11/2023

Déplacements aléatoires du robot RAPE

Dans ce cours nous allons vous proposer un programme qui fonctionne de manière aléatoire et qui ne concerne pas les positions de passage seulement une alerte quand les batteries sont presque vides, le cycle s'interrompt sur une fin de mouvement.

L'objectif est d'avoir un robot qui ne s'arrête jamais quelques soient les situations.

Nous en profiterons pour mettre l'aspiration et les brosses actives quand on est mode avance en ligne droite.

Le nombre de pas maxi en ligne droite est fixé.(évite de rester sur une position avec les roues qui patinent jusqu'à épuisement des batteries.)

Le capteur analogique avant (cellule) et les fins de course sur le pare choc avant sont actifs. La cellule avant lors d'une détection d'environ 25 cm réduit le pwm donc la vitesse du robot.

Un blocage par fin de course entraine un recul pour dégager le pare choc.(par tempo).

Il y a trois types de blocage : par la cellule, par le pare choc avant enfin par manque d'impulsion sur les roues.

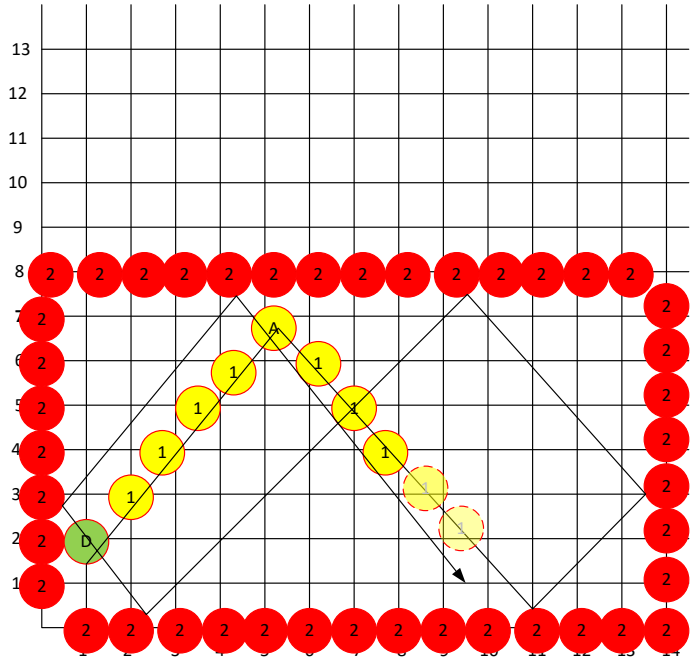
La vitesse moyenne est de 260 mais il peut arriver lors d'une rotation que la puissance ne soit pas suffisante, on a donc mis un système qui augmente la valeur du pwm en fonction du blocage jusqu'à la valeur maximum de 312 ensuite si cela perdure on change d'orientation et on essaie à nouveau.

Toujours basé sur un grafcet.

