```
/** @file
 *
 * @brief Fonctions de gestion du port série et de la caméra
 * @date 2003-2004
 * @author L'équipe Unitec, http://ae.utbm.fr/unitec/
 * @note Code réalisé pour la Coupe de France de Robotique 2004
 */

#include "global.h"
#include "x24x_inc.h"

/** Buffer de réception pour la gestion du port série */
char serial_buffer[64];

/** Position courante dans le buffer de réception du port série */
int serial_buffer_rx_ptr;

/** Variable permettant de signaler au programme principal la fin de
 * réception du message sur le port série. Le programme principal la
 * met à 0, envoie la commande sur le port série. La caméra va alors
 * envoyer ses informations, qui vont être lues par le handler
 * d'interruption. Lorsque celui-ci rencontrera ':', c'est à dire la
 * fin de la commande, il mettra cette variable à 1 pour signaler au
 * programme principal que la commande est reçue
 */
int receptionCommandeTerminee = 0;

/** Comptage des interruptions, pour débugger */
int nbInterruptionSerie = 0;

/** Réinitialisation du port série pour l'envoi d'une nouvelle
 * commande
 *
 */
void serial_init_command(void)
{
  int i;

  /* On remet cette valeur a 0 pour pouvoir etre signale de la fin de
     la commande */
  receptionCommandeTerminee = 0;

  /* Réinitialisation du buffer de réception */
  serial_buffer_rx_ptr = 0;
  for (i = 0 ; i < 64 ; i++)
  {
        serial_buffer[i] = 0;
  }
}

/** Initialisation matérielle du port série
 *
 */
void serial_port_init(void)
{
  int i;

  /* Activation de l'horloge pour le port série */
  SCSR1 = SCSR1 | 0x40;

  /* Active les entrees / sorties pour le port série (0003) */
  MCRA = MCRA | 0x0003;
```



```
  /* Initialise les indicateurs (reset) */
  SCICTL1 = 0x0;

  /* Active les lignes RX et TX (RX Enable + TX Enable + SW Reset) */
  SCICTL1 = 0x23;

  /* Format des données (1 bit de stop, pas de parite, 8 bits de données) */
  SCICCR = 0x7;

  /* Configuration de la vitesse (vitesse = 19 200 => BRR = 0x103) */
  SCIHBAUD = 0x0;
  SCILBAUD = 0xBF;

  /* Active les interruptions */
  SCICTL2 = 0x2;

  SCIPRI = 0x18;

  serial_init_command();

  asm(" SETC INTM");
  IFR = 0xFFFF;
  IMR = IMR | 0x1;
  asm(" CLRC INTM");
}

/** Envoi d'un caractere sur le port serie
 *
 * @param c Caractère à envoyer */
 */
static void serial_port_send(unsigned char c)
{
  /* Attente du vidage du buffer de transmission */
  while((SCICTL2 & 0x80) == 0);

  /* Envoi */
  SCITXBUF = c;
}

/** Reception d'un caractere du port serie
 *
 * @return Valeur reçue par le port série
 */
static unsigned char serial_port_receive(void)
{
  /* Attente de la disponibilité d'une donnée dans le buffer de réception.
   * Cette attente n'est plus necessaire, car nous appelons cette fonction
   * quand l'interruption serie a ete levee, c'est a dire quand une donnee
   * est disponible */
  /* while((SCIRXST & 0x40) == 0); */

  /* Réception */
  return (SCIRXBUF & 0xFF);
}

/** Change le mode de la LED de la caméra.
 *
 * @param mode Le mode de la LED de la caméra (0 = LED éteinte, 1 =
 * LED allumée, 2 = LED automatique)
 */
void camera_led_change_mode(int mode)
```

```
{
    serial_init_command();
    mode = mode + '0';
    serial_port_send('L');
    serial_port_send('1');
    serial_port_send(' ');
    serial_port_send(mode);
    serial_port_send('\r');
}

/** Change le "polling mode" de la caméra
 *
 * Lorsque le polling mode n'est pas activé la caméra envoie à
 * longueur de temps des informations lors d'un suivi de couleur.
 *
 * @param mode Le "polling mode" (0 = Pas de polling, 1 = Du polling)
 */
void camera_pm_mode(int mode)
{
    serial_init_command();
    mode = mode + '0';
    serial_port_send('P');
    serial_port_send('M');
    serial_port_send(' ');
    serial_port_send('1');
    serial_port_send('\r');
}

/** Réinitialise la caméra
*
*
*/
void camera_reset(void)
{
    serial_init_command();
    serial_port_send('R');
    serial_port_send('S');
    serial_port_send('\r');
}

/** Désactive l'auto-gain
 *
 * L'auto-gain permet de corriger la luminosité de l'image en fonction
 * de la lumière. Pour notre cas, l'auto-gain est désactivé.
