```
/** @file
 *
 * @brief Déplacement et asservissement du robot
 * @date 2003-2004
 * @author L'équipe Unitec, http://ae.utbm.fr/unitec/
 * @note Code réalisé pour la Coupe de France de Robotique 2004
 * @todo Éliminer le maximum de variables globales, beaucoup sont inutiles
 */

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

/** Nombre de pas des codeurs incrémentaux pour chaque tour */
#define NBRE_PAS 4000

/** Diamètre de la roue libre de droite (en mètre) */
#define D_ROUE_DROIT 0.070145

/** Diamètre de la rouge libre de gauche (en mètre) */
#define D_ROUE_GAUCHE 0.069855

/** Écart entre les deux roues libres (en mètre) */
#define ECART_ROUE_LIBRE 0.0875

/** Écart entre les deux roues motrices (en mètre) */
#define ECART_ROUE_MOTRICE 0.230

/** Périmètre de la roue libre droite (en mètre) */
#define PERIMETRE_DROIT (PI*D_ROUE_DROIT)

/** Périmètre de la roue libre gauche (en mètre) */
#define PERIMETRE_GAUCHE (PI*D_ROUE_GAUCHE)

/** Distance en dessous de laquelle on considère qu'on est arrivé à
    l'objectif défini (en mètre) */
#define DIST_MIN 0.04

/** Gain proportionnel de position qui correspond à l'accélaration à
    donner au robot */
#define KDIST 5000

/** Gain proportionnel pour la rotation */
#define KTETA 1000

/** Gain proportionnel du régulateur de vitesse droit */
#define KP_DROIT 3.0

/** Temps d'intégration droit */
#define TI_DROIT 0.8

/** Gain proportionnel du régulateur de vitesse gauche */
#define KP_GAUCHE 3.0

/** Temps d'intégration gauche */
#define TI_GAUCHE 0.8

/** Période d'échantillonage (en secondes) */
#define TE 0.01

/** Constante de temps des moteurs */
#define TM 0.8
```



```
/** Vitesse maximum de consigne */
#define VMAX 900

/** Angle entre le robot et l'objectif à partir duquel on considère
    que le robot doit d'abord tourner sur lui même avant d'avancer
    vers l'objectif */
#define THETA0 (PI/6)

/** Pente de l'accélération maximale de la consigne moyenne. Avec
    cette constante, il faut 0.8*TM secondes pour passer de 0 à
    VMAX */
#define C (VMAX*TE)/(0.8*TM)

/** Temps maximum de bloquage du robot en nombre d'interruptions */
#define TEMPS_BLOCAGE 100

/** On considère ête bloqué si l'écart entre la moyenne des dernières
    positions et la position actuelle du robot est inférieure à cette
    valeur */
#define DIST_TEST_BLOCAGE 0.002

/** On considère être bloqué si l'écart entre la moyenne des derniers
    angles du robot et l'angle actuel du robot est inférieur à cette
    valeur */
#define TETA_TEST_BLOCAGE ((5*PI)/180)

/** Tableau contenant les TEMPS_BLOCAGE dernières abcisses du robot,
    pour le système d'anti-blocage */
double x_position_backup[TEMPS_BLOCAGE];

/** Tableau contenant les TEMPS_BLOCAGE dernières ordonnées du robot,
    pour le système d'anti-blocage */
double y_position_backup[TEMPS_BLOCAGE];

/** Tableau contenant les TEMPS_BLOCAGE dernièrs angles du robot, pour
    le système d'anti-blocage */
double teta_position_backup[TEMPS_BLOCAGE];

/** Position courante dans les tableaux anti-blocage. Les tableaux
    sont utilisés de manière circulaire */
int anti_block_pos = 0;

/** Mémorisation de la valeur du codeur droit */
int memo_imp_droite= 0;

/** Mémorisation de la valeur du codeur gauche */
int memo_imp_gauche= 0;

/** Abcisse absolue courante du robot */
double x_robot = 0.0;

/** Ordonnée absolue courante du robot */
double y_robot = 0.0;

/** Angle absolu courant du robot */
double teta_robot = 0.0;

/** Abcisse absolue de l'objectif du robot pour le déplacement
   courant */
double x_arrivee;
```

```
/** Ordonnée absolue de l'objectif du robot pour le déplacement
    courant */
double  y_arrivee;

