Sujet Travaux pratiques

Site: MOODLE ENSEA
Cours: 2526-S9-ESE_3-Actionneur et automatique appliquée
Livre: Sujet Travaux pratiques

Imprimé par: Houssam Hakki (3ESE)
Date: lundi 1 décembre 2025, 14:53

Table des matières

1. Objectifs, livrables et évaluation.

A partir d'un hacheur complet et d'une carte Nucleo-STM32G474RE, vous devez :

1. Réaliser la commande des 4 transistors du hacheur en commande complémentaire décalée,
2. Faire l'acquisition des différents capteurs,
3. Réaliser l'asservissement en temps réel.

Les livrables demandés :

- Un dépôt github partagé dès la première minute avec l'enseignant (nicolas.papazoglou@ensea.fr ou alexis.martin@ensea.fr) avec le nom : **2526_ESE_AAA_<nom1>_<nom2>**
- Tout le projet stm32cubeIDE **synchronisé sur un github à chaque séance et dès la première séance de TP**,
- Un fichier **readme du Github à valeur de compte-rendu** reprenant l'ensemble des éléments demandés et des informations importantes **complété à chaque séance**.
- Un code propre et documenté.

L'évaluation aura lieu tout au long des séances de TP sur plusieurs points :

- Avancement dans le sujet,
- Aptitude a réutiliser les notions vue en cours,
- Habilité à aller chercher dans la documentation les informations utiles,
- Qualité de la documentation,
- Lisibilité du code et qualité du code (mise en place de fonctions, factorisation du code, choix des noms de variables/fonctions explicites)
- Présence et ponctualité,

2. Bonnes pratiques

Pour une meilleure lisibilité du code :

- Toutes les variables sont écrites en minuscule avec un underscore pour séparer deux mots si besoin dans le nom de la variable, par exemple : *uart_rx_buffer*
- Les directives (macro avec #define) sont écrites en majuscule, par exemple : *#define UART_TX_SIZE*
- **Aucun nombre magique ne peut exister**, par exemple : remplacer if(value == 512) par if(value == VAL_MAX) avec #define VAL_MAX 512
- Si vous écrivez deux fois le même code, il faut écrire une fonction pour **factoriser votre code**.
- Il faut **tester vos appels de fonction**, les fonctions HAL renvoient pour la plupart du temps une valeur pour savoir si son exécution s'est déroulé correctement. Tester les erreurs possibles et les gérer le cas échéant.

3. Github

1. Télécharger les fichiers de base : https://github.com/DBXYD/2526_MSC_SAC
    - **git clone git@github.com:DBXYD/2526_ESE_AAA.git**
2. Créer un projet github, et synchroniser votre projet CubeIDE avec :
    - Sur GitHub, créer un nouveau repository
    - Récupérer l'URL (https://....)
    - Ouvrir GitBash
    - Se placer dans le répertoire dans lequel sera créé votre Git
    - Récupérer la configuration du projet git :
        - git clone XXX
    - Copier dans ce dossier les fichiers téléchargés depuis le github de l'enseignant
    - Ouvrir CubeIDE
    - Ouvrir le projet
    - Pour mettre à jour votre GitHub :
        - git add .
        - git commit -m "Nom_de_la_modif"
        - git push

4. app.c et organisation des fichiers

App.c

Ce fichier sert principalement à initialiser tous les autres modules du projet. Il fournit une fonction *init_device* dont le rôle est de configurer les différents sous-systèmes comme l'interface utilisateur, les entrées analogiques, les moteurs ou encore les capteurs. Dans l'état actuel, seules certaines parties sont actives, les autres sont commentées mais montrent ce qui devra être initialisé plus tard par vos soins.

init_device

Configuration du shell

Dans init_device, la première étape configure le shell, c'est-à-dire l'interface texte accessible via l'UART. Pour cela, les fonctions qui permettent d'envoyer et de recevoir des caractères via l'UART2 sont associées à la structure du shell. Ensuite le shell est initialisé, ce qui crée les commandes de base. Une interruption de réception UART est démarrée afin que le microcontrôleur appelle automatiquement une routine lorsqu'un caractère est reçu.

