Introduction aux miniprojets

Francesco Mondada, Daniel Burnier
IEM - STI - EPFL

Programme (rappel)

Faire encore un TP pour se roder et passer en revue les différents capteurs
Donner plus de temps au projet, pas d'examen (selon plan)

22.Feb.22		Cours 1		12.Apr.22	**Examen théorie**
24.Feb.22		Intro + TP		14.Apr.22	miniprojet
01.Mar.22		Cours 2		19.Apr.22	Vacances de Pâques
03.Mar.22		TP		21.Apr.22	Vacances de Pâques
08.Mar.22		Cours 3		26.Apr.22	miniprojet
10.Mar.22		TP		28.Apr.22	miniprojet
15.Mar.22		Cours 4		03.May.22	miniprojet
17.Mar.22		TP		05.May.22	miniprojet
22.Mar.22		Cours 5		10.May.22	miniprojet
24.Mar.22		TP		12.May.22	miniprojet
29.Mar.22		Cours 6		17.May.22	présentations miniprojet
31.Mar.22		TP		19.May.22	présentations miniprojet
05.Apr.22		**Test à blanc**		24.May.22	présentations miniprojet
07.Apr.22		miniprojet		26.May.22	ascension
				31.May.22	présentations miniprojet
				02.Jun.22	présentations miniprojet

Donnée:

Le but du miniprojet est de partir sur la base des éléments que vous aurez vu lors des TPs 1-5 pour créer plusieurs tâches plus complexes à résoudre par le robot e-puck2.

Donnée:

Vous êtes libres de déterminer vous-même les tâches que le robot doit effectuer, et donc la forme de la démonstration de votre programme. Les contraintes sont les suivantes :

1) Le projet doit être fait sur la base de la librairie e-puck2_main-processor vue lors des TPs 4-5.

2) Vous devez obligatoirement utiliser les éléments suivants du robot e-puck2 dans votre projet :

a. **Les deux moteurs pas-à-pas**. Par exemple Régulation PID, odométrie précise, forme géométrique, etc.

b. **Un des capteurs de distance** (Capteurs de proximités infrarouges ou capteur Time-of-Flight). Par exemple détection d'obstacle petite ou grande distance.

c. **Un capteur parmi ceux que vous avez investigués pendant les TPs 3-5, donc un capteur parmi : la caméra, les micros, l'IMU**. Par exemple avec la caméra : détection d'objet, suivi de lignes. Par exemple avec les micros : détection de sons sur la base de l'amplitude et/ou fréquence. Par exemple avec l'accéléromètre, détection d'un plan incliné, d'un choc. Par exemple avec le gyroscope, détection du mouvement d'un plan incliné.

Donnée:

3) Chaque capteurs/actuateurs doit être géré par un Thread dédié dans le code.

4) Le code doit être rendu sous la forme d'une librairie (avec divers .c/.h) qui s'intègre avec la librairie ChibiOS que vous avez utilisée lors des Travaux pratiques.

Voici donc un exemple de projet : Le robot e-puck2 reproduit les mouvements d'une balle sur un plan incliné en utilisant l'accéléromètre comme détecteur de la direction du vecteur gravité. Le robot "rebondit" lorsqu'il détecte un obstacle avec ses capteurs de proximité infrarouges.

Forme du rendu:

Le rendu du miniprojet sera composé de:

- Une démonstration du programme final, d'une durée de deux minutes
- Un rapport de 3-4 pages qui donne un aperçu de la méthode de travail, des analyses, la conception du logiciel et des résultats obtenus.
- Le code structuré et commenté
- Une discussion sur le projet lors du rendu/presentation
  + demo = Durée totale de passage ~15 minutes

Critères d'évaluation:

Les critères d'évaluations seront :

- Clarté et propreté du code source (respect de certaines conventions du C, des commentaires utiles, des valeurs définies, des fonctions et variables avec des noms clairs, etc.)
- Utilisation de GitHub (nous donner accès, prevue d'un certain nombre de commits séparés par les members du groupe, projet final sur GitHub)
- Clarté et propreté du rapport (**Numéros de sections, légendes aux figures, axes aux graphiques, citations et références**)
  Jusqu'à un point enlevé sur la note pour des erreurs de ce type
- Efficacité du code en termes d'utilisation des ressources (temps, mémoire).
- Originalité de la démonstration.

Miniprojets

Commencez par comprendre et structurer!!!

Code: attention aux warnings, temps d'exécution, ressources utilisées, etc.

