Documentation du code – Liaison automate et code MSP430

---

1. Introduction

Cette documentation présente le fonctionnement de la communication entre l'automate **Unitronics** et la carte **MSP430**, ainsi que la structure du code côté microcontrôleur.

L'objectif est d'assurer une correspondance claire entre les **messages transmis par UniLogic (automate)** et les **fonctions exécutées sur la carte MSP430**, notamment pour le contrôle des moteurs, la mise en position initiale et la gestion des vitesses.

2. Structure générale et communication

Les différentes commandes envoyées depuis l'automate sont reçues par le MSP430 via le port **UART**. Chaque message est interprété selon son identifiant initial (ex. *P*, *V*, *H*), permettant de déterminer la fonction à exécuter.

Exemple de structure générale d'un message :

```
<Px-y>     // Commande de position
<Vx-y-z>   // Commande de vitesse + mode
<H>        // Home Position
```

Ces messages sont reçus dans une interruption UART, stockés dans un buffer puis analysés dans la fonction `get_command()` qui oriente vers la fonction correspondante :

```c
void get_command(const char *command) {
    switch(command[0]){
        case 'P': set_position_command(command, &robot); break;
        case 'V': set_speed_command(command, &robot); break;
        case 'H': home_position_command(command, &robot); break;
        default: uart_put_string("ERROR - Invalid Command"); break;
    }
}
```