Deux fonctions permettent de relier concrètement le shell au matériel UART. La fonction *shell_uart2_transmit* utilise HAL_UART_Transmit pour envoyer des données via l'UART2. La fonction *shell_uart2_receive* lit simplement le caractère qui vient d'être reçu et stocké par l'interruption.

La fonction HAL_UART_RxCpltCallback est appelée automatiquement par la bibliothèque HAL lorsqu'un caractère est reçu sur l'UART2. Elle relance immédiatement la réception d'un prochain caractère, puis appelle *shell_run* pour permettre au shell de traiter ce qui vient d'être reçu. C'est de cette manière que le shell fonctionne en tâche de fond, sans bloquer le reste du programme.

Le shell fait l'interface entre les fonctions écrites par le développeur et le hardware, à aucun autre moment l'uart 2 doit être spécifié dans votre code afin de le rendre portable.

Configuration de la LED

Ensuite, la LED est initialisée avec *led_init*.

Autres configurations

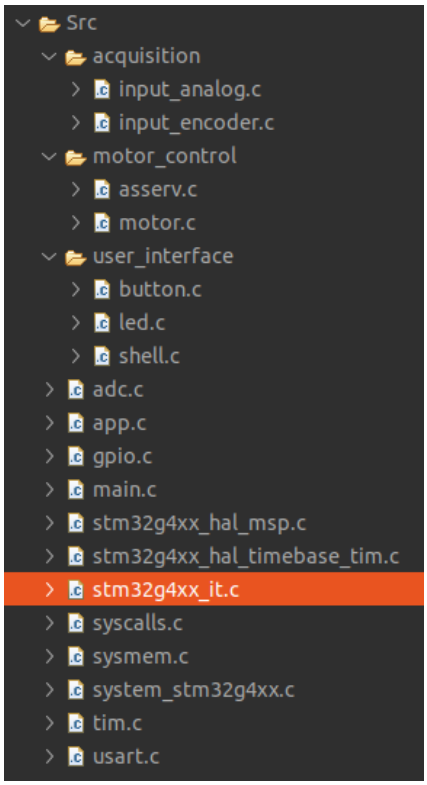
D'autres initialisations comme les boutons, la partie moteur, les correcteurs PID ou les entrées analogiques sont indiquées mais laissées en commentaire. Elles seraient traitées de manière similaire : chaque module fournit une fonction d'initialisation appelée depuis *init_device*.

loop

Enfin, une fonction *loop* est définie mais ne contient rien pour le moment. Elle est destinée à accueillir plus tard des fonctions principales de l'application qui seront exécutées à interval de temps régulier.

Organisation des fichiers

L'organisation du code est basée sur une séparation claire par grandes fonctions du système. Chaque dossier représente un sous-ensemble cohérent du projet, correspondant à un domaine fonctionnel précis du système embarqué.



Acquisition

Le dossier acquisition contient les fichiers associés à la récupération des données issues du matériel. *input_analog* regroupe tout ce qui concerne la mesure des tensions ou courants via les convertisseurs analogiques. *input_encoder* concerne les codeurs incrémentaux. Ces modules ont pour rôle de fournir au reste du système des mesures propres et utilisables.

Motor_control

Le dossier *motor_control* contient les modules liés à l'action sur le moteur. Le fichier *motor.c* gèrera les signaux PWM, l'activation du pont de puissance ou tout autre élément nécessaire au pilotage du moteur. Le fichier *asserv.c* met en œuvre les algorithmes de contrôle (PID) pour réguler la vitesse à partir des mesures fournies par l'acquisition.

