Compte rendu PST drone Jumping Sumo

Github du projet : https://github.com/Natu181/JumpingSumoTrackingPST

Le principe de de PST est de se connecter un à drone Parrot Jumping Sumo, de récupérer l'image de la caméra du drone et d'effectuer des actions en fonction de l'image reçue. Le projet original comprenait un module permettant de contrôler manuellement le drone ainsi qu'un module permettant de suivre un cercle de couleur verte.

Nous avons refait l'application en nettoyant certaines parties et ajouté deux fonctionnalitées: un détecteur qui permet de suivre une ligne noire ainsi qu'un détecteur d'obstacle naïf. Nous avons essayé d'implémenter d'autres pistes qui seront détaillés à la fin du document.

I. L'application

A. Vue d'ensemble

Ce projet a été développé sur Android Studio, avec l'ARSDK de Parrot (ajoutée via Gradle, lien: http://developer.parrot.com/docs/SDK3/) et la version Java Android de OpenCV (ajoutée via Android Studio car elle n'est pas disponible sur Gradle, lien: https://opencv.org/releases.html)
L'application a été remise au propre pour faciliter la prise en main par de futurs groupes, il est cependant très probable que certaines parties soient optimisables et/ou à refaire.
L'application est composée de plusieurs parties :

- **activities :** comporte les activités TrackingModeSelectionActivity, l'activité principale qui permet à l'utilisateur de choisir entre les différents modes de contrôles du Drone et l'activité DroneControlActivity qui est celle que l'on utilisera pour gérer le drone.
- **detectors :** comporte les différentes implémentations de l'interface Detector (CircleDetector, LineDetector, ObstacleDetector), un Detector est une classe dont une de ces méthodes sera appelée à chaque Frame reçue du drône.
- **drone :** comporte la classe JumpingDiscoverer permettant de lister les drones découverts et la classe JumpingSumo représentant le drone et permettant de le manipuler dans nos activités.
- **views :** comporte la classe FrameByFrameImageView permettant de récupérer la vidéo du drone. (classe récupérée du projet original, certainement optimisable)
- **TrackingModes :** les différents modes de fonctionnement du drone

B. Détails

Dans l'application nous utilisons différents TrackingModes liés aux Detector:

- **Manual :** mode permettant le pilotage manuel du drone.
- **Circle Tracking (CircleDetector) :** mode déjà implémenté permettant au drone de suivre des cercles.
- **Obstacle Detection (ObstacleDetector) :** permettant au drone de détecter des obstacles et de sauter par dessus.
- **Line Detection (LineDetector) :** permettant au drone de détecter une ligne noire et de la suivre.

La classe TrackingModes est une énumération dont chaque valeur possède 4 paramètres:

- un id unique
- une String pour l'identifier sur l'application
- une List de Detectors, correspondant aux détecteurs à employer pour ce mode (ceux-ci sont activés en série)
- un Boolean pour l'utilisation ou non des commandes manuelles

Pour ajouter un mode, il faut:

- ajouter sa valeur dans l'enum TrackingModes avec un nouvel id, une String pour l'affichage et le boolean si l'on veut les commandes manuelles ou non
- ajouter dans la méthode "modesSetup" les lignes de code pour ajouter une instance du ou des détecteurs que l'on souhaite dans la config du mode.

II. Traitement d'images

pb : sauts de frames -> perte d'information

Avant tout traitement, il faut convertir l'image reçue dans un format exploitable, on convertit donc les bitmap reçus en matrice OpenCV.

Ensuite pour traiter la vidéo nous avons utilisé les fonctions de traitement d'image d'OpenCV.

HoughLinesP Cany Core.Ellipse traiter l'image pour la rendre exploitable bitmapToMat

III. Les detectors

A. ObstacleDetector

Un détecteur d'obstacles naïf, il compte la proportion de 'edges' dans une zone de l'image correspondante à la zone immédiatement devant le drone. On utilise une ellipse pour définir la zone critique

Cany edges pour détecter les contours des objets, saut si les lignes d'un objet rentrent en contact avec l'ellipse.

B. LineDetector

Le détecteur se base sur les 10 premières lignes de l'image, il y calcule le point noir médian et le suit. Ce détecteur fonctionne sur le principe que la ligne est noire et continue, il arrive que la connexion avec le drone pose des problèmes de saut d'images ce qui peut faire sortir le drone de la trajectoire.

IV. Pistes à exploiter

Lors de ce PST nous avons essayé de faire en sorte que le drone émette un avertissement sonore lors d'une détection d'obstacle mais ça n'a pas été fructueux. Ainsi serait-il intéressant de se pencher sur les classes AudioRecorder et AudioPlayer.
La détection d'obstacle est également rudimentaire et pourrait être plus approfondie ainsi que le comportement du drone en fonction de l'obstacle rencontré.

Déroulement du PST :

- Détection d'obstacle → ellipse et dès que objet dans la zone → saut
- Avertissement sonore (à partir d'un son du telephone [stream audio] puis à partir des sons existants dans le drone) → échec critique
- Suivi de chemin : segment au milieu de l'image et faire en sorte que la ligne du chemin ne dépasse pas un certain angle pour que le drone avance sinon il tourne pour se recaler → Bien mais pas suffisamment efficace
- Suivi de chemin à partir d'une zone de la vidéo → rectangle en bas de la vidéo dans lequel on détecte le chemin en fonction de son niveau de noir, on prend la moyenne de tous les points noirs détectés et on suit ce chemin.
- Remise du projet au propre pour faciliter la prise en main par de futurs groupes.