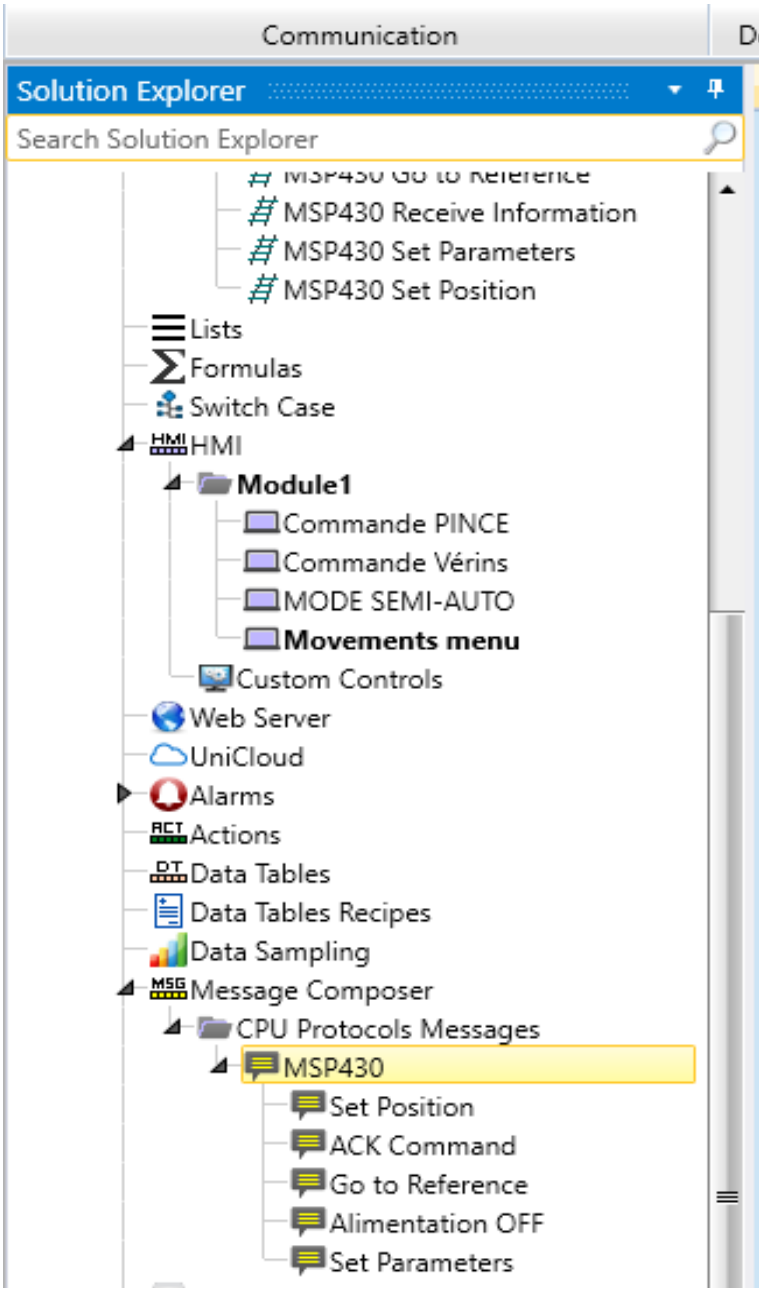

Figure 1 – Organisation des messages dans le Solution Explorer

Cette figure montre la structure du projet UniLogic.

Dans l'arborescence du Solution Explorer, les messages de communication sont regroupés dans ***Message Composer*** → ***CPU Protocols Messages*** → ***MSP430***.

Chaque message correspond à une fonction du code MSP430 (Set Position, ACK Command, Go to Reference, Set Parameters,..). Cette hiérarchie facilite la communication entre l'automate et le microcontrôleur.