 */
void camera_settings(void)
{
    serial_init_command();
    serial_port_send('C');
    serial_port_send('R');
    serial_port_send(' ');
    serial_port_send('1');
    serial_port_send('9');
    serial_port_send(' ');
    serial_port_send('3');
    serial_port_send('2');
}

/** Envoie une demande de suivi de couleur dans le champ de vision de
 *  la caméra
 *
 * @param color La couleur à rechercher. Peut prendre les valeurs
```



```
 * RED_COLOR, GREEN_COLOR, BLACK_COLOR.
 */
void camera_track_color(int color)
{
  /* Définitions des plages de couleurs correspondantes aux couleurs à
     rechercher */
  char red[] = "TC 149 170 15 17 15 17\r";   /*"TC 110 140 5 25 5 25\r"*/
  char green[] = "TC 70 120 140 190 50 80\r";
  char black[] = "TC 15 40 25 35 8 22\r";
  int i;
  int length;

  /* Réinitialiser le buffer de réception */
  serial_init_command();

  /* Si la couleur recherchée est le rouge */
  if(color == RED_COLOR)
    {
      length = strlen(red);

      for (i = 0; i < length; i++)
        {
          /* On envoie la commande correspondant à la recherche de la
             couleur rouge à la caméra. La fonction n'envoie qu'un
             caractère à la fois. On boucle afin d'envoyer toute la
             commande. */
          serial_port_send(red[i]);
        }
    }
  /* Si la couleur recherchée est le vert */
  else if(color == GREEN_COLOR)
    {
      length = strlen(green);

      for (i = 0; i < length; i++)
        {
          /* On envoie la commande correspondant à la recherche de la
             couleur verte à la caméra. La fonction n'envoie qu'un
             caractère à la fois. On boucle afin d'envoyer toute la
             commande. */
          serial_port_send(green[i]);
        }
    }
  /* Si la couleur recherchée est le noir */
  else if(color == BLACK_COLOR)
    {
      length = strlen(black);

      for (i = 0; i < length; i++)
        {
          /* On envoie la commande correspondant à la recherche de la
             couleur noire à la caméra. La fonction n'envoie qu'un
             caractère à la fois. On boucle afin d'envoyer toute la
             commande. */
          serial_port_send(black[i]);
        }
    }
}


/** Handler d'interruption pour la réception de données sur le port série.
 *
```

```
 */
void c_int1(void)
{
  char val;

  asm(" SETC INTM");
  IFR = IFR | 0x1;

  /* On compte les interruptions (débuggage) */
  nbInterruptionSerie = nbInterruptionSerie + 1;

  /* Récupération de la valeur sur le port */
  val = serial_port_receive();

  /* Si c'est un ':', alors la réception de la commande est terminée */
  if(val == ':')
  {
        receptionCommandeTerminee = 1;
  }

  /* Sinon, ajout dans le buffer et incrémentation de la position */
  else
  {
    serial_buffer[serial_buffer_rx_ptr] = val;
    serial_buffer_rx_ptr = serial_buffer_rx_ptr + 1;
  }

  asm(" CLRC INTM");
}

/** Vérifie que la commande précedente s'est bien déroulée.
 *
 * Cette fonction vérifie que le résultat de la commande précédente
 * est correct. Pour cela, on regarde la valeur des 3 premiers
 * octets. Si ils valent ACK, alors la commande s'est bien déroulée,
 * si ils valent NCK, alors la commande ne s'est pas bien déroulée.
 *
 * @return 0 si la commande s'est bien déroulée, -1 sinon et -2 si le
 * résultat de la commande est indéfini
 */
static int serial_command_successful(void)
{
  if(serial_buffer[0] == 'A' && serial_buffer[1] == 'C' && serial_buffer[2] ==
'K')
  {
        return 0;
  }

  if(serial_buffer[0] == 'N' && serial_buffer[1] == 'C' && serial_buffer[2] ==
'K')
  {
        return -1;
  }

  return -2;
}


/** Attend la fin de la réception du résultat de la commande.
 *
 * Cette fonction attend que le handler d'interruption de réception du
 * port série ait reçu le résultat de la commande.
```



```
 *
 * @return 0 si la commande s'est bien déroulée, -1 sinon, -2 si le
 * résultat est indéfini ou -3 si le résultat n'est pas arrivé assez
 * rapidement (timeout).
 */
int serial_command_wait(void)
{
  int ret;
  int delay = 7000;

  /* On attend que le handler d'interruption nous signale la fin de la
     réception */
  while(receptionCommandeTerminee == 0)
  {
     int i;

     /* On attend un peu */
     for (i = 0; i < 100; i++) { i = i + 1 ; i = i - 1; }

     /* On décrémente une variable delay. Ainsi au bout de "delay"
        itérations, on va sortir de la fonction pour gérer une sorte
        de timeout afin de ne pas bloquer */
     delay--;
     if(delay == 0)
       {
         return -3;
       }
  }

  /* Vérification du résultat de la commande */
  ret = serial_command_successful();

  return ret;
}

/** Vérification de la structure d'un paquet de type M
 *
 * Cette fonction vérifie la structure d'un paquet de type M. Les
 * paquets de type M sont ceux renvoyés par la caméra lors du suivi
 * d'une couleur. Ces paquets contiennent les coordonnées de la zone
 * de la couleur donnée, les coordonnées du barycentre ainsi qu'un
 * indice de confiance. Cette fonction va vérifier la structure de
 * manière à ce que la fonction serial_receive_m_command puisse
 * analyser le paquet sans problème.
