COHEN--URRUTIA Florian | BACONNAIS Olivier | BADANIN Roman | BLANC Indie

TP1 : Appontage automatique d'un avion de chasse

Introduction

Un appontage est l'action d'atterir sur une plateforme. Pour notre sujet, nous nous intéressons aux avions de chasse qui attérissent sur un porte avion car c'est la catégorie d'avion qui enregistre le plus d'échac pour cette manœuvre. C'est une action délicate en raison des conditions dans lesquelles elle est exécutée : visibilité pour le pilote, mouvement de la masse d'air autour du porte avion, mouvement du porte avion fortement dépendant de l'amplitude et de la houle, …

On considère que l'orientation d'un avion en vol est déterminée par 3 axes fixes :

Axe 1. L'axe longitudinale, dont la rotation correspond au roulis. On le définit par Ψ

Axe 2. L'axe vertical, pour mesurer le lacet. On le définit par θ

Axe 3. L'axe latéral, pour mesure le tangage. On le définit par φ

La manœuvre d'appontage se déroule en 3 phases :

Phase 1. Le pilote place l'appareil en position d'approche. Il manœuvre pour atteindre 3 objectifs :

- Objectif 1. Aligner l'axe longitudinale de l'avion à l'axe d'appontage
- Objectif 2. Ramener les paramètres de navigation à une configuration initiale
- Objectif 3. Enclencher le pilote d'appontage automatique

Phase 2. Lancer l'algorithme à bord du porte avion qui va générer en temps réel la trajectoire d'appontage. Il transmet aussi les paramètres suivants :

- Paramètre 1 : $V\Psi$ qui est la vitesse de l'avion suivant l'axe longitudinale de l'avion
- Paramètre 2 : Ψ qui est l'angle d'inclinaison de l'axe longitudinale de l'avion vers le porte avion

Phase 3. Fin de manœuvre d'appontage. Les paramètres de l'avion sont ramenés à la configuration suivante :

- $\Psi = 0$
- $\theta = \theta_0$
- $\varphi = 0$
- $V\Psi = 0$

Cela signifit que l'avion apponte et finit par s'immobiliser sur la piste le long de l'axe d'appontage.

Le but de ce projet est de développer une application qui permet d'automatiser la manœuvre d'appontage d'un avion de chasse sur un porte avion.

Question I

Nous avons recours à un développement basé sur une méthode formelle telle que la méthode B car les méthodes formelles permettent d'obtenir une très forte assurance de l'absence de bug dans les logiciels, c'est pourquoi elles sont utilisées pour les logiciels critiques. L'application pour automatiser la manœuvre d'appontage d'un avion de chasse sur un porte avion est une application critique qui requière 0 bug car une simple erreur pourrait avoir des conséquences catastrophiques. Elles permettent aussi de donner une spécification détaillée du système à développer, qui nous permettera de vérifier si la réalisation final est conforme à ce qui était demandé au départ.

Question II

Question 1 : Analyse du système d'appontage

Le système est organisé autour des systèmes suivants :

- Système Pilote
  - Représente le système de contrôle automatique de l'appontage.
  - Etat :
    - Sous-système Contexte
    - Sous-système Appontage
  - Fonction :
    - **EnclenchementPilote(Contexte, Appontage)** : Vérifie que le contexte et l'appontage est optimal pour enclencher le pilote automatique et démarrer la manœuvre. Retourne 0 si tout est optimal, sinon -1.

- Sous-système Contexte
  - Définit le contexte de fonctionnement global du système.
  - Etat :
    - Axe longitudinal : roulis (float)
    - Axe vertical : lacet (float)
    - Axe latéral : tangage (float)
    - Vitesse de l'avion : vitesse (float)
  - Fonction :
    - **Initialisation(roulis, lacet, tangage)** : initialisation du système applique 0 à tout les paramètres, puis les retourne.

- Sous-système Appontage
  - Système opérationnel chargé de coordonner le maintien de l'avion dans le plan d'approche et la trajectoire de descente de l'avion dans ce plan.
  - Etat :
    - Sous-système PlanApproche
    - Sous-système Trajectoire
  - Fonction :
    - **Appontage(PlanApproche, Trajectoire)** : permet de coordonner la trajectoire avec le plan d'approche. Les paramètres seront égaux lorsque l'avion sera posé. Retourne 0 si la manœuvre s'est bien passée, sinon -1.

- Sous-système PlanApproche ***A FAIRE***
  - Garantit le maintien de l'avion dans le plan d'approche.
  - Etat :
    - Sous-système RegulateurPlan
    - Sous-sytème CapteurPlan
  - Fonction :
    - **Maintien(RegulateurPlan, CapteurPlan)** : permet de maintenir l'avion grâce au RegulateurPlan

- Sous-sytème CapteurPlan
  - Récupère les données réelles fournies par les capteurs du roulis et du lacet de l'avion.
  - Etat :
    - Sous-sytème CapteurLacet
    - Sous-sytème CapteurRoulis
    - Axe longitudinal : roulis (float)
    - Axe vertical : lacet (float)
  - Fonctions :
    - **RecupRoulis(CapteurRoulis, roulis)** : récupère la donnée de CapteurRoulis pour la mettre dans roulis. Retourne roulis s'il a pu lrécupérer la donnée sinon retourne un code d'erreur.
    - **RecupLacet(CapteurLacet, lacet)** : récupère la donnée de CapteurLacet pour la mettre dans lacet. Retourne lacet s'il a pu récupérer la donnée sinon un code d'erreur.