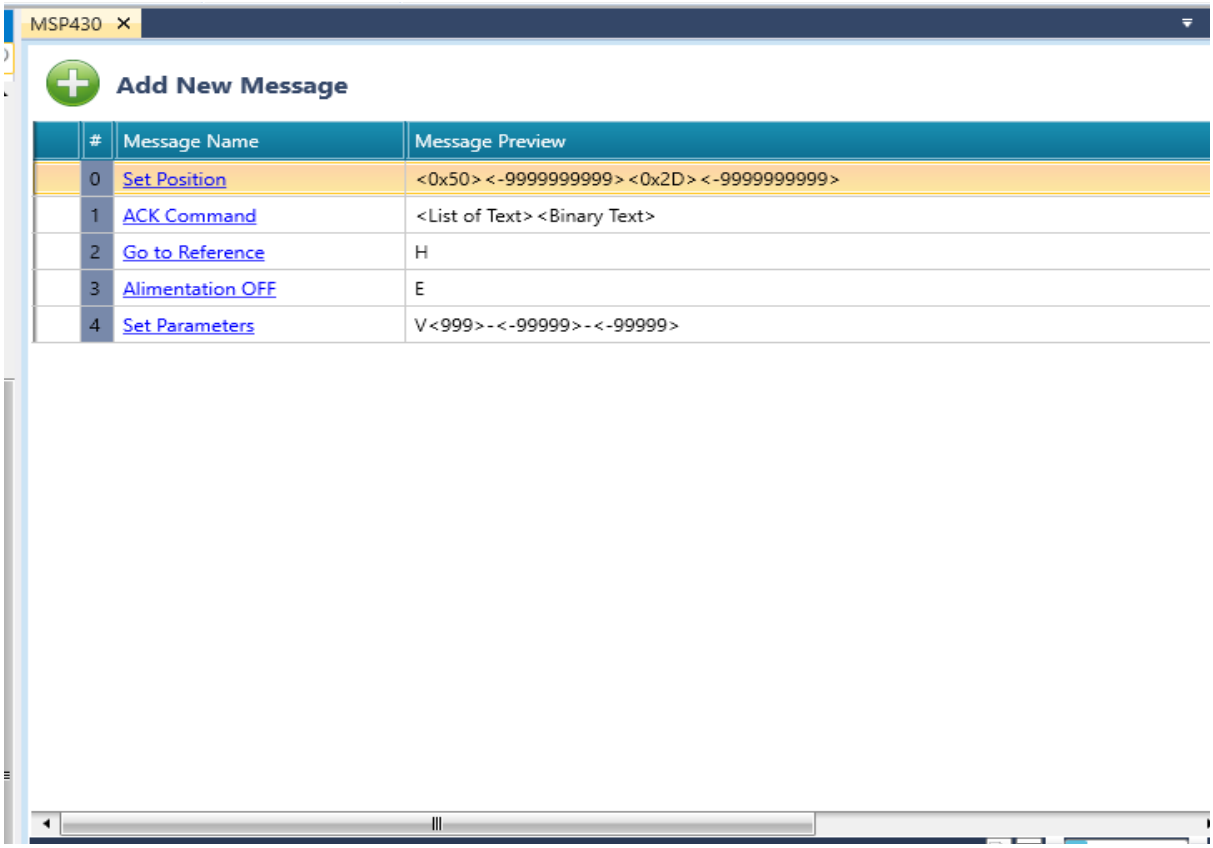

Figure 2 – Liste des messages MSP430 dans UniLogic

Ici, chaque message est défini avec son format d'envoi et sa prévisualisation.
Par exemple, Set Position utilise la structure `<Px-y>`, Set Parameters correspond à `V<999>-<9999>-<9999>`, etc.
Ces formats sont interprétés côté MSP430 dans la fonction `get_command()`, permettant l'exécution de la commande correspondante.

3. Commandes principales et code associé

3.1 Set Position

Cette commande positionne les moteurs X et Y à une coordonnée absolue ou relative.
Le message ***'P%d-%d'*** contient les valeurs X et Y.