User_interface

Le dossier *user_interface* regroupe les éléments en contact direct avec l'utilisateur. *button.c* contient la gestion des boutons physiques et de leur détection fiable. *led.c* contient la gestion des voyants lumineux, comme une LED d'état. *shell.c* contient le code du terminal texte accessible via l'UART, permettant à l'utilisateur d'envoyer des commandes ou de lire des informations.

Cette organisation permet de séparer proprement les fonctionnalités selon leur rôle et facilite la maintenance, l'évolution et la réutilisation éventuelle des modules. Chaque dossier regroupe une logique cohérente, ce qui permet de travailler sur un aspect du système sans perturber les autres.

5. Shell

Le module présenté met en place un terminal utilisable via l'UART, ce qu'on appelle un shell. Il permet à l'utilisateur de taper une commande sous forme de texte et d'exécuter automatiquement la fonction associée.

shell_init

La fonction shell_init prépare le shell avant utilisation. Elle efface les anciennes commandes enregistrées, affiche un texte d'accueil, puis ajoute deux commandes intégrées : help et test. Cette fonction doit être appelée une seule fois au démarrage du programme. Cela est déjà fait dans le fichier app.c.

shell_add

La fonction shell_add permet d'ajouter une nouvelle commande. Elle enregistre trois informations : le nom de la commande (par exemple "led"), la fonction à exécuter lorsque l'utilisateur tape ce nom dans le terminal, et une description utilisée par la commande help. Cette fonction est en général appelée juste après l'initialisation du shell, dans le main, pour chacune des commandes que tu veux rendre disponibles.

shell_run

La fonction shell_run est celle qui fait fonctionner le shell en continu. Elle lit les caractères que l'utilisateur envoie via l'UART, elle gère le retour à la ligne, le backspace et l'écho des caractères. Quand l'utilisateur appuie sur Entrée, cette fonction déclenche l'exécution de la commande. Dans notre programme, cette fonction est appelé depuis une interruption qui survient lors de la réception d'un caractère (depuis HAL_UART_RxCpltCallback).

shell_exec

La fonction shell_exec est interne. Elle prend la ligne de commande complète, sépare le nom de la commande et les arguments, cherche si la commande a été enregistrée via shell_add, et appelle la fonction associée. Si la commande n'existe pas, un message d'erreur est envoyé. **L'utilisateur du module n'appelle jamais directement shell_exec**, c'est shell_run qui s'en charge.

sh_help

La fonction sh_help est une commande intégrée. Lorsqu'on tape "help" dans le terminal, elle affiche la liste des commandes enregistrées ainsi que leur description. La fonction sh_test_list est une autre commande intégrée, utilisée pour montrer comment les arguments d'une commande sont transmis. Par exemple, si l'on tape "test a b c", elle affichera chaque argument reçu.

is_character_valid et is_string_valid

Les fonctions is_character_valid et is_string_valid servent en interne à vérifier que les caractères et les noms de commandes sont valides. Elles ne sont pas destinées à être utilisées directement.

Conclusion

Pour utiliser ce mini-shell, on écrit :

- une fonction correspondant à une commande, par exemple une fonction pour allumer ou éteindre une LED (voir led.c)
- Ensuite, après l'appel à shell_init, on ajoute cette commande avec shell_add.

Une fois ceci en place, l'utilisateur peut taper par exemple "help", "test x y", ou toute commande que tu auras ajoutée.

6. Commande MCC basique

Commande de MCC - niveau basique

Objectifs :

- Générer 4 PWM en complémentaire décalée pour contrôler en boucle ouverte le moteur en respectant le cahier des charges,
- Inclure le temps mort,
- Vérifier les signaux de commande à l'oscilloscope,
- Prendre en main le hacheur,
- Faire un premier essai de commande moteur.

6.1. Génération de 4 PWM

Générer quatre PWM sur les bras de pont U et V pour controler le hacheur à partir du timer déjà attribué sur ces pins.

Cahier des charges :

- Fréquence de la PWM : 20kHz
- Temps mort minimum : à voir selon la datasheet des transistors (faire valider la valeur)
- Résolution minimum : 10bits.