/** Vitesse maximum demandée pour le robot. En cas de rotation, cette
    vitesse sera 0, sinon elle sera égale à VMAX. */
double vitesse_maximum;

/** Gain de rotation demandé pour le robot */
int gain_rotation;

/** État du déplacement. Vaut ASSERV_IDLE si aucun déplacement n'est
    en cours, ASSERV_RUNNING si un déplacement est en cours et
    ASSERV_BLOCK si le robot est bloqué */
int ordre_deplacement = ASSERV_IDLE;


/** Donnée du régulateur proportionnel-intégral pour le moteur droit */
double Integ_Er_droite;

/** Donnée du régulateur proportionnel-intégral pour le moteur gauche */
double Integ_Er_gauche;

/** Différence entre le nombre d'impulsions à l'itération courante et
    à l'itération précédente pour le codeur droit (avancement de la
    roue droite en pas) */
int delta_imp_droite;

/** Différence entre le nombre d'impulsions à l'itération courante et
    à l'itération précédente pour le codeur gauche (avancement de la
    roue gauche en pas) */
int delta_imp_gauche;

/** Vitesse de la roue droite en mètre par seconde */
double Vmes_droite;

/** Vitesse de la roue gauche en mètre par seconde */
double Vmes_gauche;

/** Vitesse de consigne moyenne */
double Vmoy = 0.0;

double ervit1,ervit2;

/** Le timer permettant de mesurer le temps et d'arrêter le robot à la
    fin du match ou de réaliser des temporisations pour les marches
    arrières */
int timer = -1;

/** Calcule la distance en ligne droite entre le robot et le point
 *  d'arrivee
 *
 * @return distance entre le robot et le point d'arrivée en ligne
 * droite en mètre
 */
double Erreur_distance(void)
{
  return sqrt((x_arrivee - x_robot)*(x_arrivee - x_robot) +
              (y_arrivee - y_robot )*(y_arrivee - y_robot));
}

/** Calcule l'angle absolu de la droite passant par le robot et le point d'arriv
```



```
ée
 *
 * @return angle absolu de la droite passant par le robot et le point d'arrivée
 */
double Teta_RobotArrivee(void)
{
  /* Le teta à calculer pour s'orienter vers le point d'arrivee */
  double teta;

  /* La distance entre le robot et le point d'arrivée */
  double distance;

  /* Le cosinus de l'angle entre le robot et le point d'arrivée */
  double cos_angle;


  /* Calcul de la distance robot -> arrivée */
  distance = (sqrt((x_arrivee - x_robot)*(x_arrivee - x_robot) +
                   (y_arrivee - y_robot )*(y_arrivee - y_robot) ));

  cos_angle = (double)( x_arrivee - x_robot)/(double)(distance);

  /* On s'assure que le cosinus de l'angle est bien entre -1 et 1, sinon acos(<
-1) = 0 */
  if(cos_angle < -1)
    {
      cos_angle = -1;
    }
  else if(cos_angle > 1)
    {
      cos_angle = 1;
    }

  /* En fonction du coté du point d'arrivée par rapport au robot, angle négatif
ou positif */
  if((y_arrivee-y_robot)<0)
    {
      teta = -acos(cos_angle);
    }
  else
    {
      teta = acos(cos_angle);
    }

  return teta;
}

/** Calcule l'angle entre la droite passant par le robot et le point
 *  d'arrivée, et la direction du robot
 *
 * @return angle calculé
 */
double Erreur_orientation(void)
{
  double temp;

  temp = (Teta_RobotArrivee() - teta_robot);

  if (temp  > PI)
    {
      temp = temp - (2*PI);
    }
```

```
  else if (temp  < -PI)
    {
      temp = temp + (2*PI);
    }

  return temp;
}

/** Calcul de la nouvelle position du robot
 *
 * Cette fonction calcule la nouvelle position du robot à partir de la
 * mesure de la rotation des roues libres montées sur les codeurs
 * incrémentaux
*/
void positionnement(void)
{

  double delta_moy, delta_diff;
  double dx=0.0, dy=0.0;

  /* Angle de rotation du robot entre deux calcul de d'asservissement */
  double dteta;
  double vit_roue_libre_droite,vit_roue_libre_gauche;