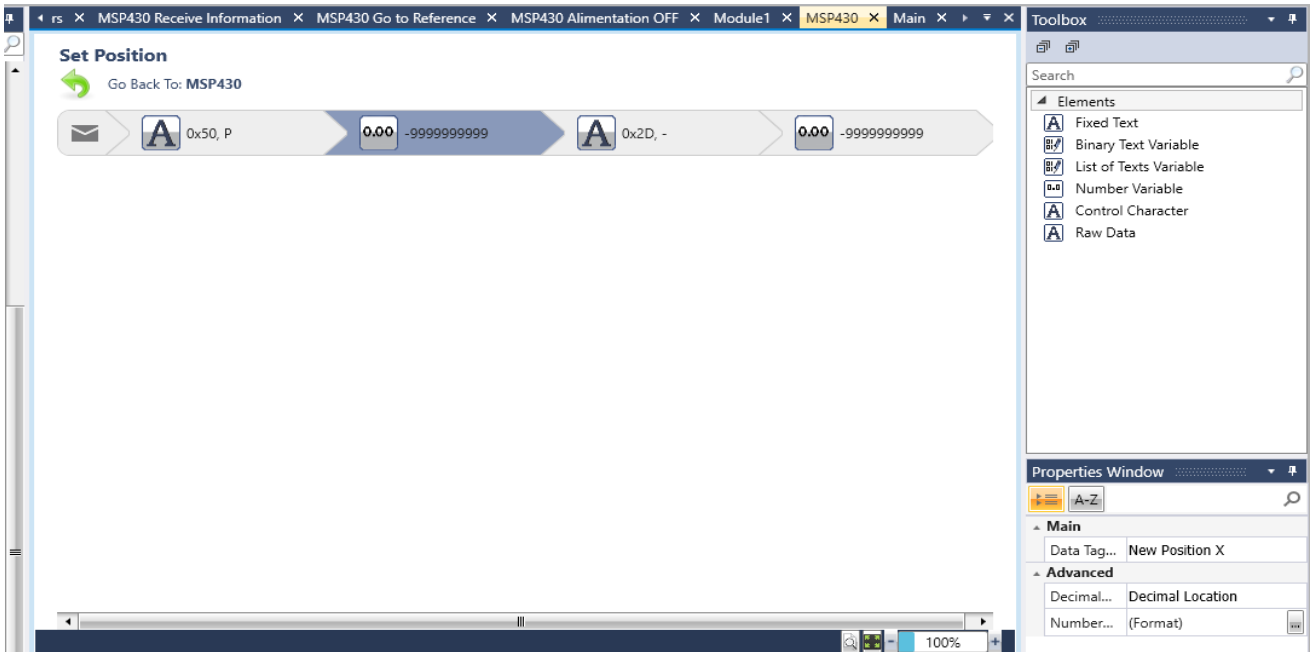

Figure 3 – Liaison de la commande Set Position dans UniLogic

Code associé :

```
void set_position_command(const char *command, Robot *robot) {
    // Lecture de la commande reçue et extraction des coordonnées X et Y
    sscanf(command, "P%d-%d", &(robot->x), &(robot->y));

    // Conversion du déplacement de mm en pas moteur
    int steps_x = mmToStep(robot->x);
    int steps_y = mmToStep(robot->y);

    // Détermination du sens de rotation selon le signe du déplacement
    if (steps_x > 0) {
        GPIO_setOutputLowOnPin(DIR_MX1_PORT, DIR_MX1_PIN);   // Mouvement en
avant (axe X)
        GPIO_setOutputLowOnPin(DIR_MX2_PORT, DIR_MX2_PIN);   // Mouvement en
avant (axe X)
    } else {
        GPIO_setOutputHighOnPin(DIR_MX1_PORT, DIR_MX1_PIN);  // Mouvement en
arrière (axe X)
        GPIO_setOutputHighOnPin(DIR_MX2_PORT, DIR_MX2_PIN);  // Mouvement en
arrière (axe X)
    }

    if (steps_y > 0) {
        GPIO_setOutputHighOnPin(DIR_MY_PORT, DIR_MY_PIN);    // Mouvement en
avant (axe Y)
    } else {
        GPIO_setOutputLowOnPin(DIR_MY_PORT, DIR_MY_PIN);     // Mouvement en
```

```
arrière (axe Y)
    }

    abs_steps_x = steps_x >=0 ? steps_x : -steps_x;
    abs_steps_y = steps_y >=0 ? steps_y : -steps_y;

    // Activation des moteurs et lancement du timer
    enableMotors(1, 1);
    timer_on();

    // Envoi d'un message d'accusé de réception (ACK) à l'automate
    uart_put_string("<P1>"); // ACK de confirmation
}
```

3.2 Home Position

Cette commande permet de renvoyer les moteurs en position initiale (définie par les capteurs de fin de course).

