```
/** @file
 *
 * @brief Programme principal du robot principal
 * @date 2003-2004
 * @author L'équipe Unitec, http://ae.utbm.fr/unitec/
 * @note Code réalisé pour la Coupe de France de Robotique 2004
 */

#include <stdlib.h>
#include <math.h>
#include "x24x_inc.h"
#include "global.h"

/** L'ordre de déplacement courant. Les valeurs possibles sont
    ASSERV_IDLE, ASSERV_RUNNING, ASSERV_BLOCK */
extern int ordre_deplacement;

/** Abcisse absolue courante du robot */
extern double x_robot;

/** Ordonnée absolue courante du robot */
extern double y_robot;

/** Angle absolu courant du robot */
extern double teta_robot;

/** Valeur courante du timer de décrémentation permettant de se situer
    temporellement dans le match */
extern int timer;

/** Nombre de points stratégiques */
#define NB_STRATEGIC_POINTS 13

/** Les abscisses des points stratégiques */
double strategic_points_x[NB_STRATEGIC_POINTS] = { 2.1, 1.8, 1.8, 1.5, 1.5, 1.2,
 1.2, 0.9, 0.9, 0.6, 0.6, 0.3, 0.3 };

/** Les ordonnées des points stratégiques */
double strategic_points_y[NB_STRATEGIC_POINTS] = { 0.3, 0.3, 1.8, 1.8, 0.3, 0.3,
 1.8, 1.8, 0.3, 0.3, 1.8, 1.8, 0.3 };

/** État balade : le robot se déplace sur tout le terrain en suivant les points
stratégiques */
#define ETAT_BALADE 1

/** État balle trouvée : une balle a été aperçue par le robot, qui s'oriente ver
s elle */
#define ETAT_BALLE_TROUVEE 2

/** État choppe balle : le robot est en face de la balle, il s'avance pour la ré
cupérer */
#define ETAT_CHOPPE_BALLE 3

/** État marque but : le robot a récupéré une balle, il doit aller marquer un bu
t */
#define ETAT_MARQUE_BUT 4

/** État départ : l'état de départ du robot, avec l'action prédéfinie de début d
e match */
#define ETAT_DEPART 5

/** État final du robot */
```



```
#define ETAT_FIN 6

/** Temporisation active
 *
 * La temporisation est réalisée à l'aide d'une boucle contenant quelques
 * instructions. La durée précise d'éxécution ne peut pas être connue.
 *
 * @param n Une grandeur permettant de déterminer la longueur de la temporisatio
n
 */
void tempo(int n)
{
  int i;

  for (i = 0; i < n; i++) { i = i + 1 ; i = i - 1; }
}

/** Rotation
 *
 * Fait pivoter le robot sur lui-même de l'angle donné à la vitesse
 * donnée.
 *
 * @param angle L'angle de rotation
 * @param vitesse La vitesse de rotation, soit
 * ROTATION_VITESSE_NORMAL, soit ROTATION_VITESSE_LENTE
 *
 * @return 0 si la rotation s'est bien déroulée, -1 si un blocage est
 * intervenu durant la rotation
 */
int Rotation(double angle, int vitesse)
{
  double val;

  /* Pour effectuer une rotation, on appelle simplement la fonction
     Initialisation_asservissement en lui précisant qu'on ne souhaite
     pas que le robot se déplace (3ème paramètre à 0). On doit lui
     fournir les coordonnées d'un point vers lequel le robot doit
     s'orienter pour effectuer la rotation du bon angle. */

  /* La tangente n'est pas définie pour l'angle PI/2 qui correspond à
     une rotation de 90° à gauche. */
  if(angle+teta_robot == PI/2)
    {
      Initialisation_asservissement(x_robot, y_robot+1.0, 0, vitesse);
    }
  /* De même pour -PI/2, qui correspond à une rotation de 90° à
     droite */
  else if(angle+teta_robot == -PI/2)
    {
      Initialisation_asservissement(x_robot, y_robot-1.0, 0, vitesse);
    }
  /* Si l'angle est supérieur à PI/2 ou inférieur à -PI/2, il faut
     corriger le calcul de la tangente pour avoir un résultat qui
     convient */
  else if((angle+teta_robot > PI/2) || (angle+teta_robot < -PI/2))  {
    val = tan(PI+angle+teta_robot);
    Initialisation_asservissement(x_robot-1.0, y_robot-val, 0, vitesse);
  }
  /* Sinon, un simple calcul de tangente suffit */
  else
    {
      val = tan(angle+teta_robot);
```

```
      Initialisation_asservissement(x_robot+1.0, y_robot+val, 0, vitesse);
    }

  /* Attente de la terminaison de l'ordre de déplacement */
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Un blocage a-t-il eu lieu ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  return 0;
