0) & (RepTelecommande[1] == 2));
      }

      if ((RepTelecommande[0] == 2) & (RepTelecommande[1] == 1))
      {
         do
          {
          commandeMoteurs (2,1);
          lectureTelecommande();
          delay(100);
          } while ((RepTelecommande[0] == 2) & (RepTelecommande[1] == 1));
      }

      if ((RepTelecommande[0] == 1) & (RepTelecommande[1] == 2))
      {
         do
          {
          commandeMoteurs (1,2);
          lectureTelecommande();
          delay(100);
          } while ((RepTelecommande[0] == 1) & (RepTelecommande[1] == 2));
      }

    delay(100);
}

//*************************************************************************************************
void affichage (char prefixe)
```

```
{
       if (RepTableau[5] == 2)
      {
       LCD.clear();
       LCD.setCursor(0, 0);
       LCD.print (prefixe + String(Nfich) +" Lgi: " + String(RepGPS[1]-D_longi0));
       // affichage longitude (axe X orienté positivement vers l'est)
       LCD.setCursor(0, 1);
       LCD.print (prefixe + String(Nfich) + " Lti: "  + String(RepGPS[0]-D_lati0));
       // affichage latitude (axe Y orienté positivement vers le nord)
       }

       if (RepTableau[5] == 1)
       {
       LCD.clear();
       LCD.setCursor(0, 0);
       LCD.print (prefixe + String(Nfich) + " Nsat: " + String(int(RepGPS[2])));
       LCD.setCursor(0, 1);
//       LCD.print (prefixe + String(Nfich) + " Hdop: " + String(RepGPS[3]));
       LCD.print (prefixe + String(Nfich) + " STS: " + String(RepGPS[4]));
       }

       if (RepTableau[5] == 0)
       {
       LCD.clear();
       LCD.print (prefixe + String(Nfich) +" Az: "+ String(RepBoussole[0]) );
       LCD.setCursor (0,1);
       LCD.print ("Ro "+ String(RepBoussole[2]) + "  Pi " + String(RepBoussole[1]));
       }
       delay(100);
}

//*************************************************************************************
*************************************
void affichageFauchage()  // Affichages de la position et des buts
{
       if (RepTableau[5] == 2)
      {
       LCD.clear();
       LCD.setCursor(0, 0);
       LCD.print (String((RepGPS[1]-D_longi0),1) + " F" + String(Nfich) + " "  +
       String(but_longi,1));   // affichage longitude (axe X orienté positivement
       vers l'est)
       LCD.setCursor(0, 1);
       LCD.print (String((RepGPS[0]-D_lati0),1) + " F" + String(Nfich) + " "  + String
       (but_lati,1));     // affichage latitude (axe Y orienté positivement vers le
       nord)
       }

       if (RepTableau[5] == 1)
       {
       LCD.clear();
       LCD.setCursor(0, 0);
       LCD.print ("F" + String(Nfich) + " Nsat: " + String(int(RepGPS[2])));
       LCD.setCursor(0, 1);
       LCD.print ("F" + String(Nfich) + " Hdop: " + String(RepGPS[3]));
       }

       if (RepTableau[5] == 0)
       {
       LCD.clear();
       LCD.print (String(RepBoussole[0]) + " F" + String(Nfich)+ " " + String(but_azi)
        );
       LCD.setCursor (0,1);
       LCD.print ("Ro "+ String(RepBoussole[2]) + "  Pi " + String(RepBoussole[1]));
       }
       delay(100);
}

//*************************************************************************************
*************************************