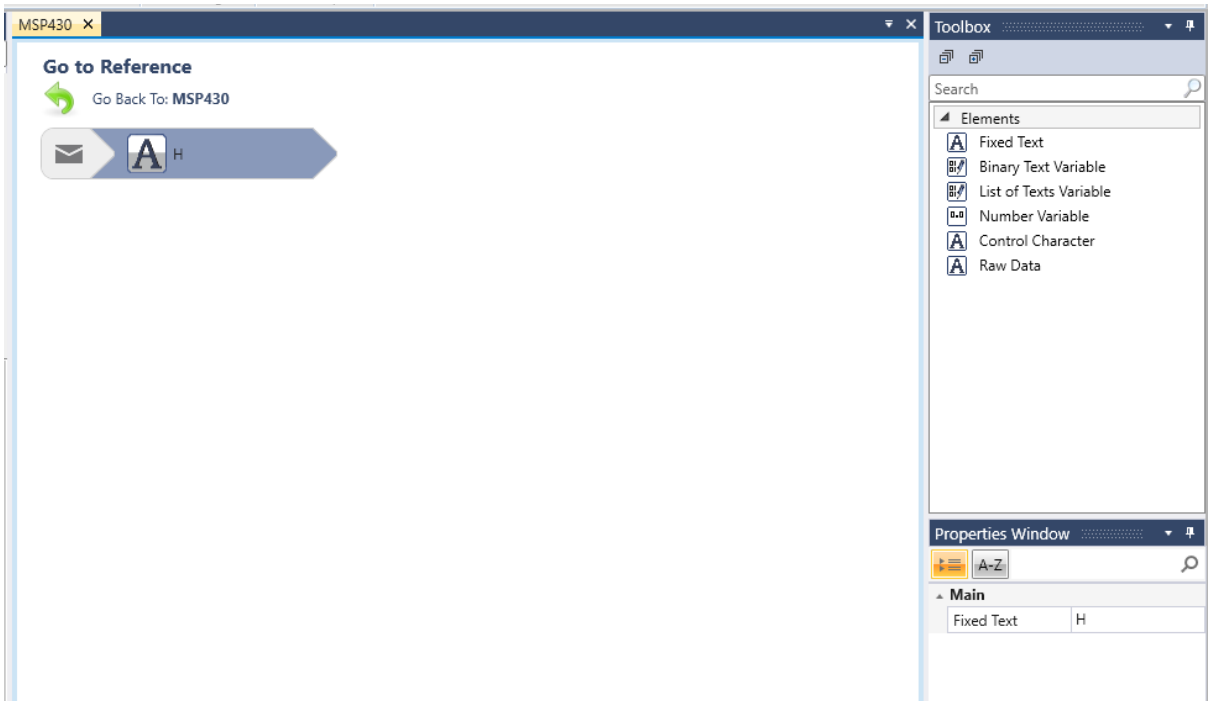

Figure 4 – Commande Home Position et retour capteurs

Code associé :

```
void home_position_command(const char *command, Robot *robot){
    HomePositionXY(1,1);
    abs_steps_x = 1000; abs_steps_y = 1000;
    enableMotors(1,1);
    timer_on();
}
```

La fonction `HomePositionXY()` déclenche le mouvement en arrière jusqu'à détection des capteurs :

```
void HomePositionXY(int flagX, int flagY) {
    if(flagX){
      GPIO_setOutputHighOnPin(DIR_MX1_PORT, DIR_MX1_PIN); //en arrière
      GPIO_setOutputHighOnPin(DIR_MX2_PORT, DIR_MX2_PIN); //en arrière
      enableMotorsX(1);
    }
    if(flagY){
      GPIO_setOutputLowOnPin(DIR_MY_PORT, DIR_MY_PIN); //en arrière
      enableMotorsY(1);
    }
}
```

N.B : cette fonction est à modifier selon la demande du professeur. Le calcul des positions sera effectué côté automate : le code MSP430 recevra directement les valeurs réelles de x et y (enregistrées à chaque déplacement). Ces valeurs permettront de revenir à la position d'origine réelle.

Par exemple, si l'axe x s'est déplacé successivement de 50, puis 70, puis 5 mm, la valeur totale reçue sera 125. Le retour à la source consistera donc à reculer de 125 pas (ou à convertir directement la valeur reçue en pas si elle est déjà exprimée en mm).

3.3 Set Speed Command

Cette commande gère la vitesse des moteurs ainsi que deux modes :
- **Simultaneous / Alternative**
- **Relative / Absolute**

Le message ***'V%d-%d-%d'*** contient respectivement :
- la vitesse,
- le mode simultané/alternatif,
- le mode relatif/absolu.

Figure 5 – Commande Set Speed et structure du message

N.B. prob 1 :

Avant, il n'y avait **aucun appel à cette fonction** dans le Ladder, ce qui empêchait son déclenchement.