}

/** Indique si une balle est dans le champ de vision
 *
 * @return 1 si une balle est trouvée, 0 sinon
 */
int balle_trouvee(void)
{
  int ret;
  int barycentre_x, barycentre_y, confiance;
  int i;

  /* Petite temporisation pour ne pas surcharger la caméra */
  tempo(25);

  /* Demande de suivi de couleur */
  camera_track_color(RED_COLOR);

  /* Attente du message de retour */
  ret = serial_command_wait();

  if(ret < 0)
    {
      return 0;
    }

  /* Analyse du paquet reçu */
  ret = serial_receive_m_command(& barycentre_x, & barycentre_y, & confiance);

  if(ret < 0)
    {
      return 0;
    }

  /* Si un objet de couleur rouge est vu avec une confiance
     suffisante, alors nous avons une balle dans le champ de vision */
  if(confiance > 30)
    {
      return 1;
    }

  return 0;
}

/** Positionne le robot en face d'une balle
 *
 * @return 0 si succès, -1 si la balle a été perdue, -2 si le robot
 * s'est bloquée durant le positionnement
 */
```



```
int balle_track(void)
{
  while(1)
    {
      long i;
      int barycentre_x, barycentre_y, confiance;
      int ret;

      /* On demande un suivi de couleur rouge */
      camera_track_color(RED_COLOR);

      /* Attente du message */
      ret = serial_command_wait();

      /* Si erreur, on refait un essai */
      if(ret < 0)
        {
          goto nexttry;
        }

      /* Analyse du message */
      ret = serial_receive_m_command(& barycentre_x, & barycentre_y, & confiance
);

      /* Si erreur, on refait un essai */
      if(ret < 0)
        {
          goto nexttry;
        }

      /* La confiance est devenue trop faible, la balle a été perdue */
      if(confiance < 10)
        {
          return -1;
        }

      /* La balle est trop à gauche dans le champ de vision, on se
         repositionne par une rotation de PI/16. La valeur 32 a été
         déterminée de façon expérimentale en fonction de la largeur
         en pixel de la fenêtre de vision de la caméra. */
      if(barycentre_x < 32)
        {
          ret = Rotation(PI/16, ROTATION_VITESSE_NORMALE);
          if(ret < 0)
            {
              return -2;
            }
        }

      /* La balle est trop à droite dans le champ de vision, on se
         repositionne par une rotation de -PI/16. La valeur 58 a été
         déterminée de façon expérimentale en fonction de la largeur
         en pixel de la fenêtre de vision de la caméra. */
      else if(barycentre_x > 58)
        {
          ret = Rotation(-PI/16, ROTATION_VITESSE_NORMALE);
          if(ret < 0)
            {
              return -2;
            }
        }
```

```
      /* On est en face de la balle */
      else
        {
          return 0;
        }

      /* Avant le prochain essai, on attend un peu */
    nexttry:
      tempo(250);
    }


}

/** Récupération de la balle qui est située dans l'axe du robot
 *
 * @return 0 si la balle a été récupérée avec succès, -1 si la balle a
 * été perdue, -2 si un blocage a eu lieu
 */
int avance_vers_balle(void)
{
  /* La balle est située dans l'axe du robot, on avance donc tout
     droit d'un mètre */
  Initialisation_asservissement(x_robot+cos(teta_robot)*1.0, y_robot+sin(teta_ro
bot)*1.0,
                                1, ROTATION_VITESSE_NORMALE);

  /* Pendant que le déplacement est en cours, on effectue divers tests */
  while(ordre_deplacement == ASSERV_RUNNING)
    {
      long i;
      int barycentre_x, barycentre_y, confiance;
      int ret;

      /* Suivi de la couleur rouge */
      camera_track_color(RED_COLOR);

      /* Réception du message */
      ret = serial_command_wait();

      if(ret < 0)
        {
          goto nexttry;
        }

      /* Analyse du message de la caméra */
      ret = serial_receive_m_command(& barycentre_x, & barycentre_y, & confiance
);

      if(ret < 0)
        {
          goto nexttry;
        }

      /* Si la balle est trop à gauche, on se recentre dessus */
      if(barycentre_x < 32)