- Sous-système CapteurRoulis
  - Mesure le roulis de l'avion.
  - Etat :
    - Axe longitudinal : roulis (float)
    - Sous-système BASIC_IO
  - Fonction :
    - **MesureRoulis(BASIC_IO, roulis)** : mesure le roulis réel de l'avion avec BASIC_IO qui récupère la donnée en entrée et la met dans roulis. Retourne roulis s'il a pu récupérer la donnée en entrée sinon retourne un code d'erreur.

- Sous-sytème CapteurLacet
  - Mesure le lacet de l'avion.
  - Etat :
    - Axe vertical : lacet (float)
    - Sous-système BASIC_IO
  - Fonction :
    - **MesureLacet(BASIC_IO, lacet)** : mesure le lacet réel de l'avion avec BASIC_IO qui récupère la donnée en entrée et la met dans lacet. Retourne lacet s'il a pu récupérer la donnée en entrée sinon retourne un code d'erreur.

- Sous-sytème RegulateurPlan ***A FAIRE***
  - Corrige en temps réel les paramètres (lacet-roulis) de sorte à maintenir l'avion dans le plan d'approche
  - Etat :
    - Sous-système RegulateurLacet
    - Sous-système RegulateurRoulis
  - Fonction :
    - Fonction

- Sous-système RegulateurRoulis ***AFAIRE***
  - Corrige le paramètre roulis de sorte que l'avion soit maintenu dans le plan d'approche
  - Etat :
    - Axe longitudinal : roulis (float)
  - Fonction :
    - Fonction

- Sous-sytème RegulateurLacet ***AFAIRE***
  - Corrige le paramètre lacet de sorte que l'avion soit maintenu dans le plan d'approche
  - Etat :
    - Axe vertical : lacet (float)
  - Fonction :
    - Fonction

- Sous-système Trajectoire ***AFAIRE***
  - Contrôle la trajectoire de descente de l'avion maintenu dans le plan d'approche
  - Etat :
    - Sous-système Aiguilleur
    - Sous-système CapteurTrajectoire
    - Sous-système RegulateurTrajectoire
  - Fonction :
    - Fonction

- Sous-système Aiguilleur
  - Reçoit les données réelles sur les paramètres vitesse et tangage de l'avion et retourne les données pour corriger en temps réel la trajectoire d'appontage.
  - Etat :
    - Vitesse de l'avion : AiguilVitesse (float)
    - Axe latéral : AiguilTangage (float)
    - Sous-système CapteurVitesse avec :
      - Vitesse de l'avion mesurée par le capteur : vitesseCap (float)
    - Sous-système CapteurTangage avec :
      - Axe latéral mesuré par le capteur : tangageCap (float)
  - Fonctions :
    - **MesureVitesse(AiguilVitesse, vitesseCap)** : Calcul la vitesse nécessaire à l'avion pour continuer la manœuvre d'appontage grâce à vitesseCap. Met la valeur calculée dans AiguilVitesse. Retourne AiguilVitesse.
    - **MesureTangage(AiguilTangage, tangageCap)** : Calcul le tangage nécessaire à l'avion pour continuer la manœuvre d'appontage grâce à tangageCap. Met la valeur calculée dans AiguilTangage. Retourne AiguilTangage.

- Sous-système BASIC_IO
  - Gère les entrées/sorties du simulateur.
  - Etat :

        - Entrées du simulateur : simulateurE
        - Sorties du simulateur : simulateurS
    - Fonctions :
        - **RecepDonnée(simulateurE)** : récupère une donnée en entrée du simulateur et la met dans simulateurE. Retourne simulateurE ou -1 si la donnée n'est pas lue.
        - **EnvoiDonnée(simulateurS)** : envoi la donnée contenue dans simulateurS par la sortie du simulateur. Retourne simulateurS ou -1 si la donnée n'a pas été envoyée.

- Sous-système CapteurTrajectoire
    - Récupère les données réelles sur la vitesse et le tangage de l'avion.
    - Etat :
        - Sous-système CapteurVitesse avec :
            - Vitesse de l'avion mesurée par le capteur : vitesseCap (float)
        - Sous-système CapteurTangage avec :
            - Axe latéral mesuré par le capteur : tangageCap (float)
        - Axe latéral : tangage (float)
        - Vitesse de l'avion : vitesse (float)
    - Fonctions :
        - **RecupVitesse(vitesseCap, vitesse)** : récupère la donnée de vitesseCap pour la mettre dans vitesse. Retourne vitesse.
        - **RecupTangage(tangageCap, tangage)** : récupère la donnée de tangageCap pour la mettre dans tangage. Retourne tangage.