Table des matières

Nom du programme choisi : ***test_cycle_aleatoire_P7.c***

Principe de déplacement



2 Blocage sur obstacle

1 passage

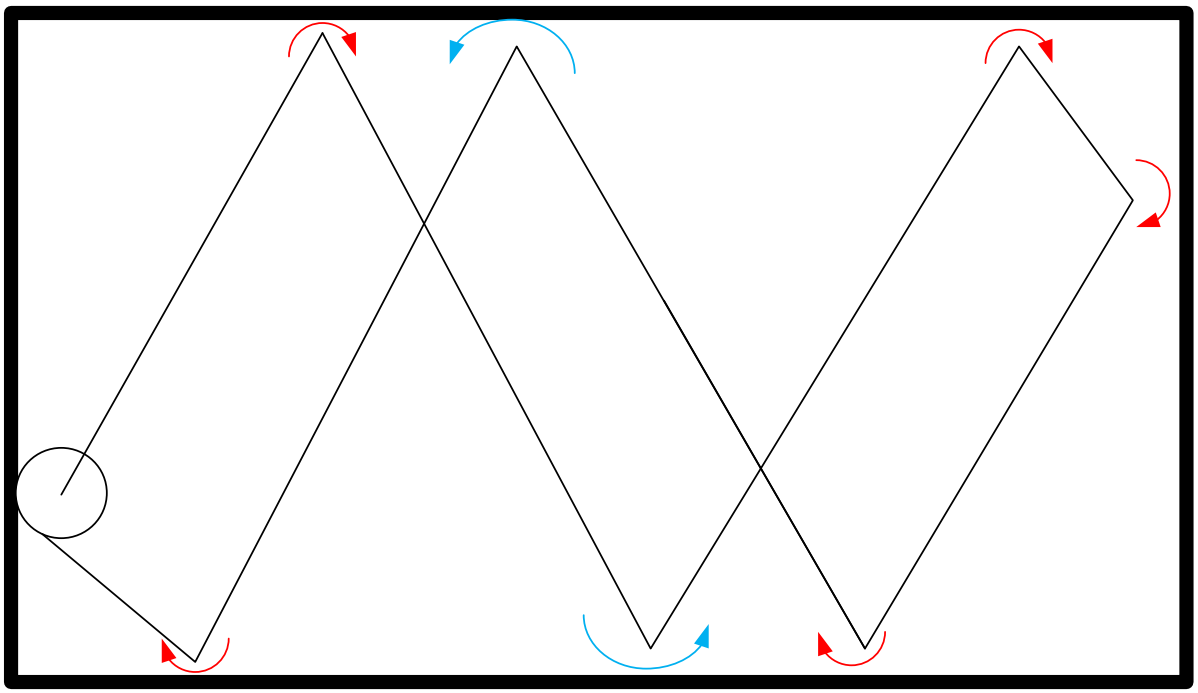
D départ

A chaque fois que l'on rencontre un obstacle on fait un quart de tour. La difficulté est de faire le quart de tour dans le bon sens en effet dans un cas on est en sens horaire dans l'autre cas en sens anti-horaire.

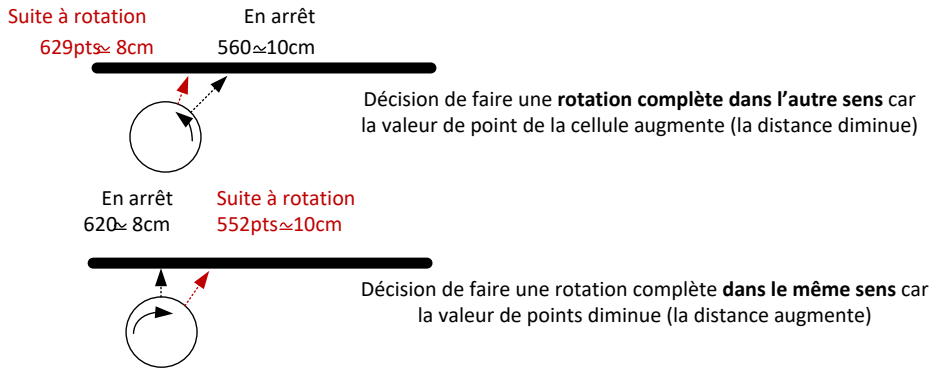
11/2023

horaire

Anti-horaire



On peut voir que les sens horaire ou anti-horaire ne sont pas alternatifs, il nous faudra trouver une autre solution que de faire des rotations alternatives. Le robot fera une petite rotation, si la valeur augmente (l'espace devant la cellule diminue) cela signifie que le robot part dans le mauvais sens donc on changera de sens de rotation. En revanche si la valeur de la cellule diminue(l'espace devant la cellule augmente) cela signifie que l'on est dans le bon sens.