        {
          Rotation(PI/16, ROTATION_VITESSE_NORMALE);
        }

      /* Si la balle est trop à droite, on se recentre dessus */
      else if(barycentre_x > 58)
```



```
        {
          Rotation(-PI/16, ROTATION_VITESSE_NORMALE);
        }

      /* La balle arrive dans le bas de la fenêtre de vision de la
         caméra. Elle est très proche du robot */
      if(barycentre_y < 25)
        {
          /* On arrête l'ordre d'avancement d'un mètre */
          ordre_deplacement = ASSERV_IDLE;

          /* On avance de 35 cm en direction de la balle pour qu'elle soit à l'i
ntérieur du robot */
          Initialisation_asservissement(x_robot+cos(teta_robot)*0.35, y_robot+si
n(teta_robot)*0.35,
                                        1, ROTATION_VITESSE_NORMALE);

          /* On attend la fin de l'ordre de déplacement */
          while(ordre_deplacement == ASSERV_RUNNING);

          /* On s'assure qu'il n'y a pas bloquage */
          if(ordre_deplacement == ASSERV_BLOCK)
            {
              return -2;
            }

          /* Ça y est, la balle est dans le robot ! */
          return 0;
        }

      /* La confiance est devenue trop faible, on a perdu la balle */
      if(confiance < 10)
        {
          return -1;
        }

      /* Prochain essai */
    nexttry:
      tempo(250);
    }

  /* A-t-on bloqué pendant le déplacement ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -2;
    }

  /* On ne devrait jamais arriver ici, sauf si la balle a été
     perdue */
  return -1;
}

/** Détection de palmier
 *
 * Cette fonction détecte les palmiers en détectant les zones verte ou
 * noire (car la caméra CMUCam semble confondre le vert des palmiers
 * avec le noir). La fonction prend garde à l'orientation du robot et
 * à sa position pour ne pas confondre les palmiers avec les poteaux
 * des buts. Il est à noter que cette fonction n'a pas été utilisée
 * dans le robot final : la détection des palmiers se faisait par
 * collision.
 *
```

```
 * @return 0 si il n'y a pas de palmier, 1 si il y a un palmier
 */
int detect_palmier(void)
{
  int ret;
  int green_barycentre_x, green_barycentre_y, green_confiance;
  int black_barycentre_x, black_barycentre_y, black_confiance;
  int i;

  /* Si le robot est au bout adverse du terrain, et qu'il regarde vers les
     poteaux de buts, alors on ne peut pas voir de palmiers */
  if ((x_robot > 180) && ((teta_robot < ((5*PI)/12)) && (teta_robot > ((-5*PI)/1
2))))
    {
      return 0;
    }

  /* Si le robot est de notre coté du terrain, et qu'il regarde vers
     les poteaux de buts, alors on ne peut pas voir de palmiers */
  if ((x_robot < 60) && ((teta_robot > ((5*PI)/12)) && (teta_robot < ((-5*PI)/12
))))
    {
      return 0;
    }

  /* Tempo pour laisser le temps à la caméra */
  tempo(25);

  /* Détection des zones vertes */
  camera_track_color(GREEN_COLOR);
  ret = serial_command_wait();

  if(ret < 0)
    {
      return 0;
    }

  /* Décodage du message */
  ret = serial_receive_m_command(& green_barycentre_x,
                                 & green_barycentre_y,
                                 & green_confiance);

  if(ret < 0)
    {
      return 0;
    }

  /* Détection des zones noires */
  camera_track_color(BLACK_COLOR);
  ret = serial_command_wait();

  if(ret < 0)
    {
      return 0;
    }

  /* Décodage du message */
  ret = serial_receive_m_command(& black_barycentre_x,
                                 & black_barycentre_y,
                                 & black_confiance);

  if(ret < 0)
```



```
    {
      return 0;
    }

  /* Si les niveaux de confiance sont suffisamment élevés, il y a un
     palmier */
  if((green_confiance > 45) || (black_confiance > 45))
    {
      return 1;
    }

  return 0;
}

/** Détection du robot adverse
 *
 * La détection du robot adverse utilise le télémètre
 * infrarouge. Cette fonction n'a pas été utilisée dans le robot
 * final, la détection du robot adverse se faisant par collision.