void rotationCentre (byte but)      // Tourne autour d'un point
```

```
819   {
820       lectureBoussole();
821       int difference = but - RepBoussole[0];
822       if (abs(difference) > 128)  difference = -difference;  // si la différence
          dépasse 128 en module, il faut inverser le sens
823
824       if (difference < -1)  // RepBoussole[0] diminue quand on tourne vers la gauche
825       {
826          do
827          {
828            commandeMoteurs(0,2); // rotation vers la gauche
829            lectureBoussole();
830            affichageFauchage();
831            delay(100);
832          } while (abs(RepBoussole[0] -  but) > 1);
833          commandeMoteurs(1,1);
834       }
835
836       if (difference > 1)     // RepBoussole[0] augmente quand on tourne vers la droite
837       {
838          do
839          {
840            commandeMoteurs(2,0);    // rotation vers la droite
841            lectureBoussole();
842            affichageFauchage();
843            delay(100);
844          } while (abs(RepBoussole[0] -  but) > 1);
845          commandeMoteurs(1,1);
846       }
847
848   }
849
850   //
851
852   void rotationRoue (byte but)      // Tourne autour d'une roue
853   {
854       lectureBoussole();
855       int difference = but - RepBoussole[0];
856       if (abs(difference) > 128)  difference = -difference;    // si la différence
          dépasse 128 en module, il faut inverser le sens
857
858       if (difference < -1)              // RepBoussole[0] diminue quand on tourne vers
          la gauche
859       {
860          do
861          {
862            commandeMoteurs(1,2);        // rotation vers la gauche
863            lectureBoussole();
864            affichageFauchage();
865            delay(100);
866          } while (abs(RepBoussole[0] -  but) > 1);
867          commandeMoteurs(1,1);
868       }
869
870       if (difference > 1)             // RepBoussole[0] augmente quand on tourne vers
          la droite
871       {
872          do
873          {
874            commandeMoteurs(2,1);      // rotation vers la droite
875            lectureBoussole();
876            affichageFauchage();
877            delay(100);
878          } while (abs(RepBoussole[0] -  but) > 1);
879          commandeMoteurs(1,1);
880       }
881
882   }
883
884
885   //*************************************************************************************
      ****************************************
886   void ecriture_SD(char prefixe)       // Ecriture dans un fichier lors de
```

```
    l'enregistrement du parcours (prefixe = E) ou lors du fauchage (prefixe F)
887
888     {
889
890
891         //  création d'un fichier E_X.txt  ou F_X.txt  avec X = Nfich déterliné au mode
            manuel
892             String NomFich = String(prefixe) + "_";
893             NomFich = NomFich + String(Nfich)+ ".txt";
894             File dataFile = SD.open(NomFich, FILE_WRITE);
895
896             dataFile.print(RepDateHeure[0]);  dataFile.print("/"); dataFile.print(
                RepDateHeure[1]);  dataFile.print("/");dataFile.print(RepDateHeure[2]);
                dataFile.print(",");
897             dataFile.print(RepDateHeure[3]); dataFile.print(":"); dataFile.print(
                RepDateHeure[4]); dataFile.print(":"); dataFile.print(RepDateHeure[5]);
                dataFile.print(",");
898
899             dataFile.print(RepGPS[0]-D_lati0); dataFile.print(","); dataFile.print(RepGPS[
                1]-D_longi0); dataFile.print(",");
900             dataFile.print(int(RepGPS[2])); dataFile.print(","); dataFile.print(RepGPS[3
                ]); dataFile.print(","); dataFile.print(RepGPS[4]); dataFile.print(",");
901
902             dataFile.print(RepBoussole[0]); dataFile.print(","); dataFile.print(
                RepBoussole[1]); dataFile.print(","); dataFile.print(RepBoussole[2]); dataFile
                .print(",");
903
904             if (RepTableau[0] == 2)
905             {
906               dataFile.print(ang) ; dataFile.print(","); dataFile.print(but_azi);
907             }
908
909
910
911             if ((RepTableau[1] == 0) & (RepTableau[0] != 2) )    // le marqueur est
                utilisé en mode Manuel et Enregistrement, mais pas en mode Fauchage
912             {
913              dataFile.print(",WP");
914              LCD.clear();
915              LCD.print("Way Point");
916             }
917
918             dataFile.println();
919             dataFile.close();
920
921       }
922
923
924     //*************************************************************************************
        ***********************************************
925     void lecture_WP(byte N_WP)        // lecture sur la carte SD de la longitude, latitude
        , azimut du N_WP-ième WayPoint et Marche ou Arrêt de la barre de coupe, Mode de virage
926     {
927         char c;
928         char cc[20];
929         byte n =1;
930         byte np = 0; byte nf = 0;
931         int i=0;
932
933         but_longi = 0;
934         but_lati = 0;
935
936     //      Ouverture fichier en lecture
937             String NomFich = "Champ_";
938             NomFich = NomFich + String(Nfich)+ ".txt";
939             File dataFile = SD.open(NomFich, FILE_READ);
940     //
941     //      Lecture de la première ligne donnant Nwp = Nombre de way points
942             Nwp = 10 * (dataFile.read() - '0');
943             Nwp = Nwp +  (dataFile.read() - '0');
944             dataFile.read(); dataFile.read();   // lecture du CR et du LF de la première
                ligne
945
```