Le programme :

Quelques remarques préliminaires :

On a quelques fois besoin d'un bit à zéro

Exemple : je veux bloquer le programme dans une étape afin de lire une valeur, je mets dans la ligne le bit_a _zero il sera toujours à zéro donc la condition ne sera jamais vraie :

```
if ((copie_etapes_graf_cycle == 2) && fin_tempo_attente_fin_mouvement && bit_a_zero  )
      {etapes_graf_cycle = 3;}
```

acquisition des entrées :

Copie des entrées en début de grafcet afin que le programme ne prenne en considération qu'une seule fois lors d'une scrutation le changement d'état. Cette action est réalisée dans une fonction en début de programme. L'entrée est copiée dans un bit que nous utiliserons dans le programme.

```
void acquisition_entrees(void)
{
  det_AV_ID = entree_det_AV_ID;
  bp1 = entree_bp1;
  bp2=entree_bp2;
        // mesures des entrées analogiques
   capteur_central_avant_AN0 = ADCC_GetSingleConversion(channel_ANA0);
   if (copie_etapes_graf_cycle ==0) {tension_batterie_AN3 =
ADCC_GetSingleConversion(channel_ANA3);
}
```

Restitution des sorties dans une fonction :

Afin d'éviter les marches arrêt intempestifs (déjà expliqué précédemment).

```
void commande_sorties(void)
 {
   sortie_mot_brosse = cde_sortie_mot_brosse;
   sortie_mot_IN1_recul_DR =cde_sortie_mot_IN1_recul_DR;
   sortie_mot_IN2_avance_DR=cde_sortie_mot_IN2_avance_DR;
   sortie_mot_IN3_recul_GA =cde_sortie_mot_IN3_recul_GA;
   sortie_mot_IN4_avance_GA=cde_sortie_mot_IN4_avance_GA;
```

```
sortie_buzzer = bp1;


PWM3_LoadDutyValue( duty_cycle_DR );  //moteur  //
PWM4_LoadDutyValue( duty_cycle_GA ); //
}
```

Remarque sur les fronts montants

Ne pas confondre la fonction avec « FM » en majuscule et le bit qui lui est avec « fm » en minuscule mais qui porte le même nom l'intérêt de ce même nom est dans la recherche dans le programme lors d'un dysfonctionnement.

Commande des leds :

Led bleue :

Allumage fixe si on a un blocage

Allumage clignotant de 0.1s si on a la cellule central analogique qui détecte un arrêt

Allumage clignotant de 0.5s si on a un blocage sur les roues.

```
sortie_led_bleue =(blocage || (blocage_par_detecteur_central && fm_bit_100ms)||  (fin_tempo_impuls_codeur  &&  bit_500ms));
```

La led verte indique toujours le cœur mais quand on est en cycle la fréquence est plus lente :

```
if (fm_bit_100ms || (marche_cy_aleatoire && bit_500ms))  {sortie_led_coeur_ve = !sortie_led_coeur_ve;}
```

télérupteur de mise en service

à chaque impulsion si l'inter est à 1 il passe à zéro sinon c'est l'inverse.

```
if (fm_bp1) {inter_marche = !inter_marche;} // télérupteur
```

mesure de la tension de batterie : *

la valeur analogique de la tension d'entrée batterie de *770=12v*

```
if (tension_batterie_AN3<=470 || !inter_marche ) {marche_cy_aleatoire=0;arret_demande=1;}
if ( tension_batterie_AN3>470 && inter_marche ) { marche_cy_aleatoire=1;arret_demande=0;}//
```

tout le temps que l'entrée sera inférieure à la consigne (470) on aura les valeurs suivantes qui seront mises à zéro et aucun cycle ne pourra être lancé.

```
if (arret_demande) {etapes_graf_cycle = copie_etapes_graf_cycle =0;
cde_avance_un_pas=fin_cde_avance_un_pas
=cde_recul_pas=cde_rotation_45
=start_tempo_attente_fin_mouvement
```

=cde_demi_tour_horaire

=cde_demi_tour_antihoraire

=cde_sortie_mot_brosse=0;}

Gestion du blocage par manque d'impulsion :

Lors de la commande de mouvements, il est possible que les roues soient bloquées. Dans ce cas les roues étant interdépendantes, elles s'arrêtent toutes les 2. Un mécanisme permet d'augmenter jusqu'au maximum la puissance (donc la tension PWM) jusqu'au mouvement détecté sur les codeurs. Au mouvement suivant, la valeur moyenne sera reprise comme valeur en cours.