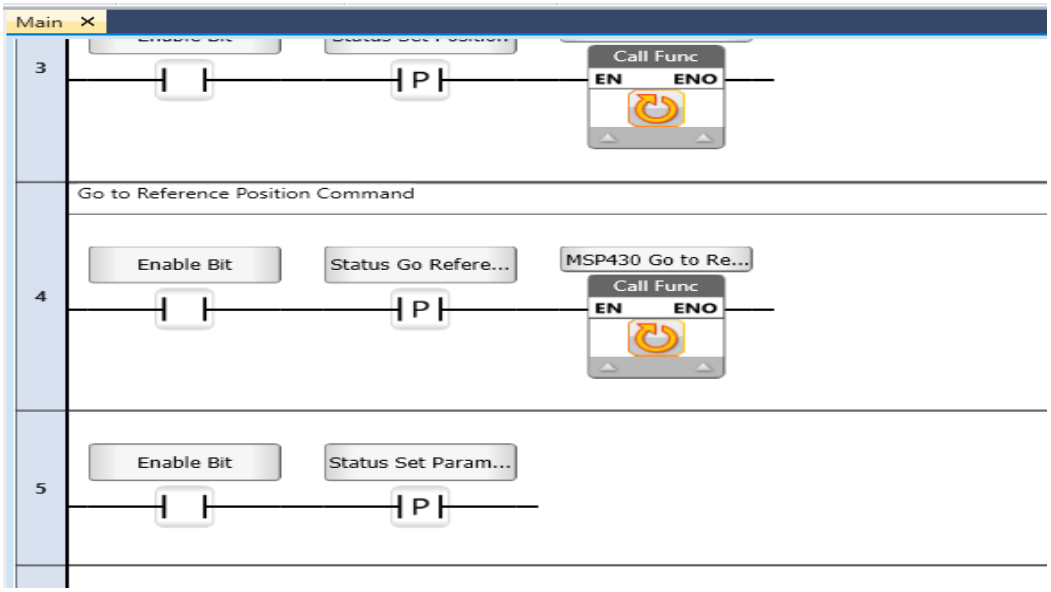

Figure 6 – Absence d'appel à la fonction Set Parameters

Après rectification, l'appel à la fonction a été ajouté dans le Ladder (*MSP430 Set Param*), ce qui permet désormais l'exécution correcte de la commande.

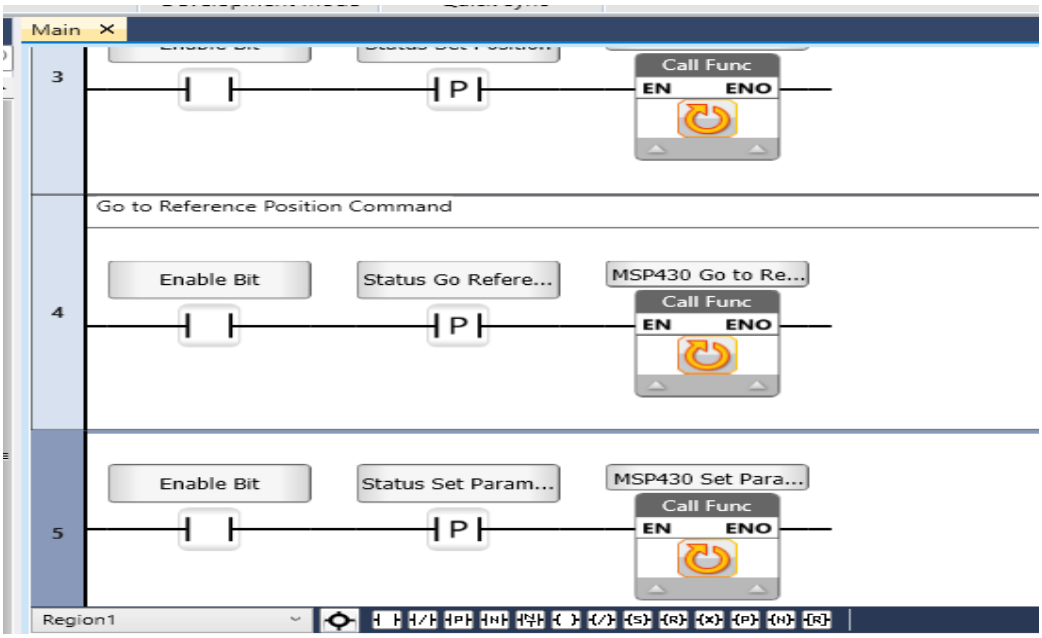

Figure 7 – Présence d'appel à la fonction Set Parameters

N.B. prob 2 :