Pour les tests, fixer le rapport cyclique à 60%.
Une fois les PWM générées, les afficher sur un oscilloscope et les faire vérifier par votre professeur.

6.2. Commande de vitesse

Pour controler la vitesse du moteur, nous allons envoyer une séquence via la liaison UART (par l'USB) de la forme :

```
speed XXXX
```

Le traitement de cette chaine de caractère se faire de la manière suivant :

- Détection des premiers caractères "speed"
- Conversion de tous les caractères représentant des chiffres XXXX en nombre entier
- Vérification de la valeur (si la valeur est supérieur au maximum autorisé (bien spécifier cette valeur), on l'abaisse à cette valeur),
- Application de cette vitesse au moteur à travers le registre gérant le rapport cyclique de la PWM

6.3. Premiers tests

Brancher le moteur en respectant les données constructeur du moteur.

Faites vérifier l'ensemble des signaux par votre professeur.

Faire un premier test dans les conditions suivantes (dans l'ordre) :

- Rapport cyclique de 50%
- Rapport cyclique de 70%

Quels problèmes observez vous ?

Pour palier à ce problème, générer une montée progressive du rapport cyclique jusqu'à arriver à la vitesse cible commandé par la commande définie précédemment.

7. Commande en boucle ouverte, mesure de Vitesse et de courant

Dans cette partie vous devez :

- Commander en boucle ouverte le moteur avec une accélération limitée,
- Mesurer le courant aux endroits adéquat dans le montage,
- Mesurer la vitesse à partir du codeur présent sur chaque moteur.

7.1. Commande de la vitesse

Rajouter quelques fonctionnalités à votre projet :

- Commande *start* : permet de fixer le rapport cyclique à 50% (vitesse nulle) et d'activer la génération des pwm (HAL_TIM_PWM_Start et HAL_TIMEx_PWMN_Start),
- Commande *stop* : permet de désactiver la génération des PWM.
- Commande speed XXXX : permet de définir le rapport cyclique à XXXX/PWM_MAX, mais afin de réduire l'appel à courant, vous devez établir une montée progressive à cette vitesse en quelques secondes. Vous pouvez effectuer une rampe entre la valeur actuelle et la valeur cible avec un incrément bien réfléchi de la PWM à un intervalle de temps régulier. Par la suite votre asservissement fera cela tout seul.

7.2. Mesure de courant

A partir de la documentation (schéma KiCad) :

- Définir quel(s) courant(s) vous devez mesurer,
- Définir les fonctions de transfert des capteurs de mesure de courant (lecture datasheet),
- Déterminer les pin du stm32 utilisés pour faire ces mesures de courant,
- Etablir une première mesure de courant avec les ADC en Pooling. Faites des tests à vitesse nulle, non nulle, et en charge (rajouter un couple resistif en consommant du courant sur la machine synchrone couplée à la MCC).
- Une fois cette mesure validée, modifier la méthode d'acquisition de ces données en établissant une mesure à interval de temps régulier avec la mise en place d'une la chaine d'acquisition Timer/ADC/DMA.
- Vous pouvez utiliser le même timer que celui de la génération des PWM pour que les mesures de courant soit synchrone aux PWM. Pour vérifier cela, utiliser un GPIO disponible sur la carte pour établir une impulsion lors de la mesure de la valeur.

7.3. Mesure de vitesse

A partir de la documentation (schéma KiCad, datasheets et expérimentation) :

- Déterminer la fonction de transfert du capteur de vitesse,
- Déterminer la constant de temps mécanique du moteur :
  - Pour cela, vous pouvez envoyer un échelon de tension au moteur et analyser la vitesse à partir de la sonde tachymétrique (que l'on n'utilisera pas pour le reste du TP).
- Déterminer les pin du stm32 utilisés pour faire cette mesure de vitesse,
- Déterminer la fréquence à laquelle vous allez faire l'asservissement en vitesse du moteur.
- Etablir le code de mesure de vitesse et le tester.