Analyse du programme manque de puissance :

Dés qu'un mouvement est lancé :

Dans cette ligne on commande dés qu'un mouvement est lancé, le bit *marche_moteurs_DR_GA* est mis à un et lance en dernière ligne la tempo de manque de puissance.

if (cde_avance_un_pas || cde_recul_pas || cde_quart_tour_horaire ||
cde_quart_tour_antihoraire || cde_demi_tour_horaire || cde_demi_tour_antihoraire ||
cde_rotation_45) {marche_moteurs_DR_GA=1;} else {marche_moteurs_DR_GA=0;}

// si le robot à des difficultés à se déplacer on le constate par le manque de points codeur

// alors on augmente le PWM par tranche de 10 toutes les 2 secondes jusqu'au maximum de 312

consigne_tempo_manque_puissance=10; // consigne

<u>compteur de la tempo</u>

dès que le bit *start_tempo_manque_puissance* est actif toutes les 0.1s le compteur est incrémenté.

if (start_tempo_manque_puissance && fm_bit_100ms) { cpt_tempo_manque_puissance++;}

quand le compteur dépasse la consigne on relance la tempo

if (cpt_tempo_manque_puissance > consigne_tempo_manque_puissance)
{fin_tempo__manque_puissance=1 ;manque_puissance_moteurs=1;} //

//à chaque fois que l'on a une impulsion sur les moteurs le compteur est remis à zéro c'est géré dans ext_int.c du MCC Generate Files

<u>reset du bit sur absence de mouvement</u>

if (!marche_moteurs_DR_GA) {manque_puissance_moteurs=0;fin_tempo__manque_puissance;} //

if (marche_moteurs_DR_GA) {start_tempo_manque_puissance=1;} else
{start_tempo_manque_puissance=0;}

augmentations successives des valeurs de puissance à chaque fin de tempo

```
if (fin_tempo__manque_puissance)
                {fin_tempo__manque_puissance=0;vitesse_moyenne_GA=vitesse_moyenne_GA +
                10;vitesse_moyenne_DR=vitesse_moyenne_DR + 10;cpt_tempo_manque_puissance=0;
```

limitation du seuil de puissance pour ne pas dépasser la puissance maxi de 312

```
if (vitesse_moyenne_GA>=312) {vitesse_moyenne_GA=312; puissance_maxi_demande=1;} else
                {puissance_maxi_demande=0;}// gestion du maxi du pwm

if (vitesse_moyenne_DR>=312) {vitesse_moyenne_DR=312; }
```

dès lors que l'on a atteint la puissance maxi et qu'un mouvement n'a pas été détecté on doit changer de stratégie de déblocage aussi le robot on fait une rotation et on relance le processus complet.

```
consigne_tempo_manque_impuls_codeur=10;

 if ( fm_bit_100ms && puissance_maxi_demande)                    { cpt_tempo_manque_impuls_codeur++;}

 if ((cpt_tempo_manque_impuls_codeur > consigne_tempo_manque_impuls_codeur) )
                                                                  {fin_tempo_impuls_codeur=1 ;} //
```