- Sous-système CapteurVitesse
    - Mesure la vitesse réelle de l'avion.
    - Etat :
        - Vitesse de l'avion mesurée par le capteur : vitesseCap (float)
        - Sous-système BASIC_IO
    - Fonction :
        - **VitesseReelle(BASIC_IO, vitesseCap)** : mesure la vitesse réel de l'avion avec BASIC_IO qui récupère la donnée en entrée et la met dans vitesse. Retourne vitesseCap.

- Sous-système CapteurTangage
    - Mesure l'angle de tangage réel de l'avion.
    - Etat :
        - Axe latéral mesuré par le capteur : tangageCap (float)
        - Sous-système BASIC_IO
    - Fonction :
        - **TangageReel(BASIC_IO, tangageCap)** : mesure le tangage réel de l'avion avec BASIC_IO qui récupère la donnée en entrée et la met dans tangage. Retourne tangageCap.

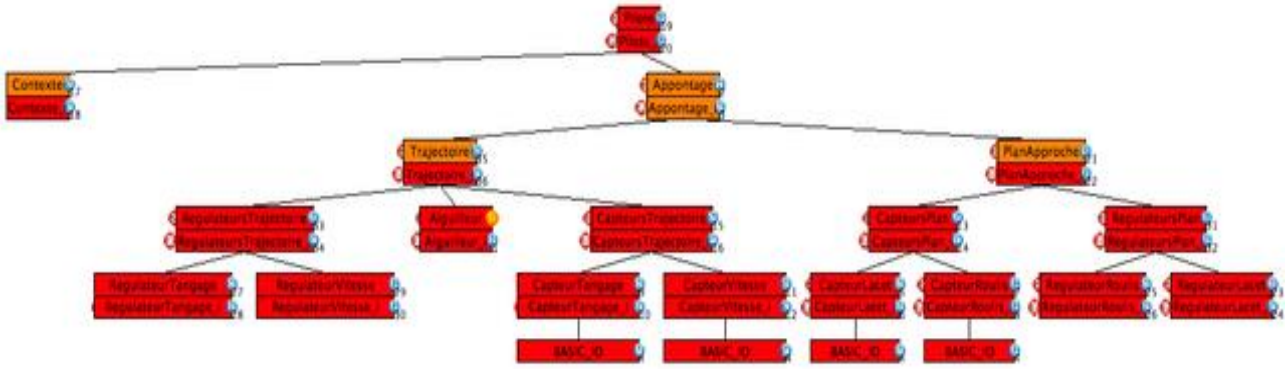
- Sous-système RegulateurTrajectoire
    - Commande les régulateurs de vitesse et de tangage afin de corriger la trajectoire de l'avion.

- Etat :
  - Sous-système RegulateurVitesse avec :
    - Différence entre la vitesse réelle et la vitesse nécessaire : différenceV (float)
  - Sous-système RegulateurTangage avec :
    - Différence entre le tangage réel et le tangage nécessaire : différenceT (float)
  - Vitesse de l'avion : vitesse (float)
  - Axe latéral de l'avion : tangage (float)
- Fonctions :
  - **CorrVitesse(differenceV, vitesse)** : Ajoute differenceV à vitesse. Retourne vitesse.
  - **CorrTangage(differenceT, tangage)** : Ajoute differenceT à tangage. Retourne tangage.

- Sous-système RegulateurVitesse
  - Corrige la vitesse de l'avion en appliquant la consigne de l'aiguilleur automatique.
  - Etat :
    - Vitesse de l'avion mesurer par le capteur de vitesse : vitesse (float)
    - Sous-système Aiguilleur avec :
      - Vitesse de l'avion mesurer par l'aiguilleur : AiguilVitesse (float)
    - Différence entre la vitesse réelle et la vitesse nécessaire : différenceV (float)
  - Fonction :
    - **DiffVitesse(AiguilVitesse, vitesse, differenceV)** : Mesure la différence entre la vitesse réelle (vitesse) et la vitesse nécessaire (AiguilVitesse). Retourne différenceV.

- Sous-sytème RegulateurTangage
  - Corrige le tangage de l'avion en appliquant la consigne de l'aiguilleur automatique.
  - Etat :
    - Axe latéral récupérer par le capteur tangage : tangage (float)
    - Sous-système Aiguilleur avec :
      - Axe latéral mesurer par l'aiguilleur : AiguilTangage (float)
    - Différence entre le tangage réel et le tangage nécessaire : différenceT (float)
  - Fonction :
    - **DiffTangage(AiguilTangage, tangage, differenceT)** : Mesure la différence entre le tangage réel (tangage) et le tangage nécessaire (AiguilTangage). Retourne différenceT.

Question 2 : Modélisation en B du système



Question 3 : Validation du modèle précédent

Question 4 : Génération du code source C++ du logiciel qui contrôle l'appontage