  double rayon_courbure;
  int sens_virage;


  /* Récupération des valeurs des codeurs et calcul de l'avancement de
     chaque roue libre en pas codeur */

  /* Roue droite */
  delta_imp_droite = T4CNT - memo_imp_droite;
  memo_imp_droite = T4CNT;

  /* Roue gauche */
  delta_imp_gauche = T2CNT - memo_imp_gauche;
  memo_imp_gauche = T2CNT;

  delta_moy  = (((double)(delta_imp_droite*PERIMETRE_DROIT +
                          delta_imp_gauche*PERIMETRE_GAUCHE)) /
                (2.0 * ((double)NBRE_PAS)));

  delta_diff = (((double)(delta_imp_droite*PERIMETRE_DROIT -
                          delta_imp_gauche*PERIMETRE_GAUCHE)) /
                ((double)NBRE_PAS));

  dx = delta_moy * cos(teta_robot);
  dy = delta_moy * sin(teta_robot);
  dteta = delta_diff / ((double) ECART_ROUE_LIBRE);

  /* Mise à jour de la position actuelle du robot */
  x_robot = x_robot + dx;
  y_robot = y_robot + dy;
  teta_robot = teta_robot + dteta;

  /* Ajustement de teta_robot pour qu'il soit toujours compris entre
     -PI et PI */
  if (teta_robot > PI)
    {
      teta_robot = teta_robot - (2*PI);
    }
  else if (teta_robot  <= -PI)
```



```
    {
      teta_robot = teta_robot + (2*PI);
    }

  /* Calcul des vitesses des roues motrices */

  /* Roue droite */
  vit_roue_libre_droite = ((double)delta_imp_droite *
                           ((double)PERIMETRE_DROIT)) / ((double)NBRE_PAS);

  /* Vitesse de la roue droite en mètre par seconde */
  vit_roue_libre_droite = vit_roue_libre_droite / TE;

  /* Roue gauche */
  vit_roue_libre_gauche = ((double)delta_imp_gauche *
                           ((double)PERIMETRE_GAUCHE)) / ((double)NBRE_PAS);

  /* Vitesse de la roue gauche en mètre par seconde */
  vit_roue_libre_gauche = vit_roue_libre_gauche / TE;

  /* Le robot va tout droit */
  if (delta_diff == 0)
    {
      Vmes_droite = vit_roue_libre_droite;
      Vmes_gauche = vit_roue_libre_gauche;
    }
  /* Le robot ne va pas tout droit */
  else
    {
        rayon_courbure = delta_moy / dteta;

        /* Virage à gauche */
        if (delta_diff > 0)
          {
            sens_virage = 1;
          }
        /* Virage à droite */
        else
          {
            sens_virage = -1;
          }

        Vmes_droite = ((((double)(sens_virage * ECART_ROUE_MOTRICE/2 + rayon_cou
rbure)) *
                        ((double)(1/(sens_virage * ECART_ROUE_LIBRE/2 + rayon_co
urbure))) *
                       vit_roue_libre_droite);

        Vmes_gauche = ((((double)(-sens_virage * ECART_ROUE_MOTRICE/2 + rayon_co
urbure)) *
                        ((double)(1/(-sens_virage * ECART_ROUE_LIBRE/2 + rayon_c
ourbure))) *
                       vit_roue_libre_gauche);
    }
}

/** Calcule la vitesse moyenne à appliquer au robot
 *
 * @param distance Distance à l'objectif
 * @param Vmoyen Vitesse moyenne
 * @param angle Angle par rapport à l'objectif
 */
```

```
double Vitesse_moyenne(double distance, double Vmoyen, double angle)
{
  double Vemoyen;

  /* Si l'angle entre le robot et le point d'arrivée est trop grand
     (supérieur à THETA0), alors on donne une vitesse de déplacement
     nulle pour que le robot effectue une rotation sur lui même */
  if(( angle>THETA0)||(angle<-THETA0))
    {
      Vmoyen = 0;
    }
  else
    {
      Vemoyen = KDIST * distance;

      /* Saturation de la vitesse moyenne de consigne*/
      if (Vemoyen > vitesse_maximum)
        {
          Vemoyen = vitesse_maximum;
        }

      /* Limitation de l'accélération de l'évolution de la vitesse moyenne */
      if (abs(((int)(Vemoyen - Vmoyen))) < C)
        {
          Vmoyen = Vemoyen;
        }
      else
        {
          if ((Vemoyen - Vmoyen) > 0)