Rapport:

- seulement ce qui n'est pas dans le code, comme structure générale, tests initiaux, graphes de capteurs, etc
- **Respectez le format (sections numérotées, figures et tables numérotées avec légendes, axes sur les graphes, références)**

Etape 1: Définir

Commencez par définir votre projet sur papier:

- **quels capteurs vous allez utiliser:** est-ce que vous êtes au clair sur les caractéristiques? Est-ce que vous devez valider des aspects?
- **quelle application vous voulez faire:** quels choix sont délicats, où est-ce qu'il peut y avoir des problèmes? Quelle démo finale? Faisable par vidéo interposée
- **quelles ressources vous voulez utiliser:** quelles représentations de nombres, mémoire, FPU, DSP, périphériques?

Evitez de changer votre projet en cours de route, sinon vous n'allez pas réussir à faire quelque chose de propre.

Etape 2: Organiser

Avant d'écrire le code et les algorithmes en détail, prenez le temps de planifier SUR PAPIER l'architecture de votre code, avec la construction de différentes librairies, gestion correcte et optimale des Threads, etc.

Un conseil: allez voir comment sont faits les exemples de code dans la librairie e-puck2_main-processor (src/main.c).

Attention! Ne partez pas sur la base des TP1-3. L'idéal serait de partir du TP4 ou TP5 qui se base sur e-puck2_main-processor et d'inclure les parties des autres TPs ou de la librairie.

Etape 2: Organiser

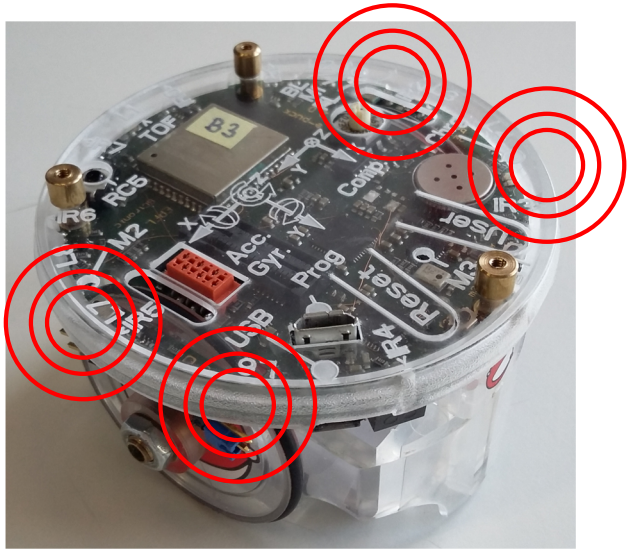
- Il est demandé dans la donnée d'avoir un thread par capteur/actuateur. Cependant si vous regardez dans la librairie e-puck2_main-processor, vous verrez que les librairies des ces mêmes capteurs/actuateurs fonctionnent déjà avec des threads dédiés lorsque c'est nécessaire. Il n'est donc pas utile de créer des threads dans votre code juste pour un capteur. En revanche ça peut être nécessaire pour faire du traitement des données d'un capteur par exemple.

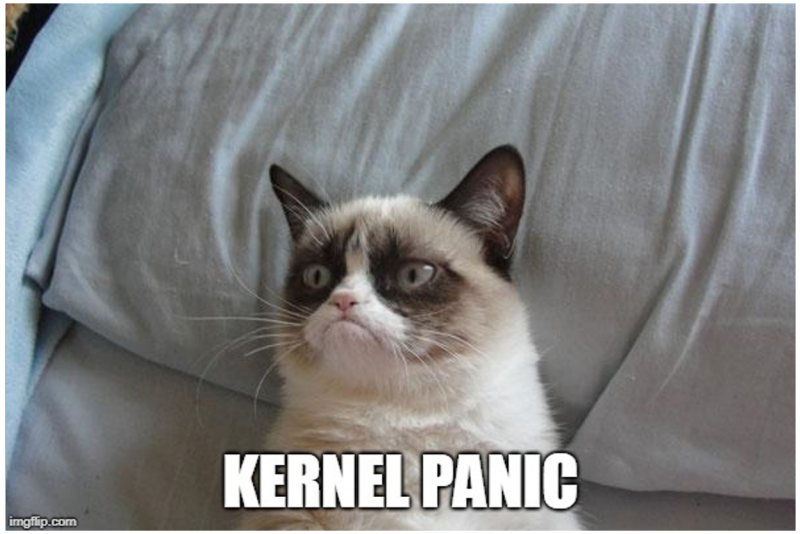
Etape 2: Organiser

- Il faut créer et utiliser un thread uniquement si cela vous paraît utile dans la façon de fonctionner de votre projet (besoin de gérer deux chose en parallèles ou à des fréquences différentes par exemple). Inutile de créer un thread juste pour en créer un. C'est une perte de ressource mémoire inutile et ça n'apporte rien au projet.