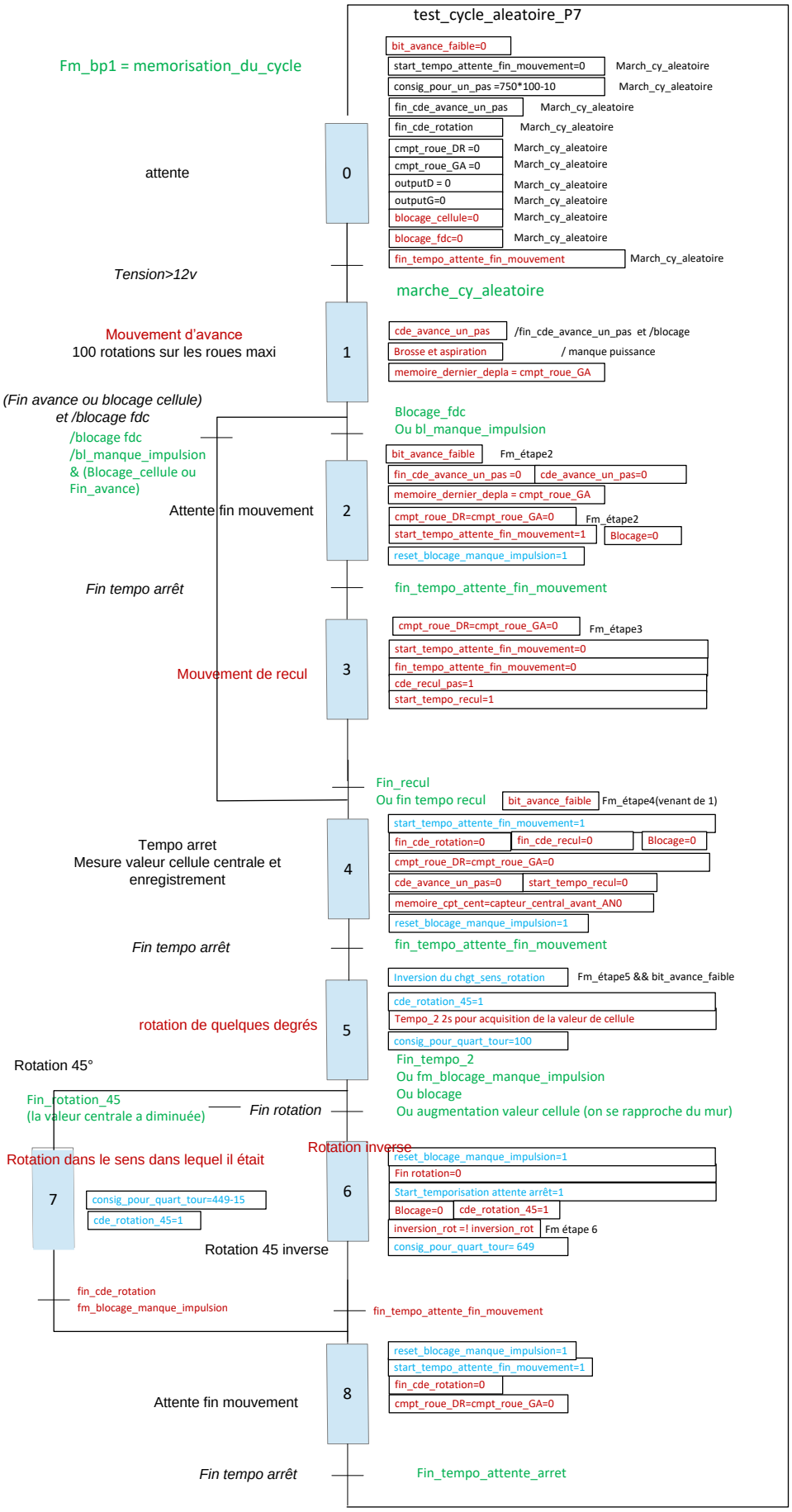
le reset blocage vient des étapes grafcet, on a fait un mouvement de rotation et on relance le processus

```
if ( reset_blocage_manque_impulsion)
 {reset_blocage_manque_impulsion=0;cpt_tempo_manque_impuls_codeur=0;fin_tempo_impuls_codeur=0;}

   if (!puissance_maxi_demande)      {cpt_tempo_manque_impuls_codeur=0;fin_tempo_impuls_codeur=0;}

  blocage_manque_impulsion = fin_tempo_impuls_codeur;
```

réduction de la vitesse lorsque l'on approche d'un mur

```
if (( capteur_central_avant_AN0 > 200) && !manque_puissance_moteurs) {vitesse_demandee_GA =
 vitesse_min_GA;}
```