```
//      Recherche ligne du N_WP-ième Way Point
        while (dataFile.available())
        {
         if (n == N_WP) break;
         c =  dataFile.read();
        if (c == 10) {    // 13 == CR   10 == LF
          n = n + 1;
        }
        }
 //
 //     Lecture WP et mise en vecteur de la longitude

       for ( i = 0; i <= 10; i = i + 1)  //
       {
       c = dataFile.read();
       cc[i] = c;
       if (c == '.') np = i; // point
       if (c == ' ') {nf = i; break;}  // espace
      }
 //
//    Calcul de la longitude

     but_longi = (cc[np-1]-'0') * 1.0 + (cc[np+1]-'0') * 0.1 + (cc[np+2]-'0') * 0.01;

      for ( i = np-2; i >= 0; i = i - 1)
     {
      if (cc[i] != '-')
        {
        but_longi = but_longi + (cc[i]-'0') * pow(10., (np-1-i)*1.) ;
        }
        else
       {
        but_longi = but_longi * -1.;
        break;
        }
      }
//
//     Lecture WP et mise en vecteur de la latitude
       for ( i = 0; i <= 10; i = i + 1)
      {
       c = dataFile.read();
       cc[i] = c;
       if (c == '.') np = i; // point
       if (c == ' ') {nf = i; break;}  // espace
      }
//
//    Calcul de la latitude
       but_lati = (cc[np-1]-'0') * 1.0 + (cc[np+1]-'0') * 0.1 + (cc[np+2]-'0') * 0.01;

       for (int i = np-2; i >= 0; i = i - 1)
      {
        if (cc[i] != '-')
        {
        but_lati = but_lati + (cc[i]-'0') * pow(10., (np-1-i)*1.) ;
        }
        else
       {
        but_lati = but_lati * -1.;
        break;
        }
      }


     //  lecture M/A barre de coupe
       c = dataFile.read();   // lecture de l'avant dernier caratère de la ligne
       if (c == 'A') marcheCoupe = false;
       if (c == 'M') marcheCoupe = true;

     //  lecture Methode de virage
       c = dataFile.read();   // lecture de l'espace
       c = dataFile.read();   // lecture du dernier caratère de la ligne
       if (c == 'C') Virage = 0;    // pour virage autour du centre
       if (c == 'G') Virage = 1;   // pour virage autour de la roue gauche
```

```
1018         if (c == 'D') Virage = 2;   // pour virage autour de la roue droite
1019
1020        dataFile.close();
1021    }
1022
1023    //*************************************************************************************************
1024
1025    byte boussole(float angle)       // Transformation d'un angle d'azimut en diretion boussole
1026    {
1027    int angle_boussole;
1028
1029    if (angle > 180) angle = angle - 360;  // l'angle doit toujours être compris entre 180° et -180°
1030    if (angle < 180) angle = angle + 360;  // l'angle doit toujours être compris entre 180° et -180°
1031
1032    if (angle < 0) {angle = angle + 360 ;}
1033    angle_boussole = -255. / 360. * angle + 61 ;         // offset = 61 lié à l'orientation de la boussole sur la machine
1034    angle_boussole = angle_boussole - sin(6.28*(angle -30.)/380.)* 8.  ; // correction erreur cyclique fittée sur des mesures
1035     if (angle_boussole > 255) {angle_boussole = angle_boussole - 255;}
1036     if (angle_boussole <0 ) {angle_boussole = angle_boussole + 255;}
1037
1038      return angle_boussole;
1039    }
1040
1041    //*************************************************************************************************
1042    boolean OK_GPS()    // Fonction qui vérifie si les données GPS sont valides
1043    {
1044    float seuil_ = 10;
1045    boolean STS1 = false;
1046    boolean STS2 = false;
1047    boolean STS3 = false;
1048
1049    if (abs(RepGPS[0]-reflati) < seuil_)   //  Ecart de latitude < 10 mètres
1050    {
1051    STS1 = true;
1052    reflati = RepGPS[0];
1053    }
1054
1055    if (abs(RepGPS[1]-reflongi) < seuil_)  // Ecart de longitude < 10 mètres
1056    {
1057    STS2 = true;
1058    reflongi = RepGPS[1];
1059    }
1060
1061
1062    if (RepGPS[4]==4) {STS3 = true;}     // Status du GPS égal à 4 (FIX RTK)
1063
1064
1065    return (STS1 && STS2 && STS3);  // les 3 conditions sont nécessaires pour avoir une position valide
1066    }
1067
1068
1069
1070
```