Pièges à éviter: Kernel Panic

Vu que vous codez avec un RTOS, il y a des risques d'obtenir des erreurs du système (Kernel Panic). Dans ce cas, le robot allume certaines LED en rouge et se met en attente.





Exemple de causes d'un Kernel Panic

- Violation de la mémoire (Exemple
  pas assez de mémoire allouée à un Thread)

- Appels invalide (exemple Thread non-initialisé)

Si vous avez un Kernel Panic, avant d'appeler un assistant, faites vous même le check de ces points pour vérifier que cela ne vient pas de ça.

Etape 3: Optimiser

- Tenez compte des warnings du compilateur, à la fin il ne devrait pas en rester!
- Évitez d'inclure des librairies non-nécessaires, ou d'initialiser des fonctions (Threads) non-nécessaires
- Ne pas surcharger la pile
- Utilisez les bons types de variables (grosse taille en mémoire vs risque d'overflow)
- Trouvez les bons compromis entre précision et rapidité d'exécution (exemple look-up table, calculs lourds dans un Thread devant s'exécuter rapidement, buffers circulaires pour du filtrage)

Etape 3: Optimiser

- Si vous faites quelque chose qui paraît non optimisé mais que vous pouvez le justifier et expliquer pourquoi c'est nécessaire de faire comme ça alors ok, autrement, ça fait des points en moins…
- Il n'est pas demandé de faire de l'optimisation ultra pointue comme la librairie arm_cfft_32 vue au TP5 par exemple. Cependant si vous utilisez des float, int32_t, etc pour rien ou que vous itérez dans de nombreuses boucles imbriquées alors qu'on pourrait faire facilement plus simple, là vous pouvez perdre des points, faute d'optimisation évidente.

Etape 4: Respecter les conventions

- L'utilisation de variables globales est acceptée uniquement si c'est nécessaire et si elles sont déclarées en static (accessibles uniquement depuis le fichier). Par exemple lorsqu'un thread et une autre fonction doivent accéder à la même variable.
- Pas de nombres magiques dans le code. Utilisez des #define pour les constantes que vous utilisez.
- Mettez des petits commentaires dans vos fonctions lorsqu'on ne peut pas comprendre au premier coup d'œil ce qui s'y passe.
- Utilisez des noms de fonctions et de variables qui permettent de comprendre facilement à quoi elles servent.

Soumission du projet

- Soumettre le rapport (format pdf) et votre logiciel (un archive zip du dossier du projet, on doit pouvoir recompiler chez nous votre projet) sur moodle

- S'inscrire dans le calendrier disponible sur moodle

Rapport

- Il contient seulement ce qui n'est pas dans le code, comme structure générale, tests initiaux, graphes de capteurs, etc

- Respectez le format (sections numérotées, figures et tables numérotées avec légendes, axes sur les graphes, références). Le non respect de ces règles peut amener à une sanction allant jusqu'à un point de moins sur la note finale.

- Une description complète de comment rédiger le rapport est sur moodle

Présentation

Une séance zoom sera organisée et vous allez vous connecter à l'heure convenue dans l'horaire de passage.

La présentation va durer environ 20 minutes:

- 3 minutes (!) de présentation: 3 slides en partage d'écran
- 2 minutes de démonstration live depuis votre caméra, en montrant au mieux votre démonstration.
- 15 minutes de réponse aux question live sur votre projet, avec des questions pouvant creuseur aussi des éléments de théorie en lien à vos choix du projet, par exemple.

Répétez tout cela à l'avance! Lors des tests, enregistrez une fois la vidéo de votre démo, afin d'avoir un backup si la démonstration live ne marche pas et assurer quelques points tout-de-même.

Miniprojets

Rendu du rapport + code: lundi 16 mai, 01h00

Présentations: lundi 16 mai -> vendredi 3 juin

Choix de l'heure de passage: Google Sheet sur moodle, publié le 14 avril. Inscrivez vous par groupe (G01, G56, etc.). Présentations par zoom ou en présentiel.

Enjoy the miniprojet!