 *
 * @see telem.c
 *
 * @return 0 si il n'y a pas de robot, 1 si il y a un robot
 */
int detect_robot(void)
{
  int val;

  /* Récupération de la valeur du télémètre */
  val = valeur_telemetre();

  /* Si elle est supérieure à un seuil donné, alors un obstacle est
     proche de nous */
  if(val > 190)
    {
      return 1;
    }

  return 0;
}

/** Marquer un but
 *
 * Cette fonction effectue les déplacements nécessaires pour marquer un but
 *
 * @return 0 si le but a été marqué avec succès, -1 si le robot s'est
 * bloqué durant l'action
 */
int marque_but (void)
{
  int end;
  int ret;

  /* On déplace le robot jusqu'à un point assez proche des buts, et
     centré par rapport aux poteaux */
  Initialisation_asservissement(2.15, 1.05, 1, ROTATION_VITESSE_LENTE);
  while (ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
```

```
    }

  /* On continue à avancer sur la ligne centrée en faisant entrer le
     robot assez profondément dans les cages */
  Initialisation_asservissement(2.30, 1.05, 1, ROTATION_VITESSE_LENTE);
  while (ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  /* Pendant 150 interruptions, on fait reculer lentement le
     robot. Ceci est fait à la main, sans asservissement, car
     l'asservissement ne permet pas de faire de marche arrière */
  end = timer - 150;
  while(end != timer)
    {
      CMPR1 = 600;
      CMPR2 = 600;
    }

  /* Demi-tour */
  Rotation(PI, ROTATION_VITESSE_NORMALE);

  /* Pendant 200 interruptions, on fait reculer lentement le robot,
     pour repousser les balles dans les buts */
  end = timer - 200;
  while(end != timer)
    {
      CMPR1 = 600;
      CMPR2 = 600;
    }

  /* On refait sortir le robot des buts, pour qu'il puisse s'orienter
     sans problème vers son objectif suivant */
  Initialisation_asservissement(2.15, 1.05, 1, ROTATION_VITESSE_LENTE);
  while (ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  return 0;
}

/** Action de débloquage
 *
 * Cette fonction est appelée lorsque le robot est en situation de bloquage
 */
void debloquage(void)
{
  /* Pendant 30 interruptions, on recule le robot relativement
     rapidement */
  int end = timer - 30;
  while(timer != end)
    {
      CMPR1 = 400;
      CMPR2 = 400;
```



```
    }

  /* On se tourne d'un angle de -PI/3 pour essayer de contourner l'obstacle */
  Rotation(-PI/3, ROTATION_VITESSE_NORMALE);

  /* On avance de 26cm pour contourner l'obstacle */
  Initialisation_asservissement(x_robot + 0.26 * cos(teta_robot), y_robot + 0.26
 * sin(teta_robot), 1, ROTATION_VITESSE_NORMALE);
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Si jamais un bloquage se produit de nouveau, on sera sorti du
     while() précédent. Le prochain objectif ne pourra être atteint,
     et cette fonction sera appelée de nouveau pour dégager le
     robot. */
}

/** Action de départ
 *
 * Cette fonction est appelée pour faire réaliser au robot l'action de
 * départ, toujours identique
 *
 * @return 0 si succès, -1 si bloquage du robot
 */
int nouveau_depart(void)
{
  int end;

  /* On avance pour récupérer la balle fixe */
  Initialisation_asservissement(0.50, 1.5, 1, ROTATION_VITESSE_NORMALE);
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  /* L'oblique permettant de se rendre sur la droite ne pouvant pas
     contenir de palmier */
  Initialisation_asservissement(0.9, 1.8, 1, ROTATION_VITESSE_NORMALE);
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  /* Traversée complète du terrain sur la ligne ne pouvant pas
     contenir de palmiers, mais pouvant contenir des balles */
  Initialisation_asservissement(2.0 , 1.8, 1, ROTATION_VITESSE_NORMALE);