            {
              Vmoyen = Vmoyen + C;
            }
          else
            {
              Vmoyen = Vmoyen - C;
            }
        }
    }

  return Vmoyen;
}

/** Calcule la vitesse à appliquer au moteur droit
 *
 * @param vitesse Vitesse moyenne
 * @param angle Angle avec l'objectif
 *
 * @return consigne à appliquer au moteur droit avant régulation
 */
double Consigne_droite(double vitesse, double angle)
{
  double temp;

  if((angle>=0 && angle<THETA0)||(angle<0 && angle>-THETA0))
    {
      temp = vitesse + KTETA * angle;
    }
  else
    {
      temp = vitesse + gain_rotation * angle;
    }
```



```
  /* Saturation à -1024 */
  if (temp < -1024)
    {
      temp = -1024;
    }

  /* Saturation à 1024 */
  if (temp > 1024)
    {
      temp = 1024;
    }

  return temp;
}

/** Calcule la vitesse à appliquer au moteur gauche
 *
 * @param vitesse Vitesse moyenne
 * @param angle Angle avec l'objectif
 *
 * @return consigne à appliquer au moteur gauche avant régulation
 */
double Consigne_gauche(double vitesse, double angle)
{
  double temp;

  if((angle>=0 && angle<THETA0)||(angle<0 && angle>-THETA0))
    {
      temp = vitesse - KTETA * angle;
    }
  else
    {
      temp = vitesse - gain_rotation * angle;
    }

  /* Saturation à -1024 */
  if (temp < -1024)
    {
      temp = -1024;
    }

  /* Saturation à 1024 */
  if (temp > 1024)
    {
      temp = 1024;
    }

  return temp;
}

/** Calcule l'écart entre la consigne et la mesure de vitesse
 *
 * @param consigne Consigne du moteur
 * @param mesure Mesure effective
 *
 * @return Écart entre la consigne et la mesure de vitesse
*/
double Erreur_vitesse(double consigne, double mesure)
{
  return (consigne - mesure);
}
```

```
/** Regulateur de vitesse droit, fourni la valeur à donner au registre
 *  de PWM
 *
 * @param Er Erreur de vitesse
 *
 * @return Consigne à fournir au registre PWM pour le moteur droit
*/
double Regulateur_PI_droit(double Er)
{
  double Er_corr;

  /* Calcul de l'intégrale sur l'erreur */
  Integ_Er_droite = Integ_Er_droite + Er;

  if (Integ_Er_droite > 1024)
    Integ_Er_droite = 1024;

  if (Integ_Er_droite < -1024)
    Integ_Er_droite = -1024;

  /* Loi de régulation PI */
  Er_corr = KP_DROIT * Er + ((KP_DROIT * TE)/TI_DROIT) * Integ_Er_droite;

  /* Saturations de la commande en vitesse par la PWM */
  if (Er_corr < -1024)
    Er_corr = -1024.0;

  if (Er_corr > 1024)
    Er_corr = 1024.0;

  /* Offset de l'erreur corrigée pour une valeur comprise entre 0 et
     2048 -> valeur à donner au registre de PWM pour la commande de
     vitesse du moteur droit */
  Er_corr = Er_corr + 1024;

  return Er_corr;
}

/** Regulateur de vitesse gauche, fourni la valeur à donner au registre
 *  de PWM
 *
 * @param Er Erreur de vitesse
 *
 * @return Consigne à fournir au registre PWM pour le moteur gauche
*/
double Regulateur_PI_gauche(double Er)
{
  double Er_corr;

  /* Calcul de l'intégrale sur l'erreur */
  Integ_Er_gauche = Integ_Er_gauche + Er;
  if (Integ_Er_gauche > 1024)
    Integ_Er_gauche = 1024;

  if (Integ_Er_gauche < -1024)
    Integ_Er_gauche = -1024;

  /* Loi de régulation */
  Er_corr = KP_GAUCHE * Er + ((KP_GAUCHE * TE)/TI_GAUCHE) * Integ_Er_gauche;

  /* Saturations de la commande en vitesse par la PWM */
  if (Er_corr < -1024)
```



```
    Er_corr = -1024.0;

  if (Er_corr > 1024)
    Er_corr = 1024.0;

  /* Offset de l'erreur corrigée pour une valeur comprise entre 0 et
     2048 -> valeur à donner au registre de PWM pour la commande de
     vitesse du moteur droit */
  Er_corr = Er_corr + 1024;

  return Er_corr;
}

/** Gestionnaire de l'interruption périodique qui s'occupe de gérer
 *  l'asservissement.