grafcet


test_cycle_aleatoire_P7
Fm_bp1 = memorisation_du_cycle
bit_avance_faible=0
start_tempo_attente_fin_mouvement=0
March_cy_aleatoire
consig_pour_un_pas =750*100-10
March_cy_aleatoire
fin_cde_avance_un_pas
March_cy_aleatoire
fin_cde_rotation
March_cy_aleatoire
attente
0
cmpt_roue_DR =0
March_cy_aleatoire
cmpt_roue_GA =0
March_cy_aleatoire
outputD = 0
March_cy_aleatoire
outputG=0
March_cy_aleatoire
blocage_cellule=0
March_cy_aleatoire
blocage_fdc=0
March_cy_aleatoire
fin_tempo_attente_fin_mouvement
March_cy_aleatoire
Tension>12v
marche_cy_aleatoire
Mouvement d'avance
100 rotations sur les roues maxi
1
cde_avance_un_pas
/fin_cde_avance_un_pas et /blocage
Brosse et aspiration
/ manque puissance
memoire_dernier_depla = cmpt_roue_GA
(Fin avance ou blocage cellule)
et /blocage fdc
Blocage_fdc
Ou bl_manque_impulsion
/blocage fdc
/bl_manque_impulsion
& (Blocage_cellule ou
Fin_avance)
bit_avance_faible
Fm_étape2
Attente fin mouvement
2
fin_cde_avance_un_pas =0
cde_avance_un_pas=0
memoire_dernier_depla = cmpt_roue_GA
cmpt_roue_DR=cmpt_roue_GA=0
Fm_étape2
start_tempo_attente_fin_mouvement=1
Blocage=0
reset_blocage_manque_impulsion=1
Fin tempo arrêt
fin_tempo_attente_fin_mouvement
cmpt_roue_DR=cmpt_roue_GA=0
Fm_étape3
Mouvement de recul
3
start_tempo_attente_fin_mouvement=0
fin_tempo_attente_fin_mouvement=0
cde_recul_pas=1
start_tempo_recul=1
Fin_recul
Ou fin tempo recul
bit_avance_faible
Fm_étape4(venant de 1)
start_tempo_attente_fin_mouvement=1
Tempo arret
Mesure valeur cellule centrale et
enregistrement
4
fin_cde_rotation=0
fin_cde_recul=0
Blocage=0
cmpt_roue_DR=cmpt_roue_GA=0
cde_avance_un_pas=0
start_tempo_recul=0
memoire_cpt_cent=capteur_central_avant_AN0
reset_blocage_manque_impulsion=1
Fin tempo arrêt
fin_tempo_attente_fin_mouvement
Inversion du chgt_sens_rotation
Fm_étape5 && bit_avance_faible
cde_rotation_45=1
rotation de quelques degrés
5
Tempo_2 2s pour acquisition de la valeur de cellule
consig_pour_quart_tour=100
Rotation 45°
Fin_tempo_2
Ou fm_blocage_manque_impulsion
Ou blocage
Ou augmentation valeur cellule (on se rapproche du mur)
Fin_rotation_45
(la valeur centrale a diminuée)
Fin rotation
Rotation inverse
Rotation dans le sens dans lequel il était
reset_blocage_manque_impulsion=1
Fin rotation=0
Start_temporisation attente arrêt=1
Blocage=0
cde_rotation_45=1
inversion_rot =! inversion_rot
Fm étape 6
consig_pour_quart_tour= 649
6
7
consig_pour_quart_tour=449-15
cde_rotation_45=1
Rotation 45 inverse
fin_cde_rotation
fm_blocage_manque_impulsion
fin_tempo_attente_fin_mouvement
reset_blocage_manque_impulsion=1
start_tempo_attente_fin_mouvement=1
fin_cde_rotation=0
Attente fin mouvement
8
cmpt_roue_DR=cmpt_roue_GA=0
Fin tempo arrêt
Fin_tempo_attente_arret