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  /* L'oblique pour rejoindre le centre des buts */
  Initialisation_asservissement(2.0, 1.05, 1, ROTATION_VITESSE_LENTE);
  while(ordre_deplacement == ASSERV_RUNNING);
```

```
  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  /* On avance profondément dans les buts */
  Initialisation_asservissement(2.30, 1.05, 1, ROTATION_VITESSE_LENTE);
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  /* On recule pendant 150 interruptions */
  end = timer - 150;
  while(end != timer)
    {
      CMPR1 = 600;
      CMPR2 = 600;
    }

  /* Demi tour */
  Rotation(PI, ROTATION_VITESSE_NORMALE);

  /* On recule pendant 200 interruptions pour repousser les balles
     dans les buts par l'arrière du robot */
  end = timer - 200;
  while(end != timer)
    {
      CMPR1 = 600;
      CMPR2 = 600;
    }

  /* On ressort des buts pour pouvoir s'orienter correctement vers le prochain o
bjectif */
  Initialisation_asservissement(2.0, 1.05, 1, ROTATION_VITESSE_LENTE);
  while(ordre_deplacement == ASSERV_RUNNING);

  /* Bloquage ? */
  if(ordre_deplacement == ASSERV_BLOCK)
    {
      return -1;
    }

  return 0;
}

/** L'état courant de la machine à état guidant le robot */
int state;

/** Les 1000 derniers états de la machine à état */
int states[1000];

/** L'index courant dans le tableau states */
int stateidx = 0;

/** L'objectif courant (index dans les tableaux strategic_points_x, strategic_po
ints_y) */
```



```
int objectif = 0;

void main(void)
{
  /* Booléen permettant de savoir si le retour aux buts obligatoire
     au bout d'une minute a été effectué */
  int test_minute = 0;

  /* Initialisation des timers, des PWM, des QEP */
  inittimer();

  /* Initialisation de l'entrée sur laquelle est branchée le signal de
     départ */
  MCRC = MCRC & 0xFFFE;
  PEDATDIR=0x0;

  /* Initialisation du port série */
  serial_port_init();

  /* Réinitialisation de la caméra */
  camera_reset();
  serial_command_wait();

  /* Passage en mode polling de la caméra (un seul paquet renvoyé pour
     chaque demande de suivi de couleur) */
  camera_pm_mode(1);
  serial_command_wait();

  /* Configuration de la caméra */
  camera_settings();
  serial_command_wait();

  /* Réinitialisation du tableau des états */
  for (stateidx = 0; stateidx < 1000; stateidx++)
    {
      states[stateidx] = -1;
    }
  stateidx = 0;

  /* Tant que l'entrée est à 0, la tirette de départ n'a pas été tirée
     et on bloque. Dès que l'entrée passe à 1, le match commence */
  while((PEDATDIR & 1) == 0);

  /* Durant un match, il y a 9000 interruptions. Ce timer va être
     décrémenté toutes les interruptions et le robot sera bloqué
     lorsqu'il arrivera à 0 */
  timer = 9000;

  /* L'état initial de la machine à état est ETAT_DEPART */
  state = ETAT_DEPART;

  /* Les coordonnées initiales du robot (il part à gauche des buts,
     collé contre le muret) */
  x_robot = 0;
  y_robot = 1.5;
  teta_robot = 0;

  /* La machine à état */
  while(1)
    {
      /* Si on est à moins de 30 secondes de la fin et que le passage
         aux buts forcé n'a pas été réalisé */
```

```
      if((timer < 3000 ) && (test_minute == 0))
        {
          /* Passage au but forcé */
          state = ETAT_MARQUE_BUT;

          /* Changement du booléen pour indiquer que le passage aux
             buts a été fait */
          test_minute = 1;
        }

      /* Enregistrement de l'état courant si il a changé par rapport à
         la précédente itération */
      if(states[stateidx] != state)
        {
          stateidx++;
          states[stateidx] = state;
        }

      /* Dans l'état départ, appel de l'action de départ */