 *
 * Cette fonction est la fonction principale de la gestion de
 * l'asservissement et du déplacement.
 *
 * Successivement, elle s'occupe de :
 *  - réactiver les interruptions
 *  - décrémenter la valeur du "timer" global
 *  - mettre à jour la position du robot
 *  - déterminer si l'on est arrivé à l'objectif
 *  - recalculer les consignes à appliquer aux moteurs
 *  - déterminer si l'on est en situation de bloquage contre un obstacle
 */
void c_int2 ()
{
  /* Distance en ligne droite entre le robot et le point d'arrivee */
  double ErDist;

  /* Angle entre la droite passant par le robot et le point d'arrivée,
     et la direction du robot */
  double ErTeta;

  /* Consignes de vitesse à appliquer aux moteurs */
  double Vcons_droite,Vcons_gauche;

  int moyenneidx;
  double moyenne_x;
  double moyenne_y;
  double moyenne_teta;


  asm(" CLRC INTM");

  /* Réaction des interruptions, pour permettre à la réception par le
     port série de fonctionner */
  IFR=IFR | 0x2;
  EVBIFRA=0xFFFF;

  /* Le programme est en fonctionnement, on décrémente le timer */
  if(timer > 0)
    {
      timer--;
    }
  /* On est arrivé à expiration du temps */
  else if(timer == 0)
    {
      /* Arret complet des moteurs */
      CMPR1 = 1024;
```

```
    CMPR2 = 1024;

    /* Désactivation des interruptions */
    asm("     SETC INTM");

    /* Arret du programme */
    while(1);
  }


/* Actualisation de la position du robot */
positionnement();

if (ordre_deplacement == ASSERV_RUNNING)
  {
    ErDist = Erreur_distance();
    ErTeta = Erreur_orientation();

    /* Si la commande était de rester sur place, alors la condition
       d'arret est sur l'erreur d'angle */
    if((vitesse_maximum == 0) && (ErTeta < 0.1) && (ErTeta > -0.1))
      {
        ordre_deplacement = ASSERV_IDLE;
      }

    /* Sinon, la commande est de se déplacer, alors la condition d'arret
       est sur la distance */
    if ((ErDist < DIST_MIN))
    {
      ordre_deplacement = ASSERV_IDLE;
    }
  }

if (ordre_deplacement == ASSERV_RUNNING)
  {
    Vmoy = Vitesse_moyenne(ErDist, Vmoy, ErTeta);

    /* Moteur gauche */
    Vcons_gauche = Consigne_gauche(Vmoy, ErTeta);

    /* 1765 : permet de passer de m/s en valeur du registre PWM,
       consigne étant entre -1024 et 1024, sachant qu'à la valeur de
       vitesse max de 1024 dans le PWM, on a une vitesse de 0.58
       m/s */
    ervit1 = Erreur_vitesse(Vcons_gauche, Vmes_gauche*1765);
    CMPR1 = (int)Regulateur_PI_gauche(ervit1);

    /* Moteur droit */
    Vcons_droite = Consigne_droite(Vmoy, ErTeta);
    ervit2 = Erreur_vitesse(Vcons_droite, Vmes_droite*1765);
    CMPR2 = (int)Regulateur_PI_droit(ervit2);


    /* Système anti blocage : on note les valeurs de la position courante */
    x_position_backup[anti_block_pos] = x_robot;
    y_position_backup[anti_block_pos] = y_robot;
    teta_position_backup[anti_block_pos] = teta_robot;

    /* On incrémente la position dans les tableaux de stockage. Si
