TP2: Développement d'un Pilote Automatique

Nizar Ouarti

Résumé

Dans ce TP notre objectif sera de coder un pilote automatique. Nous nous concentrerons sur l'aspect types de données et prototypes. Pour finir nous verrons comment compiler des projets un peu plus importants.

1 Introduction

Vous allez concevoir un pilote automatique. Le programme doit en permanence tester ces différent aspects lié à l'avion. Votre algorithme doit vérifier plusieurs choses :

- Si un bouton est enfoncé si oui lequel
- Si l'altitude n'est pas en dessous d'un seuil
- Si aucun angle de l'attitude est supérieur à un trop gros angle
- Si le niveau du carburant est supérieur au nombres de litres requis pour faire les kilomètres restant
- Il doit vérifier l'état de fonctionnement des 4 moteurs

2 détails du problème

2.1 Altitude, Attitude

- Des information venant d'un gyromètre sont disponibles
- Donc des vitesses angulaires
- Il ne faut pas dépasser 30° d'angle
- Nous avons aussi un altimètre qui donne l'altitude en mètre
- Il ne faut pas être plus bas que (500 mètres d'altitude
- Ces deux évènements sont associés à une alerte spécifique

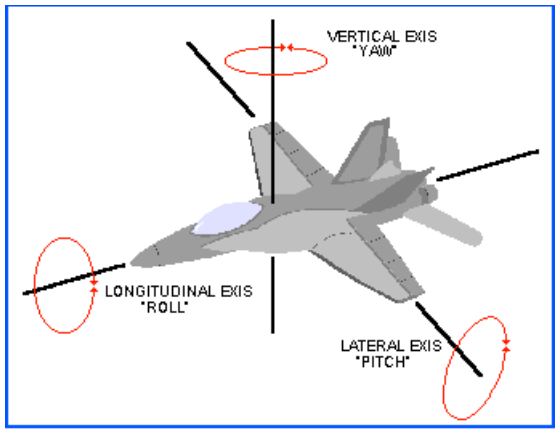

FIGURE 1: Différents angles d'attitude

2.2 Le carburant

- Il suffit de tester la jauge
- Sachant que l'avion consomme L litre par kilomètres
- Sachant que la distance à parcourir est de K kilomètres
- Calculer s'il reste suffisamment de carburant ou sinon donner une alerte

2.3 Les 4 moteurs

- Il faut tester les paramètres des moteurs les uns après les autres
- paramètres : température, problème d'électronique, problème mécanique

2.4 Les boutons

- Il ne peuvent être allumés qu'un à la fois
- ils ont différentes couleurs
  - bleu, rouge, vert, jaune
- Chacun étant lié à une action
  - Arrêt du pilote automatique
  - Ouverture des vannes des réserves d'essences
  - Allumer les lumières extérieures
  - Ouverture de la porte du cockpit

3 Questions

1. Créer un nouveau répertoire TP2
2. Créer un répertoire pilotAuto01 dans TP2
3. Dans ce répertoire, écrire le squelette d'un fichier C s'appelant pilotAuto.c possédant un main.
4. Quelles sont les structures de données à définir pour ce problème ?
5. Ecrire les structures de données en question dans un header. Ce header sera placé dans un répertoire include que vous allez créer et mettre dans pilotAuto01.
6. Comment lier le header au .c ?
7. Usage de gcc : **gcc fichierSource.c -o binaire -lbibliothèque -I include**, avec bibliothèque le chemin vers un répertoire de bibliothèque(autres .o) et include le chemin vers le repertoire où se trouves les headers.
8. Quelles sont les prototypes à définir pour ce problème ?
9. Ecrire les prototypes en question dans le header.
10. Ecrire le contenu du main.
11. Ecrire le contenu de vos fonctions.

Conseils : n'oubliez pas de compiler régulièrement pour trouver les erreurs

4 Le makefile

1. Créer un nouveau répertoire pilotAuto02 dans TP2
2. Ecrire les functions définies dans les prototypes dans un fichier .c séparé du main.
3. On aura deux fichier .c : main.c et pilotAuto.c et le header pilotAuto.h.
4. Adapter le makefile écrit en dessous pour qu'il puisse bien atteindre le header qui est dans le répertoire include.
5. Lancer le makefile avec make

Dans le makefile, chaque règle à un nom suivi de " : " puis une commande après une tabulation
cible : dépendance
Commande (les tabulations sont importantes)

Dans la première ligne ou ligne de dépendance on observe si la cible est plus récente ou ancienne que les fichiers dont elle dépend Si la cible est plus ancienne la commande du bas régénère la cible. On peut utiliser \ si la ligne est trop longue cela indique que la ligne suivante est considéré sur la même ligne.

Le makefile s'utilise de la manière suivante :

```
#commentaires
pilotAuto: pilotAuto.o main.o  #si pilotAuto plus ancient que pilotAuto.o et main.o
        gcc -o pilotAuto pilotAuto.o main.o #alors cette ligne s'éxécute

pilotAuto.o: pilotAuto.c #si pilotAuto.0 plus ancien que pilotAuto.c
        gcc -o pilotAuto.o -c pilotAuto.c -W -Wall -ansi -pedantic #alors cette
            ligne s'éxécute

main.o: main.c pilotAuto.h #si pilotAuto.0 est plus ancien que pilotAuto.c
        gcc -o main.o -c main.c -W -Wall -ansi -pedantic #alors cette ligne s'
            éxécute

clean:
        rm *.o #Efface tous les .o intermédiaires après création de l'exécutable
```

Les options -W -Wall -pedantic servent à indiquer des warnings. Particulièrement -w donnera des warnings sur du code qui peut compiler normalement mais peut être source d'erreur (exemple comparaison d'entiers signés et non signés). -ansi sert à ne pas faire d'interférence avec des variables prédéfini de l'environnement.

Variables d'environnement dans le makefile :

```
# commentaires du makefile
CC=gcc

CFLAGS=-W -Wall -ansi -pedantic

LDFLAGS=

EXEC=pilotAuto

all: $(EXEC)

pilotAuto: pilotAuto.o main.o
        $(CC) -o pilotAuto pilotAuto.o main.o $(LDFLAGS)

pilotAuto.o: pilotAuto.c
        $(CC) -o pilotAuto.o -c pilotAuto.c $(CFLAGS)

main.o: main.c pilotAuto.h
        $(CC) -o main.o -c main.c $(CFLAGS)

clean:
        rm *.o

mrproper: clean #efface
        rm $(EXEC)       #tout

again: clean             #efface
        make ${EXEC}     #et refait
```

Le bonne usage pour l'utilisation de make (qui lance le makefile) est :
make cible
exemple 1 : make pilotAuto, exemple 2 : make clean, exemple 3 : make all

Pour aller plus loin : http ://www.siteduzero.com/tutoriel-3-31992-compilez-sous-gnu-linux.html