      if(state == ETAT_DEPART)
        {
          int ret;

          ret = nouveau_depart();

          if(ret < 0)
            {
              debloquage();
            }

          /* Une fois l'action de départ terminée, on passe en
             ETAT_BALADE */
          state = ETAT_BALADE;
        }

      /* Etat de balade */
      else if(state == ETAT_BALADE)
        {
          /* On se déplace jusqu'au point stratégique de l'objectif courant */
          Initialisation_asservissement(strategic_points_x[objectif], strategic_
points_y[objectif],
                                        1, ROTATION_VITESSE_NORMALE);
          while(ordre_deplacement == ASSERV_RUNNING)
            {
              /* Pendant tout le déplacement, on regarde si on détecte une balle
 */
              if(balle_trouvee() == 1)
                {
                  /* Si oui, on passe directement dans l'état
                     ETAT_BALLE_TROUVEE */
                  state = ETAT_BALLE_TROUVEE;
                  goto nextstate;
                }
            }

          if(ordre_deplacement == ASSERV_BLOCK)
            {
              debloquage();
            }

          /* L'objectif a été atteint sans rencontrer de balle, on
             passe à l'objectif suivant */
```



```
          objectif++;

          /* Si tous les objectifs ont été étudiés, on repart sur le
             premier objectif */
          if(objectif == NB_STRATEGIC_POINTS)
            {
              objectif = 0;
            }
        }

      /* État balle trouvée */
      else if(state == ETAT_BALLE_TROUVEE)
        {
          int tst;

          /* On arrête le robot */
          ordre_deplacement = ASSERV_IDLE;

          /* On se positionne en face de la balle */
          tst = balle_track();

          /* On a été bloqué */
          if(tst == -2)
            {
              debloquage();
              state = ETAT_BALADE;
            }

          /* La balle a été perdue */
          if(tst == -1)
            {
              state = ETAT_BALADE;
            }

          /* On est en face de la balle, on passe dans l'état suivant
             permettant de récupérer la balle */
          else
            {
              state = ETAT_CHOPPE_BALLE;
            }
        }

      /* État de récupération de la balle */
      else if(state == ETAT_CHOPPE_BALLE)
        {
          int tst;

          /* On avance vers la balle */
          tst = avance_vers_balle();

          /* La balle a été perdue */
          if(tst == -1)
            {
              state = ETAT_BALADE;
            }

          /* On a été bloqué */
          else if(tst == -2)
            {
              debloquage();
              state = ETAT_BALADE;
```

```
            }

          /* La balle est dans le robot, on peut aller marquer */
          else
            {
              state = ETAT_MARQUE_BUT;
            }
        }

      /* État de marquage de but */
      else if(state == ETAT_MARQUE_BUT)
        {
          int ret;

          /* On va marquer le but */
          ret = marque_but();

          /* Si on a été bloqué, on repasse en état ballade */
          if(ret < 0)
            {
              debloquage();
              state = ETAT_MARQUE_BUT;
            }

          /* Si la balle a été marquée, on repasse également en état
             ballade pour recherche de nouvelles balles */
          else
            {
              state = ETAT_BALADE;
            }
        }
      else
        {
          break;
        }

      /* Label permettant de forcer le passage à l'état suivant */
    nextstate:
      continue;
    }

  while(1);
}
```