       on est arrivé au bout du tableau, on repart à 0 (tableaux
       circulaires) */
    anti_block_pos++;
```



```
    if(anti_block_pos >= TEMPS_BLOCAGE)
      {
        anti_block_pos = 0;
      }

    /* Calcul de la moyenne de chaque coordonnée sur les
       TEMPS_BLOCAGE dernières interruptions */
    moyenne_x = 0.0;
    moyenne_y = 0.0;
    moyenne_teta = 0.0;
    for(moyenneidx = 0; moyenneidx < TEMPS_BLOCAGE; moyenneidx++)
      {
        moyenne_x = moyenne_x + x_position_backup[moyenneidx];
        moyenne_y = moyenne_y + y_position_backup[moyenneidx];
        moyenne_teta = moyenne_teta + teta_position_backup[moyenneidx];
      }
    moyenne_x = moyenne_x / TEMPS_BLOCAGE;
    moyenne_y = moyenne_y / TEMPS_BLOCAGE;
    moyenne_teta = moyenne_teta / TEMPS_BLOCAGE;

    /* Si le robot n'a pas suffisamment bougé en x, en y ou en angle
       sur les TEMPS_BLOCAHE dernières interruptions, alors on
       considère qu'il est bloqué contre un obstacle */
    if ( ( ((moyenne_x - x_robot) < DIST_TEST_BLOCAGE) &&
           ((moyenne_x - x_robot) > -DIST_TEST_BLOCAGE) ) &&
         ( ((moyenne_y - y_robot) < DIST_TEST_BLOCAGE) &&
           ((moyenne_y - y_robot) > -DIST_TEST_BLOCAGE) ) &&
         ( ((moyenne_teta - teta_robot) < TETA_TEST_BLOCAGE) &&
           ((moyenne_teta - teta_robot) > -TETA_TEST_BLOCAGE) ) )
      {
        ordre_deplacement = ASSERV_BLOCK;
      }
  }

 else
  {
    /* Pas d'ordre de déplacement, les moteurs sont à l'arrêt */
    CMPR1 = 1024;
    CMPR2 = 1024;
  }
}

/** Initialisation d'un nouveau déplacement du robot
 *
 * @param coord_x_arrivee Abscisse absolue du point d'arrivée
 * @param coord_y_arrivee Ordonnée absolue du point d'arrivée
 * @param deplace Valeur indiquant si le robot doit se déplacer ou
 * non. Si deplace est à 0, alors le robot ne bougera pas, ce qui
 * permettra d'effectuer des rotations sur lui-même. Sinon, le robot
 * se déplacera pour atteindre l'objectif.
 * @param grotation Gain de rotation, vaut soit
 * ROTATION_VITESSE_NORMALE ou ROTATION_VITESSE_LENTE
 */
void Initialisation_asservissement(double coord_x_arrivee, double coord_y_arrive
e,
                                   int deplace, int grotation)
{
  /* Affectation du nouveau point d'arrivée */
  x_arrivee = coord_x_arrivee;
  y_arrivee = coord_y_arrivee;

  /* Remise à zero des intégrales des lois de régulation PI des moteurs */
```

```
  Integ_Er_droite = 0;
  Integ_Er_gauche = 0;

  /* Éxecution des fonctions d'asservissement */
  ordre_deplacement = ASSERV_RUNNING;
  gain_rotation = grotation;

  /* Vitesse maximum du robot */
  if(deplace == 0)
    {
      vitesse_maximum = 0;
    }
  else
    {
      vitesse_maximum = VMAX;
    }

  /* Remise à zéro du système d'anti-bloquage */
  for(anti_block_pos = 0; anti_block_pos < TEMPS_BLOCAGE; anti_block_pos++)
  {
    x_position_backup[anti_block_pos] = 12.0;
    y_position_backup[anti_block_pos] = 12.0;
    teta_position_backup[anti_block_pos] = 12.0;
  }

  anti_block_pos = 0;
}
```