 *
 * @return 0 si le paquet est bien formé, -1 sinon
 */
int serial_check_m_packet()
{
  int i;
  int spaces = 0;

  /* Les octets 0, 1, 2 ont déjà été vérifiés et valent 'ACK'. L'octet
     3 est un '\r'. On vérifie ici le début d'un paquet M à savoir 'M
     ' (M suivi d'un espace) */
  if(serial_buffer[4] != 'M' || serial_buffer[5] != ' ')
  {
    return -1;
  }

  /* On vérifie que dans les 64 octets du buffer de réception, on
     trouve la fin du paquet, marquée par un '\r' */
```

```
  for (i = 4; i < 64 ; i++)
  {
    if(serial_buffer[i] == '\r')
    {
      break;
    }
  }

  /* Si on a pas trouvé de '\r', on considère le paquet comme
     invalide */
  if(i == 63)
  {
     return -1;
  }

  /* On compte le nombre d'espaces du paquet pour vérifier que le bon
     nombre de paramètre a été reçu */
  for (i = 4; serial_buffer[i] != '\r' ; i++)
  {
     if(serial_buffer[i] == ' ')
     {
       spaces++;
     }
  }

  /* Il faut 8 espaces */
  if(spaces != 8)
  {
     return -1;
  }

  return 0;
}

/** Analyse d'un paquet M
 *
 * Cette fonction analyse le contenu d'un paquet de type M. Cette
 * fonction va extraire 3 valeurs intéressantes du paquet M : les
 * coordonnées du barycentre de la zone de la couleur donnée ainsi que
 * l'indice de confiance.
 *
 * @param x L'adresse à laquelle sera retournée l'abcisse du
 * barycentre
 * @param y L'adresse à laquelle sera retournée l'ordonnée du
 * barycentre
 * @param confidence L'adresse à laquelle sera retournée l'indice de
 * confiance
 *
 * @return 0 si l'analyse du paquet a fonctionné, -1 en cas d'erreur
 */
int serial_receive_m_command(int *x, int *y, int *confidence)
{
   char buf[4];
   int i;
   int j;
   int valx, valy, valconf;
   int ret;

   *x = 0;
   *y = 0;
   *confidence = 0;
```



```
   /* Vérification de la structure du paquet M */
   ret = serial_check_m_packet();
   if(ret < 0)
   {
     return ret;
   }

   /* Réinitialisation du petit buffer temporaire */
   for (j = 0; j < 4 ; j++) buf[j] = 0;

   /* Récupération de la coordonnée 'x' */
   for (i = 6, j = 0 ; serial_buffer[i] != ' ' ; i++, j++)
   {
      buf[j] = serial_buffer[i];
   }

   /* Passer le ' ' */
   i++;

   /* Conversion en entier */
   buf[3] = 0;
   valx = atoi(buf);
   *x = valx;

   for (j = 0; j < 4 ; j++) buf[j] = 0;

   /* Conversion de la coordonnée 'y' */
   for (j = 0 ; serial_buffer[i] != ' ' ; i++, j++)
   {
          buf[j] = serial_buffer[i];
   }

   /* Conversion en entier */
   buf[3] = 0;
   valy = atoi(buf);
   *y = valy;

   /* Passer le ' ' */
   i++;

   /* Coin gauche x => Inutile */
   for ( ; serial_buffer[i] != ' ' ; i++);
   i++;
   /* Coin gauche y => Inutile */
   for ( ; serial_buffer[i] != ' ' ; i++);
   i++;
   /* Coin droite x => Inutile */
   for ( ; serial_buffer[i] != ' ' ; i++);
   i++;
   /* Coin droite y => Inutile */
   for ( ; serial_buffer[i] != ' ' ; i++);
   i++;
   /* Nombre de pixels => Inutile */
   for ( ; serial_buffer[i] != ' ' ; i++);
   i++;

   /* Réinitialisation du buffer */
   for (j = 0; j < 4 ; j++) buf[j] = 0;

   /* Récupération de l'indice de confiance */
   for (j = 0 ; ((serial_buffer[i] != '\r') && (serial_buffer[i] != ' ')) ; i++,
j++)
```

```
   {
      buf[j] = serial_buffer[i];
   }

   /* Passer le ' ' */
   i++;

   /* Conversion en entier */
   buf[3] = 0;
   valconf = atoi(buf);
   *confidence = valconf;

   return 0;
}
```