De plus, au niveau du code, seule la **vitesse** était initialement récupérée, sans les modes de fonctionnement. Le code original était :

```
void set_speed_command(const char *command, Robot *robot){
    int commanded_speed;
    sscanf(command, "V%d", &commanded_speed);

    if(commanded_speed<= MAX_SPEED && commanded_speed >= MIN_SPEED){
        robot->v = commanded_speed;
        timer_off();
        TB0CCR0 = (MM_BY_TOUR*TIMER_FREQ)/(2*STEP_BY_TOUR*robot->v);
        uart_put_string("<S1>"); // ACK message from MSP430 board to PLC
    } else {
        uart_put_string("<S0>"); // ACK message from MSP430 board to PLC
    }
}
```

Après rectification (mais avec améliorations encore possibles) :

```
void set_speed_command(const char *command, Robot *robot) {
    int commanded_speed = 0;
    int mode_sim_alt = 0;
    int mode_rel_abs = 0;

    sscanf(command, "V%d-%d-%d", &commanded_speed, &mode_sim_alt,
&mode_rel_abs);

    if (commanded_speed <= MAX_SPEED && commanded_speed >= MIN_SPEED) {
        robot->v = commanded_speed;
        // → bit 1 = simultaneous/alternative, bit 0 = relative/absolute
        robot->mode = (mode_sim_alt << 1) | mode_rel_abs;

        // Stop and reconfigure timer
        timer_off();
        TB0CCR0 = (MM_BY_TOUR * TIMER_FREQ) / (2 * STEP_BY_TOUR * robot->v);

        /* À ajouter : gestion du comportement selon le mode :
        if (mode_sim_alt == 0) {
            // Simultaneous mode → X et Y bougent ensemble
        } else {
            // Alternative mode → X puis Y
        }

        if (mode_rel_abs == 0) {
            // Relative mode → déplacement par rapport à la position
actuelle
        } else {
```

```
            // Absolute mode → déplacement vers la position donnée
        }*/

        uart_put_string("<S1>"); // ACK OK
    } else {
        uart_put_string("<S0>"); // Erreur : vitesse hors limites
    }
}
```

3.4 ISR Fin de course (PORT1_VECTOR)

Cette interruption permet d'arrêter automatiquement les moteurs lorsqu'un capteur de fin de course est activé.

Code associé :

```
#pragma vector = PORT1_VECTOR
__interrupt void ISR_CAPTEURS_FDC(void){
    if(P1IFG & FCX_PIN){ enableMotorsX(0); robot.x = 0; }
    if(P1IFG & FCY_PIN){ enableMotorsY(0); robot.y = 0; }
    P1IFG &= ~(FCX_PIN | FCY_PIN);
}
```

4. Problèmes rencontrés et améliorations

1. **Problème de synchronisation initiale** : certaines commandes (*Set Parameters*) ne se déclenchaient pas car la fonction n'était pas encore appelée dans le code principal.
   Correction : ajout de l'appel à la fonction correspondante et gestion des modes via `sscanf()`.

2. **Limitation du calcul de position pour retour à l'origine** : la version initiale ne tenait pas compte de l'enregistrement cumulatif des positions X/Y.
   Correction prévue : récupérer directement la valeur totale X/Y de l'automate et effectuer le retour en pas selon cette valeur.

3. **Validation de la communication UART** : ajout de messages de confirmation *ACK* (`<S1>`, `<P1>`) pour vérifier la réception correcte des ordres.

5. Conclusion

Cette documentation prouve la mise en place d’une communication fonctionnelle entre l’automate **Unitronics** et la carte **MSP430**.
Les prochaines évolutions concerneront les issues liées au retour à l’origine ainsi que la gestion dynamique des positions absolues/relatives et l’intégration de modes de déplacement simultanés/alternatifs plus avancés.